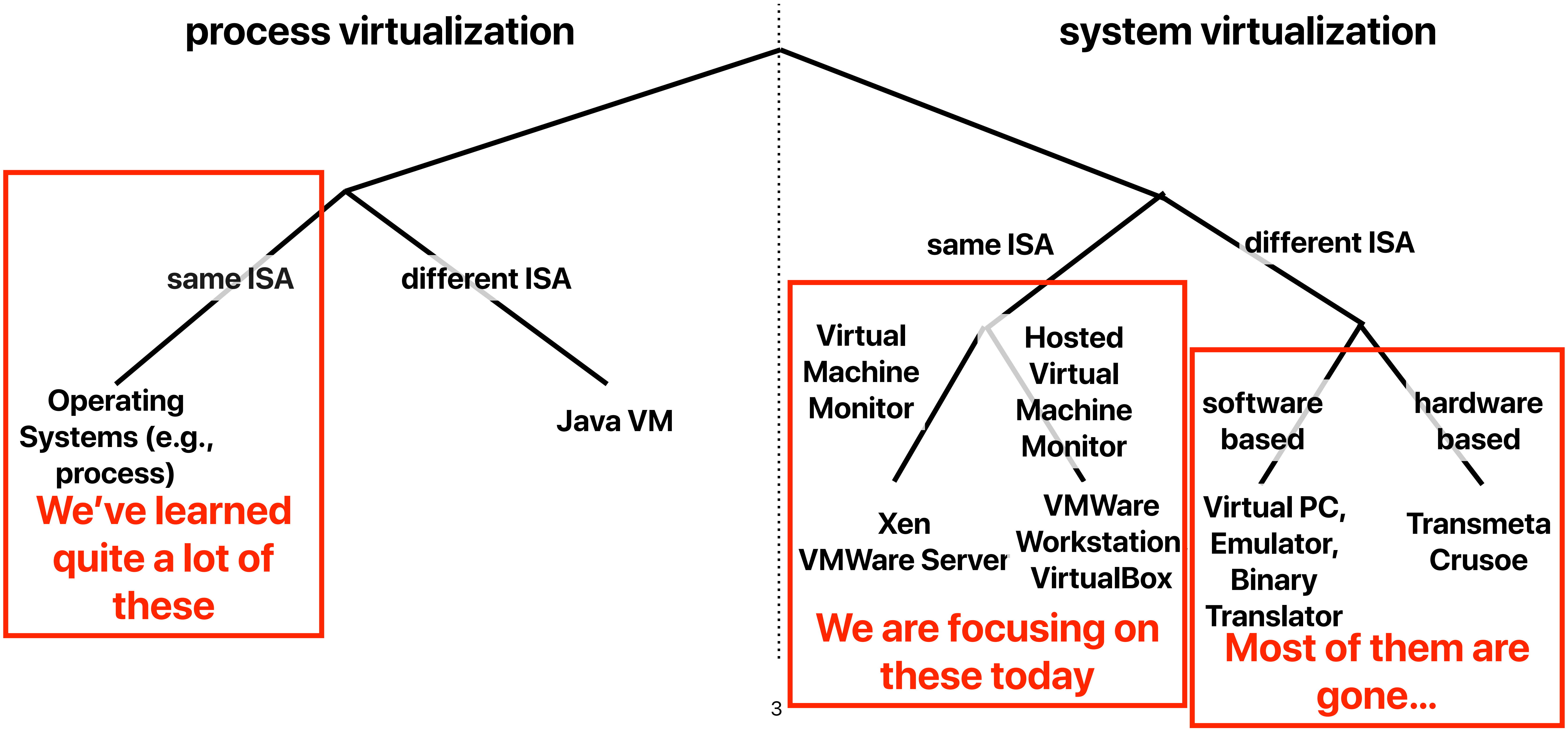
Virtual Machines & Reflections

Hung-Wei Tseng

Virtual Machines

Taxonomy of virtualization



Virtual machine architecture

Applications

Guest OS

Virtual Machine Monitor

The Machine

Three Laws of Robotics

- A robot may not injure a human being or, through inaction, allow a human being to come to harm.
- A robot must obey orders given it by human beings except where such orders would conflict with the First Law.
- A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

https://s3-ap-southeast-1.amazonaws.com/cloud-skcript/wp-content/uploads/2014/05/25090337/robots.jpg

Back to 1974...

Formal Requirements for Virtualizable Third Generation Architectures

Gerald J. Popek
University of California, Los Angeles
and
Robert P. Goldberg
Honeywell Information Systems and
Harvard University

A virtual machine is taken to be an *efficient, isolated duplicate* of the real machine. We explain these notions through the idea of a *virtual machine monitor* (VMM). See Figure 1. As a piece of software a VMM has three essential characteristics. First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.

Fidelity

Performance

Safety and isolation

Recap: virtualization

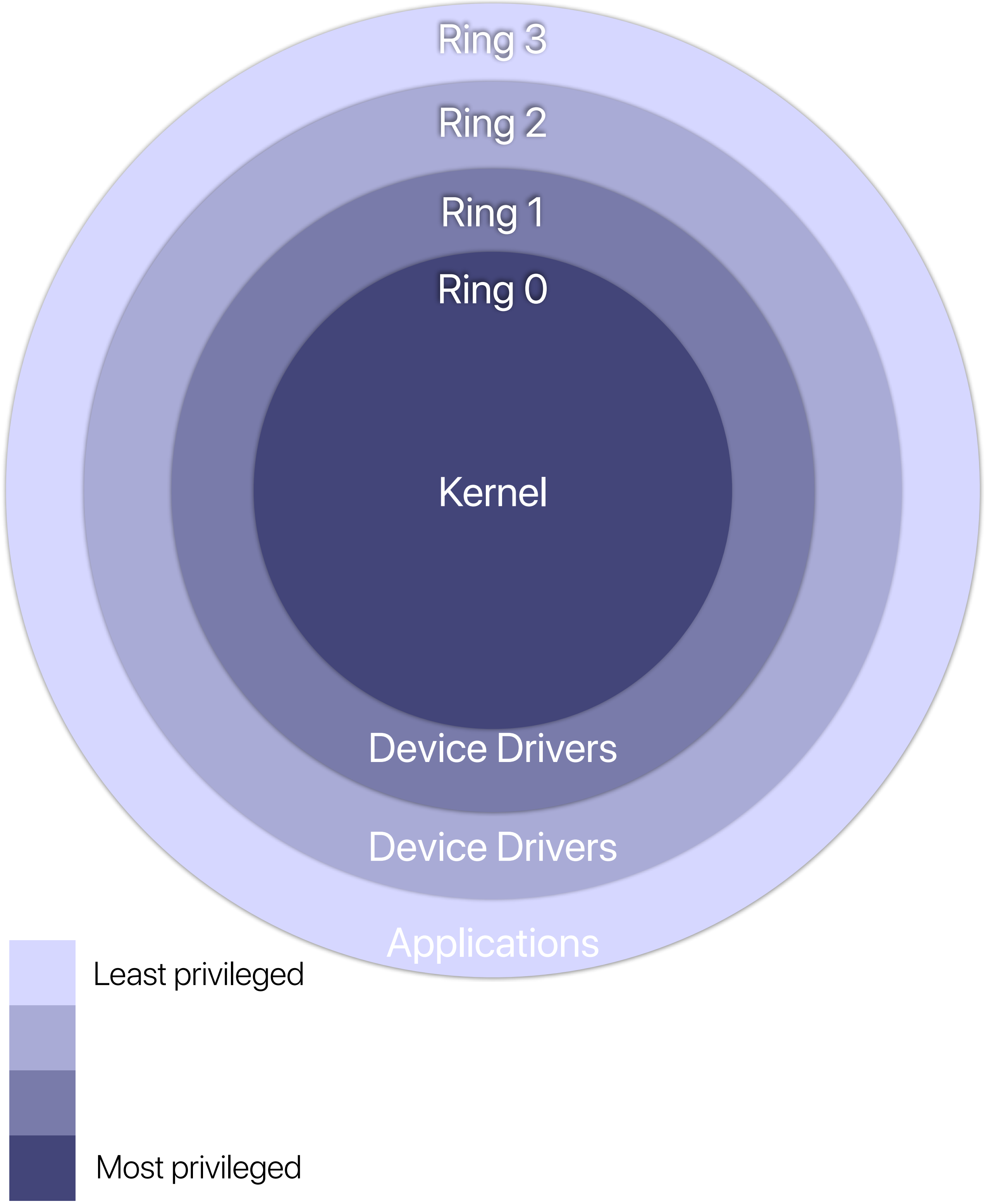
However, we don't want everything to pass through this API!

Too slow!!!

Do you really need to track all intermediate states?

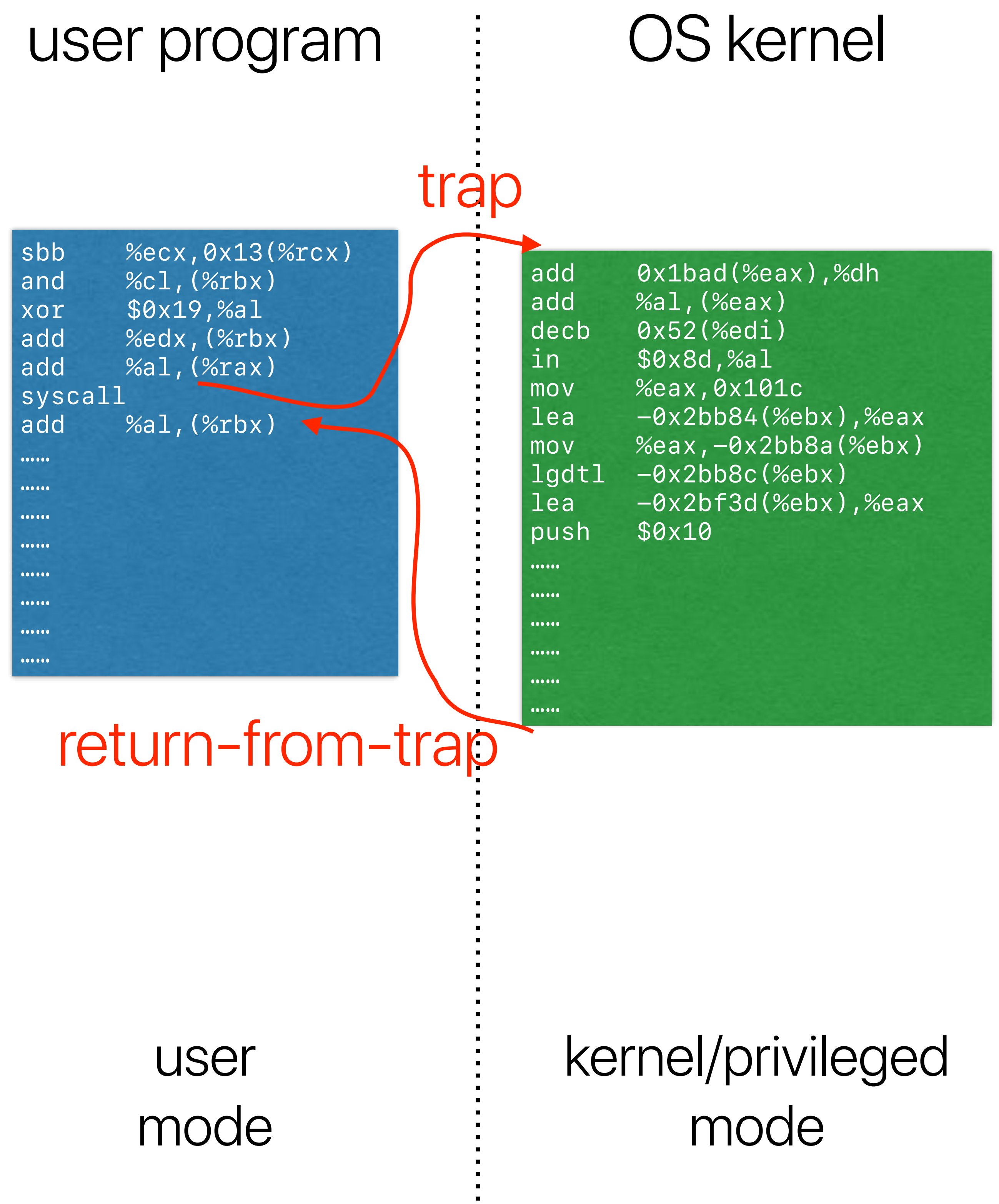
Recap: privileged instructions

- The processor provides **normal** instructions and **privileged** instructions
  - Normal instructions: ADD, SUB, MUL, and etc ...
  - Privileged instructions: HLT, CLTS, LIDT, LMSW, SIDT, ARPL, and etc...
- The processor provides different modes
  - User processes can use normal instructions
  - Privileged instruction can only be used if the processor is in proper mode



Recap: How applications can use privileged operations?

- Through the API: **System calls**
- Implemented in “trap” instructions
  - Raise an exception in the processor
  - The processor saves the exception PC and jumps to the corresponding exception handler in the OS kernel



Hosted virtual machine

Applications | Applications | Applications

OS | OS | OS

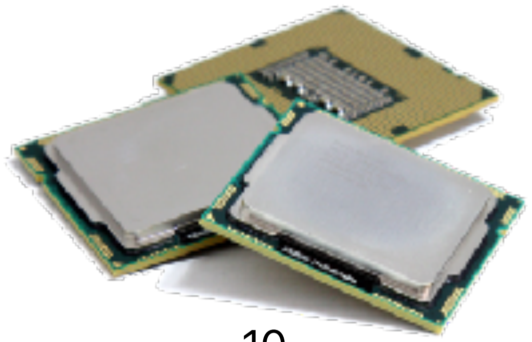
device emulation, virtualization

Virtualized memory | Virtualized storage | Virtualized network | Virtualized ...

Hosted virtual machine monitor

device emulation, virtualization

Hosted operating system


Virtual machine monitors on bare machines

Applications	Applications	Applications
OS	OS	OS

device emulation, virtualization

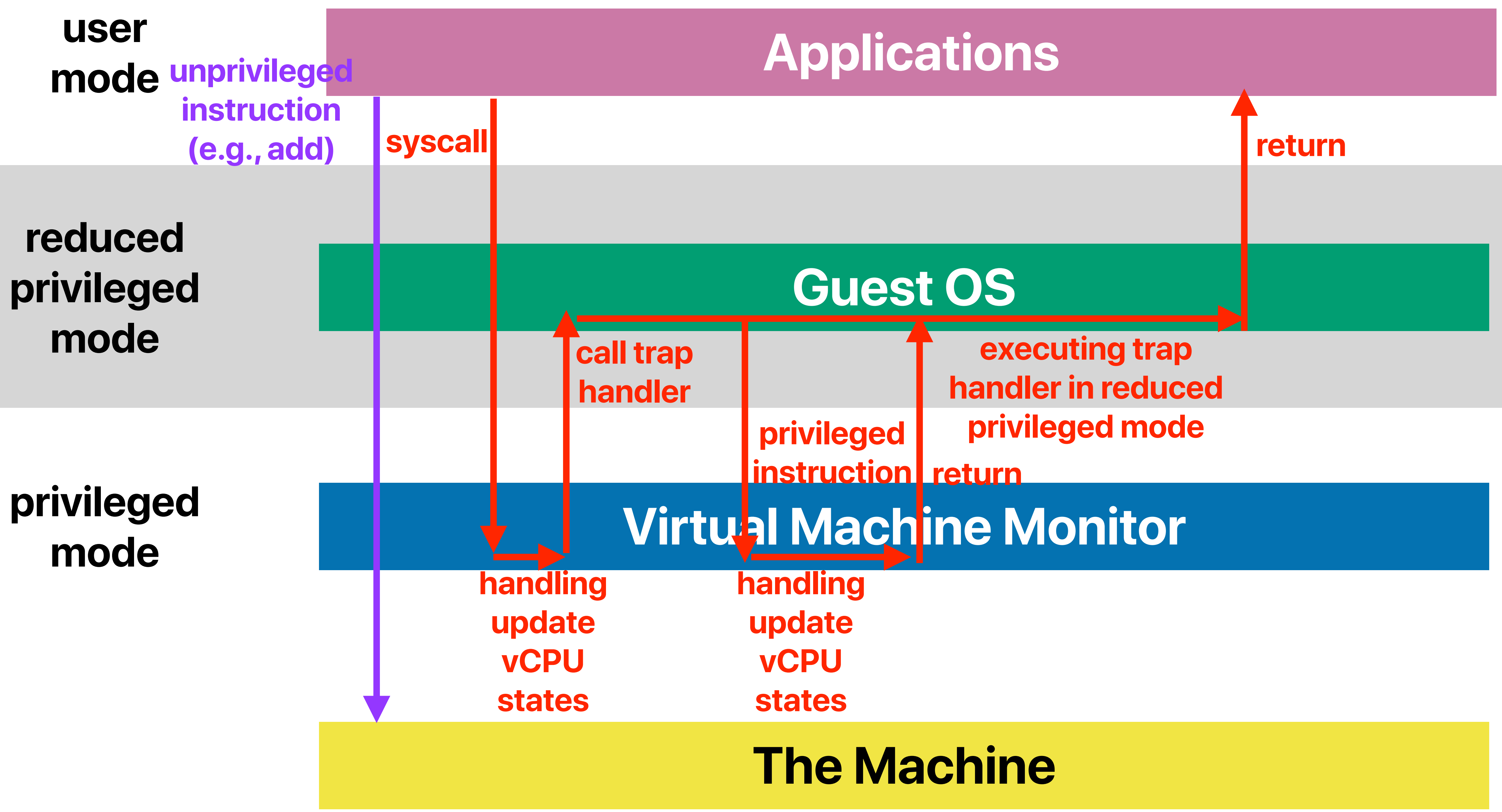
Virtualized memory | Virtualized storage | Virtualized network | Virtualized …

Virtual machine monitor

Three main ideas to classical VMs

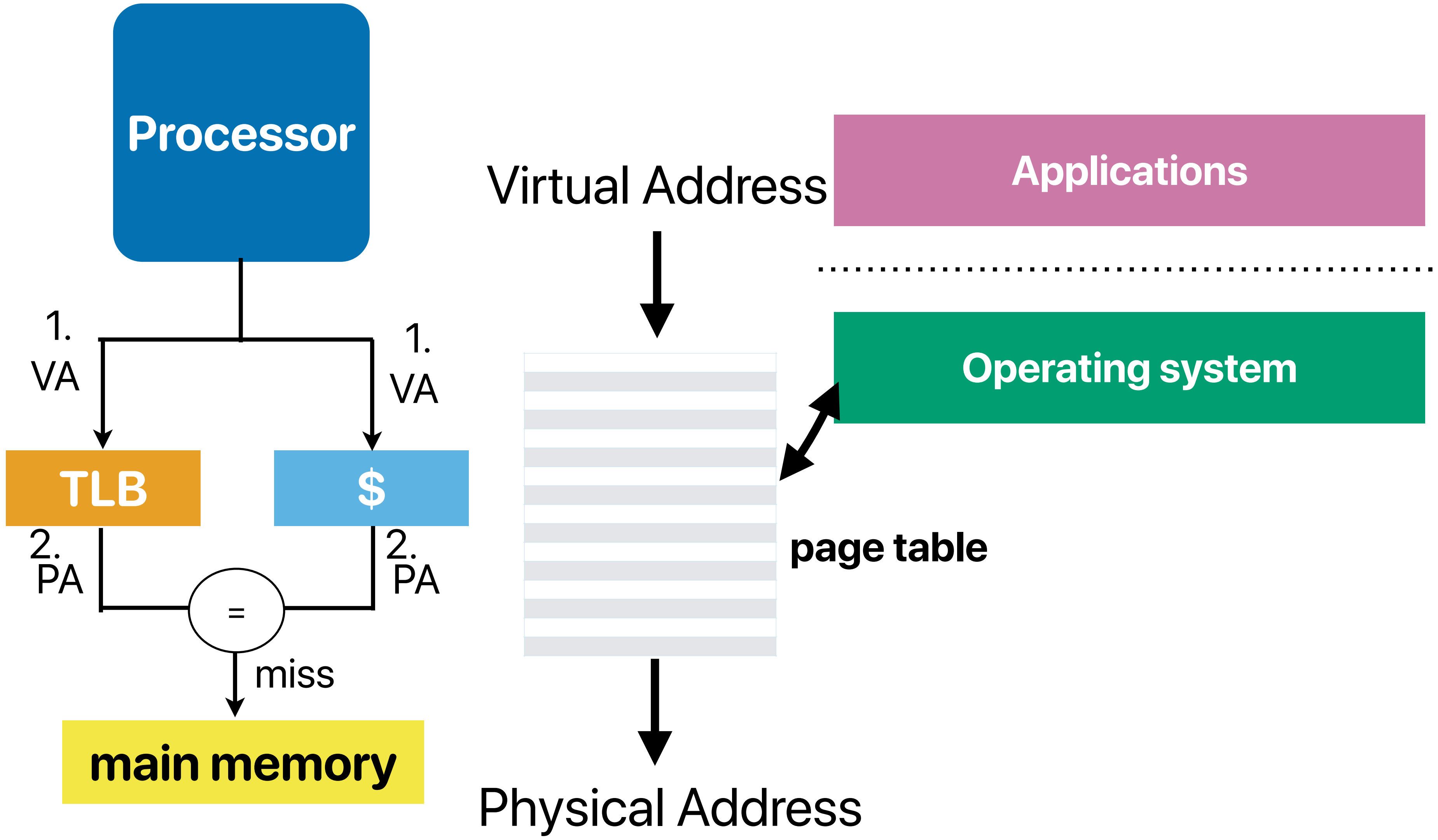
- De-privileging
- Primary and shadow structures
- Tracing

CPU Virtualization: Trap-and-emulate

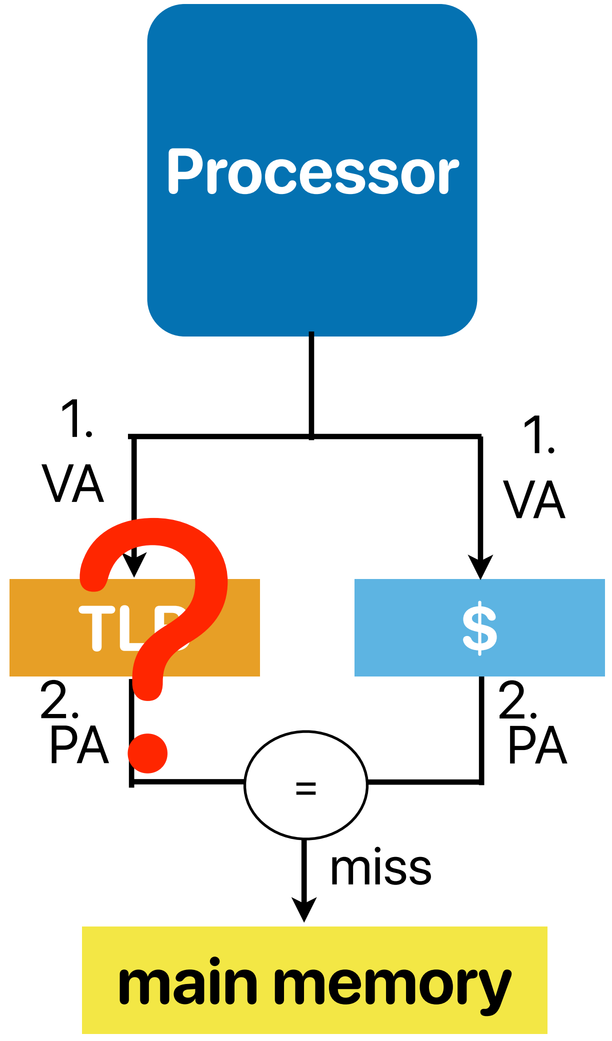


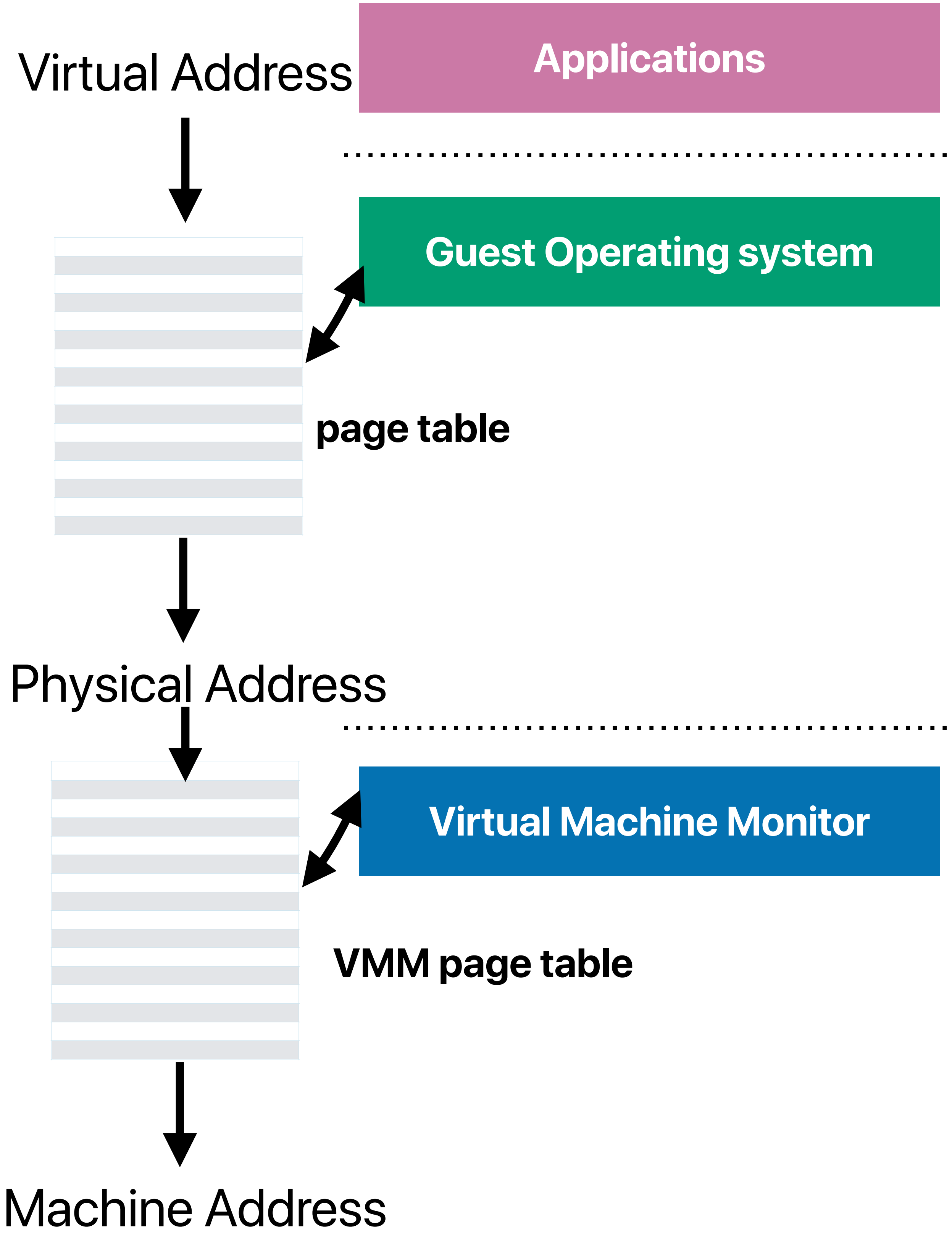
Recap: address translation with TLB

- This is called **virtually indexed, physically tagged cache**
- TLB hit: the translation is in the TLB, no penalty
- TLB miss: fetch the translation from the page table in main memory

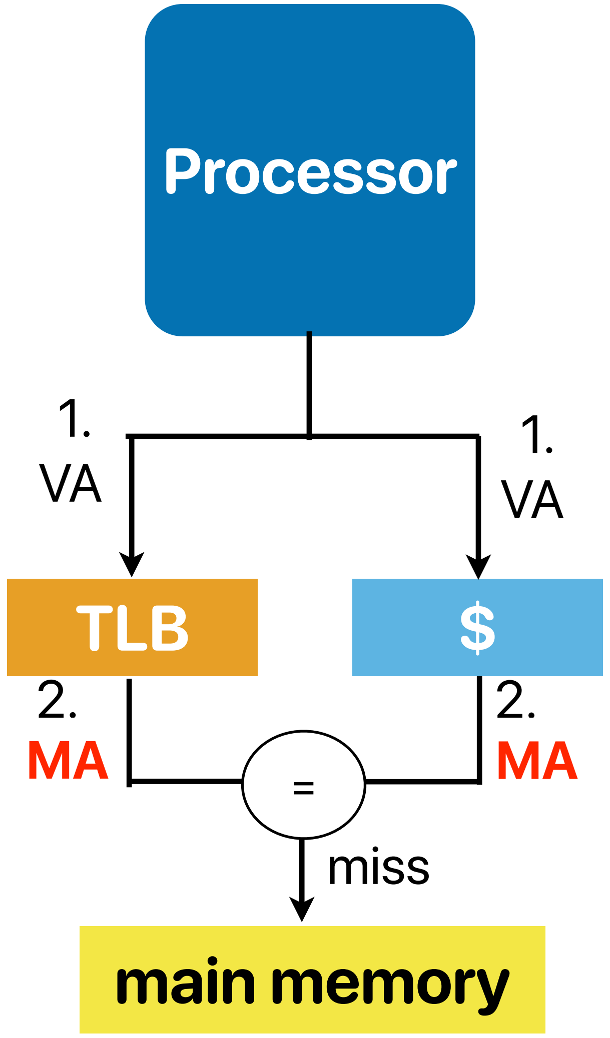


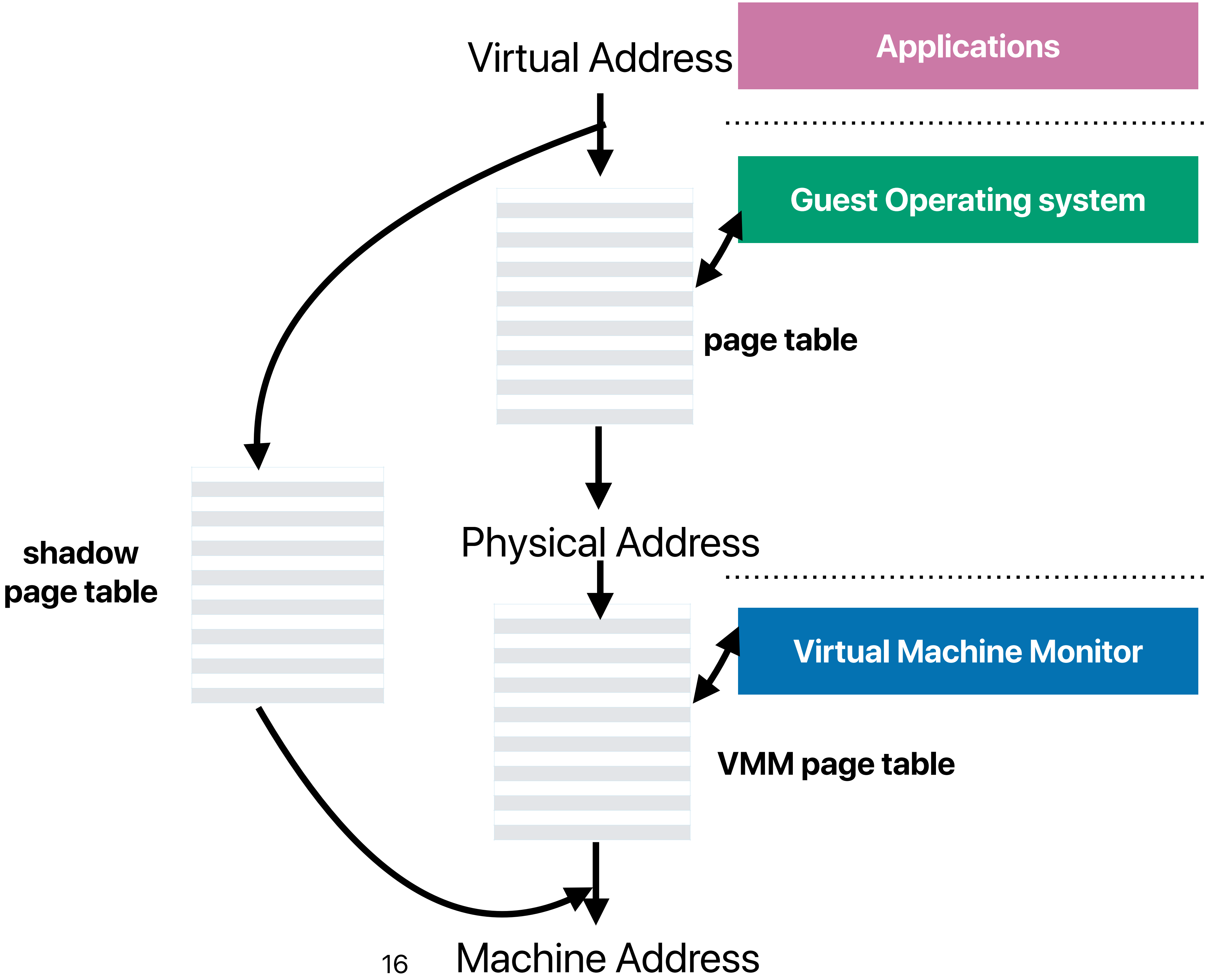
Address translation in VM





Address translation in VM





Tracing

- You need to make the shadow page table consistent with guest OS page table
- Protect these structures with write-protected
  - If anyone tries to modify the protected PTE — trigger a segfault handler
  - The segfault handler will deal with these write-protected locations and consistency issues for both tables

A Comparison of Software and Hardware Techniques for x86 Virtualization

Keith Adams and Ole Agesen
VMware

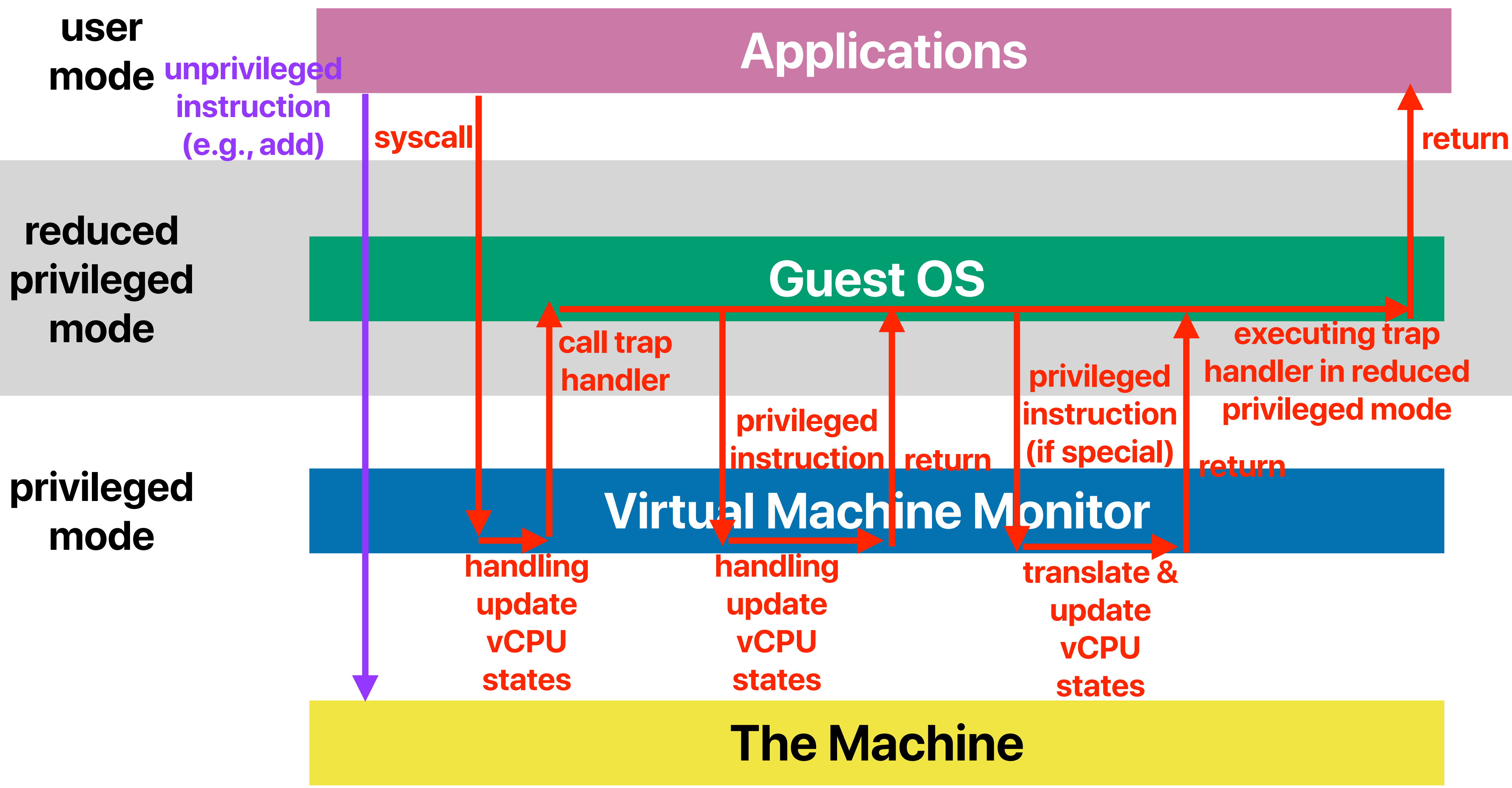
Binary translator

- Binary
- Dynamic
- On demand
- System level
- Subsetting
- Adaptive

Binary translation on x86

- If the virtualized CPU is in user mode
  - Instructions execute directly
- If the virtualized CPU is in kernel mode
  - VMM examines every instruction that the guest OS is about to execute in the near future by prefetching and reading instructions from the current program counter
  - Non-special instructions run natively
  - Special instructions (those instruction may have missing flags set) are "translated" into equivalent instructions with flags set

Trap-and-emulate with Binary Translation



Hardware virtualization in modern x86

- VMCB (Virtual machine control block)
  - Settings that determine what actions cause the guest to exit to host
  - All CPU state for a guest is located in VMCB data-structure
- A new, less privileged execution mode, guest mode
  - `vmrun` instruction to enter VMX mode
  - Many instructions and events cause VMX exits
  - Control fields in VMCB can change VMX exit behavior

How hardware VM works

- VMM fills in VMCB exception table for Guest OS
  - Sets bit in VMCB not exit on syscall exception
- VMM executes vmrun
- Application invokes syscall
- CPU —> CPL #0, does not trap, vectors to VMCB exception table

Virtualization overhead

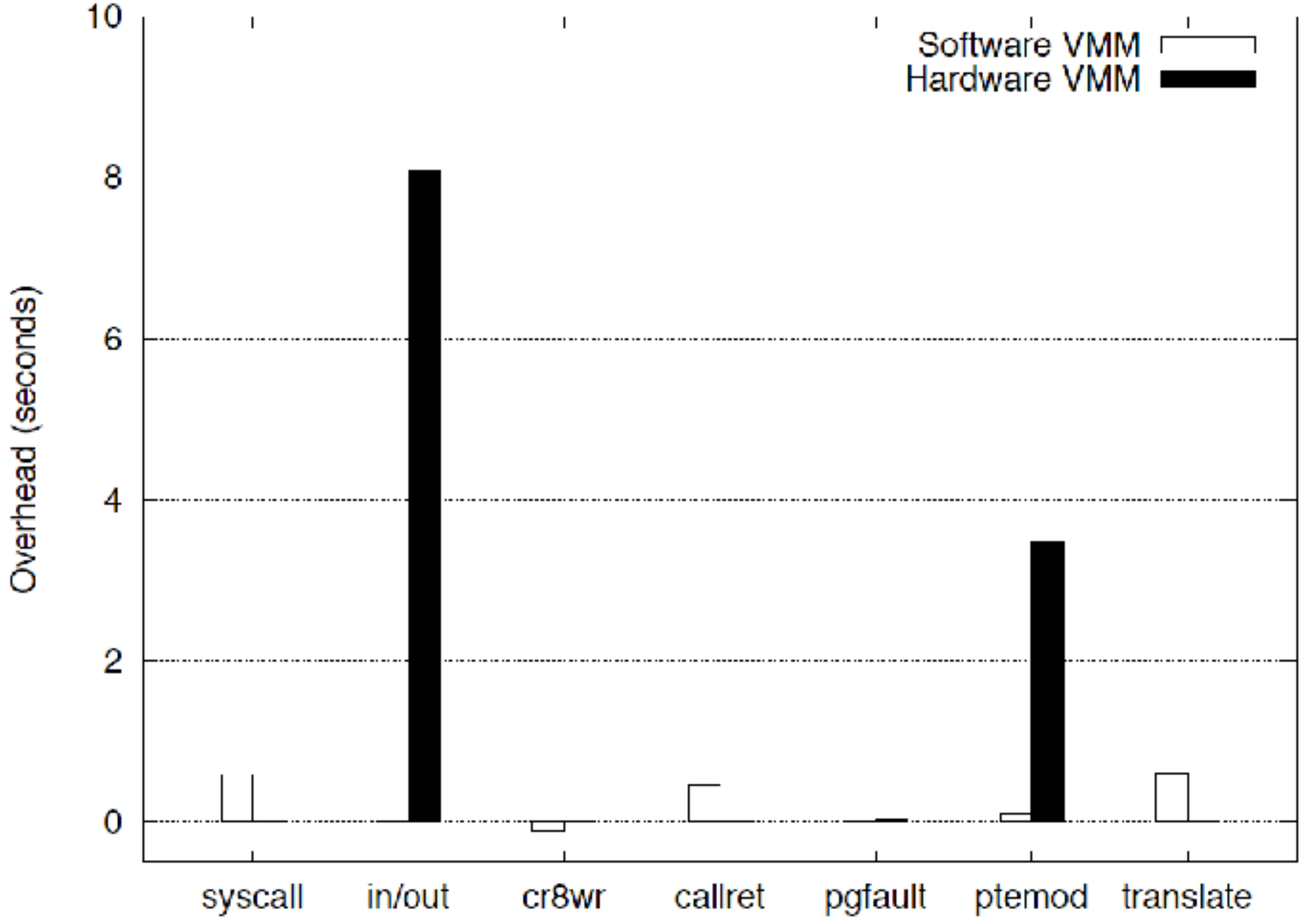


Figure 5. Sources of virtualization overhead in an XP boot/halt.

Nanobenchmarks

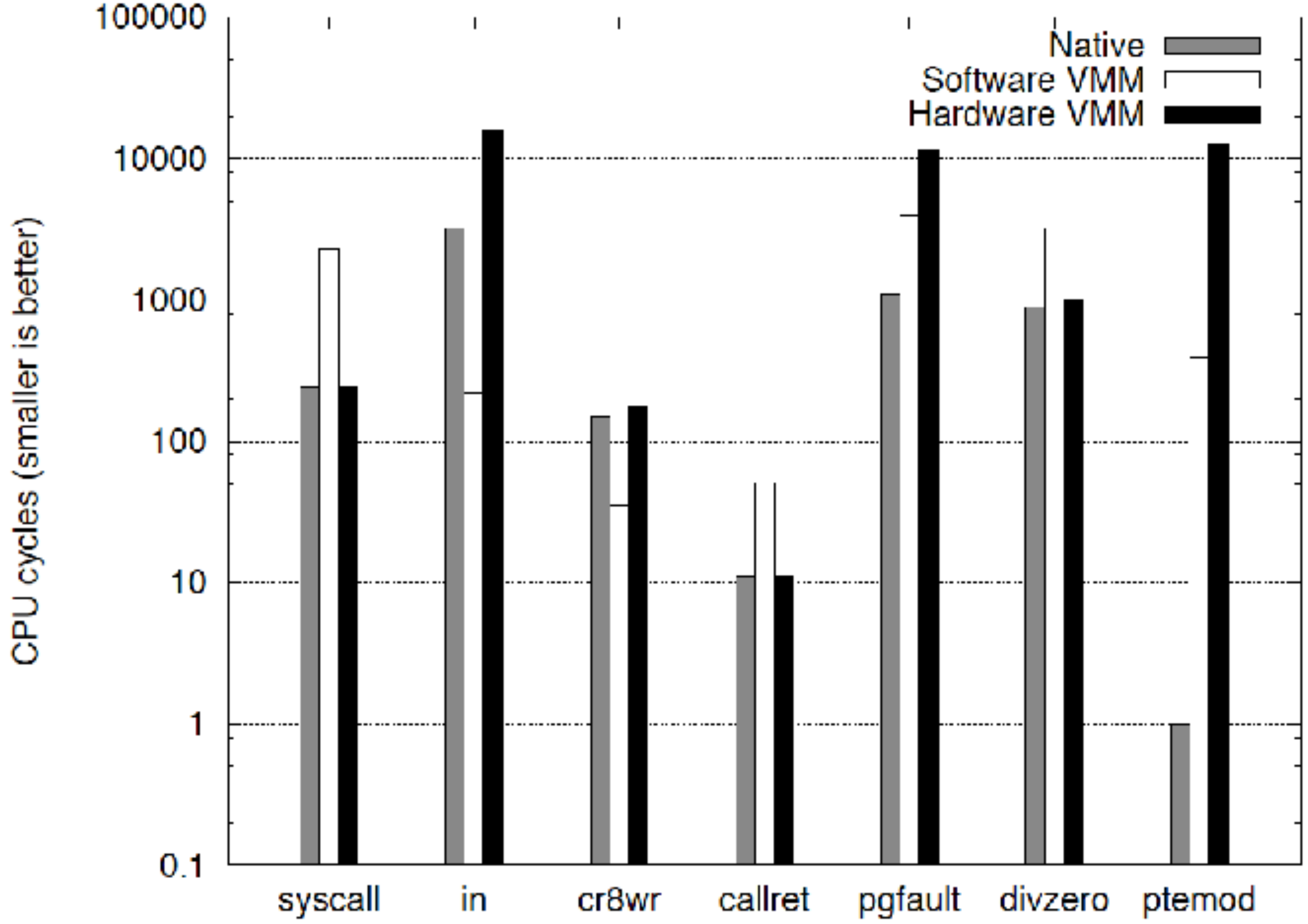


Figure 4. Virtualization nanobenchmarks.

Macrobenchmarks

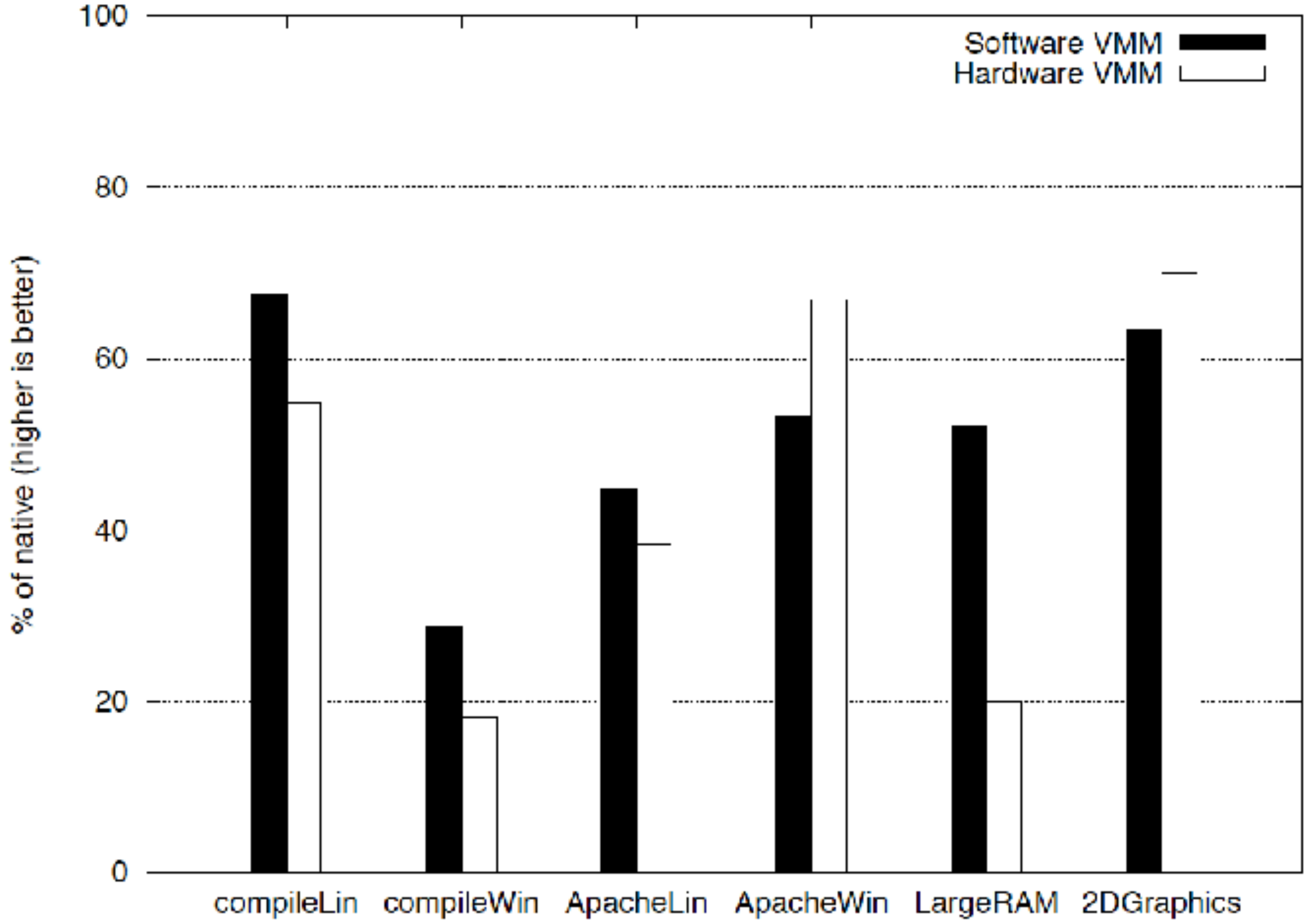


Figure 3. Macrobenchmarks.

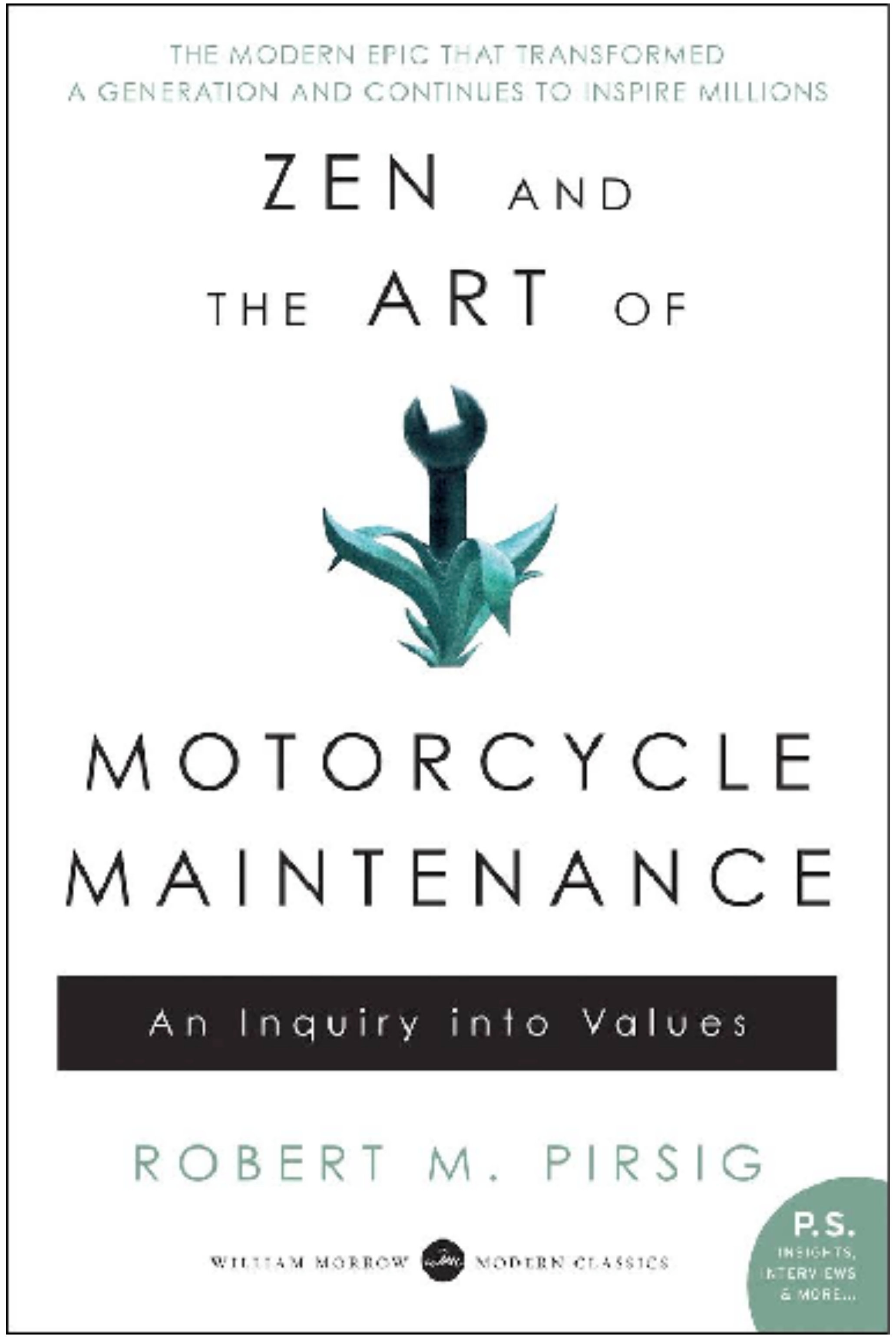
Side-by-side comparison

- Binary Translation VMM:
  - Converts traps to callouts
    - Callouts faster than trapping
  - Faster emulation routine
    - VMM does not need to reconstruct state
  - Avoids callouts entirely
- Hardware VMM:
  - Preserves code density
  - No precise exception overhead
  - Faster system calls

Xen and the Art of Virtualization

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield
University of Cambridge Computer Laboratory

Why “Xen and the Art of Virtualization”?



Why Xen?

- Server consolidation: improve the server utilization
- Server co-location
- Secure distributed computing
- We want to host many full OS instances efficiently
  - The overhead of full virtualization/resource container is large
  - Hard to achieve Quality of Service guarantee because a VM is treated as a process in the host operating system

Xen hypervisor

user mode

Applications | Applications | Applications

reduced privileged mode (ring 1)

Modified OS | Modified OS | Modified OS

device emulation, virtualization

Para-Virtualized CPU | Para-Virtualized memory | Para-Virtualized storage | Para-Virtualized network | Para-Virtualized ...

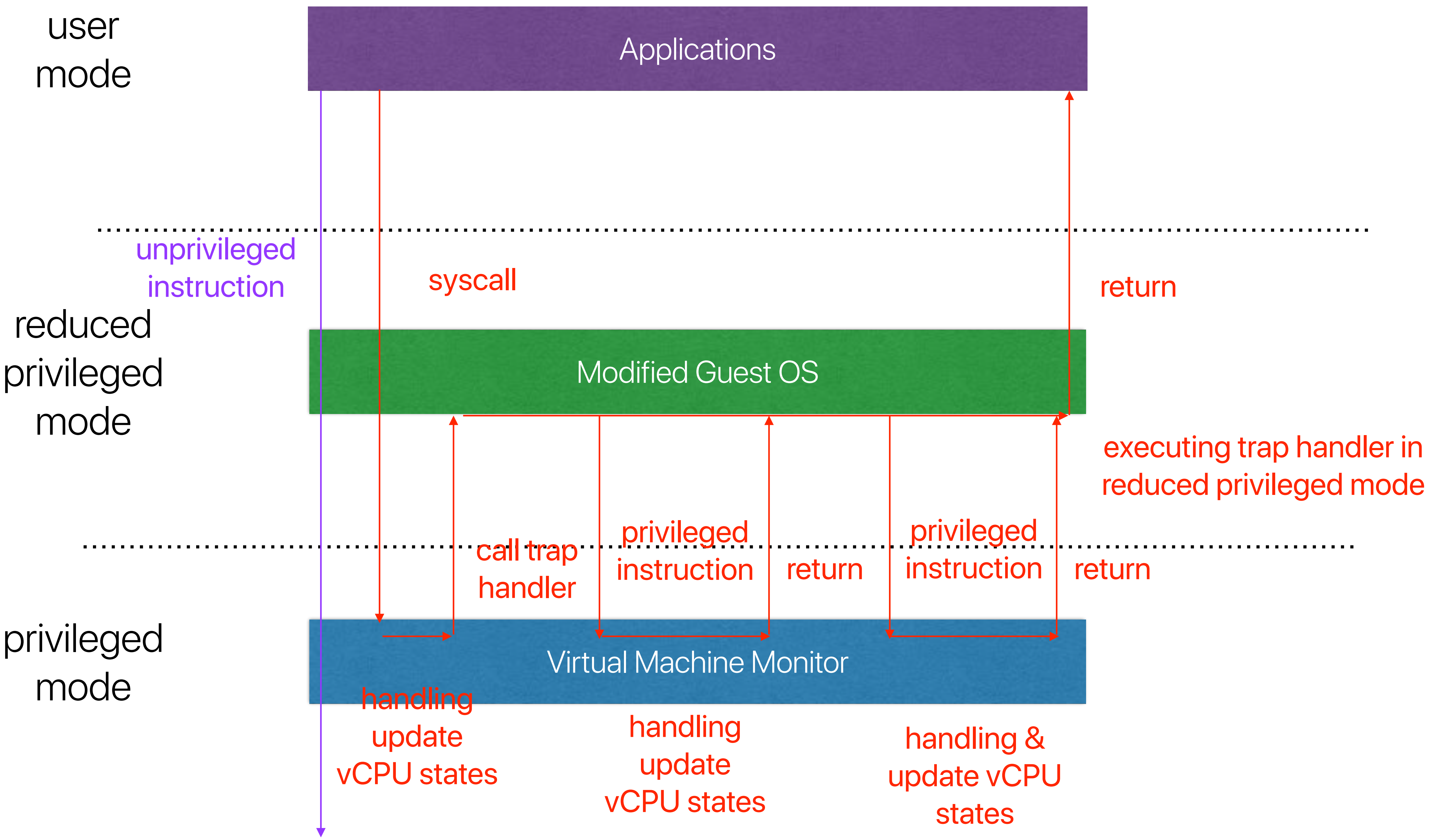
privileged mode

Paravirtualization

- Solution to issues with x86 instruction set
  - Don’t allow guest OS to issue sensitive instructions
  - Replace those sensitive instructions that don’t trap to ones that will trap
- Guest OS makes “hypercalls” (like system calls) to interact with system resources
  - Allows hypervisor to provide protection between VMs
- Exceptions handled by registering handler table with Xen
  - Fast handler for OS system calls invoked directly
  - Page fault handler modified to read address from replica location
- Guest OS changes largely confined to arch-specific code
  - Compile for ARCH=xen instead of ARCH=i686
  - Original port of Linux required only 1.36% of OS to be modified

Trap-and-emulate

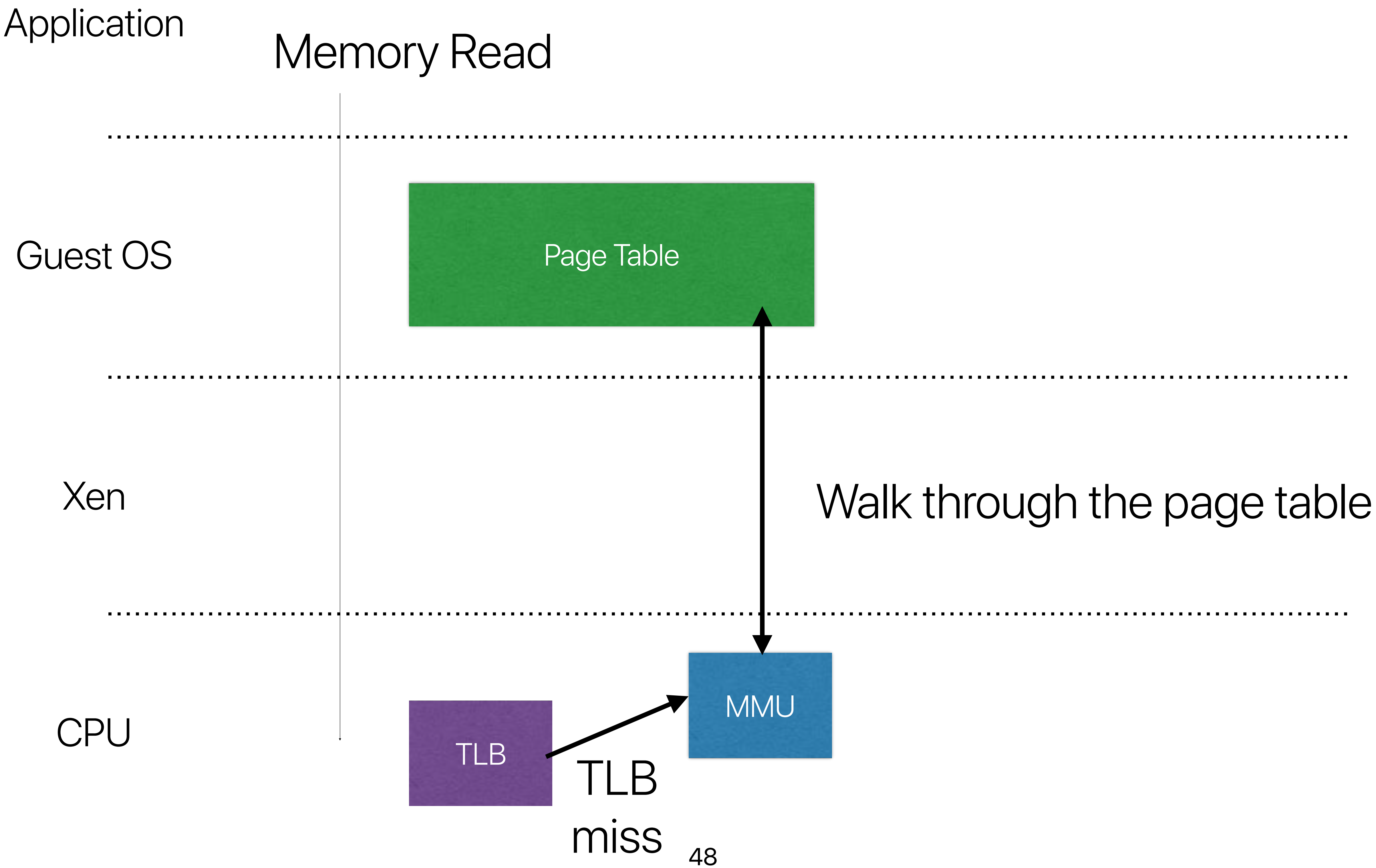
As we modified the OS code, no binary translation is necessary



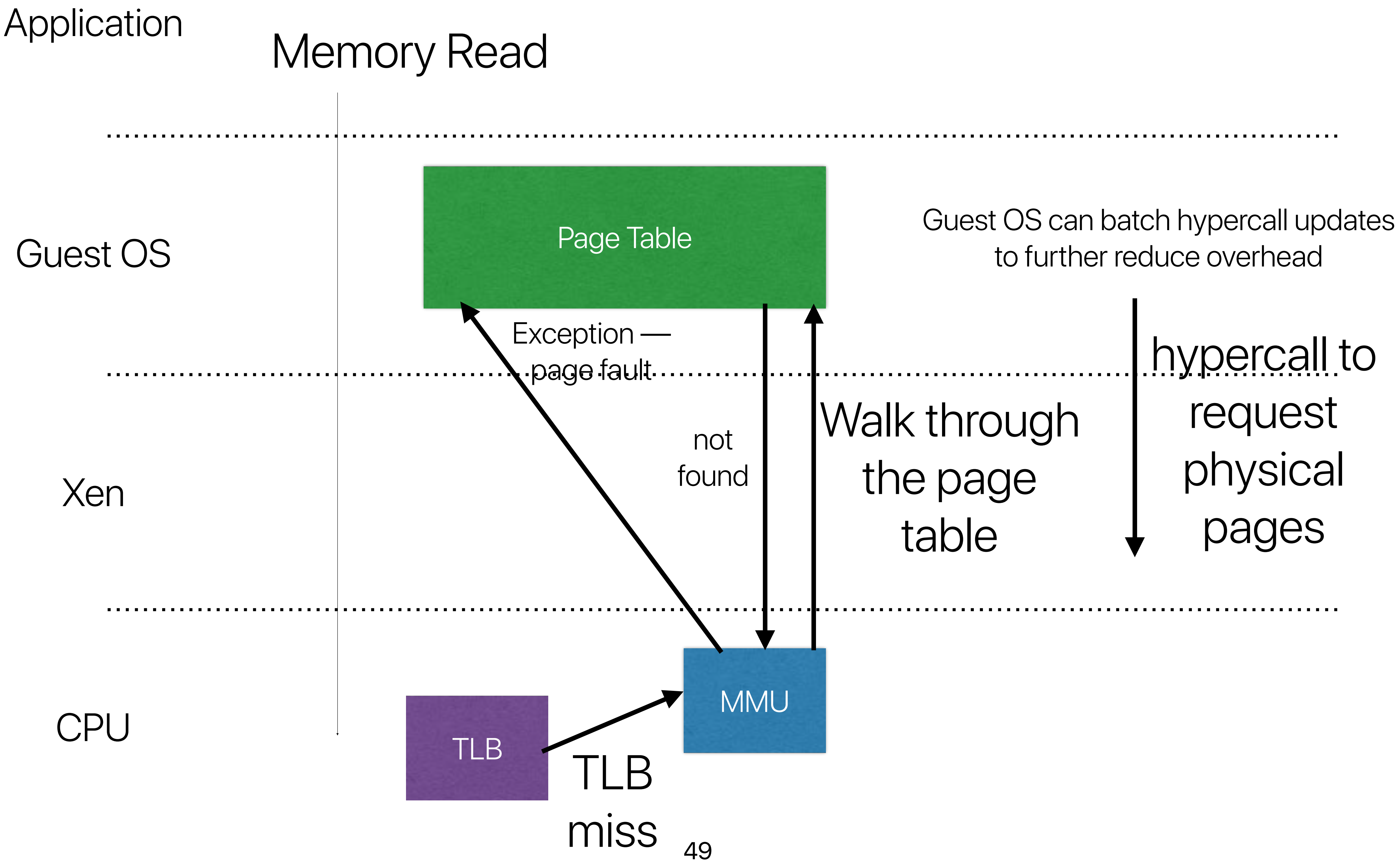
MMU Virtualization: Direct mode

- Modifying the guest OS to be involved only for page table updates
- Restricting the guest OS to have only read access
- Writing to page tables is protected and must use a hypercall — Xen can verify and allocate pages

Accessing a page — TLB miss



Accessing a page — page fault



Balloon driver

- Mechanism that forces guest OS to give up memory
- Balloon driver consumes physical memory allocated in the guest OS
- The memory consumed by Balloon is given to Xen
- The guest OS uses hypercalls to see and change the state

I/O virtualization

- Exposes I/O devices as asynchronous I/O rings to guest OS
- Exposes the device abstraction to minimize the change in device drivers
- Xen pins a few physical memory as DMA buffers and exposes to the guest OS to avoid copying overhead
- Use an up call to notify the guest OS as opposed to interrupts

Network virtualization

- Virtual firewall for each physical network interface
- Virtual interface for each physical network interface in each guest OS
- Circular Queue — Mechanism supporting I/O between Xen and guest OSes
  - Ring buffers for exchanging requests
  - Producer-consumer problem
    - Producers: guest OSes
    - Consumer: Xen

Performance

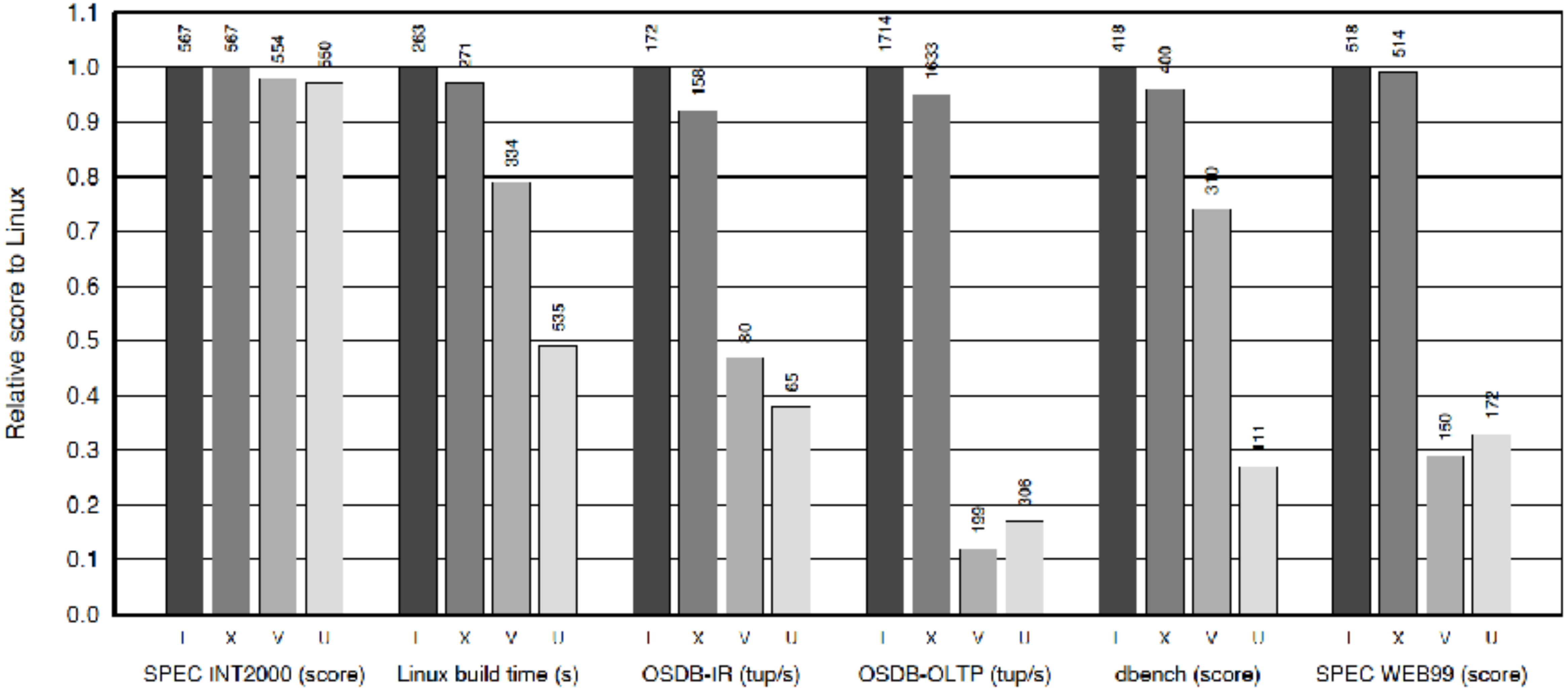


Figure 3: Relative performance of native Linux (L), XenoLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).

Overhead

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

	TCP MTU 1500 TX	TCP MTU 1500 RX	TCP MTU 500 TX	TCP MTU 500 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

Effort of porting

- Do you buy this?

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%)**

Later evolution of Xen

- x86-64 removes ring 1, 2
  - Both applications and guest OSes in ring 3
  - Using guest mode in Intel VT-X/AMD VMX when necessary
- Higher performance NIC through segment offload
- Enhanced support for unmodified guest OSes using hardware virtualization
- Secure isolation between VMs

Hints for computer system design

Butler W. Lampson
Computer Science Laboratory Xerox Palo Alto Research Center

Hints for computer system design

Why?	*Functionality* Does it work?	*Speed* Is it fast enough?	*Fault-tolerance* Does it keep working?
Where?			
Completeness	Separate normal and worst case	Shed load	
		End to end	End to end
		Safety first	
Interface	Do one thing well:	Make it fast	End-to-end
	Don't generalize	Split resources	Log updates
	Get it right	Static analysis	Make actions atomic
	Don't hide power	Dynamic translation	
	Use procedure arguments		
	Leave it to the client		
	Keep basic interfaces stable		
	Keep a place to stand		
Implementation	Plan to throw one away	Cache answers	Make actions atomic
	Keep secrets	Use hints	Use hints
	Use a good idea again	Use brute force	
	Divide and conquer	Compute in background	
		Batch processing	

Completeness

- Separate normal and worst case
- Make normal case fast
- The worst case must make progress
  - Saturation
  - Thrashing

Interface — Keep it simple, stupid

- Do one thing at a time or do it well
  - Don’t generalize
  - Example
    - Interlisp-D stores each virtual page on a dedicated disk page
      - 900 lines of code for files, 500 lines of code for paging
      - fast — page fault needs one disk access, constant computing cost
    - Pilot system allows virtual pages to be mapped to file pages
      - 11000 lines of code
      - Slower — two disk accesses in handling a page fault, under utilize the disk speed
- Get it right

More on Interfaces

- Make it fast, rather than general or powerful
  - CISC v.s. RISC
- Don’t hide power
  - Are we doing all right with FTL?
- Use procedure arguments to provide flexibility in an interface
  - Thinking about SQL v.s. function calls
- Leave it to the client
  - Monitors’ scheduling
  - Unix’s I/O streams

Implementation

- Keep basic interfaces stable
  - What happen if you changed something in the header file?
- Keep a place to stand if you do have to change interfaces
  - Mach/Sprite are both compatible with existing UNIX even though they completely rewrote the kernel
- Plan to throw one away
- Keep secrets of the implementation — make no assumption other system components
  - Don't assume you will definitely have less than 16K objects!
- Use a good idea again
  - Caching!
  - Replicas
- Divide and conquer

Speed

- Split resources in a fixed way if in doubt, rather than sharing them
  - Processes
  - VMM: Multiplexing resources Guest OSs aren’t even aware that they’re sharing
- Use static analysis — compilers
- Dynamic translation from a convenient (compact, easily modified or easily displayed) representation to one that can be quickly interpreted is an important variation on the old idea of compiling
  - Java byte-code
  - LLVM
- Cache answers to expensive computations, rather than doing them over
- Use hints to speed up normal execution
  - The Ethernet: carrier sensing, exponential backoff

Speed

- When in doubt, use brute force
- Compute in background when possible
  - Free list instead of swapping out on demand
  - Cleanup in log structured file systems: segment cleaning could be scheduled at nighttime.
- Use batch processing if possible
  - Soft timers: uses trigger states to batch process handling events to avoid trashing the cache more often than necessary
  - Write buffers
- Safety first
- Shed load to control demand, rather than allowing the system to become overloaded
  - Thread pool
  - MLQ scheduling
  - Working set algorithm
  - Xen v.s. VMWare

Fault-tolerance

- End-to-end
  - Network protocols
- Log updates
  - Logs can be reliably written/read
  - Logs can be cheaply forced out to disk, which can survive a crash
    - Log structured file systems
    - RAID5 in Elephant
- Make actions atomic or restartable
  - NFS
  - atomic instructions for locks