# Virtualization

## Virtualization Overview

1. Virtualization allows concurrent execution of multiple OSs (and their applications) on the same physical machine
    - Invented at IBM in the 1960s
2. Virtual resources: Each OS thinks that it "owns" hardware resources
3. Virtual machine: OS + applications + virtual resources (guest domain)
4. Virtualization layer: Management of physical hardware (virtual machine monitor, hypervisor)

## Defining Virtualization

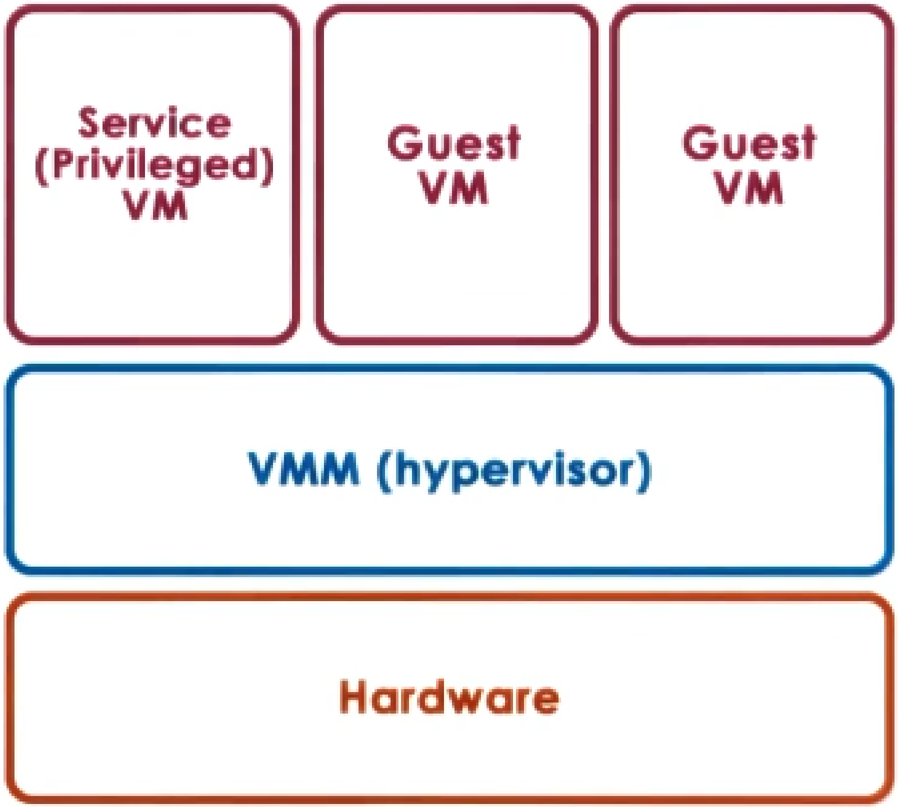
1. Virtual machine: An efficient, isolated duplicate of the real machine
2. Supported by a virtual machine monitor (VMM)
    - Provides environment essentially identical with the original machine
    - Programs show at worst only minor decrease in speed
    - VMM is in complete control of system resources
3. VMM goals: Fidelity, Performance, Safety/Isolation

## Benefits of Virtualization

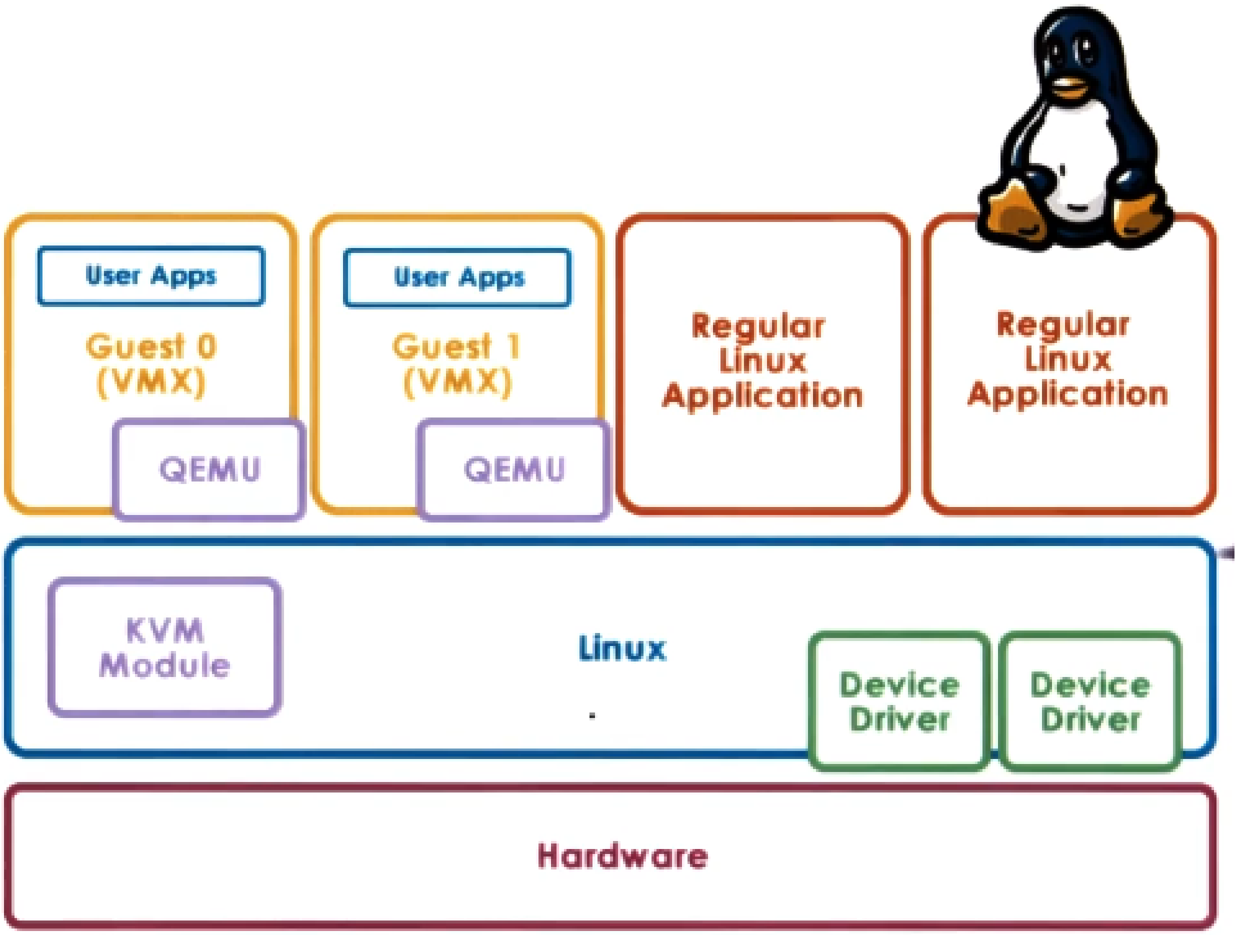
1. Consolidation: Can run multiple VMs with separate OS and applications on a single physical platform
    - Decrease cost, improve manageability
2. Migration: Move OS/applications from one physical machine to another
    - Availability, reliability
3. Security: Malicious behavior is contained to one instance
4. Debugging: Can quickly introduce new OS feature and test it
5. Support for legacy OSs

## Virtualization Models

1. Bare Metal/Hypervisor (type 1)
    - VMM (hypervisor) manages all hardware resources and supports execution of VMs
    - Priveleged, service VM to deal with devices (and other configuration and management tasks)
    - Xen (open source of Citrix XenServer)
        - VMs are referred to as domains
        - dom0 is priveleged domain
        - domUs is guest domains
        - Drivers in dom0
    - ESX (VMware)
        - Many open APIs
        - Drivers in VMM
        - Used to have Linux control core, now remote APIs
2. Hosted (type 2)
    - Host OS owns all hardware
    - Special VMM module provides hardware interfaces to VMs and deals with VM context switching
    - KVM (kernel-based VM)
        - Based on Linux
        - KVM kernel module + QEMU for hardware virtualization
        - Leverages Linux open-source community




Bare Metal Virtualization Model




Hosted Virtualization Model

### Hardware Protection Levels

1. Commodity hardware has more than two protection levels
    - x86 has 4 protection levels (rings)
        - ring 0: Highest privelege (OS)
        - ring 3: Lowest privelege (applications)
    - For virtualization:
        - ring 0: Hypervisor
        - ring 1: OS
        - ring 3: Applications
    - x86 also has 2 protection modes
        - Non-root: VMs (ring 0: OS, ring 3: applications)
        - Root: (ring 0: hypervisor)
        - VMexit: Trap to root mode
        - VMentry: Return to non-root mode

### Processor Virtualization

1. Guest instructions
    - Executed directly by hardware (VMM doesn't interfere)
    - For non-priveleged operations: Hardware speeds -> efficiency

- For priveleged operations: Trap to hypervisor
- Hypervisor determines what needs to be done
  - If illegal operration: Terminate VM
  - If legal operation: Emulate the behavior the guest OS was expecting from the hardware

2. Called trap-and-emulate; key component in achieving efficiency

## x86 Virtualization in the Past

1. Problems with Trap-and-Emulate (x86 pre-2005)
   - 4 rings, no root/non-root modes yet
   - Hypervisor in ring0, guest OS in ring1
   - 17 priveleged instructions do not trap! Fail silently!
     - Interrupt enable/disable bit in priveleged register; POPF/PUSHF instructions that access it from ring1 fail silently
     - Hypervisor doesn't know, so it doesn't try to change settings
     - OS doesn't know, so it assumes the change was successful

## Binary Translation

1. Main idea: Rewrite the VM binary to never issue those 17 instructions
   - Pioneered by Mendel Rosenblum's group at Stanford, commercialized as VMware (received ACM fellow for "reinventing virtualization")
2. Binary translation:
   - Goal: Full virtualization == guest OS is not modified
   - Approach: Dynamic binary translation
     - Inspect code blocks to be executed
     - If needed, translate to alternate instruction sequence (e.g., to emulate desired behavior, possibly even avoiding trap)
     - Otherwise, run at hardware speeds and cache translated blocks to amortize translation costs

## Paravirtualization

1. Goal: Performance, but give up on running unmodified guests
2. Approach: Modify guest so that it...
   - knows it's running virtualized
   - makes explicit calls to the hypervisor (hypercalls)
3. Hypercalls are analogous to system calls
   - Package context information
   - Specify desired hypercall
   - Trap to VMM
4. Xen: Open source hypervisor (XenSource -> Citrix)

## Full Memory Virtualization

1. Full virtualization
   - All guests expect contiguous physical memory, starting at 0
   - Virtual vs physical vs machine addresses and page frame numbers
   - Still leverages hardware MMU, TLB
2. Option 1
   - Essentially two page tables; One for OS, one for hypervisor
   - Guest page table: VA -> PA (software)
   - Hypervisor: PA -> MA (hardware)
   - Too expensive!
3. Option 2
   - Guest page table: VA -> PA

- Hypervisor shadow PT: VA -> MA
- Hypervisor maintains consistence
  - e.g., invalidate on context switch, write-protect guest page table to track new mappings

## Paravirtualized Memory Virtualization

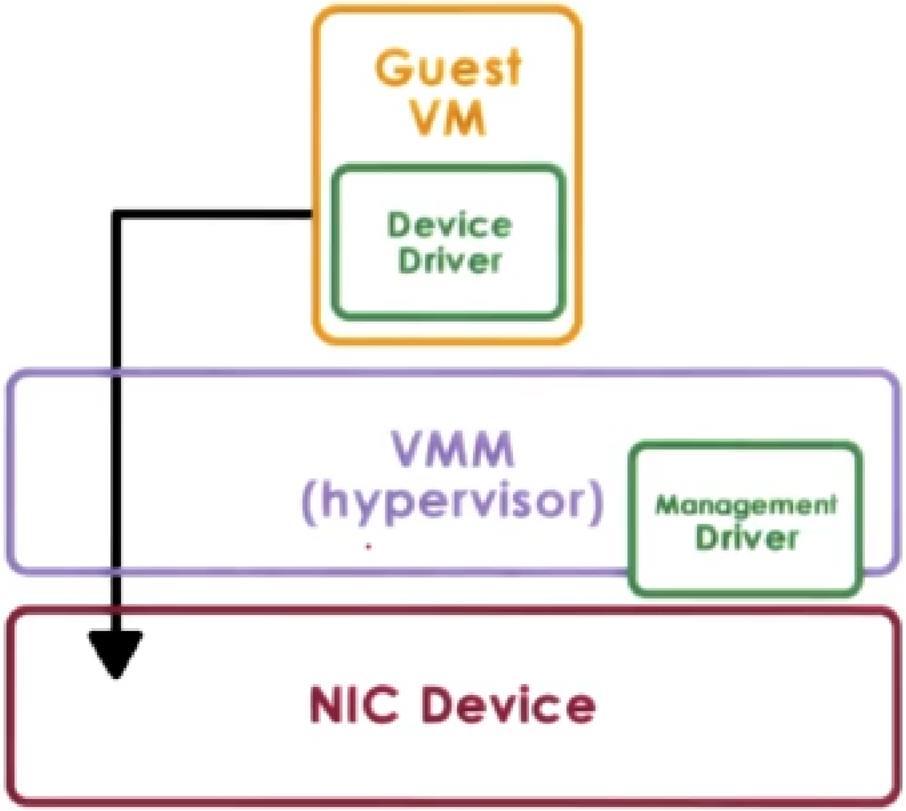
1. Paravirtualized
   - Guest is aware of virtualization
   - No longer strict requirement on contiguous physical memory starting at 0
   - Explicitly registers page tables with hypervisors
   - Can "batch" page table updates to reduce VM exits and other optimizations
2. Overheads are eliminated or reduced on newer platforms

## Device Virtualization

1. For CPUs and memory:
   - Less diversity at the ISA-level ("standardization" of interface)
2. For devices:
   - High diversity
   - Lack of standard specification of device interface and behavior
3. 3 Key Models for device virtualization (pre-virtualization HW extensions)

## Passthrough Model

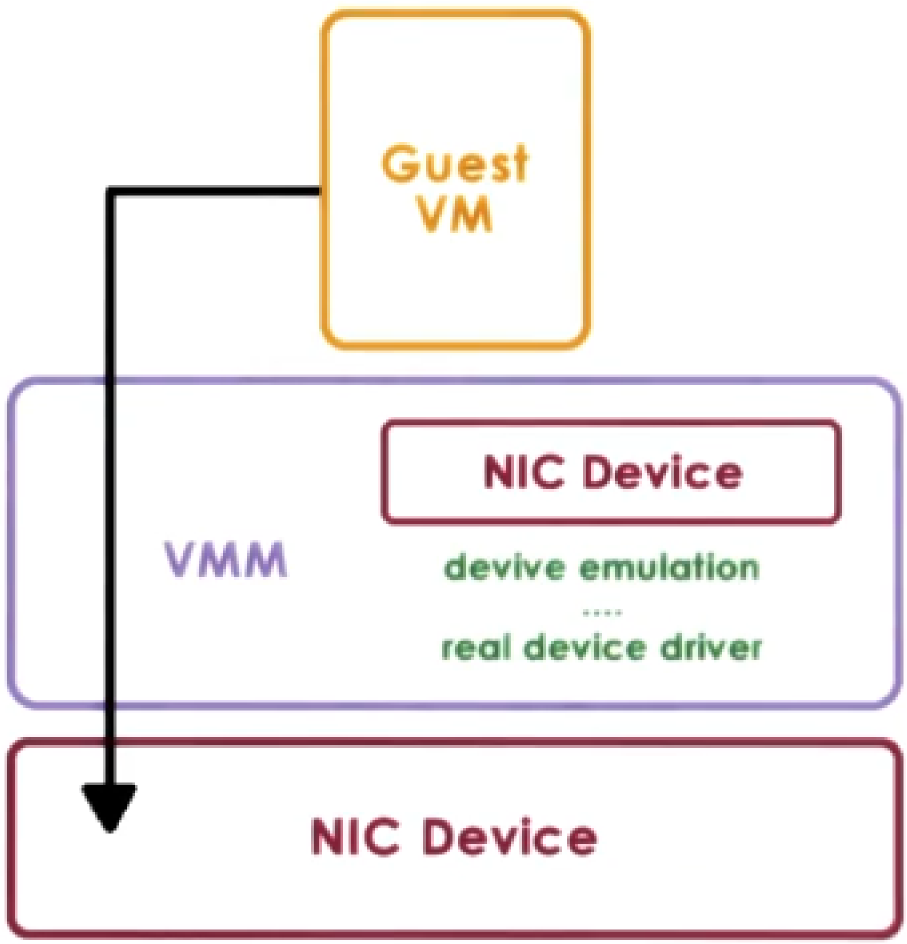
1. Approach: VMM-level driver configures device access permissions
2. Pros:
   - VM provided with exclusive access to the device
   - VM can directly access the device (VMM-bypass)
3. Cons:
   - Device sharing is difficult
   - VMM must have exact type of device as what VM expects
   - VM migration becomes more difficult




Hosted Virtualization Model

### Hypervisor-Direct Model

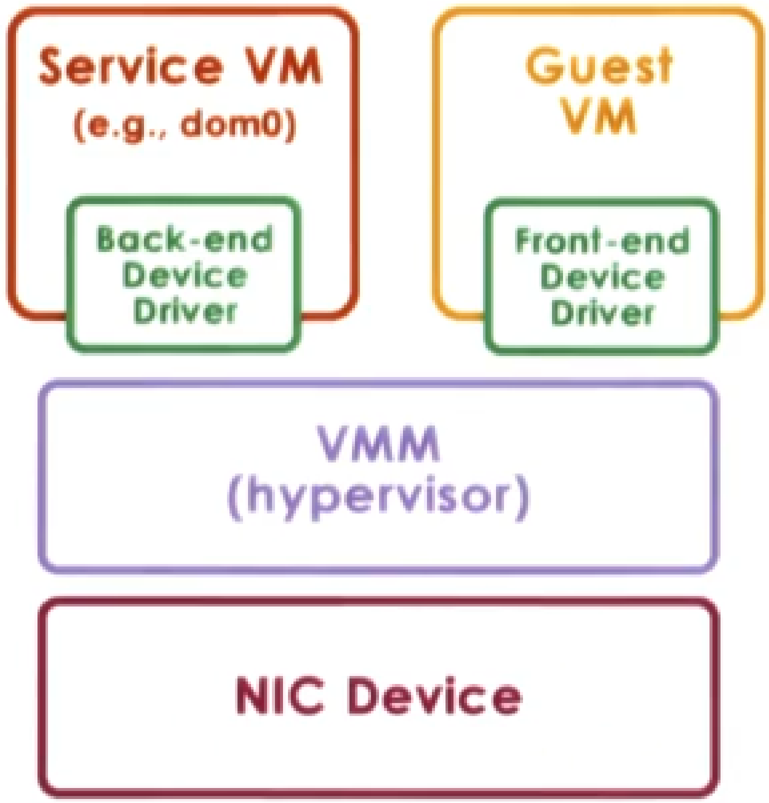
1. Used by VMware ESX
2. Approach: VMM intercepts all device accesses
   - Emulate device operation:
     - Translate to generic I/O operation
     - Traverse VMM-resident I/O stack
     - Invoke VMM-resident driver
3. Pros:
   - VM decoupled from physical device
   - Sharing, migration, dealing with device specifics all become simpler
4. Cons:
   - Latency of device operations
   - Device driver ecosystem complexities in hypervisor




Hypervisor-Direct Virtualization Model

### Split-Device Driver Model

1. Approach: Device access control split between...
    - Front-end driver in guest VM (device API)
    - Back-end driver in service VM (or host)
    - Modified guest drivers to interact with back-end
2. Pros:
    - Eliminate emulation overhead
    - Allow for better management of shared devices
3. Cons:
    - Limited to paravirtualized guests

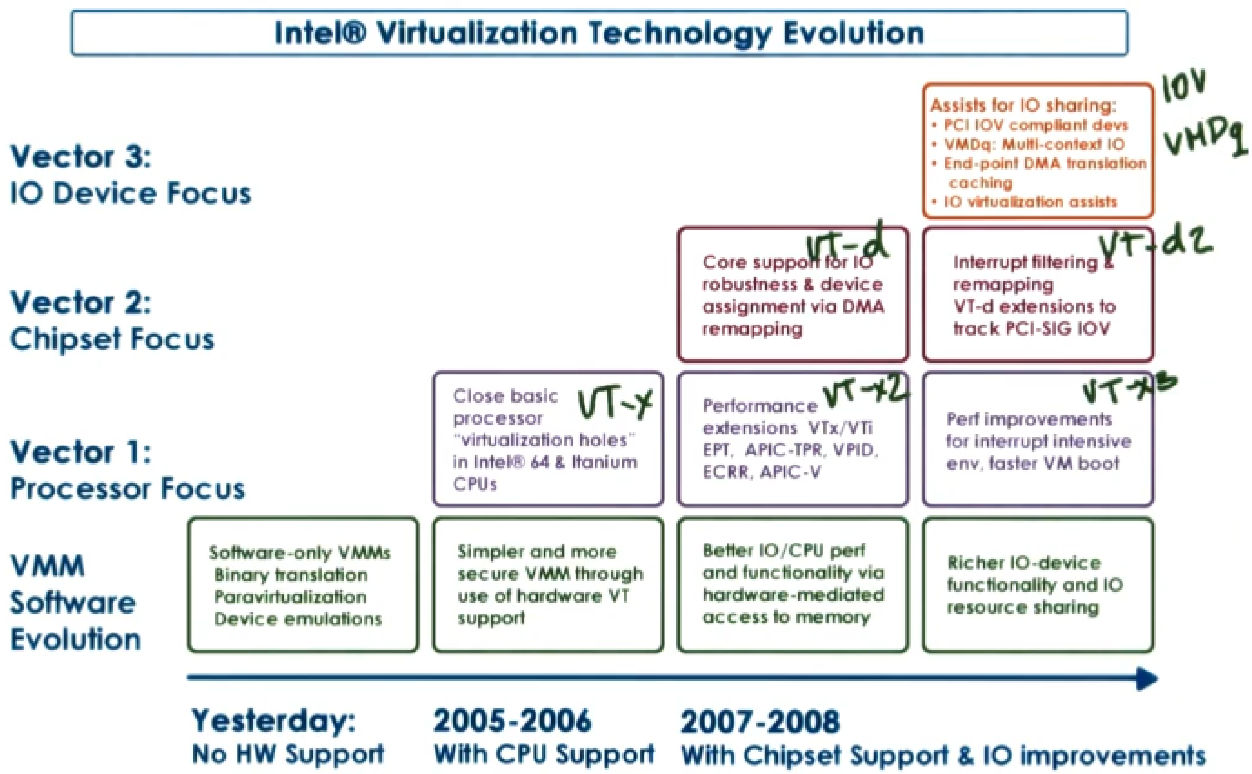



Split-Device Virtualization Model

## Hardware Virtualization

1. AMD Pacifica & Intel Vanderpool Technology (Intel-VT) circa 2005
   - Close holes in x86 ISA
   - Modes: root/non-root (or 'host' and 'guest' mode)
   - VM Control Structure
     - Per vCPU; 'walked' by hardware (can specify whether a system call should trap or not)
   - Extended page tables and tagged TLB with VM IDs
     - Context switch between VMs ("world switch") doesn't have to flush TLB; MMU can check if the address is valid for this VM
     - Context switches are much more efficient
   - Multiqueue devices and interrupt routing
     - Can deliver an interrupt to a specific VM
   - Security and management support
     - Protect VMs from each other
2. Added new instructions to x86 to exercise the above features
   - Manipulate state in VM control data structure

## x86 Virtualization Technology (VT) Revolution




Virtualization Revolution