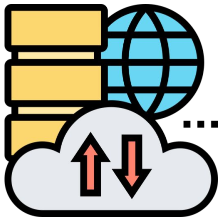
COEN 241
Introduction to Cloud Computing

Lecture 4 - Advanced Virtualization Concepts II

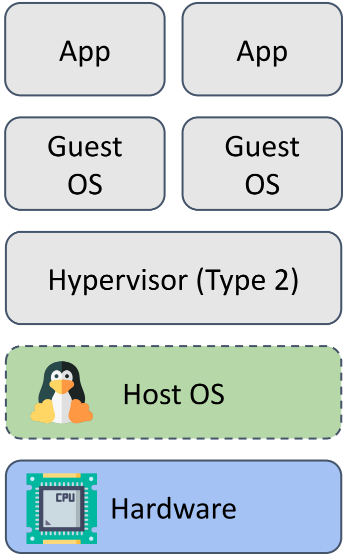


Lecture 3 Recap

- Advanced Virtualization Concepts
  - Hypervisor
    - Xen
  - CPU Support for Virtualization
- Emulation
- QEMU

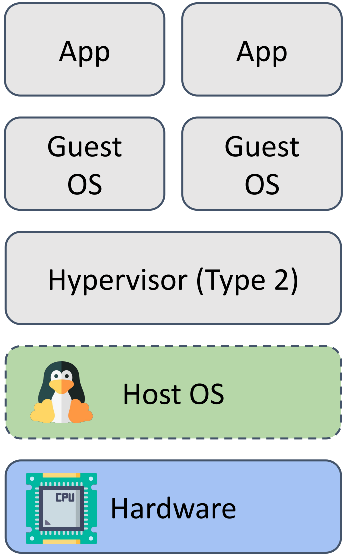
Type 1 & Type 2 Hypervisor

- Type 1 Hypervisor
  - Runs directly on the host's hardware
  - Used for most production environment
  - E.g., Citrix/Xen Server, VMware ESXi, Microsoft Hyper-V
  - Pros: Secure, performant
  - Cons: Require hardware support, harder to use
- Type 2 Hypervisor
  - Runs on an operating system similar to other processes
  - E.g., Oracle Virtual Box, VMware Workstation
  - Pro: Easier to use (We will use this for HW1)
  - Con: Less performant and less secure



Hypervisor Capabilities

- Manages CPU, RAM and device I/O (input/output)
    - Device I/O include disk, network, graphics, USB, ...
- Share concepts with microkernel operating systems
    - i.e., microkernel can't do the complete job of an operating system
- For device I/O, must prevent guests “breaking out”
- Base security approach on per-device capabilities:
    - Network card might be VM-aware and can isolate functions
    - Need to be very careful about capabilities such as DMA
        - Direct Memory Access lets devices read/write memory without CPU

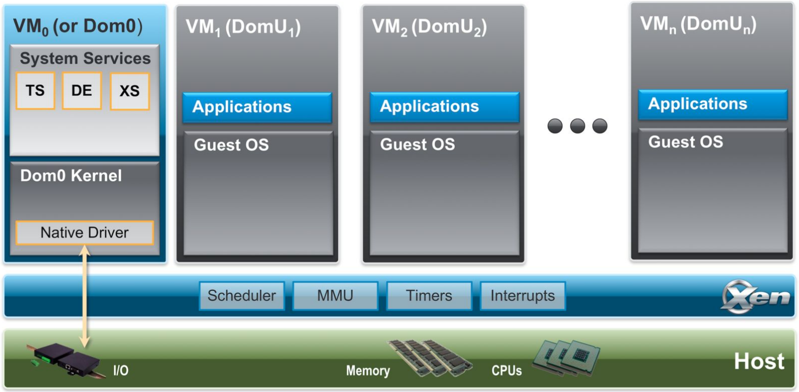


Digression: Xen Project

- Xen was presented in 2003 SOSP
  - One of the top CS Conferences
  - https://www.cl.cam.ac.uk/research/srg/netos/papers/2003-xensosp.pdf
- Xen (then) required paravirtualised OSs as VMs
  - We will discuss paravirtualization soon
- Demonstrated on both Linux (XenoLinux) and Windows XP
  - Was much faster than VMWare
  - Able to scale to 100 VMs in a single commodity machine
  - Open Source

Xen Architecture

- Xen divides hosts into Domains
  - Dom0 is special: the Linux that runs actual hardware
  - Then the DomUs can make *hypercalls* to access hardware
  - Xen hypervisor delegates hardware interaction to Dom0



Hypervisor Security

- Users expect complete isolation
  - As if each VM is on a separate computer
- Aggregation must result in resource sharing
  - Security risk in control interactions with the hypervisor
  - Hypervisor management commands have exposed security holes
- Hypervisor must be designed to have a **small attack surface area**
  - Kernels have a large attack area, and are hard to secure
  - **Hyperjacking**: taking over (hijacking) the hypervisor from within a guest
- A Hard Problem!

Prior Hypervisor Security Holes

- VENOM (Virtualized Environment Neglected Operations Manipulation)
    - Problem with the floppy disk controller in QEMU
        - VirtualBox, Xen and KVM also used QEMU’s code
    - Basically, guest accesses “floppy drive” via “I/O port”
    - QEMU driver keeps track of floppy drive commands in a buffer
    - but specially crafted requests could overflow buffer
    - Malicious VM can then take control of QEMU system
- Spectre & Meltdown
    - CVE-2017-5715, 2017-5753, 2017-5754
    - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5715
- https://en.wikipedia.org/wiki/Virtual_machine_escape

Live Migration

- Live migration: Moving running VMs to another host
    - A good demonstration that your hypervisor is efficient!
    - “Live” usually means no detectable downtime
    - i.e., cannot pause VM, copy VM state, resume VM on new host
    - Repeatedly stream memory updates until can do switch-over

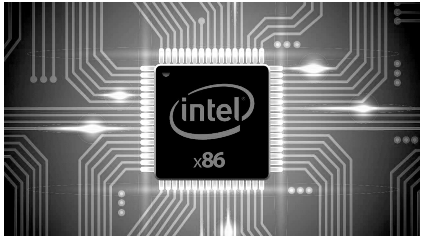
- Requirements of physical hosts supporting migration:
    - NICs are receiving the same MAC address
    - Simplest for storage to be network-based (e.g., iSCSI / NFS)

High Availability

- High availability (HA) means VMs are robust to failure:
  - VMs may restart on the same host (e.g., given typical OS crash)
  - Or host may fail: ensure VMs can continue on a different host
- Live migration & high availability have common needs
  - HA requires VM that might take over being up-to-date
  - Similar to a persistent live migration from leader to follower
- Care needed to ensure safe and consistent failover

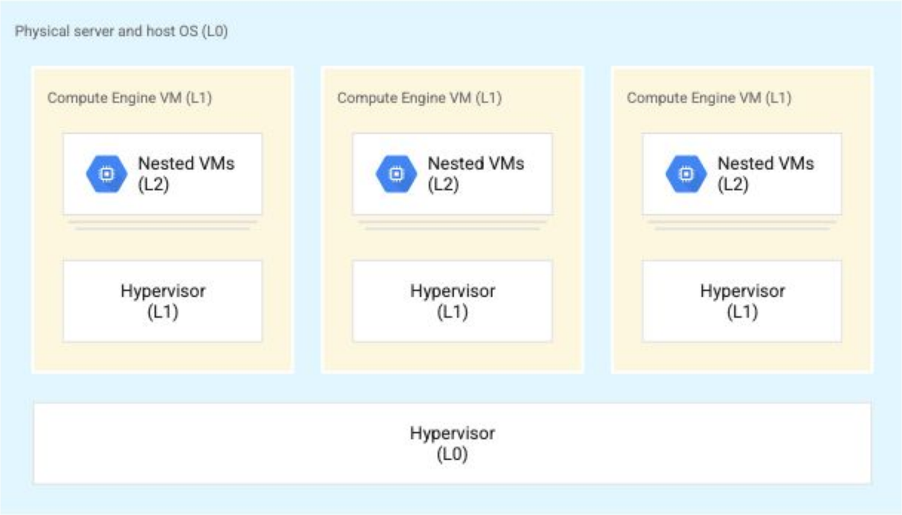
Challenges in x86/x64 Virtualization

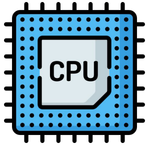
- 20 years between first CPU and the first hardware virtualization efforts
- CPU **protected mode** and CPU **long mode** (64-bit operation)
  - These modes weren't designed without virtualization in mind
- Hidden CPU state
  - VMM can't save/restore this state later when switching VMs
- Memory management inefficiency
- I/O interactions
  - Again, designed without virtualization in mind



Nested Virtualization

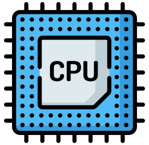
- Running VMs inside another VM





Hardware Virtualization Support in x86/x64

- Intel VT-x released in 2005 for some Pentium 4 CPUs
  - Virtual Machine Extension and adds new instructions for virtualization
  - Subsequent CPUs include it (except some Atom processors)
  - AMD released an equivalent technology in 2006 (AMD-v)
- CPUs gain a guest mode within protected mode
  - For guest OSes, guest mode looks like protected mode
  - For hosts, guest mode is lower privilege than protected mode
- More memory virtualization support still to come…



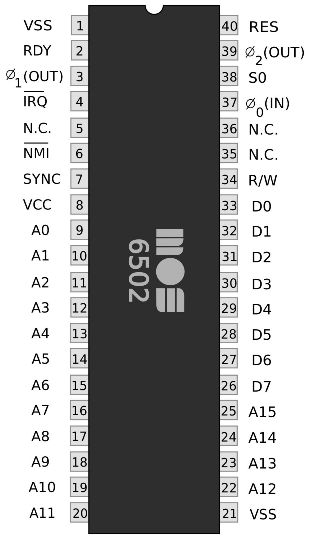
Agenda for Today

- Advanced Virtualization Concepts II
  - Memory Virtualization
  - Paravirtualization
  - I/O Virtualization
  - Vagrant
- HW 1 Preview
- Readings
  - Recommended: None
  - Optional
    - https://www.pagetable.com/?p=25
    - https://learn.hashicorp.com/vagrant

Memory Virtualization

Another Virtualization Challenge: RAM

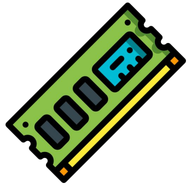
- RAM access patterns are surprisingly complex
- CPU's address pin used to indicate address to read/write
- Example
  - MOS 6502 has 16 address wires, thus 64KB RAM (2^{16} bytes)
    - 8-bit processor
    - D0-7: Data bus
    - A0-15: Address bus



Another Virtualization Challenge: RAM

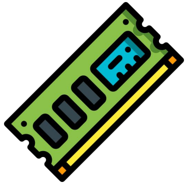
- Early Intel 80x86 chips addressed offsets of 'segments'
  - **Segmentation** is the process in which the main memory of the computer is logically divided into different segments and each segment has its own base address
    - https://www.geeksforgeeks.org/memory-segmentation-8086-microprocessor/
  - Originally invented as a method by which system software could isolate different software processes (tasks) and data they are using
  - Costly memory management algorithms required
  - Segmentation must find free memory area big enough
  - Originally no memory protection
  - **Thankfully segmented memory model has died off in x64**

- Intel 80386 added **page-based memory mapping**

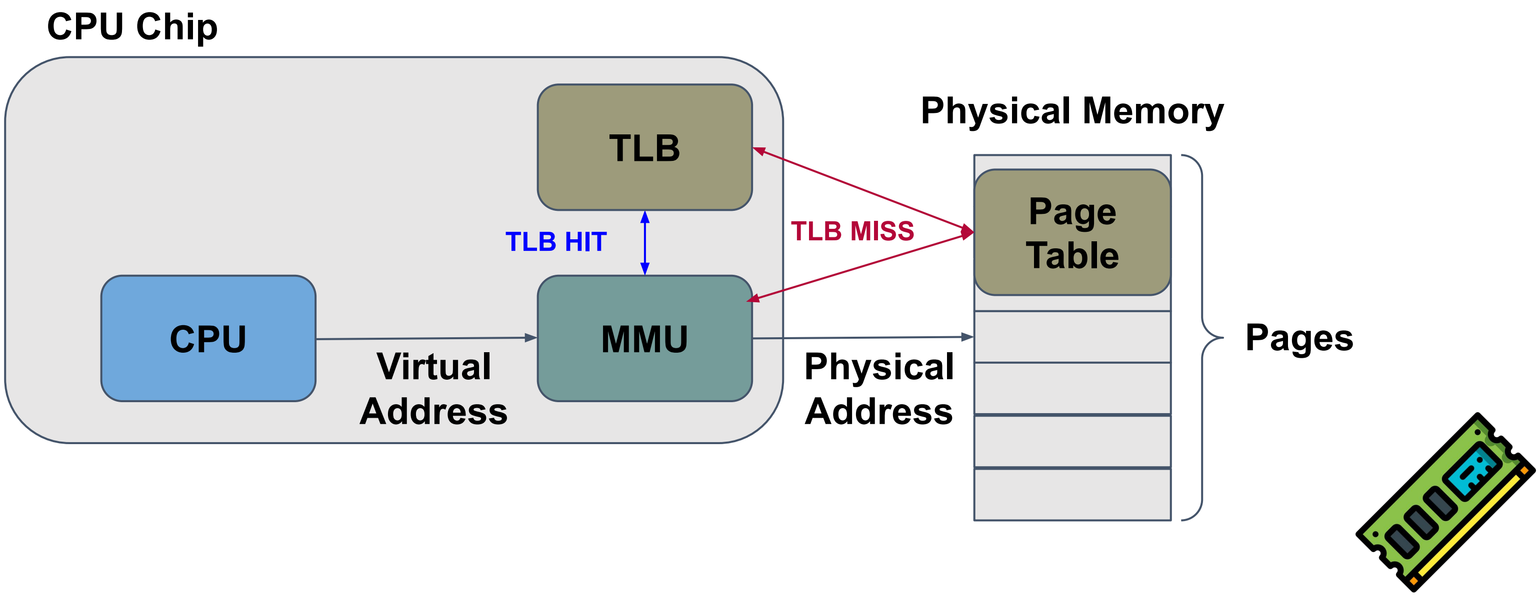


Page-based Memory Access

- Modern CPUs manage memory within **pages**
  - Mostly configured by the chip architecture
  - Can have more than 1 page size
- CPU **memory management unit (MMU)** does the work of translating **virtual addresses** into **physical addresses** and vice versa
- **Page tables** describe virtual to physical mapping
  - The page tables are stored in memory, themselves as well
  - Page tables define process' address space
- Translation Lookaside Buffer **(TLB)** is a faster Page Table within a CPU
  - Prefer hitting the TLB than going to page the table
- Virtual addresses help OSs manage processes' memory
  - Swap parts of an address space in & out of **physical memory**



Page-based Memory Access

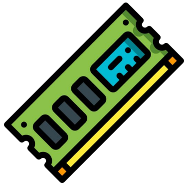


Segmentation vs Paging

Segmentation	Paging
Variable Sized	Fixed Sized
Size determined by user	Size is determined by hardware
Slower	Faster
Logical address is split into section number and section offset.	Logical address is split into page number and page offset.
Visible to user	Invisible to the user

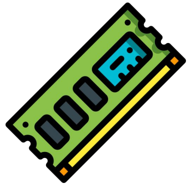
Challenges in Virtualizing Hardware Page Tables

- Hypervisor has no chances to intercept on TLB misses
  - Why?
- Solutions around this issue
  - **Software-based**: Shadow Paging
  - **Hardware-based**: Hardware support for virtualizing memory

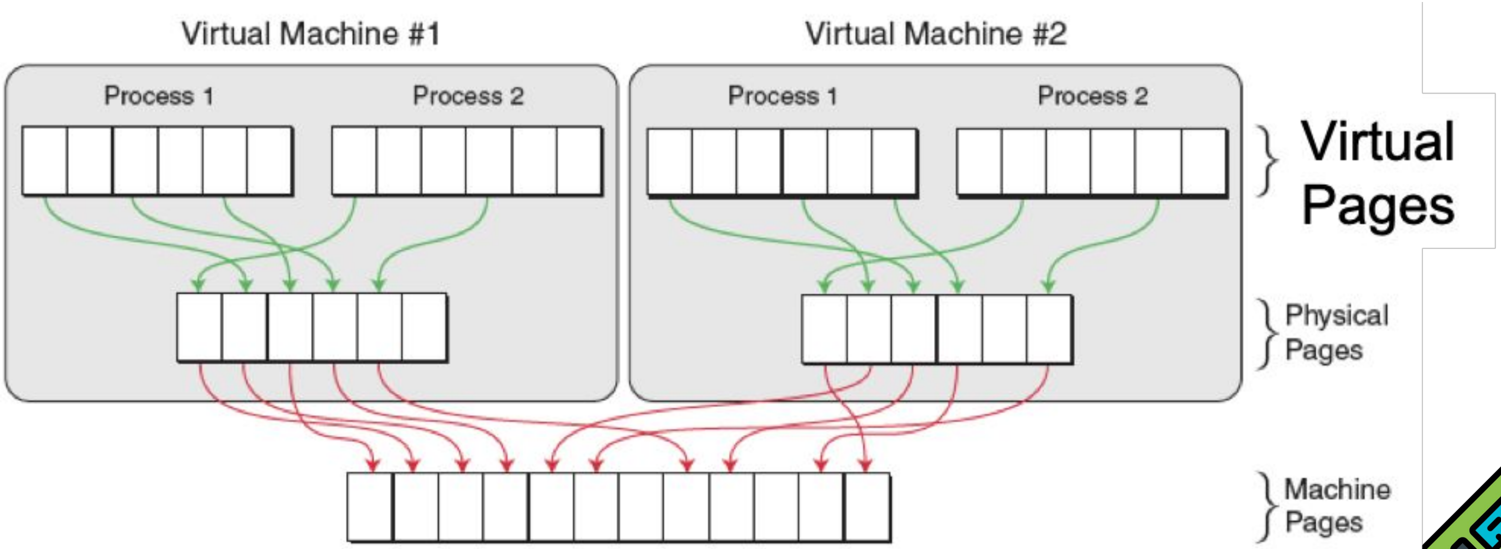


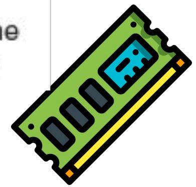
Virtualizing Paged Memory - Shadow Page Table

- Page tables themselves are managed by **guest OSs**
- Older CPUs: Hypervisor creates a **shadow page table per guest**
  - An indirection between guest physical and host physical memory
  - Every time guest OS changes its page tables, hypervisor must change shadow page tables as well
- Slow and very CPU intensive!
  - Each memory access goes through additional searches



Virtualizing Paged Memory - Shadow Page Table

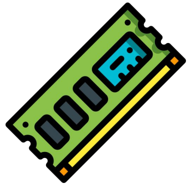




Virtualizing Paged Memory - Shadow Page Table

- Hypervisor creates and manages page tables that map virtual pages directly to machine pages
  - These tables are loaded into the MMU on a context switch
  - Hypervisor page tables are the shadow page tables

- Hypervisor needs to keep its V => M tables consistent with changes made by OS to its V => P tables
  - Hypervisor maps OS page tables as read only
  - When OS writes to page tables, trap to Hypervisor
  - Hypervisor applies write to shadow table and OS table, returns
  - Also known as memory tracing
  - High overhead!

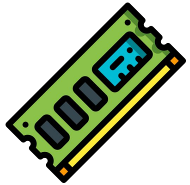


Virtualizing Paged Memory - Nested Paging

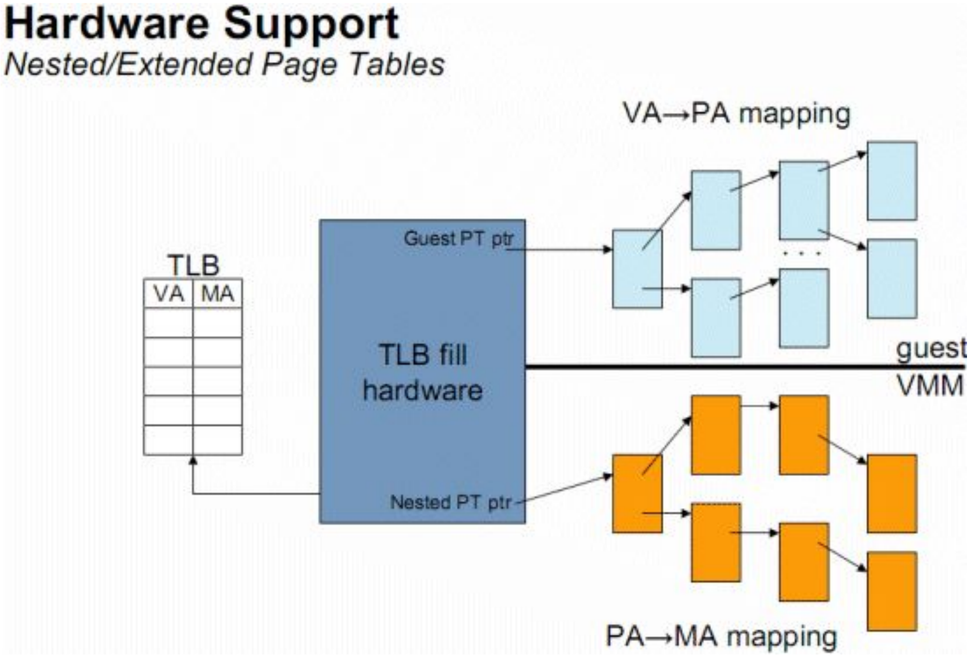
- Newer CPUs: Nested paging support in **hardware**
  - Guest's physical addresses treated as a host virtual address
  - Caching of virtual to physical address translation is important for speed up
  - Examples
    - Intel Extended Page Table (EPT)
    - AMD Nested Page Table (NPT)

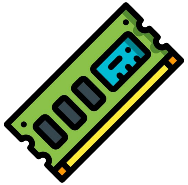
- Much simpler hypervisor memory management

- Lower memory overhead



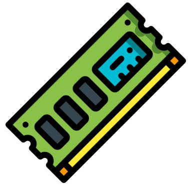
Virtualizing Paged Memory - Nested Paging





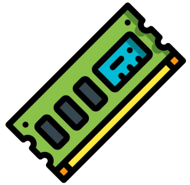
TLB Tagging

- Traditionally, every time a hypervisor switched between different VMs, the VM and its data structure had to be flushed out of the TLB
- Intel and AMD have facilitated TLB tagging since 2008
  - Intel Virtual Processor IDs (VPIDs) allow VMM to assign VM IDs
- Flush TLB entries of a particular VM only
  - So switching between VMs and VMM may leave TLB entries
  - Significant boost to memory access speed

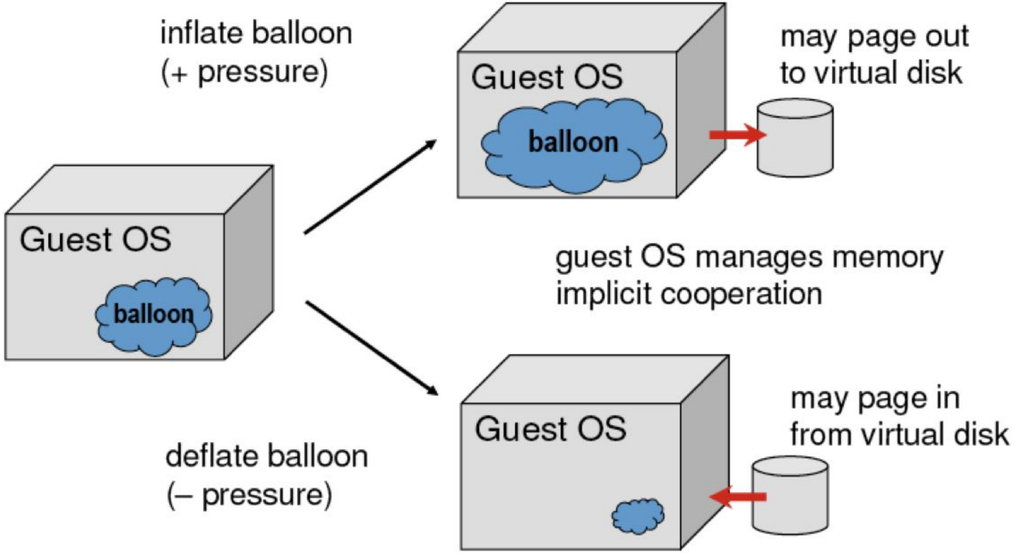


Memory Management / Allocation

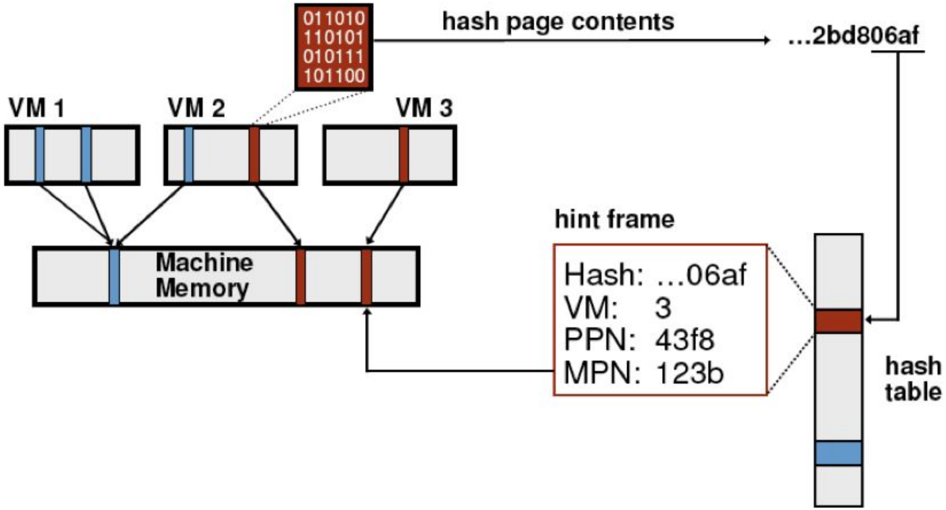
- Hypervisors tend to have simple memory management
  - Static policy: VM gets 8GB at start
  - Dynamic adjustment is hard since OS cannot handle
  - No swapping to disk
- More sophistication: Overcommit with ballooning
  - Balloon driver runs inside OS => consume hardware pages
  - Balloon grows or shrinks (gives back mem to other VMs)
- Even more sophistication: memory de-duplication
  - Identify pages that are shared across VMs!



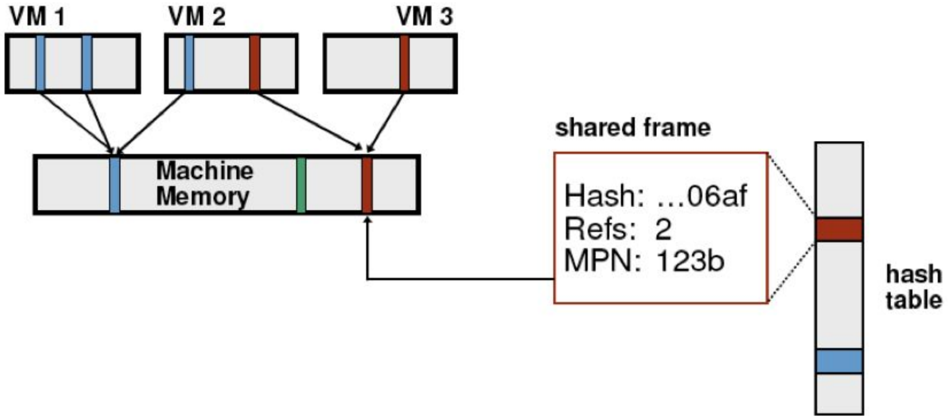
Memory Ballooning



Page Sharing



Page Sharing

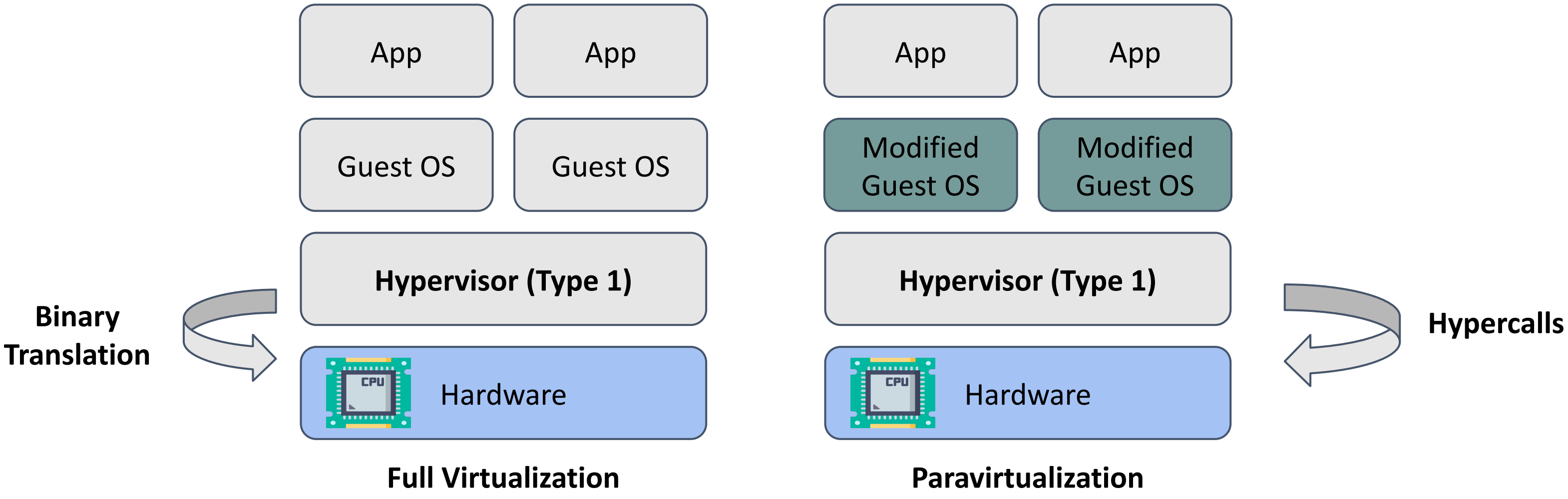


Paravirtualization

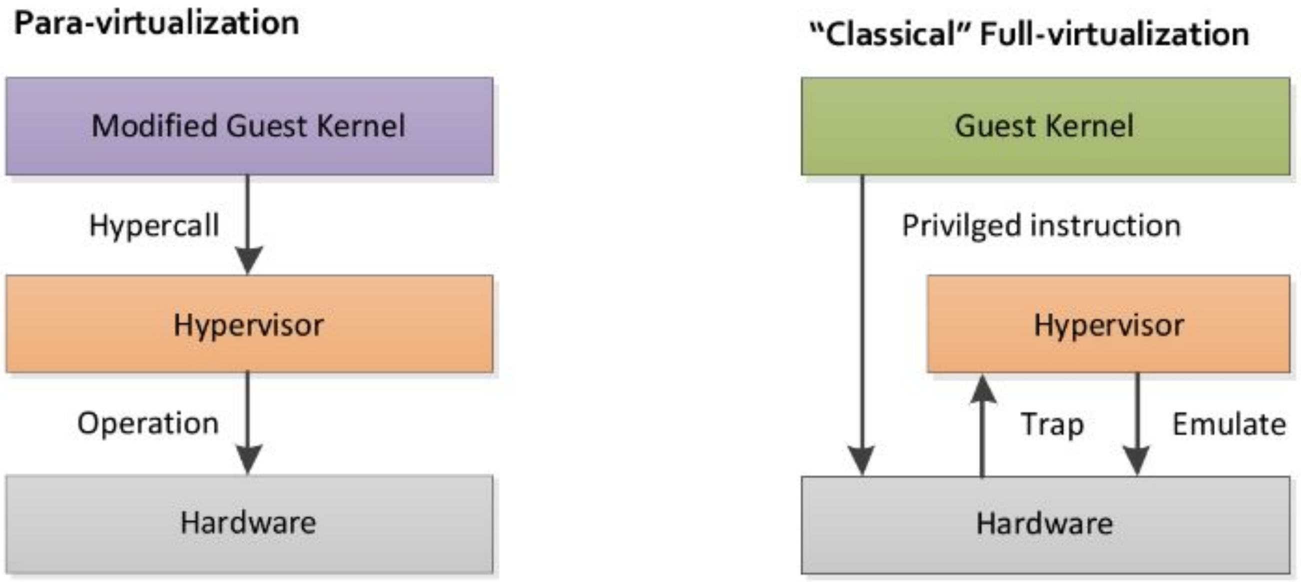
Paravirtualization

- Complete isolation of VMs implies VMs are not aware of virtualization
- Sometimes it can be better to let VMs know that they're virtual
    - E.g., otherwise VMs may waste time managing fake devices
- **Paravirtualization** is a technique that allows VMs to have an interface similar to that of the host hardware
    - Translator vs Native Speaker
    - Also called OS Assisted Virtualization

Paravirtualization



Paravirtualization



Paravirtualization

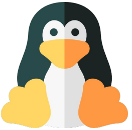
Paravirtualization	Full Virtualization
Guest OS communicates with hypervisor	Guest OS executes independently
Faster Performance	Lower Performance
Guest OS directly communicates with hardware via drivers	Guest OS issues hardware call
More secure	Less secure
Require guest OS changes	No need to change guest OS

Xen and Paravirtualization

- Back to Xen!
  - Xen relied heavily on paravirtualization
- Using paravirtualization meant that Xen is very small
  - Made it practical for development within a university
- Xen was released first, and then OSs ported to it
  - Enabling paravirtualization on Linux was fairly straightforward
  - Paravirtualized Windows XP existed too

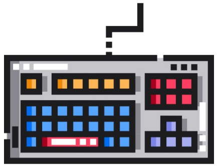
Linux Paravirtualization

- In 2006 Xen, IBM, Red Hat, and VMware met and agreed to collaborate on **paravirt-ops initiative**
  - Provides a way for Linux to know to para-virtualize itself
    - While also enabling it to boot normally if not running in as a VM
  - Agnostic to the underlying VMMs
    - Supports Xen, VMware, VirtualBox
- Microsoft did not release the Xen-compatible Windows
  - Better CPU support for virtualization arrived
  - Thus Microsoft no longer needed to cooperate with Xen



Paravirtualized Device Drivers

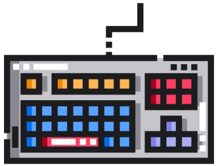
- We only discussed paravirtualized OS kernel functions
- Often hardware is accessed through device drivers
  - (Too many different types to build directly into OS effectively!)
- Can use paravirtualized dev. drivers in unmodified OS
  - VirtualBox’s guest extensions; VMware’s Guest Tools;…
- **virtio** provides a set of common emulated devices
  - Specifically the front-end drivers within the guest OS
  - Back-end drivers map virtio API to real device drivers in host OS



Typical Front-end Drivers in virtio

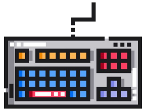
- vrtio-blk: i.e., block devices: hard disks, DVD drives, …
- vrtio-net: i.e., network adapters
- virtio-pci: i.e., PCI pass through
  - Recall that PCI is for interconnecting peripherals with the CPU
  - e.g., hot-pluggable storage devices
- virtio-console: i.e., the keyboard and screen
  - Very basic versions of them, but useful for diagnostics
- virtio-balloon: for managing guest memory size

- Try to find where to set these up in HW 1!



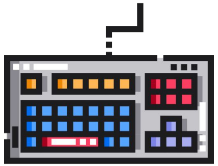
Transparency & Optimization

- Butler Lampson, a famous computer scientist and Turing award winner once gave two conflicting advice on the nature of implementations.
- "**Keep secrets of the implementation.** Secrets are assumptions about an implementation that client programs are not allowed to make... Obviously, it is easier to program and modify a system if its parts make fewer assumptions about each other."
- "One way to improve performance is to increase the number of assumptions that one part of a system makes about another; the additional assumptions often allow less work to be done, sometimes a lot less."
- On one hand we should **hide an implementation** for ease of development on the other, we should **expose our implementations** for speed



Transparency & Optimization

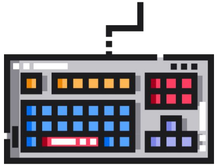
- Xen
  - Chose to increase transparency to increase optimization
- VMWare
  - Chose to decrease transparency to increase ease of development and use



I/O Virtualization

Challenges and Solution in I/O Virtualization

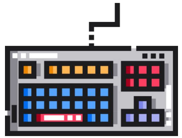
- Challenge: Lots of I/O devices available in the world!
- Problem: Writing device drivers for all I/O devices in the hypervisor layer is not a feasible option
- But!, Device drivers are already written for popular Operating Systems
- **Solution**: Present **virtual** I/O devices to guest VMs and channel I/O requests to a trusted host VM (popular OS)



Challenges and Solution in I/O Virtualization

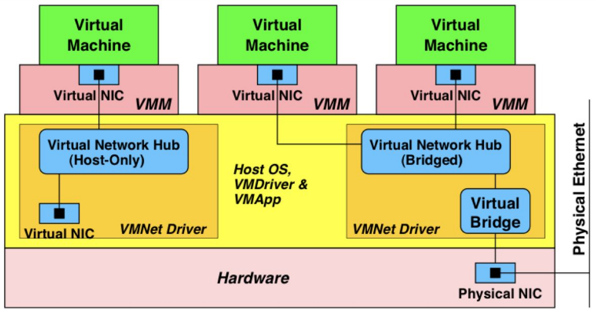
- Even so, I/O is complicated to virtualize in the hypervisor
  - Many short paths for I/O devices and OSes for performance
  - Better if hypervisor needs to **do less** for I/O for guests

- Possible Solutions
  - Direct device access
  - DMA pass-through
  - Direct interrupt delivery (need H/W support!)



Challenges and Solution in I/O Virtualization

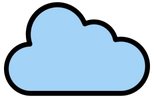
- Networking also complex as hypervisor and guests all need network access
  - Hypervisor can bridge guest to network (direct access)
  - Hypervisor can provide network address translation (NAT)
    - NAT address local to machine on which guest is running, hypervisor provides address translation to guest to hide its address



IAC & Vagrant

Infrastructure As Code (IAC)

- IAC covers configuration management & provisioning
  - Also involves avoiding hardware configuration (e.g., switches)
  - Goal: complete automation from machine readable files
  - For cloud, cluster of servers or single server management
- Cost reduction
  - Focus on business needs rather than device management
  - Continuous integration pipelines often integrated
- Now a requirement for most businesses running on cloud
- Will talk about this in more detail in upcoming lectures



Vagrant

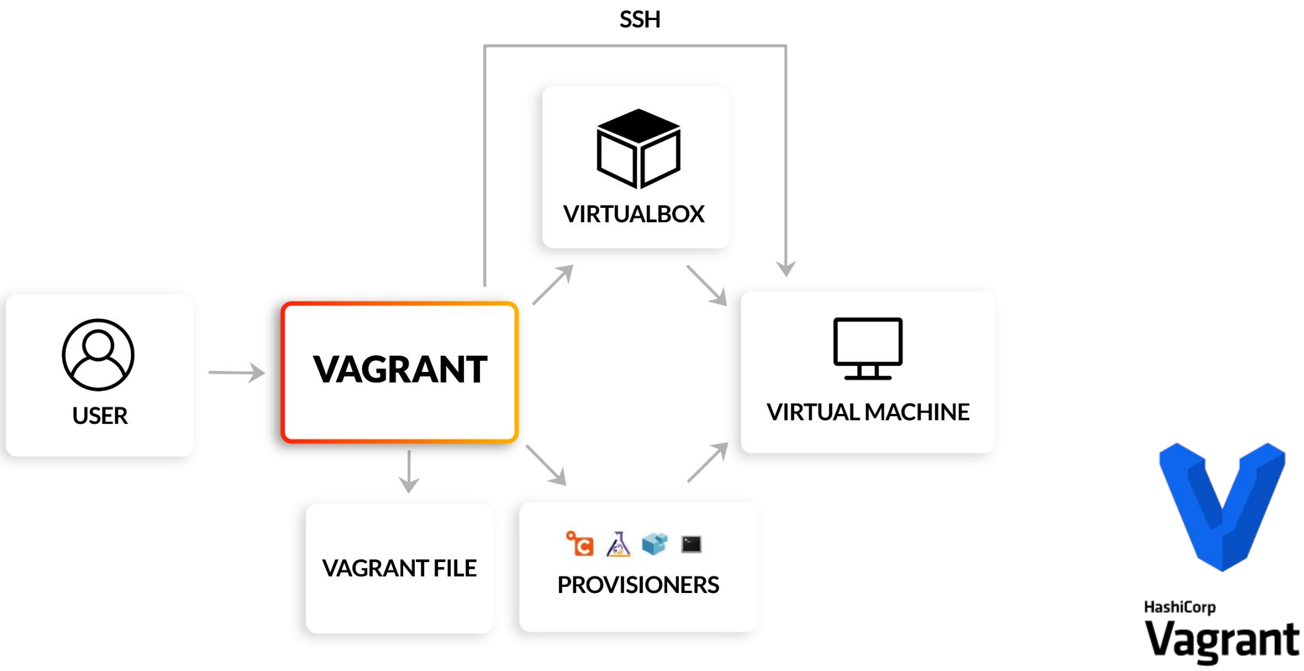
- A tool for automatically creating and configuring VMs via simple scripts
- Operates on a **Vagrantfile** that contains instructions for how to download and run a particular VM image
- Requires a "provider" on the host machine to start and manage the VM
    - E.g., Virtualbox, QEMU, Hyper-V, ...

HashiCorp Vagrant

Vagrant

- First started as a personal side-project by Mitchell Hashimoto in January 2010
- In 2012 Founded Hashicorp now valued at $5.1B
- Now used by many corporations like BBC, Nokia and Expedia
- Enables **Infrastructure as Code (IAC)**
    - Along with Kubernetes, which we will cover soon

Vagrant Architecture



Vagrant File (Single VM)

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|

  config.vm.box = "ubuntu/trusty64"

  config.vm.network "private_network", ip: "192.168.33.10"
  config.vm.network "public_network"
  config.vm.network "forwarded_port", guest: 80, host: 8080

  config.vm.synced_folder "code/", "/app/code"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = 2048
    vb.cpus = 1
  end

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y apache2
    service apache2 start
  SHELL
end
```

Vagrant File (Multi VM)

```
Vagrant.configure("2") do |config|
  config.vm.provision "shell", inline: "echo Hello"

  config.vm.define "web" do |web|
    web.vm.box = "ubuntu/trusty64"
    web.vm.network "private_network", ip: "192.168.33.20"
    web.vm.synced_folder "code/", "/app/code"
    web.vm.provider "virtualbox" do |vb|
        vb.memory = 1048
        vb.cpus = 1
    end
  end

  config.vm.define "db" do |db|
    db.vm.box = "ubuntu/trusty64"
    db.vm.network "private_network", ip: "192.168.33.30"
    db.vm.synced_folder "data/", "/db/data"
    db.vm.provider "virtualbox" do |vb|
        vb.memory = 2048
        vb.cpus = 1
    end
  end
end
```

HashiCorp Vagrant

Vagrant File Details

- `Vagrant.configure("2")`
- `web.vm.box = "ubuntu/trusty64"`
- `web.vm.network "private_network", ip: "192.168.33.20"`
- `web.vm.synced_folder "code/", "/app/code"`
- `vb.memory = 2048`
- `vb.cpus = 1`

Vagrant Commands

- `vagrant up`
- `vagrant status`
- `vagrant ssh`
- `vagrant destroy`
- `vagrant remove`

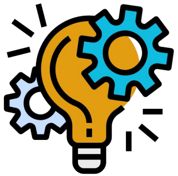
HW 1 Preview

HW 1 Preview

- QEMU and Docker

- 3 Main Parts
  - Being able to install and run these technologies on your machine
    - Can get frustrating and hard
  - Running the experiments
  - Report on your findings

- Must submit via git
  - Make sure to create a single repository for all your HWs

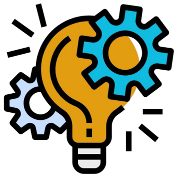


Agenda for Today

- Advanced Virtualization Concepts II
  - Memory Virtualization
  - Paravirtualization
  - I/O Virtualization
  - Vagrant
- HW 1 Preview
- Readings
  - Recommended: None
  - Optional
    - https://www.pagetable.com/?p=25
    - https://learn.hashicorp.com/vagrant

TODOs

- Quiz 1 is out!
  - 10 questions T/F, MC and multiple answers
- HW 1
- Project



Questions?