Xen and the Art of Virtualizations

Chuhan Ouyang

9/24/24

Background

Authors

- Paul Barham
  - Principal Researcher at Microsoft Research Cambridge
  - TensorFlow lead
  - Google's ML infra
  - Xen, Barrelfish multi-kernel OS
- Boris Dragovic
  - Chief Strategy Officer at Hyperoptic
  - McKinsey & Company
  - Xen, XenoTrust
- Xen began at Cambridge University and is being further developed primarily by Citrix.

Timeline - VM/VMM and Microkernels

1981: IBM/370

1997: Disco/VMWare

2002: Denali

2003: Xen

VM/VMM

1981:"OS support for database management"

1980s: Mach

1990s: SPIN, Exokernel

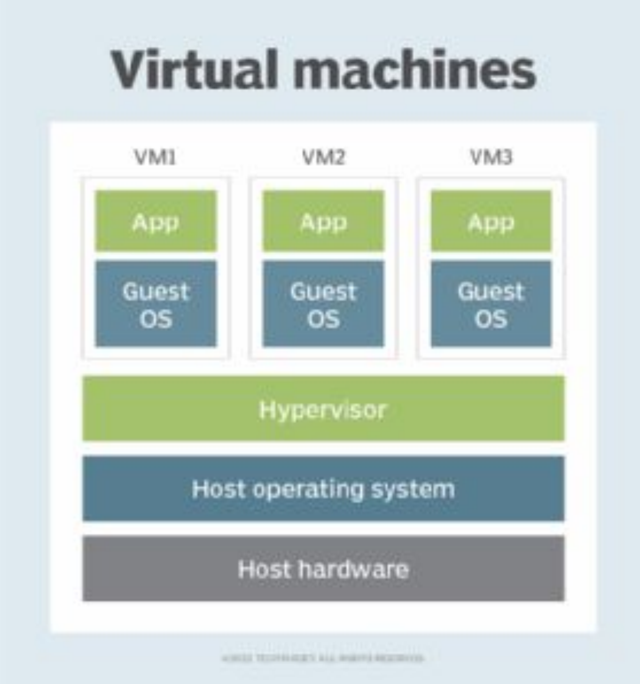
1995: L4

2003: Xen

Microkernels

VM and VMM

- Paper published in 2003
- Computer become powerful to support many virtual machines that each run a separate OS
- Virtual Machine Monitor (VMM)/Hypervisor
    - Enables the creation, running, and management of virtual machines (VMs) on a host system.
- Key Challenges
    - isolate VM from each other
    - support variety of OS (heterogeneity), low performance overhead
- Xen: hypervisor that multiplexes resources at the granularity of an entre OS



Microkernel and VM: Similarity and Differences

VMM transforms the single machine interface into the illusion of many

Microkernel minimizes the kernel and implements whatever possible outside of the kernel

Flexibility
Fault isolation
Maintainability
Restricted interdependencies
Minimality

Software reliability
Data security
Alternative system APIs
Improved mechanism
Minimality

Papers: “Are VMM Microkernels Done Right” (Hand et al., HotOS 2005)

“Are VMM Microkernels Done Right” (Heiser et al., SIGOPS 2006)

Granularity of Multiplexing

- Alternate approach: run multiple applications on the same OS
  - does not support performance isolation
  - difficult to ensure all resource usage is accounted to the correct process
- Xen: multiplexes resources at the granularity of an entre OS
  - performance isolation
  - flexibility: allow multiple OS to coexist
  - drawback: more heavyweight than process-level multiplexing for initializing processes and resource consumption

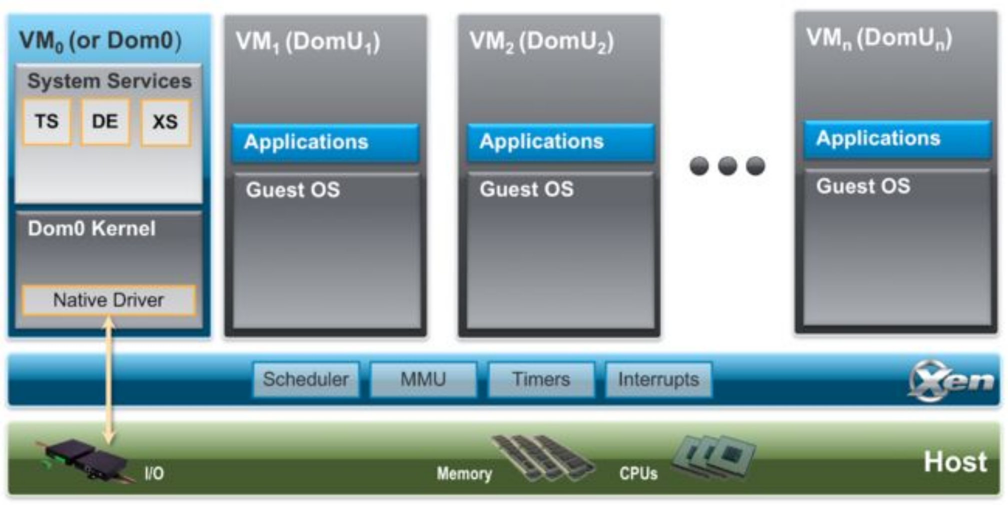
Virtualization Approaches

Traditional VMM: Full Virtualization

- Virtual hardware exposed is functionally identical to the underlying machine
- Benefit
  - OS needs no modifications to be run
- Drawbacks
  - Not compatible with x86
  - Supervisor instruction needs be handled by VMM for correct virtualization
  - Executing instructions with insufficient privilege silently fails
  - High-cost for virtualizing x86 MMU

Xen: Paravirtualization

- Provide virtual machine abstraction similar but not identical to the underlying hardware
- Benefits
  - improved performance
- Drawbacks
  - require modifying the guest OS
  - but no changes to the ABI and no changes to the guest applications



Xen: Design Principles

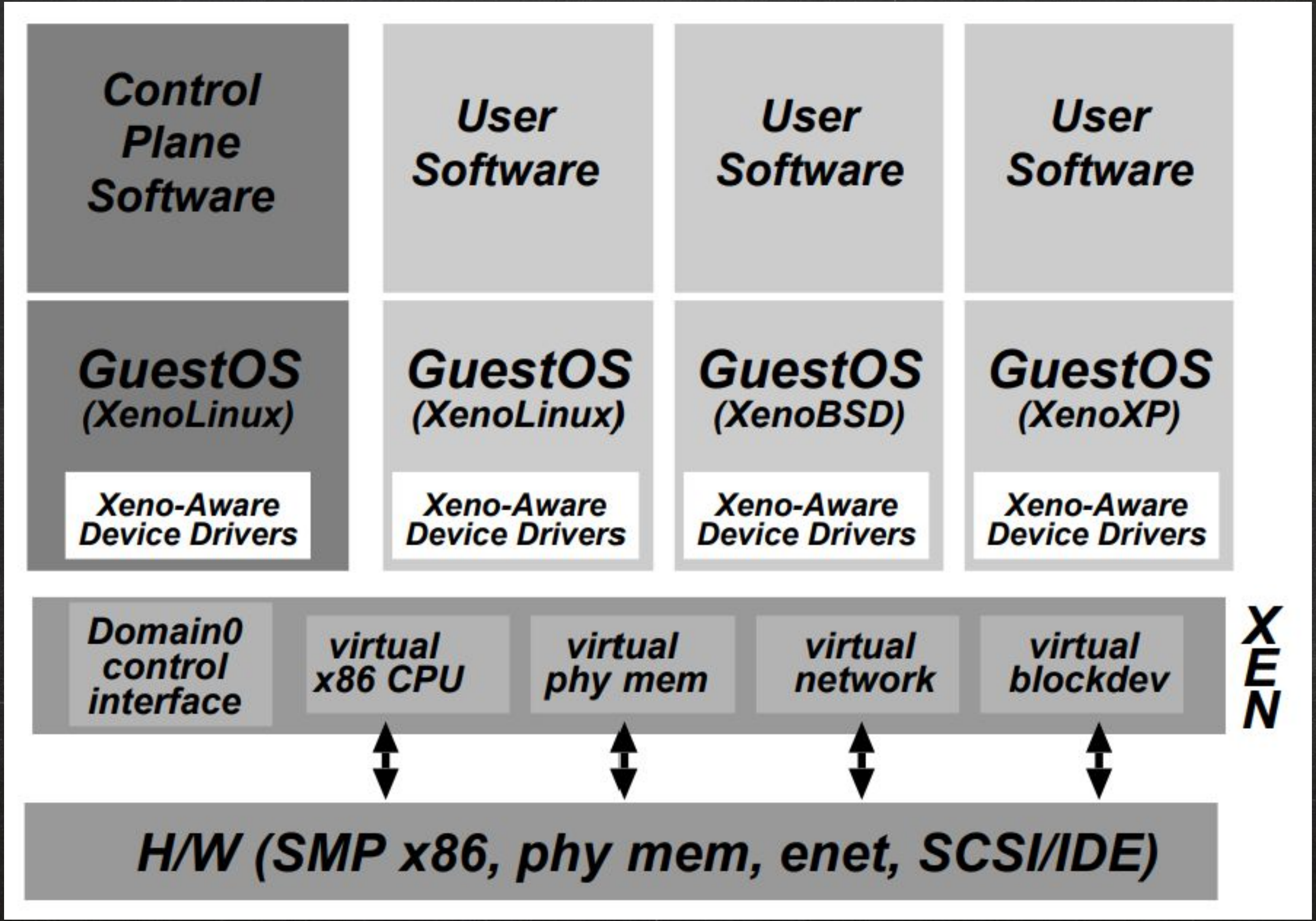
1. Unmodified application binaries is essential. Must virtualize all architectural features required by existing standard ABIs.

2. Supporting full multi-application operating systems. (Denali only supports VM hosting single-user single-application unprotected OS)

3. Paravirtualization for high performance and strong resource isolation.

4. Completely hiding the effects of resource virtualization from guest OSes risks both correctness and performance.

Note: domain = running instance of a VM


Control Plane Software
User Software
User Software
User Software
GuestOS (XenoLinux)
Xeno-Aware Device Drivers
GuestOS (XenoLinux)
Xeno-Aware Device Drivers
GuestOS (XenoBSD)
Xeno-Aware Device Drivers
GuestOS (XenoXP)
Xeno-Aware Device Drivers
Domain0 control interface
virtual x86 CPU
virtual phy mem
virtual network
virtual blockdev
XEN
H/W (SMP x86, phy mem, enet, SCSI/IDE)

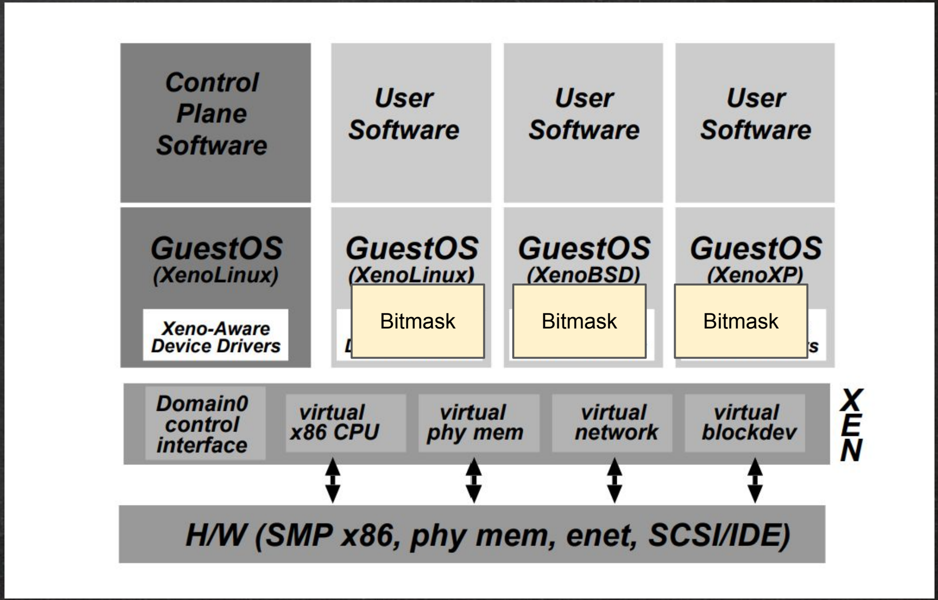
Discussion Questions

1. Besides from the incompatibility associated with x86, is there any other drawbacks to using full virtualization compared to paravirtualization?
2. What are some challenges to support virtualizing heterogeneous OS (mixes of different operating systems)?
3. What is the benefit of this principle : “Completely hiding the effects of resource virtualization from guest OSes risks both correctness and performance.”

Virtualization Mechanisms

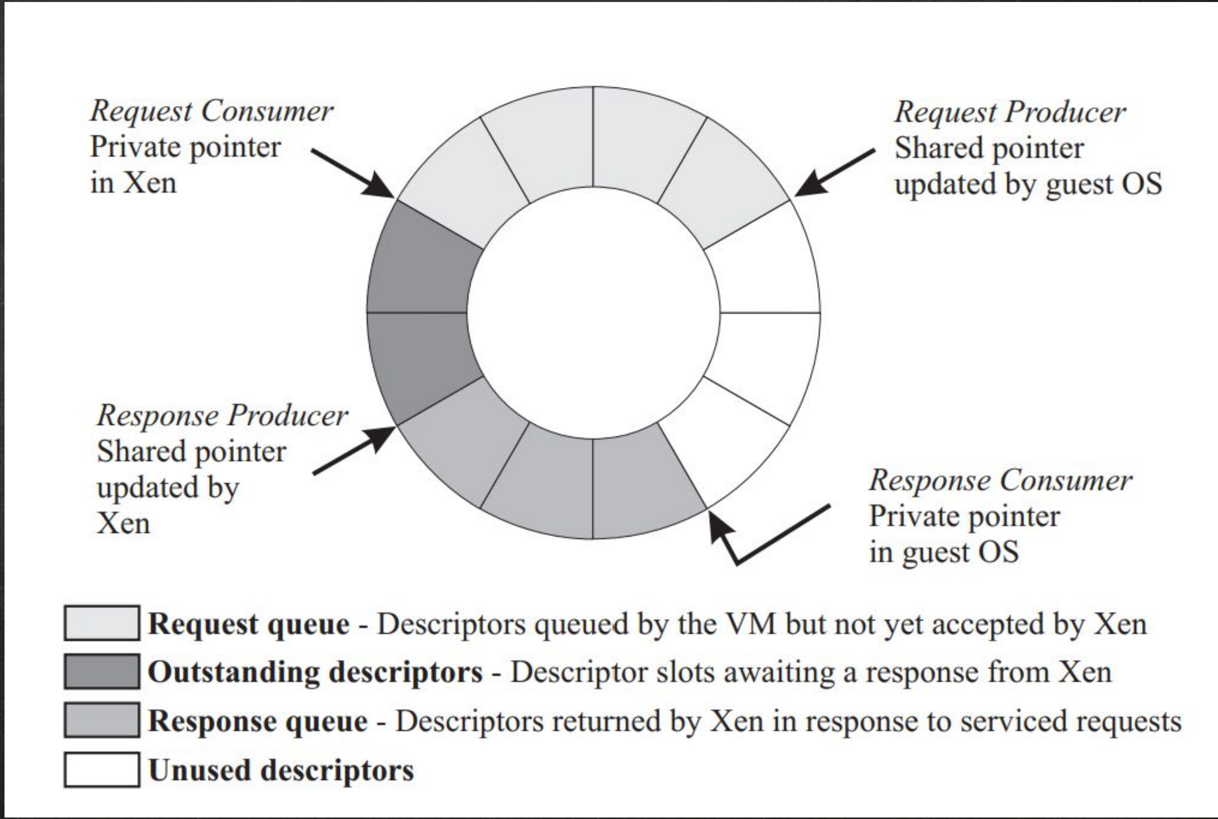
Design - Control Transfer

- Xen <-> domain communication
- Domain to Xen: synchronous call using hypercall
- Xen to domain: asynchronous event mechanism
- Hypercall
  - synchronous software trap into hypervisor
  - perform privileged operation
- Event mechanism
  - replace device interrupts with lightweight notification
  - Represent event using flags
  - Pending event are stored in bitmask, per-domain
  - Ex. got data over network, completed a virtual disk update



Design - Data Transfer

- Communication between guest OS and I/O devices
- Goal: little overhead
- Circular queue of I/O descriptor
- Allocated by domain but accessible within Xen
- Two pairs of producer, consumer pointer
  - Requests: Domains are producer, Xen is consumer
  - Response: Xen is producer, Domains are consumer
- Reorder I/O for scheduling/priority



Discussion Questions

1. Are there performance or concurrency issues with using the shared-memory ring buffer system as the communication system?
2. Is it possible for domains to be blocked on an IO event due to inefficient resource multiplexing?

Virtual Machine Interface

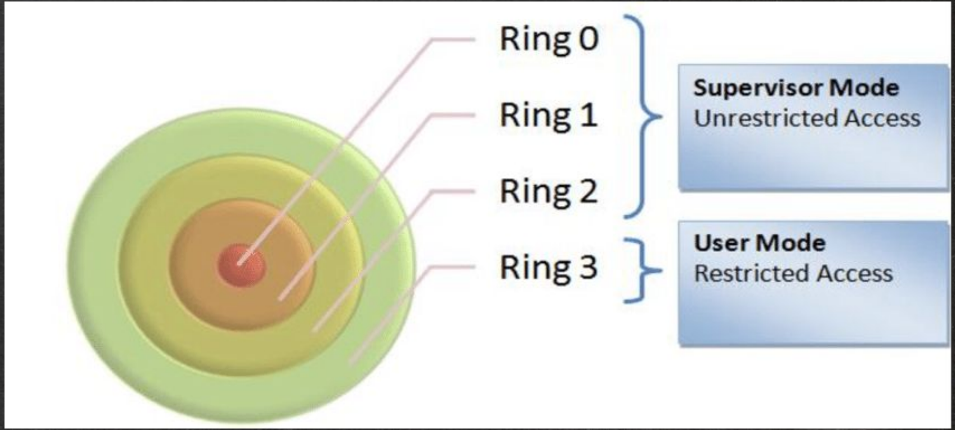
Memory Management	
Segmentation	Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.
Paging	Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated discontiguous machine pages.
CPU	
Protection	Guest OS must run at a lower privilege level than Xen.
Exceptions	Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.
System Calls	Guest OS may install a 'fast' handler for system calls, allowing direct calls from an application into its guest OS and avoiding indirecting through Xen on every call.
Interrupts	Hardware interrupts are replaced with a lightweight event system.
Time	Each guest OS has a timer interface and is aware of both 'real' and 'virtual' time.
Device I/O	
Network, Disk, etc.	Virtual devices are elegant and simple to access. Data is transferred using asynchronous I/O rings. An event mechanism replaces hardware interrupts for notifications.

Virtualization Interface - Memory Management

- Guest OS are responsible for allocating and managing individual hardware page tables
- Xen is at the top of every address space (64MB)
- New page table need to be registered with Xen
- OS relinquish direct write privileges
- Can batch update requests
  - Amortize for the overhead of entering Xen
  - useful for creating new address space
  - Must commit updates before TLB flush
- Updates are validated by Xen, using hypercalls
- Memory allocation specified at creation for each domain
  - Can claim additional pages
  - Can release memory pages

Virtualization Interface - CPU

- Guest OS are modified to run at a lower privileged level
- Rings for privilege levels
  - 0 (most privileged), 3 (least)
  - Modify OS to execute at ring 1
- Privileged instruction need to be validated and executed within Xen
- Exceptions: table describing handler for each type is registered with Xen
- Frequent exceptions: system calls
- Fast exception handler access directly by the processor without indirecting to Xen

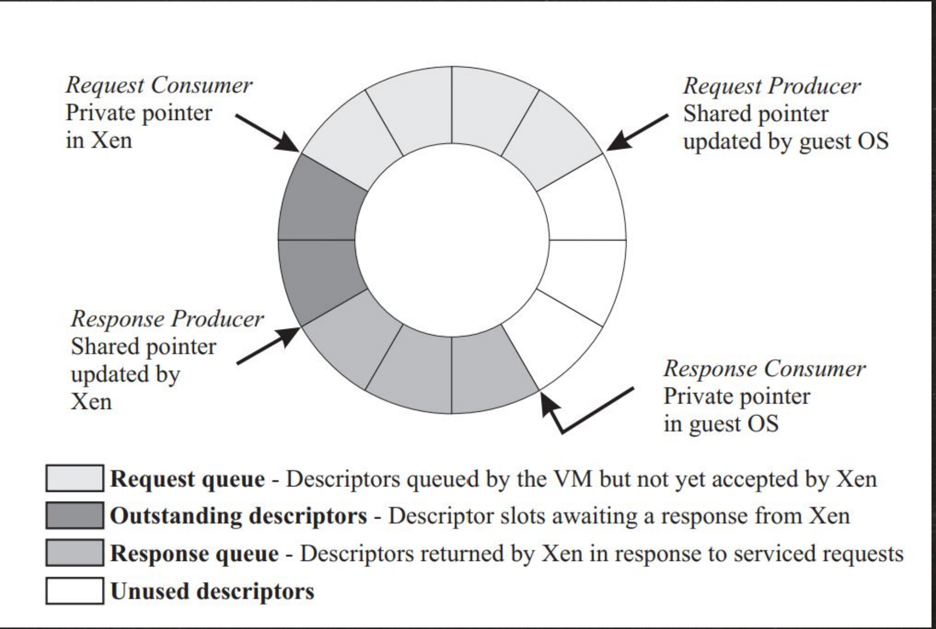


Virtualization Interface - CPU Scheduling

- Borrowed Virtual Time (BVT) scheduling algorithm
- Benefits
  - Work-conserving
  - Low-latency wake-up of a domain when receiving an event
  - Fast dispatch → minimize effect of virtualization
- Drawbacks
  - Violae ‘idea’ fair sharing
  - Favor recently-woken domains
- Note: **low-latency dispatch** refers to the ability of the scheduler to quickly dispatch or execute a thread that requires immediate CPU time, especially for real-time or interactive tasks that are latency-sensitive.
- In BVT, threads can "borrow" virtual time by warping their virtual time to an earlier point, making them appear to have a higher priority

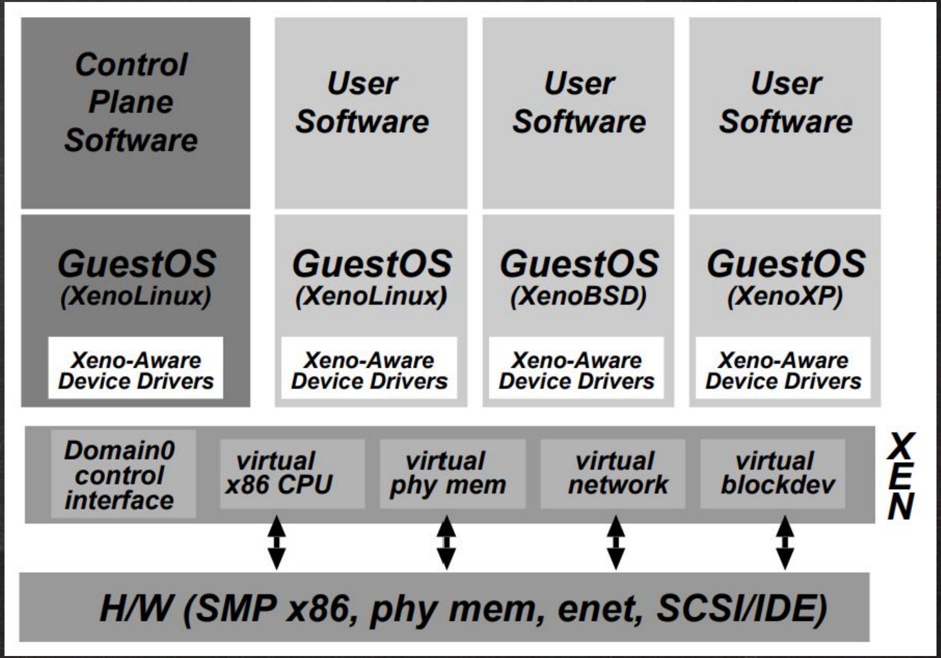
Virtualization Interface - Device IO

- Using shared-memory, asynchronous buffer-descriptor rings
- High-performance communication to pass buffer information
- Xen perform validation check (is address in the domain’s memory reservation?)
- Use lightweight event delivery mechanism
- Xen will call event handler specified by the guest OS
- Using shared-memory, asynchronous buffer-descriptor rings



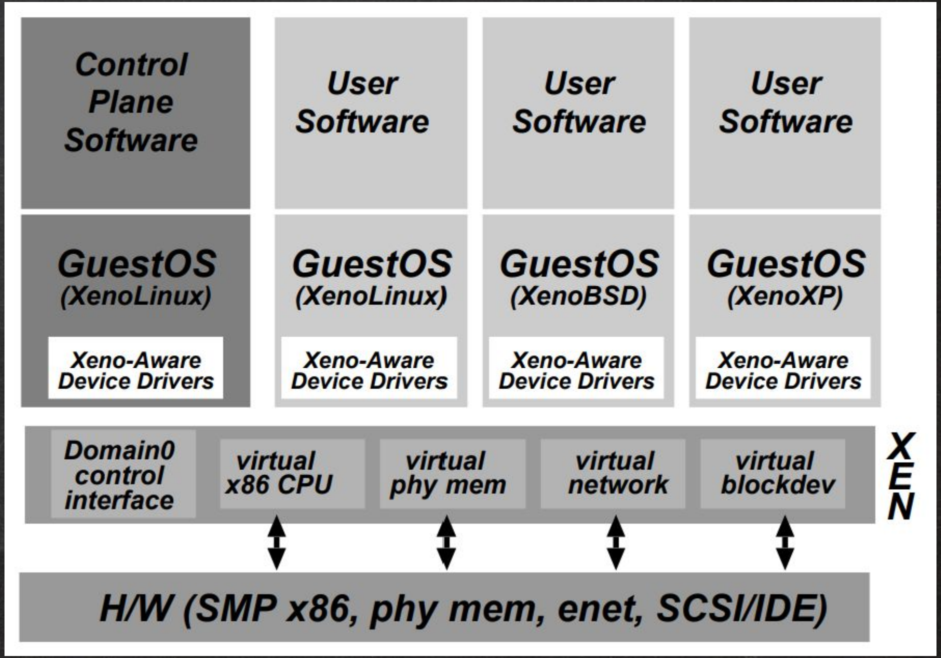
Virtualization Interface - Split Drivers

- Domain0 is responsible for hosting application-level management software
  - Create/terminate other domains, control their scheduling parameters, physical memory allocations, and accesses for physical disks and network devices
- Split-Driver model: technique for creating efficient virtual hardware
  - One device driver runs inside guest to interact directly with applications
  - They communicates with another corresponding device driver inside Domain0 that manage hardware
  - Pair of drivers function together (Ex. block and network device drivers)



Virtualization Interface - Network

- Virtual firewall-router (VFR): each domain has one or more network interfaces (VIFs)
- I/O ring buffers with associated rules
  - <pattern> <action>
  - if pattern matches then action is applied
  - Domain0 insert and remove rules
  - Rules can prevent IP src spoofing and ensure correct demultiplexing
- Transmit
  - guestOS: enqueue buffer descriptor on transmit ring
  - Xen: copy header and execute rule
- Receive
  - Xen: determines destination, exchange buffer to page frame
  - Page frame must be pinned
  - Exchange unused page frame for each packet received



Virtualization Interface - Disks

- Domain0 has access to physical disks
- Other domains access through virtual block devices (VBD)
- VBD has extents and ownership info, accessed using I/O ring
- Xen maintains translation table for each VBD
- When receive disk request, Xen inspect VBD identifier and offset
- Zero-copy data transfer using DMA to transfer between disk content and pinned memory pages in the requesting domain
- Xen batches requests

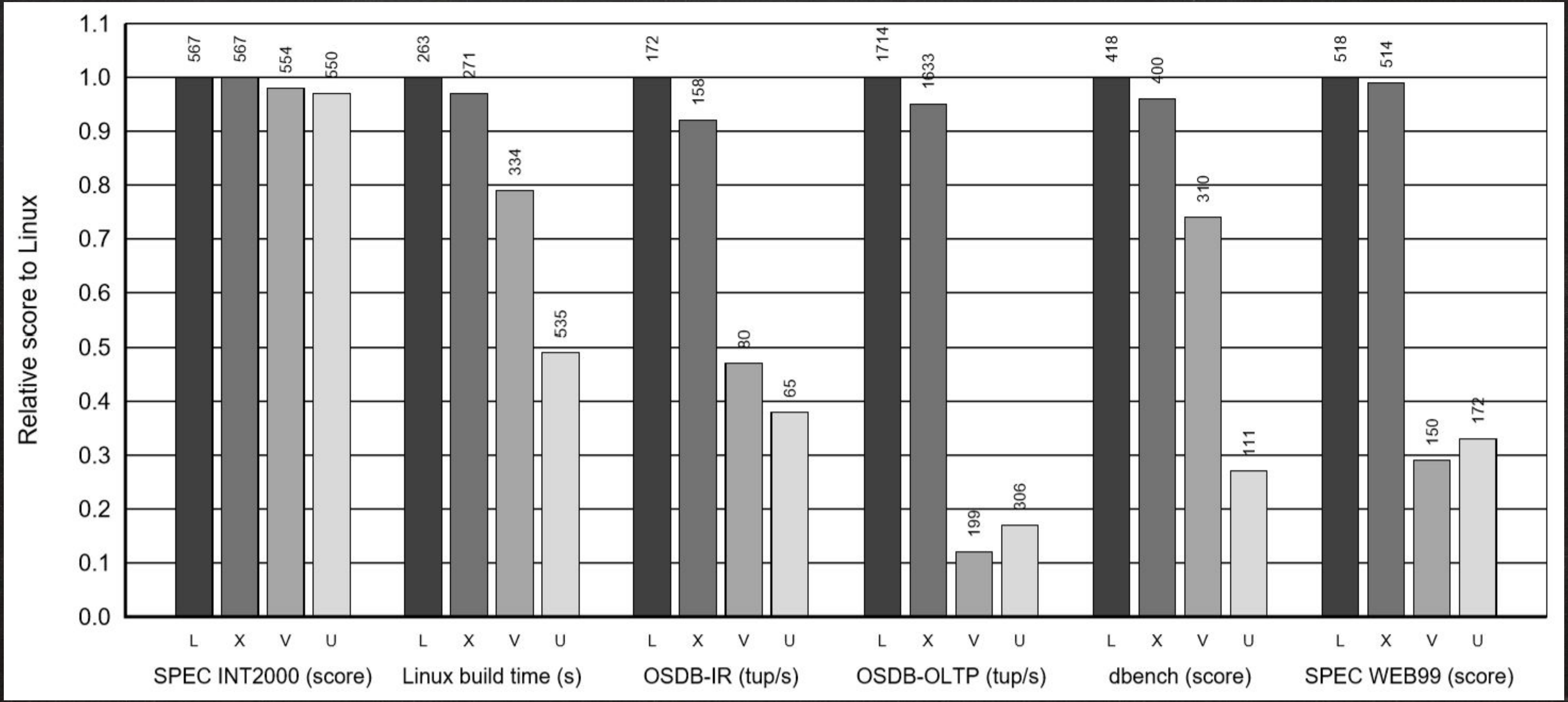
Discussion Questions

1. For OS with only 2 levels, what is the approach for putting the hypervisor at a priority higher than the OS?
2. Is there security risks for the way that Xen virtualizes the CPU, memory, disk, or network?
3. Is it a problem to exchange a unused page for every packet received?

Performance

Evaluation

- Xen use XenoLinux (based on Linux 2.4.21) as guest OS
  - developing for XP and NetBSD
- VMware Workstation 3.2
- User-mode Linux (UML)
- Native linux, executing multiple applications on the OS (vs. applications in separate VMs for Xen)
- Dell 2650 dual-processor
  - 2.4GHz Xeon server
  - 2GB Ram



Relative score to Linux
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
567
567
554
550
263
271
334
535
172
158
80
65
1714
1633
199
306
418
400
310
111
518
514
150
172
L X V U
SPEC INT2000 (score)
Linux build time (s)
OSDB-IR (tup/s)
OSDB-OLTP (tup/s)
dbench (score)
SPEC WEB99 (score)

Config	null call	null I/O	stat	open close	slct TCP	sig inst	sig hndl	fork proc	exec proc	sh proc
L-SMP	0.53	0.81	2.10	3.51	23.2	0.83	2.94	143	601	4k2
L-UP	0.45	0.50	1.28	1.92	5.70	0.68	2.49	110	530	4k0
Xen	0.46	0.50	1.22	1.88	5.69	0.69	1.75	**198**	**768**	**4k8**
VMW	0.73	0.83	1.88	2.99	11.1	1.02	4.63	874	2k3	10k
UML	24.7	25.1	36.1	62.8	39.9	26.0	46.0	21k	33k	58k

Table 3: `lmbench`: Processes - times in μs

Config	2p 0K	2p 16K	2p 64K	8p 16K	8p 64K	16p 16K	16p 64K
L-SMP	1.69	1.88	2.03	2.36	26.8	4.79	38.4
L-UP	0.77	0.91	1.06	1.03	24.3	3.61	37.6
Xen	**1.97**	**2.22**	**2.67**	**3.07**	**28.7**	**7.08**	39.4
VMW	18.1	17.6	21.3	22.4	51.6	41.7	72.2
UML	15.5	14.6	14.4	16.3	36.8	23.6	52.0

Table 4: `lmbench`: Context switching times in μs

Config	0K File create	0K File delete	10K File create	10K File delete	Mmap lat	Prot fault	Page fault
L-SMP	44.9	24.2	123	45.2	99.0	1.33	1.88
L-UP	32.1	6.08	66.0	12.5	68.0	1.06	1.42
Xen	32.5	5.86	68.2	13.6	**139**	1.40	**2.73**
VMW	35.3	9.3	85.6	21.4	620	7.53	12.4
UML	130	65.7	250	113	1k4	21.8	26.3

Table 5: `lmbench`: File & VM system latencies in μs

	TCP MTU 1500		TCP MTU 500	
	TX	RX	TX	RX
Linux	897	897	602	544
Xen	897 (-0%)	897 (-0%)	516 (-14%)	467 (-14%)
VMW	291 (-68%)	615 (-31%)	101 (-83%)	137 (-75%)
UML	165 (-82%)	203 (-77%)	61.1(-90%)	91.4(-83%)

Table 6: `ttcp`: Bandwidth in Mb/s

Discussion

1. What are the trends in performance evaluation, especially for cases where Xen has worse performances than native Linux?
2. What are the most significant performance optimizations that reduced the overhead for Xen?

Summary

- Xen: paravirtualization, strong performance isolation, OS-granularity VMM that does not require the applications to change their ABIs and supports multi-application OS
- Historically, VMM has high performance overhead
- Xen shows performance comparable with native Linux and significantly better than VMWare and User-Space Linux

Xen and Microkernels

- "Xen in particular, are in fact a specific point in the microkernels design space; that VMMs are microkernels done right" (Hand et al.)

Microkernels	Xen
Liability inversion: applications depend on user-level components (external pagers)	Avoid liability inversion: isolation, partitions memory and allows limited sharing
Depends on IPC performance	Less IPC between VM; Control (synchronous IPC) and data path (async rings) split
Changing ABIs	Support out-of-the-box code
Academic research	Developed in industry