Virtualization with XEN

Trusted Computing
CS599 Spring 2007

Arun Viswanathan
University of Southern California

Agenda

- Introduction
  - Virtualization approaches
  - Basic XEN Architecture
  - Setting up XEN
  - Bootstrapping XEN dom-0 and dom-U
- Meeting the Devil
  - XEN Virtual Machine Monitor Design
  - XEN 3.0 architecture
- Practicality and applicability
  - Benefits of XEN for Trusted Computing Research
  - Building a virtual network using XEN (DEMO)
- References

Virtualization Approaches

- **Full Virtualization**
  - ✔ Hyper visor provides fully emulated machine eg. Vmware
  - ✗ Performance is a concern
- **Single Kernel Image a.k.a Lightweight virtualization**
  - ✔ Host OS spawns additional copies of itself eq. Virtuozzo, Solaris Zones
  - ✗ Not possible to run different OS/OS versions on same machine
  - ✗ Exploitation of base OS leads to exploitation of all
- **Para Virtualization**
  - ✔ Virtualization technique that presents a software interface to virtual machines that is similar but not identical to that of the underlying hardware eg. XEN
  - ✔ Faster, scalable and more secure than above two approaches

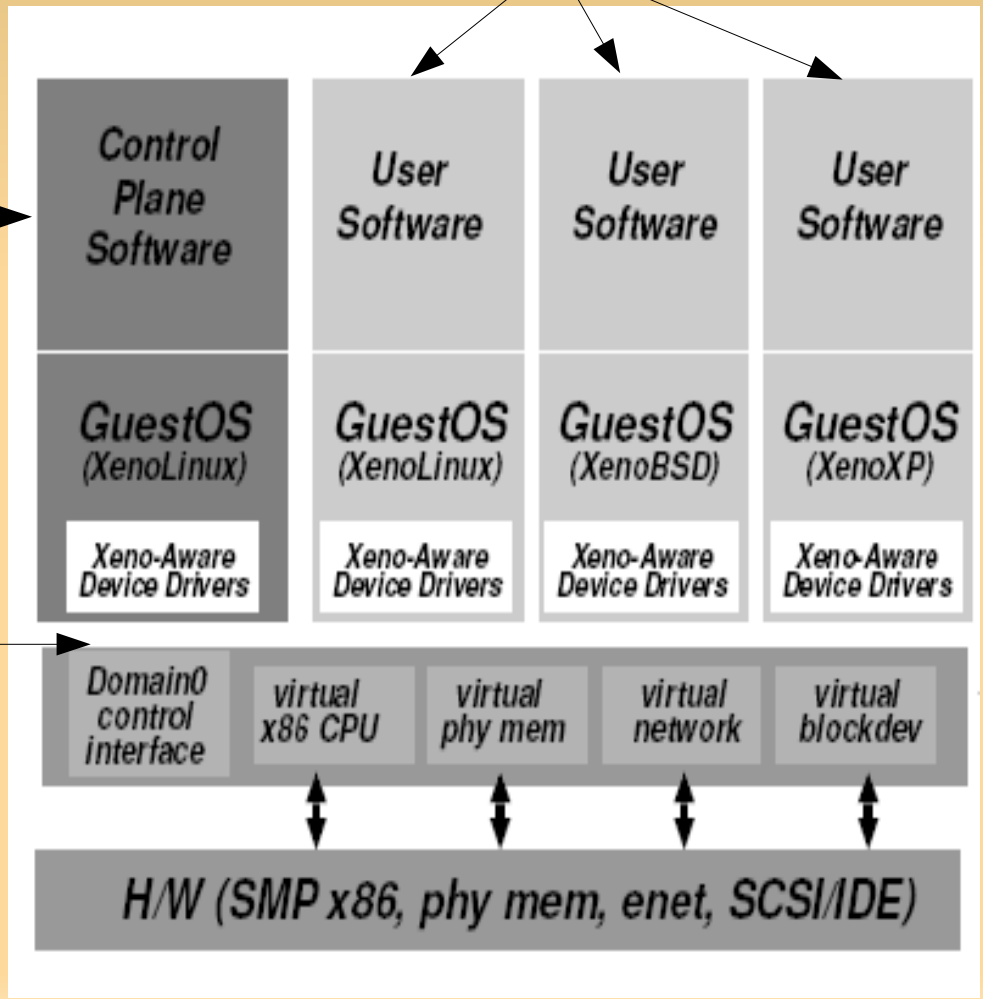
Introduction

Basic XEN Architecture

Domain-U
Runs a lean kernel with virtual drivers

Domain-0
- Full kernel
- Native DD
- XEN control tools (xend/xm etc)

Hyper call Interface to trap from Ring 1 to Ring 0



Ring 3

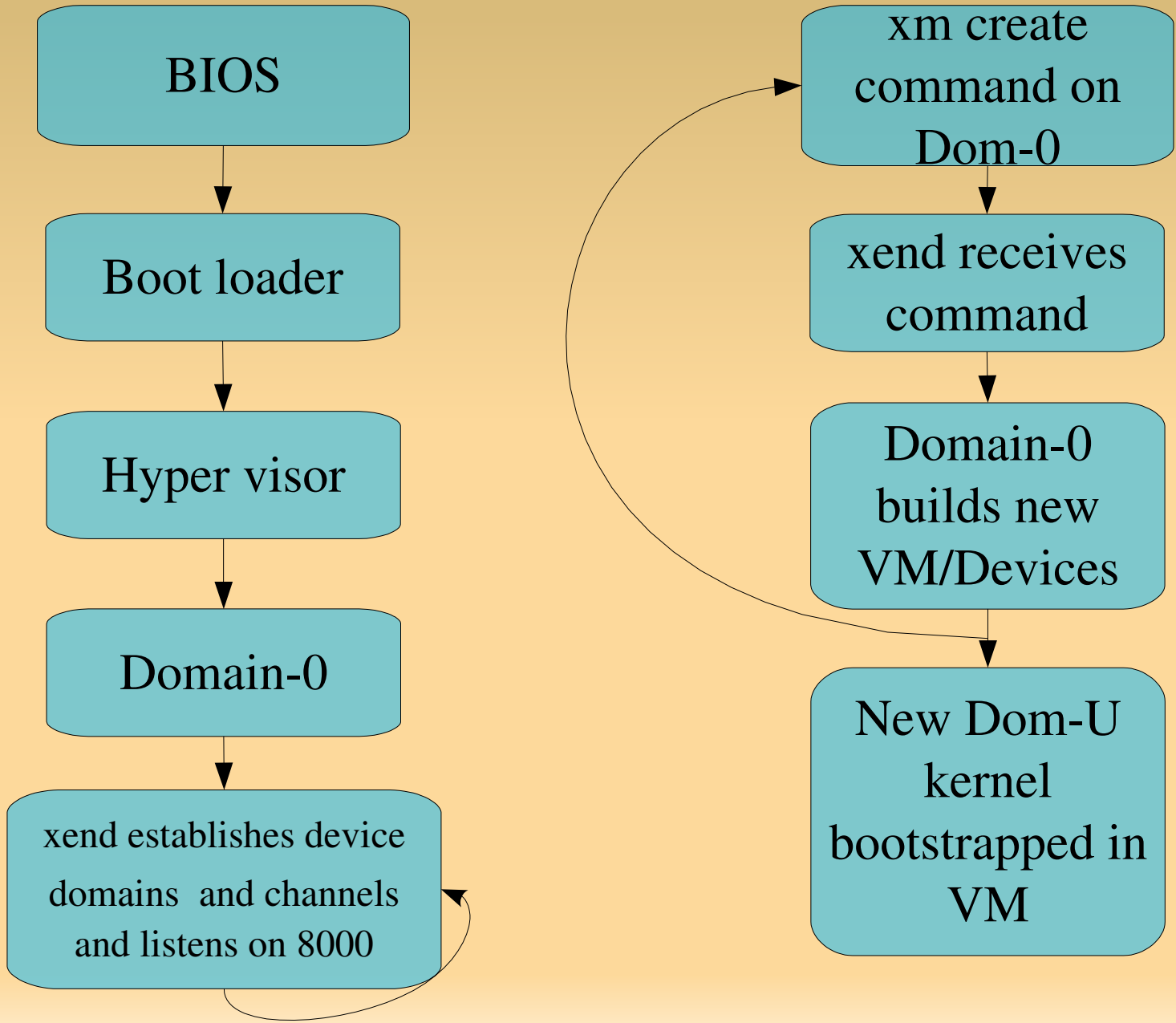
Ring 1

Ring 0

Setting up XEN

- My Configuration
  - Pentium-M, 1GB Ram (no VT, no TPM :(
  - Ubuntu `Edgy Eft' ver 6.10
  - Linux Kernel 2.6.16.29 with Xen 3.0.3
  - Dom-0 uses 512MB memory, Dom-U's are allocated 128MB
- Steps to get up and running
  - Download the xen sources
  - Compile xen hypervisor, dom-0 and dom-U kernels
  - Edit grub and place the hypervisor and dom-0 kernel entries
  - Boot using the hypervisor/dom-0 kernel
  - Build images for guest kernels (i used debian)
    - One can also use stripped down linux systems like **tty-linux**
  - Create a VM config file for XEN
  - Create new domains using dom-U kernels and guest images

Bootstrapping Xen dom-0 and dom-U



Meet the Devil

XEN Design Goals

- ✔ **Efficient Paravirtualization of the x86 interfaces**
  - CPU
  - Memory management
  - Time and Timers
  - I/O
- ✔ **Safe hardware access**
  - Unmodified device drivers are shared across isolated OS instances
  - System as a whole is protected from driver bugs and failures.

CPU Virtualization

- x86 CPUs have 4 rings of operation
- Xen runs in ring 0, whereas the kernel moved to ring 1 or 2.
- Privileged instructions are paravirtualized
- 17 / 250 instructions in IA32 cause problems when run in ring 1 (eg. SGDT, SLDT, PUSPF, POPF, SMSW, HLT etc)
- In some cases like HLT the instruction is replaced with a hypercall which is like a system call (INT 0x82) and is used to move from ring 1 to ring 0.
- In some cases like CLTS (Clear Task Switch flag in CR0) the control passes onto ring 0 and the instruction is then executed in ring 0 by XEN.
- Exceptions including memory faults and software traps are virtualized by associating a handler with the exception and registering with XEN for validation.
- System calls can be serviced by 'fast' exception handlers without indirecting via XEN. At registration time XEN checks that the exception handlers do not specify execution in ring 0.

CPU Virtualization (cont...)

- **Protecting XEN from Guest OSes**
  - Paging provides a supervisor/user permission check, but there's only one bit, and it says that rings 0-2 are all "supervisor" rings, and only ring 3 is prohibited from seeing those pages.
  - Xen sets up the segment registers so that the top 64MB of memory is beyond the segment limit whenever the guest OS is running.
  - This also helps in preventing costly TLB flushes when making a hypercall.

Time and Timers

- Xen provides guest OSes with notions of real time, virtual time and wall-clock time.
- Real time is expressed in nanoseconds passed since machine boot and is maintained to the accuracy of the processor's cycle counter
- A domain's virtual time only advances while it is executing: this is typically used by the guest OS scheduler to ensure correct sharing of its timeslice between application processes.
- Finally, wall-clock time is specified as an offset to be added to the current real time. This allows the wall-clock time to be adjusted without affecting the forward progress of real time.
- For a real kernel, a second going by in kernel time is a second going by on the clock on the wall. However, when a kernel is being virtualised, a second going by for the kernel can be several seconds of real time as it is sharing the hardware with all the other kernels on that same computer.
- Therefore a virtualised kernel must be aware of both real wall clock time, and virtual processor time - the time which it has actual access to the hardware.

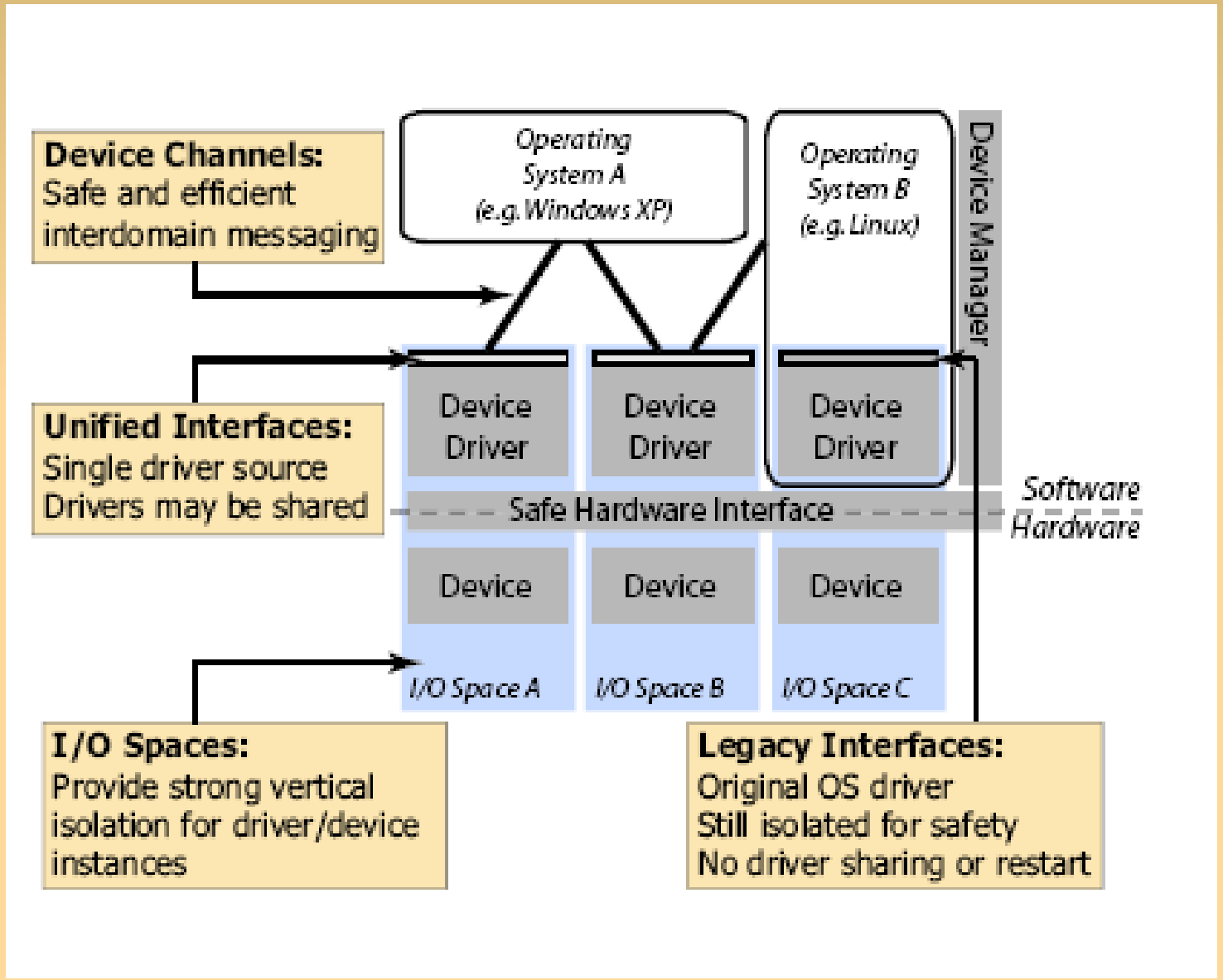
I/O Handling

- dom0 is a privileged domain that can touch all hardware in the system
- dom0 exports some subset of the the devices in the system to the other domains
- Devices are exported via device channels - an asynch shared memory transport coupled with event rings for interrupts
- dom0 runs the backend of the device, which is exported to each domain via a frontend
  - netback, netfront
  - blockback, blockfront
  - there’s a PCI pass through for other kinds of devices (e.g. sound)
- backends and frontends communicate via a high level device abstraction - block class, network class, etc domains other than dom0 may be granted physical device access, securely
- virtual PCI configuration space and virtual interrupts

Safe Hardware Access requirements

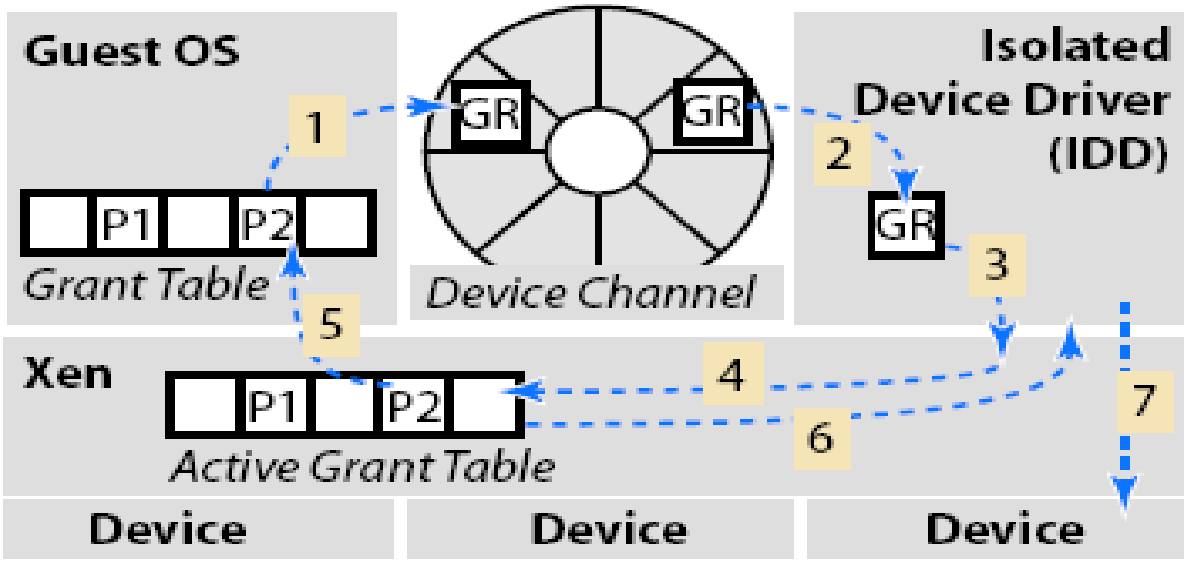
- **Requirement 1 : Driver Isolation**
  - Memory : Execute in logical fault domain
  - CPU : Schedule access to prevent excessive consumption
  - Privilege : Limit access to instruction set
- **Requirement 2: Driver – Device Isolation**
  - I/O registers : restrict access to permitted ranges
  - Interrupt : allow to mask/receive only device's interrupts
- **Requirement 3: Device Isolation**
  - Memory : Prevent DMA to arbitrary host memory
  - Other Devices: Prevent access to arbitrary other devices

Safe Hardware Model



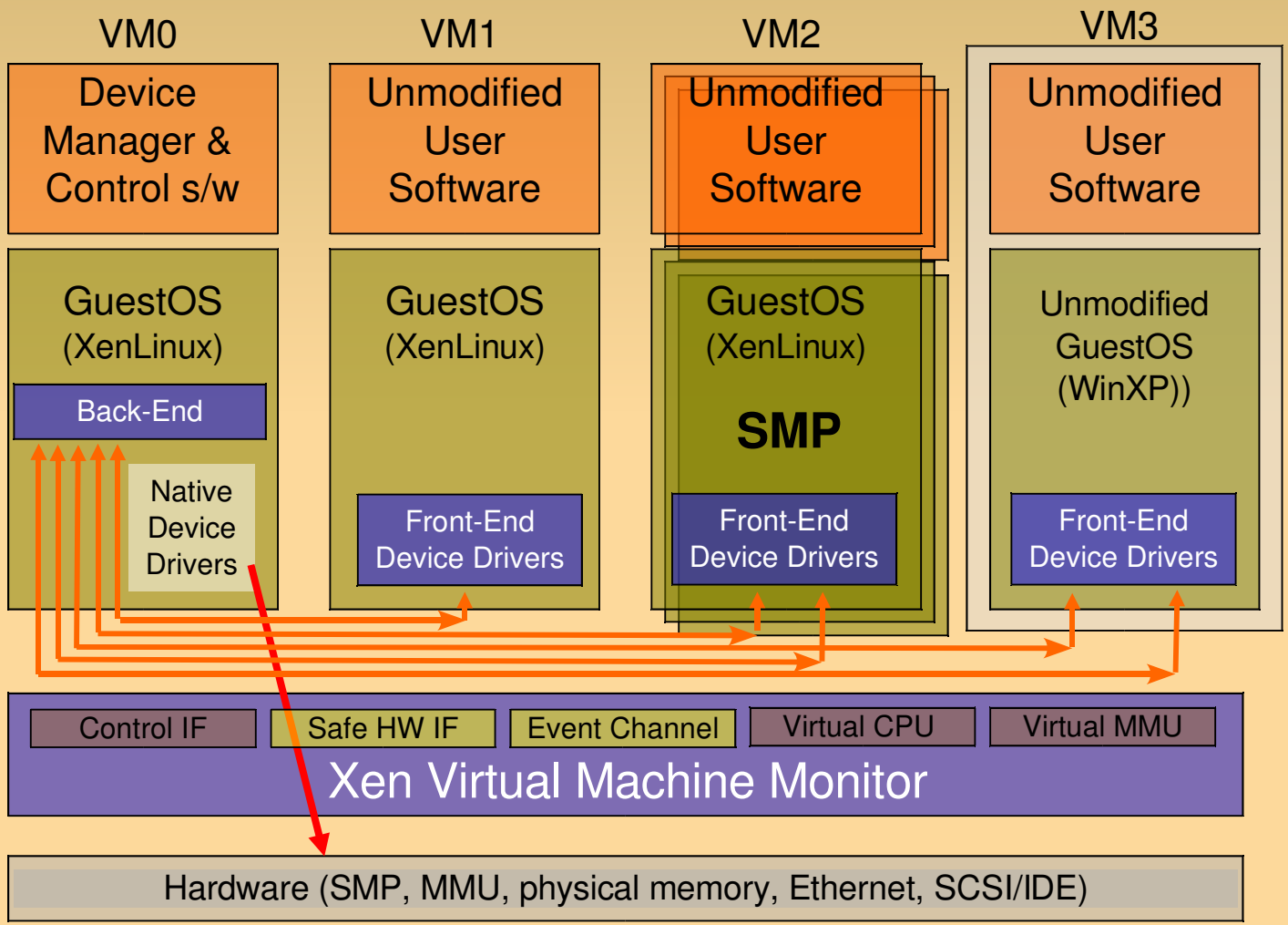
Safe Device Access

Guest Requests DMA:
1. Grant Reference for Page P2 placed on device channel
2. IDD removes GR
3. Sends pin request to Xen



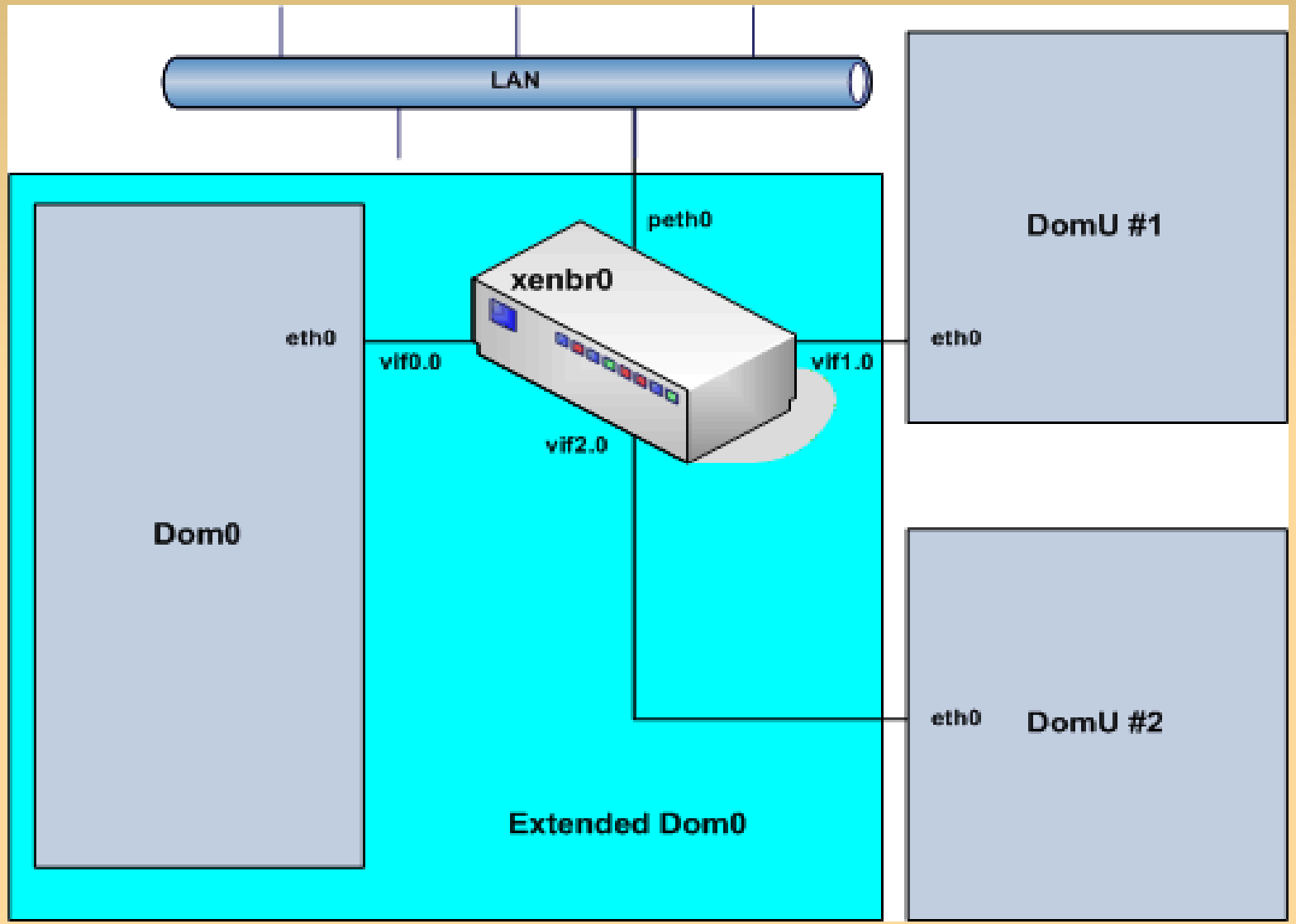
4. Xen looks up GR in active grant table
5. GR validated against Guest (if necessary)
6. Pinning is acknowledged to IDD
7. IDD sends DMA request to device

The Big Picture (Xen 3.0)



Practicality and Applicability

Building a virtual network using XEN (Ref [2])



Benefits of XEN for Trusted Computing Research

- XEN includes a facility called **vTPM** (a virtual TPM) which is supposed to emulate the full TPM. You still need to have a real TPM in place first just like other devices. But one could then create many TPM enabled boxes easily.
- XEN allows easy creation of virtual networks. Combined with the vTPM one could test 'remote attestation' implementations easily.
- VT technology allows one to run Windows/linux unmodified under the XEN hypervisor. This facility would be really helpful as one could then create hetrogeneous networks within a laptop.
- A laptop equipped with TPM / Processor supporting VT and Xen can realize most of our prototypes that we may conjure in CS599. XEN allows me to carry my virtual lab with me in my laptop.
- Paravirtualization could allow one to spawn 'thin virtual machines' on the fly because there is no driver overhead. I am currently toying with this idea and i speculate that XEN is the closest to realizing such systems.

References

1. http://www.howtoforge.com/perfect_xen_setup_debian_ubuntu
    1. Caution : Some critical material in this is out of date ! Did not apply completely to my configuration. Send me email (aviswana@usc.edu) if you want to setup XEN using my config.
2. http://www.shorewall.net/Xen.html
3. Xen and the Art of Virtualization, Paul Barham et al, SOSP 2003
4. Safe Hardware Access with the Xen Virtual Machine Monitor, Keir Fraser et al, OASIS ASPLOS 2004 workshop