Cloud Architecture and Virtualisation

Lecture 4

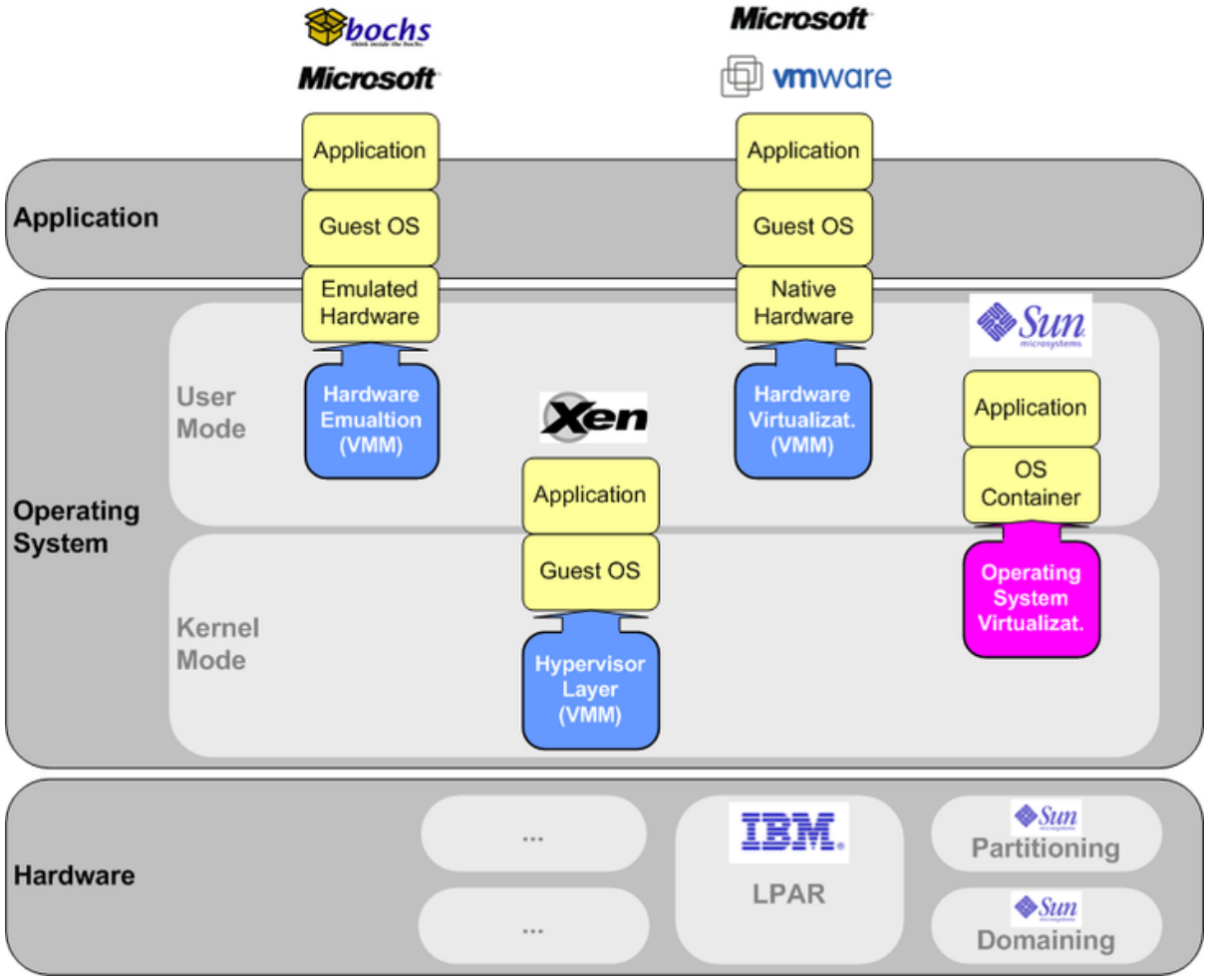
Virtualisation

TOC

- Introduction to virtualisation
- Layers and interfaces
- Virtual machines and virtual machine managers
- Hardware support
- Security

Virtualisation overview

Areas of virtualisation



David E.Y. Sarna, "Implementing and Developing Cloud Computing Applications"
Wikipedia

Why to virtualise?

- There are distinguished three classes of fundamental abstractions necessary to describe the operation of a computing systems:
  - Interpreters (e.g., processors which transform information),
  - Memory (e.g., primary and the secondary memory for storing information),
  - Communications links (communication channels which allow different systems to communicate with one another)
- The physical realisation of each one of these abstractions can vary in terms of:
  - bandwidth (the number of operations per unit of time),
  - latency (the time elapsed from the instance an operation is initiated and its effect is sensed),
  - Reliability,
  - Other physical characteristics
- Software systems, such as operating systems are responsible for the management of the system resources, the physical implementations of the three abstractions

Why to virtualise?

- Resource management for a community of users with a wide range of applications running under different operating systems is a very difficult problem
- Resource management becomes even more complex when resources are oversubscribed and users are uncooperative
- In addition to external factors, resource management is affected by internal factors, such as:
  - The heterogeneity of the hardware and software systems,
  - The ability to approximate the global state of the system and to redistribute the load,
  - The failure rates of different components,
  - Many other factors

Why to virtualise?

- The traditional solution for a data centre is to install standard operating systems on individual systems and rely on conventional OS techniques to ensure resource sharing, application protection, and performance isolation
  - System administration, accounting, security, and resource management are very challenging for the providers of service in this setup
  - Application development and performance optimisation are equally challenging for the users
- The alternative is resource virtualisation

Why to virtualise?

- Virtualisation is a basic tenet of cloud computing, it simplifies some of the resource management tasks
- For example, 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
- At the same time, virtualisation allows users to operate in environments they are familiar with, rather than forcing them to work in idiosyncratic environments

What is virtualisation?

- Virtualisation simulates the interface to a physical object by any one of four means:
  - Multiplexing: create multiple virtual objects from one instance of a physical object, e.g.:
    - a processor is multiplexed among a number of processes or threads.
  - Aggregation: create one virtual object from multiple physical objects, e.g.:
    - a number of physical disks are aggregated into a RAID disk.
  - Emulation: construct a virtual object from a different type of a physical object, e.g.:
    - a physical disk emulates a Random Access Memory.
  - Multiplexing and emulation, e.g.:
    - virtual memory with paging multiplexes real memory and disk and a virtual address emulates a real address; the TCP protocol emulates a reliable bit pipe and multiplexes a physical communication channel and a processor.

What is virtualisation?

- Virtualisation abstracts the underlying resources and simplifies their use, isolates users from one another, and supports replication which, in turn, increases the elasticity of the system
- Virtualisation is a critical aspect of cloud computing, equally important for the providers and the consumers of cloud services, and plays an important role 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

Virtualisation in a cloud

- In a cloud computing environment a virtual-machine monitor runs on the physical hardware and exports hardware-level abstractions to one or more guest operating systems
- A guest OS interacts with the virtual hardware in the same manner it would interact with the physical hardware, but under the watchful eye of the VMM which traps all privileged operations and mediates the interactions of the guest OS with the hardware.
  - For example, a VMM would control I/O operations to two virtual disks implemented as two different set of tracks on a physical disk. New services can be added without the need to modify an operating system

Virtualisation in a cloud

- User convenience is a necessary condition for the success of the utility computing paradigm
  - one of the multiple facets of user convenience is the ability to run remotely using the system software and libraries required by the application
- User convenience is a major advantage of a Virtual Machine architecture versus a traditional operating system
  - a user of the Amazon Web Services (AWS) could submit an Amazon Machine Image (AMI) containing the applications, libraries, data, and the associated configuration settings
  - the user could choose the operating system for the application, then start, terminate, and monitor as many instances of the AMI as needed, using the web service APIs and the performance monitoring and management tools provided by the AWS.

Virtualisation drawbacks

- There are side effects of virtualisation, notably the performance penalty and the hardware costs:
  - All privileged operations of a virtual machine must be trapped and validated by the VMM which, ultimately, controls the system behaviour the increased overhead has a negative impact on the performance.
  - The cost of the hardware for a virtual machine is higher than the cost for a system running a traditional operating system because the physical hardware is shared among a set of guest operating systems and it is typically configured with faster and/or multi-core processors, more memory, larger disks, and additional network interfaces as compared with a system running a traditional operating system

Layering and virtualisation

Layers

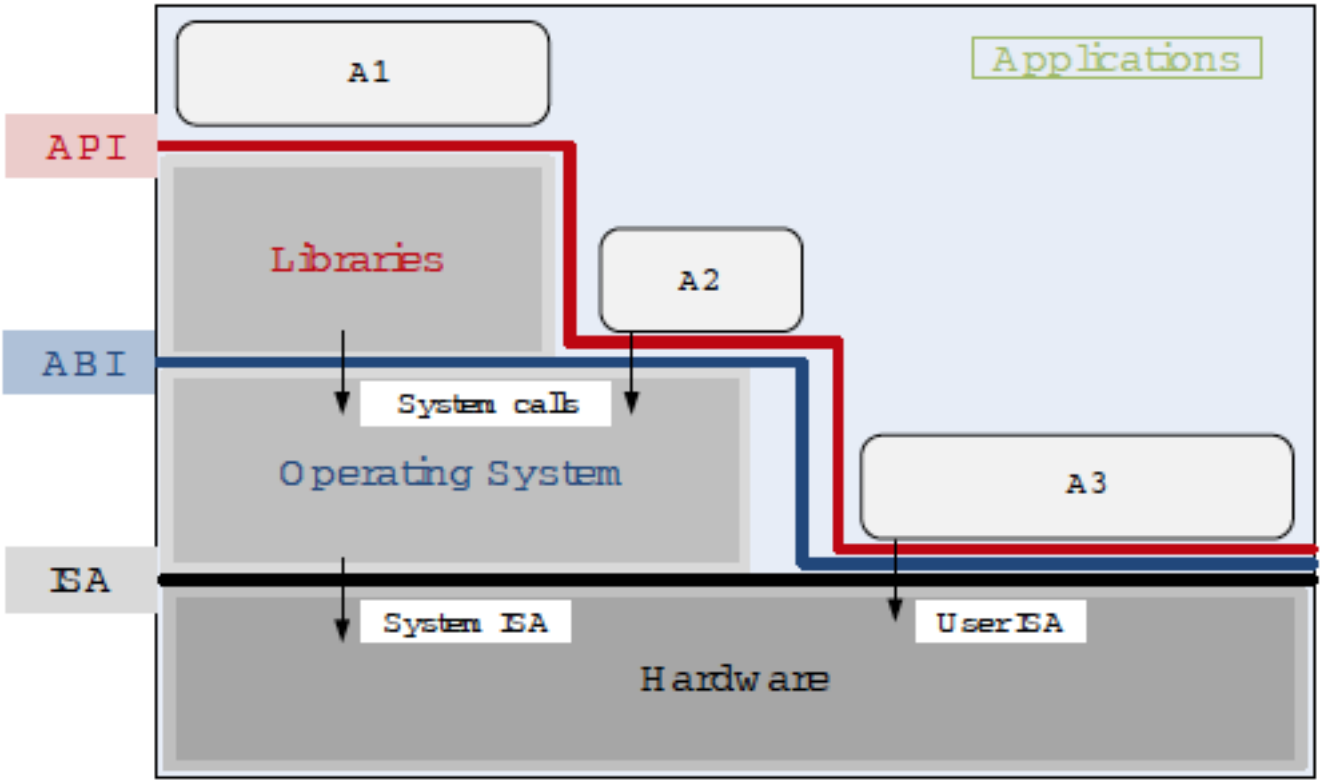
- A common approach to manage system complexity is to identify a set of layers with well defined interfaces among them – the interfaces separate different levels of abstraction
- Layering minimises the interactions among the subsystems and simplifies the description of the subsystems
  - each subsystem is abstracted through its interfaces with the other subsystems thus, we are able to design, implement, and modify the individual subsystems independently

Layers

- The software components including applications, libraries, and operating system interact with the hardware via several interfaces:
  - the Application Programming Interface (API),
  - the Application Binary Interface (ABI),
  - the Instruction Set Architecture (ISA)

Layers

- Interfaces between software components and the hardware
  - An application uses library functions (A1), makes system calls (A2), and executes machine instructions (A3):



Layers and interfaces

- Computer systems are fairly complex and their operation is best understood when we consider a model with distinct layers communicating with dedicated interfaces:
  - The hardware consists of one or more multi-core processors
  - A system interconnect, (e.g., one or more busses) a memory translation unit, the main memory, and I/O devices, including one or more networking interfaces
  - Applications written mostly in High Level languages (HLL) often call library modules and are compiled into object code
  - Privileged operations, such as I/O requests, cannot be executed in user mode; instead, application and library modules issue system calls and the operating system determines if the privileged operations required by the application do not violate system security or integrity and, if so, executes them on behalf of the user.
  - The binaries resulting from the translation of HLL programs are targeted to a specific hardware architecture.

ISA

- The Instruction Set Architecture (ISA) at the boundary of the hardware and the software
- The ISA (Instruction Set Architecture) defines the set of instructions of a processor
  - E.g., the Intel architecture is represented by the x86-32 and x86-64 instruction sets for systems supporting 32-bit addressing and 64-bit addressing, respectively

ISA

- The hardware supports two execution modes, a privileged, or kernel mode and a user mode
- The instruction set consists of two sets of instructions:
  - Privileged instructions that can only be executed in kernel mode
  - The non-privileged instructions that can be executed in user mode
- There are also sensitive instructions that can be executed in kernel and in user mode, but behave differently

ABI

- The Application Binary Interface (ABI) which 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

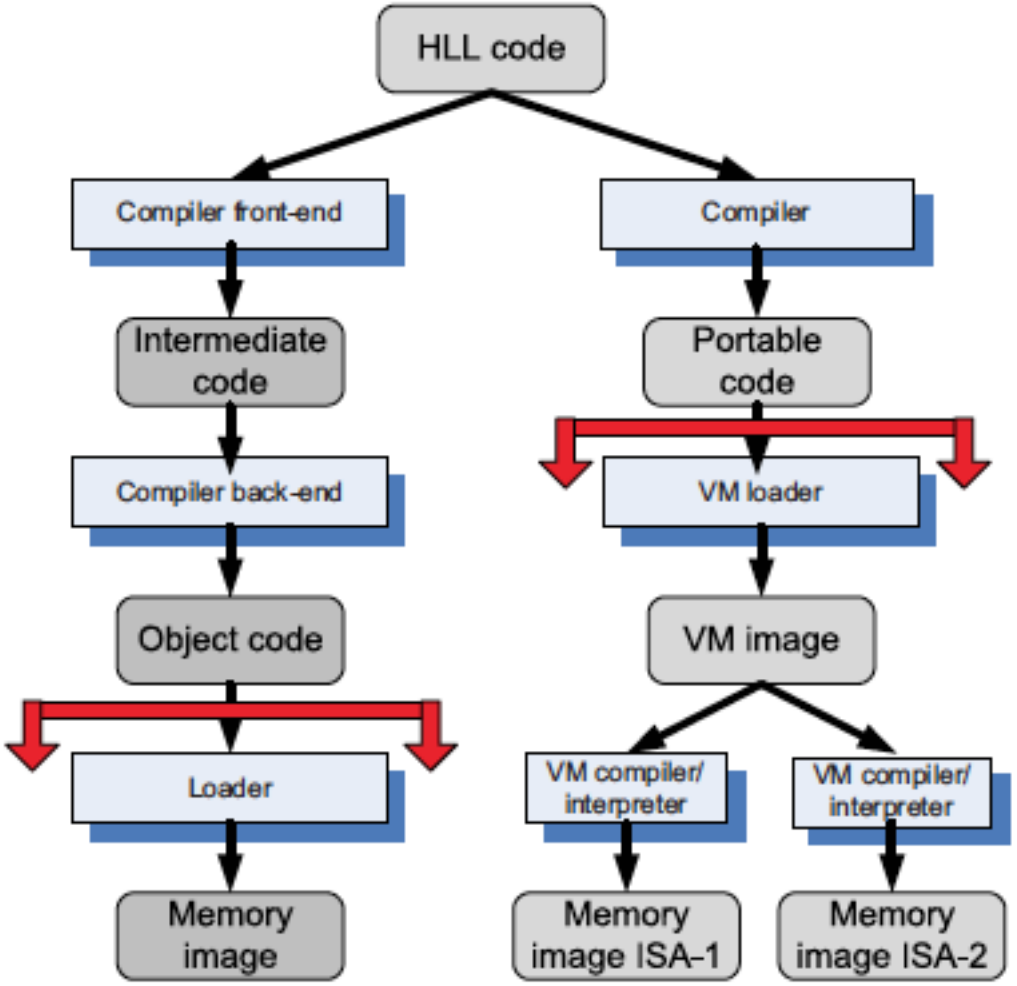
API

- The 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
- A process is the abstraction for the code of an application at execution time; a thread is a light-weight process
- The ABI is the projection of the computer system seen by the process and the API is the projection of the system from the perspective of the HLL program.

HLL execution

- High Level Language (HLL) code can be translated for a specific architecture and operating system
- The HLL code can also be compiled into portable code and then the portable code be translated for systems with different ISAs
- The code shared/distributed is the object code in the first case and the portable code in the second case

HLL execution



HLL execution

- Clearly, the binaries created by a compiler for a specific ISA and a specific operating systems are not portable. Such code cannot run on a computer with a different ISA, or on the computer with the same ISA, but a different operating system
- 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

Virtual machine monitors

VMMs or hypervisors

- A Virtual Machine Monitor (VMM) also called hypervisor is the software that securely partitions the resources of computer system into one or more virtual machines
- A guest operating system is an operating system that runs under the control of a VMM rather than directly on the hardware
- The VMM runs in kernel mode while a guest OS runs in user mode; sometimes the hardware supports a third mode of execution for the guest OS

VMMs or hypervisors

- The VMMs allow several operating systems to run concurrently on a single hardware platform
  - VMMs enforce isolation among these systems, thus security
- A VMM controls how the guest operating system uses the hardware resources; the events occurring in one VM do not affect any other VM running under the same VMM
  - The VMM enables:
    - Multiple services to share the same platform
    - The movement of a server from one platform to another, the so-called live migration
    - System modification while maintaining backward compatibility with the original system

VMMs or hypervisors

- When a guest OS attempts to execute a privileged instruction the VMM traps the operation and enforces the correctness and safety of the operation
- The VMM guarantees the isolation of the individual virtual machines and thus, ensures security and encapsulation, a major concern in cloud computing
- At the same time, the VMM monitors the system performance and takes corrective actions to avoid performance degradation, e.g.:
  - the VMM may swap out a Virtual Machine (copies all pages of that VM from real memory to disk and makes the real memory frames available for paging by other VMs) to avoid thrashing

VMMs or hypervisors

- A VMM virtualises the CPU and the memory, e.g.:
  - the VMM traps interrupts and dispatches them to the individual guest operating systems
  - if a guest OS disables interrupts, the VMM buffers such interrupts until the guest OS enables them
- The VMM 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 hardware component called the Memory Management Unit (MMU) for dynamic address translation

Memory virtualisation

- Memory virtualisation has important implications on the performance
- VMMs use a range of optimisation techniques, e.g.:
  - VMware systems avoid page duplication among different virtual machines, they maintain only one copy of a shared page and use copy-on-write policies
  - Xen imposes total isolation of the VM and does not allow page sharing
- VMMs control the virtual memory management and decide what pages to swap out, e.g.:
  - when the ESX VMware Server wants to swap out pages it uses a balloon process inside a guest OS and requests it to allocate more pages to itself and thus, swapping out pages of some of the processes running under that VM. Then it forces the balloon process to relinquish control of the free page frames

Hypervisors

- There are three leading open source hypervisors:
  - Kernel-based Virtual Machine (KVM),
  - Xen,
  - QEMU
  - (Microsoft Azure uses its own)

KVM

- KVM is developed and maintained at www.linux-kvm.org
  - It has been embraced by Red Hat, a popular distributor of Linux software
  - A current FAQ (http://www.linuxkvm.org/page/FAQ) includes information on supported processors
- KVM offers a live migration feature to move virtual machines from one host to another without downtime

Xen

- The Xen hypervisor is developed and maintained by the Xen.org community as a free solution licensed under the GNU General Public License
  - A customised version of Xen is deployed by Amazon in EC2
- Xen provides support for x86, x86-64, Itanium, Power PC, and ARM processors, which allows the Xen hypervisor to run on a wide variety of computing devices
  - Citrix also provides a free version of Xen, which it calls XenServer (it supports 64-bit versions of Intel and AMD processors)
- Currently Xen supports Linux, NetBSD, FreeBSD, Solaris, Windows, and other common operating systems as guests running on the hypervisor

QEMU

- QEMU is a generic and open source machine emulator and virtualiser
  - When used as a machine emulator, QEMU can run operating systems and programs made for one machine (e.g., an ARM board) on a different machine (e.g., your own PC)
  - By using dynamic translation, it achieves very good performances
  - When used as a virtualiser, QEMU achieves near native performance by executing the guest code directly on the host CPU
- QEMU supports virtualisation when executing under the Xen hypervisor or using the KVM kernel module in Linux. When using KVM, QEMU can virtualise x86, server and embedded PowerPC, and S390 guests

KVM vs. Xen

- Xen is an external hypervisor – it assumes control of the machine and divides resources among guests
- On the other hand, KVM is part of Linux and uses the regular Linux scheduler and memory management
  - This means that KVM is much smaller and simpler to use; it is also provides some features not available in Xen
  - For example KVM can swap guests to disk in order to free RAM.

KVM vs. Xen

- KVM only runs on processors that support x86 hardware virtual machines (hvm), Intel Virtualization Technology (VT), and AMD SVM (Secure Virtual Machine) instruction sets, known as vt/svm
- Xen also allows running modified operating systems on non-hvm x86 processors using a technique called paravirtualisation
- KVM does not support paravirtualisation for CPUs but may support paravirtualisation for device drivers to improve I/O performance

KVM vs. QEMU

- QEMU uses emulation
- KVM uses processor extensions (HVM) for virtualisation

Parallels

- Parallels, Inc. has a product for the Apple Mac environment, Parallels Desktop for Mac, that is also based on hypervisor technology.
- Parallels Server for Mac is also hypervisor-based server virtualisation software
  - It enables IT managers to run multiple Windows, Linux, and Mac OS X Server operating systems on a single Mac Xserve, Apple's line of 1U rack-mounted servers
- At present, it is the only server virtualisation solution for the Mac OS X server platform that allows users to virtualise the Mac OS X Leopard Server

Microsoft Azure and Hyper-V

- Microsoft has included the Hyper-V hypervisor in Microsoft Server 2008 (code-named Viridian)
  - Hyper-V is also available in a free, reduced-function stand-alone version. However, Hyper-V, which provides partition-level isolation, is not the basis for the Hypervisor in Microsoft Azure
- That was written, like the rest of Azure, from the ground up and is particularly optimised for multitenancy
- The Windows Azure Hypervisor is tightly optimised with the Windows Azure kernel. However, Microsoft has stated that some of the features of the Azure Hypervisor will ultimately make their way into the next version of Hyper-V

Virtual machines

Virtual machine

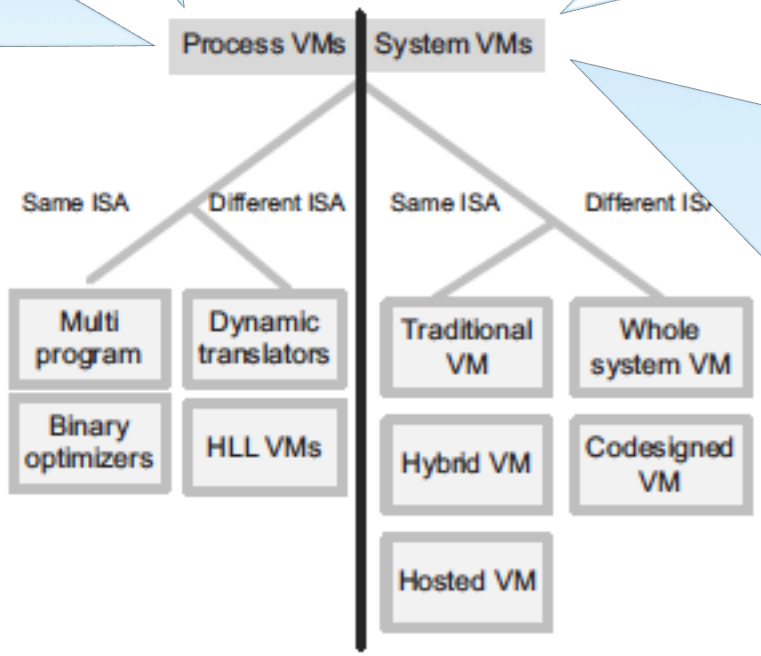
- A Virtual Machine (VM) is an isolated environment that appears to be a whole computer, but actually only has access to a portion of the computer resources
- Each virtual machine appears to be running on the bare hardware, giving the appearance of multiple instances of the same computer, though all are supported by a single physical system
- Virtual machines have been around since early 1970s when IBM released its VM 370 operating system.

VM types

A process VM is a virtual platform created for an individual process and destroyed once the process terminates

A system VM supports an operating system together with many user processes. When the VM runs under the control of a normal OS and provides a platform-independent host for a single application we have an application virtual machine, e.g., Java Virtual Machine (JVM)

Virtually all operating systems provide a process VM for each one of the applications running, but the more interesting process VMs are those which support binaries compiled on a different instruction set

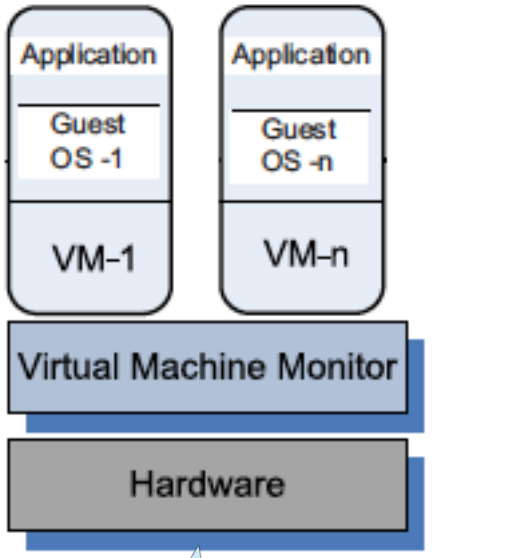


A system virtual machine provides a complete system; each VM can run its own OS, which in turn can run multiple applications. Systems such as Linux Vserver, OpenVZ, FreeBSD Jails, and Solaris Zones based on Linux, FreeBSD, and Solaris, respectively, implement operating system-level virtualisation technologies.

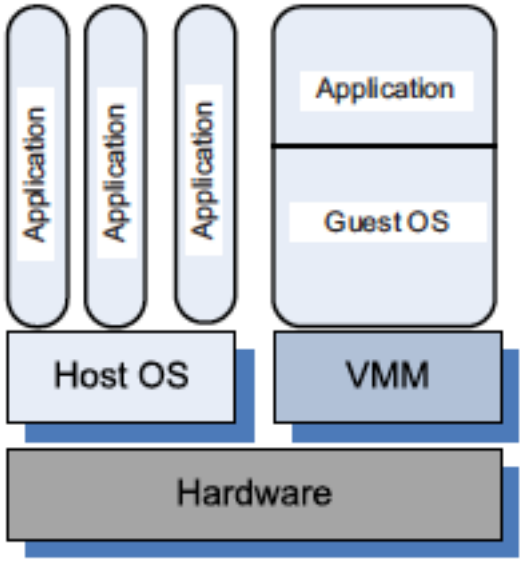
System VMs

- The operating system-level virtualisation allows a physical server to run multiple isolated operating system instances subject to several constrains
- The instances are known as containers, Virtual Private Servers (VPSs), or Virtual Environments (VEs)
  - For example, OpenVZ requires both the host and the guest OS to be Linux distributions. These systems claim performance advantages over the systems based on a Virtual Machine Monitor such as Xen or Vmware, there is only a 1% to 3% performance penalty for OpenVZ as compared to a standalone Linux server
- Several organisations of the software stack are possible: traditional, hybrid, and hosted are three classes of VMs for systems with the same ISA

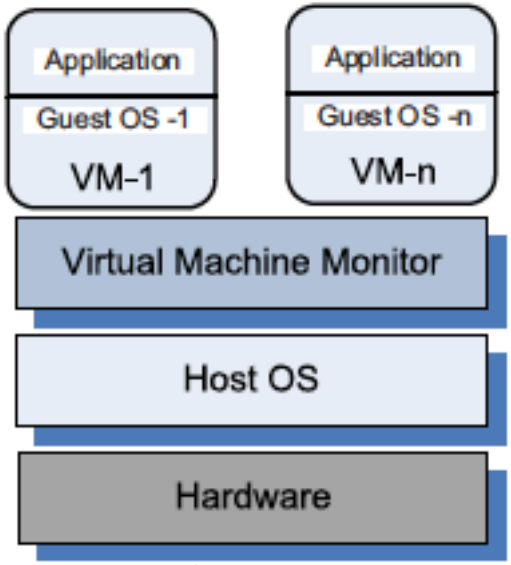
System VM stacks



Traditional VMs: the VMM supports multiple virtual machines and runs directly on the hardware



Hybrid VM: the VMM shares the hardware with a host operating system and supports multiple virtual machines



Hosted VM: the VMM runs under a host operating system

System VMs stacks

- Traditional – a VM also called a "bare metal" VMM – a thin software layer that runs directly on the host machine hardware
  - Its main advantage is performance
  - Examples: VMWare ESX, ESXi Servers, Xen, OS370, and Denali
- Hybrid – the VMM shares the hardware with existing OS.
  - Example: VMWare Workstation

System VMs stacks

- Hosted – the VM runs on top of an existing OS
  - The main advantage of this approach is that the VM is easier to build and install. Also, the VMM could use several components of the host OS, such as the scheduler, the pager and the I/O drivers, rather than providing its own
  - A price to pay for this simplicity is the increased overhead and the associated performance penalty; indeed, the I/O operations, page faults, and scheduling requests from a guest OS are not handled directly by the VMM, instead they are passed to the host OS
  - Performance, as well as the challenges to support complete isolation of VMs make this solution less attractive for servers in a cloud computing environment
  - Example: User-mode Linux.

Performance and security isolation

Performance isolation

- Performance isolation is a critical condition for Quality of Service (QoS) guarantees in shared computing environments.
- Indeed, if the run-time behavior of an application is affected by other applications running concurrently and thus, competing for CPU cycles, cache, main memory, disk and network access, it is rather difficult to predict the completion time
- Moreover, it is equally difficult to optimise the application

Processor virtualisation

- Processor virtualisation presents multiple copies of the same processor or core on multicore systems
- The code is executed directly by the hardware, while processor emulation presents a model of another hardware system
  - Instructions are "emulated" in software much slower than virtualisation
- An example is Microsoft's VirtualPC that could run on chipsets other than the x86 family
  - It was used on Mac hardware until Apple adopted Intel chips

Performance isolation

- Traditional operating systems multiplex multiple processes or threads while a virtualisation supported by a VMM (Virtual Machine Monitor) multiplexes full operating systems
- Obviously, there is a performance penalty as an OS is considerably more heavyweight than a process and the overhead of context switching is larger
- A VMM executes directly on the hardware a subset of frequently used machine instructions generated by the application and emulates privileged instructions including device I/O requests
- The subset of the instructions executed directly by the hardware includes arithmetic instructions, memory access and branching instructions

Performance isolation

- Operating systems use the process abstraction not only for resource sharing but also to support isolation
- Unfortunately, this is not sufficient from a security perspective, once a process is compromised it is rather easy for an attacker to penetrate the entire system
- On the other hand, the software running on a virtual machine has the constrains of its own dedicated hardware: it can only access virtual devices emulated by the software
- This layer of software has the potential to provide a level of isolation nearly equivalent to the isolation presented by two different physical systems
- Thus, the virtualisation can be used to improve security in a cloud computing environment.

Security isolation

- A VMM is a much simpler and better specified system than a traditional operating system; for example, the Xen VMM has approximately 60 000 lines of code while the Denali VMM has only about half, 30 000 lines of code
- The security vulnerability of VMMs is considerably reduced as the systems expose a much smaller number of privileged functions; for example, the Xen VMM can be accessed through 28 hypercalls while a standard Linux allows hundreds, e.g., Linux 2.6.11 allows 289 system calls
- In addition to a plethora of system calls a traditional operating system supports special devices (e.g., /dev/kmem) and many privileged programs from a third party (e.g., sendmail and sshd).

Full virtualisation and paravirtualisation

Virtualisation conditions

- In 1974 Gerald J. Popek and Robert P. Goldberg gave a set of sufficient conditions for a computer architecture to support virtualisation and allow a VMM to operate efficiently:
  - A program running under the VMM should exhibit a behaviour essentially identical to that demonstrated when running on an equivalent machine directly
  - The VMM should be in complete control of the virtualised resources.
  - A statistically significant fraction of machine instructions must be executed without the intervention of the VMM

Virtualisation conditions

- Another way to identify an architecture suitable for a virtual machine is to distinguish two classes of machine instructions:
  - sensitive, instructions which require special precautions at execution time and innocuous:
    - Control sensitive, instructions that attempt to change either the memory allocation, or the privileged mode
    - Mode sensitive, instructions whose behaviour is different in the privileged mode,
  - instructions that are not sensitive

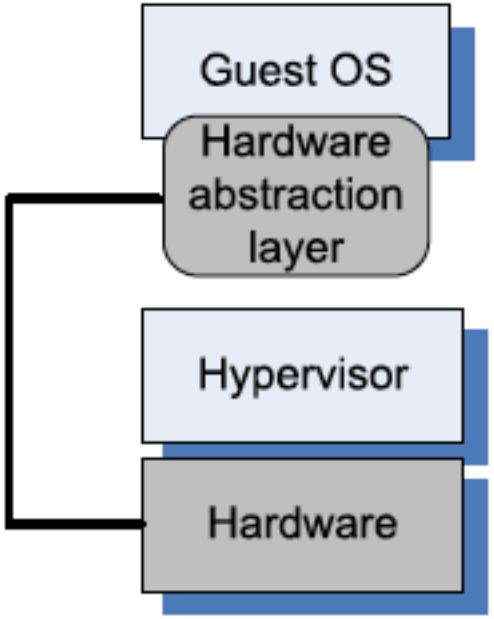
Virtualisation conditions

- An equivalent formulation of the conditions for efficient virtualisation can be based on this classification of machine instructions:
  - a VMM for a third or later generation computer can be constructed if the set of sensitive instructions is a subset of the privileged instructions of that machine. To handle non-virtualisable instructions one could resort to two strategies:
    - Binary translation. The VMM monitors the execution of guest operating systems; non-virtualisable instructions executed by a guest operating system are replaced with other instructions.
    - Paravirtualisation. The guest operating system is modified to use only instructions that can be virtualised

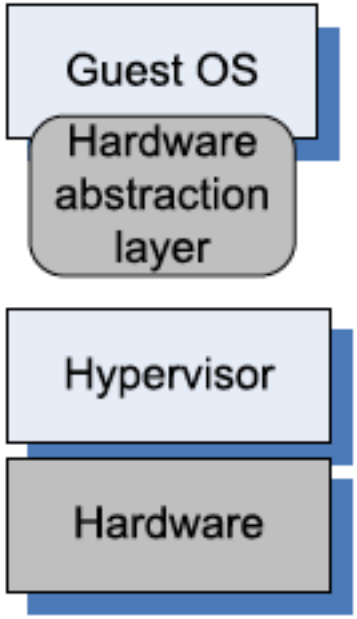
Two approaches

- There are two basic approaches to processor virtualisation:
  - Full virtualisation when each virtual machine runs on an exact copy of the actual hardware
  - Paravirtualisation when each virtual machine runs on a slightly modified copy of the actual hardware
- The reasons why paravirtualisation is often adopted are:
  - some aspects of the hardware cannot be virtualised
  - to improve performance
  - to present a simpler interface
- VMware VMMs are example of full virtualisation. Xen and Denali are based on paravirtualisation

Full vs. paravirtualisation



The full virtualisation requires the hardware abstraction layer of the guest OS to have some knowledge about the hardware



The paravirtualisation avoids this requirement and allows full compatibility at the Application Binary Interface (ABI)

Full vs. paravirtualisation

- Full virtualisation requires a virtualisable architecture
  - The hardware is fully exposed to the guest OS which runs unchanged and this ensure that this direct execution mode is efficient
- Paravirtualisation is done because some architectures such as x86 are not easily virtualisable
  - Paravirtualisation demands that the guest OS be modified to run under the VMM; also the guest OS code must be ported for individual hardware platforms

Full vs. paravirtualisation

- Systems such as VMware EX Server support full virtualisation on x86 architecture
- The virtualisation of the MMU (Memory Management Unit) and the fact that privileged instructions executed by a guest OS fail silently pose some challenges, e.g.:
  - To address the later problem one has to insert traps whenever privileged instructions are issued by a guest OS
  - The system must also maintain shadow copies of system control structures, such as page tables, and trap every event affecting the state of these control structures
  - The overhead of many operations is substantial

Performance and cache isolation

- Application performance under a virtual machine is critical
  - Generally, virtualisation adds some level of overhead which affects negatively the performance
- In some cases an application running under a virtual machine performs better than one running under a classical OS
  - This is the case of a policy called cache isolation. The cache is generally not partitioned equally among processes running under a classical OS, as one process may use the cache space better than the other

Performance and cache isolation

- For example:
  - In the case of two processes, one write-intensive and the other read-intensive, the cache may be aggressively filled by the first. Under the cache isolation policy the cache is divided between the VMs and it is beneficial to run workloads competing for cache in two different VMs

Other performance factors

- The application I/O performance running under a virtual machine depends on factors such as, the disk partition used by the VM, the CPU utilisation, the I/O performance of the competing VMs, and the I/O block size
  - On a Xen platform discrepancies between the optimal choice and the default are as high as 8% to 35%

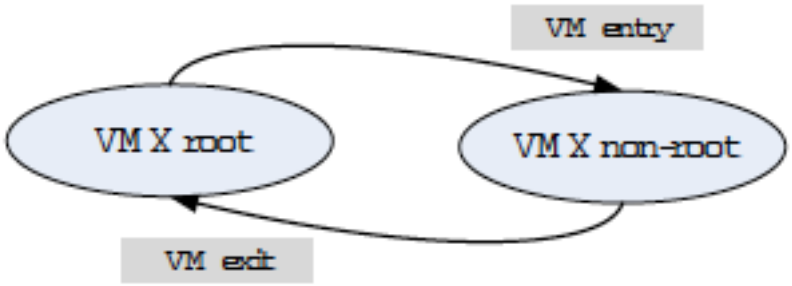
Hardware support

Beginnings

- In the early 2000 it became obvious that hardware support for virtualisation was necessary and Intel and AMD started work on the first generation virtualisation extensions of the x8659 architecture
  - In 2005 Intel released two Pentium 4 model supporting VT-x and in 2006 AMD announced Pacifica and then several Athlon 64 models

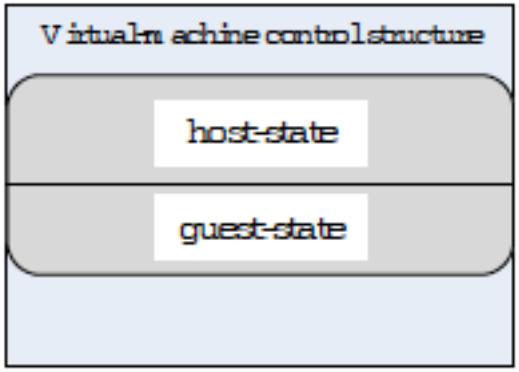
Current state

- A major architectural enhancement provided by the VT-x is the support for two modes of operations and a new data structure called the Virtual Machine Control Structure (VMCS) including host-state and guest-state areas:
  - VMX root: intended for VMM operations, and very close to the x86 without VT-x
  - VMX non-root: intended to support a VM



Current state

- The Virtual Machine Control Structure (VMCS) includes host-state and guest-state areas which control the VM entry and VM exit transitions:



Current state

- When executing a VM entry operation the processor state is loaded from the it guest-state of the VM scheduled to run
- Then the control is transferred from the VMM to the VM
- A 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, and finally transfers control to the VMM
- Note that all VM exit use a common entry point to the VMM

Current state

- Each VM exit operation saves the reason for the exit and eventually some qualifications in VMCS
- Some of this information is stored as bitmaps
  - For example, the exception bitmap specifies which one of 32 possible exceptions caused the exit. The I/O bitmap contains one entry for each port in a 16-bit I/O space

Current state

- The VMCS area is referenced with a physical address and its layout is not fixed by the the architecture but can be optimised by a particular implementation
- The VMCS includes control bits that facilitate the implementation of virtual interrupts
  - For example, external interrupt exiting when set causes the execution of a VM exit operation; moreover the guest is not allowed to mask these interrupts
  - When the interrupt window exiting is set a VM exit operation is triggered if the guest is ready to receive interrupts

Current state

- Processors based on two new virtualisation architectures, VT-d (AMD-Vi in AMD) and VT-c have been developed
  - The first supports the I/O Memory Management Unit (I/O MMU) virtualisation and the second network virtualisation
  - Also known as PCI pass-through the I/O MMU virtualisation 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

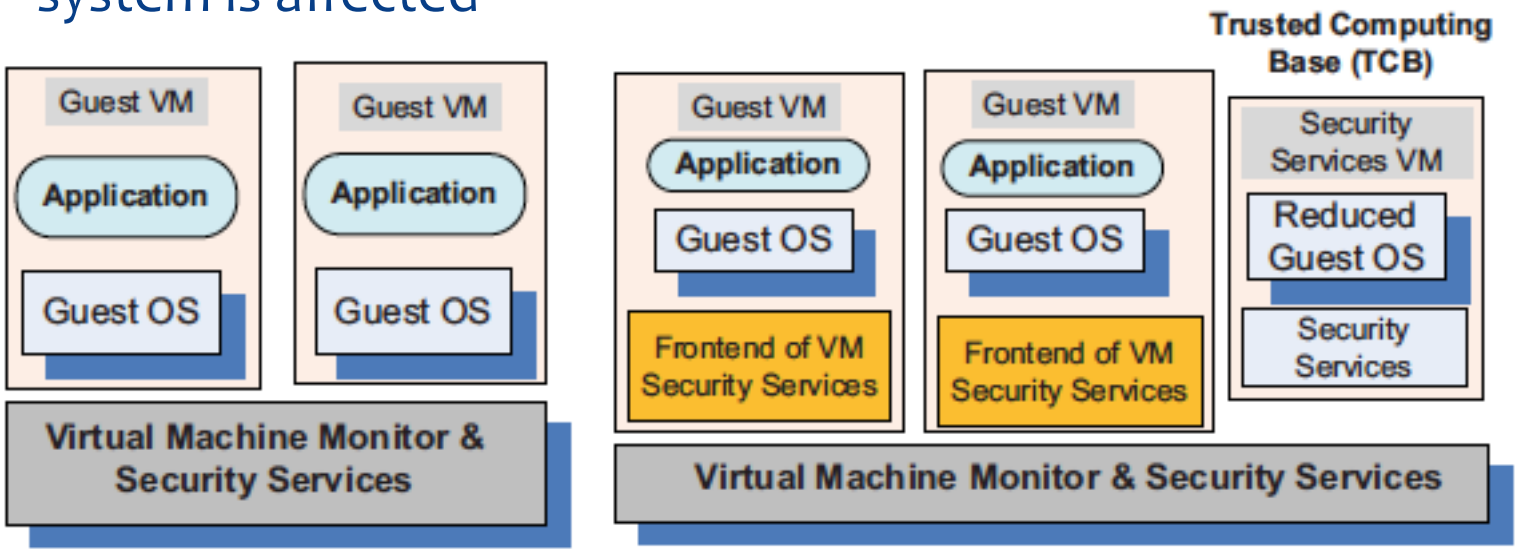
VM security

Traditional, hybrid and hosted VMs

- The hybrid and the hosted VM models expose the entire system to the vulnerability of the host operating system thus, we will not analyse these models
- Our discussion of virtual machine security is restricted to the traditional system VM model when the Virtual Machine Monitor controls the access to the hardware

Traditional VMs

- Virtual security services are typically provided by the VMM
- Another alternative is to have a dedicated security services VM
  - A secure TCB (Trusted Computing Base) is a necessary condition for security in a virtual machine environment
  - If the TCB is compromised then the security of the entire system is affected



Intrusion detection

- A recent VM-based intrusion detection systems such as Livewire and Siren exploit the three capabilities of a virtual machine for intrusion detection:
  - Isolation
    - Discussed in details in the following slides,
  - Inspections
    - The VMM has the ability to review the state of the guest VMs
  - Interposition
    - The VMM can trap and emulate the privileged instruction issued by the guest VMs

Isolation

- The analysis of Xen and vBlades shows that the VM technology provides a stricter isolation of virtual machines from one another than the isolation of processes in a traditional operating system.
  - A Virtual Machine Monitor controls the execution of privileged operations and can thus enforce memory isolation as well as disk and network access
- The VMMs are considerably less complex and better structured than traditional operating systems thus, in a better position to respond to security attacks
- A major challenge is that a VMM sees only raw data regarding the state of a guest operating system while security services typically operate at a higher logical level, e.g., at the level of a file rather than a disk block

Isolation

- A guest OS runs on simulated hardware and the VMM has access to the state of all virtual machines operating on the same hardware
- The state of a guest virtual machine can be saved, restored, cloned, and encrypted by the VMM. Replication can ensure not only reliability but also support security, while cloning could be used to recognise a malicious application by testing it on a cloned system and observing if it behaves normally
- We can also clone a running system and examine the effect of potentially dangerous applications
- Another interesting possibility is to have the guest VM's files moved to a dedicated VM and thus, protect it from attacks
  - This is possible because inter-VM communication is faster than communication between two physical machines.

Other attack types

- Sophisticated attackers are able to fingerprint virtual machines and avoid virtual machine honey pots designed to study the methods of attack
- They can also attempt to access VM-logging files and thus, recover sensitive data
  - such files have to be very carefully protected to prevent unauthorised access to cryptographic keys and other sensitive data

Price

- There is no free lunch thus, we expect to pay some price for the better security provided by virtualisation:
  - Higher hardware costs because a virtual system requires more resources such as CPU cycles, memory, disk, network bandwidth
  - The cost of developing VMMs and modifying the host operating systems in case of paravirtualisation
  - The overhead of virtualisation as the VMM is involved in privileged operations

VMM threats (by NIST)

- Starvation of resources and denial of service for some VMs. Probable causes:
  - badly configured resource limits for some VMs
  - a rogue VM with the capability to bypass resource limits set in VMM
- VM side-channel attacks: malicious attack on one or more VMs by a rogue VM under the same VMM. Probable causes:
  - lack of proper isolation of inter-VM traffic due to misconfiguration of the virtual network residing in the VMM
  - limitation of packet inspection devices to handle high speed traffic, e.g., video traffic
  - presence of VM instances built from insecure VM images, e.g., a VM image having a guest OS without the latest patches
- Buffer overflow attacks

VM threats (by NIST)

- Deployment of rogue or insecure VM; unauthorised users may create insecure instances from images or may perform unauthorised administrative actions on existing VMs. Probable cause:
  - improper configuration of access controls on VM administrative tasks such as instance creation, launching, suspension, reactivation and so on.
- Presence of insecure and tampered VM images in the VM image repository. Probable causes:
  - lack of access control to the VM image repository
  - lack of mechanisms to verify the integrity of the images, e.g., digitally signed image.

Malware problems

- Can virtualisation empower the creators of malware to carry out their mischievous activities with impunity and minimal danger of being detected?
- How difficult is it to implement such a system?
- What are the means to prevent this type of malware to be put in place?

Malware problems

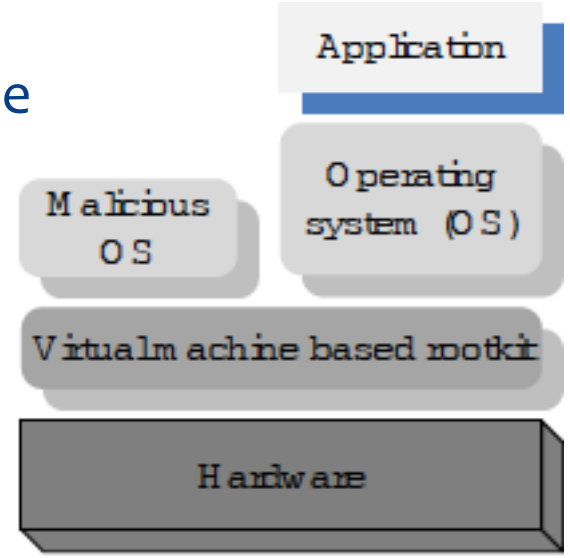
- It is well understood that in a layered structure a defense mechanism at some layer can be disabled by malware running at a layer below it
- Thus, the winner in the continuous struggle between the attackers and the defenders of a computing system is the one in control of the lowest layer of the software stack, the one which controls the hardware

Malware problems

- A VMM allows a guest operating system to run on virtual hardware
  - the VMM offers to the guest operating systems a hardware abstraction and mediates its access to the physical hardware.
  - A VMM is simpler and more compact than a traditional operating system thus, it is more secure
  - But what if the VMM itself is forced to run above another software layer thus, it is prevented to exercise direct control of the physical hardware?

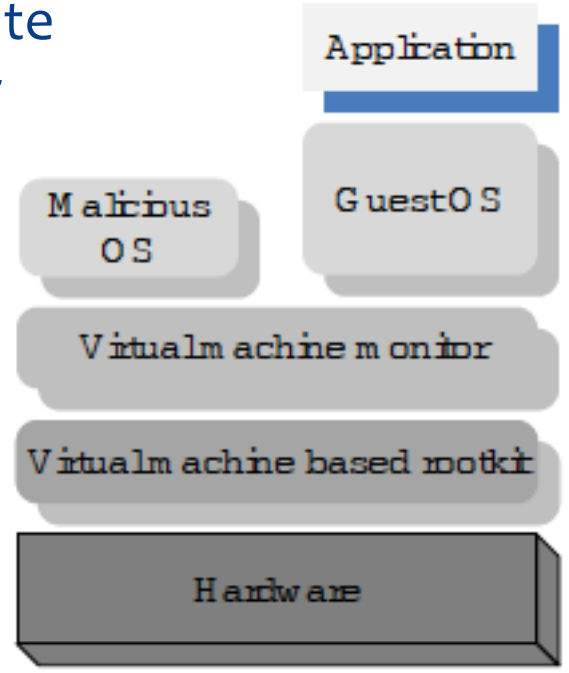
Malware problems

- Is it feasible to insert a "rogue VMM" between the physical hardware and an operating system?
  - Such a rogue VMM is called a Virtual-Machine Based Rootkit (VMBR)
    - The term rootkit refers to malware with a privileged access to a system; the name comes from root, the most privileged account on a Unix system, and kit, a set of software components
  - The VMBR enables a malicious OS to run surreptitiously and makes it invisible to the genuine or the guest OS and to the application



Malware problems

- Is it also feasible to insert the VMBR between the physical hardware and a "legitimate VMM"
  - As a virtual machine running under a legitimate VMM sees a virtual hardware, the guest OS will not notice any change of the environment
  - So the only trick is to present the legitimate VMM with a hardware abstraction, rather than allow it to run on the physical hardware



Malware problems

- The malware runs either inside a VMM or with the support of a VMM
  - A VMM is a very potent engine for the malware, it prevents the software of the guest operating system or the application to detect malicious activities
  - A VMBR can record key strokes, system state, data buffers sent to, or received from the network, data to be written to, or read from the disk with impunity; moreover, it can change any data at will

Malware problems

- The only way for a VMBR to take control of a system is to modify the boot sequence and to first load the malware and only then load the legitimate VMM, or the operating system
  - This is only possible if the attacker has root privileges
  - Once the VMBR is loaded it must also store its image on the persistent storage