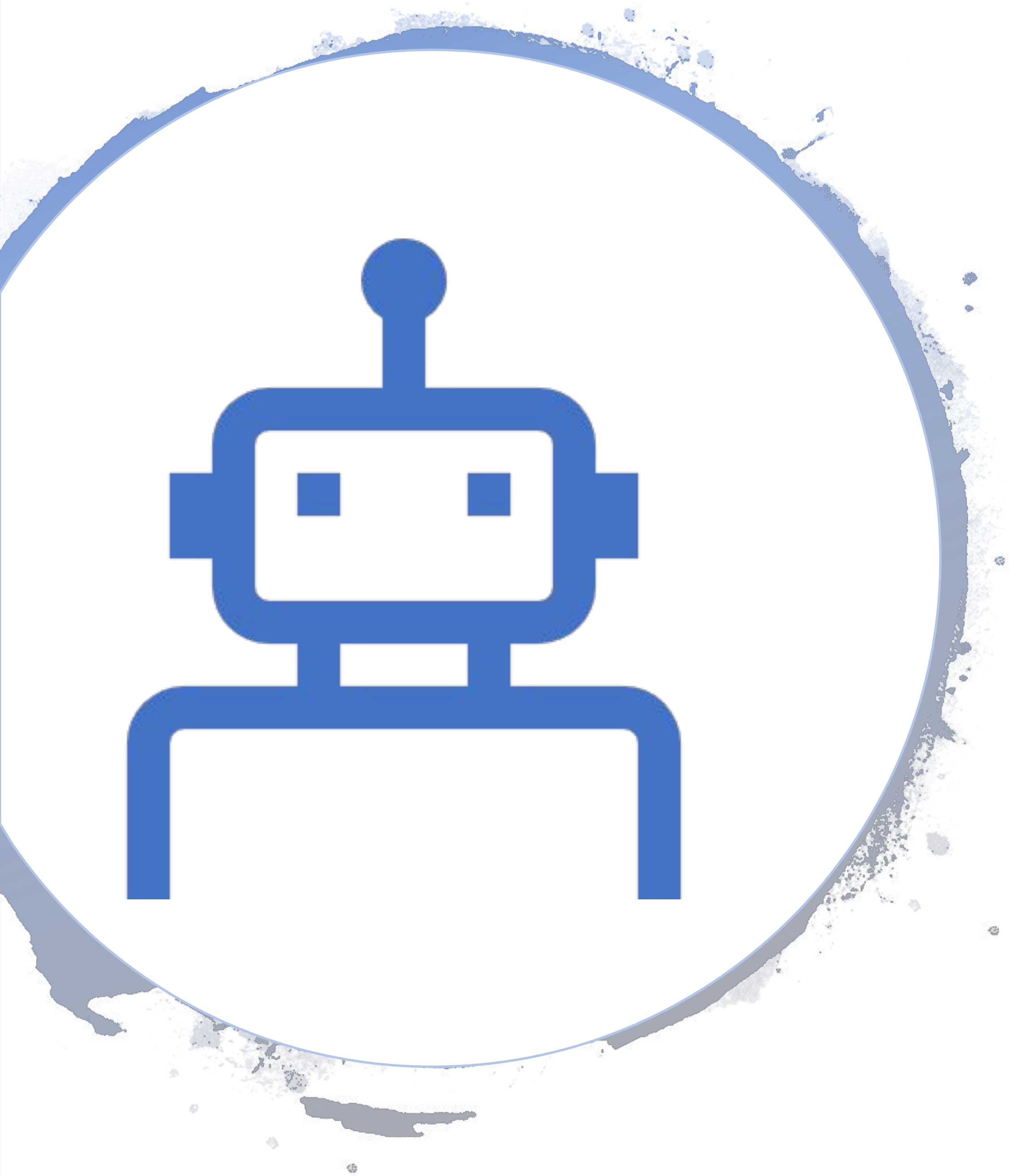


CS323

Week 12
Virtual Machines

Laurent BINDSCHAEDLER
Calin IORGULESCU

May 22, 2019

Outline

- Virtualization
- Virtual Machines
- Virtual Machine Monitor (VMM)
- VMM Construction
  - Direct execution
  - Feasibility (Popek-Goldberg, x86)
  - Address translation
- Summary

Virtualization

Principle of Indirection

"Any problem in computer science can be solved with another layer of indirection. But that usually will create another problem."

- David Wheeler

Virtualization is an instance of indirection, specifically layering.

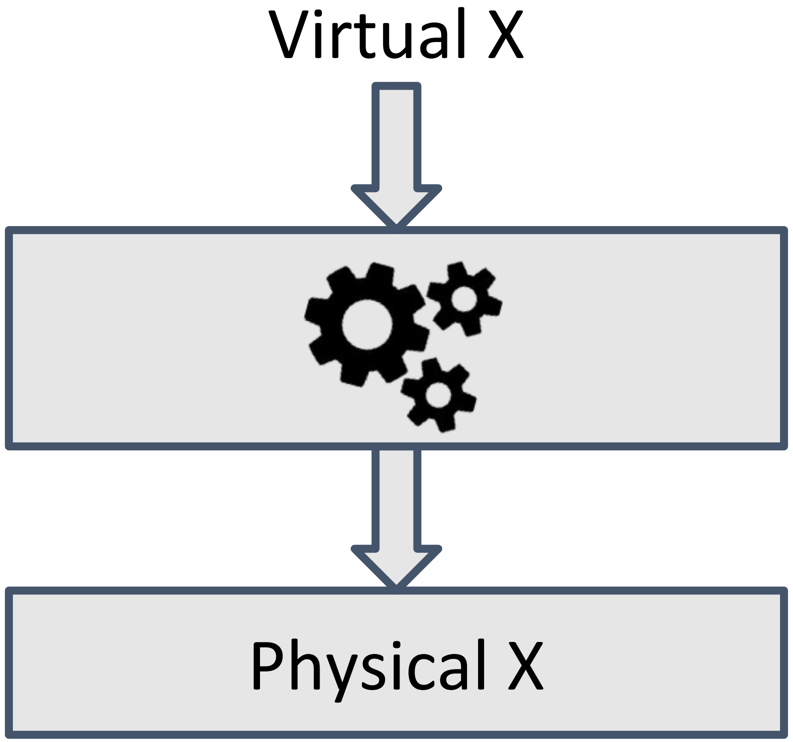
Caution:

- Virtual != imaginary
- Virtual is an overloaded term

Virtualization

A layer that exports the same abstraction as the layer it relies upon

- Provides isolation by hiding the physical names of underlying resources
- Enforces modularity



Q: Can you give examples of Virtualization?

Virtualization Examples

Threads as Virtual CPUs

- Abstraction (x86 instruction set)
- Physical resource: core or hyper thread
- OS scheduler allows
    - Physical core used by a thread to change
    - More threads than physical cores

Virtualization Examples

Virtual Memory

- Abstraction: byte-addressable content
- Physical resource: random access semiconductor memory
- Hiding actual memory location allows
  - Memory location to not be present at all (see paging)
  - Memory location to change transparently (see COW)

More Virtualization Examples

In operating systems:

- Sockets, pipes = Virtual links
- RAID volumes = Virtual disks

Elsewhere:

- Data / database virtualization
- Virtual Private Networks (VPN), VLANs
- ...

Virtualization Mechanisms

Multiplexing

Expose one resource as multiple virtual entities

Aggregation

Make multiple resources appear as one virtual resource

Emulation

Make a resource appear as a different type of resource

Virtualization by Multiplexing

Expose one resource as multiple virtual entities

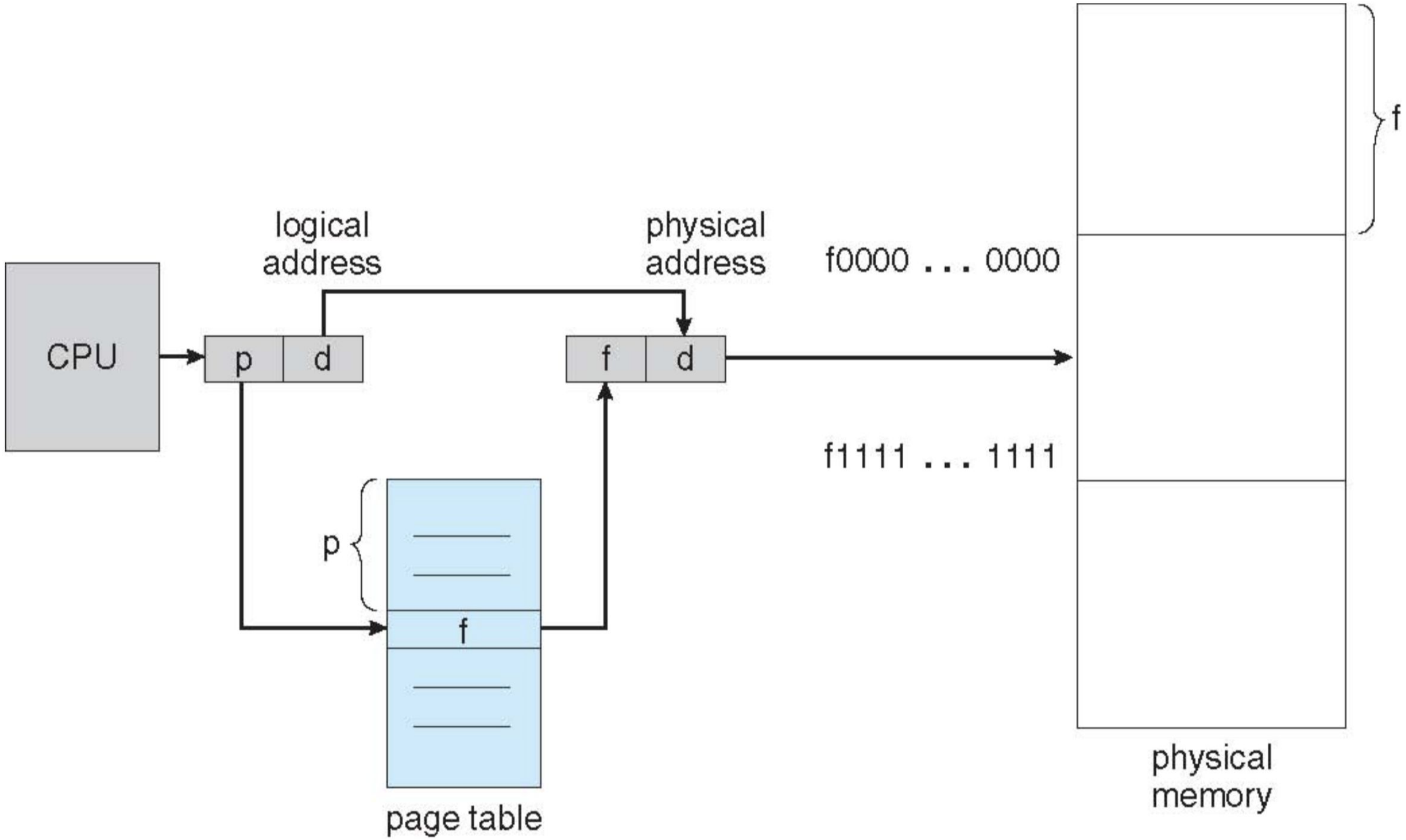
- Each entity appears to have the full resource
- Applies in space and/or time

Indirection hides physical names

Often relies on hardware-based mechanism, e.g.,

- Virtual memory uses the MMU
- Registers saved/restored on trap

Virtualization by Multiplexing – Virtual Memory

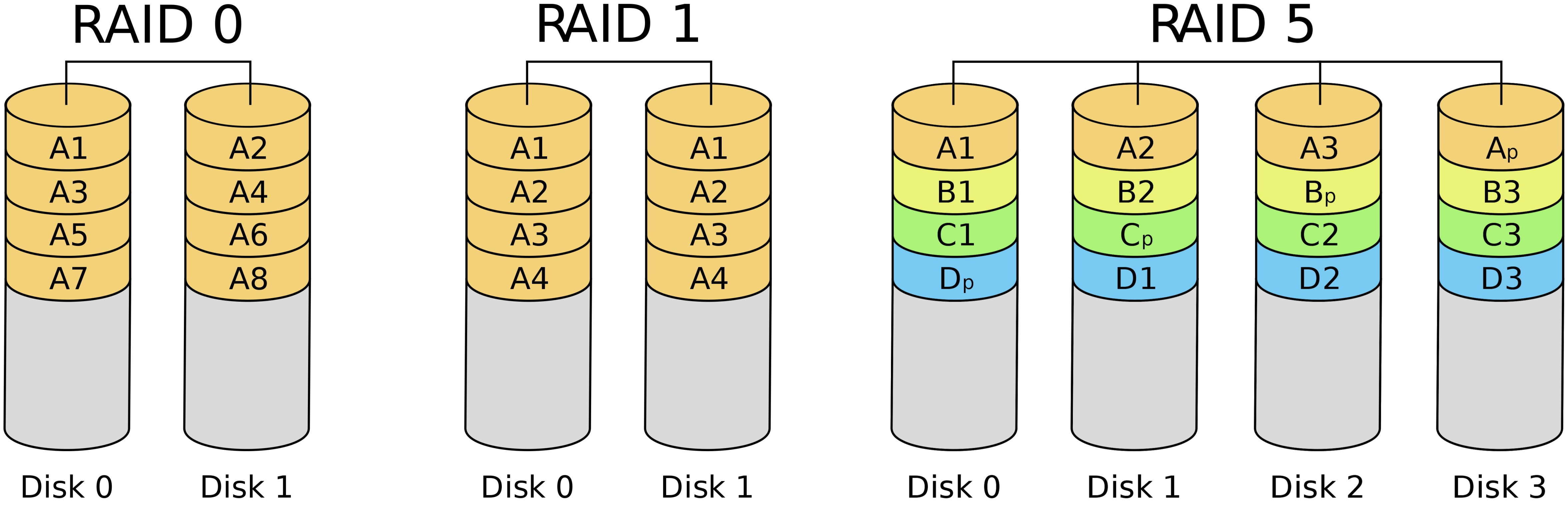


Virtualization by Aggregation

Aggregate multiple resources into one single virtual resource

- Virtual resource typically has enhanced properties
  - More capacity
  - Better availability
  - Redundancy

Virtualization by Aggregation – RAID



Expose multiple disks as one virtual disk with more capacity and/or availability

Virtualization by emulation

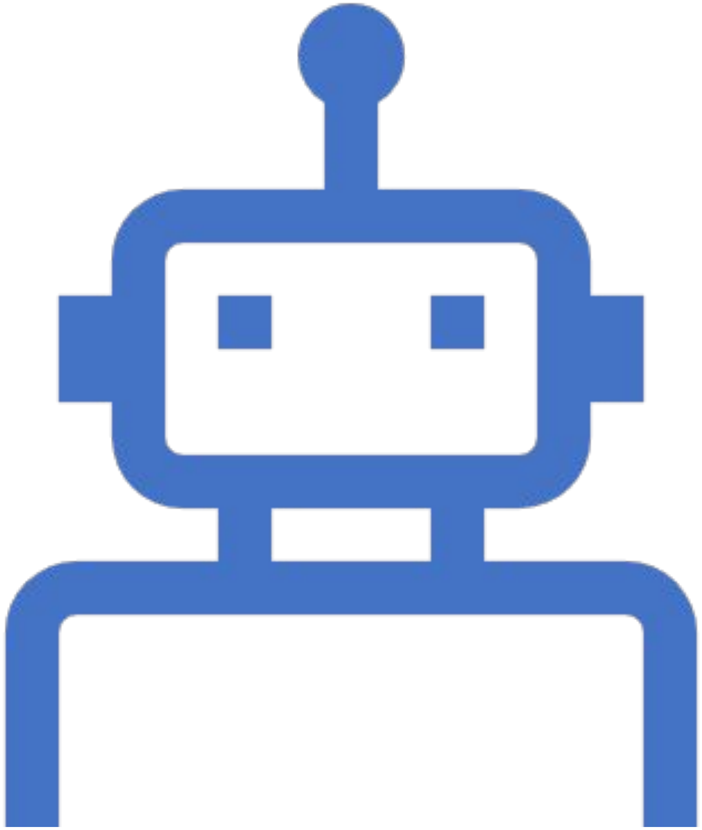
Use software to emulate a virtual resource which is different from the underlying physical resource

- Very useful in some cases, e.g., backwards compatibility

	Physical resource	Virtual resource
RAM disk	Memory	Disk
Virtual tape	Disk	Tape drive
Java Virtual Machine	x86 core	Java bytecode processor

Virtualization by emulation – Android Emulator

Facilitate development of Android apps by running them on your dev machine

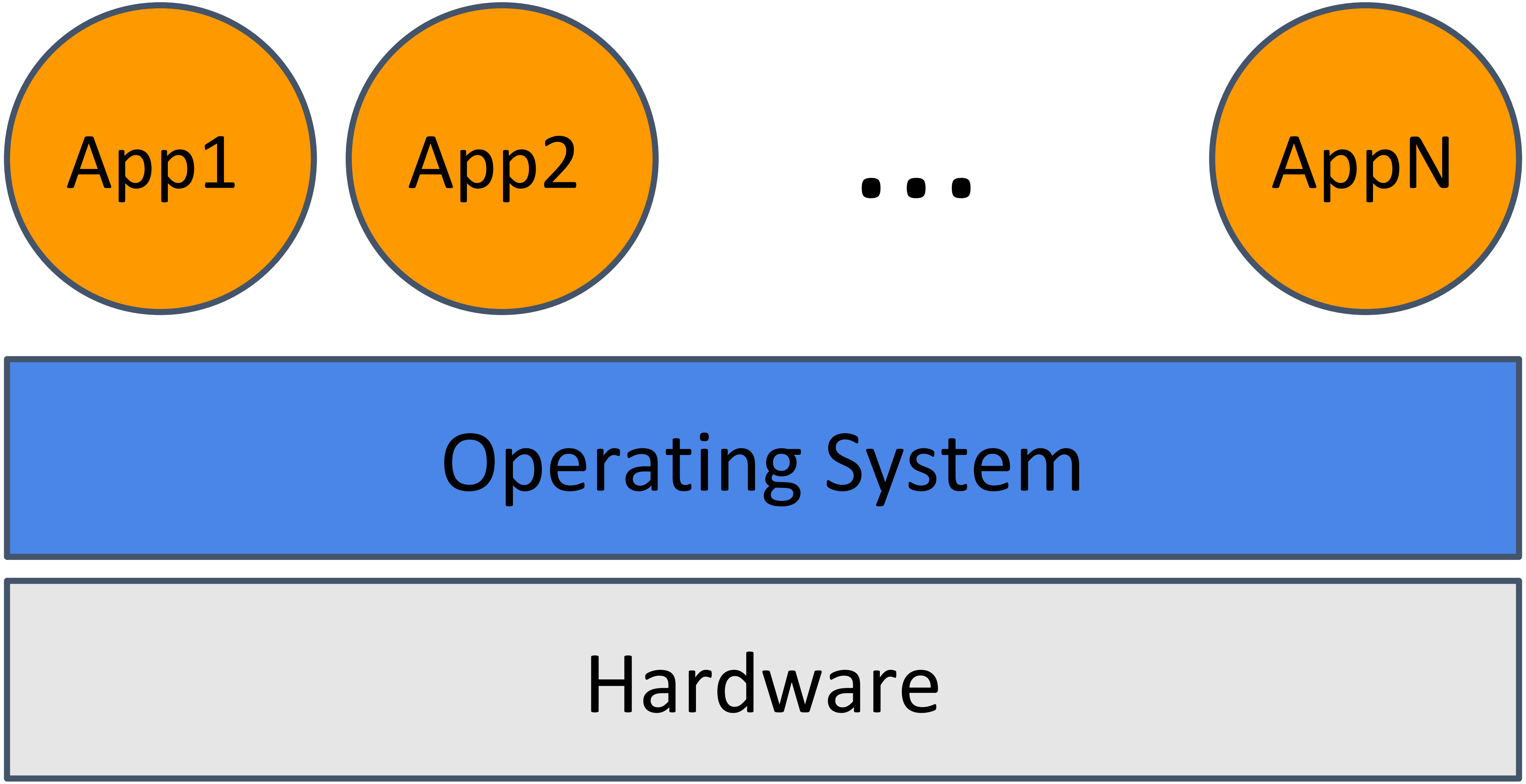


Virtual Machines

Virtual Machines

Virtualization applied to an entire computer.

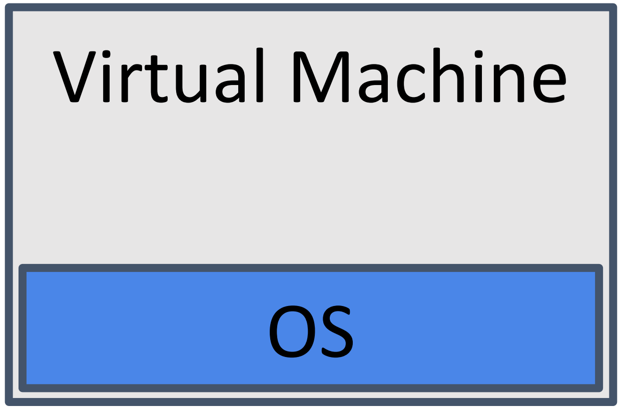
Reminder – Operating System



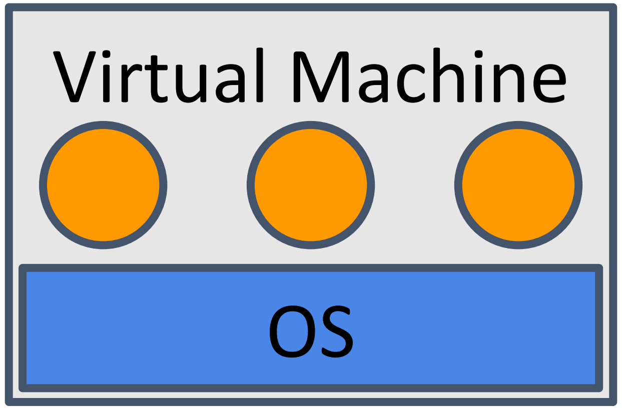
Virtual Machines

Virtual Machine

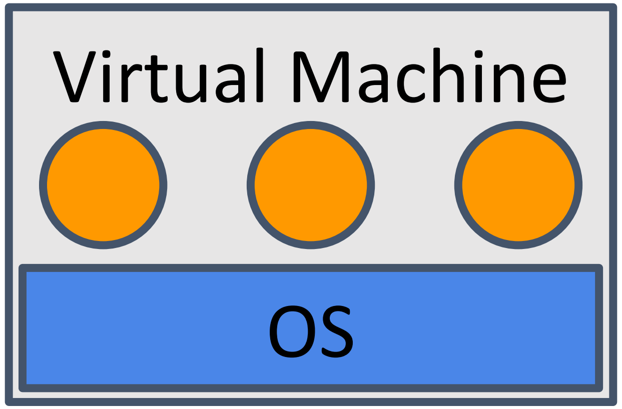
Virtual Machines



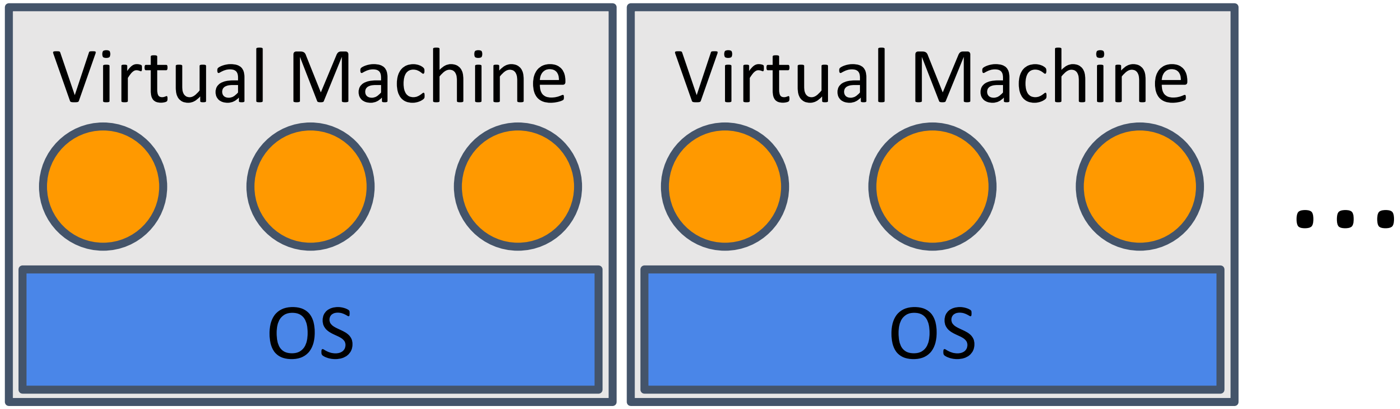
Virtual Machines



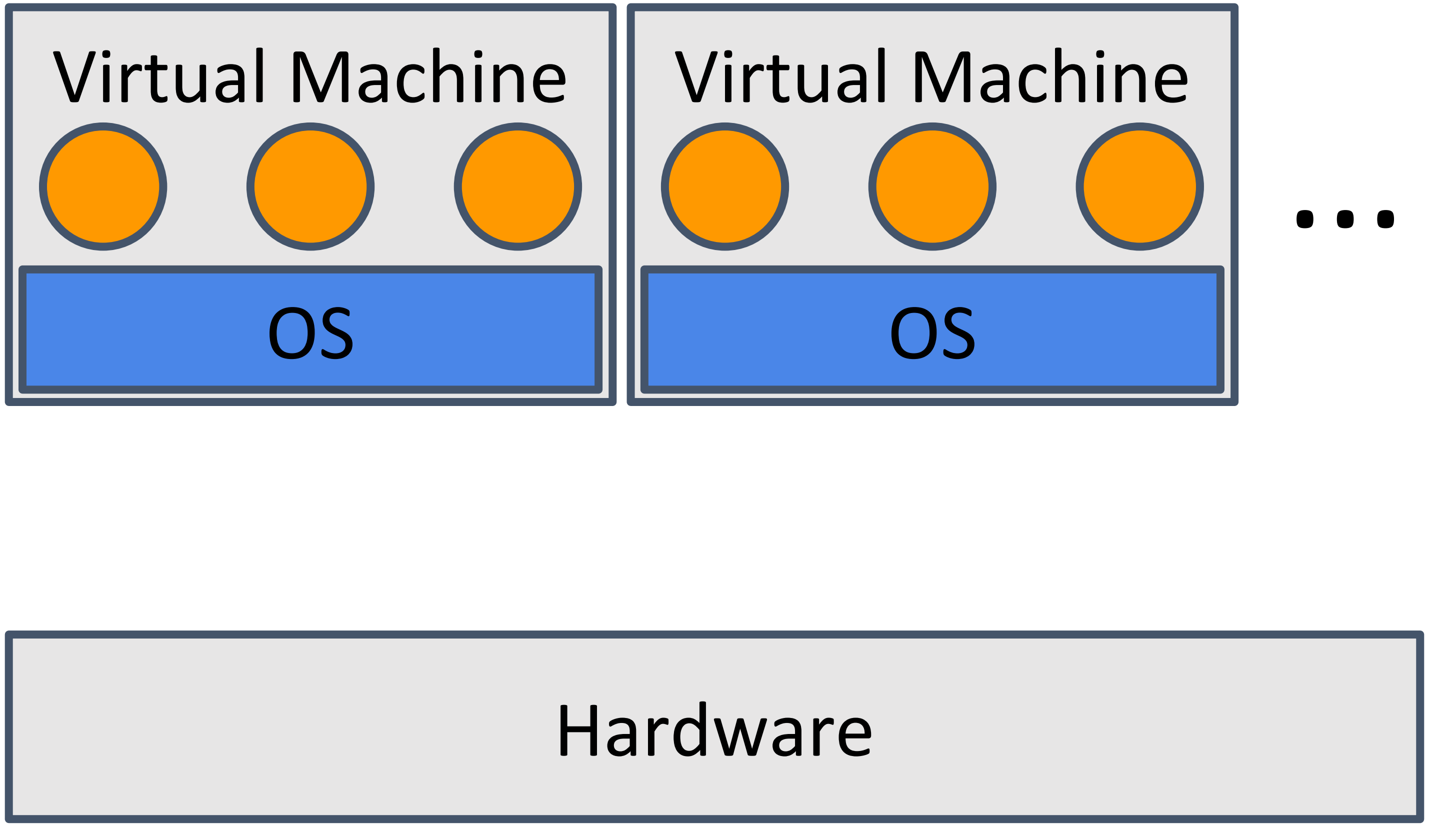
Virtual Machines



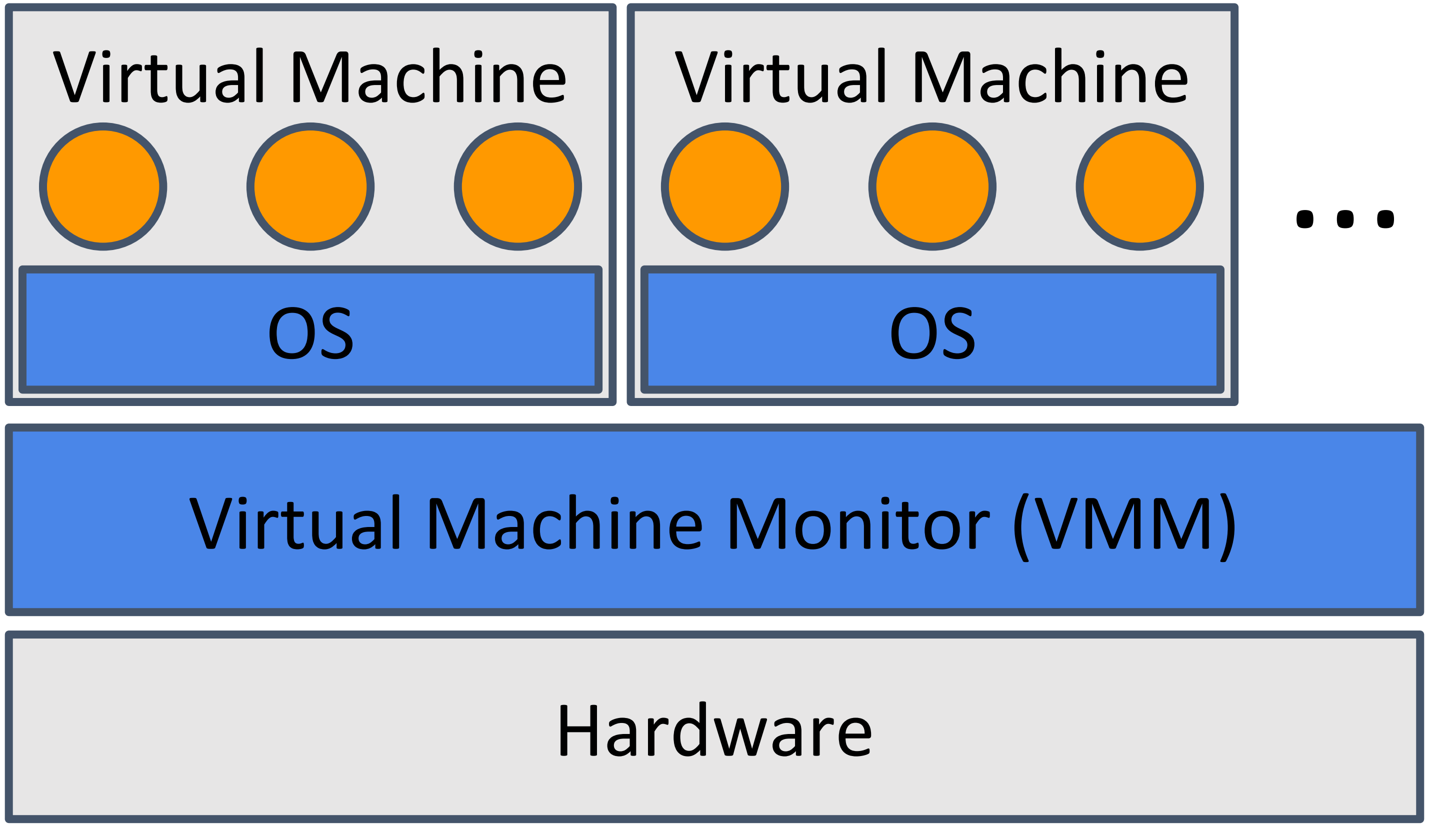
Virtual Machines



Virtual Machines



Virtual Machines



Virtual Machine

OS

Virtual Machine

OS

…

Virtual Machine Monitor (VMM)

Hardware

Virtual Machines

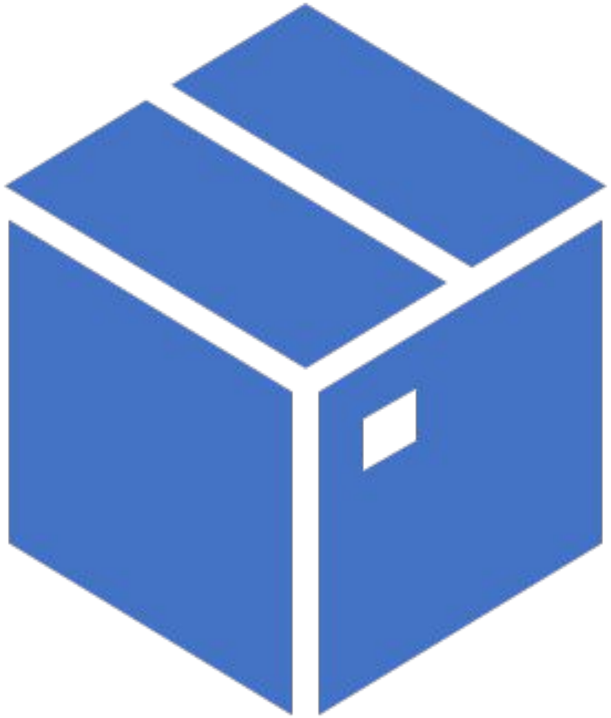
Virtualization applied to an entire computer.

A Virtual Machine (VM) is an abstraction that is sufficiently equivalent to the underlying hardware that it can run an operating system.

- In particular, the OS within the VM can itself run multiple applications

i.e., a VM is an efficient, isolated duplicate of the real machine.

VM abstractions are materialized using a piece of software called a Virtual Machine Monitor (VMM), sometimes also known as Hypervisor.



Virtual Machine Monitor

Virtual Machine Monitor (VMM)

Resource manager for VMs

Provides

- Creation, destruction, and scheduling of VMs
- Memory management for VMs
- Disk management for VMs
- I/O management for VMs

Similar to what an OS does, with VMs instead of processes

Current VMM Tools

Type I architecture – VMM is the host operating system (a.k.a. hypervisor)

- Xen (open-source)
- VMware vSphere / ESXi
- Microsoft Hyper-V

Type II architecture – VMM separate from host OS

- KVM (Linux host) (open-source)
- VMware Workstation (Windows host) and Fusion (OS X host)
- Parallels (Windows and OS X hosts)

Terminology

VM = Virtual Machine
Guest OS = operating system running in the VM
Host OS = operating system running on the "metal" (i.e. not in the VM)
VMM = Virtual Machine Monitor
Hypervisor = VMM that is also a host OS (a.k.a. type I VMM)
Hosted VMM = VMM that runs on a separate host OS (a.k.a. type II VMM)

Virtual Machine Monitor – Requirements

Equivalence: the virtual hardware needs to be sufficiently equivalent to the underlying hardware that you can run the same software in the virtual machine that you would normally run on the computer

- In particular, you can run the same (unmodified) operating system

Safety: VM must be completely isolated from other VMs and from the VMM

- i.e., you can think of the VM as running on its own dedicated hardware

Performance: the overhead of virtualization must be sufficiently low that a VM can be used in the same way as if it was running on the hardware

- i.e. virtualization slowdown must not have a significant impact on execution

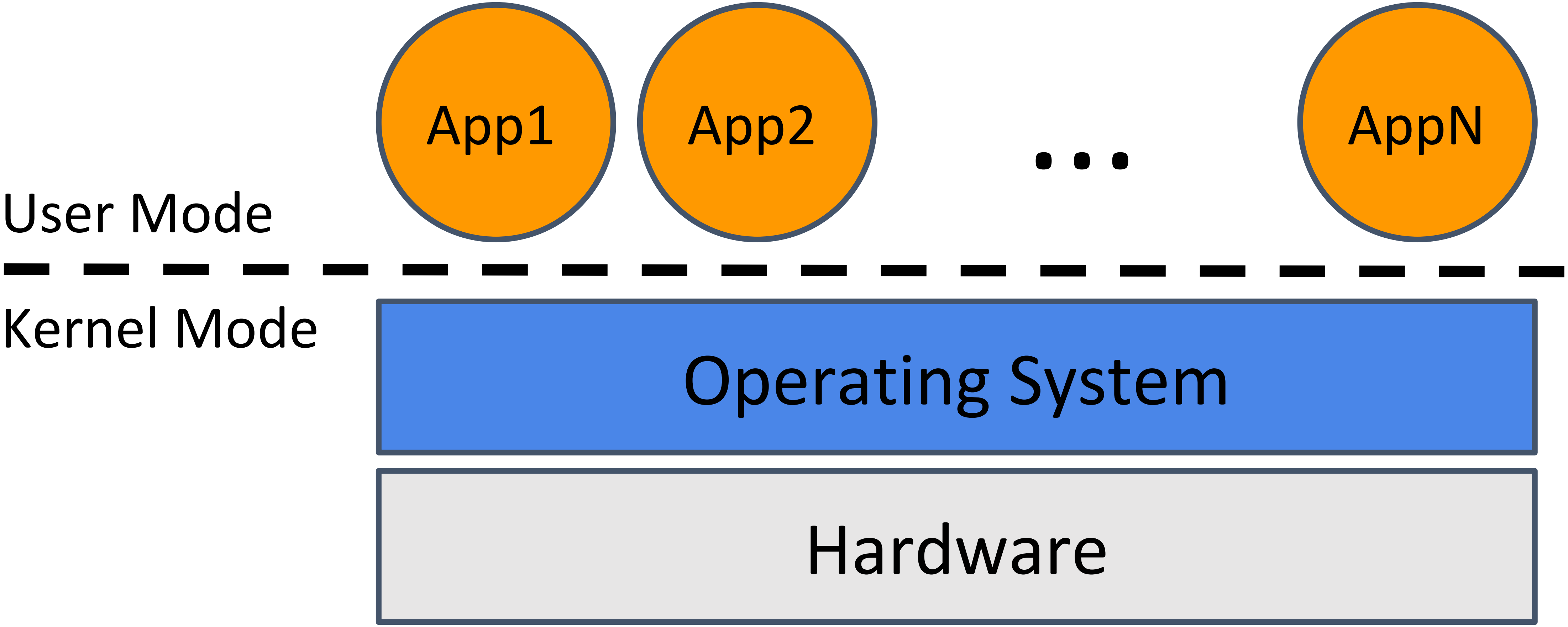
Q: Is it always possible to build a VMM?

Intuition – Guest OS must be protected

3 fundamental requirements for a protected OS:

- User / kernel mode bit
- Virtual memory
- Trap architecture

OS Requirement 1: User/Kernel Mode



OS Requirement 2: Virtual Memory

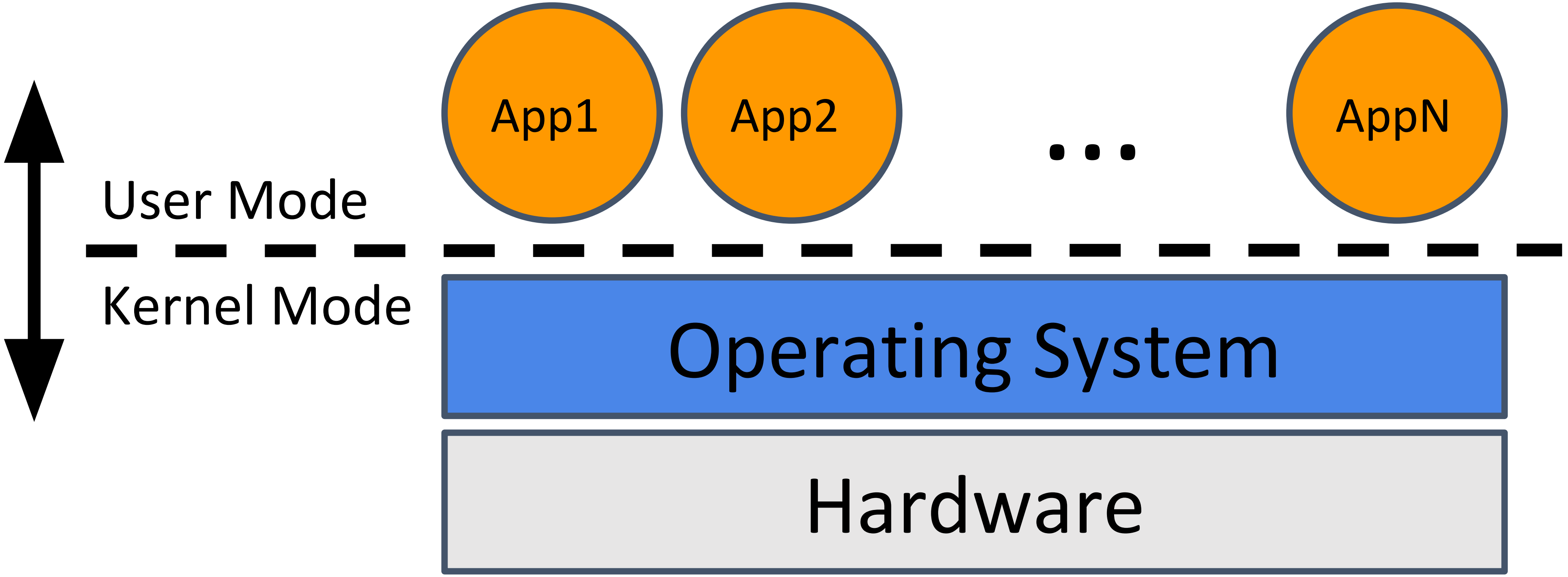
Level of indirection between

- Address space used by the CPU (virtual memory addresses)
- Underlying physical memory (physical memory addresses)

Protects OS memory from applications

OS Requirement 3: Trap Architecture

OS must be able to take traps and return from them

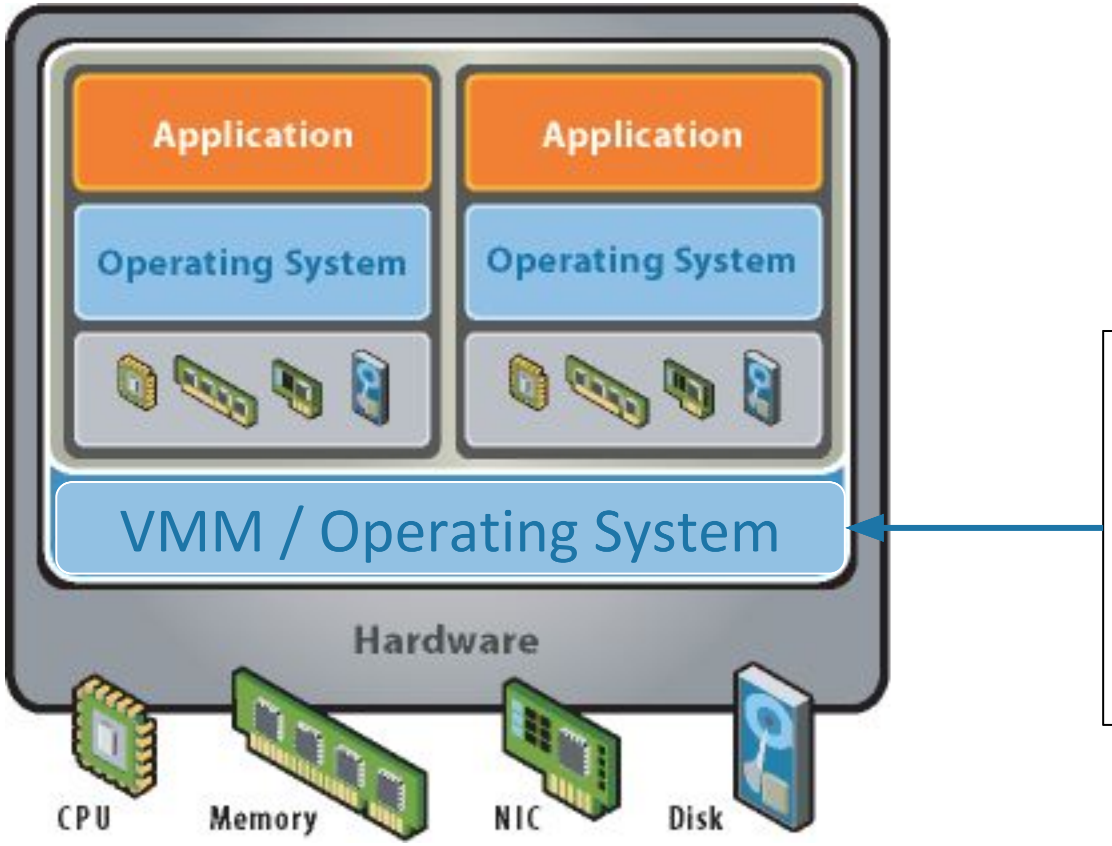


VMM Construction

VMM: goals

- **Hardware multiplexing**
    - multiple VMs can access underlying hardware
- **Isolation**
    - VMs <u>cannot</u> interfere with one another, or hog resources
- **Low overhead**
    - most of the resources must go to the VMs

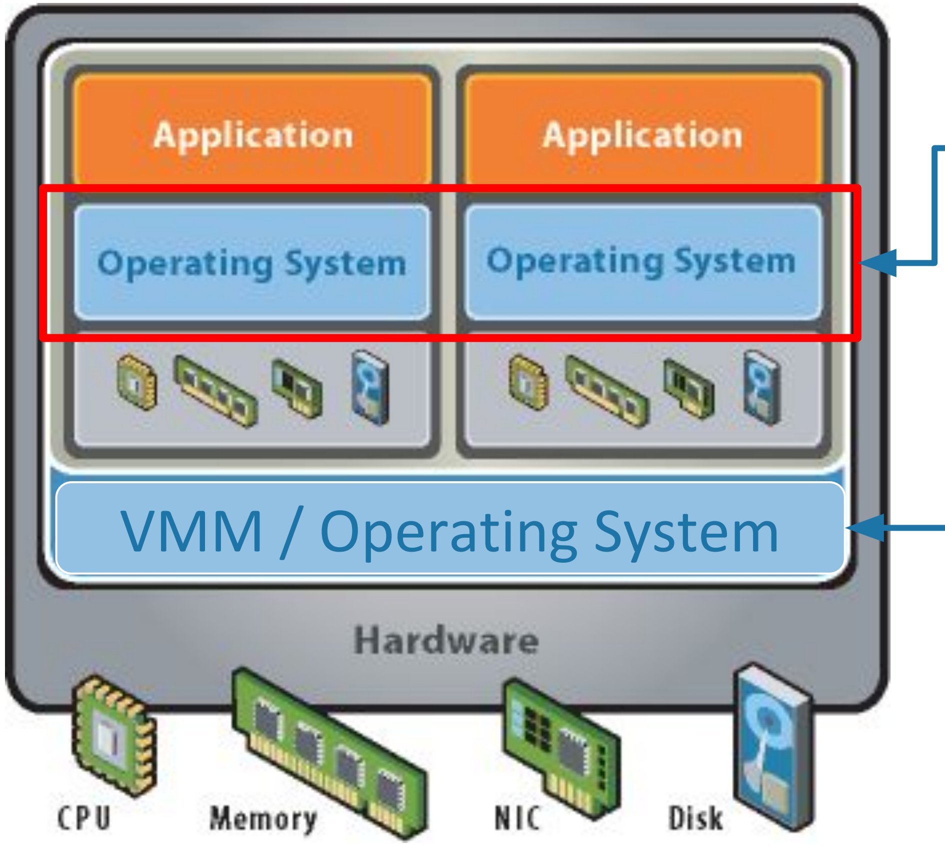
Reminder: Virtual Machines



VMM types:

- **Type 1: VMM == OS**
  - e.g., Xen, Microsoft HyperV
- **Type 2: VMM runs on top of OS**
  - e.g., KVM, VMware Fusion, etc.

Reminder: Virtual Machines



Each OS thinks it runs directly on the hardware!

- the VMM must make sure the OS <u>cannot</u> deduce it runs in a VM

VMM types:

- **Type 1: VMM == OS**
  - e.g., Xen, Microsoft HyperV
- **Type 2: VMM runs on top of OS**
  - e.g., KVM, VMware Fusion, etc.

Virtual Hardware

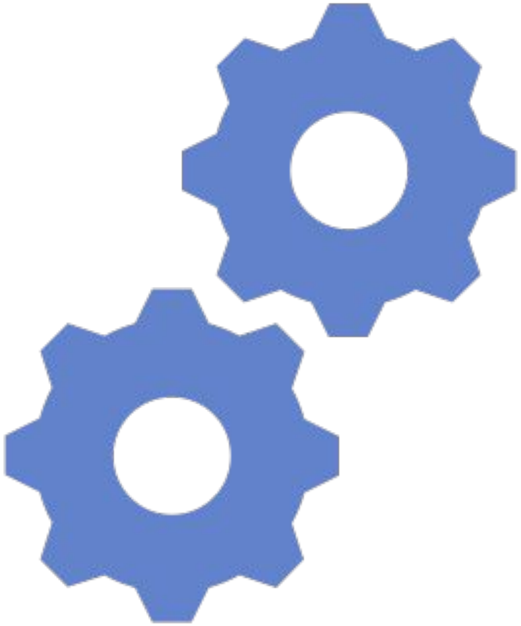
Virtual Machine must behave as if they are running on “real hardware”

- CPU
    - each VM has its own **Virtual CPU**
- Physical Memory
    - each VM has its own **Virtual “Physical Memory”**
- I/O
    - each VM has its own **Virtual Disk** and other peripherals
- Network
    - VMs can communicate with real and virtual machines through the **Virtual Network**

Virtual Hardware

Virtual Machine must behave as if they are running on “real hardware”

- CPU
  - each VM has its own **Virtual CPU**
- Physical Memory
  - each VM has its own **Virtual “Physical Memory”**
- I/O
  - each VM has its own **Virtual Disk** and other peripherals
- Network
  - VMs can communicate with real and virtual machines through the **Virtual Network**



Virtual CPU

Virtual CPU

- What architecture can the Virtual CPUs have?

	Different than the Physical CPUs	**Same as Physical CPUs**
Example scenario	e.g., develop code for other devices: ARM, PlayStation, etc.	e.g., run different OSes on same machine without needing to restart
Mechanism used	Dynamic Binary Translation	Limited Direct Execution

Virtual CPU

- What architecture can the Virtual CPUs have?

	Different than the Physical CPUs	**Same as Physical CPUs**
Example scenario	e.g., develop code for other devices: ARM, PlayStation, etc.	e.g., run different OSes on same machine without needing to restart
Mechanism used	Dynamic Binary Translation	Limited Direct Execution

Dynamic Binary Translation

- Change one type of instructions to another
- Example: for `a = 5` and `b = 6`, compute `c = a + b`

Guest: ARM 7

```
mov r3, #5
str r3, [fp, #-8]
mov r3, #6
str r3, [fp, #-12]
ldr r2, [fp, #-8]
ldr r3, [fp, #-12]
add r3, r2, r3
str r3, [fp, #-16]
```

Host: Intel x86

```
movl     $0, -4(%ebp)
movl     $5, -8(%ebp)
movl     $6, -12(%ebp)
movl     -8(%ebp), %ecx
movl     -12(%ebp), %edx
addl     %edx, %ecx
movl     %ecx, -16(%ebp)
```

Dynamic Binary Translation

- Change one type of instructions to another
- E

Pros	Cons
Applicable for most architectures No special hardware support needed	Typically **very high overhead**

```
mov r3, #5
str r3, [fp, #-8]
mov r3, #6
str r3, [fp, #-12]
ldr r2, [fp, #-8]
ldr r3, [fp, #-12]
add r3, r2, r3
str r3, [fp, #-16]
```

```
movl      $0, -4(%ebp)
movl      $5, -8(%ebp)
movl      $6, -12(%ebp)
movl      -8(%ebp), %ecx
movl      -12(%ebp), %edx
addl      %edx, %ecx
movl      %ecx, -16(%ebp)
```

Dynamic Binary Translation

- Change one type of instructions to another
- E

Pros	Cons
• Applicable for most architectures • <u>No special hardware</u> support needed	• Typically <u>**very high overhead**</u>

```
mov r3, #5
str r3, [fp, #-8]
mov r3, #6
st
ld
ldr r3, [fp, #-12]
add r3, r2, r3
str r3, [fp, #-16]
```

```
movl     $0, -4(%ebp)
movl     $5, -8(%ebp)
movl     -12(%ebp), %edx
addl     %edx, %ecx
movl     %ecx, -16(%ebp)
```

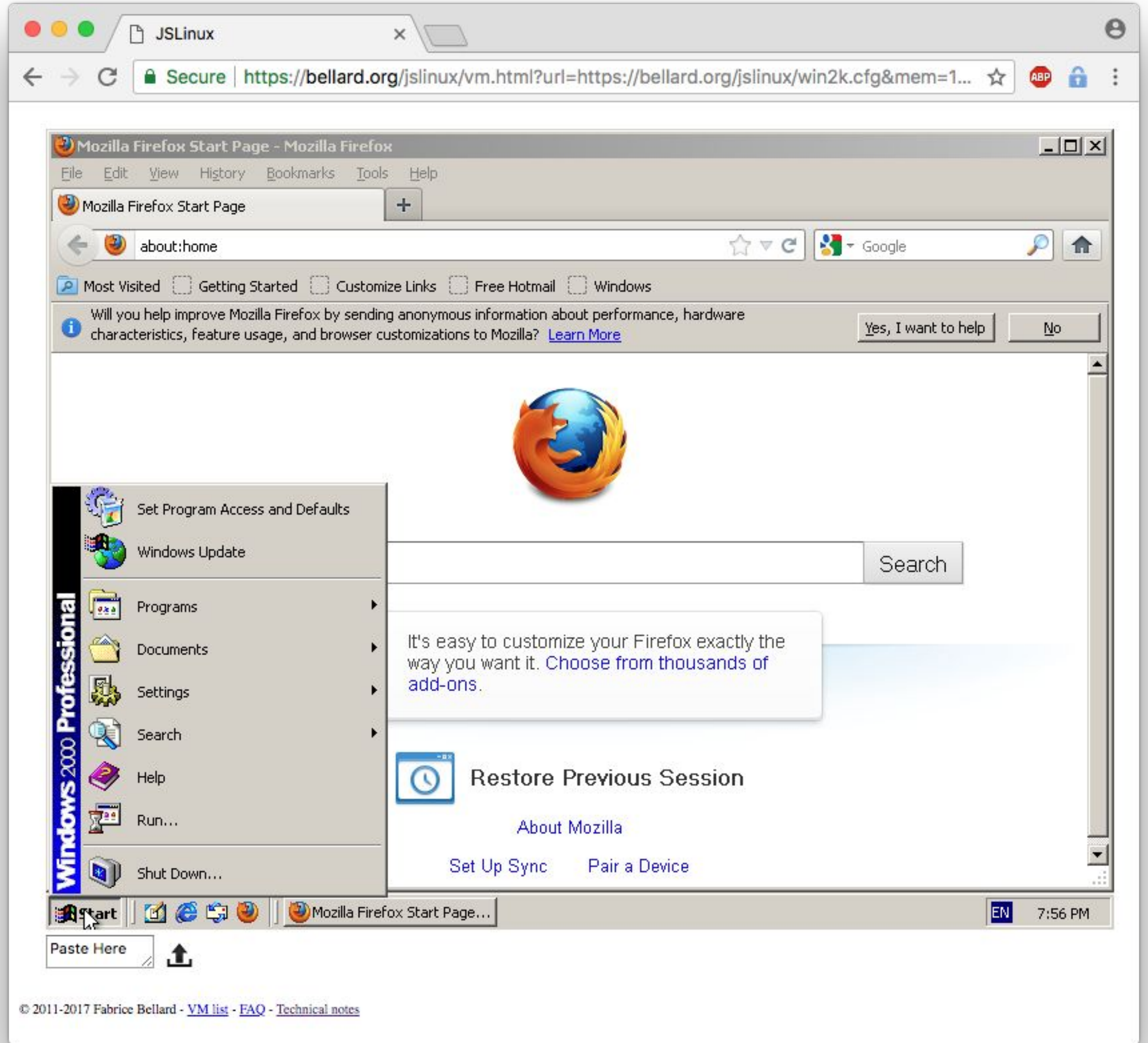
Can also translate same-architecture instructions into different ones!

Example: JSLinux -- a VM in your browser!

Chrome on **OS X**...

… running **Windows 2000** ...

… running **Firefox**!



https://bellard.org/jslinux/vm.html?url=https://bellard.org/jslinux/win2k.cfg&mem=1...

Will you help improve Mozilla Firefox by sending anonymous information about performance, hardware characteristics, feature usage, and browser customizations to Mozilla? Learn More

It's easy to customize your Firefox exactly the way you want it. Choose from thousands of add-ons.

© 2011-2017 Fabrice Bellard - VM list - FAQ - Technical notes

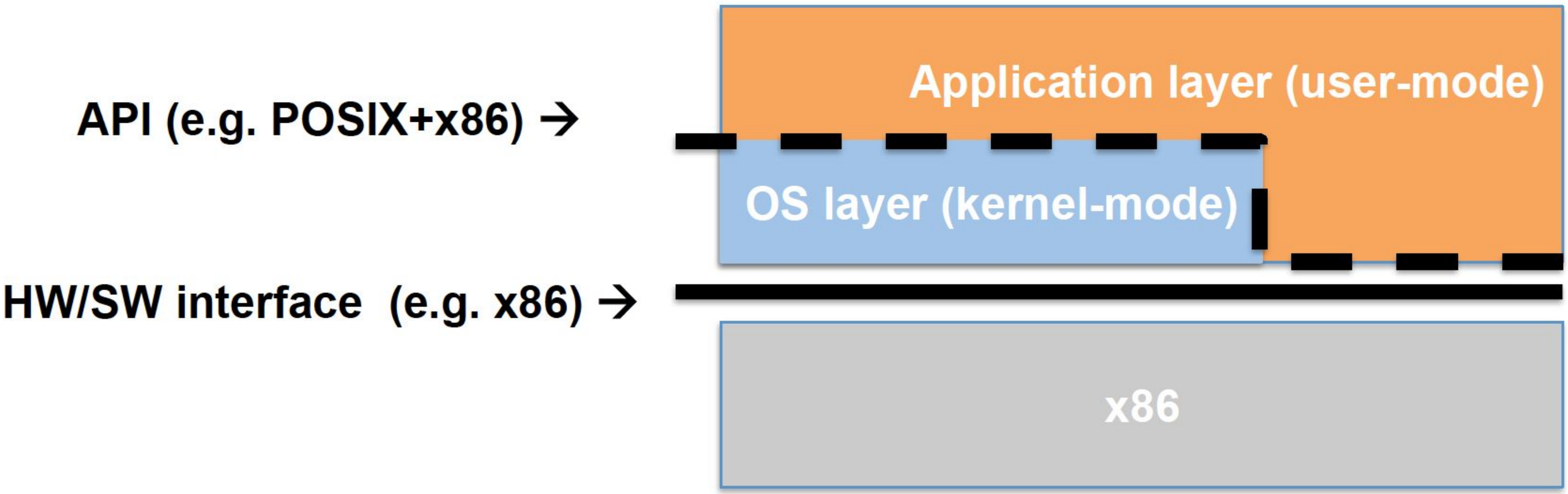
Virtual CPU

- What architecture can the Virtual CPUs have?

	Different than the Physical CPUs	**Same as Physical CPUs**
Example scenario	e.g., develop code for other devices: ARM, PlayStation, etc.	e.g., run different OSes on same machine without needing to restart
Mechanism used	Dynamic Binary Translation	Limited Direct Execution

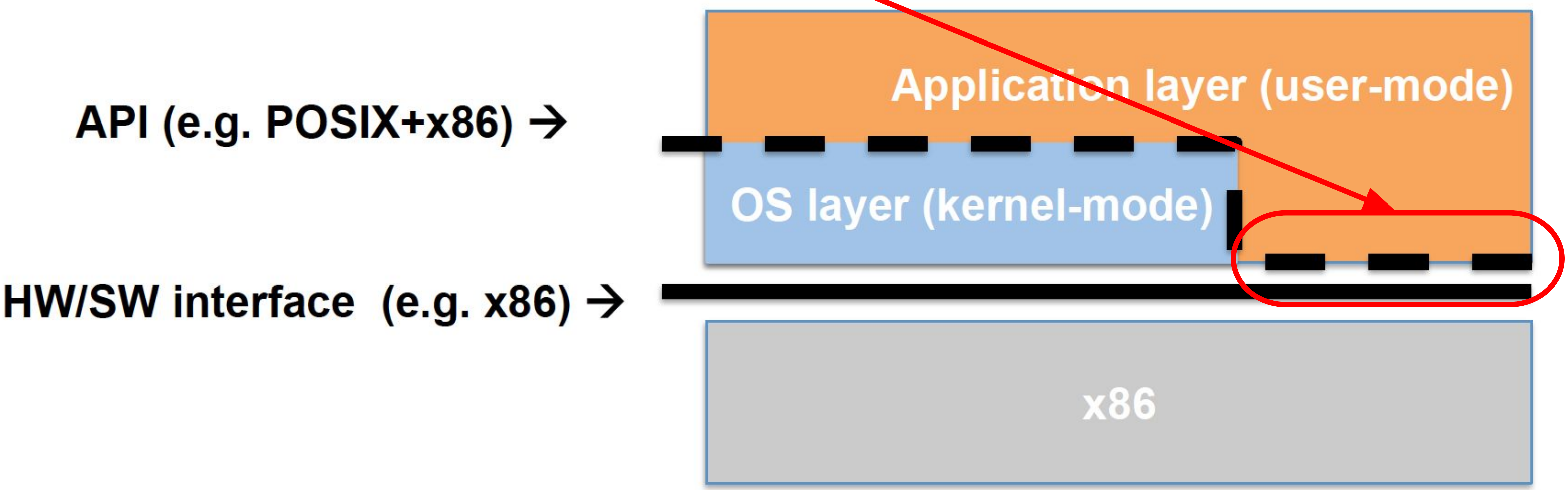
Reminder: OS / Application layering

- **Unprivileged** instructions ⟶ run in user-mode (e.g., `add`, `sub`, `div`, etc.)
- **Privileged** instructions ⟶ run in kernel-mode (e.g., `lidt` - change interrupt behavior)



Reminder: OS / Application layering

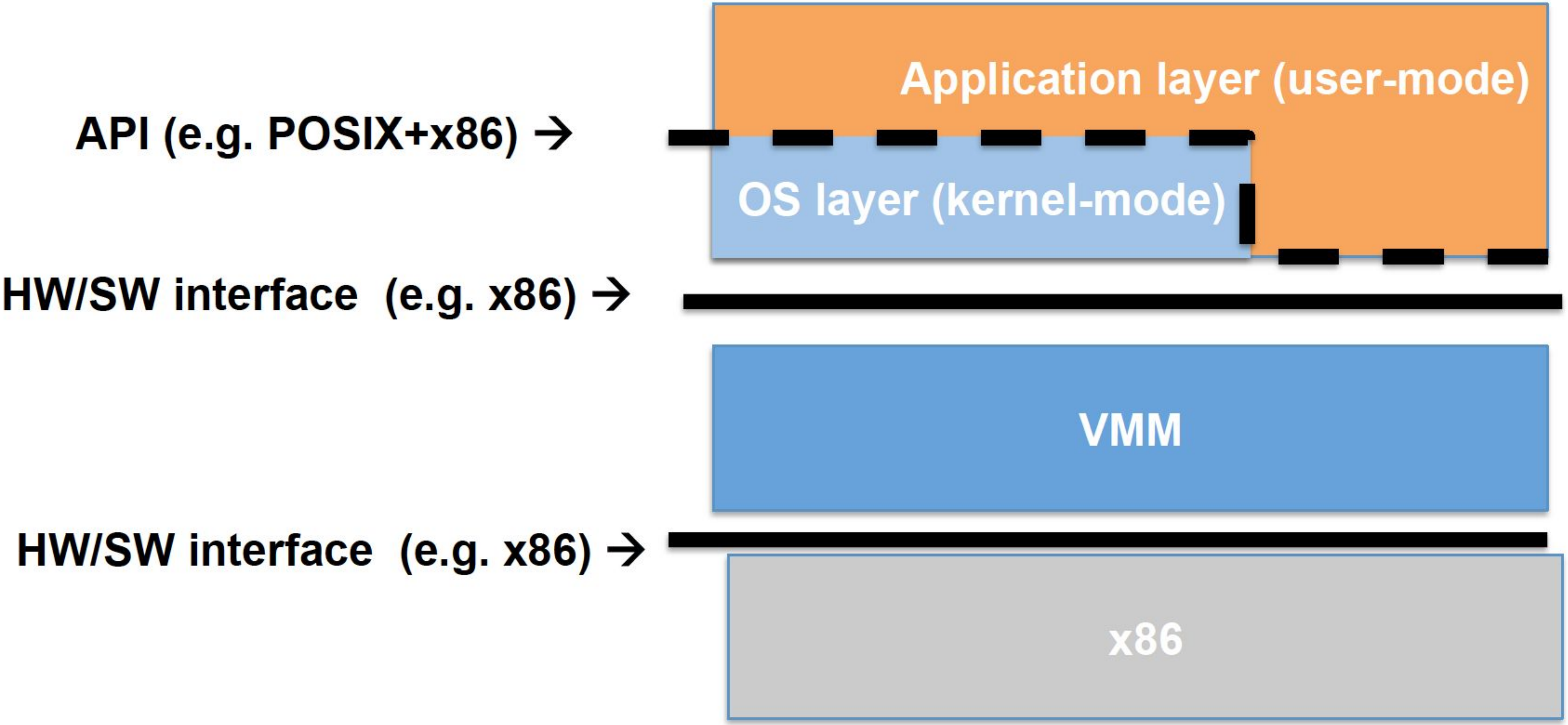
Application executes unprivileged instructions directly on the CPU



API (e.g. POSIX+x86) →

HW/SW interface (e.g. x86) →

VMM as another layer of indirection

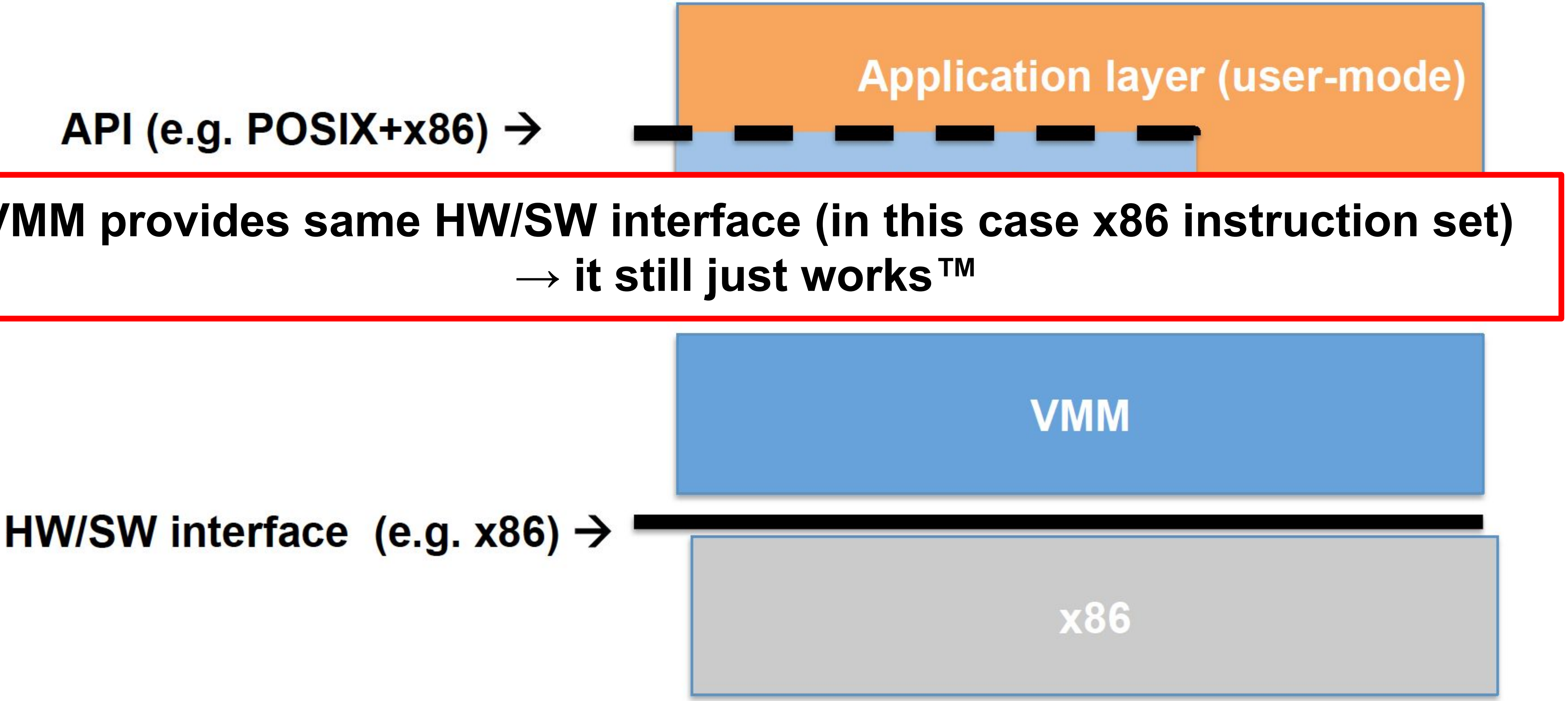


API (e.g. POSIX+x86) →

HW/SW interface (e.g. x86) →

HW/SW interface (e.g. x86) →

VMM as another layer of indirection



Application layer (user-mode)

API (e.g. POSIX+x86) →

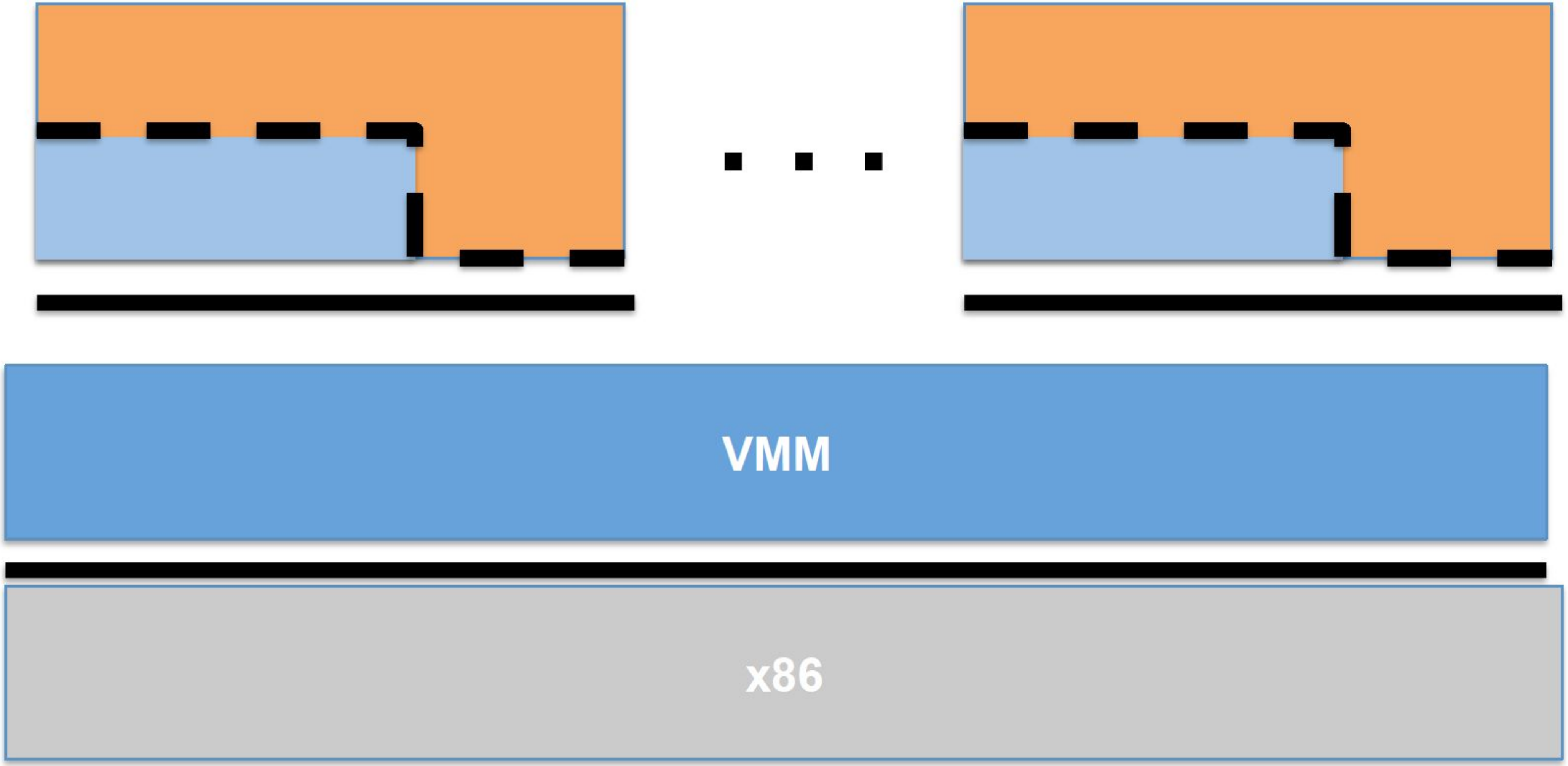
VMM provides same HW/SW interface (in this case x86 instruction set)
→ it still just works™

VMM

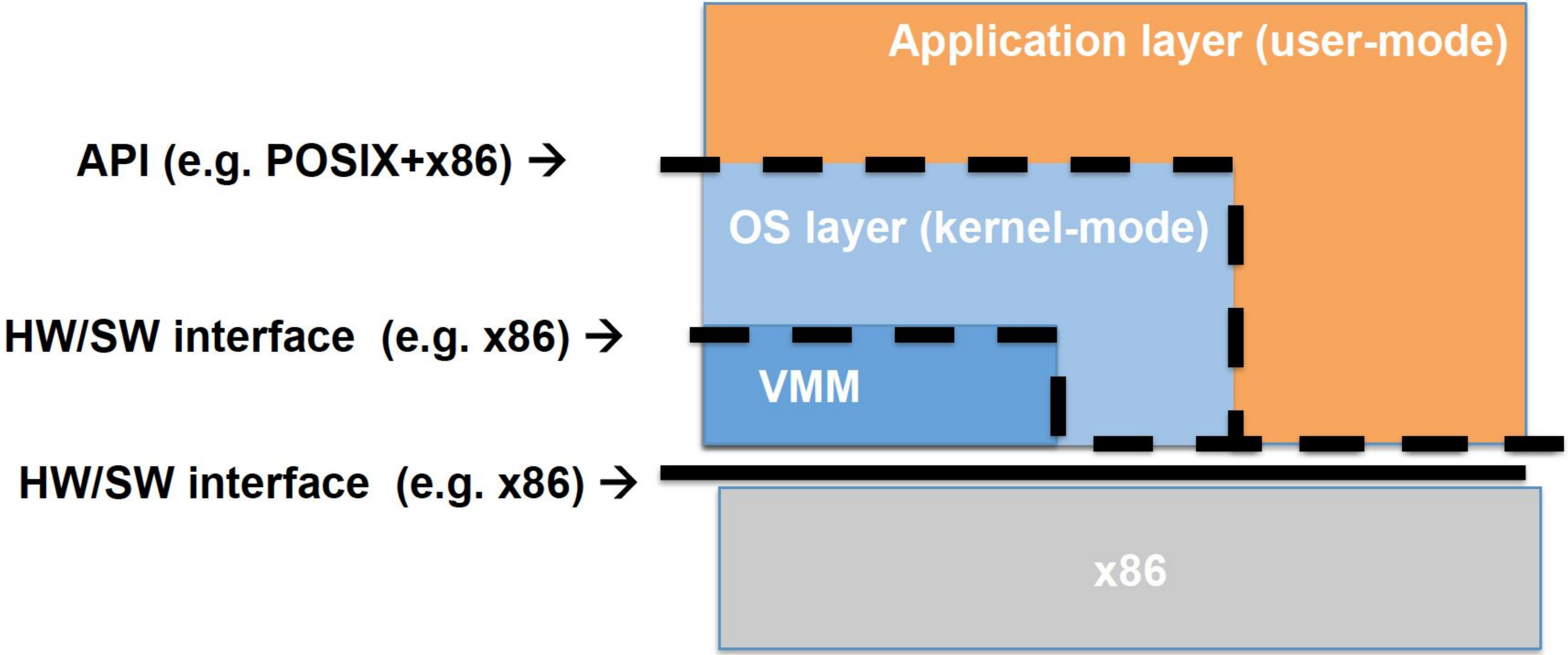
HW/SW interface (e.g. x86) →

x86

VMM → multiplexing of the CPU



Direct Execution

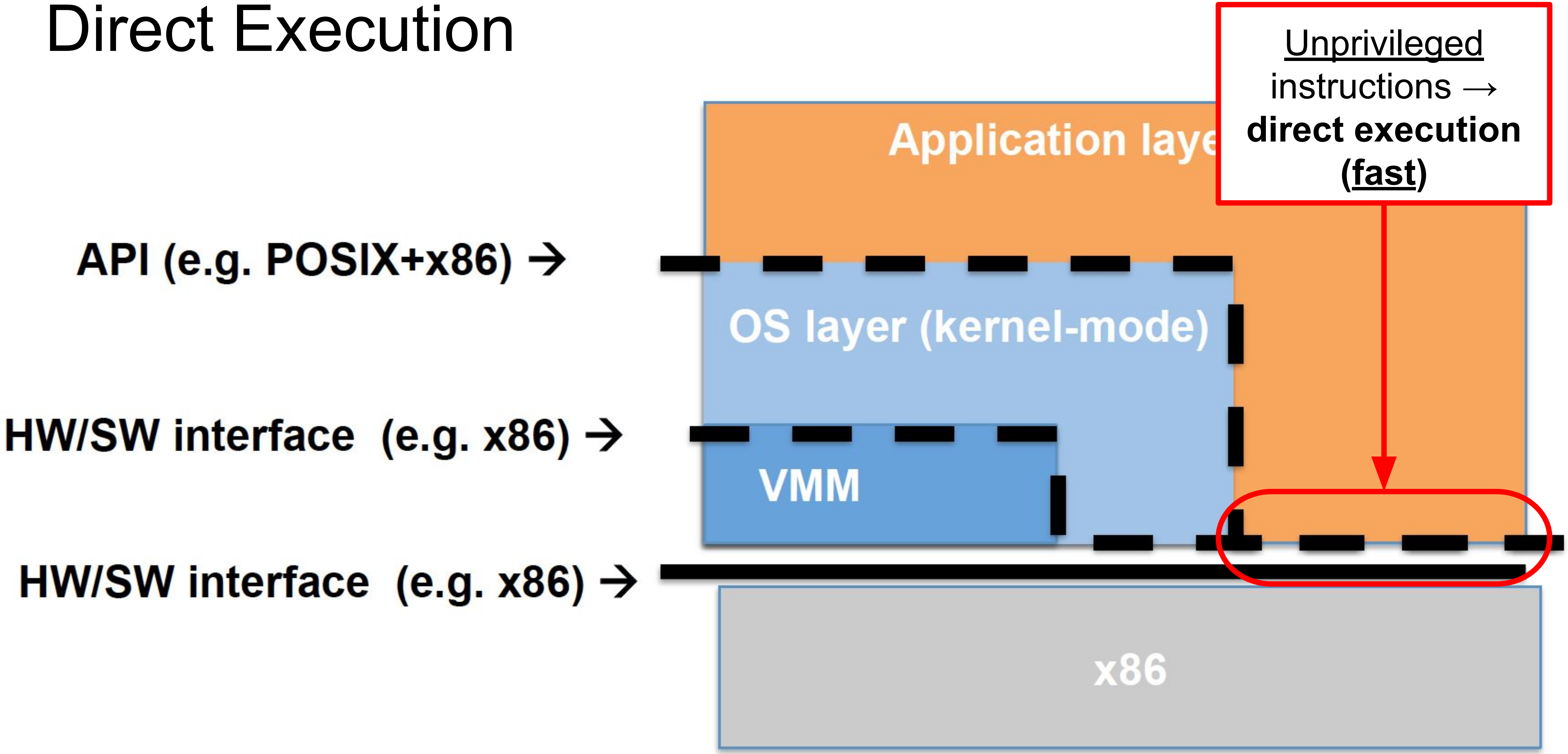


API (e.g. POSIX+x86) →

HW/SW interface (e.g. x86) →

HW/SW interface (e.g. x86) →

Direct Execution



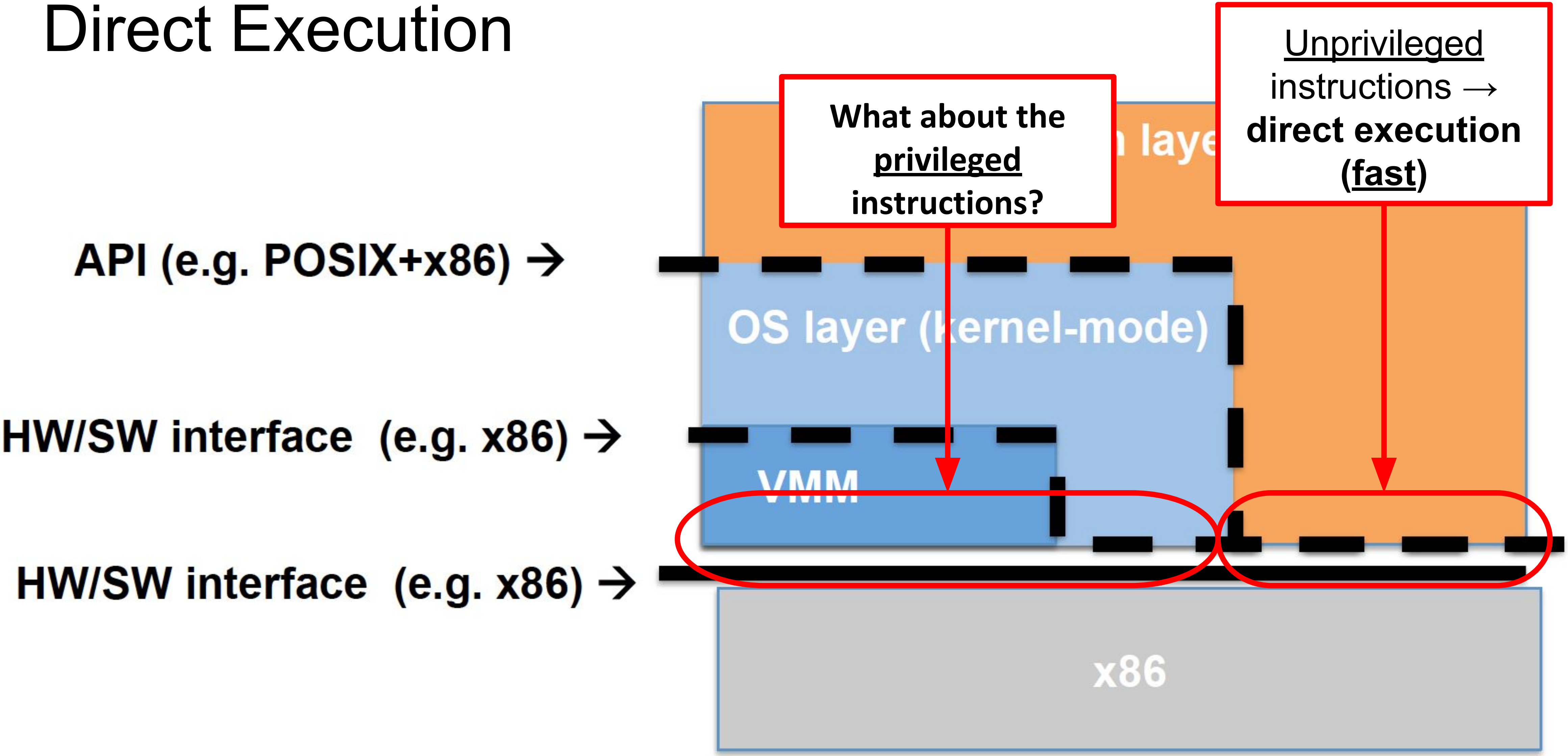
API (e.g. POSIX+x86) →

HW/SW interface (e.g. x86) →

HW/SW interface (e.g. x86) →

<u>Unprivileged</u> instructions → **direct execution (<u>fast</u>)**

Direct Execution



What about the **privileged** instructions?

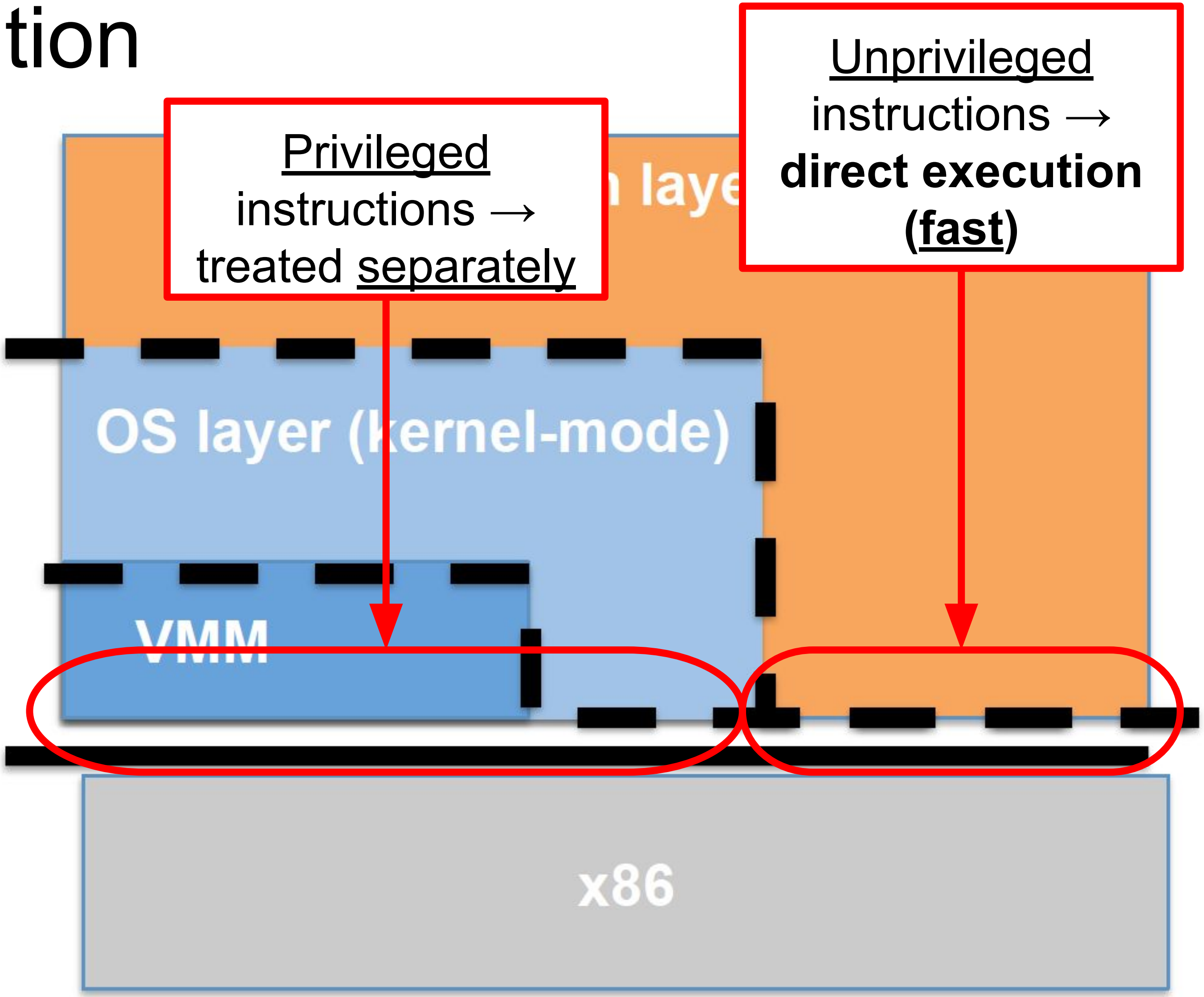
Unprivileged instructions → **direct execution (fast)**

API (e.g. POSIX+x86) →

HW/SW interface (e.g. x86) →

HW/SW interface (e.g. x86) →

Limited Direct Execution



API (e.g. POSIX+x86) →

HW/SW interface (e.g. x86) →

HW/SW interface (e.g. x86) →

<u>Privileged</u> instructions → treated <u>separately</u>

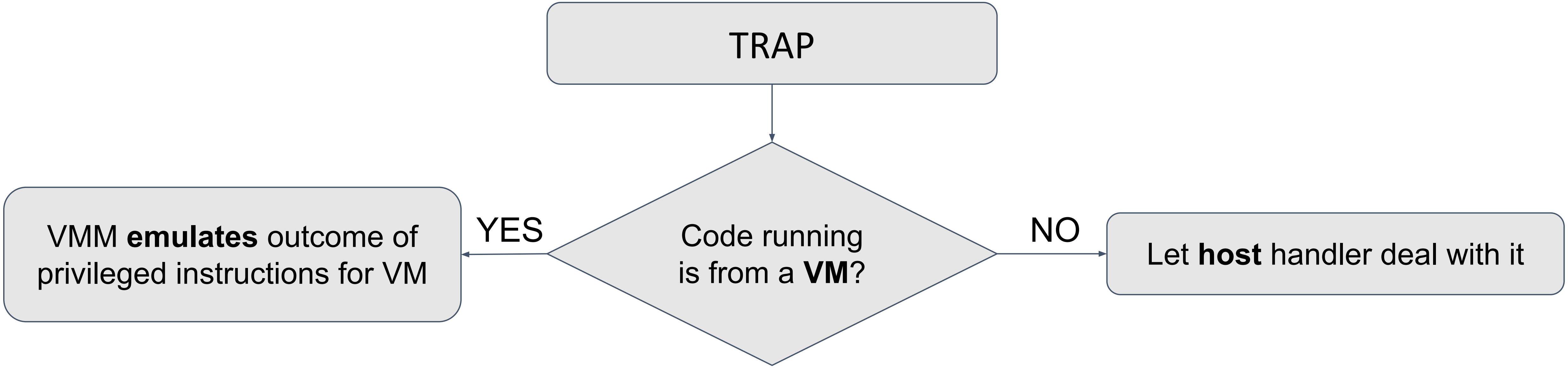
<u>Unprivileged</u> instructions → **direct execution (<u>fast</u>)**

Handling Privileged Instructions

- Solution 1: **Dynamic Binary Translation**
    - Insight: translate privileged instruction to an **unprivileged** one
- VM thinks you are executing its instruction …
    - … in fact you are executing something else
- No hardware support needed! ✓
- High overhead → very slow ✗

Trap-and-Emulate

- Run kernel code in user-mode → privileged instructions <u>fault</u> → **TRAP**
- VMM installs a new fault handler:


TRAP

Code running is from a **VM**?

YES

VMM **emulates** outcome of privileged instructions for VM

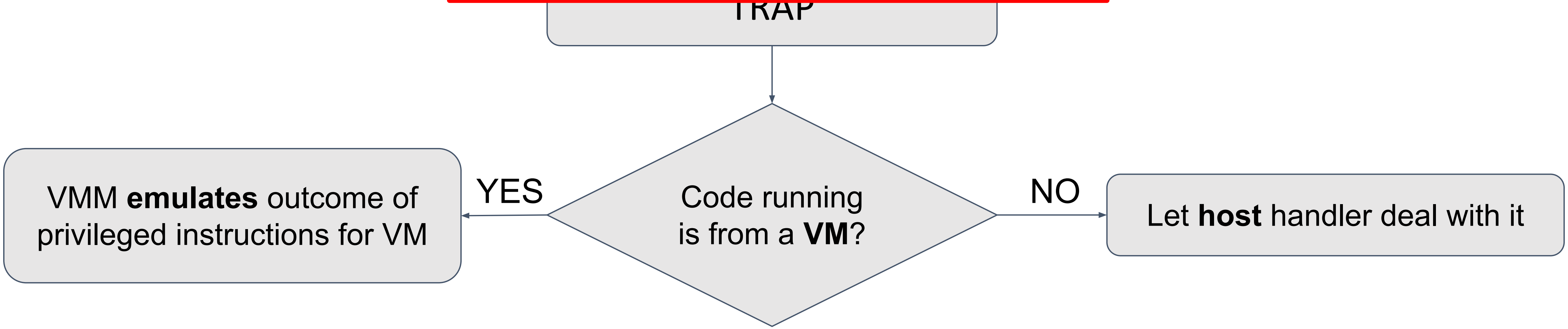
NO

Let **host** handler deal with it

Trap-and-Emulate

- Run kernel code in user-mode → privileged instructions fault → **TRAP**
- VMM installs a nev

Everything
still works™!



TRAP

Code running
is from a **VM**?

YES

VMM **emulates** outcome of
privileged instructions for VM

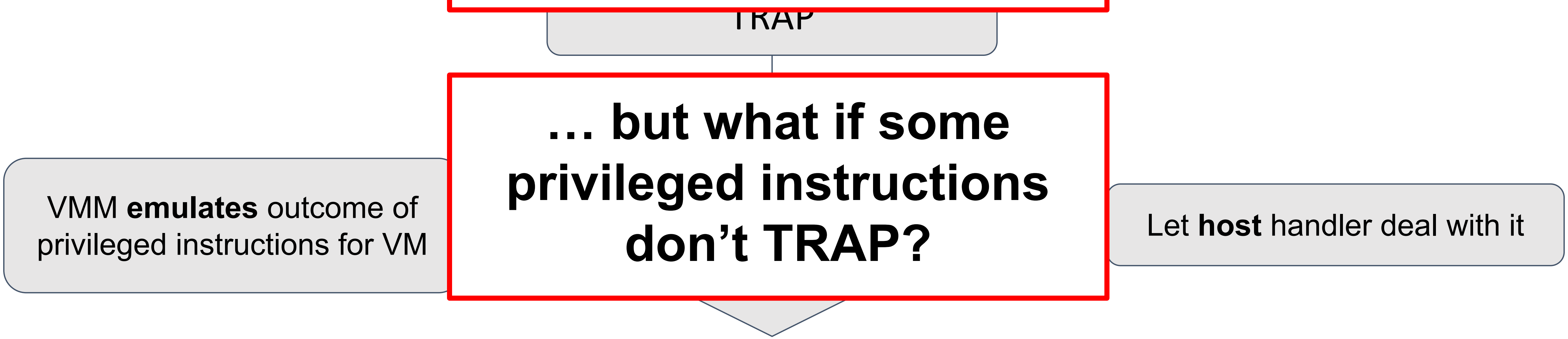
NO

Let **host** handler deal with it

Trap-and-Emulate

- Run kernel code in user-mode → privileged instructions fault → **TRAP**
- VMM installs a ne

Everything
still works™!



VMM **emulates** outcome of privileged instructions for VM

… but what if some privileged instructions don’t TRAP?

Let **host** handler deal with it

Popek/Goldberg Theorem (1974)

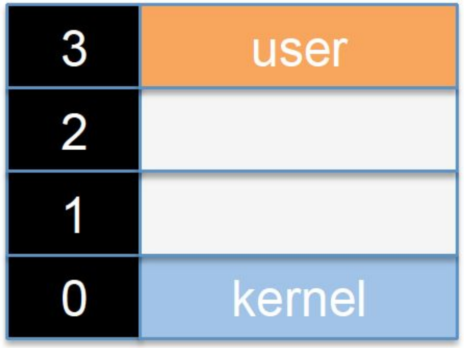
- **Privileged** instruction → runs only in kernel-mode
- **Sensitive** instruction → behaves differently in kernel-mode vs. user-mode
  - aka... *doesn't trap!*

- **VMM exists for an architecture iff {sensitive} ⊆ {privileged}.**

- Rephrased: trap-and-emulate works only if all sensitive instructions are privileged.

Is x86 a virtualizable architecture?

- **32-bit x86 architecture**
  - 4 protection rings
  - Segments and paging support
  - Ring 1 and 2 are never used



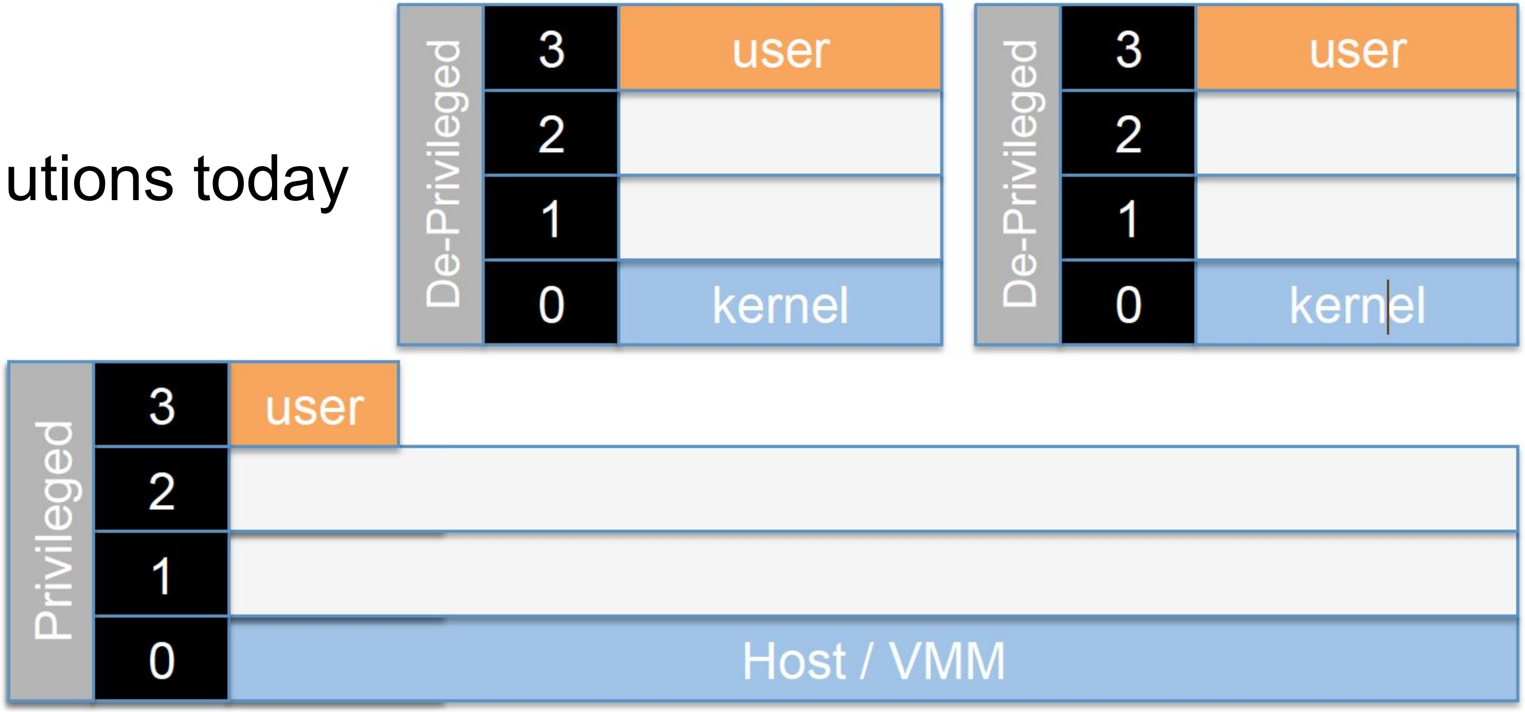
3	user
2	
1	
0	kernel

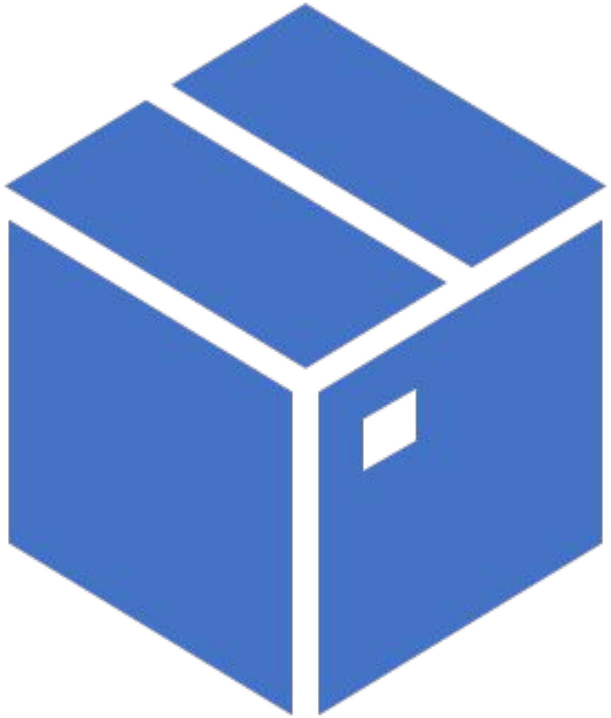
- **17 sensitive, unprivileged instructions** ☹

- **VMM still possible, but more complicated**

Intel VT-x and AMD-v (2005)

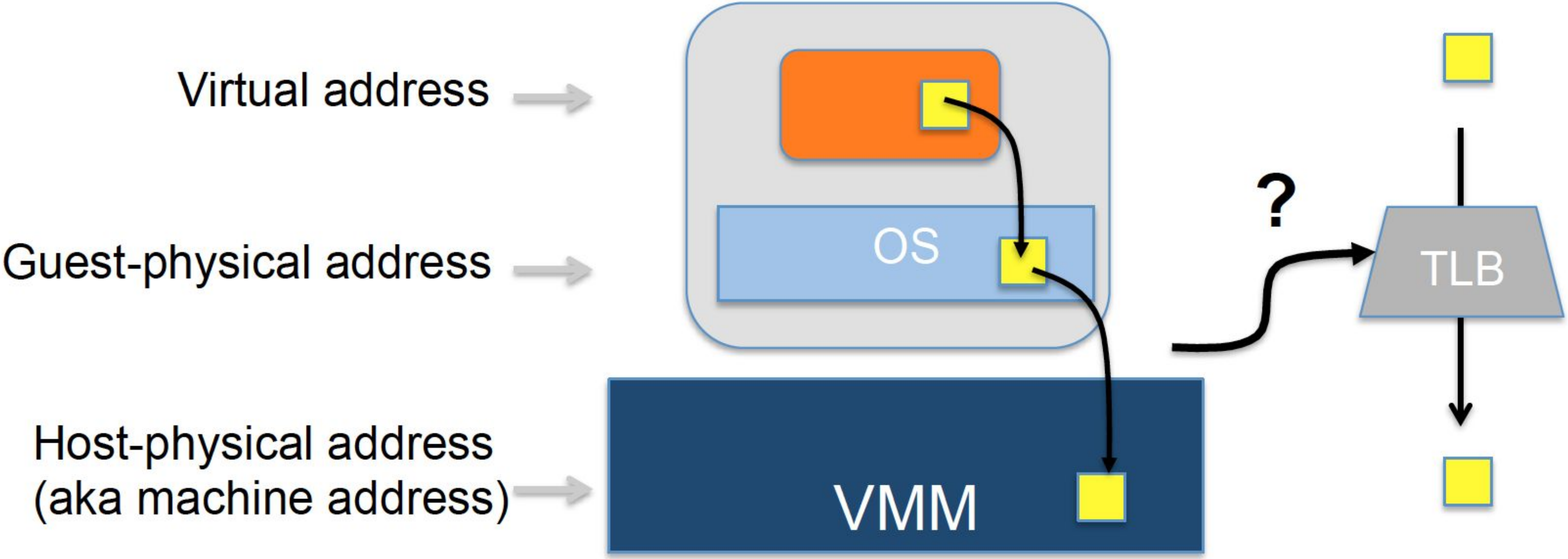
- Available on all current 64-bit processors
  - Duplicate the 4 protection rings
  - Meets Popek/Goldberg criteria
- Used by all virtualization solutions today





Virtual “Physical Memory”

Virtualizing physical memory

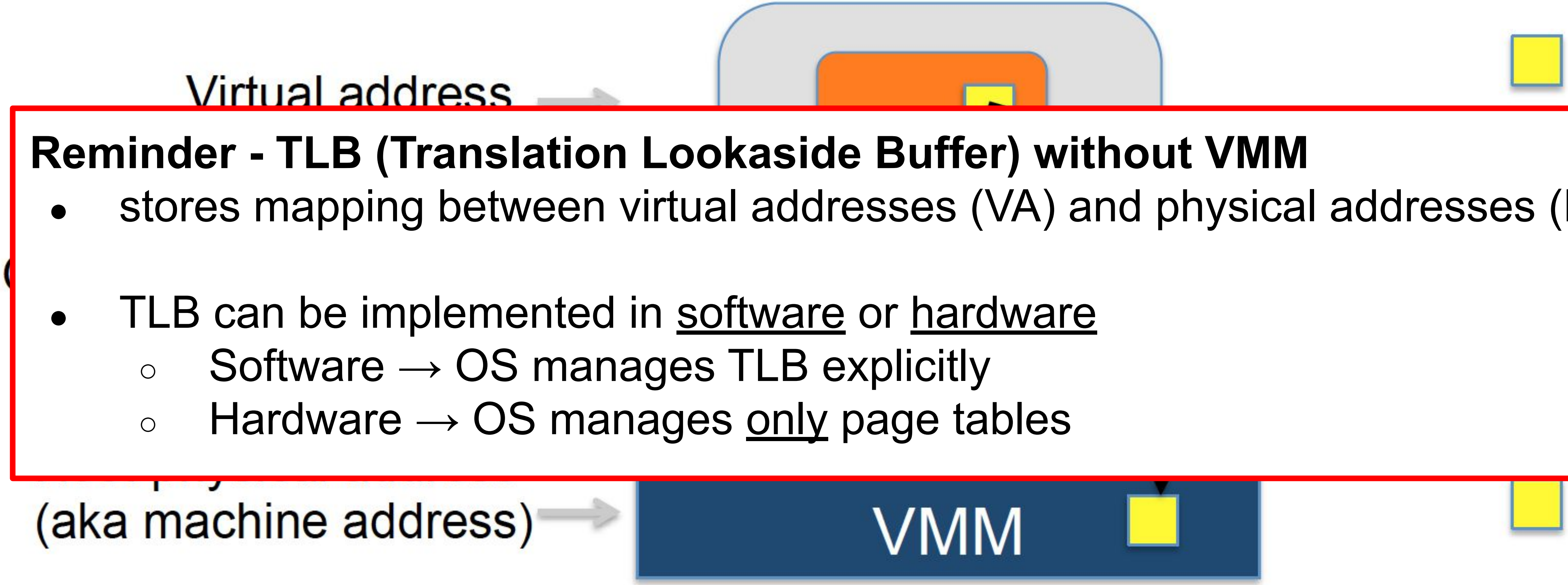


Virtual address

Guest-physical address

Host-physical address
(aka machine address)

Virtualizing physical memory



Reminder - TLB (Translation Lookaside Buffer) without VMM

- stores mapping between virtual addresses (VA) and physical addresses (PA)

- TLB can be implemented in <u>software</u> or <u>hardware</u>
  - Software → OS manages TLB explicitly
  - Hardware → OS manages <u>only</u> page tables

Virtualizing physical memory

* Only the VAs in the **guest** are useful. We do not consider hVAs (for the host).

What is given:

- VA* → gPA (managed by the **guest** OS)
- gPA → hPA (managed by the **VMM**)

What is needed:

- VA* → hPA

Challenge:

- **How to insert VA → hPA mappings into TLB ?**

Virtualizing physical memory

* Only the VAs in the **guest** are useful. We do not consider hVAs (for the host).

What is given:

- VA* → gPA (managed by the **guest** OS)
- gPA → hPA (managed by the **VMM**)

N.B.: There is only **<u>one</u>** physical TLB for the whole physical machine and for all the VMs!

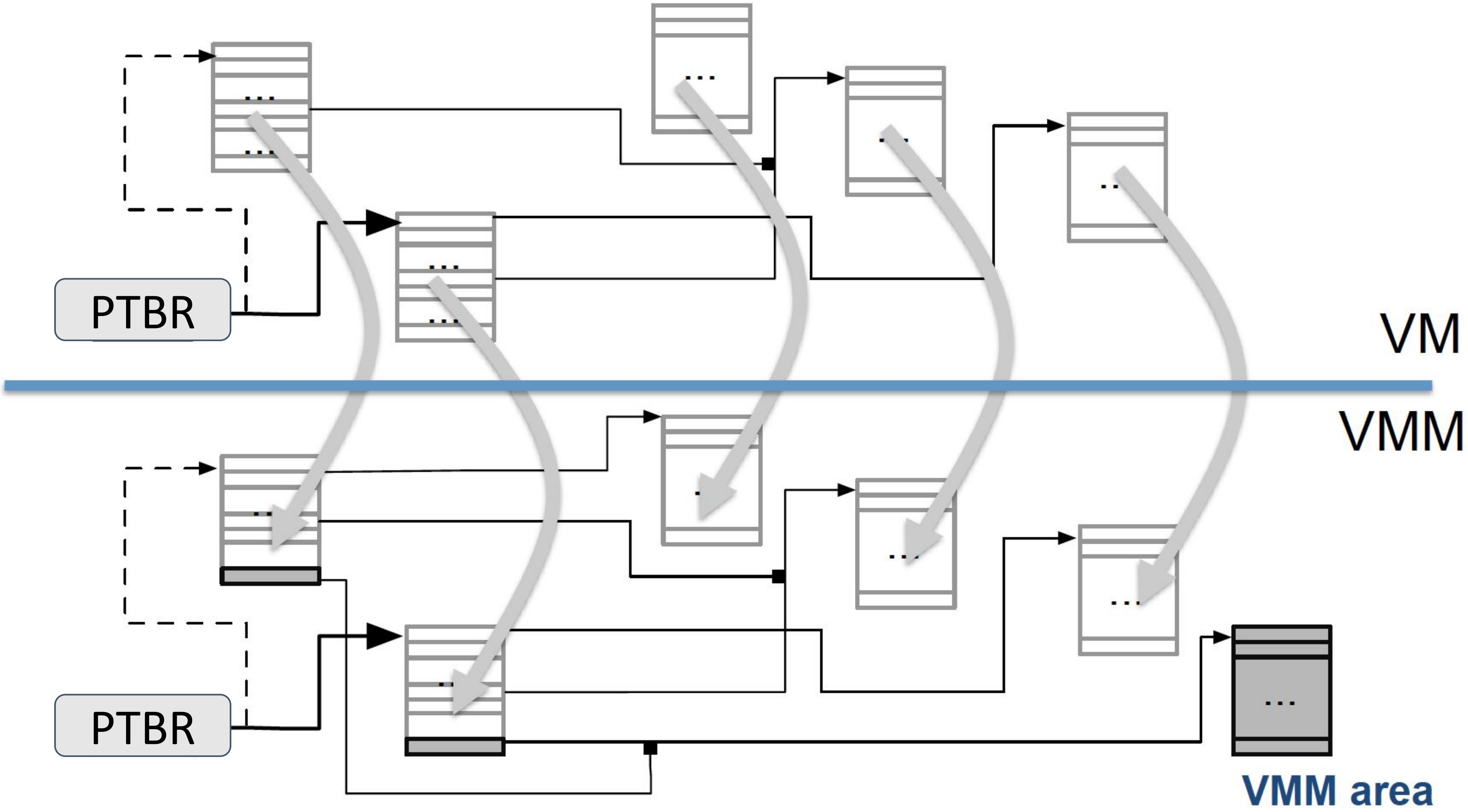
What is

- VA* → hPA

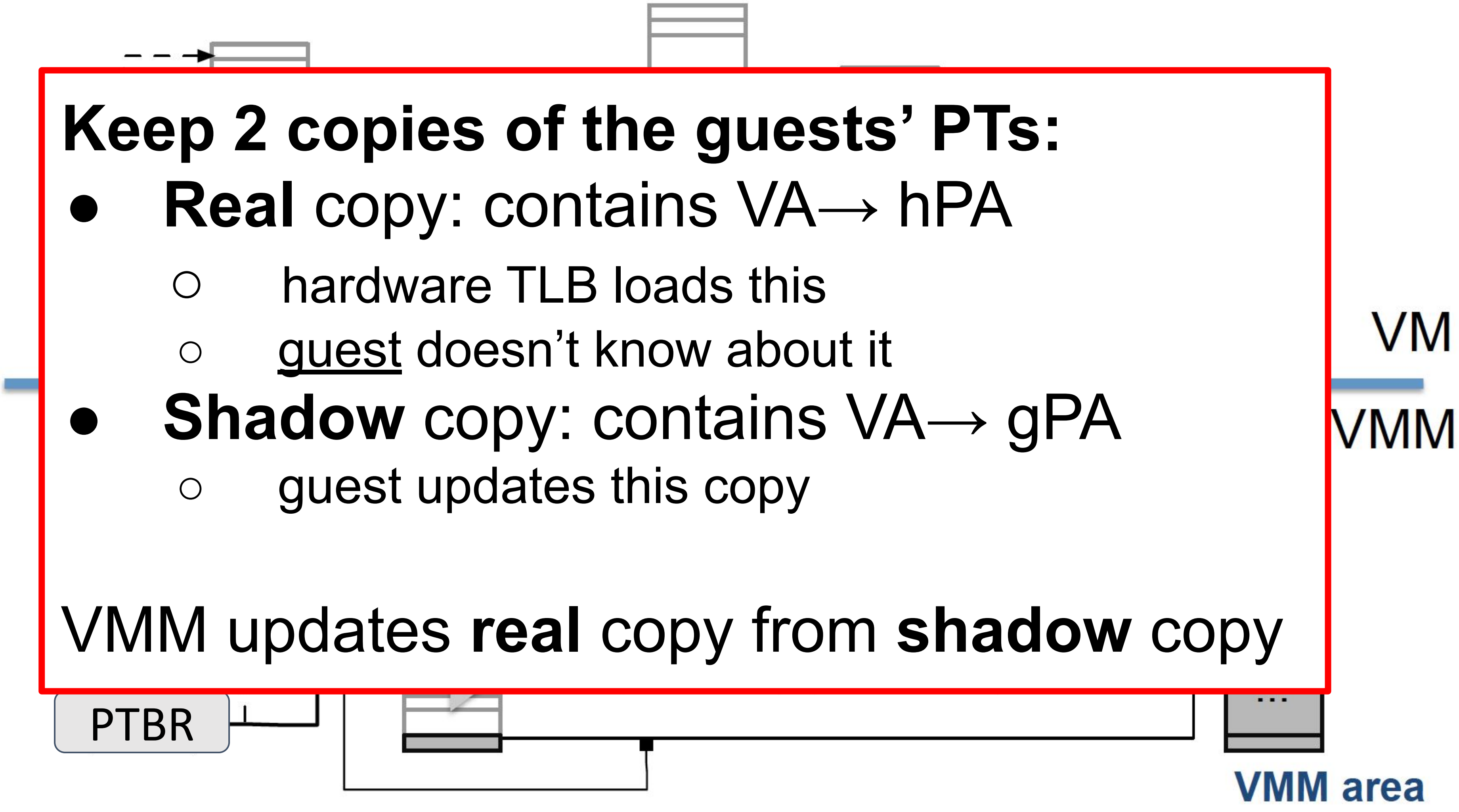
Challenge:

- **How to insert VA → hPA mappings into TLB ?**

Solution 1: Shadow Page Tables



Solution 1: Shadow Page Tables

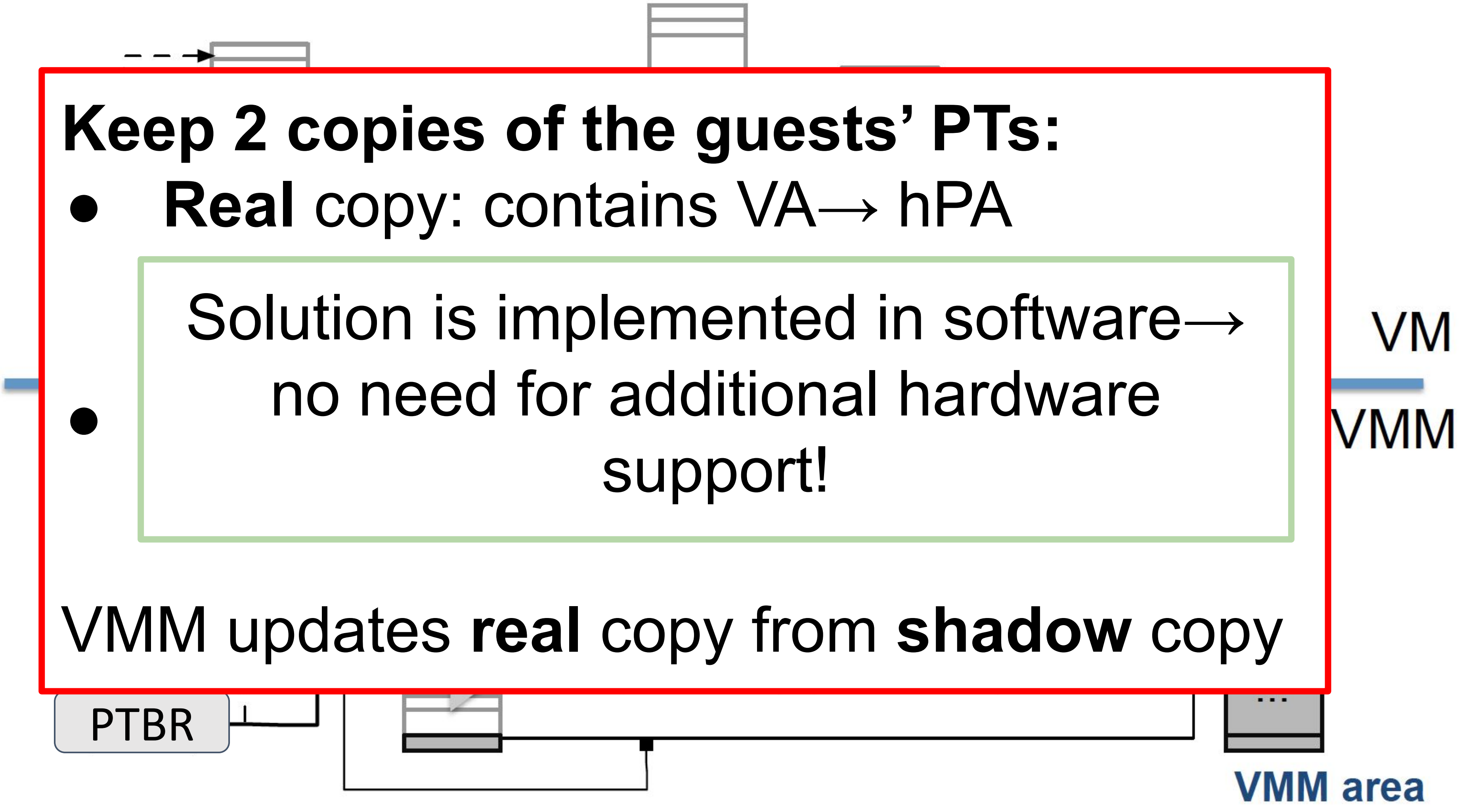


Keep 2 copies of the guests' PTs:

- **Real** copy: contains VA→ hPA
  - hardware TLB loads this
  - guest doesn't know about it
- **Shadow** copy: contains VA→ gPA
  - guest updates this copy

VMM updates **real** copy from **shadow** copy

Solution 1: Shadow Page Tables



Keep 2 copies of the guests' PTs:

- **Real** copy: contains VA→ hPA
-

Solution is implemented in software→ no need for additional hardware support!

VMM updates **real** copy from **shadow** copy

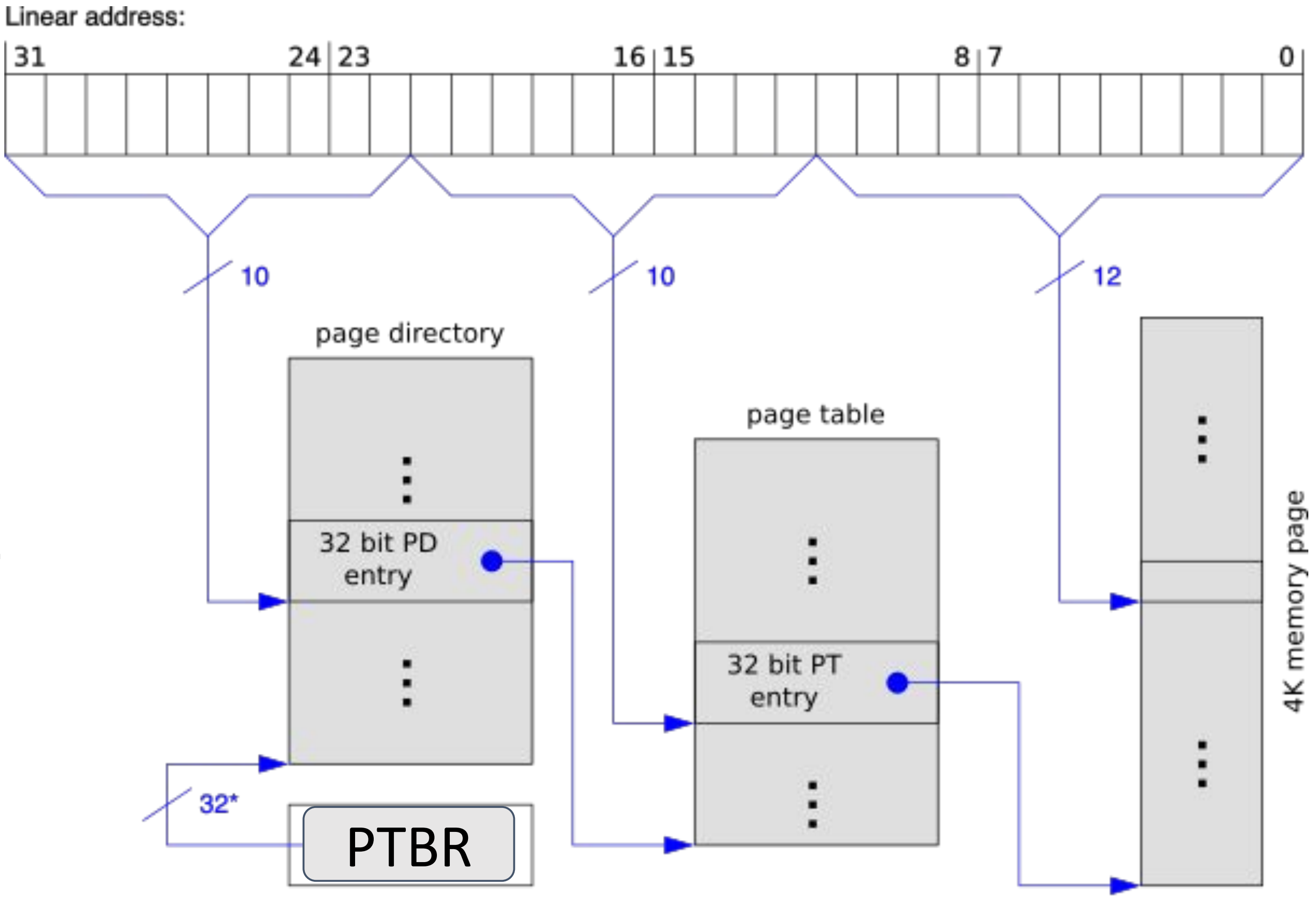
Solution 2: Nested Page Tables

- Requires hardware support
- Intel calls it “Extended Page Tables”
- AMD calls it “Rapid Virtualization Indexing”

… so how does it work?

Reminder – Multi-level Page Tables

- Example: two-level PT
- Successive lookup phases:
  - PTBR → 1st PT
  - 1st PT entry → 2nd PT
  - 2nd PT entry → page



*) 32 bits aligned to a 4-KByte boundary

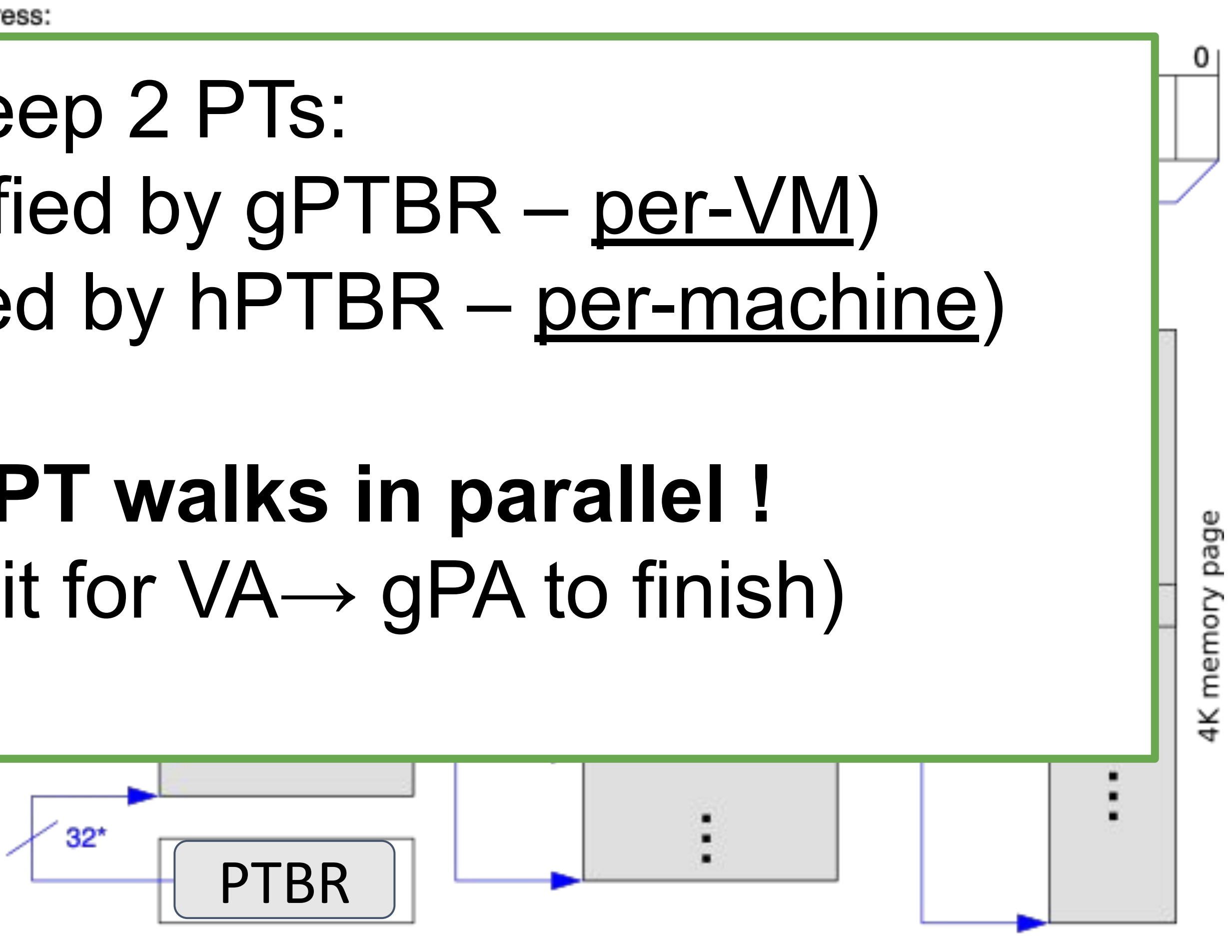
Reminder – Multi-level Page Tables

We keep 2 PTs:

- **VA→ gPA** (identified by gPTBR – per-VM)
- **gPA → hPA** (identified by hPTBR – per-machine)

We can perform PT walks in parallel !

(e.g., no need to wait for VA→ gPA to finish)



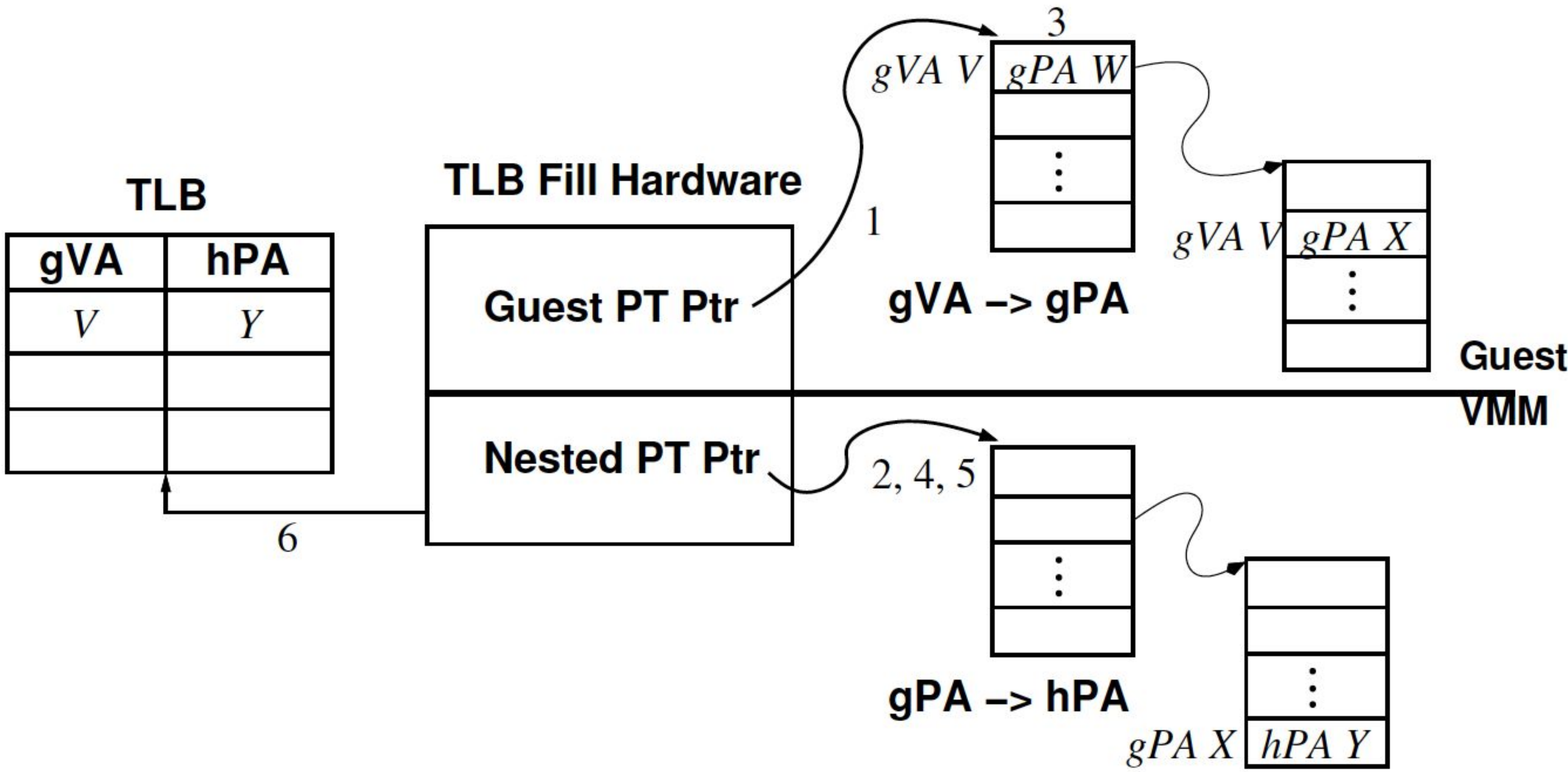
*) 32 bits aligned to a 4-KByte boundary

Solution 2: Nested Page Tables

TLB filling algorithm:

1. VA → gPA (1st level lookup)
2. **gPA → hPA (1st level lookup)**
3. **VA → gPA (2nd level lookup)**
4. gPA → hPA (2nd level lookup)
5. Return hPA to OS
6. Fill TLB with gVA → hPA

Steps 2 & 3 execute in parallel!



Solution 2: Nested Page Tables

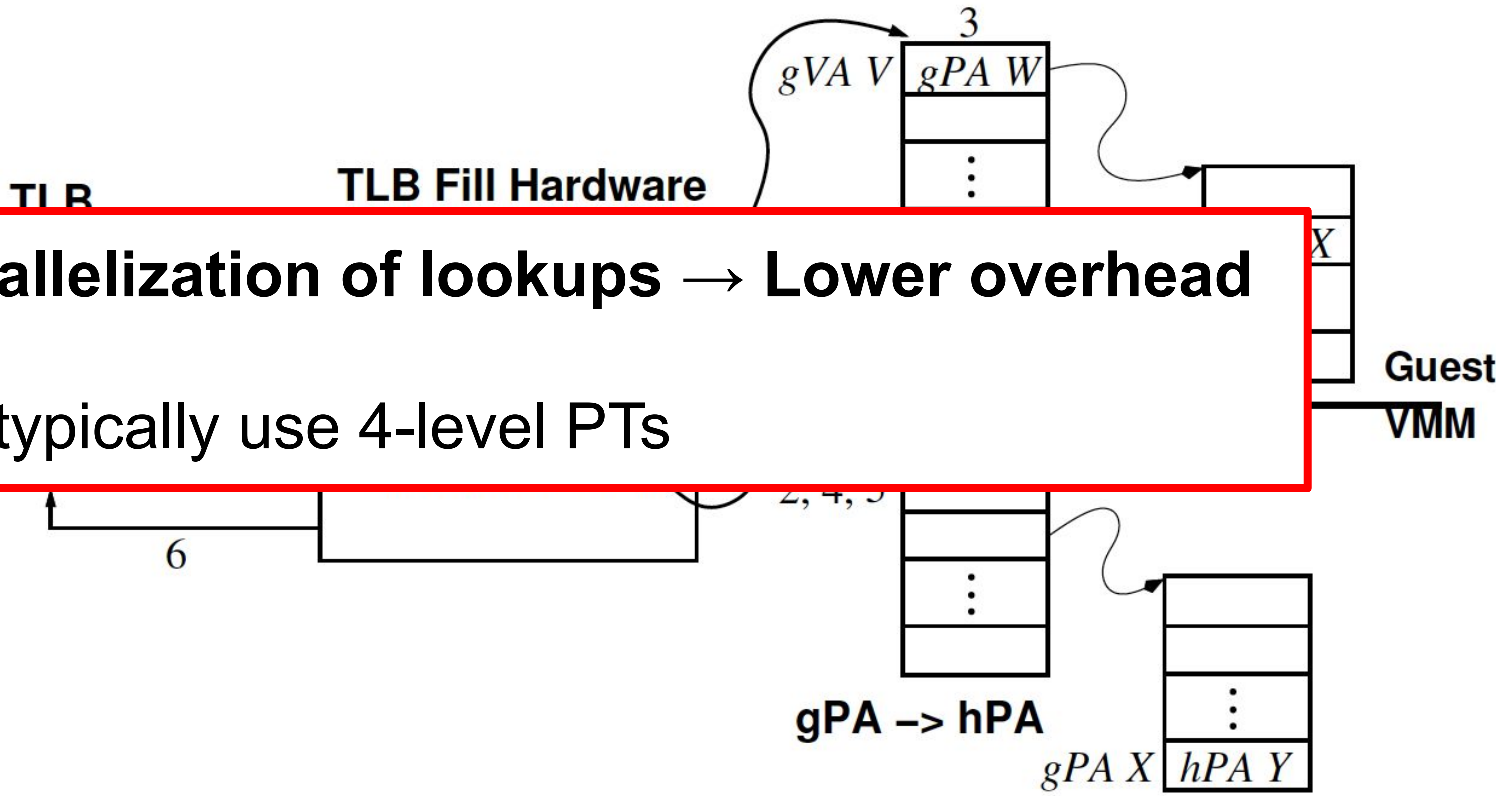
TLB filling algorithm:

More PT levels → Higher parallelization of lookups → Lower overhead

e.g., Intel and AMD processors typically use 4-level PTs

6. Fill TLB with gVA → hPA

Steps 2 & 3 execute in parallel!



Summary

Summary

Virtualization: “virtual” abstraction layer

VM: virtualizing an entire computer

VMM: how VMs are implemented

Popek-Goldberg: sufficient and necessary conditions to build a VMM

Direct execution and address translation in VMMs

A bit of history…

Very popular in the old days (60’s, 70’s)

- Hardware was expensive
- Operating systems were primitive

Out of favor for two decades (80's, 90's)

- Hardware became cheap
- Operating systems became powerful (UNIX, ...)

Back in favor since 2000

- Because the operating system is special (compatibility)
- Because hardware exceeds the capacity of a user (cloud)
- Because it is easier to provision a virtual machine than a physical machine
- ...

VMs and Virtualization are the future!

- Cloud computing is the new paradigm for IT
  - Trend is now going towards serverless computing

- Enterprise IT is moving to desktop virtualization and converged infrastructure

- Docker has changed the way we do software development and delivery

- Many challenges ahead
  - Mobile virtualization
  - Virtualized blockchains
  - …