Platform Virtualization: Model, Challenges and Approaches

Fangzhou Jiao, Yuan Luo
School of Informatics and Computing
Indiana University
{fjiao, yuanluo}@indiana.edu

Outlines

- Virtualization Overview
- Virtualization Models
- Formal Definition of Virtualization:
  - Popek and Goldberg's Virtualization Requirements
- Overview of x86 Virtualization: Obstacles and Solutions
- Virtualization "on the move"

Virtualization Overview

- What’s Virtualization?
- Why Virtualization?

What's Virtualization

- Virtualization is a technology that combines or divides (computing) resources to present one or many operating environments.

Virtualization: What is it, really?

- Virtualization using methodologies such as,
  - Hardware/software partitioning (or aggregation)
  - Partial or complete machine simulation
  - Emulation (again, can be partial or complete)
  - Time-sharing (in fact, sharing in general)
  - In general, can be M-to-N mapping (M “real” resources, N “virtual” resources)
  - Examples: VM (M-N), Grid Computing (M-1) , Multitasking (1-N)

Virtualization Everywhere

Driver

Gas Machine

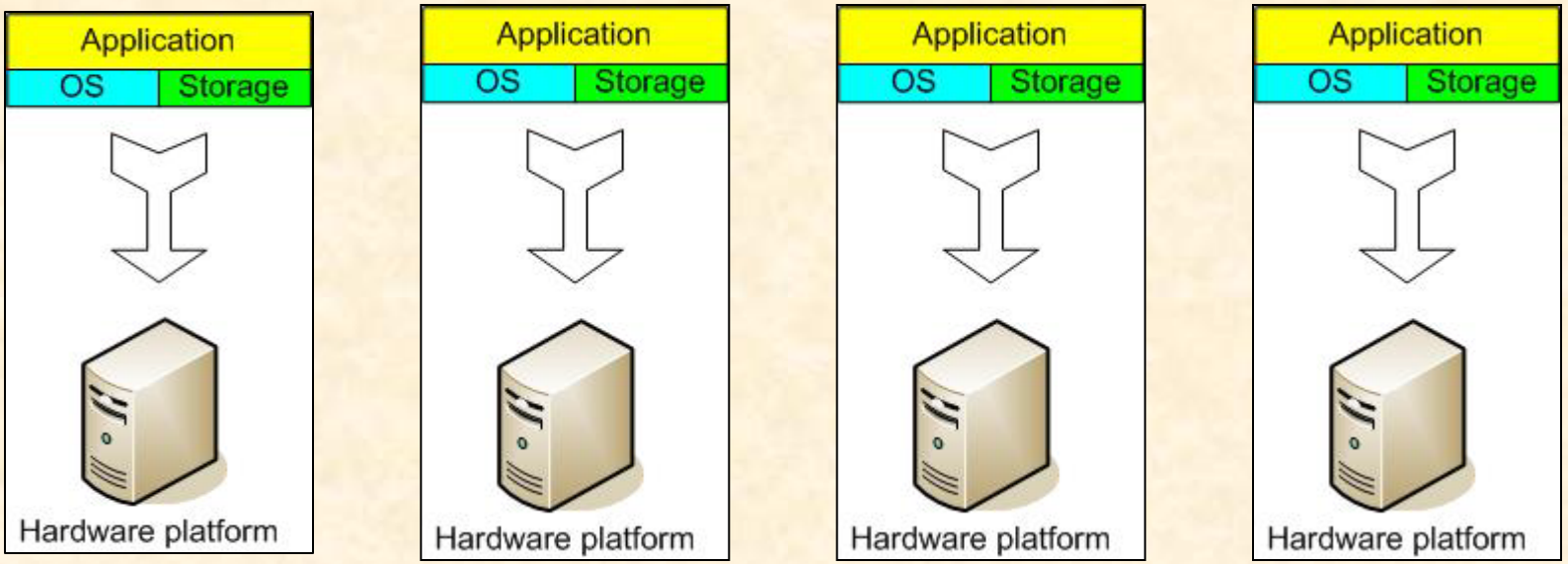
Gas Station Attendants

Gas Pump

Virtualization: Any demands?

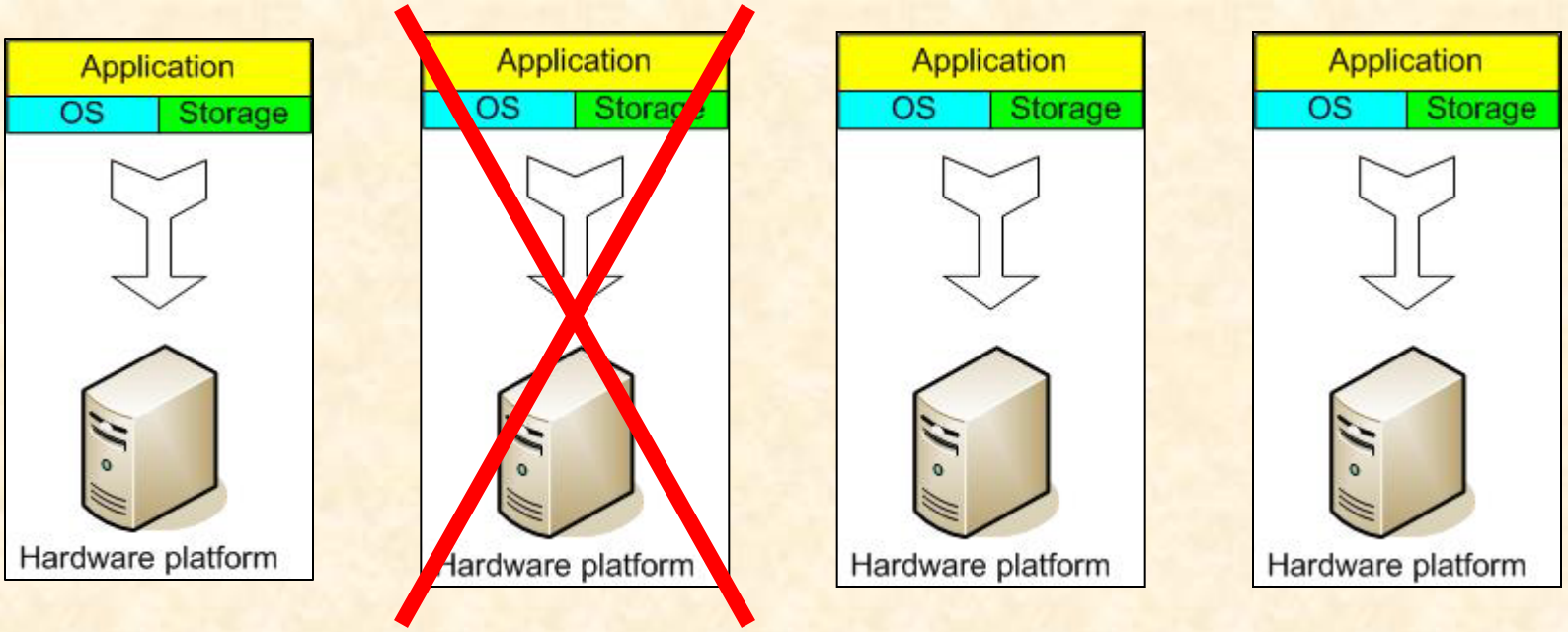
- Server consolidation
- Application Consolidation
- Sandboxing
- Multiple execution environments
- Virtual hardware
- Debugging
- Software migration (Mobility)
- Appliance (software)
- Testing/Quality Assurance

The Traditional Server Concept



Web Server	App Server	DB Server	EMail
Windows	Linux	Linux	Windows
IIS	Glassfish	MySQL	Exchange

And if something goes wrong ...



Web Server

Windows

IIS

App Server

DOWN!

DB Server

Linux

MySQL

EMail

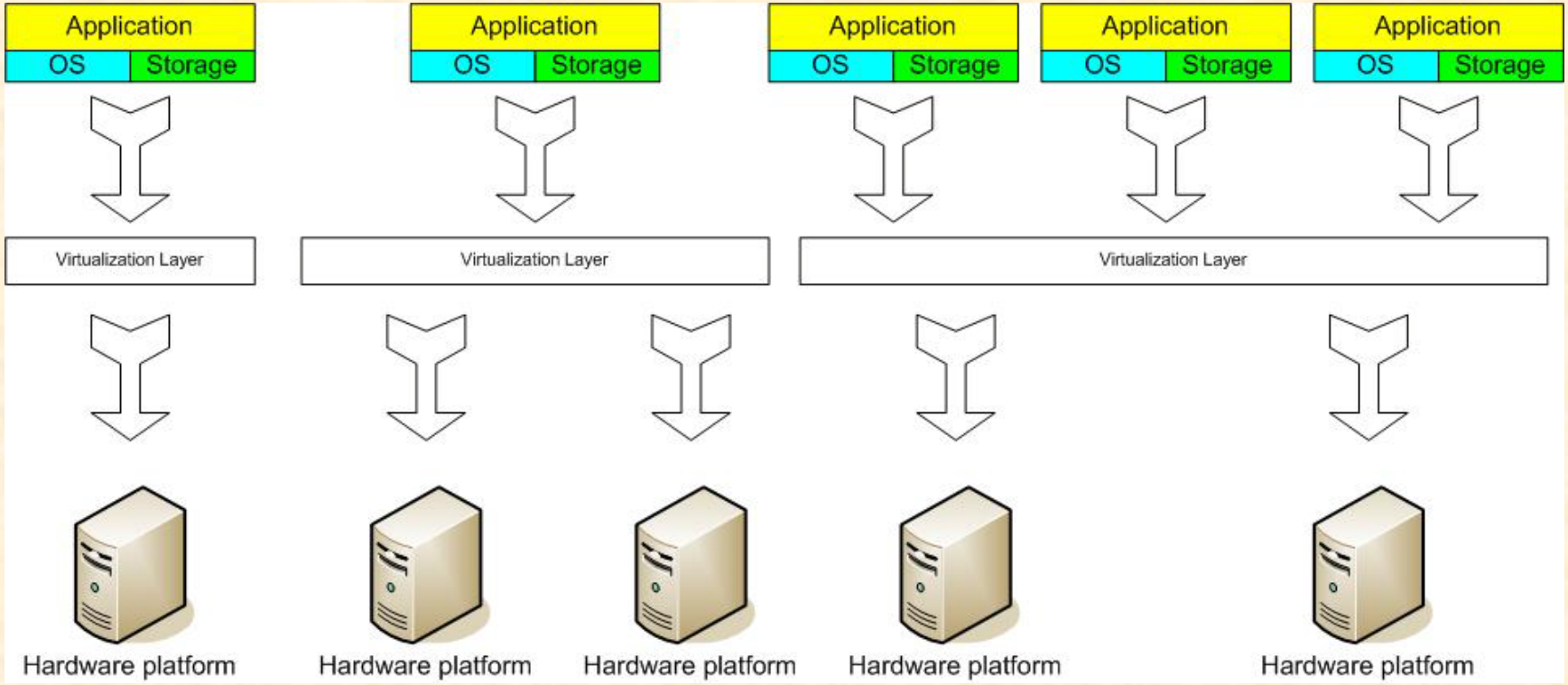
Windows

Exchange

The Traditional Server Concept

- Pros
  - Easy to conceptualize
  - Fairly easy to deploy
  - Easy to backup
  - Virtually any application/service can be run from this type of setup
- Cons
  - Expensive to acquire and maintain hardware
  - Not very scalable
  - Difficult to replicate
  - Redundancy is difficult to implement
  - In many cases, processor is under-utilized

The Virtual Server Concept



Virtual Machine Monitor (VMM) layer between *Guest OS* and hardware

The Virtual Server Concept

- Pros
  - Resource pooling
  - Highly redundant
  - Highly available
  - Rapidly deploy new servers
  - Easy to deploy
  - Reconfigurable while services are running
  - Optimizes physical resources by doing more with less
- Cons
  - Slightly harder to conceptualize
  - Slightly more costly (must buy hardware, OS, Apps, and now the abstraction layer)

BACK

Virtualization Abstraction Levels

- Instruction Set Architecture (ISA)
  - Emulate the ISA in software
    - Interprets, translates to host ISA (if required)
    - Device abstractions implemented in software
    - Inefficient
  - Optimizations: Caching, Code reorganization
  - Applications: Debugging, Teaching, multiple OS
- Hardware Abstraction Layer (HAL)
  - Between “real machine” and “emulator” (maps to real hardware)
  - Handling non-virtualizable architectures (code scanning, dynamic instruction rewriting)
  - Applications: Fast and usable, virtual hardware, consolidation, migration.

Virtualization Abstraction Levels cont'd

- Operating System Level
  - Virtualized SysCall Interface (may be same)
  - May or may not provide all the device abstractions
  - Easy to manipulate (create, configure, destroy)
- Library (user-level API) Level
  - Presents a different subsystem API to application
  - Complex implementation, if kernel API is limited
  - User-level device drivers
- Application (Programming Language) Level
  - Virtual architecture (ISA, registers, memory, …)
  - Platform-independence (highly portable)
  - Less control on the system (extremely high-level)

Overall Picture

	ISA	HAL	OS	Library	PL
Performance	*	****	****	***	**
Flexibility	****	***	**	**	**
Ease of Implementation	**	*	***	**	**
Degree of Isolation	***	****	**	**	***

(more stars are better)

Instruction Set Architecture Level Virtualization

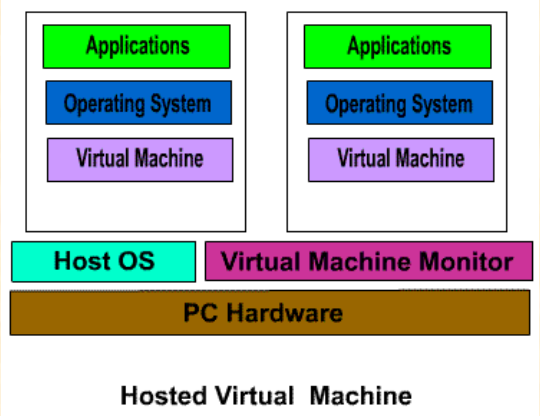
- Technologies
  - Emulation: Translates guest ISA to native ISA
  - Translation Cache: Optimizes emulation by making use of similar recent instructions
  - Code rearrangement
  - Speculative scheduling (alias hardware)
- Issues
  - Efficient Exception handling
  - Self-modifying code

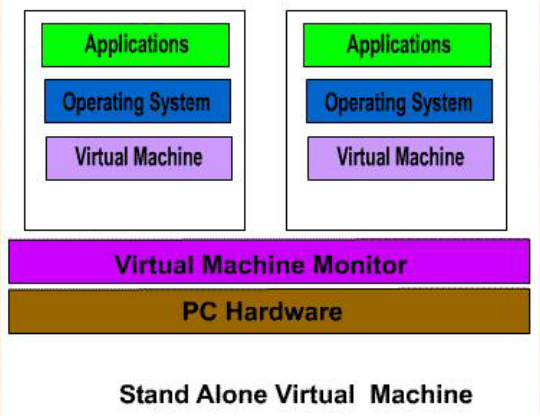
ISA Level Virtualization: Examples

- Bochs: Open source x86 emulator
  - Emulates whole PC environment
    - x86 processor and most of the hardware (VGA, disk, keyboard, mouse, …)
    - Custom BIOS, emulation of power-up, reboot
    - Host ISAs: x86, PowerPC, Alpha, Sun, and MIPS
- Crusoe (Transmeta)
  - “Code morphing engine” – dynamic x86 emulator on VLIW processor
  - 16 MB memory as “translation cache”
  - Shadow registers: Enables easy exception handling
- QEMU:
  - User space only, and full system emulation
  - Multiple target ISAs: x86, ARM, PowerPC, Sparc
  - Full-software MMU and simulation(using mmap() system call)
  - Dynamic Translation

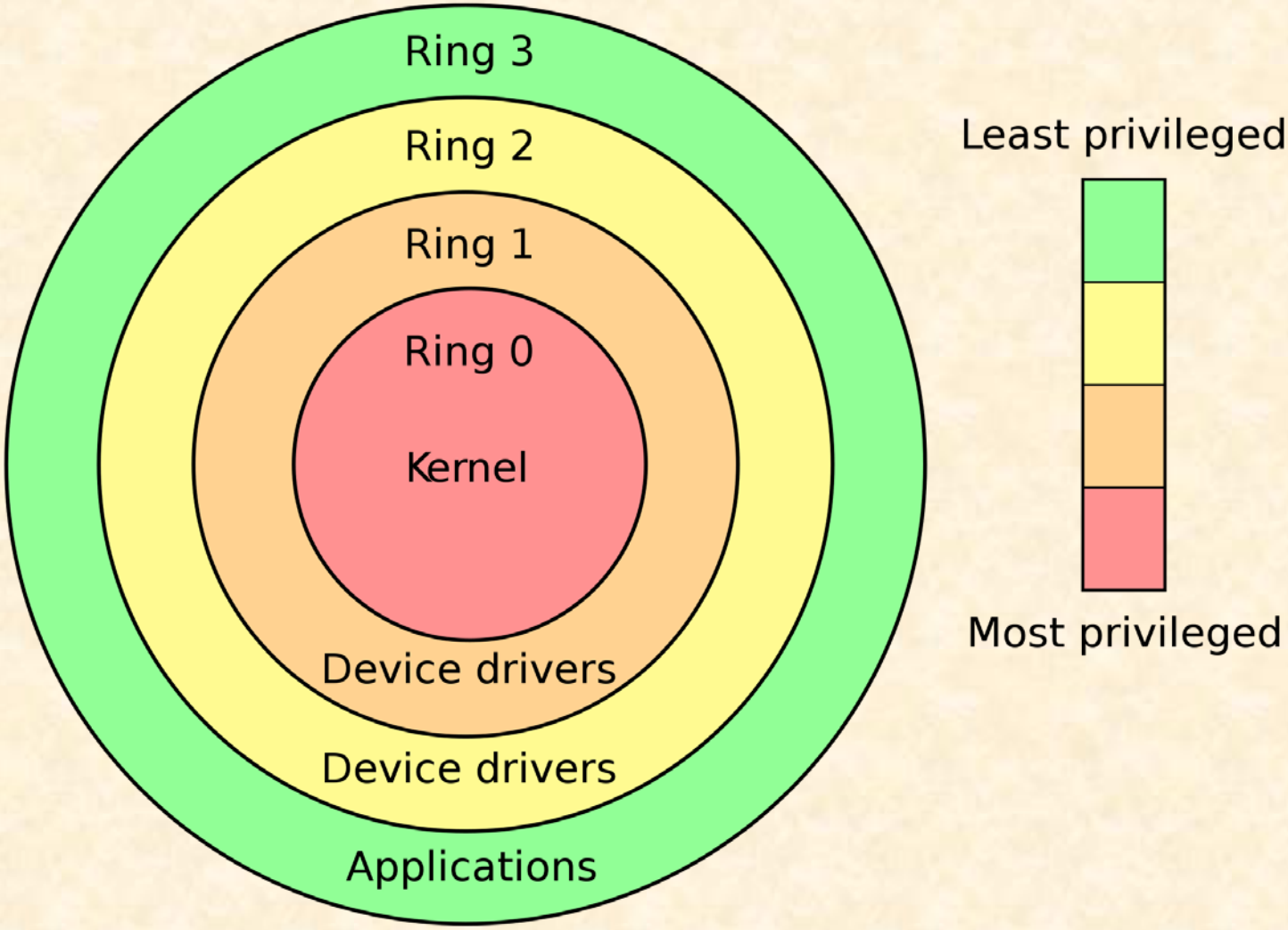
HAL Virtualization Techniques

- Standalone vs. Hosted
  - Drivers
  - Host and VMM worlds
  - I/O
- Protection Rings
  - Multilevel privilege domains
- Handling “silent” fails
  - Scan code and insert/replace artificial traps





Intel x86 Protection Rings



Paravirtualization

- Traditional architectures do not scale
  - Interrupt handling
  - Memory management
  - World switching
- Virtualized architecture interface
  - Much simpler architectural interface
  - Virtual I/O and CPU instructions, registers, …
- Portability is lost

Examples

- Denali
- Xen

Programming Level Virtualization

- Java Virtual Machine (JVM)
  - Executes Java byte code (virtual instructions)
  - Provides the implementation for the instruction set interpreter (or JIT compiler)
  - Provides code verification, garbage collection, etc
  - Hardware access through underlying OS
- JVM Architecture
  - Stack-based architecture
  - No MMU
  - Virtual hardware: PC, register-set, heap, method (code) areas
  - Rich instruction set
    - Direct object manipulation, type conversion, exception throws
- Provides a runtime environment through JRE
- Other Examples: .NET CLI, Parrot (PERL 6)

BACK

Virtual Machine Implementation: Issues

- Only one “bare” machine interface to **guest system**!
- What architectures are considered as virtualizable?

> “A virtualizable architecture allows any instruction inspecting/modifying machine state to be trapped when executed in any but the most privileged mode”
>
> - Popek & Goldberg (1974)

- What about x86 machines?

BACK

x86 Virtualization: Obstacles and Solutions

Processor Virtualization

- Goal of processor virtualization:

Processor Virtualization

- Goal of processor virtualization:
- **Identical** environment and behavior with physical machine.
- **Efficiency:** most instructions should be directly executed by processor.
- **Control to resource:** i. Isolation of resources; ii. Hypervisor can gain control for allocated resources if needed.

Processor Virtualization

- These goals needs instruction set support.
- What features of modern processors can contribute to these goals of virtualization?

Processor Virtualization

- **Privilege Levels:**
- Hypervisor should run in higher privilege level than guest OS.
- **Memory protection system:**
- Virtual memory: offered address isolation
- ***And more?***

Processor Virtualization

- First we define the “sensitive instructions”

Processor Virtualization

- First we define the “sensitive instructions”
- **Sensitive instructions:**
- *Instructions which can interfere the global status of system.*

Processor Virtualization

1. Instructions which changes or references the state of VM or machine.
2. Instructions which changes or reads the sensitive register or memory location.
3. Instructions which changes or references the memory protection system.
4. Instructions which execution results depends on the state of machine, or memory protection system, or privilege level.

Processor Virtualization

- Popek and Goldberg's virtualization requirements:
  - ***If all the sensitive instructions were privileged, the instruction set is ready for virtualization.***
- What about x86 instructions?

Obstacles of x86 Virtualization

- Instructions involved sensitive registers:
- SGDT, SIDT, SLDT
- SMSW
- PUSHF, POPF

Obstacles of x86 Virtualization

- Sensitive Registers
  - GDTR (Global descriptor table register)
  - IDTR (Interrupt descriptor table register)
  - LDTR (Local descriptor table register)
- Store segment descriptors which containing base address, type, length and access rights for memory segments.

Obstacles of x86 Virtualization

- GDTR, IDTR, LDTR
  - Typically, x86 processors only have one register of these three.
  - But each virtual machine needs to keep the segmentation information of themselves.
- When all guest OS sharing one segment register, the guest OS may see the segment descriptor of other guest OS or hypervisor, since the segment register only can be loaded in CPL 0.
- MSW register similar with this.

Obstacles of x86 Virtualization

- Instructions: SMSW, PUSHF, POPF
- SMSW, PUSHF:
  - Executing these two instructions involved checking the content of sensitive registers (CR0, EFLAGS)
  - So the guest OS is able to found it was running in virtual machine.

Obstacles of x86 Virtualization

- Ambiguous result in different privilege ring:
  - POPF (Push/pop EFLAGS)
- Different bits in EFLAGS have different write privilege levels.
- Guest OS usually not executing in CPL 0, so POPF will not raise exception not trapped, only generated results without modify some bits in EFLAGS.

Obstacles of x86 Virtualization

- Instructions which result relies on memory protection system or privilege level:
  - LAR, LSL, VERR, VERW
  - POP, PUSH
  - CALL, JUMP, INT, RET
  - MOVE

Obstacles of x86 Virtualization

- When a guest OS executing such instructions, it is possible for discovering itself was not running in CPL=0, thus the guest OS will not execute properly.

Instruction	Sensitive	Privileged	Violated Rules	Source	Destination	Semantic	Explanation
SGDT	Y	N	3B	[Register] GDTR	Memory	Store	The registers GDTR, LDTR, IDTR, and CR0, EFLAGS should be maintained as different registers for different VMs. So the operations to these registers could lead to result that one VM modified other VM's state.
SIDT	Y	N	3B	[Register] IDTR	Memory / Register	Store	
SLDT	Y	N	3B	[Register] LDTR	Memory	Store	
SMSW	Y	N	3B	[Register] CR0	Memory / Register	Store	
SMSW	Y	N	3B	[Register] CR0	Memory / Register	Store	Still involved registers should be per-VM. Also these instructions can be executed when have no enough privilege, but the whole system was left as an ambiguous state.
PUSHF	Y	Ambiguous	3B	[Register] EFLAGS	Stack	Push stack	
POPF	Y	Ambiguous	3B	Stack	[Register] EFLAGS	Pop stack	
LAR	Y	N	3C	Segment Descriptor	[Register] GP	Load (Access Rights)	Execution of these instructions involved checking the CPL. The result relies on the CPL. In guest OS the CPL could be different with the OS expected, so there would be trouble.
LSL	Y	N	3C	Segment Descriptor	[Register] GP	Load (Segment Limit)	
VERR	Y	N	3C	Segment Descriptor	N/A	Verify (Readable)	
VERW	Y	N	3C	Segment Descriptor	N/A	Verify (Writable)	
POP	Y	N	3C	Stack	[Register] GP	Pop stack	
PUSH	Y	N	3C	Stack	[Register] GP	Push stack	
CALL	Y	N	3C			Call subroutine	Only cross privilege level calls are sensitive. Croo privilege level calls involved a lot of CPL check.
JMP	Y	N	3C			Jump to	
INT	Y	N	3C			Interrupt	
RET	Y	N	3C			Return	
STR	Y	N	3C	Task Register [Segment Selector]	[Register] GP/Memory	Store	When CS or SS register involved, the CPL bits in CS or SS can be referenced or updated, and the guest OS can found inconsistency in CPL.
MOVE	Y	N	3C			Move	

Obstacles of x86 Virtualization: Memory Management

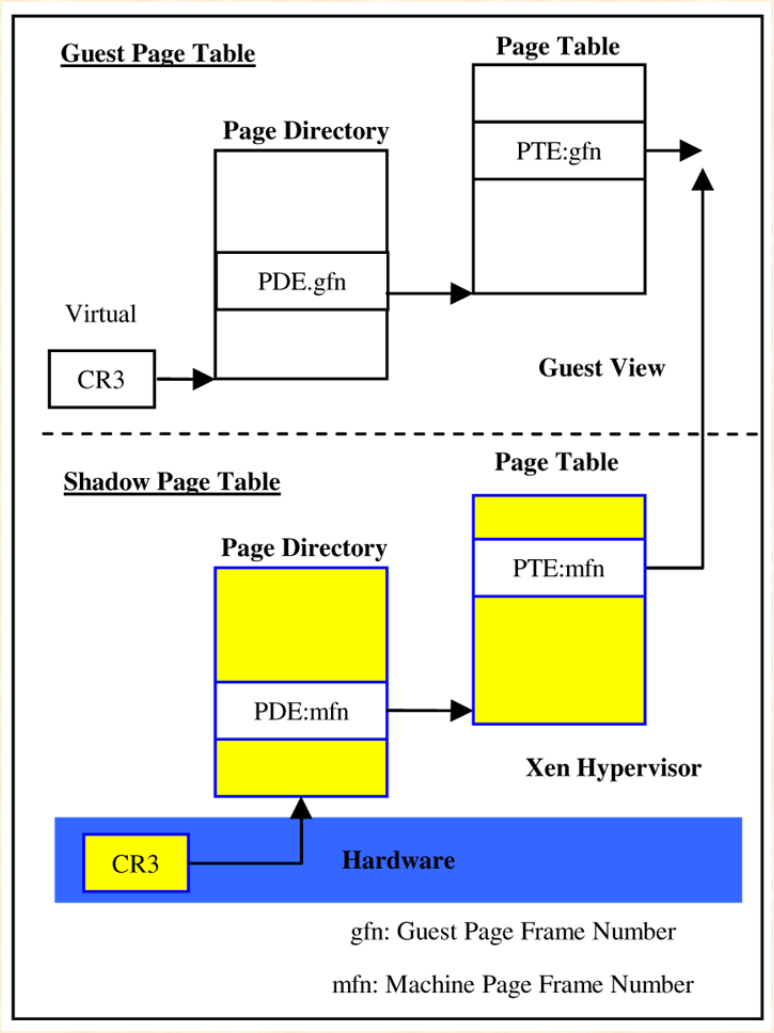
- For each virtual machine, the address should be maintained independently and isolated.

Obstacles of x86 Virtualization: Memory Management

- So in virtualized system, two level address translation were needed:
  - **Guest Virtual Address** to **Guest Physical Address**
  - **Guest Physical Address** to **Native Physical Address**

Obstacles of x86 Virtualization: Memory Management

- Traditional solution: Shadow page tables
- MMU was not aware of the exist of shadow page table
- So we need generate page fault to update the TLB to keep it updated with both page tables



Obstacles of x86 Virtualization: Memory Management

- Shadow Page Table and Guest Page Table

	Guest Page Table	**Shadow Page Table**
Address Translation	Guest Virtual Address to Guest Physical Address	Guest Physical Address to Native Physical Address
Visible	Guest	Hypervisor
Updated by	Guest	Hypervisor
Numbers	Per VM	Per VM

Obstacles of x86 Virtualization

- x86 machines cannot satisfy the virtualization requirement...
- So how to cope with these obstacles of x86 virtualization?

x86 Virtualization: Traditional approach

- First one and the older one:
  - **Binary translation**
- The virtual machine monitoring all the instructions executed by guest OS, when founding a sensitive instruction, it executes a series of dynamic generated instructions instead of directly execute the original sensitive instruction.

x86 Virtualization: Traditional approach

- Another, quite new solution:
  - **Paravirtualization**
- Modify the guest operating system, replace the sensitive operations with virtualization-safe routines.

Processor Virtualization

- Limitations of traditional approach

	Binary translation	Paravirtualization
Performance	Relatively Low	Faster
Guest OS Portability	Need not modify Guest OS	Guest OS should be rewrite

Processor Virtualization

- One major advantage of x86 platform is plenty of software.
- Modify the instruction set of x86 for better support to virtualization is not practical considering the legacy software.
- So what can we do..?

x86 Virtualization: New approach

- **Intel VT and AMD-V: Hardware assisted x86 virtualization.**

x86 Virtualization: New approach

- New processor states:
  - VMX non-root operation mode: for guest OS
  - VMX root operation mode: for hypervisor

- State transition between these two state were assisted by hardware

x86 Virtualization: New approach

Example of VMX instructions

1. VMCALL: This simply calls the VM monitor, causing the VM to exit.
2. VMCLEAR: copies VMCS data to memory in case it is written there
3. VMLAUNCH: launches a virtual machine, and changes the launch state of the VMCS to be launched, if it is clear.
4. VMPTRLD: loads a pointer to the VMCS.
5. VMPTRST: stores a pointer to the VMCS.
6. VMREAD: read specified field from VMCS.
7. VMRESUME: resumes a virtual machine.
8. VMWRITE: write specified field in VMCS.
9. VMXOFF: terminates VMX operation.
10. VMXON: starts VMX operation.

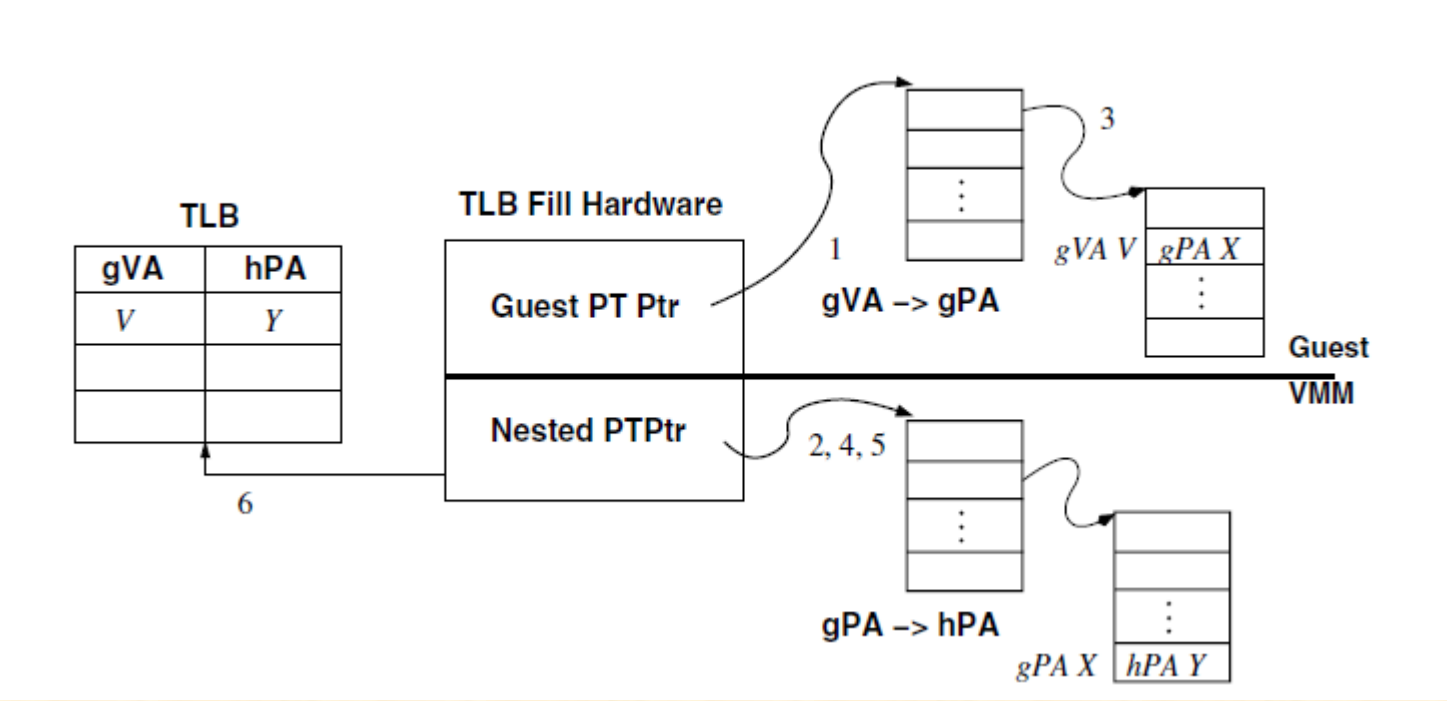
x86 Virtualization: New approach

- VMCS: Virtual Machine Control Structure
  - Guest Area: Per-VM information, included the sensitive registers (CRs, DRs, RSP/RIP/RFLAGS, Segment registers, MSR) and other non-register states.
  - Host Area: Sensitive registers, and other machine states which belongs to hypervisor.

BACK

x86 Virtualization: New approach

- MMU Virtualization: Nested page table
  - MMU knows the exist of “shadow page tables”, MMU handled gPA->nPA translation



BACK

Brief overview to x86 Virtualization: Obstacles and Solutions

Popek and Goldberg's Ideal Virtual Machine Characteristics	**Identical** environment with physical machine	**Efficiently** execution of virtual machine	Resource Control: VMM complete control to resources and **Resource isolation** between different guest OS.	
Obstacles to virtualization in x86 architecture	Many instructions need to check CPL (Current Privilege Level), so a Guest OS is able to found it was running in a virtual machine since it could found the CPL is not in 0, which is the privilege level for operating system kernels. Or some instructions could have ambiguous result when executed in different CPLs.	Circumventing other obstacles can lead to serious performance downgrade…	Memory virtualization: "Guest physical address space" should be isolated for different guest operating systems. Just like an additional level of virtual memory. However the performance could be lower since it caused more TLB miss and page faults.	Virtual machines were considered as "Logical processors", so some processor state should be kept as per-VM manner, like segmentation registers, flag registers, machine state word, etc. However x86 only have one register for these "sensitive" registers, this could lead to the violation of resource isolation.
Traditional solutions	The "Hypercalls": modify guest OS to make it can communicate with hypervisor explicitly; or binary translation	"Hypercalls" can reduce the overhead of binary translation, however it need to modify guest OS.	Shadow page tables, etc.	Use binary translation, "Hypercalls" to implement the save and load of guest states.
New solution: Hardware-assisted x86 virtualization	New **processor modes**: VMX Root operations and VMX Non-root operations. You can just think it as "**Privilege Level -1**", like a new privilege level for hypervisors.	Hardware based **state transition** between guest state (VMX Non-root) and hypervisor state (VMX Root). When sensitive instructions were encountered, the processor automatically signals the hypervisor to handle it.	**Nested page tables**(aka EPT for Intel): CR3 become a per-VM register and can be automatically switched when switching between different guest OS. Guest physical to hypervisor physical address translation were managed by hardware MMU. Guest OS now can perform address translation transparently, without too much overhead.	**VMCS: Virtual Machine Control Structure** For tracking each guest OS's private state. And can be automatically switched when context switching between different guest OS.

Virtualization: on the move

BACK

Virtualization on the move

- GPU virtualization?
  - GPU is used more and more in HPC
  - Allocate the computing resources of GPU more efficiently.
- Virtualization as resource integration?
  - m physical machines to 1 or n (n<=m) virtual machines?
  - ScaleMP

BACK

References

[1] POPEK, G. J., AND GOLDBERG, R. P. , Formal requirements for virtualizable third generation architectures. Commun. ACM 17, 7 (1974), 412–421.

[2] J. S. Robin and C. E. Irvine, Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. In Proceedings of the 9th USENIX Security Symposium, Denver, CO, USA, pages 129--144, Aug. 2000.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, Xen and the art of virtualization. In Proceedings of the nineteenth ACM symposium on Operating Systems Principles (SOSP19), pages 164-177. ACM Press, 2003.

[4] Keith Adams, Ole Agesen, A Comparison of Software and Hardware Techniques for x86 Virtualization. ASPLOS 06

[5] NEIGER, G., SANTONI, A., LEUNG, F.,RODGERS, D., AND UHLIG, R., Intel virtualization technology: Hardware support for efficient processor virtualization. Intel Technology Journal 10, 3 (2006).

[6] INTEL CORPORATION. Intel Virtualization Technology Specification for the IA-32 Intel R Architecture, April 2005

[7] Advanced Micro Devices, Inc. White paper on AMD-V Nested Paging, Rev. 1.0, July, 2008

[8] Susanta Nanda, Tzi-cker Chiueh, A Survey on Virtualization Technologies. RPE Report, pages 1–42, 2005.

[9] I. Pratt et al, Xen 3.0 and the Art of Virtualization. In Proc. of the Ottawa Linux Symposium, 2005.

[10] VMWare, Inc. Understanding full virtualization, paravirtualization, and hardware assist. http://www.vmware.com/files/pdf/VMware/paravirtualization.pdf, 2007.

[11] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux virtual machine monitor. In OLS '07: The 2007 Ottawa Linux Symposium, pages 225–230, July 2007.

Thank You!