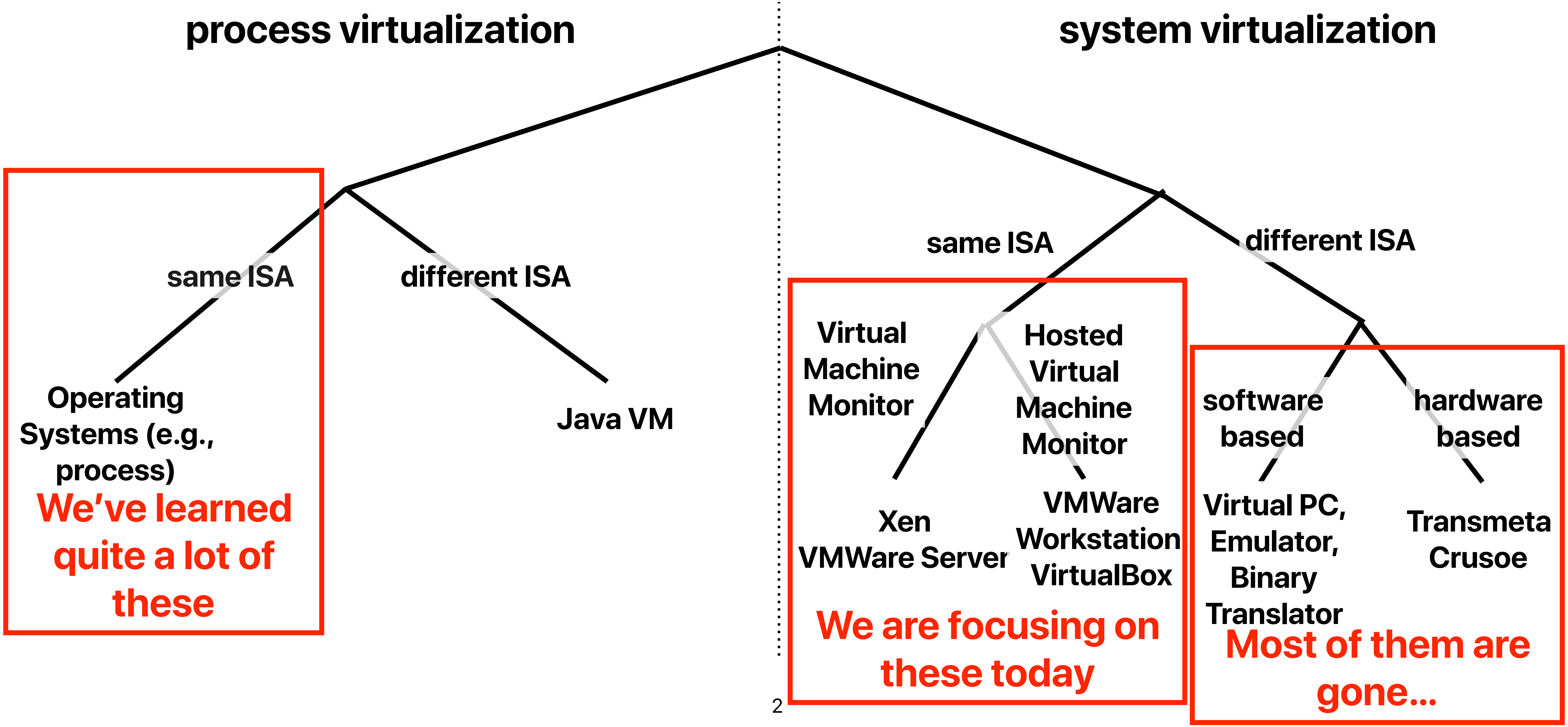
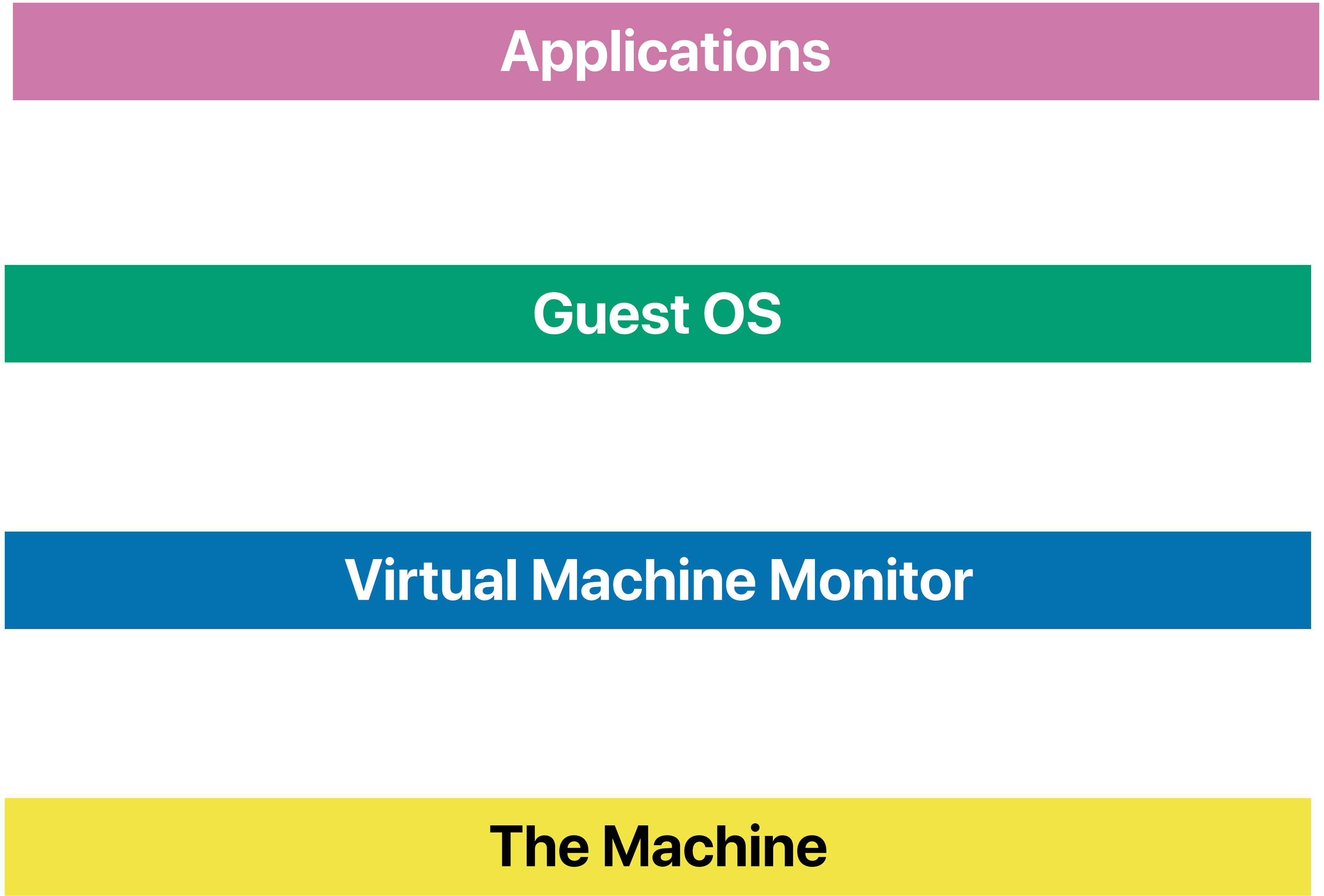
Virtual Machines & Revisiting Computer System Designs

Hung-Wei Tseng

Taxonomy of virtualization

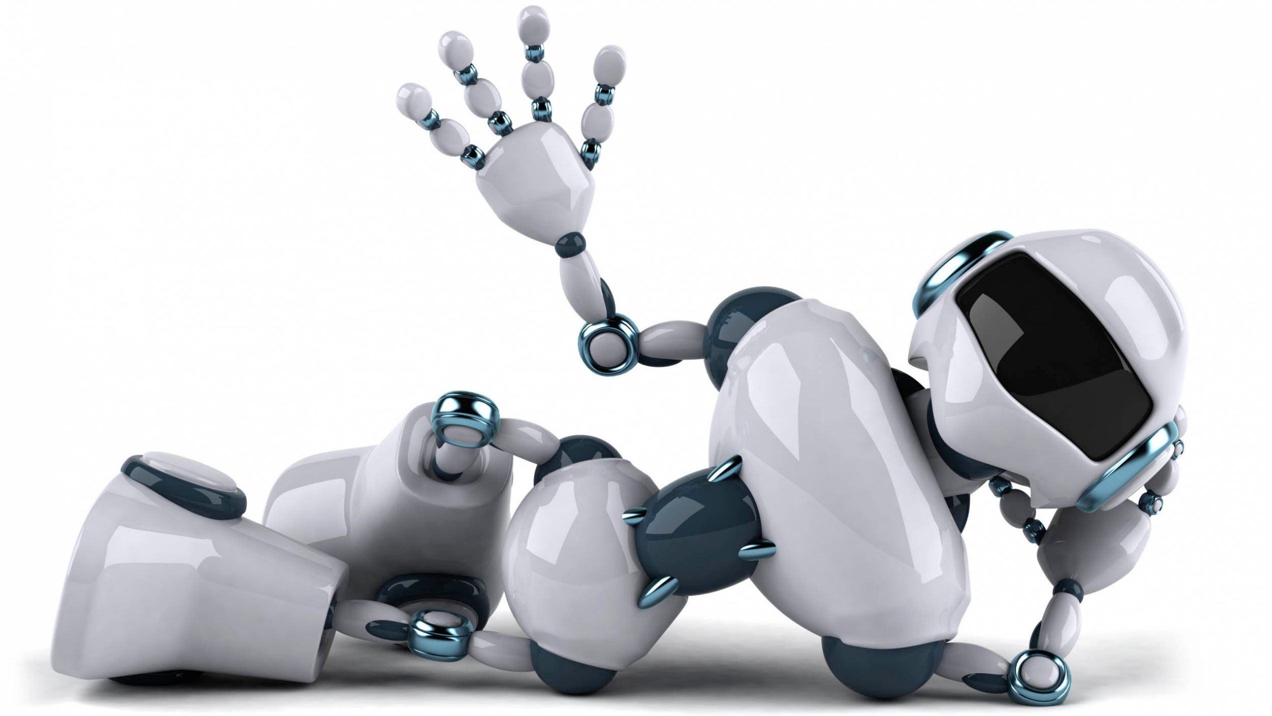


Virtual machine architecture



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

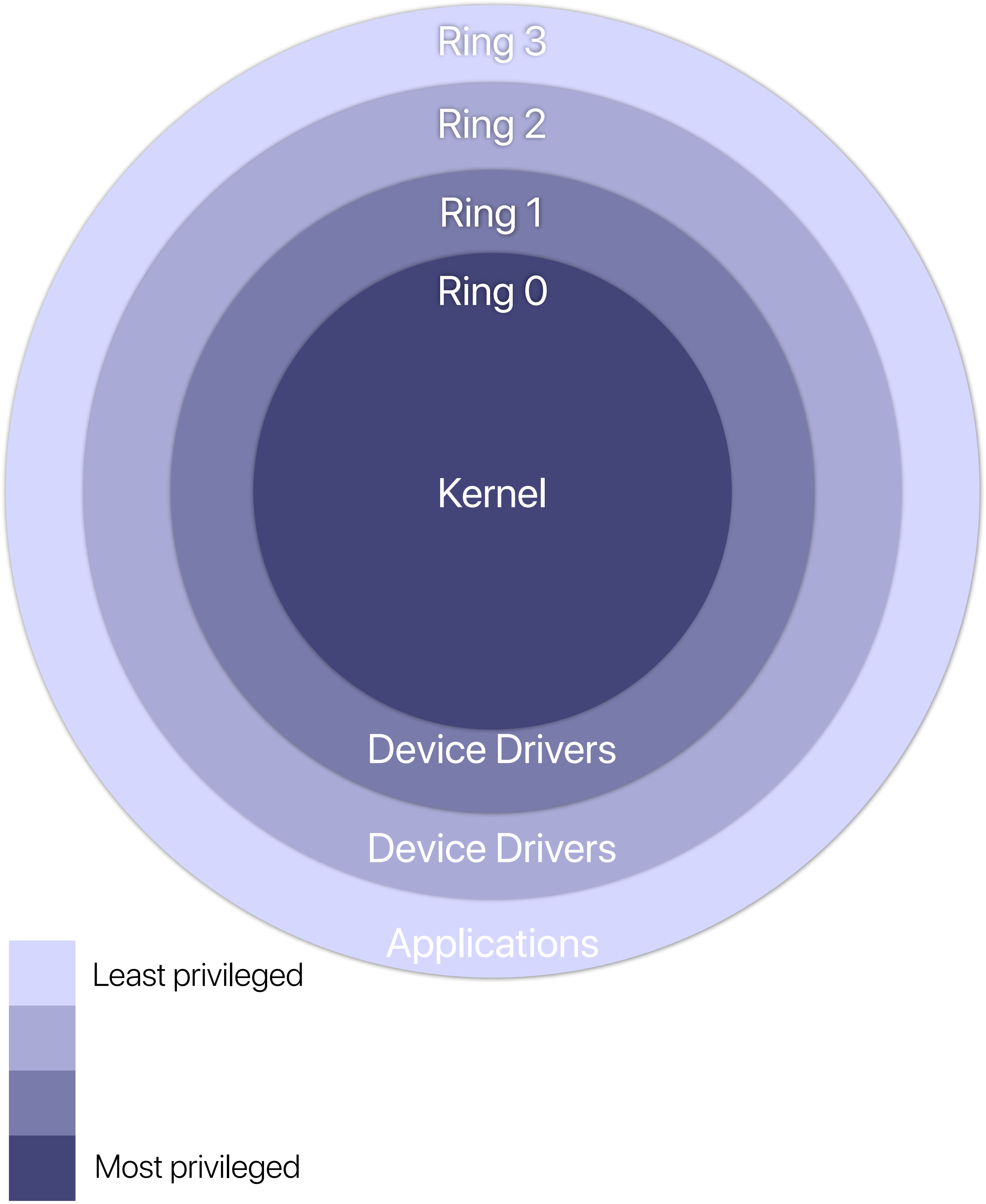
However, we don't want everything to pass through this API!

Too slow!!!

Do you really need to track all intermediate states?

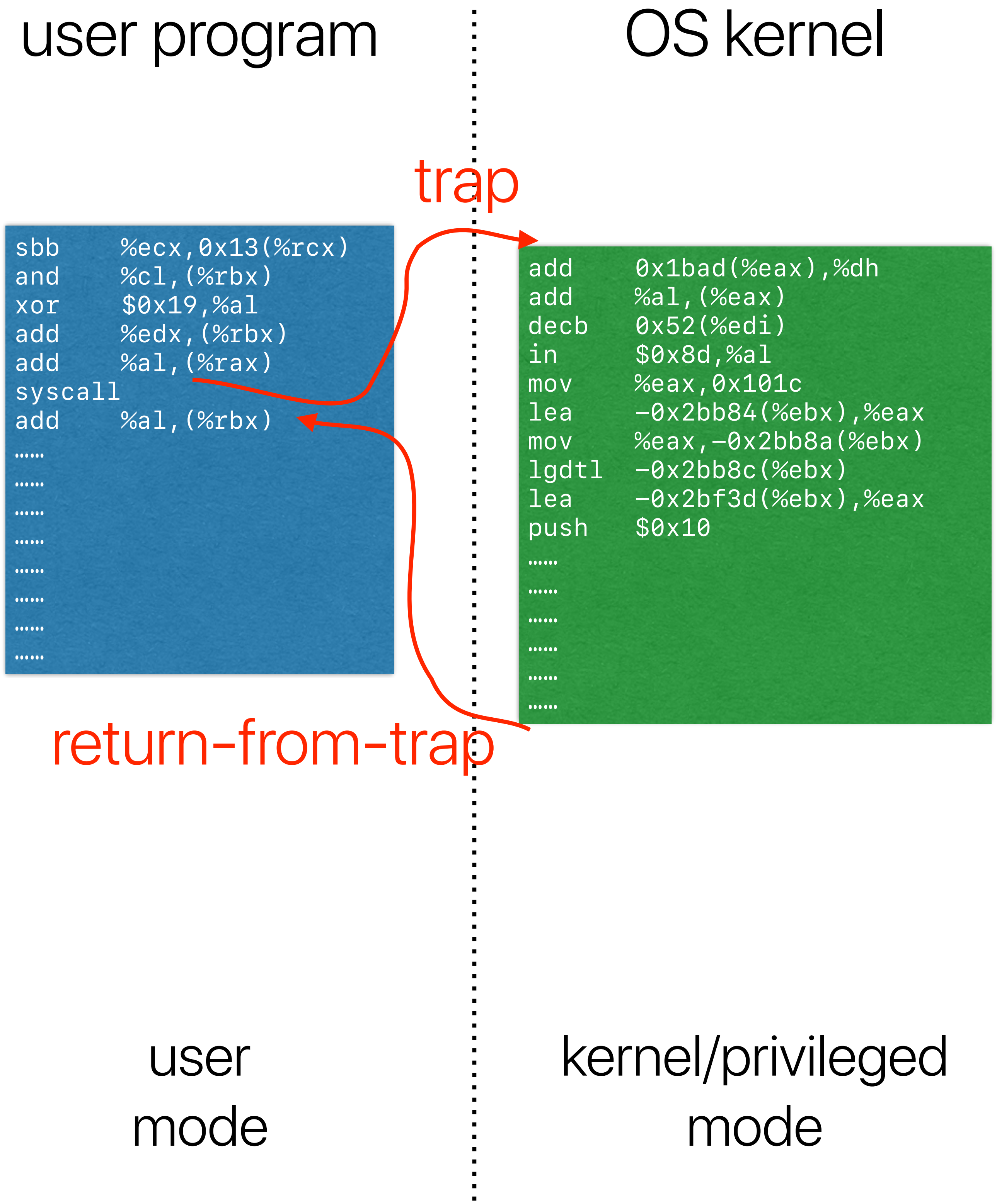
Recap: privileged instructions

- 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

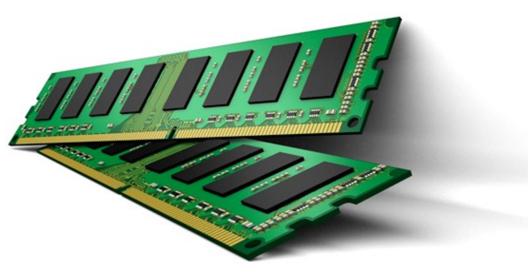
device emulation, virtualization

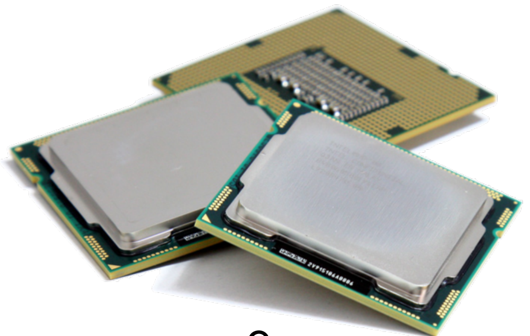
Virtualized memory | Virtualized storage | Virtualized network | Virtualized ...

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

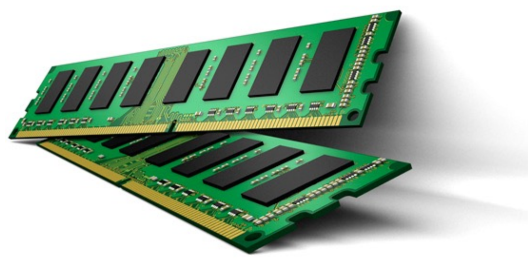
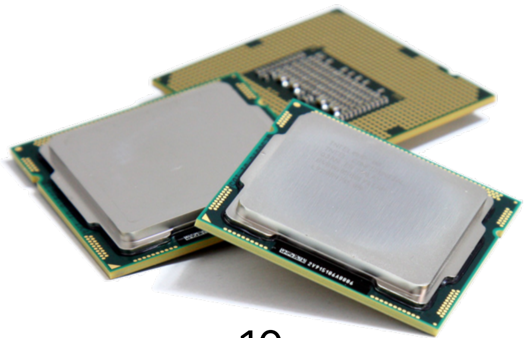
Virtualized memory

Virtualized storage

Virtualized network

Virtualized ...

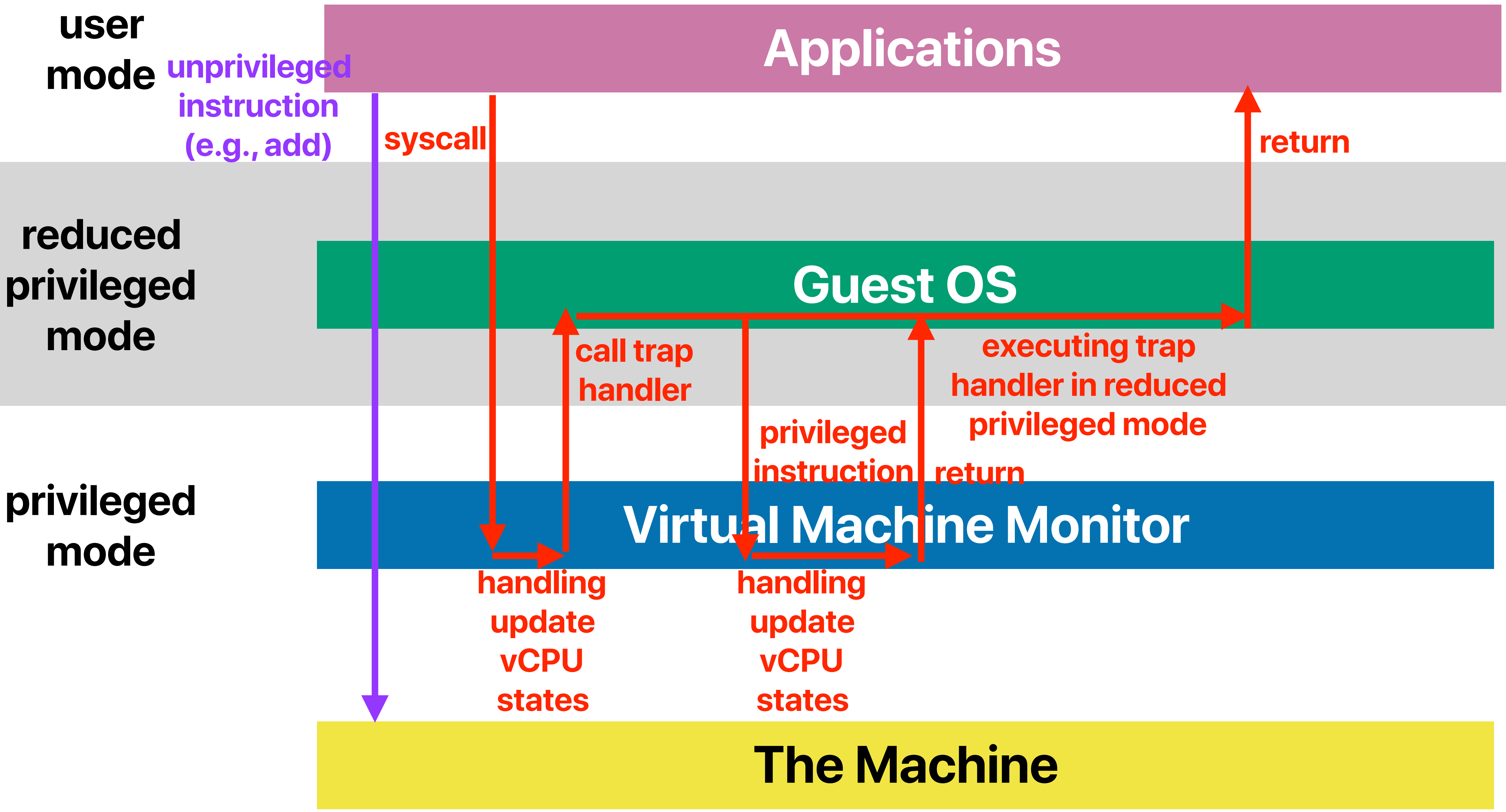
Virtual machine monitor


Three main ideas to classical VMs

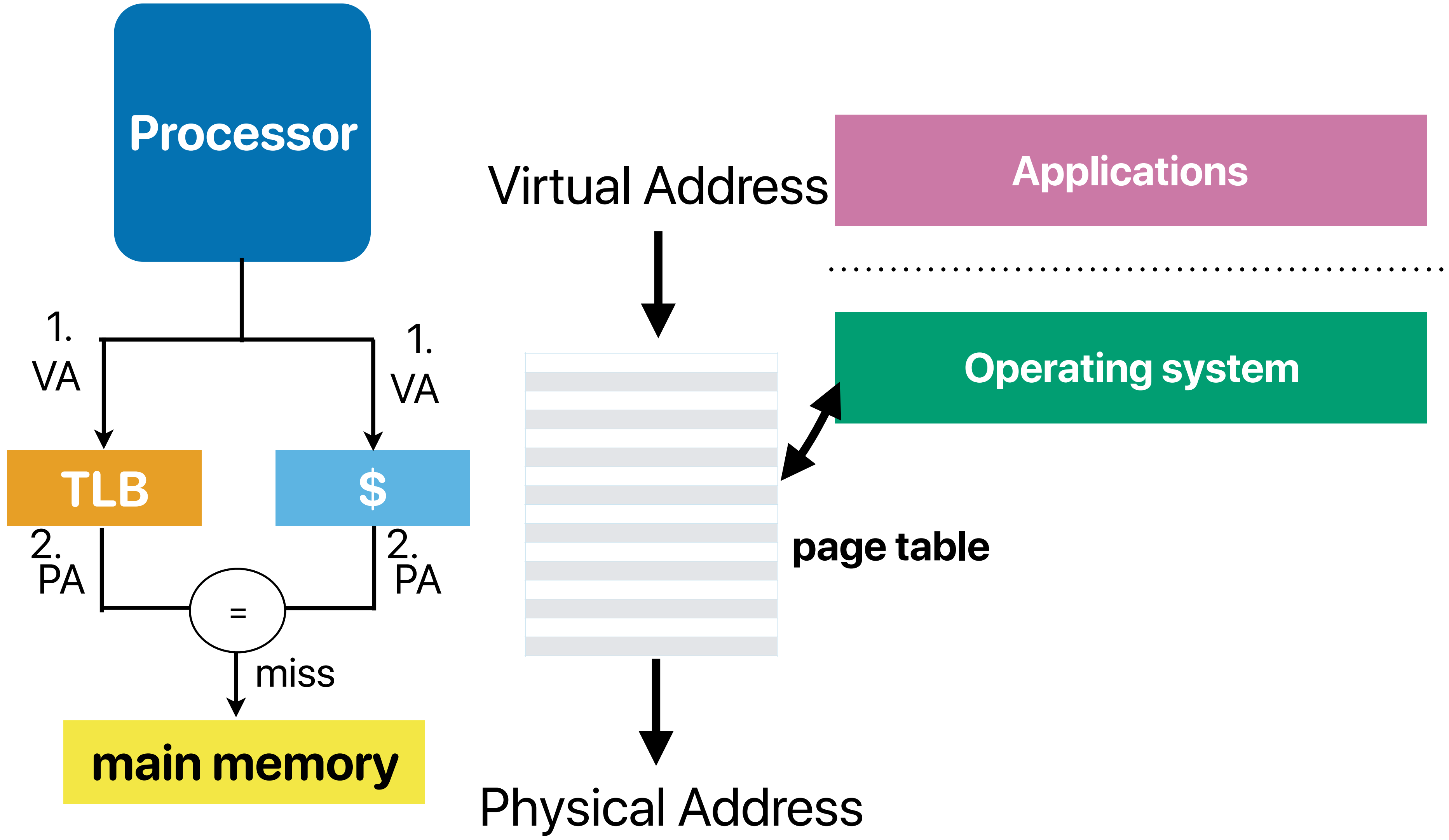
- De-privileging
- Primary and shadow structures
- Tracing

CPU Virtualization: Trap-and-emulate

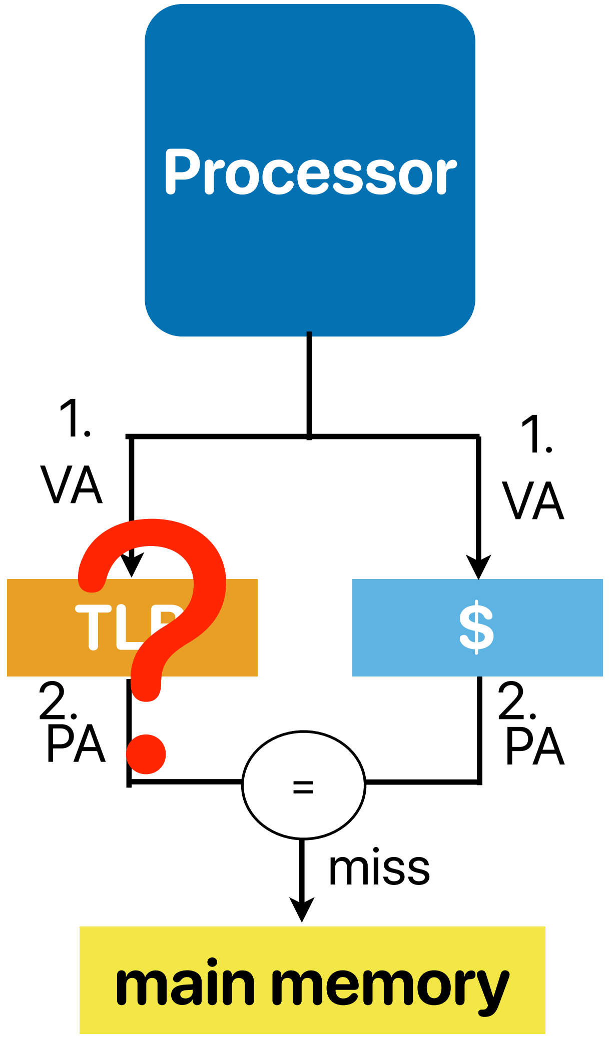


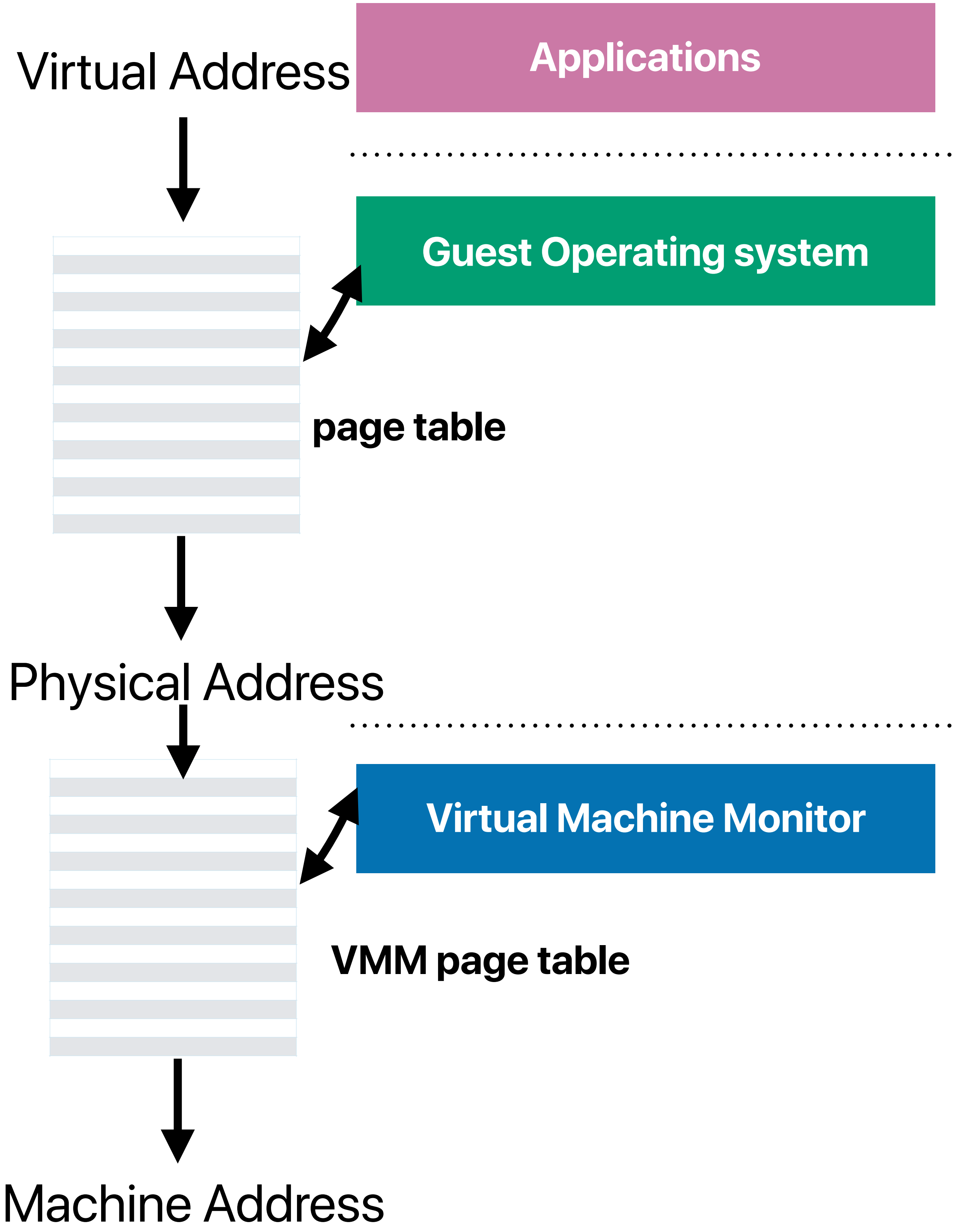
Recap: address translation with TLB

- 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

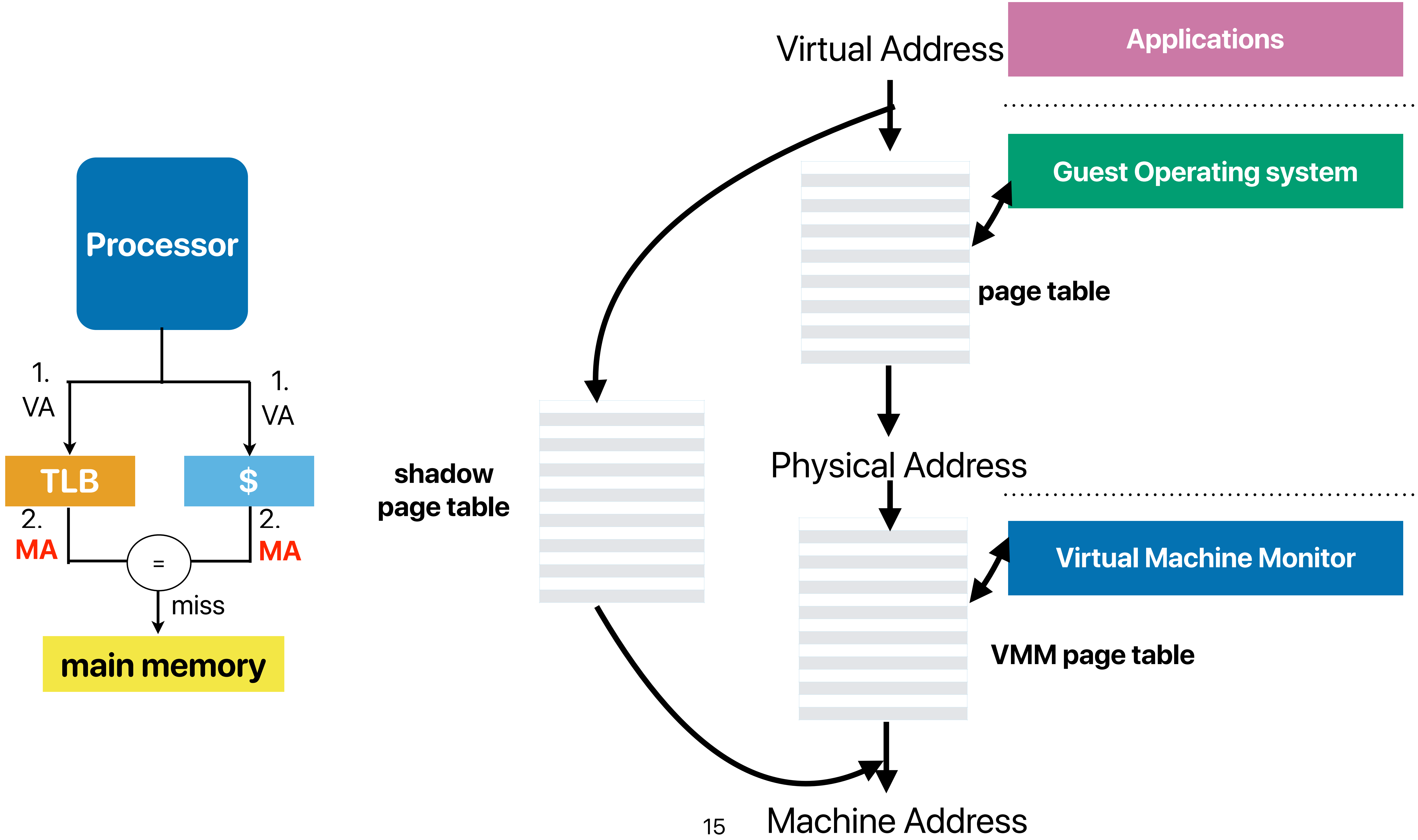


Address translation in VM





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

Why this doesn't work with x86

- The classical x86 architectures cannot allow the VMM to use the classical trap-and-emulation for virtualizing guest operating systems. How many of the following best describes the reasons?
  - ① The guest OS can be aware that it's not running in a privileged mode
  - ② A privileged instruction in the guest OS may not trigger a trap
  - ③ x86 does not provide a mechanism to set write-protected pages and handlers for tracing
  - ④ x86's hardware-walk hierarchical page table structure prevents the use of shadow page tables.

A. 0

B. 1

C. 2

D. 3

E. 4

- *Visibility of privileged state*. The guest can observe that it has been deprivileged when it reads its code segment selector (`%cs`) since the current privilege level (*CPL*) is stored in the low two bits of `%cs`.

- *Lack of traps when privileged instructions run at user-level*. For example, in privileged code `popf` ("pop flags") may change both ALU flags (e.g., ZF) and system flags (e.g., IF, which controls interrupt delivery). For a deprivileged guest, we need kernel mode `popf` to trap so that the VMM can emulate it against the virtual IF. Unfortunately, a deprivileged `popf`, like any user-mode `popf`, simply suppresses attempts to modify IF; no trap happens.

A Comparison of Software and Hardware Techniques for x86 Virtualization

Keith Adams and Ole Agesen
VMware

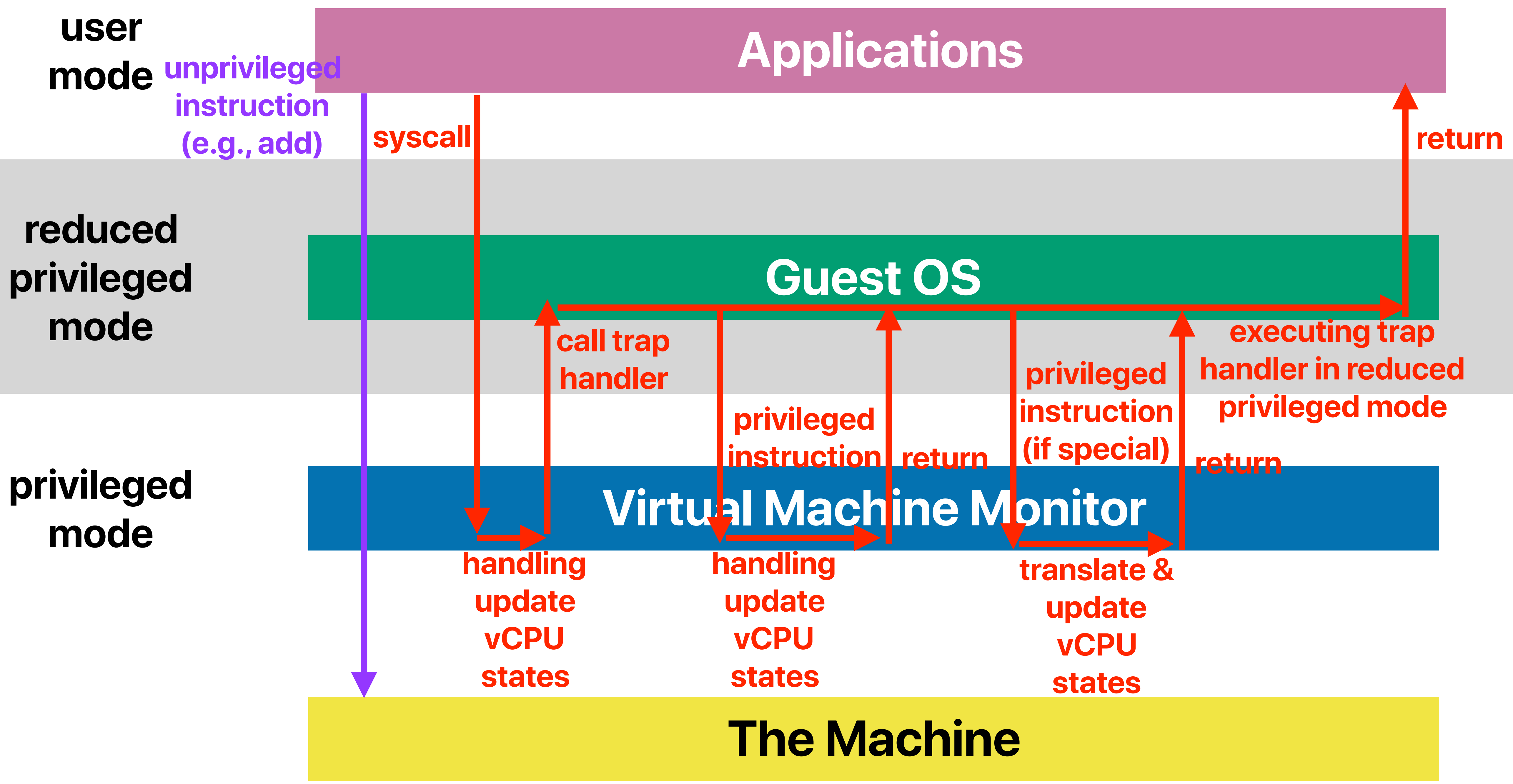
Binary translator

- 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

When to use hardware support for VM

- How many of the following situations can x86 VMX/VT-X instruction set extensions help improve the performance of VMM?
  ① Executing system calls
  ② Handling page faults
  ③ Modifying a page table entry
  ④ Calling a function
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Virtualization overhead

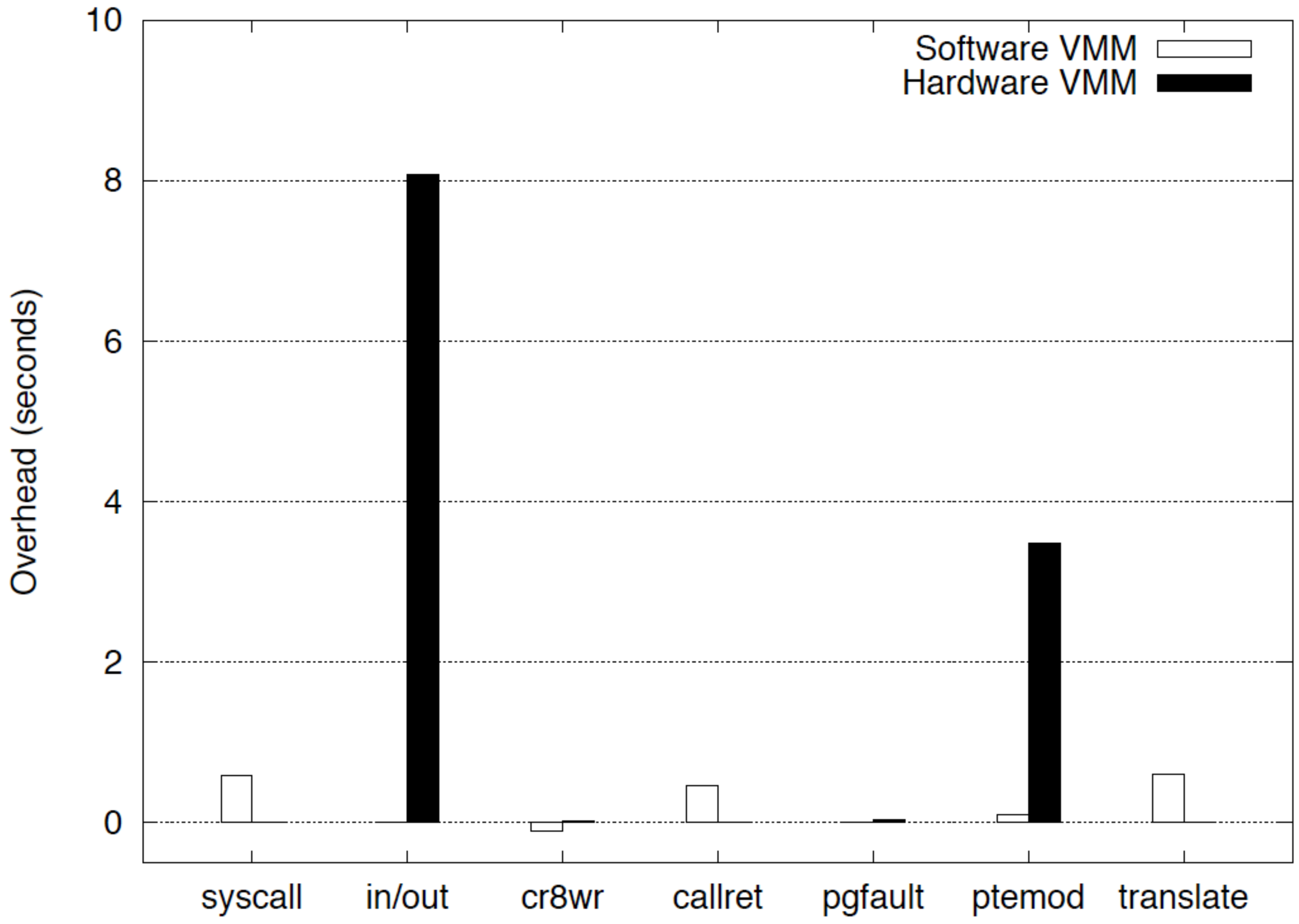


Figure 5. Sources of virtualization overhead in an XP boot/halt.

Nanobenchmarks

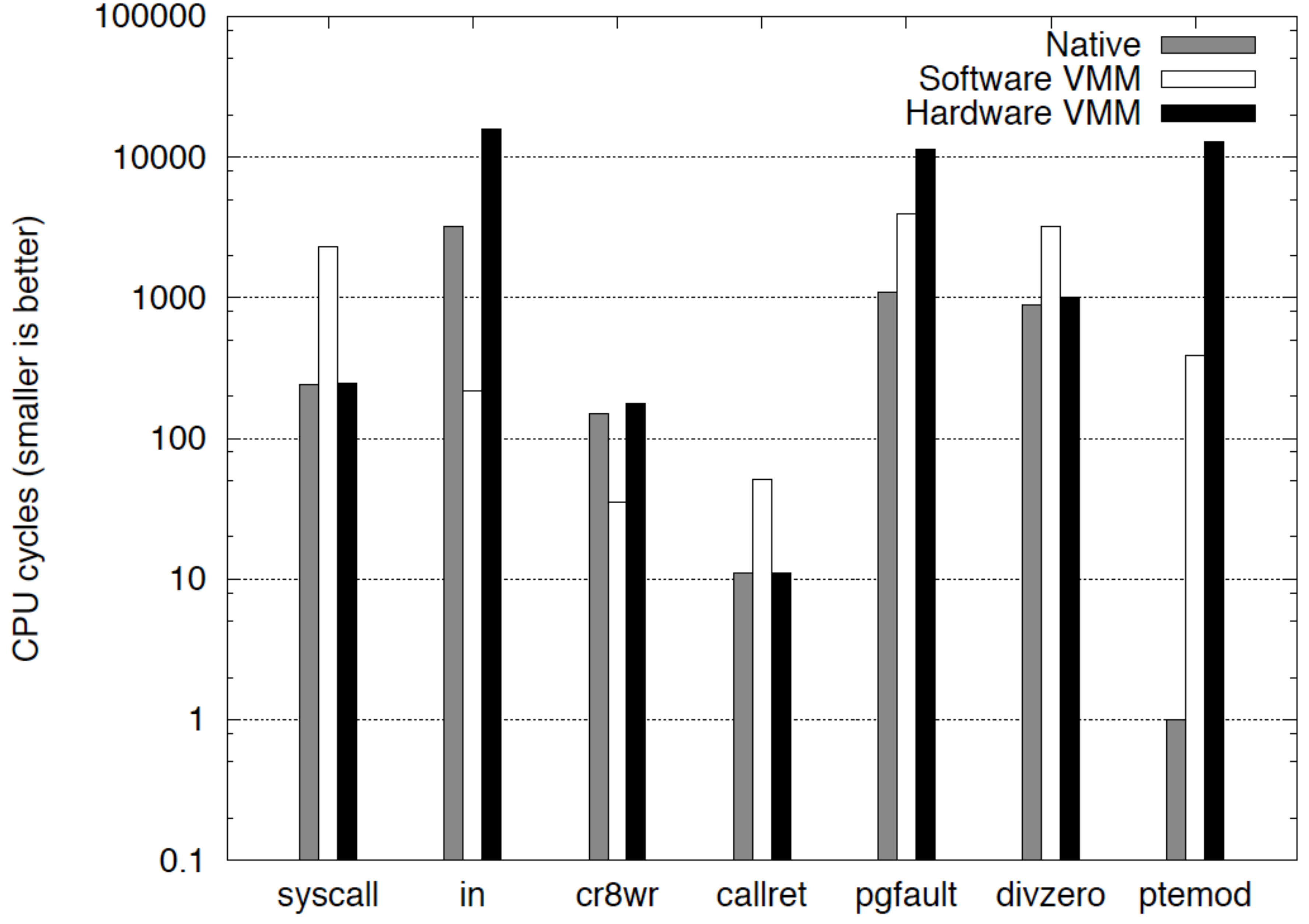


Figure 4. Virtualization nanobenchmarks.

Macrobenchmarks

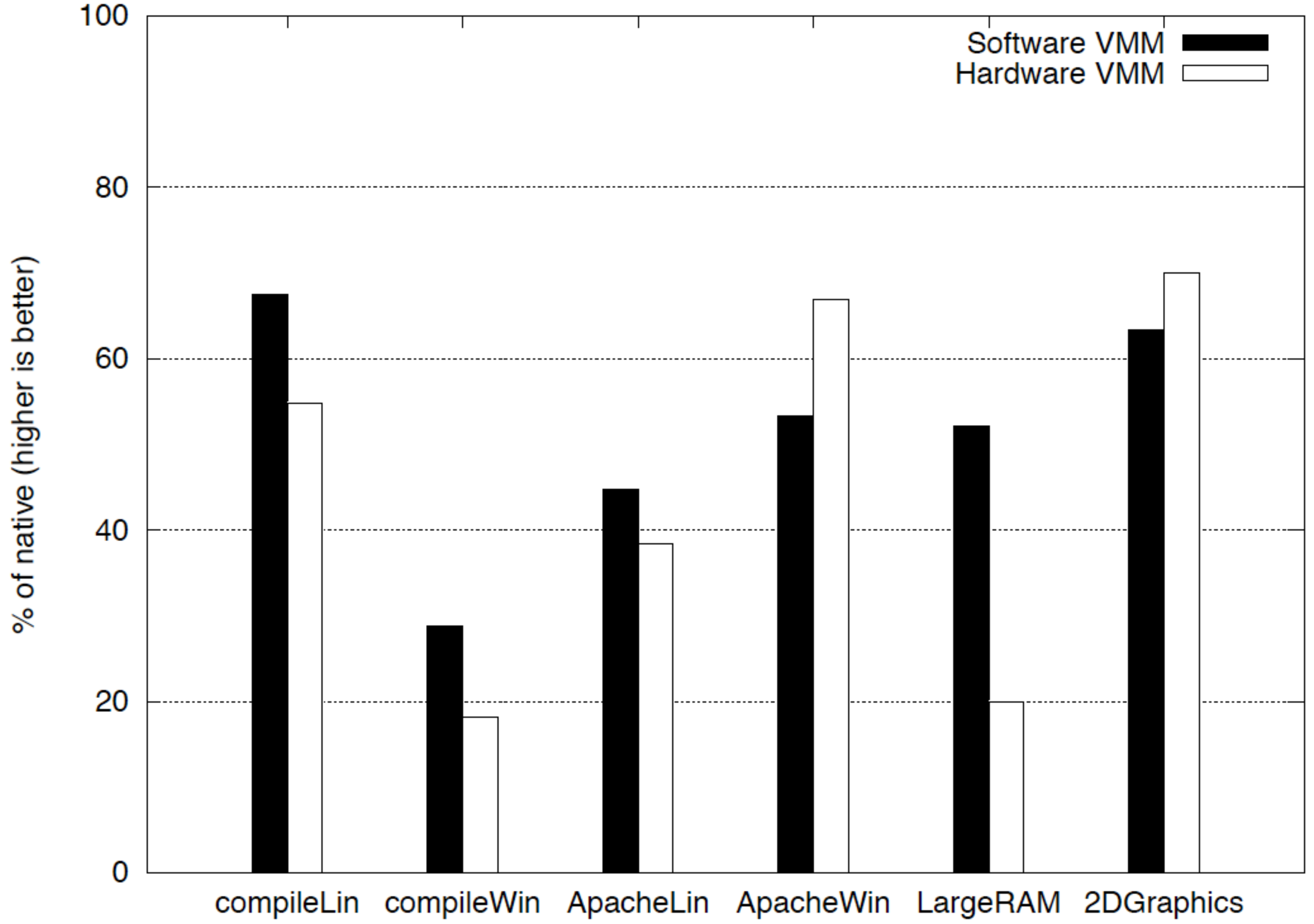


Figure 3. Macrobenchmarks.

When to use hardware support for VM

- How many of the following situations can x86 VMX/VT-X instruction set extensions help improve the performance of VMM?
  1. Executing system calls — guest OS runs in VM mode, no VMM intervention
  2. Handling page faults — software VMM doesn't need to use vmrun and exit
  3. Modifying a page table entry
  4. Calling a function — hardware VMM doesn't need BT

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

	3.8GHz P4 672	2.66GHz Core 2 Duo
VM entry	2409	937
Page fault VM exit	1931	1186
VMCB read	178	52
VMCB write	171	44

Table 1. Micro-architectural improvements (cycles).

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

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

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 Safety first	End-to-end
Interface	Do one thing well: Don't generalize Get it right Don't hide power Use procedure arguments Leave it to the client Keep basic interfaces stable Keep a place to stand	Make it fast Split resources Static analysis Dynamic translation	End-to-end Log updates Make actions atomic
Implementation	Plan to throw one away Keep secrets Use a good idea again Divide and conquer	Cache answers Use hints Use brute force Compute in background Batch processing	Make actions atomic Use hints

Cloud storage and Lampson's paper

- How many of the following cloud storage system represents the idea of "Separate normal and worst case"
  1. Facebook's f4
  2. Google's GFS
  3. Microsoft's Window Azure Storage
  4. NetApp's NFS

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

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

Final

Logistics

- Two of the questions are considered as comprehensive exam
- Final is cumulative
- Final exam will be online for any three hours you pick (starting from 3/14 8am — 3/15 4pm)
- If you help others, you're hurting yourself — since grades are given according to your relative rank in the class.

It’s a 3-hour long test

- Short answer questions **on papers** and Lampson’s paper * 10 — limits your answer to **20 words** — it’s a strict policy this time. You won’t get “any” point if you have more than 20 words
- Free answer question (2) — you need to know how to write code or what functions to use in designing a system module
- Brainstorming questions (4 or 5)— research questions, design decisions. Not actually a standard answer
  - Keep it short
  - If you’re asked to make a design decision, make sure you cover the following aspects
    - Why your choice makes sense to the problem asked/needs to be addressed
    - Why **other listed options are not competitive** as your choice

Short answer questions (20-word limit)

- What is thrashing? Which paper addresses this problem?
- What is saturation? Which paper addresses this problem?
- Which paper is about microkernel design?
- Which paper talks about capability?
- What’s TLB? What’s hardware-assisted TLB? What’s software-assisted TLB? Pros and cons for each? Which paper can you find the x86 TLB, page table design?
- What is garbage collection? Which paper uses that?
- What is free list? Which paper contains free list?

Short answer questions (20-word limit)

- Which paper is designing FS for read-intensive data access?
- Which paper is designing FS for write-intensive data access?
- Which paper is designing FS for MapReduce?
- Which paper talks about diskless system design? What design decisions they made?
- What are the three important properties that virtual machines need to hold?
- **Can you relate papers with Butler Lampson's "Hints for Computer System Design"?**

Reading more papers and see if …

- Xen and the Art of Virtualization
  - How to implement Balloon driver?
  - How to implement the circular queue for sharing devices in Xen?
- Simultaneous multithreading: maximizing on-chip parallelism
  - If you have an SMT processor like intel Core i7 or AMD RyZen, how does this architecture change your scheduling policies/mechanisms to maximize throughput?
- Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures
  - Giving a few reasons why Mach's VM is designed in this way?
  - Pros? Cons?

Programming

- How are multithreaded programs different from multi-process programs? How can we use them to generate desired results?
- How can we use semaphores to achieve similar goals of monitors?
- How is kernel programming different from user-space programming?
- If you want to implement binary translator in C on Linux, what functions do you need?

Brainstorming

- Revisiting ideas from papers you've read and think if those ideas work in modern scenarios
  - Segmentation
- If you're designing a file system for MapReduce, which of the file systems we learned in class will be a best fit? Why?
- If you can design a new interface for flash-based SSD, what this interface would look like?

Brainstorming

- What kind of application behavior is especially bad for flash SSDs? What kind of mechanism/optimization can you provide to mitigate that?
- What's the problem in the current Linux driver implementation in terms of system stability? What is potential solution to address that problem?
- What's log-on-log problem? Can you give an example of similar problems in modern system design?
- Comparison of UNIX FFS, LFS, GoogleFS, WAS. Under what scenario will you use each?

Announcement

- iEVAL
  - We highly value your opinions
  - Submit your screenshot of confirmation, equivalent to a full-credit reading quiz
- Check your grades on iLearn as soon as possible
  - We drop 2 of your lowest reading quizzes
  - We allow 4 absences through out the whole quarter
  - Midterm grade is up. One week regrading policy applies — check the website regarding how to initiate that
  - “Weighted Total” is your current total.

Thank you all for this great quarter!