CSE490H: Virtualization

It's turtles all the way down…

Steve Gribble
Associate Professor, CSE
[on sabbatical at Google as a visiting scientist]

Some simple terms

a *virtual machine:*

- a software-based implementation of some real (hardware-based) computer
- in its pure form, supports booting and execution of unmodified OSs and apps

a *virtual machine monitor*:

- the software that creates and manages the execution of virtual machines
- a VMM is essentially a simple operating system

VM demo

Outline

- **The history of virtualization**
- How virtualization works
- Applications of virtualization

Before there were data centers…

Many early commercial computers were *mainframes:*

- originally housed in enormous, room-sized metal frames
- computationally powerful, though less so than a supercomputer
- highly reliable, with redundancy engineered into hardware and software
- extensive I/O capabilities for data-intensive business and scientific apps
- “IBM and the seven dwarfs” – their heyday was the late ‘50s through ‘70s

IBM 704 (1954)

$250K - millions

IBM z9 (2005)

$100k - millions

Issues with early mainframes

Early mainframe families had some disadvantages

- successive (or even competing!) models were not architecturally compatible
  - massive headache to upgrade HW: gotta port software!
- the systems were primarily batch-oriented

In the meantime, project MAC at MIT was kicking off

- responsible for developing Multics
- invented many of the modern ideas behind time-sharing operating systems
  - e.g., fundamentals of protection systems (access control lists, capabilities)
- the computer was becoming a multiplexed tool for a community of users, instead of being a batch tool for wizard programmers
  - and the mainframe companies were about to be left in the dust

Big blue's bold move

IBM bet the company on the **System/360** hardware family [1964]

- S/360 was the first to clearly distinguish architecture and implementation
- its architecture was virtualizable (with the addition of virtual memory support in the 360-67)

And, unexpectedly, the **CP/CMS** system software is a hit [1968]

- CP: a "control program" that created and managed virtual S/360 machines
- CMS: the "Cambridge monitor system" -- a lightweight, single-user OS
  - run several different OSs concurrently on the same HW
    - one CMS instance per user: CP/CMS is now great for timesharing!
    - older, batch-oriented jobs on batch-oriented OSs (PCP)
    - presumably, any of the other s/360 compatible OSs (OS/360, DOS/360, etc.)
    - any S/360 software could run in a VM, and hence became time-sharable
- CP/CMS also enabled OS development and experimentation

Thus began the family tree of IBM mainframes

- system/360 (1964-1970)
  - ended up supporting virtualization via CP/CMS, channel I/O, virtual memory, byte-addressable, 32-bit registers with 24 bit addressing, EBCDIC, …
  - several orders of magnitude of performance and cost
- system/370 (1970-88)
  - shipped as dual-processors, virtual memory support via DAT boxes, moved to 31-bit architecture; reimplementation of CP/CMS OS as VM/370
- system/390 (1990-2000)
  - clustering, aka “parallel sysplex”
- zSeries (2000-present)
  - hot hardware swap and failover, redundant software execution, wide-area failover

Huge moneymaker for IBM, and many business still depend on these!

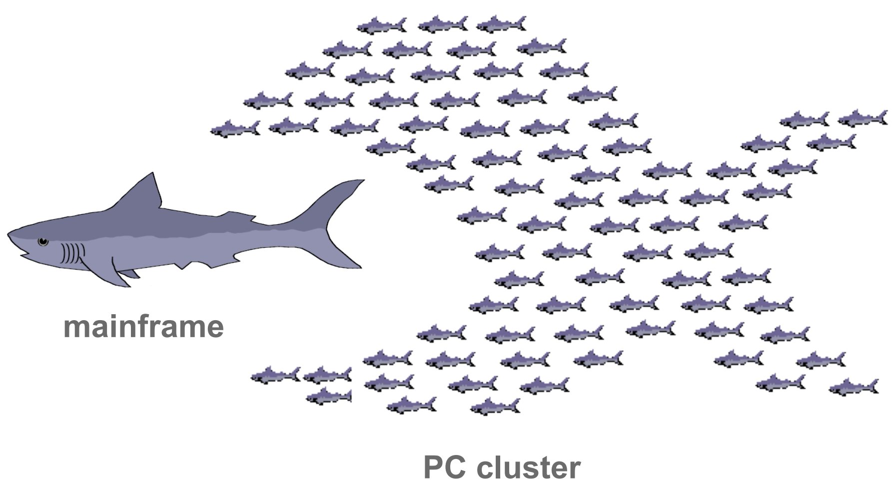
In the meantime…the PC revolution happened

PCs are much less powerful, but enjoy massive ***economies of scale***

- “a computer for every desktop”   (1980s)
- ship hundreds of millions of units, not hundreds of units
- much better price/performance (operations per $)
- much lower reliability

Cluster computing (1990s)

- build a cheap mainframe or supercomputer out of a cluster of commodity PCs
- use clever software to get fault tolerance



Mendel Rosenblum makes it big

VMware co-founded by Mendel Rosenblum and Diane Green in 1998

- commercialized ideas incubated in Stanford DISCO project
- brought CP/CMS-style virtualization to PC computers

Their initial market was software developers

- often need to develop and test software on multiple OSs (windows, linux, …)
  - (or, similar to CP/CMS, might want to do OS development)
- can afford multiple PCs, or could dual-boot, but this is very inconvenient
- instead, run multiple OSs simultaneously in separate VMs
  - very similar to mainframe VM motivation, but for opposite reason – too many computers now, not too few!

The real PC virtualization moneymaker

Enterprise consolidation

- big companies usually have their own machine rooms or data centers
  - operate many services: mail servers, file servers, Web servers, remote cycles
  - want to run at most **one service per machine** (administrative best practices)
  - leads to low utilization, lots of machines, high power bills, administrative hassles
- instead, run **one service per virtual machine**
  - and consolidate many VMs per physical machine
  - leads to better utilization, easier management

The forefront of virtualization

Large-scale, hosted cloud computing (e.g., Amazon EC2)

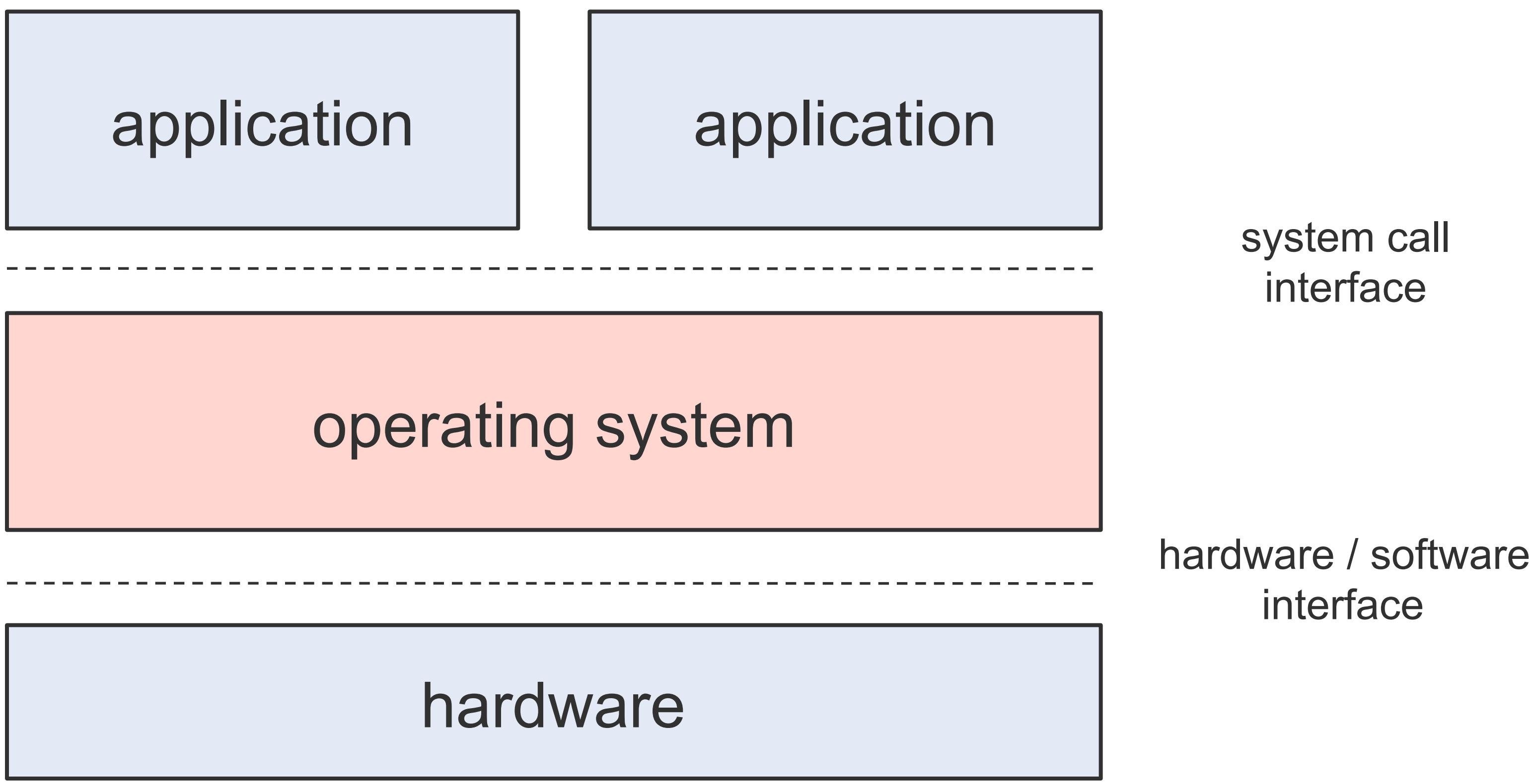
- the cloud provider buys a bazillion computers and operates a data center
- your run your software in a VM on their computers, and pay them rent
  - the VM is a convenient container for uploading software, and is a safe sandbox that prevents you and other customers from harming each other
- run 1,000 VMs images for a day, and pay just $2400.00.

Outline

- The history of virtualization
- **How virtualization works**
- Applications of virtualization

How do virtual machines work?

Start with a “simpler” question: how do (regular) machines work?



What is computer hardware?

Just a bag of devices…

- CPU
  - defines the instruction set of the machine
  - provides registers, processes instructions, handles interrupts
  - defines privilege modes (e.g., supervisor, user)
- memory hierarchy
  - physical memory words accessible via load/store instructions
  - MMU provides paging / segmentation, and therefore virtual memory support
  - MMU controlled via special registers, and via page tables (see CSE451)
- I/O devices
  - disks, NICs, etc., controlled by programmed I/O (inb, outb) or by DMA (load/store)
  - events delivered to software via polling or interrupts
- Other devices
  - graphics cards, clocks, USB controllers, etc.

What is an OS?

It's just a program!

- you write it in some language (C/C++), and compile it into a program image
- it runs like any other program, but in a privileged (supervisor) CPU mode
  - this allows it to interact with hardware devices using "sensitive" instructions

Looking downwards:

- an OS issues instructions to control hardware devices
- it does so to allocate and manage hardware resources on behalf of programs

Looking upwards:

- OS gives apps a high-level programming interface (system call interface)
- OS implements this interface using low-level hardware devices
  - file open / read / write close vs. disk block read / write

What’s an application?

A program that relies on the system call interface

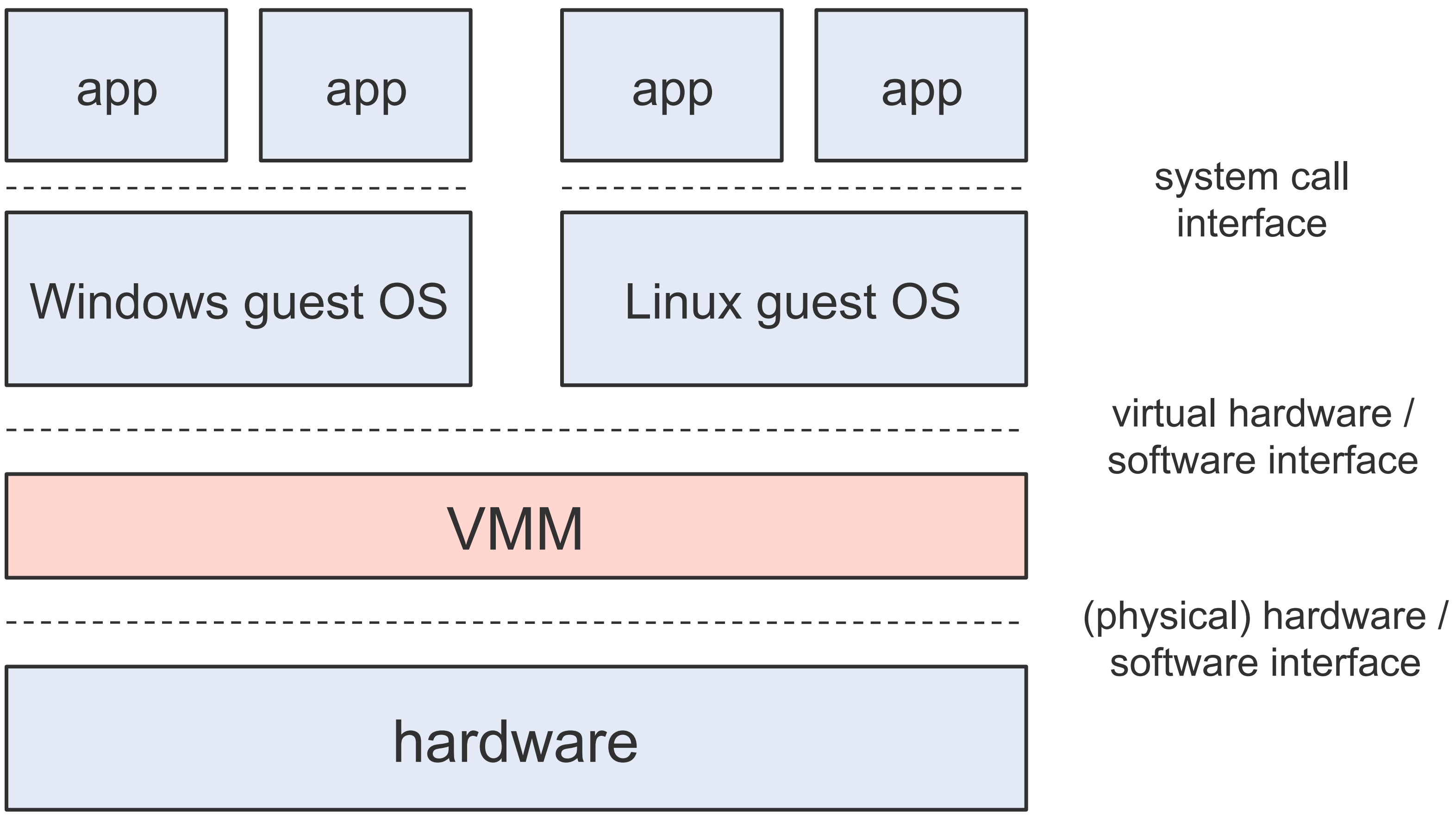
- While executing it, the CPU runs in unprivileged (user) mode
- a special instruction (“intc” on x86) lets a program call into the OS
    - the OS uses this to expose system calls
    - the program uses system calls to manipulate file system, network stack, etc.
- OS provides a program with the illusion of its own memory
    - MMU hardware lets the OS define the “virtual address space” of the program

Is this safe?

- most instructions run directly on the CPU (fast)
    - but sensitive instructions cause the CPU to throw an exception to the OS
- address spaces prevent program from stomping on OS memory, each other
- it’s as though each program runs in its own, private machine (the “process”)

Here’s the goofy idea…

What if we run the Windows kernel as a user-level program?



The goofy idea almost works, but…

What happens when Windows issues a sensitive instruction?

What (virtual) hardware devices should Windows see?

How do you prevent apps running on Windows from hurting Windows?

- or apps from hurting the VMM…
- or Windows from hurting Linux…or the VMM…

Trap-and-emulate, and Goldberg

Answer: rely on CPU to trap sensitive instructions and hand off to VMM

- VMM emulates the effect of sensitive instruction on the virtual hardware that it provides to its guest OSs
- instead of OS providing high-level abstractions to process via system calls…
  - VMM provides a virtual HW/SW interface to guest OSs by trapping and emulating sensitive instructions

Goldberg (1974): two classes of instructions

- **privileged** instructions: those that trap when CPU is in user-mode
- **sensitive** instructions: those that modify hardware configuration or resources, and those whose behavior depends on HW configuration

A VMM can be constructed efficiently and safely if the set of sensitive instructions is a subset of the set of privileged instructions.

Performance implications of trap-and-emulate

There is almost no overhead to non-sensitive instructions

- they execute directly on the CPU, and do not cause traps
- CPU-bound code (e.g., many SPEC benchmarks, some scientific programs) execute at the same speed on a VM as on a physical machine

There is a large potential performance hit to sensitive instructions

- they raise a trap and must be vectored to and emulated by VMM
- I/O or system-call intensive applications get hit hard
  - recent hardware extensions try to improve this by letting the hardware handle instructions that used to cause trap/emulate
  - in essence, these extensions make the CPU aware of VM boundaries

A hard problem (and why VMware made $$)

Until 2005, the Intel architecture did not meet Goldberg's requirement

- 17 instructions were not virtualizable
- they do not trap, and they behave differently in supervisor vs. user mode
  - some leak processor mode (e.g., SMSW, or store machine status word)
  - some behave differently (e.g., CALL or JMP to addresses that reference the protection mode of the destination)

How to make Intel virtualizable

You have four choices…

1. **Emulate**: do not execute instructions directly, but instead interpret each
   - very slow (Virtual PC on the Mac)
2. **Paravirtualize**: modify the guest OS to avoid non-virtualizable instructions
   - very fast and safe, but not "pure" or backwards compatible (Denali, Xen)
3. Use **binary translation** instead of trap-and-emulate.
   - this is rocket science; and it is what VMware does
4. **Fix the CPUs.**
   - In 2005/2006, Intel introduced "VT", and AMD introduced "Pacifica"
     - re-implemented ideas from VM/370 virtualization support
     - basically added a new CPU mode to distinguish VMM from guest/app
   - now building a VMM is easy!
     - and VMware must make money some other way…

Outline

- The history of virtualization
- How virtualization works
- **Applications of virtualization**

Cool properties of VM-based systems

A full-blown computer image can be stored in a file

- VMM manifestly sees all of the state of the virtual hardware
  - virtual disk blocks, virtual (physical) memory pages, virtual CPU registers, virtual I/O device state, etc.
- if the VMM saves all this state into a file, it has created a VM snapshot
  - and if it loads this state from a file, it is restoring a VM from a snapshot
- Pop quiz: if all you save in the snapshot is the disk state, what do you have?

You can copy VM image to a new machine and run it there (migration)

- install a complicated app in an image, and ship it (virtual appliances)
- optimize the copy, and do the copy while the VM is running (live migration)

More cool properties of VMs

A virtual machine is a (pretty) secure sandbox

- run malicious code in a VM, and it won't harm other VMs or the host OS
  - e.g., run a web browser in a VM and not worry about malware
  - what assumption does this make?

The VMM can observe and log all HW/SW interactions of its guest OSs

- log non-deterministic interactions to build a flight-data-recorder for replay
  - forensics, software-based fault tolerance, time-travel debugging, …

The virtual data center

A cluster of machines, each running a set of VMs

- drive up utilization by packing many VMs onto each cluster node
- fault recovery is simplified
  - if hardware fails, copy VM image elsewhere
  - if software fails, restart VM from snapshot
- can safely allow third parties to inject VM images into your data center
  - hosted VMs in the sky, commercial computing grids

Pop quiz:

- should a big cloud app provider (Google, Yahoo, Microsoft, …) run VMs on all of its machines?

Amazon web services

EC2, S3 etc.

- customer uploads and runs Xen virtual machines; Amazon charges:
  - 10 cents per CPU hour
  - 10 cents per GB-month of storage
  - 10 cents per million I/O requests
  - 10 cents per wide-area network EC GB in, 17 per GB out.
- is very much a low-level utility
  - you decide what software images to run
  - you must manage your fleet of virtual machine images
  - you get to worry about fault tolerance, scalability (sharding), etc.
- ecosystem is growing around it
  - third-party companies like RightScale help solve these problems, if you run LAMP

For comparison, Google's AppEngine

Let's customers implement and execute Web services on Google's machines

- programmers write to a Python-based execution environment
  - you implement code to handle a Web request
  - your code can store and retrieve data from something that looks like BigTable
- Google figures out…
  - how many machines to run your code on
  - how to route requests to your machines
  - where to store your data, and how to manage data replication
  - how to hide faults from you and your users
  - the geolocation of your code
- Google chose to rely on Python + OS as sandbox, rather than a VM