Chapter 5 – Cloud Resource Virtualization

Contents

- Virtualization.
- Layering and virtualization.
- Virtual machine monitor.
- Virtual machine.
- Performance and security isolation.
- Architectural support for virtualization.
- x86 support for virtualization.
- Full and paravirtualization.
- Xen 1.0 and Xen 2.0.
- Performance comparison of virtual machine monitors.
- The darker side of virtualization.
- Software fault isolation.

Motivation

- There are many physical realizations of the fundamental abstractions necessary to describe the operation of a computing systems.
  - Interpreters.
  - Memory.
  - Communications links.
- Virtualization is a basic tenet of cloud computing, it simplifies the management of physical resources for the three abstractions.
- The state of a virtual machine (VM) running under a virtual machine monitor (VMM) can de saved and migrated to another server to balance the load.
- Virtualization allows users to operate in environments they are familiar with, rather than forcing them to idiosyncratic ones.

Motivation (cont'd)

- Cloud resource virtualization is important for:
  - System security, as it allows isolation of services running on the same hardware.
  - Performance and reliability, as it allows applications to migrate from one platform to another.
  - The development and management of services offered by a provider.
  - Performance isolation.

Virtualization

- Simulates the interface to a physical object by:
  - Multiplexing: creates multiple virtual objects from one instance of a physical object. Example - a processor is multiplexed among a number of processes or threads.
  - Aggregation: creates one virtual object from multiple physical objects. Example - a number of physical disks are aggregated into a RAID disk.
  - Emulation: constructs a virtual object from a different type of a physical object. Example - a physical disk emulates a Random Access Memory (RAM).
  - Multiplexing and emulation. Examples - virtual memory with paging multiplexes real memory and disk; a virtual address emulates a real address.

Layering

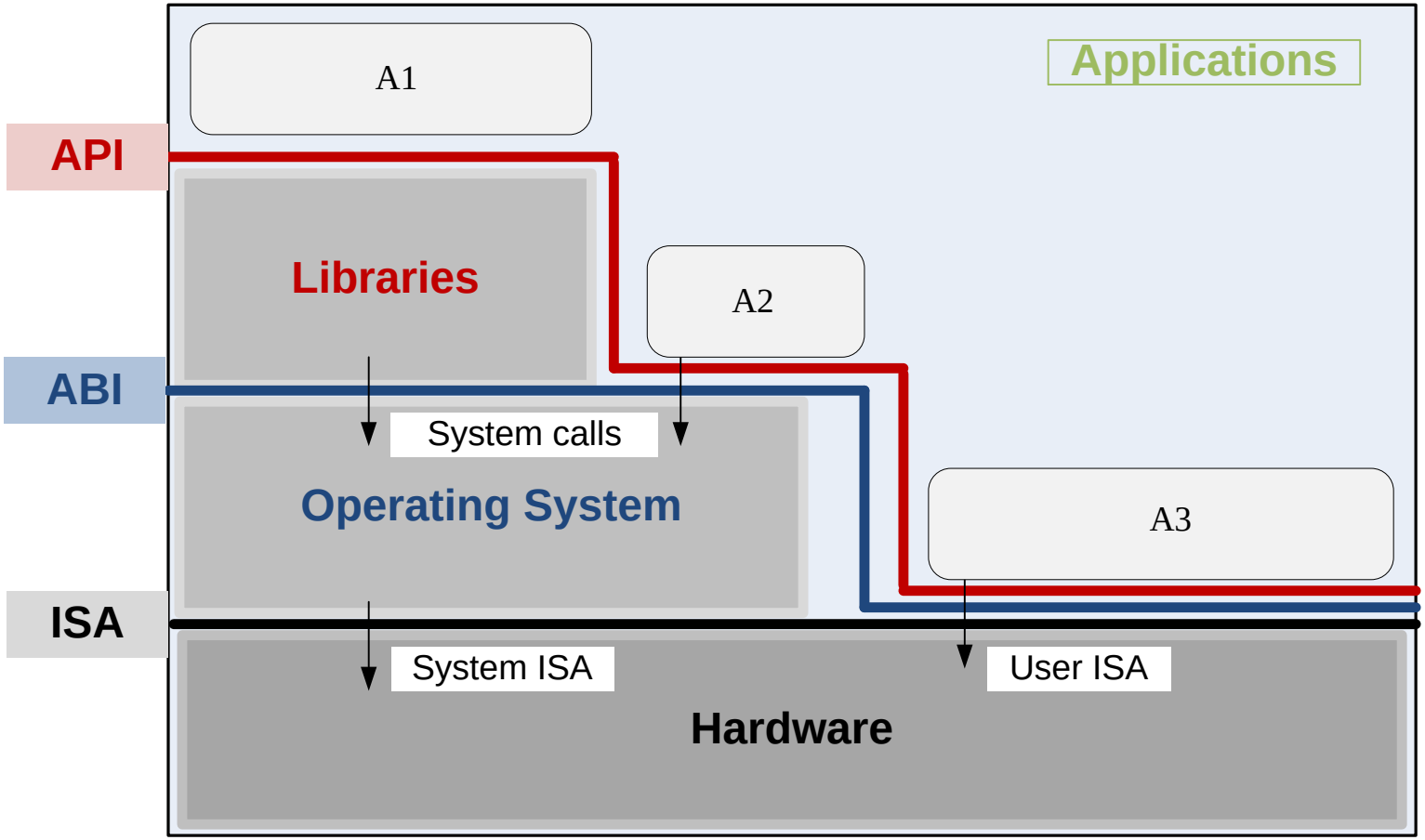
- Layering – a common approach to manage system complexity.
  - Minimizes the interactions among the subsystems of a complex system.
  - Simplifies the description of the subsystems; each subsystem is abstracted through its interfaces with the other subsystems.
  - We are able to design, implement, and modify the individual subsystems independently.
- Layering in a computer system.
  - Hardware.
  - Software.
    - Operating system.
    - Libraries.
    - Applications.

Interfaces

- Instruction Set Architecture (ISA) – at the boundary between hardware and software.

- Application Binary Interface (ABI) – allows the ensemble consisting of the application and the library modules to access the hardware; the ABI does not include privileged system instructions, instead it invokes system calls.

- Application Program Interface (API) - defines the set of instructions the hardware was designed to execute and gives the application access to the ISA; it includes HLL library calls which often invoke system calls.



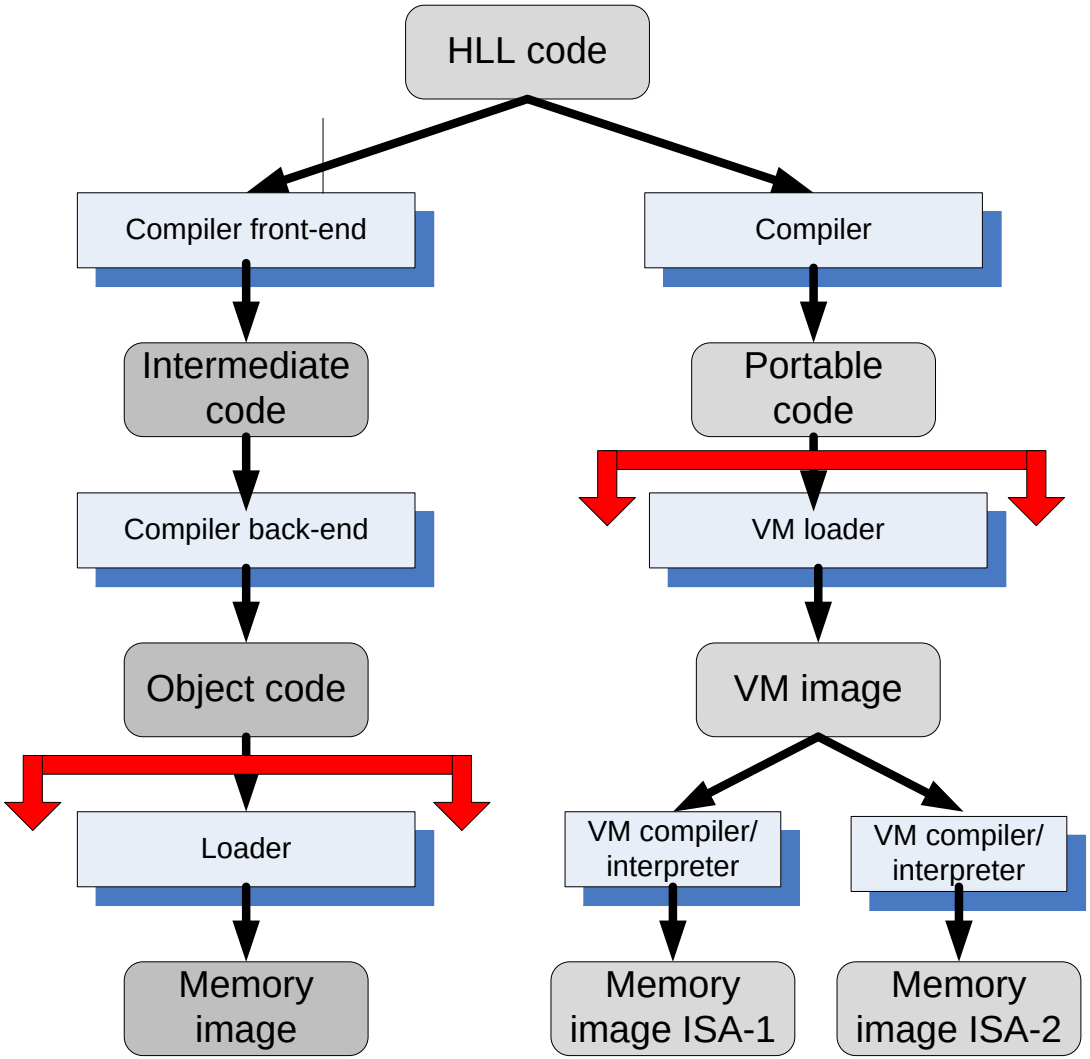
Application Programming Interface, Application Binary Interface, and Instruction Set Architecture . An application uses library functions (A1), makes system calls (A2), and executes machine instructions (A3).

Code portability

- Binaries created by a compiler for a specific ISA and a specific operating systems are not portable.

- It is possible, though, to compile a HLL program for a virtual machine (VM) environment where portable code is produced and distributed and then converted by binary translators to the ISA of the host system.

  A dynamic binary translation converts blocks of guest instructions from the portable code to the host instruction and leads to a significant performance improvement, as such blocks are cached and reused


HLL code
Compiler front-end
Compiler
Intermediate code
Portable code
Compiler back-end
VM loader
Object code
VM image
Loader
VM compiler/ interpreter
VM compiler/ interpreter
Memory image
Memory image ISA-1
Memory image ISA-2

Virtual machine monitor (VMM / hypervisor)

- Partitions the resources of computer system into one or more virtual machines (VMs). Allows several operating systems to run concurrently on a single hardware platform.

- A VMM allows
  - Multiple services to share the same platform.
  - Live migration - the movement of a server from one platform to another.
  - System modification while maintaining backward compatibility with the original system.
  - Enforces isolation among the systems, thus security.

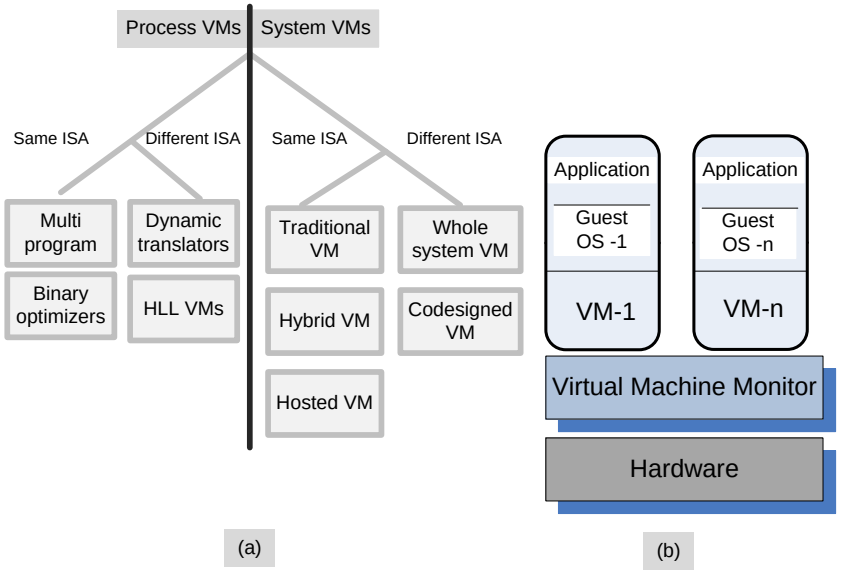
VMM virtualizes the CPU and the memory

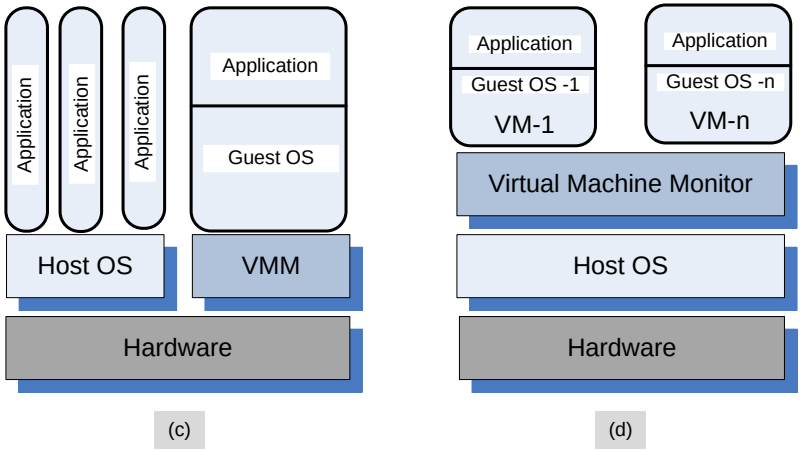
- A VMM
  - Traps the privileged instructions executed by a guest OS and enforces the correctness and safety of the operation.
  - Traps interrupts and dispatches them to the individual guest operating systems.
  - Controls the virtual memory management.
  - Maintains a shadow page table for each guest OS and replicates any modification made by the guest OS in its own shadow page table. This shadow page table points to the actual page frame and it is used by the Memory Management Unit (MMU) for dynamic address translation.
  - Monitors the system performance and takes corrective actions to avoid performance degradation. For example, the VMM may swap out a Virtual Machine to avoid thrashing.

Virtual machines (VMs)

- VM - isolated environment that appears to be a whole computer, but actually only has access to a portion of the computer resources.
- Process VM - a virtual platform created for an individual process and destroyed once the process terminates.
- System VM - supports an operating system together with many user processes.
- Traditional VM - supports multiple virtual machines and runs directly on the hardware.
- Hybrid VM - shares the hardware with a host operating system and supports multiple virtual machines.
- Hosted VM - runs under a host operating system.

Traditional, hybrid, and hosted VMs





Name	Host ISA	Guest ISA	Host OS	guest OS	Company
Integrity VM	*x86*-64	*x86*-64	HP-Unix	Linux,Windows HP Unix	HP
Power VM	Power	Power	No host OS	Linux, AIX	IBM
z/VM	z-ISA	z-ISA	No host OS	Linux on z-ISA	IBM
Lynx Secure	*x86*	*x86*	No host OS	Linux, Windows	LinuxWorks
Hyper-V Server	*x86*-64	*x86*-64	Windows	Windows	Microsoft
Oracle VM	*x86*, *x86*-64	*x86*, *x86*-64	No host OS	Linux, Windows	Oracle
RTS Hypervisor	*x86*	*x86*	No host OS	Linux, Windows	Real Time Systems
SUN xVM	*x86*, SPARC	same as host	No host OS	Linux, Windows	SUN
VMware EX Server	*x86*, *x86*-64	*x86*, *x86*-64	No host OS	Linux, Windows Solaris, FreeBSD	VMware
VMware Fusion	*x86*, *x86*-64	*x86*, *x86*-64	MAC OS *x86*	Linux, Windows Solaris, FreeBSD	VMware
VMware Server	*x86*, *x86*-64	*x86*, *x86*-64	Linux, Windows	Linux, Windows Solaris, FreeBSD	VMware
VMware Workstation	*x86*, *x86*-64	*x86*, *x86*-64	Linux, Windows	Linux, Windows Solaris, FreeBSD	VMware
VMware Player	*x86*, *x86*-64	*x86*, *x86*-64	Linux Windows	Linux, Windows Solaris, FreeBSD	VMware
Denali	*x86*	*x86*	Denali	ILVACO, NetBSD	University of Washington
Xen	*x86*, *x86*-64	*x86*, *x86*-64	Linux Solaris	Linux, Solaris NetBSD	University of Cambridge

Performance and security isolation

- The run-time behavior of an application is affected by other applications running concurrently on the same platform and competing for CPU cycles, cache, main memory, disk and network access. Thus, it is difficult to predict the completion time!

- Performance isolation - a critical condition for QoS guarantees in shared computing environments.

- A VMM is a much simpler and better specified system than a traditional operating system. Example - Xen has approximately 60,000 lines of code; Denali has only about half, 30,000.

- The security vulnerability of VMMs is considerably reduced as the systems expose a much smaller number of privileged functions.

Computer architecture and virtualization

- Conditions for efficient virtualization:
  - A program running under the VMM should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly.
  - The VMM should be in complete control of the virtualized resources.
  - A statistically significant fraction of machine instructions must be executed without the intervention of the VMM.
- Two classes of machine instructions:
  - Sensitive - require special precautions at execution time:
    - Control sensitive - instructions that attempt to change either the memory allocation or the privileged mode.
    - Mode sensitive - instructions whose behavior is different in the privileged mode.
  - Innocuous - not sensitive.

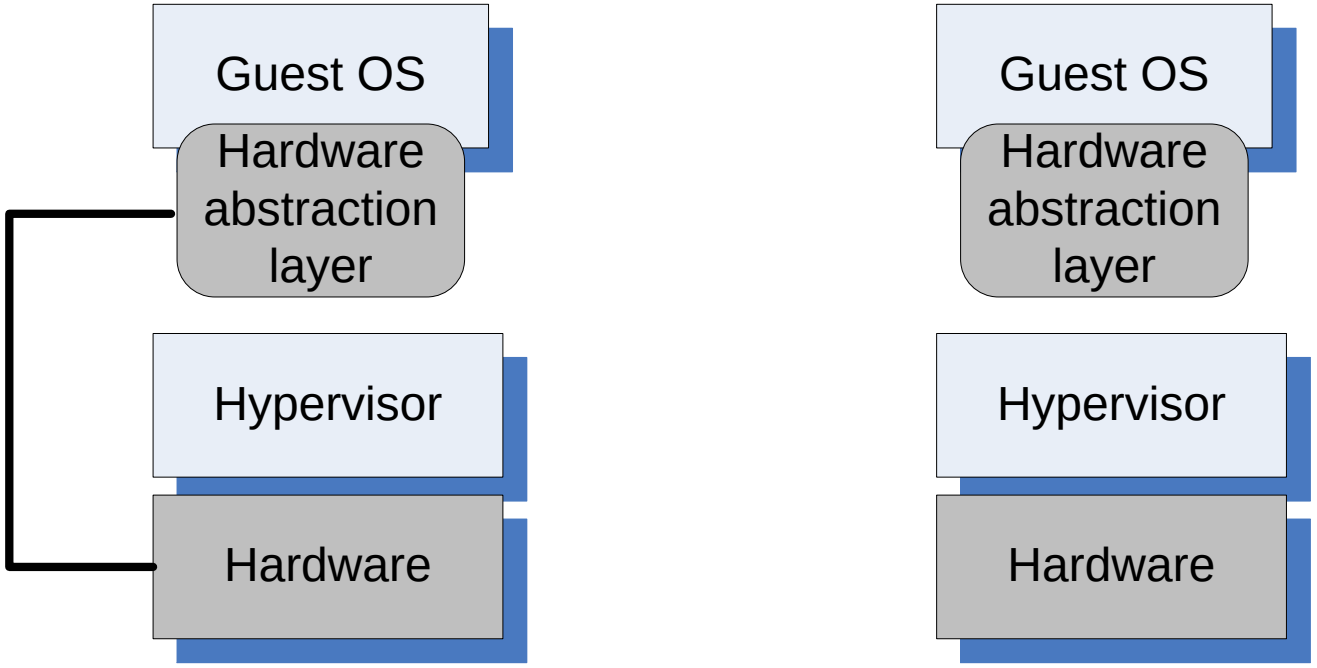
Full virtualization and paravirtualization

- Full virtualization – a guest OS can run unchanged under the VMM as if it was running directly on the hardware platform.
  - Requires a virtualizable architecture.
  - Examples: Vmware.

- Paravirtualization - a guest operating system is modified to use only instructions that can be virtualized. Reasons for paravirtualization:
  - Some aspects of the hardware cannot be virtualized.
  - Improved performance.
  - Present a simpler interface.

  Examples: Xen, Denaly

Full virtualization and paravirtualization



(a) Full virtualization

(b) Paravirtualization

Virtualization of x86 architecture

- Ring de-privileging - a VMMs forces the operating system and the applications to run at a privilege level greater than 0.
- Ring aliasing - a guest OS is forced to run at a privilege level other than that it was originally designed for.
- Address space compression - a VMM uses parts of the guest address space to store several system data structures.
- Non-faulting access to privileged state - several store instructions can only be executed at privileged level 0 because they operate on data structures that control the CPU operation. They fail silently when executed at a privilege level other than 0.
- Guest system calls which cause transitions to/from privilege level 0 must be emulated by the VMM.
- Interrupt virtualization - in response to a physical interrupt, the VMM generates a ``virtual interrupt'' and delivers it later to the target guest OS which can mask interrupts.

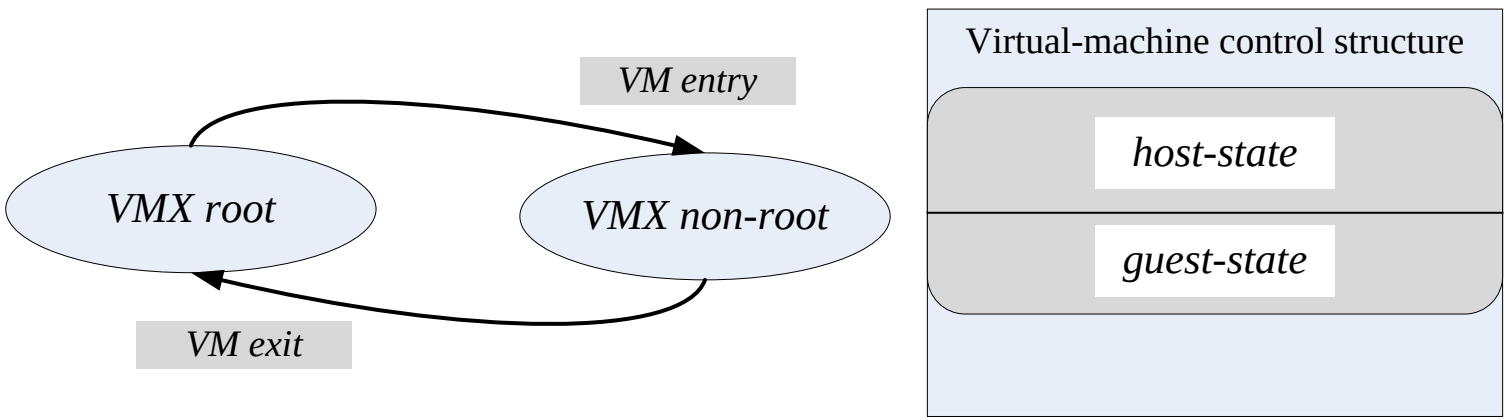
Virtualization of x86 architecture (cont'd)

- Access to hidden state - elements of the system state, e.g., descriptor caches for segment registers, are hidden; there is no mechanism for saving and restoring the hidden components when there is a context switch from one VM to another.
- Ring compression - paging and segmentation protect VMM code from being overwritten by guest OS and applications. Systems running in 64-bit mode can only use paging, but paging does not distinguish between privilege levels 0, 1, and 2, thus the guest OS must run at privilege level 3, the so called (0/3/3) mode. Privilege levels 1 and 2 cannot be used thus, the name ring compression.
- The task-priority register is frequently used by a guest OS; the VMM must protect the access to this register and trap all attempts to access it. This can cause a significant performance degradation.

VT-x, a major architectural enhancement

- Supports two modes of operations:
  - VMX root - for VMM operations.
  - VMX non-root - support a VM.
- The Virtual Machine Control Structure including host-state and guest-state areas.
  - VM entry - the processor state is loaded from the guest-state of the VM scheduled to run; then the control is transferred from VMM to the VM.
  - VM exit - saves the processor state in the guest-state area of the running VM; then it loads the processor state from the host-state area, finally transfers control to the VMM.

VT- x


VM entry
VMX root
VMX non-root
VM exit
Virtual-machine control structure
host-state
guest-state
(a)
(b)

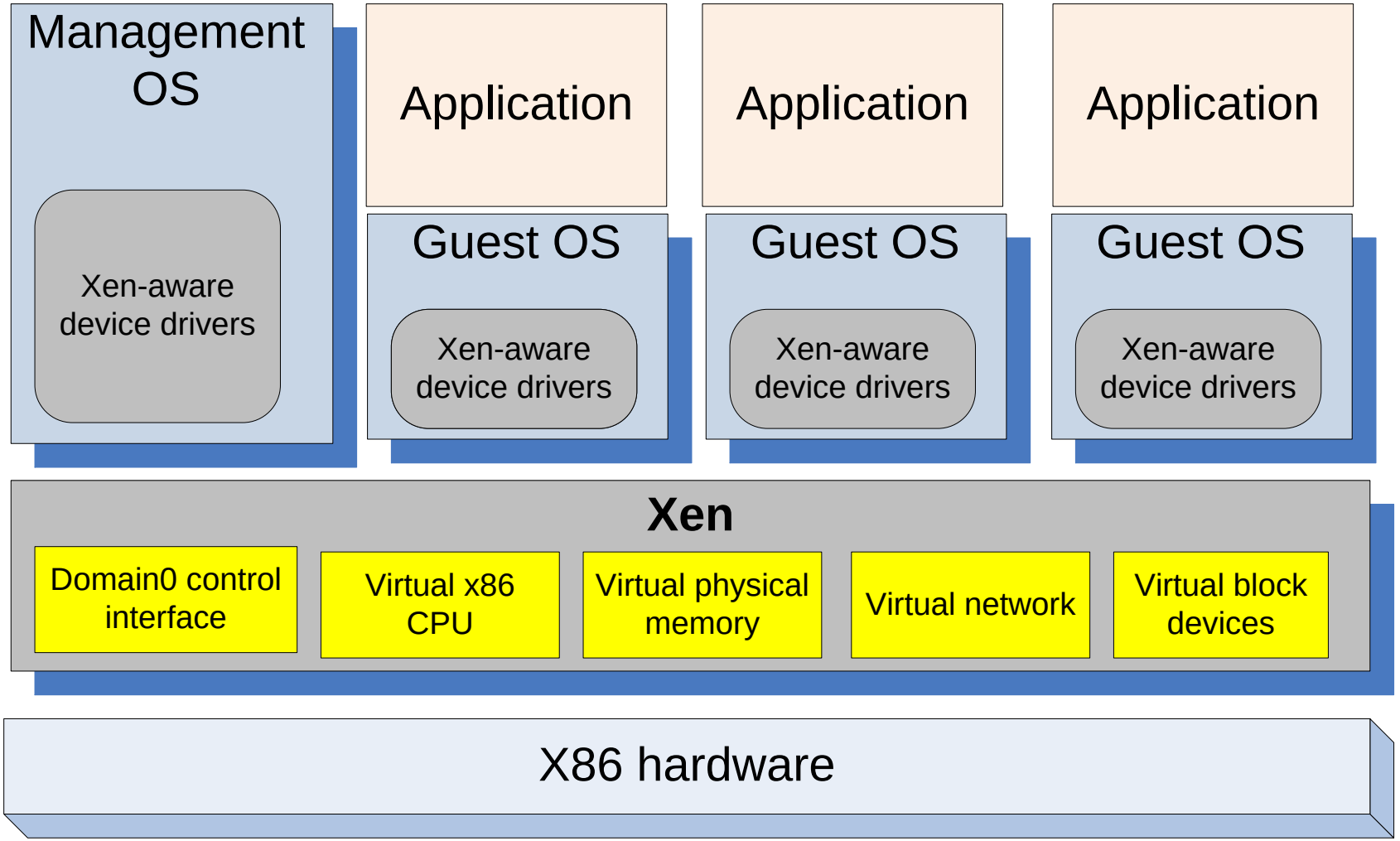
VT-d, a new virtualization architecture

- I/O MMU virtualization gives VMs direct access to peripheral devices.
- VT-d supports:
  - DMA address remapping, address translation for device DMA transfers.
  - Interrupt remapping, isolation of device interrupts and VM routing.
  - I/O device assignment, the devices can be assigned by an administrator to a VM in any configurations.
  - Reliability features, it reports and records DMA and interrupt errors that my otherwise corrupt memory and impact VM isolation.

Xen - a VMM based on paravirtualization

- The goal of the Cambridge group - design a VMM capable of scaling to about 100 VMs running standard applications and services without any modifications to the Application Binary Interface (ABI).
- Linux, Minix, NetBSD, FreeBSD, NetWare, and OZONE can operate as paravirtualized Xen guest OS running on x86, x86-64, Itanium, and ARM architectures.
- Xen domain - ensemble of address spaces hosting a guest OS and applications running under the guest OS. Runs on a virtual CPU.
  - Dom0 - dedicated to execution of Xen control functions and privileged instructions.
  - DomU - a user domain.
- Applications make system calls using hypercalls processed by Xen; privileged instructions issued by a guest OS are paravirtualized and must be validated by Xen.

Xen


Management OS
Xen-aware device drivers
Application
Guest OS
Xen-aware device drivers
Application
Guest OS
Xen-aware device drivers
Application
Guest OS
Xen-aware device drivers
Xen
Domain0 control interface
Virtual x86 CPU
Virtual physical memory
Virtual network
Virtual block devices
X86 hardware

Xen implementation on x86 architecture

- Xen runs at privilege Level 0, the guest OS at Level 1, and applications at Level 3.
- The x86 architecture does not support either the tagging of TLB entries or the software management of the TLB. Thus, address space switching, when the VMM activates a different OS, requires a complete TLB flush; this has a negative impact on the performance.
- Solution - load Xen in a 64 MB segment at the top of each address space and delegate the management of hardware page tables to the guest OS with minimal intervention from Xen. This region is not accessible or re-mappable by the guest OS.
- Xen schedules individual domains using the Borrowed Virtual Time (BVT) scheduling algorithm.
- A guest OS must register with Xen a description table with the addresses of exception handlers for validation.

Dom0 components

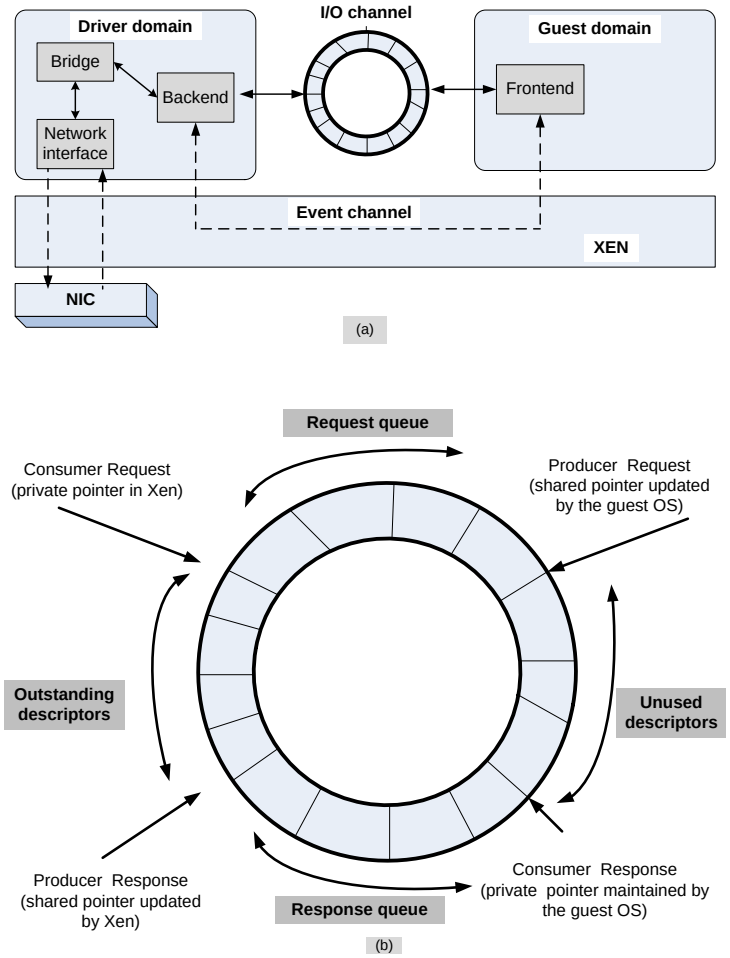
- XenStore – a Dom0 process.
  - Supports a system-wide registry and naming service.
  - Implemented as a hierarchical key-value storage.
  - A watch function informs listeners of changes of the key in storage they have subscribed to.
  - Communicates with guest VMs via shared memory using Dom0 privileges.
- Toolstack - responsible for creating, destroying, and managing the resources and privileges of VMs.
  - To create a new VM, a user provides a configuration file describing memory and CPU allocations and device configurations.
  - Toolstack parses this file and writes this information in XenStore.
  - Takes advantage of Dom0 privileges to map guest memory, to load a kernel and virtual BIOS and to set up initial communication channels with XenStore and with the virtual console when a new VM is created.

Strategies for virtual memory management, CPU multiplexing, and I/O devices

Function	Strategy
Paging	A domain may be allocated discontinuous pages. A guest OS has direct access to page tables and handles pages faults directly for efficiency; page table updates are batched for performance and validated by *Xen* for safety.
Memory	Memory is statically partitioned between domains to provide strong isolation. `XenoLinux` implements a *balloon driver* to adjust domain memory.
Protection	A guest OS runs at a lower priority level, in ring 1, while *Xen* runs in ring 0.
Exceptions	A guest OS must register with *Xen* a description table with the addresses of exception handlers previously validated; exception handlers other than the page fault handler are identical with *x86* native exception handlers.
System calls	To increase efficiency, a guest OS must install a "fast" handler to allow system calls from an application to the guest OS and avoid indirection through *Xen*.
Interrupts	A lightweight event system replaces hardware interrupts; synchronous system calls from a domain to *Xen* use *hypercalls* and notifications are delivered using the asynchronous event system.
Multiplexing	A guest OS may run multiple applications.
Time	Each guest OS has a timer interface and is aware of "real" and "virtual" time.
Network and I/O devices	Data is transferred using asynchronous I/O rings; a ring is a circular queue of descriptors allocated by a domain and accessible within *Xen*.
Disk access	Only *Dom0* has direct access to IDE and SCSI disks; all other domains access persistent storage through the Virtual Block Device (VBD) abstraction.

Xen abstractions for networking and I/O

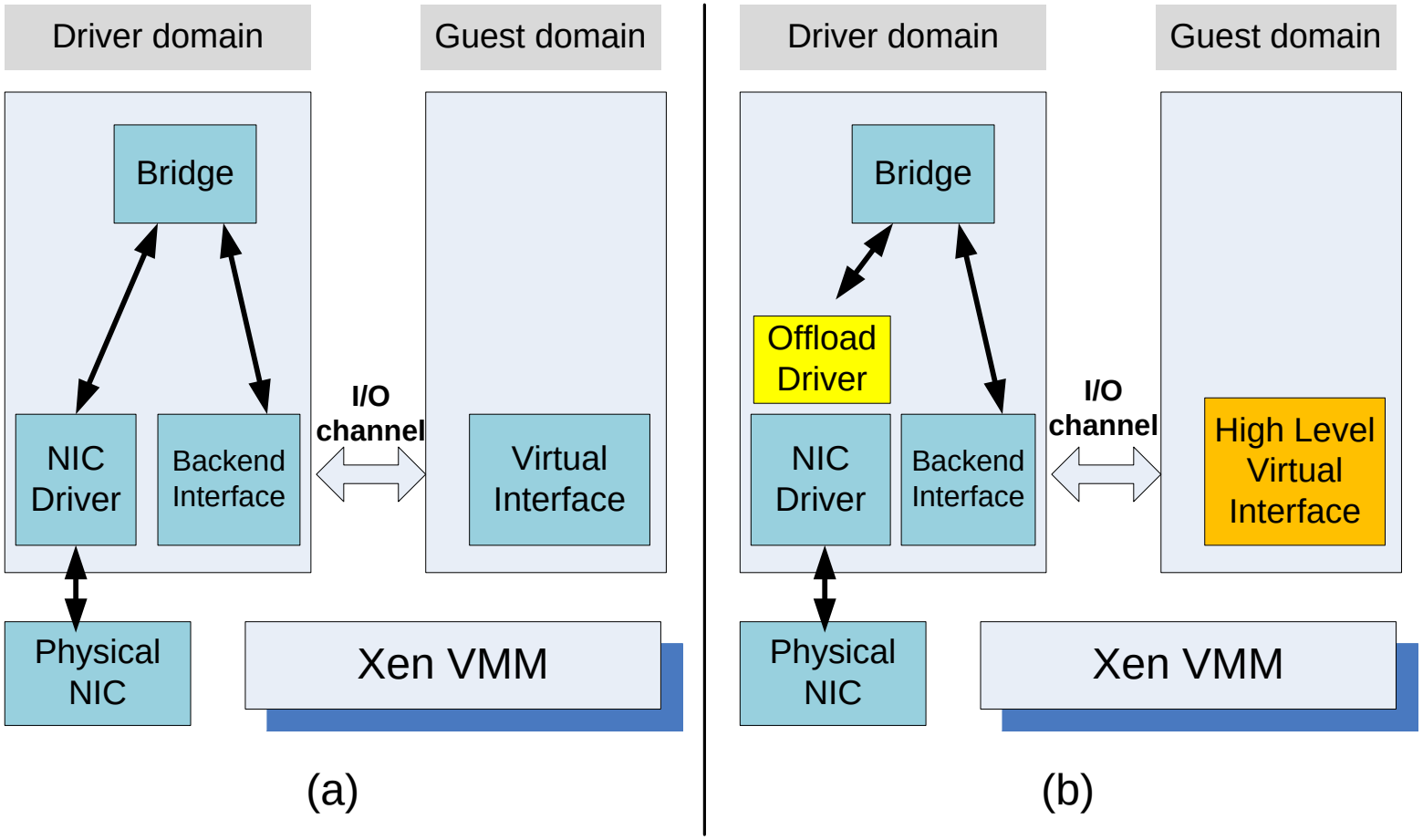
- Each domain has one or more Virtual Network Interfaces (VIFs) which support the functionality of a network interface card. A VIF is attached to a Virtual Firewall-Router (VFR).
- Split drivers have a front-end in the DomU and the back-end in Dom0; the two communicate via a ring in shared memory.
- Ring - a circular queue of descriptors allocated by a domain and accessible within Xen. Descriptors do not contain data, the data buffers are allocated off-band by the guest OS.
- Two rings of buffer descriptors, one for packet sending and one for packet receiving, are supported.
- To transmit a packet:
  - a guest OS enqueues a buffer descriptor to the send ring,
  - then Xen copies the descriptor and checks safety,
  - copies only the packet header, not the payload, and
  - executes the matching rules.



Xen zero-copy semantics for data transfer using I/O rings. (a) The communication between a guest domain and the driver domain over an I/O and an event channel; NIC is the Network Interface Controller. (b) the circular ring of buffers.

Xen 2.0

- Optimization of:
  - Virtual interface - takes advantage of the capabilities of some physical NICs, such as checksum offload.
  - I/O channel - rather than copying a data buffer holding a packet, each packet is allocated in a new page and then the physical page containing the packet is re-mapped into the target domain.
  - Virtual memory - takes advantage of the superpage and global page mapping hardware on Pentium and Pentium Pro processors. A superpage entry covers 1,024 pages of physical memory and the address translation mechanism maps a set of contiguous pages to a set of contiguous physical pages. This helps reduce the number of TLB misses.



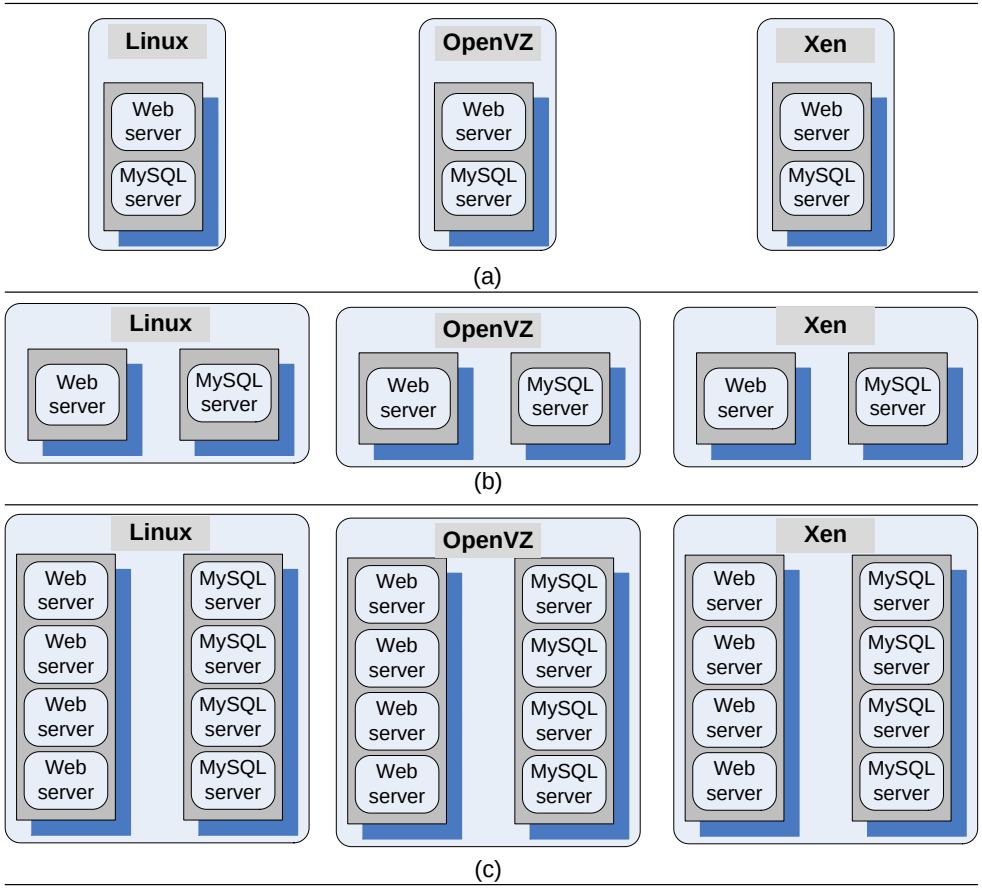
Xen network architecture .(a) The original architecture;
(b) The optimized architecture

System	Receive data rate (Mbps)	Send data rate (Mbps)
Linux	2 508	3 760
Xen driver	1 728	3 760
Xen guest	820	750
optimized *Xen* guest	970	3 310

A comparison of send and receive data rates for a native Linux system, the Xen driver domain, an original Xen guest domain, and an optimized Xen guest domain.

Performance comparison of virtual machines

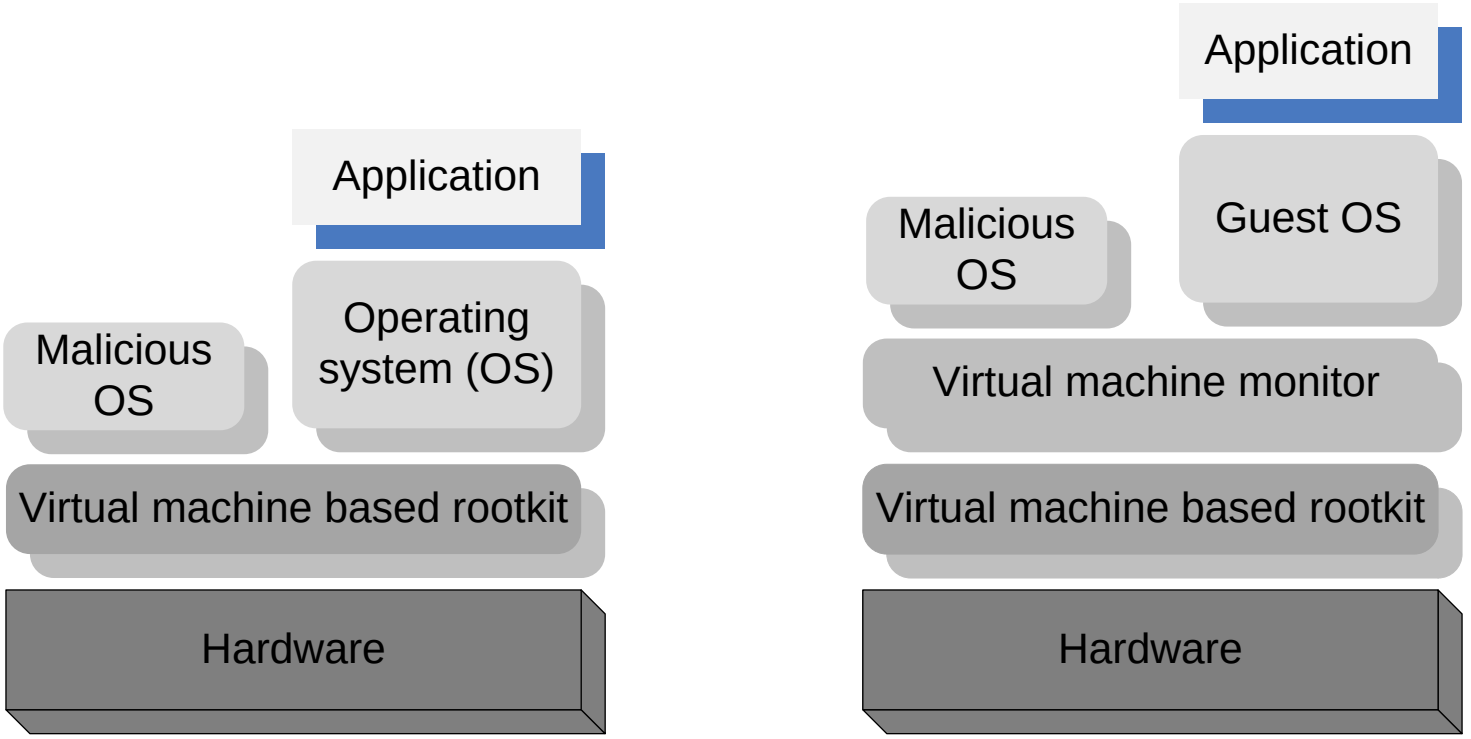
- Compare the performance of Xen and OpenVZwith, a standard operating system, a plain vanilla Linux.
- The questions examined are:
  - How the performance scales up with the load?
  - What is the impact of a mix of applications?
  - What are the implications of the load assignment on individual servers?
- The main conclusions:
  - The virtualization overhead of Xen is considerably higher than that of OpenVZ and that this is due primarily to L2-cache misses.
  - The performance degradation when the workload increases is also noticeable for Xen.
  - Hosting multiple tiers of the same application on the same server is not an optimal solution.



The setup for the performance comparison of a native Linux system with OpenVZ, and the Xen systems. The applications are a web server and a MySQL database server. (a) The first experiment, the web and the DB, share a single system; (b) The second experiment, the web and the DB, run on two different systems; (c) The third experiment, the web and the DB, run on two different systems and each has four instances.

The darker side of virtualization

- In a layered structure, a defense mechanism at some layer can be disabled by malware running at a layer below it.
- It is feasible to insert a *rogue VMM,* a Virtual-Machine Based Rootkit (VMBR) between the physical hardware and an operating system.
- Rootkit - malware with a privileged access to a system.
- The VMBR can enable a separate malicious OS to run surreptitiously and make this malicious OS invisible to the guest OS and to the application running under it.
- Under the protection of the VMBR, the malicious OS could:
  - observe the data, the events, or the state of the target system.
  - run services, such as spam relays or distributed denial-of-service attacks.
  - interfere with the application.



The insertion of a Virtual-Machine Based Rootkit (VMBR) as the lowest layer of the software stack running on the physical hardware; (a) below an operating system; (b) below a legitimate virtual machine monitor. The VMBR enables a malicious OS to run surreptitiously and makes it invisible to the genuine or the guest OS and to the application.

Feature /Architecture	*x86*-32	*x86*-64	*ARM*
Addressable memory	1 GB	4 GB	1 GB
Virtual base address	any	44GB	0
Data model	ILP32	ILP32	ILP 32
Reserved registers	0 of 8	1 of 16	0 of 16
Data address mask	None	Implicit in result width	Explicit instruction
Control address mask	Explicit instruction	Explicit instruction	Explicit instruction
Bundle size (bytes)	32	32	16
Data in text segment	forbidden	forbidden	allowed
Safe address registers	all	rsp, rbp	sp
Out-of-sandbox store	trap	wraps mod 4 GB	No effect
Out-of-sandbox jump	trap	wraps mod 4 GB	wraps mod 1 GB

The features of the SFI for the Native Client on the x86-32, x86-64 , and ARM.