

CS202 – Advanced Operating Systems

Virtualization

March 12, 2025

Virtualization

3

- One of the natural consequences of the extensibility research we discussed
- What is virtualization and what are the benefits?

Virtualization motivation

5

- Cost: multiplex multiple virtual machines on one hardware machine
 - ▣ Cloud computing, data center virtualization
 - ▣ Why not processes?
- Heterogeneity:
 - ▣ Allow one machine to support multiple OS's
 - ▣ Maintaining compatibility
- Other: security, migration, energy optimization, customization, ...

How do we virtualize?

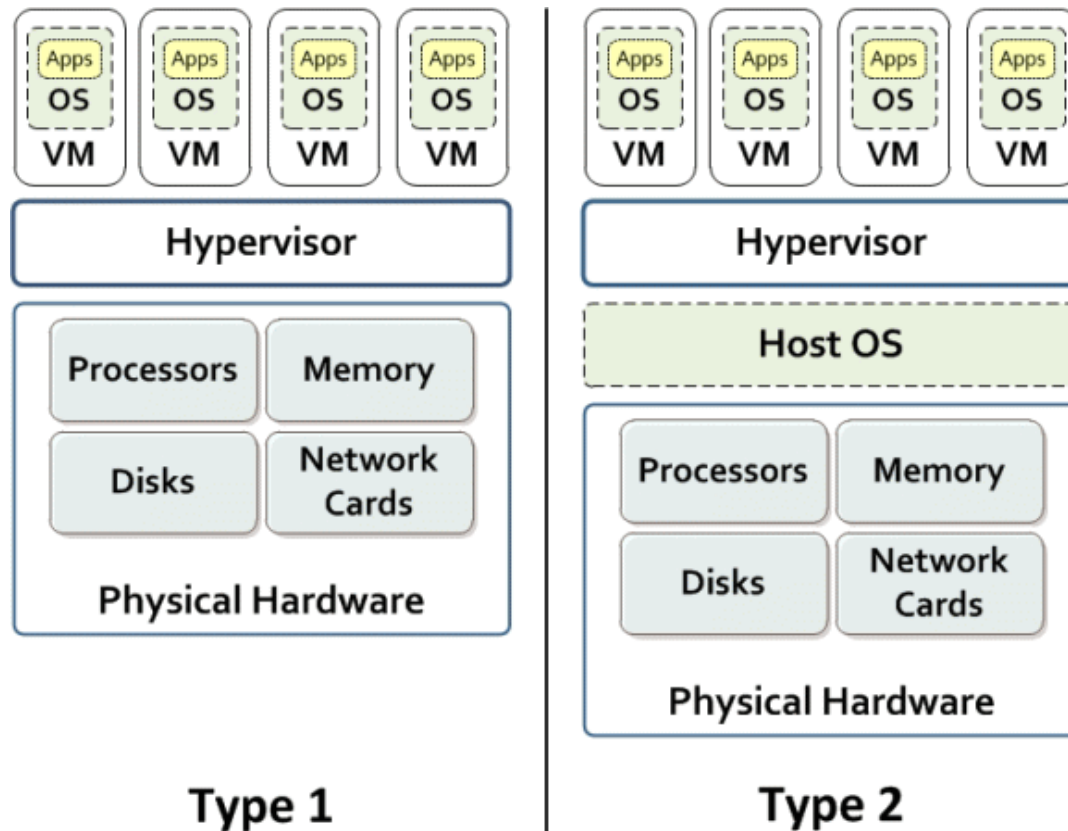
6

- Create an operating system to multiplex resources among operating systems!
 - ▣ Exports a virtual machine to the operating systems
 - ▣ Called a hypervisor or Virtual Machine Monitor (VMM)

VIRTUALIZATION MODELS

Two types of hypervisors

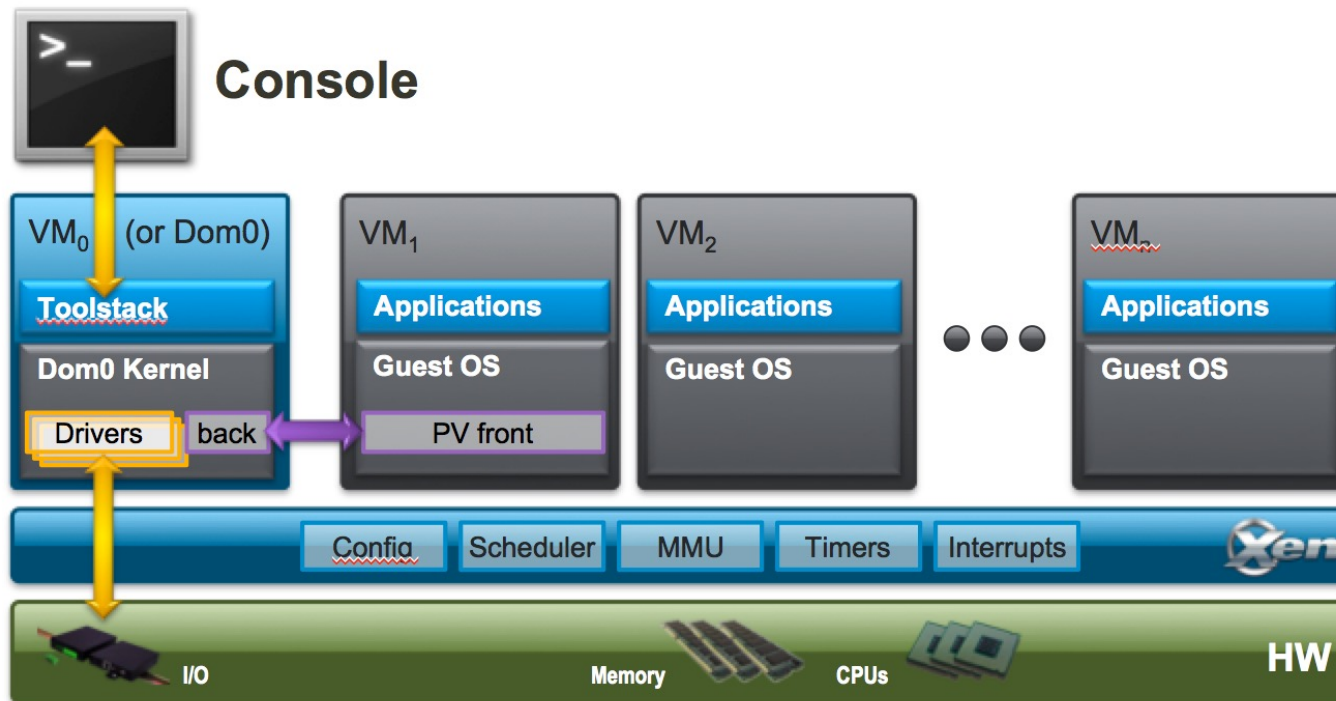
8



- Type 1: Native (bare metal)
 - ▣ Hypervisor runs on top of the bare metal machine
 - ▣ e.g., KVM
- Type 2: Hosted
 - ▣ Hypervisor is an emulator
 - ▣ e.g., VMWare, virtual box, QEMU

Hybrid organizations

9



- Some hybrids exist, e.g., Xen
 - ▣ Mostly bare metal
 - ▣ VM0/Dom0 to keep device drivers out of VMM

Stepping back – some history

10

- IBM VM 370 (1970s)
- Microkernels (late 80s/90s)
- Extensibility (90s)
- SIMOS (late 90s)
 - ▣ Eventually became VMWare (2000)
- Xen, VMWare, others (2000s)
- Ubiquitous use, Cloud computing, data centers, ...
 - ▣ Makes computing a utility

Full virtualization

11

- Idea: run guest operating systems unmodified
- However, hypervisor (type 1) is the real privileged software
- When OS executes privileged instruction, trap to hypervisor who executes it for the OS
- This can be very expensive
- Also, subject to quirks of the architecture
 - ▣ Example, x86 fails silently if some privileged instructions execute without privilege
 - E.g., popf
 - ▣ Not all security-sensitive instructions (which should be run in the hypervisor) are actually privileged

Example: Disable Interrupts

- Guest OS tries to disable interrupts
 - ▣ the instruction is trapped by the VMM which makes a note that interrupts are disabled for that virtual machine
- Interrupts arrive for that machine
 - ▣ Buffered at the VMM layer until the guest OS enables interrupts.
- Other interrupts are directed to VMs that have not disabled them

Binary translation--making full virtualization practical

13

- Use binary translation to modify OS to rewrite instructions that impact execution (silent failure)
- More aggressive translation can be used
 - ▣ Translate OS-mode instructions to equivalent VMM instructions
 - Some operations still expensive
 - Cache for future use
 - Used by VMWare ESXi and Microsoft Virtual Server
- Performance on x86 typically ~80-95% of native

Binary Translation Example

14

Guest OS Assembly

do_atomic_operation:

cli

mov eax, 1

xchg eax, [lock_addr]

test eax, eax

jnz spinlock

...

...

mov [lock_addr], 0

sti

ret

Translated Assembly

do_atomic_operation:

call [vmm_disable_interrupts]

mov eax, 1

xchg eax, [lock_addr]

test eax, eax

jnz spinlock

...

...

mov [lock_addr], 0

call [vmm_enable_interrupts]

ret

Paravirtualization

15

- Modify the OS to make it aware of the hypervisor
 - ▣ Can avoid the tricky features
 - ▣ Aware of the fact it is virtualized
 - Can implement optimizations
- Comparison to binary translation?
- Amount of code change?
 - ▣ 1.36% of Linux, 0.04% for Windows

Hardware supported virtualization (Intel VT-x, AMD-V)

17

- Hardware support for virtualization
- Makes implementing VMMs much simpler
- Streamlines communication between VM and OS
- Removes the need for paravirtualization/binary translation
 - ▣ Although some paravirtualization may be leveraged to improve performance
- **Extended Page Table** Switching: Support for shadow page tables – used in LXDs

IMPLEMENTATION

What needs to be done?

19

- Virtualize hardware
 - ▣ Memory hierarchy
 - ▣ CPUs
 - ▣ Devices
- Implement data and control transfer between guests and hypervisor
- We'll cover this by example – Xen paper
 - ▣ Slides modified from presentation by Jianmin Chen

Xen



- Design principles:
 - ▣ Unmodified applications: essential
 - ▣ Full-blown multi-tasking OSeS: essential
 - ▣ Paravirtualization: necessary for performance and isolation

Xen

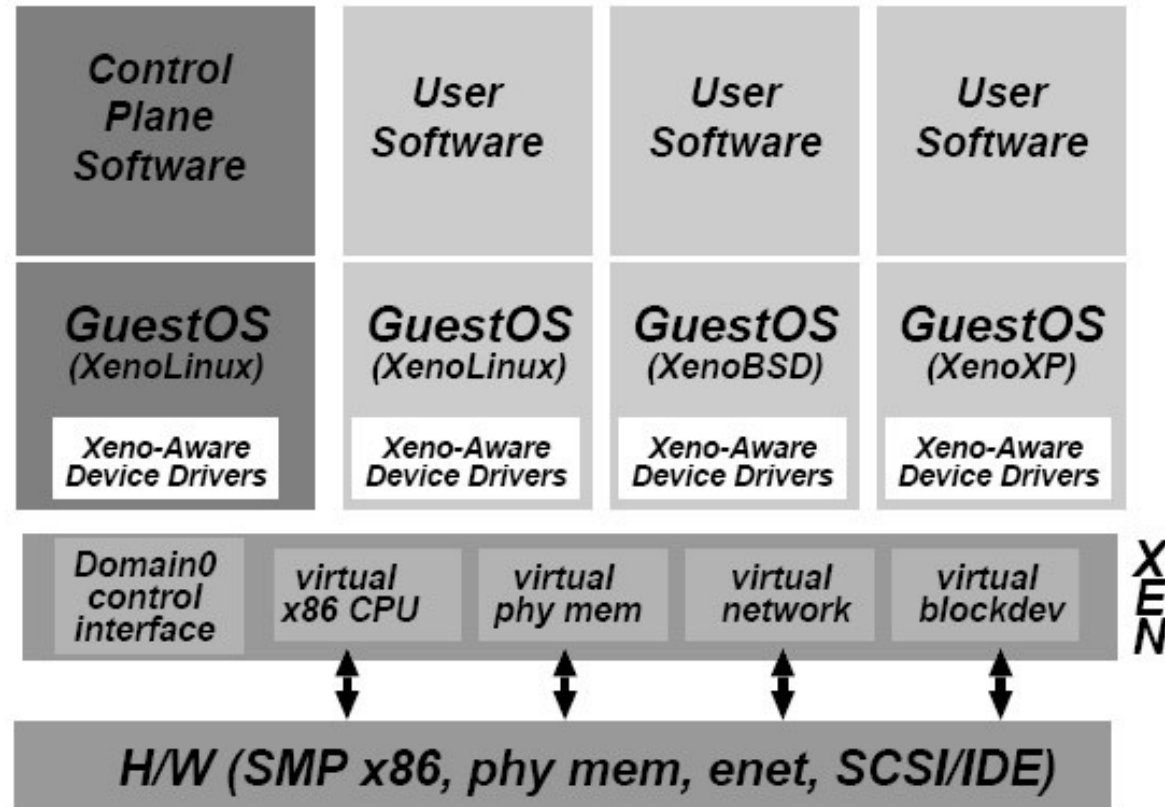


Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including *Domain0* running control software in a XenoLinux environment.

Implementation summary

22

Memory Management	
Segmentation	Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.
Paging	Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated discontinuous machine pages.
CPU	
Protection	Guest OS must run at a lower privilege level than Xen.
Exceptions	Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.
System Calls	Guest OS may install a 'fast' handler for system calls, allowing direct calls from an application into its guest OS and avoiding indirecting through Xen on every call.
Interrupts	Hardware interrupts are replaced with a lightweight event system.
Time	Each guest OS has a timer interface and is aware of both 'real' and 'virtual' time.
Device I/O	
Network, Disk, etc.	Virtual devices are elegant and simple to access. Data is transferred using asynchronous I/O rings. An event mechanism replaces hardware interrupts for notifications.

Table 1: The paravirtualized x86 interface.

Implementation summary

23

Memory Management Segmentation	Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.
Paging	Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated discontinuous machine pages.
CPU Protection Exceptions	Guest OS must run at a lower privilege level than Xen. Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.
System Calls	Guest OS may install a 'fast' handler for system calls, allowing direct calls from an application into its guest OS and avoiding indirecting through Xen on every call.
Interrupts Time	Hardware interrupts are replaced with a lightweight event system. Each guest OS has a timer interface and is aware of both 'real' and 'virtual' time.
Device I/O Network, Disk, etc.	Virtual devices are elegant and simple to access. Data is transferred using asynchronous I/O rings. An event mechanism replaces hardware interrupts for notifications.

Table 1: The paravirtualized x86 interface.

Xen VM interface: Memory

- How to handle memory management when the OS is not running in privileged mode?
 - ▣ Guest OSes cannot manage the memory devices, TLBs, and MMUs
 - ▣ E.g., How are page tables managed? How are page faults processed?

Xen VM interface: Memory



- Memory management
 - ▣ Guest cannot install highest privilege level segment descriptors; top end of linear address space is not accessible
 - ▣ Guest has direct (not trapped) read access to hardware page tables; writes are trapped and handled by the VMM
 - ▣ Physical memory presented to guest is not necessarily contiguous

Two Layers of Virtualization

Virtual address →
physical address

Physical address →
machine address

Host OS's
View of RAM

Guest OS's
View of RAM

0xFFFF

0xFFFFFFFF

Guest App's
View of RAM

0xFF

0x00

Page 3

Page 2

Page 1

Page 0

Page 0

Page 1

Page 3

Page 2

Page 2

Page 0

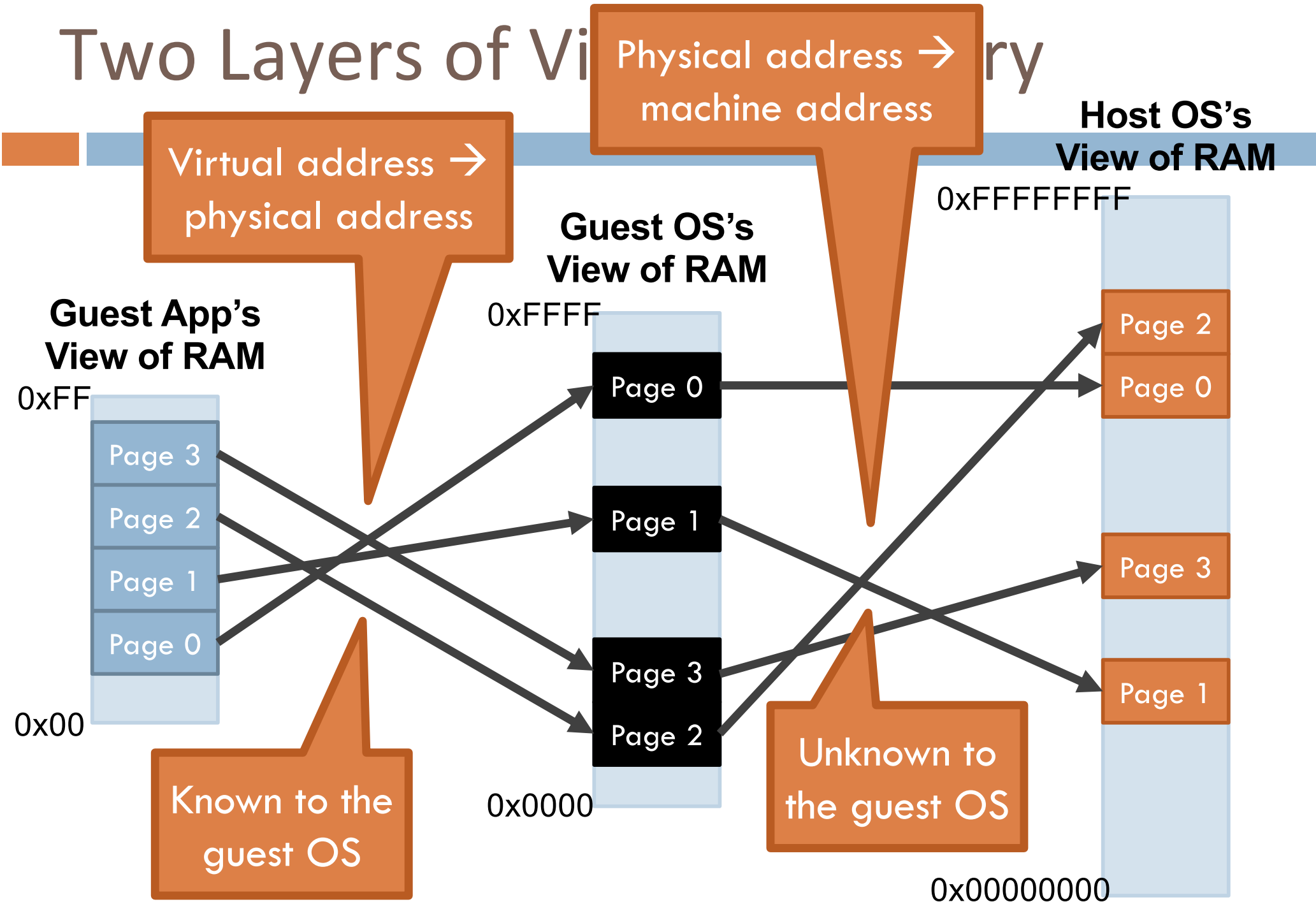
Page 3

Page 1

Known to the
guest OS

Unknown to
the guest OS

0x00000000



Guest's Page Tables Are Invalid

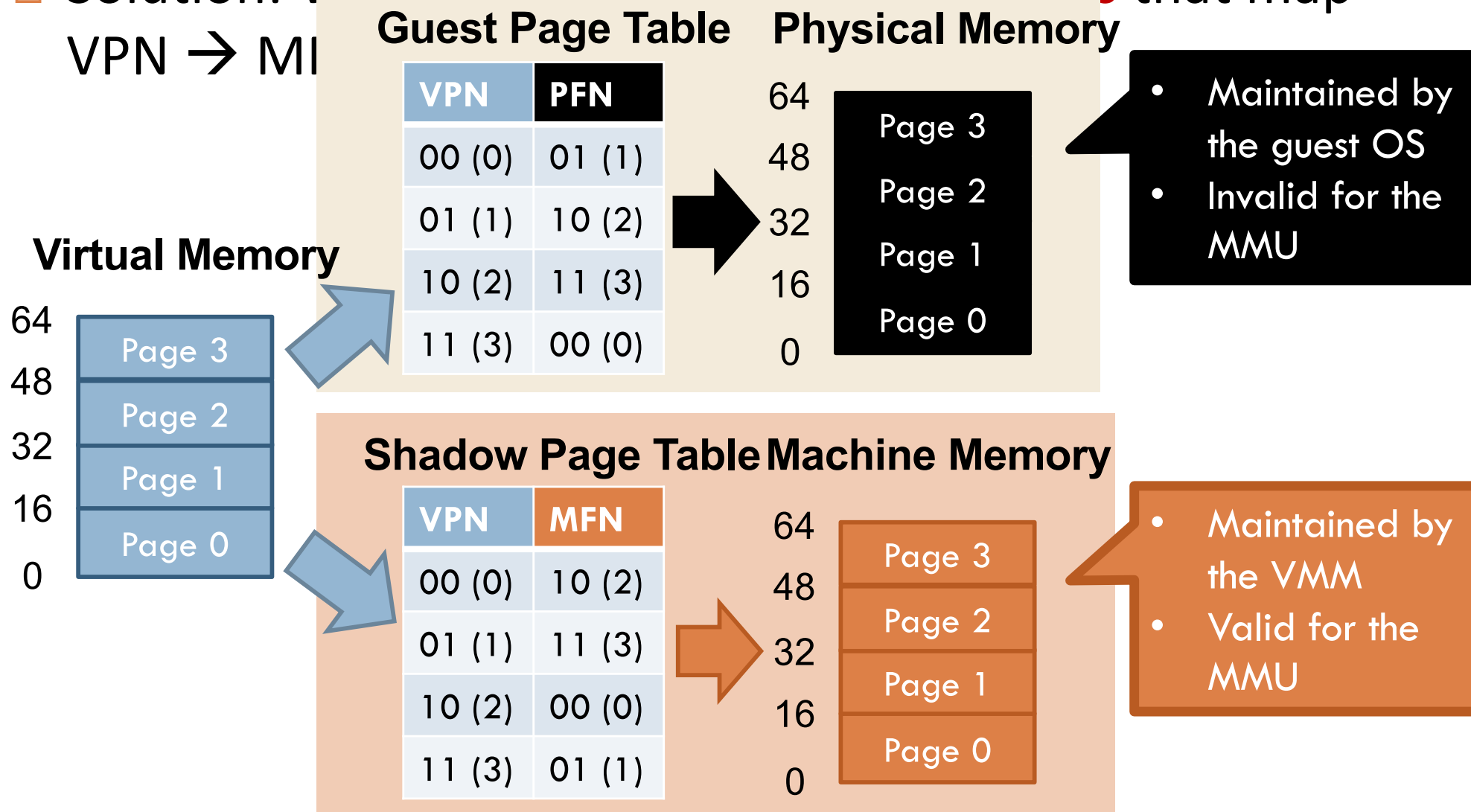
27

- Guest OS page tables map **virtual page numbers (VPNs)** to **physical frame numbers (PFNs)**
- Problem: the guest is virtualized, doesn't actually know the true PFNs
 - ▣ The true location is the **machine frame number (MFN)**
 - ▣ MFNs are known to the VMM and the host OS
- Guest page tables cannot be installed in *cr3*
 - ▣ Map VPNs to PFNs, but the PFNs are incorrect
- How can the MMU translate addresses used by the guest (VPNs) to MFNs?

Shadow Page Tables

28

- Solution: VMM creates **shadow page tables** that map VPN → MI



Building Shadow Tables

29

- Problem: how can the VMM maintain consistent shadow pages tables?
 - ▣ The guest OS may modify its page tables at any time
 - ▣ Modifying the tables is a simple memory write, not a privileged instruction
 - Thus, no helpful CPU exceptions :(
- Solution: mark the hardware pages containing the guest's tables as **read-only**
 - ▣ If the guest updates a table, an exception is generated
 - ▣ VMM catches the exception, examines the faulting write, updates the shadow table

Xen VM interface: I/O



- How is I/O performed when the device driver is not in the guest OS?
 - ▣ E.g., How are requests handled? How is data communicated to the guest efficiently?

Xen VM interface: I/O



- I/O

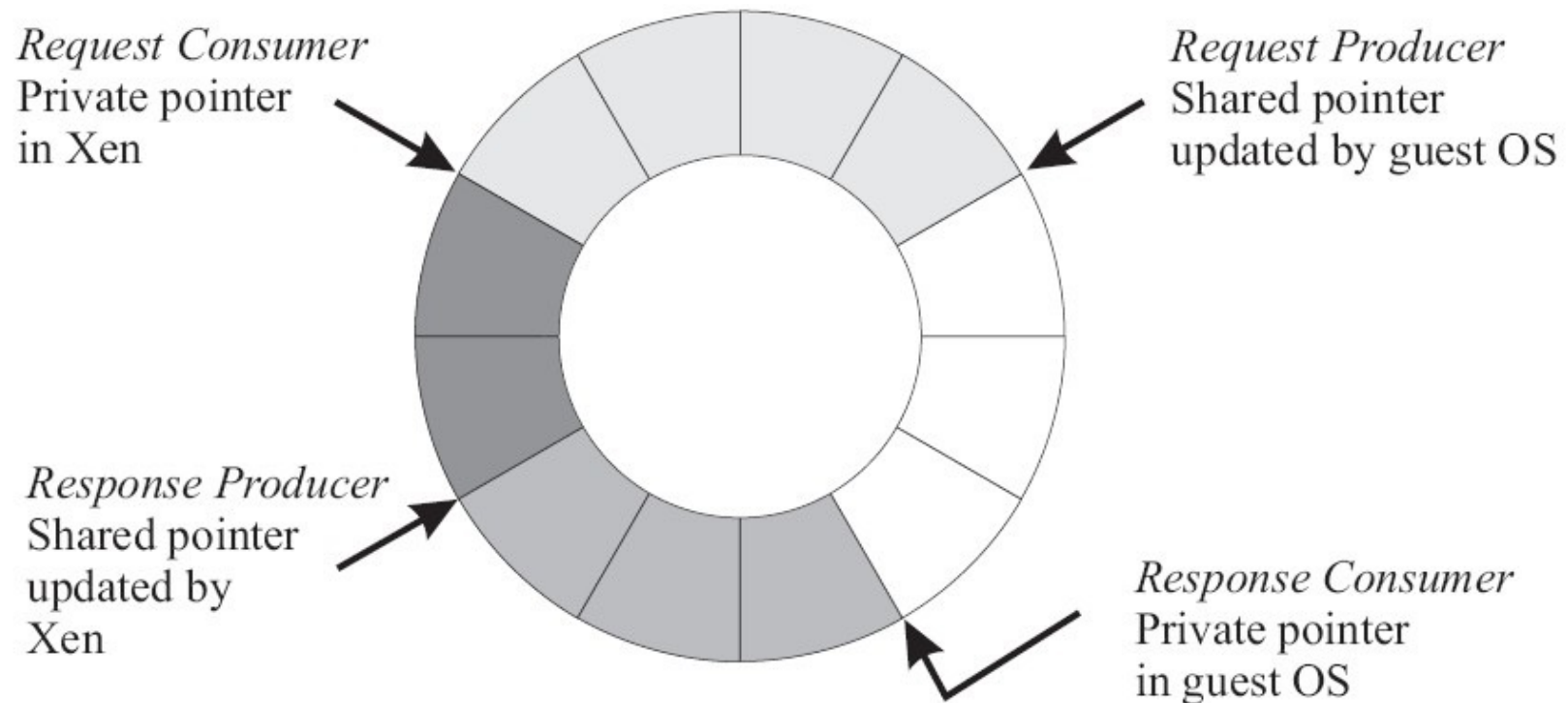
- ▣ Virtual devices exposed as asynchronous I/O rings to guests
- ▣ Event notification replaces interrupts





Details: I/O Paravirtualization

- Xen does not emulate hardware devices
 - ▣ Device drivers are run in domain 0 (Xen's VM)
 - Perform I/O requests on behalf of guest OSes
 - ▣ Exposes device abstractions for simplicity and performance
 - ▣ I/O data transferred to/from guest via Xen using shared-memory buffers
 - ▣ Virtualized interrupts: light-weight event delivery mechanism from Xen-guest
 - Update a bitmap in shared memory
 - Optional call-back handlers registered by OS

Details: I/O

□ I/O Descriptor Ring:



-  **Request queue** - Descriptors queued by the VM but not yet accepted by Xen
-  **Outstanding descriptors** - Descriptor slots awaiting a response from Xen
-  **Response queue** - Descriptors returned by Xen in response to serviced requests
-  **Unused descriptors**

Data Transfer: Descriptor Ring



- Descriptor rings are allocated by a domain (guest) and accessible from Xen domain 0
- Descriptors do not contain I/O data; instead, point to data buffers also allocated by a guest
 - ▣ Facilitate zero-copy transfers of I/O data into a domain

Network Virtualization



- Each domain has 1+ network interfaces (VIFs)
 - ▣ Each VIF has 2 I/O rings (send, receive)
 - ▣ Each direction also has rules of the form (<pattern>,<action>) that are inserted by domain 0 (management)
- Xen models a virtual firewall+router (VFR) to which all domain VIFs connect

Network Virtualization

□ Packet transmission:

- ▣ Guest adds request descriptor to I/O ring
- ▣ Xen copies packet header, applies matching filter rules
 - E.g. change header IP source address for NAT
 - No change to payload; pages with payload must be pinned to physical memory until DMA to physical NIC for transmission is complete
- ▣ Round-robin packet scheduler

Network Virtualization

□ Packet reception:

- ▣ Xen applies pattern-matching rules to determine destination VIF
- ▣ Guest OS is required to exchange unused page frame for each packet received
 - Xen exchanges packet buffer for page frame in VIF's receive ring
 - If no receive frame is available, the packet is dropped
 - Avoids Xen-guest copies; requires page-aligned receive buffers to be queued at VIF's receive ring

Evaluation

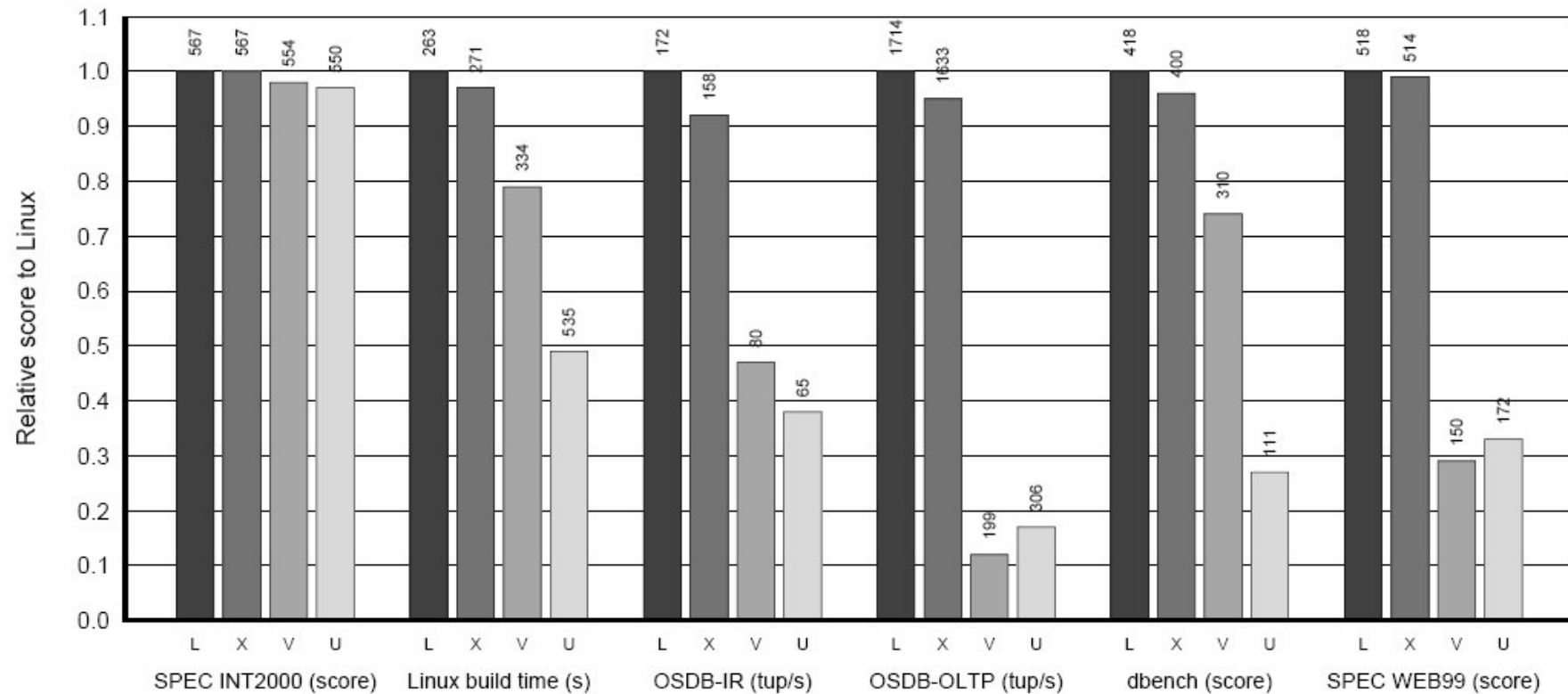


Figure 3: Relative performance of native Linux (L), XenLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).

Conclusions

70

- Brief look at **virtualization**
 - ▣ Run multiple OSes on one machine efficiently
 - ▣ Enables improved machine utilization
- Virtualization architectures can vary
 - ▣ Different security, performance, usability trade-offs
- Examined the **Xen hypervisor system**
 - ▣ Hybrid virtualization using a bare-metal hypervisor (for security) and a system VM (domain 0 for I/O)
 - ▣ **Memory**: Guest management that achieves isolation
 - ▣ **I/O**: paravirtualization of devices

Questions

71

