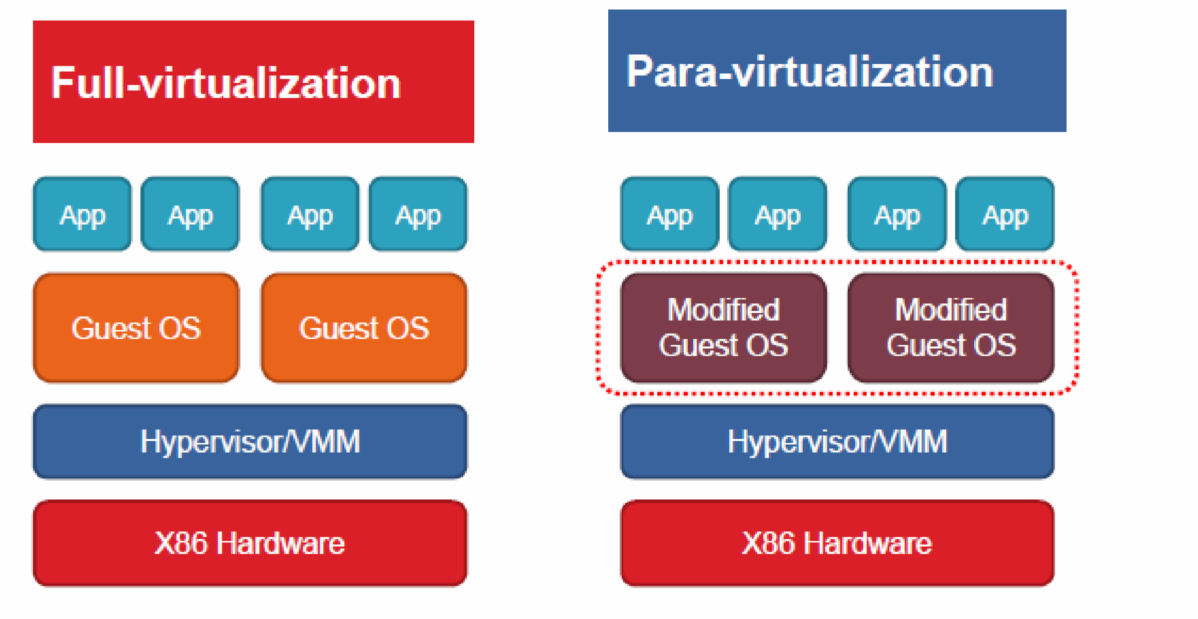
Xen and The Art of Virtualization

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt & Andrew Warfield

SOSP 2003

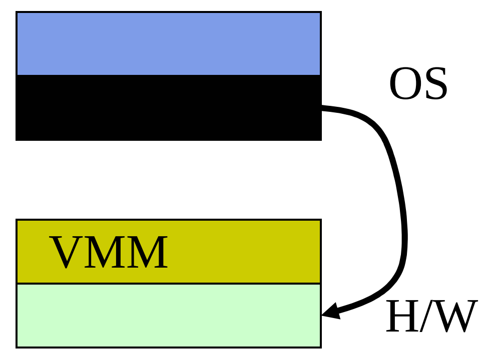
Additional source: Ian Pratt on xen (xen source)
www.cl.cam.ac.uk/research/srg/netos/papers/2005-xen-may.ppt

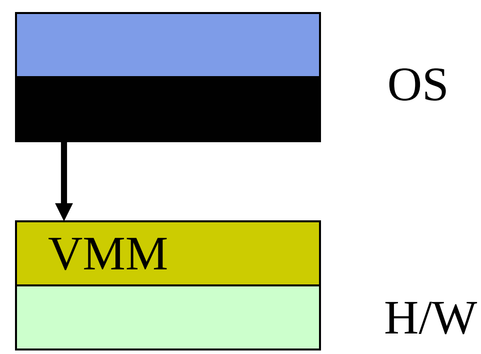
Para virtualization



Virtualization approaches

- Full virtualization
  - OS sees exact h/w
  - OS runs unmodified
  - Requires virtualizable architecture or work around
  - Example: Vmware
- Para Virtualization
  - OS knows about VMM
  - Requires porting (source code)
  - Execution overhead
  - Example Xen, denali





The Xen approach

- Support for unmodified binaries (**but not OS**) essential
  - Important for app developers
  - Virtualized system exports has same Application Binary Interface (ABI)
- Modify guest OS to be aware of virtualization
  - Gets around problems of x86 architecture
  - Allows better performance to be achieved
- Expose some effects of virtualization
  - Translucent VM OS can be used to optimize for performance
- Keep hypervisor layer as small and simple as possible
  - Resource management, Device drivers run in privileged VMM
  - Enhances security, resource isolation

Paravirtualization

- Solution to issues with x86 instruction set
  - Don't allow guest OS to issue sensitive instructions
  - **Replace** those sensitive instructions that don't trap to ones that will trap
- Guest OS makes "hypercalls" (like system calls) to interact with system resources
  - Allows hypervisor to provide protection between VMs
- Exceptions handled by registering handler table with Xen
  - Fast handler for OS system calls invoked directly
  - Page fault handler modified to read address from replica location
- Guest OS changes largely confined to arch-specific code
  - Compile for ARCH=xen instead of ARCH=i686
  - Original port of Linux required only 1.36% of OS to be modified

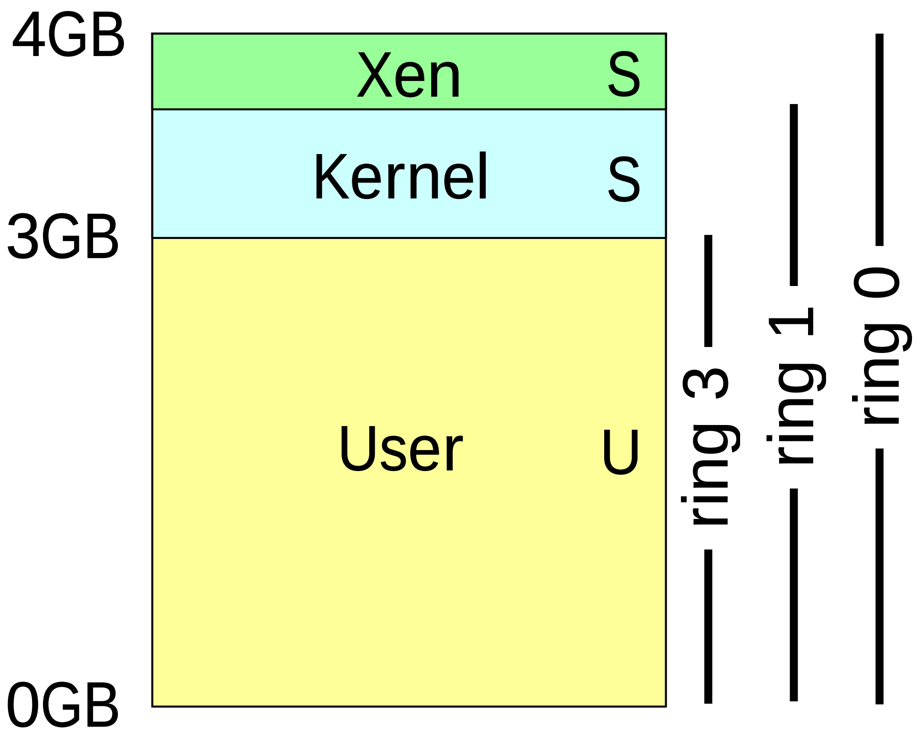
Para-Virtualization in Xen

- Arch xen_x86 : like x86, but Xen hypercalls required for privileged operations
  - Avoids binary rewriting
  - Minimize number of privilege transitions into Xen
  - Modifications relatively simple and self-contained
- Modify kernel to understand virtualized environment.
  - Wall-clock time vs. virtual processor time
    - Xen provides both types of alarm timer
  - Expose real resource availability
    - Enables OS to optimise behaviour

x86 CPU virtualization

- Xen runs in ring 0 (most privileged)
- Ring 1/2 for guest OS, 3 for user-space
  - General Processor Fault if guest attempts to use privileged instruction
- Xen lives in top 64MB of linear address space
  - Segmentation used to protect Xen as switching page tables too slow on standard x86
- Hypercalls jump to Xen in ring 0
- Guest OS may install ‘fast trap’ handler
  - Direct user-space to guest OS system calls
- MMU virtualisation: shadow vs. direct-mode

x86_32



- Xen reserves top of VA space
- Segmentation protects Xen from kernel
- System call speed unchanged

- Xen 3.0 now supports >4GB mem with Processor Address Extension (64 bit etc)

Xen VM interface: CPU

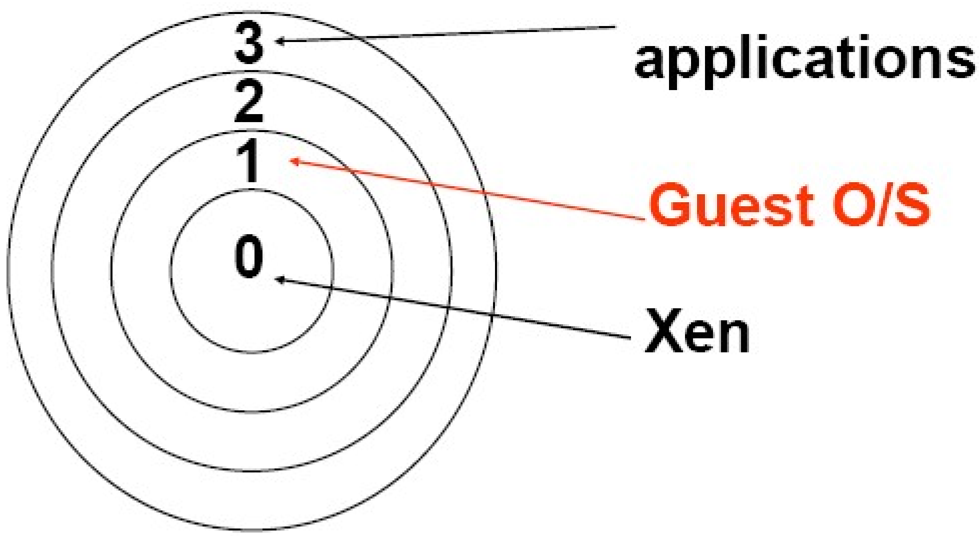
- CPU
  - Guest runs at lower privilege than VMM
  - Exception handlers must be registered with VMM
  - Fast system call handler can be serviced without trapping to VMM
  - Hardware interrupts replaced by lightweight event notification system
  - Timer interface: both real and virtual time

Xen virtualizing CPU

- Many processor architectures provide only 2 levels (0/1)
- Guest and apps in 1, VMM in 0
- Run Guest and app as separate processes
- Guest OS can use the VMM to pass control between address spaces
- Use of software TLB with address space tags to minimize CS overhead

XEN: virtualizing CPU in x86

- x86 provides 4 rings (even VAX processor provided 4)
- Leverages availability of multiple "rings"
  - Intermediate rings have not been used in practice since OS/2; x86-specific
  - An O/S written to only use rings 0 and 3 can be ported; needs to modify kernel to run in ring 1



CPU virtualization

- Exceptions that are called often:
  - Software interrupts for system calls
  - Page faults
- Improve Allow “guest” to register a ‘fast’ exception handler for system calls that can be accessed directly by CPU in ring 1, without switching to ring-0/Xen
  - Handler is validated before installing in hardware exception table: To make sure nothing executed in Ring 0 privilege.
  - Doesn’t work for Page Fault
  - Only code in ring 0 can read the faulting address from register

Xen

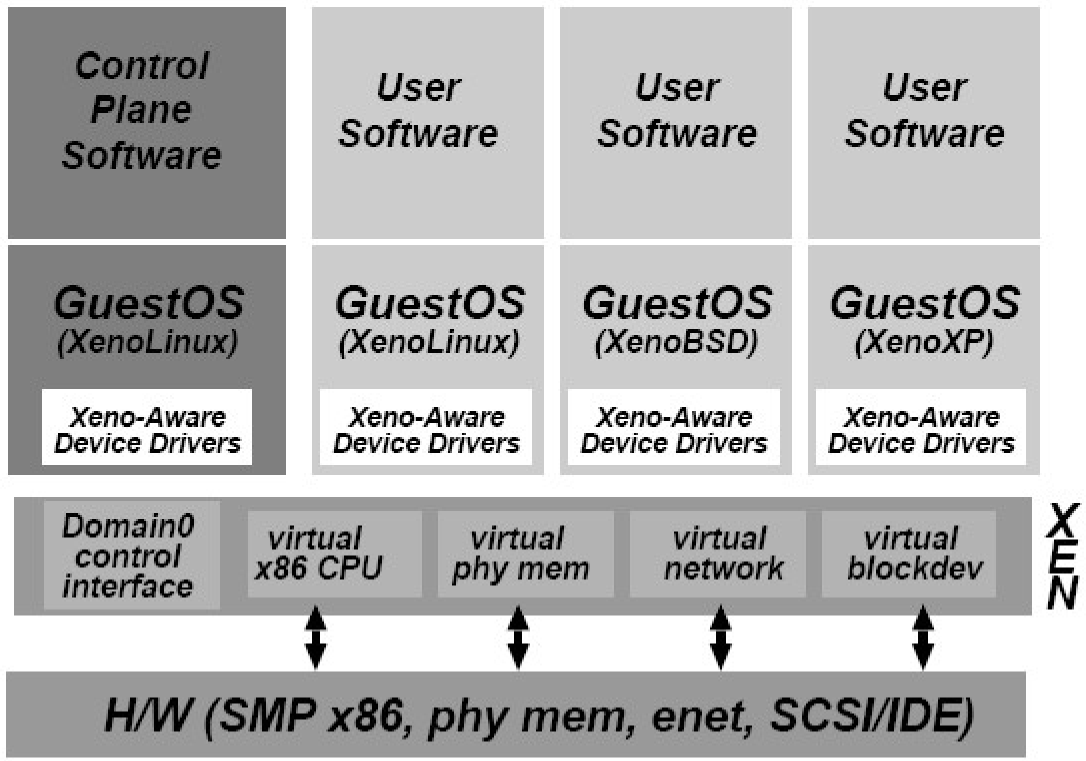


Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including *Domain0* running control software in a XenoLinux environment.

Some Xen hypercalls

- See http://lxr.xensource.com/lxr/source/xen/include/public/xen.h

```
#define __HYPERVISOR_set_trap_table        0
#define __HYPERVISOR_mmu_update            1
#define __HYPERVISOR_sysctl               35
#define __HYPERVISOR_domctl               36
```

Xen VM interface: Memory

- Memory management
  - Guest cannot install highest privilege level segment descriptors; top end of linear address space is not accessible
  - Guest has direct (not trapped) read access to hardware page tables; writes are trapped and handled by the VMM
  - Physical memory presented to guest is not necessarily contiguous

Memory virtualization choices

- TLB: challenging
  - Software TLB can be virtualized without flushing TLB entries between VM switches
  - Hardware TLBs tagged with address space identifiers can also be leveraged to avoid flushing TLB between switches
  - x86 is hardware-managed and has no tags…
- Decisions:
  - Guest O/Ss allocate and manage their own hardware page tables with minimal involvement of Xen for better safety and isolation
  - Xen VMM exists in a 64MB section at the top of a VM’s address space that is not accessible from the guest

Xen memory management

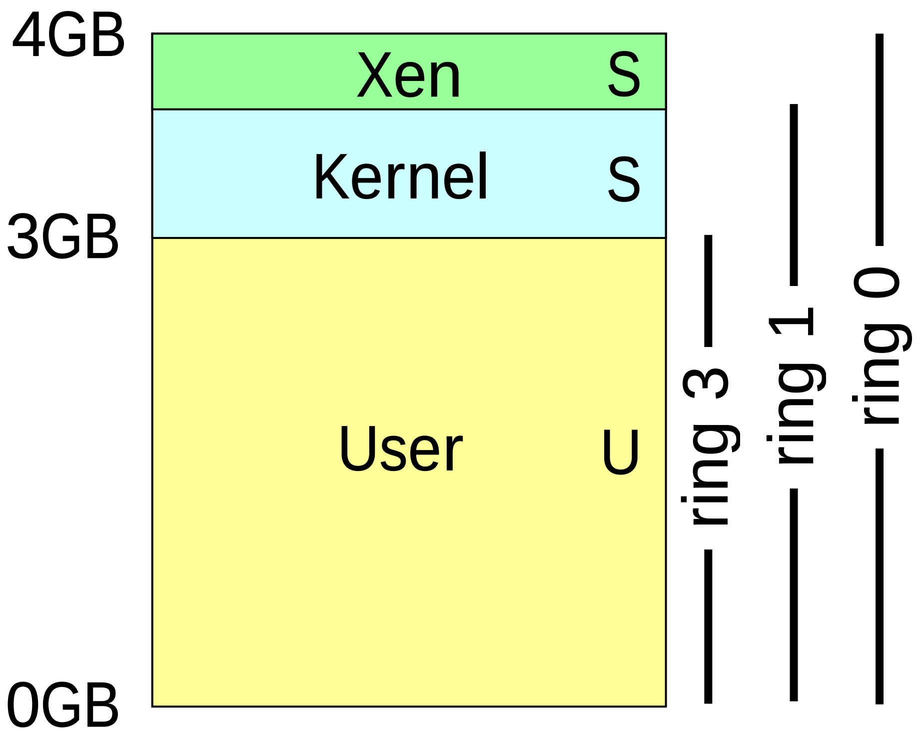
17

- x86 TLB not tagged
  - Must optimise context switches: allow VM to see physical addresses
  - Xen mapped in each VM's address space
- PV: Guest OS manages own page tables
  - Allocates new page tables and registers with Xen
  - Can read directly
  - Updates batched, then validated, applied by Xen

Memory virtualization

- Guest O/S has direct read access to hardware page tables, but updates are validated by the VMM
  - Through “hypercalls” into Xen
  - Also for segment descriptor tables
  - VMM must ensure access to the Xen 64MB section not allowed
  - Guest O/S may “batch” update requests to amortize cost of entering hypervisor

x86_32



- Xen reserves top of VA space
- Segmentation protects Xen from kernel
- System call speed unchanged

- Xen 3.0 now supports >4GB mem with Processor Address Extension (64 bit etc)

Virtualized memory management

- Each process in each VM has its own VAS
- Guest OS deals with real (pseudo-physical) pages, Xen maps physical to machine
- For PV, guest OS uses hypercalls to interact with memory
- For HVM, Xen has shadow page tables (VT instructions help)

VM1

1	2
2	3

1	5
2	6

VM2

1	2
2	3

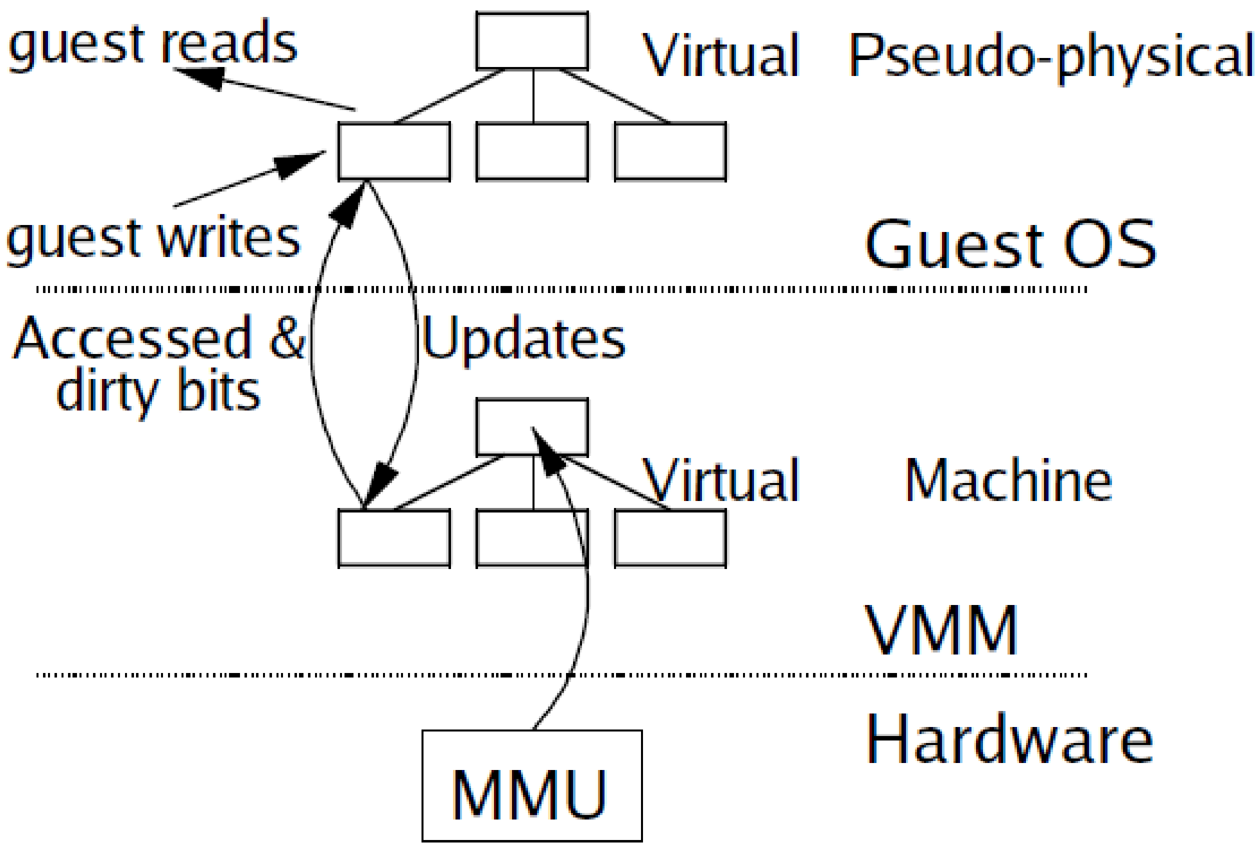
1	5
2	6

2	8
3	12
5	14
6	9
2	4
3	16
5	17
6	18

TLB when VM1 is running

VP	PID	PN
1	1	?
2	1	?
1	2	?
2	2	?

MMU Virtualization: shadow mode



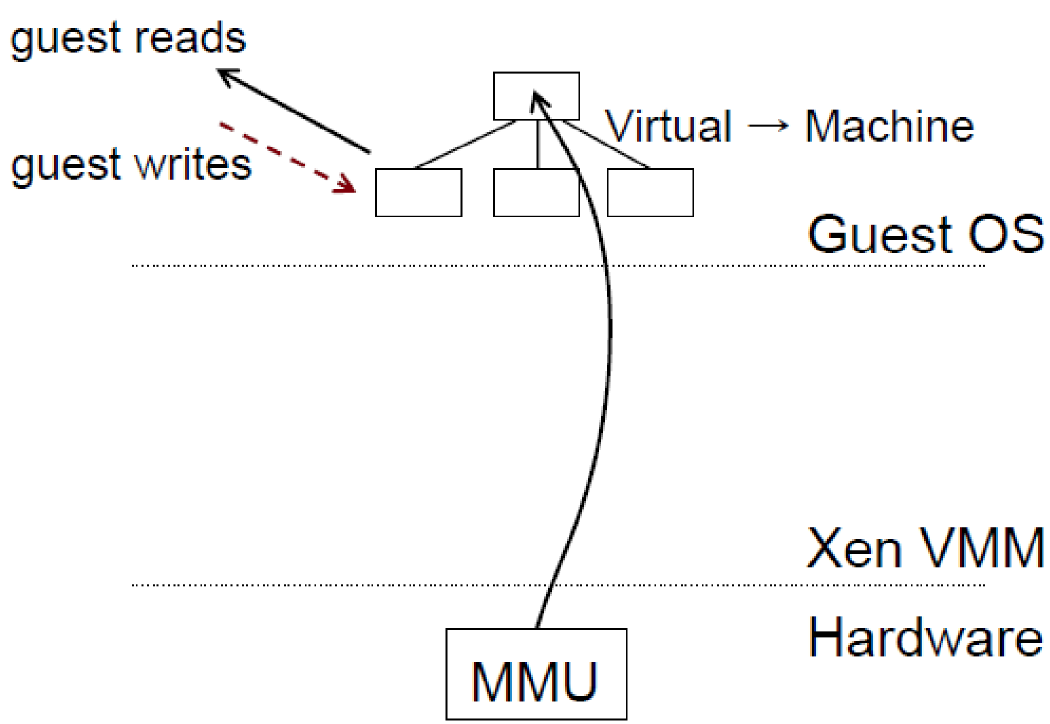
Shadow page table

- Hypervisor responsible for trapping access to virtual page table
- Updates need to be propagate back and forth between Guest OS and VMM
- Increases cost of managing page table flags (modified, accessed bits)
- Can view physical memory as contiguous
- Needed for full virtualization

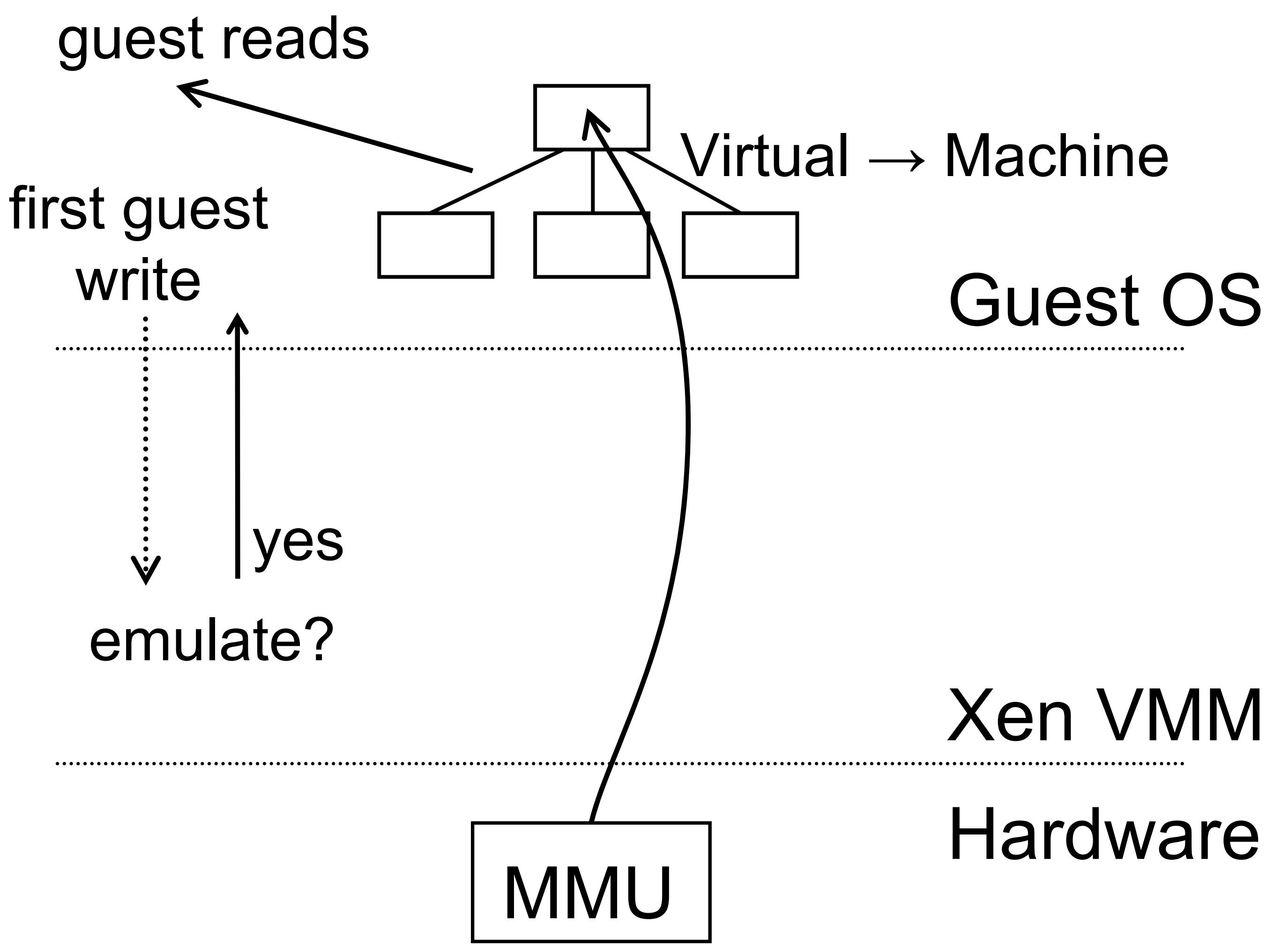
MMU virtualization: direct mode

- Take advantage of Paravirtualization
- OS can be modified to be involved only in page table updates
- Restrict guest OSes to read only access
- Classify Page frames into frames that holds page table
- Once registered as page table frame, make the page frame R_ONLY
- Can avoid the use of shadow page tables

Single PTE update



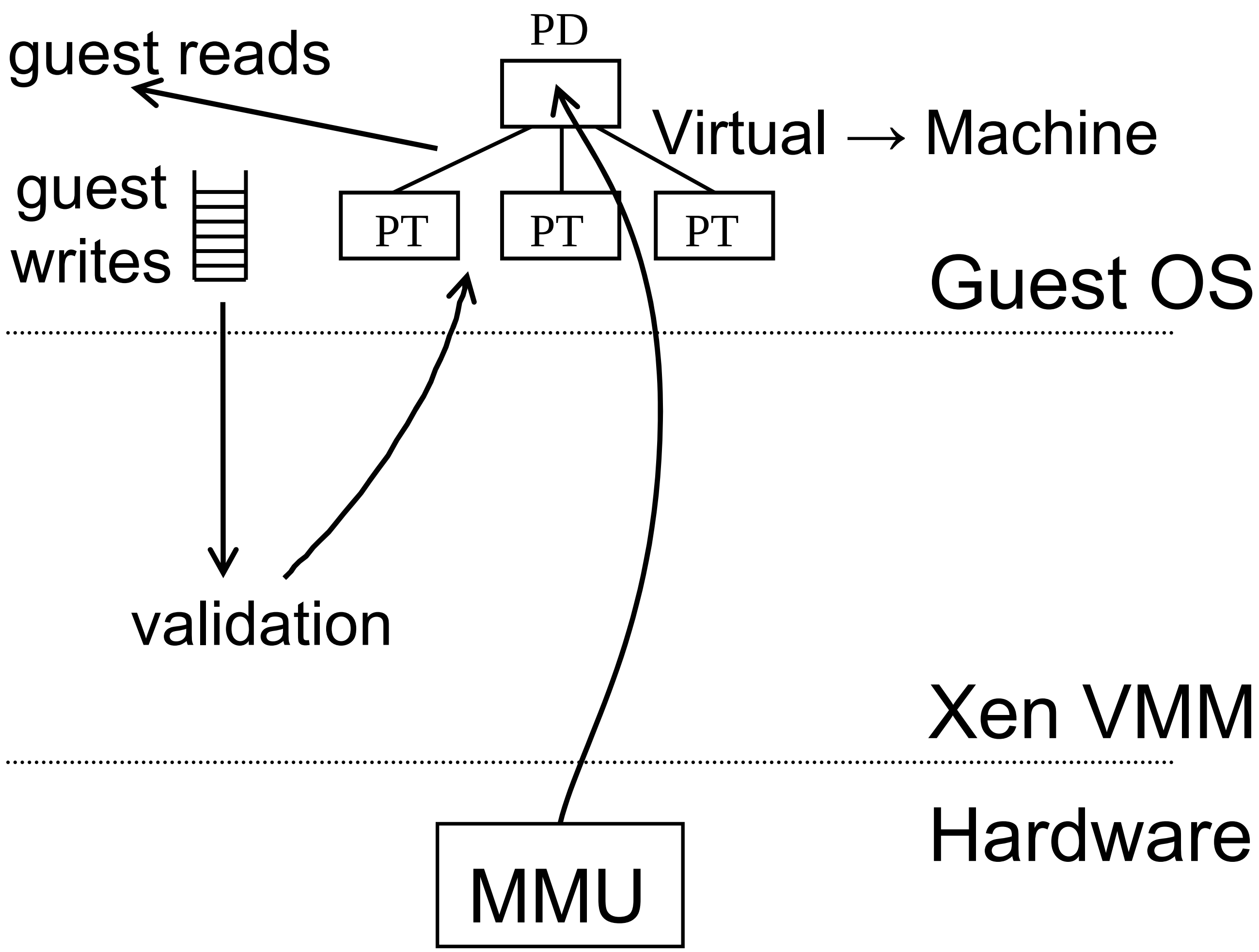
On write PTE : Emulate



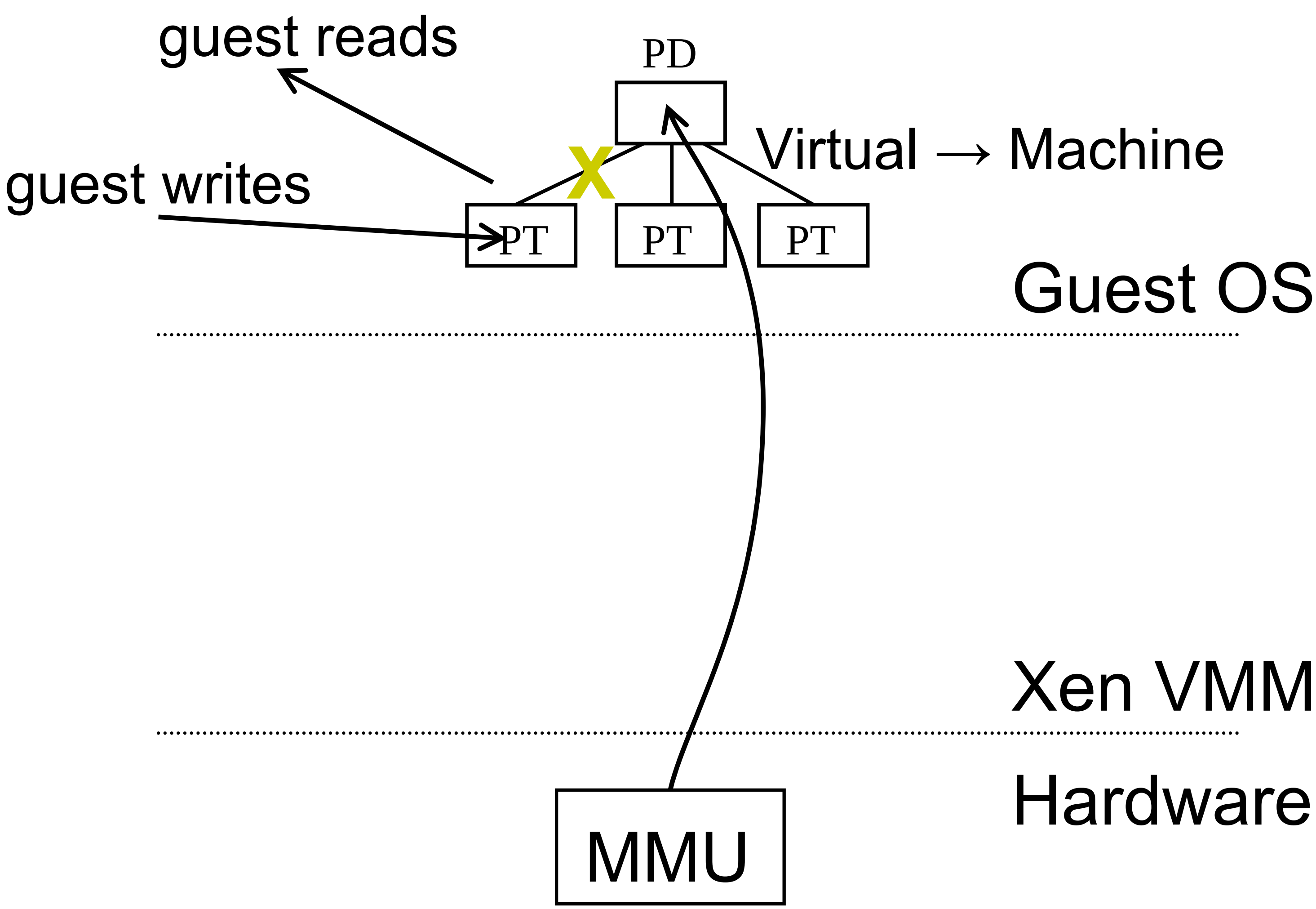
Bulk update

- Useful when creating new Virtual address spaces
- New Process via fork and Context switch
- Requires creation of several PTEs
- Multipurpose hypercall
  - Update PTEs
  - Update virtual to Machine mapping
  - Flush TLB
  - Install new PTBR

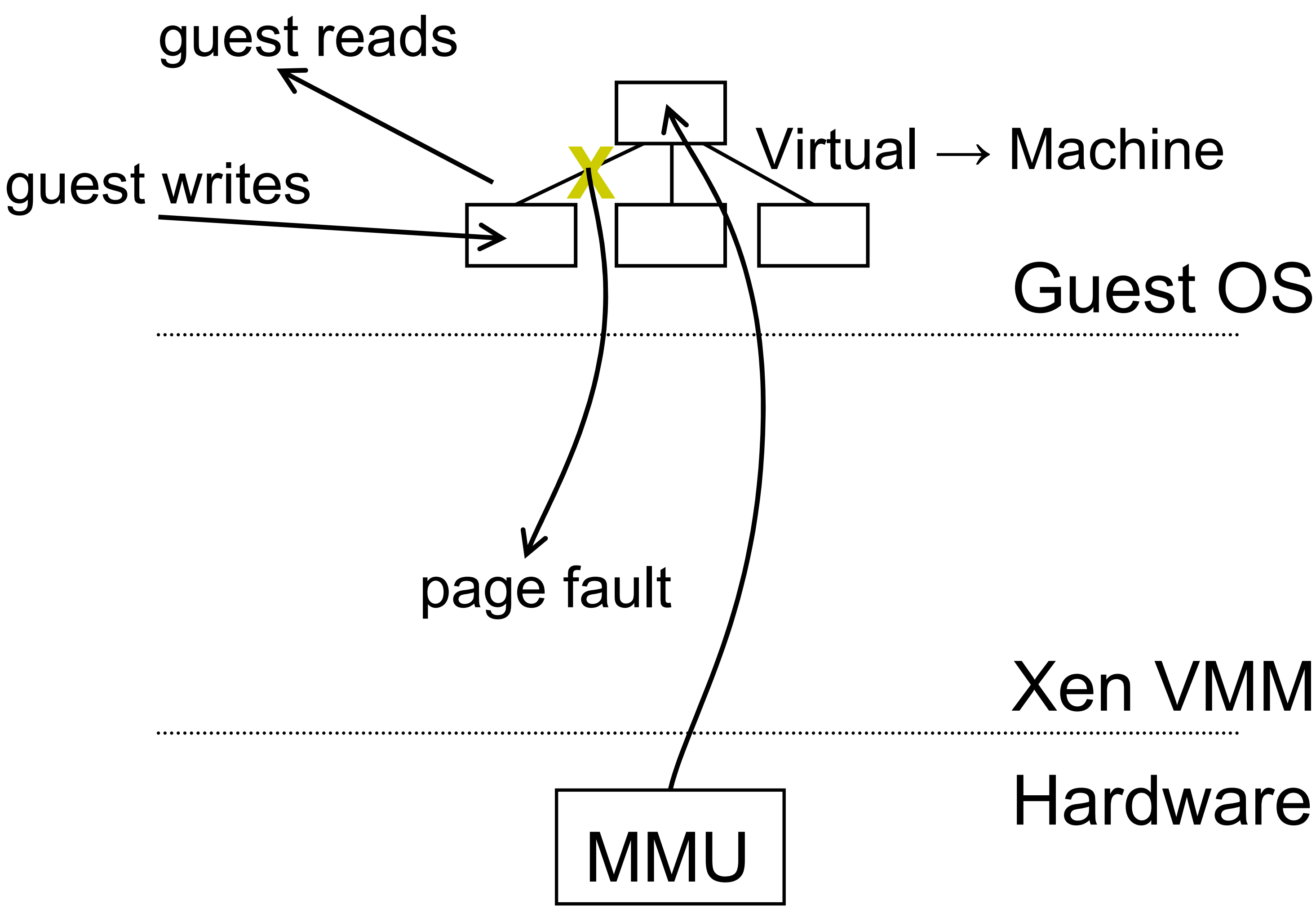
Batched Update Interface



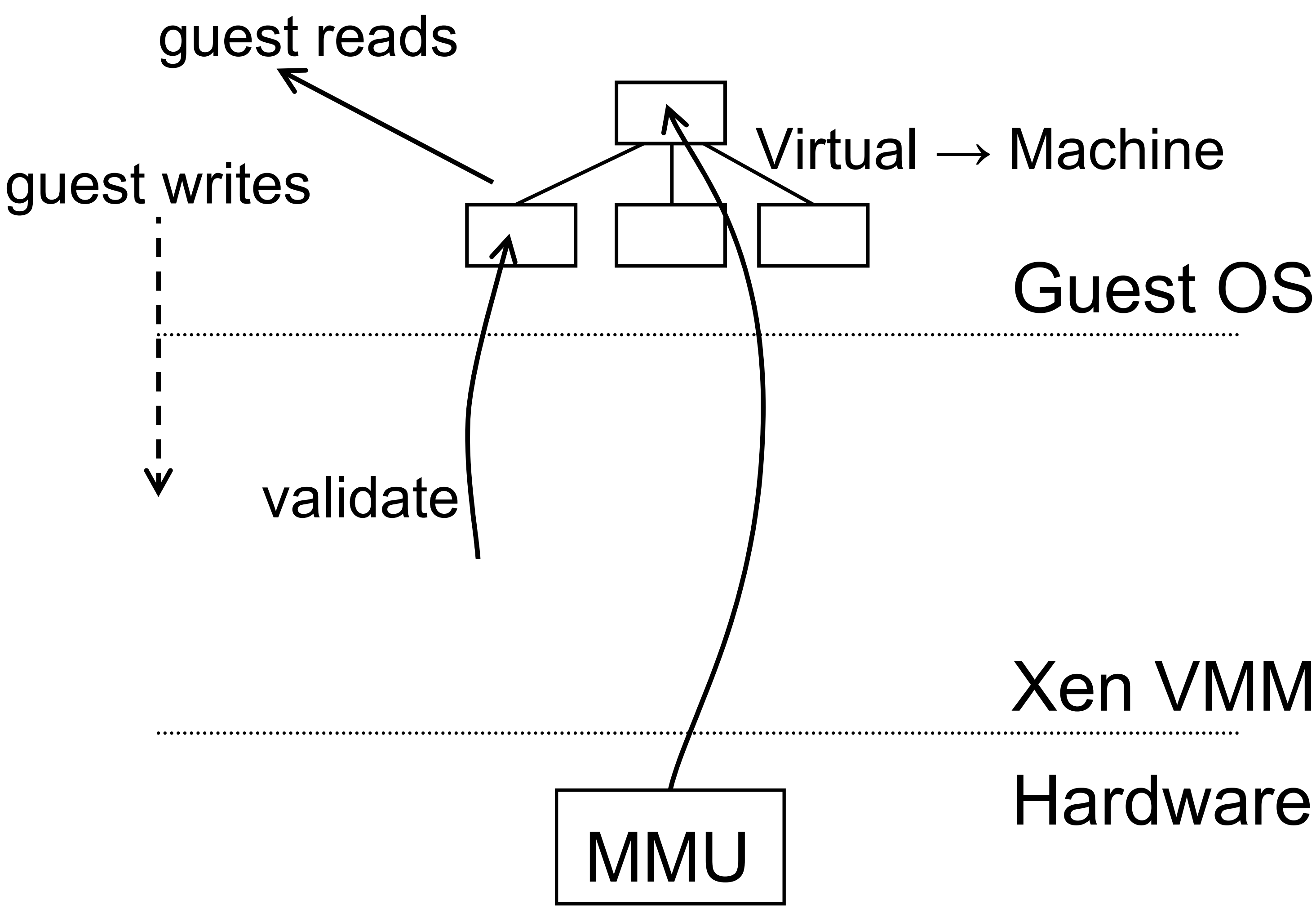
Writeable Page Tables: create new entries



Writeable Page Tables : First Use—validate mapping via TLB



Writeable Page Tables : Re-hook



Physical memory

- Memory allocation for each VM specified at boot
  - Statically partitioned
  - No overlap in machine memory
  - Strong isolation
- Non-contiguous (Sparse allocation)
  - Balloon driver
  - Add or remove machine memory from guest OS

Xen memory management

33

- Xen does not swap out memory allocated to domains
  - Provides consistent performance for domains
  - By itself would create inflexible system (static memory allocation)
- Balloon driver allows guest memory to grow/shrink
  - Memory target set as value in the XenStore
  - If guest above target, free/swap out, then release to Xen
  - If guest below target, can increase usage
- Hypercalls allow guests to see/change state of memory
  - Physical-real mappings
  - “Defragment” allocated memory

Xen VM interface: I/O

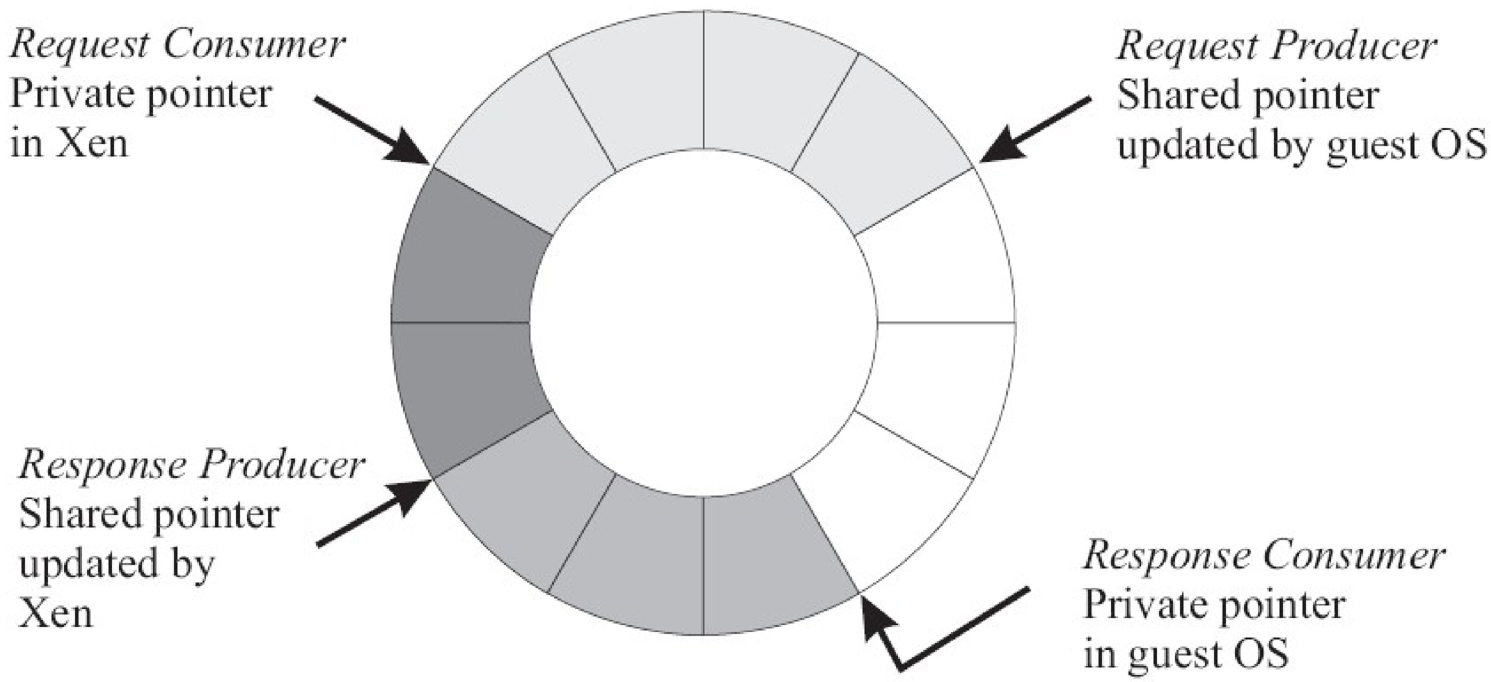
- I/O
  - Virtual devices (device descriptors) exposed as asynchronous I/O rings to guests
  - Event notification is by means of an upcall as opposed to interrupts

I/O

- Handle interrupts
- Data transfer
- Data written to I/O buffer pools in each domain
- These Page frames pinned by Xen

Details: I/O

- I/O Descriptor Ring:



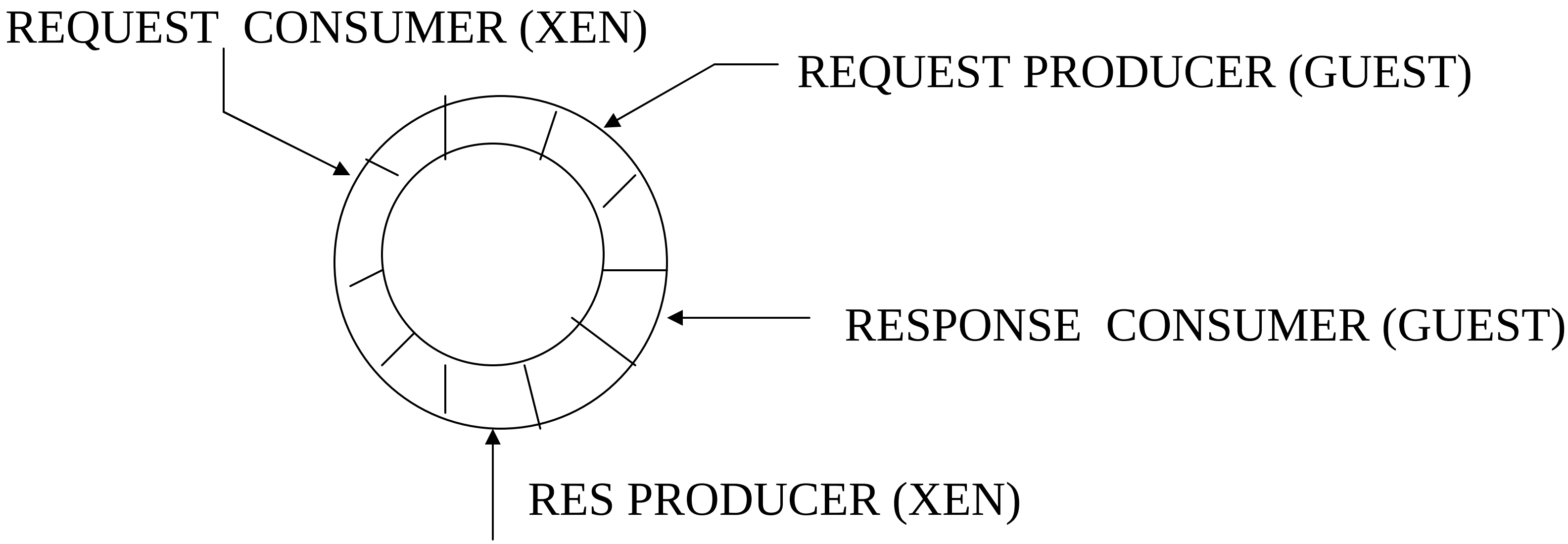
Request queue - Descriptors queued by the VM but not yet accepted by Xen

Outstanding descriptors - Descriptor slots awaiting a response from Xen

Response queue - Descriptors returned by Xen in response to serviced requests

Unused descriptors

I/O rings


REQUEST CONSUMER (XEN)
REQUEST PRODUCER (GUEST)
RESPONSE CONSUMER (GUEST)
RES PRODUCER (XEN)

I/O virtualization

- Xen does not emulate hardware devices
  - Exposes device abstractions for simplicity and performance
  - I/O data transferred to/from guest via Xen using shared-memory buffers
  - Virtualized interrupts: light-weight event delivery mechanism from Xen-guest
    - Update a bitmap in shared memory
    - Optional call-back handlers registered by O/S

Network Virtualization

- Xen models a virtual firewall-router (VFR) to which one or more VIFs of each domain connect
- Two I/O rings: one for send and another for receive
- Policy enforced by a special domain
  - Each direction also has rules of the form (if <pattern> → <action>) that are inserted by domain 0 (management)

Network Virtualization

- Packet transmission:
  - Guest adds request to I/O ring
  - Xen copies packet header, applies matching filter rules
  - Round-robin packet scheduler

Network Virtualization

- Packet reception:
  - Xen applies pattern-matching rules to determine destination VIF
  - Guest O/S required to provide PM for copying packets received
    - If no receive frame is available, the packet is dropped
    - Avoids Xen-guest copies;

Disk Virtualization

- Uses Split driver approach
- Front end, back end drivers
- Front end
  - Guest OSes use a simple generic driver per class
- Domain 0 provides the actual driver per device
- Back end runs in own VM (domain 0)

Disk virtualization

- Domain0 has access to physical disks
  - Currently: SCSI and IDE
- All other domains are offerred virtual block device (VBD) abstraction
  - Created & configured by management software at domain0
  - Accessed via I/O ring mechanism
  - Possible reordering by Xen based on knowledge about disk layout

Disk virtualization

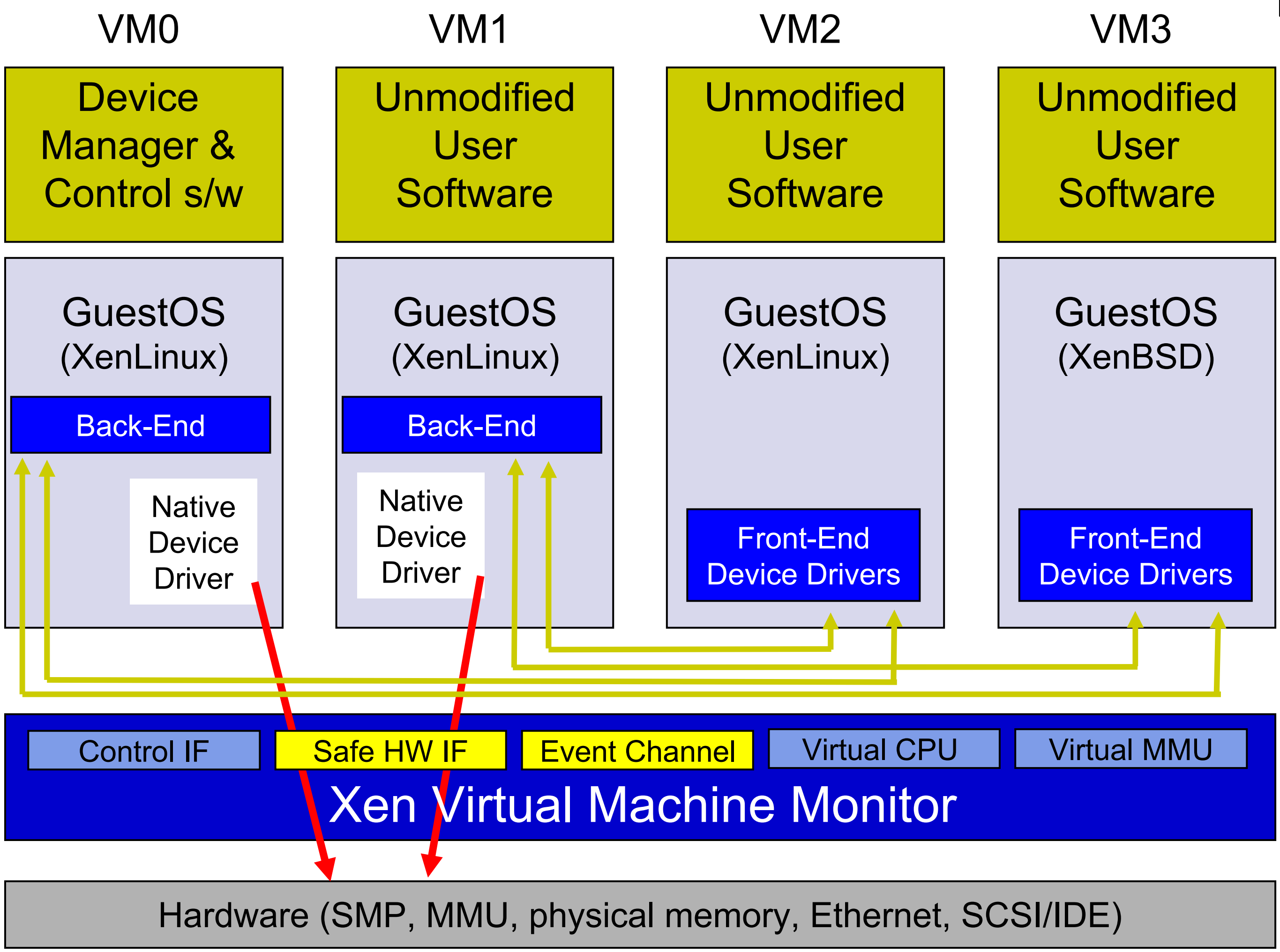
- Xen maintains translation tables for each VBD
  - Used to map requests for VBD (ID,offset) to corresponding physical device and sector address
  - Zero-copy data transfers take place using DMA between memory pages pinned by requesting domain
- Scheduling: batches of requests in round-robin fashion across domains

Advanced features

- Support for HVM (hardware virtualisation support)
  - Very similar to “classic” VM scenario
  - Uses emulated devices, shadow page tables
  - Hypervisor (VMM) still has important role to play
  - “Hybrid” HVM paravirtualizes components (e.g. device drivers) to improve performance

- Migration of domains between machines
  - Daemon runs on each Dom0 to support this
  - Incremental copying used to for live migration (60ms downtime!)

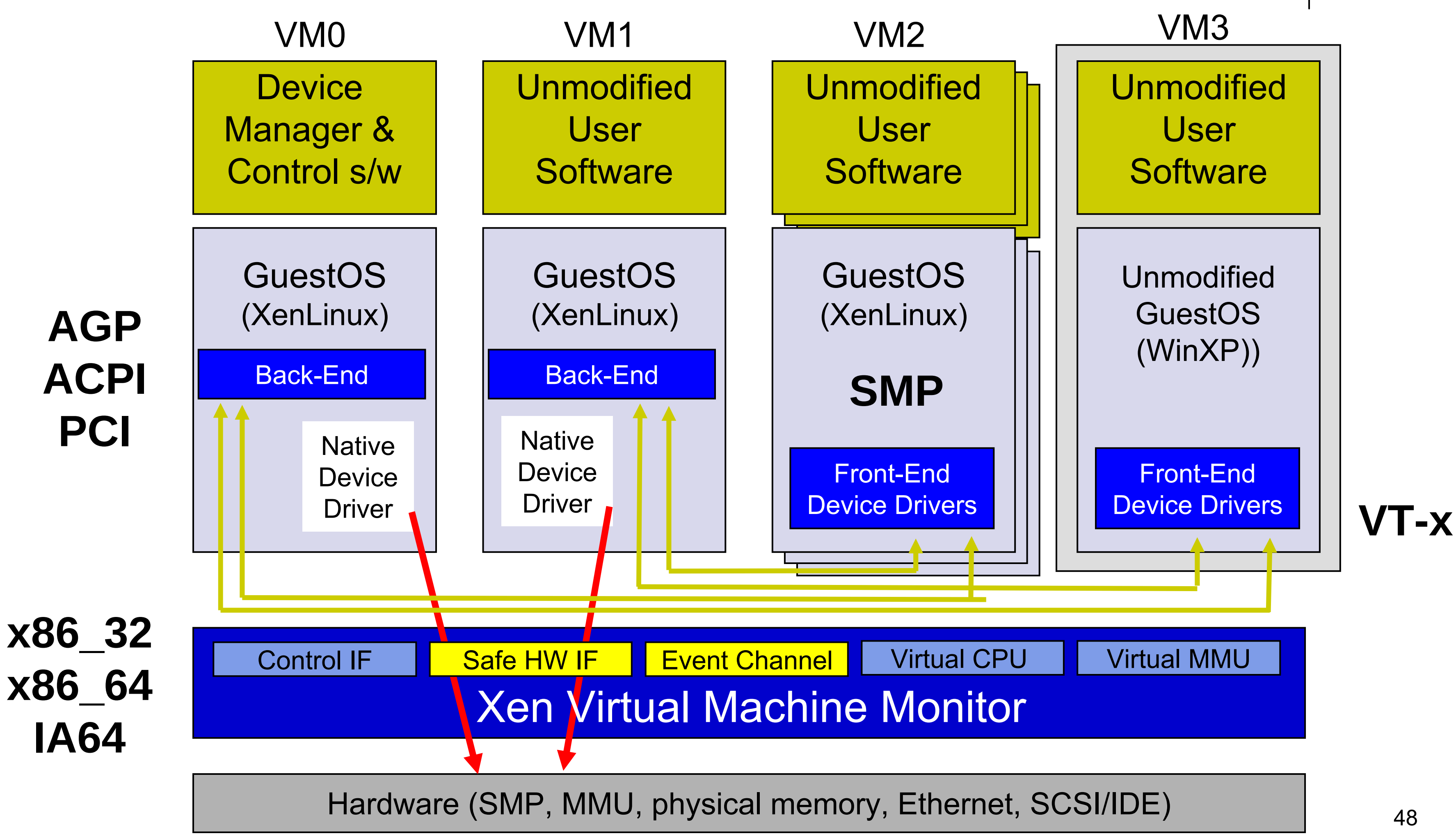
Xen 2.0 Architecture



Xen Today : 2.0 Features

- Secure isolation between VMs
- Resource control and QoS
- Only guest kernel needs to be ported
  - All user-level apps and libraries run unmodified
  - Linux 2.4/2.6, NetBSD, FreeBSD, Plan9
- Execution performance is close to native
- Supports the same hardware as Linux x86
- Live Relocation of VMs between Xen nodes

Xen 3.0 Architecture



Xen 3.0 features

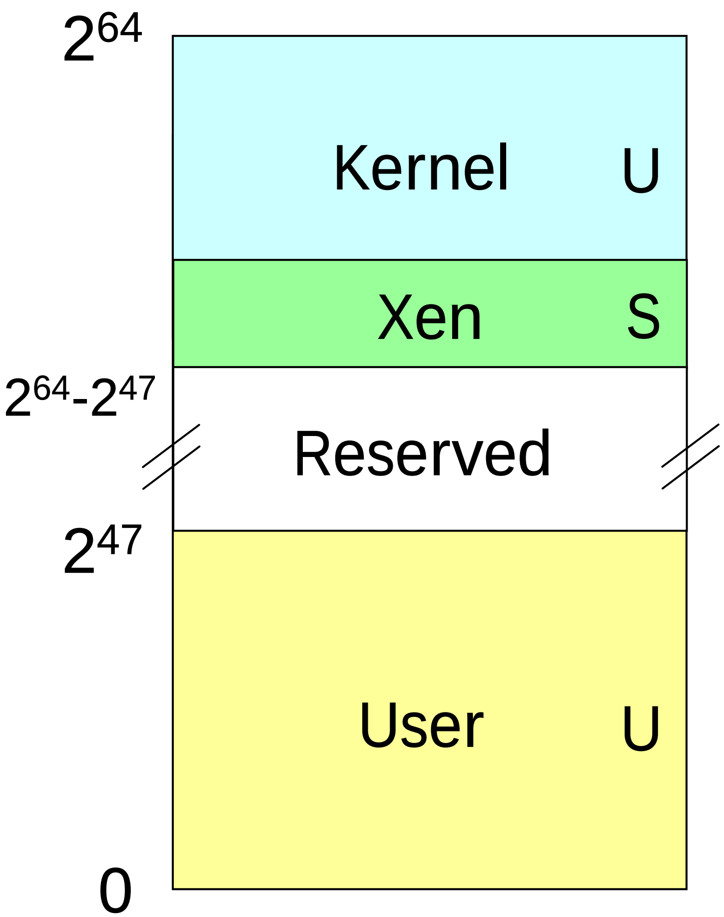
- Support for up to 32-way SMP guest
- Intel® VT-x and AMD Pacifica hardware virtualization support
- PAE support for 32 bit servers with over 4 GB memory
- x86/64 support for both AMD64 and EM64T

- New easy-to-use CPU scheduler including weights, caps and automatic load balancing
- Much enhanced support for unmodified ('hvm') guests including windows and legacy linux systems
- Support for sparse and copy-on-write disks
- High performance networking using segmentation off-load

Xen protection levels in PAE

- x86_64 removed rings 1,2
  - Xen in ring 0
  - Guest OS and apps in ring 3

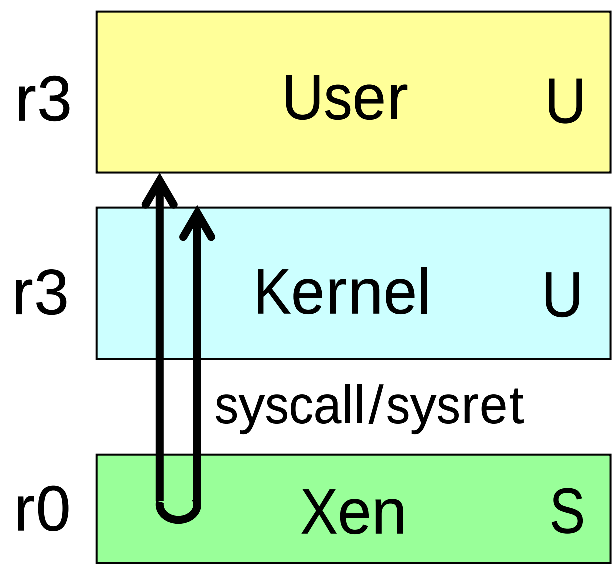
x86_64



- Large VA space makes life a lot easier, but:
- No segment limit support

➔ Need to use page-level protection to protect hypervisor

x86_64



- Run user-space and kernel in ring 3 using different pagetables
  - Two PGD's (PML4's): one with user entries; one with user plus kernel entries
- System calls require an additional syscall/ret via Xen
- Per-CPU trampoline to avoid needing GS in Xen

Additional resources on Xen

- “Xen 3.0 and the art of virtualization”, Presentation by Ian Pratt
- Virtual machines by Jim Smith and ravi nair
- “The definitive guide to the Xen hypervisor” (Kindle Edition), David Chisnall
- The source code: http://lxr.xensource.com/lxr/source/xen/