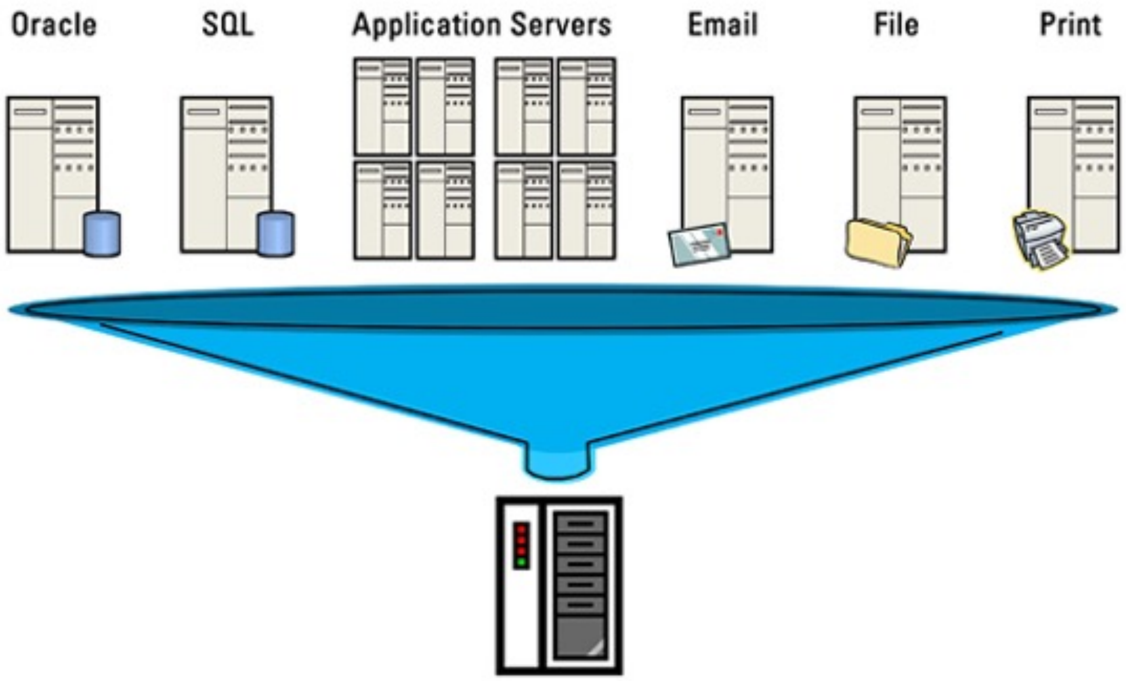
Operating Systems
Lecture

Virtual Machine

Prof. Mengwei Xu

Virtualization



- a technology that allows for the creation of virtual versions of physical resources

Virtual Machine

- VM is a very hot topic in both academia and industry
  - Industry commitment
    - ❑ Software: VMware, Virtualbox, Xen, Microsoft Virtual PC
    - ❑ Hardware: Intel VT, AMD-V
      - If Intel and AMD add it to their chips, you know it's serious…
  - Academia: so many papers on OSDI/SOSP

- An old idea, actually: developed by IBM in 60s and 70s
- But becomes super successful with cloud computing..

- Virtual Machine Monitor (VMM)
  - Also named Hypervisor

What is a VM

- We have seen that an OS already virtualizes
  - Syscalls, processes, virtual memory, file system, sockets, etc.
  - Applications program to this interface
- A VMM virtualizes an entire physical machine
  - Interface supported is the hardware
    - ❑ In contrast, OS defines a higher-level interface
  - VMM provides an illustration that software/OS has the full control over the hardware (of course, VMM is in control)
  - VMM "applications" run in virtual machines
- What does it mean?
  - You can boot an operating system in a virtual machine
  - Run multiple instances of an OS on same physical machine
  - Run different OSes simultaneously on the same machine
    - ❑ Linux on Windows, Windows on Mac, etc.

Why VM?

- Resource utilization
  - Machines today are powerful, want to multiplex their hardware
    - ❑ Cloud hosting can divvy up a physical machine to customers
  - Can migrate VMs from one machine to another without shutdown
- Software use and development
  - Can run multiple OSes simultaneously
    - ❑ No need to dual boot
  - Can do system (e.g., OS) development at user-level
- Many other cool applications
  - Debugging, emulation, security, speculation, fault tolerance…
- Common theme is manipulating applications/services at the granularity of a machine
  - Specific version of OS, libraries, applications, etc., as package

VM Requirement

- Fidelity
  - OSes and applications work the same without modification
    - ❑ (although we may modify the OS a bit)
- Isolation
  - VMM protects resources and VMs from each other
- Performance
  - VMM is another layer of software…and therefore overhead
    - ❑ As with OS, want to minimize this overhead
  - VMware (early):
    - ❑ CPU-intensive apps: 2-10% overhead
    - ❑ I/O-intensive apps: 25-60% overhead (much, much better today)

How VM Works at High Level

- VMM runs with privilege
  - OS in VM runs at "lesser" privilege (think user-level)
  - VMM multiplexes resources among VMs
- Want to run OS code in a VM directly on CPU
  - Think in terms of making the OS a user-level process
  - What OS code can run directly, what will cause problems?
  - Otherwise emulation..
- Ideally, want privileged instructions to trap
  - Exception vectors to VMM, it emulates operation, returns
  - Nothing modified, running unprivileged is transparent
  - Known as trap-and-emulate
- Unfortunately on architectures like x86, not so easy

Virtualization vs. Emulation vs. Simulation

- Virtualization (虚拟化)
- Emulation (仿真)
  - An emulator is a software that mimics the hardware and software environment.
- Simulation (模拟)
  - A simulator is a software which can mimic certain process or object.

- They are different concepts, but all involve creating managed environments where resources behave differently than they inherently do

Virtualization vs. Emulation vs. Simulation

- Emulation vs. Simulation
  - Fidelity to Original System
    - ❑ Emulation aims for high fidelity to the original system, replicating its hardware and software behavior.
    - ❑ Simulation aims to replicate the experience or outcome, not necessarily the underlying hardware or software processes.
  - Implementation
    - ❑ Emulation involves a detailed replication of the original platform's architecture, requiring a deep understanding of how the original system worked.
    - ❑ Simulation can be more flexible and might involve creating new code and algorithms to mimic the behavior of the original system.
  - Resource Requirements
    - ❑ Emulation can be resource-intensive, as it needs to mimic the original hardware's operations in real-time.
    - ❑ Simulation might be less demanding, especially if it only aims to replicate certain aspects of the original system.

Virtualization vs. Emulation vs. Simulation

- Emulation vs. Simulation
- The followings are..
  - QEMU: an [redacted] that enables running software designed for one type of CPU or architecture to run on a different one
  - Microsoft Flight [redacted]: A flight [redacted] that provides the experience of flying an aircraft. It models the flight environment and physics but does not replicate the specific hardware of an actual aircraft's control system.
  - PCSX2: an [redacted] that enables playing PS2 games on a PC.
  - SimCity: A city-building [redacted] game that [redacted] urban planning and management but does not emulate the software or hardware of real-world city management tools.

Virtualization vs. Emulation vs. Simulation

- Emulation vs. Simulation
- The followings are..
  - QEMU: an emulator that enables running software designed for one type of CPU or architecture to run on a different one
  - Microsoft Flight simulator: A flight simulator that provides the experience of flying an aircraft. It models the flight environment and physics but does not replicate the specific hardware of an actual aircraft's control system.
  - PCSX2: an emulator that enables playing PS2 games on a PC.
  - SimCity: A city-building simulation game that simulates urban planning and management but does not emulate the software or hardware of real-world city management tools.

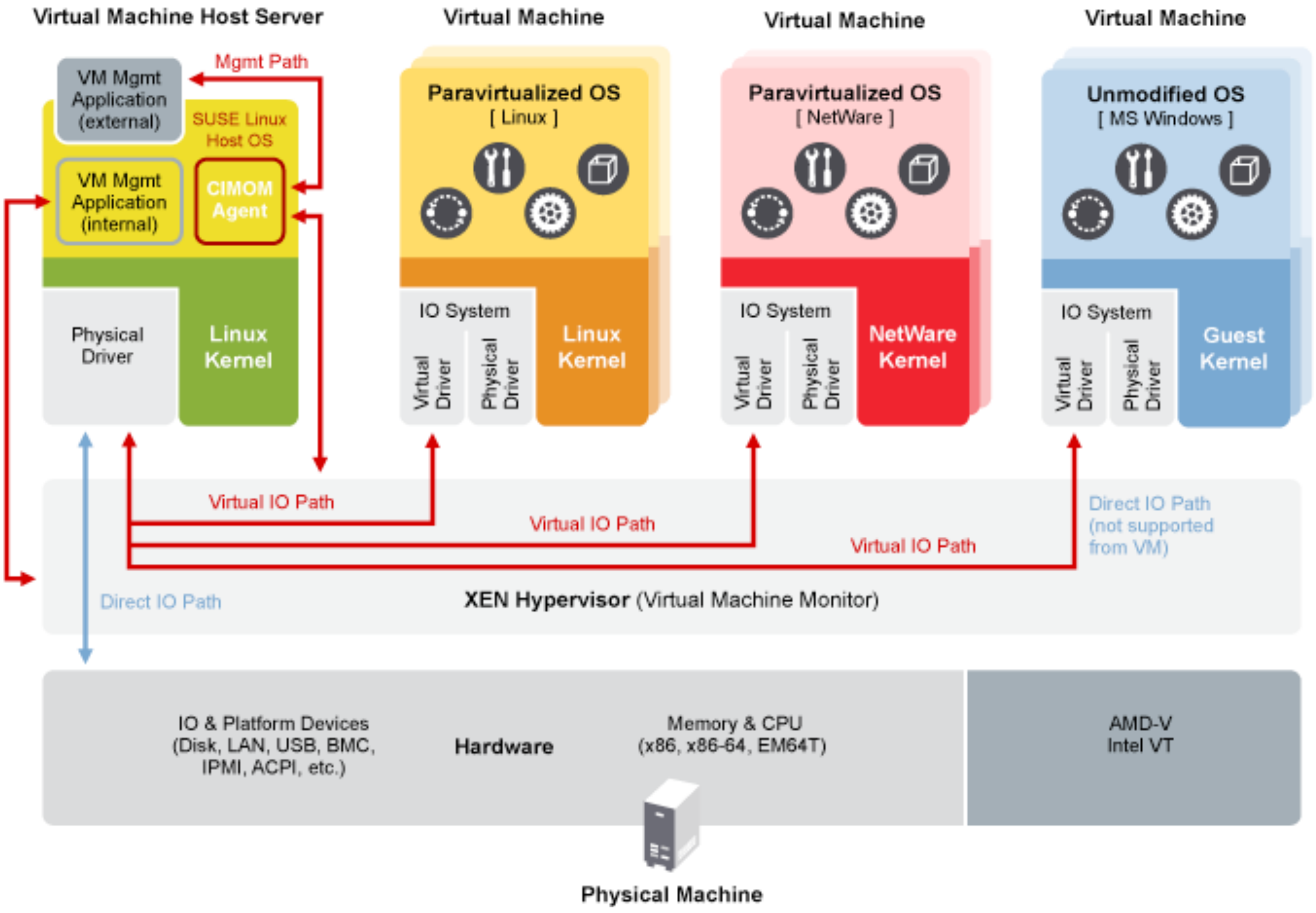
Virtualizing the x86

- Ease of virtualization influenced by the architecture
  - x86 is perhaps the last architecture you would choose
  - But it's what everyone uses, so…that's what we deal with
- Issues
  - Unvirtualizable events
    - popf does not trap when it cannot modify system flags (simply does nothing), so the host OS cannot capture this event!
  - Hardware-managed TLB
    - VMM cannot easily interpose on a TLB miss (more in a bit)
  - Untagged TLB
    - Have to flush on context switches (just a performance issue)
- Why Intel and AMD have added virtualization support

Xen

- Early versions use "paravirtualization"
  - Fancy word for "we have to modify & recompile the OS"
  - Since you're modifying the OS, make life easy for yourself
  - Create a VMM interface to minimize porting and overhead
- Xen hypervisor (VMM) implements interface
  - VMM runs at privilege, VMs (domains) run unprivileged
  - Trusted OS (Linux) runs in own domain (Domain0)
    - Use Domain0 to manage system, operate devices, etc.
- Most recent version of Xen does not require OS modifications
  - Because of Intel/AMD hardware support
- Commercialized via XenSource, but also open source

Xen Architecture

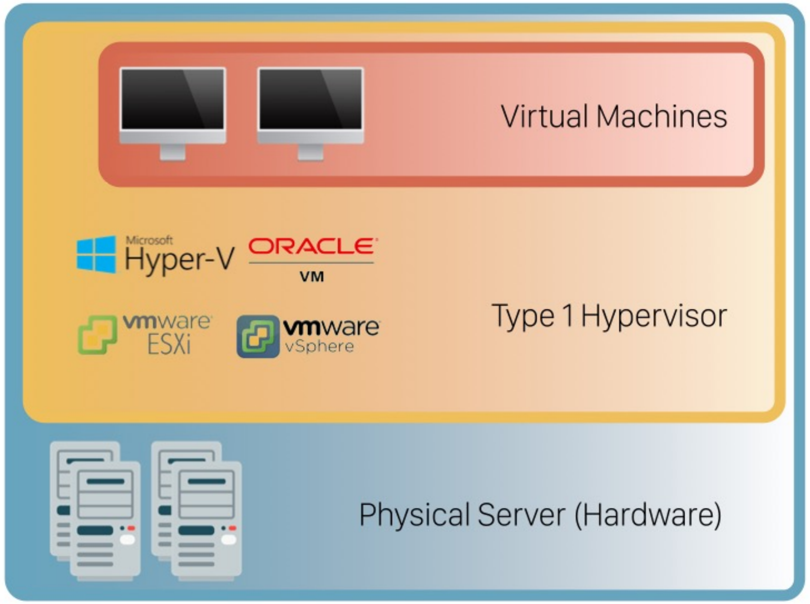


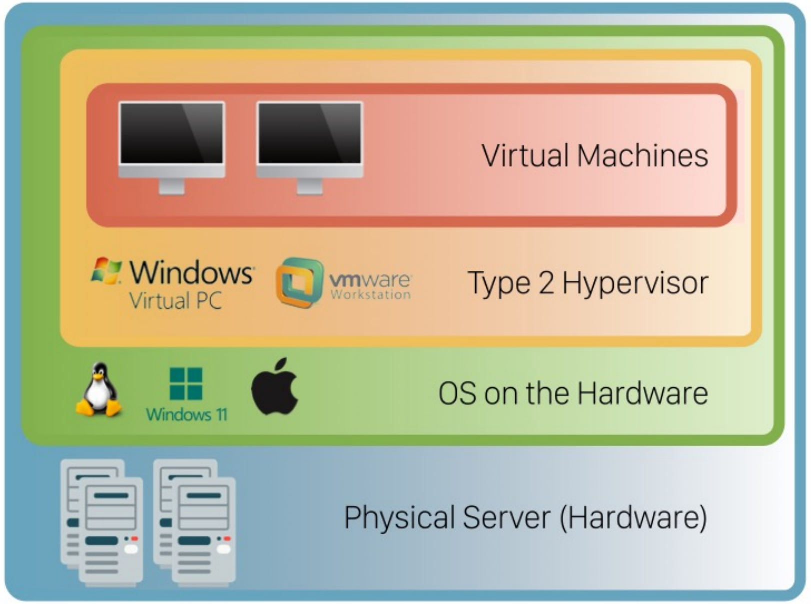
VMWare

- VMware workstation uses hosted model
  - VMM runs unprivileged, installed on base OS (+ driver)
  - Relies upon base OS for device functionality
- VMware ESX server uses hypervisor model
  - Similar to Xen, but no guest domain/OS
- VMware uses software virtualization
  - Dynamic binary rewriting translates code executed in VM
    - ❑ Rewrite privileged instructions with emulation code (may trap)
  - CPU only executes translated code
  - Think JIT compilation for JVM, but
    - ❑ full binary x86 ☐IR code ☐safe subset of x86
  - Incurs overhead, but can be well-tuned (small % hit)

VMM Types

- Type-1 hypervisor, also known as bare-metal or native.
  - Xen (or Citrix now), VMware vSphere with ESX/ESXi, KVM (Kernel-Based Virtual Machine), Microsoft Hyper-V, Oracle VM
- Type-2 hypervisor, also known as hosted hypervisors.
  - Oracle Virtualbox, VMware Workstation, Windows Virtual PC, Parallels Desktop

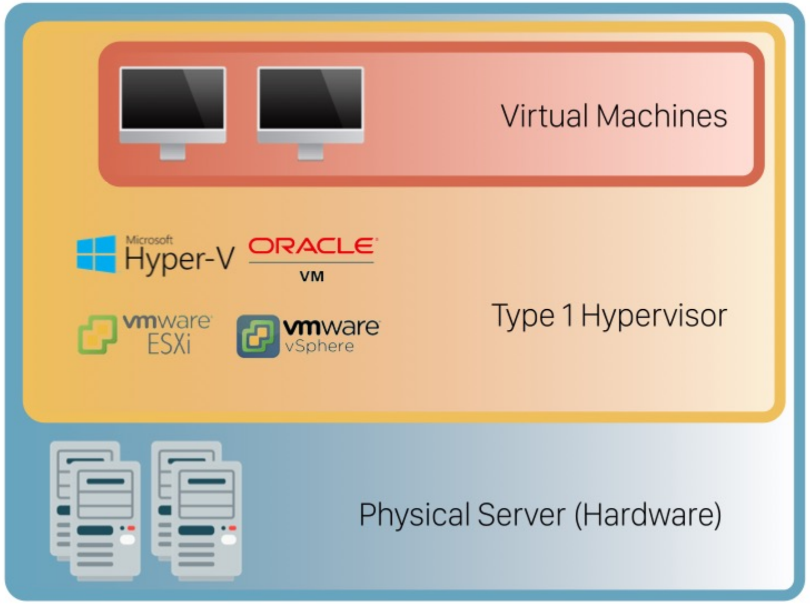


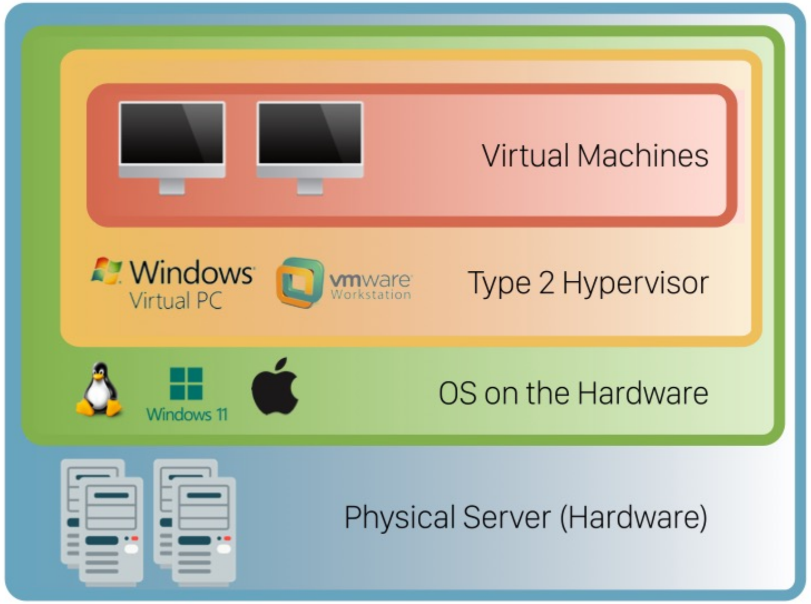


VMM Types

- Pros
  - VM mobility
  - Security
  - Resource over-allocation
- Cons
  - Limited functionality
  - Complicated management
  - Price (license)

- Pros
  - Easy to manage
  - Convenient for testing
  - Allows access to additional productivity tools
- Cons
  - Less flexible resource management
  - Performance
  - Security





https://phoenixnap.com/kb/what-is-hypervisor-type-1-2

Implementation: What Needs to be Virtualized?

- Exactly what you would expect
  - CPU
  - Events (exceptions and interrupts)
  - Memory
  - I/O devices
- Isn't this just duplicating OS functionality in a VMM?
  - Yes and no
  - Approaches will be similar to what we do with OSes
    - ❑ Simpler in functionality, though (VMM much smaller than OS)
  - But implements a different abstraction
    - ❑ Hardware interface vs. OS interface

Recall: Dual Mode

- Hardware-assisted isolation and protection
  - User mode (用户态) vs. kernel mode (内核态)
  - Teachers & TAs are in ?? mode, while students are in ?? mode
- What hardware needs to provide?
  - Privileged instructions (特权指令)
  - Memory protection
  - Timer interrupts
  - Safe mode transfer (in next course)

Recall: Privileged Instructions

- Instructions available in kernel mode but not user mode

Privileged Instructions	Non-privileged Instructions
I/O read/write	Performing arithmetic operations
Context switch	Call a function
Changing privilege level	Reading status of processor
Set system time	Read system time
..	..

Any instructions that could affect other processes are likely to be privileged.

Recall: Privileged Instructions (3/3)

- What if app executes a privileged instruction without permission?
  - Processor detects it in its hardware logic, and throws an exception (next course)
  - Process halted, OS takes over
- Demonstration with assembly code
  - Demonstration without assembly code (e.g., in pure C) is challenging

```
rtos@localhost:~/test $ ./a.out
Illegal instruction
rtos@localhost:~/test $ cat t.c
#include <stdio.h>

int main() {
    int cpsr;

    // Attempt to execute a privileged instruction (MRS - Move to Register from Status)
    // This instruction is only allowed in privileged modes (kernel mode).
    __asm__ __volatile__ ("MRS %0, s3_3_c13_c2_1" : "=r" (cpsr));

    // This code will execute after the privileged instruction above
    // without causing a compilation error.
    printf("Hello, World!\n");

    return 0;
}
rtos@localhost:~/test $
```

Recap: Hardware Support for OS

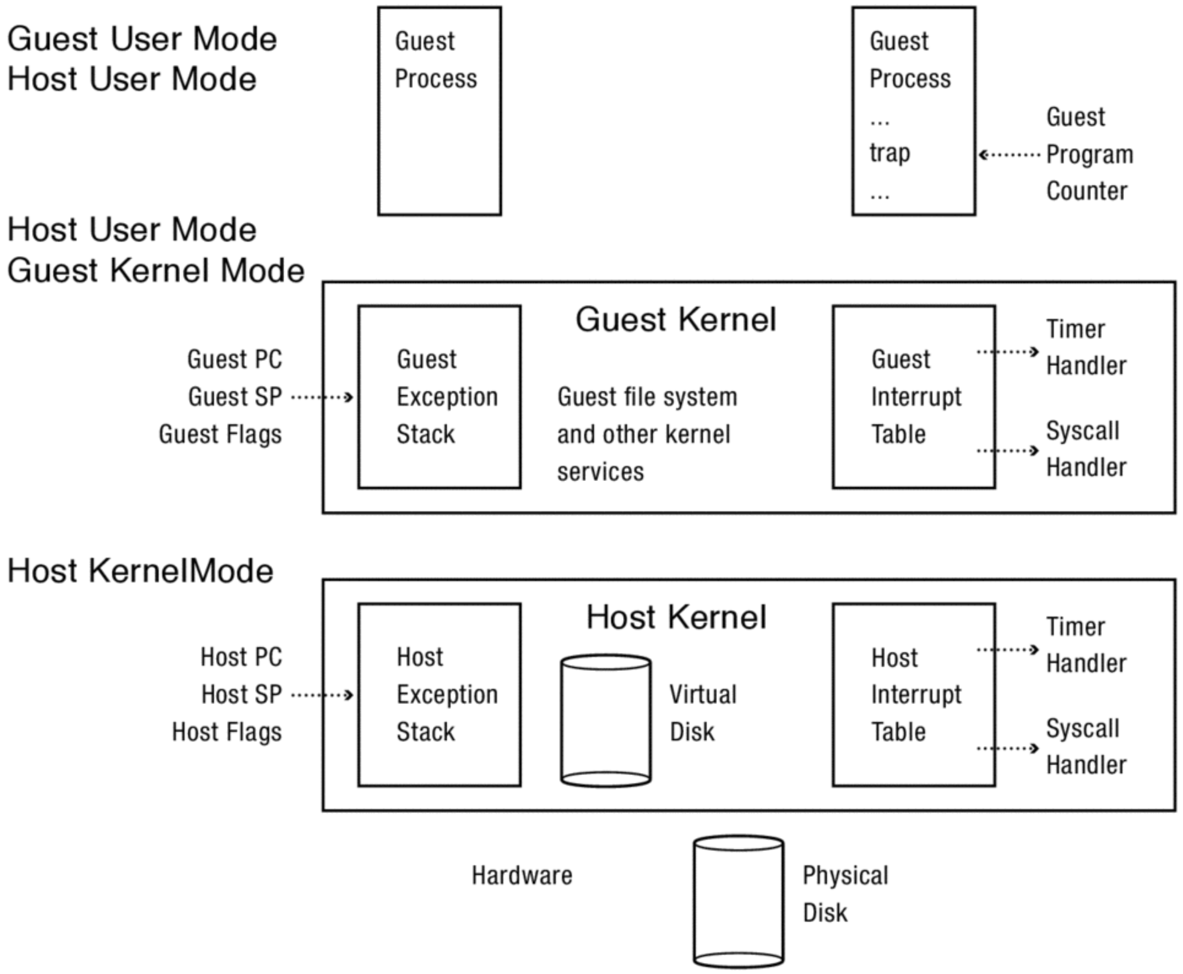
- Privilege levels, user and kernel.
- Privileged instructions: instructions available only in kernel mode.
- Memory translation prevents user programs from accessing kernel data structures and aids in memory management.
- Processor exceptions trap to the kernel on a privilege violation or other unexpected event.
- Timer interrupts return control to the kernel on time expiration.
- Device interrupts return control to the kernel to signal I/O completion.
- Inter-processor interrupts cause another processor to return control to the kernel.
- Interrupt masking prevents interrupts from being delivered at inopportune times.
- System calls trap to the kernel to perform a privileged action on behalf of a user program.
- Return from interrupt: switch from kernel mode to user mode, to a specific location in a user process.

Virtualizing Privileged Instructions

- OSes can no longer successfully execute privileged instructions
  - Virtual memory registers, interrupts, I/O, halt, etc.
- For those instructions that cause an exception
  - Trap to VMM, take care of business, return to OS in VM
- For those that do not…
  - Xen: modify OS to hypervisor call into VMM
  - VMware: rewrite OS instructions to emulate or call into VMM
  - H/W support: add new CPU mode, instructions to support trap and emulate

Cast Study: Mode Transfer

- When guest user process issues a syscall
  1. Traps into the host kernel's syscall handler (why?)
  2. The host kernel saves the $PC, $FLAGS, and user stack pointer on the interrupt stack of the guest kernel.
  3. The host kernel transfers control to the guest kernel, which runs with user-mode privilege.
  4. The guest kernel performs the system call — saving user state and checking arguments.
  5. When the guest kernel attempts to return from the system call back to guest user process (iret), this causes a processor exception, dropping back into the host kernel.
  6. The host kernel can then restore the state of the user process, running at user level, as if the guest OS had been able to return there directly.



Cast Study: Mode Transfer

- When guest user process issues a syscall
  1. Traps into the host kernel's syscall handler
  2. The host kernel saves the $PC, $FLAGS, and user stack pointer on the interrupt stack of the guest kernel.
  3. The host kernel transfers control to the guest kernel, which runs with user-mode privilege.
  4. The guest kernel performs the system call — saving user state and checking arguments.
  5. When the guest kernel attempts to return from the system call back to guest user process (iret), this causes a processor exception, dropping back into the host kernel.
  6. The host kernel can then restore the state of the user process, running at user level, as if the guest OS had been able to return there directly.

- How about processor exception? Similarly
  1. If the exception is caused by the guest user process, the host kernel forwards it to the guest kernel for handling
  2. If the exception is caused by the guest kernel (e.g., executing privileged instruction), the host kernel simulates it by itself.
  - Therefore, the host kernel must track whether the VM is executing in virtual user mode or virtual kernel mode.

Virtualizing the CPU

- VMM needs to multiplex VMs on CPU
- How? Just as you would expect
  - Timeslice the VMs
  - Each VM will timeslice its OS/applications during its quantum
- Typically relatively simple scheduler
  - Round robin, work-conserving (give unused quantum to other VMs)

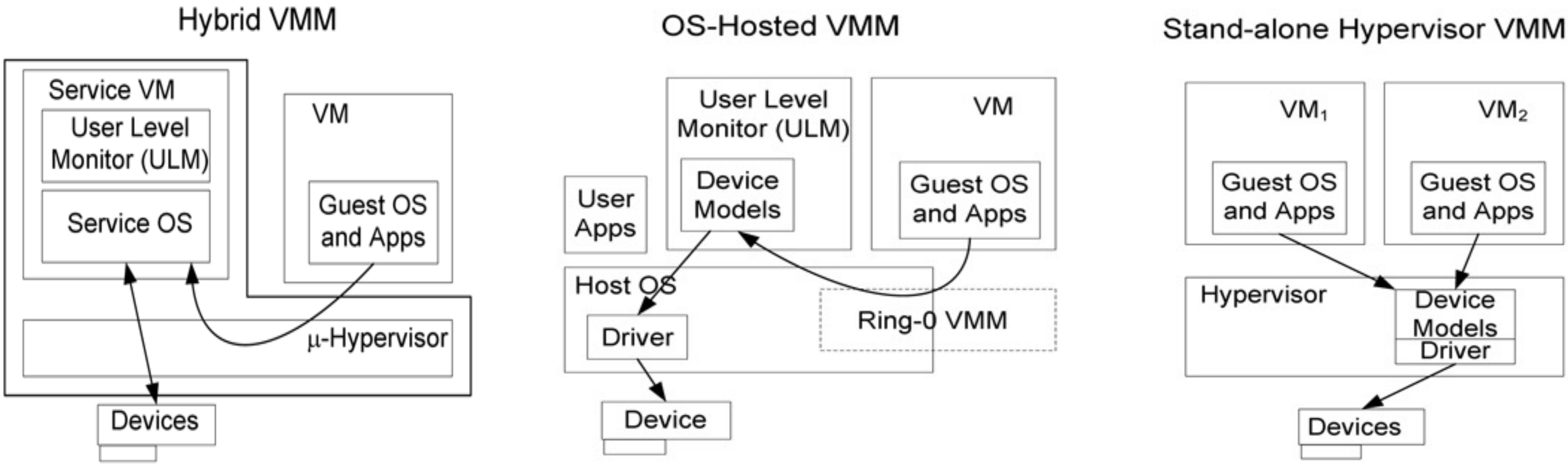
Virtualizing Events

- VMM receives interrupts, exceptions
- Needs to vector to appropriate VM
  - Xen: modify OS to use virtual interrupt register, event queue
  - VMware: craft appropriate handler invocation, emulate event registers

Virtualizing I/O

- OSes can no longer interact directly with I/O devices
- Xen: modify OS to use low-level I/O interface (hybrid)
  - Define generic devices with simple interface
    - ❑ Virtual disk, virtual NIC, etc.
  - Ring buffer of control descriptors, pass pages back and forth
  - Handoff to trusted domain running OS with real drivers
- VMware: VMM supports generic devices (hosted)
  - E.g., AMD Lance chipset/PCNetEthernet device
  - Load driver into OS in VM, OS uses it normally
  - Driver knows about VMM, cooperates to pass the buck to a real device driver (e.g., on underlying host OS)
- VMware ESX Server: drivers run in VMM (hypervisor)

Virtualizing I/O



Abramson et al., "Intel Virtualization Technology for Directed I/O", Intel Technology Journal, 10(3) 2006

Virtualizing Memory

- OSes assume they have full control over memory
  - Managing it: OS assumes it owns it all
  - Mapping it: OS assumes it can map any virtual page to any physical page
- But VMM partitions memory among VMs
  - VMM needs to assign hardware pages to VMs
  - VMM needs to control mappings for isolation
    - ❑ Cannot allow an OS to map a virtual page to any hardware page
    - ❑ OS can only map to a hardware page given to it by the VMM
- Hardware-managed TLBs make this difficult
  - When the TLB misses, the hardware automatically walks the page tables in memory
  - As a result, VMM needs to control access by OS to page tables

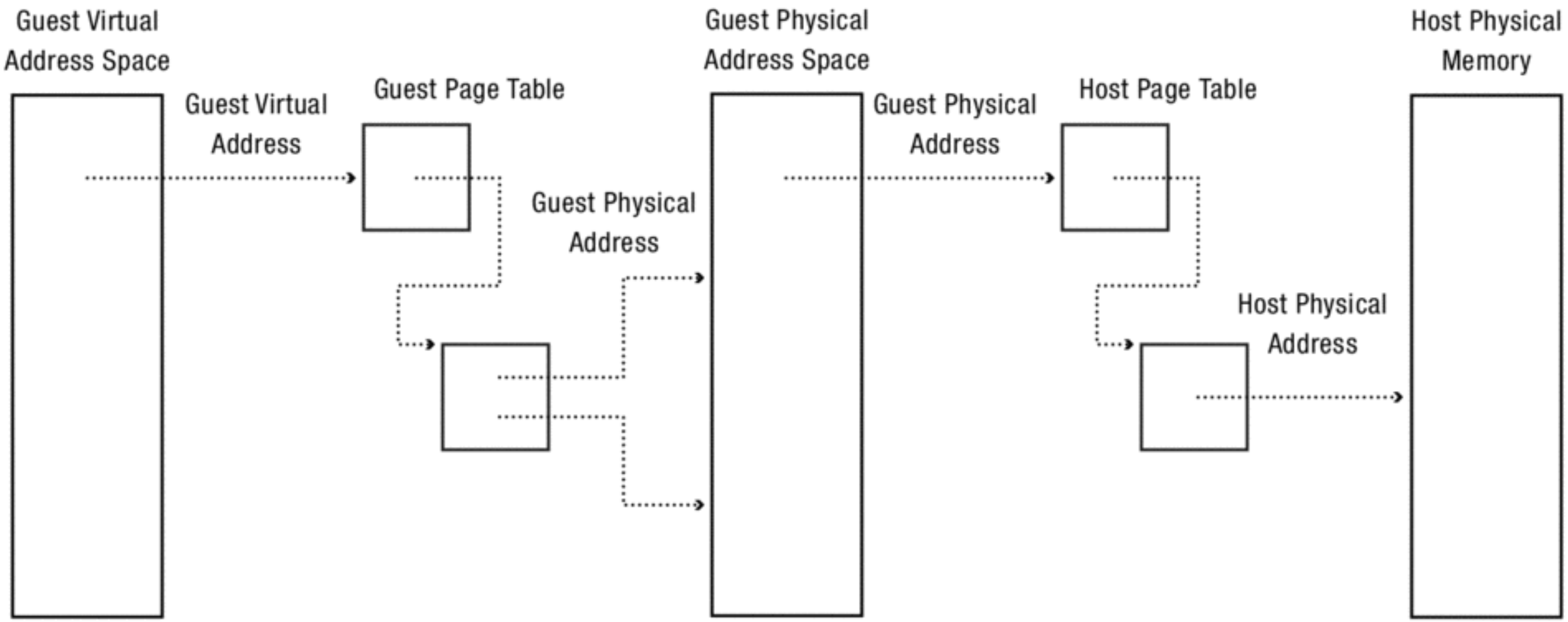
Xen Paravirtualization

- Xen uses the page tables that an OS creates
  - These page tables are used directly by hardware MMU
- Xen validates all updates to page tables by OS
  - OS can read page tables without modification
  - But Xen needs to check all PTE writes to ensure that the virtual-to-physical mapping is valid
    - That the OS "owns" the physical page being used in the PTE
  - Modify OS to hypervisor call into Xen when updating PTEs
    - Batch updates to reduce overhead
- Page tables work the same as before, but OS is constrained to only map to the physical pages it owns
- Works fine if you can modify the OS. If you can't…

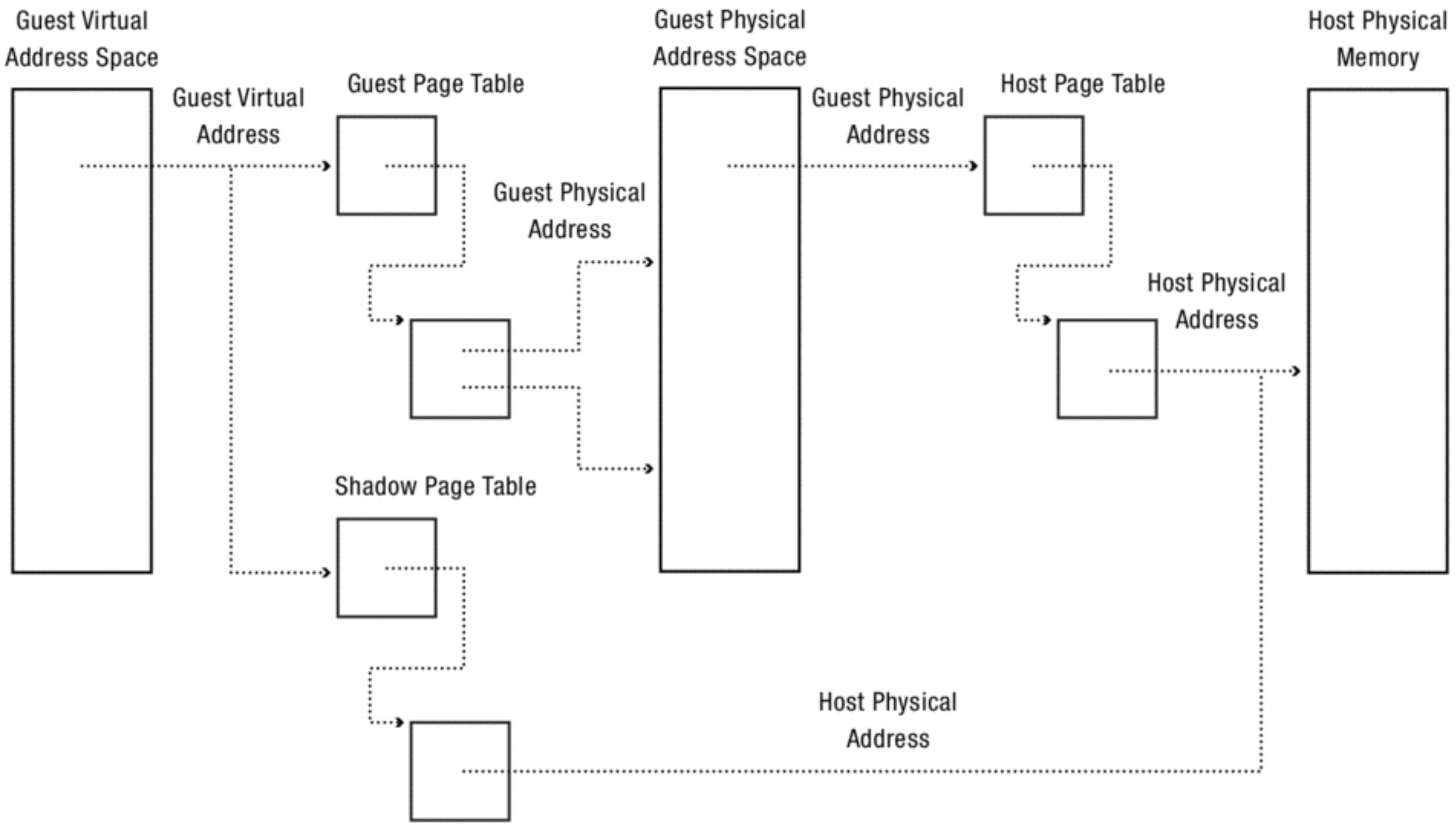
Three Abstractions of Memory

- Three abstractions of memory
  - Machine: actual hardware memory
    - ❑ 16 GB of DRAM
  - Physical: abstraction of hardware memory managed by OS
    - ❑ If a VMM allocates 512 MB to a VM, the OS thinks the computer has 512 MB of contiguous physical memory
    - ❑ (Underlying machine memory may be discontiguous)
  - Virtual: virtual address spaces you know and love
- In each VM, OS creates and manages page tables for its virtual address spaces without modification
  - But these page tables are not used by the MMU hardware

Three Abstractions of Memory



Shadow Page Tables



Shadow Page Tables

- VMM creates and manages page tables that map virtual pages directly to machine pages
  - These tables are loaded into the MMU on a context switch
  - VMM page tables are the shadow page tables
- VMM needs to keep its shadow page tables up-to-date
  - When host kernel changes its page table, it knows it's changed
  - When guest kernel changes its page table, however..
    - ❑ VMM maps OS page tables as read only
    - ❑ When OS writes to page tables, trap to VMM
    - ❑ VMM applies write to shadow table and OS table, returns
    - ❑ Also known as memory tracing
    - ❑ Again, more overhead…

Shadow Page Tables

- In recent Intel architecture adds direct support for shadow page table
  - The hardware can be set up with 2 page tables
  - When a TLB miss occurs in guest process
    - ❑ The hardware translates the guest VM address to the guest PM address, and then to the host PM address directly
    - ❑ As if the host maintained an explicit shadow page table (but it doesn't)
  - Pros
    - ❑ Simplifies the VM implementation
    - ❑ Switch between guest/host page table is easier and faster
  - Cons
    - ❑ The handling of a TLB miss (a full address translation) is slower since the host operating system must consult two page tables instead of one

Memory Allocation

- VMMs tend to have simple hardware memory allocation policies
  - Static: VM gets 512 MB of hardware memory for life
  - No dynamic adjustment based on load
    - ❑ OSes not designed to handle changes in physical memory…
  - No swapping to disk
- More sophistication: Overcommit with balloon driver
  - Balloon driver runs inside OS to consume hardware pages
    - ❑ Steals from virtual memory and file buffer cache (balloon grows)
  - Gives hardware pages to other VMs (those balloons shrink)
- Identify identical physical pages (e.g., all zeroes)
  - Map those pages copy-on-write across VMs

Hardware Support

- Intel and AMD implement virtualization support in their recent x86 chips (Intel VT-x, AMD-V)
  - Goal is to fully virtualize architecture
  - Transparent trap-and-emulate approach now feasible
  - Echoes hardware support originally implemented by IBM
- Execution model
  - New execution mode: guest mode
    - ❑ Direct execution of guest OS code, including privileged insts
  - Virtual machine control block (VMCB)
    - ❑ Controls what operations trap, records info to handle traps in VMM
  - New instruction *vmenter* enters guest mode, runs VM code
  - When VM traps, CPU executes new *vmexit* instruction
  - Enters VMM, which emulates operation

Hardware Support

- Memory
  - Intel extended page tables (EPT), AMD nested page tables (NPT)
  - Original page tables map virtual to (guest) physical pages
    - ❑ Managed by OS in VM, backwards compatible
    - ❑ No need to trap to VMM when OS updates its page tables
  - New tables map physical to machine pages
    - ❑ Managed by VMM
  - Tagged TLB w/ virtual process identifiers (VPIDs)
    - ❑ Tag VMs with VPID, no need to flush TLB on VM/VMM switch
- I/O
  - Constrain DMA operations only to page owned by specific VM
  - AMD DEV: exclude pages (c.f. Xen memory paravirtualization
  - Intel VT d: IOMMU address translation support for DMA

Java Virtual Machine (JVM)

- Java program runs atop JVM, and that's why it's ubiquitous..
  - And also why it runs slower than native program written in C/C++.
  - Garbage collection
  - Just-in-time Compile (JIT, 即时编译)

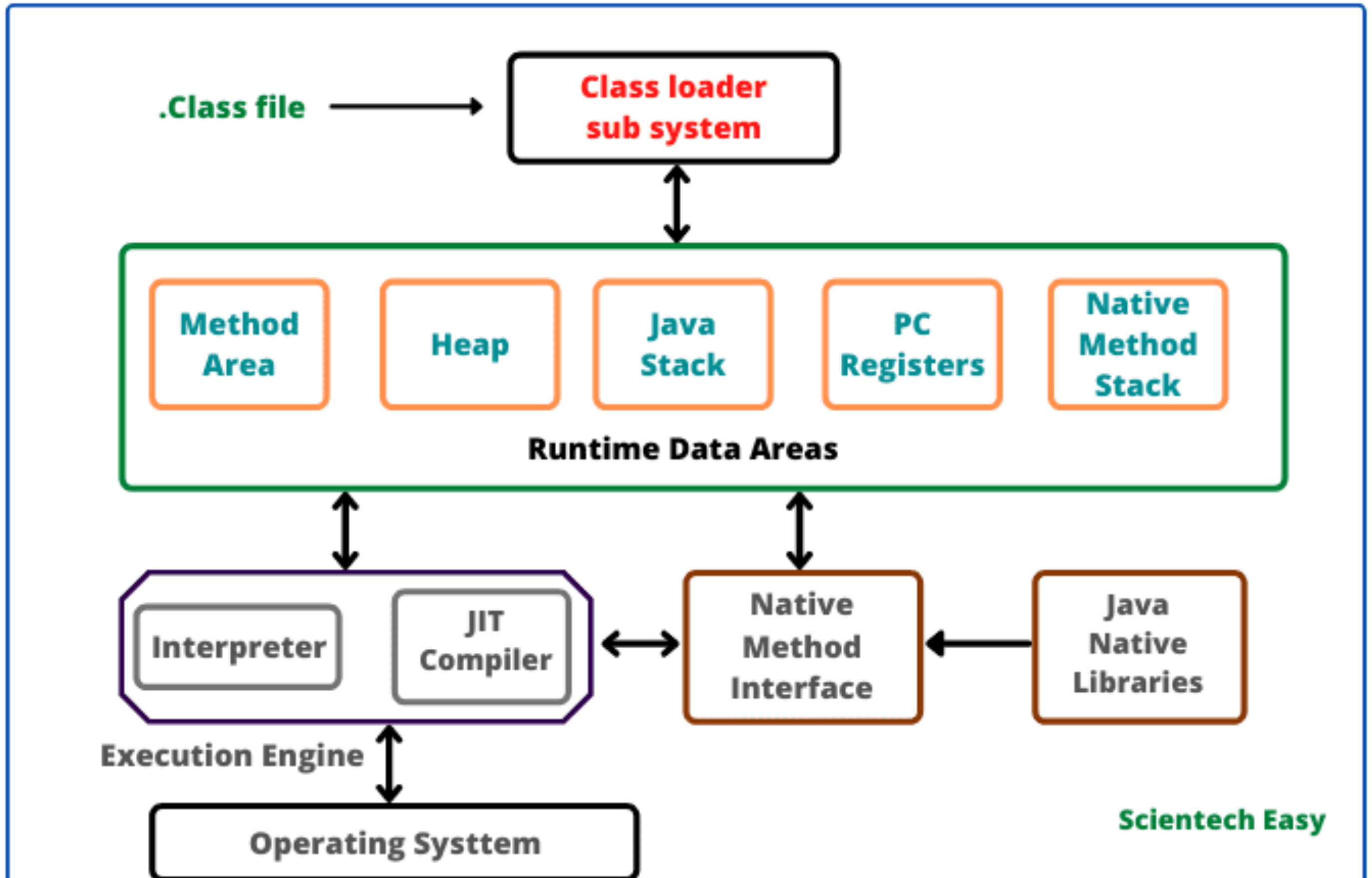
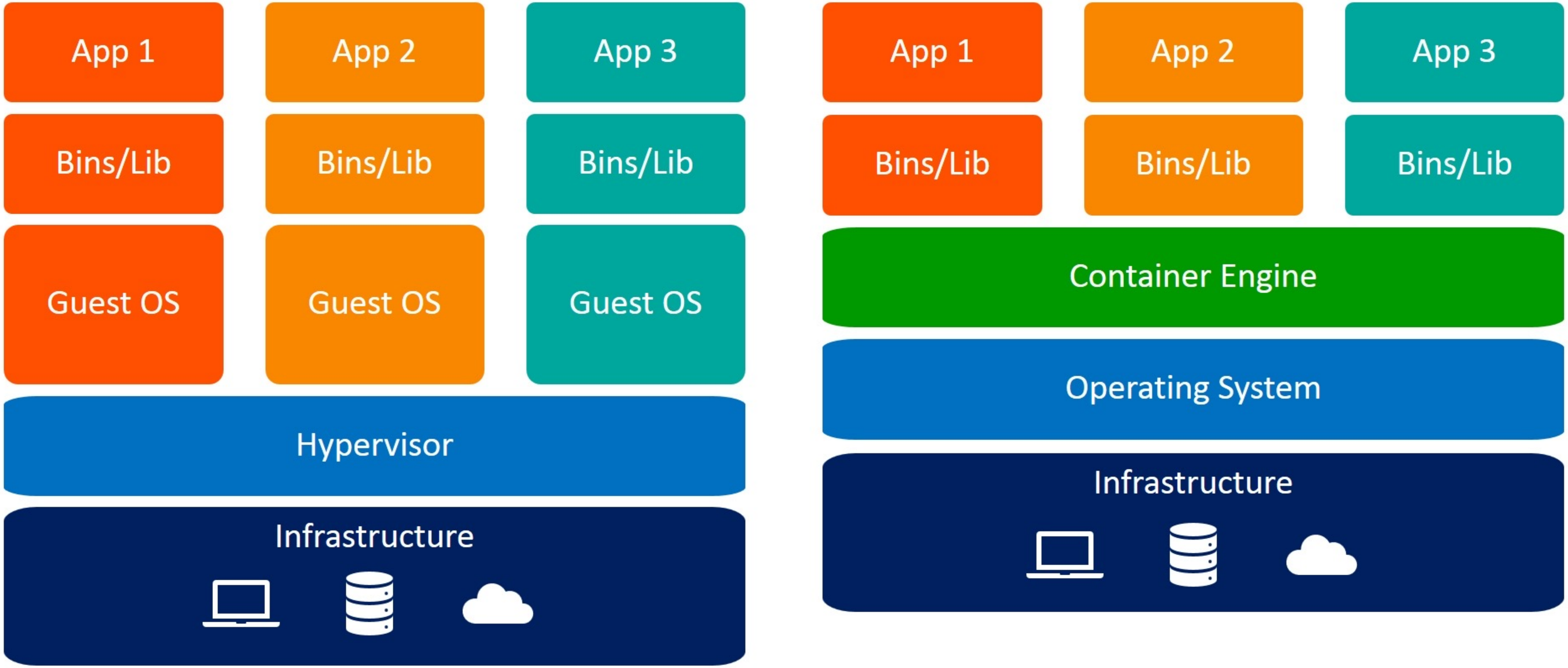


Fig: Components in JVM Architecture

VM vs. Containers (容器)



Virtual Machines

Containers