The University for
world-class professionals

Manchester Metropolitan University

Advanced Operating Systems
Lecture 5: Virtualisation

Soufiene Djahel
Office: John Dalton E151

Email: s.djahel@mmu.ac.uk

Telephone: 0161 247 1522

Office hours: Monday 10 -11, Thursday 11-13

This lecture was adapted and extended from the slides originally designed by Emma Norling, MMU, UK

Recap from last week

- Distributed file systems (DFS)
- DFS requirements
- DFS examples:
  - NFS, AFS, NTFS
  - GFS, HDFS

Today’s objectives

- To explore the use of virtualisation technologies in operating systems
  - Considering different reasons for virtualisation
- To describe the various approaches to virtualisation
- To discuss performance issues in virtualisation

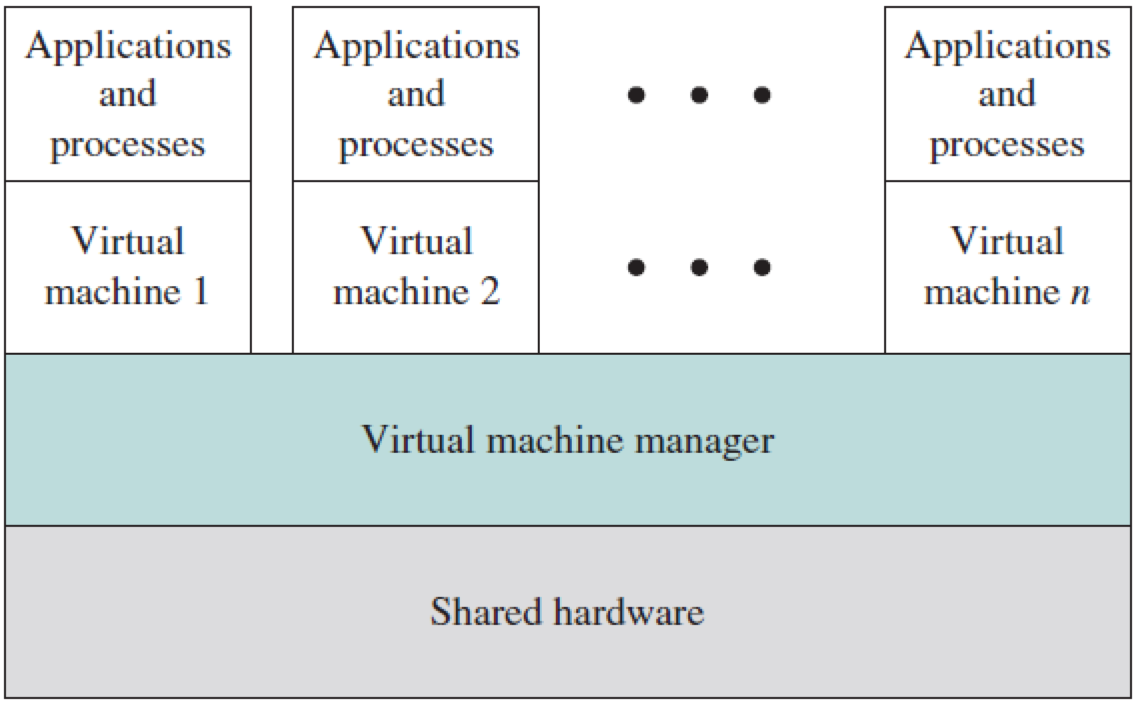
Virtualisation

- Virtualization technology enables a single PC or server to simultaneously run multiple operating systems or multiple sessions of a single OS
- A machine with virtualization software can host numerous applications, including those that run on different operating systems, on a single platform

Virtualisation (cont.)

- The solution that enables virtualization is a ***Virtual Machine Monitor (VMM)***, or ***hypervisor***
- This software sits between the hardware and the VMs
  - acting as a resource broker
  - allowing multiple VMs to safely coexist on a single physical server host and share that host's resources

Virtualisation (cont.)



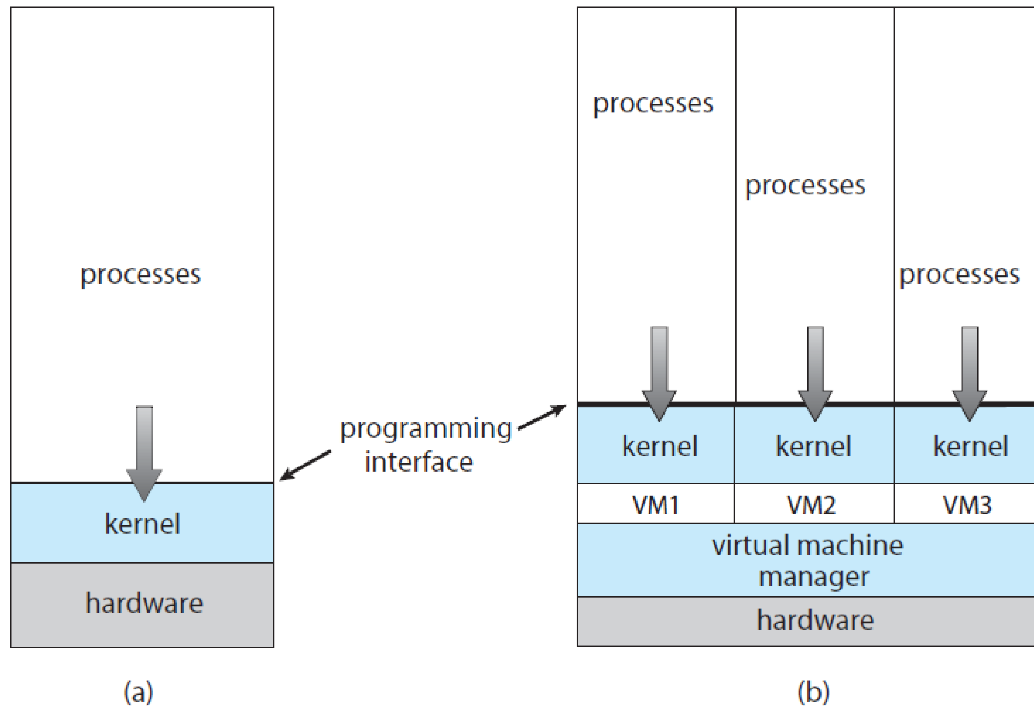
Virtual machine concept

Virtualisation (cont.)

- Several components
  - **Host** – underlying hardware system
  - **Virtual Machine Monitor** (VMM) or **hypervisor** – creates and runs virtual machines by providing interface that is ***identical*** to the host
    - Except in the case of paravirtualization
  - **Guest** – process provided with virtual copy of the host
    - Usually an operating system

Virtualisation (cont.)

- Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

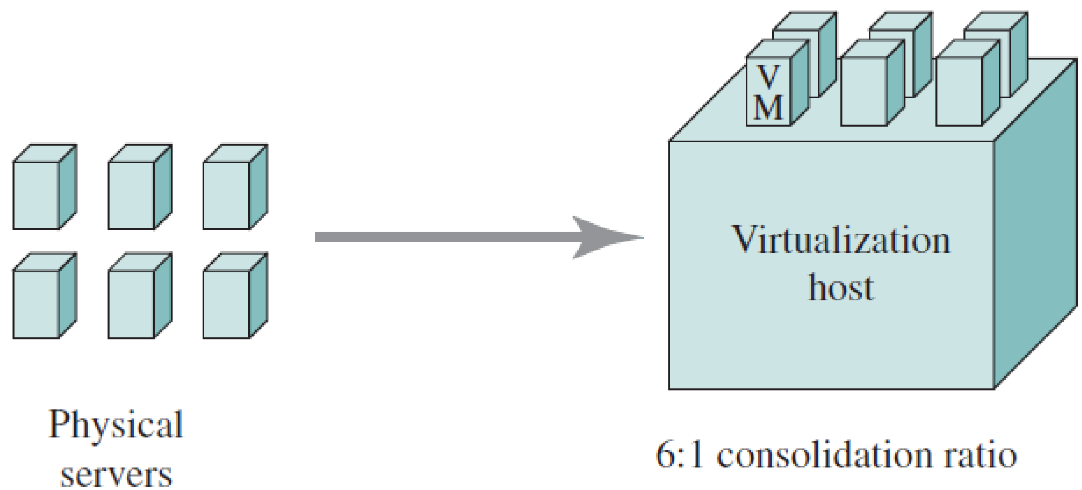


Non-virtual machine

Virtual machine

Consolidation ratio

- The number of Guests that can exist on a single host is measured as Consolidation ratio
  - e.g., a host supporting 6 VMs has a consolidation ratio of 6 to 1, written as 6:1



Virtual machine consolidation

History

- Virtual machines invented by IBM in late 1960s
- Original usage:
  - Each user ran a different guest OS
  - Single shared hardware platform
- Interest died out in the 1980's and 1990's:
  - Each user had a private machine
- Reinvented, made practical by Mendel Rosenblum and graduate students at Stanford, led to VMware.

Why Use Virtualisation?

- Cloud computing (IaaS)
- Multiple OSes on one machine
  - Multi-platform development/support
- Fault isolation
- Encapsulation

Encapsulation

- VMM can encapsulate entire state of a VM in a file.
- Can save, continue, restore old state.
- Datacenter example:
  - Can migrate VM's between machines to balance load
- Software development:
  - Tests may corrupt the state of the machine
  - Solution:
    - Run tests in a VM
    - Always start tests from a saved VM configuration
    - Discard VM state after tests
    - Results: reproducible tests

Supporting Virtualisation

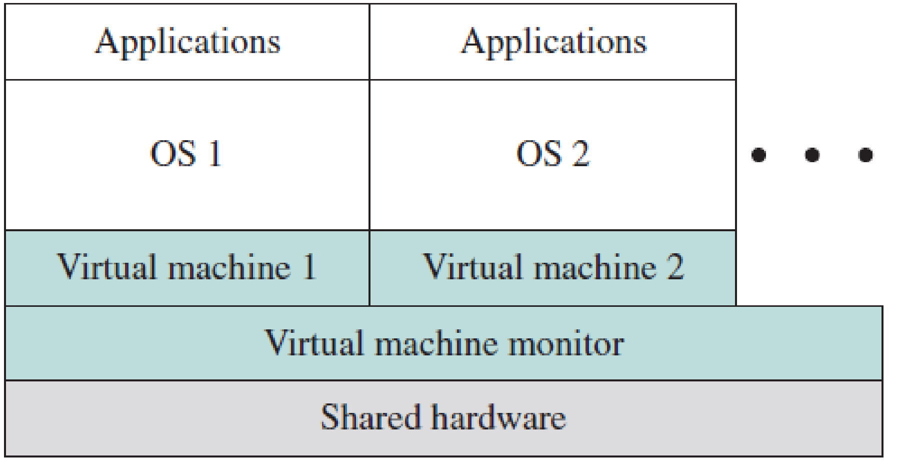
- Need a process that is an abstraction of a machine, which looks exactly like the underlying hardware:
  - The complete instruction set of the underlying machine
  - Physical memory
  - Memory management unit (page tables, etc.)
  - I/O devices
  - Traps and interrupts
- Virtual Machine Monitor (VMM) (aka *hypervisor*)

Virtual Machine Monitor (VMM)

- Implements sharing of real hardware resources by multiple OS VMs that each think they have a complete copy of the machine
  - Popular in early days of computing to allow mainframe to be shared by multiple groups developing OS code
  - Used in modern mainframes to allow multiple versions of OS to be running simultaneously → OS upgrades with no downtime!
  - Example for PCs: VMware allows Windows OS to run on top of Linux (or vice-versa)

VMM types

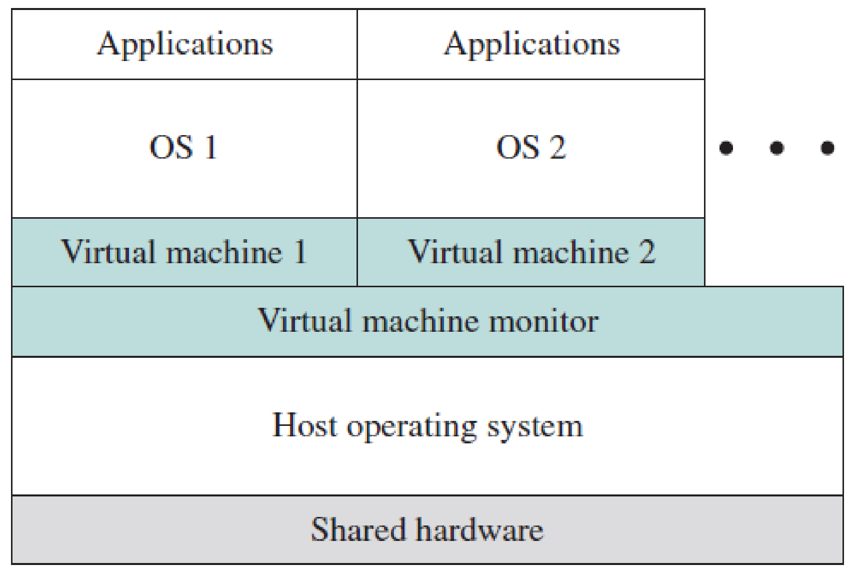
- Type 1 VMM, aka "bare metal" VMM
  - Loaded as a thin software layer into the physical server
  - VMM *is* the host OS
  - Thus, it can directly control the resources of the host
  - E.g., VMware ESXi. Microsoft Hyper -V



(a) Type 1 VMM

VMM types

- Type 2 VMM, hosted by the underlying OS
  - Works as a traditional app, a program code loaded on top of OS
  - Relies on the OS to handle all hardware interactions
  - It has lower performance than Type 1 VMM
  - E.g., VMware Workstation, ORACLE Virtual Box



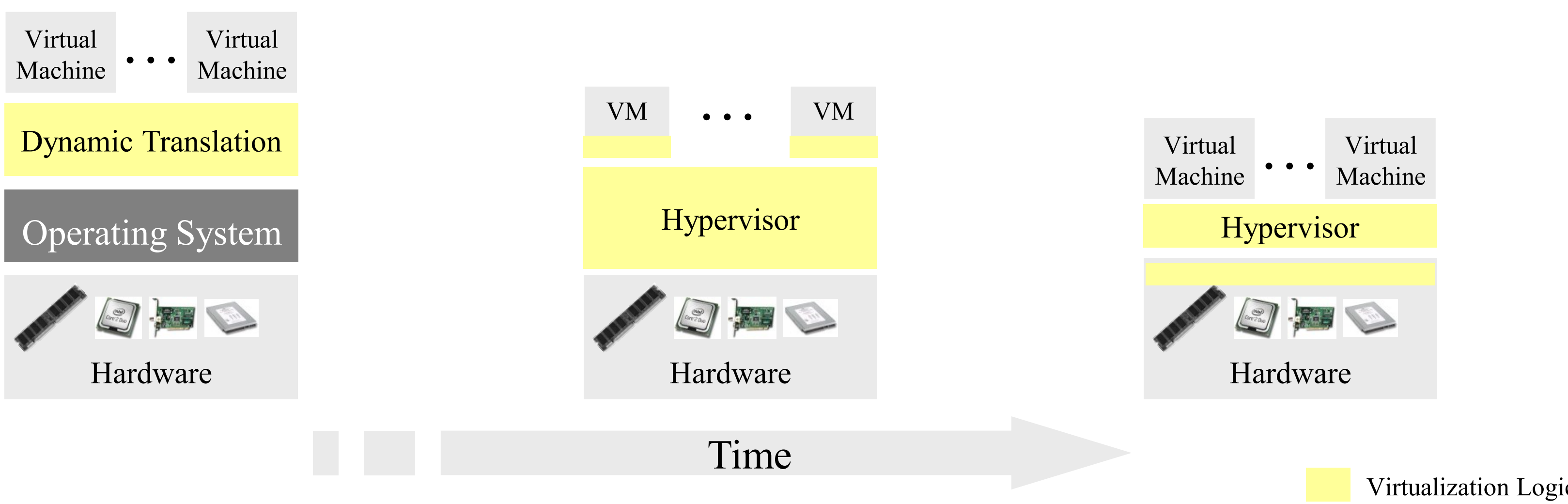
(b) Type 2 VMM

Approaches to VMMs

- Simulation
  - sometimes called emulation, but it is not full emulation
- Use CPU, but trap on access to privileged hardware state
- Paravirtualisation
- Hardware-assisted virtualisation

Evolution of Software solutions*

- 1st Generation: Full virtualization (Binary rewriting)
  - Software Based
  - VMware and Microsoft
- 2nd Generation: Paravirtualization
  - Cooperative virtualization
  - Modified guest
  - VMware, Xen
- 3rd Generation: Silicon-based (Hardware-assisted) virtualization
  - Unmodified guest
  - VMware and Xen on virtualization-aware hardware platforms



*This slide is from Intel® Corporation

Simulation

- Most basic form of virtualisation
- All underlying hardware is simulated
  - e.g. QEMU
- SLOW

Instruction Traps

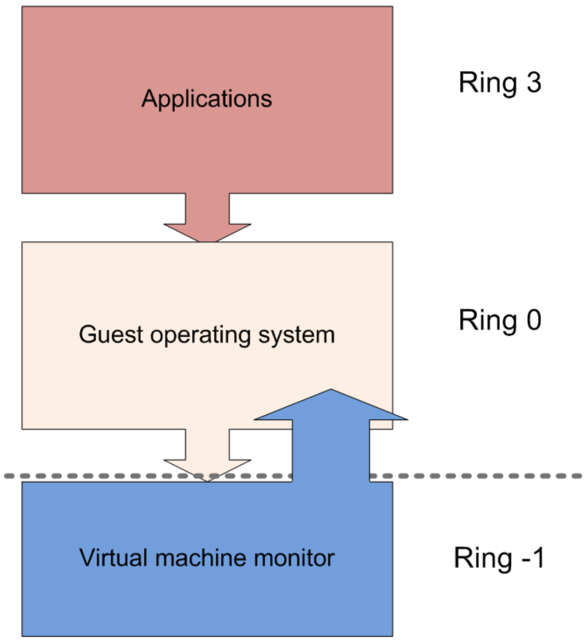
- Most common form of virtualisation
- Guest OS runs as an application program
  - Attempts to access privileged instructions cause exceptions
- When guest OS issues a privileged instruction, VMM traps it, takes appropriate action, and returns result

Paravirtualisation

- Used to improve performance
- Guest OSes “know” that they are modified
- Make use of specialist instructions to improve speed and reliability
- Must be supported by underlying hardware *and* guest OS

Hardware-assisted Virtualization

- Server hardware is virtualization aware
- VMM loads at privilege Ring -1 (firmware)
- Removes CPU emulation bottleneck
- Memory virtualization coming in quad core AMD and Intel CPUs



Hardware-assisted virtualization

What If We Can't Trap…?

- If there's no way to tell when guest OS is trying to access privileged instructions…
  - One possibility: VMM checks *every* instruction when it is executed
    - costly, but not as bad as simulation
  - Binary translation: Instruction set translated in advance to replace sections that require virtualisation
    - Upfront cost, but can also be optimised

- Although VMM isolates VMs, they can affect each other
  - resource usage
- Different VMMs manage resource usage in different ways
- In cloud computing, additional options of *migration exist*
  - VM moves from one VMM (and one underlying hardware platform) to another

VMM overhead

- Depends on workload
- Goal for system VMs:
  - Run almost all instructions directly on native hardware
- User-level CPU-bound programs have near-zero virtualization overhead
  - Run at native speeds since OS rarely invoked

VMM overhead (cont.)

- I/O-intensive workloads are OS-intensive
  - Execute many system calls and privileged instructions
  - Can result in high virtualization overhead
- But if I/O-intensive workload is also I/O-bound
  - Processor utilization is low (since waiting for I/O)
  - Processor virtualization can be hidden in I/O costs
  - So virtualization overhead is low

Full Virtualisation

Processor Management

- Two main strategies:
  1. Emulate chip as software (e.g. QEMU or Android Emulator in Android SDK).
     - Easily transportable but inefficient
  2. Provide each VM with segments of time on the physical processors (pCPUs) to host the virtual processors (vCPUs)
     - Number of processors (vCPUs) is important metric in system
     - In general, CPUs are underutilised ➔ start with one, add extras if needed

Processor Management (cont.)

- Operating systems in general have at least two levels of protection: kernel mode and user mode
- Most systems have *four* levels of protection, described as rings:
  - Ring 0 is most privileged, where kernel runs
  - Ring 3 is least privileged, where user processes run
  - Rings 1 & 2 in theory for intermediate levels of protection, such as device drivers
- A guest OS believes to be operating at ring 0, but in reality is only doing so on the virtualised machine
  - In terms of the underlying hardware, it's operating in ring 3

Memory Management

- A VM needs to be configured with enough memory to function efficiently
- If each guest OS in every VM manages its own set of page tables, what happens to the virtual memory managed by the underlying OS?

Memory Management (cont.)

- Three abstractions of memory:
  - Real (aka machine) memory: actual hardware memory, e.g., 8GB of DRAM
  - Physical memory: abstraction of hardware memory managed by a guest OS
    - If the VMM allocates 512 MB to a VM, its OS thinks the computer has 512MB of contiguous physical memory
  - Virtual memory:
    - a continuous virtual address space presented by the guest OS to applications running inside the VM.

Memory Management (cont.)

- VMM separates real and physical memory
  - Makes real memory a separate, intermediate level between virtual memory and physical memory
  - Guest OS maps virtual memory to physical memory via its page tables
  - VMM page tables map real memory to physical memory

Memory Management (cont.)

- VMM manages page sharing, so the virtual machine OSs are unaware of what is happening in the physical system
  - Multiple VMs share pages if they run similar version of OS or the same applications

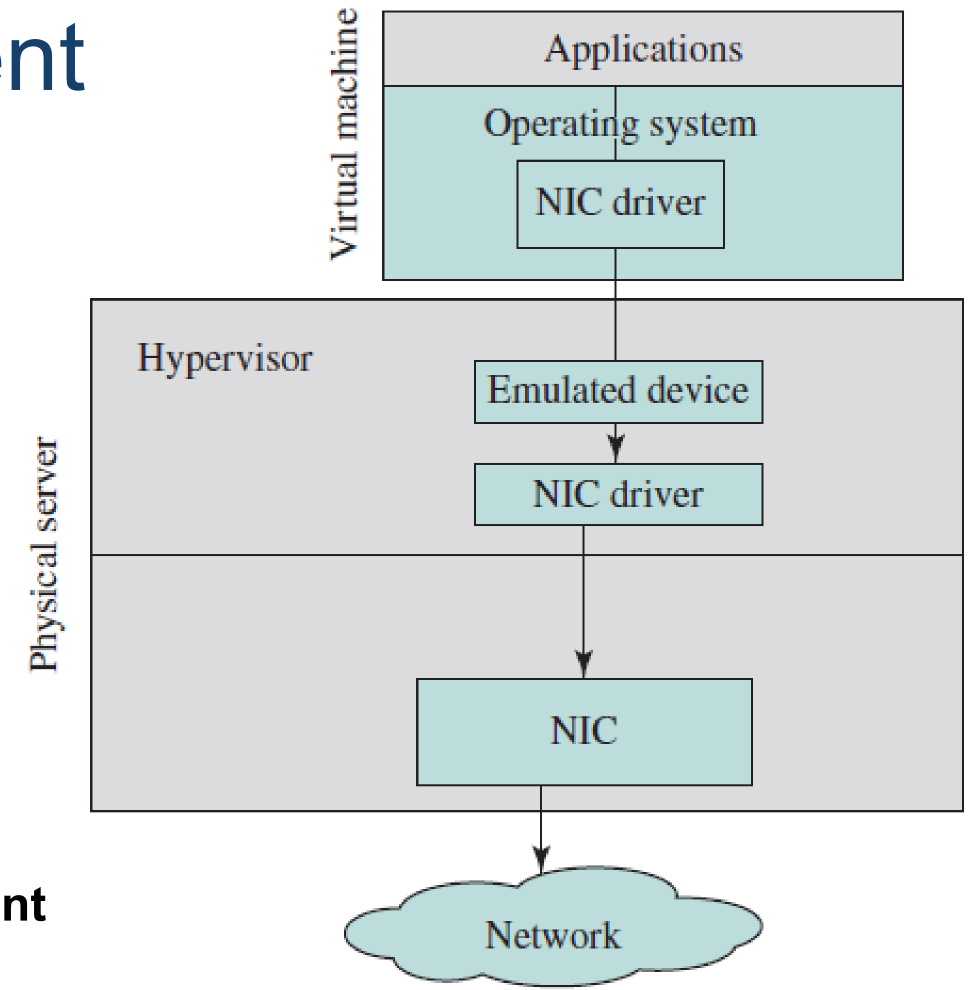
Memory Management (cont.)

- Ballooning
  - the VMM activates a balloon driver that (virtually) inflates and presses the guest OS to flush pages to disk
  - once the pages are cleared, the balloon driver deflates and the VMM can use the physical memory for other VMs
- Memory overcommit
  - the capability to allocate more memory than physically exists on a host

I/O Management

- Most difficult part of virtualization
  - Increasing number of I/O devices attached to the computer
  - Increasing diversity of I/O device types
  - Sharing of a real device among multiple VMs,
  - Supporting the myriad of device drivers that are required, especially if different guest OSes are supported on the same VM system

I/O Management



I/O in a virtual environment

I/O Management

- An advantage of virtualizing the workload's I/O path is enabling hardware independence by abstracting vendor-specific drivers to more generalized versions that run on the hypervisor
- This abstraction enables:
  - live migration, which is one of virtualization's greatest availability strengths
  - the sharing of aggregate resources, such as network paths

I/O Management

- The memory overcommit capability is another benefit of virtualizing the I/O of a VM
- The trade-off for this is that the hypervisor is managing all the traffic and requires processor overhead
  - this was an issue in the early days of virtualization but now faster multicore processors and sophisticated hypervisors have addressed this concern

VMM Example

Java VM – not a VM in the same sense!

- The goal of a Java Virtual Machine (JVM) is to provide a runtime space for a set of Java code to run on any operating system staged on any hardware platform without needing to make code changes to accommodate the different operating systems or hardware
- The JVM can support multiple threads
- Promises "Write Once, Run Anywhere"

Java VM

- Example of programming-environment virtualization
- Includes language specification (Java), API library, Java virtual machine (JVM)
- Java objects specified by class construct, Java program is one or more objects

Java VM

- The JVM is described as being an abstract computing machine consisting of:
    - an instruction set
    - a program counter register
    - a stack to hold variables and results
    - a heap for runtime data
    - a method area for code and constants

Android Virtual Machine

- Referred to as Dalvik
- The Dalvik VM (DVM) executes files in the Dalvik Executable (.dex) format
- The Dalvik core class library is intended to provide a familiar development base for those used to programming with Java Standard Edition, but it is geared specifically to the needs of a small mobile device
- Each Android application runs in its own process, with its own instance of the Dalvik VM
- Dalvik has been written so that a device can run multiple VMs efficiently

Summary

- Range of reasons for virtualisation
  - Cloud computing is just one
- Various approaches to virtualisation
- Virtualisation leads to performance issues
- Range of commercial and open-source examples

Interesting VM tools

- Xen:
  - https://www.xenproject.org/
  - https://xenserver.org/
- VMWare: https://www.vmware.com/
- VirtualBox: https://www.virtualbox.org/

Next lecture: Fault tolerance