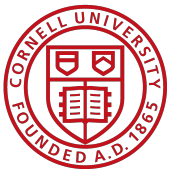
Xen and the art of virtualization

Paper presentation for CS 6410

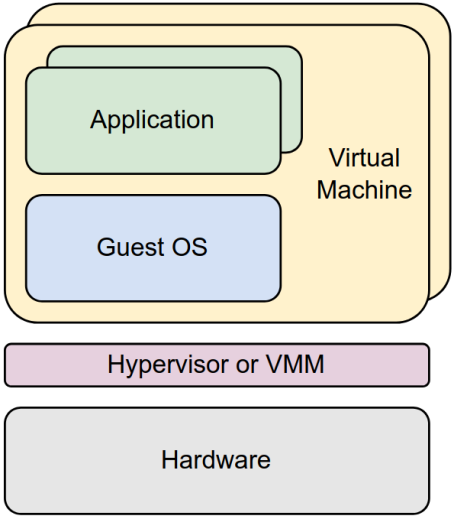
Irene Simo Munoz

Cornell HPC

September 23rd 2025

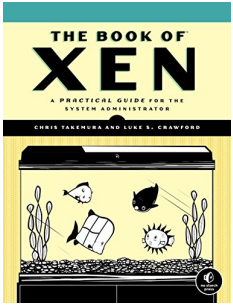


Virtualization



Introduction

- Published in 2003, but VM had been around for a while already. IBM main frames had VM back in the 70s on System/370
- New approach on virtualization: how to do it on the x86 architecture



Xen was the first practical open-source **high-performance hypervisor**

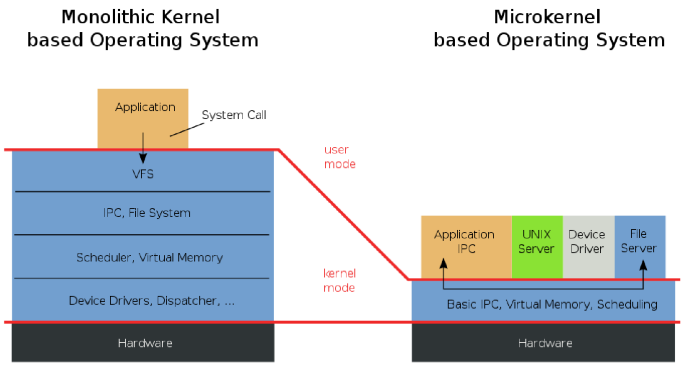
About the authors

Graduate student + professor collaboration out of University of Cambridge.

- **Ian Pratt**: Faculty at University of Cambridge, later founded XenSource
- **Keir Fraser**: PhD student at Cambridge, lead developer of Xen
- **Paul Barham**: Google MI infra, TensroFlow, Microsoft Research
- **Boris Dragovic, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Andrew Warfield**: Part of the original Cambridge Systems Research Group

From microkernels to virtualization

- **Monolithic kernels**: all services run in kernel space
- **Microkernels**: Kernel runs the bare essentials (IPC, scheduling, etc.), other services run in user space as separate processes



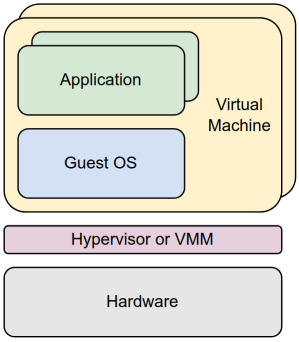
The x86 architecture

"Un-cooperative" design for full virtualization

- Insufficient permission on supervisor instructions on VMM **fail silently**
- Complicated privilege model
  - 4 privilege levels (rings)
  - Ring 0: kernel mode
  - Ring 3: user mode
  - No clean way for the hypervisor to sit "below" the guest OS while maintaining the illusion that it's running in ring 0.
  - Complex workarounds for full virtualization (binary translation, device emulation)

Full virtualization

- No need to modify the guest OS
- **Binary translation**: Hypervisor traps and emulates privileged instructions
- Each VM gets a fully virtualized hardware environment
- Needs hardware support or heavy **device emulation**



Architectures like x86 are not natively virtualizable and require costly techniques, causing huge **performance overhead**

A tradeoff: from full virtualization to paravirtualization

Paravirtualization

Trade off small changes to the guest OS for big improvements in performance and VMM simplicity. Guest OS are *aware* that they run in a virtualized enviroment.

- DomUs will explicitly call the hypervisor using a safe and efficient interface (like system calls, but from the guest OS to the hypervisor)
- Hypervisor is simpler and faster

Xen overview

- Paravirtualization for speed
  - Keep hypervisor as small as possible Split drivers
  - Modify guest OS to be aware of virtualization issues of x86 architecture
  - Ensure strong isolation between Guest OSes
- No changes to Application Binary Interface (ABIs)
  - Propetiary software, modular applications can be executed in all guest OSes
- Domain0

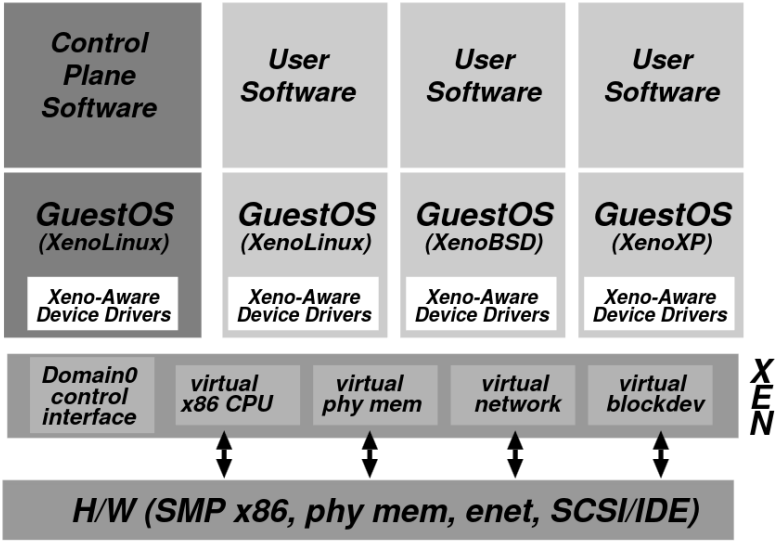
Discussion

1. What are the differences between virtualization and exokernels?
2. Why was virtualization much more successful than exokernels?

Xen Architecture roadmap

- Architecture
- Domain0
- Control transfer
- Split drivers
- Data transfer

Diagram of architecture



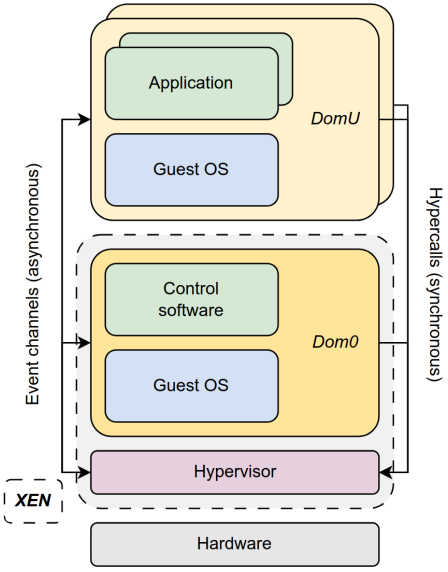
Domain0 kernel

- Automatically started at boot time
- Special management privileges: only domain with direct access to physical devices and the control interface to the hypervisor
- Responsibilities:
  - Creating, destroying, and managing DomUs
  - Device management (network, disk, etc.)
  - I/O request mediation for DomUs
  - Resource allocation for DomUs

Control transfer: Hypercalls and events

Xen provides two mechanisms:

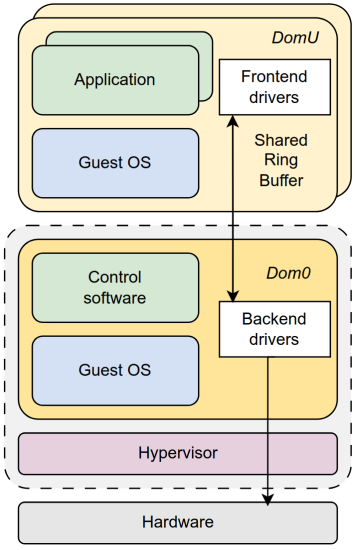
- **Hypercall (synch)**: DomU /Dom0 requests a privileged service, allocating resources or managing memory
- **Event channel (asynch)**: Signals, interrupts, notifications between domans and hypervisor.



Split drivers or paravirtualized I/O

Multiple DomUs need access to hardware

- Xen doesn't include drivers in the hypervisor itself, they are run in domains
- Dom0 is the only domain with direct access to hardware
- DomU domains must communicate with Dom0 to perform I/O

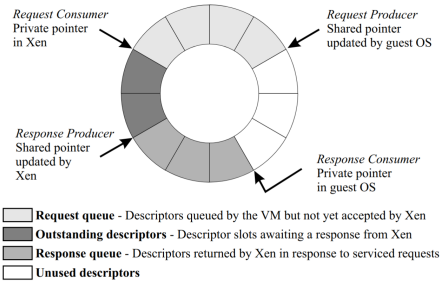


Data transfer: I/O rings

I/O ring

Shared-memory circular queue of descriptors allocated by a domain but accessible from within Xen

- Descriptors do not hold the I/O data themselves; instead, they point to buffers allocated by the Guest OS.
- Requests are processed asynchronously



Discussion

1. What happens if Dom0's security is compromised?
2. What makes Xen's architecture not suitable for the cloud (as presented in this paper)?

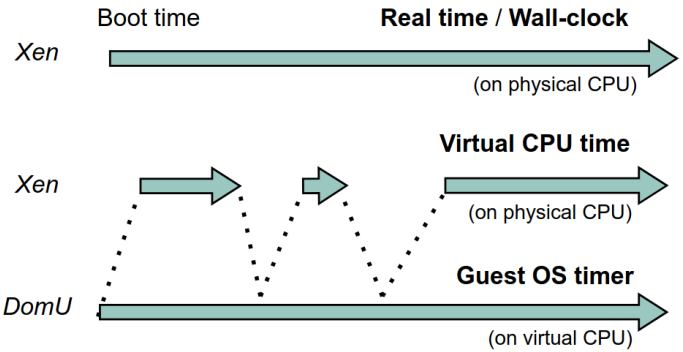
CPU virtualization; scheduling

- Protection
  - Xen in ring 0, Guest kernel in ring 1
  - DomUs cannot execute privileged instructions directly.
  - **Privileged instructions are replaced with hypercalls**
- Xen uses Borrowed Virtual Time (BVT) to schedule virtual CPUs
  - Threads can "borrow" virtual time by warping their virtual time to an earlier point, making them appear to have a higher priority
- Xen handles page fault exceptions on behalf of DomUs

Timers and time virtualization

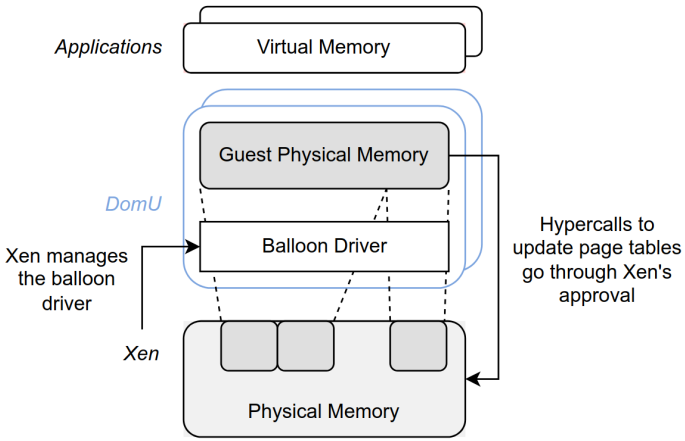
Running in a virtual machine, OS are not used to being de-scheduled, stopped. Not enough CPUs for all the VMs running at once!

- Network: When do I need to resend a packet? Did I receive something while I was off?



Virtualizing physical memory

- Reservations are specified at time of creation
- Balloon driver (dynamic adjustment)
- Shared translation array (between guest virtual memory and actual machine memory)

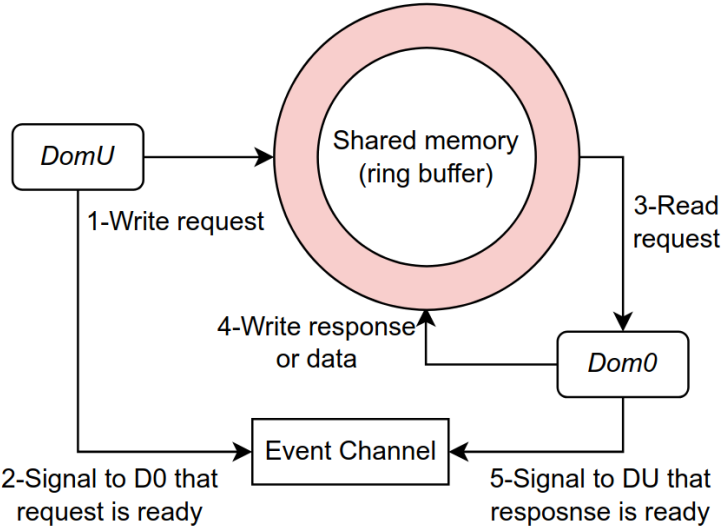


Memory access virtualization

- x86 uses hardware page tables
- Xen exists in a 64MB section at the top of every address space
- DomUs allocate and manage their own page tables
  - DomU requires new page table $\rightarrow$ allocates and initializes from its own memory reservation and registers it to Xen
  - Xen is only involved in updates via hypercall
  - Updates must be validated before being applied
  - Can queue updates and apply in batch

Network and disk virtualization

- DomU has $\geq$ network interface
- Each one contains two I/O ring buffers with associated rules (Dom0)
- Each direction (transmit, receive) has a list of associated rules of the form (pattern,action) – if the pattern matches then the associated action applied
- DomU access through virtual block device (VBD)
- For each VBD a translation table is maintained at the hypervisor (entries controlled by Dom0)
- Domain0 has access to physical disks
- VBD info accessed using I/O ring


DomU
Shared memory
(ring buffer)
1-Write request
3-Read
request
4-Write response
or data
Dom0
Event Channel
2-Signal to D0 that
request is ready
5-Signal to DU that
resposnse is ready

Discussion

1. For OS with only 2 levels, what is the approach for putting the hypervisor at a priority higher than the OS?
2. Note on Balloon Dirver

CPU/memory microbenchmarks.

Xen performs almost as good as native Linux for all microbenchmarks!

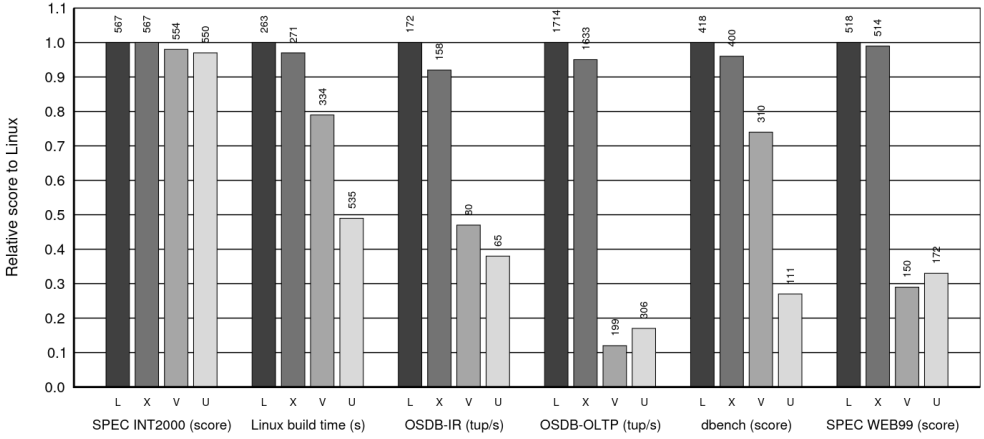


Figure 3: Relative performance of native Linux (L), XenoLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).

Other nice results

- Scalability (target of 100 domains)
  - Domains able to reduce its memory footprint up to 90%
  - Context switching between large numbers of domains, Xen only loses 7.5% of total throughput relative to Linux

Discussion

1. Are VM microkernels done right?
2. Why do VMs seem to be more successful than microkernels?

Are VM microkernels done right?

1. *Steven Hand, Andrew Wareld, Keir Fraser* (2005)
   - Microkernels rely heavily on user-level components. Xen, keeps the hypervisor tiny and doesn't push so much into user space.
   - Performance in microkernels is dominated by IPC latency, but Xen focuses on VM isolation and paravirtualized performance, so IPC isn't central.
   - Supporting legacy apps is painful in a pure microkernel world, but Xen treats entire OSes as components, so it inherits all their compatibility.
2. *Gernot Heiser, Volkmar Uhlig, Joshua LeVasseur* (2006)
   - Xen also relies on Dom0!
   - Xen also ends up doing a lot of IPC-style communication (via event channels and shared memory)
   - L4Linux project showed you can run full OSes on microkernels with acceptable performance

Legacy

- Industry adoption: Basis for commercial XenSource, then bought by Citrix ($500 million)
- Backbone of Amazon's AWS EC2 in early days, used Xen hypervisor until late 2010s → Cloud ran on Xen!
- Set stage for hardware virtualization extensions (Intel VT-x, AMD-V): paravirtualization influenced Intel/AMD to add new hardware instructions to make virtualization easier

References

1. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A. (2003). Xen and the art of virtualization. ACM SIGOPS Operating Systems Review, 37(5), 164–177. https://doi.org/10.1145/1165389.945462
2. Bitwise Botcast. (2025, April 3). Xen and the Art of Virtualization [Video]. YouTube https://www.youtube.com/watch?v=UG7pzbBnxe8
3. Shen, Zhiming (n.d.). Virtualization technology. Fall 2016. https://www.cs.cornell.edu/courses/cs6410/2016fa/slides/07-virtualization.pdf
4. Ouyang, Chuhan. Xen and the Art of Virtualization. Fall 2024. https://www.cs.cornell.edu/courses/cs6410/2024fa/schedule/slides/09-virtualization.pdf
5. Weatherspoon, Hakim "VIRTUALIZATION: IBM VM/370 AND XEN" Fall 2019. https://www.cs.cornell.edu/courses/cs6410/2019fa/slides/09-virtualization.pdf