Xen and the Art of Virtualization

UCL Computer Science

CS M038 / GZ06

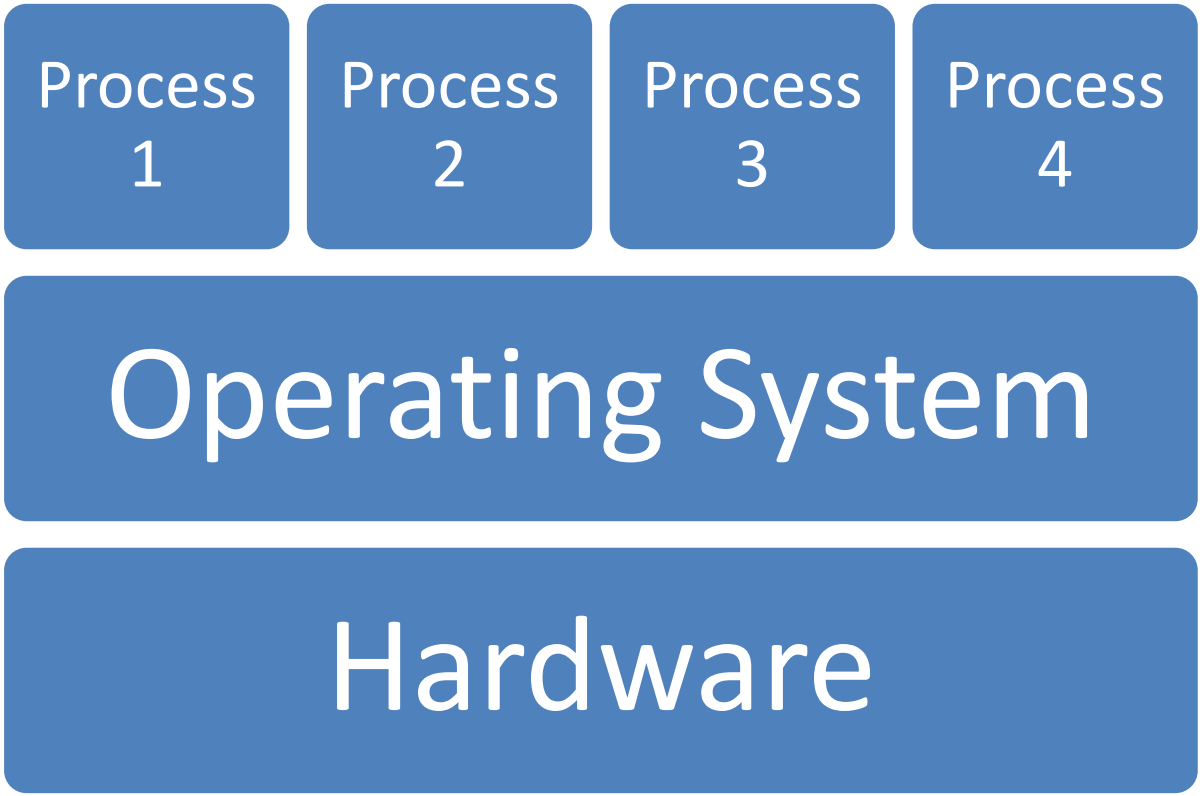
Nikola Gvozdiev
Georgian Mihaila

Outline

Xen and the Art of Virtualization – Ian Pratt *et al.*

I. **The Art of Virtualization**
II. Xen, goals and design
III. Xen evaluation
IV. The future looks bright

Typical system components



Basic idea

Linux

Mac OS X

Windows XP

Virtualization layer

Hardware

Some definitions

- **Host/guest**: machine or software
- **Domain**: running VM + guest OS executes
- **Hypervisor** or Virtual Machine Monitor (**VMM**): software or firmware that creates a virtual machine on the host hardware

Why virtualize?

- More resources, subdivide for better use

- 1+ app(s)/OS/server:

servers ↗ sys admin complexity & time ↗

- Scarce resources & unwilling cooperation:

resource containers & accountability

Some applications

- Resource optimization
- Infrastructure as a Service (IaaS)
- Application mobility/migration
- Containers/virus/worm (sandboxing)

Types of hardware virtualization

(software, memory, storage, data, network)

- **Full virtualization** (VMware)
  - run OS/software unmodified
- **Partial virtualization**
  - software may need modification to run
- **Paravirtualization** (Xen)
  - software unmodified runs in modified OS as separate system

Outline

Xen and the Art of Virtualization – Ian Pratt *et al.*

I. The Art of Virtualization
II. Xen, goals and design
III. Xen evaluation
IV. The future looks bright

Virtualization, at what cost?

- Specialized hardware $ ↗
- No commodity OS ♥ ↘
- 100% binary compatibility, speed ↘
- Speed ↗ => security ☠ ↘ or functionality ↘

- Resource isolation/performance guarantees?
- So far, best effort provisioning => risk DoS

Xen goals

- x86 architecture => $+=0
- Commodity OS (Linux, BSD, XP) => ♥++
- Share resources safely & in a managed fashion => ☠-- ♥++
- No performance/functionality sacrifice => ♥++
- Up to 100 VM instances on the same server => ?++

- (Free -GNU/GPL- => $--)

The Xen paper

Focus on the VMM design

- How to multiplex physical resources at the granularity of an entire OS and provide performance isolation between guest OSes?

Prototype

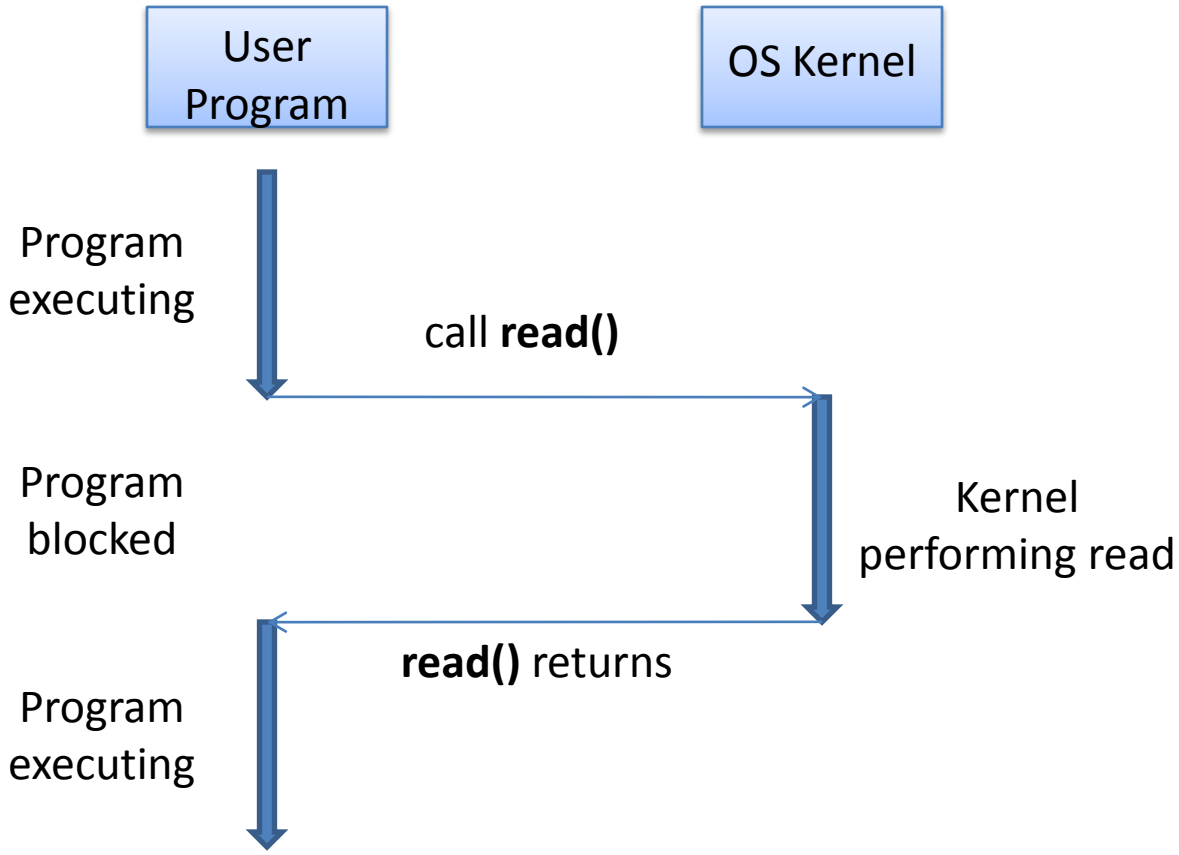
- Full Linux, partial Win XP, ongoing NetBSD port

The Virtual Machine Interface (VMI)

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.

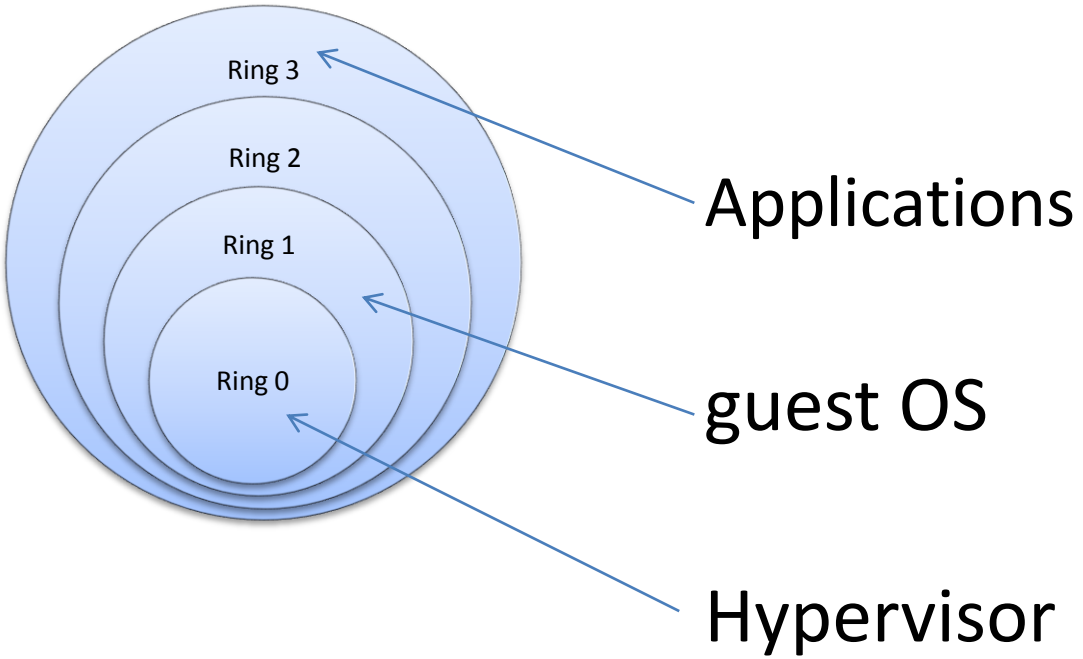
Table 1: The paravirtualized x86 interface.

Typical System call



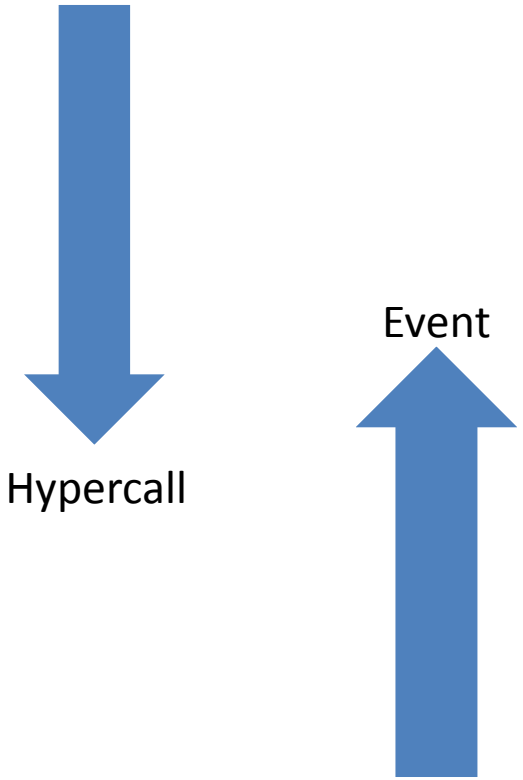
Virtualizing CPU

~~*OS is most privileged*~~ hypervisor



x86

Control transfer



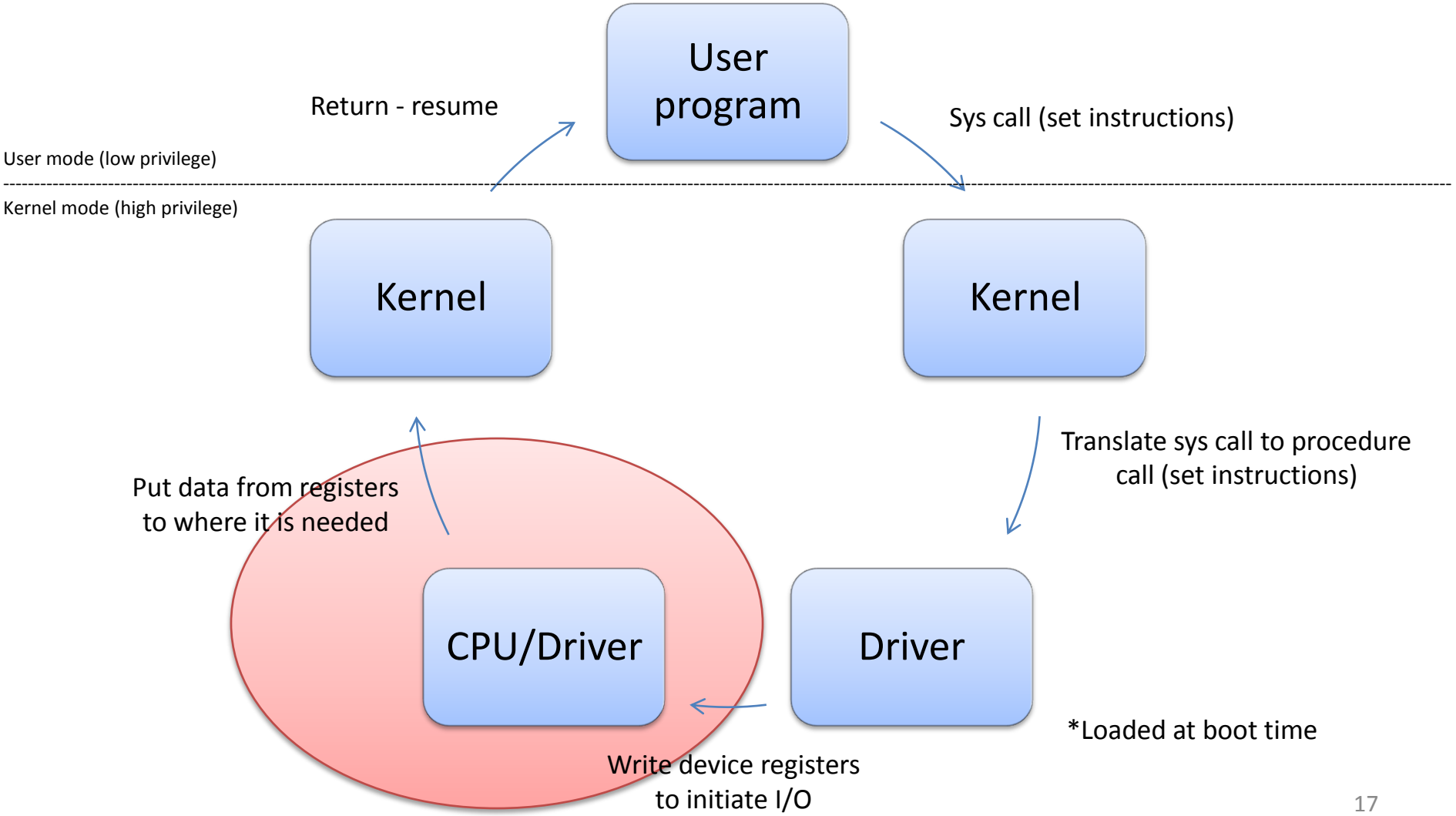
Domain (guest OS)

- Hypercall interface (synchronous software trap)

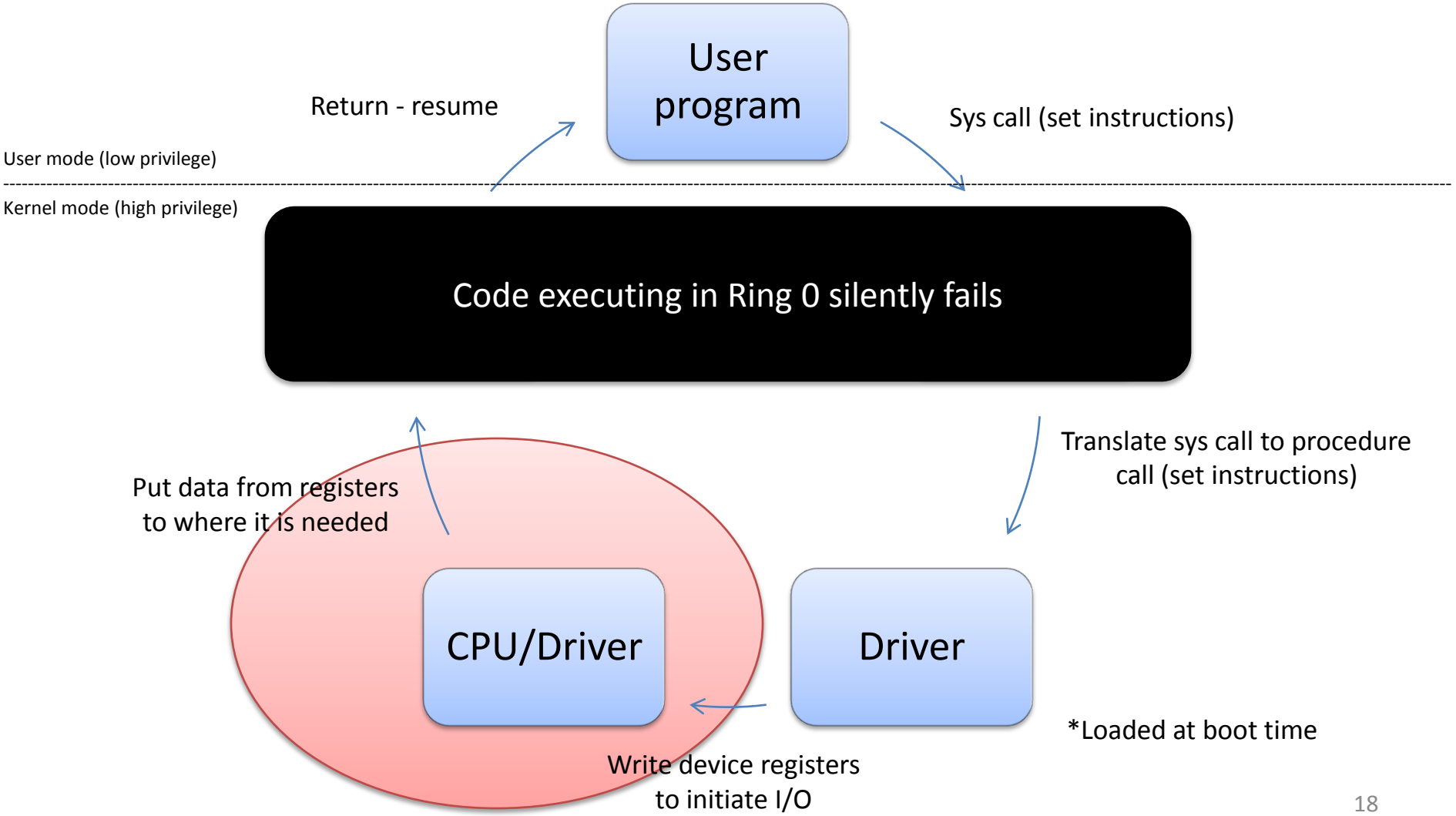
Xen (VMM)

- Event mechanism (asynchronous)

High level view



High level view



Exceptions

- Typical exceptions
  - **System calls**: register ‘fast’ exception handler (in Exception Vector Table) which is accessed directly by the processor, without indirecting via ring 0
  - **Page faults**: code executing in ring 0 only can read the faulting address from register CR2, therefore faults are delivered via Xen

Virtualizing CPU cont'd

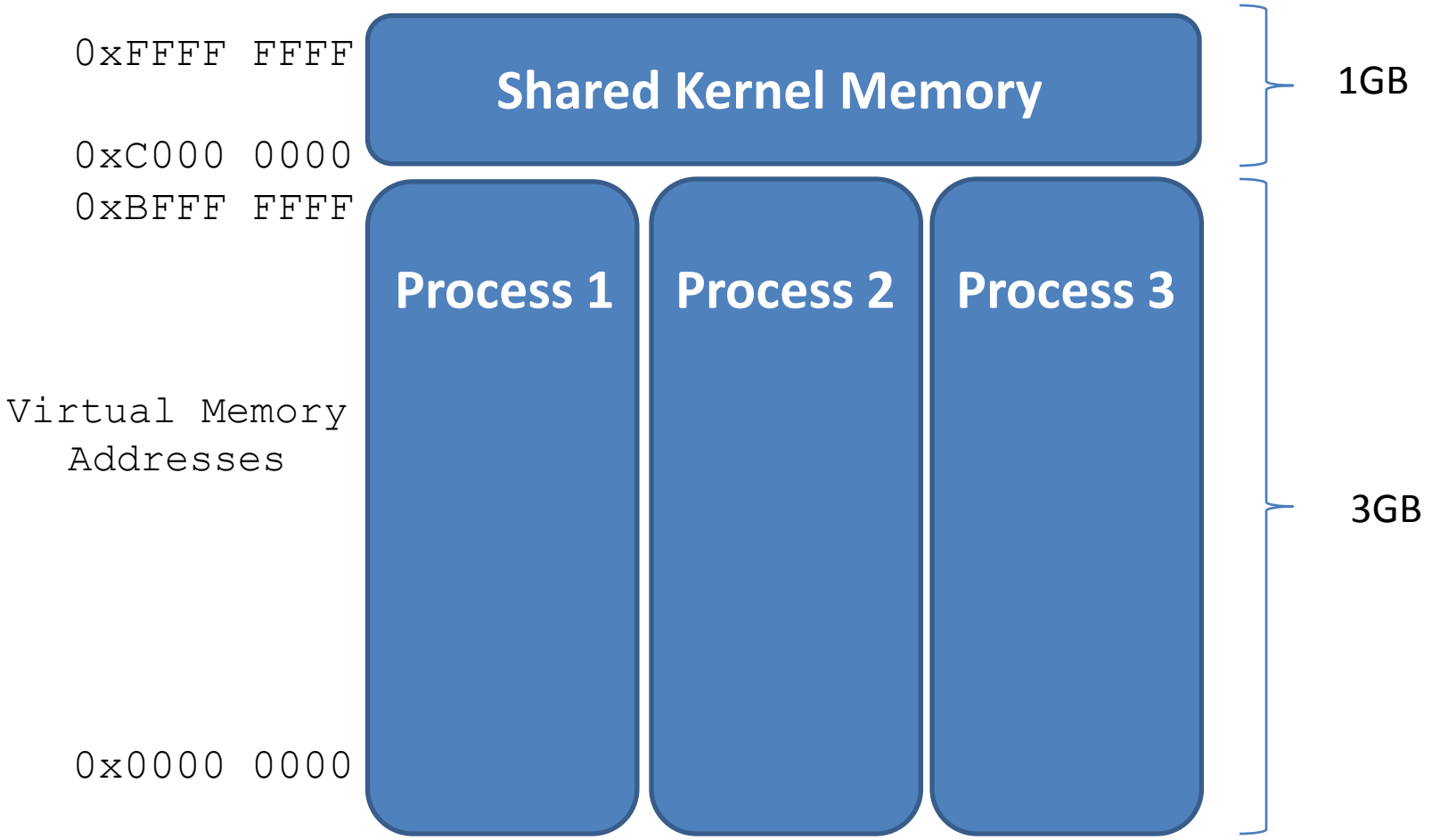
- CPU scheduling
  - Borrowed Virtual Time (BVT) scheduling algorithm

Work-conserving

Low-latency wake-up (dispatch)

Timers : real, virtual, wall-clock

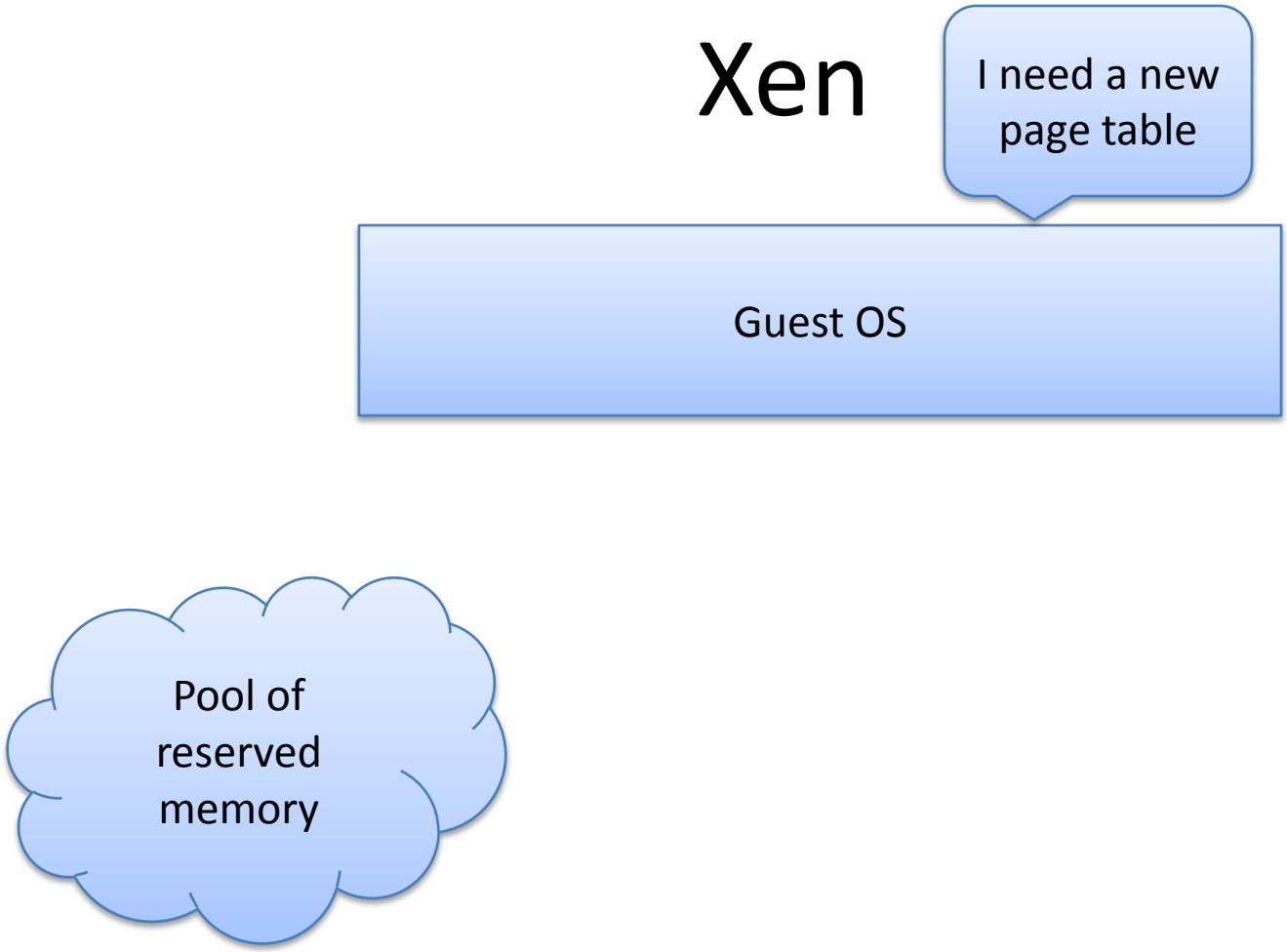
Memory mapping (relocation) cont'd



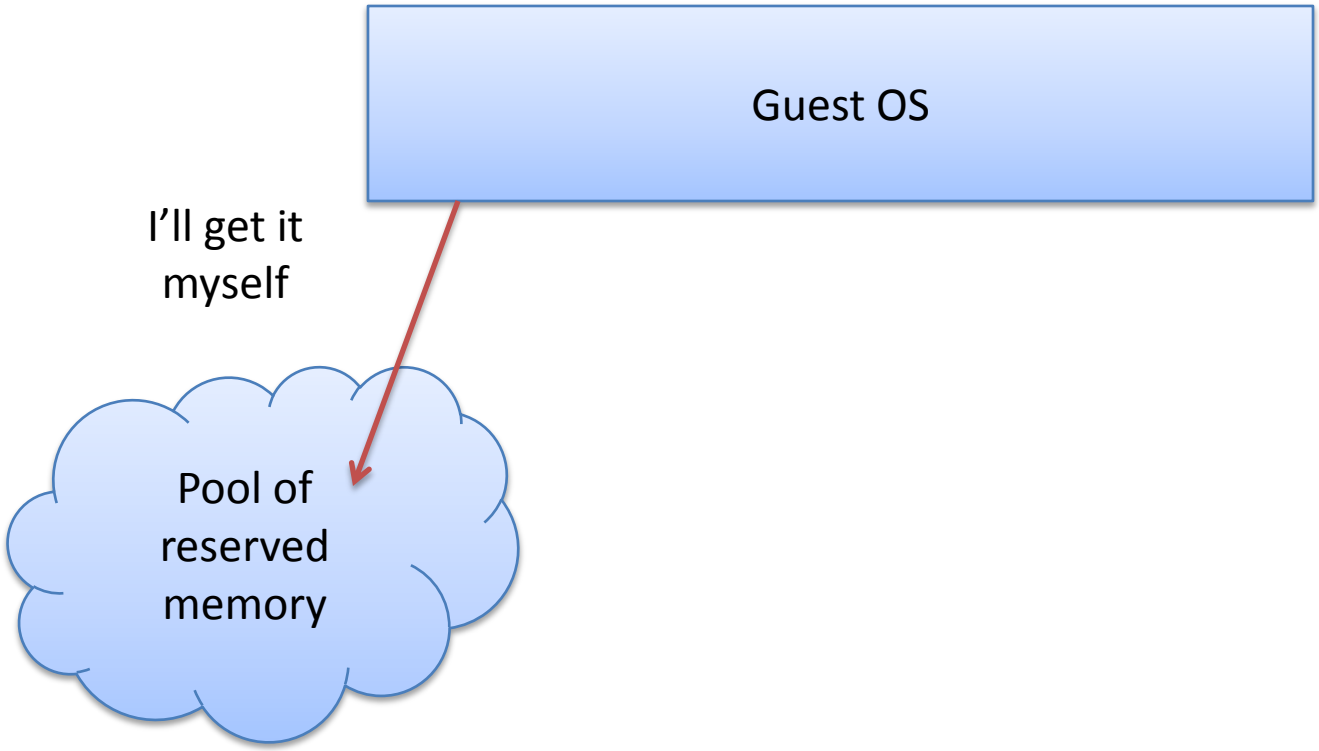
Virtualizing memory management (cont'd)

- x86 architecture
  - **No** software managed TLB (walk the page table structure in hardware to recover misses) and no tags (flush TLB for address space switch)

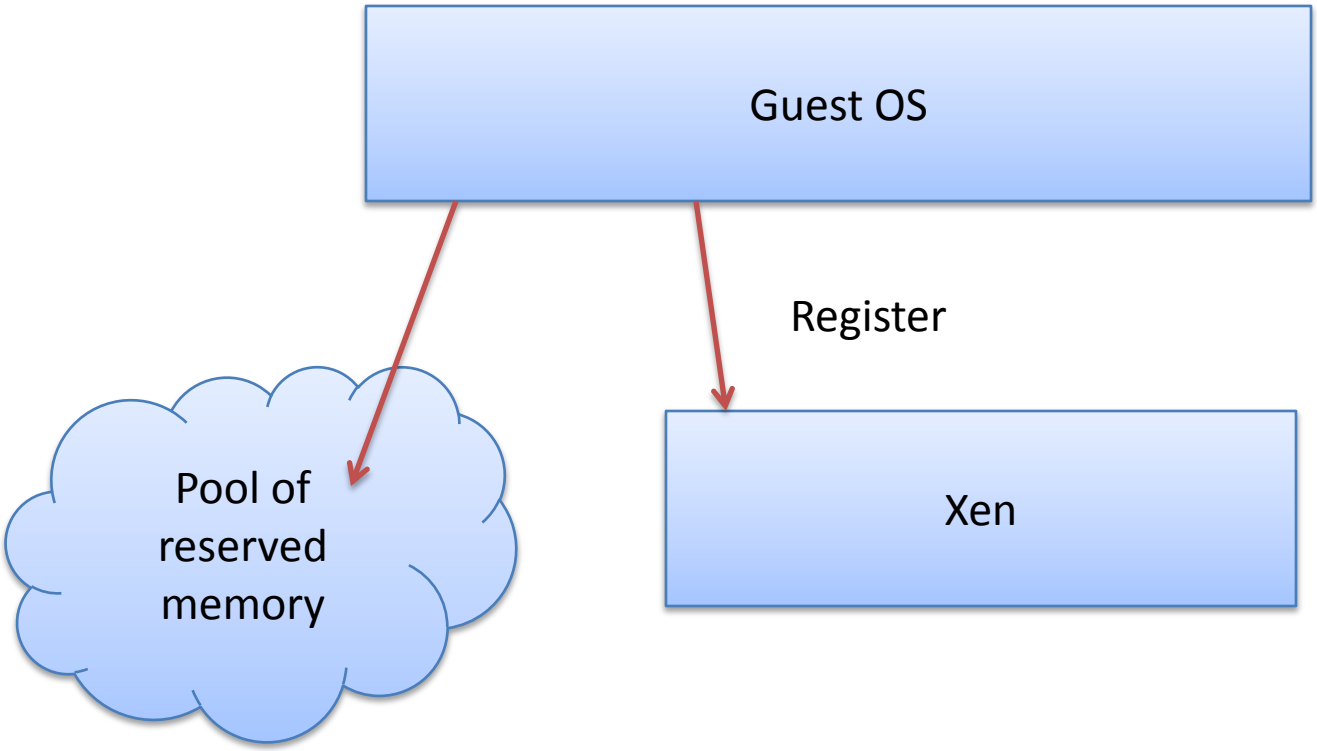
- Guest OSes are responsible for allocating and managing the hardware page tables, minimizing Xen involvement
- Xen exists in a 64MB section at the top of every address space, avoiding a TLB flush when entering and leaving the hypervisor


Xen
I need a new page table
Guest OS
Pool of reserved memory

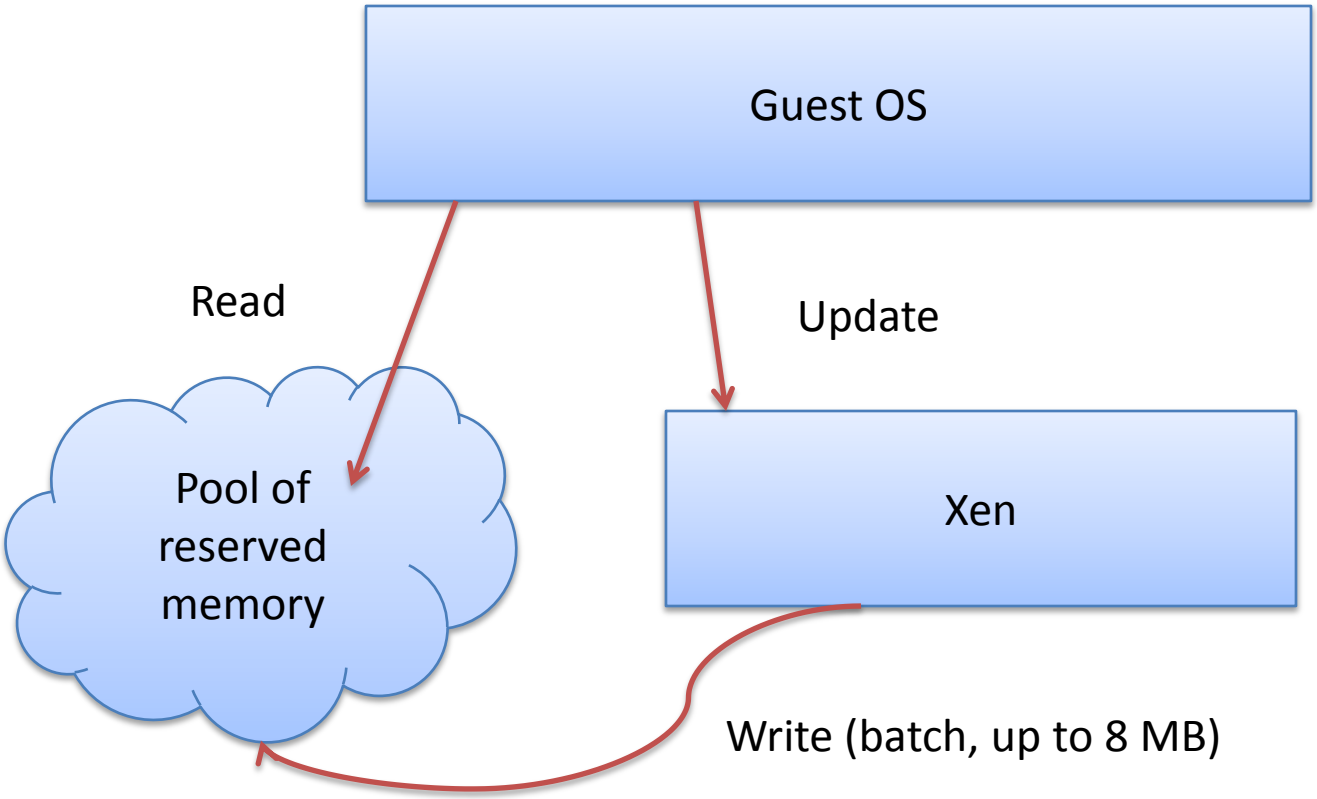
Xen



Xen



Xen


Guest OS
Read
Update
Pool of reserved memory
Xen
Write (batch, up to 8 MB)

Xen

- Reduce memory footprint
- Avoid copies

Virtualizing Device I/O

(disk, network interface)

- Xen exposes device abstractions

- I/O data is transferred to and from each domain via Xen, using shared memory, asynchronous buffer descriptor rings

Data Transfer : I/O Rings

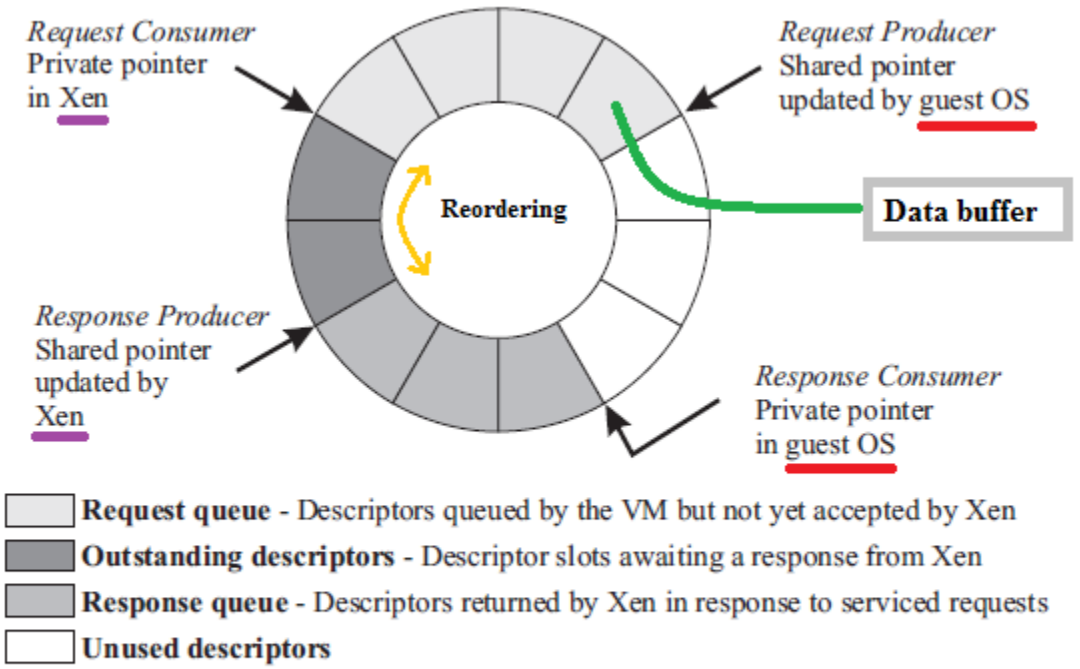


Figure 2: The structure of asynchronous I/O rings, which are used for data transfer between Xen and guest OSes.

Xen design

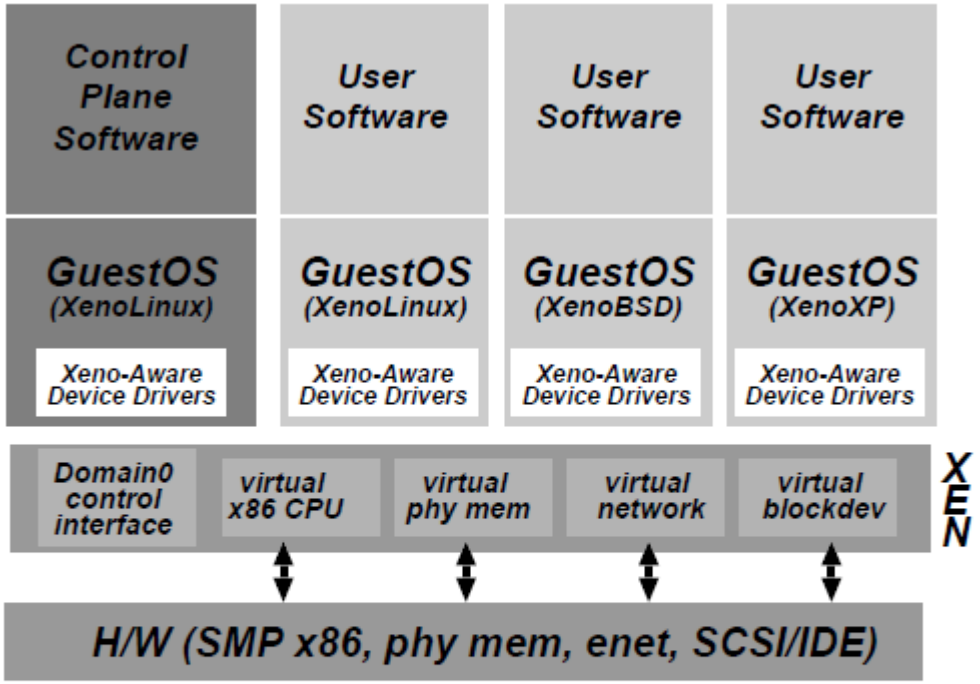


Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including *Domain0* running control software in a XenoLinux environment.

Outline

Xen and the Art of Virtualization – Ian Pratt *et al.*

I. The Art of Virtualization
II. Xen, goals and design
III. Xen evaluation
IV. The future looks bright

Cost of porting an OS

OS subsection	# lines	
	Linux	XP
Architecture-independent	78	1299
Virtual network driver	484	–
Virtual block-device driver	1070	–
Xen-specific (non-driver)	1363	3321
Total	**2995**	**4620**
(Portion of total x86 code base	**1.36%**	**0.04%)**

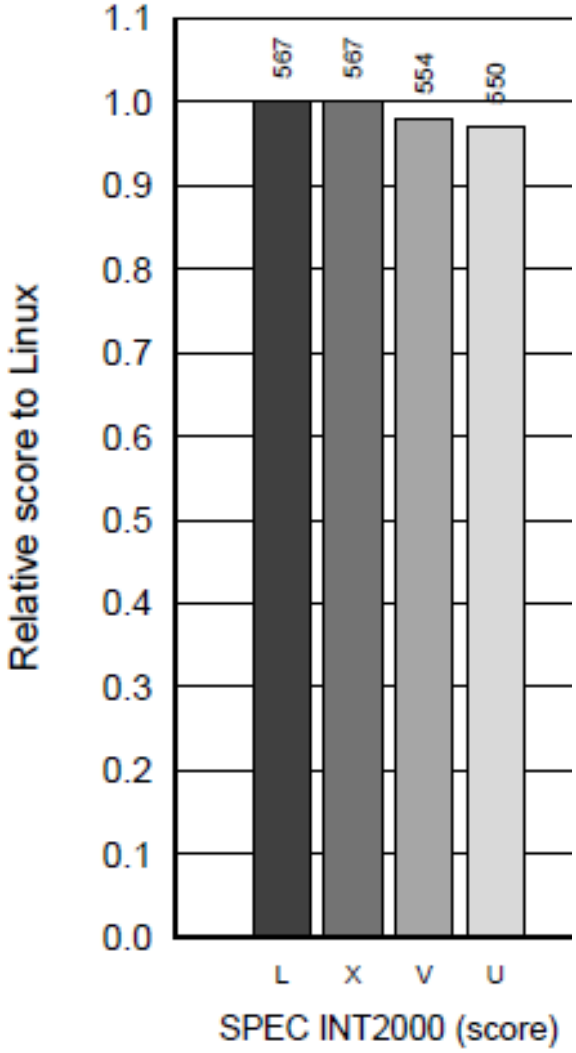
Table 2: The simplicity of porting commodity OSes to Xen. The cost metric is the number of lines of reasonably commented and formatted code which are modified or added compared with the original x86 code base (excluding device drivers).

In both OSes, the architecture-specific sections are effectively a port of the x86 code to their paravirtualized architecture.

Evaluation

- Based on Linux 2.4.21 (neither XP nor NetBSD fully functional)
- Thoroughly compared to 2 other systems
  - VMware Workstation (binary translation)
  - UML (run Linux as a Linux process)
- Performs better than solutions with restrictive licenses (ESX Server)

Relative Performance



CPU-bound tasks

- Relatively easy for all VMMs
- Little interaction with the OS

Legend

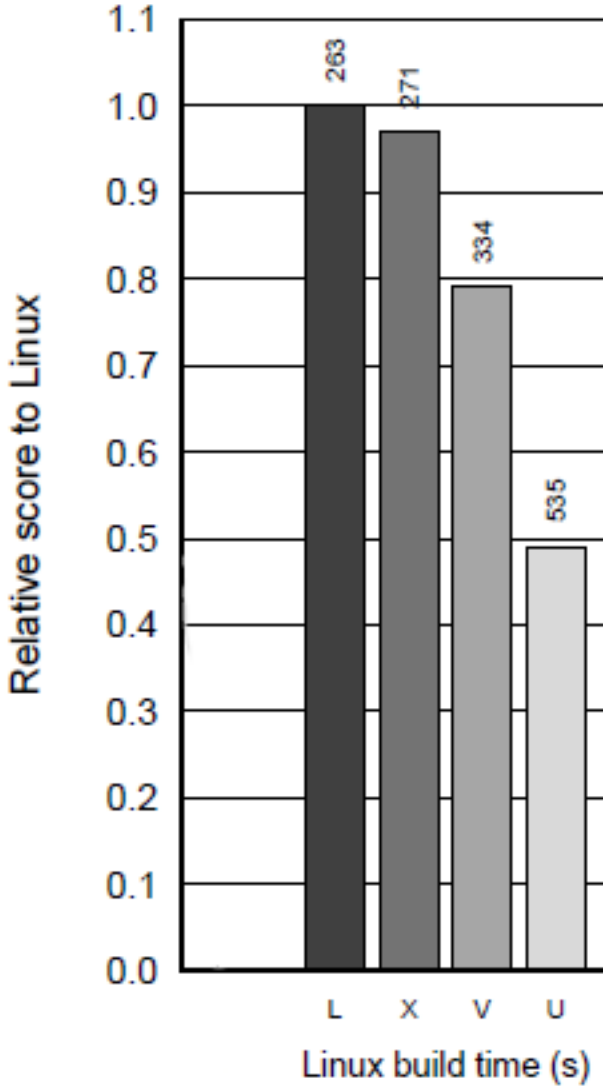
L – Linux

X – XenoLinux

V – VMware

U – User-mode Linux

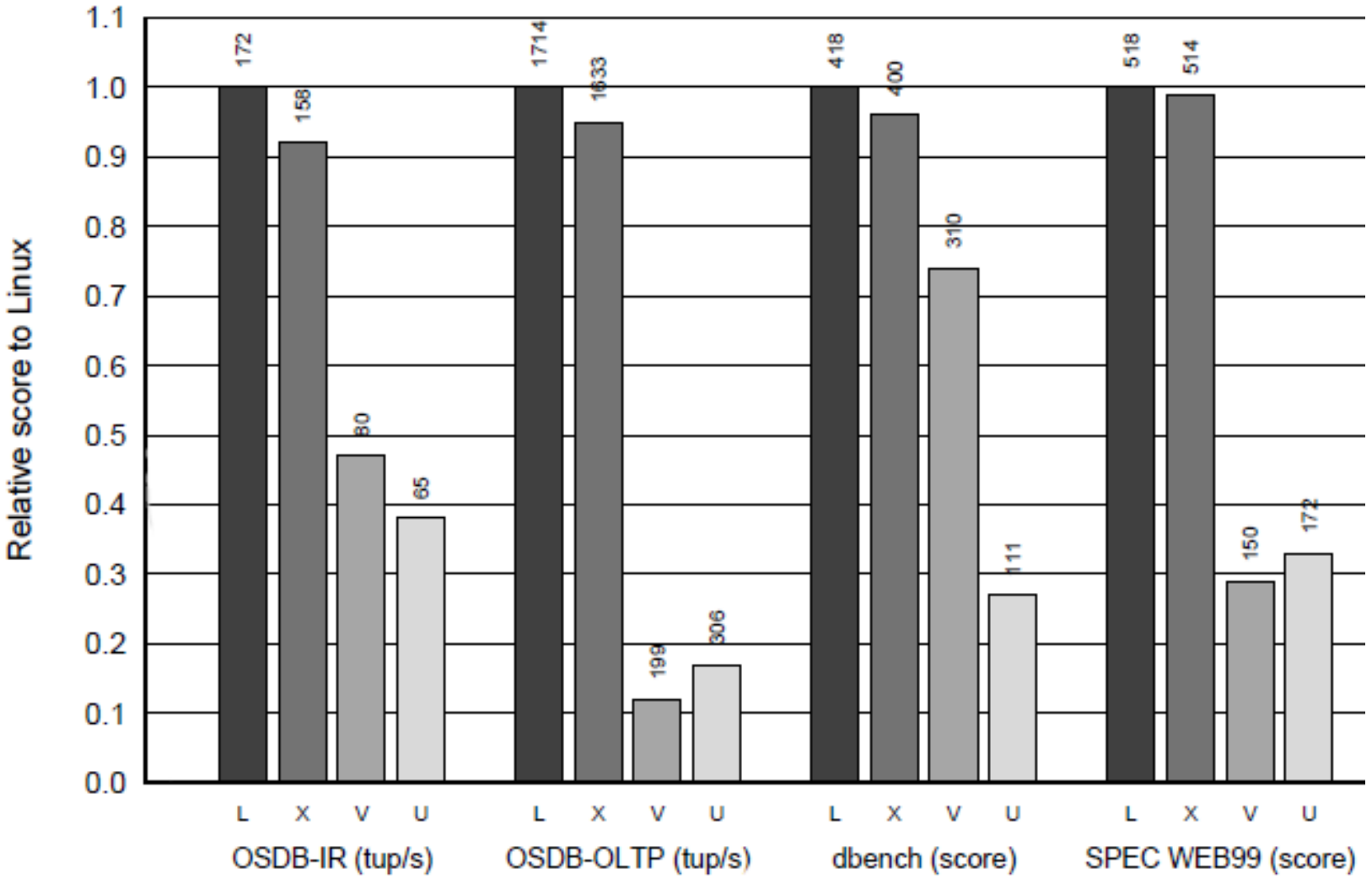
Relative Performance



Tasks with more I/O

- About 7% of the time spent in OS doing I/O and memory management
- This portion of time gets expanded for each VMM but to a different extent

Relative Performance



Microbenchmarks

Config	null call	null I/O	stat	open closeTCP	slct	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

- As expected *fork*, *exec* and *sh* require large number of page updates which slow things down
- On the up side these can be batched (up to 8MB of address space constructed per hypercall)

Microbenchmarks

Config	null call	null I/O	stat	open closeTCP	slct	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

- Hmmm …. no calls into XEN yet …

Microbenchmarks

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

- Overhead due to a hypercall when switching context in a guest OS (in order to change base of page table)
- The larger the working set the smaller the relative overhead

Microbenchmarks

Config	0K File		10K File		Mmap	Prot	Page
	create	delete	create	delete	lat	fault	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

- 2 transitions into XEN
  - One for the page fault handler
  - One to actually get the page

Microbenchmarks

	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

- Page flipping really pays off – no unnecessary data copying
- More overhead for smaller packets – we still need to deal with every header

Concurrent VMs

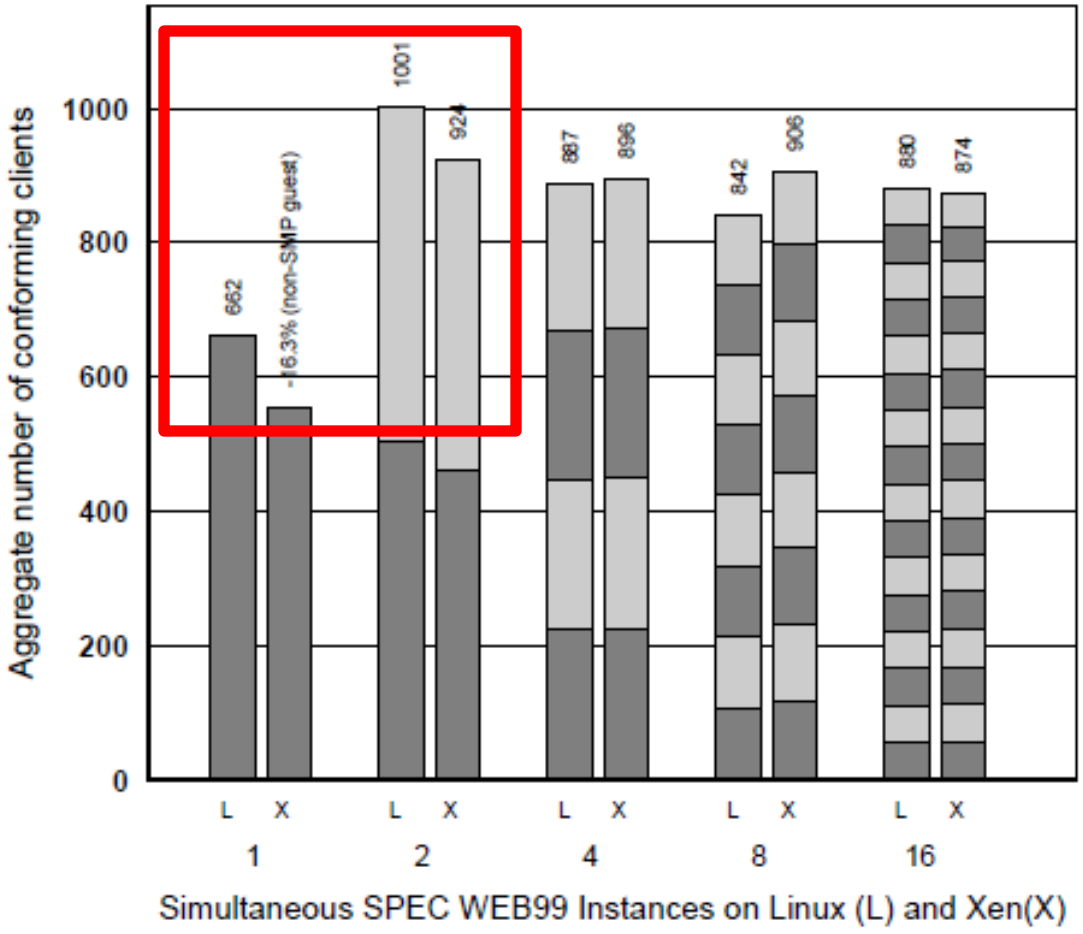
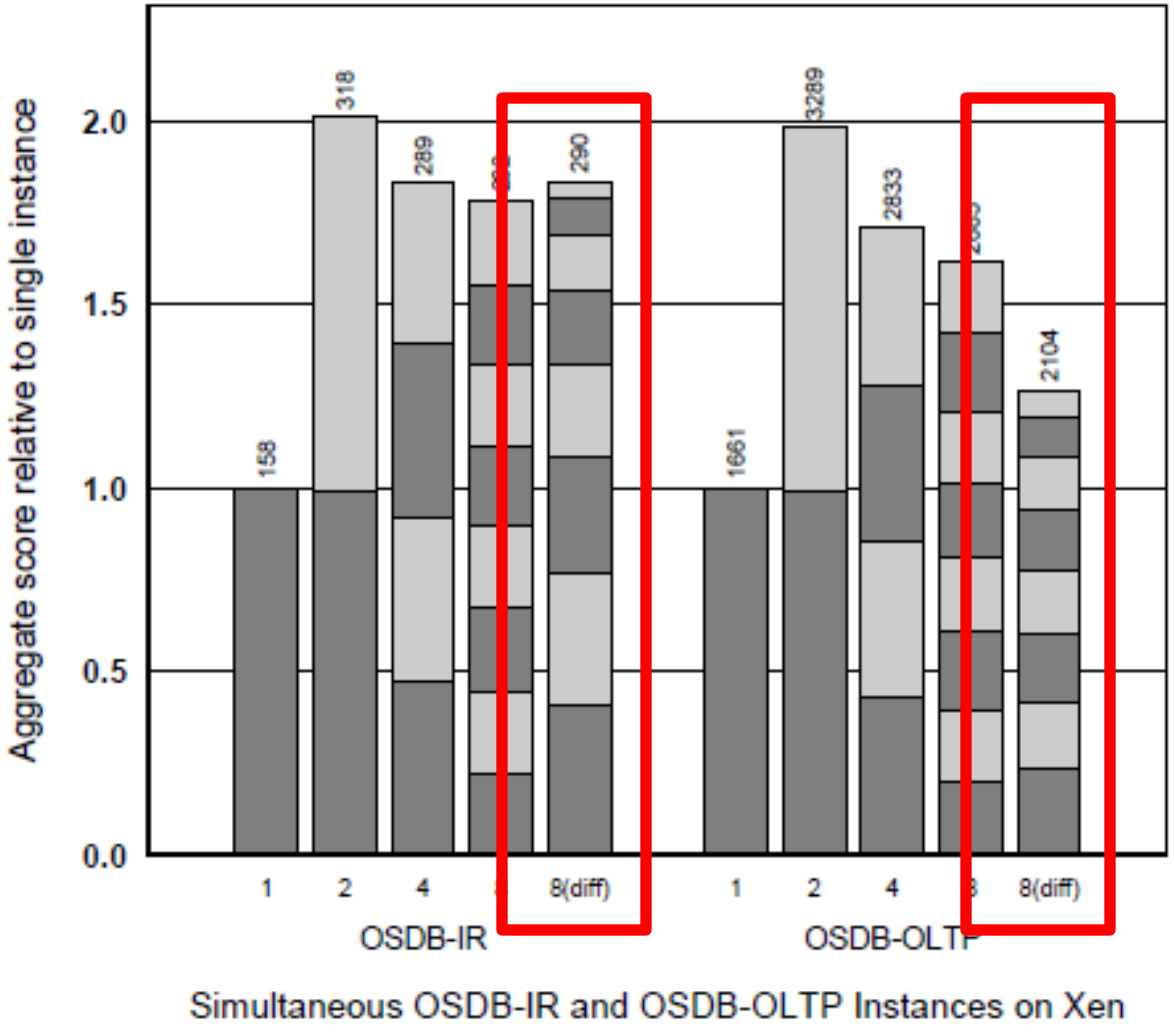


Figure 4: SPEC WEB99 for 1, 2, 4, 8 and 16 concurrent Apache servers: higher values are better.

- Unexpectedly low SMP performance for 1 instance of Apache
- As expected adding another domain leads to a sharp jump in performance under XEN
- More domains – more overhead

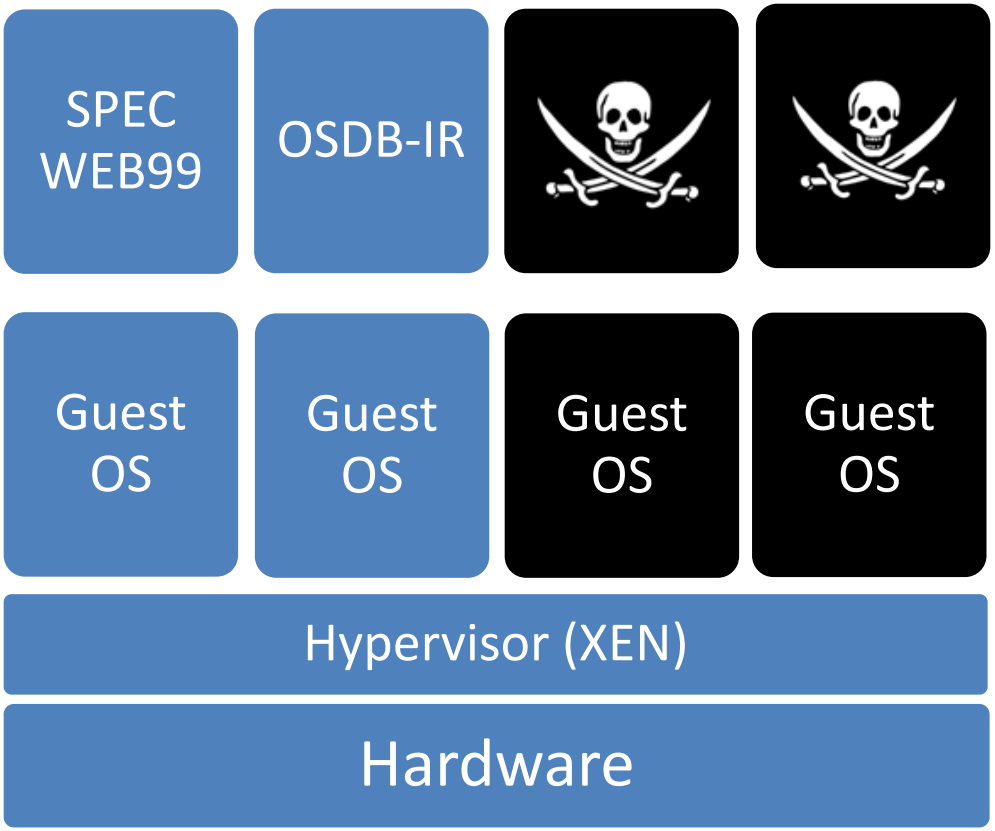
Concurrent VMs



Simultaneous OSDB-IR and OSDB-OLTP Instances on Xen

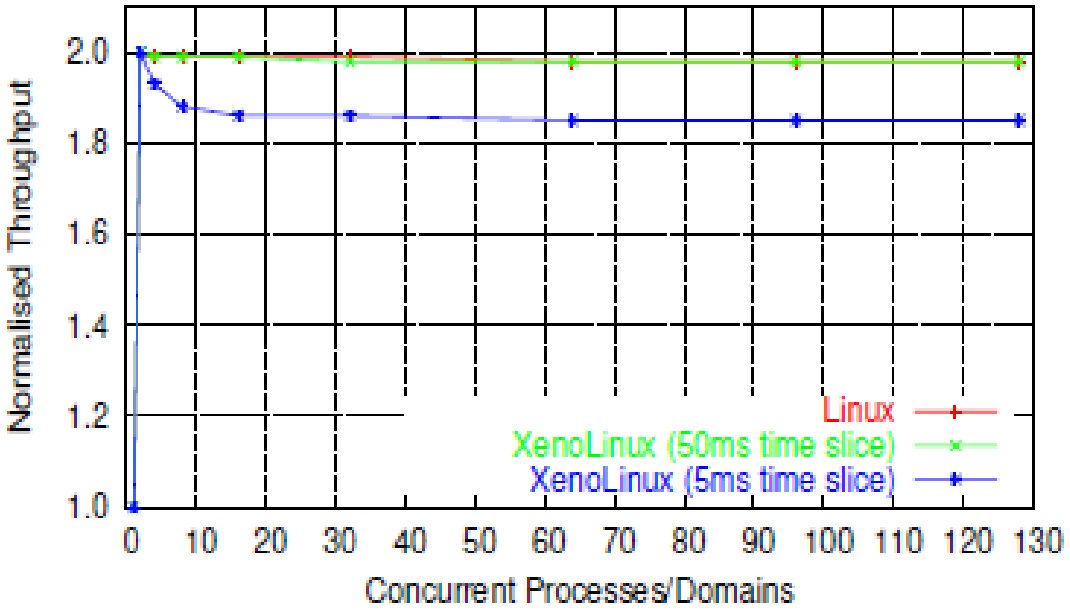
- Performance differentiation works as expected with IR
- But fails with OLTP
- Probably due to inefficiencies with the disk scheduling algorithm
- Bands matter !!!

Isolation



- Run uncooperative user applications, see if they bring down the system
- 2 “bad” domains vs 2 “good” ones
- XEN delivers good performance even in this case
- What about an uncooperative OS ?

Scalability



- Very low footprint per domain (4 - 6MB memory, 20KB state)
- Benchmark is compute-bound and Linux assigns long time slices, XEN needs some tweaking
- Even without it does pretty well (but no absolute values)

Criticism

- No comparison between fundamentally similar techniques (e.g. the big IBM mainframe)
- Memory footprint almost not mentioned
- Most tests performed with limited number of domains, while the paper's main goal is to demonstrate performance with 100 domains
- Benchmarks used relevant today ?

Outline

Xen and the Art of Virtualization – Ian Pratt *et al.*

I. The Art of Virtualization
II. Xen, goals and design
III. Xen evaluation
IV. The future looks bright

Related work

- Started by IBM in the 70s
- Resurgence lately (at time of publication) as hardware got fast enough to run multiple OS
- Unlike in other solutions isolation/security is “implicitly” enforced by the hypervisor
- One of the first attempts at paravirtualization the other two are:
  - IBM zSeries mainframes (Linux)
  - Denali (quite a bit different)

Xen vs. Denali ?

Feature	Xen	Denali
# VM	100	1000
Target existing ABI	Yes	No (does not fully support x86 segmentation, used in Net BSD, Linux, Windows XP)
Supports application multiplexing	Yes	No (1 app/OS)
Supports multiple address spaces	Yes	*No (prototype virtual MMU developed, helping Denali) => no evaluation*
Memory management support at the virtualization layer	Yes	No (VMM performs all paging to and from disk)
Performance isolation	Yes, each guest OS performs own paging using guaranteed memory reservation and disk allocation	No, malicious VM can encourage thrashing behavior
Virtualizes namespaces of all machine resources	No (Secure access control within hypervisor only)	Yes (resource isolation through impossible naming)

Future work and impact

- Huge impact of the paper, sparked a lot of interest, the project is still very much alive
- They did actually complete the XP port, but due to licensing restrictions it never got published
- Currently only supports Windows as guest if the hardware supports virtualization
- More recent versions of the project try to push complexity away from the hypervisor
- However, paravirtualization nowadays is only used if the hardware does not support virtualization natively

Any questions?