

Windows Server Virtualization & The Windows Hypervisor

Background and Architecture Reference

Brandon Baker

Lead Security Engineer

Windows Kernel Team
Microsoft Corporation



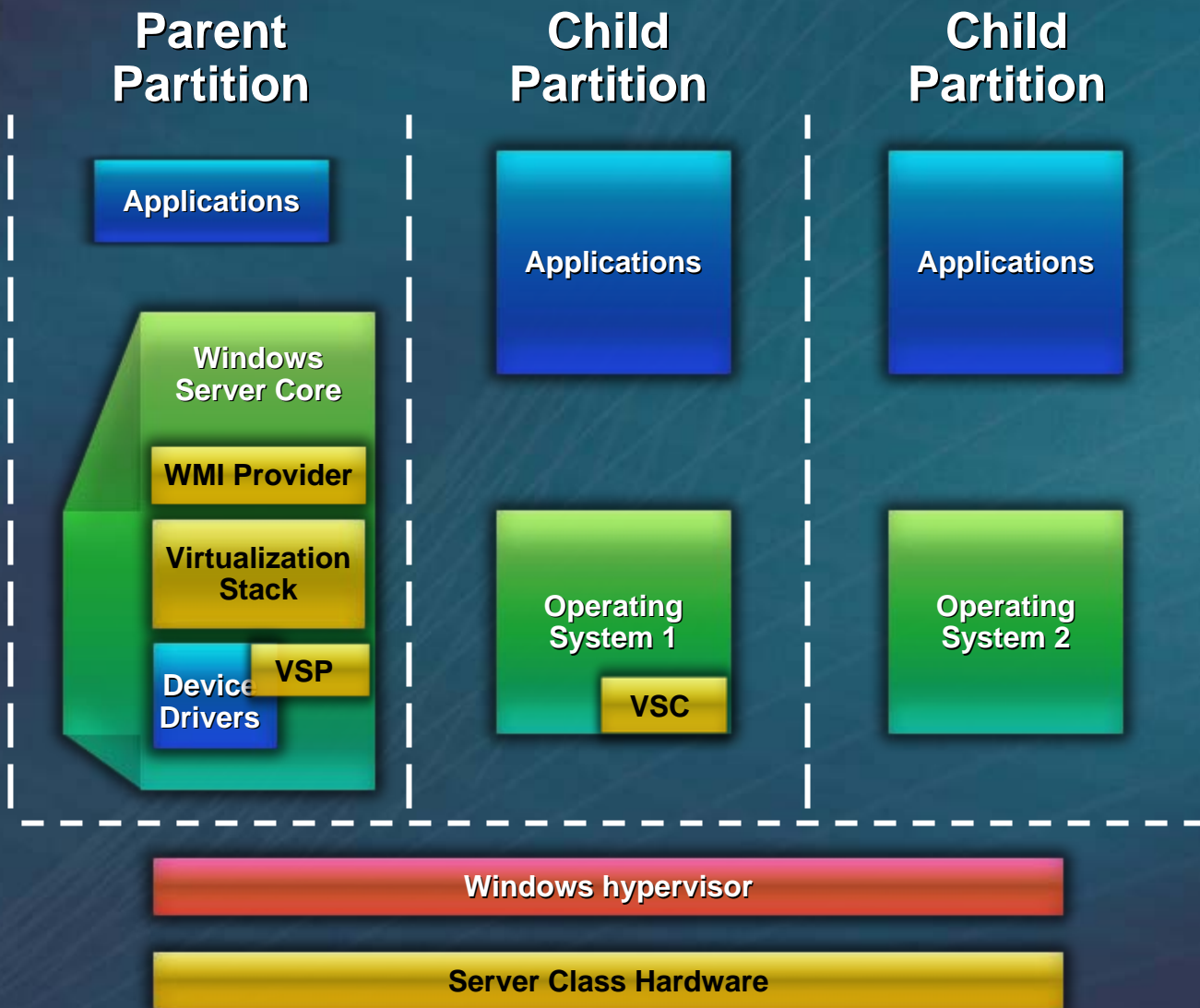
Agenda - Windows Server Virtualization (WSV)

- Background
- Architecture
 - Hypervisor
 - Virtualization Stack
 - Device Virtualization
- Security Goals and Characteristics
- Deployment Considerations
- Future Directions

Background

- Project code name “Viridian”
- Full machine virtualization for guest operating systems
- Component of Windows Server 2008
- Final version available within 180 days of Windows Server 2008 RTM
- Installs as a role on Server Core
- Hypervisor Based
 - Takes advantage of (and requires) processor virtualization extensions
 - Supported on x64 hosts only, 32/64bit guest support
- Has three major components:
 - Hypervisor
 - Virtualization Stack
 - Virtual Devices

WSV Architecture



WSV Targeted Use Scenarios

- Server consolidation
 - Lower total cost of ownership (TCO)
 - Maximize hardware utilization
 - Reduce datacenter heat, space, power
- Dynamic datacenter management
 - Decouple workloads from hardware
 - Simplify management of complex systems
- Business continuity
 - Disaster recovery
 - Reduce service interruptions
- Software development and testing
 - Multi-tier applications (enterprise-in-a-box)
 - Snapshots allow easy rollback, provisioning, and sharing

WSV Features

- 32-bit and 64-bit guests
- Guest multiprocessing
- WMI management and control API
- Save & restore
- Snapshotting
- CPU and I/O resource controls
- Dynamic virtual resource addition & removal
- Authorization model for administration

System Virtualization

- At hardware machine interface level
- Virtual Machine Monitor (VMM) virtualizes underlying hardware resources
- Multiple OSes execute concurrently
- Individual OSes manage virtualized resources like
 - Processors
 - Memory
 - IO Devices
- Resource virtualization techniques
 - Partitioning
 - Time sharing
 - Emulating

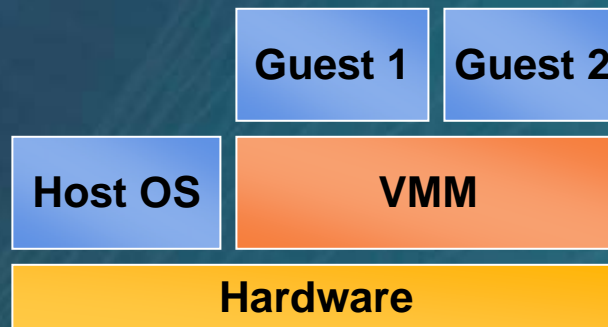
VMM Arrangements

Type-2 VMM



Examples:
JVM
CLR

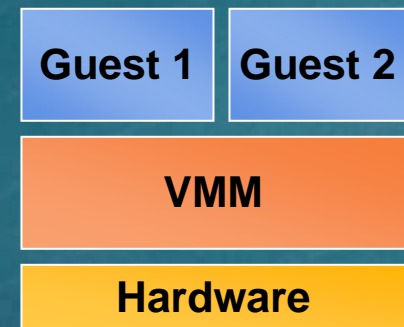
Hybrid VMM



Examples:
Microsoft Virtual PC
and Virtual Server

What we have today

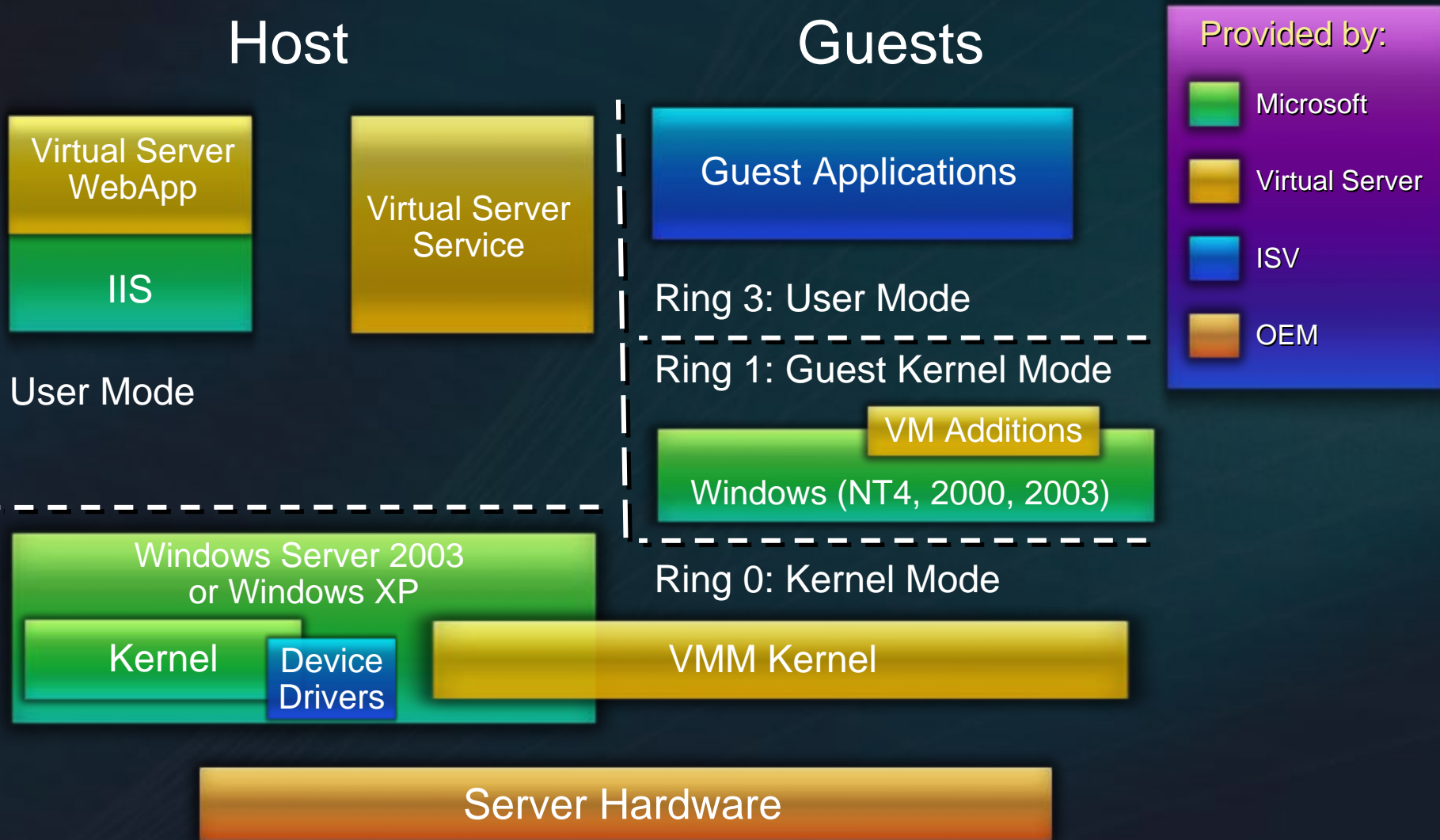
Type-1 VMM (Hypervisor)



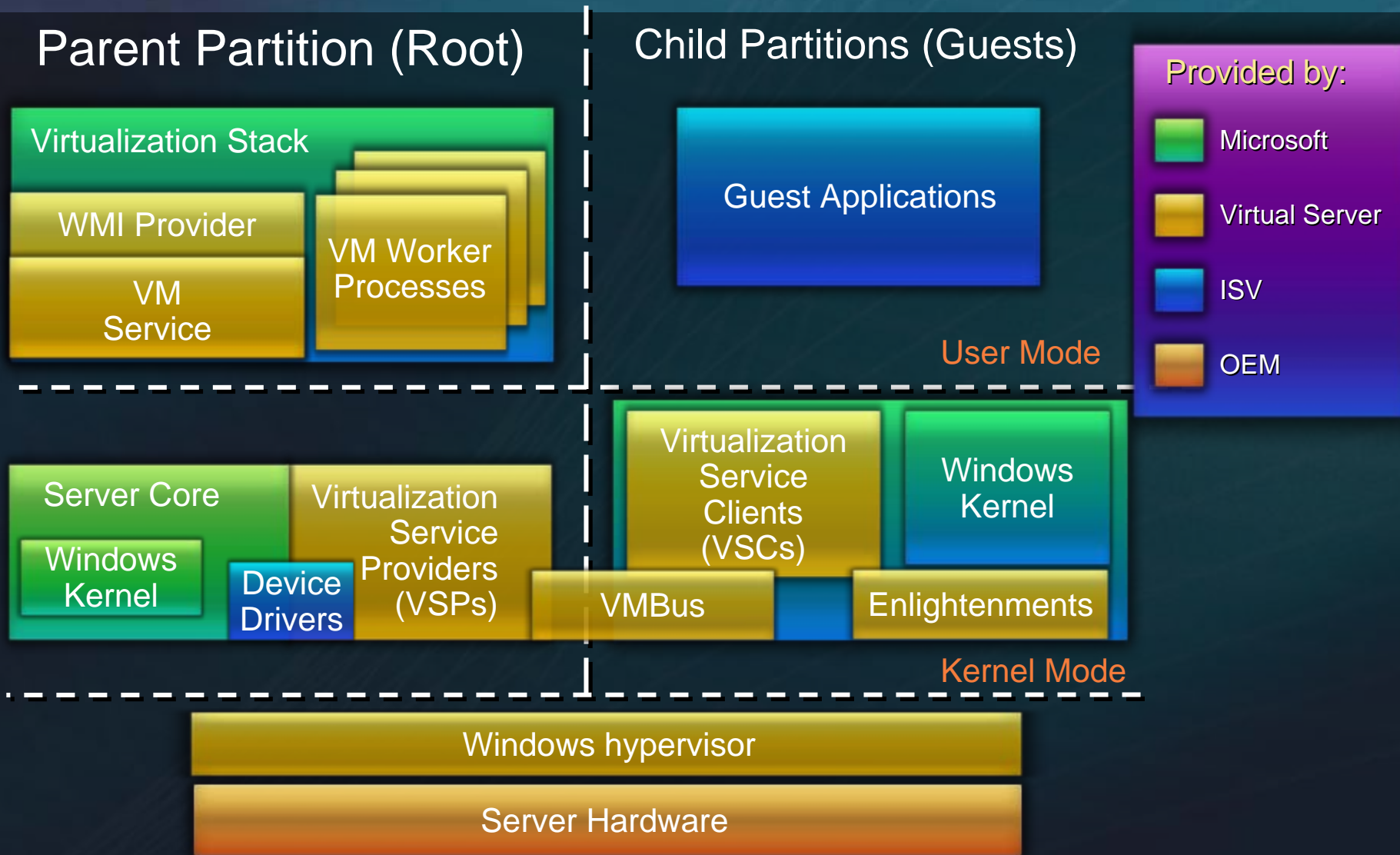
Examples:
Windows
Server
Virtualization

What's coming

Virtual Server Architecture



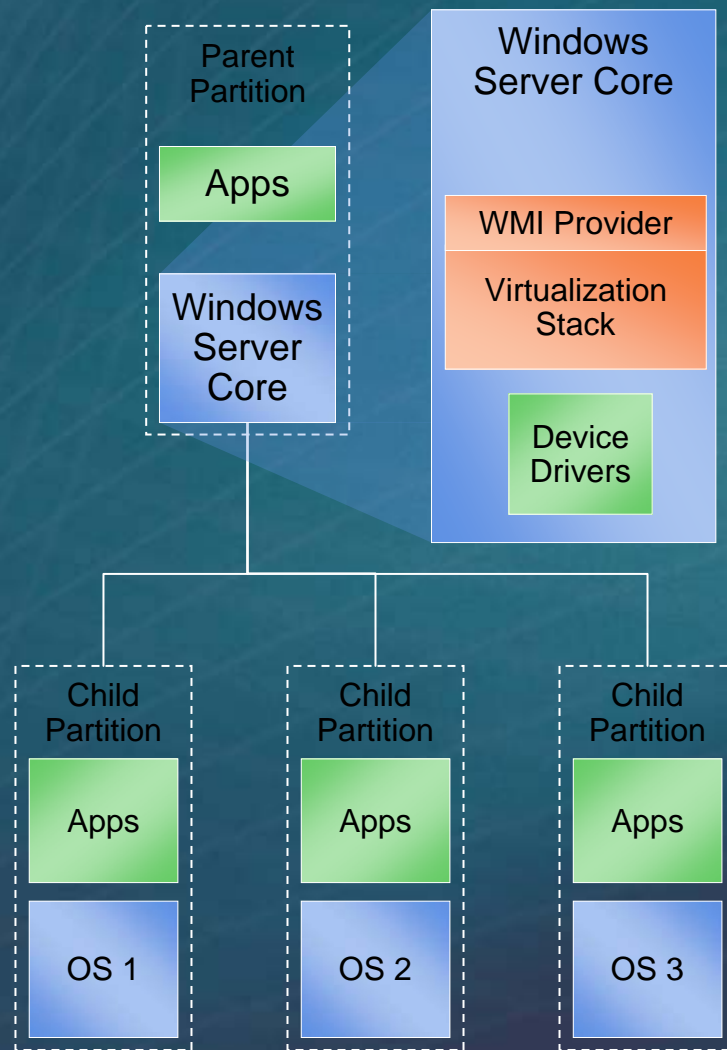
Windows Server Virtualization Architecture



WSV Architecture

Partition hierarchies

- Partitions are arranged in a tree
 - Parents manage children
- Each parent contains a **Virtualization stack**
 - Manages child's memory
 - Manages virtual devices
- In WSV, there is only one parent
 - We are considering deeper hierarchies for future versions



Hardware Innovations

- Processor virtualization extensions
 - Intel VT / AMD-V
 - Widely available today
 - Provides a new “monitor mode” (effectively, ring minus 1)
 - Decreases complexity and increases efficiency of VMMs
- Processor/Chipset security extensions
 - Intel TXT / AMD SVM
 - Coming soon / Available today
 - Provides a way to securely launch a VMM / hypervisor
 - Working on ways to allow for policy enforcement
- DMA remapping (IOMMU)
 - Intel, AMD & other chipset vendors
 - Coming soon
 - Provides support for “device assignment”
 - Provides protections against malicious DMA transfers

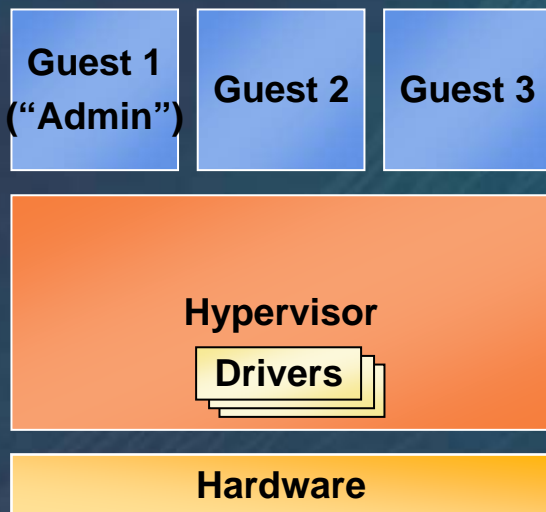
Hypervisor

- Partitioning Kernel
 - “Partition” is isolation boundary
 - Few virtualization functions; relies on virtualization stack
- Very thin layer of software
 - Microkernel
 - Highly reliable
 - Basis for smaller Trusted Computing Base (TCB)
- No device drivers
 - Two versions, one for Intel and one for AMD
 - Drivers run in a partition
 - Leverage the large base of Windows drivers
- Well-defined interface
 - Allow others to create support for their OSes as guests

Monolithic versus Microkernelized

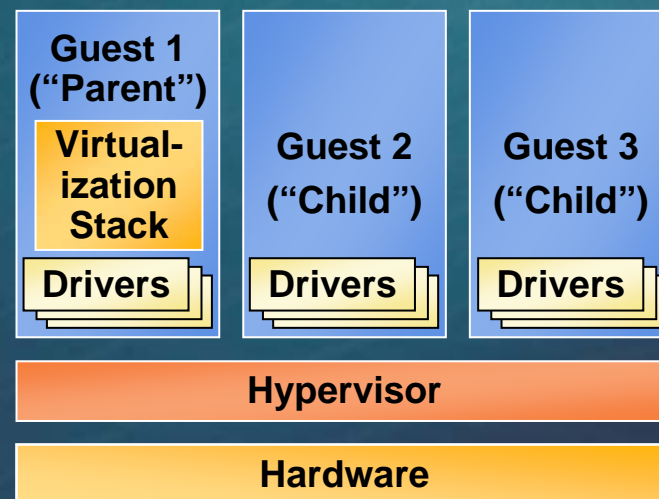
- Monolithic hypervisor

- Simpler than a modern kernel, but still complex
- Contains its own drivers model



- Microkernelized hypervisor

- Simple partitioning functionality
- Increase reliability and minimize lowest level of the TCB
- No third-party code
- Drivers run within guests



Hypervisor Design Goals

- Isolation
 - Security isolation
 - Fault isolation
 - Resource isolation
- Reliability
 - Minimal code base
 - Strictly layered design
 - Not extensible
- Scalability
 - Scale to large number of cores
 - Large memory systems

Hypervisor Design Details

- Modules strictly layered
- Resource accounting
 - Most CPU cycles and memory associated with a partition
- Hypercalls have must-complete semantics
 - Time bounded
- Designed for concurrency but not preemptible
- Cooperative deadline scheduler
- Shadow page tables for GPA (guest physical address) virtualization
- Intercept routing for virtual machine monitor extension

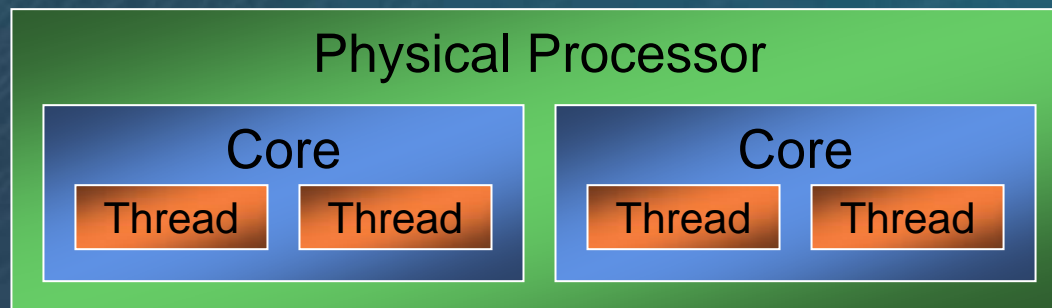
Hypercalls

Low level API

- Guests communicate with the hypervisor via hypercalls
 - Hypervisor equivalent of a syscall
 - Detected via Cpuid
 - Configured via MSR (Model Specific Register)
- Simple format
 - One input page, one output page
 - Specify pages by physical address, then jump to known address

Physical World Processors

- Physically
 - Customers plug processors into **sockets**
 - There are one or more **cores** per socket
 - There are one or more **threads/logical processors** per core
- OSes typically schedule logical processors
 - Cache pollution if multiple threads per core
 - Secrets recoverable from cache access patterns (inference attacks)
- The hypervisor schedules cores



Virtual World

Mapping to the physical world

- Partitions are the unit of containment
 - **Virtual Machine** refers to the partition and its state
- **Guests** are software that run in a partition
 - Such as a “Guest OS”
- **Virtual processors** correspond to logical processors
 - Abbreviated as “VP”



Address Space Concepts (1/5)

GPA and SPAs

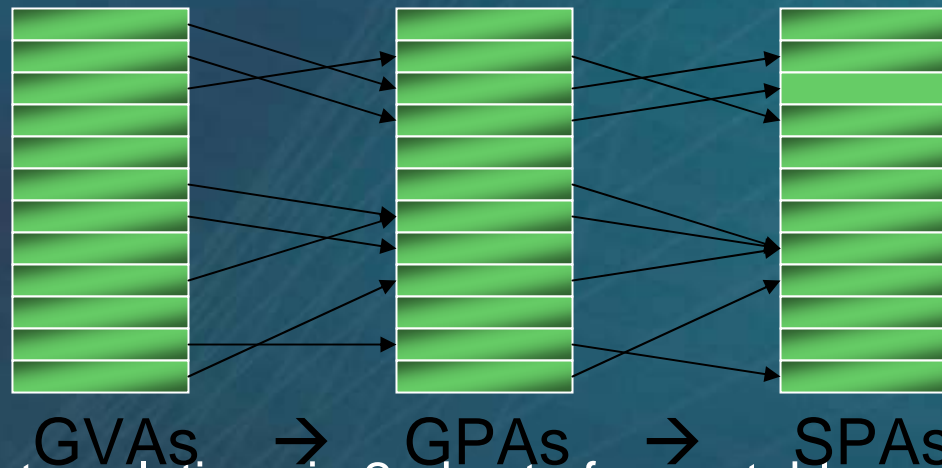
- Most guests expect physically contiguous memory starting at zero
 - Not everybody can start at zero
 - Contiguous memory hard to find after boot
- Solution: another layer of indirection
 - **SPAs**: System physical addresses
 - What the CPU, hardware* sees
 - **GPA**s: Guest physical addresses
 - What the guest OS sees
 - Can start at zero, appear contiguous

* modulo device apertures

Address Space Concepts (2/5)

GPA and SPAs

- Address translation converts
 - GVAs (Guest Virtual Addresses) to GPAs to SPAs



- GPA → SPA translation via 2nd set of page tables
 - Software today
 - Hardware assisted in the future
 - R/W/X access bits in GPA page table entries as well

Address Space Concepts (3/5)

Shadow Page Tables (SPT)

- Viridian's GVA → SPA software solution:
Shadow Page Tables
 - Hypervisor owns real CPU page tables
 - GVA → SPA
 - These tables shadow the guest's page tables
- Hypervisor demand faults in entries
 - Initially all zero, that is, not-present
 - Hypervisor walks guest's page tables to convert GVA → GPA
 - Hypervisor maintains internal tables to convert GPA → SPA
- CPU honors access rights (r,w,NX) set in shadow tables
 - These typically reflect guest page table access bits, but don't have to

Address Space Concepts (4/5)

Shadow Page Tables (SPT)

- This is transparent to guests
 - Hypervisor intercepts TLB/address translation instructions
 - `invlpg`
 - `mov to/from cr0, 3, 4`
 - SPTs entries removed on flushes
 - Guest always sees its own CR3
 - Overall, CPU behaves like it has a very large TLB
- Some common OS operations are more expensive
 - One example: A loop calling `invlpg` multiple times
 - Solution: **OS Enlightenment**
 - Make OS aware of hypervisor
 - More on this later

Address Space Concepts (5/5)

Overlay pages

- The hypercall page is an example of an **overlay** page
 - These pages “overlay” the guest’s normal GPA space
- Overlaid page (RAM, etc.) is “obscured” and unreachable
 - But uncovered when the overlay page is disabled or moved
 - Principle is similar to APIC in PC
- Small number of these in the design
 - Hypervisor chooses (undefined) order in case of overlapping overlay pages

Time Virtualization

Three types of time

- **Calendar time**
 - Affected by Daylight Savings changes
 - Source is parent-created virtual RTC device
- **Machine time**
 - Unaffected by Daylight Savings changes
 - 5 seconds in the future, etc.
 - Sources
 - Per-VP virtualized APIC timer (periodic or single-shot)
 - Four per-VP SynIC timers (periodic or single-shot)
 - Per-partition constant-rate monotonically-increasing reference counter
- **Scheduling time**
 - How long has this processor been scheduled

Time Virtualization

Design Choice

- How to handle RDTSC?
 - When a VP is intercepted, a single instruction can appear to take a long time – namely, the time it takes to enter the hypervisor, perform actions, and return to a guest
- TSC is recorded and can be modified in guest control structure (VMCS/VMCB)

“Allow it to advance naturally”

- Just leave it alone
- But...
- A VP can be rescheduled on a different LP, whose TSC could be smaller
- Can't allow TSCs to jump backwards in time

“Modify it to appear unchanged”

- On entry into the Hv, record guest TSC.
- On return to guest, reload original TSC value minus some amount
- But...
- Never know how long the return instruction will take (caches!)
- Still observable at a certain granularity

Some software depends on knowing cycle counts between instruction blocks (video/audio codecs)

So, we allow it to advance naturally, with a guarantee that it will never appear to go backwards on a given VP

Virtualization Support (1/5)

Intercepts

- A parent partition can install **intercepts** for certain child events
- Intercepts are triggered by child VP actions
 - Accessing I/O ports
 - Accessing MSR's
 - Exceptions
 - Etc.
- The hypervisor sends the parent an **intercept message**
 - The VP is left in a **suspended** state
- The virtualization stack in the parent partition must
 - Resolve the issue
 - Resume the VP

Virtualization Support (2/5)

Intercepts

- Intercepts can be installed for GPAs as well
 - GPAs = Guest Physical Addresses per earlier slide
 - Various combinations of “read/write/execute”
- Uses of intercepts
 - Simulating hardware
 - Profiling
 - Monitoring
 - Page sharing (copy-on-write)

Virtualization Support (3/5)

Virtual processors

- The parent can set any processor register
 - Virtualization stack would do this to emulate an instruction
 - Multiple registers can be set at once, including...
 - A pseudo “resume VP” register
- Registers also include
 - The basics
 - General-purpose registers
 - Selectors, MMX, XMM, CRn’s, DRn’s, xxTR’s
 - MSRs
 - Architecture-defined: TSC, EFER, APIC base, etc.
 - Hypervisor-defined: SynIC, hypercall, etc.
 - x86 oddities
 - In NMI handler
 - In Interrupt shadow (pop ss)
 - Etc.

Virtualization Support (4/5)

Memory

- The parent virtualization stack can
 - Read & write to guest memory
 - Map and unmap guest memory
 - Restrict access
 - r/w/nx
 - Install intercepts on memory
 - Map shared pages between two children
- Most operations take GPAs
 - A parent can ask for VAs to be translated though

Virtualization Support (5/5)

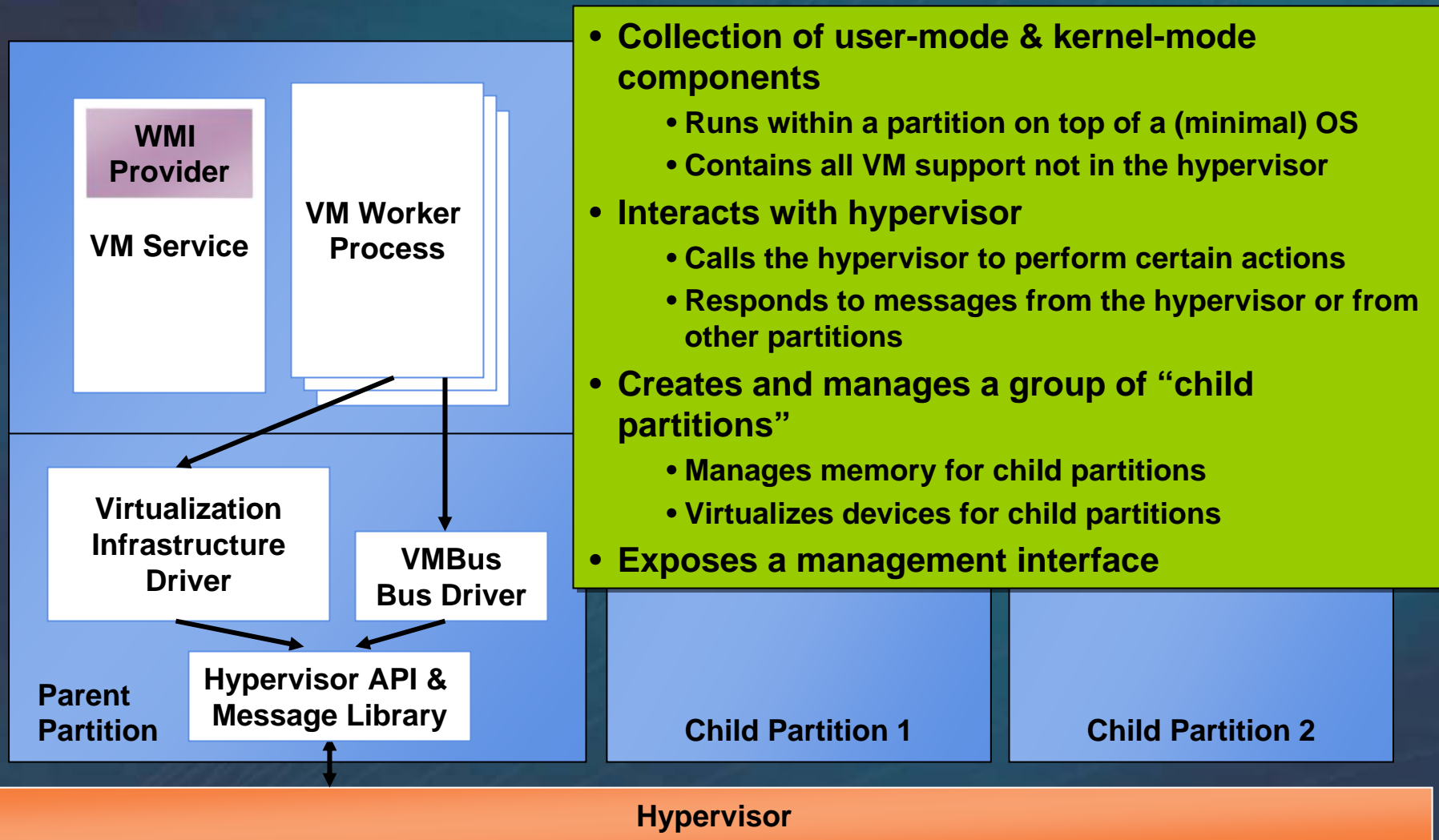
Local APIC

- The hypervisor virtualizes the local APIC
 - True for all guests
- Virtualized APIC can differ slightly from physical APIC
 - Only some instructions supported
 - Operands must be four-byte aligned
 - APIC base MSR can be implemented as a global MSR
 - One write affects all processors at once
 - APIC timer in periodic mode may be less accurate
- Parent can inject virtual interrupts into child
 - Allows parent to emulate a legacy 8259 PIC
 - Edge- or level-triggered
 - Parent can install intercepts on APIC EOI register
 - Find out when interrupt is dismissed

Virtualization Stack

- Portion of traditional hypervisor that has been “pushed up and out” to make a micro-hypervisor
- Runs within a “parent” partition
- Manages a set of “child” partitions
- Handles intercepts passed up by hypervisor
- Includes
 - Legacy device emulation
 - Fast device access
 - VM lifecycle management (start, stop, save, restore)
 - VM management APIs

Windows Virtualization Stack



Partition Lifecycle

- Parent
 - Creates child partition
 - Creates VPs for child partition
 - Sets initial register values for child VPs
 - Sets up child's GPA space
 - GPA → SPA mappings, memory contents
 - Installs intercept handlers
 - I/O ports, memory, fault handlers, etc.
 - Creates ports and connections for
 - parent ↔ child I/O
 - Responds to intercept messages
 - Terminates, deletes child
- Parent deposits memory in child's pool when required

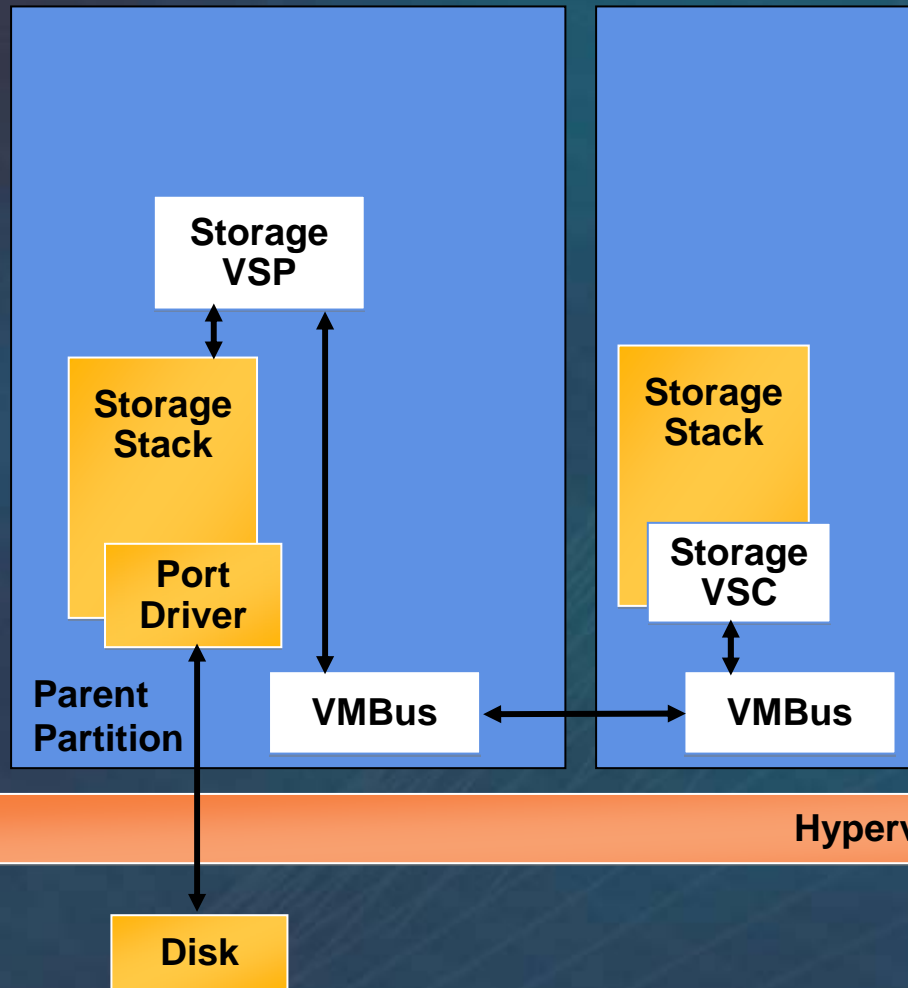
Device Virtualization (1/2)

- Method for sharing hardware efficiently
 - No emulation
- Physical devices controlled by existing device drivers
 - No new device drivers required
- Virtualization Service Provider (VSP)
 - Runs within parent partition (or other partition that owns the hardware device)
 - Talks to device driver
 - Acts as multiplexer, offering hardware services

Device Virtualization (2/2)

- Virtualization Service Clients(VSC)
 - Runs within child partition
 - Consumes service
- VSP/VSC pair per device type
 - Protocol is specific to device type, but is generally OS-agnostic
- Microsoft-provided VSP/VSC pairs
 - Storage, networking, video, input, USB

Device Virtualization



- **Physical devices**
 - Managed by traditional driver stacks
- **Virtualization service providers (VSPs)**
 - Virtualize a specific class of device (e.g. networking, storage, etc.)
 - Expose an abstract device interface
 - Run within the partition that owns the corresponding physical device
- **Virtualization service clients (VSCs)**
 - Consume virtualized hardware service
- **VMBus**
 - Software “bus” (enumeration, hot plug, etc.)
 - Enables VSPs and VSCs to communicate efficiently
 - Uses memory sharing and hypervisor IPC messages

Security Design Assumptions

- Guests are considered adversarial.
- Adversarial code in the guest will run in all available processor modes, rings, and segments.
- Adversarial software executing in a guest will be able to detect that it is running on a hypervisor and determine the specific version of that hypervisor.
- The interface from a guest to the hypervisor will be well documented and widely available to attackers.
- The internal design of the hypervisor will be well understood by attackers through public means as well as reverse engineering.
- All hypercalls and interceptable events can be attempted by an adversarial guest.
- Root must be trusted by hypervisor; parent must be trusted by children.

Security Goals

- Strong isolation between partitions at machine level
- Protect confidentiality and integrity of guest data
- Separation
 - Unique hypervisor resource pools per guest
 - Separate worker processes per guest
 - Guest-to-parent communications over unique channels
- Non-interference
 - Guests cannot affect the contents of other guests, parent, hypervisor
 - Guest computations protected from other guests
 - Guest-to-guest communications not allowed through VM interfaces

Security Non-Goals

- Things we don't do in Windows Server Virtualization*
 - Mitigate hardware bleed-through (inference attacks)
 - Guarantee availability
 - Protect children from their parent
 - Mitigate covert channels
 - Protect the hypervisor from root
 - Provide support for trusted hardware
 - TPM, Device Assignment, DMA protection, Secure Launch

*at least, not in this version

Hypervisor Security Model

- Memory
 - GPA to partition map maintained in Hv
 - Parent Child ownership model on memory
 - Can supersede access rights in guest page tables (R, W, X)
- CPU
 - Hardware guarantees cache & register isolation, TLB flushing, instruction interception
- I/O
 - Hypervisor enforces Parent policy for all guest access to I/O ports
 - WSV v1 policy is guests have no access to real hardware
- Hypervisor Interface
 - Partition privilege model
 - Guests access to hypercalls, instructions, MSRs with security impact enforced based on Parent policy
 - WSV v1 policy is guests have no access to privileged instructions

WSV Security Characteristics

- Hypervisor maintains own address space separate from any “guest”
- Guest addresses != Hypervisor addresses
- No 3rd party code in the Hypervisor
- Limited number of channels, code paths from guests to hypervisor
- Guest to guest communication is channeled through message passing mechanism
- Parent can map shared memory between itself and children
- Root partition controls DMA hardware

Deployment Considerations

- Why two virtual machines can't have the same degree of isolation as two physical machines:
 - Inference Attacks
 - Covert Channels
- Not recommended to host two VMs of vastly differing trust levels on the same system
 - e.g. a front-end web server and a certificate server
- Minimize the Root Partition
 - Don't run arbitrary apps, no web surfing
 - Run your apps and services in guests
 - Ideally only connected to a back-end management network
 - Only expose guests to internet traffic

Future Security Benefits

- Many types of virtualization (app, OS, machine) each with increasing levels of isolation
- Powerful tool for virus isolation and analysis
- Improved forensic capability for compromised operating systems
- Investments in OS hardening through hypervisor features
- Potential for greater intra-OS isolation (e.g. Ring 0 separation of drivers)
- VMs can be leveraged for hosting security appliances

Security Challenges

- VM to VM network monitoring
- Managing VM OS patch levels
- Leakage of information between partitions due to shared hardware
- Larger attack surface than air-gapped machines
- Threat of malicious, unauthorized hypervisors (hypervirus, hyperjacking)

Conclusion

- Hypervisors kick ass.
- Preview will be available soon (weeks!)
- We want your feedback

Microsoft[®]

Your potential. Our passion.[™]

© 2007 Microsoft Corporation. All rights reserved.

This presentation is for informational purposes only. Microsoft makes no warranties, express or implied, in this summary.

Black Hat 2007