CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

Virtual Machines Building Blocks

Ch 18.4-18.6

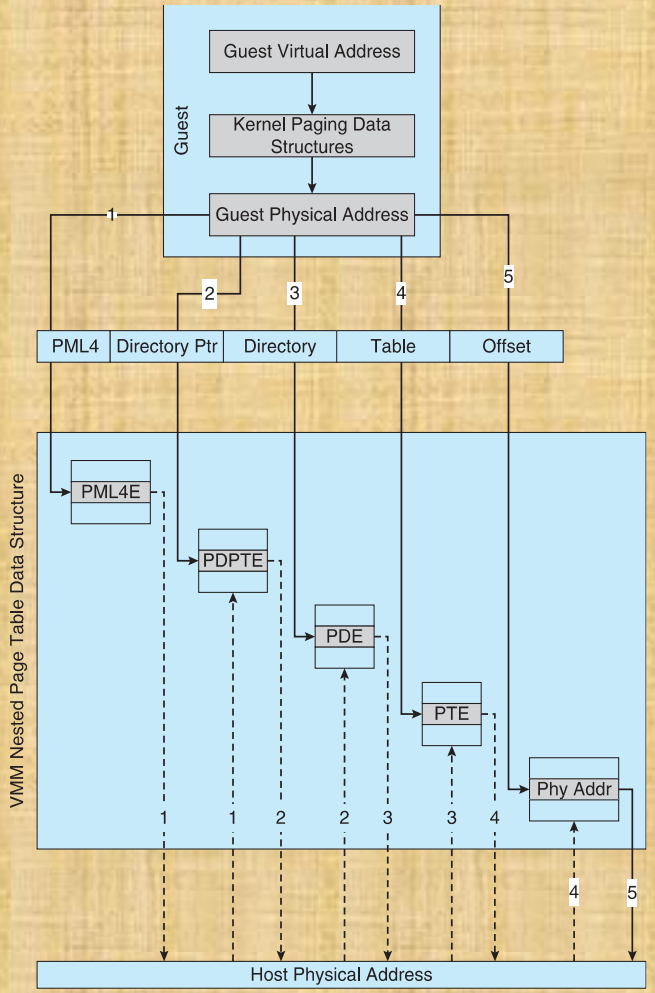
This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

Building Blocks – Hardware Assistance

- All virtualization needs some HW support
- More support -> more feature rich, stable, better performance of guests
- Intel added new **VT-x** instructions in 2005 and AMD the **AMD-V** instructions in 2006
  - CPUs with these instructions remove need for binary translation
  - Generally define more CPU modes – “guest” and “host”
  - VMM can enable host mode, define characteristics of each guest VM, switch to guest mode and guest(s) on CPU(s)
  - In guest mode, guest OS thinks it is running natively, sees devices (as defined by VMM for that guest)
    - Access to virtualized device, priv instructions cause trap to VMM
    - CPU maintains VCPU, context switches it as needed
- HW support for Nested Page Tables, DMA, interrupts as well over time

https://binarydebt.wordpress.com/2018/10/14/intel-virtualisation-how-vt-x-kvm-and-qemu-work-together/

Nested Page Tables



See details from:
http://developer.amd.com/wordpress/media/2012/10/NPT-WP-1%201-final-TM.pdf

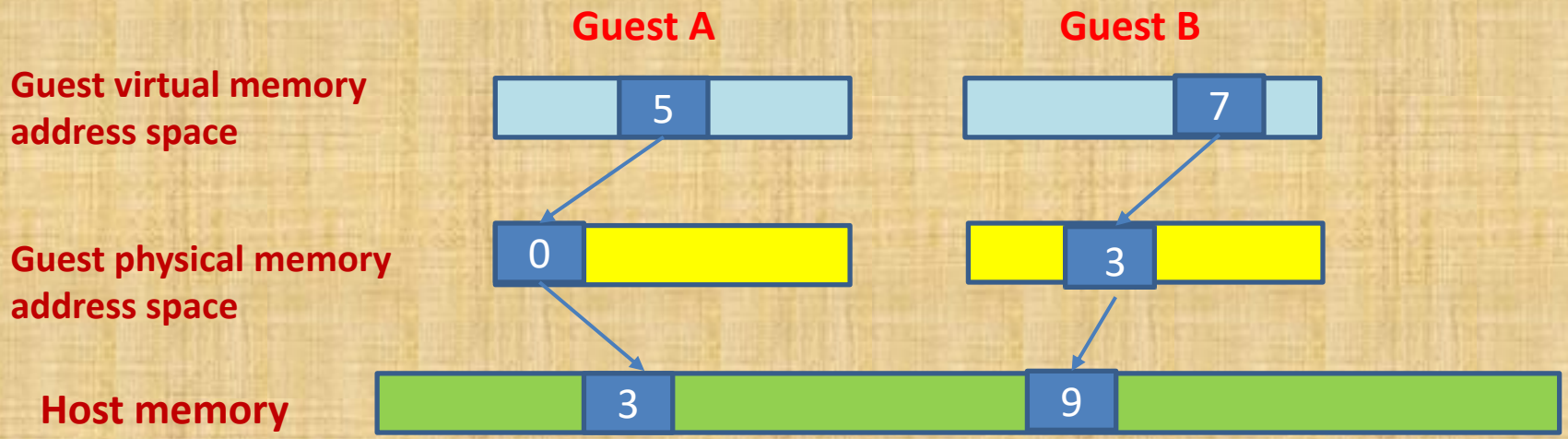
A total of 48 bit virtual address space.

- PML4: Page Map Level 4 offset, 9 bits
- PDPTE: Page Directory Pointer Offset, 9 bits
- PDE: Page Directory Offset, 9 bits
- PTE: Page Table Entry, 9 bits
- Offset: 12-bit page offset (4 K page size)

Example implantation with 4 K page size.

A Simplified, Logical View

The actual AMD implementation shown in previous page, which is representative, looks complicated. But the idea isn't. Here is a simplified view.
https://www.cs.cmu.edu/~dga/15-440/F11/lectures/vm-ucsd.pdf



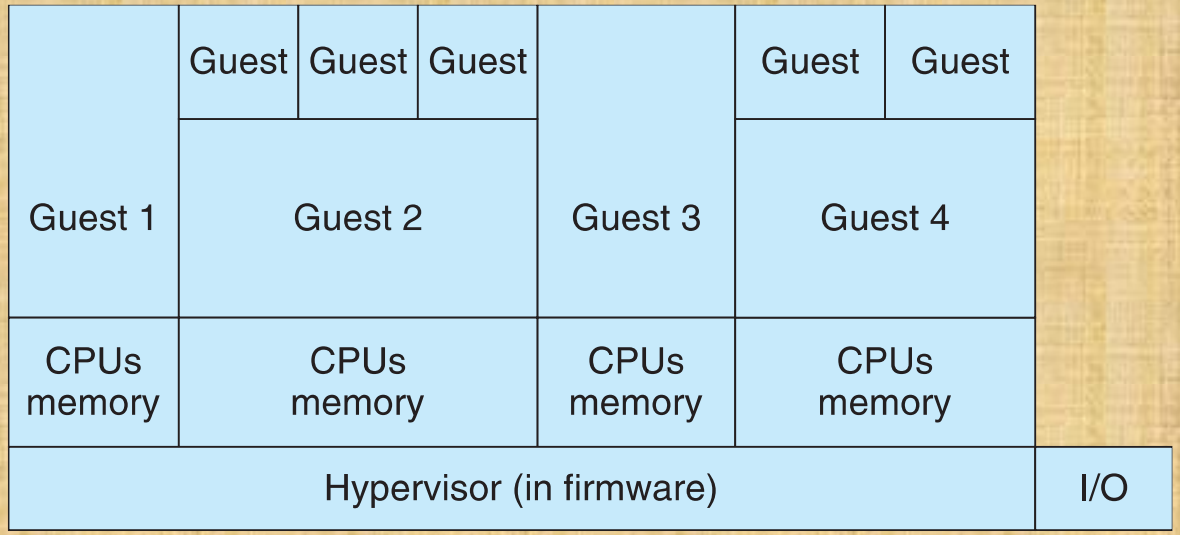
Types of Virtual Machines and Implementations

- Many variations as well as HW details
  - Assume VMMs take advantage of HW features
    - HW features can simplify implementation, improve performance
- Whatever the type, a VM has a lifecycle
  - Created by VMM
  - Resources assigned to it (number of cores, amount of memory, networking details, storage details)
  - In type 0 hypervisor, resources usually dedicated
  - Other types dedicate or share resources, or a mix
  - When no longer needed, VM can be deleted, freeing resources
- Steps simpler, faster than with a physical machine install
  - Can lead to **virtual machine sprawl** with lots of VMs, history and state difficult to track

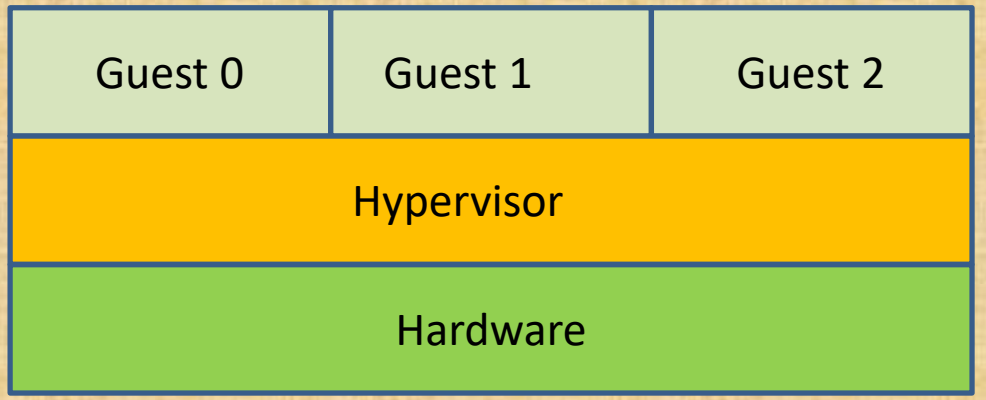
Types of VMs – Type 0 Hypervisor

- Old idea, under many names by HW manufacturers
  - "partitions", "domains"
  - A HW feature implemented by firmware
  - OS need to nothing special, VMM is in firmware
  - Smaller feature set than other types
  - Each guest has dedicated HW
- I/O a challenge as difficult to have enough devices, controllers to dedicate to each guest
- Sometimes VMM implements a **control partition** running daemons that other guests communicate with for shared I/O
- Can provide virtualization-within-virtualization (guest itself can be a VMM with guests
  - Other types have difficulty doing this

Type 0 Hypervisor



Type 1 Hypervisor



https://www.cs.dartmouth.edu/~sergey/cs258/2014/TorreyGuestLecture-Hypervors.pdf

Types of VMs – Type 1 Hypervisor

- Commonly found in company datacenters
  - In a sense becoming “datacenter operating systems”
    - Datacenter managers control and manage OSes in new, sophisticated ways by controlling the Type 1 hypervisor
    - Consolidation of multiple OSes and apps onto less HW
    - Move guests between systems to balance performance
    - Snapshots and cloning

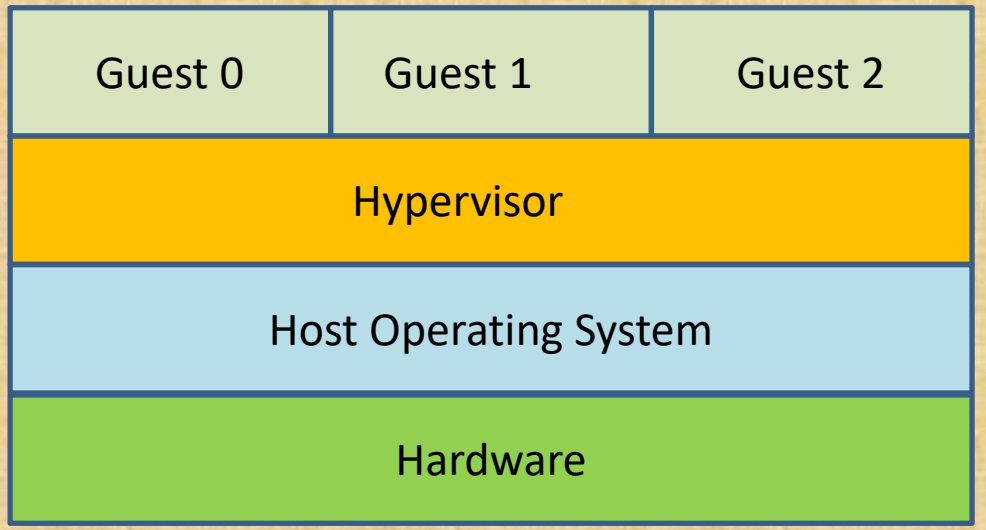
Types of VMs – Type 1 Hypervisor (Cont.)

- Special purpose operating systems that run natively on HW
  - Rather than providing system call interface, create run and manage guest OSes
  - Can run on Type 0 hypervisors but not on other Type 1s
  - Run in kernel mode
  - Guests generally don’t know they are running in a VM
  - Implement device drivers for host HW because no other component can
  - Also provide other traditional OS services like CPU and memory management

Types of VMs – Type 1 Hypervisor (Cont.)

- Another variation is a general purpose OS that also provides VMM functionality
  - RedHat Enterprise Linux with KVM, Windows with Hyper-V, Oracle Solaris
  - Perform normal duties as well as VMM duties
  - Typically less feature rich than dedicated Type 1 hypervisors
- In many ways, treat guests OSes as just another process
  - Albeit with special handling when guest tries to execute special instructions

Type 2 Hypervisor



https://www.cs.dartmouth.edu/~sergey/cs258/2014/TorreyGuestLecture-Hypervors.pdf

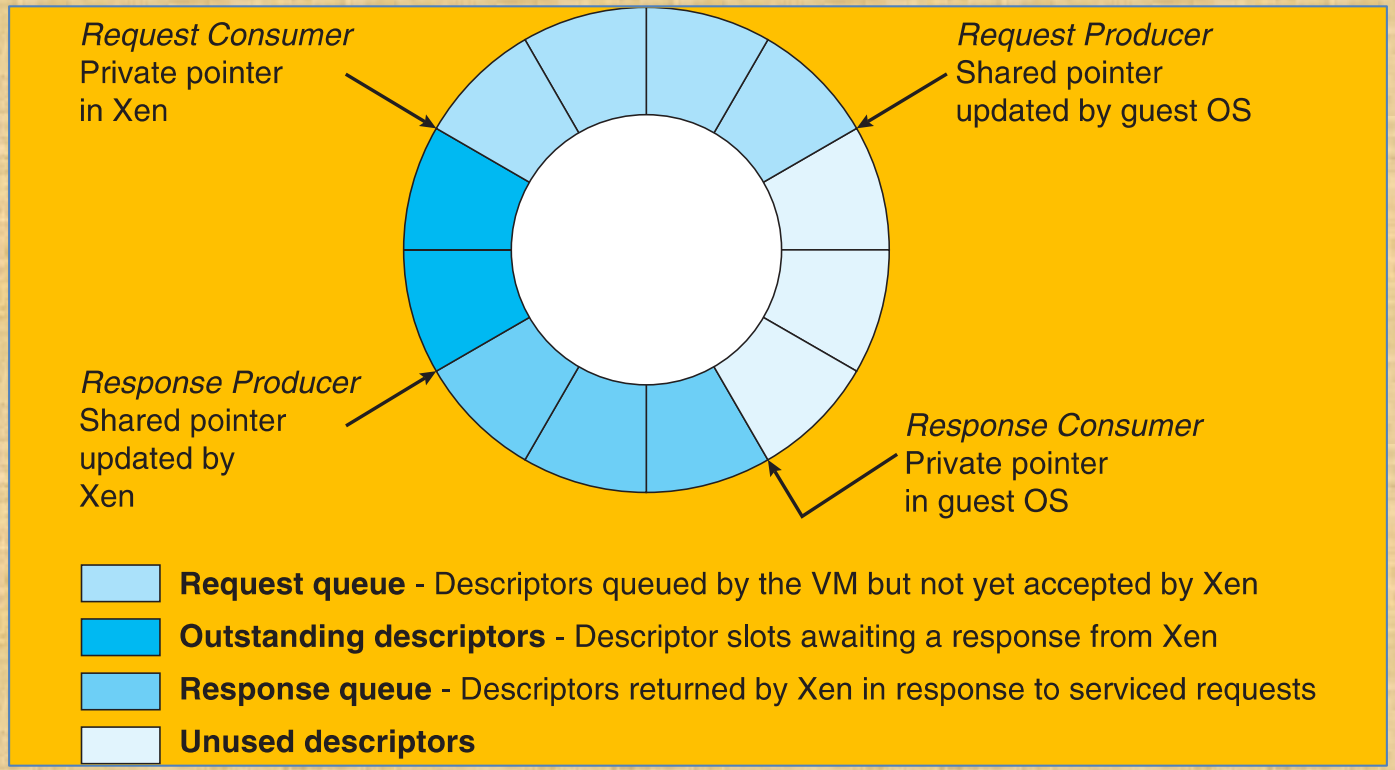
Types of VMs – Type 2 Hypervisor

- Less interesting from an OS perspective
  - Very little OS involvement in virtualization
  - VMM is simply another process, run and managed by host
    - Even the host doesn't know they are a VMM running guests
  - Tend to have poorer overall performance because can't take advantage of some HW features
  - But also a benefit because require no changes to host OS
    - Student could have Type 2 hypervisor on native host, run multiple guests, all on standard host OS such as Windows, Linux, MacOS

Types of VMs – Paravirtualization

- Does not fit the definition of virtualization – VMM not presenting an exact duplication of underlying hardware
  - But still useful!
  - VMM provides services that guest must be modified to use
  - Leads to increased performance
  - Less needed as hardware support for VMs grows
- Xen, leader in paravirtualized space, adds several techniques
  - For example, clean and simple device abstractions
    - Efficient I/O
    - Good communication between guest and VMM about device I/O
    - Each device has circular buffer shared by guest and VMM via shared memory

Xen I/O via Shared Circular Buffer



Types of VMs – Paravirtualization (Cont.)

- Xen, leader in paravirtualized space, adds several techniques (Cont.)
  - Memory management does not include nested page tables
    - Each guest has own read-only tables
    - Guest uses **hypercall** (call to hypervisor) when page-table changes needed
- Paravirtualization allowed virtualization of older x86 CPUs (and others) without binary translation
- Guest had to be modified to use run on paravirtualized VMM
- But on modern CPUs Xen no longer requires guest modification -> no longer paravirtualization

Types of VMs – Programming Environment Virtualization

- Also not-really-virtualization but using same techniques, providing similar features
- Programming language is designed to run within custom-built virtualized environment
  - For example Oracle Java has many features that depend on running in **Java Virtual Machine** (**JVM**)
- In this case virtualization is defined as providing APIs that define a set of features made available to a language and programs written in that language to provide an improved execution environment
- JVM compiled to run on many systems (including some smart phones even)
- Programs written in Java run in the JVM no matter the underlying system
- Similar to **interpreted languages**

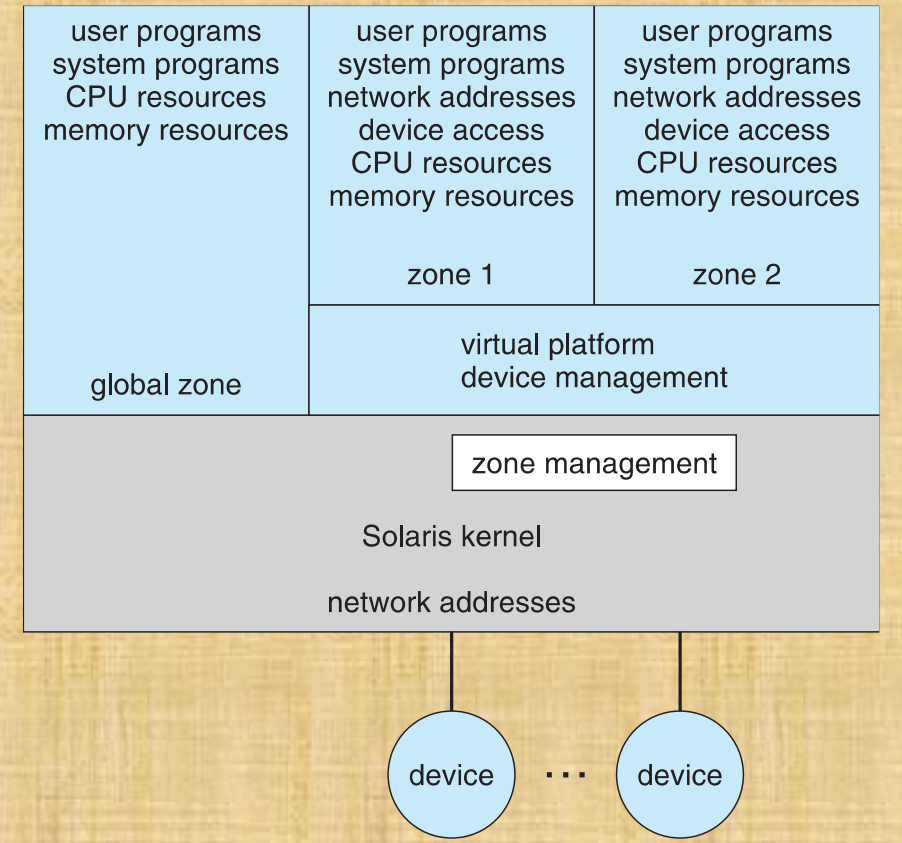
Types of VMs – Emulation

- Another (older) way for running one operating system on a different operating system
  - Virtualization requires underlying CPU to be same as guest was compiled for
  - Emulation allows guest to run on different CPU
- Necessary to translate all guest instructions from guest CPU to native CPU
  - Emulation, not virtualization
- Useful when host system has one architecture, guest compiled for other architecture
  - Company replacing outdated servers with new servers containing different CPU architecture, but still want to run old applications
- Performance challenge – order of magnitude slower than native code
  - New machines faster than older machines so can reduce slowdown
- Very popular – especially in gaming where old consoles emulated on new

Types of VMs – Application Containment

- Some goals of virtualization are segregation of apps, performance and resource management, easy start, stop, move, and management of them
- Can do those things without full-fledged virtualization
  - If applications compiled for the host operating system, don't need full virtualization to meet these goals
- Oracle **containers** / **zones** for example create virtual layer between OS and apps
  - Only one kernel running – host OS
  - OS and devices are virtualized, providing resources within zone with impression that they are only processes on system
  - Each zone has its own applications; networking stack, addresses, and ports; user accounts, etc
  - CPU and memory resources divided between zones
    - Zone can have its own scheduler to use those resources

Solaris 10 with Two Zones



Virtualization and Operating-System Components

- Now look at operating system aspects of virtualization
  - CPU scheduling, memory management, I/O, storage, and unique VM migration feature
    - How do VMMs schedule CPU use when guests believe they have dedicated CPUs?
    - How can memory management work when many guests require large amounts of memory?

OS Component – CPU Scheduling

- Even single-CPU systems act like multiprocessor ones when virtualized
  - One or more virtual CPUs per guest
- Generally VMM has one or more physical CPUs and number of threads to run on them
  - Guests configured with certain number of VCPUs
    - Can be adjusted throughout life of VM
  - When enough CPUs for all guests -> VMM can allocate dedicated CPUs, each guest much like native operating system managing its CPUs
  - Usually not enough CPUs -> CPU **overcommitment**
    - VMM can use standard scheduling algorithms to put threads on CPUs
    - Some add fairness aspect

OS Component – CPU Scheduling (Cont.)

- Cycle stealing by VMM and oversubscription of CPUs means guests don't get CPU cycles they expect
  - Consider timesharing scheduler in a guest trying to schedule 100ms time slices -> each may take 100ms, 1 second, or longer
    - Poor response times for users of guest
    - Time-of-day clocks incorrect
  - Some VMMs provide application to run in each guest to fix time-of-day and provide other integration features

https://www.vmware.com/files/pdf/techpaper/Timekeeping-In-VirtualMachines.pdf

OS Component – Memory Management

- Also suffers from oversubscription -> requires extra management efficiency from VMM
- For example, VMware ESX guests have a configured amount of physical memory, then ESX uses 3 methods of memory management
  1. Double-paging, in which the guest page table indicates a page is in a physical frame but the VMM moves some of those pages to backing store
  2. Install a **pseudo-device driver** in each guest (it looks like a device driver to the guest kernel but really just adds kernel-mode code to the guest)
     - **Balloon** memory manager communicates with VMM and is told to allocate or de-allocate memory to decrease or increase physical memory use of guest, causing guest OS to free or have more memory available
  3. De-duplication by VMM determining if same page loaded more than once, memory mapping the same page into multiple guests

OS Component – I/O

- Easier for VMMs to integrate with guests because I/O has lots of variation
  - Already somewhat segregated / flexible via device drivers
  - VMM can provide new devices and device drivers
- But overall I/O is complicated for VMMs
  - Many short paths for I/O in standard OSes for improved performance
  - Less hypervisor needs to do for I/O for guests, the better
  - Possibilities include direct device access, DMA pass-through, direct interrupt delivery
    - Again, HW support needed for these
- Networking also complex as VMM and guests all need network access
  - VMM can **bridge** guest to network (allowing direct access)
  - And / or provide **network address translation** (**NAT**)
    - NAT address local to machine on which guest is running, VMM provides address translation to guest to hide its address

OS Component – Storage Management

- Both boot disk and general data access need be provided by VMM
- Need to support potentially dozens of guests per VMM (so standard disk partitioning not sufficient)
- Type 1 – storage guest root disks and config information within file system provided by VMM as a **disk image**
- Type 2 – store as files in file system provided by host OS
- Duplicate file -> create new guest
- Move file to another system -> move guest
- **Physical-to-virtual** (**P-to-V**) convert native disk blocks into VMM format
- **Virtual-to-physical** (**V-to-P**) convert from virtual format to native or disk format
- VMM also needs to provide access to network attached storage (just networking) and other disk images, disk partitions, disks, etc.

OS Component – Live Migration

- Taking advantage of VMM features leads to new functionality not found on general operating systems such as live migration
- Running guest can be moved between systems, without interrupting user access to the guest or its apps
- Very useful for resource management, maintenance downtime windows, etc.
  1. The source VMM establishes a connection with the target VMM
  2. The target creates a new guest by creating a new VCPU, etc.
  3. The source sends all read-only guest memory pages to the target
  4. The source sends all read-write pages to the target, marking them as clean
  5. The source repeats step 4, as during that step some pages were probably modified by the guest and are now dirty
  6. When cycle of steps 4 and 5 becomes very short, source VMM freezes guest, sends VCPU's final state, sends other state details, sends final dirty pages, and tells target to start running the guest
     - Once target acknowledges that guest running, source terminates guest

Live Migration of Guest Between Servers