6.1800 Spring 2025

Lecture #6: Virtual Machines

even more virtualization, plus kernel designs

Katrina LaCurts | lacurts@mit.edu | 6.1800 2025

6.1800 in the news

POSTED ON FEBRUARY 14, 2025 TO CONNECTIVITY, NETWORKING & TRAFFIC

Unlocking global AI potential with next-generation subsea infrastructure

With Project Waterworth, we continue to advance engineering design to maintain cable resilience, enabling us to build the longest 24 fiber pair cable project in the world and enhance overall speed of deployment. We are also deploying first-of-its-kind routing, maximizing the cable laid in deep water — at depths up to 7,000 meters — and using enhanced burial techniques in high-risk fault areas, such as shallow waters near the coast, to avoid damage from ship anchors and other hazards.

all of the physical infrastructure we require to build our systems has an **impact** on the world around us

we'll return to this issue at different points the class

operating systems enforce modularity on a single machine using **virtualization**

in order to enforce modularity + have an effective operating system, a few things need to happen

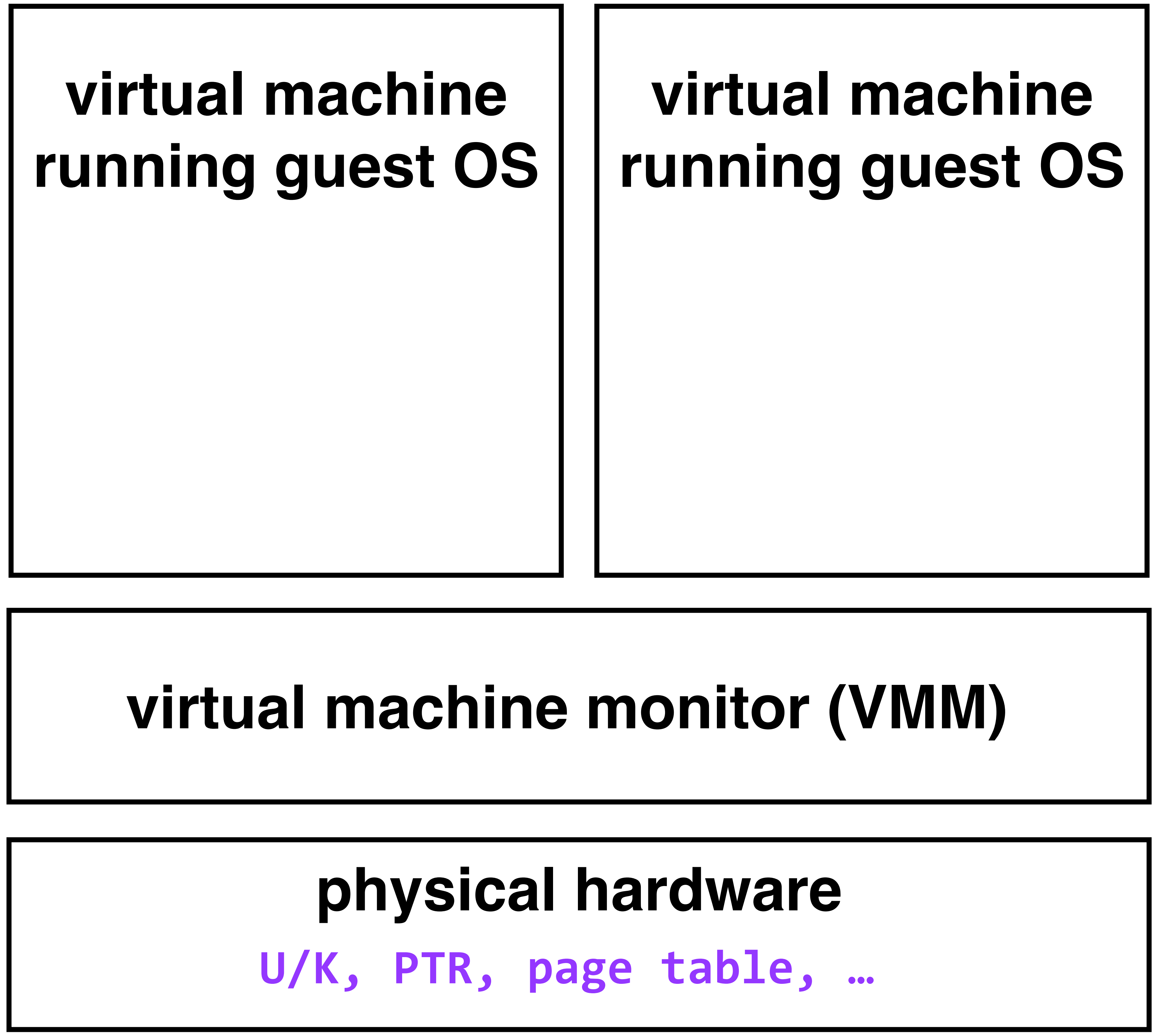
1. programs shouldn't be able to refer to (and corrupt) each others' **memory** → **virtual memory**

2. programs should be able to **communicate** with each other → **bounded buffers**
(virtualize communication links)

3. programs should be able to **share a CPU** without one program halting the progress of the others → **threads**
(virtualize processors)

today's goal: run multiple operating systems at once

virtual machine monitor virtualizes the physical hardware for the guest OSes



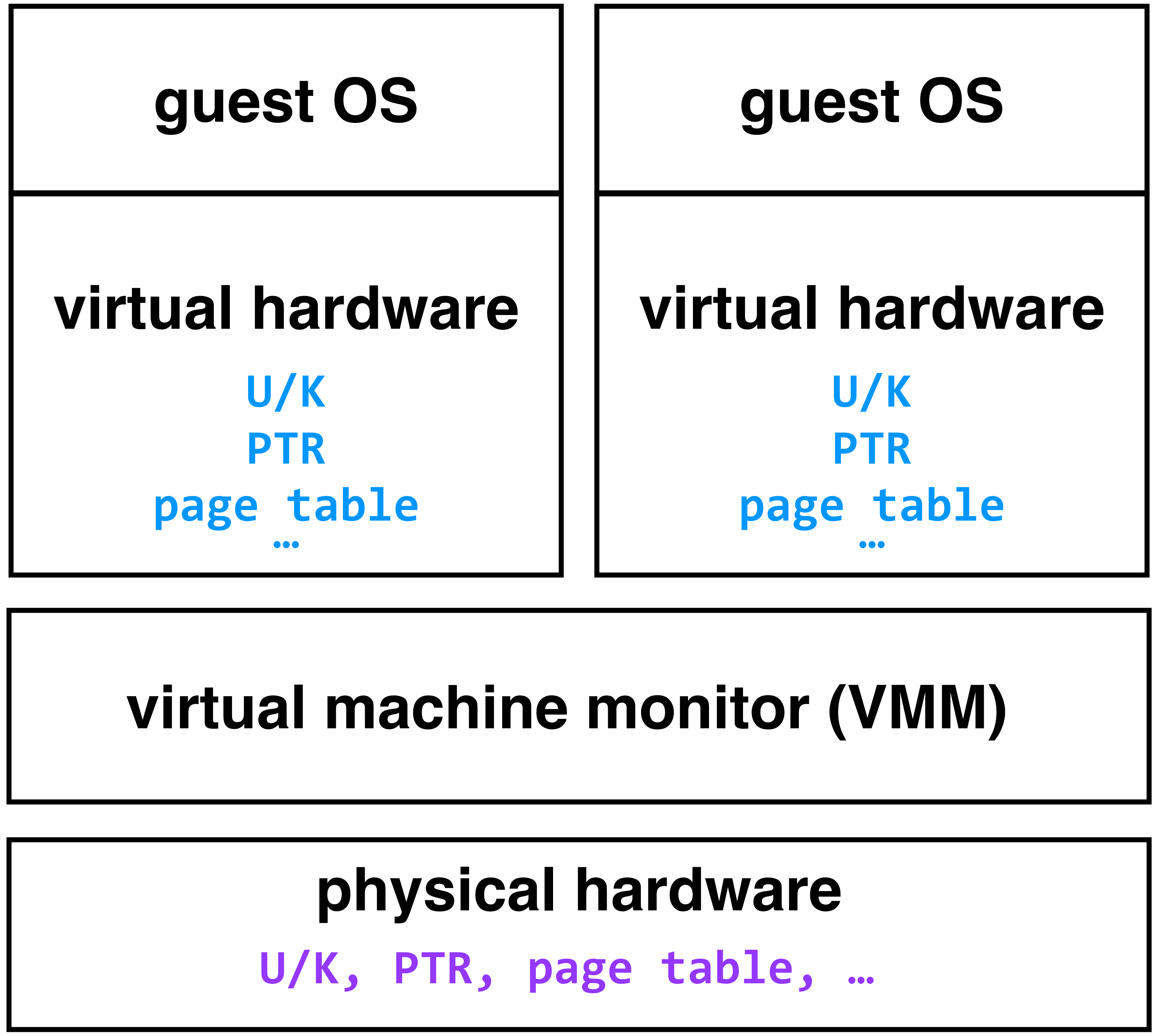
guest OSes run in user mode

privileged instructions in guest OS will cause an exception, which the VMM will intercept (“**trap**”) and **emulate**

if the VMM *can’t* emulate an instruction, it will send the exception back to the guest OS for handling

first question: what does it mean to emulate?

virtual machine monitor virtualizes the physical hardware for the guest OSes



guest OSes run in user mode

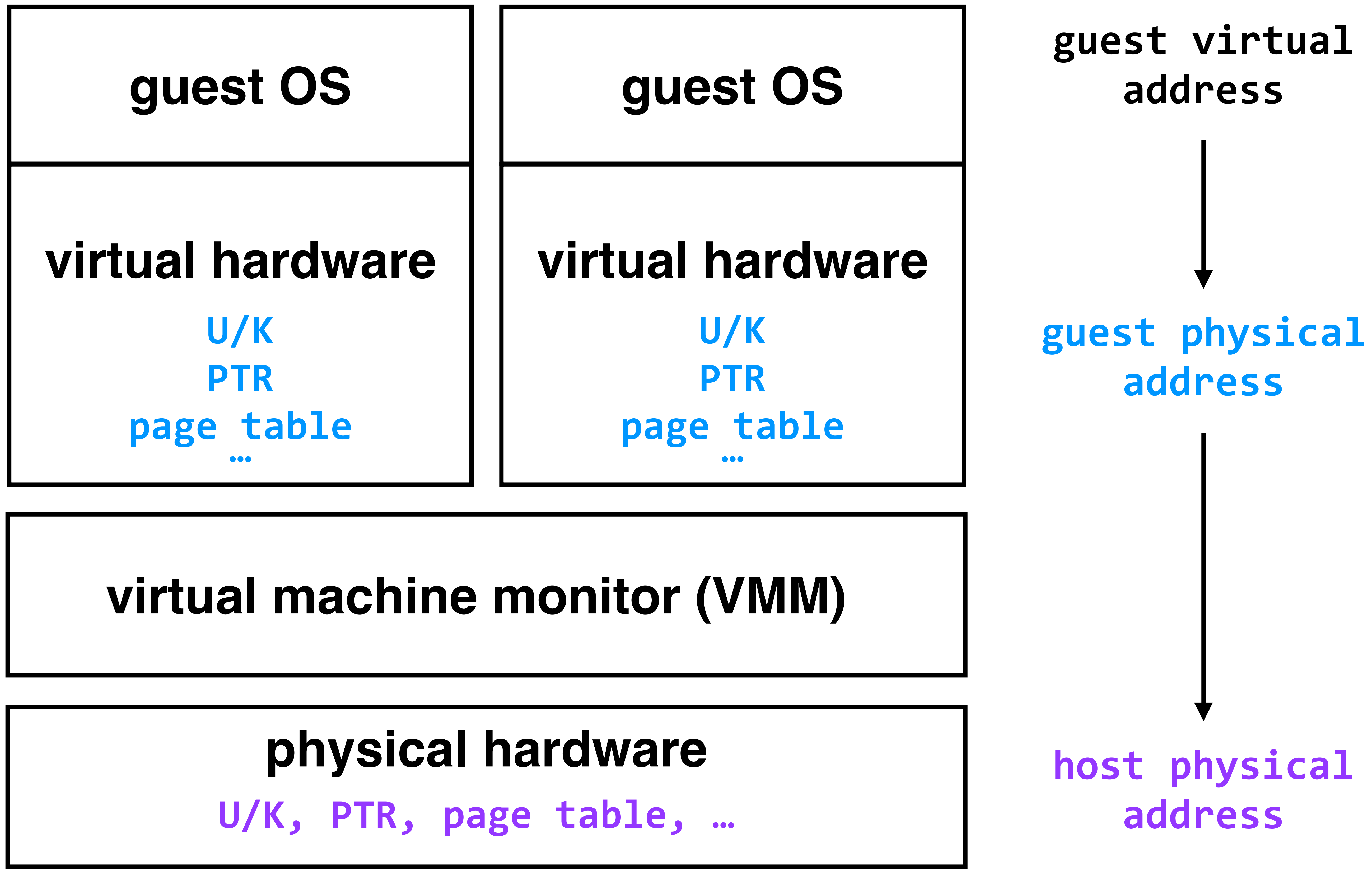
privileged instructions in guest OS will cause an exception, which the VMM will intercept ("**trap**") and **emulate**

if the VMM *can't* emulate an instruction, it will send the exception back to the guest OS for handling

first question: what does it mean to emulate?

virtual machine monitor virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again!)

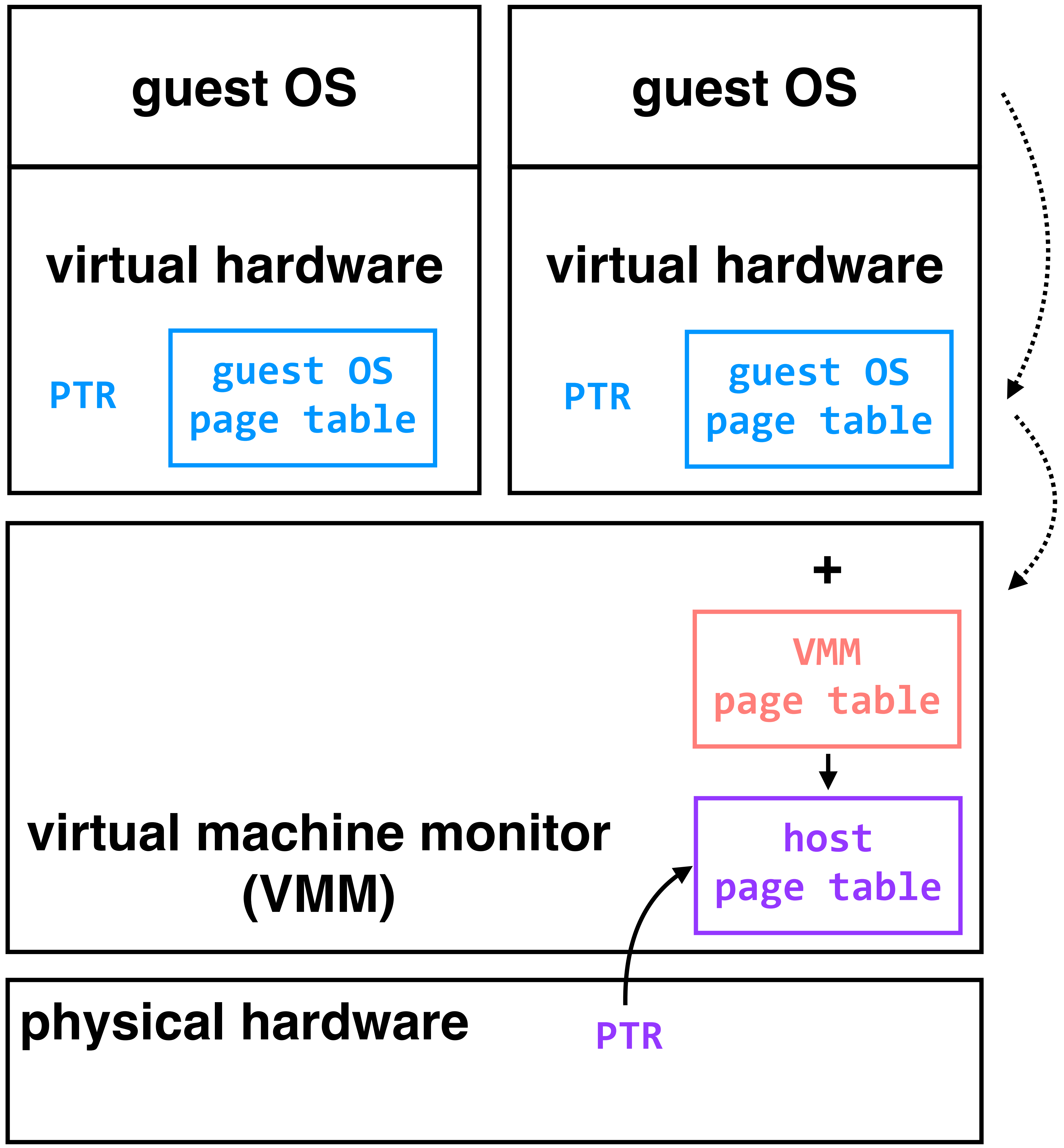


first question: what does it mean to emulate?

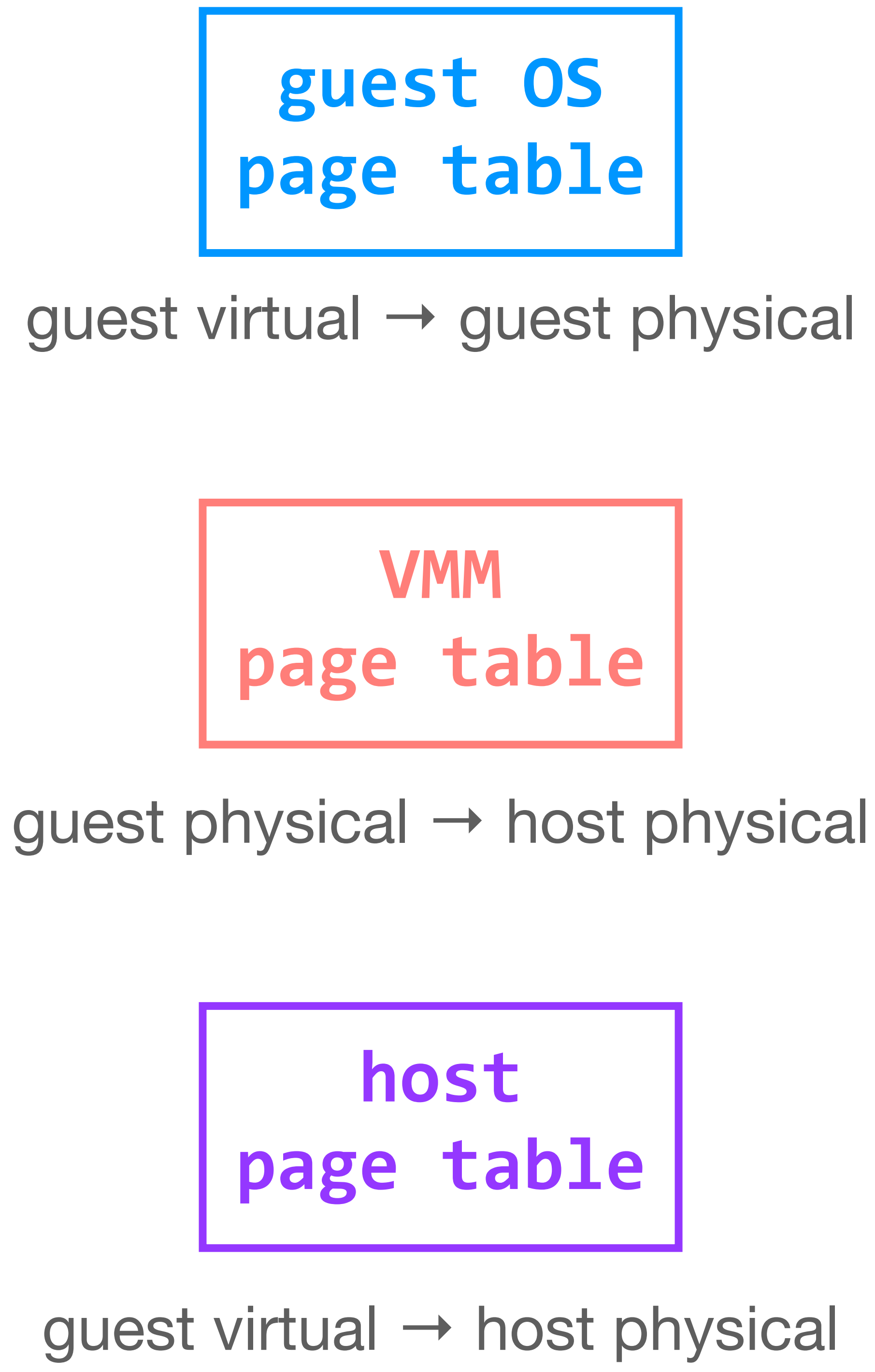
in this example, it means that the VMM needs to step in and translate guest physical addresses to host physical addresses

virtual machine monitor virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again!)

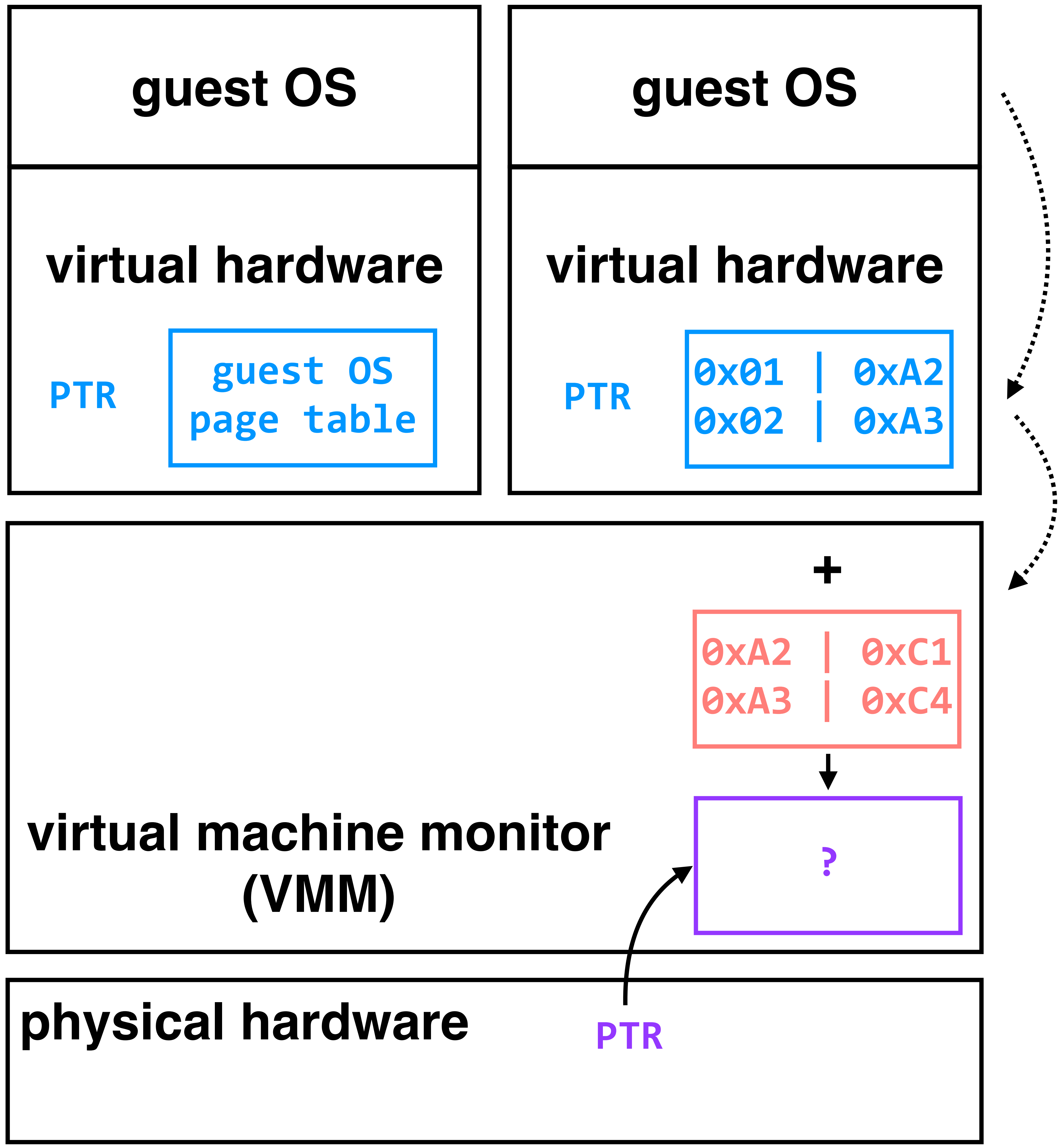


1. guest OS loads its PTR, which triggers an exception; the VMM intercepts
2. VMM combines the guest page table with its own page table to create a host page table
3. physical hardware uses the host page table

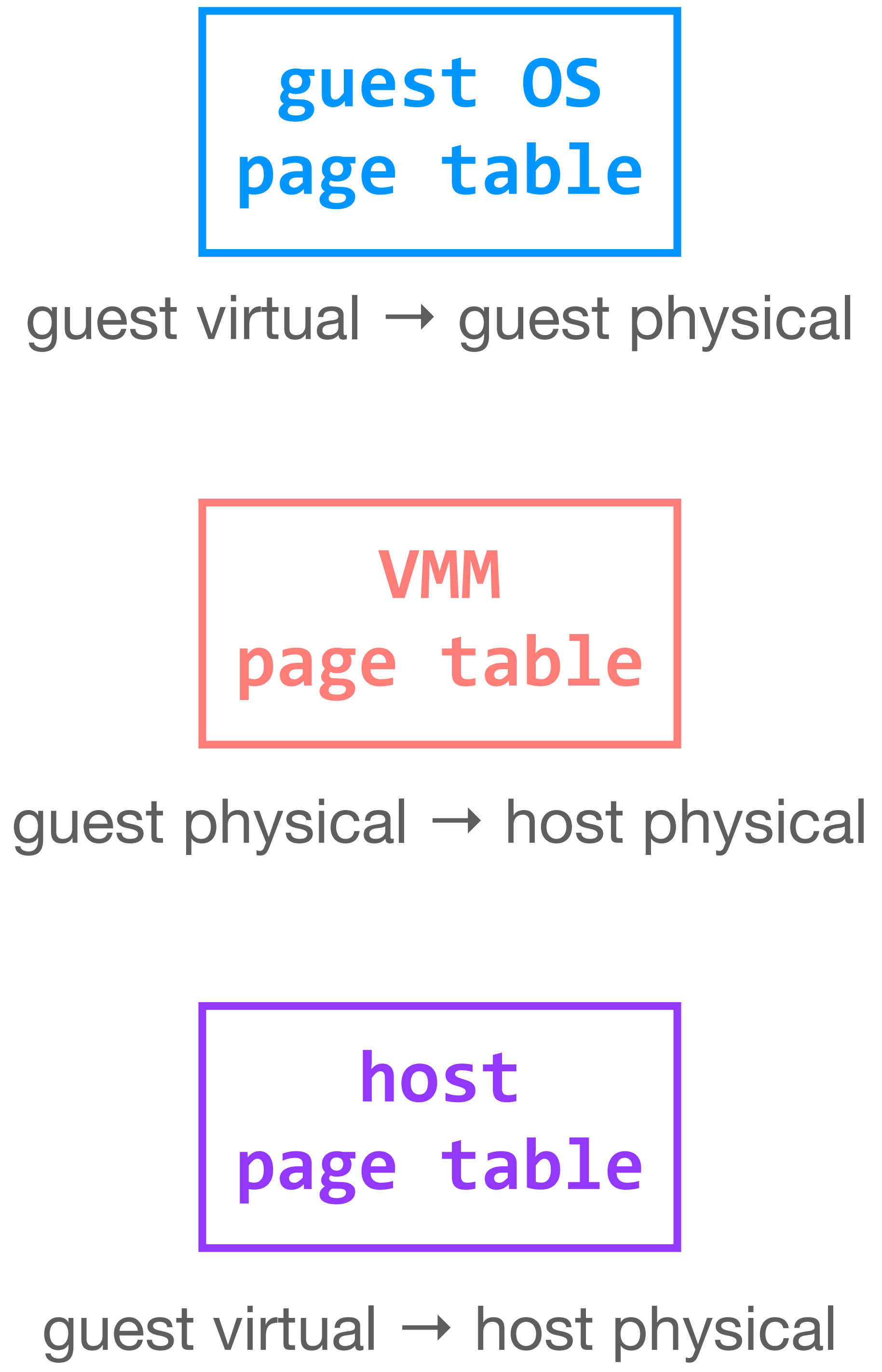


virtual machine monitor virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again!)

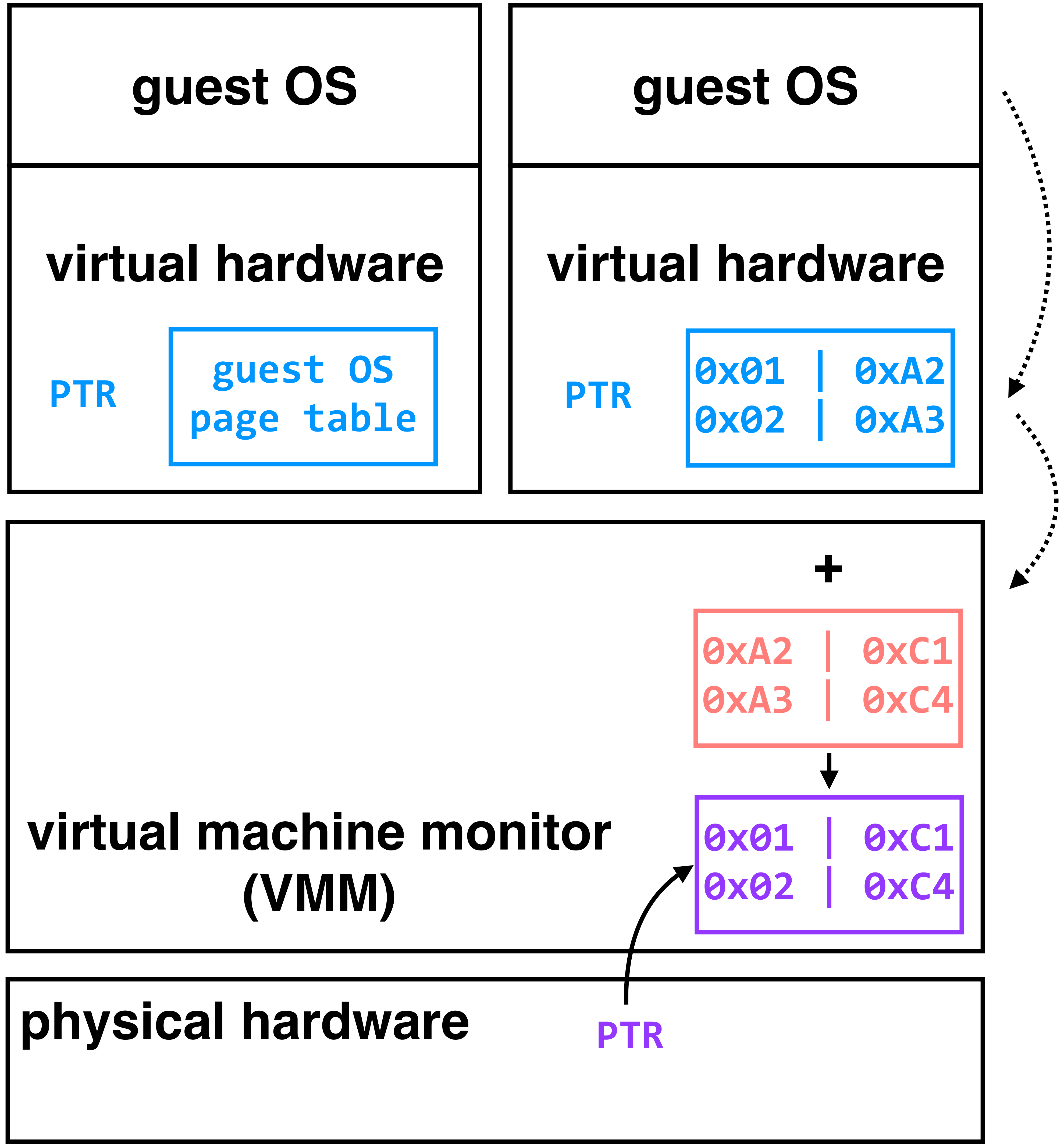


1. guest OS loads its PTR, which triggers an exception; the VMM intercepts
2. VMM combines the guest page table with its own page table to create a host page table
3. physical hardware uses the host page table

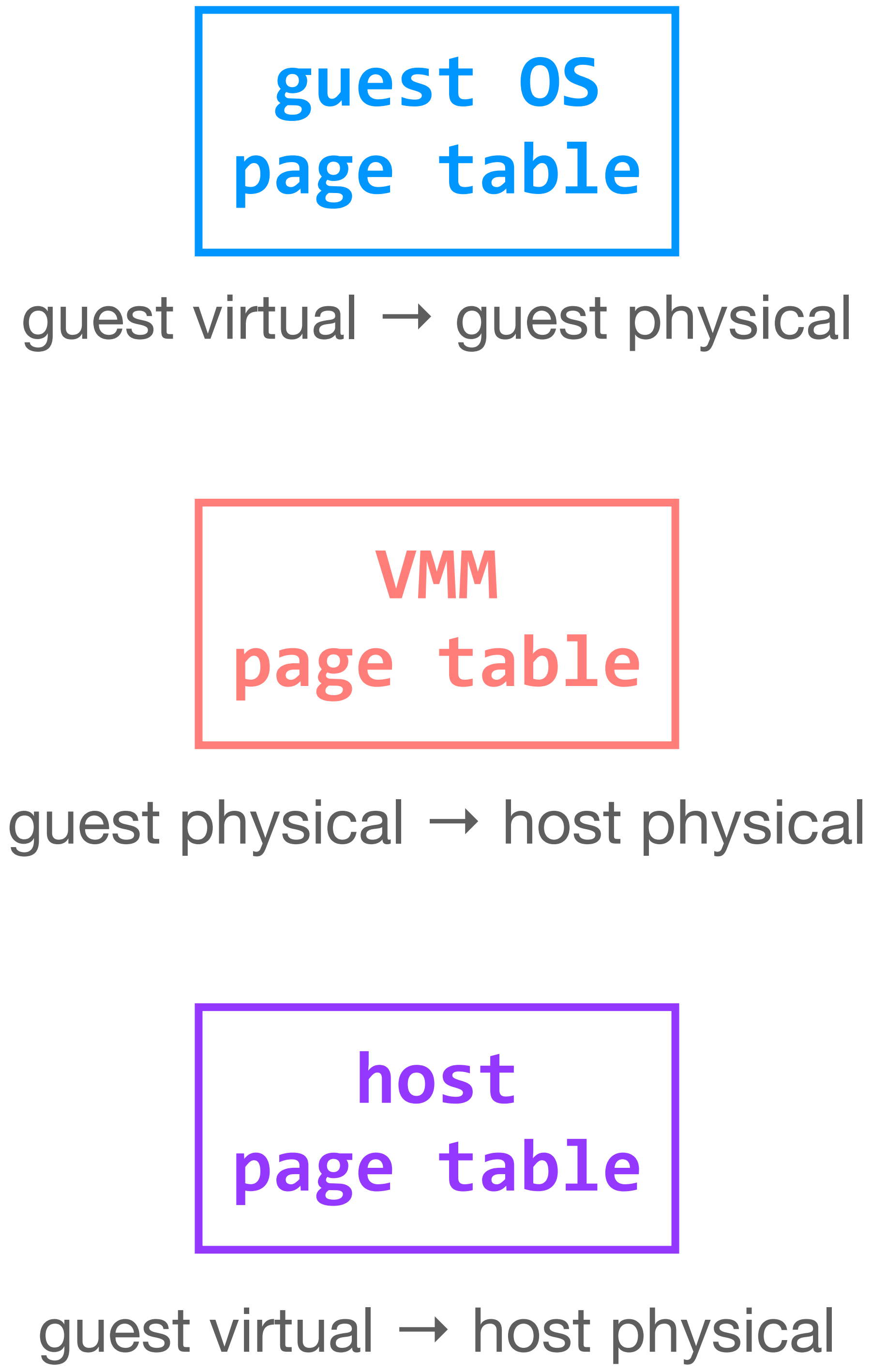


virtual machine monitor virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again!)

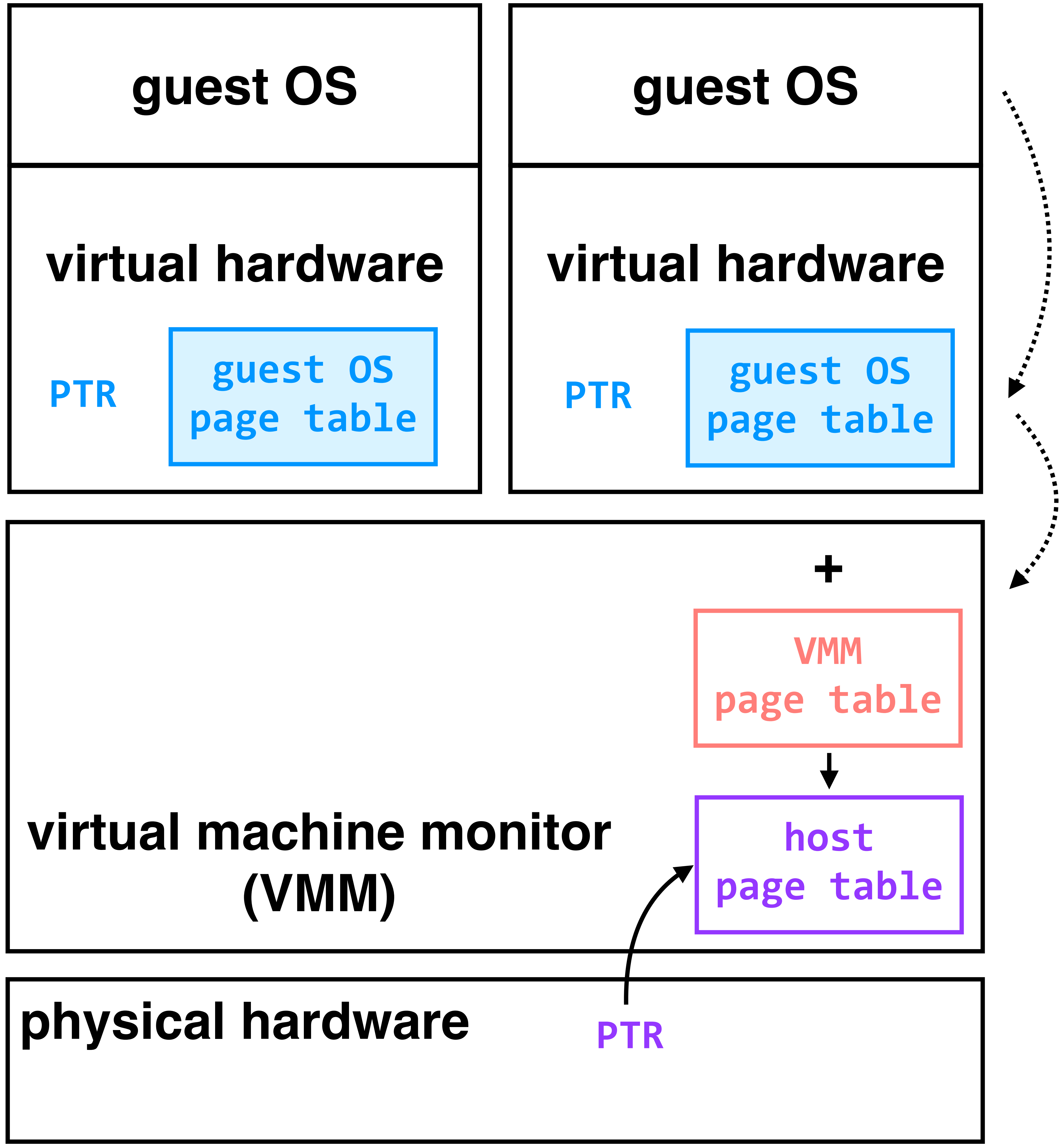


1. guest OS loads its PTR, which triggers an exception; the VMM intercepts
2. VMM combines the guest page table with its own page table to create a host page table
3. physical hardware uses the host page table



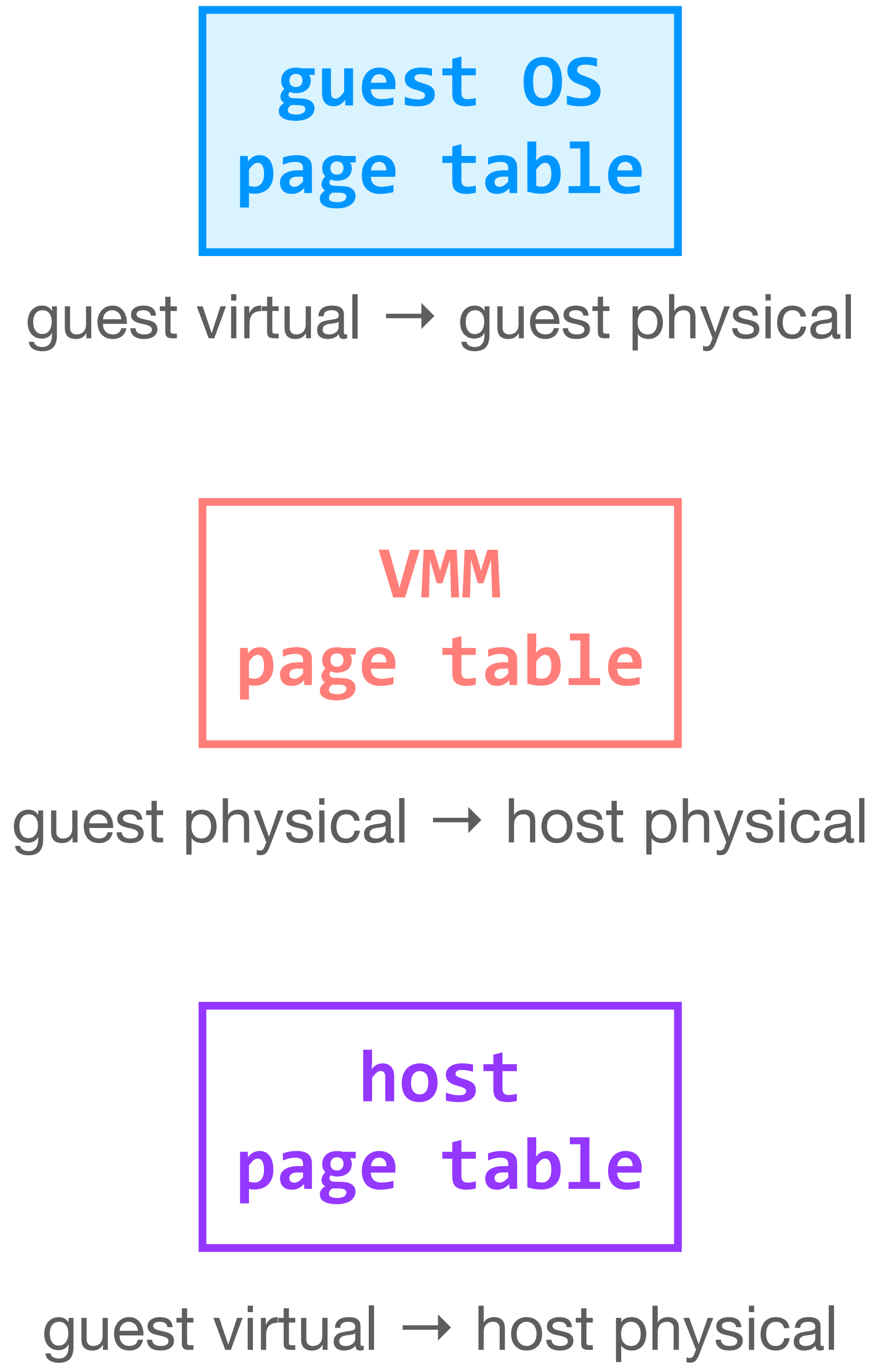
virtual machine monitor virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again!)



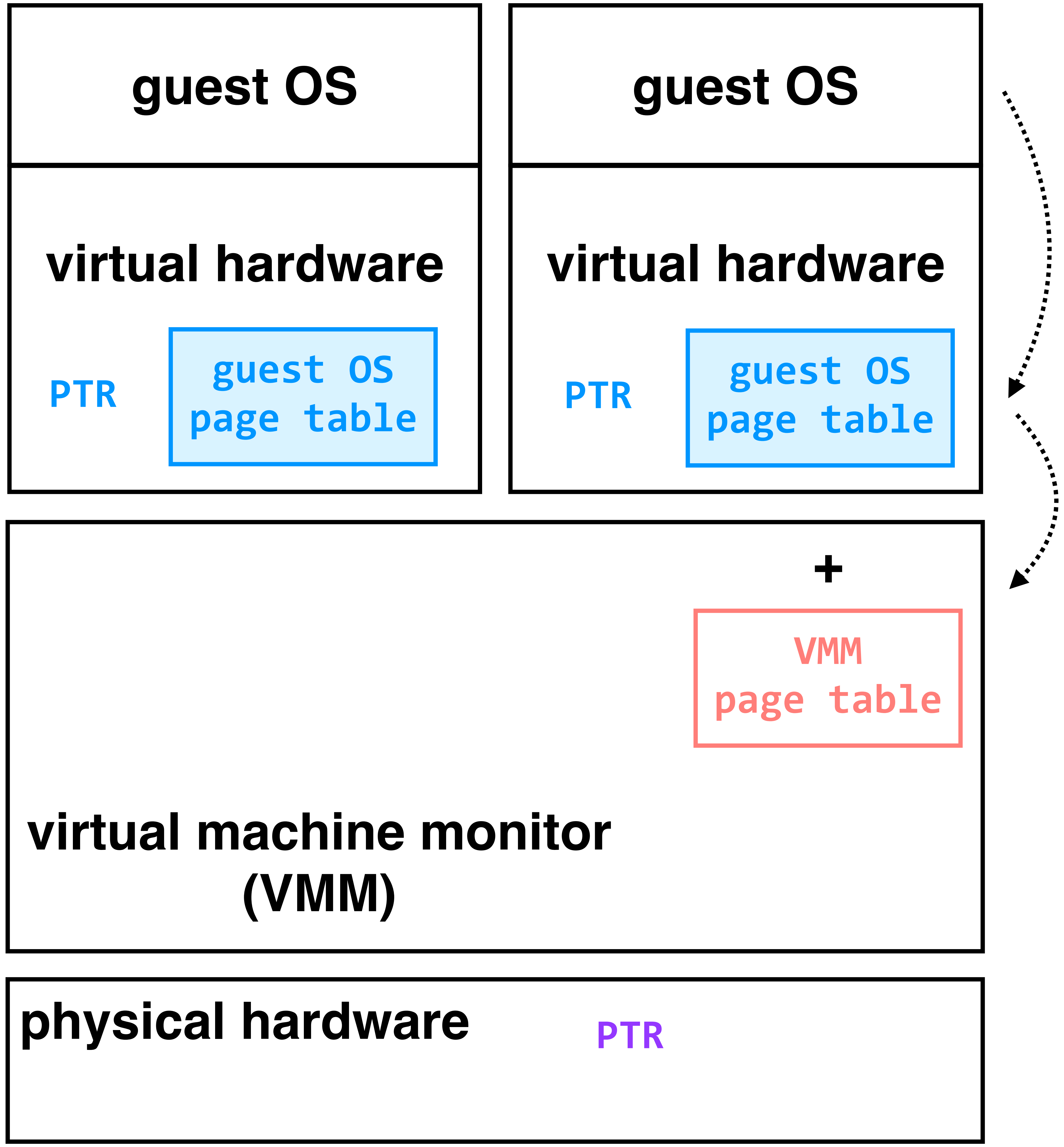
1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

guest OS page tables are marked as **read-only memory** so that modifications to these page tables also trigger exceptions (and thus allow the VMM to update the other tables)

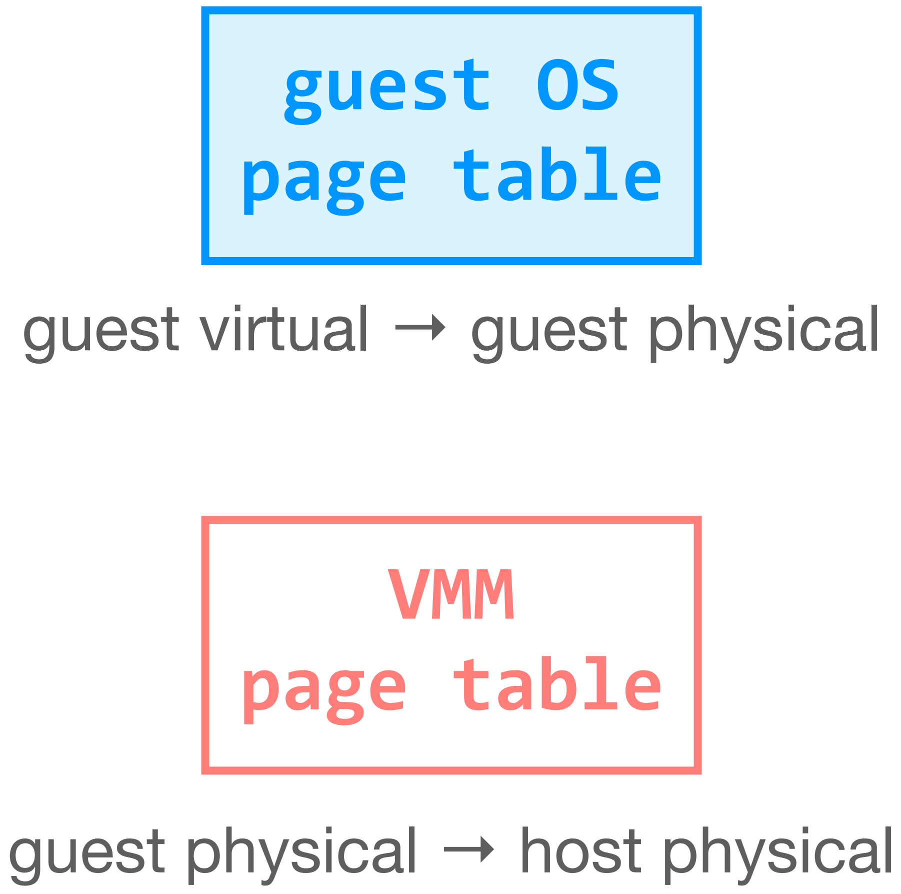


virtual machine monitor virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again!)

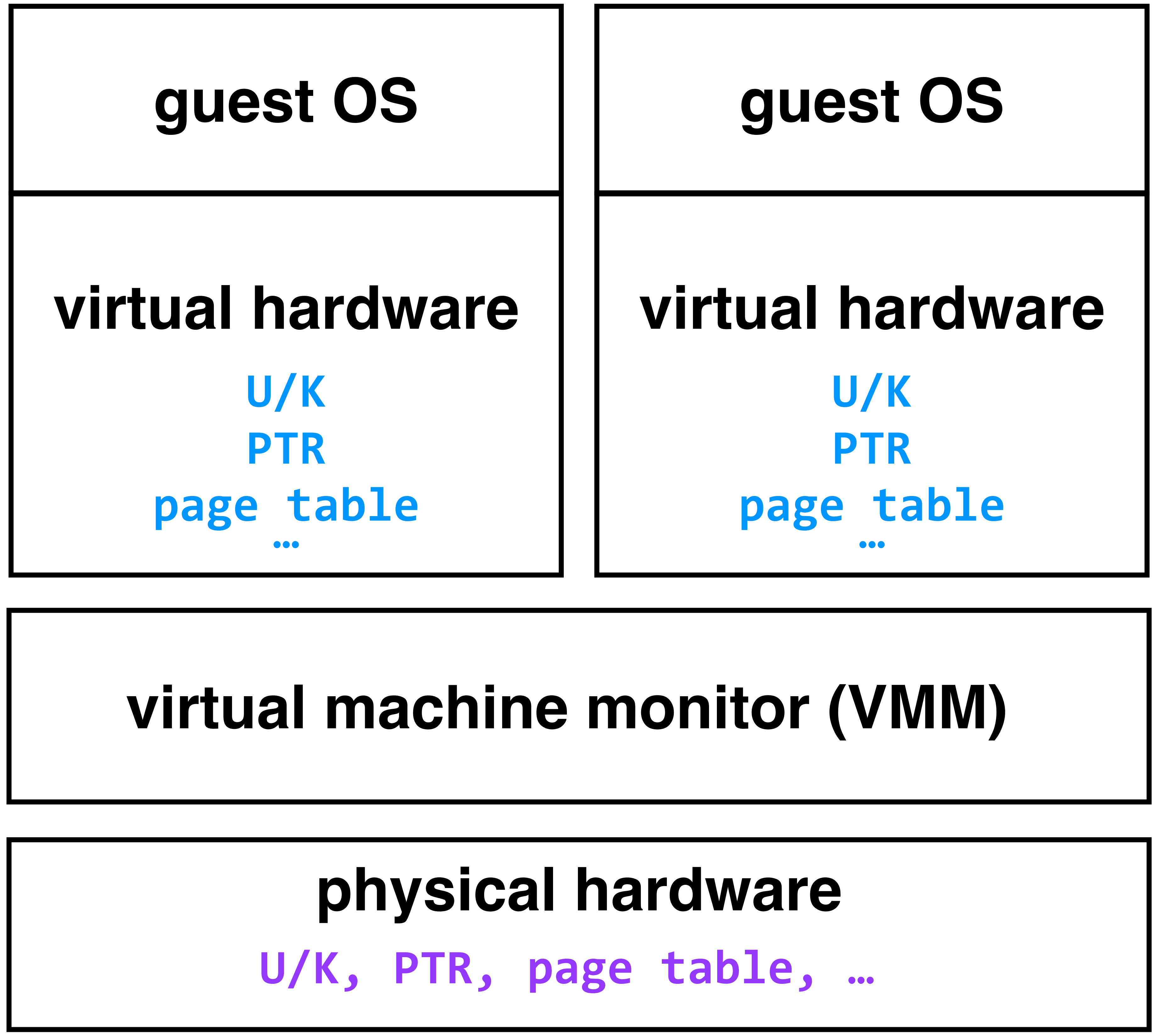


1. guest OS loads its PTR, which triggers an exception; the VMM intercepts



in modern hardware, the physical hardware is aware of both page tables, and performs the translation from guest virtual to host physical itself

virtual machine monitor virtualizes the physical hardware for the guest OSes



guest OSes run in user mode

privileged instructions in guest OS will cause an exception, which the VMM will intercept ("**trap**") and **emulate**

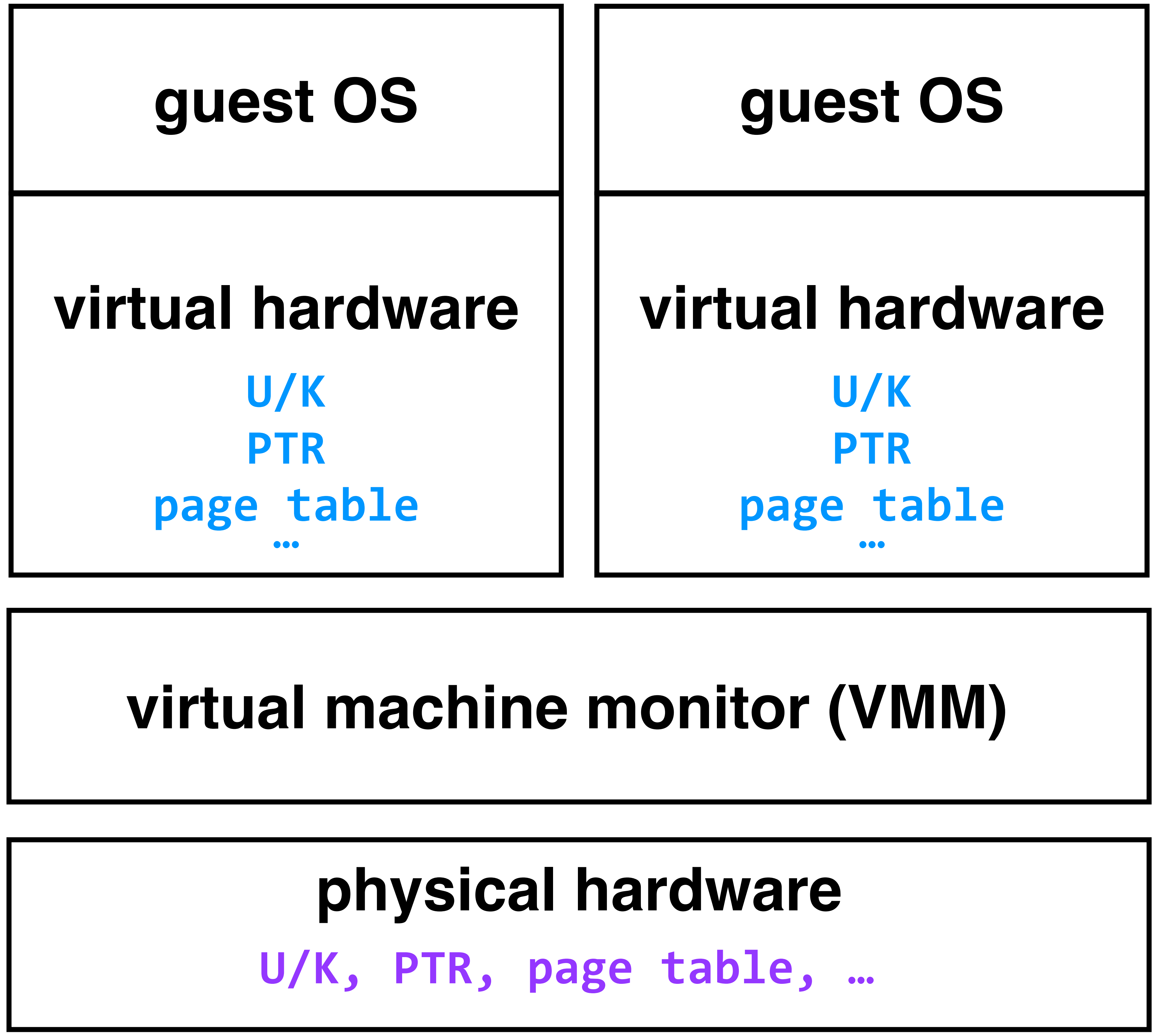
if the VMM *can't* emulate an instruction, it will send the exception back to the guest OS for handling

figuring out how to emulate an instruction is not enough; we also need to make sure that the VMM is trapping all relevant instructions

Katrina LaCurts | lacurts@mit.edu | 6.1800 2025

virtual machine monitor virtualizes the physical hardware for the guest OSes

second example: virtualizing the U/K bit



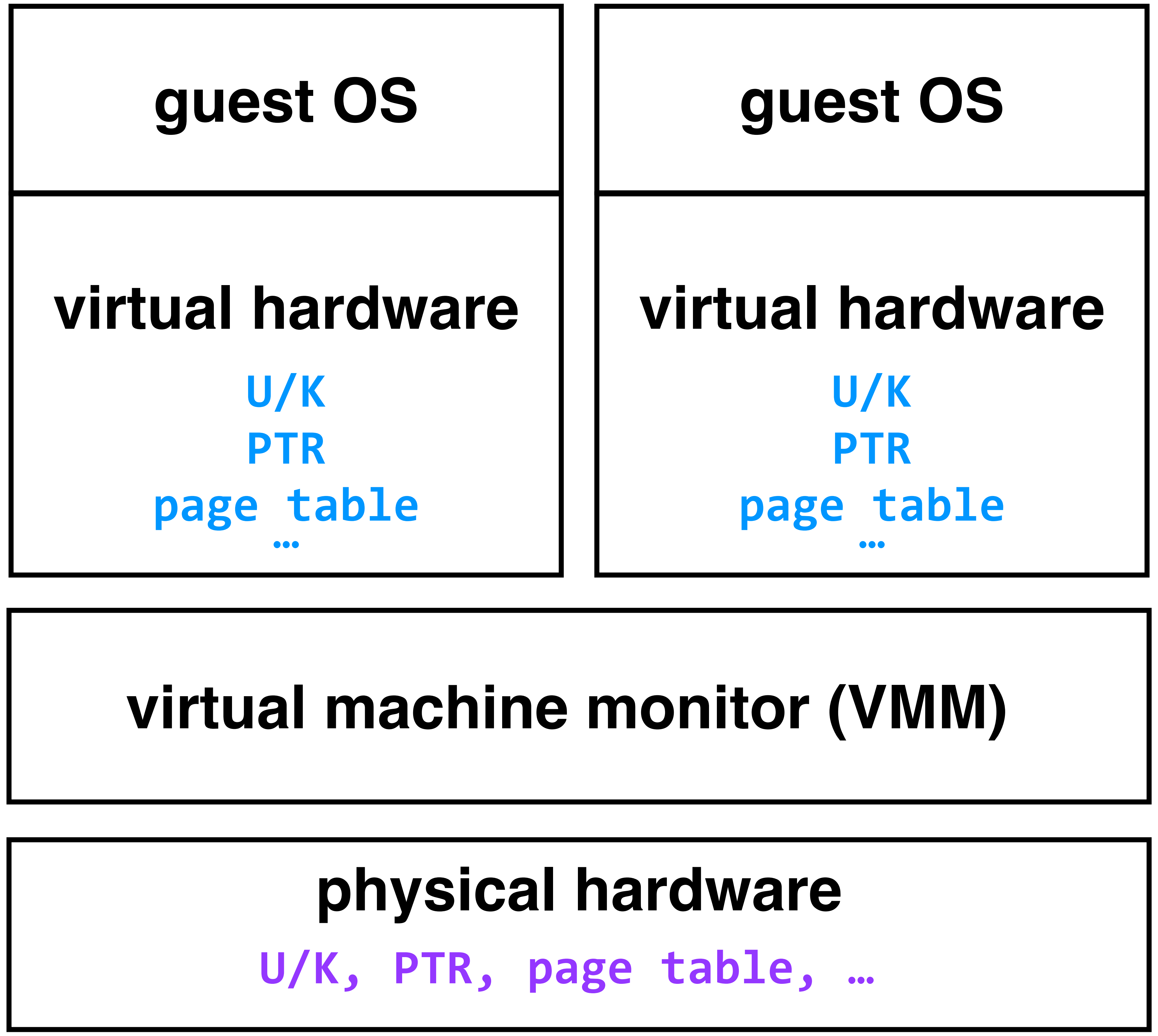
para-virtualization: modify guest OS slightly

binary translation: VMM replaces problematic instructions with ones that it can trap and emulate

hardware support: architecture provides a special operating mode for VMMs in addition to user mode, kernel mode

figuring out how to emulate an instruction is not enough; we also need to make sure that the VMM is trapping all relevant instructions

virtual machine monitor virtualizes the physical hardware for the guest OSes



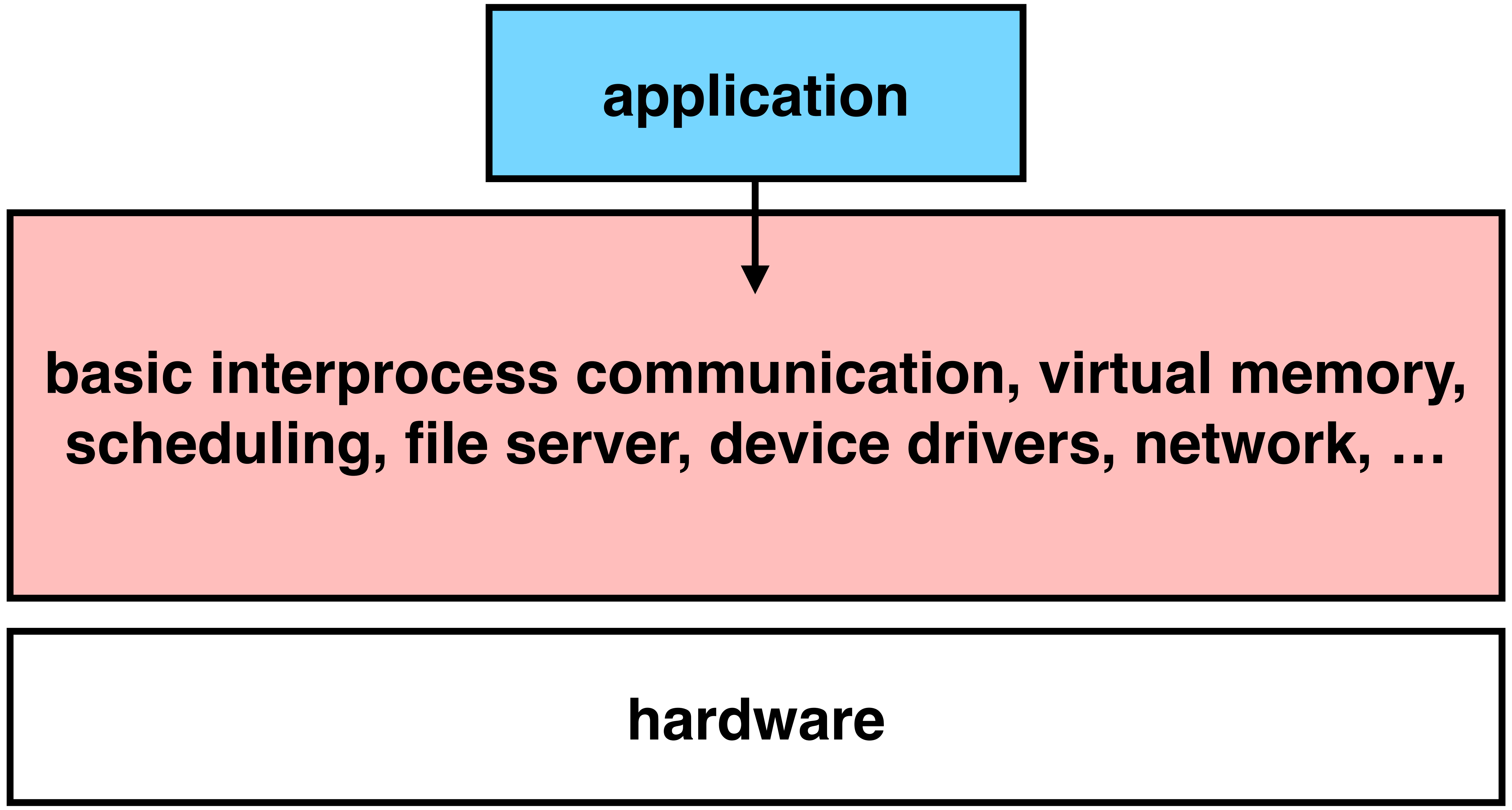
VMMs work by **trapping** and **emulating** important instructions

the actual **emulation** looks different depending on what we're trying to do. at times — e.g., in the case of virtual memory — it's a fairly straightforward extension of what the OS does

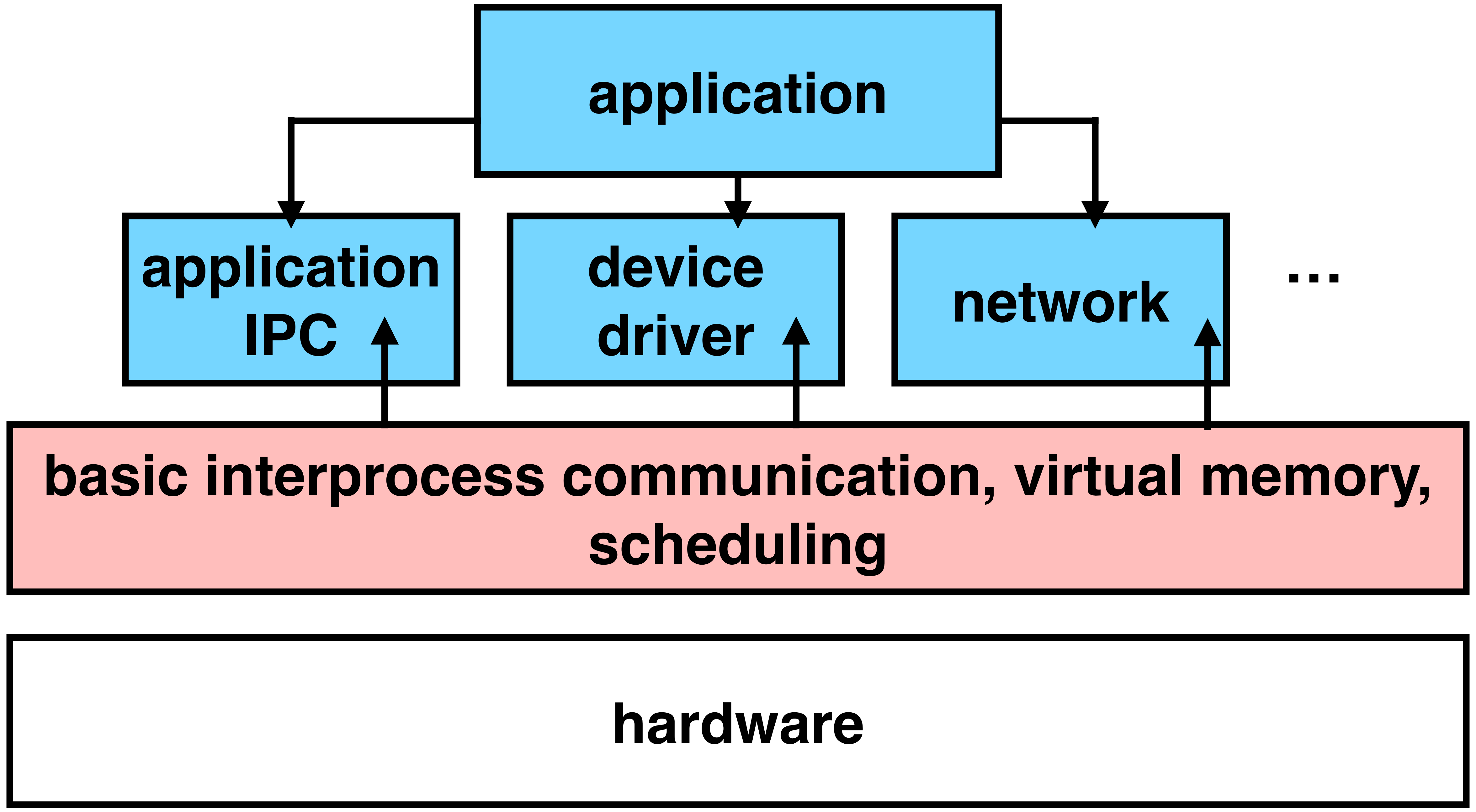
modern architectures build support for virtualization into their CPUs, which allow the VMM to operate **efficiently**

this is all yet another application of **virtualization**. the details change depending on what problem we're solving, but the goal of virtualization remains the same.

monolithic kernel: no enforced modularity within the kernel itself



microkernels: enforce modularity by putting subsystems in user programs



despite the modularity, it's not clear that redesigning an operating system from a monolithic kernel to a microkernel is a good idea, in part for reasons of **performance**

virtual machines allow us to run multiple **isolated** OSes on a single physical machine, similar to how we used an OS to run multiple programs on a single CPU

this set-up also enables many cloud compute infrastructures, which back many of the applications you use today

monolithic kernels provide no enforced modularity within the kernel. **microkernels** do, but redesigning monolithic kernels as microkernels is challenging

we have cared about **performance** in all aspects of our operating systems journey so far, and next time we'll start to think about performance more generally

you have now seen **virtualization** applied as a solution to many different problems. the details change depending on what problem we're solving, but the goal of virtualization remains the same.