Virtualization

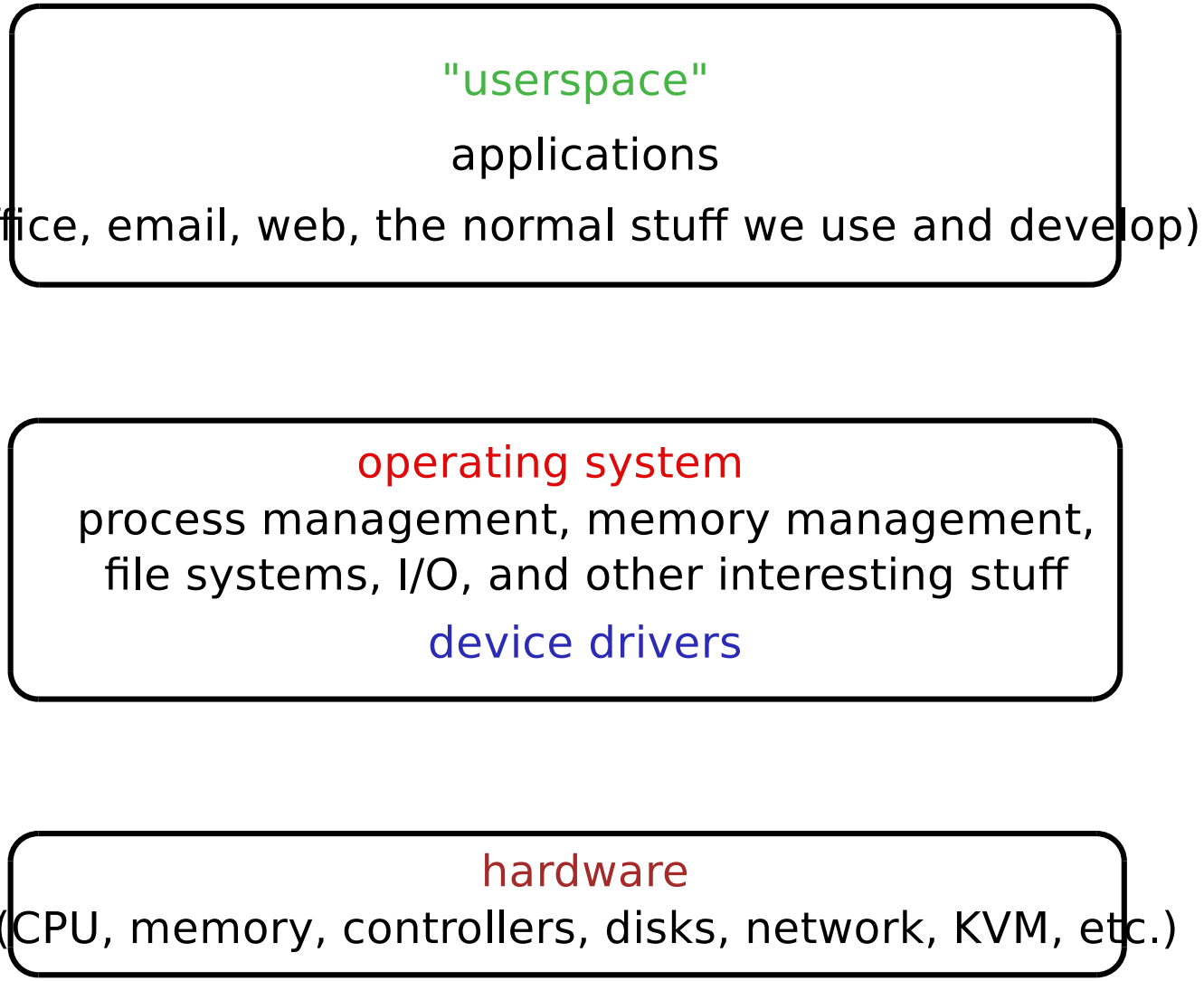
Operating Systems

Oleg Goldshmidt

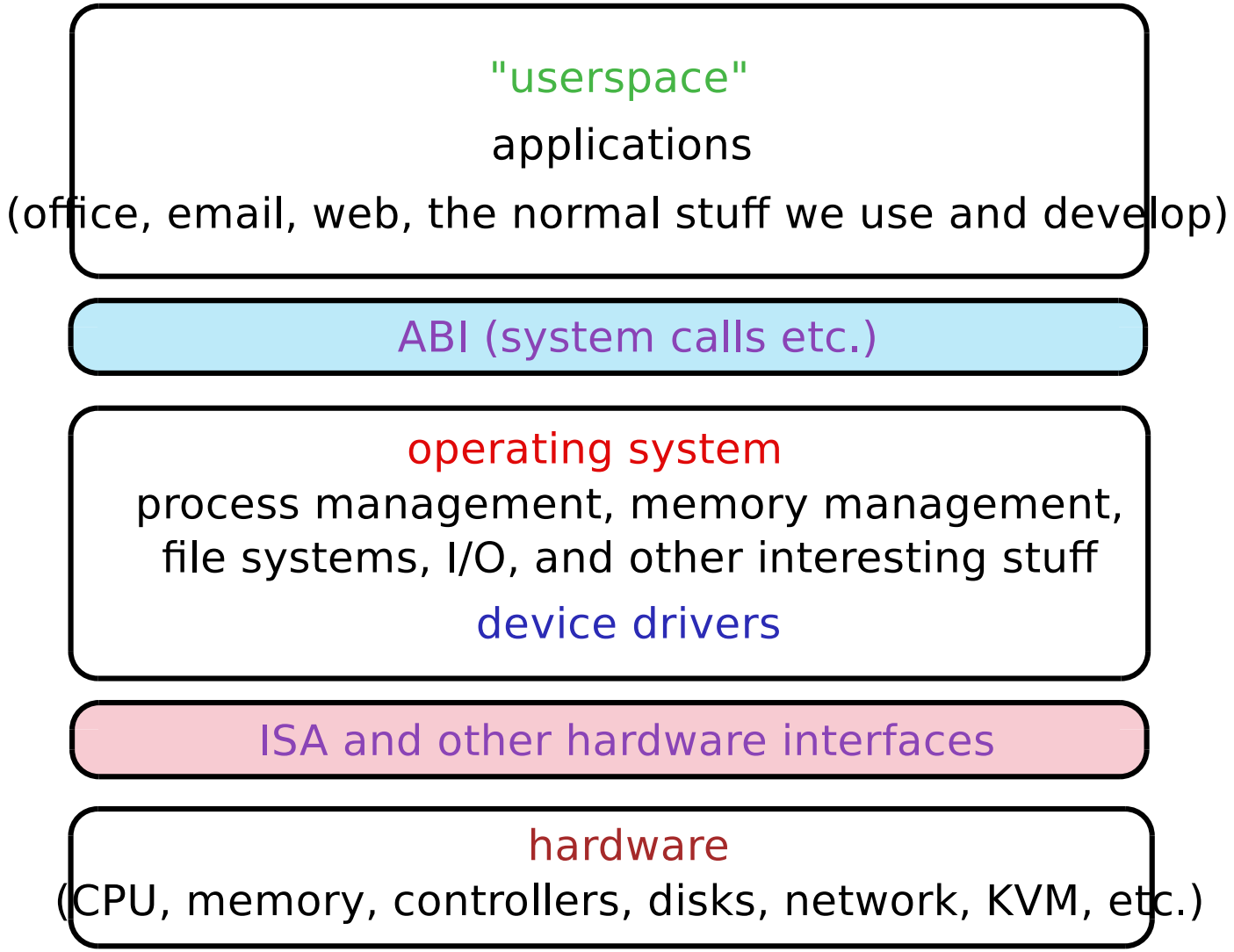
ogoldshmidt@computer.org

Lecture 12

Computer System Architecture



Computer System Interfaces



Process Virtual Machines

- "process" — an execution unit
- what is a "machine" from the point of view of a process?
  - logical memory address space ("virtual memory")
  - user-level (unpriviledged) instructions and registers
  - all the rest (priviledged instructions, I/O) is done through the OS
- the "machine" is a combination of the OS and the underlying hardware
- the process interacts with the "machine" via the ABI
- the process does not see the machine the way the OS does
- even the time does not fly in the same manner!

System Virtual Machine

- “system” — an execution environment that can support multiple process that can belong to different users
- the processes share the physical resources (CPU, memory, I/O devices, etc.)
- the system allocates and schedules the resources
- from the system’s point of view the “machine” is hardware
- the system interacts with the “machine” through ISA

Generic Virtual Machine

- what do process and system virtual machines have in common?
  - they execute software instructions of the supported entity (process or system)
  - they present the supported entity an abstraction of the supporting “machine”
  - they map the real machine resources to virtual resources accessible by the supported entity
  - ultimately, the real hardware executes the actions specified by the virtual machine instructions
- the virtual resources are not necessarily the same as the real resources
- performance is not necessarily equivalent to “native”

Computers and Operating Systems

- the familiar picture: one computer, one OS
- multiprocessor picture: one CPU, one OS
  - recall, e.g., the master-slave configuration
- multicomputer picture: one node, one OS
- SMP picture: many CPUs, one OS
- where is: one CPU, many OSes?

Can we run more than one OS on a single CPU (or, more generally, on a single computer, e.g., several SMP OSes on an SMP machine) simultaneously? And why would we want to do it?

Server Consolidation

- IT needs of enterprizes grow
  - more computing power
  - more new servers with different roles, disparate operating environments
  - very high acquisition, provisioning, maintenance costs
- servers are provisioned on a “just in case” basis, thus underutilized
- workloads change
- granularity of resource control is low

Server Consolidation II

- example: IT infrastructure servers (file servers, print servers, directory infrastructure servers, firewalls, NAT, DNS, DHCP, etc.)
  - typical utilization: $10 \div 15\%$
  - architectural, security, compatibility issues lead to deployment on separate machines
    - hardware expenses
    - operational expenses
    - space, facilities, power, cooling, etc.
- example: mission critical servers
  - typical utilization: $30 \div 40\%$
  - varying workload: today need more web servers here, tomorrow need more DB servers there, etc.
  - maintenance, high availability, recovery time

Desktop Consolidation

- typical desktop utilization: 5%
- today: investment bank branch
  - 200 users with different requirements
  - IT operation, personnel, etc to maintain 200 desktops + spares
  - “I need more memory, faster CPU, bigger disk”
    - temporarily
- tomorrow
  - users only have peripheral equipment (KVM)
  - one physical platform runs 4 different operating environments
  - 50 consolidated machines + 5 spares
  - new environment is created as need arises

Capacity Optimization

- treat the available hardware as a single resource pool
- flexible capacity planning
- provisioning to optimize resource utilization
- “just in time” provisioning
- moving functionality around the resource pool
- dealing with dynamic workloads

Testing, Upgrading, Debugging

- test in multiple operating environments without the need for separate physical platforms
- e.g., test a new version of software (including OS kernels) before upgrading
- live update: shut down an old version and bring up a new one while the rest of the environment keeps working
  - bring up a new version alongside the old one and switch!
- rollback: shut down the new version and restore the old one if needed
- debugging does not affect the rest of the system

Research and Development

- what happens if there is a serious bug in your brand new OS?
  - your computer freezes, catches fire, or becomes generally unusable
- how do you debug a running OS?
  - you do it from a different computer connected to the target via, e.g., a serial cable
- wouldn't it be great if
  - you could develop, test, debug a new OS using only one computer?
  - a bug would not render your computer unusable?
  - you had control over the OS behaviour from the outside?

Security

- a VM provides an isolated environment that can be destroyed without permanent damage to the system
- single-application VM: e.g., run IE in a virtual machine
  - a VM infected with a virus, or 0WNed, can be brought down and restarted afresh without affecting the rest of the system
  - needs appropriate protection for permanent storage
- multiple secure environments
  - how to allow multiple users share a Windows system simultaneously
  - Citrix — awkward solution, the underlying system is not designed for multiuser operation
  - let each user have a virtual machine

Using The Best Tools

- I want more then one OS on my machine!
  - simultaneously!
- my main development environment is Linux
- I may need Windows for specific applications or tasks
  - without rebooting
- terminal server solutions (e.g., Citrix) are not always available
  - need to be on the network
  - securing Citrix is a problem
- example: I teach OS, and I may want to demonstrate some things on different systems during a lecture

A Bit of History

- IBM System/360 (circa 1965)
  - extended the (novel) concept of virtual memory
  - provide the full System/360 ISA (rather than ABI)
    - perceived as better protection as system architecture evolves
  - HAL — hardware abstraction layer
- IBM System/370 (early 1970-ies)
  - VM become mainstream
- IBM zSeries (today)

So What Seems To Be The Problem?

- the problem is that OS likes to think it is the sole owner of all the system's resources
- resource management is a major OS task, and one of the main resons for OS existence
- OS manages sharing the system resources between processes
- kernel (system) space vs. user space
  - Intel CPUs have a special register
  - 2 bits — 4 protection rings
  - only 2 are used: "ring 0" for kernel, "ring 3" for userspace
  - no OS used rings 1 or 2 since the late OS/2

What Are the Options?

- cooperative virtual machine: multiple OSes share ring 0
- emulation: run one host in ring 0, other OSes in ring 3 (as applications)
  - usually slow — not native execution
- para-virtualization: add a protection level, e.g., run a "hypervisor" in ring 0, OSes ("partitions") — in ring 1, applications — in ring 3
  - good performance
  - need to modify the OS!
- binary translation (full virtualization): try to figure out what an OS does, change it on the fly
  - significant performance overhead
  - lots of headaches you would not have otherwise

Cooperative Linux (coLinux)

- a port of Linux kernel that allows it to run alongside other operating systems (e.g., Windows) on a single machine (135K patch, compile time parameter)
- runs in the driver context of another OS
- runs at the same protection level as the host kernel
- the driver gives the guest kernel a fixed amout of memory
- guest kernel has its own page tables, etc.
- there is a “passage page” shared between the host driver and the guest kernel — 4K of host and guest CPU state

Cooperative Linux II

- in the host OS, a userspace daemon ioctl()'s the driver, the driver switches to guest while preserving the full CPU state
- when the guest needs data from the host or an interrupt occurs the driver switches back
- drawbacks: stability and security
- measures can be taken to gracefully shut down on the first "oops" or panic
- the two kernels must be trusted and cooperative
- an attack on one of them leads to a compromise of the other
- more details: `http://www.colinux.org`

Emulation vs. Virtualization

- emulation: providing the functionality of the target hardware completely in software
  - you can emulate an architecture using a completely different architecture
  - tends to be slow
- virtualization: partitioning real hardware into multiple contexts, which (usually) timeshare the real hardware
  - typically faster than emulation
  - need the right type of hardware
- a lot of confusion, especially since both words are so overloaded

Hardware Emulation

- emulate the computer hardware in an application
- run an application that emulates the hardware, run an OS and an application stack inside this application
- can be used for cross-platform development
- user mode emulation
  - used to run processes compiled for one architecture on a different architecture
- full system emulation
  - used to emulate a full system used to run an OS and applications

Hardware Emulation: bochs

- emulates an x86 PC
  - 386, 486, Pentium, PentiumPro, AMD64, including MMX, SSE, etc.
  - common I/O devices: disk, floppy, CD, NIC, etc.
  - includes BIOS and VGA BIOS
- runs on UNIX, Linux, Windows, MacOS X, BeOS, OS/2, etc.
- runs Linux, DOS, Windows (95, NT), MacOS X
- a C++ application providing a faithful HW emulation
- the BIOS and OS are stored in files on host
- emulates every instruction — slow
- more details: `http://bochs.sourceforge.net/`

Hardware Emulation: QEMU

- open source CPU emulator
- full system emulation for x86, x86_64, PowerPC
- user mode emulation for Linux for x86, x86_64, PowerPC, SPARC
- Accelerator Module for PC — runs most of the target code directly on the host CPU to achieve good performance
- lots of OSes, including Linux, Windows, QNX, NetBSD, L4, Solaris, Minix, etc.
- slowdown
  - without acceleration: factor of 5 to 10
  - with acceleration: factor of 1 to 2

QEMU: Dynamic Translation

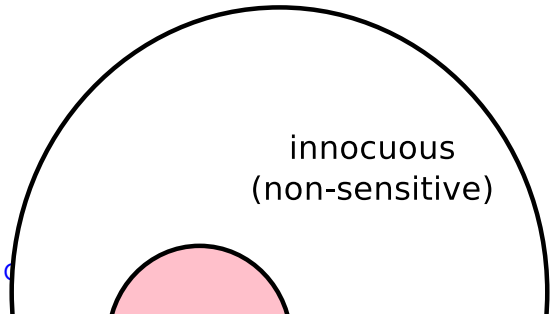
- QEMU converts target code to host instruction set
  - splits, e.g., x86 instructions into fewer simpler instructions
  - each simple instruction is implemented in C
  - a compile time tool (`dyngen`) dynamically generates code by concatenating simple instructions
- a lot of optimizations — condition code, translation cache
- MMU emulation via `mmap()`
- QEMU can even emulate itself (self-virtualization)
- more details:
  `http://fabrice.bellard.free.fr/qemu/`

CPU Virtualization

- we need a “hypervisor” (a.k.a. “virtual machine monitor”) to control the many OSes running on a single system
  - “one ring to rule them all...”
- to maintain complete control, the OS must be kept out of ring 0
  - if one of the OSes can mess with the hypervisor it will defeat the original purpose
- but OSes are used to running in ring 0 and having complete control themselves!
- the 0/1/3 model: change the OS so it plays nice
- the 0/3 model: push the OS to ring 3, but do everything else as if it were 0/1/3 (differently from emulation)

Instruction Taxonomy

- sensitive
  - attempts to change resource configuration
  - depend of resource configuration
- privileged
  - trap when executed in user mode
- virtualization is possible if all sensistive instructions and privileged



CPU Virtualization Woes I

- problem: some instructions will only work from ring 0
- bigger problem: some instructions will behave “oddly” if issued from a wrong ring
- long-term problem: modern architectures (x86_64) tend to “clean up” by getting rid of rings 1 and 2, under the assumption that no one cares
  - but virtualization folks do!
- categories of “odd behaviour”
  - instructions that check which ring they are in
  - instructions that do not save the CPU state correctly when in a wrong ring
  - instructions that do not fault when they should
  - instructions that fault when they should not

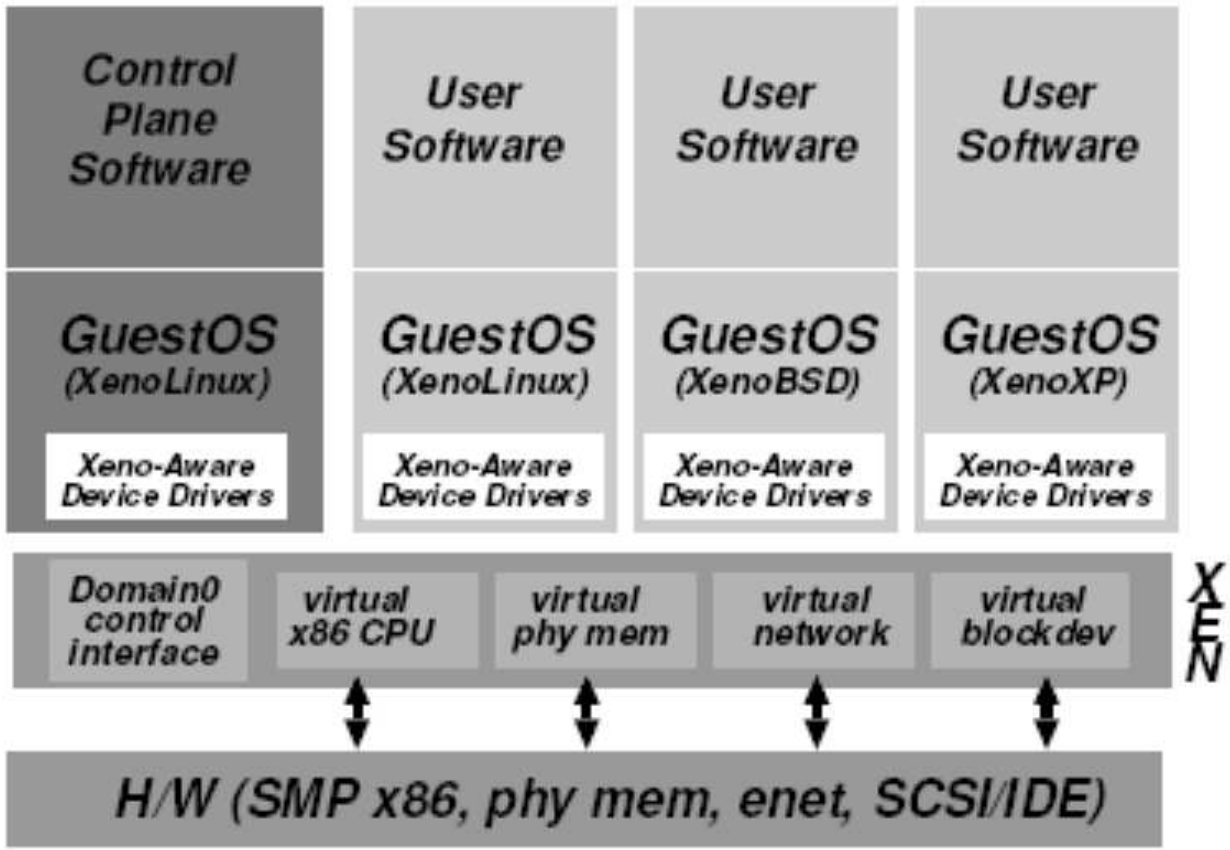
CPU Virtualization Woes II

- imagine an OS checking which ring it is in
  - it finds it is in ring 1, not ring 0
  - returns 1, not 0 — treated as (severe?) error
  - BSOD, panic, core dump, or other unpleasantries
  - binary translation can notice and fake 0, but it will take lots of instructions rather than one — performance hit
- saving CPU state
  - some CPU registers are not easy to save on context switch once loaded into memory
  - memory-resident portions and the real CPU state are not the same
  - workarounds are tricky and expensive

CPU Virtualization Woes III

- things that do not fail when they should
  - nothing happens when you expect a fault
  - you cannot catch the fault when you want to
  - example: `POPF` ignored by Intel when issued in userspace
- things that fail when they should not
  - example: writing to `CR0` or `CR4` (on Intel)
    - CR0: cache control, paging control, task switching...
    - e.g., on every HW context switch the `TS` flag is set in `CR0`
    - will fault if not performed in the correct ring
    - either crash or lots of effort and overhead

Para-Virtualization: Xen



http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf

Control Mechanisms

- hypercalls
  - synchronous calls from a guest OS (“domain,” “partition”) to the hypervisor
  - analogous to system calls in regular OSes
  - synchronous software trap into the hypervisor to perform a priviledged operation
  - example: request for page table updates
- asynchronous events
  - replace device interrupts on real hardware
  - examples: data received over network, virtual disk request has been completed
  - events can be deferred (like disabling interrupts)
  - quite similar to UNIX signals, handlers

Memory Virtualization: TLB

- software-managed TLB is relatively easy
- tagged TLB (supported by Alpha, MIPS, SPARC, and other RISC arcitectures) also helps
  - an address space tag is associated with each TLB entry
  - the hypervisor and the OSes co-exist in separate address spaces
  - no need to flush the entire TLB when execution is transferred
- Intel did it “wrong”:
  - TLB is hardware-managed by walking through HW page tables in case of miss
  - TLB is not tagged, so it must be flushed at address space switch

Memory Virtualization: Page Tables

- shadow page tables approach: each OS keeps a set of page tables distinct from hardware
  - hypervisor is responsible for trapping updates, validation, updating the real hardware page tables
- direct page table access: each OS has read-only access to the real page tables
  - hypervisor ensures isolation and protection

Memory Virtualization In Xen

- hypervisor is mapped into the top 64MB of every address space, so no TLB flushes when entering and exiting
- guest OSes register the page tables with Xen
- each OS allocates and initializes pages from its own memory reservation
- all subsequent write access must be validated by Xen
  - OS can map only pages it owns
  - no writeable mappings of page table
  - Xen's 64MB is not mappable (not required by any x86 ABI)
  - update requests to Xen may be batched

Segmentation Virtualization In Xen

- segmentation support is required by thread libraries with TLS (Thread-Local Storage) support
- similar approach to virtualization of hardware segment descriptor tables
  - hypervisor validates updates
- Xen itself is protected by a segment, so there can be no segments that overlap with it
- NPT (New POSIX Thread library) TLS uses segmentation in a way that conflicts with Xen
  - Xen needs binary rewriting to cope

Page Frame Types In Xen

- each machine page frame has a type and a reference count
- page types are mutually exclusive:
  - PD — page directory
  - PT — page table
  - GDT — global descriptor table
  - LDT — local descriptor table
  - RW — writeable
- e.g., writeable mappings are disallowed: a page frame cannot be PT and RW simultaneously
- a page frame may be retasked only if its reference count is 0

Page Validation And Pinning In Xen

- OS indicates when a page is allocated for page table use
  - page frame type is pinned as PD or PT, as appropriate
  - until a subsequent unpin request from the OS
- safety check: a page cannot be retasked until it is both unpinned and its reference count is reduced to 0
  - the hypervisor does not trust the guest OSes, and wants to protect against circumventing the reference counting mechanism by unpinning a used page

Processing Page Table Updates

- OSes can queue update requests locally
  - very useful optimization, e.g., when creating a new address space
  - requests must be committed early enough for correctness
- OS flushes TLB before using a new mapping
  - to ensure that cached translations are invalidated
  - committing pending updates before the flush will be enough to ensure correctness
- OS doesn't flush TLB if there are no stale entries
  - first use of new mapping will generate a page fault
  - the page fault handler must check for outstanding updates before retrying the faulting instruction

Ballooning

- most OSes happily utilize all the memory they have
  - they don't release unused memory until some process needs it
- we would like to manage memory dynamically
  - including taking memory from a guest OS when it is needed elsewhere
- "balloon driver":
  - tells the OS it needs so many MB of memory
  - OS allocates the memory which is not really used
  - the rest of the domain makes do with a smaller amount of memory
  - the hypervisor knows it can give the memory allocated to but unused by the balloon driver to another domain

I/O Virtualization In Xen

- Xen starts a priviledged control domain (dom0) that can touch all hardware devices in the system
- dom0 exports subsets of devices to other, “guest” domains (domU)
  - mechanism: asynchronous shared memory channels
  - even rings for interrupt handling
- dom0 runs a “backend”, exports “frontends” to domU
  - high level interface: block class, network class
- virtual PCI configuration space, virtual interrupts
- domU may be granted physical device access, securely

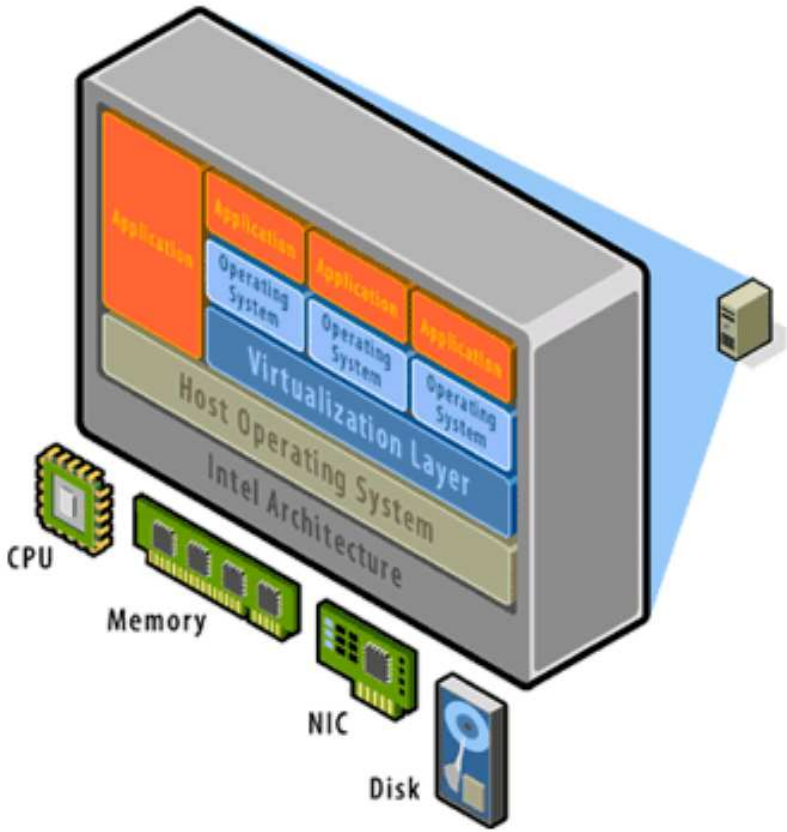
Time, Timers, And Scheduling In Xen

- real time: nanoseconds since boot of physical machine
  - maintained to the accuracy of the real timer
  - can be synced with a remote source (NTP)
- virtual time: advances only while a domain is executing
  - to ensure fair scheduling between the domain's processes
- wall-clock time: offset to be added to real time
  - can be adjusted without affecting the flow of real time
- each guest OS programs a pair of alarm timers: real and virtual
- there are numerous schedulers to time-share between domains

Full Virtualization: VMWare

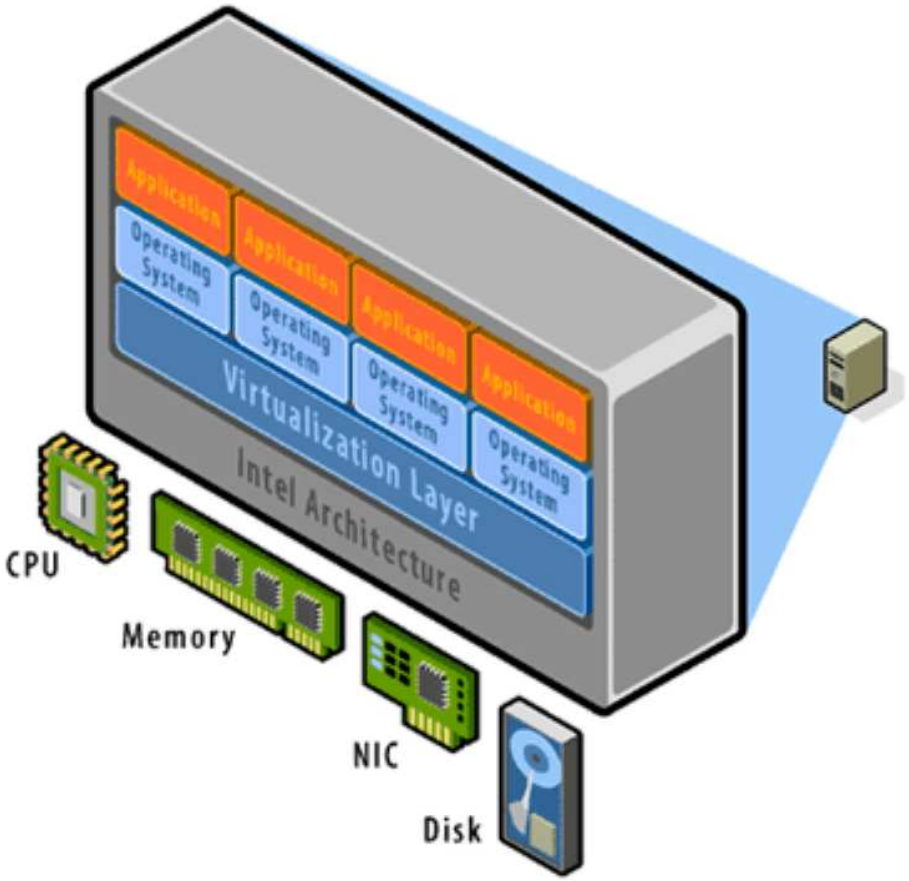
- what if we must run unmodified OS (e.g., Windows)?
- use full virtualization:
  - employ binary code translation
  - detect privileged instructions executed by a guest OS, change them on the fly
- can do what para-virtualization cannot
- suffers from performance degradation, because of the overhead of the virtualization mechanism

VMWare Workstation and GSX



http://www.vmware.com/products/server/gsx_features.html

VMWare ESX Server



http://www.vmware.com/products/server/esx_features.html

VMWare vs. Xen (or Full vs. Para)

- VMWare: 0/3, Xen: 0/1/3
- Xen is quite a bit faster:
  - Xen claim 97% of native performance
  - VMWare: slowdown of dozens of per cent
- VMWare: shadow page tables, Xen: access
- balloon support — both
- migration supported — both
- no modifications of application stack
- VMWare does not modify guest OS
  - closed source, licensing issues
  - e.g., Xen cannot run Windows as guest OS
    - it can — see below!

Hardware Support I

- how to avoid modifying guest OS and avoid the complexity and performance problems of full virtualization?
- support virtualization in hardware!
- keep the OS in ring 0, add another protection ring (-1?) below it
- solves all problems
- Intel's VT-x (for x86) and VT-i (Itanium) support announced
  - expect desktop-grade CPUs real soon now
  - possibly exists today in new machines, just disabled
- AMD's Pacifica — full spec released on 25/05/2005

Hardware Support II

- Xen has support for both VT and Pacifica built in
  - (partial) specs have been available for quite a while
  - (much) more implementation work is needed
  - folks at IBM and Intel run Windows in domU in the labs
- other architectures (e.g., Power) have had it since forever
  - trivia: Apple's G5 is Power 970 with virtualization support disabled
- catch up with VMWare on OS support, beat it on performance?
- the market will likely decide based on management tools