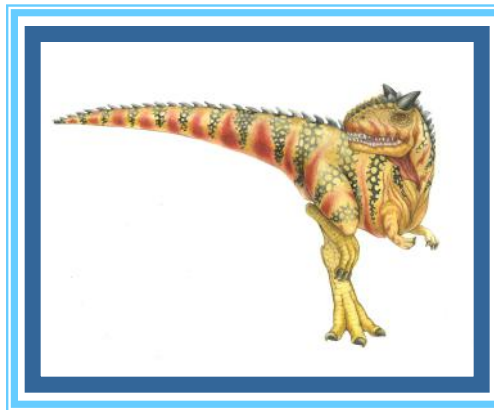


Chapter 16: Virtual Machines



Wisdom from Alex Stamos

- Be part of the product, not the plumbing
- You have the most leverage when you have an offer.
- Make friends in the industry and network.
- Be nice, we are in a small industry.

Superblock

- <http://www.linfo.org/superblock>

But First

- Most interesting man in the world
- VMWare
- To have more data on disk, the tracks have gotten narrow.
- This is not like a record with grooves. The tracks are getting squeezed.
- There is little margin for error.
- Yelling at a disk drive

Chapter 16: Virtual Machines

- Overview
- History
- Benefits and Features
- Building Blocks
- Types of Virtual Machines and Their Implementations
- Virtualization and Operating-System Components
- Examples

Chapter Objectives

- To explore the history and benefits of virtual machines
- To discuss the various virtual machine technologies
- To describe the methods used to implement virtualization
- To show the most common hardware features that support virtualization and explain how they are used by operating-system modules

Before VM, lets talk about Rings

- User Code Runs at Ring Level 3
- Kernel Code Runs at Ring Level 0
- https://en.wikipedia.org/wiki/Protection_ring#/media/File:Priv_rings.svg
- Intel Management Engine
 - Not a lot is known about the IME
 - Runs at Ring Level -3?!?!?
 - https://en.wikipedia.org/wiki/Intel_Management_Engine
- <https://www.cnx-software.com/2017/11/07/minix-based-intel-management-engine-firmware-uefi-are-closed-source-insecure-nerf-to-the-rescue/>

Overview

- Fundamental idea – **abstract hardware of a single computer** into several different execution environments
 - Similar to layered approach
 - But layer creates virtual system (**virtual machine**, or **VM**) on which operation systems or applications can run
- Several components
 - **Host** – underlying hardware system
 - **Virtual machine manager (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
- Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

Hypervisors



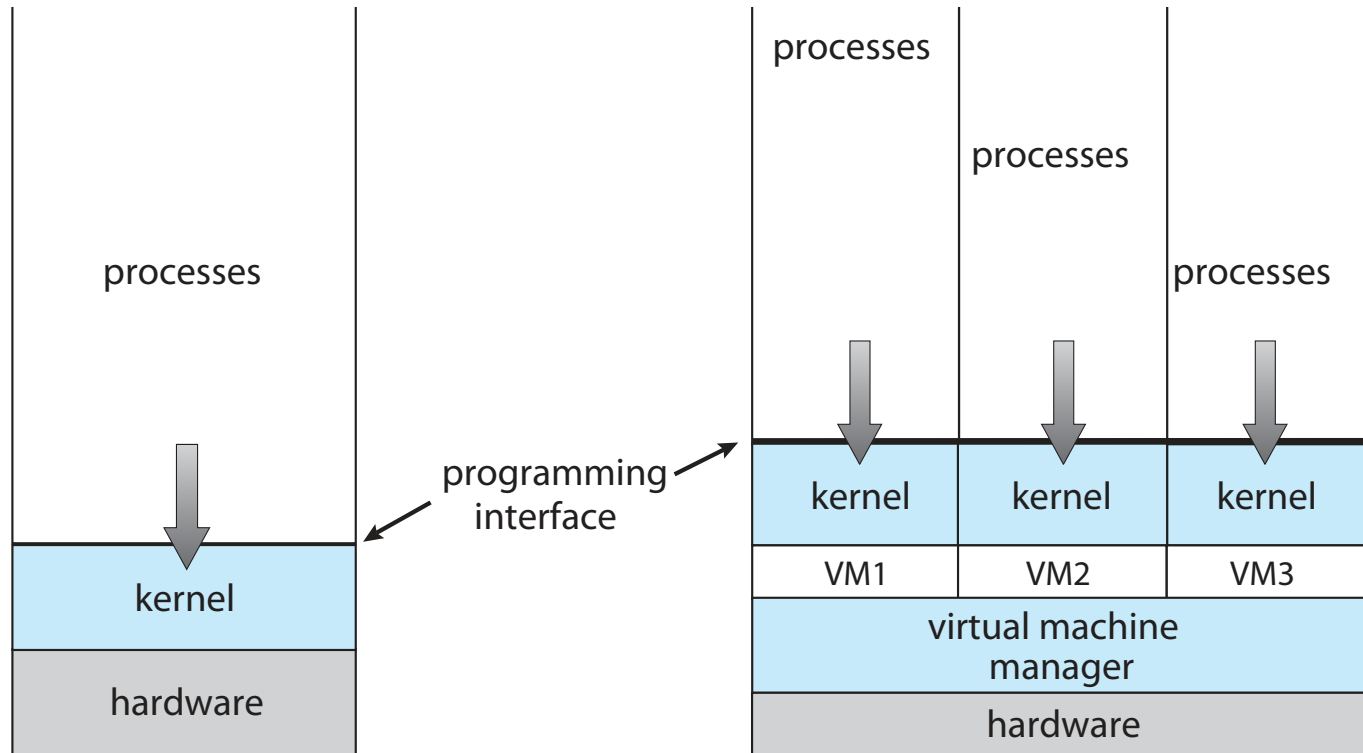
2009



VirtualBox



System Models



Non-virtual machine

Virtual machine

Implementation of VMMs

- Vary greatly, with options including:
 - **Type 0 hypervisors** - Hardware-based solutions that provide support for virtual machine creation and management via firmware
 - ▶ IBM LPARs and Oracle LDOMs are examples
 - **Type 1 hypervisors** - Operating-system-like software built to provide virtualization
 - ▶ Including VMware ESX, Joyent SmartOS, and Citrix XenServer
 - **Type 1 hypervisors** – Also includes general-purpose operating systems that provide standard functions as well as VMM functions
 - ▶ Including Microsoft Windows Server with HyperV and RedHat Linux with KVM
 - **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems
 - ▶ Including VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox

Implementation of VMMs (cont.)

■ Other variations include:

- **Paravirtualization** - Technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance
- **Programming-environment virtualization** - VMMs do not virtualize real hardware but instead create an optimized virtual system
 - ▶ Used by Oracle Java and Microsoft.Net
- **Emulators** – Allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU. The CPU is being emulated. Slow.
- **Application containment** - Not virtualization at all but rather provides virtualization-like features by segregating applications from the operating system, making them more secure, manageable
 - ▶ Including Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs

■ Much variation due to breadth, depth and importance of virtualization in modern computing

History

- First appeared in IBM mainframes in 1972
- Allowed multiple users to share a batch-oriented system
- Formal definition of virtualization helped move it beyond IBM
 1. A VMM provides an environment for programs that is essentially identical to the original machine
 2. Programs running within that environment show only minor performance decreases
 3. The VMM is in complete control of system resources
- In late 1990s Intel CPUs fast enough for researchers to try virtualizing on general purpose PCs
 - **Xen** and **VMware** created technologies, still used today
 - Virtualization has expanded to many OSES, CPUs, VMMs

Benefits and Features

- Host system protected from VMs, VMs protected from each other
 - I.e. A virus less likely to spread
 - Sharing is provided via shared file system volume, network communication
- Freeze, **suspend**, running VM
 - Then move or copy somewhere else and **resume**
 - Snapshot of a given state, able to restore back to that state
 - ▶ Some VMMs allow multiple snapshots per VM
 - **Clone** by creating copy and running both original and copy
- Great for OS research, better system development efficiency
- Run multiple, different OSes on a single machine
 - **Consolidation**, app dev, ...

Benefits and Features (cont.)

- **Templating** – create an OS + application VM, provide it to customers, use it to create multiple instances of that combination
 - https://solutionexchange.vmware.com/store/category_groups/19
 - Great for a classroom
- **Live migration** – move a running VM from one host to another!
 - No interruption of user access
- All those features taken together -> **cloud computing**
 - Using APIs, programs tell cloud infrastructure (servers, networking, storage) to create new guests, VMs, virtual desktops

Benefits and Features (cont.)

- **Consolidation**– less hardware to break, power, space and cooling.
- **Flexibility** – Bring up and down instances on demand
- **Thin Client** – Build a new machine each time you use it. Move the heat out of the office where the people are into a machine room perhaps miles away.
- **Backwards compatibility** – backwards compatibility

<https://www.tnhh.net/posts/google-maps-insane-backward-compatibility.html>

Benefits and Features (cont.)

- **Security** – for a personal VM, especially when doing secure transactions, this might discourages Ransomware and tracking
 - <https://blogs.vmware.com/vcloud/2017/03/security-landscape-pwn2own-2017.html>

- **Qubes OS** - <https://www.qubes-os.org/video-tours/>
 - Does not work on a virtual machine such as VMWare
 - Does not work on every computer due it being open source software.
 - Also, it is more or less an still in the Alpha stage of maturity.

Building Blocks

- Generally difficult to provide an **exact** duplicate of underlying machine
 - Especially if only dual-mode (user, kernel) (Ring 0, Ring3) operation available on CPU
 - But getting easier over time as CPU features and support for VMM improves
 - Most VMMs implement **virtual CPU (VCPU)** to represent state of CPU per guest as guest believes it to be
 - ▶ When guest context switched onto CPU by VMM, information from VCPU loaded and stored
 - Several techniques, as described in next slides

Building Block – Trap and Emulate

■ What is a trap?

- <https://www.shellscript.sh/trap.html>
- Dual mode CPU means
 - ▶ guest executes in user mode
 - ▶ Kernel runs in kernel mode
- Not safe to let guest kernel run in kernel mode too
- So VM needs two modes – virtual user mode and virtual kernel mode
 - ▶ Both of which run in real user mode???
- Actions in guest that usually cause switch to kernel mode must cause switch to virtual kernel mode
- I think of this like learning to ride a bicycle, but you have training wheels. You are not going to fall over.

Trap-and-Emulate (cont.)

■ Privileged Instructions

- <http://www.cs.princeton.edu/courses/archive/spr01/cs217/slides/21.os.pdf>
- User mode code in guest runs at same speed as if not a guest. Emulation will require more instructions.

■ But kernel mode privilege mode code runs slower due to trap-and-emulate

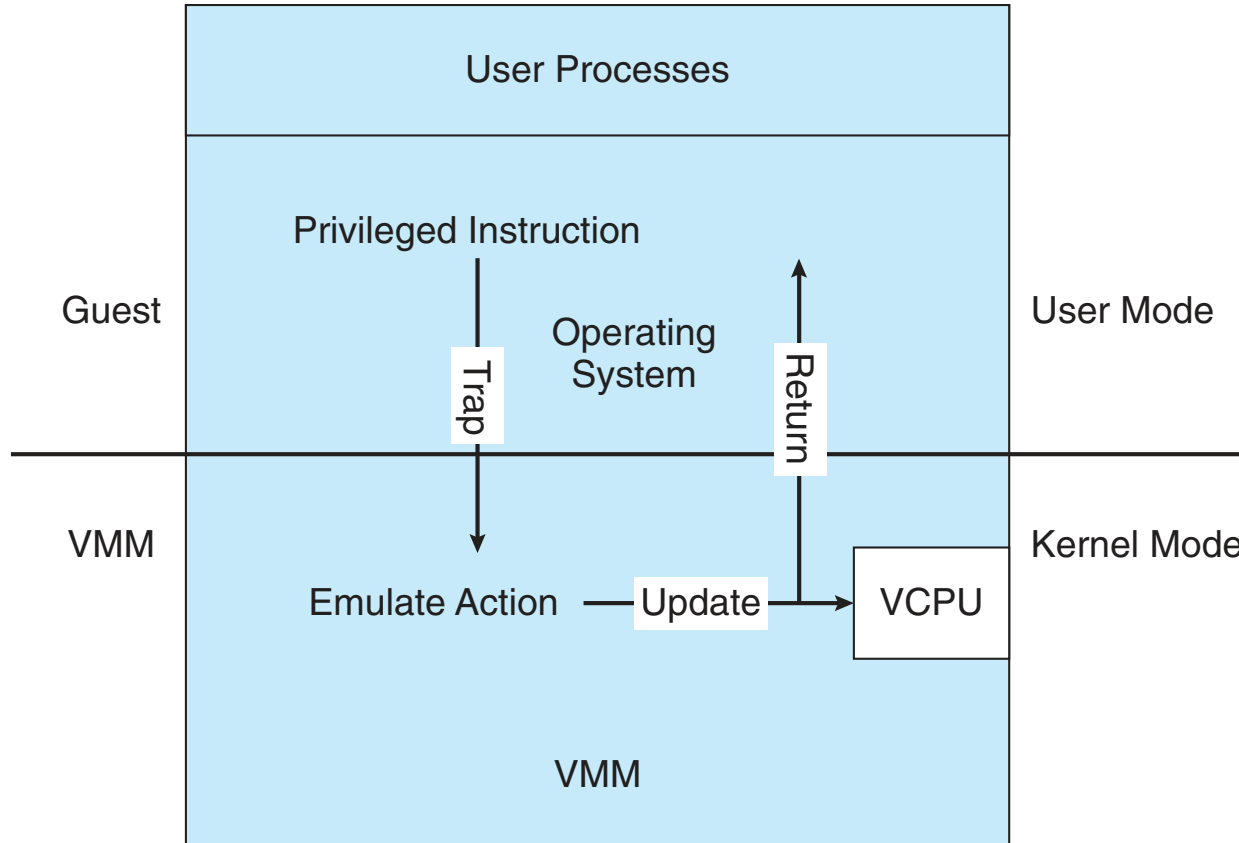
- Especially a problem when multiple guests running, each needing trap-and-emulate
- Thus, there is a performance penalty

■ CPUs adding hardware support, mode CPU modes to improve virtualization performance. This is done by caching “replacement” instructions by Intel and AMD.

Trap-and-Emulate (cont.)

- User mode code in guest runs at same speed as if not a guest
- But kernel mode privilege mode code runs slower due to trap-and-emulate
 - Especially a problem when multiple guests running, each needing trap-and-emulate
- CPUs adding hardware support, mode CPU modes to improve virtualization performance

Trap-and-Emulate Virtualization Implementation



Building Block – Binary Translation

- Some CPUs don't have clean separation between privileged and nonprivileged instructions
 - Earlier Intel x86 CPUs are among them
 - ▶ Earliest Intel CPU designed for a calculator
 - Backward compatibility means difficult to improve
 - Consider Intel x86 `popf` instruction
 - ▶ Loads CPU flags register from contents of the stack
 - ▶ If CPU in privileged mode -> all flags replaced
 - ▶ If CPU in user mode -> on some flags replaced
 - No trap is generated

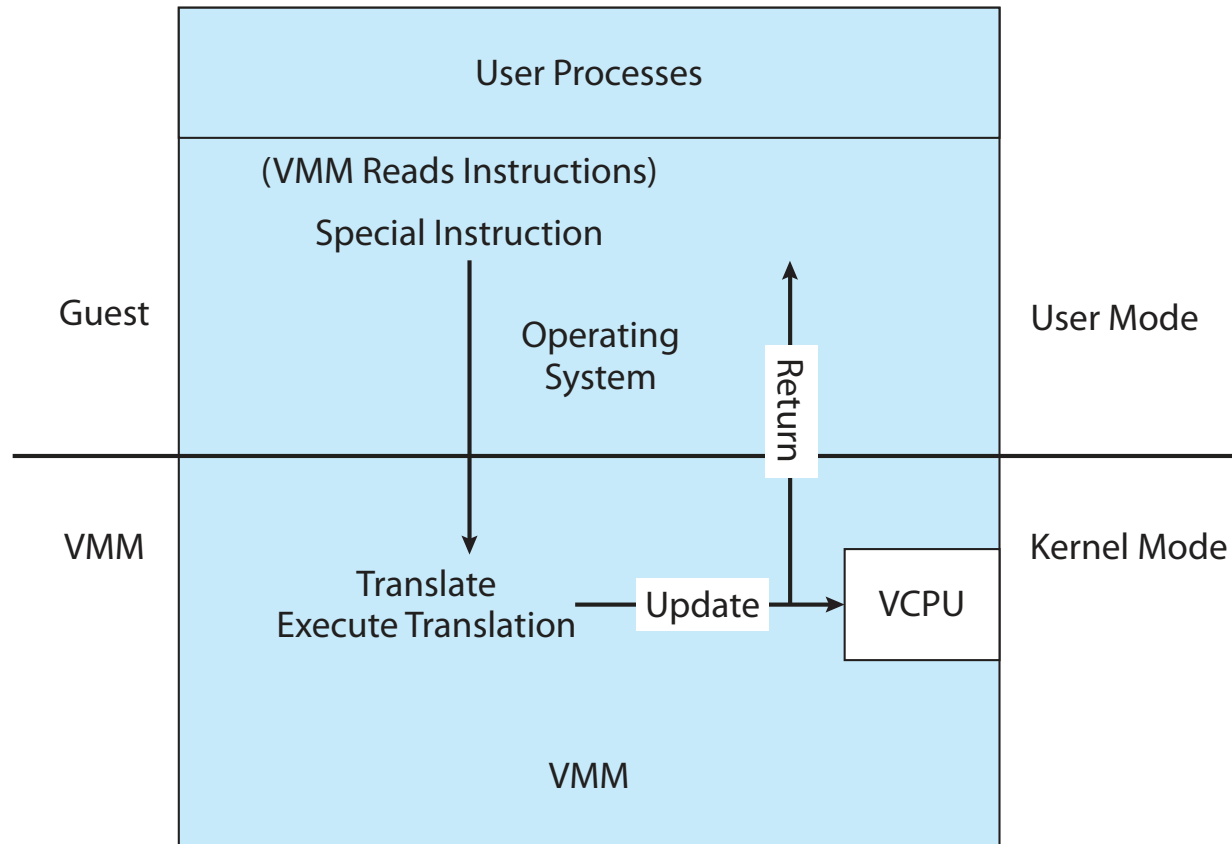
Binary Translation (cont.)

- Other similar problem instructions we will call ***special instructions***
 - Caused trap-and-emulate method considered impossible until 1998
- Binary translation solves the problem
 - Basics are simple, but implementation very complex
 1. If guest VCPU is in user mode, guest can run instructions natively
 2. If guest VCPU in kernel mode (guest believes it is in kernel mode)
 1. VMM examines every instruction guest is about to execute by reading a few instructions ahead of program counter
 2. Non-special-instructions run natively
 3. Special instructions translated into new set of instructions that perform equivalent task (for example changing the flags in the VCPU)

Binary Translation (cont.)

- Implemented by translation of code within VMM
- Code reads native instructions dynamically from guest, on demand, generates native binary code that executes in place of original code
- Performance of this method would be poor without optimizations
 - Products like VMware use caching
 - ▶ Translate once, and when guest executes code containing special instruction cached translation used instead of translating again
 - ▶ Testing showed booting Windows XP as guest caused 950,000 translations, at 3 microseconds each, or 3 second (5 %) slowdown over native

Binary Translation Virtualization Implementation



Nested Page Tables

- Memory management another general challenge to VMM implementations
- How can VMM keep page-table state for both guests believing they control the page tables and VMM that does control the tables?
- Common method (for trap-and-emulate and binary translation) is **nested page tables** (**NPTs**)
 - **Each guest maintains page tables to translate virtual to physical addresses**
 - VMM also maintains per guest NPTs to represent guest's page-table state
 - ▶ Just as VCPU stores guest CPU state
 - When guest on CPU -> VMM makes that guest's NPTs the active system page tables
 - ***Guest tries to change page table -> VMM makes equivalent change to NPTs and its own page tables***
 - Can cause many more TLB misses -> much slower performance
- This is commonly done with the assistance of H/W instructions in Intel/AMD

Building Blocks – Hardware Assistance

- All virtualization needs some HW support
 - <http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html>
- More support -> more feature rich, stable, better performance of guests
- Intel added new **VT-x** instructions in 2005 and AMD the **AMD-V** instructions in 2006
 - CPUs with these instructions remove need for binary translation
 - Generally define more CPU modes – “guest” and “host”
 - VMM can enable host mode, define characteristics of each guest VM, switch to guest mode and guest(s) on CPU(s)
 - In guest mode, guest OS thinks it is running natively, sees devices (as defined by VMM for that guest)
 - ▶ Access to virtualized device, priv instructions cause trap to VMM
 - ▶ CPU maintains VCPU, context switches it as needed
- HW support for Nested Page Tables, DMA, interrupts as well over time

Type 0 Hypervisor

Guest 1	Guest	Guest	Guest	Guest 3	Guest	Guest
	Guest 2				Guest 4	
CPU's memory	CPU's memory			CPU's memory	CPU's memory	
Hypervisor (in firmware)						I/O

Types of VMs – Type 1 Hypervisor

- Commonly found in company datacenters
 - In a sense becoming “datacenter operating systems”
 - ▶ Datacenter managers control and manage OSeS in new, sophisticated ways by controlling the Type 1 hypervisor
 - ▶ Consolidation of multiple OSeS and apps onto less HW
 - ▶ Move guests between systems to balance performance
 - ▶ Snapshots and cloning
- Special purpose operating systems that run natively on HW
 - Rather than providing system call interface, create run and manage guest OSeS
 - Can run on Type 0 hypervisors but not on other Type 1s
 - Run in kernel mode
 - Guests generally don’t know they are running in a VM
 - Implement device drivers for host HW because no other component can
 - Also provide other traditional OS services like CPU and memory management

Types of VMs – Type 1 Hypervisor (cont.)

- Another variation is a general purpose OS that also provides VMM functionality
 - RedHat Enterprise Linux with KVM, Windows with Hyper-V, Oracle Solaris
 - Perform normal duties as well as VMM duties
 - Typically less feature rich than dedicated Type 1 hypervisors
- In many ways, treat guests OSes as just another process
 - Albeit with special handling when guest tries to execute special instructions

Types of VMs – Type 2 Hypervisor

- Less interesting from an OS perspective
 - Very little OS involvement in virtualization
 - VMM is simply another process, run and managed by host
 - ▶ Even the host doesn't know they are a VMM running guests
 - Tend to have poorer overall performance because can't take advantage of some HW features
 - But also a benefit because require no changes to host OS
 - ▶ Student could have Type 2 hypervisor on native host, run multiple guests, all on standard host OS such as Windows, Linux, MacOS

Programming Environment Virtualization

- Also not-really-virtualization but using same techniques, providing similar features
- Programming language is designed to run within custom-built virtualized environment
 - For example Oracle Java has many features that depend on running in **Java Virtual Machine (JVM)**
- In this case virtualization is defined as providing APIs that define a set of features made available to a language and programs written in that language to provide an improved execution environment
- JVM compiled to **run on many systems** (including some smart phones even)
- Programs written in Java run in the JVM no matter the underlying system
- Similar to **interpreted languages**

Types of VMs – Emulation

- Another (older) way for running one operating system on a different operating system
 - Virtualization requires underlying CPU to be same as guest was compiled for
 - Emulation allows guest to run on different CPU
- Necessary to translate all guest instructions from guest CPU to native CPU
 - Emulation, not virtualization
- Useful when host system has one architecture, guest compiled for other architecture
 - Company replacing outdated servers with new servers containing different CPU architecture, but still want to run old applications
- Performance challenge – order of magnitude slower than native code
 - New machines faster than older machines so can reduce slowdown
- Very popular – especially in gaming where old consoles emulated on new

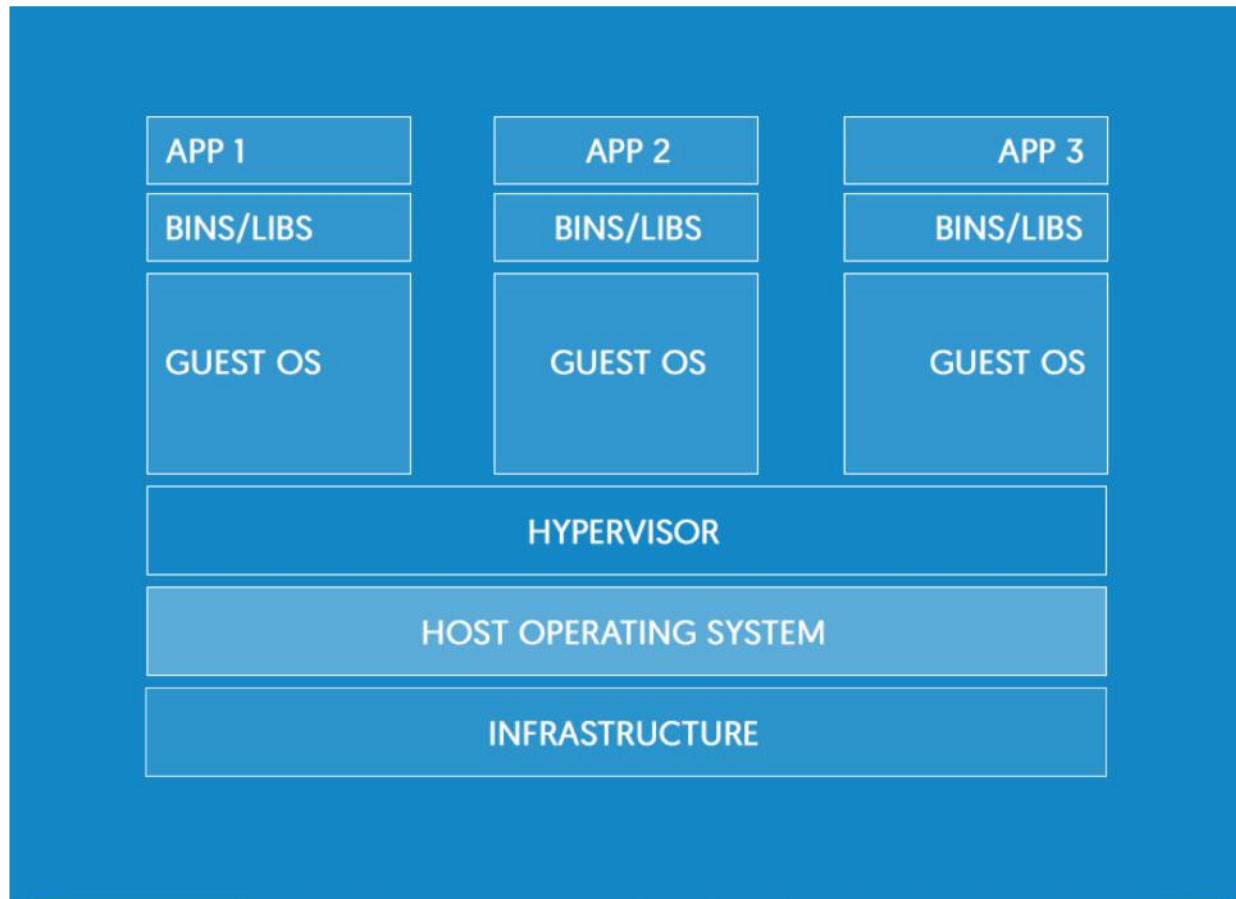
Emulate MacOS in your browser

- https://archive.org/details/mac_MacOS_7.0.1_compilation

VM vs Containers

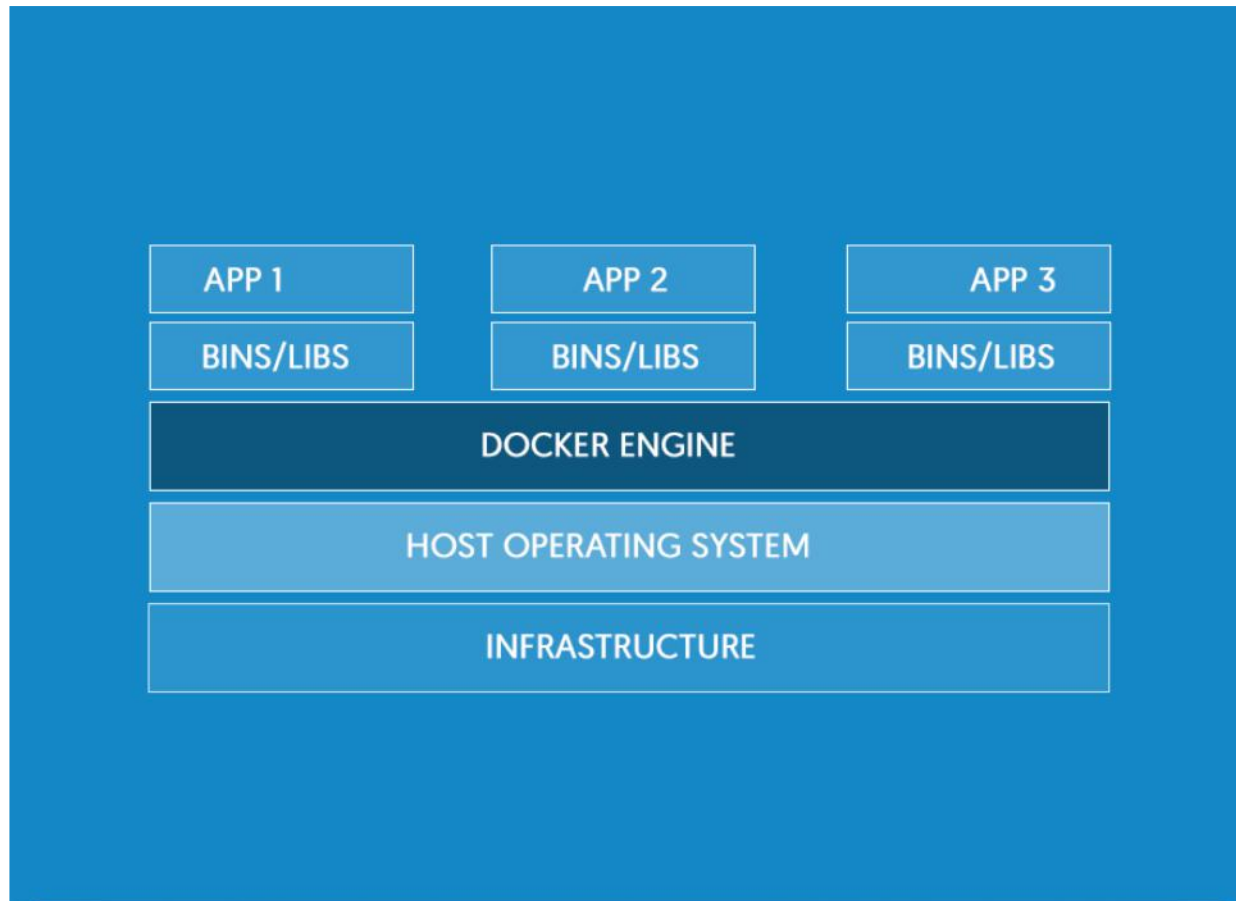
- <https://www.docker.com/what-container>
- <https://www.youtube.com/watch?v=R-3dfURb2hA&t=19s>
- Each VM runs not just a full copy of an operating system, but a virtual copy of all the hardware that the operating system needs to run.
- Container requires is enough of an operating system, supporting programs and libraries, and system resources to run a specific program.
- A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.

VM



<https://www.docker.com/what-docker>

Container



<https://www.docker.com/what-docker>

Kubernetes

- Automates the deployment, scaling and management of containerized applications.
- <https://www.youtube.com/watch?v=R-3dfURb2hA&t=19s>

Types of VMs – Containers

- Some goals of virtualization are segregation of apps, performance and resource management, easy start, stop, move, and management of them
- Can do those things without full-fledged virtualization
 - If applications compiled for the host operating system, don't need full virtualization to meet these goals
- Oracle **containers** / **zones** for example create virtual layer between OS and apps
 - Only one kernel running – host OS
 - OS and devices are virtualized, providing resources within zone with impression that they are only processes on system
 - Each zone has its own applications; networking stack, addresses, and ports; user accounts, etc
 - CPU and memory resources divided between zones
 - ▶ Zone can have its own scheduler to use those resources

Docker

- https://www.youtube.com/watch?v=dz5_IsWlfTU

VMWare explained by VMWare

- <https://twit.tv/shows/this-week-in-enterprise-tech/episodes/265?autostart=false>
- Start at 30 minutes into the podcast
- Degraded network start 38 minutes

Virtualization and Operating-System Components

- Now look at operating system aspects of virtualization
 - CPU scheduling, memory management, I/O, storage, and unique VM migration feature
 - ▶ How do VMMs schedule CPU use when guests believe they have dedicated CPUs?
 - ▶ How can memory management work when many guests require large amounts of memory?

OS Component – CPU Scheduling

- Even single-CPU systems act like multiprocessor ones when virtualized
 - One or more virtual CPUs per guest
- Generally VMM has one or more physical CPUs and number of threads to run on them
 - Guests configured with certain number of VCPUs
 - ▶ Can be adjusted throughout life of VM
 - When enough CPUs for all guests -> VMM can allocate dedicated CPUs, each guest much like native operating system managing its CPUs
 - Usually not enough CPUs -> CPU **overcommitment**
 - ▶ VMM can use standard scheduling algorithms to put threads on CPUs
 - ▶ Some add fairness aspect

OS Component – CPU Scheduling (cont.)

- Cycle stealing by VMM and oversubscription of CPUs means guests don't get CPU cycles they expect
 - Consider timesharing scheduler in a guest trying to schedule 100ms time slices -> each may take 100ms, 1 second, or longer
 - ▶ Poor response times for users of guest
 - ▶ Time-of-day clocks incorrect
 - Some VMMs provide application to run in each guest to fix time-of-day and provide other integration features

OS Component – Memory Management

- Also suffers from oversubscription -> requires extra management efficiency from VMM
- For example, VMware ESX guests have a configured amount of physical memory, then ESX uses 3 methods of memory management
 1. Double-paging, in which the guest page table indicates a page is in a physical frame but the VMM moves some of those pages to backing store
 2. Install a **pseudo-device driver** in each guest (it looks like a device driver to the guest kernel but really just adds kernel-mode code to the guest)
 - ▶ **Balloon** memory manager communicates with VMM and is told to allocate or deallocate memory to decrease or increase physical memory use of guest, causing guest OS to free or have more memory available
 3. Deduplication by VMM determining if same page loaded more than once, memory mapping the same page into multiple guests

OS Component – I/O

- Easier for VMMs to integrate with guests because I/O has lots of variation
 - Already somewhat segregated / flexible via device drivers
 - VMM can provide new devices and device drivers
- But overall I/O is complicated for VMMs
 - Many short paths for I/O in standard OSES for improved performance
 - Less hypervisor needs to do for I/O for guests, the better
 - Possibilities include direct device access, DMA pass-through, direct interrupt delivery
 - ▶ Again, HW support needed for these
- Networking also complex as VMM and guests all need network access
 - VMM can **bridge** guest to network (allowing direct access)
 - And / or provide **network address translation (NAT)**
 - ▶ NAT address local to machine on which guest is running, VMM provides address translation to guest to hide its address

OS Component – Storage Management

- Both boot disk and general data access need be provided by VMM
- Need to support potentially dozens of guests per VMM (so standard disk partitioning not sufficient)
- Type 1 – storage guest root disks and config information within file system provided by VMM as a **disk image**
- Type 2 – store as files in file system provided by host OS
- Duplicate file -> create new guest
- Move file to another system -> move guest
- **Physical-to-virtual (P-to-V)** convert native disk blocks into VMM format
- **Virtual-to-physical (V-to-P)** convert from virtual format to native or disk format
- VMM also needs to provide access to network attached storage (just networking) and other disk images, disk partitions, disks, etc

OS Component – Live Migration

- Taking advantage of VMM features leads to new functionality not found on general operating systems such as live migration
- Running guest can be moved between systems, without interrupting user access to the guest or its apps
- Very useful for resource management, maintenance downtime windows, etc
 1. The source VMM establishes a connection with the target VMM
 2. The target creates a new guest by creating a new VCPU, etc
 3. The source sends all read-only guest memory pages to the target
 4. The source sends all read-write pages to the target, marking them as clean
 5. The source **repeats** step 4, as during that step some pages were probably modified by the guest and are now dirty
 6. When cycle of steps 4 and 5 becomes very short, source VMM freezes guest, sends VCPU's final state, sends other state details, sends final dirty pages, and tells target to start running the guest
 - ▶ Once target acknowledges that guest running, source terminates guest

End of Chapter 16

