Virtualization

Dr. Yingwu Zhu

What is virtualization?

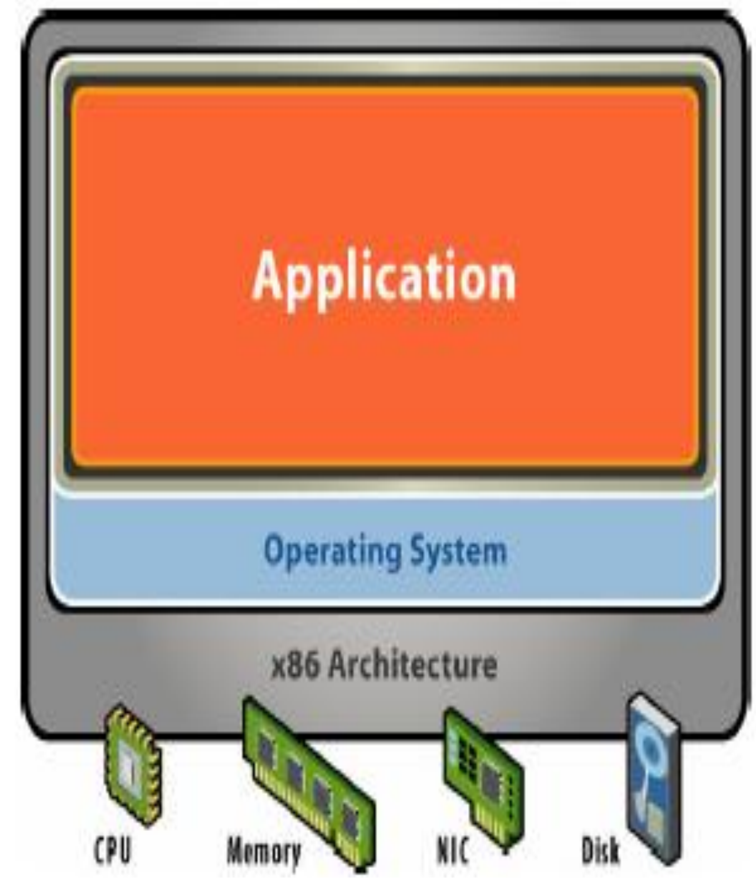
- Virtualization allows one computer to do the job of multiple computers.
- Virtual environments let one computer host multiple operating systems at the same time

Virtualization

- Definition
  - Framework or methodology of dividing the resources of a computer into multiple execution environments.
- Types
  - Platform Virtualization: Simulate a full computer environment (Our current concern).
  - Resource Virtualization: Simulate combined, fragmented or simplified computer resources (RAID, NAT, VPN, ...).

OS vs. Physical Machines

- Physical Hardware
  - Processors, memory, chipset, I/O bus and devices, etc.
  - Physical resources often underutilized
- Software
  - Tightly coupled to hardware
  - Single active OS image
  - OS controls hardware



Basic Concepts

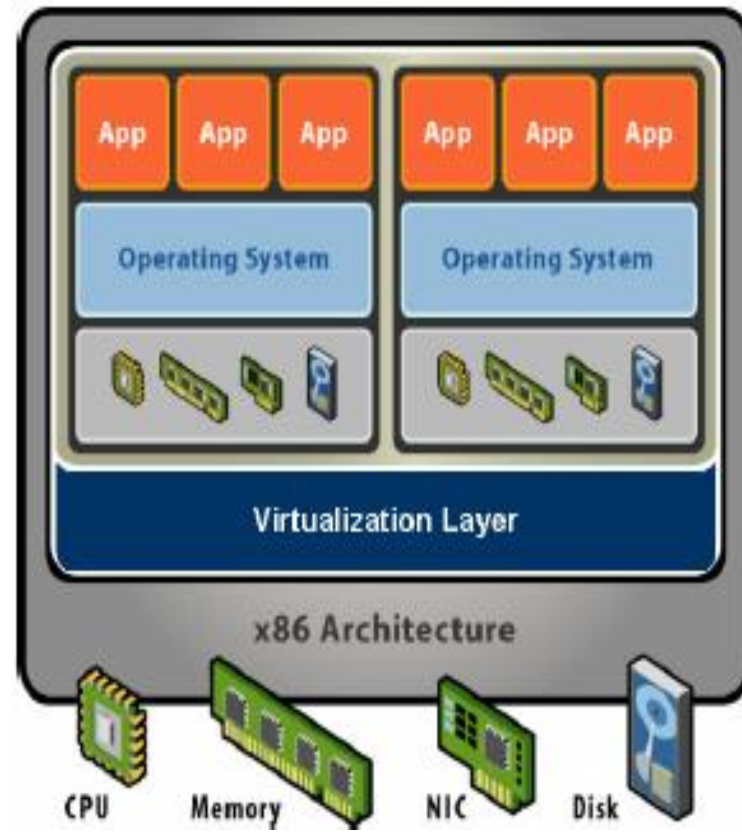
- Traditional OS implements a virtual machine
  - OS gives processes virtual memory
  - Each process runs on a virtualized CPU
- Virtual machine
  - “A virtual machine (VM) is a software implementation of a machine (i.e. a computer) that executes instructions like a physical machine.”
- Virtualization is not new: IBM was providing virtual versions of hardware platforms in the 1960s

How does virtualization work?

- Virtualization transforms hardware into software.
- It is the creation of a fully functional virtual computer that can run its own applications and operating system.
- Creates virtual elements of the CPU, RAM, and hard disk.

What is a virtual machine?

- Hardware-Level Abstraction
  - Virtual hardware: processors, memory, chipset, I/O devices, etc.
  - Encapsulates all OS and application state
- Virtualization Software
  - Extra level of indirection decouples hardware and OS
  - Multiplexes physical hardware across multiple “guest” VMs
  - Strong isolation between VMs
  - Manages physical resources, improves utilization

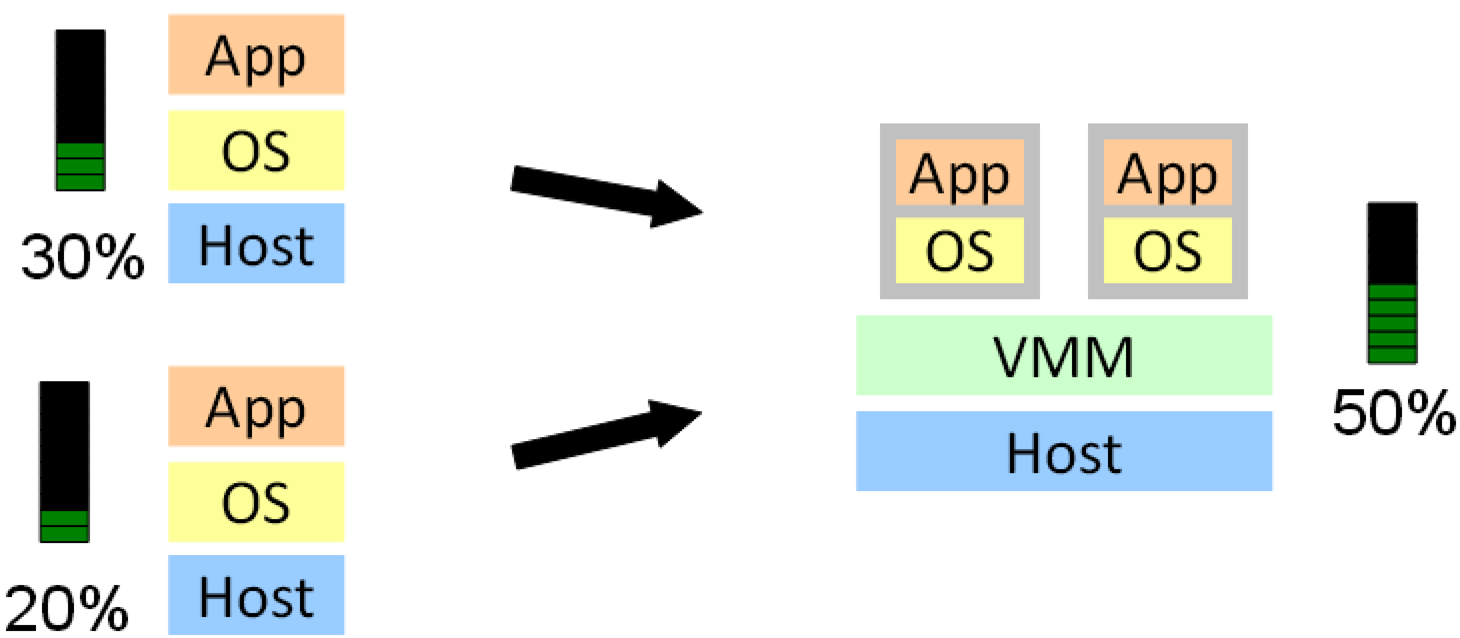


Why VM?

- Manage big machines
  - Multiplex CPUs/memory/devices at VM granularity
  - E.g., Amazon EC2
- Multiple OS on one machine
  - E.g., use Windows on Linux OS
- Isolate faults/break-ins
  - One VM is compromised/crashes, others OK
- Kernel development
  - Like QEMU, but faster
- OS granularity checkpoint/record/replay

Why VM?

- Server Consolidation
  - Underutilized physical servers
  - Consolidate to improve utilization / lower cost
    - Also simplifies management of OS and system configs



VM isolation

- Secure Multiplexing
  - Run multiple VMs on single physical host
  - Processor hardware isolates VMs, *e.g. MMU*
- Strong Guarantees
  - Software bugs, crashes, viruses within one VM cannot affect other VMs
- Performance Isolation
  - Partition system resources
  - Example: VMware controls for reservation, limit, shares

VM encapsulation

- Entire VM is a File, capturing all of the state
  - OS, applications, data
  - Memory and device state
- Snapshots and Clones
  - Capture VM state on the fly and restore to point-in-time
  - Rapid system provisioning, backup, remote mirroring
- Easy Content Distribution
  - Virtual appliances :Pre-configured VM with OS/apps pre-installed
  - Just download and run (no need to install/configure)
  - Software distribution using appliances

VM compatibility

- Hardware-Independent
  - Physical hardware hidden by virtualization layer
  - Standard virtual hardware exposed to VM
- Create Once, Run Anywhere
  - No configuration issues
  - Migrate VMs between hosts
- Legacy VMs
  - Run ancient OS on new platform
  - *E.g. DOS VM drives virtual IDE* and vLance devices, mapped to modern SAN and GigE hardware

Virtual machine monitor (VMM)

- A piece of software, also called Hypervisor
- Characteristics
  - Fidelity: provides an environment for programs which is essentially identical with the original machine
  - Performance: programs run in this environment show at worst minor decreases in speed
  - Isolation/safety: the VMM is in complete control of system resources

VMM responsibilities

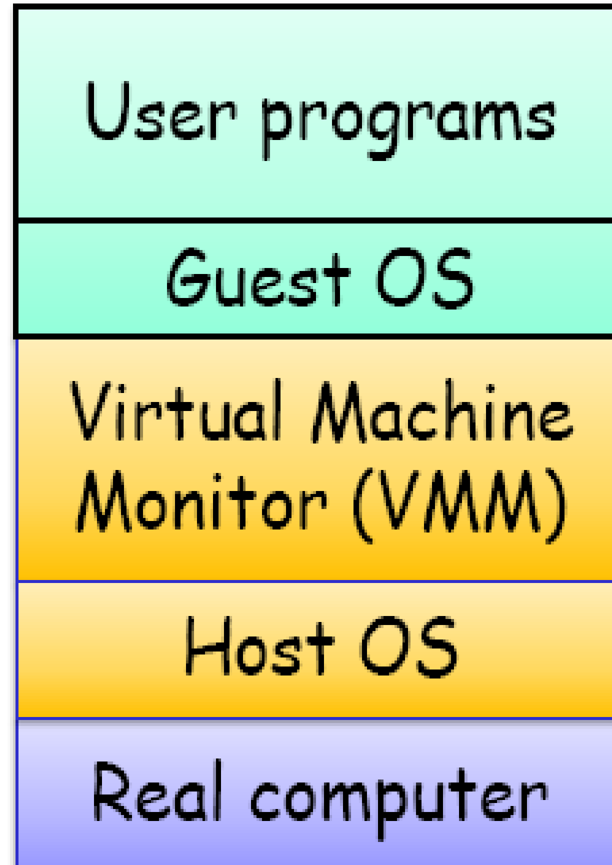
- VMM must fully control CPUs, memory, I/O devices
- Ways hat a VMM can share resources between VMs
  - Time-share CPU among guests
  - Space-share (partition) memory among guests
  - Simulate disk, network, and other devices
    - Often multiplex on host devices

VMM platform types

- Hosted Architecture
- Bare-Metal Architecture

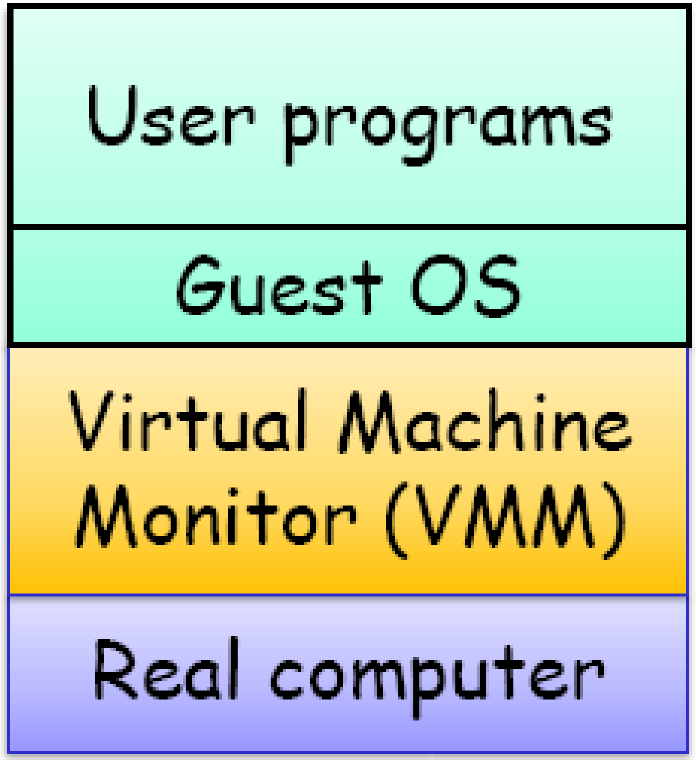
VMM platform types

- Hosted Architecture
  - Install as application on existing x86 "host" OS, *e.g. Windows, Linux, OS X*
    - Leverages device drivers and services of a "host OS"
  - Small context-switching driver
  - Leverage host I/O stack and resource management
  - Examples: VMware Player/Workstation/Server, Microsoft Virtual PC/Server, Parallels Desktop



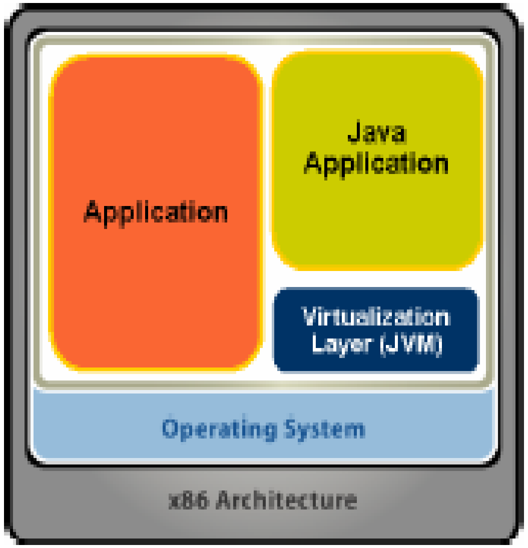
VMM platform types

- Bare-Metal Architecture
  - "Hypervisor" installs directly on hardware
    - provides its own device drivers and services
  - Acknowledged as preferred architecture for high-end servers
  - Examples: VMware ESX Server, Xen, Microsoft Viridian (2008)

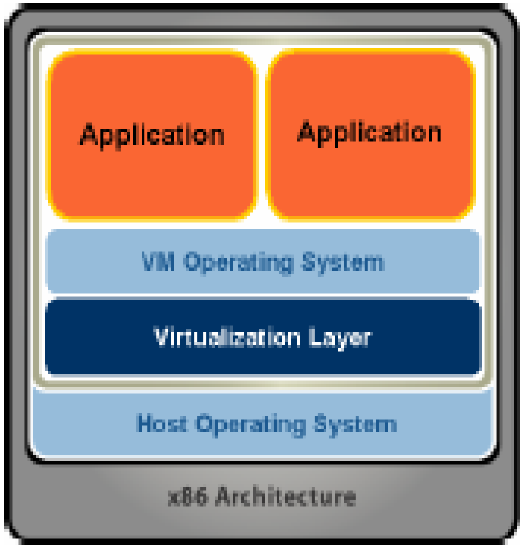


System virtualization alternatives

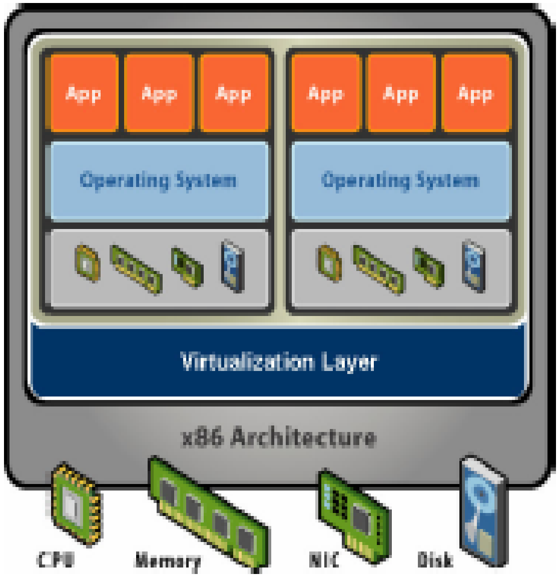
- Virtual machines abstracted using a layer at different places



Language Level

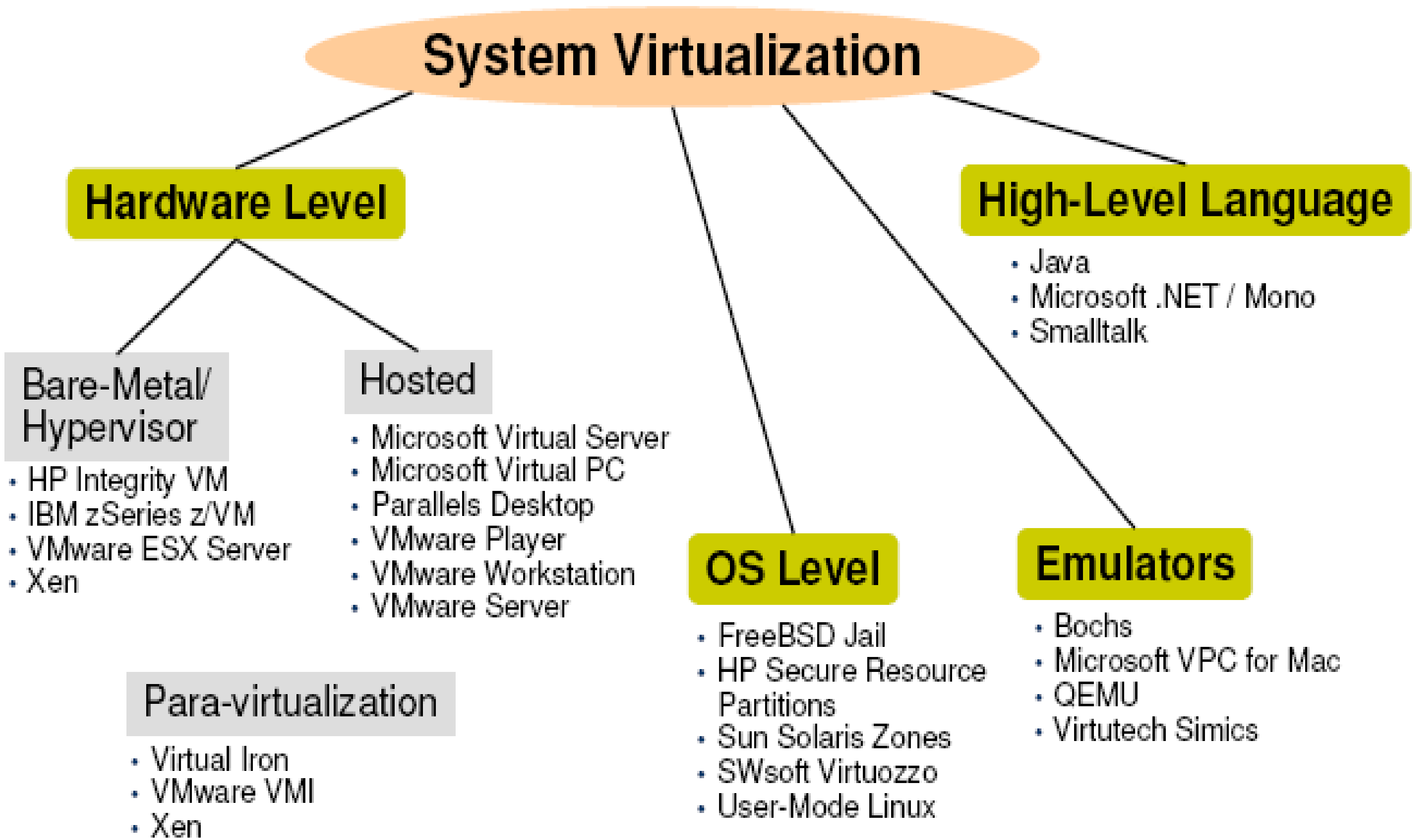


OS Level



Hardware Level

System virtualization taxonomy



Naïve approach: simulation

```
int32_t regs[8];
#define REG_EAX 1;
#define REG_EBX 2;
#define REG_ECX 3;
...
int32_t eip;
int16_t segregs[4];
...
```

```
for (;;) {
        read_instruction();
        switch (decode_instruction_opcode()) {
        case OPCODE_ADD:
                int src = decode_src_reg();
                int dst = decode_dst_reg();
                regs[dst] = regs[dst] + regs[src];
                break;
        case OPCODE_SUB:
                int src = decode_src_reg();
                int dst = decode_dst_reg();
                regs[dst] = regs[dst] - regs[src];
                break;
        ...
        }
        eip += instruction_length;
}
```

- Interpret each guest instruction
- Maintain each VM state purely in software
- Problem: too slow!

VMM implementation

- Should efficiently virtualize the hardware
  - Provide illusion of multiple machines
  - Retain control of the physical machine
- Subsystems
  - Processor Virtualization
  - Memory Virtualization
  - I/O virtualization

Processor virtualization

Popek and Goldberg (1974)

- All instructions that can inspect and modify privileged machine state will trap when executed from any but the most privileged state
- CPU architecture virtualizable if it supports running VMs on real CPU (*direct execution*), and VMM retains real control of CPU

Classical instruction virtualization

- Trap and Emulate
  - Run guest operating system *deprivileged*
  - All privileged instructions *trap into VMM*
  - VMM emulates instructions against virtual state
  - *e.g. disable virtual interrupts, not physical interrupts*
  - Resume direct execution from next guest instruction
- Implementation Technique
  - This is just one technique
  - Popek and Goldberg criteria permit others

x86 Processor Virtualization

- x86 architecture is not fully *virtualizable*
  - The Intel IA-32 (x86, Pentium, ...) architecture did not support trapping of privileged instructions until the introduction of the Intel Core 2 Duo processor Techniques to address inability to virtualize x86
  - If a process not running in privileged mode attempted to execute a privileged instruction *nothing would happen*
  - We couldn't just run the operating system code as an unprivileged process and have a hypervisor trap and emulate the special instructions.

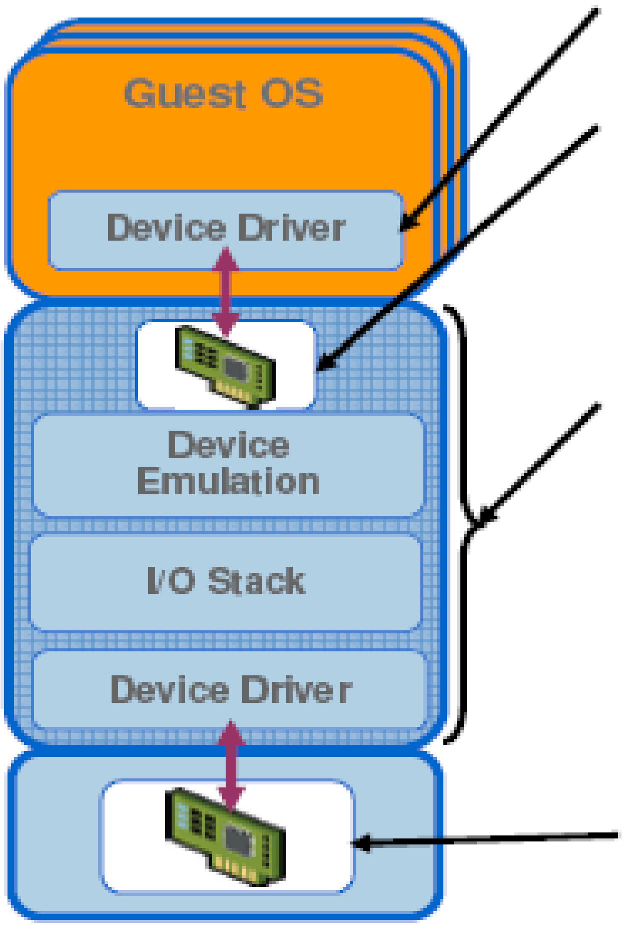
x86 Processor Virtualization

- Two approaches to dealing with the issue
  - Binary translation (by VMWare)
    - Pre-scan the instruction stream for code
    - Replace the privileged instructions with traps that VMM can trap
    - Code is executed at full speed: instructions are executed by the processor and not interpreted
  - Paravirtualization (by Xen)
    - Need to modify guest Oss not to use privileged instructions
    - Replace with API calls to the VMM which act like OS system calls, causing a trap and a context switch to the VMM
    - Yields higher performance than binary translation
    - But requires access to the kernel source code

I/O virtualization

- Issue: lots of I/O devices (richer & diverse), making virtualization challenging
- Problem: Writing device drivers for all I/O device in the VMM layer is not a feasible option
- Insight: Device driver already written for popular OSs
- Solution: Present *virtual I/O devices to guest VMs* and channel I/O requests to a trusted *host VM* running popular OS

I/O virtualization



Guest Device Driver

Virtual Device

- Model existing device, *e.g.* e1000
- Model an idealized device, *e.g.* vmxnet

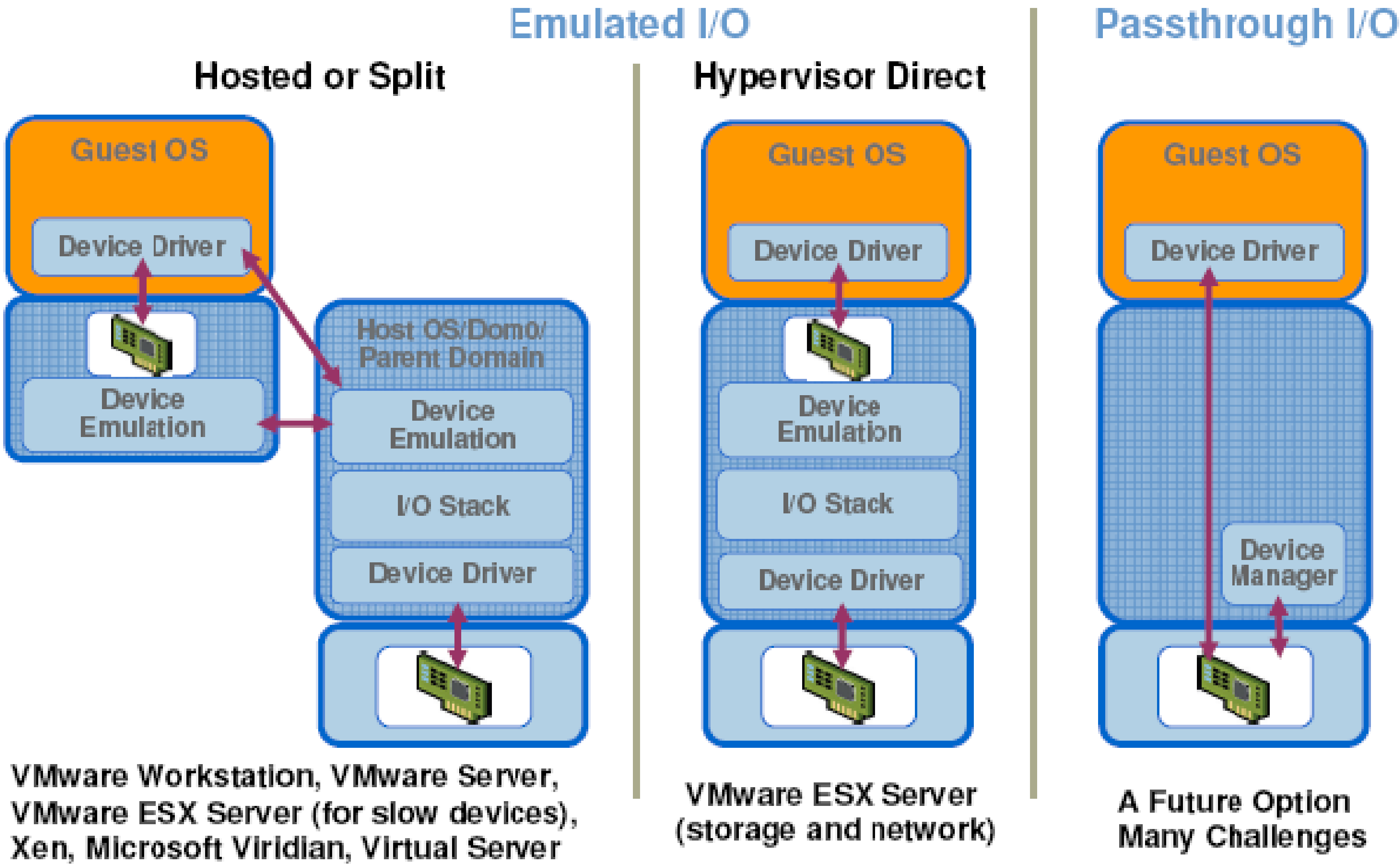
Virtualization Layer

- Emulates the virtual device
- Remaps guest and real I/O addresses
- Multiplexes and drives physical device
- Provides additional features, *e.g.* transparent NIC teaming

Real Device

- Physical hardware, *e.g.* bcm5700
- Likely to be different than virtual device

I/O virtualization



Eliminate I/O virt. overhead

Memory virtualization

- Traditional way is to have the VMM maintain a shadow of the VM’s page table
- The shadow page table controls which pages of machine memory are assigned to a given VM
- When OS updates it’s page table, VMM updates the shadow

Memory virtualization

- Desirable capabilities
  - Efficient memory overcommitment
  - Accurate resource controls
  - Exploit sharing opportunities
- Challenges
  - Allocations should reflect both importance and working set
  - Best data to guide decisions known only to guest OS
  - Guest and meta-level policies may clash

Memory virtualization: addr. translation

- Two levels of translation
  - Guest virtual addr. -> guest physical addr.
  - Guest physical addr. -> host physical addr.
- Who controls these mappings?
- Shadow page table
  - Guest OS will maintain its own virtual memory page table in the guest physical memory frames.
  - For each guest physical memory frame, VMM should map it to host physical memory frame.
  - Shadow page table maintains the mapping from guest virtual address to host physical address.

Example

Guest VA	Guest PA
0	20
1	30
2	10
3	40

Guest OS

Guest PA	Host PA
10	300
20	200
30	400
40	100

VMM

Guest VA	Host PA

Hardware translation (single level)

Case Study:

Virtualization on VMware ESX Server

VMware ESX Server

- Bare-metal VMM
  - Runs on bare hardware
- Full-virtualized
  - Legacy OS can run unmodified on top of ESX server
- Fully controls hardware resources and provides good performance

ESX Server – CPU Virtualization

- Most user code executes in Direct Execution mode;
  - near native performance
- Uses *runtime Binary Translation for x86* virtualization
  - Privileged mode code is run under control of a Binary Translator, which emulates problematic instructions
  - Fast compared to other binary translators as source and destination instruction sets are nearly identical

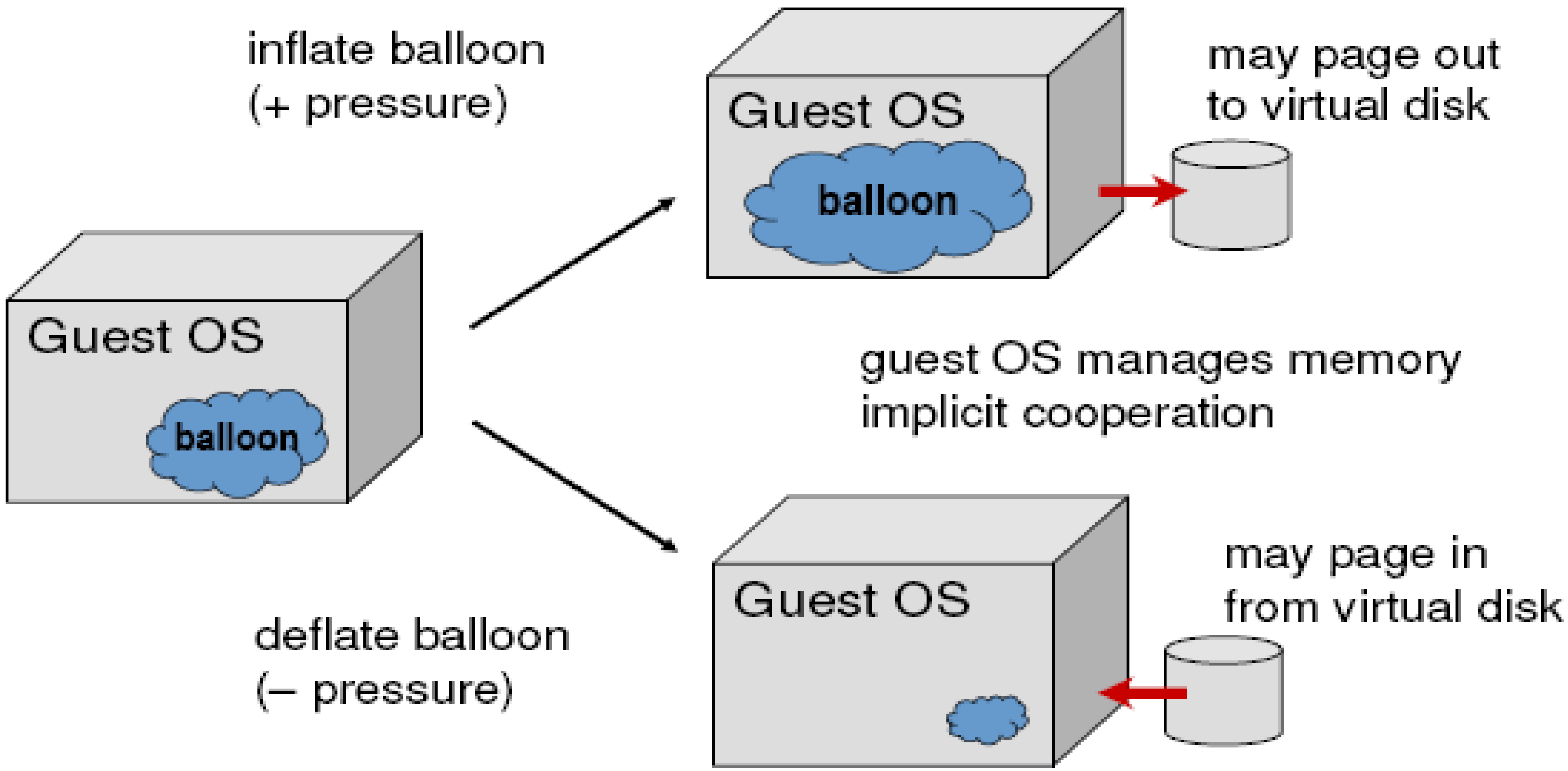
ESX Server – Memory Virtualization

- Maintains shadow page tables with virtual to machine address mappings.
- Shadow page tables are used by the physical processor
- ESX maintains the pmap data structure for each VM with “physical” to machine address mappings
- ESX can easily remap a machine page

ESX Server – Memory Management

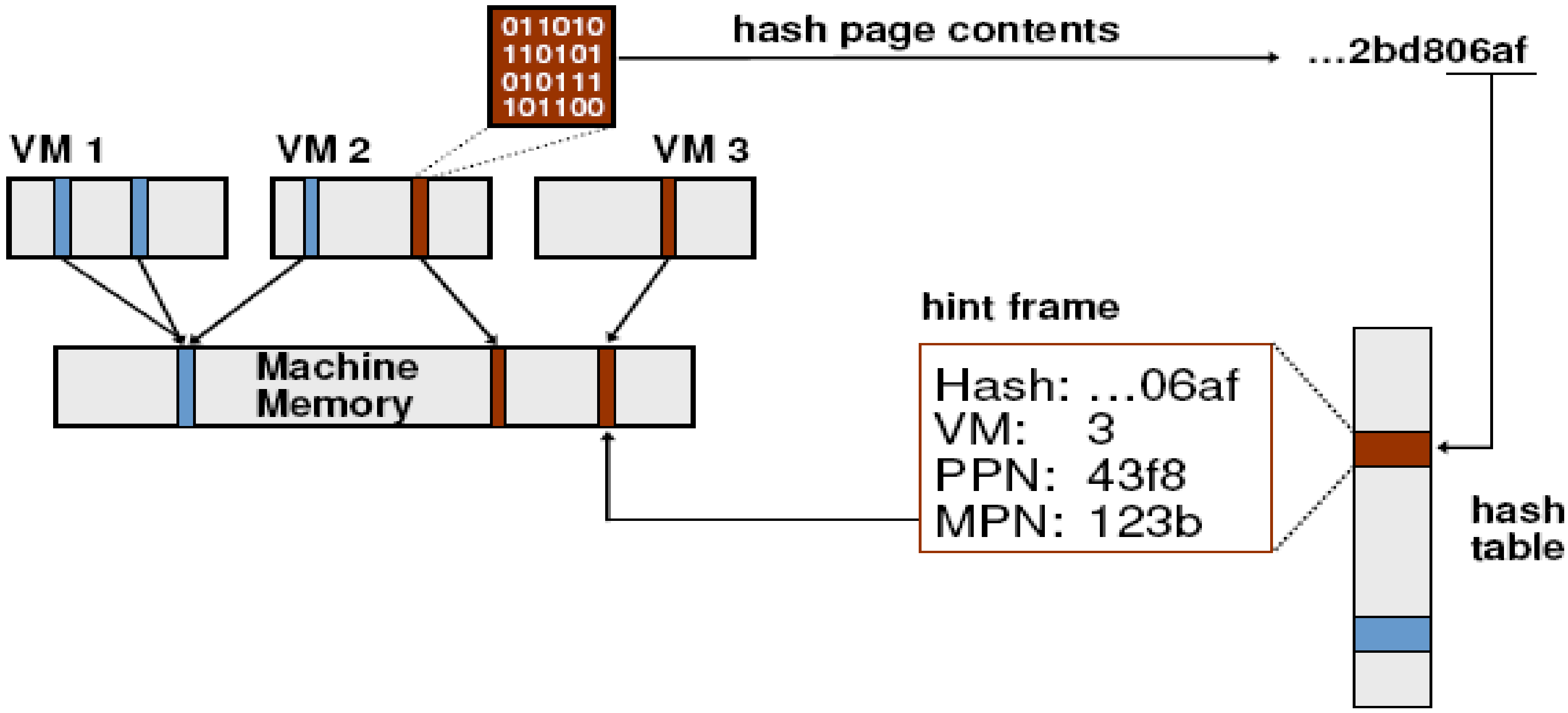
- Page reclamation – Ballooning technique
  - Reclaims memory from other VMs when memory is overcommitted
- Page sharing – Content based sharing
  - Eliminates redundancy and saves memory pages when VMs use same operating system and applications

ESX Server- Ballooning



1. A *balloon process running inside the* GuestOS can communicate with the VMM
2. When VMM wants to reclaims mem. from a given VM, asks its balloon process to allocate more memory, then the page repacement in GuestOS gives pages to the balloon process, which passes them to the VMM for reallocation

ESX Server – Page Sharing



the VMM tracks the contents of physical pages, noting if they are identical. If so, the VMM modifies the virtual machine's shadow page tables to point to only a single copy

ESX Server – I/O Virtualization

- Has highly optimized storage subsystem for networking and storage devices
  - Directly integrated into the VMM
  - Uses device drivers from the Linux kernel to talk directly to the device
- Low performance devices are channeled to special “host” VM, which runs a full Linux OS

Virtualization vs. Data Centers

What is data center?

- Large server & storage farms
  - Used by enterprises to run server applications
  - Used by Internet companies: Google, Facebook, Youtube, Amazon….
  - Sizes can vary depending on needs

Data center architecture

- Traditional: applications run on physical servers
  - Manual mapping of apps to servers
    - Apps can be distributed
    - Storage may be on a SAN or NAS
  - IT admins deal with “change”
- Modern: virtualized data centers
  - Apps run inside virtual servers; VMs mapped onto physical servers
  - Provides flexibility in mapping from virtual to physical resources

Virtualized data centers

- Resource management is simplied
  - Apps can be started from preconfigured VM images/appliances
  - Virtualization layer / hypervisor permits resource allocations to be varied dynamically
  - VMs can be migrated without app down-time

Workload management

- Internet apps ➔ dynamic workloads
- How much capacity to allocate to an app?
  - Incorrect workload estimate: over or under provisioned capacity
  - Major issue for Internet facing apps
    - Workload surges/ flash crowds cause overloads
    - Long-term incremental growth
  - Traditional approach: IT admins estimate peak workloads and provision sufficient

Dynamic provisioning

- Track workload and dynamically provision capacity
- Monitor -> Predict -> Provision
- Predictive vs. reactive provisioning
  - Predictive: predict future workload and provision
  - Reactive: react whenever capacity falls short of demand
- Traditional data centers: bring up a new server
  - Borrow from free pool or reclaim under-used server
- Virtualized data center: exploit virtualization