CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

Virtual Machines Building Blocks Summary

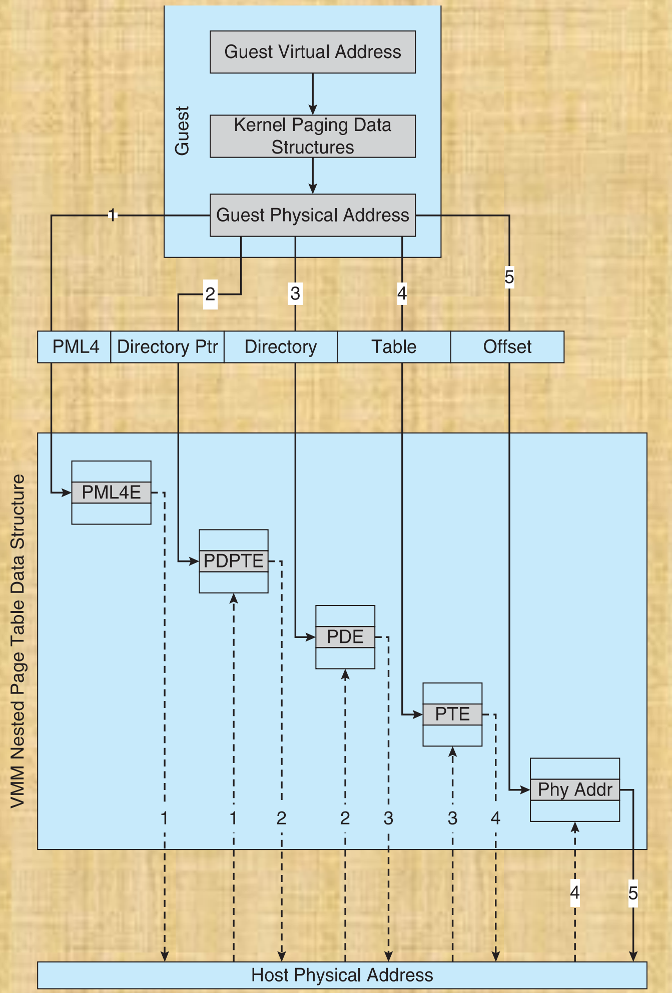
Ch 18.4-18.6

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

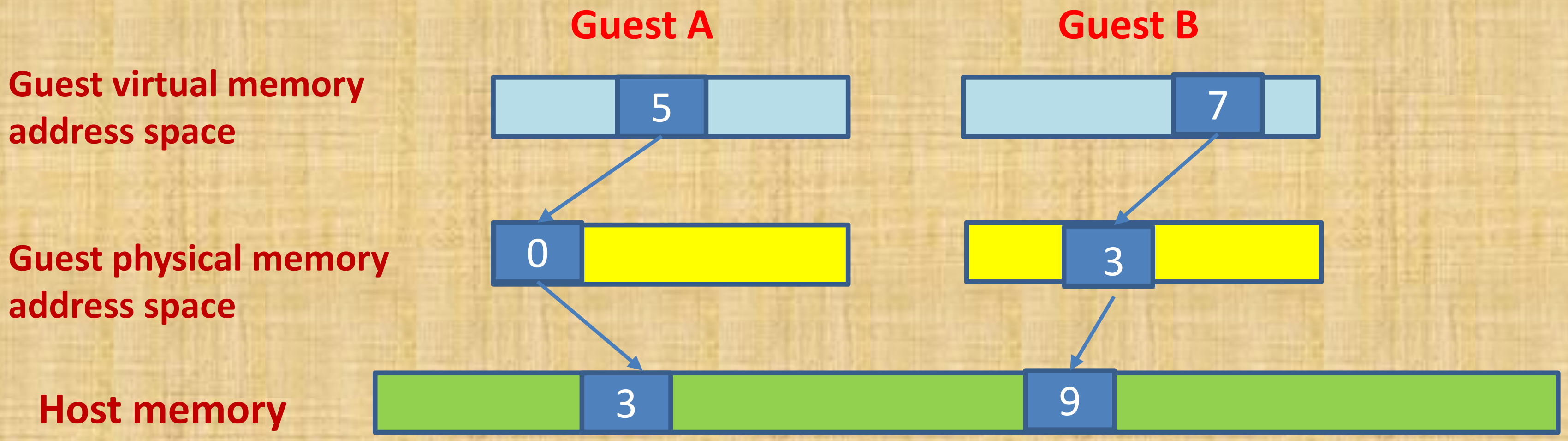
A total of 48 bit virtual address space.

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

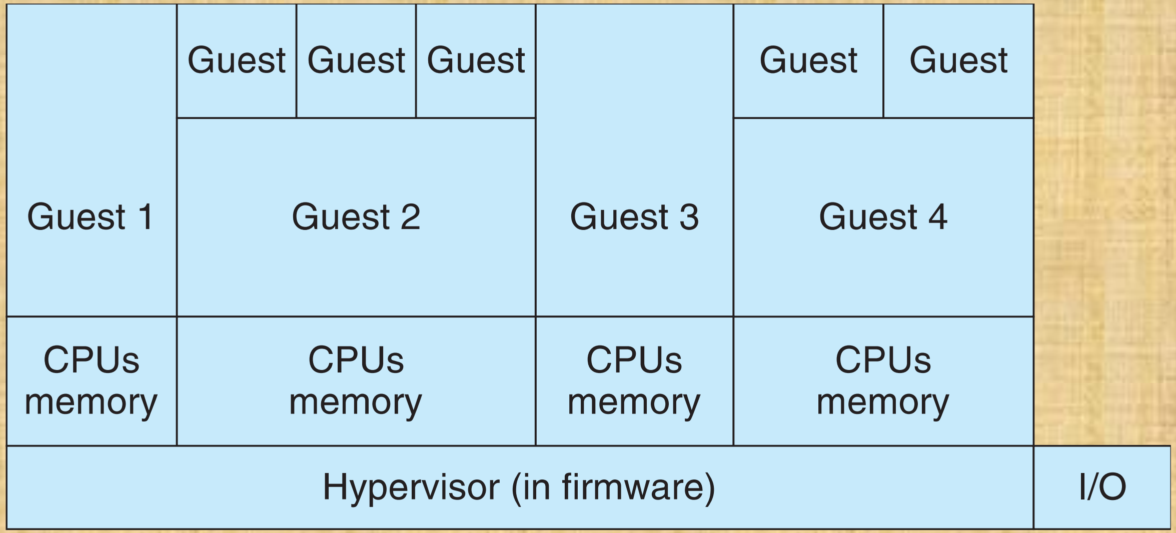
Example implantation with 4 K page size.

A Simplified, Logical View

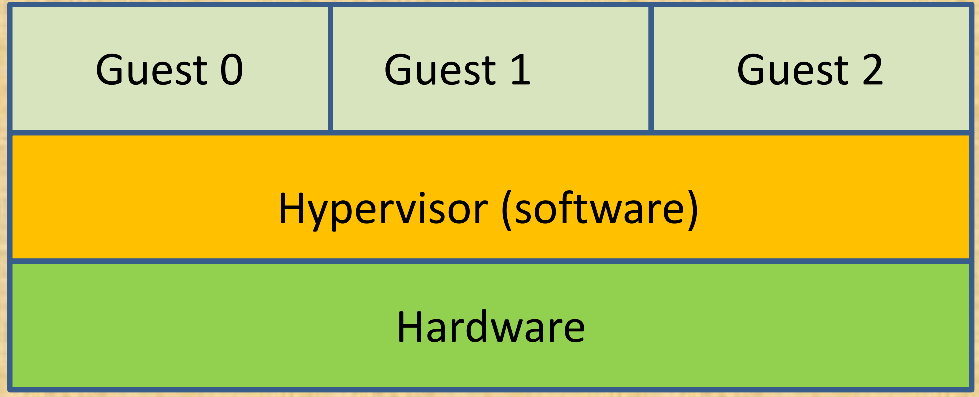
The actual AMD implementation shown in previous page, which is representative, looks complicated. But the idea isn't. Here is a simplified view.
https://www.cs.cmu.edu/~dga/15-440/F11/lectures/vm-ucsd.pdf



Type 0 Hypervisor

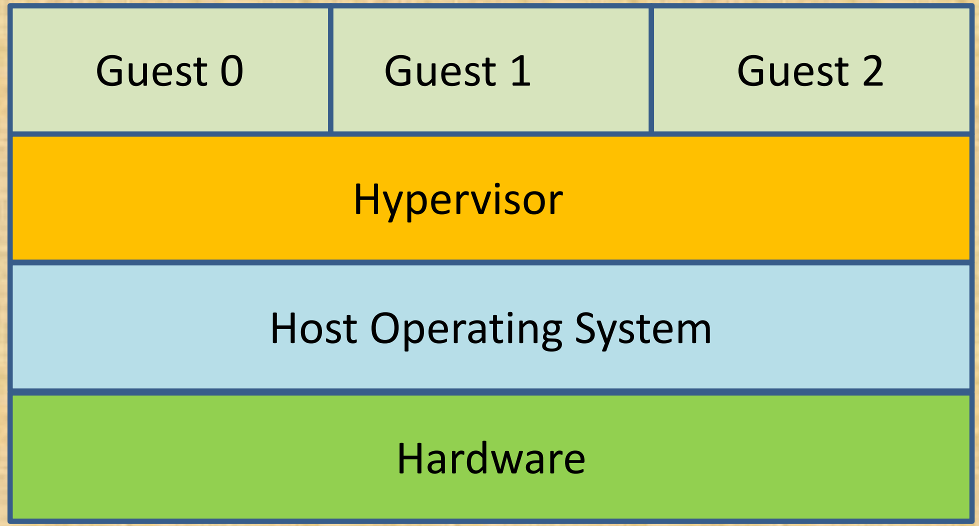


Type 1 Hypervisor



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

Type 2 Hypervisor



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

Types of VMs – Paravirtualization

- **Paravirtualization**: guest OS is aware of the virtualization and revise itself to make more efficient use of the hardware resources.
- 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

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

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.