Virtual Machines

To do …

- VM over time
- Implementation methods
- Hardware features supporting VM
- Next time: Midterm?

*Partially based on notes from C. Waldspurger, VMware, 2010 and Arpaci-Dusseau's Three Easy Pieces

Too many computers!

- An organization can have *too many* machines
  - Why many? Running different services (email, web) to
    - Ensure each has enough resources
    - … fails independently
    - ... is sandboxed (for security)
    - ... survives if another one gets attack
    - ... different services may require different OSes
  - *Why **too** many?* Hard and expensive to maintain!
- Virtualization as an alternative
  - A VMM (or hypervisor) creates the illusion of multiple (virtual) machines

Virtualization applications

- Server consolidation
  - Convert underutilized servers to VMs, saving cost
  - Increasingly used for virtual desktops
- Simplified management
  - Datacenter provisioning and monitoring
  - Dynamic load balancing
- Improved availability
  - Automatic restart, fault tolerance, disaster recovery
- Test and development
- Cloud support
  - Isolation for clients

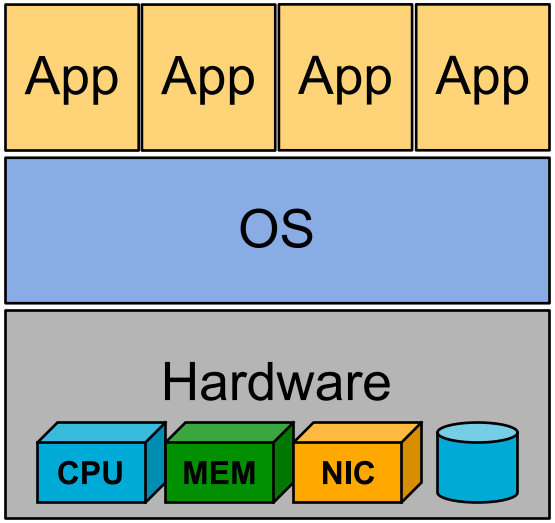
- *But are you not putting all your eggs in one basket?*

Types of virtualization

- Process virtualization
  - Language-level: Java, .NET, Smalltalk
  - OS-level: processes, Solaris Zones, BSD Jails
  - Cross-ISA emulation: Apple 68K-PPC-x86
- Device virtualization
  - Logical vs. physical: VLAN, VPN, LUN, RAID
- System virtualization
  - Xen, VMware Fusion, KVM, Palacios …

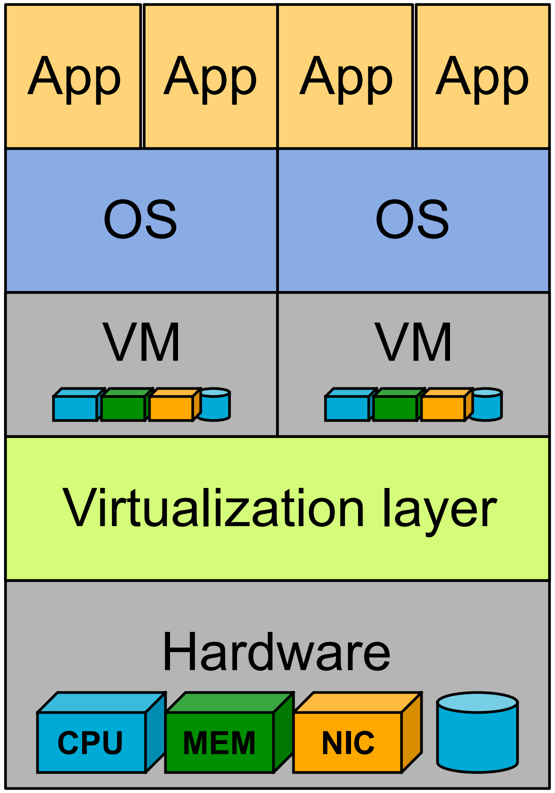
System virtualization starting point

- Physical hardware
  - Processors, memory, chipset, I/O devices, etc.
  - Resources often grossly underutilized
- Software
  - Tightly coupled to physical hardware
  - Single active OS instance
  - OS controls hardware



Adding a virtualization layer

- Software abstraction
  – Behaves like hardware
  – Encapsulates all OS and application state
- Virtualization layer
  – Extra level of indirection
  – Decouples hardware, OS
  – Enforces isolation
  – Multiplexes physical hardware across VMs



Virtual Machine Monitor

- Classic definition*

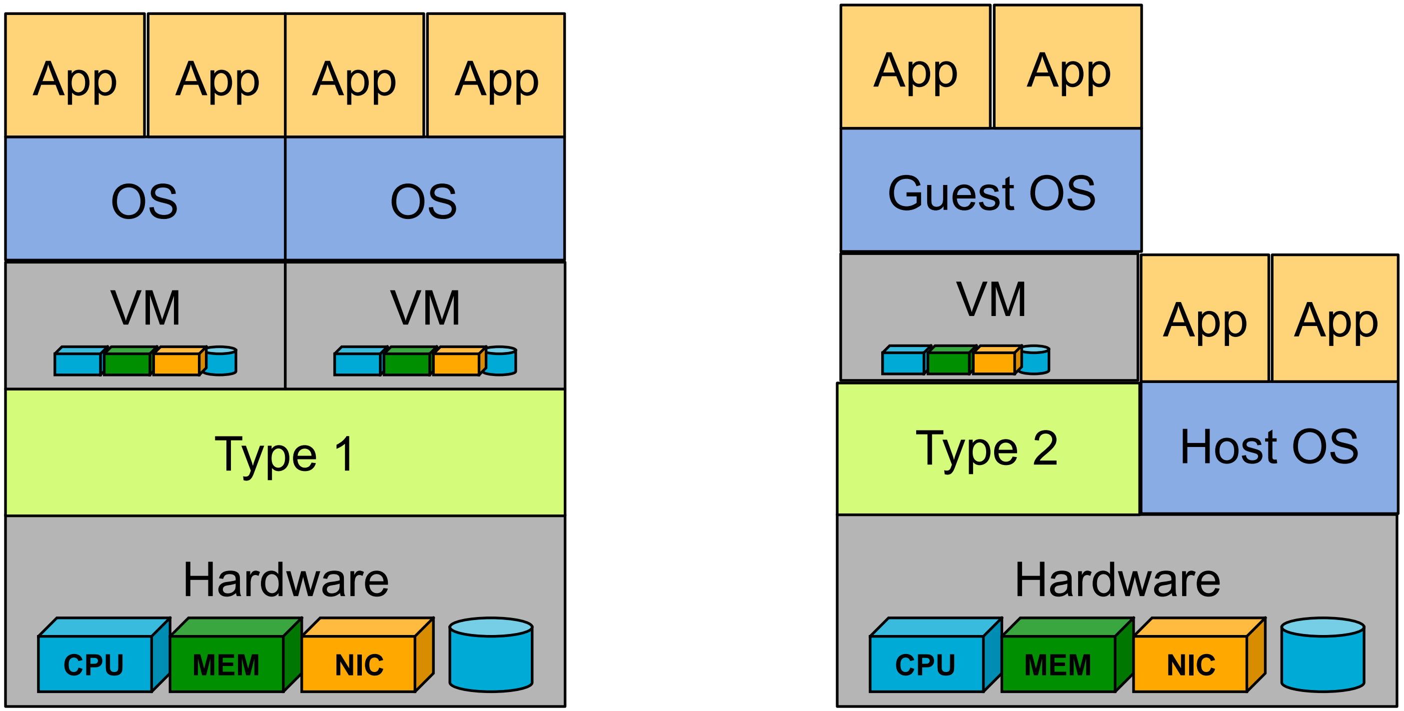
> … an efficient, isolated duplicate of the real machine. … the VMM provides an environment … essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.

- ➔VMM properties
  - Fidelity – Software on the VMM executes same as on the hardware, other than for timing effects
  - Performance – Hardware runs most of the instructions w/o the VMM involvement
  - Safety and isolation – VMM manages all hardware resources

*Popek, G. J.; Goldberg, R. P. "Formal requirements for virtualizable third generation architectures". Communications of the ACM, July 1974.

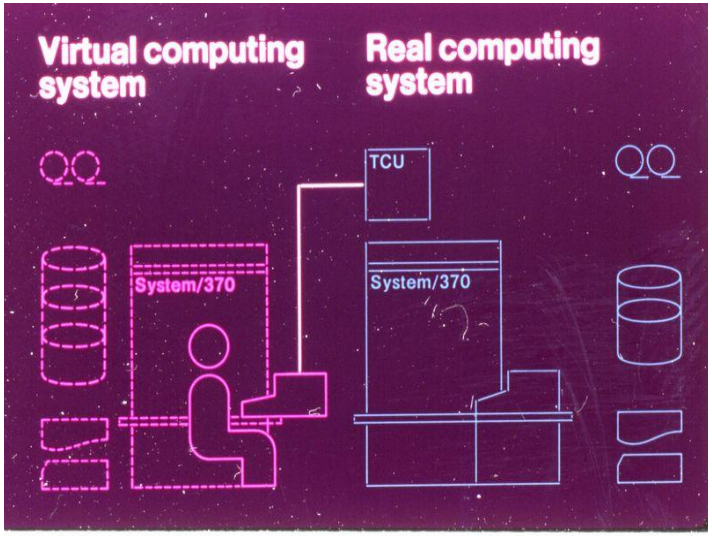
Type 1 and 2 hypervisors

Virtualization method	Type 1	Type 2
Without HW support	ESX Server 1.0	VMware Workstation 1
Paravirtualization	Xen 1.0, Denali	
With HW support	vSphere, Xen, Palacios	VMware Fusion, KVM
Process virtualization		Wine



Classic virtualization

- Classical VMM, back to the 1960s
  - A couple of projects SIMMON and CP-40 for IBM System/360
    - CP-40 ➔CP-67 ➔CP/CMS➔IBM VM/370
  - IBM mainframes in ’72 System/370 (later /390 and z-series)
  - Co-designed proprietary hardware, OS, VMM
  - “Trap and emulate” model
- Applications
  - Timeshare several single-user OS instances on expensive hardware
  - Compatibility



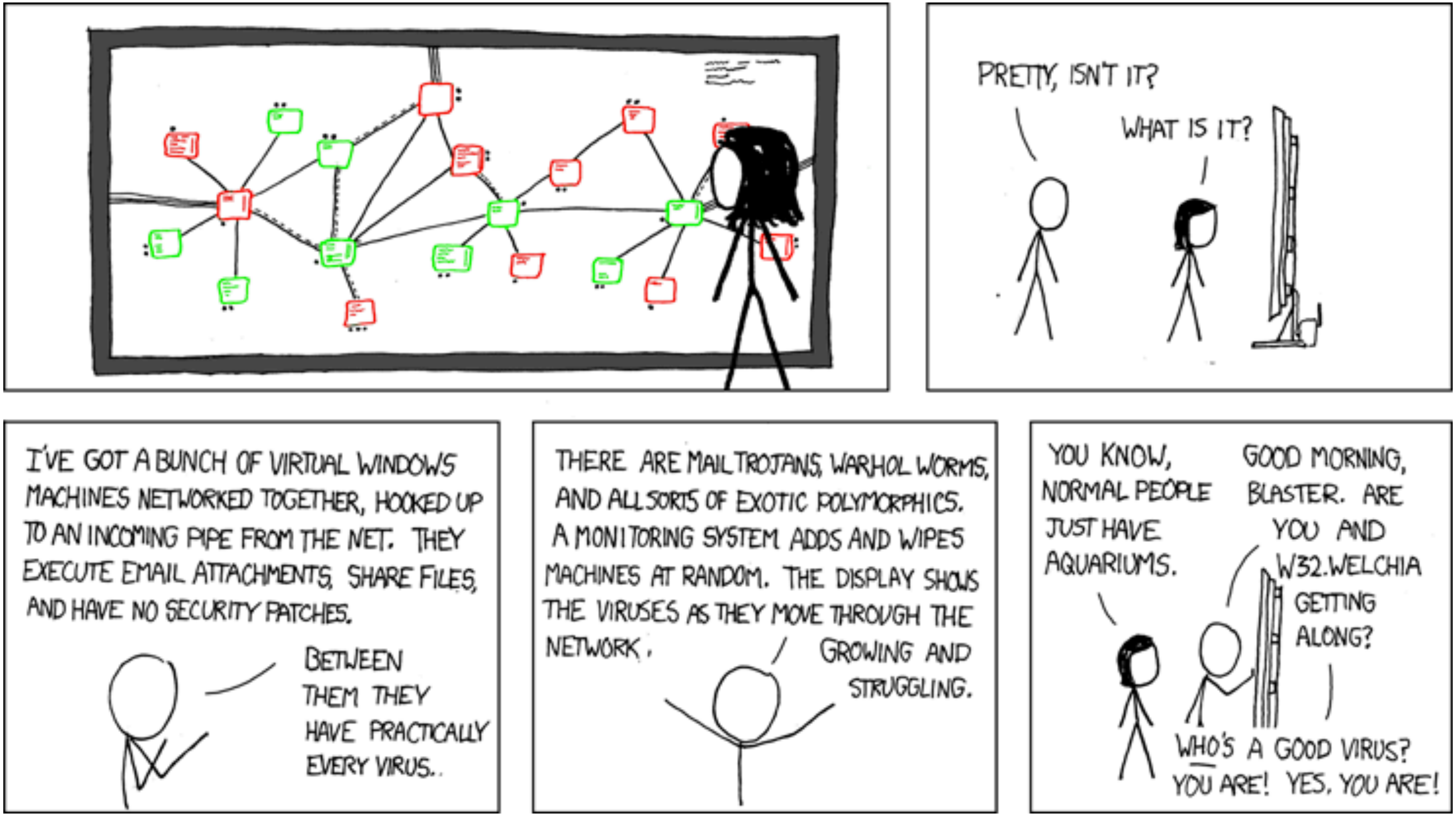
From IBM VM/370 product announcement, *ca*. 1972

Modern virtualization renaissance

- Recent proliferation of VMs
  – Considered exotic mainframe technology in 90s
  – Now pervasive in datacenters and clouds
  – Huge commercial success (partly lead by Rosenblum's VMware, started with DISCO '97)

- Why?
  – Introduction on commodity x86 hardware
  – Ability to "do more with less" saves $$$
  – Innovative new capabilities
  – Extremely versatile technology

And now a short break …

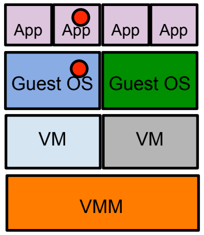


Virtualization’s building blocks

- Processor virtualization
  - Trap and Emulate
  - Binary Translation
- Memory virtualization
- I/O virtualization

Virtualizing the CPU

- Running a virtual machine
  - Limited direct execution – remember processes?
  - Want to boot a VM, jump to first address and go …
- To switch between two VMs, a machine switch
  - Like a processes switch, but using a Virtual CPU
  - VCPU – the sate of the CPU as the guest machine believes it to be; a VCPU per guest (~PCB)
  - The VM loaded can be either within the OS itself (i.e., executing a sys call) or within a process running on that OS
- *Easy right?*



Virtualizing the CPU

- What if the OS or running app tries to perform a privileged instruction?
  - E.g., update the TLB in a SW managed TLB, or make a syscall
  - The OS cannot be allowed to do it, the VM must intercept it
  - Consider `open(path, flags, mode)` in FreeBSD

```
push dword mode
push dword flags
push dword path
mov eax, 5
push eax
int 80h
```

Process	**Hardware**	**OS**
1.Execute instruction (add, load, etc) **2.System call: trap to OS**		
	3.Switch to kernel mode; jump to trap handler	
		4.In kernel mode; handle syscall; return from trap
	5.Switch to user mode; return to user code	
6.Resume execution		

Trap and emulate

- On a virtualized environment …trap and emulate
  - Kernel in guest attempts a privileged instruction, traps to VMM
  - VMM emulates the requested action, updates VCPU, and returns control to VM

Process	OS	VMM
1.System call: trap to OS		
		2.Process trapped, call OS trap handler (at reduce privilege)
	3.OS trap handler; decode, trap and execute syscall; return from trap	
		4.OS tried to return from trap; do real return
5.Resume execution		

Why the VMM doesn't handle the system call?

Trap and emulate

- How does the VMM know where the trap handler is?
  - The OS tried to installed them at boot time, a privilege instruction, and the VMM took notes

Process	OS	VMM
1.System call: trap to OS		
		2.Process trapped, call OS trap handler (at reduce privilege)
	3.OS trap handler; decode, trap and execute syscall; return from trap	
		4.OS tried to return from trap; do real return
5.Resume execution		

- Trap costs may be high

Trap and emulate

- What mode should the OS run in?
  - Not kernel mode, or it would get unrestricted access to the hardware
- VMM consumes a privilege level
  - In DISCO, use a MIPS supervisor mode
    - Access additional memory, but not privilege instructions
    - Additional memory is enough for the OS to keep its data
  - x86 has four protection mode rings – 3 (least privileged) for user processes, 0 for kernel
  - Without it, run in user mode and virtualize protection levels, use page tables and TLBs to protect OS data structs

Binary translation

- Not all architectures support trap and emulate, i.e. are
- *Strictly virtualizable*
  - A processor is strictly virtualizable if, when executed in a lesser privileged mode:
    - All instructions that access privileged state trap
    - All instructions either trap or execute identically
  - x86 was not strictly virtualizable ☹ until 2005 – example problem instruction – “pop flags” or `popf`
    - In privilege mode, popf may change system flags such as IF, which controls interrupt delivery
    - For a de-privileged guest, we need the kernel to trap so that the VMM can emulate the virtual IF
    - *But all user-mode popf simply suppresses attempts to modify IF*

Binary translation

- Simple form – fidelity and safety, but bad performance
- Dynamically translate potentially dangerous instructions (non-virtualizable) into safe ones
  - VMM inspects next block of instr (up to a control transfer)
  - Translate each instr and cache translation, jump to start of the translated block and run with VCPU state on HW

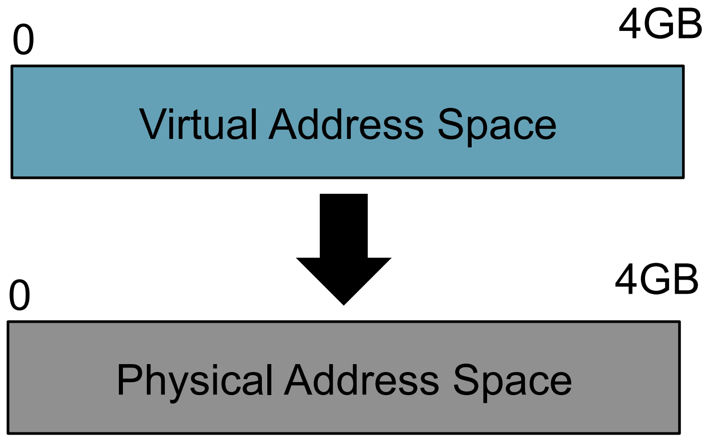
- Issues with binary translation
  - Translation cache management
  - PC synchronization on interrupts
  - Self-modifying code
    - Notified on writes to translated guest code
  - Protecting VMM from guest

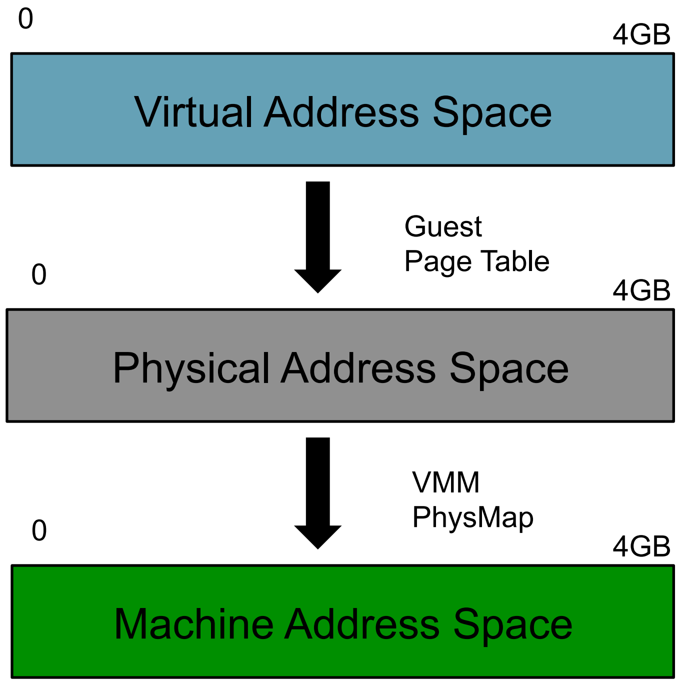
Virtualization’s building blocks

- Processor virtualization
  - Trap and Emulate
  - Binary Translation
- Memory virtualization
- I/O virtualization

Virtualizing memory

- Add another level of indirection for memory
- Physical memory is now a virtualization on top of "machine memory"
  - Each OS maps virtual-to-physical addresses via its per-process page table
  - VMM maps physical mappings to underlying machines via its per-OS page tables





Traditional address spaces

- Consider address translation with a software-managed TLB

Process	OS
1.Load from memory; TLB miss: trap to OS	
	2.OS TLB miss handler; extract VPN from VA; do page table lookup; if present and valid, get PFN, update TLB; return from trap
3.Resume execution; instruction is retried, results in TLB hit	

TLB miss flow with virtualization

- With a VMM, upon a miss TLB is not the OS TLB miss handler that runs but the VMM's

Process	OS	VMM
1.Load from memory; TLB miss: trap to OS		
		2.VMM TLB miss handler; call into OS TLB handler (reducing privilege)
	3.OS TLB miss handler; extract VPN from VA; do page table lookup; if present and valid, get PFN, update TLB	
		4.Trap handler; unprivileged code trying to update TLB; OS is trying to install VPN-PFN mapping; Update TLB instead with VPN-to-MFN (privileged); Jump back to OS (reducing privilege)
	5.Return from trap	
6.Resume execution; instruction is retried, results in TLB hit		

Issues with virtualized memory

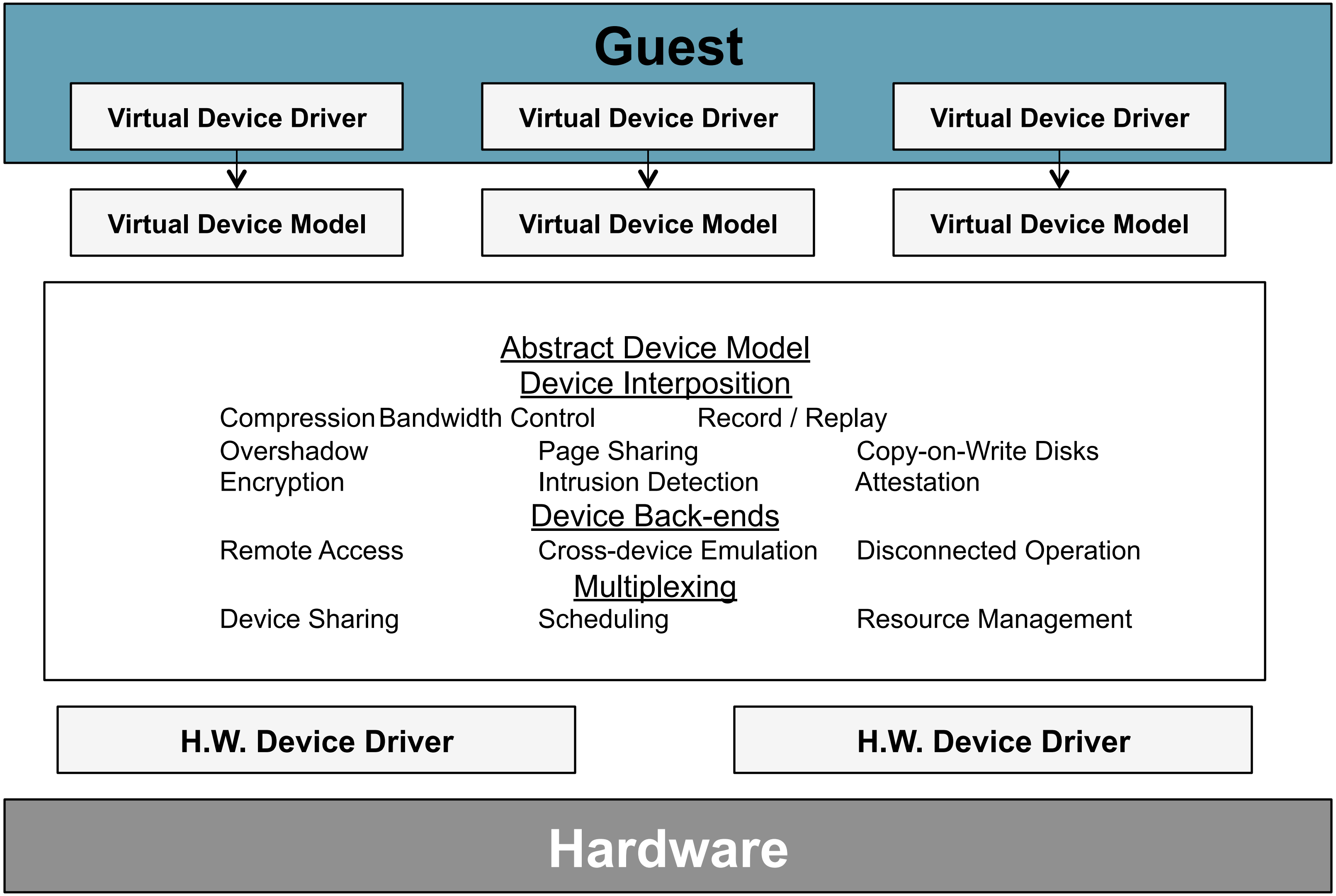
- Positives
  - Simplifies monitor design
- TLB misses significantly more expensive
  - DISCO included a software TLB to reduce this cost
- With a hardware-managed TLB
  - HW walks the TLB and updates it as needed
  - VMM must monitor changes the OS makes to each page table and keep a shadow page table mapping the virtual address of each process to the VMM’s desired machine pages
  - The VMM installs a process’ shadow table when the OS tries to install the process’ OS-level page table

Summary

- Virtualization
  - Not a new concept but technological improvements brought a renaissance and many new usages
  - Interested? Take EECS 441 – much of it focused on Linux kernel development

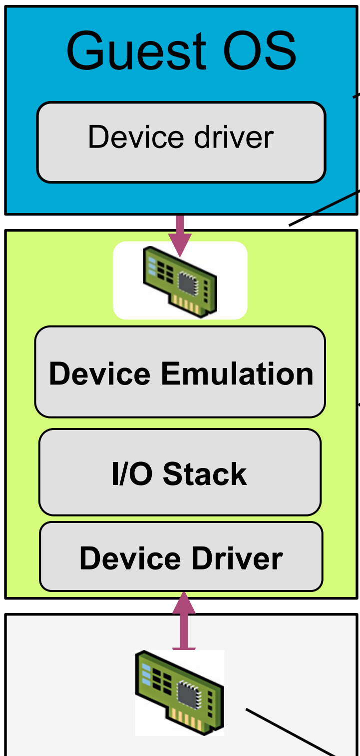
- Second half of the quarter
  - The power and danger of concurrency
  - I/O and persistent storage

I/O Virtualization



I/O Virtualization implementations

Virtualized I/O



VMware ESX

Guest device driver

Virtual device

Virtualization layer

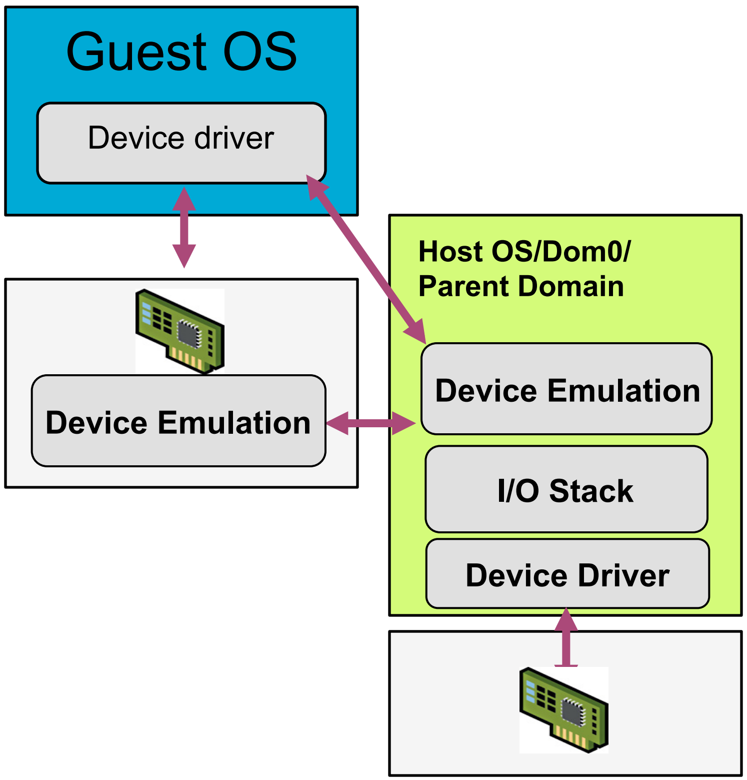
- Emulates the virtual device
- Remaps guest and real I/O addresses
- Multiplexes and drives the physical device
- I/O features

Real device

- May be different from virtual

I/O Virtualization implementations

Virtualized I/O: Hosted or Split

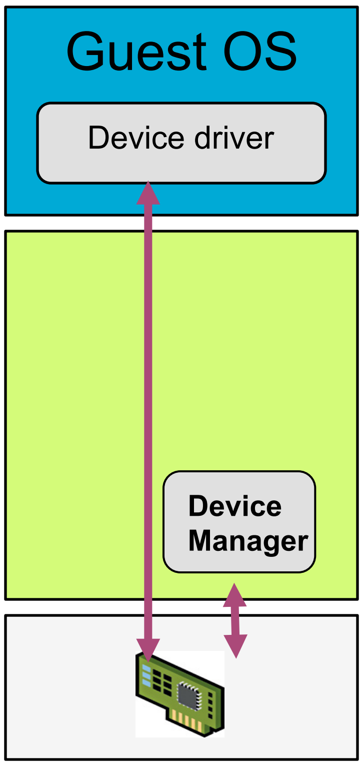


VMware Workstation, VMware Server, Xen, Microsoft Hyper-V, Virtual Server

- Only Domain0, created at boot, has direct access
  - Can also create/terminate domains, control scheduling parameters, physical memory allocation, ..
- Others domains access through virtual device
- Information pass between domains through shared-mem, asynchronous buffer-descriptor rings

I/O Virtualization implementations

Passthrough I/O



VMware ESX (FPT)

- Fast but inflexible
- How to handle
  - Discontinuous physical memory
  - Read-only physical memory (COW)
  - Paged out physical memory
  - VM migration
  - …