

# Formal Requirements for Virtualizable Third Generation Architectures

Presented by: Christopher Boggs on 10/01/2018 for EECS 582

Authored by: Gerald J. Popek and Robert P. Goldberg

# Outline

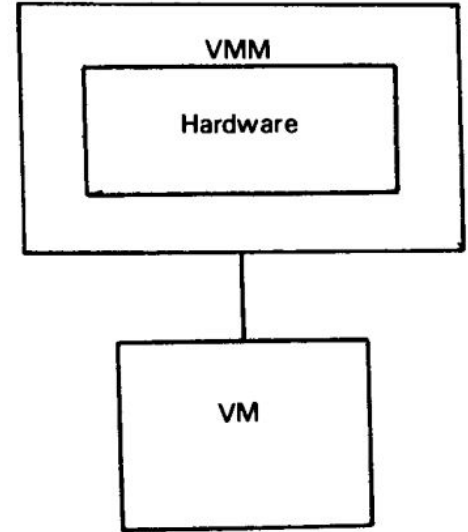
- Background
- Model of 3rd Gen Architectures
- Instructions
- Virtual Machine Monitors
- Theorems
- Strengths/Weaknesses
- Discussion



# Virtual Machines / Monitors

- Whats a VM? (1974)
- VM is *efficient, isolated duplicate* of a real machine

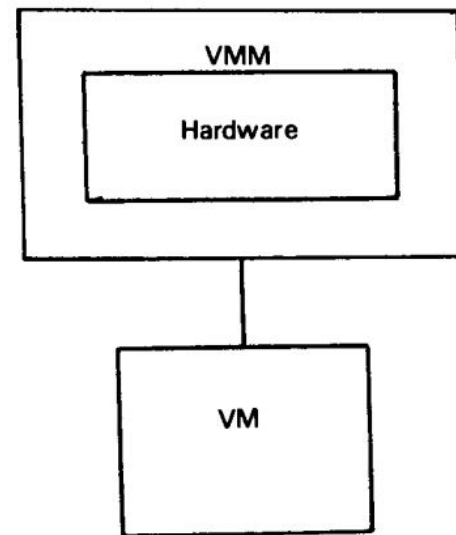
Fig. 1. The virtual machine monitor.



# Virtual Machines / Monitors

- Whats a VM? (1974)
- VM is *efficient, isolated duplicate* of a real machine
- Virtual Machine Monitor (VMM)
  - Mimics original machine for applications
  - Programs run ~normal speed
  - Completely controls system

Fig. 1. The virtual machine monitor.



# Can a Machine Support a VM?

- Actually pretty simple test (3rd gen)



# Can a Machine Support a VM?

- Actually pretty simple test (3rd gen)
- Once we define a LOT of notation/terms




# Can a Machine Support a VM?

- Model of 3rd gen computer exists
- We can derive conditions on this model
  - If sufficient, can support VMs



# Our Model

- $S = \langle E, M, P, R \rangle$
  - Model defines state
    - Specifies real computer
    - Each state defined by E, M, P, R
  - E = Executable Storage
  - M = Processor Mode
  - P = Program Counter (sometimes PC)
  - R = Relocation Register
- 



# Executable Storage (E)

- Word/byte addressed memory of size  $q$
- $E[i]$  is the contents in the  $i$ -th storage unit
- $E=E'$  iff  $E[i]=E'[i]$  for any  $i$  in  $\text{len}(q)$



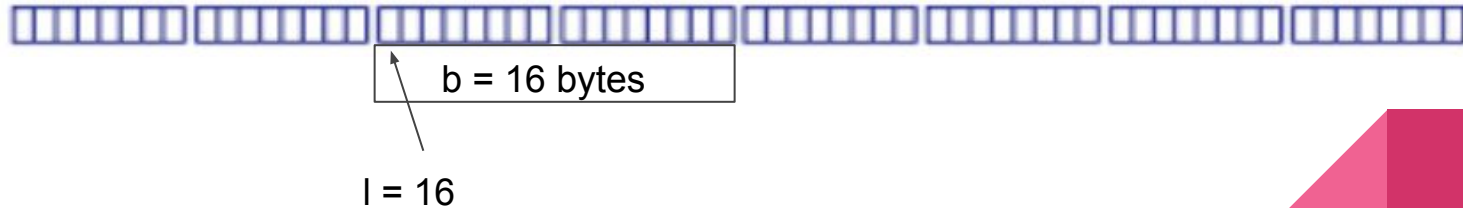
# Processor Mode (M)

- Operation modes, restrict type/scope of operations
- Supervisor mode (s)
  - All instructions available to processor
- User mode (u)
  - Not all instructions available to processor



# Program Counter (P) and Relocation Register (R)

- P indexes into R
- R indexes into E
- Relocation Register is a tuple (I,b)
  - I = absolute memory address
  - b = absolute size



# Program Status Word (PSW)

- Contents of  $\langle M, P, R \rangle$
- In one storage location
- $E[0]$  is assumed for old-PSW
- $E[1]$  is assumed for new-PSW fetched



# Instructions

- Statement of a processor
- Specifies changing of state in machine
- Transforms a state in the machine to another
- I.e  $i(S1)=S2$  or  $i(E1,M1,P1,R1)=(E2,M2,P2,R2)$



# Trap

- Instruction causes interrupt due to a condition
- Not executable in this state
- Protocol
  - Save state
  - Premade routine to reverse state
  - “Invalid” instructions have no effect
- Memory traps - instr for invalid address



# Memory Trap

- Relocation register is (l,b)
- Instruction results in location a



# Memory Trap

- Relocation register is (l,b)
- Instruction results in location a

```
if a + l >= total-real-memory-size then
    // out of real memory bounds
    memorytrap
else if a >= b then
    // out of virtual memory bounds
    memorytrap
else
    use address a+l
```



# Instructions

- Privileged
  - Can run in supervisor but not user mode
  - Traps on certain states  $i(S)$  traps but not  $i(S1)$
- Sensitive
  - Interacts with hardware
  - Behavior and Control Sensitive
- Innocuous
  - Not sensitive



# Sensitive Instructions

- Control
  - Attempts to change relocation register (R) and/or mode (M)
  - Changes available resources or mode without a trap
- Behavior
  - Effect of execution is dependent
  - Location Sensitive=depends on value in R
  - Mode Sensitive=depends on M in prev state



# Virtual Machine Monitor (VMM)

- Software, called *control program*
- Dispatcher (D) - top level control module
- Allocator (A) - allocates system resources
- Interpreter (I)
  - Simulates privileged instruction when trapped
  - One routine per privileged instruction



# Control Program

- $CP = \langle D, A, \{IR\} \rangle$  where IR set of all interpretive routines
- Dispatcher controls is in  $E[1]$
- CP runs in supervisor where everything else is user



# VM Properties

- Efficiency
  - Little to no speed decreases
  - Innocuous instructions executed normally (w/o CP)
- Resource Control
  - VMM has complete control
  - Utilizes allocator
- Equivalence
  - Environment “identical” to original machine
  - Timing and availability restrictions



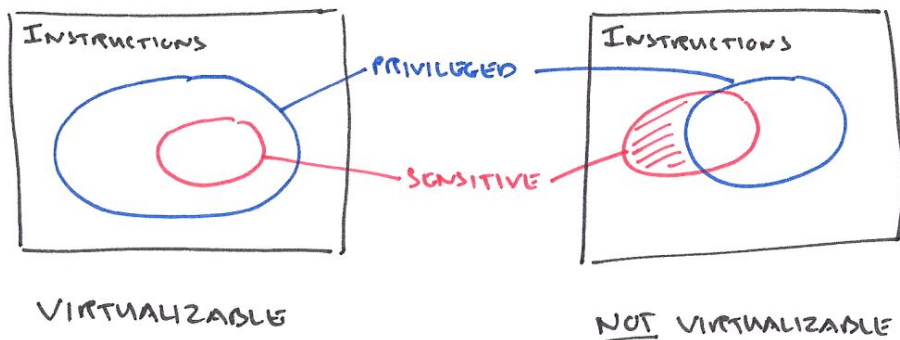
# Third Generation Computers

- Consists of:
  - Relocation mechanisms (l,b)
  - Supervisor/user mode
  - Trap mechanisms
- We now have 3 theorems for virtualization on these machines



# Theorem 1

- VMM may be constructed if set of sensitive instructions is a subset of set of privileged instructions



# Virtual Machine Map

- We can explore checking Theorem 1 for a machine
- Notation
  - $C$  - set of machine states
  - $CV$  - states with VMM in memory
  - $CR$  - states of physical machine, no VMM
  - $e$  - an instruction sequence, is  $n$  inst long

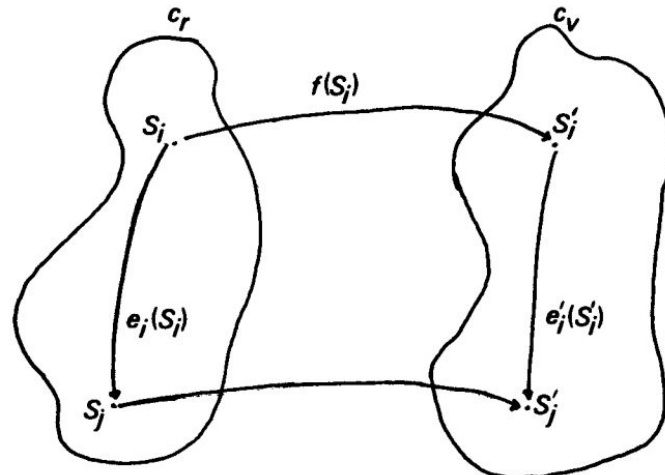




# Virtual Machine Map

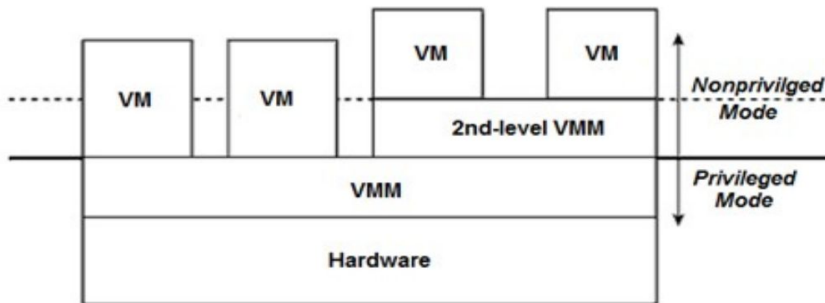
- A mapping  $f$
- Any state in CR has a mapping to CV
- The mapping still holds after any instruction sequence ( $e$ )

Fig. 2. The virtual machine map.



# Theorem 2

- A conventional third gen computer is recursively virtualizable if it is:
  - Virtualizable
  - A VMM without timing dependencies can be made for it
- The VM can run a copy of the VMM with the same properties
- Other VMMs and original are indistinguishable



# Hybrid Virtual Machines

- Hard to virtualize third gen architectures
- Hybrid Virtual Machine is more general, less efficient VM
- More instructions are interpreted



# Theorem 3

- A hybrid virtual machine monitor may be constructed for any conventional third generation machine in which the set of user sensitive instructions are a subset of the set of privileged instructions
- User sensitive instructions
  - Difficult to run in user mode
  - Control or location sensitive in user mode



# What Do We Have? (Contribution)

- Defined states in a machine
- Defined effects of instructions
- We can model third gen architectures!
- Determine if a machine can support a VMM
- Theorems to help define complex virtualizability



# Strengths



# Strengths

- “Simple” model of third gen architecture
- Intuitively makes a good amount of sense
- Overall requirement is quite simple
- Always a way to “virtualize”



# Weaknesses





# Weaknesses

- Lots of notation
- Simple explanations would help more
- Potentially oversimplified models?



# Discussion



# Discussion

- What would change in applying this to modern computers?
- How to minimize timing restrictions?
- Running multiple VMs, different OS?



# Xen and the Art of Virtualization

Paul Barham, Boris Dragovic et al.

Presented by

Liran Xiao

September 30, 2018

# Why virtualization

- Subdivide ample resources
- Consolidation
- Simulation
- Failure tolerance
- Reduce migration cost

# Resource control

- CPU, Memory, Storage...

# Popek and Goldberg Theorem on x86

## Popek and Goldberg Theorem

a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

## What about x86 instructions?

- 17+ sensitive, unprivileged instructions
- "popf problem"

# Popek and Goldberg Theorem on x86

PUSHF

ANDL \$0x003FFDFF, (%ESP)

POPF

*# %EFLAGS onto stack*

*# Clear IF on stack*

*# %EFLAGS from stack*

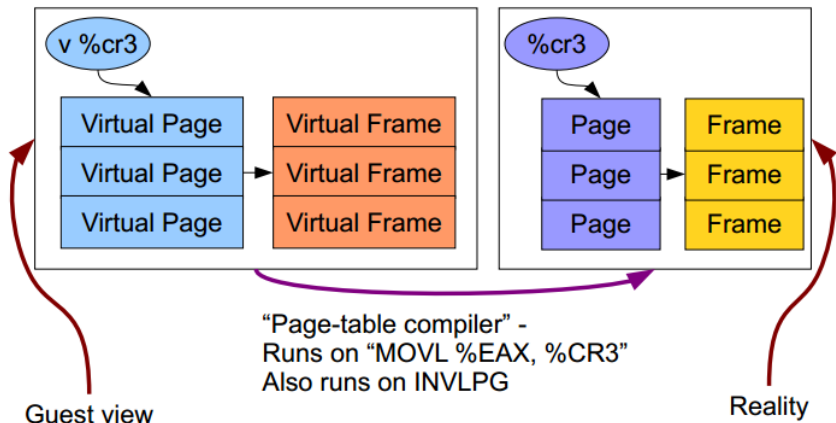
Mode	Operand Size	CPL	IOPL	Flags																Notes
				21 ID	20 VIP	19 VIF	18 AC	17 VM	16 RF	14 NT	13:12 IOPL	11 OF	10 DF	9 IF	8 TF	7 SF	6 ZF	4 AF	2 PF	
Real-Address Mode (CR0.PE = 0)	16	0	0-3	N	N	N	N	N	0	S	S	S	S	S	S	S	S	S	S	
	32	0	0-3	S	N	N	N	N	0	S	S	S	S	S	S	S	S	S	S	
Protected, Compatibility, and 64-Bit Modes (CR0.PE = 1 EFLAGS.VM = 0)	16	0	0-3	N	N	N	N	N	0	S	S	S	S	S	S	S	S	S	S	
	16	1-3	<CPL	N	N	N	N	N	0	S	N	S	S	S	S	S	S	S	S	
	16	1-3	≥CPL	N	N	N	N	N	0	S	N	S	S	S	S	S	S	S	S	
	32, 64	0	0-3	S	N	N	S	N	0	S	S	S	S	S	S	S	S	S	S	
	32, 64	1-3	<CPL	S	N	N	S	N	0	S	N	S	S	N	S	S	S	S	S	
	32, 64	1-3	≥CPL	S	N	N	S	N	0	S	N	S	S	S	S	S	S	S	S	
Virtual-8086 (CR0.PE = 1 EFLAGS.VM = 1 CR4.VME = 0)	16	3	0-2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
	16	3	3	N	N	N	N	N	0	S	N	S	S	S	S	S	S	S	S	
	32	3	0-2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
	32	3	3	S	N	N	S	N	0	S	N	S	S	S	S	S	S	S	S	
VME (CR0.PE = 1 EFLAGS.VM = 1 CR4.VME = 1)	16	3	0-2	N/X	N/X	SV/X	N/X	N/X	0/X	S/X	N/X	S/X	S/X	N/X	S/X	S/X	S/X	S/X	S/X	2,3
	16	3	3	N	N	N	N	N	0	S	N	S	S	S	S	S	S	S	S	
	32	3	0-2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
	32	3	3	S	N	N	S	N	0	S	N	S	S	S	S	S	S	S	S	



# Full virtualization (VMware)

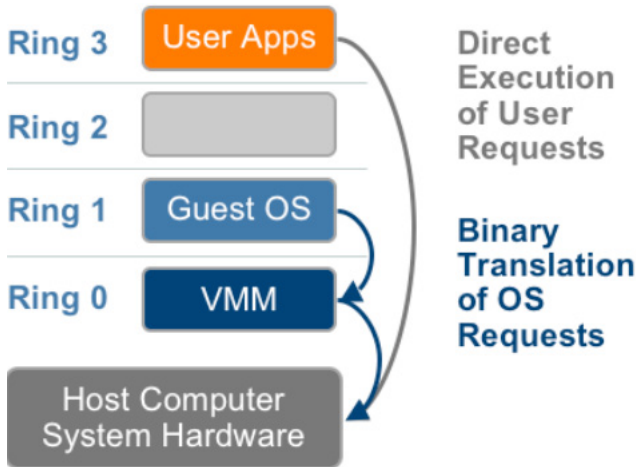
Guest OSes are blind.

- Shadow version of system structures



# Full virtualization (VMware)

- Binary translation popf → int \$99



# Full virtualization (VMware)

- Pros:
  - Allow unmodified OS and binaries
  - Don't need hardware assistance
- Cons:
  - Large performance overhead
  - Hosted OS wants both virtual and real resources

# Xen - Paravirtualization

Guest OSeS know it.

## Backward compatibility

Support for unmodified application binaries is essential.

## Support

Supporting full multi-application operating systems is important.

## VMM essentials

Paravirtualization is necessary to obtain high performance and strong resource isolation on uncooperative machine architectures such as x86.

## Don't hide power

Even on cooperative machine architectures, completely hiding the effects of resource virtualization from guest OSeS risks both correctness and performance.

# x86 virtual machine interface

<b>Memory Management</b>	
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 discontinuous machine pages.
<b>CPU</b>	
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.
<b>Device I/O</b>	
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.**

# Memory management

## Avoid TLB flush

- x86 TLB: hardware-managed, not tagged
- Xen on top 64MB sections of every address space

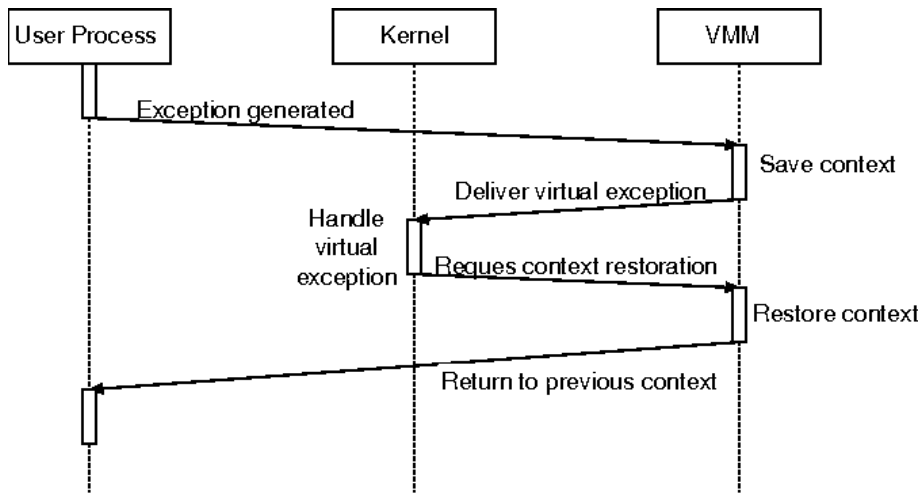
## Guest OS manages the hardware PT / segment descriptor

- register pages / segment descriptor to Xen
- read: guest OS → hardware MMU
- update (batched): guest OS (hypercall) → Xen (validation) → hardware MMU

# CPU

- privilege level
  - hypervisor (ring 0) > guest OS (ring 1) > application (ring 3)
- privileged instructions
  - hypercalls
  - validated and executed by Xen
- exception handling
  - similar to real x86 handlers
  - exception in exceptions: page fault handler
  - fast exception handler
  - double fault

## CPU





# Device I/O

- Actual device  $\Rightarrow$  set of device abstraction
- I/O data: shared asynchronous buffer-descriptor ring
- Interrupt  $\Rightarrow$  lightweight event-delivery

# Control and Management

- Hypervisor: basic control operations
- Domain0
  - created at boot
  - all application-level management

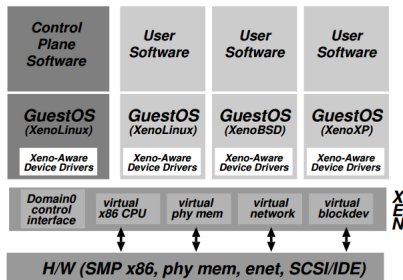


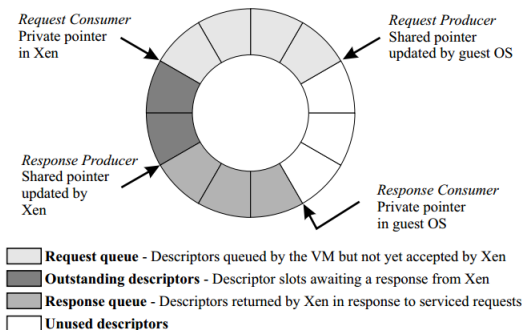
Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including *Domain0* running control software in a XenoLinux environment.

# Control transfer

- $\text{domU} \rightarrow \text{Xen}$ : synchronous hypercall
- $\text{Xen} \rightarrow \text{domU}$ : asynchronous events

# Data transfer

- Ring: circular queue of descriptors
- I/O Data: allocated out-of-band
- I/O Rings



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

# CPU scheduling

## Borrowed virtual time

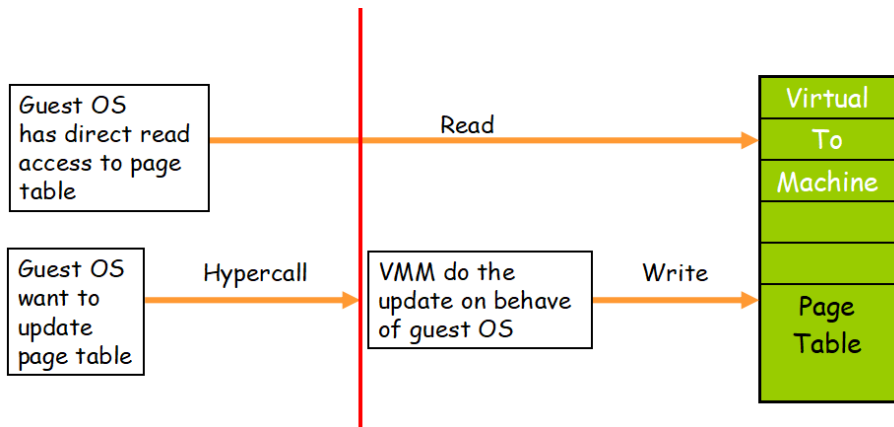
- proportional-share
- work conserving
- low-latency dispatch by warping

$$E_v = T/w_i - \text{wrap} \cdot w_i$$

# Time and timers

- real time: nanoseconds since booting
- virtual time: domain execution time
- wall-clock time: offset of real time
- guest OS alarm timers: (real time, virtual time)

# Virtual memory



# Virtual memory

## Machine frames

- reference count
- type pinning

### None

No special uses.

### L/N Page table page

Pages used as a page table at level  $N$ . There are separate types for each of the 4 levels on 64-bit and 3 levels on 32-bit PAE guests.

### Segment descriptor page

Pages used as part of the Global or Local Descriptor tables (GDT/LDT).

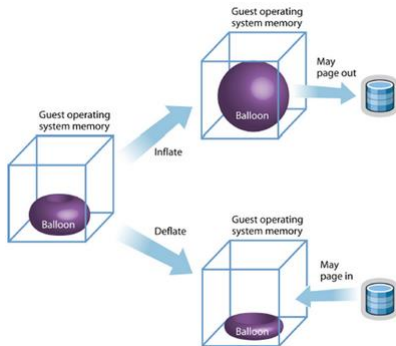
### Writeable

Page is writable.



# Physical memory

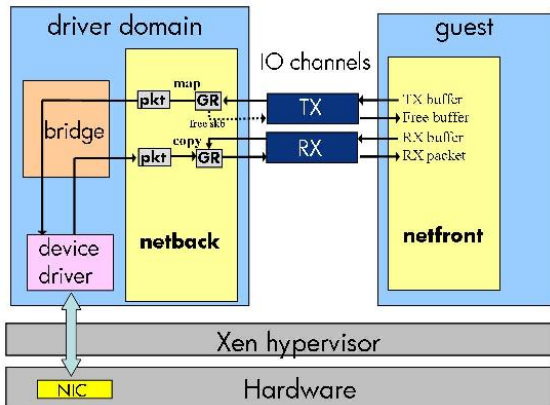
- statically initial memory reservation
- dynamic balloon driver



- guest physical address  $\rightarrow$  hardware address
  - P2M & M2P table

# Network

- VFR (virtual firewall router)
- VIF (virtual network interface)
  - rules (< pattern >, < action >)
- page flipping



# Disks

- dom0: unchecked access to physical disks
- domU: VBD (virtual block devices)
  - Reorder requests
  - DMA

# 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
<b>Total</b>	<b>2995</b>	<b>4620</b>
<b>(Portion of total x86 code base</b>	<b>1.36%</b>	<b>0.04%)</b>

**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).**

# Relative performance

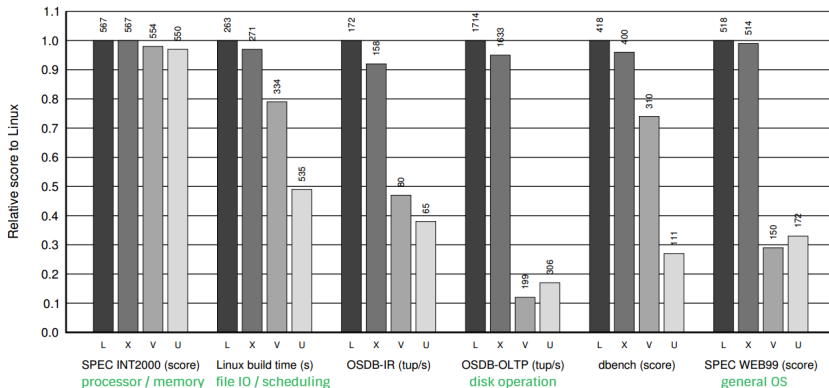


Figure 3: Relative performance of native Linux (L), XenoLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).

# OS benchmarks

Config	null call	null I/O	stat	opens close	slct TCP	sig inst	sig hdl	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	<b>198</b>	<b>768</b>	<b>4k8</b>
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: 1mbench: Processes - times in  $\mu s$ 

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	<b>1.97</b>	<b>2.22</b>	<b>2.67</b>	<b>3.07</b>	<b>28.7</b>	<b>7.08</b>	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: 1mbench: Context switching times in  $\mu s$ 

Config	0K File		10K File		Mmap lat	Prot fault	Page fault
	create	delete	create	delete			
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	<b>139</b>	1.40	<b>2.73</b>
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: 1mbench: File & VM system latencies in  $\mu s$

# Network performance

	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**

# Concurrent performance

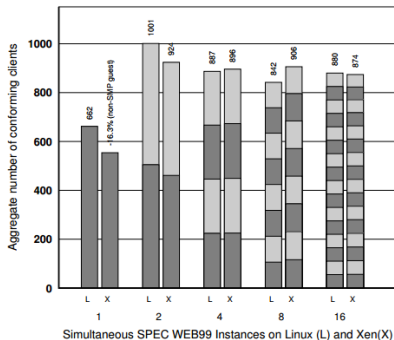


Figure 4: SPEC WEB99 for 1, 2, 4, 8 and 16 concurrent Apache servers: higher values are better.

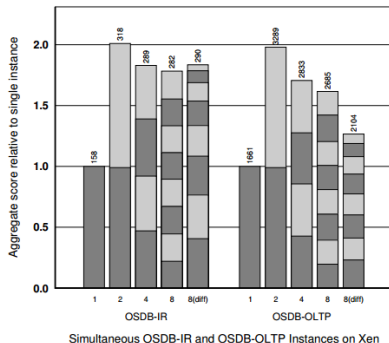


Figure 5: Performance of multiple instances of PostgreSQL running OSDB in separate Xen domains. 8(diff) bars show performance variation with different scheduler weights.



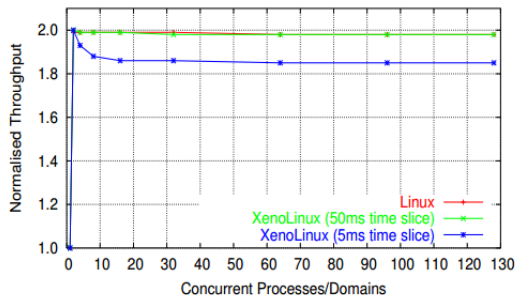
# Performance isolation

## 4 equally configured Domains

- PostgreSQL/OSDB-IR
- SPEC WEB99
- Disk bandwidth hog
- Fork bomb

# Scalability

- Memory overhead: 4-6MB footprint, 20KB state
- Context-switching overhead



**Figure 6: Normalized aggregate performance of a subset of SPEC CINT2000 running concurrently on 1-128 domains**

# Discussion

- Strength
  - High performance
- Weakness
  - Inevitability of dom0
  - Need to modify operating systems
  - Lack of SMP support (at the paper's time)
- Open questions
  - Hardware-supporting virtualization
    - Make sensitive instructions all privileged (EFLAGS.VM)
    - Provide nested paging (EPT)
    - Device I/O Intel VT-d

The End  
Thank you for listening!