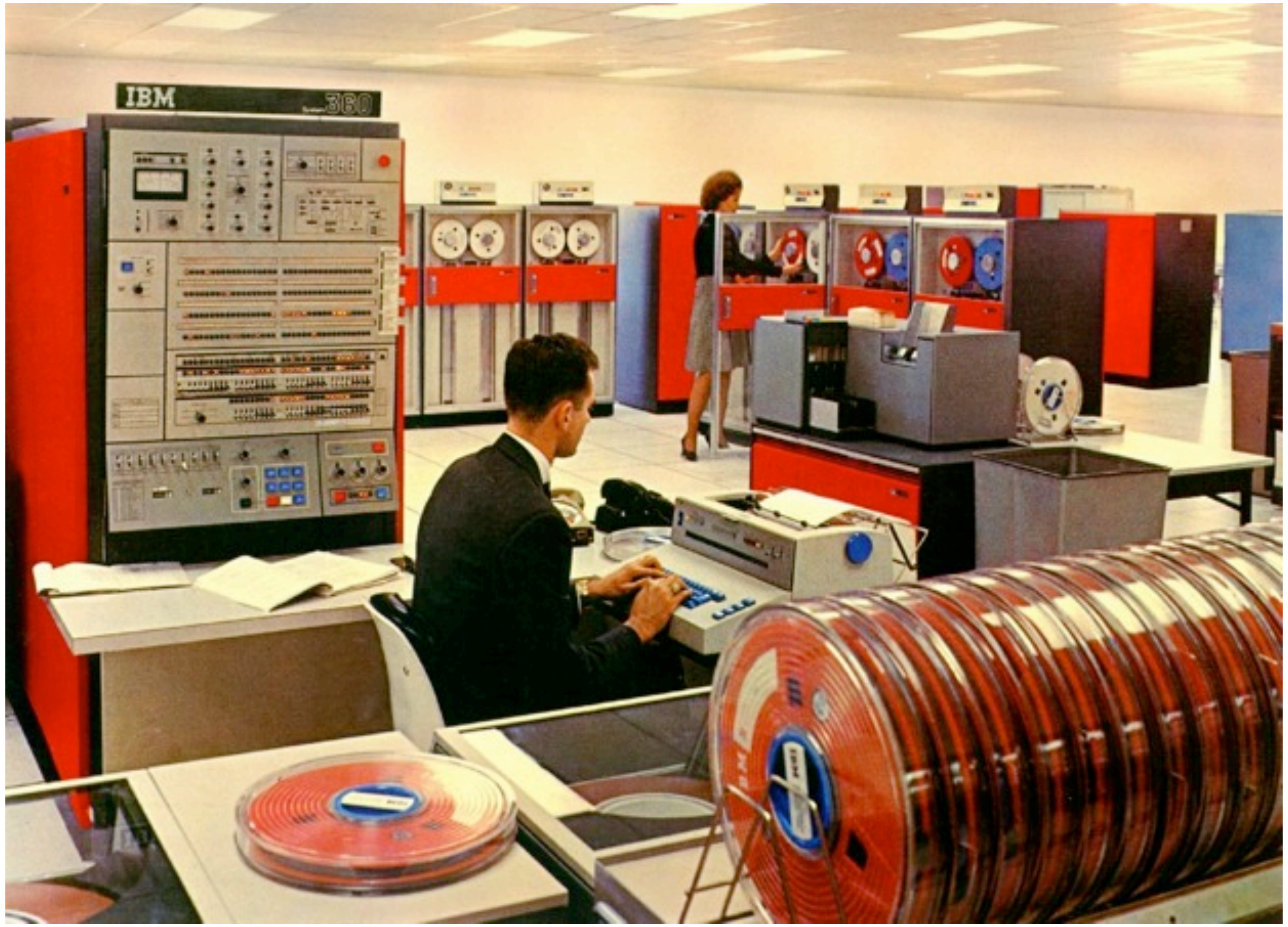
Xen and the Art of Virtualization

Panut Sookpranee

10/1/09

Virtual Machine: Origin



- IBM CP/CMS
  - CP-40
  - CP-67
  - VM/370

Why virtualize?

- Underutilized machines
- Easier to debug and monitor OS
- Portability
- Isolation
- EC2

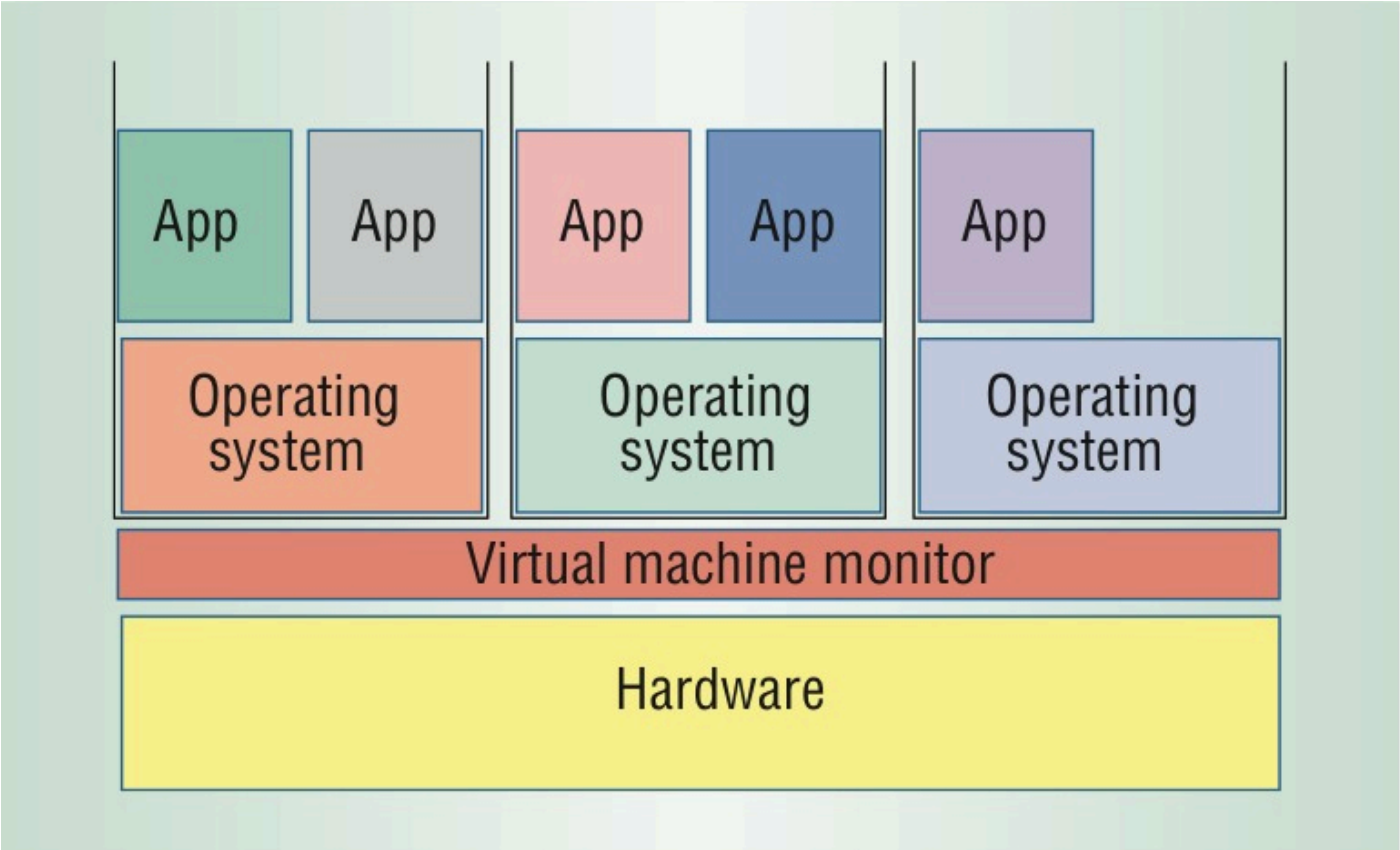
Full Virtualization

- Complete simulation of underlying hardware
- Unmodified guest OS
- Trap and simulate privileged instruction
- Was not supported by x86 (Not true anymore, Intel VT-x)
- Guest OS can't see real resources

Paravirtualization

- Similar but not identical to hardware
- Modifications to guest OS
- Hypercall
- Guest OS registers handlers
- Improved performance

Classic VMM



App
App
App
App
App
Operating system
Operating system
Operating system
Virtual machine monitor
Hardware

VMware ESX Server

- Full virtualization
- Dynamically rewrite privileged instructions
- Ballooning
- Content-based page sharing

Denali

- Paravirtualization
- 1000s of VMs
- Security & performance isolation
- Did not support mainstream OSes
- VM uses single address space

Xen

- History
- Design philosophy
- Virtual interfaces/implementation
- Evaluation
- μ-Kernel?

Xen

- University of Cambridge, MS Research Cambridge
- XenSource, Inc.
- Released in 2003
- Acquired by Critix Systems in 2007 for $500M
- Now in RHEL5, Solaris, SUSE Linux Enterprise 10, EC2

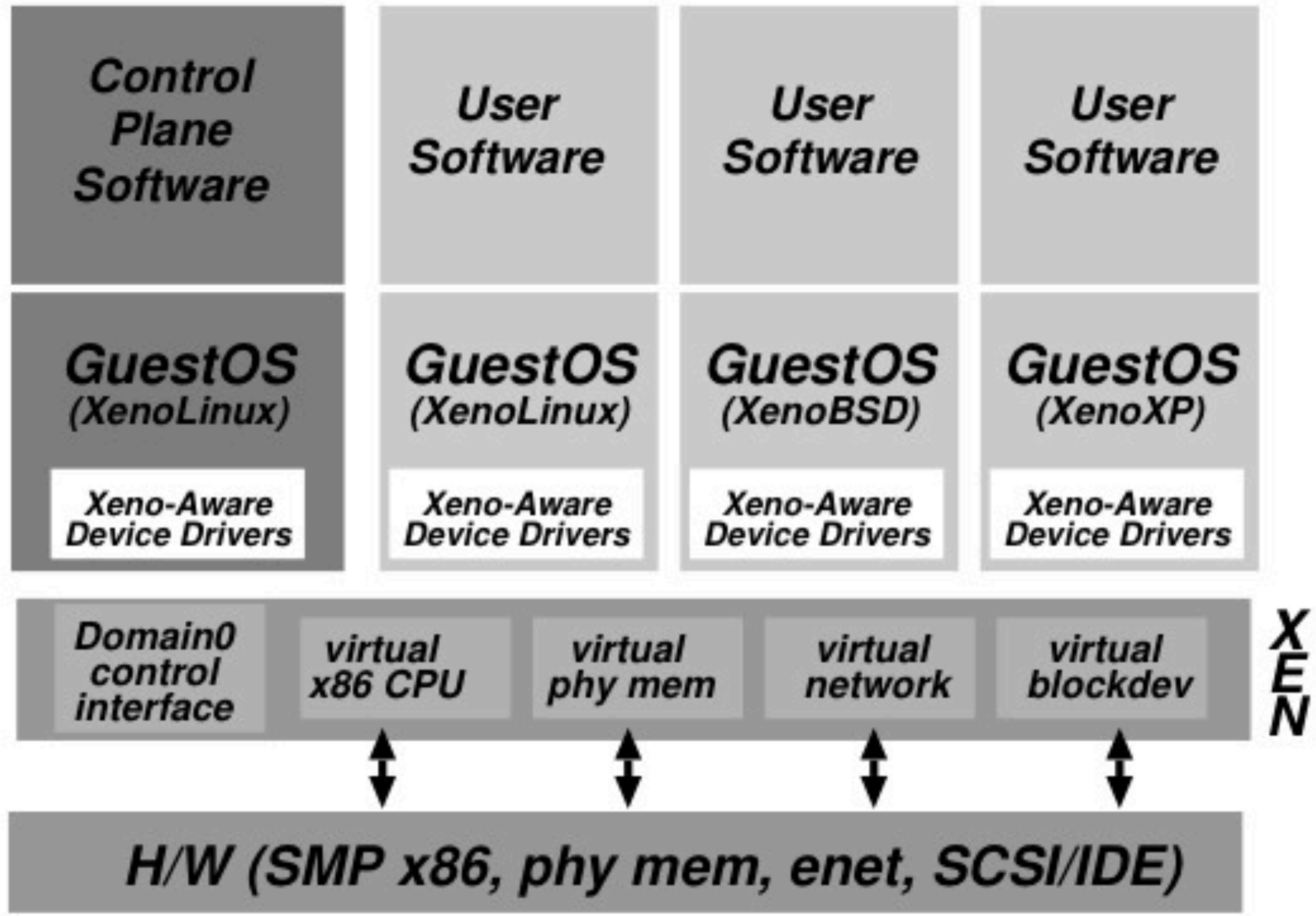
Xen

- No changes to ABI
- Full multi-application OS
- Paravirtualization
- Real and virtual resources
- Up to 100 VMs

Xen 3.0 supports full virtualization with hardware support.

Domain0

- Management interface
- Created at boot time
- Policy from mechanism
- Privileged


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)

Control Transfer

- Hypercalls
- Lightweight events

Interface: Memory Management

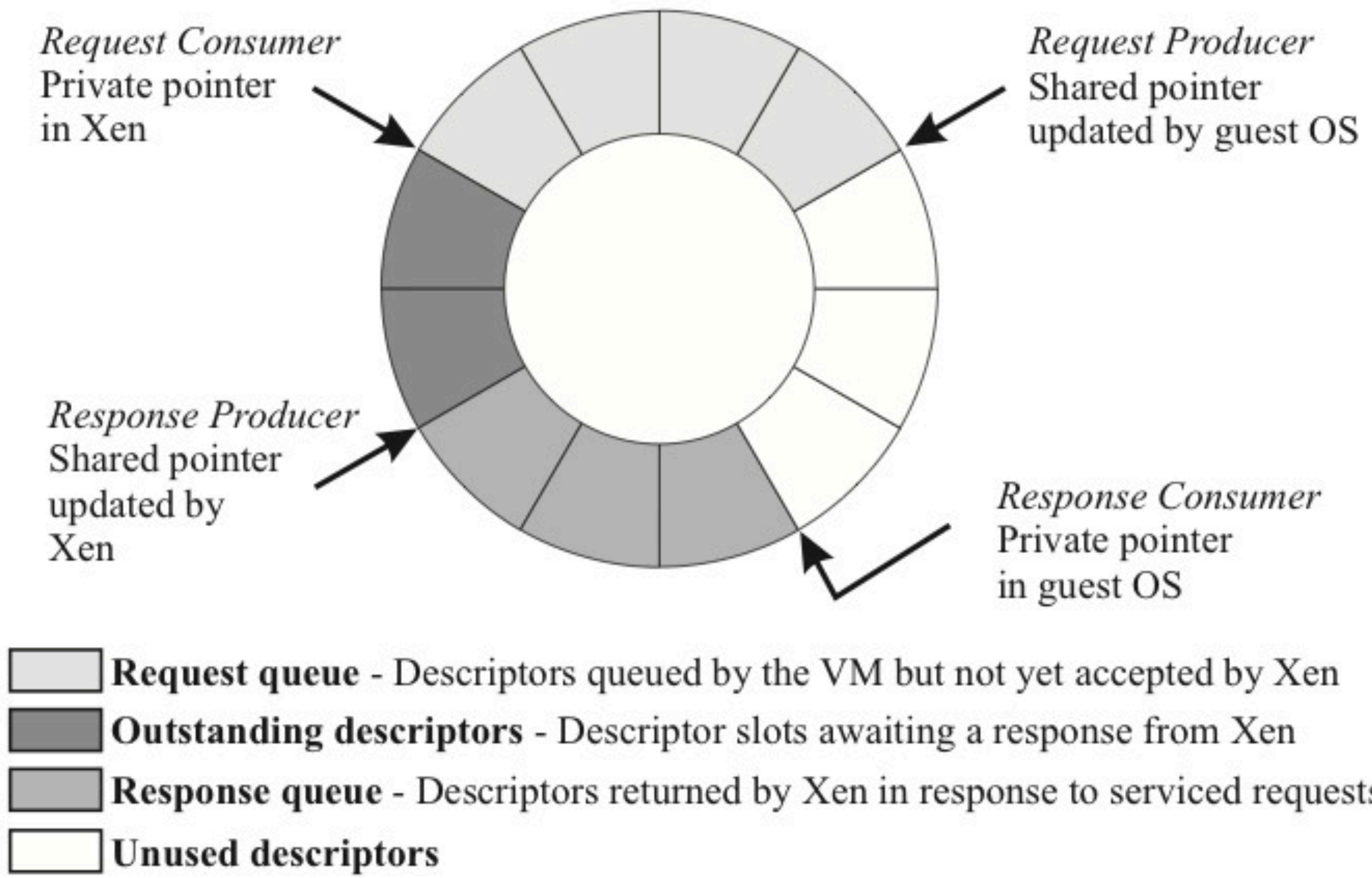
- Guest OSes manage their own page tables
- Register pages with Xen
- No direct write access
- Updates through Xen
- Hypervisor @ top 64MB of every address space

Interface: CPU

- Xen in ring 0, OS in ring 1, everything else in ring 3
- “Fast” exception handler
- Xen handles page fault exceptions
- Double faulting

Interface: Device I/O

- Shared memory
- I/O rings
- Batching

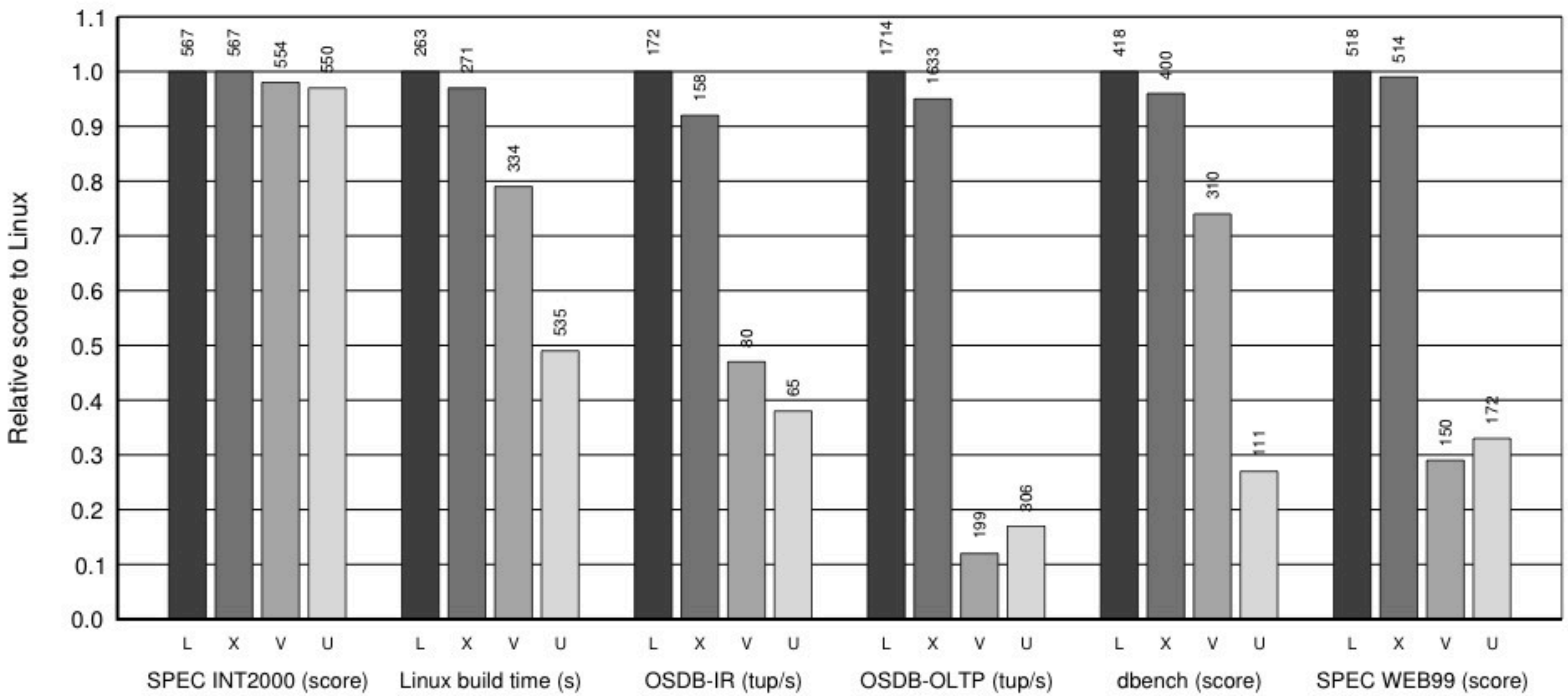


Subsystem Virtualization

- CPU Scheduling : Borrowed Virtual Time
- Real, virtual, and wall clock times
- Virtual address translation : updates through hyper call
- Physical memory : balloon driver, translation array
- Network : VFR, VIF
- Disk : VBD

Evaluation

Relative Performance



Operating System Benchmark

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

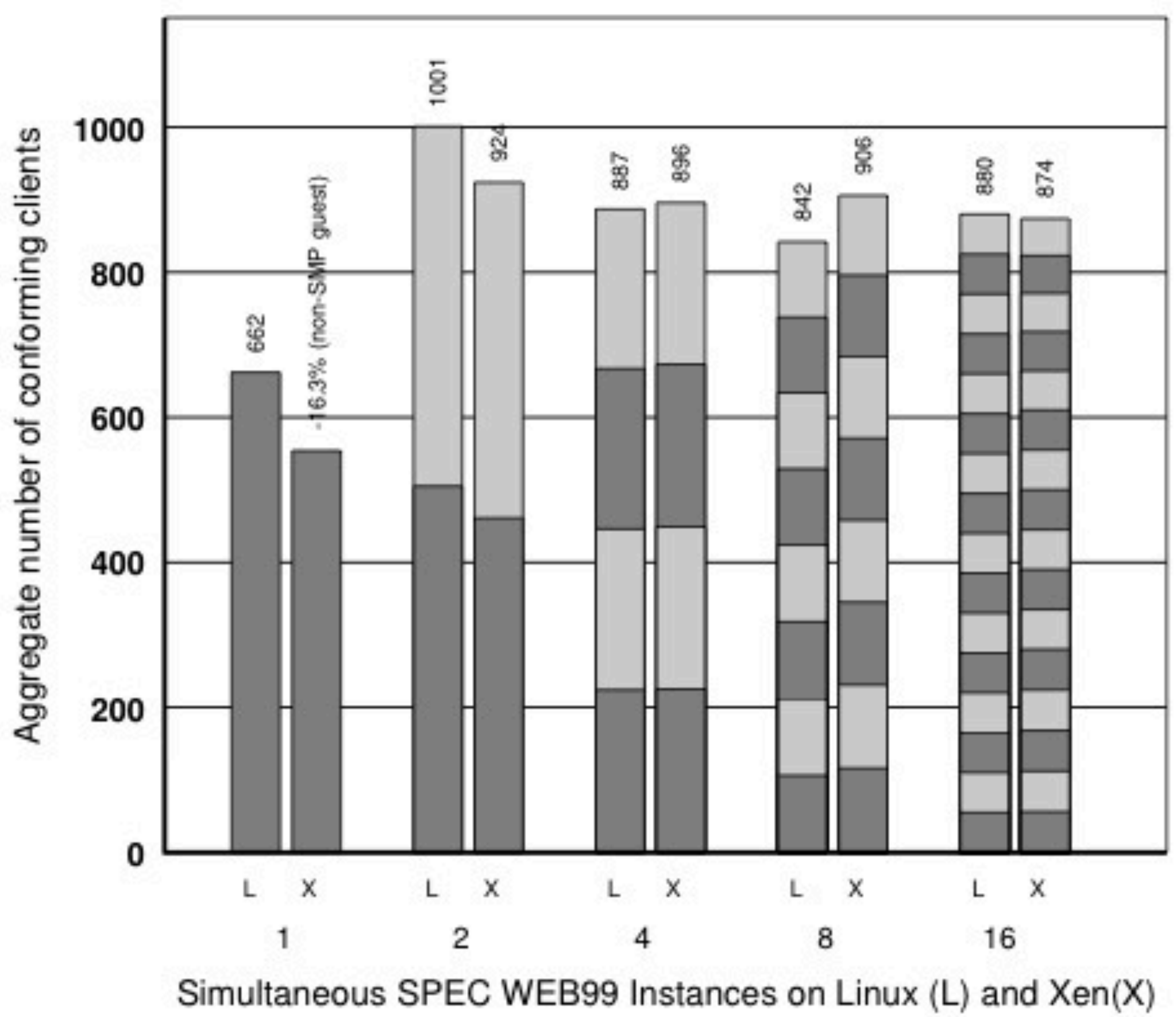
Config	2p 0K	2p 16K	2p 64K	8p 16K	8p 64K	16p 16K	16p 64K
L-SMP	1.69	1.88	2.03	2.36	26.8	4.79	38.4
L-UP	0.77	0.91	1.06	1.03	24.3	3.61	37.6
Xen	**1.97**	**2.22**	**2.67**	**3.07**	**28.7**	**7.08**	39.4
VMW	18.1	17.6	21.3	22.4	51.6	41.7	72.2
UML	15.5	14.6	14.4	16.3	36.8	23.6	52.0

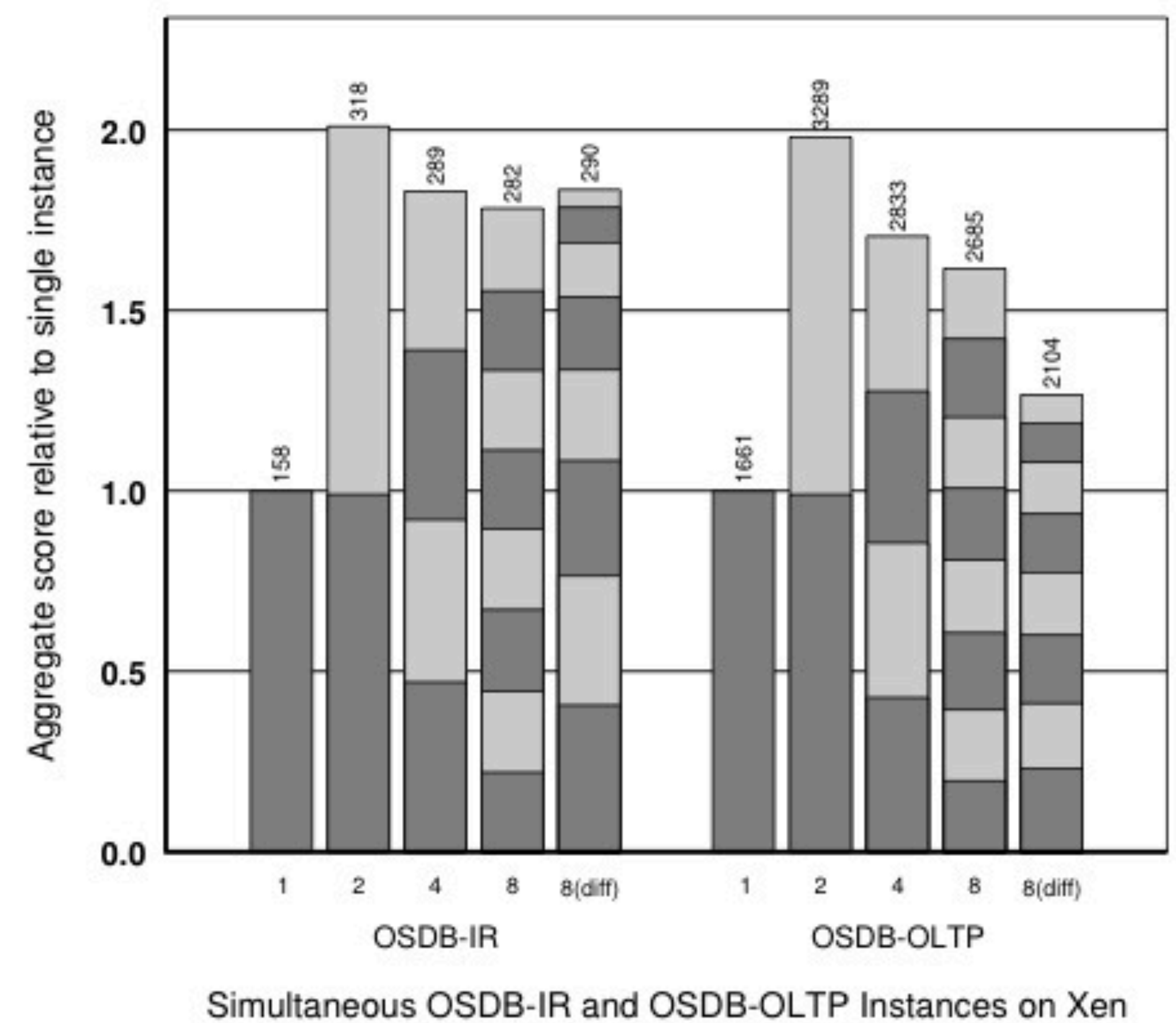
Table 4: `lmbench`: Context switching times in μs

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

Concurrent Virtual Machines

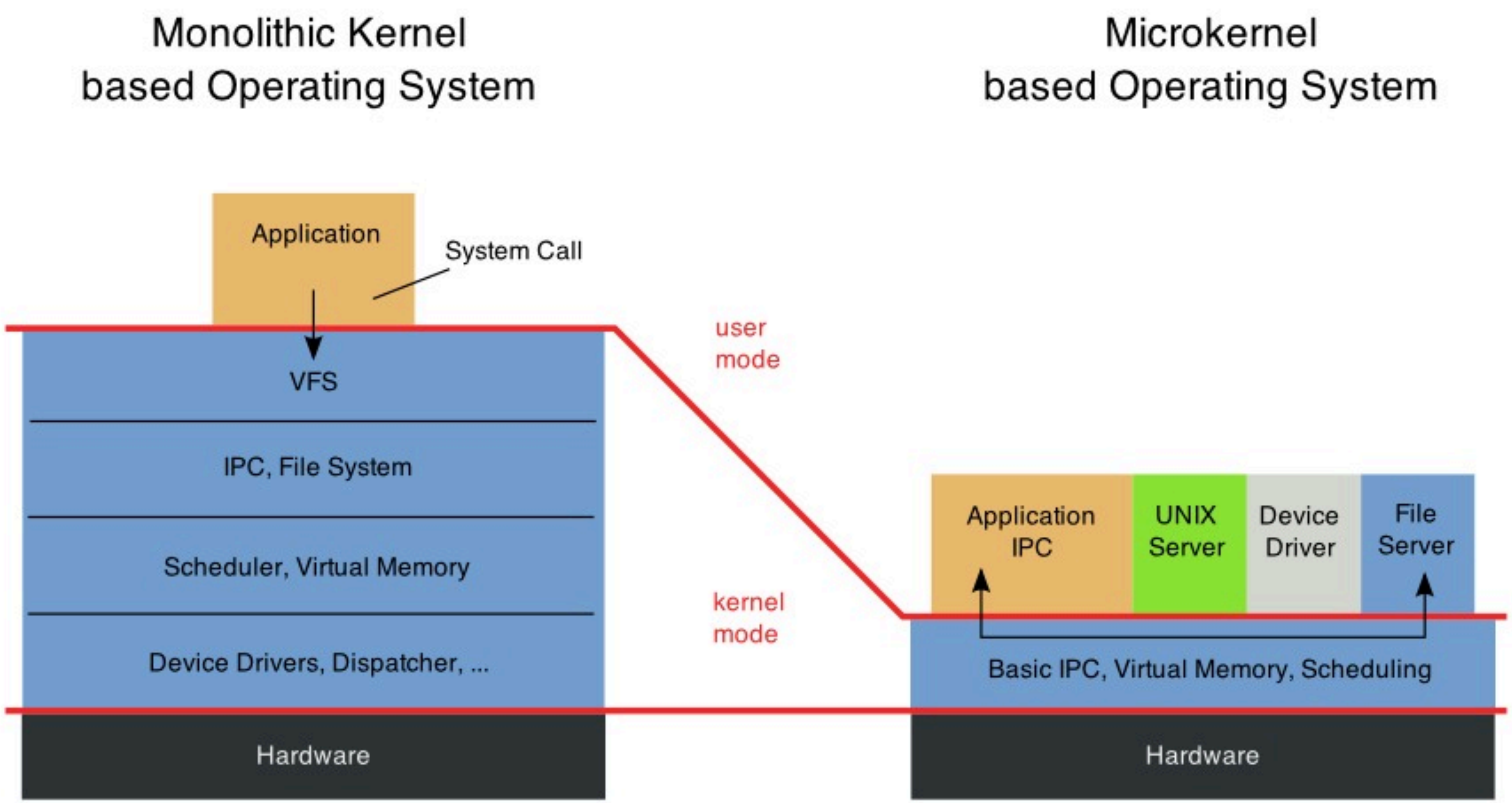




Xen

Are virtual machines μ-Kernel done right?

µ-Kernel



http://upload.wikimedia.org/wikipedia/commons/6/67/OS-structure.svg

μ-Kernel

- User-space components
- Isolation of components
- Liability inversion
- Change the interfaces for existing OSes
- IPC performance issue

VM

- Multiplexes at the level of the OS
- Isolation of VMs
- Liability inversion
- Less assumptions
- IPC irrelevant

Goals of μ-Kernel

- Extensibility by narrow interfaces
- A small code base that guarantees security
- Strong isolation to get improved manageability