

## Multi-Function PCI Pass-Through

Simon Horman <simon@valinux.co.jp>  
VA Linux Systems Japan K.K.  
Jun Kamada <kama@jp.fujitsu.com>  
Fujitsu Limited

linux.conf.au 2010  
Wellington, New Zealand  
18th – 23rd January

# Motivation

- Aim of my work:  
*To enhance Xen PCI pass-through to allow multi-function devices appear in unprivileged-domains (guests) as multi-function devices.*

# Motivation

- Aim of my work:  
*To enhance Xen PCI pass-through to allow multi-function devices appear in unprivileged-domains (guests) as multi-function devices.*
- Aim of this presentation:  
*To explain what that means and some of the challenges encountered while making it possible.*

## Part I

# Overview of PCI Pass-Through

# PCI Pass-Through

- Method of making a PCI function available to a guest.
- KVM calls this feature PCI Device Assignment
- Typically uses an IOMMU to provide isolation
  - Otherwise guests can use DMA to access memory they shouldn't.
- This discussion focuses on fully-virtualised guests, although it should also be applicable to para-virtualised guests.

# PCI Devices and Functions

- A PCI device may include between 1 and 8 functions
- Function numbers range from 0 to 7
- Function 0 must always be present
- Classified as single-function and multi-function devices

## Single-Function PCI Device

```
$ lspci -s 02:02.*
```

```
02:02.0 Ethernet controller: Realtek Semiconductor Co., Ltd.  
      RTL-8169 Gigabit Ethernet (rev 10)
```

## Multi-Function PCI Device

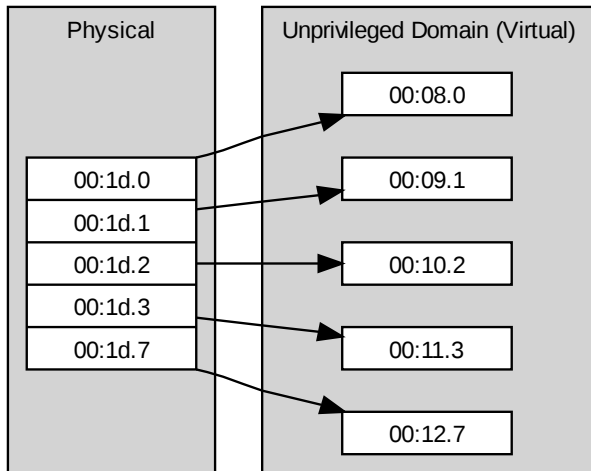
```
$ lspci -s 00:1d.*
```

```
00:1d.0 USB Controller: Intel Corporation  
      82801G (ICH7 Family) USB UHCI Controller #1 (rev 01)  
00:1d.1 USB Controller: Intel Corporation  
      82801G (ICH7 Family) USB UHCI Controller #2 (rev 01)  
00:1d.2 USB Controller: Intel Corporation  
      82801G (ICH7 Family) USB UHCI Controller #3 (rev 01)  
00:1d.3 USB Controller: Intel Corporation  
      82801G (ICH7 Family) USB UHCI Controller #4 (rev 01)  
00:1d.7 USB Controller: Intel Corporation  
      82801G (ICH7 Family) USB2 EHCI Controller (rev 01)
```



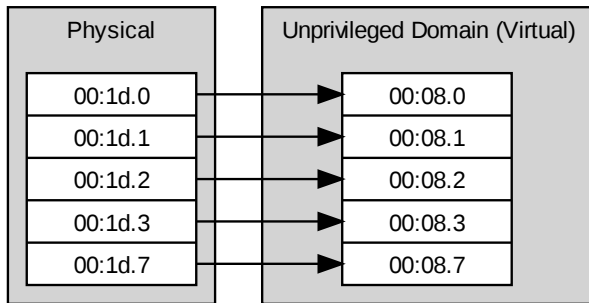
# No Multi-Function in Unprivileged Domains

Prior to this work Xen allowed functions to be passed through as single-function devices.



# Multi-Function in Unprivileged Domains

This work allows functions of a multi-function device to be passed-through as a multi-function device.



# Xen Pass-Through Architecture

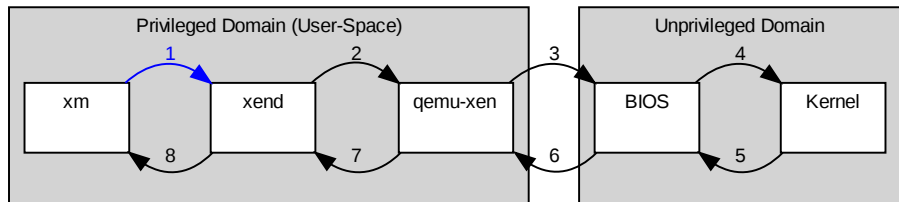
## Four operations

- Attachment
  - At unprivileged domain boot-time (static assignment)<sup>1</sup>
  - While unprivileged domain is running (hot-plug)
- Detachment
  - While unprivileged domain is running (hot-unplug)
- Listing of attached devices
- Listing of attachable devices

---

<sup>1</sup>Static assignment isn't actually static as such devices may be hot-unplugged

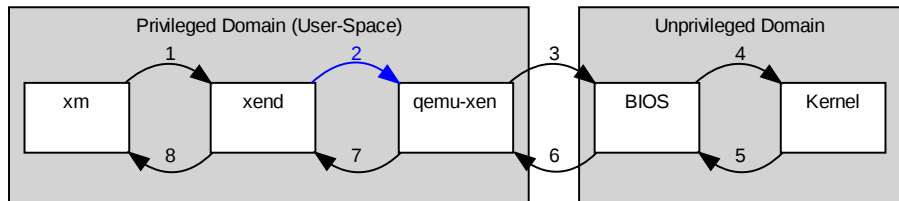
# Attachment and Detachment Events: xm



## 1 xm is a command-line tool

- Accepts commands from the end-user
- Makes corresponding requests to xend

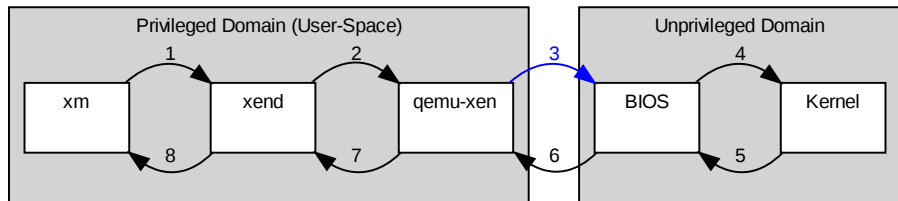
# Attachment and Detachment Events: xend



## 2 xend marshals information between sub-systems

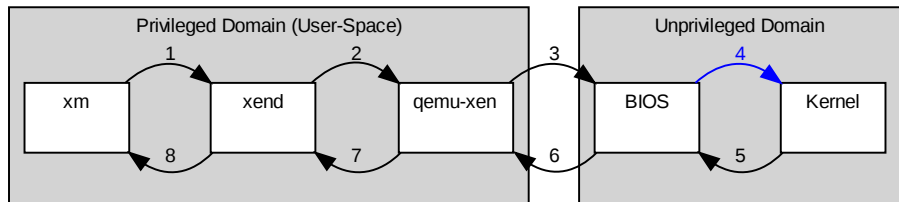
- Checks the pass-through commands sent by `xm`
- Reconciles them with the current state of the system
- Passes them on to `qemu-xend`

# Attachment and Detachment Events: qemu-xen



- 3 `qemu-xen` is used to emulate devices and control pass-through devices
- Reconfigures the xen hypervisor accordingly
  - Triggers a corresponding ACPI event in the virtual BIOS of the target unprivileged domain

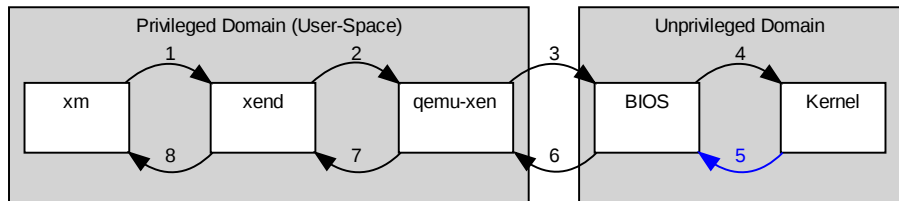
# Attachment and Detachment Events: BIOS



## 4 Unprivileged domain's virtual BIOS

- Triggers a corresponding ACPI event in the kernel

# Attachment and Detachment Events: Kernel

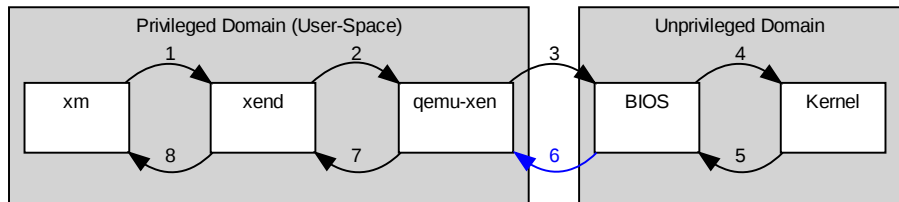


## 5 Unprivileged domain's kernel

- Hot-plugs or unplugs the device
- Sends an acknowledgement back to the BIOS



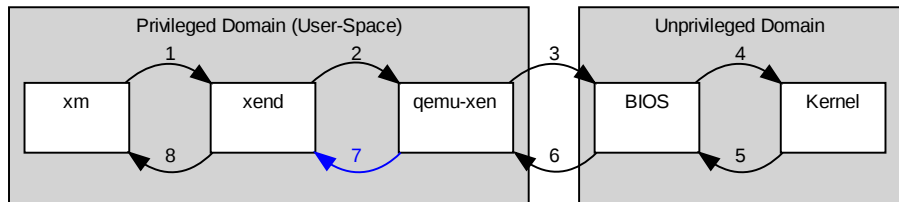
# Attachment and Detachment Events: BIOS Ack



## 6 Unprivileged domain's virtual BIOS

- Sends an acknowledgement to qemu-xen

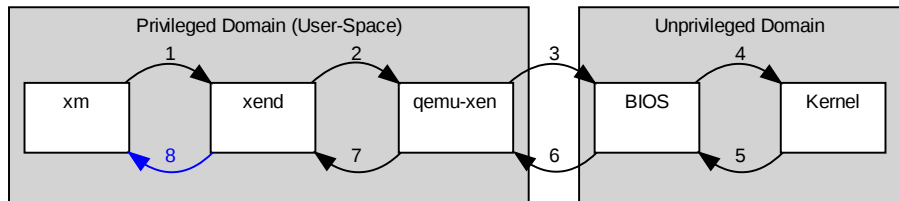
# Attachment and Detachment Events: qemu-xen ACK



## 7 qemu-xen

- Updates its internal state
- Sends an acknowledgement to xend

# Attachment and Detachment Events: xend Ack



## 8 xend

- Updates its internal state and that of xenstore
- Sends an acknowledgement to xm

## Part II

### Implementation Challenges

# User Interaction

- Problem: Need a succinct way to describe multi-function devices
- Solution: Extend BDF notation
  - BDF stands for Bus Device Function
  - Used to describe PCI and PCIe devices

# Simple BDF Notation

00:02.0

- PCI Bus number in hexadecimal
- A colon (:)
- PCI Device number in hexadecimal  
Sometimes referred to as the slot number
- A decimal point (.)
- PCI Function number

# Extended BDF Notation

Optionally prefixes simple BDF with the PCI domain<sup>2</sup>

0000:00:02.0

- PCI domain number
- A colon (:)
- Simple BDF Notation

---

<sup>2</sup>PCI domains do not correspond to Xen domains

# Extended BDF Notation with Virtual-Slots

Optionally suffixes extended BDF with the virtual-slot or pass-through options to be used.

```
0000:00:02.0@7,msitranslate=1
```

- Extended BDF Notation
- An at-sign (@)
- A virtual slot

Or any number of:

- A comma (,)
- An option name
- An equal sign (=)
- A value for the option and yes or no.

In the case where both a virtual-slot and options are specified, the virtual-slot must come first

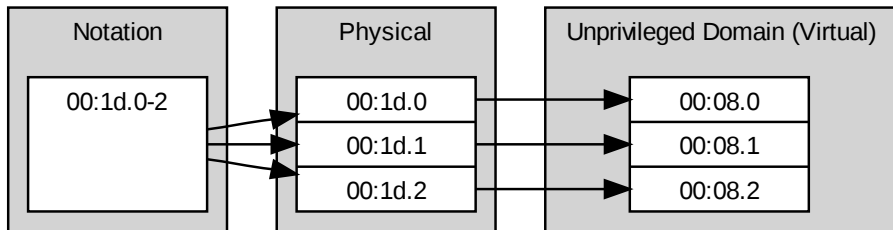


# BDF Notation for Multi-Function

The function field is expanded to accept a comma-delimited list of:

- Function numbers
- A range of function numbers, denoted by:
  - The first function number
  - A hyphen (-)
  - The last function number
- An asterisk (\*)

This notation is internally expanded into groups of functions

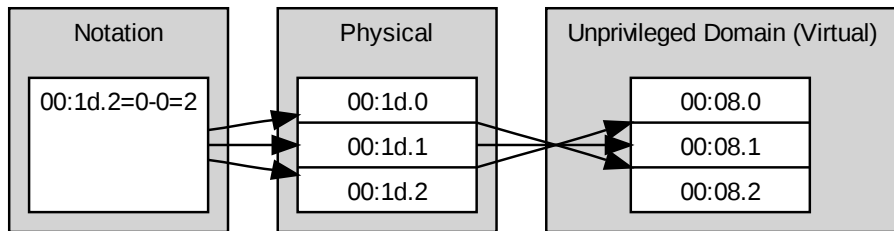


# BDF Notation for Multi-Function with Explicit Vfunctions

Allows control over the mapping of physical to virtual functions

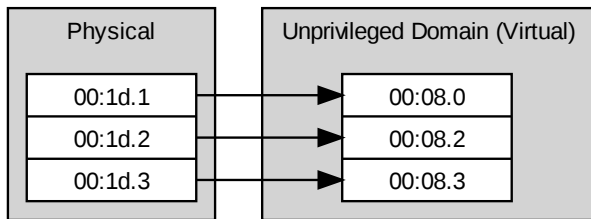
Physical function numbers are replaced by function units which comprise of:

- Physical function number and optionally;
- An equal sign and;
- A virtual function number to use



# Mapping Physical-Functions to Virtual-Functions

- Use any virtual functions in BDF
- Then, map the lowest remaining physical function to virtual function 0 as needed
- Finally, identity map the rest of the functions



- A virtual device must always include virtual function zero

# Allocating Virtual-Functions

- Virtual-Functions are assigned by `xm` (hot-plug) or `xend` (boot-time assignment) at the time BDFs are parsed
  - It knows which functions belong to the same device
  - Allows for BDF to specify virtual-functions

# Allocating Virtual-Slots

- Virtual-Slots are assigned by qemu-xen
  - It knows which slots are free
- An extended devfn scheme is used
  - Between xm and xend
  - Between xend and qemu-xen
  - Flag is set:
    - qemu-xen should allocate a free slot
    - device/slot bits are filled in by qemu-xen
  - Flag is not set:
    - BDF specifies slots
    - device/slot bits read by qemu-xen

flag (bit 9)	device/slot (bits 3-7)	function (bits 0-2)
--------------	------------------------	---------------------

# Device Keys

- xend, qemu-xen and ACPI deal with per-function requests
- Need a way to identify functions that are members of the same function device
- A key is added to the functions internal representation in xend
  - At this stage it is the BDF string used to specify the device
  - Due to insertion-time checks it is guaranteed to be unique

# Attachment

1. Find all the functions with the same key
2. Order the functions so that virtual-function zero is last
3. Attach the first function
4. If there are no more functions, finish — it is a single-function device
5. Else, if the virtual-slot is to be automatically assigned
  - 5.1 Request the virtual function of the function that was just inserted
  - 5.2 Set the virtual-slot of all remaining functions to this value
6. Attach each of the remaining functions

gemu-xen only sends an ACPI event to the BIOS for function zero, which is always last

# Detachment

1. Find all the functions with the same key
2. Order the functions so that virtual-function zero is last
3. Detach each function

xend only sends a notification to qemu-xen for function zero, which is always last



# ACPI BIOS

- Extended from 2 slots to 32
  - Removed arbitrary limitation in original Xen pass-through code
- Extended from 1 function per slot to 8
- Auto-generated the BIOS ASL code
  - Very repetitive
  - $\sim 32 \text{ lines} \times 32 \text{ slots} \times 8 \text{ functions} \approx 8000 \text{ lines}$
- Matching changes in qemu-xen
  - Not matching verbosity

# Conclusion

- Incremental improvement to pass-through for Xen
  - user/xm/xend/qemu-xen interaction was by far the most time-consuming portion
- Functions from multiple virtual devices in a single multi-function virtual-device would be interesting — possibly very broken



This work was partly funded by Ministry of Economy, Trade and Industry (METI) of Japan as the Secure Platform project of Association of Super-Advanced Electronics Technologies (ASET).

Questions?