OS

Practice in Operating System

Lecture 6: Exokernel

Dept. of Computer Sci.&Tech.

Questions

- Challenges
  - Poor performance of Mach and L4
- What can we do next?
  - Improvement?
  - Design new system?

Outline

- Overview
  - Concept
  - Design
- LibOS
- Xen

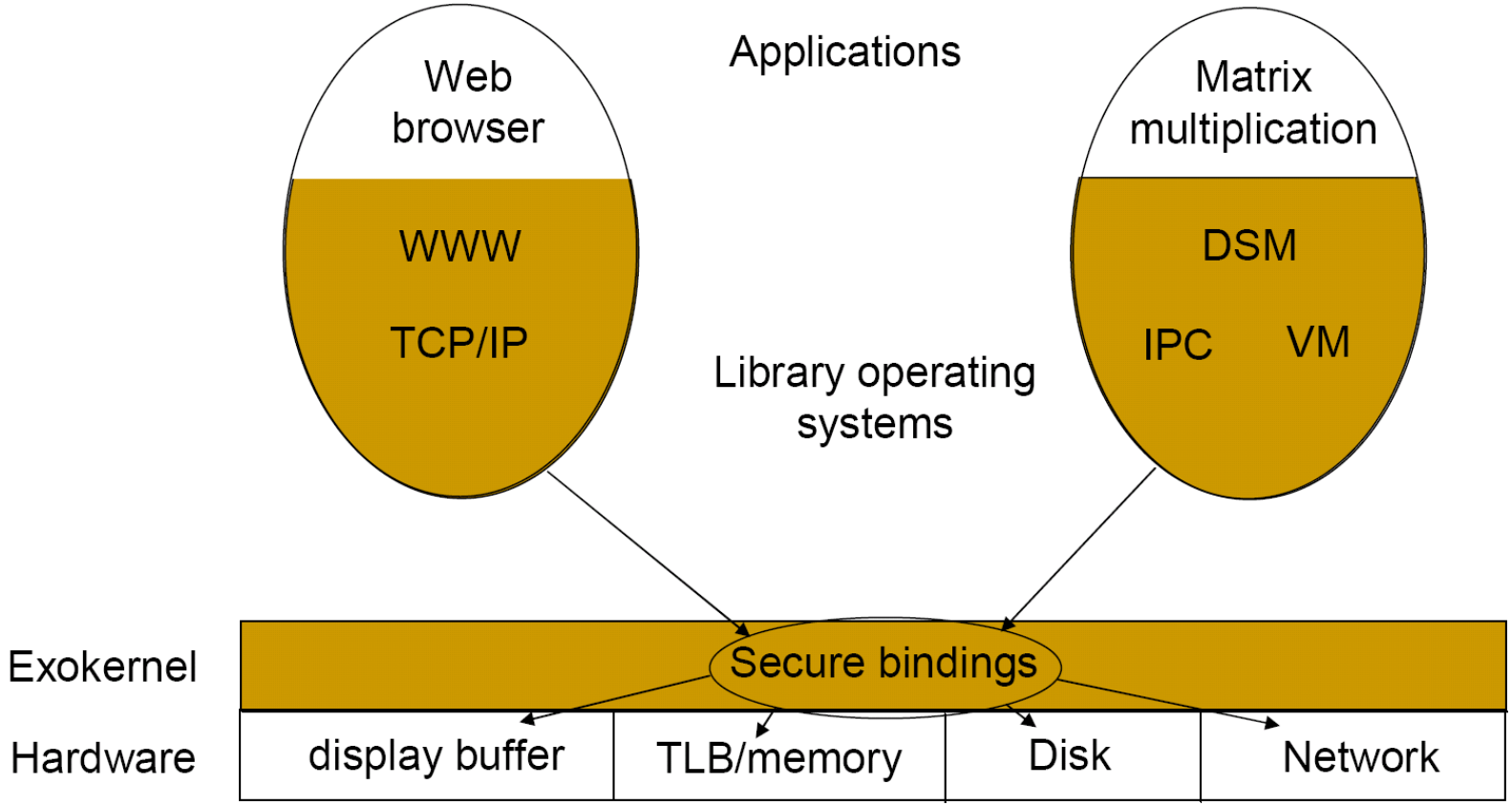
Motivation

- Traditional OS has interface, that is the centralized resource manager
  - Applications run on the virtual machine
  - Fixed high-level abstraction
  - Low performance
  - Low flexibility
  - Low functionality
- So Exokernel is designed

Exokernel

- Exokernel
  - developed at MIT in early 90s
  - Functionality of exokernels is limited to ensuring protection and multiplexing of resources
  - server functionalities are pushed to library OSes linked with individual user-level processes
- Benefits:
  - modular design
  - more reliable/secure (less code is running in kernel mode).
  - more flexibility (different user program can use different VM page replacement policies).

Architecture of Exokernel



- It includes an exokernel and untrusted (application-level) library OS
- Exokernel defines low-level interface & multiplexes (not emulate but exports) available HW resources
- Lib OS implements higher-level OS

Advantages of the architecture

- Allows extension, specialization, and replacement of abstraction : High functionality & Flexibility
- Conflict between the applications can be resolved without intervention of the kernel architects: High performance
- Easy to implement : Simplicity
- Easy to port : High portability & compatibility

Outline

- Overview
  - Concept
  - Design
- LibOS
- Xen

Exokernel Design

- Exokernel separates protection from management through a low-level interface
  - Exokernel avoids resource management
- Design principles
  - Securely expose hardware
  - Expose Allocation
  - Expose Names
  - Expose Revocation
- Resource policy decision by library OS
  - Competing is allowed as traditional OS

Exokernel Design: Secure Binding

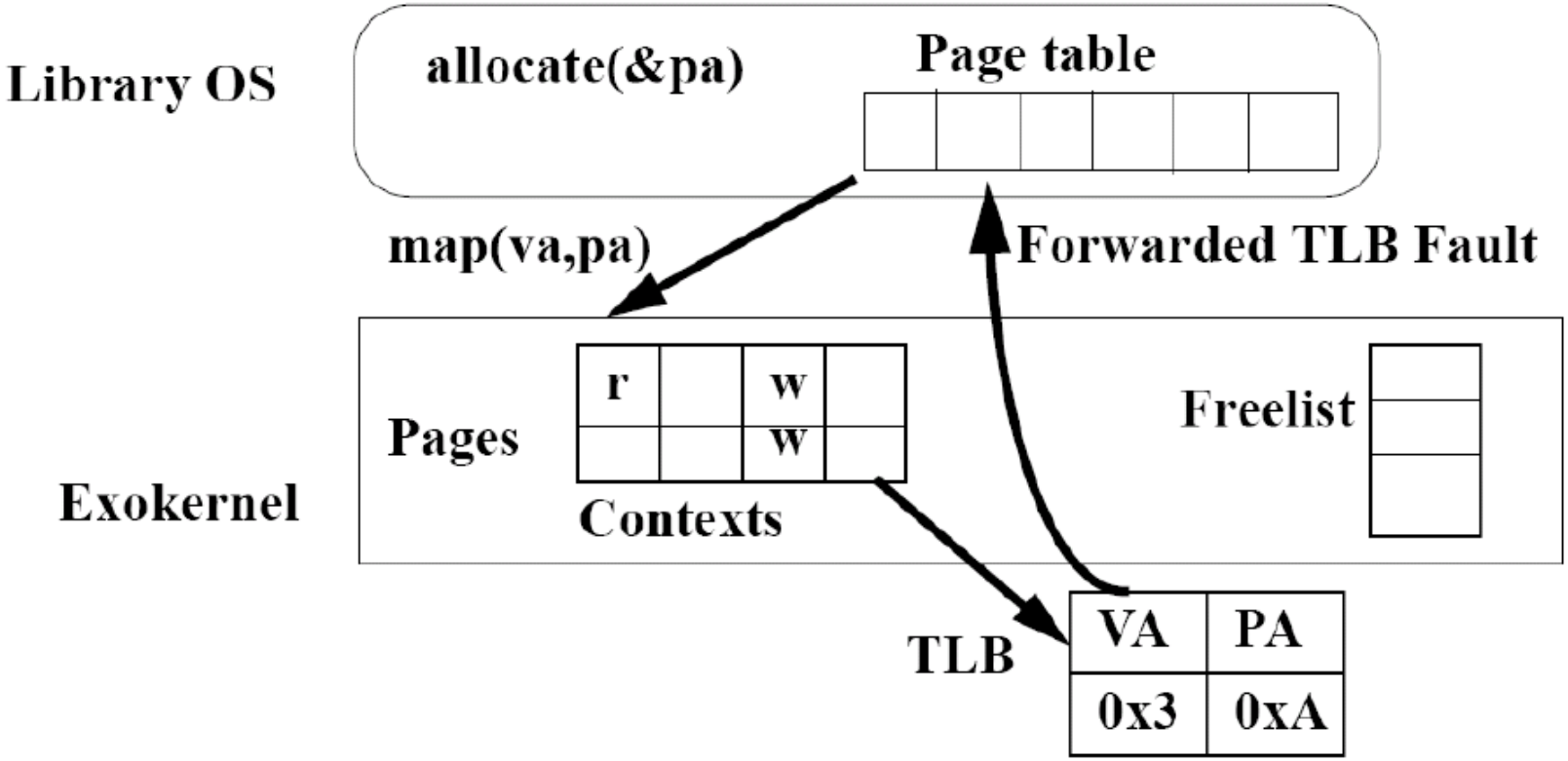
- Primary task that is protection mechanism decouples authorization
- Simple operation
- Authorization only at bind time(low management overhead)
  - No need to understand semantics at bind time
- Needs set of primitives application can use to express protection check by H/W or S/W

Exokernel Design(cont'd)

- Visible Resource Revocation
  - Higher latency but library OS can guide deallocation and have knowledge that resources are scarce
  - Frequent revoked modules can be invisible
- The Abort protocol
  - When library OS fails to response quickly in fail state, secure binding must be broken by force
  - Using repossession vector

Virtual Memory example

- When binding established authentication, etc. take place
- When binding is used, page accessed, no added overheads



Memory – Address translation

- Application AS partitioned in two regions
  - Normal app data/code
  - "guaranteed" mapping: exception code and page tables
- TLB miss steps
  - 1) Exokernel
    - Check software TLB cache (4K entries)
    - First region: dispatched to app directly
    - Second: load guaranteed mapping to TLB
  - 2) App
    - Fault
    - Construct TLB entry and capability
    - Call exokernel system call
  - 3) exokernel
    - Check capability
    - If ok, load TLB

Secure binding

- Guard everything and track ownership of everything
- Policy and mechanisms for determining access in libOS/application
  - Must have some initial bootstrapping allocation however
- Provide capability – secure binding
  - Leverage hardware support whenever possible, otherwise software
- Perform fast access checks

Multiplexing resources

- Utilize hardware, software, both
- Let applications specify exception/interrupt handlers
  - Processor context per application to include exception and interrupt handling info (+ other stuff…)
  - => visible revocation
- CPU
  - Maintain time vector; use timer interrupts to tell application of imminent context switch so "right" resources can be saved
- Memory
  - Rely on hardware address translation; add software TLB to expand set of fast VA->PA checks
- Network
  - Trickier; to parse a packet, need higher-level info to parse it;
  - Allow dynamic packet filters (DPFs) to be dynamically generated and inserted into kernel; checked for safety and sandboxed
  - ASHs – Application-specific Handlers – addition additional functionality in kernel
- Forceful revocation – abort protocol

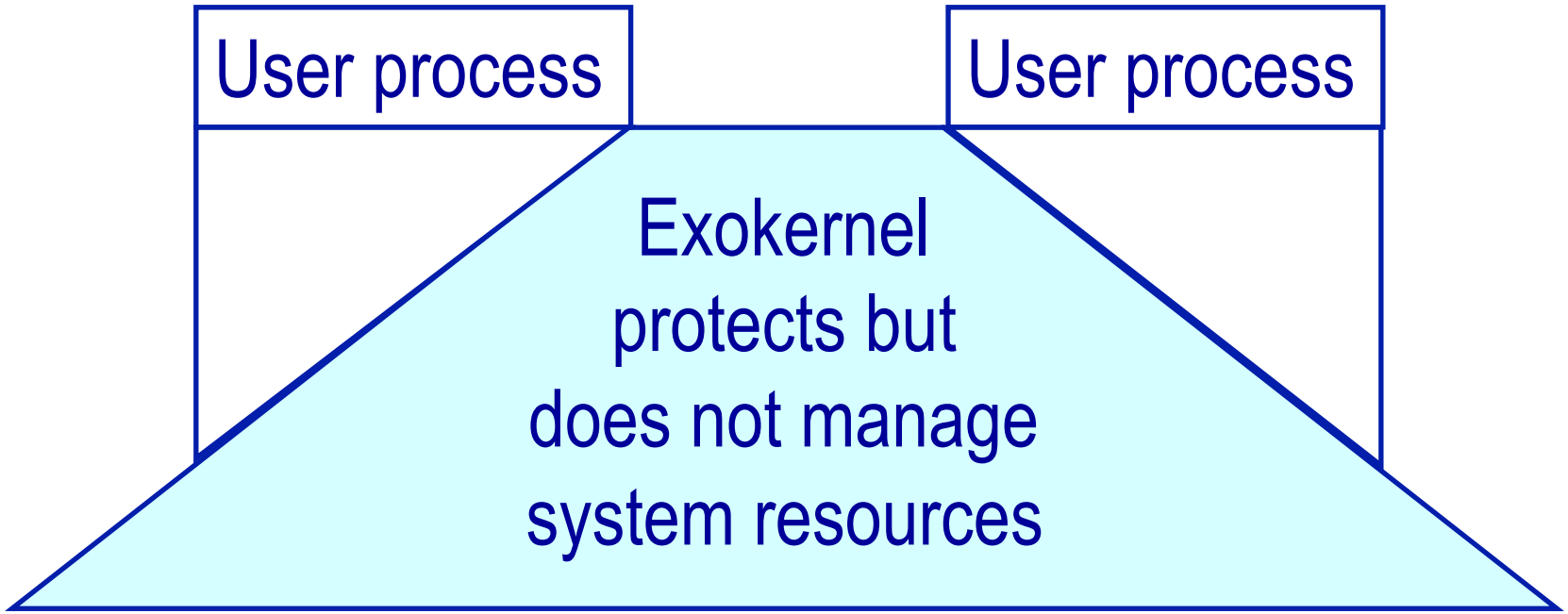
Outline

- Overview
- LibOS
  - Library OS Concept
  - Performance of LibOS
- Xen

Library OS

- The kernel does not trust the library OS
- The library OS trusts the user program; so the library OS can be implemented without the concern of protection
- The library OS and the user program are linked together
  - low cost interaction between them
  - particularly helpful when applications and the OS interact frequently (application-assisted VM page replacement)
- Applications can link with customized library OS
  - flexibility
  - e.g., one process can use LRU page replacement and another can use MRU

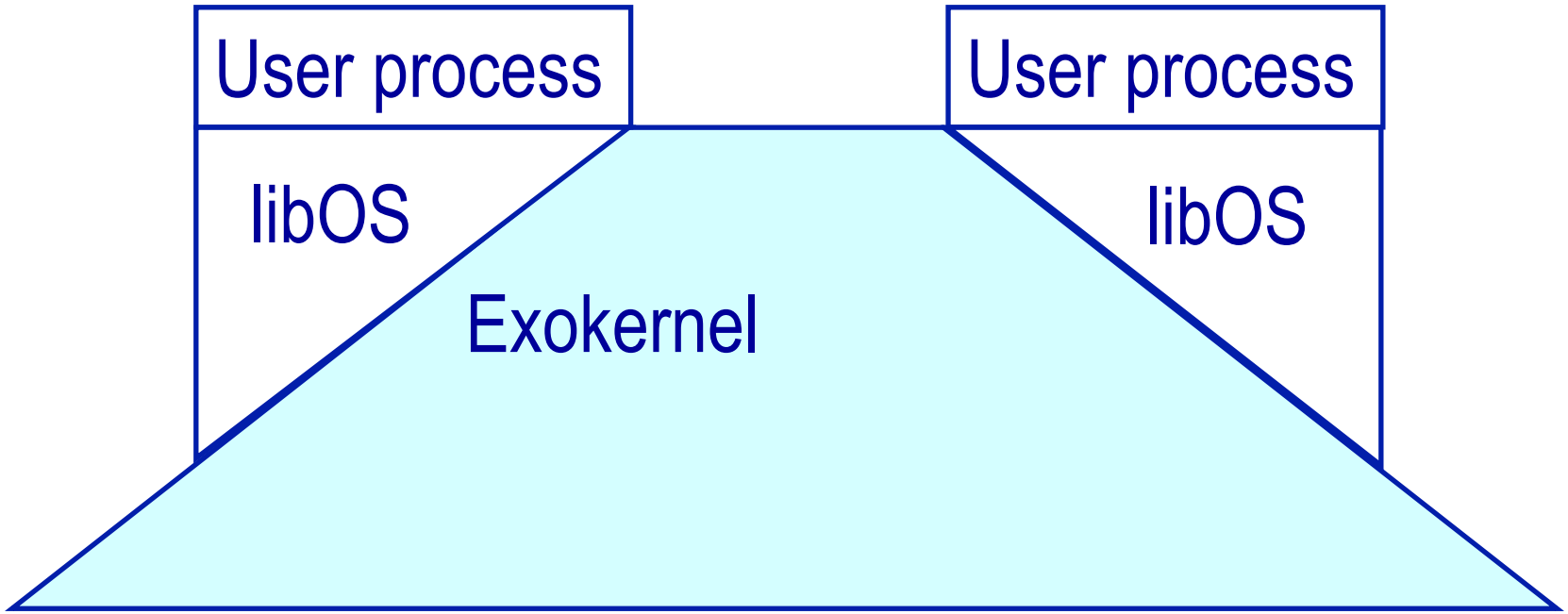
Exokernels


User process
User process
Exokernel
protects but
does not manage
system resources

LibOSes

- User-level library of functions emulating conventional system call interface
  - Manages resources for applications that do not want to do it themselves
- Can have different libOSes coexisting on the top of same exokernel
  - Allows system to emulate behaviors of several conventional operating systems

LibOSes



LibOSes

- Same interface between application and libOS as between application and a conventional kernel
- libOS runs as part of application
  - Cannot be trusted by the kernel or other user processes

Five Principles

- Separate protection and management.
- Letting applications allocate resources explicitly.
- Using physical names whenever possible.
- Expose revocation: let applications choose which instance of a resource to give up.
- Expose all kernel information.

Outline

- Overview
- LibOS
  - Library OS Concept
  - Performance of LibOS
- Xen

Results

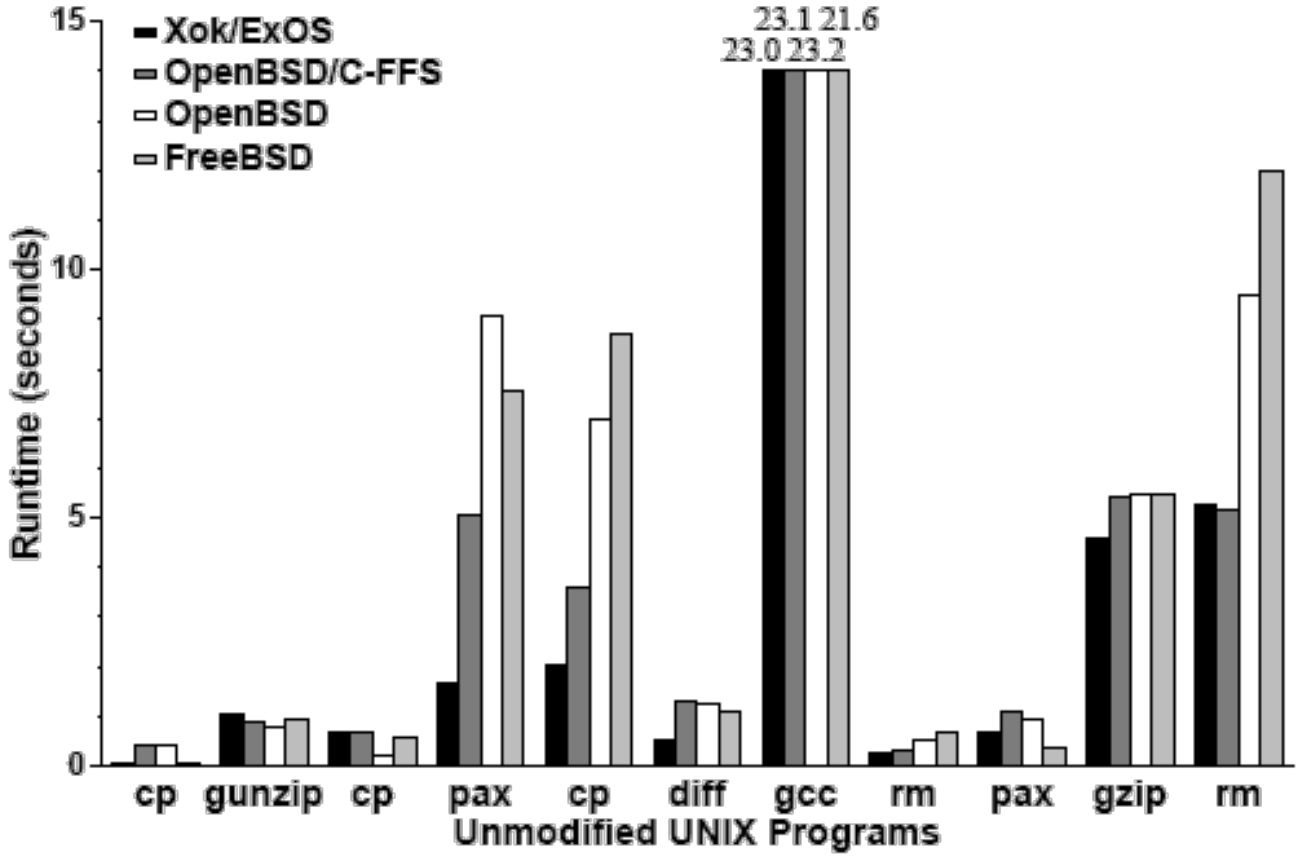


Figure 2: Performance of unmodified UNIX applications. Xok/ExOS and OpenBSD/C-FFS use a C-FFS file system while Free/OpenBSD use their native FFS file systems. Times are in seconds.

Results

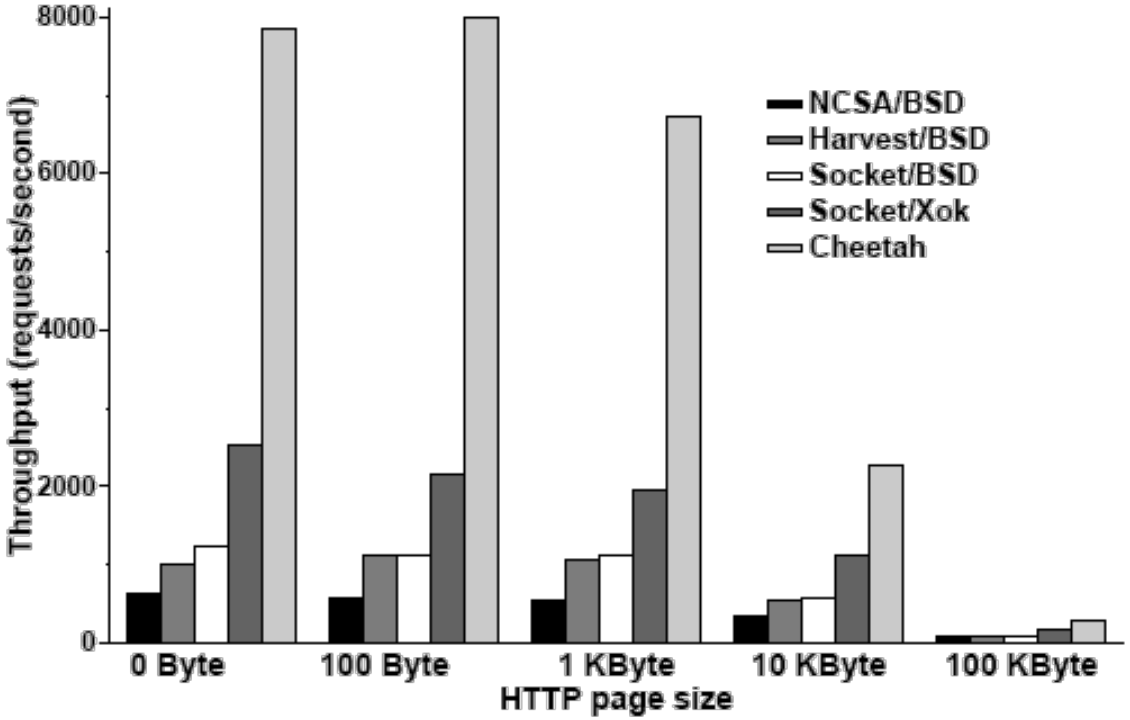


Figure 3: HTTP document throughput as a function of the document size for several HTTP/1.0 servers. **NCSA/BSD** represents the NCSA/1.4.2 server running on OpenBSD. **Harvest/BSD** represents the Harvest proxy cache running on OpenBSD. **Socket/BSD** represents our HTTP server using TCP sockets on OpenBSD. **Socket/Xok** represents our HTTP server using the TCP socket interface built on our extensible TCP/IP implementation on the Xok exokernel. **Cheetah/Xok** represents the Cheetah HTTP server, which exploits the TCP and file system implementations for speed.

Advantages and Disadvantages

- Benefits
  - Performance
  - Isolation and Stability
  - Flexibility (In terms of abstraction level)
- Drawbacks
  - Space Requirement
  - Development Resources (General case)
  - Complexity (General case)
  - Flexibility (In terms of program mobility)

Outline

- Overview
- LibOS
- Xen
  - Xen Architecture
  - Virtualization of MMU
  - SMP support
  - Xen I/O Architecture

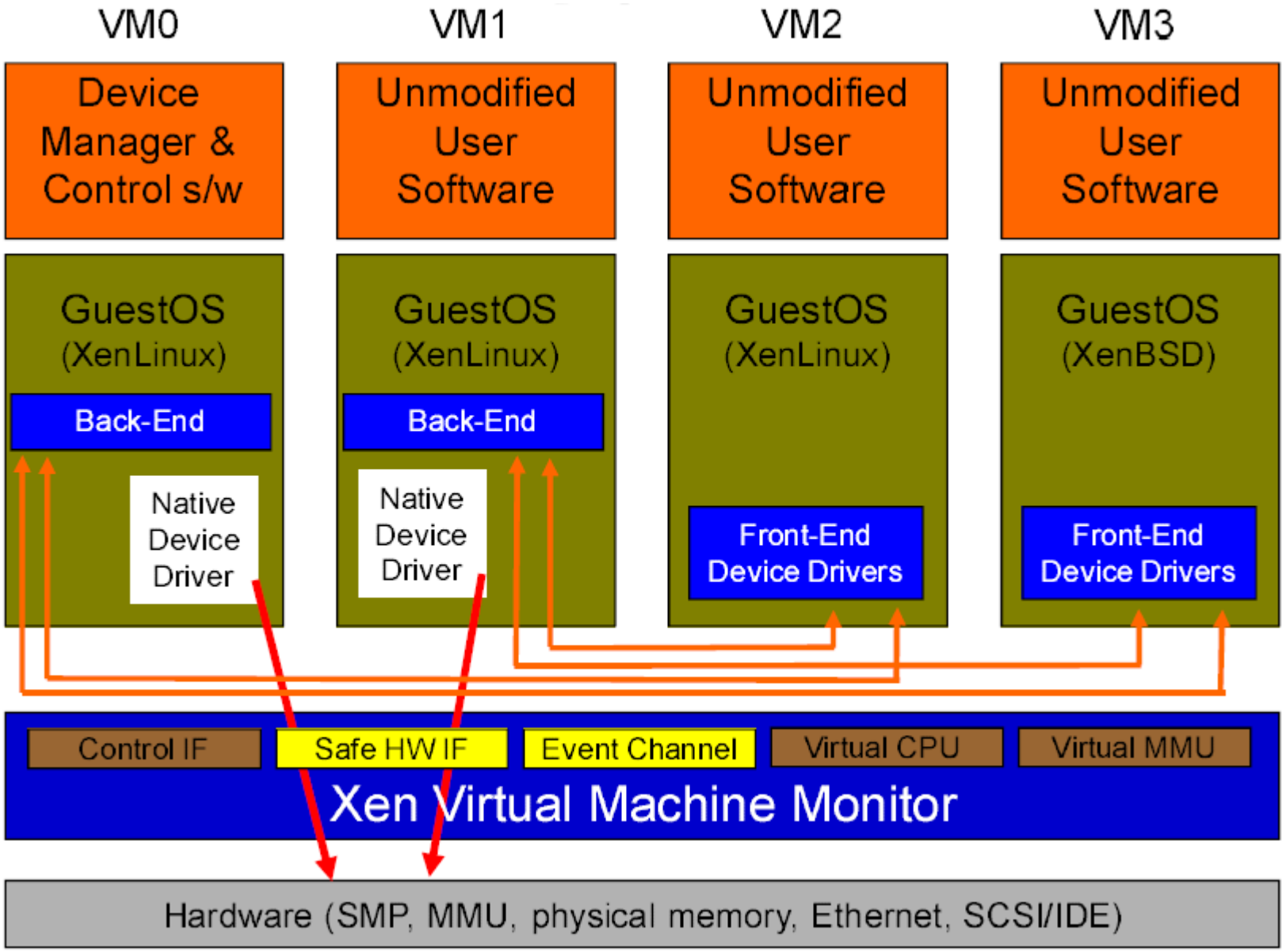
Xen Today :

- Secure isolation between VMs
- Resource control and QoS
- Only guest kernel needs to be ported
  - User-level apps and libraries run unmodified
  - Linux 2.4/2.6, NetBSD, FreeBSD, Plan9, Solaris
- Execution performance close to native
- Broad x86 hardware support
- Live Relocation of VMs between Xen nodes

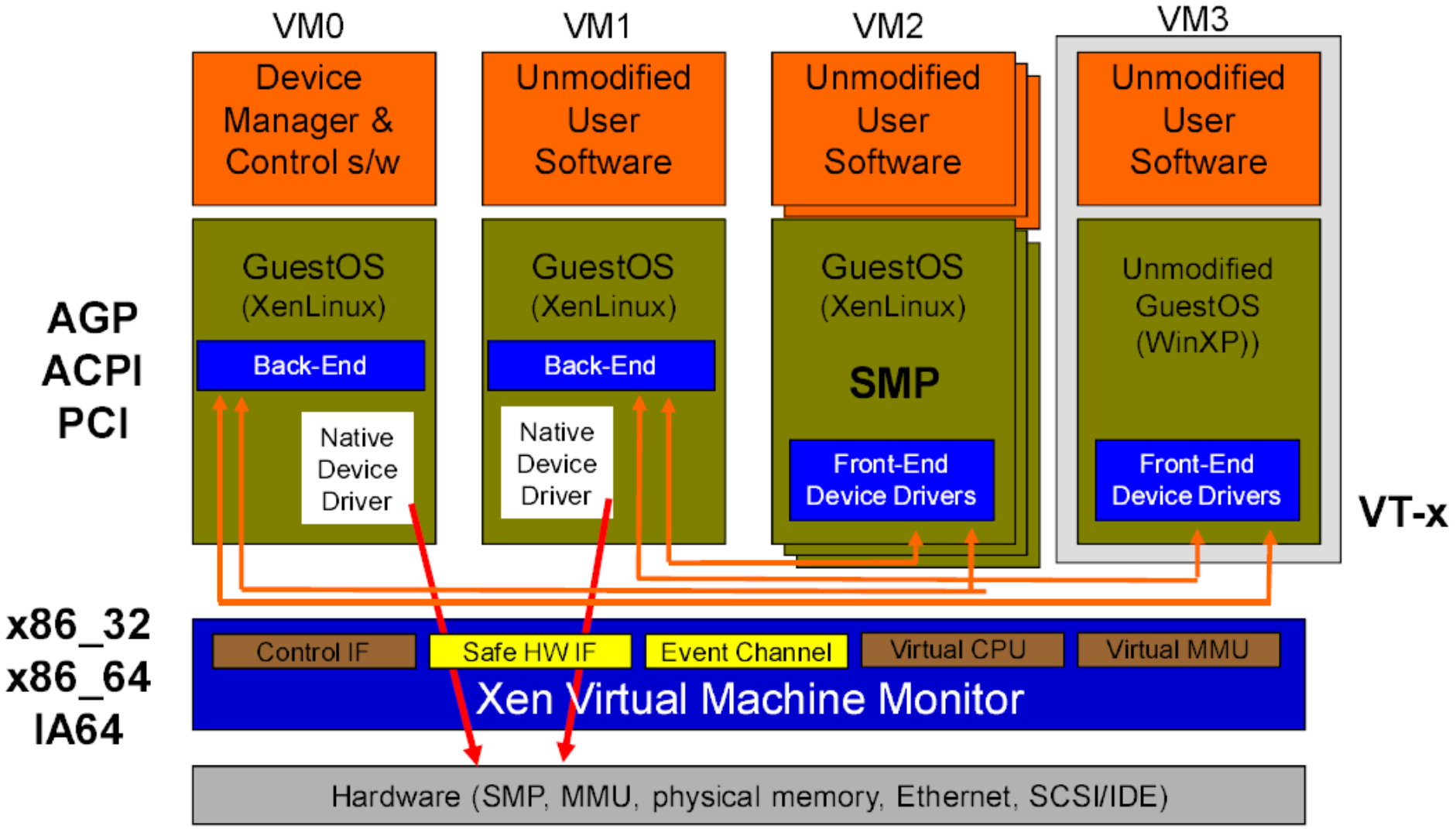
Para-Virtualization in Xen

- Xen extensions to x86 arch
  - Like x86, but Xen invoked for privileged ops
  - Avoids binary rewriting
  - Minimize number of privilege transitions into Xen
  - Modifications relatively simple and self-contained
- Modify kernel to understand virtualised env.
  - Wall-clock time vs. virtual processor time
    - Desire both types of alarm timer
  - Expose real resource availability
    - Enables OS to optimise its own behaviour

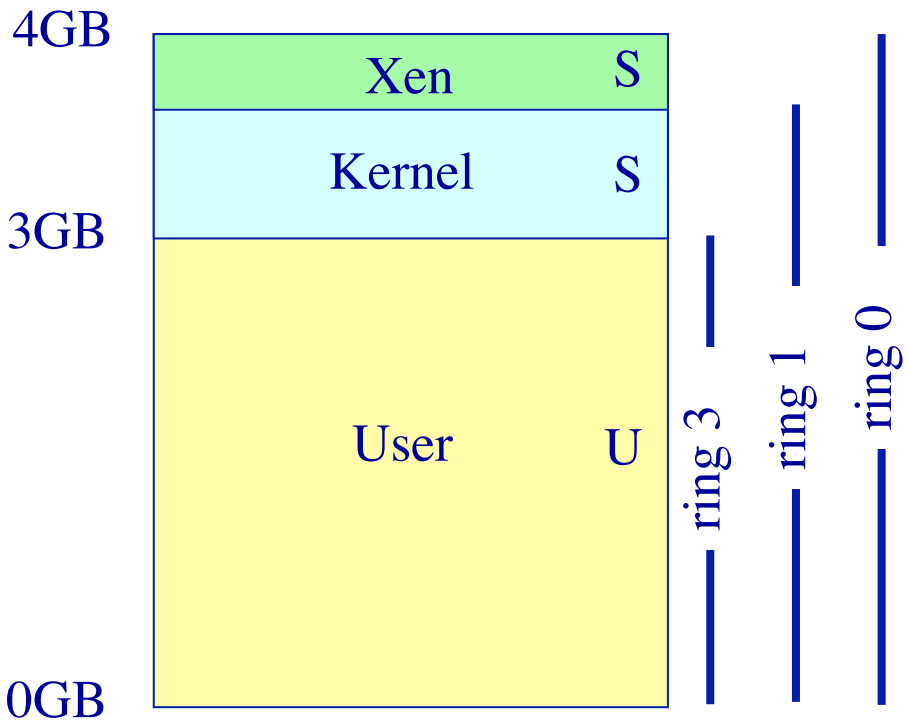
Xen 2.0 Architecture



Xen 3.0 Architecture

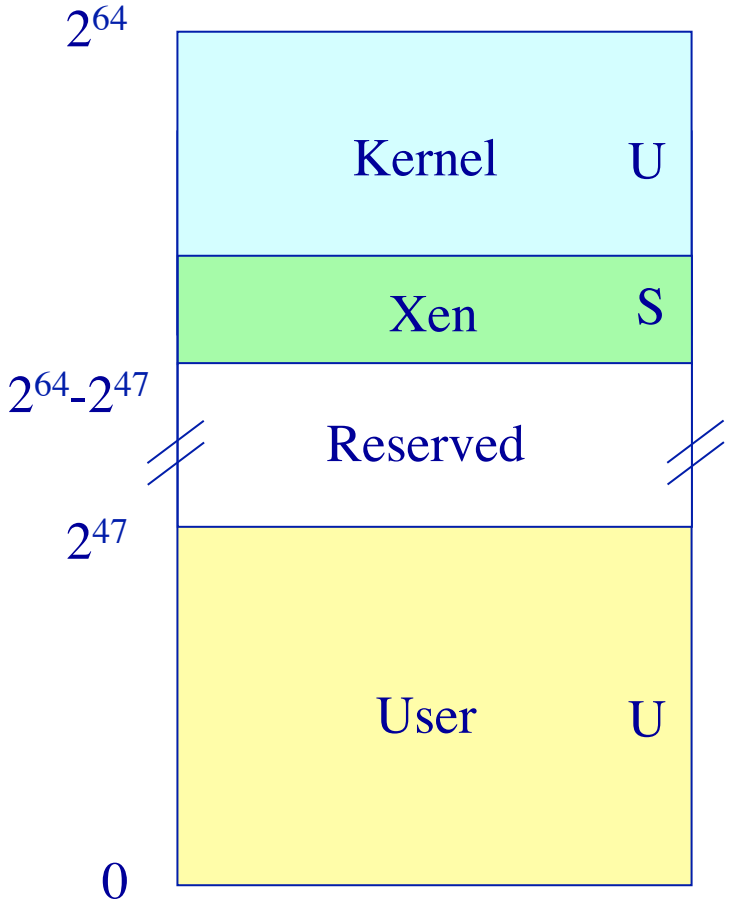


x86_32



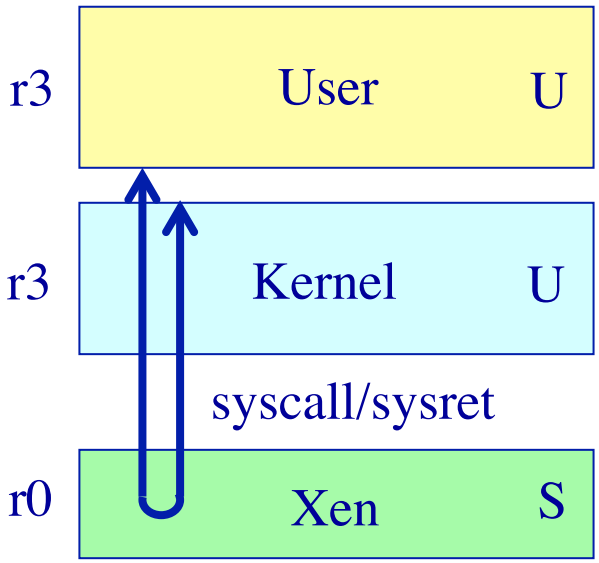
- Xen reserves top of VA space
- Segmentation protects Xen from kernel
- System call speed unchanged
- Xen 3 now supports PAE for >4GB mem

x86_64



- Large VA space makes life a lot easier, but:
- No segment limit support
- ➔ Need to use page-level protection to protect hypervisor

x86_64



- Run user-space and kernel in ring 3 using different pagetables
  - Two PGD's (PML4's): one with user entries; one with user plus kernel entries
- System calls require an additional syscall/ret via Xen
- Per-CPU trampoline to avoid needing GS in Xen

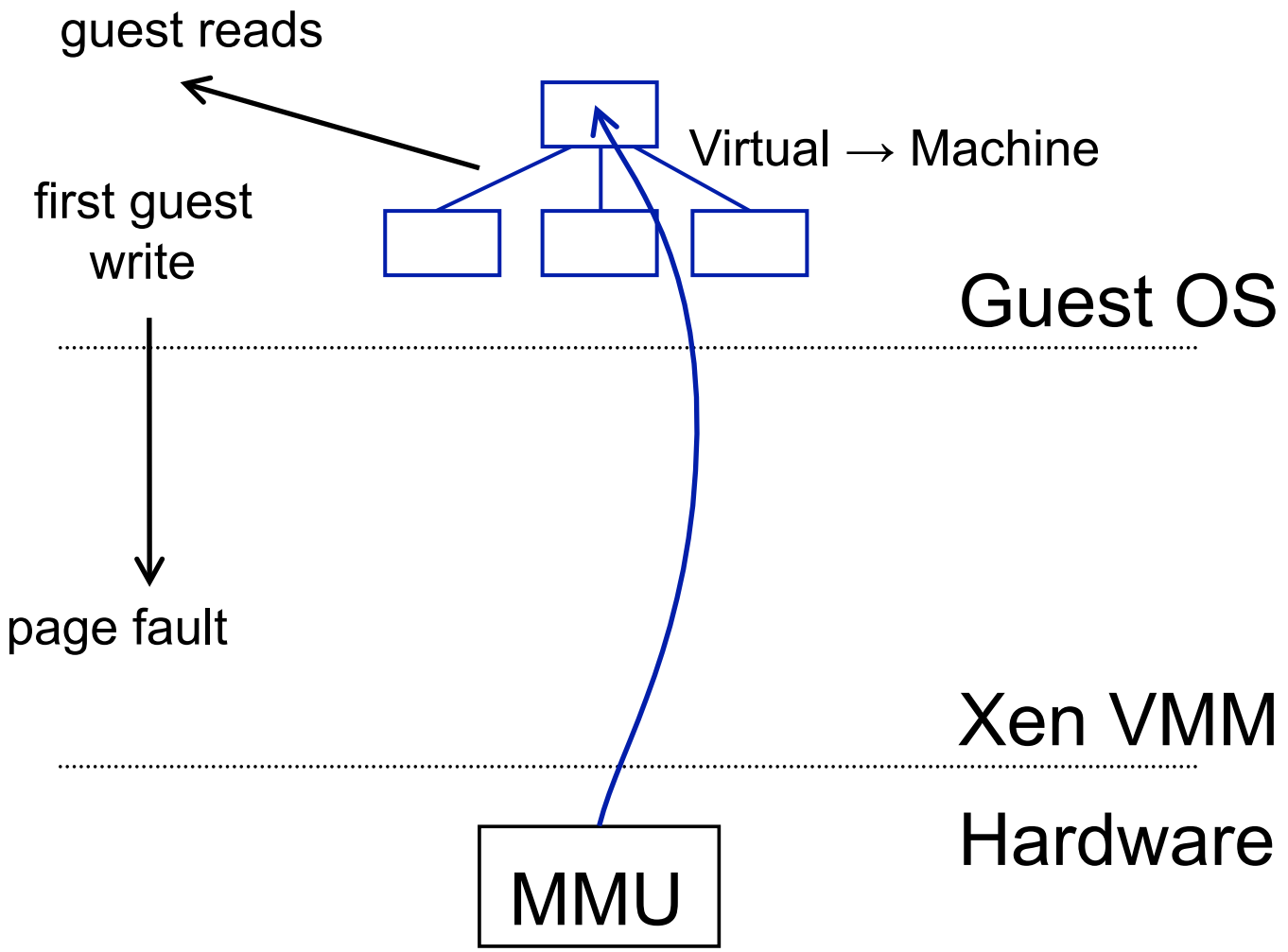
Outline

- Overview
- LibOS
- Xen
  - Xen Architecture
  - Virtualization of MMU
  - SMP support
  - Xen I/O Architecture

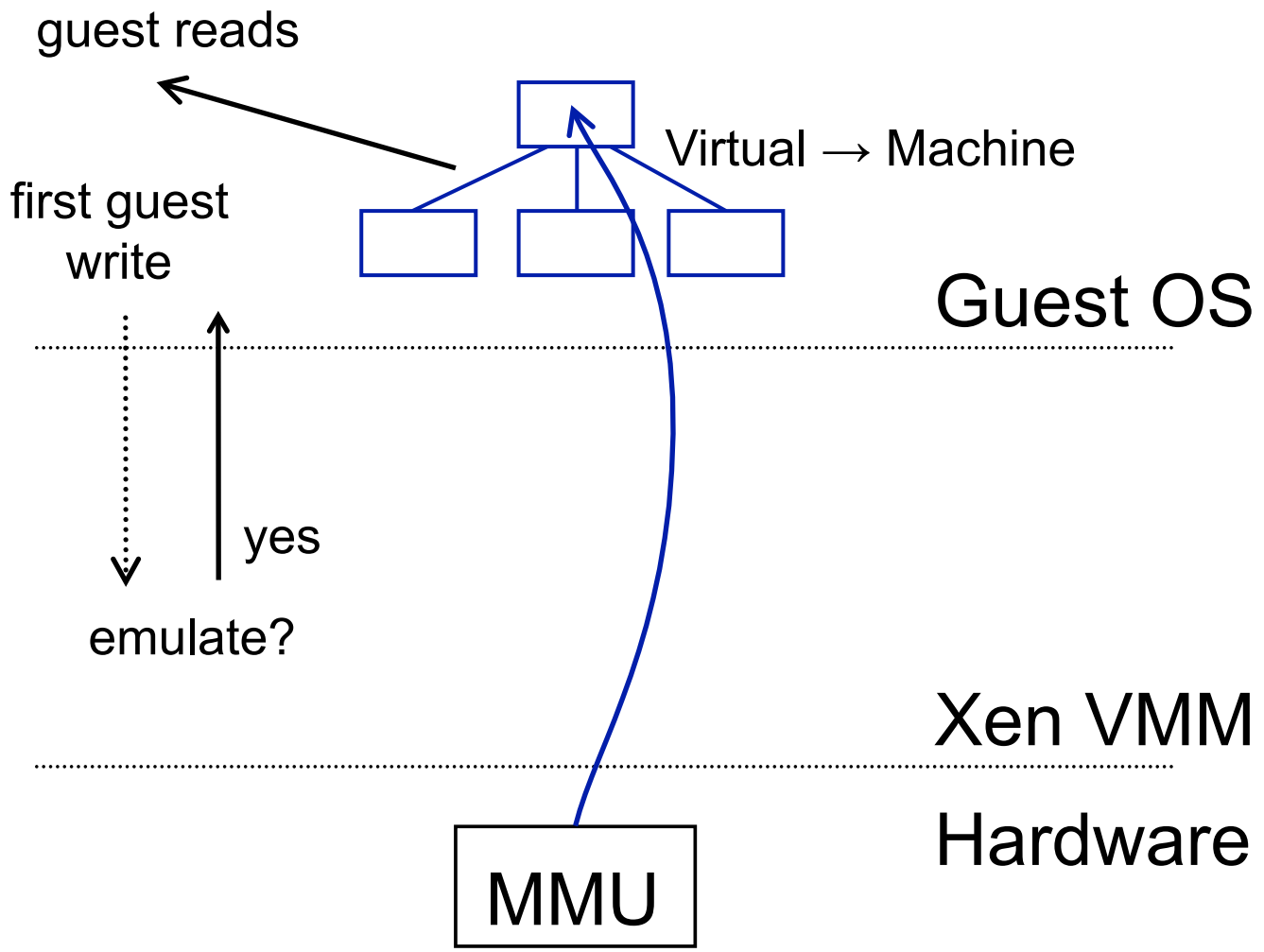
Para-Virtualizing the MMU

- Guest OSes allocate and manage own PTs
  - Hypercall to change PT base
- Xen must validate PT updates before use
  - Allows incremental updates, avoids revalidation
- Validation rules applied to each PTE:
  1. Guest may only map pages it owns*
  2. Pagetable pages may only be mapped RO
- Xen traps PTE updates and emulates, or 'unhooks' PTE page for bulk updates

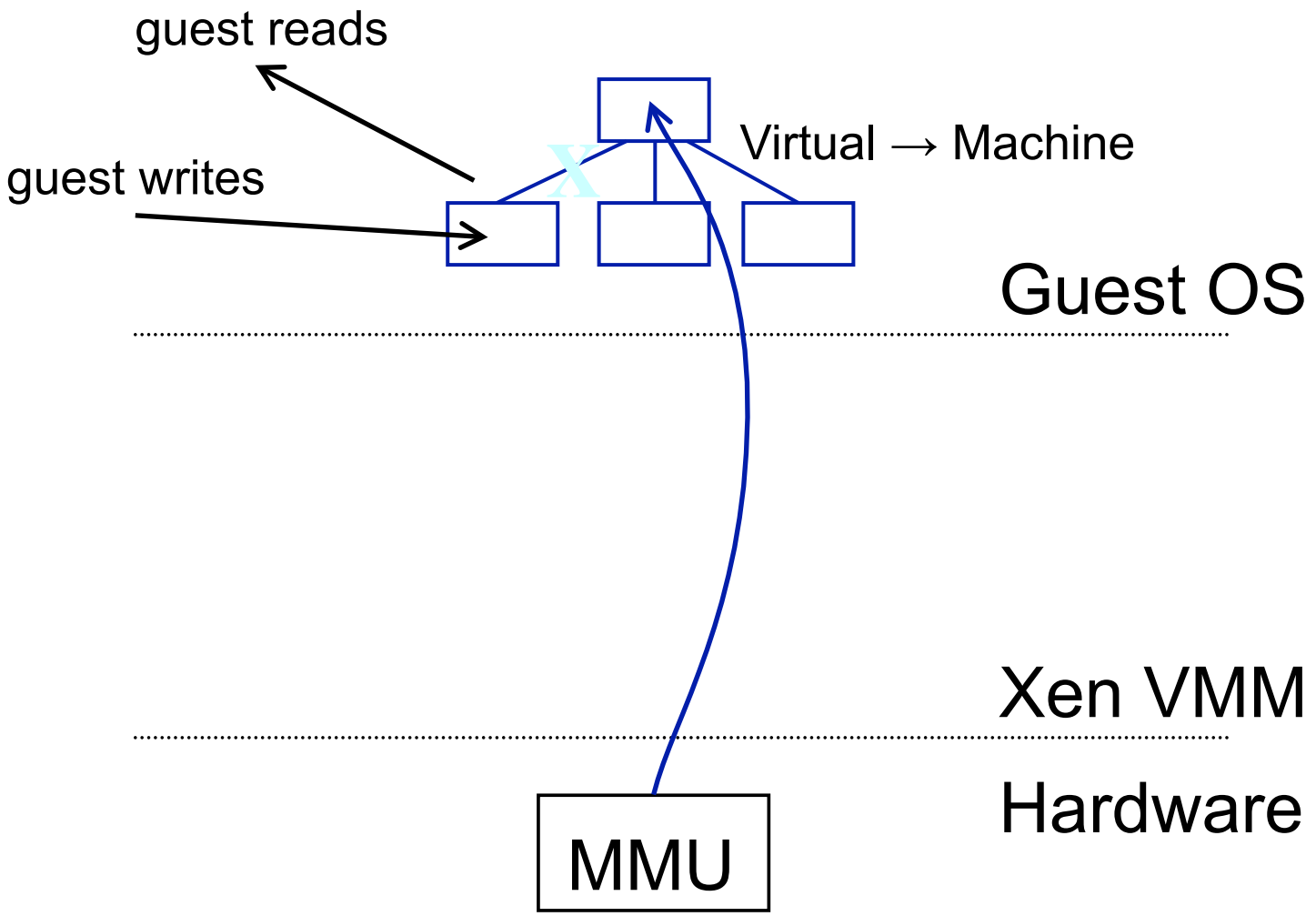
Writeable Page Tables : 1 – Write fault



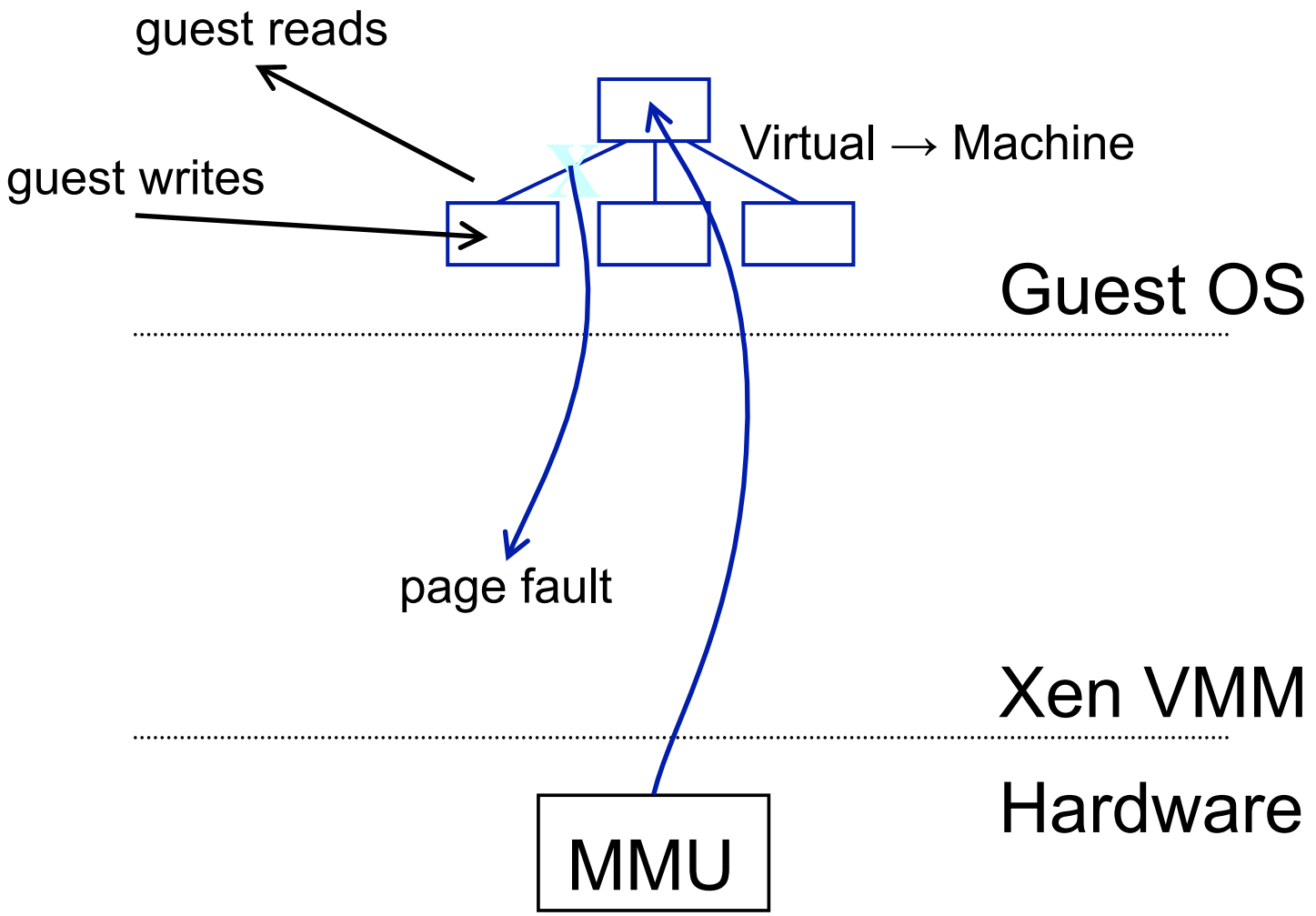
Writeable Page Tables : 2 – Emulate?



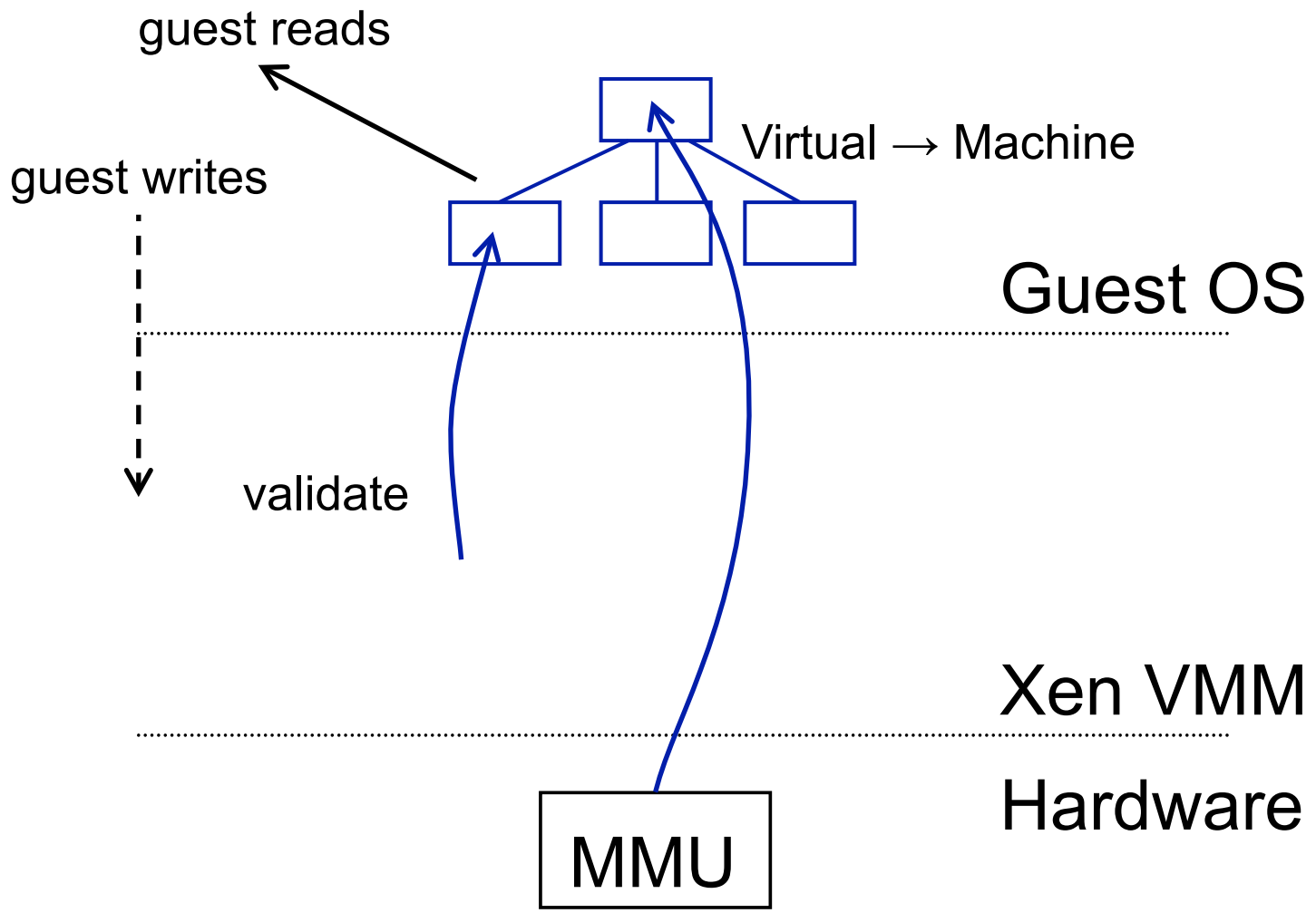
Writeable Page Tables : 3 - Unhook



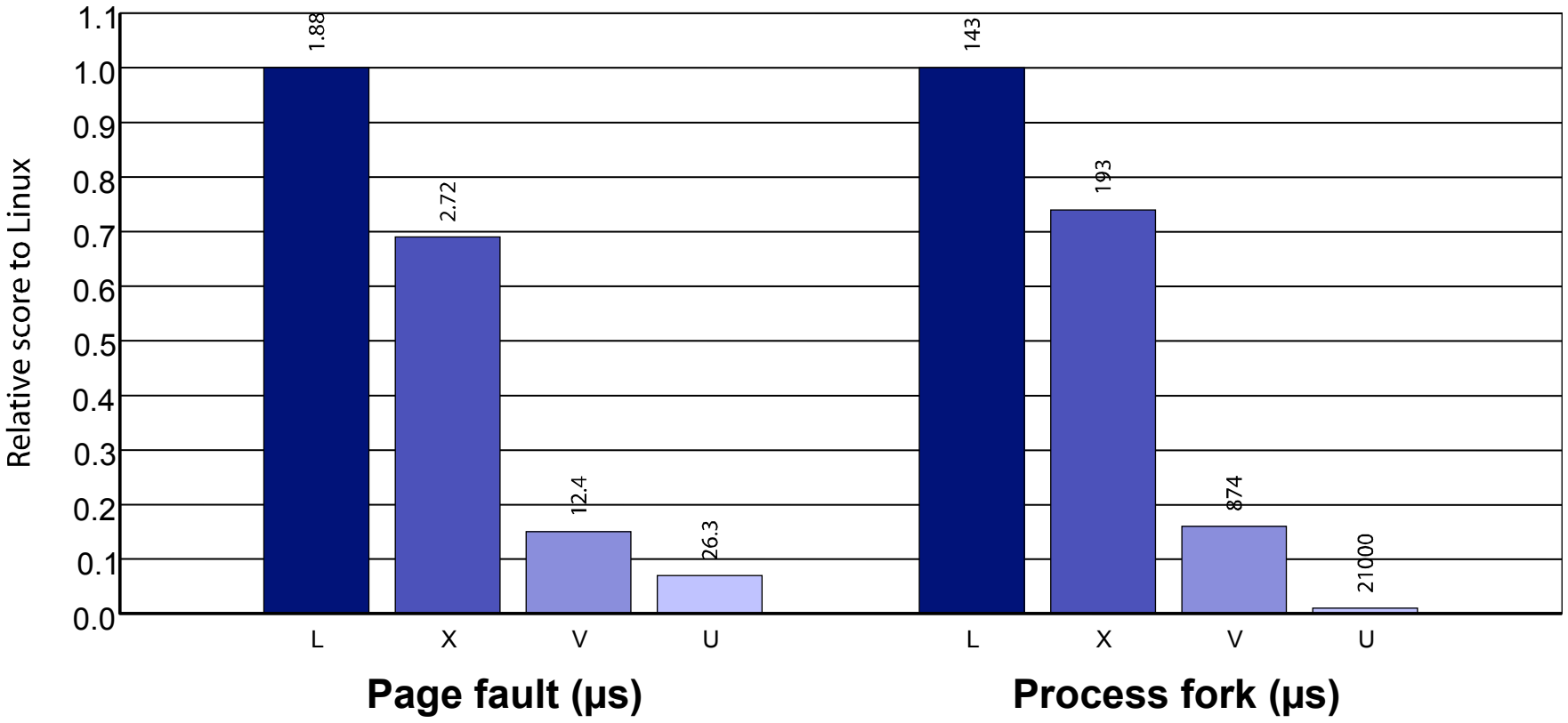
Writeable Page Tables : 4 - First Use



Writeable Page Tables : 5 – Re-hook



MMU Micro-Benchmarks



Imbench results on Linux (L), Xen (X), VMWare Workstation (V), and UML (U)

Outline

- Overview
- LibOS
- Xen
  - Xen Architecture
  - Virtualization of MMU
  - SMP support
  - Xen I/O Architecture

SMP Guest Kernels

- Xen extended to support multiple VCPUs
  - Virtual IPI's sent via Xen event channels
  - Currently up to 32 VCPUs supported
- Simple hotplug/unplug of VCPUs
  - From within VM or via control tools
  - Optimize one active VCPU case by binary patching spinlocks

SMP Guest Kernels

- Takes great care to get good SMP performance while remaining secure
  - Requires extra TLB syncronization IPIs
- Paravirtualized approach enables several important benefits
  - Avoids many virtual IPIs
  - Allows 'bad preemption' avoidance
  - Auto hot plug/unplug of CPUs
- SMP scheduling is a tricky problem
  - Strict gang scheduling leads to wasted cycles

Outline

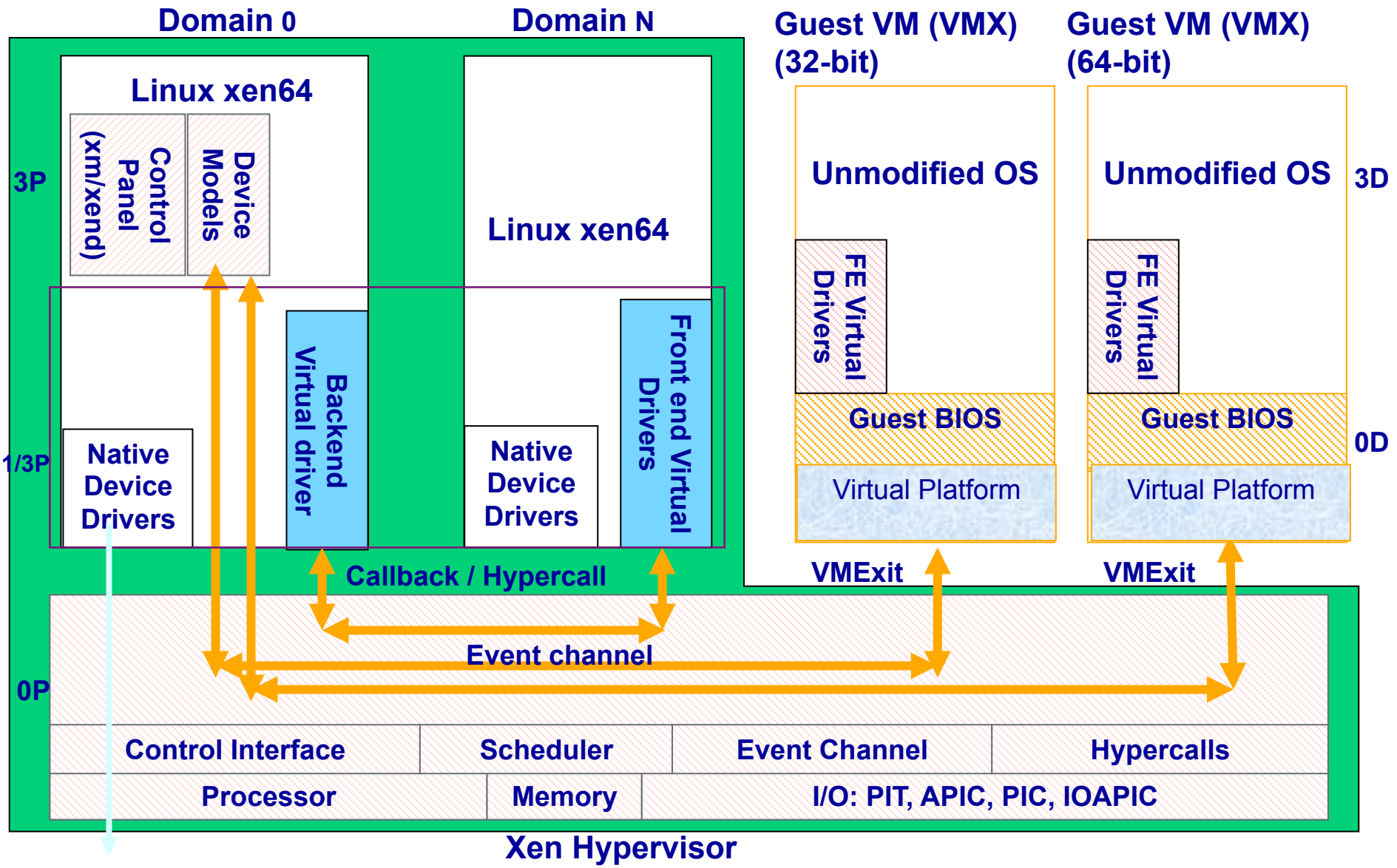
- Overview
- LibOS
- Xen
  - Xen Architecture
  - Virtualization of MMU
  - SMP support
  - Xen I/O Architecture

I/O Architecture

- Xen *IO-Spaces* delegate guest OSes protected access to specified h/w devices
  - Virtual PCI configuration space
  - Virtual interrupts
  - (Need IOMMU for full DMA protection)
- Devices are virtualised and exported to other VMs via *Device Channels*
  - Safe asynchronous shared memory transport
  - 'Backend' drivers export to 'frontend' drivers
  - Net: use normal bridging, routing, iptables
  - Block: export any blk dev e.g. sda4,loop0,vg3
- (Infiniband / Smart NICs for direct guest IO)

VT-x / (Pacifica)

- Enable Guest OSes to be run without para-virtualization modifications
  - E.g. legacy Linux, Windows XP/2003
- CPU provides traps for certain privileged instrs
- Shadow page tables used to provide MMU virtualization
- Xen provides simple platform emulation
  - BIOS, Ethernet (ne2k), IDE emulation
- (Install paravirtualized drivers after booting for high-performance IO)


Domain 0
Domain N
Guest VM (VMX) (32-bit)
Guest VM (VMX) (64-bit)
Linux xen64
Control Panel (xm/xend)
Device Models
Linux xen64
Unmodified OS
Unmodified OS
3P
3D
FE Virtual Drivers
FE Virtual Drivers
Guest BIOS
Guest BIOS
0D
1/3P
Native Device Drivers
Backend Virtual driver
Native Device Drivers
Front end Virtual Drivers
Virtual Platform
Virtual Platform
VMExit
VMExit
Callback / Hypercall
Event channel
0P
Control Interface
Scheduler
Event Channel
Hypercalls
Processor
Memory
I/O: PIT, APIC, PIC, IOAPIC
Xen Hypervisor

Reference

- http://www.cc.gatech.edu/classes/AY2009/cs6210_fall/lectures/02-Exokernel.pdf
- http://web.cecs.pdx.edu/~walpole/class/cs533/spring2005/slides/161.ppt
- http://dcslab.snu.ac.kr/research/sig-net/presentation/0427_shahn.ppt
- http://os.korea.ac.kr/course_papers/2009_AOS/2009_nos_slide_03.pdf
- https://agora.cs.illinois.edu/download/attachments/7340110/Exokernel+presentation.ppt?version=1&modificationDate=1204401253000
- http://www.cs.ucf.edu/courses/cop5611/spring05/item/exokernel.pdf
- http://www2.cs.uh.edu/~paris/6360/PowerPoint/Xok.ppt
- http://www.cl.cam.ac.uk/research/srg/netos/papers/2005-xen-ols.ppt