# CS 5600
## Computer Systems

**Lecture 11: Virtual Machine Monitors**

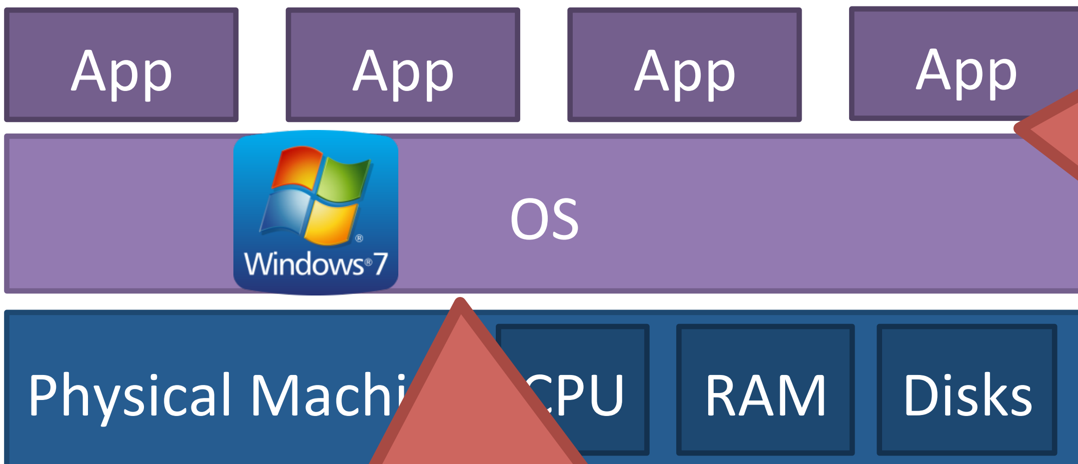
# History

- In the '70s, there were dozens of OSes
  - Unlike today, where Windows and Android dominate
- This created many problems
  - Upgrading hardware or switching hardware vendors meant changing OS
  - However, apps are typically bound to a particular OS
- Virtual machines were used to solve this problem
  - Pioneered by IBM
  - Run multiple OSes concurrently on the same hardware
  - Heavyweight mechanism for maintaining app compatibility

# Terminology

- “Virtual machine” is a loaded term
  - E.g. Java Virtual Machine refers to a runtime environment (software) that can execute Java bytecode
- “VM” is a loaded abbreviation
  - JVM (Java Virtual Machine), Virtual Memory
- For our purposes, we will talk about Virtual Machine Monitors (VMM)
  - VMM is software that allows multiple guest OSes to run concurrent on one physical machine
    - Each guest runs on a virtual machine
  - VMM is sometimes called a hypervisor

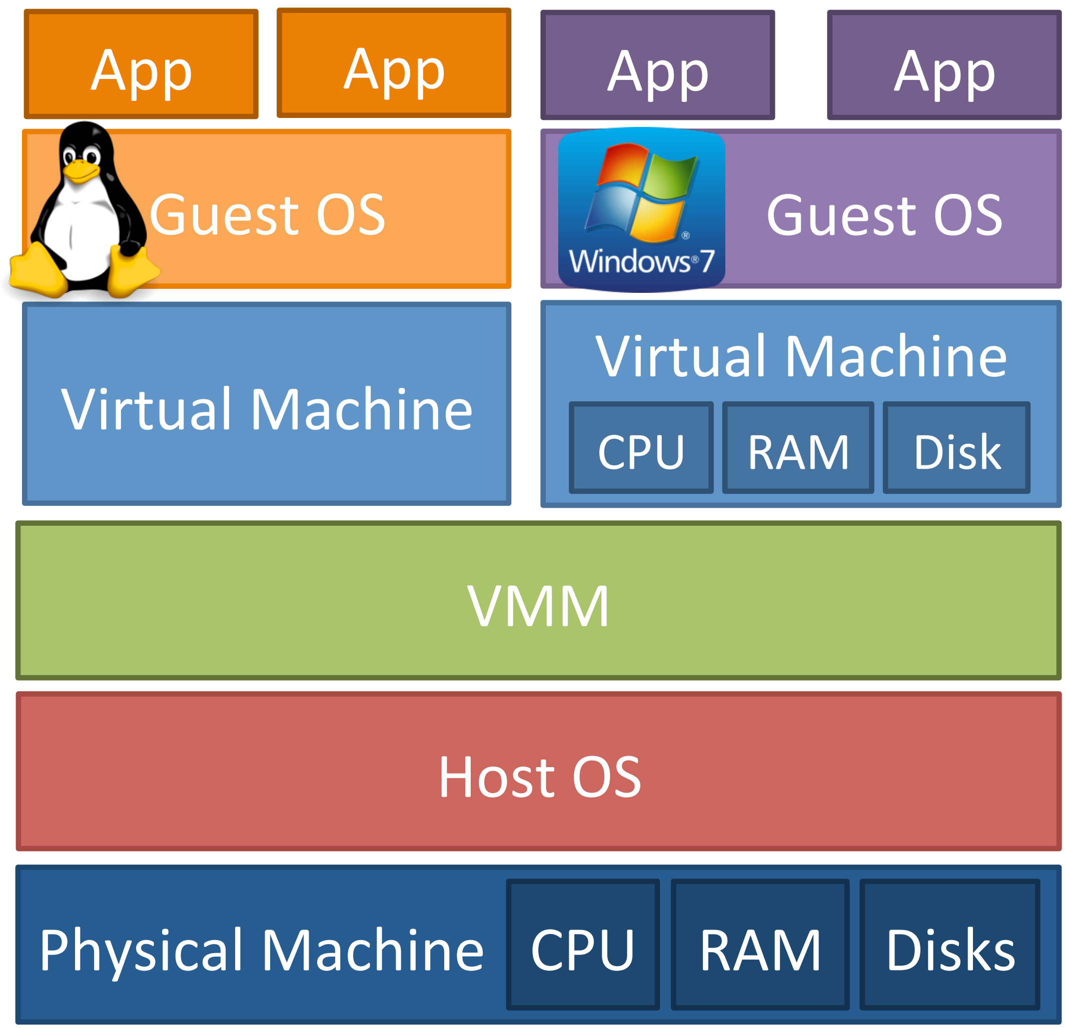
# OS Fundamentals




- OS multiplexes resources between apps
- OS enforces isolation & protection between apps (ring 3)

- The OS manages physical resources
- The OS expects to have privileged access (ring 0)

# VMM Organization and Functions




- Map operations on virtual hw. to physical hw.
- Multiplex resources between guest OSes
- Enforce protection & isolation between guest OSes

# Goals of Virtualization

- Popek and Goldberg, 1974

1. **Fidelity**: software on the VMM executes identically to its execution on hardware
   - Except for timing effects
2. **Performance**: An overwhelming majority of guest instructions are executed by the hardware without VMM intervention
   - Counterexample: the JVM
3. **Safety**: the VMM manages all hardware resources
   - Guests cannot impact each other

# Advantages of Virtualization (1)

- Compatibility and functionality
  – Guests are oblivious to low-level hardware changes
  – Windows apps on Linux or vice-versa
- Consolidation
  – Multiple machines can be combined into one by running the OSes as guests
- Checkpointing and migration
  – A guest OS can be written to disk or sent across the network, reloaded later or on a different machine

# Advantages of Virtualization (2)

- Security
  - If a guest OS is hacked, the others are safe (unless the hacker can escape the guest by exploiting the VMM)
- Multiplatform debugging
  - App writers often target multiple platforms
    - E.g. OS X, Windows, and Linux
  - Would you rather debug on three separate machines, or one machine with two guests?

# Technical Challenges

- x86 is not designed with virtualization in mind
  - Some privileged instructions don’t except properly
  - MMU only supports one layer of virtualization
- These hardware issues violate goal 1 (fidelity)
  - As we will discuss, sophisticated techniques are needed to virtualize x86
  - These techniques work, but they reduce performance
- Modern x86 hardware supports virtualization
  - AMD-V and VT-x for hypervisor context switching
  - RVI (AMD) and EPT (Intel) for MMU virtualization

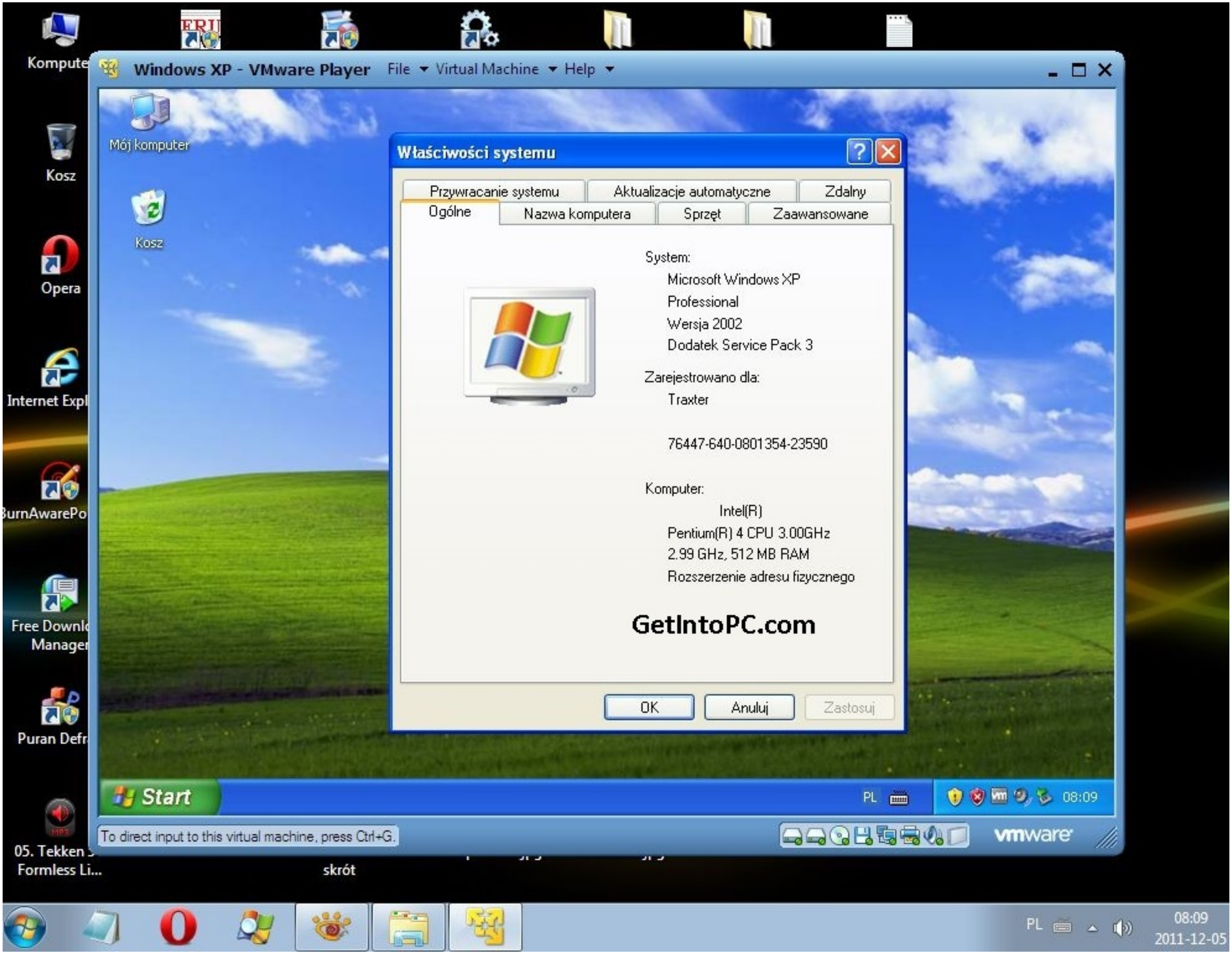
# Performance Challenges

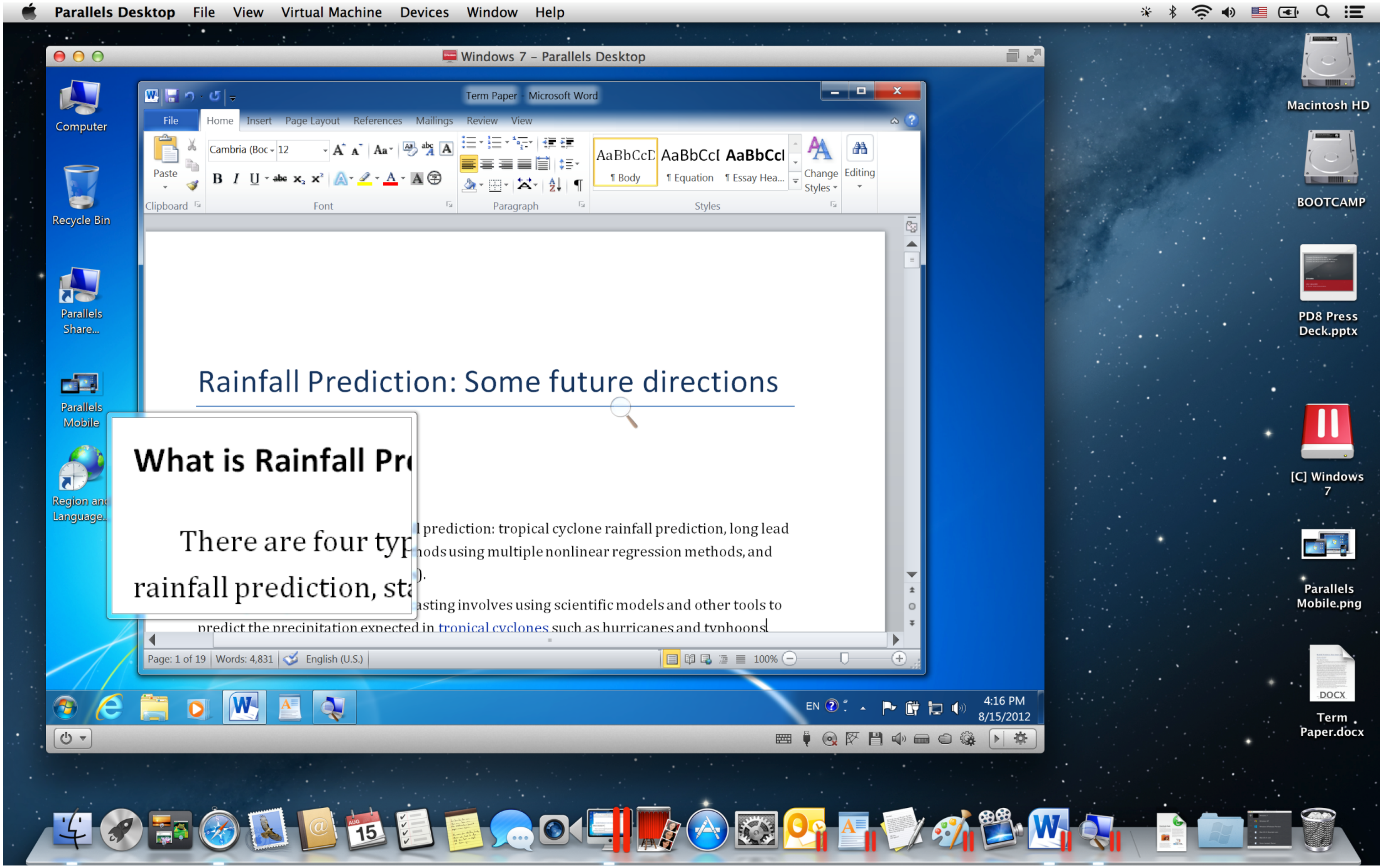
- Memory overhead
  - VMM data structures for virtualized hardware may require lots of memory
- CPU overhead
  - Context switching between VMM and each guest is costly
  - Some instructions and functions (e.g. page allocation) must be virtualized; slower than direct operations
- I/O performance
  - Devices must be shared between guests
  - Virtualized devices (e.g. disks, network) may be slower than the underlying physical devices

- Full Virtualization (VMWare)
- Hardware Support
- Paravirtualization (Xen)

# Full Virtualization

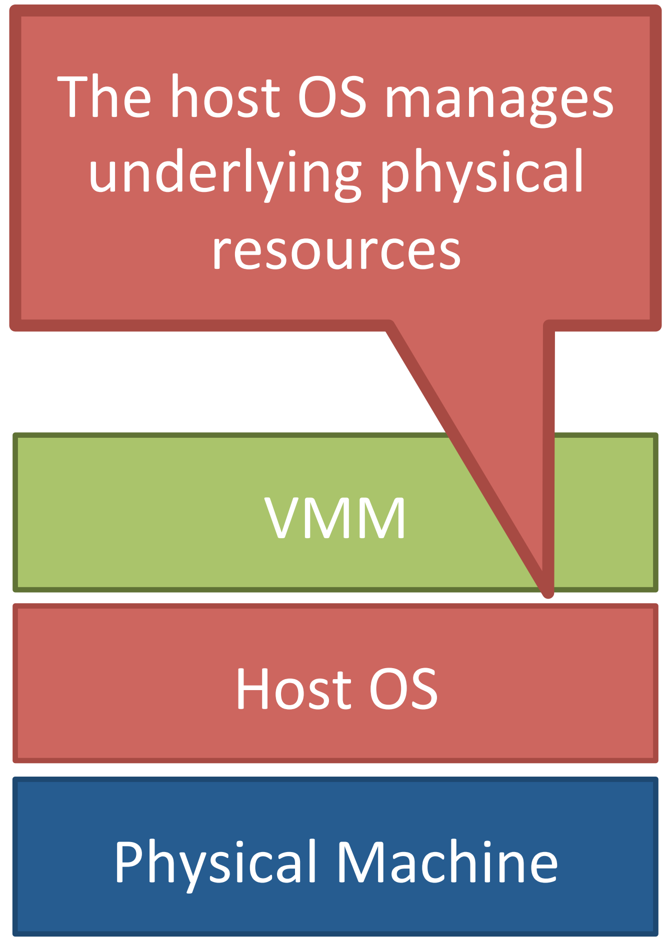
- VMWare implements full virtualization
  - Full → guest OSes do not need to be modified
- Goals:
  - Run unmodified OSes as guests
  - Isolate the guest from the host (safety/security)
  - Share physical devices and resources with the guest
    - CPU, RAM, disk, network, GPU, etc…
- Other full virtualization VMMs:
  - Parallels on OS X
  - Hyper-v on Windows


Windows XP - VMware Player
File
Virtual Machine
Help
Mój komputer
Kosz
Właściwości systemu
Przywracanie systemu
Aktualizacje automatyczne
Zdalny
Ogólne
Nazwa komputera
Sprzęt
Zaawansowane
System:
Microsoft Windows XP
Professional
Wersja 2002
Dodatek Service Pack 3
Zarejestrowano dla:
Traxter
76447-640-0801354-23590
Komputer:
Intel(R)
Pentium(R) 4 CPU 3.00GHz
2.99 GHz, 512 MB RAM
Rozszerzenie adresu fizycznego
GetIntoPC.com
OK
Anuluj
Zastosuj
Start
PL
08:09
To direct input to this virtual machine, press Ctrl+G.
vmware
Kosz
Opera
skrót
PL
08:09
2011-12-05


Parallels Desktop File View Virtual Machine Devices Window Help
Windows 7 – Parallels Desktop
Term Paper - Microsoft Word
File Home Insert Page Layout References Mailings Review View
Rainfall Prediction: Some future directions
What is Rainfall Pr
There are four typ
rainfall prediction, st
prediction: tropical cyclone rainfall prediction, long lead
hods using multiple nonlinear regression methods, and
asting involves using scientific models and other tools to
predict the precipitation expected in tropical cyclones such as hurricanes and typhoons
Page: 1 of 19 Words: 4,831 English (U.S.) 100%
4:16 PM
8/15/2012
Computer
Recycle Bin
Parallels Share...
Parallels Mobile
Region and Language
Macintosh HD
BOOTCAMP
PD8 Press Deck.pptx
[C] Windows 7
Parallels Mobile.png
Term Paper.docx

# Before We Virtualize…

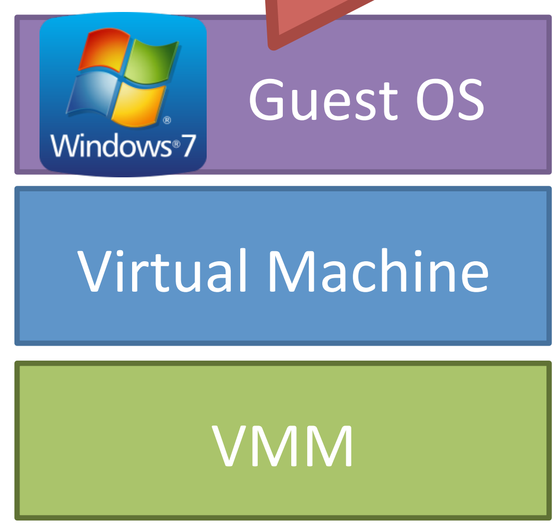
- The VMM is an application
- Like any app, it runs on top of a host OS
- VMMs exist for most OSes
  - VMWare works on Windows and Linux
  - Parallels on OS X
  - Hyper-V on Windows
- Some lightweight OSes are designed to run VMMs
  - VMWare ESX



# Booting a Guest

- When an OS boots, it expects to do so on physical hardware
- To boot a guest, the VMM provides virtual hardware
  - A fake BIOS
  - CPU: typically matches the underlying CPU (e.g. x86 on x86)
  - RAM: subset of physical RAM
  - Disks: map to subsets of the physical disk(s)
  - Network, etc…
- Guest OS is totally isolated
  - Executes in userland (ring 3)
  - Memory is contained in an x86 segment

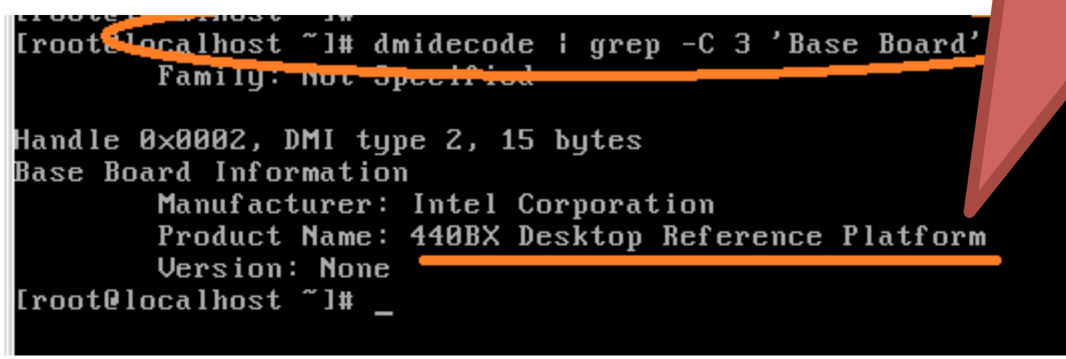
- Guest boots exactly like any other OS
- Starts at the MBR, looks for the bootloader, etc…



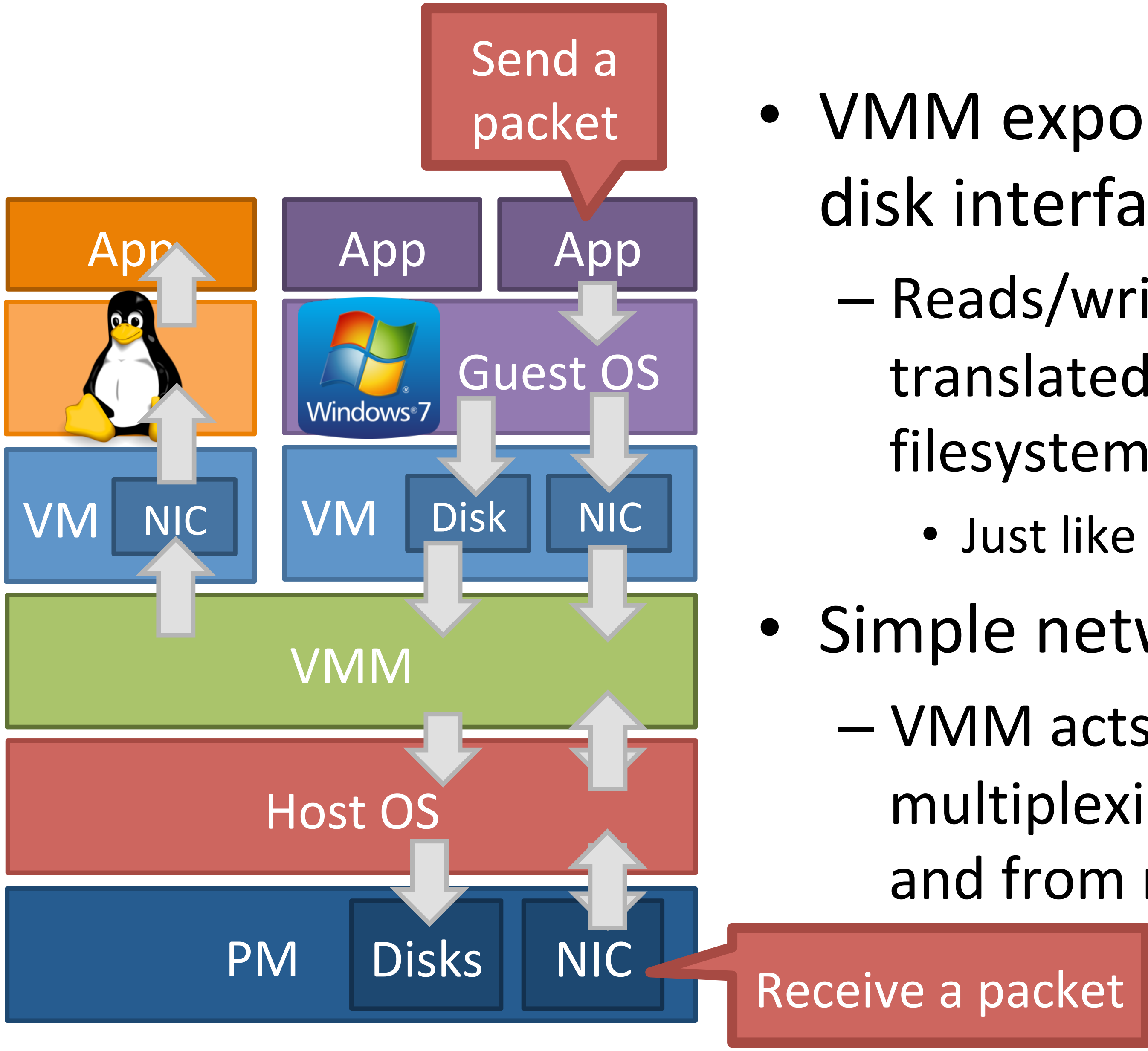
# Virtual Machine Hardware

- VMMs try to emulate hardware that is:
  - Simple
    - Emulating advan
  - Widely supporte
    - Guests should al hardware
- Example: VMWare virtual motherbo ways an Intel 440BX reference board

- This motherboard was released in 1998
- Widely supported by many OSes
- All VMWare guests run on this virtual hardware to this day



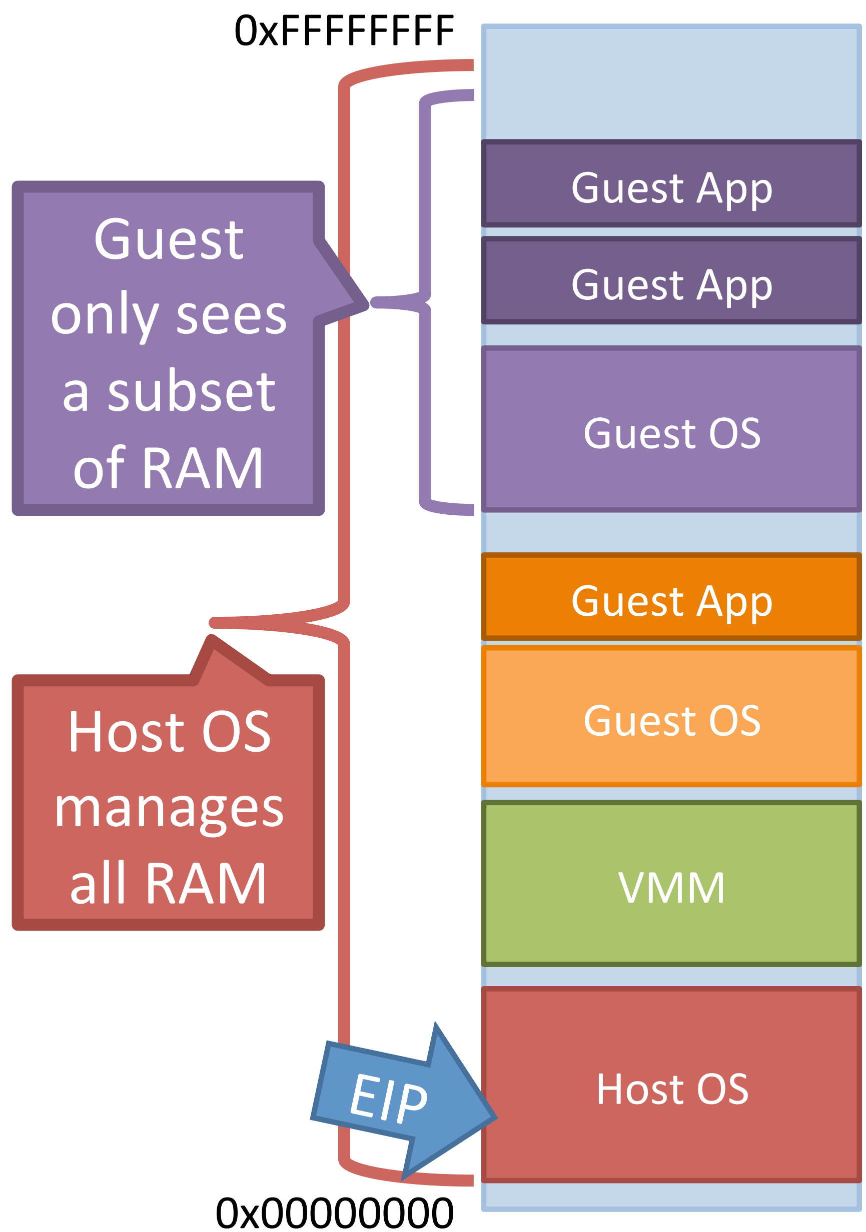
# Virtual Hardware Examples




- VMM exports a simple disk interface
  - Reads/writes are translated to a virtual filesystem on the real disk
    - Just like Pintos on QEMU
- Simple network interface
  - VMM acts like a NAT, multiplexing packets to and from multiple guests

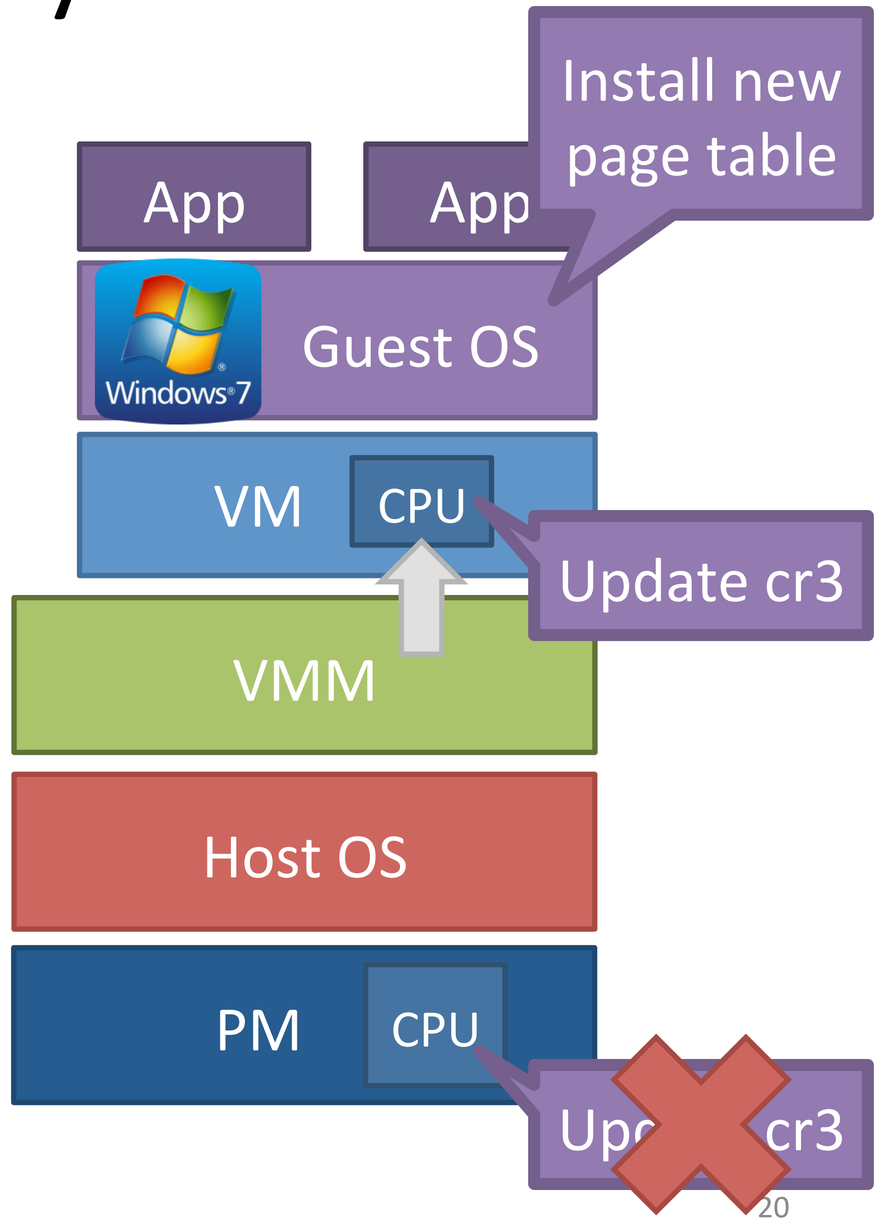
# Sharing CPU and RAM

- VMM allocates subsets of RAM for guests
  - Each guest's memory is contained in an x86 segment
  - Segments enforce strong isolation
- VMM divides CPU time between guests
  - Timer interrupts jump to the host OS
  - VMM schedules time for each guest
  - Guests are free to schedule apps as they
- In a multicore system, each guest may be assigned 1 or more CPUs



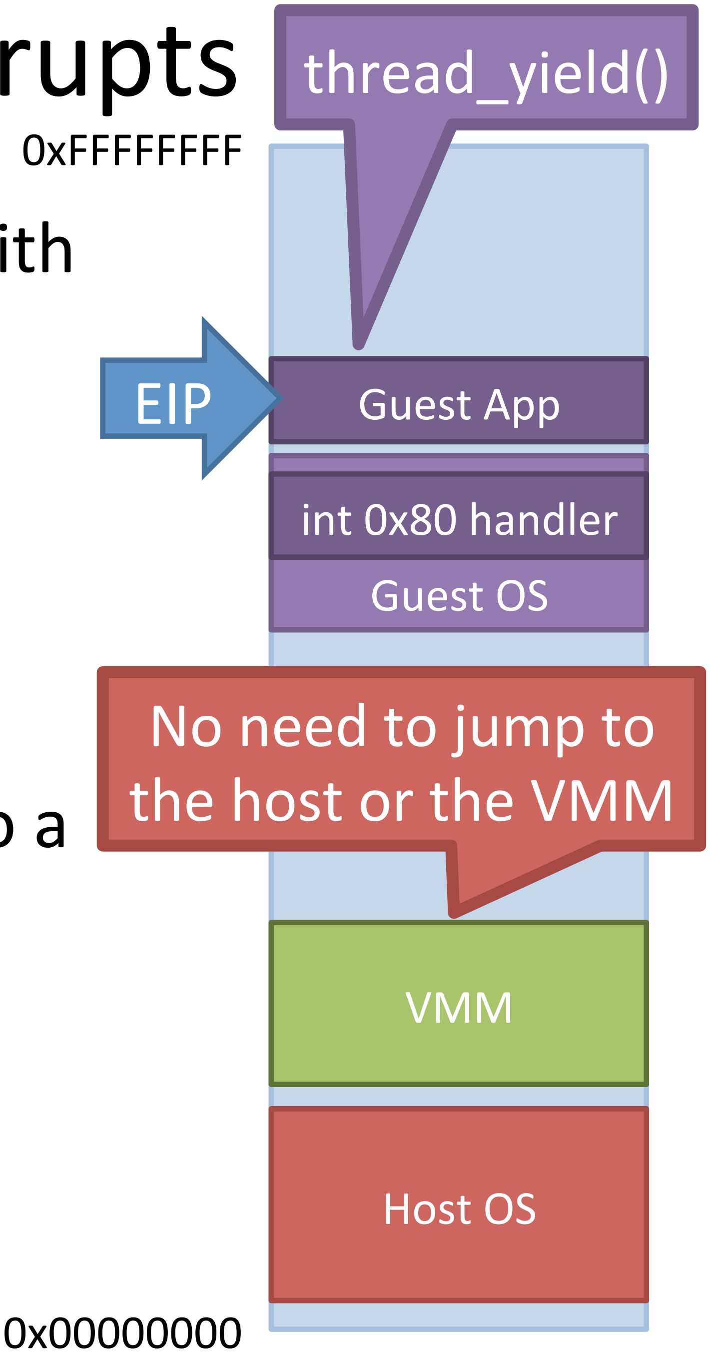
# Virtual and Physical CPU

- Each guest has a virtual CPU created by the VMM
- However, the virtual CPU is only used to store state
  - E.g. if a guest updates *cr3* or *eflags*, the new value is stored in the virtual CPU
- Guest code executes on the physical CPU
  - Keeps guest performance high
  - Guests run in userland, so security is maintained



# Handling Interrupts

- Every OS installs handlers to deal with interrupts
  - incoming I/O, timer, system call traps
- When a guest boots, the VMM records the addresses of guest handlers
- When the VMM context switches to a guest, some of its handlers are installed in the physical CPU
  - Host traps are reinstalled when the guest loses context

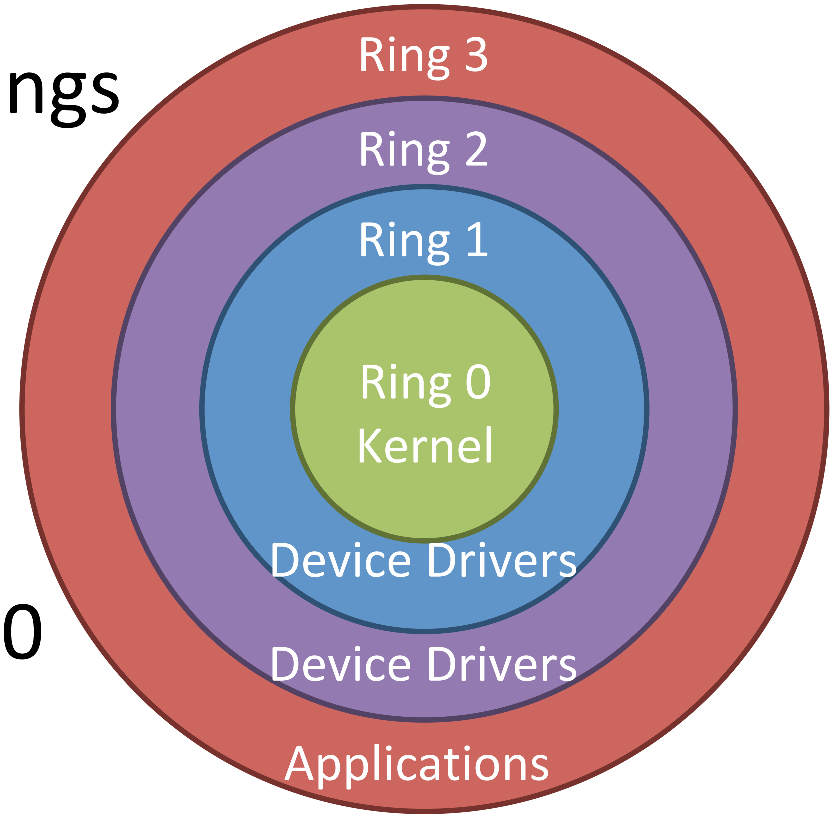


# Challenges With Virtual Hardware

1. Dealing with privileged instructions
   – OSes expect to run with high privilege (ring 0)
   – How can the VMM enable guest OSes to run in userland (ring 3)?
2. Managing virtual memory
   – OSes expect to manage their own page tables
   – This requires modifying *cr3* (high privilege) as well as updated page tables in RAM
   – How can the VMM translate between a guest's page tables and the hosts page tables?

# Protected Mode

- Most modern CPUs support protected mode
- x86 CPUs support three rings with different privileges
  - Ring 0: OS kernel
  - Ring 1, 2: device drivers
  - Ring 3: userland
- Most OSes only use rings 0 and 3



# Privileged Instructions

- OSes rely on many privileges of ring 0
  - cri, sti, popf – Enable/disable interrupts
  - hlt – Halt the CPU until the next interrupt
  - mov cr3, 0x00FA546C – install a page table
  - Install interrupt and trap handlers
  - Etc…
- However, guests run in userland (ring 3)
- VMM must somehow virtualize privileged operations

# Using Exceptions for Virtualization

- Ideally, when a guest executes a privileged instruction in ring 3, the CPU should generate an exception
- Example: suppose the guest executes hlt
  1. The CPU generates a protection exception
  2. The exception gets passed to the VMM
  3. The VMM can emulate the privileged instruction
     - If guest 1 runs hlt, then it wants to go to sleep
     - VMM can do guest1.yield(), then schedule guest 2

# Problem: x86 Doesn't Except Properly

- On x86, interrupts can be enabled/disabled by setting bit 9 of the *eflags* register
- popf pops the top value off the stack and writes it into *eflags*
- Problem: the behavior of popf varies based on privilege
  - In ring 0, all bits of *eflags* are overwritten
  - In ring 3, all bits are overwritten except bit 9
- If a guest OS uses popf to alter bit 9, then:
  1. The update will fail, and the guest's state will be inconsistent (the guest OS may crash)
  2. No CPU exception is generated, so the VMM has no idea that the guest tried to enable/disable interrupts

# Binary Translation

- x86 assembly cannot be virtualized because some privileged instructions don’t generate exceptions
- Workaround: translate the unsafe assembly from the guest to safe assembly
  - Known as binary translation
  - Performed by the VMM
  - Privileged instructions are changed to function calls to code in VMM

# Binary Translation Example

| Guest OS Assembly | Translated Assembly |
|---|---|
| do_atomic_operation: | do_atomic_operation: |
| cli | call [vmm_disable_interrupts] |
| mov eax, 1 | mov eax, 1 |
| xchg eax, [lock_addr] | xchg eax, [lock_addr] |
| test eax, eax | test eax, eax |
| jnz spinlock | jnz spinlock |
| … | … |
| … | … |
| mov [lock_addr], 0 | mov [lock_addr], 0 |
| sti | call [vmm_enable_interrupts] |
| ret | ret |

# Pros and Cons

- Advantages of binary translation
  - It makes it safe to virtualize x86 assembly code
  - Translation occurs dynamically, on demand
    - No need to translate the entire guest OS
  - App code running in the guest does not need to be translated
- Disadvantages
  - Translation is slow
  - Wastes memory (duplicate copies of code in memory)
  - Translation may cause code to be expanded or shortened
    - Thus, jmp and call addresses may also need to be patched

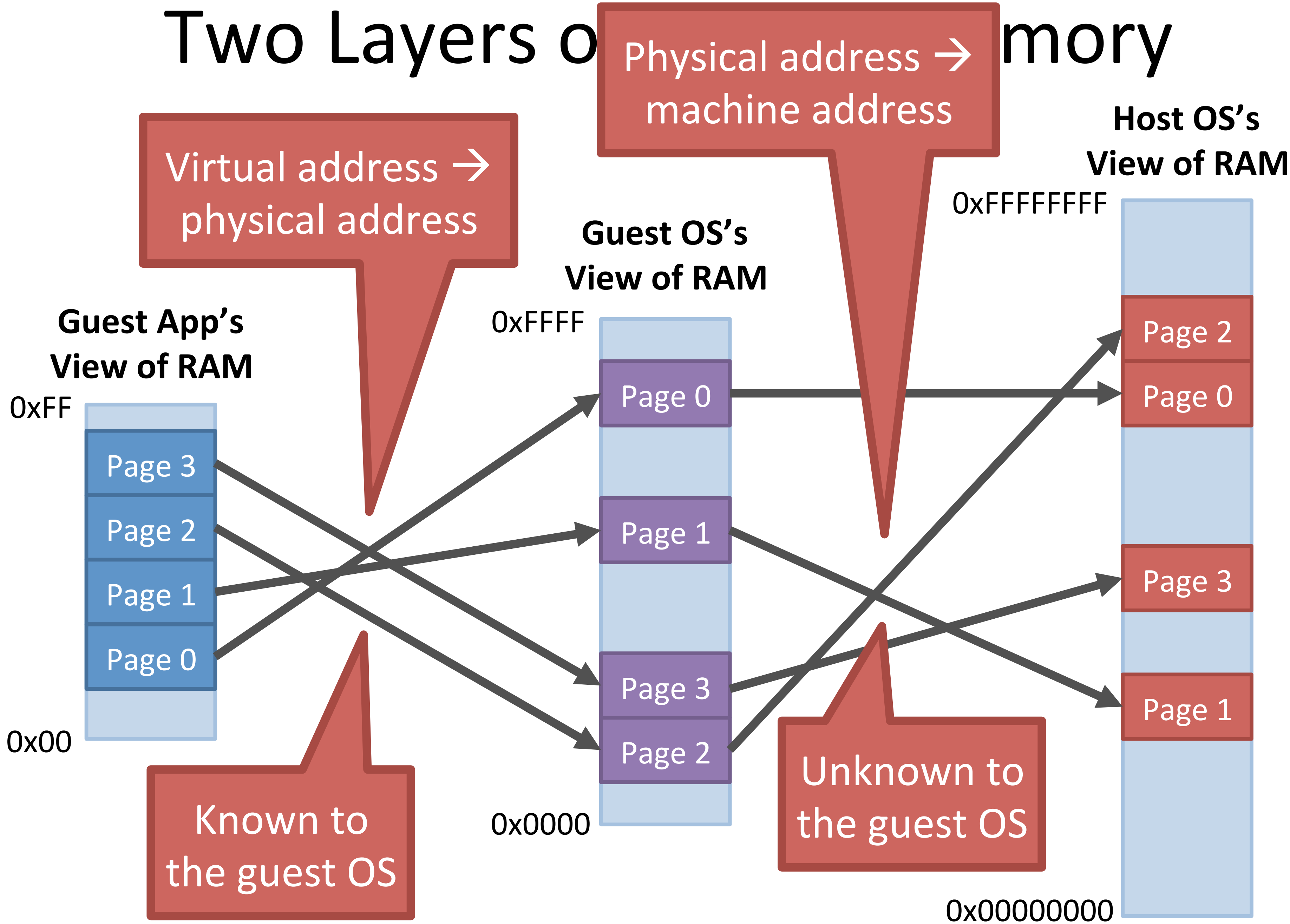
# Caching Translated Code

- Typically, VMMs maintain a cache of translated code blocks
  - LRU replacement
- Thus, frequently used code will only be translated once
  - The first execution of this code will be slow
  - Other invocations occur at native speed

# Problem: How to Virtualize the MMU?

- On x86, each OS expects that it can create page tables and install them in the *cr3* register
  - The OS believes that it can access physical memory
- However, virtualized guests do not have access to physical memory
- Using binary translation, the VMM can replace writes to *cr3*
  - Store the guest's root page in the virtual CPU *cr3*
  - The VMM can now walk to guest's page tables
- However, the guest's page tables cannot be installed in the physical CPU…

# Two Layers o mory

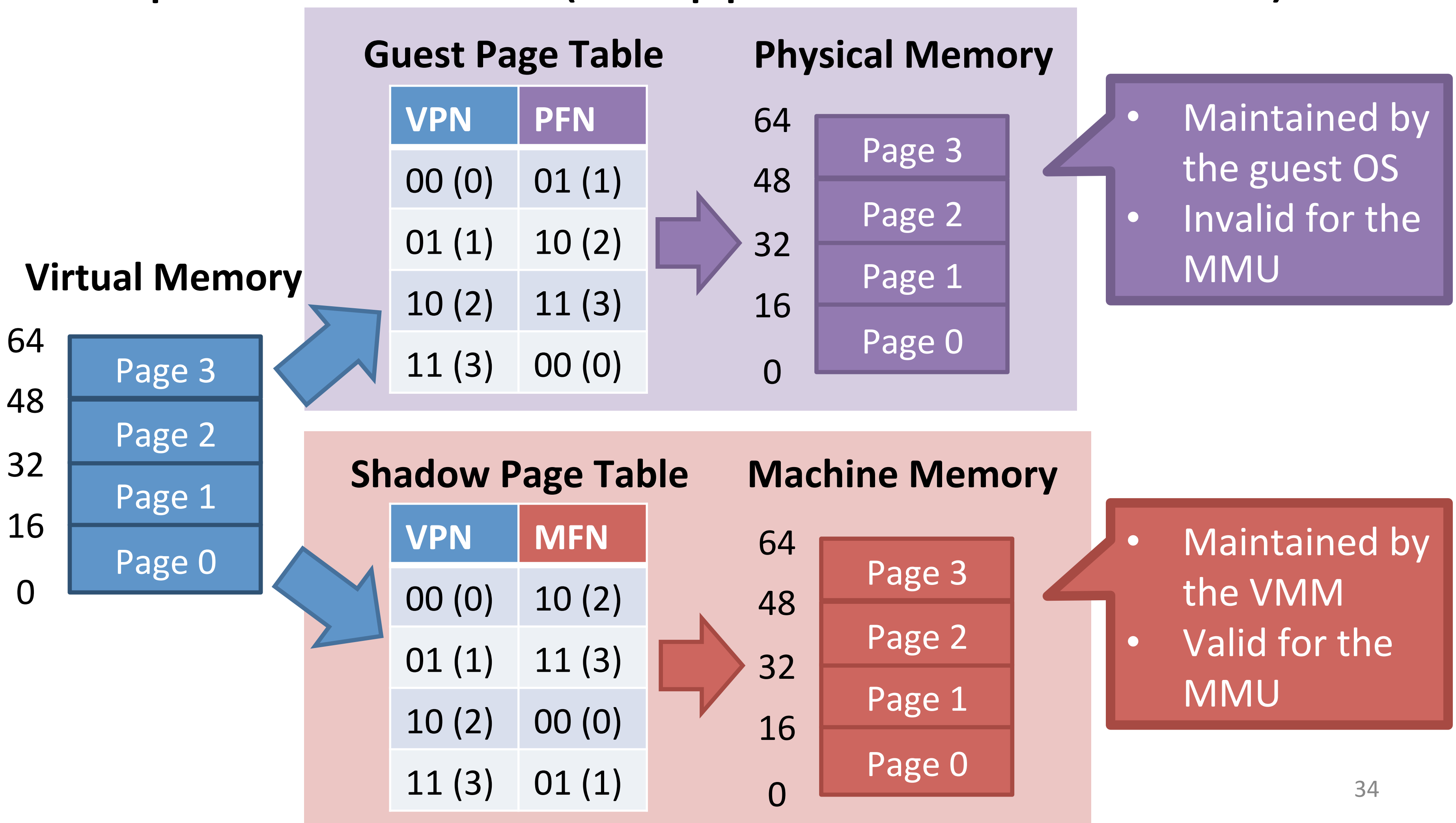


# Guest's Page Tables Are Invalid

- Guest OS page tables map virtual page numbers (VPNs) to physical frame numbers (PFNs)
- Problem: the guest is virtualized, doesn't actually know the true PFNs
  - The true location is the machine frame number (MFN)
  - MFNs are known to the VMM and the host OS
- Guest page tables cannot be installed in *cr3*
  - Map VPNs to PFNs, but the PFNs are incorrect
- How can the MMU translate addresses used by the guest (VPNs) to MFNs?

# Shadow Page Tables

- Solution: VMM creates shadow page tables that map VPN → MFN (as opposed to VPN→PFN)




**Guest Page Table**

| VPN | PFN |
|---|---|
| 00 (0) | 01 (1) |
| 01 (1) | 10 (2) |
| 10 (2) | 11 (3) |
| 11 (3) | 00 (0) |

**Shadow Page Table**

| VPN | MFN |
|---|---|
| 00 (0) | 10 (2) |
| 01 (1) | 11 (3) |
| 10 (2) | 00 (0) |
| 11 (3) | 01 (1) |

# Building Shadow Tables

- Problem: how can the VMM maintain consistent shadow pages tables?
  - The guest OS may modify its page tables at any time
  - Modifying the tables is a simple memory write, not a privileged instruction
    - Thus, no helpful CPU exceptions :(
- Solution: mark the hardware pages containing the guest's tables as read-only
  - If the guest updates a table, an exception is generated
  - VMM catches the exception, examines the faulting

# Dealing With Page Faults

- It is possible that the shadow table may be inconsistent
- If a guest page faults, this could be a:
  - **True miss**: actual page fault, guest OS/app should crash
  - **Hidden miss**: the shadow table is inconsistent; there is a valid VPN→PFN mapping in the guest's page tables
- VMM must disambiguate true and hidden misses
  - On each page fault, the VMM must walk the guest's tables to see if a valid VPN→PFN mapping exists
  - If so, this is a hidden miss
    - Update the shadow table and retry the instruction
  - Otherwise, forward the page fault to the guest OS's handler

Tracing

# Pros and Cons

- The good: shadow tables allow the MMU to directly translate guest VPNs to hardware pages
  - Thus, guest OS code and guest apps can execute directly on the CPU
- The bad:
  - Double the amount of memory used for page tables
    - i.e. the guest's tables and the shadow tables
  - Context switch from the guest to the VMM every time a page table is created or updated
    - Very high CPU overhead for memory intensive workloads

# More VMM Tricks

- The VMM can play tricks with virtual memory just like an OS can
- Paging:
  - The VMM can page parts of a guest, or even an entire guest, to disk
  - A guest can be written to disk and brought back online on a different machine!
- Shared pages:
  - The VMM can share read-only pages between guests
  - Example: two guests both running Windows XP

- Full Virtualization (VMWare)
- Hardware Support
- Paravirtualization (Xen)

# The Story So Far…

- We have discussed how systems like VMWare implement full virtualization
- Key challenges solved by VMWare:
  - Binary translation rewrites guest OS assembly to not use privileged instructions
  - Shadow page tables maintained by the VMM allow the MMU to translate addresses for guest code
- So what’s the problem?
  - **Performance**

# Virtualization Performance

- Guest code executes on the physical CPU
- However, that doesn't mean its as fast as the host OS or native applications

1. Guest code must be binary translated
2. Shadow page tables must be maintained
   - Page table updates cause expensive context switches from guest to VMM
   - Page faults are at least twice as costly to handle

# Hardware Techniques

- Modern x86 chips support hardware extensions designed to improve virtualization performance

1. Reliable exceptions during privileged instructions
   - Known as AMD-V and VT-x (Intel)
   - Released in 2006
   - Adds vmrun/vmexit instructions (like sysenter/sysret)
2. Extended page tables for guests
   - Known as RVI (AMD) and EPT (Intel)
   - Adds another layer onto existing page table to map PFN→MFN

# AMD-V and VT-x

- Annoyingly, AMD and Intel offer different implementations
- However, both offer similar functionality
- vmenter: instruction used by the hypervisor to context switch into a guest
  – Downgrade CPU privilege to ring 3
- vmexit: exception thrown by the CPU if the guest executes a privileged instruction
  – Saves the running state of the guest's CPU
  – Context switches back to the VMM

# Configuring vmenter/vmexit

- The VMM tells the CPU what actions should trigger vmexit using a VM Control Block (VMCB)
  - VMCB is a structure defined by the x86 hardware
  - Fields in the struct tell the CPU what events to trap
  - Examples: page fault, TLB flush, mov cr3, I/O instructions, access of memory mapped devices, etc.
- The CPU saves the state of the guest to the VMCB before vmexit
  - Example: suppose the guest exits due to device I/O
  - The port, data width, and direction (in/out) of the operation get stored in the VMCB

# Benefits of AMD-V and VT-x

- Greatly simplifies VMM implementation
  - No need for binary translation
  - Simplifies implementation of shadow page tables
- Warning: the VMM runs in userland, but use of AMD-V and VT-x requires ring 0 access
  - Host OS must offer APIs that allow VMMs to configure VMCB and setup callbacks for guest OS exceptions
  - Example: KVM on Linux

# Problem with AMD-V and VT-x

- Some operations are **much** slower when using vmexit vs. binary translation

| Guest OS Assembly | Translated Assembly |
|---|---|
| do_atomic_operation: | do_atomic_operation: |
| cli | call [vmm_disable_interrupts] |
| mov eax, 1 | mov eax, 1 |
| | … |

- This code is okay because cli is trapped by vmexit
- However, each vmexit causes an expensive context switch

- The VMM must generate this code via binary translation
- But, this direct call is very fast, no context switch needed

# Benefits of AMD-V and VT-x

- Greatly simplifies VMM implementation
  - No need for binary translation
  - Simplifies implementation of shadow page tables
- … however, sophisticated VMMs still use binary translation in addition to vmenter/vmexit
  - VMM observes guest code that causes frequent vmexits
  - Hot spots may be binary translated or dynamically patched to improve performance
  - Similar to Just-In-Time (JIT) compilation

# Second Level Address Translation

- AMD-V and VT-x help the VMM control guests
- … but, they don't address the need for shadow page tables
- Second level address translation (SLAT) allows the MMU to directly support guest page tables
  - Intel: Extended Page Tables (EPT)
  - AMD: Rapid Virtualization Indexing (RVI)
  - Also known as Two Dimensional Paging (TDP)
  - Introduced in 2008

# SLAT Implementation

- VMM installs first and second level tables in the MMU
  - Context switch to the guest via vmenter
- Steps to translate an address:
  1. MMU queries the level 1 (guest) table
  2. MMU queries the level 2 (VMM) table
- If any step yields an invalid PTE than page fault to the VMM (vmexit)

**Guest Page Table**

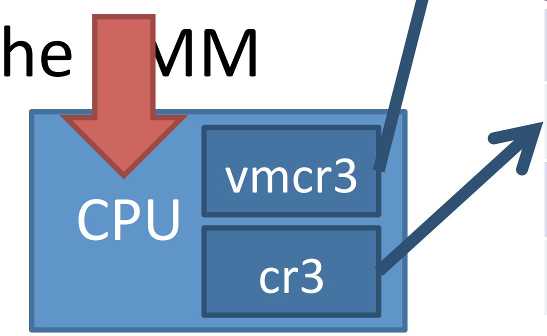
| VPN | PFN |
| --- | --- |
| 00 (0) | 01 (1) |
| 01 (1) | 10 (2) |
| 10 (2) | 11 (3) |
| 11 (3) | 00 (0) |

Maintained by the guest OS

**Extended Page Table**

| PFN | MFN |
| --- | --- |
| 01 (1) | 10 (2) |
| 10 (2) | 11 (3) |
| 11 (3) | 00 (0) |
| 00 (0) | 01 (1) |

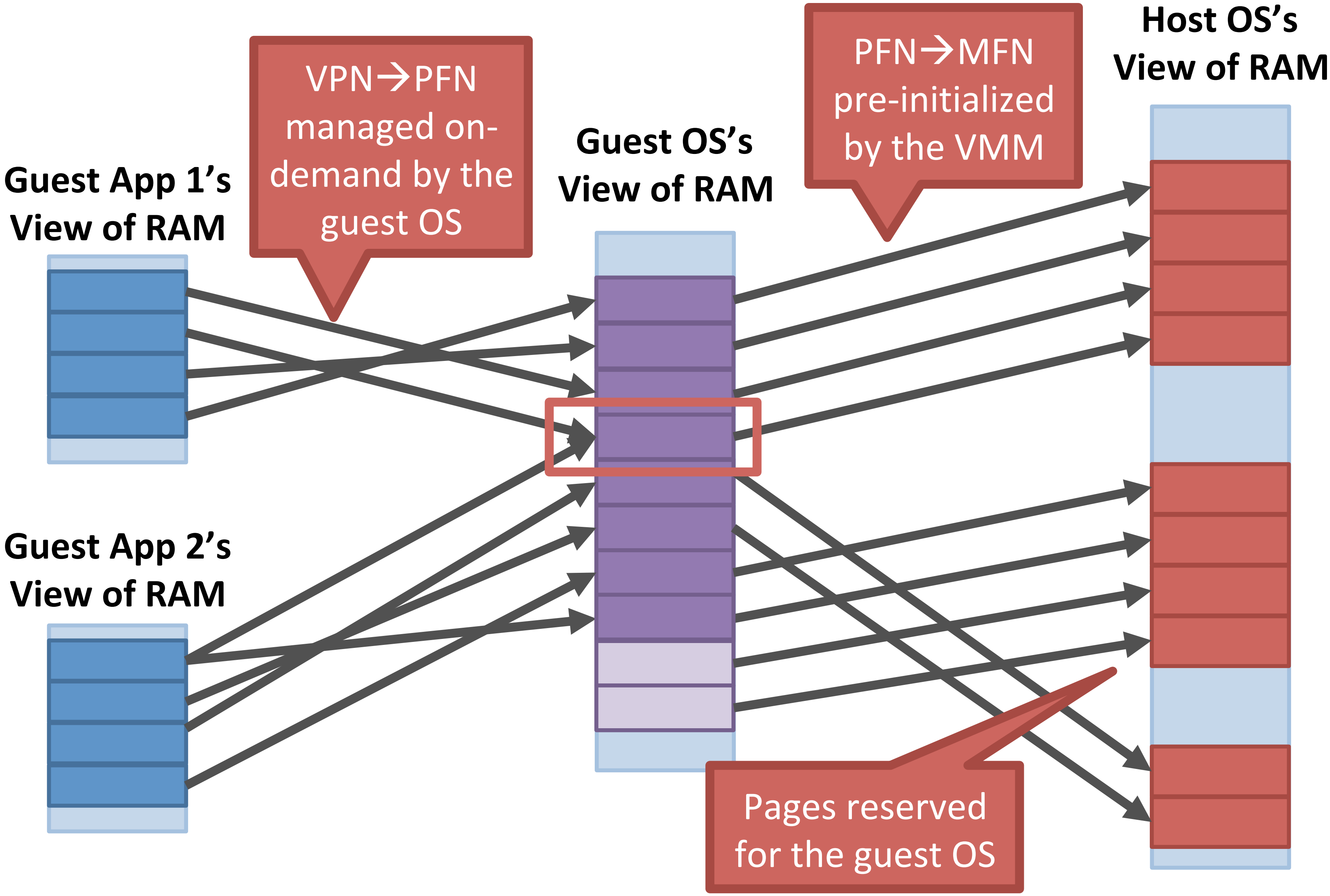
Maintained by the VMM



# Advantages of SLAT

- **Huge** performance advantages vs. shadow page tables
- When guests mov cr3, the CPU updates *vmcr3* register
  - No need to vmexit when guest OS switches context
- EPT can be filled on-demand or pre-initialized with PFN→MFN entries
  - On-demand:
    - Slower, since many address translations will trigger hidden misses
    - … but hardware pages for the guest can be allocated when needed
    - And, the EPT will be smaller
  - Preallocation:
    - No need to vmexit when the guest OS creates or modifies it's page tables
    - … but hardware pages need to be reserved for the guest
    - And, the EPT table will be larger

# Pre-initialized EPT

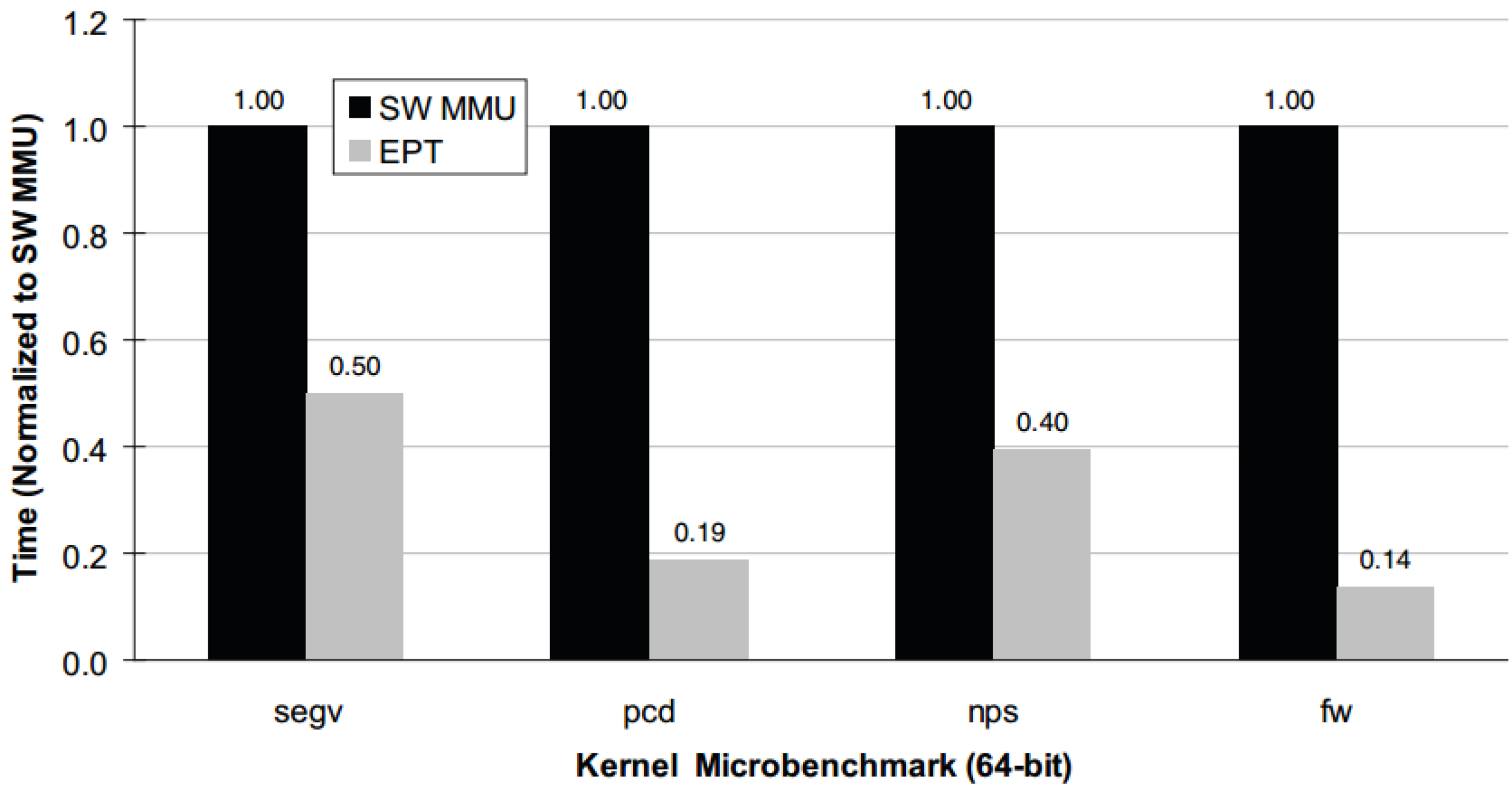


# Disadvantages of SLAT

- Memory overhead for EPT
  - … but not as much as shadow page tables
- TLB misses are twice as costly
  - SLAT makes page tables twice as deep, hence it takes twice as long to resolve PTEs

# EPT Performance Evaluation

- Microbenchmarks by the VMWare team
- Normalized to shadow page table speeds (1.0)
  - Lower times are better



# Configuring Your VMM

- Advanced VMMs like VMWare give you three options
  1. Binary translation + shadow page tables
  2. AMD-V/VT-x + shadow page tables
  3. AMD-V/VT-x + RVI/EPT

- Fastest by far
- But, requires very recent, expensive CPUs

- Which is best?
  – Choosing between 1 and 2 is more difficult
  – For some workloads, 2 is much slower than 1
  – Run benchmarks with your workload before decided on 1 or 2

- Full Virtualization (VMWare)
- Hardware Support
- Paravirtualization (Xen)

# The Story so Far…

- We have discussed full virtualization by looking at the implementation of VMWare
- We have discussed how recent advances in x86 hardware can speed up virtualization
- Thus far, we have abided by virtualization rule #1:
  - **Fidelity**: software on the VMM executes identically to its execution on hardware
- What if we relax this assumption?

# Relaxing Assumptions

- Problem: it takes a lot of work to virtualize an arbitrary guest OS
  - VMM implementation is very complicated
  - Even with hardware support, performance issues remain
- What if we require that guests be modified to run in the VMM
  - How much work is it to modify guests to “cooperate” with the VMM?
  - Will VMM implementation be simpler?
  - Can we get improved performance?

# Paravirtualization

- Denali and Xen pioneered the idea of paravirtualization
  - Require that guests be modified to run on the VMM
  - Replace privileged operations with hypercalls to the hypervisor
  - Defer most memory management to the VMM
- Our discussion will focus on Xen
  - Commercial product owned by Citrix (i.e. GoToMeeting)
  - Robust, mature hypervisor

# Hypercalls

- The Xen VMM exports a hypercall API
  - Methods replace privileged instructions offered by the hardware
    - E.g halt CPU, enable/disable interrupts, install page table
  - Guest OS can detect if it’s running directly on hardware or on Xen
    - In the former case, typical ring 0 behavior is used
    - In the latter case, hypercalls are used
- If a guest executes a privileged instruction, crash it
  - Xen VMM makes no attempt to emulate privileged instructions
  - Simplifies Xen VMM implementation

# Handling Interrupts and Exceptions

- Guests register callbacks with the Xen VMM to receive interrupts and exceptions
  - Timer interrupts
  - Page faults
  - I/O interrupts
- Xen buffers many events and passes them to the guest in batches
  - Improves performance by reducing the number of VMM→guest context switches
- In some cases, interrupts are forwarded directly to the guest without Xen's intervention
  - Example: int 0x80 system calls

# Managing Virtual Memory

- All guest memory is managed by Xen
  - Guests allocate empty page tables, registers them with Xen via a hypercall
  - Guest may read but not write page tables
  - All updates to pages must be made via hypercalls
- Advantages:
  - No extra memory needed for extended page tables
  - No need to implement shadow page tables
  - No additional overhead for TLB misses
  - No hidden misses
- Disadvantages:
  - Each updates to page tables cause a guest→VMM context switch

# Virtual Time in Xen

- Keeping track of time is hard in the guest
  - Guest cannot observe CPU ticks directly
  - VMM may context switch a guest out for an arbitrary amount of time
- Xen provides multiple times to guests
  - Real time: ticks since bootup
  - Virtual time: ticks during which the guest is active
  - Wall clock time, adjusted by timezone
- Why are real time and wall clock time separate?
  - The host OS may change the time (e.g. daylight savings)
  - Changing the clock can cause weird anomalies

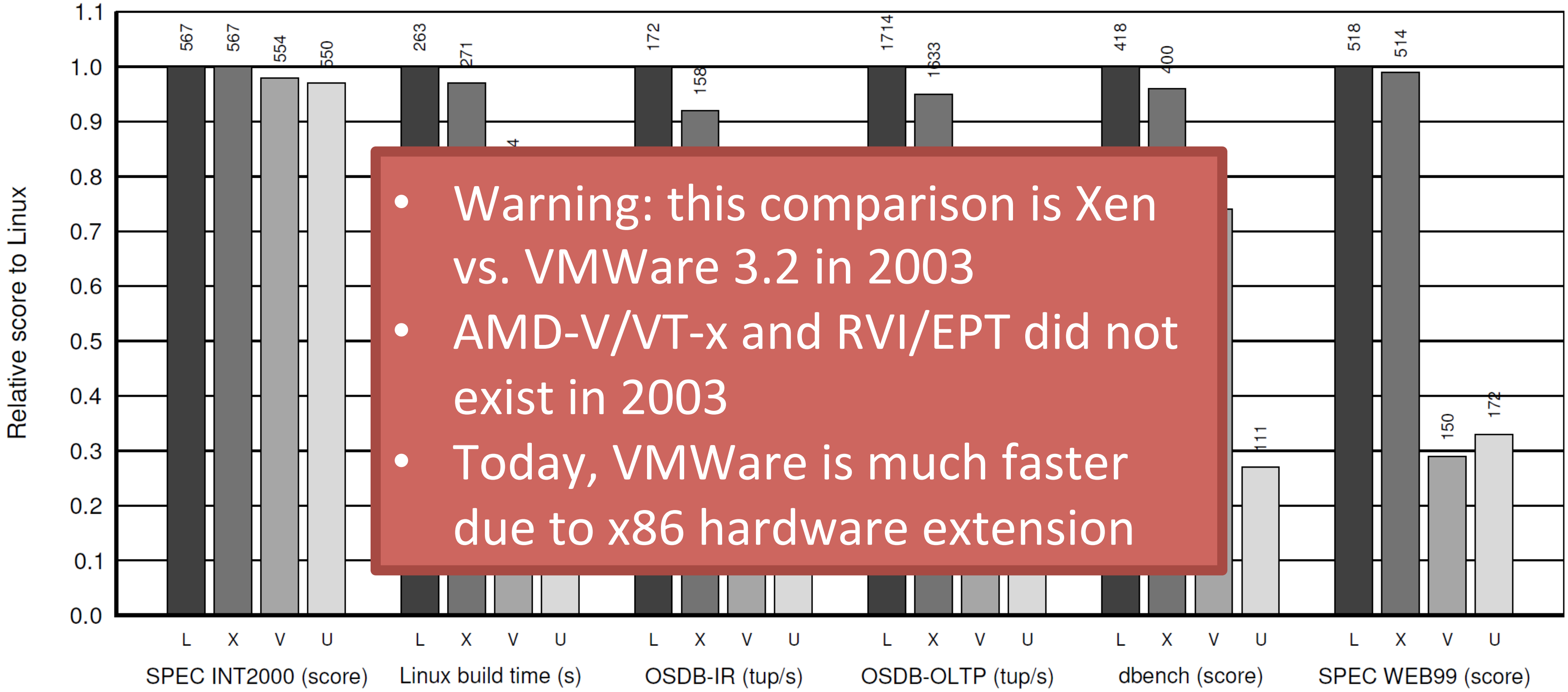
# Virtual Devices in Xen

- Xen exports simple, idealized virtual devices to guests
  - Guest needs to be modified to include drivers for these devices
  - Thankfully, the drivers are simply to write
- This is essentially the same approach used by other hypervisors (VMWare, etc.)

# Modifying Guests

- How much work does it take to modify a guest OS to run on Xen?
  - Linux: 3000 lines (1.36% of the kernel)
    - Including device drivers
  - Windows XP: 4620 lines (0.04% of the kernel)
    - Device drivers add another few hundred lines of code
- Modification isn’t trivial, but its certainly doable

# Xen Performance




- Warning: this comparison is Xen vs. VMWare 3.2 in 2003
- AMD-V/VT-x and RVI/EPT did not exist in 2003
- Today, VMWare is much faster due to x86 hardware extension

- Relative performance of native Linux (L), Linux on Xen (X), Linux on VMWare 3.2 (V), and User-Mode Linux (U)
- Normalized to native Linux

# Wrap-Up

- Virtualization has made a huge resurgence in the last 15 years
- Today, all OSes and most CPUs have direct support for hosting virtual machines, or becoming virtualized
- **Virtualization underpins the cloud**
  - E.g. Amazon EC2 rents virtual machines at low costs
  - Hugely important for innovation

# Bibliography


- Software and Hardware Techniques for x86 Virtualization
  - http://www.vmware.com/files/pdf/software_hardware_tech_x86_virt.pdf
- A Comparison of Software and Hardware Techniques for x86 Virtualization
  - http://dl.acm.org/citation.cfm?id=1168860
- Performance Evaluation of Intel EPT Hardware Assist
  - http://www.vmware.com/pdf/Perf_ESX_Intel-EPT-eval.pdf
- Xen and the Art of Virtualization
  - http://dl.acm.org/citation.cfm?id=945462