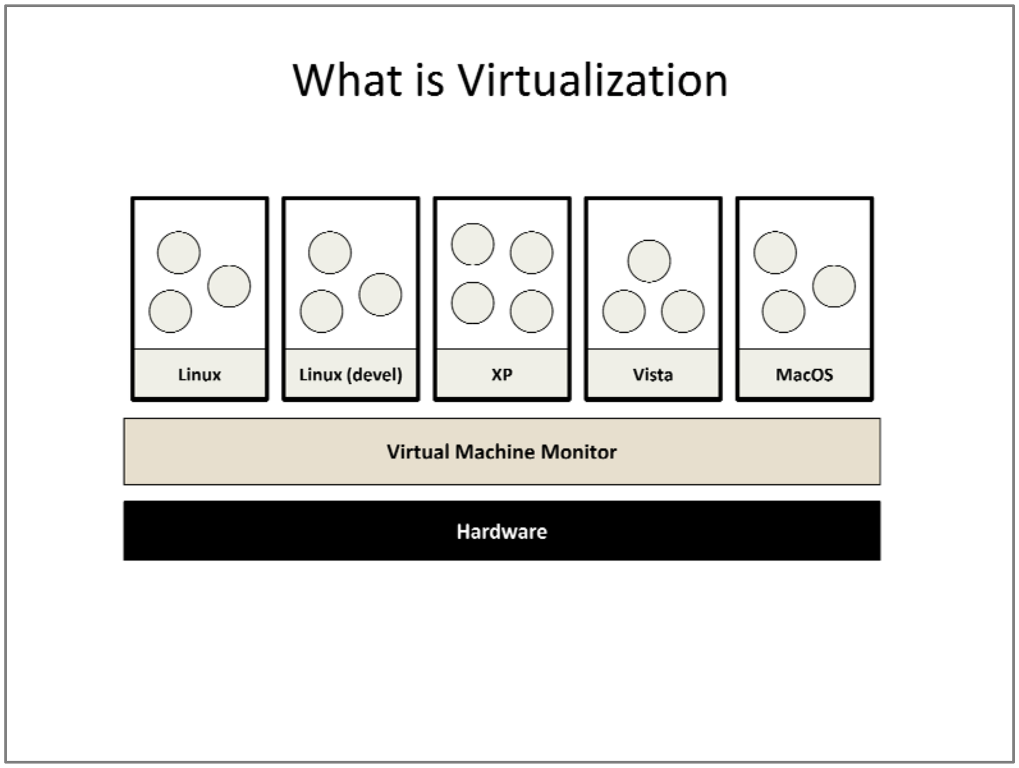
E6998 - Virtual Machines
Lecture 1
What is Virtualization?

Scott Devine
VMware, Inc.

Outline

- What is virtualization?
- Virtualization classification
- Monitor Architectures



What is Virtualization

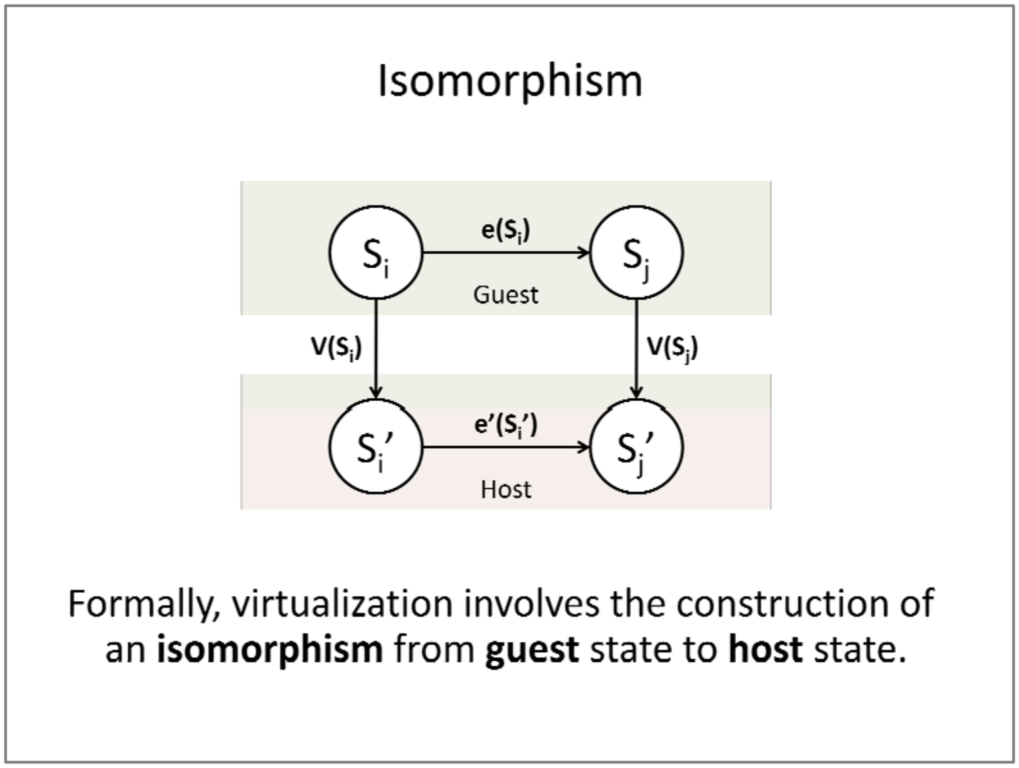
Linux
Linux (devel)
XP
Vista
MacOS
Virtual Machine Monitor
Hardware

Virtualization can be defined many ways. I will try to define it formally and also define it by giving a few examples. However loosely, virtualization is the addition of a software layer (the virtual machine monitor) between the hardware and the existing software that exports an interface at the same level as the underlying hardware.

In the strictest case the exported interface is the exact same as the underlying hardware and the virtual machine monitor provides no functionality except multiplexing the hardware among multiple VMs. This was largely the case in the old IBM VM/360 systems.

However the layer really can export a different hardware interface as the case in cross-ISA emulators. Also the layer can provide additional functionality not present in the operating system.

I think of virtualization as the addition of a layer of software that can run the original software with little or no changes.



Isomorphism

Guest

$$\begin{tikzcd} S_i \arrow[r, "e(S_i)"] \arrow[d, "V(S_i)"'] & S_j \arrow[d, "V(S_j)"] \\ S_i' \arrow[r, "e'(S_i')"] & S_j' \end{tikzcd}$$

Host

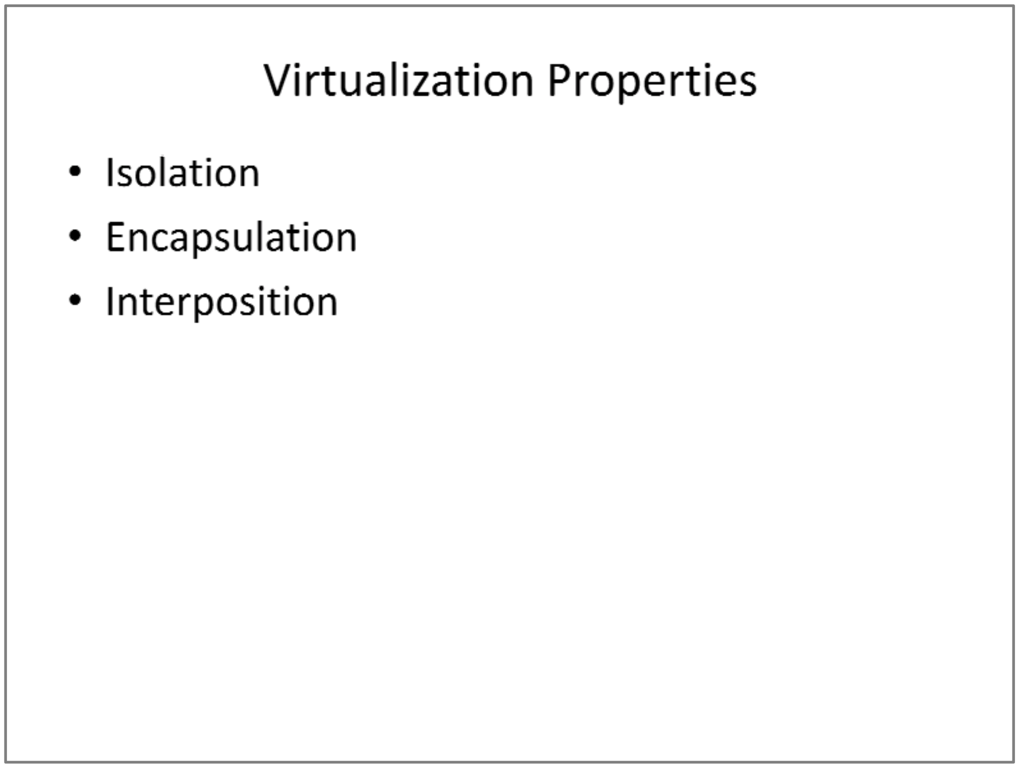
Formally, virtualization involves the construction of an **isomorphism** from **guest** state to **host** state.

Virtualization software constructs an isomorphism from guest to host.

All guest state S is mapped onto host state S' through some function V(S).

Additionally for every state changing operation e(S) in the guest there is a corresponding state changing operation e'(S') in the host.

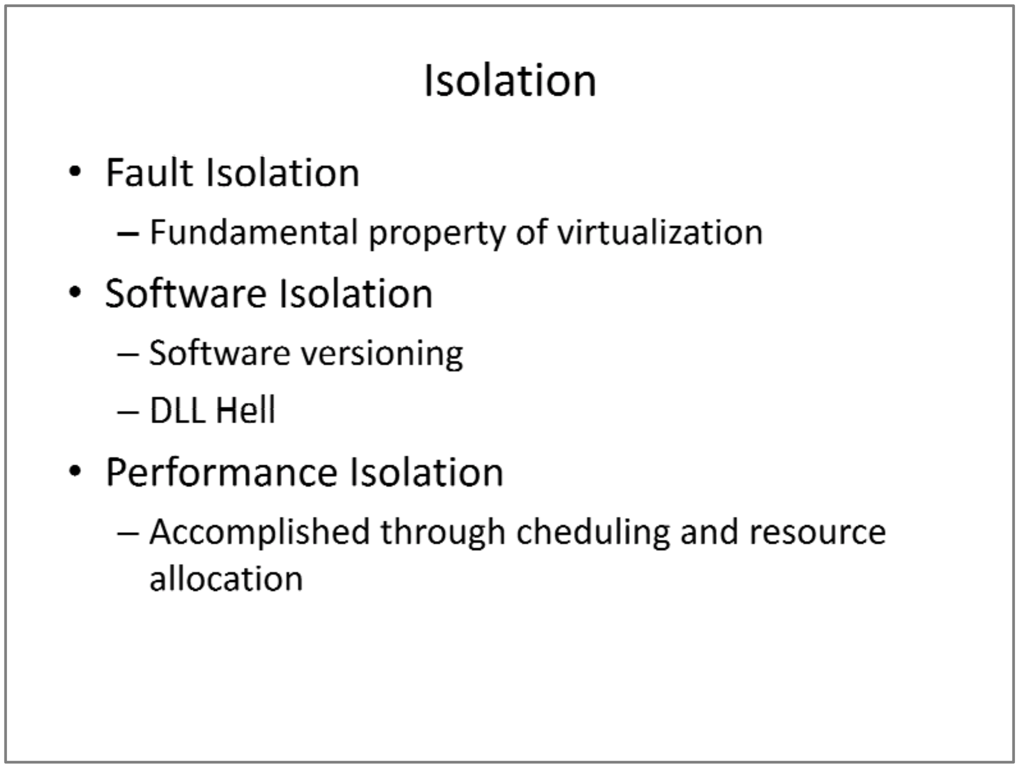
Virtualization software must implement V() and e().



Virtualization Properties

- Isolation
- Encapsulation
- Interposition

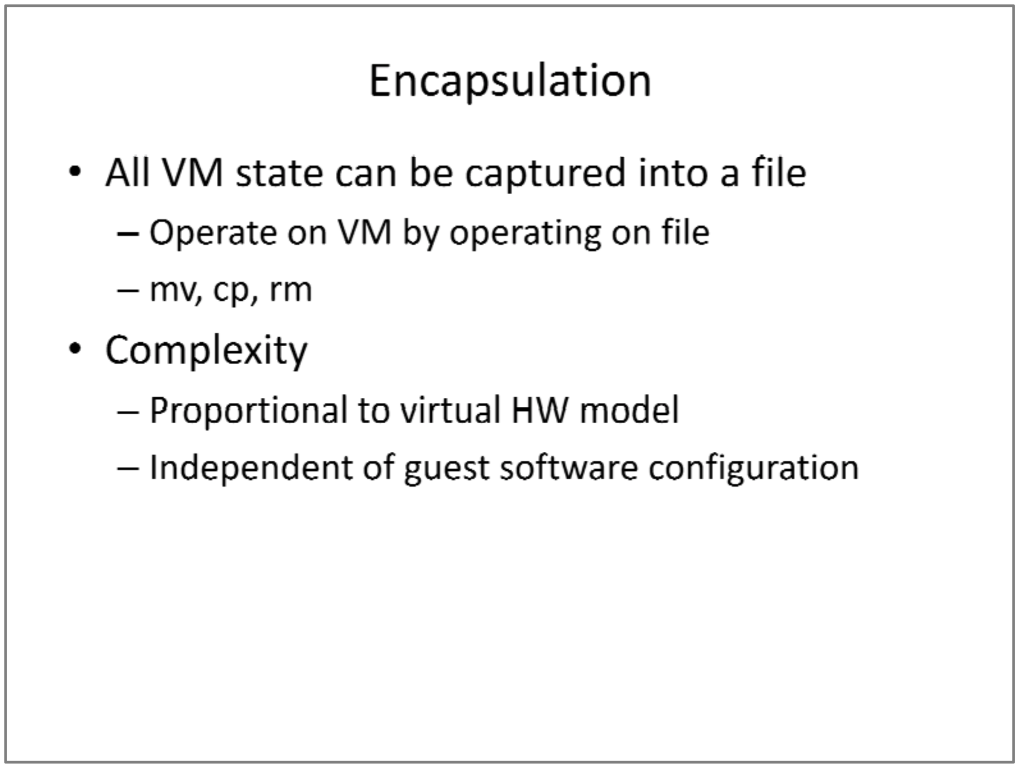
Virtualization has three main properties that give rise to all its applications.


Isolation

- Fault Isolation
  - Fundamental property of virtualization
- Software Isolation
  - Software versioning
  - DLL Hell
- Performance Isolation
  - Accomplished through cheduling and resource allocation

First, virtualization provides isolation. Isolation is key for many applications and comes in several flavors.

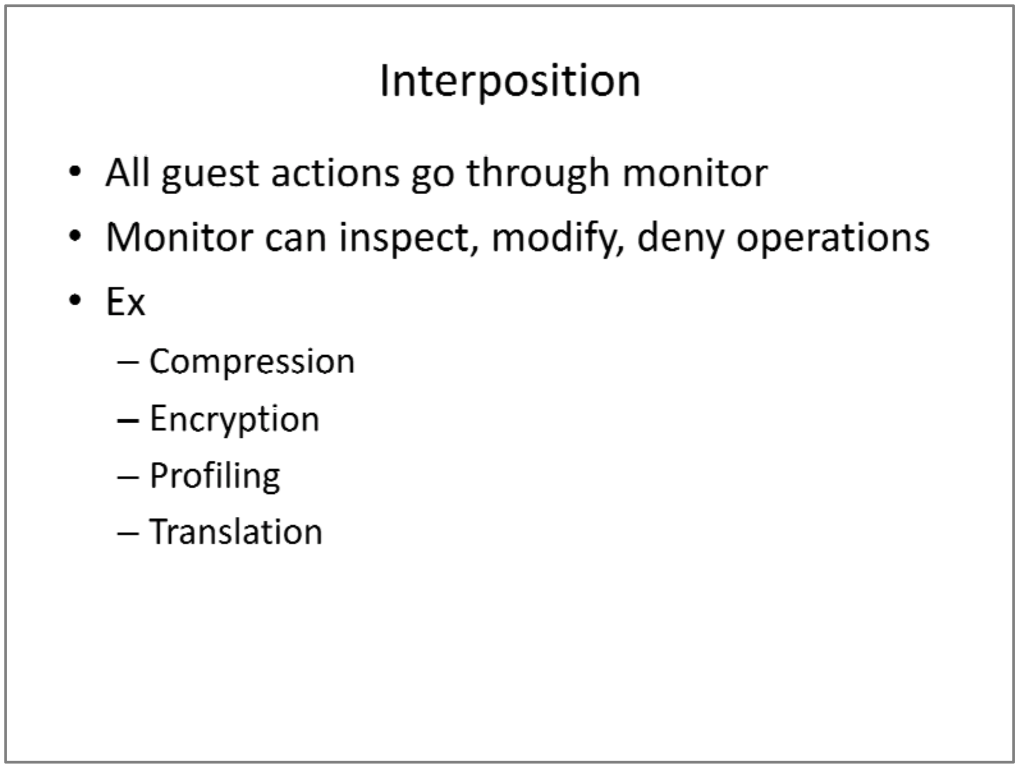
• Fault Isolation. If one virtual machine contains a buggy operating system, that OS can start scribbling all over physical memory. These wild rights must be contained within the VM boundaries.
• Performance Isolation. Ideally VMs performance would be independent of the activity going-on on the hardware. This must be accomplished by smart scheduling and resource allocation policies in the monitor.
• Software Isolation. Most of the issues with computers today are complex software configurations. DLL hell on PCs, operating system and library versions, viruses, and other security threats. VMs are naturally isolated for each other by running in separate software environments.



Encapsulation

- All VM state can be captured into a file
  - Operate on VM by operating on file
  - mv, cp, rm
- Complexity
  - Proportional to virtual HW model
  - Independent of guest software configuration

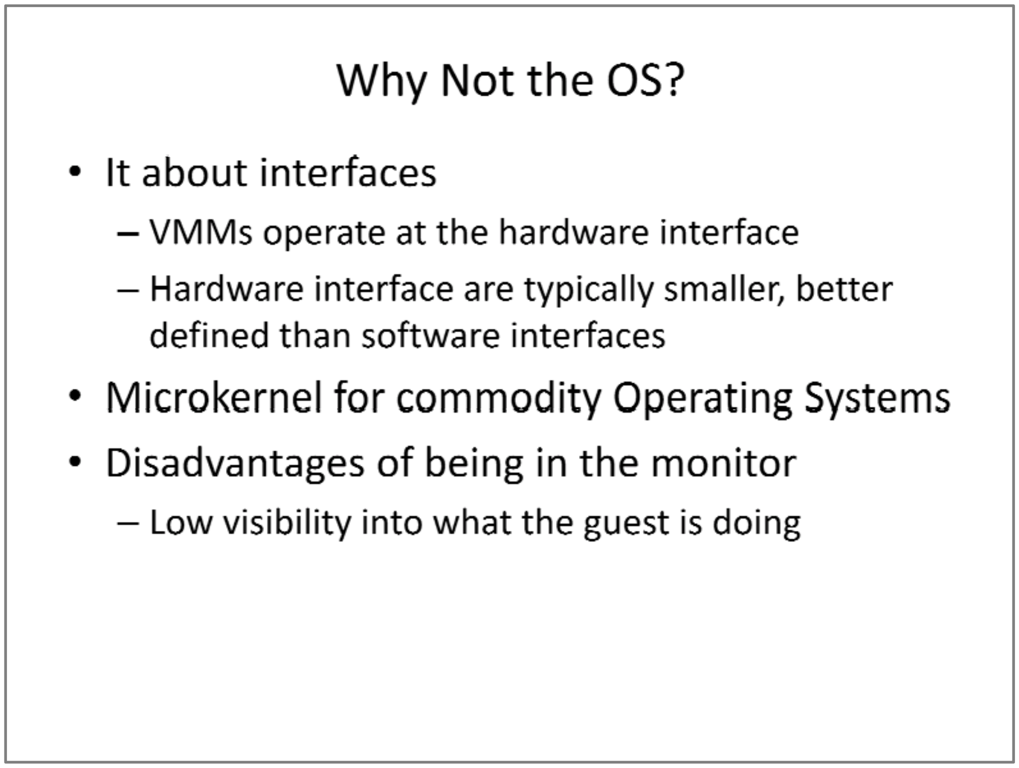
Encapsulation is the property that all VM state can be described and recorded simply. The VM state is basically the dynamic memory, static memory, and the register state of the CPU and devices. These items typically have a simple layout and are easy to describe. We can checkpoint a VM by writing out these items to a few files. The VM can be moved and copied by moving these files around. You can think about this as similar to doing a backup at the block level vs. doing a backup by recording all the packages, configuration and data files that encompass a file system.



Interposition

- All guest actions go through monitor
- Monitor can inspect, modify, deny operations
- Ex
  - Compression
  - Encryption
  - Profiling
  - Translation

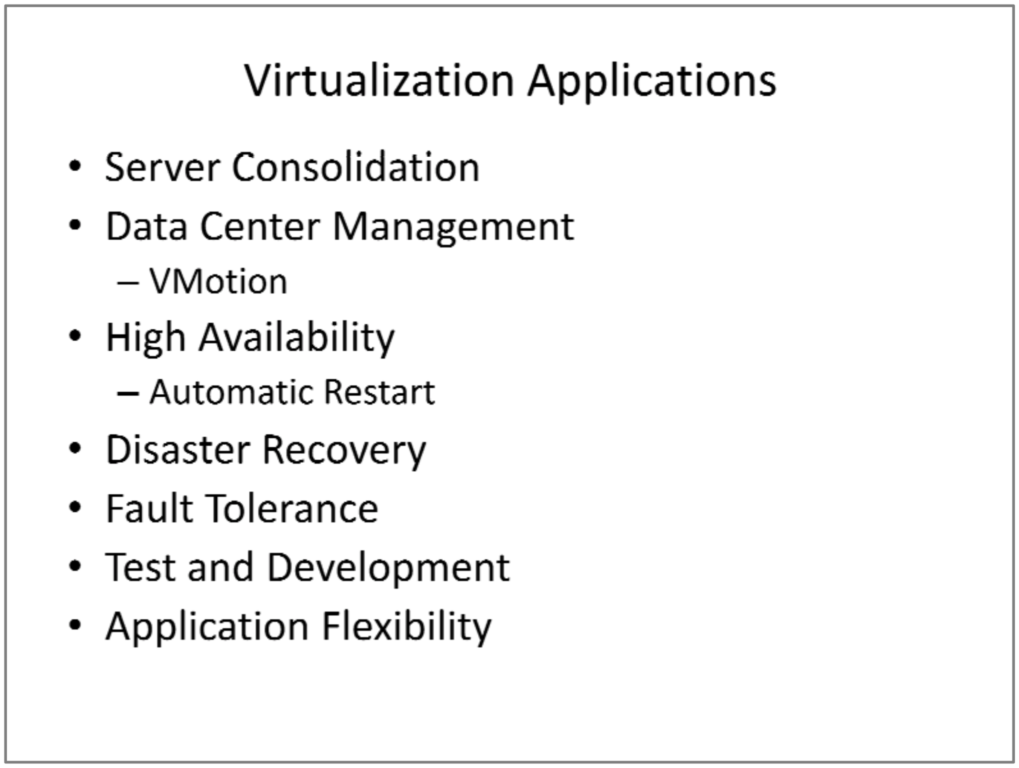
At some level all access to the hardware passes through the monitor first. This gives the monitor and chance to operate on these accesses. The best example of this is encrypting all data written to a disk. The advantage of this is that it does it without the knowledge of the OS.



Why Not the OS?

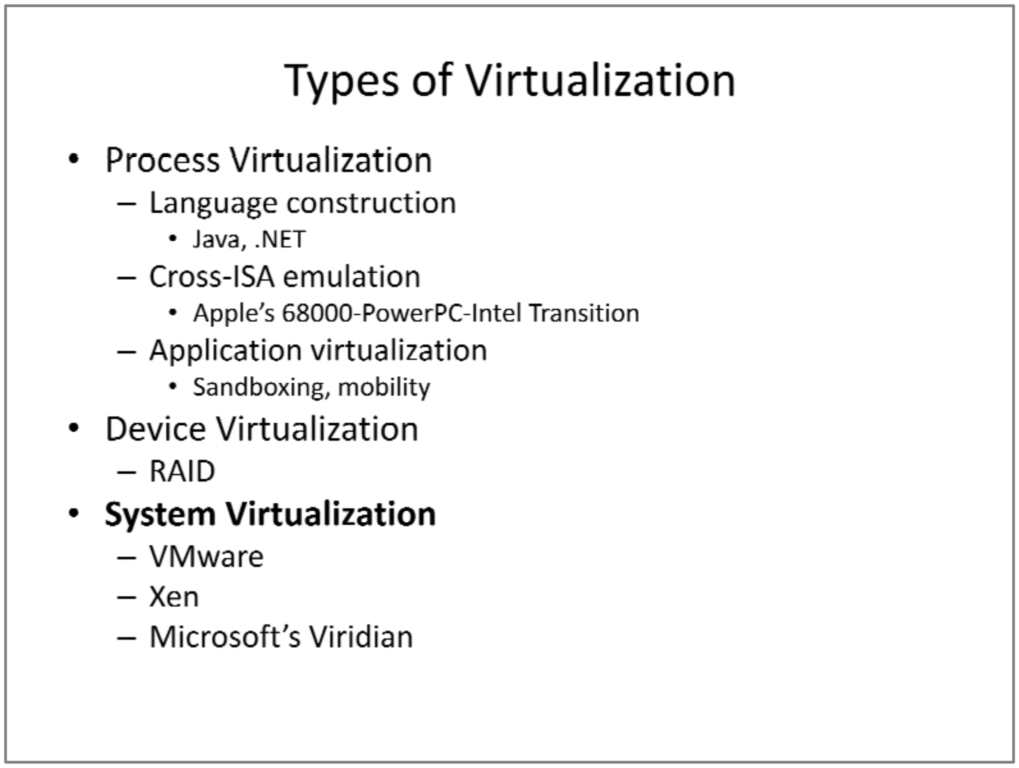
- It about interfaces
  - VMMs operate at the hardware interface
  - Hardware interface are typically smaller, better defined than software interfaces
- Microkernel for commodity Operating Systems
- Disadvantages of being in the monitor
  - Low visibility into what the guest is doing

This brings up a good point. Why not do these things in the OS. By splitting up the system this way the OS functions more like a large application library. The VMM functions more like a smart set of device drivers. This is a nice split and can simplify overall system design. It also provides a natural administration boundary. However the monitor is often at a disadvantage because it does not have the same insight into what’s happening as the OS has. For example, the OS knows the distinction between data and metadata when implementing an encrypted file system. So there is a tradeoff there.



Virtualization Applications

- Server Consolidation
- Data Center Management
  - VMotion
- High Availability
  - Automatic Restart
- Disaster Recovery
- Fault Tolerance
- Test and Development
- Application Flexibility



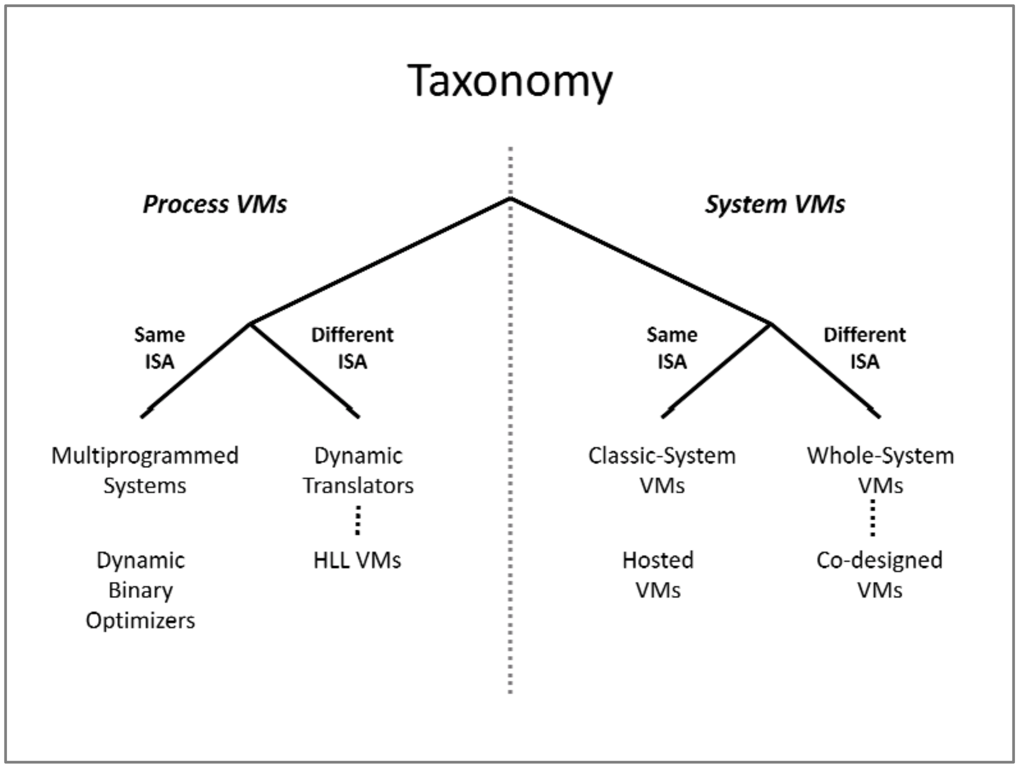
Types of Virtualization

- Process Virtualization
  - Language construction
    - Java, .NET
  - Cross-ISA emulation
    - Apple’s 68000-PowerPC-Intel Transition
  - Application virtualization
    - Sandboxing, mobility
- Device Virtualization
  - RAID
- **System Virtualization**
  - VMware
  - Xen
  - Microsoft’s Viridian

In process virtualization, only a single process is run under the control of the virtualization software. Typically both the process and the virtualization software run at user level in the same context. The most common uses of process virtualization are language construction and cross-ISA emulation. Dynamic binary optimization has also become popular in the literature. Commercial another type of virtualization called Application Virtualization aims to offer the same benefits of system virtualization by does it at the application level.

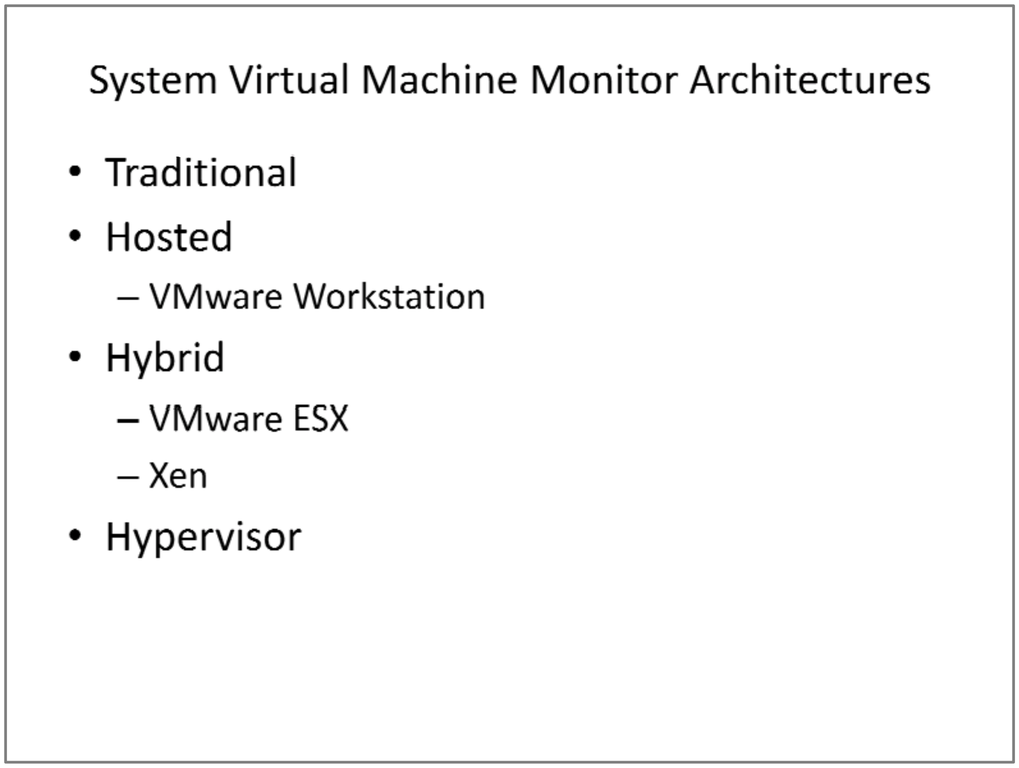
Typically the operating system is responsible for creating levels of abstraction on top of the systems devices. However many of the techniques can be considered virtualization. For example when the exported interface is at the same level as the underlying interface, I would consider this device virtualization. Specifically, RAID is a type of disk virtualization. NAT is a type of network virtualization. These virtualizations can be components of a larger system virtualization or can be implemented at the lower level of the operating system.

System Virtualization aim to virtualize the entire system with enough accuracy to run largely unmodified operating systems on top.



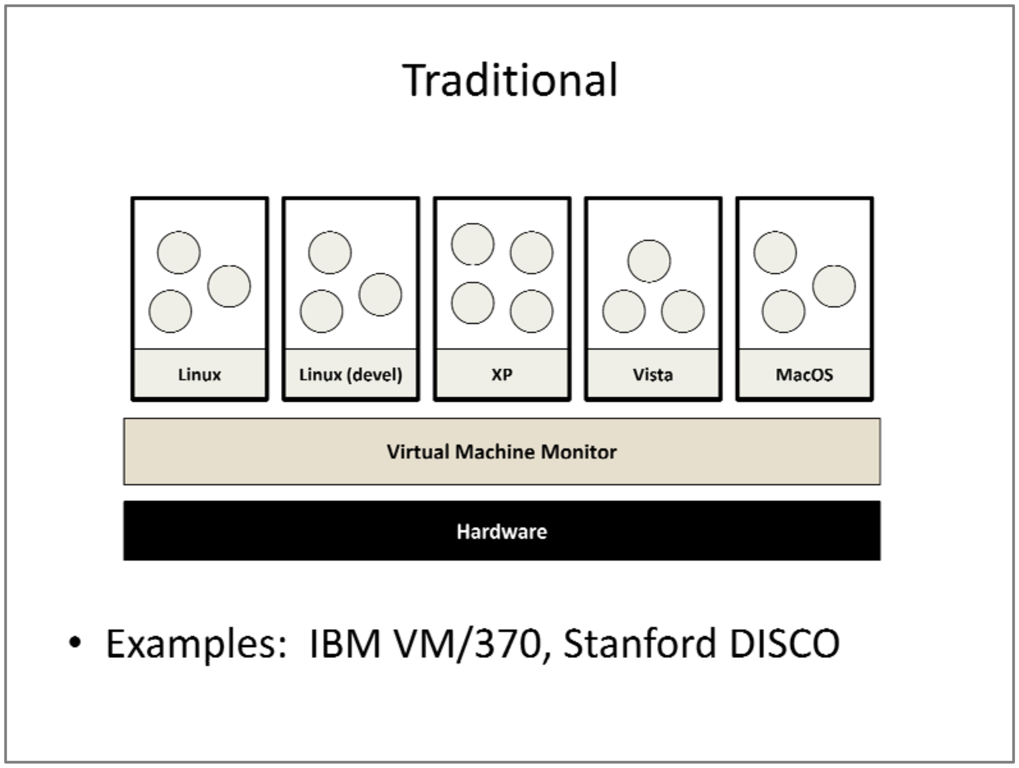
Taxonomy

Process VMs
System VMs
Same ISA
Different ISA
Same ISA
Different ISA
Multiprogrammed Systems
Dynamic Binary Optimizers
Dynamic Translators
HLL VMs
Classic-System VMs
Hosted VMs
Whole-System VMs
Co-designed VMs

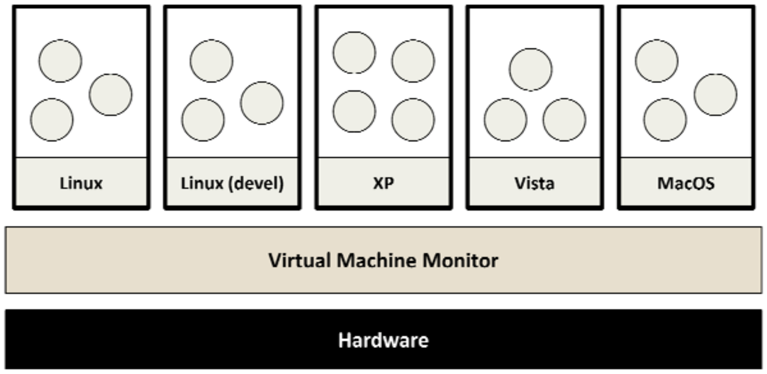


System Virtual Machine Monitor Architectures

- Traditional
- Hosted
  - VMware Workstation
- Hybrid
  - VMware ESX
  - Xen
- Hypervisor



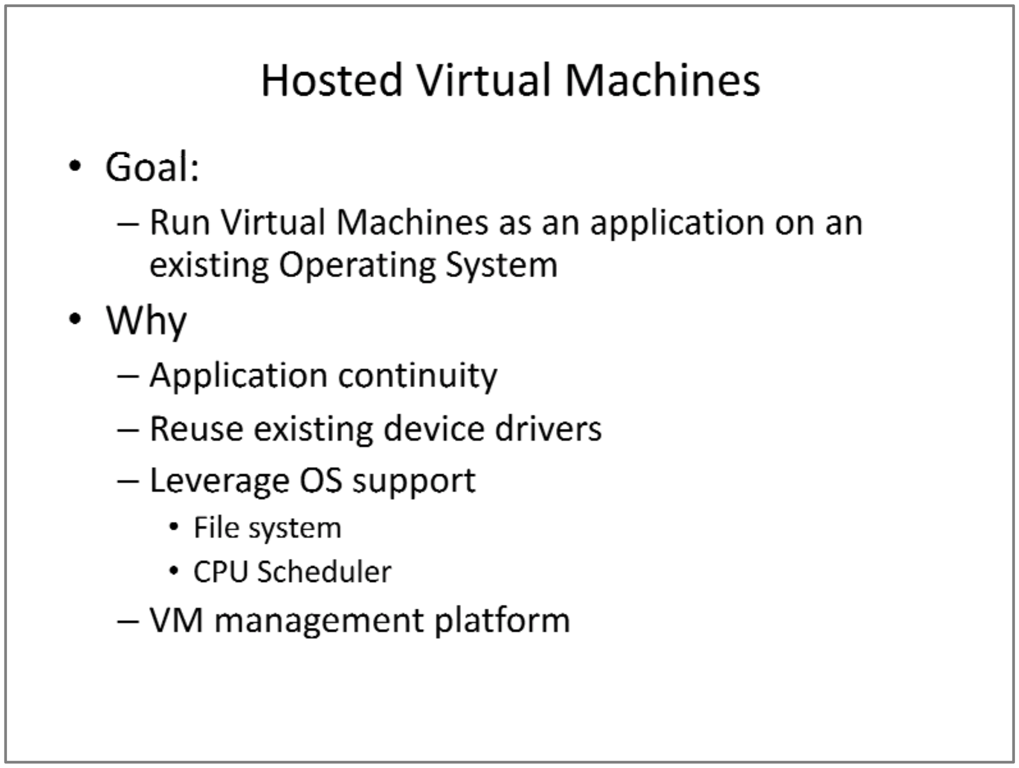
Traditional



- Examples: IBM VM/370, Stanford DISCO

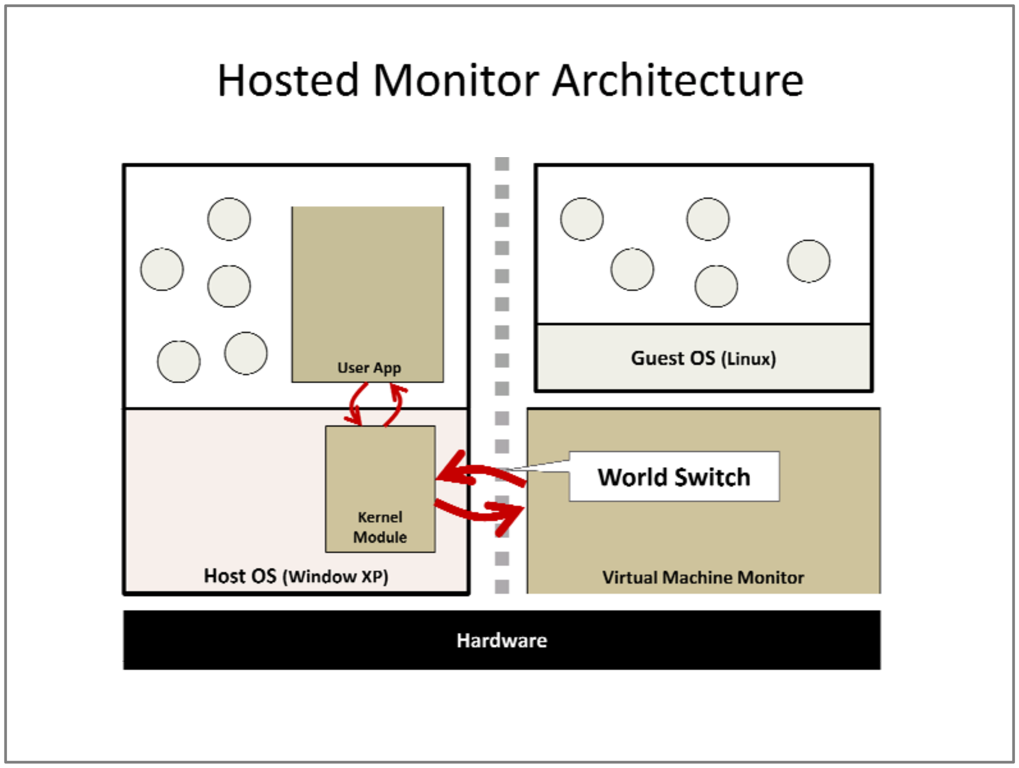
Here the monitor needs to be self-sufficient. There is no help from an Operating System.
What does this mean the monitor needs to have:

CPU scheduler
Memory allocator
Device Drivers
File system
Network stack for administration

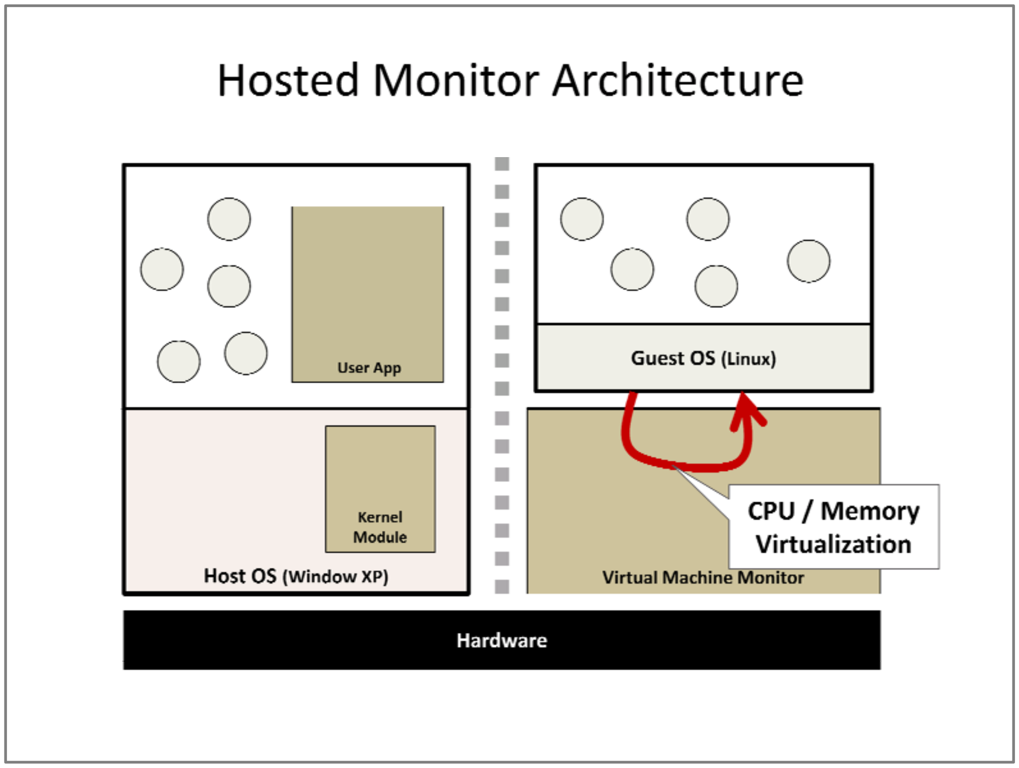


Hosted Virtual Machines

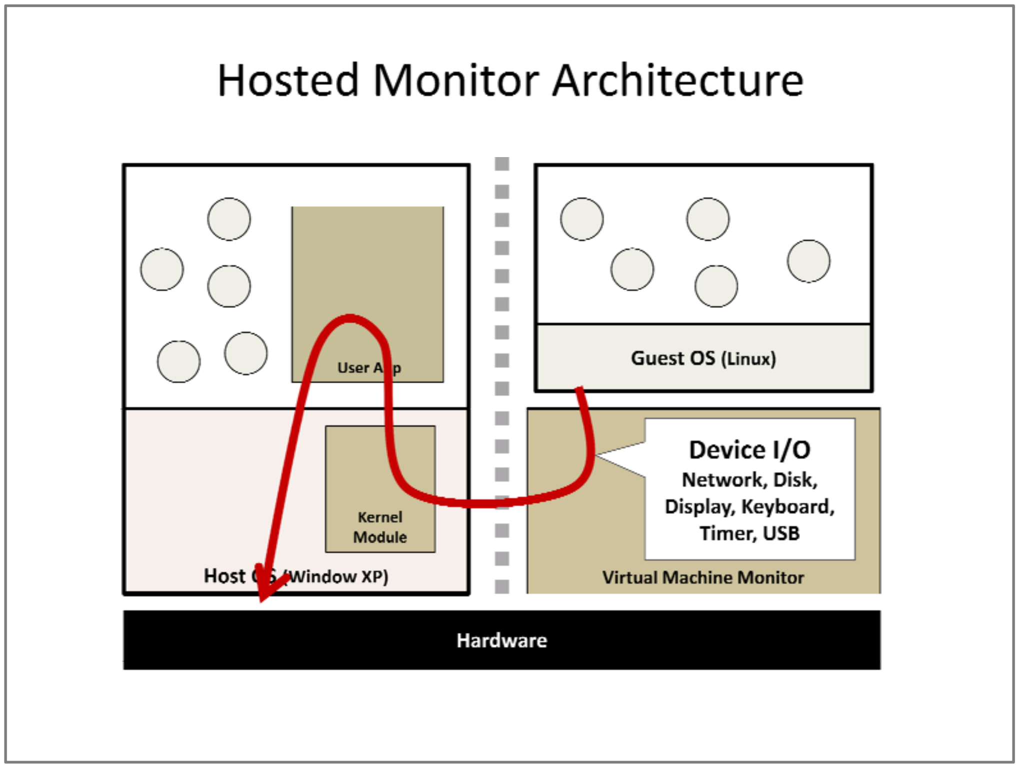
- Goal:
  - Run Virtual Machines as an application on an existing Operating System
- Why
  - Application continuity
  - Reuse existing device drivers
  - Leverage OS support
    - File system
    - CPU Scheduler
  - VM management platform



The virtual machine monitor runs in its own address space at kernel level. The VMM time shares the hardware with the host. When the UserApp runs in the host, it switches to the VMM by way of a World Switch. The world switch save all the registers, page tables, etc of the host and then loads the state of the VMM. Initially the state of the VMM is setup up by the UserApp. This includes what the registers should be and the structure and contents of the page tables. When the VMM voluntarily gives up the CPU, it World Switches back to the host.

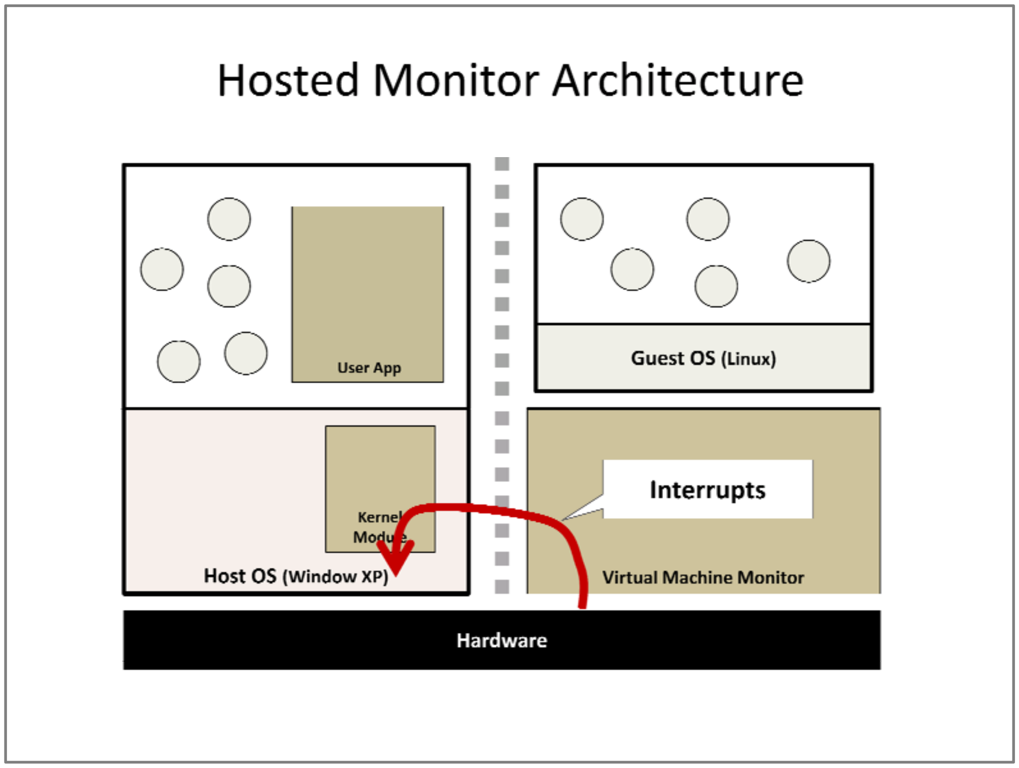


All CPU and Memory virtualization is handled internally to the VMM for performance. This is the whole idea behind this architecture. The VMM has access to all the privileged state of the CPU to provide the fastest CPU/memory virtualization possible.

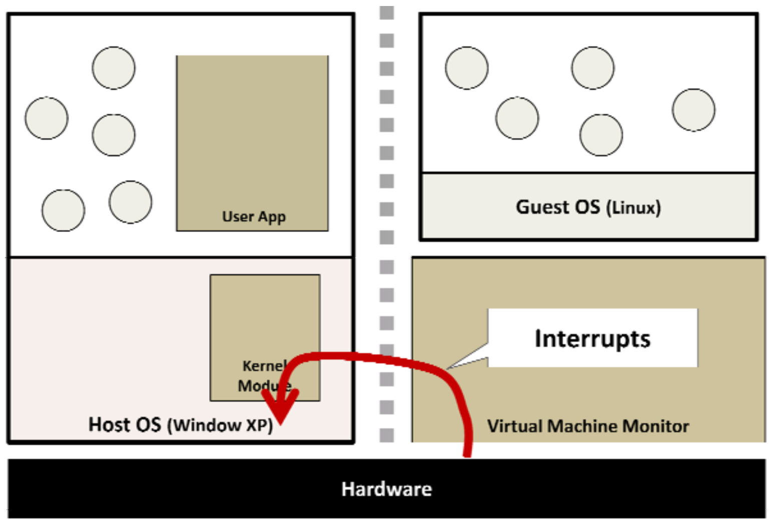


Hosted Monitor Architecture

To utilize OS abstractions and existing device drivers the VMM forwards all device requests to the UserApp. The UserApp then uses system call interfaces to access the devices.



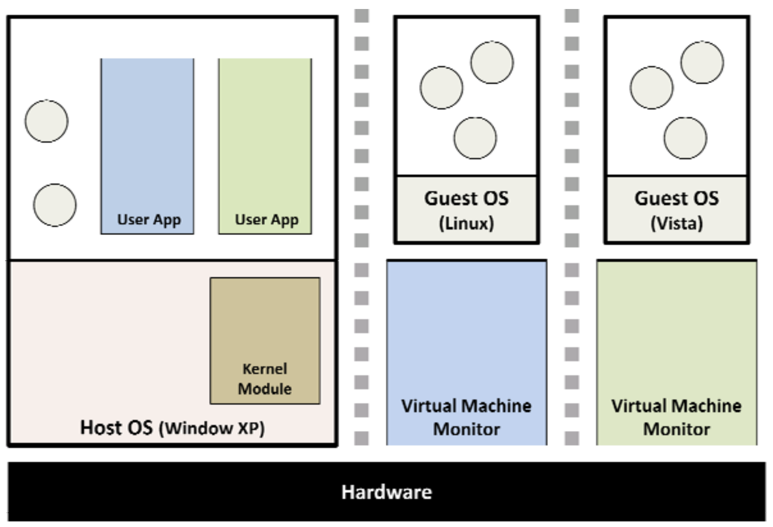
Hosted Monitor Architecture

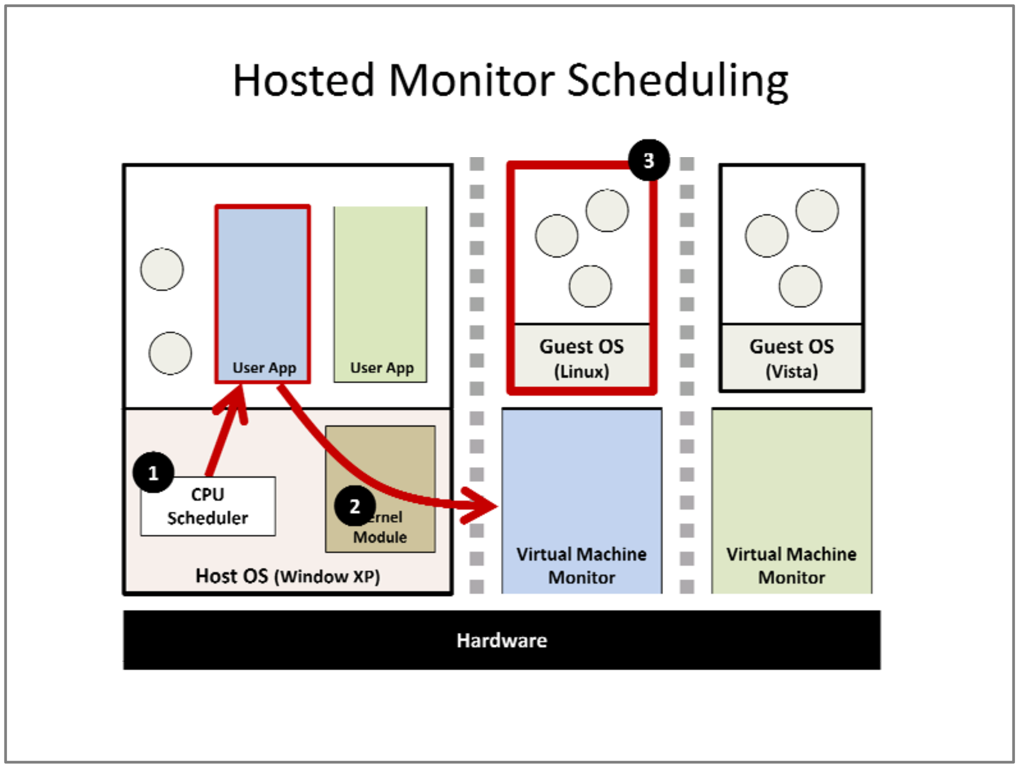


Because the VMM does not handle devices it simply forwards all interrupts on to the host.

Note the VMM must handle CPU generated exceptions like page faults and illegal instruction faults.

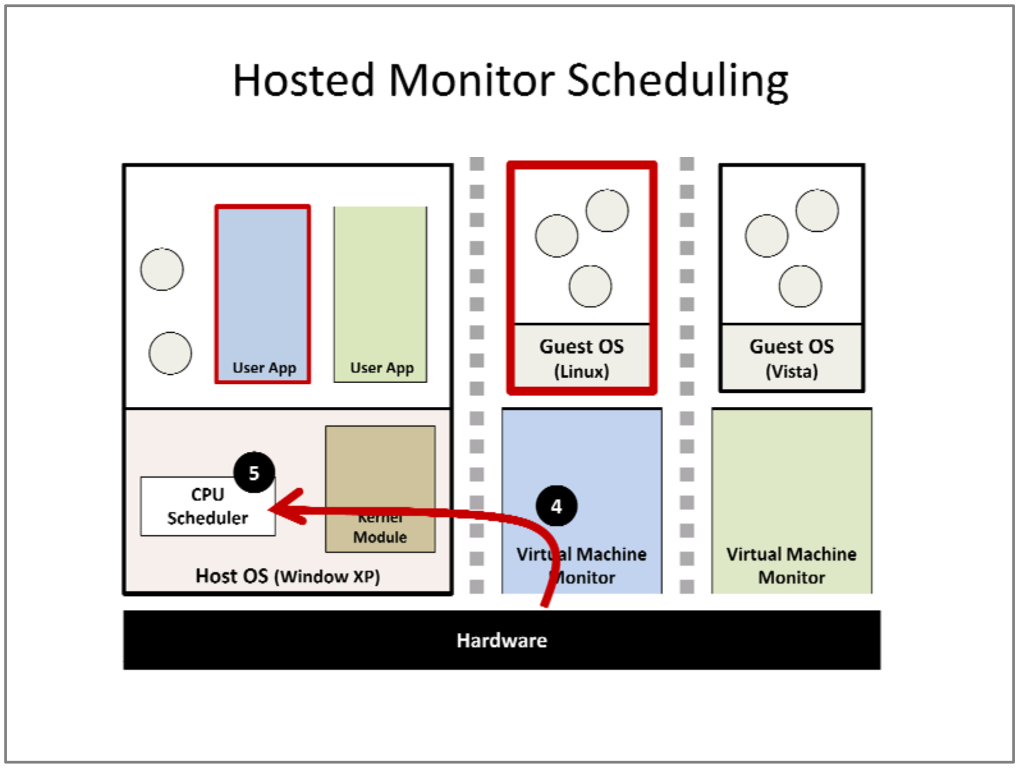
Hosted Monitor Scheduling





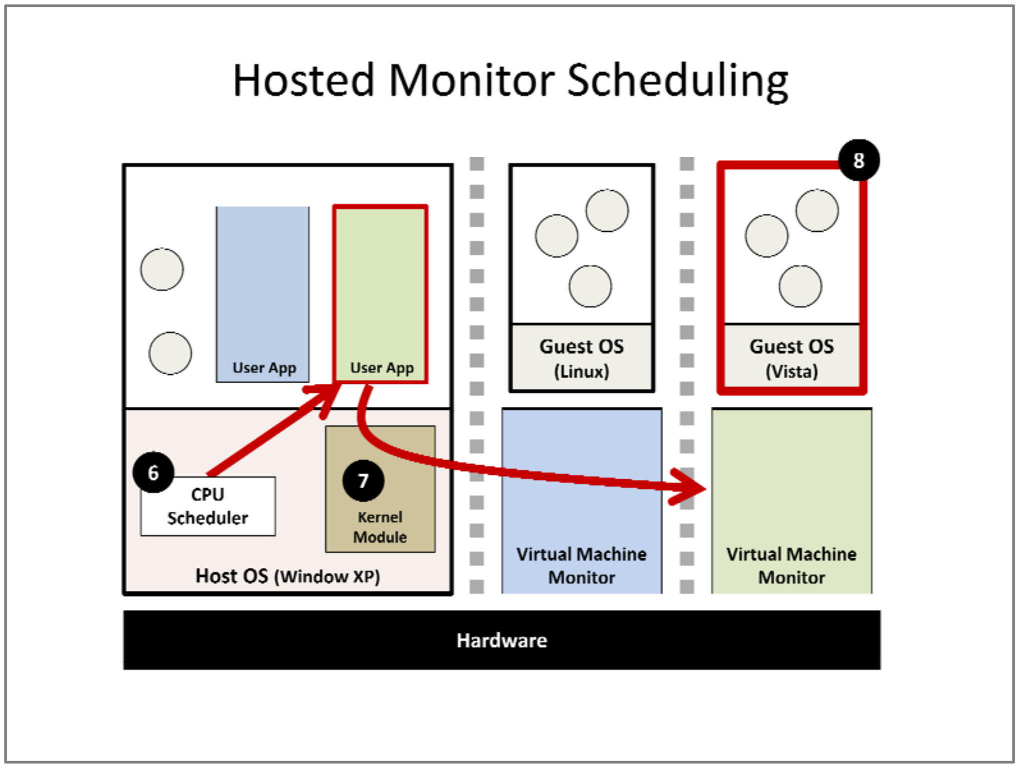
Hosted Monitor Scheduling

1. The CPU scheduler runs the blue UserApp .
2. The UserApp switches to its VMM.
3. The blue guest is run and gets CPU time.



Hosted Monitor Scheduling

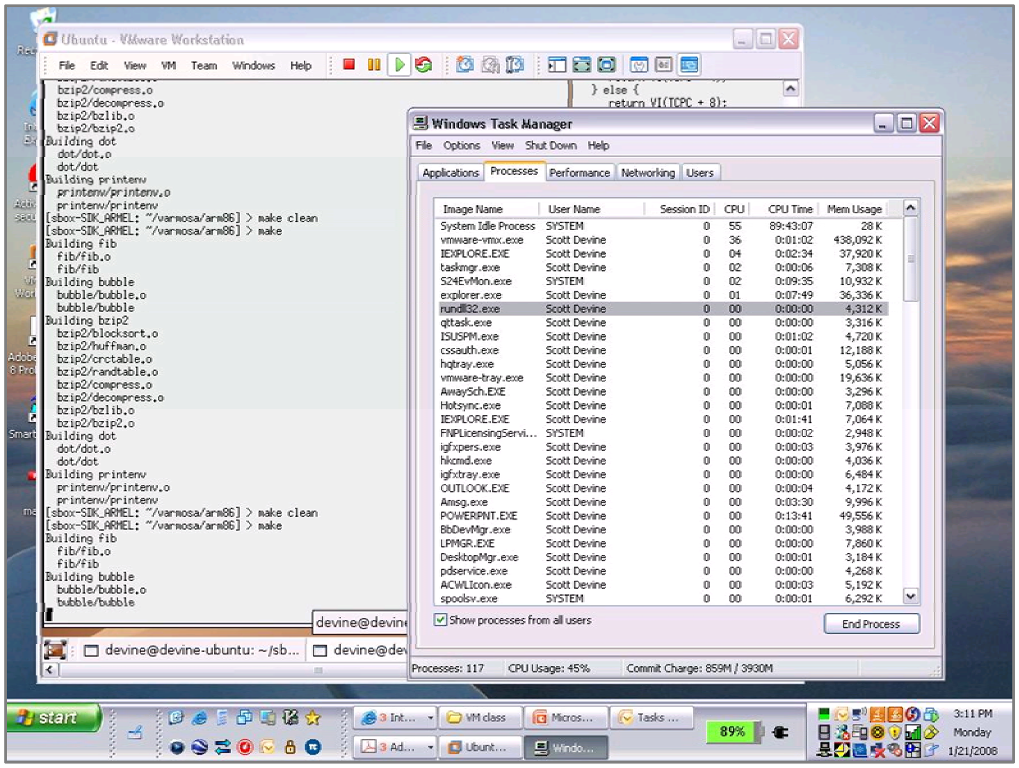
4. A time interrupt comes in.
5. The VMM forwards the timer interrupt to the host. The host scheduler runs.



Hosted Monitor Scheduling

6. The host scheduler deschedules the blue UserApp and schedules the green UserApp.
7. The green UserApp switch to its VMM.
8. The green guest gets CPU Time

How is the time run in VM accounted for?



Ubuntu - VMware Workstation
File Edit View VM Team Windows Help
Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users
End Process
devine@devine-ubuntu: ~/sb...
start
3:11 PM
Monday
1/21/2008

```
  bzip2/compress.o
  bzip2/decompress.o
  bzip2/bzlib.o
  bzip2/bzip2.o
Building dot
  dot/dot.o
  dot/dot
Building printenv
  printenv/printenv.o
  printenv/printenv
[sbox-SDK_ARMEL: ~/varmosa/arm86] > make clean
[sbox-SDK_ARMEL: ~/varmosa/arm86] > make
Building fib
  fib/fib.o
  fib/fib
Building bubble
  bubble/bubble.o
  bubble/bubble
Building bzip2
  bzip2/blocksort.o
  bzip2/huffman.o
  bzip2/crctable.o
  bzip2/randtable.o
  bzip2/compress.o
  bzip2/decompress.o
  bzip2/bzlib.o
  bzip2/bzip2.o
Building dot
  dot/dot.o
  dot/dot
Building printenv
  printenv/printenv.o
  printenv/printenv
[sbox-SDK_ARMEL: ~/varmosa/arm86] > make clean
[sbox-SDK_ARMEL: ~/varmosa/arm86] > make
Building fib
  fib/fib.o
  fib/fib
Building bubble
  bubble/bubble.o
  bubble/bubble
```

```
} else {
  return VI(TCPC + 8);
```

Image Name	User Name	Session ID	CPU	CPU Time	Mem Usage
System Idle Process	SYSTEM	0	55	89:43:07	28 K
vmware-vmx.exe	Scott Devine	0	36	0:01:02	438,092 K
IEXPLORE.EXE	Scott Devine	0	04	0:02:34	37,920 K
taskmgr.exe	Scott Devine	0	02	0:00:06	7,308 K
S24EvMon.exe	SYSTEM	0	02	0:09:35	10,932 K
explorer.exe	Scott Devine	0	01	0:07:49	36,336 K
rundll32.exe	Scott Devine	0	00	0:00:00	4,312 K
qttask.exe	Scott Devine	0	00	0:00:00	3,316 K
ISUSPM.exe	Scott Devine	0	00	0:01:02	4,720 K
cssauth.exe	Scott Devine	0	00	0:00:01	12,188 K
hqtray.exe	Scott Devine	0	00	0:00:00	5,056 K
vmware-tray.exe	Scott Devine	0	00	0:00:00	19,636 K
AwaySch.EXE	Scott Devine	0	00	0:00:00	3,296 K
Hotsync.exe	Scott Devine	0	00	0:00:01	7,088 K
IEXPLORE.EXE	Scott Devine	0	00	0:01:41	7,064 K
FNPLicensingServi...	SYSTEM	0	00	0:00:02	2,948 K
igfxpers.exe	Scott Devine	0	00	0:00:03	3,976 K
hkcmd.exe	Scott Devine	0	00	0:00:00	4,036 K
igfxtray.exe	Scott Devine	0	00	0:00:00	6,484 K
OUTLOOK.EXE	Scott Devine	0	00	0:00:04	4,172 K
Amsg.exe	Scott Devine	0	00	0:03:30	9,996 K
POWERPNT.EXE	Scott Devine	0	00	0:13:41	49,556 K
BbDevMgr.exe	Scott Devine	0	00	0:00:00	3,988 K
LPMGR.EXE	Scott Devine	0	00	0:00:00	7,860 K
DesktopMgr.exe	Scott Devine	0	00	0:00:01	3,184 K
pdservice.exe	Scott Devine	0	00	0:00:00	4,268 K
ACWLIcon.exe	Scott Devine	0	00	0:00:03	5,192 K
spoolsv.exe	SYSTEM	0	00	0:00:01	6,292 K

[✓] Show processes from all users

Processes: 117 CPU Usage: 45% Commit Charge: 859M / 3930M

The associated UserApp (VMX) gets it.

Hosted Architecture Tradeoffs

- Positives
  - Installs like an application
    - No disk partitioning needed
    - Virtual disk is a file on host file system
    - No host reboot needed
  - Runs like an application
    - Uses host schedulers
- Negatives
  - I/O path is slow
    - Requires world switch
  - Relies on host scheduling
    - May not be suitable for intensive VM workloads

VMware ESX 2.0

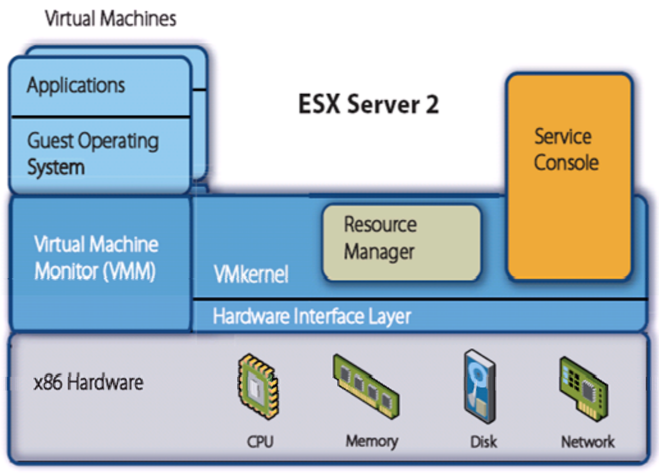


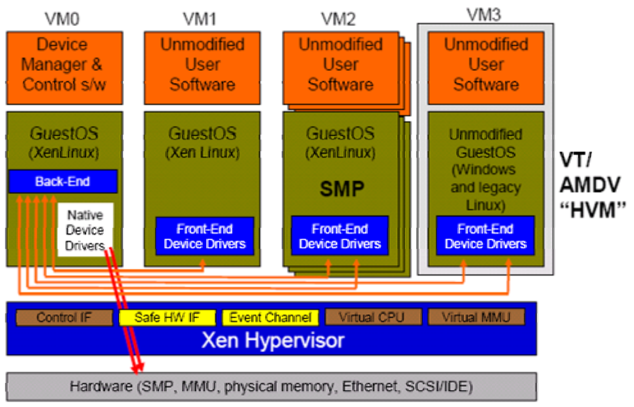
Figure 1: ESX Server architecture

Source: http://www.vmware.com/pdf/esx2_performance_implications.pdf

Traditional arch for cpu scheduling, memory allocation, virtual disk, virtual network. Console OS is there for legacy devices, USB, and as a management platform, i.e. it has the network stack.

Hybrid Ex 2 - Xen 3.0

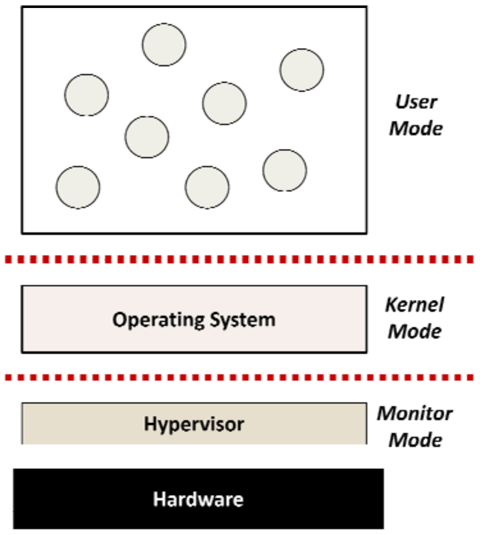
- Para –virtualization
  - Linux Guest
- Hardware-supported virtualization
  - Unmodified Windows
- Isolated Device Drivers



Source: Ottawa Linux Symposium 2006 presentation.
http://www.cl.cam.ac.uk/netos/papers/

Hypervisor

- Hardware-supported single-use monitor
- Characteristics
  - Small size
  - Runs in a special hardware mode
  - Guest OS runs in normal priviledge level
- Uses
  - Security
  - System management
  - Fault tolerance


User Mode
Operating System
Kernel Mode
Hypervisor
Monitor Mode
Hardware