# Virtualization

- Learning Objectives
  - Explain how today's virtualization movement is actually a re-invention of the past.
  - Explain how virtualization works.
  - Discuss the technical challenges to virtualization.
  - Identify the key similarities and differences between a virtual machine monitor (hypervisor) and an operating system.
- Topics
  - In the beginning: VM/370
  - What is virtualization?
  - Challenges to virtualization and how to address them.

# Virtual Machines

- Re-inventing operating systems all over again!
- In the 1960's, IBM developed a family of machines (System/360, System/370) and an operating system called VM/360.
  - Enable multiple users to share a single machine by giving each user his/her own machine.
  - Real hardware partitioned into per-"machine" components:
    - Disk
    - Processors
    - Memory
  - Compared to operating systems, it's just a different way of sharing resources.

| **Conventional OS** | **Virtual Machine Monitor** |
|---|---|
| Share processor via processes | Share processor across "machines" |
| Share disk through a global namespace | Share disk by giving each machine its own "disk" and "file system namespace" |
| Share memory through virtual memory | Share memory by partitioning among machines. |

# API

- Conventional operating systems use programmatic APIs: system calls.
- Virtual machine monitors use the hardware as its API.

*Think about this for a minute …*

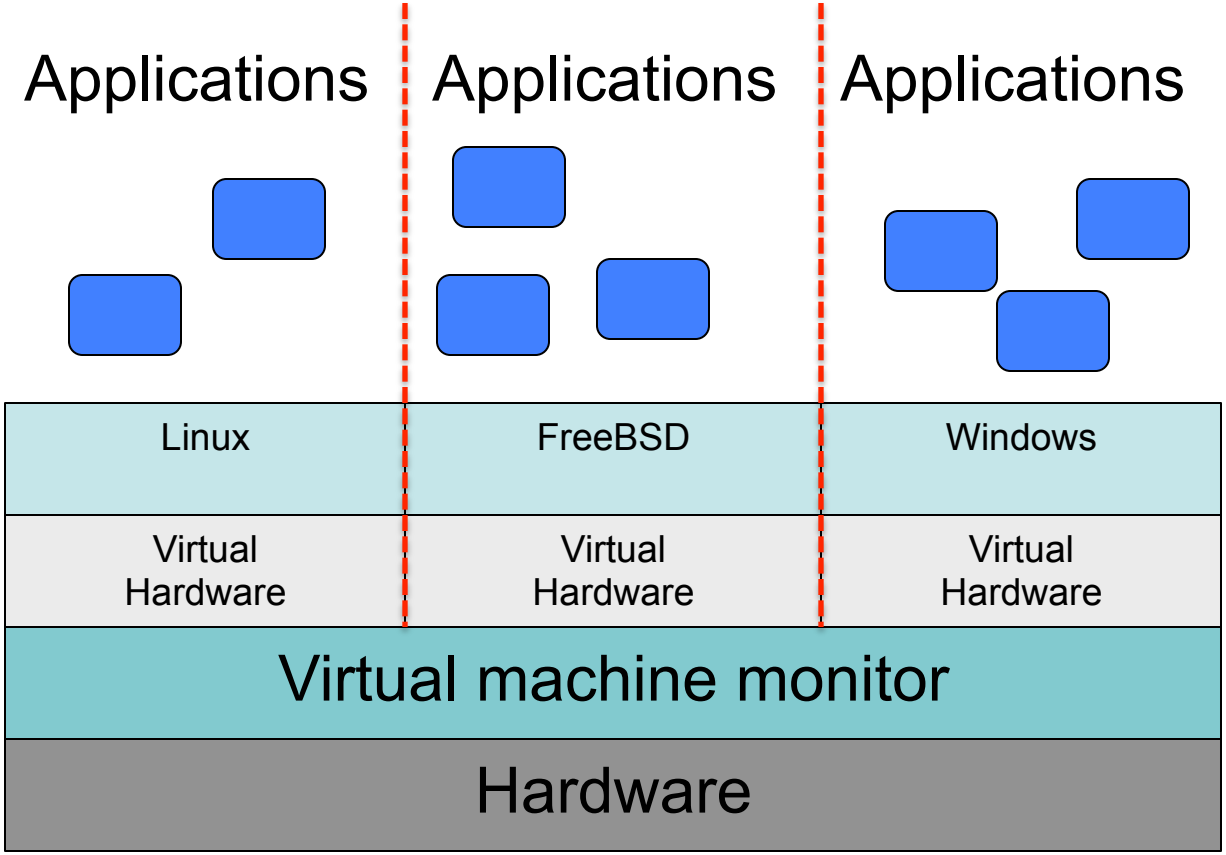
- Virtual machines largely died out in the 80's.
- They made a (major) a resurgence in the 90's and continue to do so.

*WHY?*

# Why?

- Original VM/370: Only way to provide sharing.
- Motivation when re-introduced:
  - Allow geeks like us to run UNIX/Linux and still read MS docs.
  - Provide effective fault containment on multiprocessors.
- Distracting motivations:
  - One OS can reboot while others keep running.
  - Can test out new installations (both OS or major servers).
- Today's motivations:
  - Security: can run dangerous things in a "sandbox."
  - Provide a new software delivery vehicle.
  - Optimized application and OS integration.
  - Makes it easy for infrastructure providers to use resources efficiently.
  - Provides isolation between different customers.

# Virtual Machine Architecture



# So, what is the VMM?

- A layer of software that multiplexes the actual hardware.
- While an OS multiplexes among processes, a VMM multiplexes among machines.
- *From the point of view of the OS on a virtual machine or an application, anything running on a different virtual machine looks the same as if it were running on a different piece of hardware.*

# Virtualization Goals

- Must give each virtual machine the *complete* illusion of a real machine, including:
  - I/O devices
  - interrupts,
  - virtual memory hardware
  - protection levels
  - etc...
- Must be 100% compatible with existing hardware
  - Run existing, unmodified operating systems and applications directly on the VM!!
- Must completely isolate VMs from each other
  - Should not allow one VM to stomp on another's memory or CPU state
- Must have high performance
  - Otherwise nobody will want to use it.

# Why is Virtualization hard?

- Terminology:
  - VMM:The layer of software that lets you run virtual machines.
  - HostOS: An OS that acts as a VMM; runs natively on the hardware.
  - GuestOS: The OS running in a virtual machine.
- Guest OS needs to call privileged instructions
  - E.g., to perform I/O, manipulate hardware state, etc.
  - Should we let it do this directly??
- Guest OS needs to manipulate page tables
  - To set up virtual->physical mappings for its applications
  - Why is this potentially a bad idea?
- Guest OS needs to believe it's running on a real machine
  - Must support complete instruction set of processor
  - Must support all the weird virtual memory features (segments, paging, etc.)
  - Must support (at least some) “real” I/O devices (disk, network, etc.)

# In what mode does a VM run?

- Typically an operating system runs in supervisor mode, but if we really want to prevent individual machines from executing privileged instructions or directly manipulating the memory manager, then we cannot run a virtual machine in supervisor mode!
  - Running the VM in unprivileged mode is the only way to ensure that a Guest OS doesn't do bad stuff.
  - How do we get the Guest OS to behave like an OS then?
- Option 1: Interpretation
  - The VMM is simply a software layer that interprets every CPU instruction executed by the VM.
  - Can safely emulate privileged instructions.
  - All machine accesses (CPU execution, memory access, I/O) go through VMM.
  - This works, but …
    - It's SLOW!
    - VMM is a full CPU emulator (e.g., superset of Sys161)

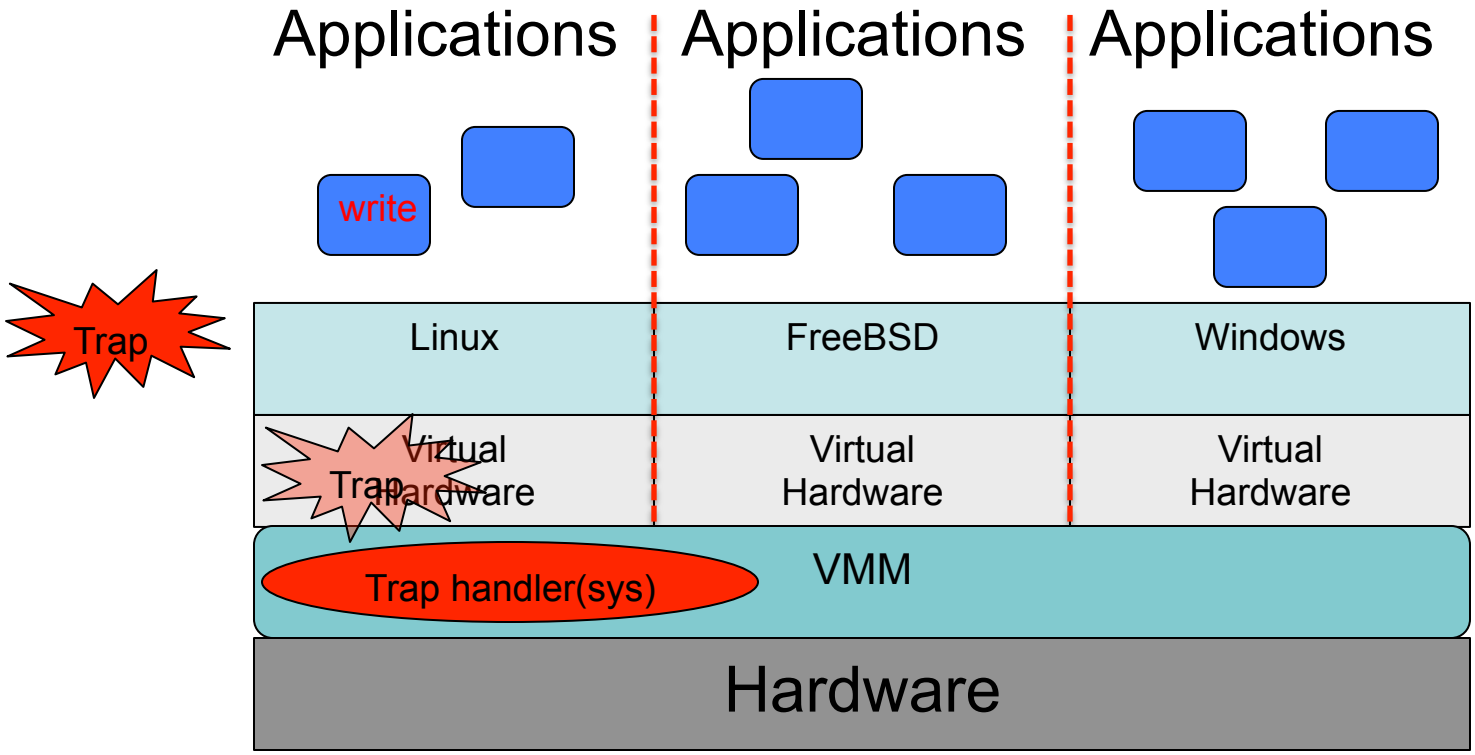
# Option 2: Direct Execution

- Let the Guest OS run directly on the machine, but in user mode.
  - Much faster.
- What happens when the guest issues a privileged instruction?

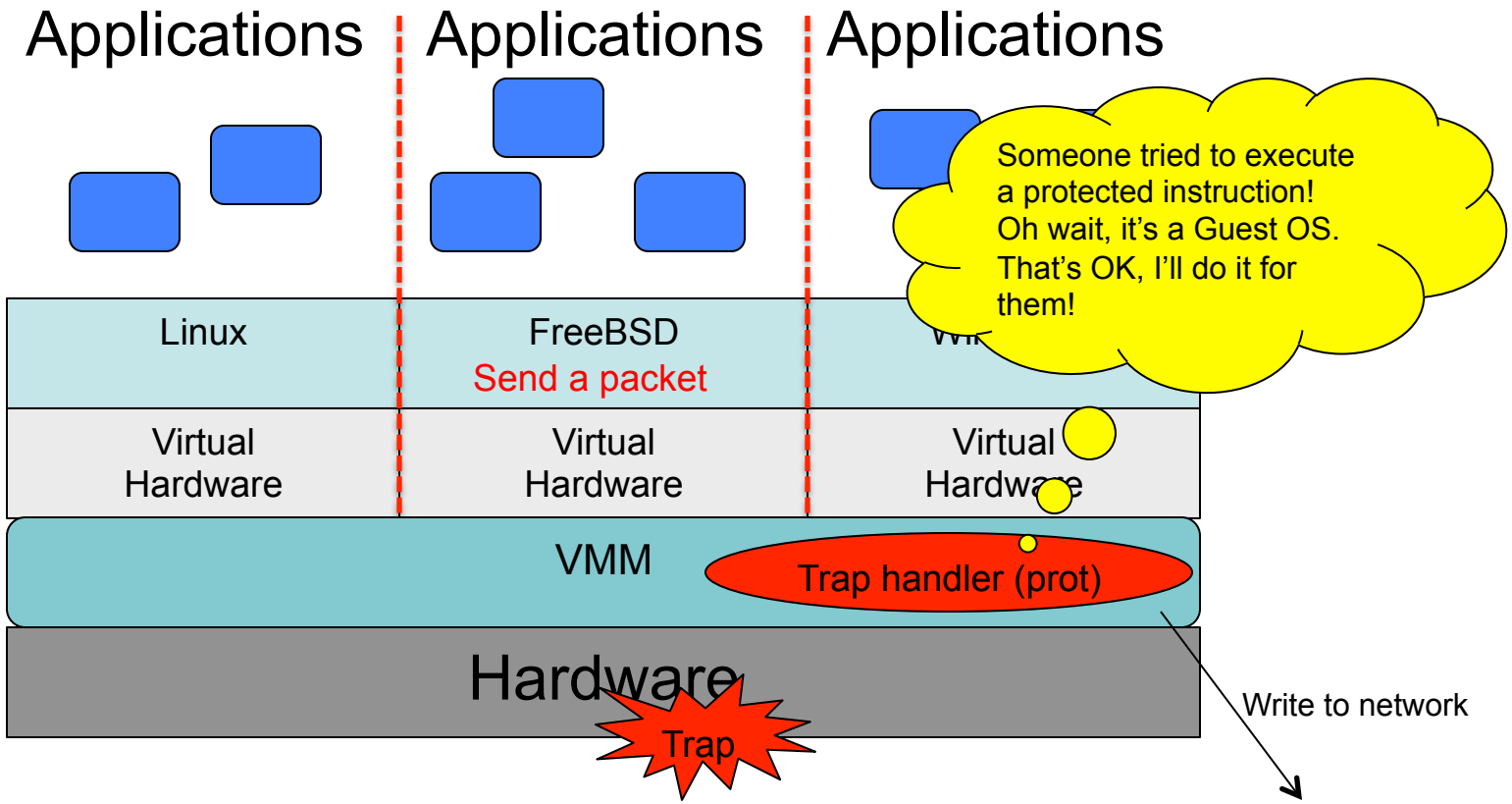
# Option 2: Direct Execution

- Let the Guest OS run directly on the machine, but in user mode.
  - Much faster.
- What happens when the guest issues a privileged instruction?
  - Trap!
  - Who catches this trap?
    - The software running in supervisor mode, which is the VMM!
  - The VMM examines the faulting instruction and decides what to do:
    - Handle the fault (issue the privileged instruction for the Guest).
    - Disallow the operation and kill the VM

# Example: Application issues System Call



# Example: Guest accessing hardware



# Protection

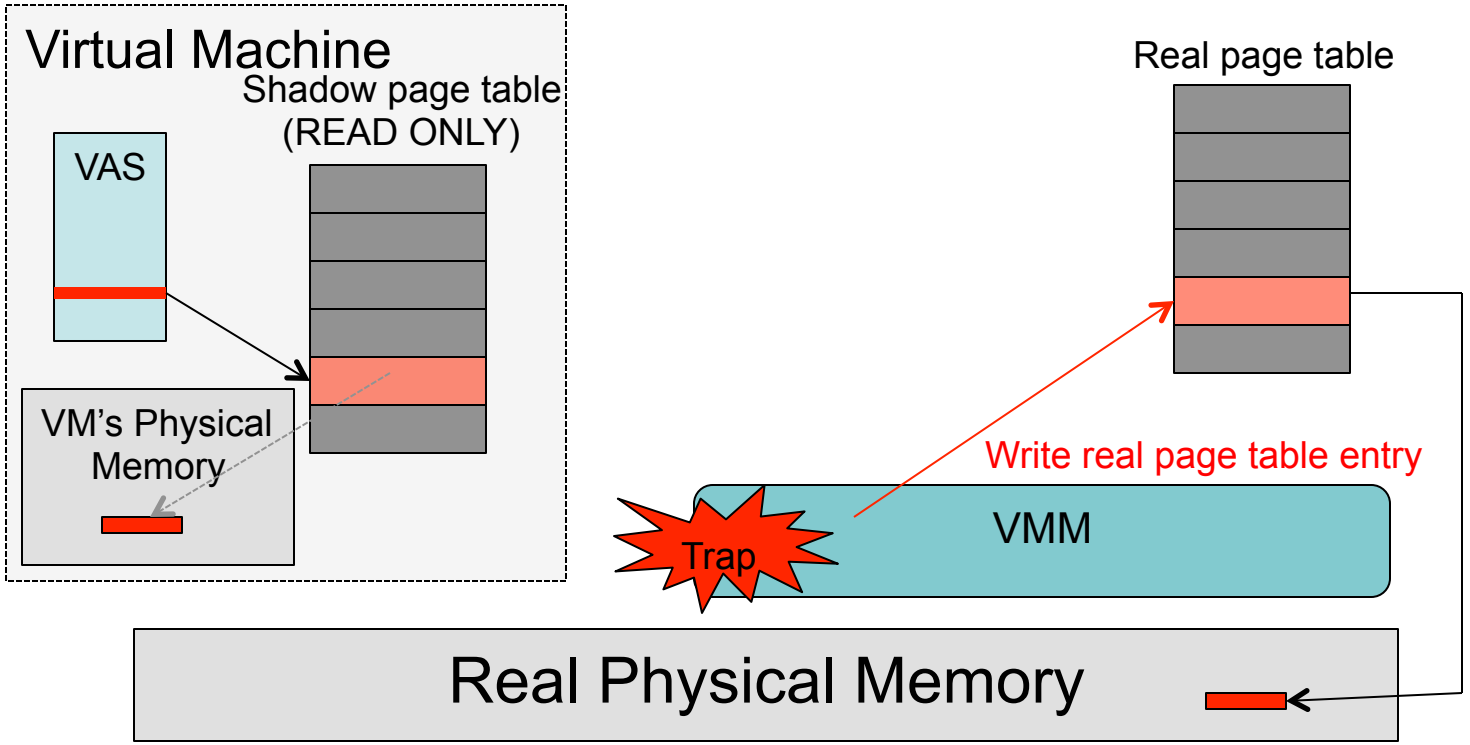
- If the OS and regular applications are both running in unprivileged mode, how do we protect them from each other?
  - For example, how do we prevent the user application from stomping all over kernel memory?
- Recall that the x86 has multiple protection levels (four to be exact).
- Let's take advantage of them:
  - Run user code in Ring 3
  - Run the GuestOS in Ring 1
  - Run the VMM in Ring 0

# Onward: Memory Management

- The Guest OS thinks that it has its own page tables (because that's what operating systems do and because when it does actually perform operations on behalf of processes, it needs to be able to translate addresses correctly).
- But … we still do not want the Guest OS to actually manipulate PTEs (or TLB entries).
  - If we let it directly manipulate PTEs, it could map *any* page.
  - Including pages from other VMs!
  - That would be *bad*™.

# Shadow Page Tables

- Give each virtual machine make-believe page tables (called *shadow page tables*).
  - The Guest OS will trap to the VMM when it manipulates page tables, allowing the VMM to update the real page tables appropriately.



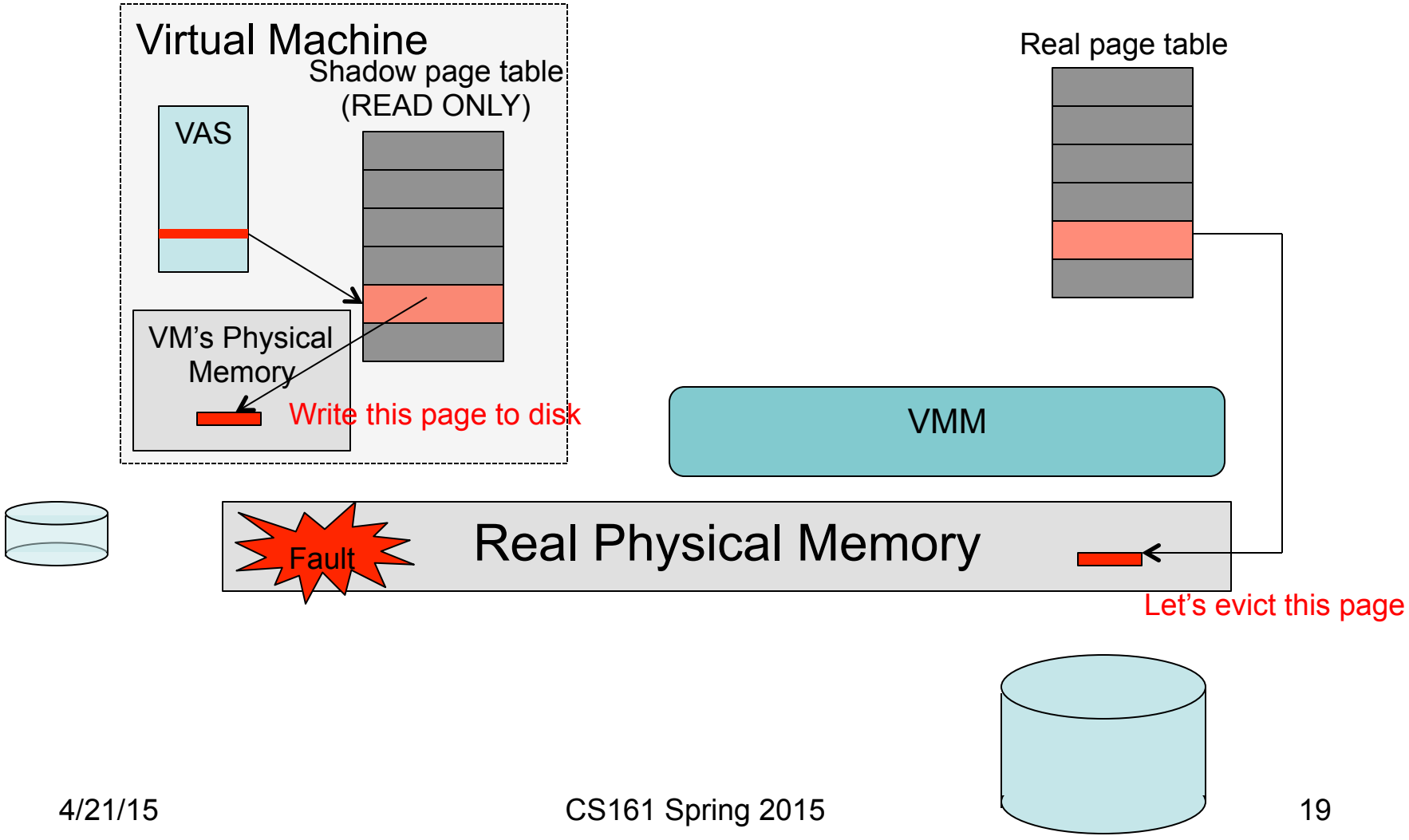
# Physical Memory Management

- How does the VMM allocate memory across virtual machines?
  - Each VM/OS needs “some” physical memory (enough to run).
  - How do you reclaim space from a VM?
- Option 1: Swap the entire VM (just like the OS swaps a process)
- Option 2: VMM-managed paging (just like the OS pages processes).
- Is the problem any different? Can’t we just use the same techniques in the VMM that we use in the OS?

# VMM-based Paging

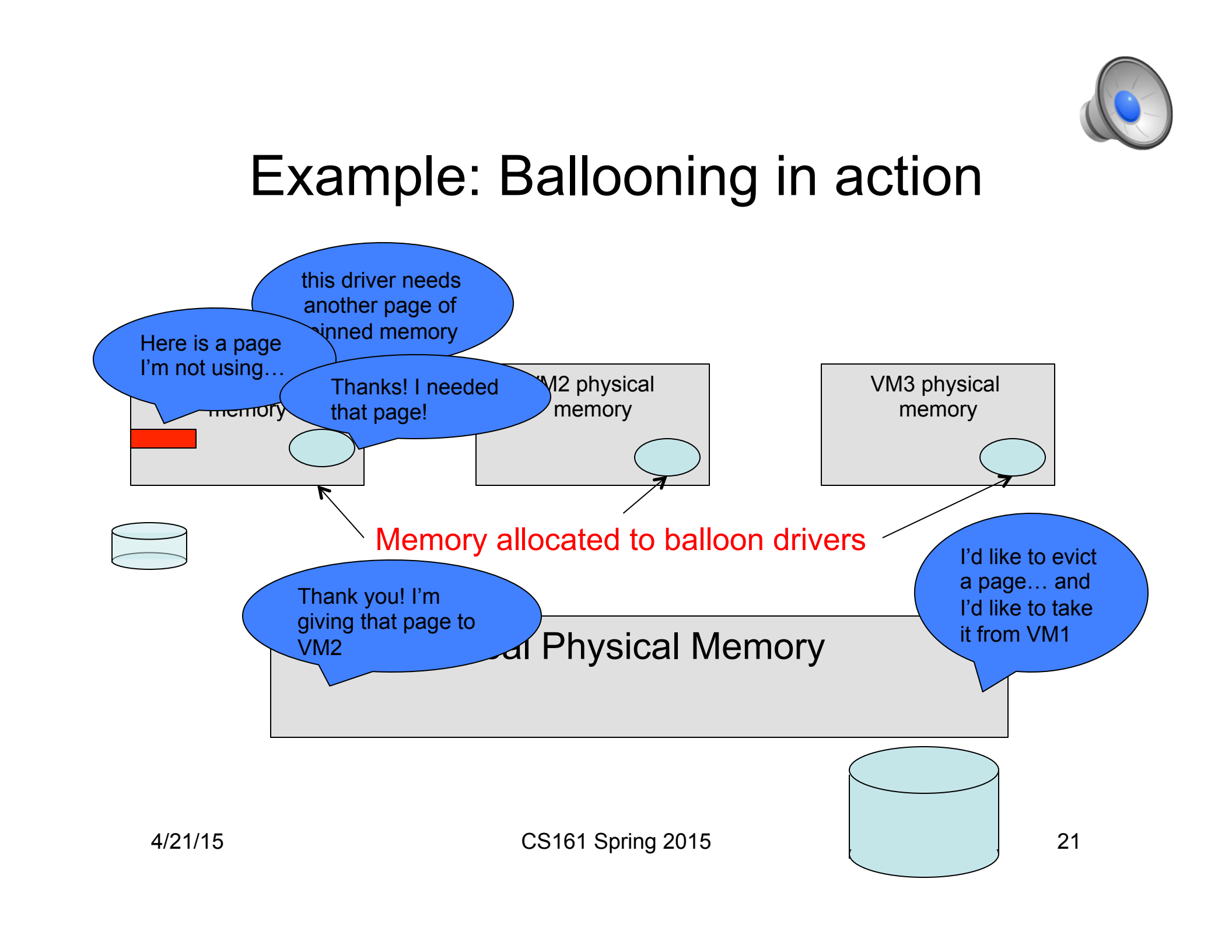
- VMM has no idea what pages are used for (or even when they were used recently).
  - VMM can't distinguish between the operating system's pages and application's pages.
  - If the VMM is solely responsible for paging, bad things can happen.
- Double Paging:
  - VMM picks a page from the Guest OS and evicts it
    - This just means that a page that appears to be in the virtual machine's physical memory is really on disk.
  - Now imagine that the Guest OS is under memory pressure and it decides to evict a page from its physical memory.
    - It could choose a page that the VMM has already evicted.
    - Now, the Guest OS is going to write it … a second time!
    - And, in order for the Guest OS to access it to write it to its swap partition, it's going to need to page it back in.
    - This is also *bad*™.

# Example: Double Paging



# Avoiding Double Paging: Ballooning

- Whenever we need to evict a real, physical page, we want to trick a Guest OS into picking the page to pick.
- This suggests that we need the Guest OS to feel memory pressure.
- How do we induce memory pressure on the guest?
- Add a balloon driver to each Guest OS
  - The driver allocates (pinned) physical pages.
  - When you "inflate the balloon" it pins more pages.
  - These pages aren't actually needed; all they do is consume part of the virtual machine's address space; thus they are perfect candidates for the VMM to take away from the Guest!



# Example: Ballooning in action

# Optimizing Memory Usage

- In many cases, you may run many copies of the same Guest OS.
  - These copies use many of the same pages.
  - Just like an OS would like to avoid multiple copies of the same text page, the VMM would like to avoid having multiple copies of the same (OS) text page.
- How can you identify redundant pages?
  - Borrow a page from the file system folks who do deduplication.
  - Maintain a table of the hash values of all the contents of all the pages in memory.
  - When you are about to allocate a new chunk of real physical memory for a page, lookup its hash value.
  - If the hash is already in the table, compare contents to be sure.
  - If they really are the same, reuse the physical page marking it copy-on-write.
- This technique is called “content-based page sharing.”

# I/O Device Interfaces

- All access to I/O devices must go through VMM
  - This can be very expensive!
- Many real I/O devices have complex interfaces.
  - Many programmed I/O operations to do simple things
  - E.g., Reading a disk block or sending an Ethernet frame require a lot of interaction with the VMM.
- Observation: The VMM provides “virtual” devices
  - Those devices can be dirt simple!
  - Example:
    - Simple network driver with two PIO interfaces.
    - One PIO transmits a packet, another receives a packet.
- Lesson: If you get to design the hardware, make it simple!

# Hardware Nastiness

- So far, we've pretended that it's actually possible to virtualize the CPU, but some architectures are not actually fully virtualizable:
  - Every privileged instruction must trap to the OS when executed in user mode.
  - Except for protection, instructions in privileged and unprivileged mode must operate identically.
- Guess what: the x86 is ***not*** virtualizable!
- Example: The POPF instruction
  - Restores the CPU state from the stack, into the EFLAGS register.
  - Some bits in the EFLAGS be modified only in supervisor mode.
    - E.g., Interrupt enabled bit
  - When execute in user mode, POPF will ***not*** modify the interrupt enabled bit.
    - This leads to different behavior in different modes.
- This means a Guest OS is not going to observe the behavior it expects if we execute it in user mode.

# VMware Approach

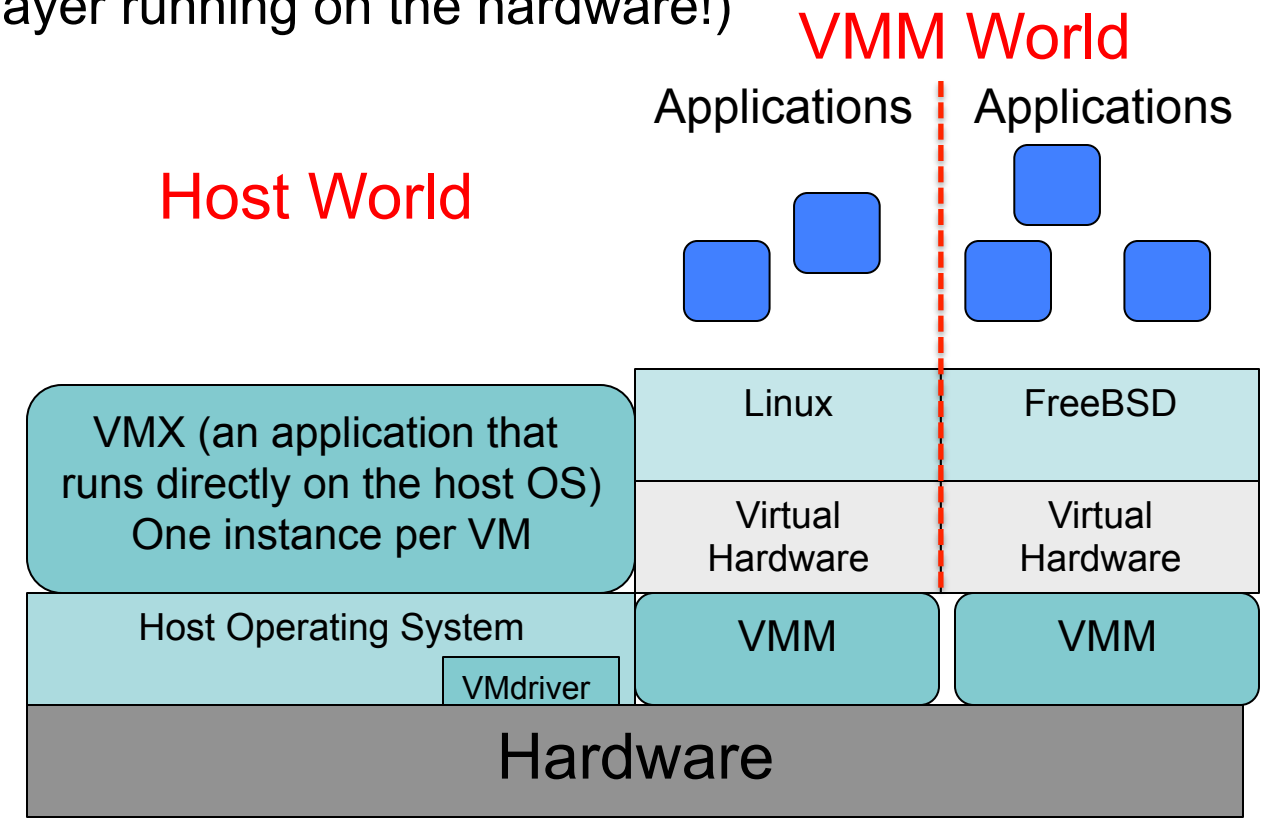
- Binary translation
  - Code running in each VM is scanned on-the-fly for “non-virtualizable” instructions
  - Code is rewritten into a *safe* form
- This allows you to play some games.
  - When you translate code in the Guest OS, you can incorporate some of the functionality from the virtual machine monitor.
  - Example: Fast versions of I/O processing code
- Binary translation can be slow, so much of the secret sauce is in making it fast.
  - Once you’ve translated a page once, you probably don’t want to do it again…
  - Maintain a *translation cache* of recently-rewritten code pages.
  - If you observe the same page again, use its already translated version.
  - Must trap writes to any code page to invalidate the cache
    - This only matters if you allow writable code pages
    - But even if you do, you can get this to work.

# Para-virtualization

- Binary translation is fairly complex and slow.
- Paravirtualization is an alternative:
  - Define a virtualizable subset of the x86 instruction set.
  - Port the Guest OS to the new instruction set architecture.
- Xen uses this approach.
- Advantages
  - Simple
  - Fast
  - Allows you to simplify OS/VMM interaction.
- Disadvantages
  - Requires that you port the Guest OS
  - Requires that Guest OS does not depend on any non-virtualizable features.

# VMware Workstation

- Virtual machine monitor runs as an application (not a special layer running on the hardware!)



# References


- The Origin of the VM/370 Time-Sharing System
  - R. J. Creasy, IBM J. Res. Develop, 25:5, Sep. '81
- The double paging anomaly
  - R. P. Goldberg, R. Hassinger, AFIPS 1974
- DISCO: Running Commodity Operating Systems on Scalable Multiprocessors
  - E. Bugnion, S. Devince, M. Rosenblum, SOSP 1997
- Virtualizing I/O Devices on VMware Workstation's hosted Virtual Machine Monitor
  - J. Sugerman, G. Venkitachalam, B. Lim, USENIX ATC, 2001
- Memory Resource Management in the VMWare ESX Server
  - C. Waldspurger, OSDI'02
- Xen and the Art of Virtualization
  - P. Barham *et al.*, SOSP'03