# 6.033 Spring 2021

## Lecture #6: Virtual Machines and Performance

wrapping up operating systems

Katrina LaCurts | lacurts@mit.edu | 6.033 2021

# **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)

## **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

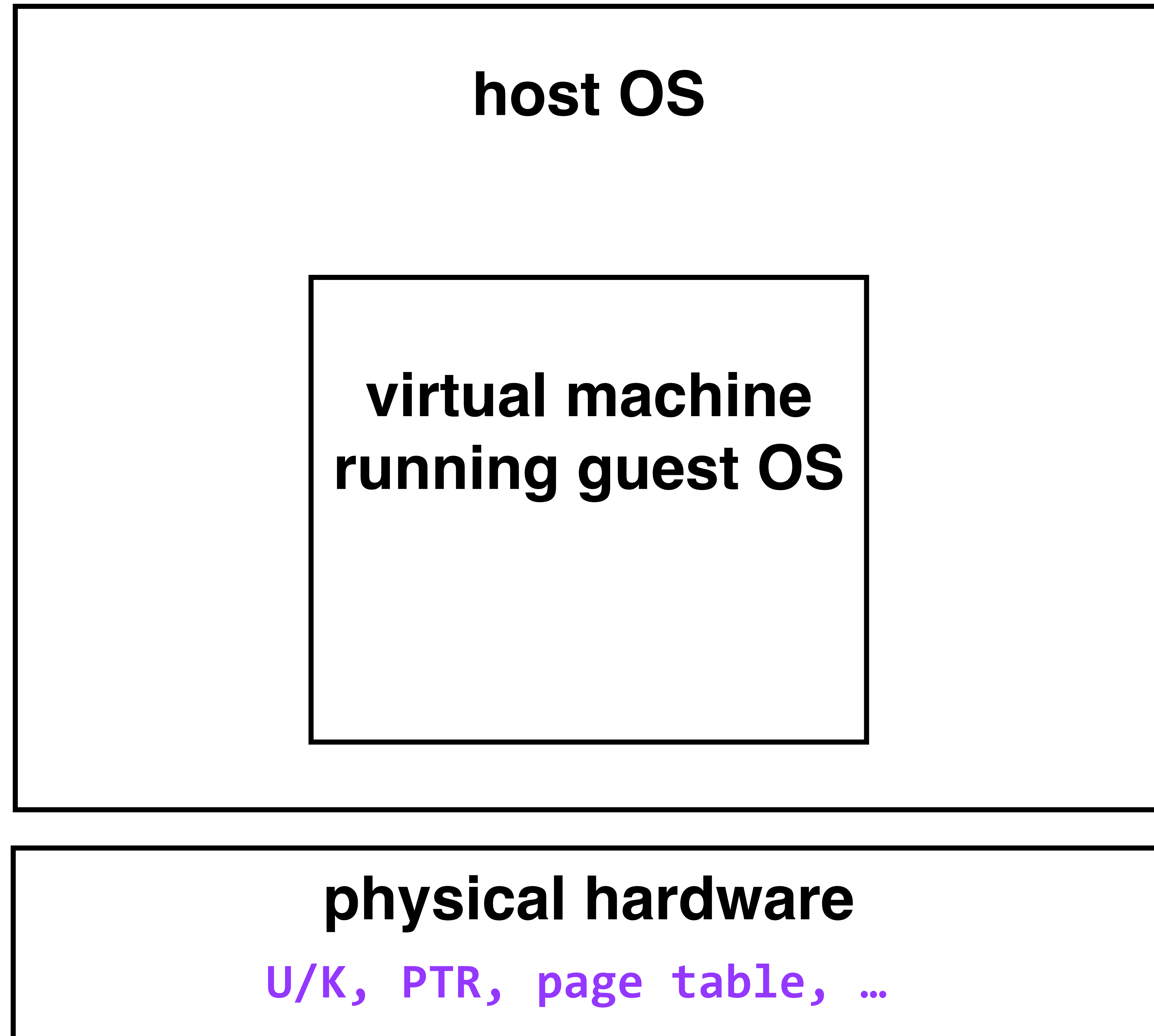
1. programs shouldn't be able to refer to (and corrupt) each others' **memory** → **virtual memory**

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

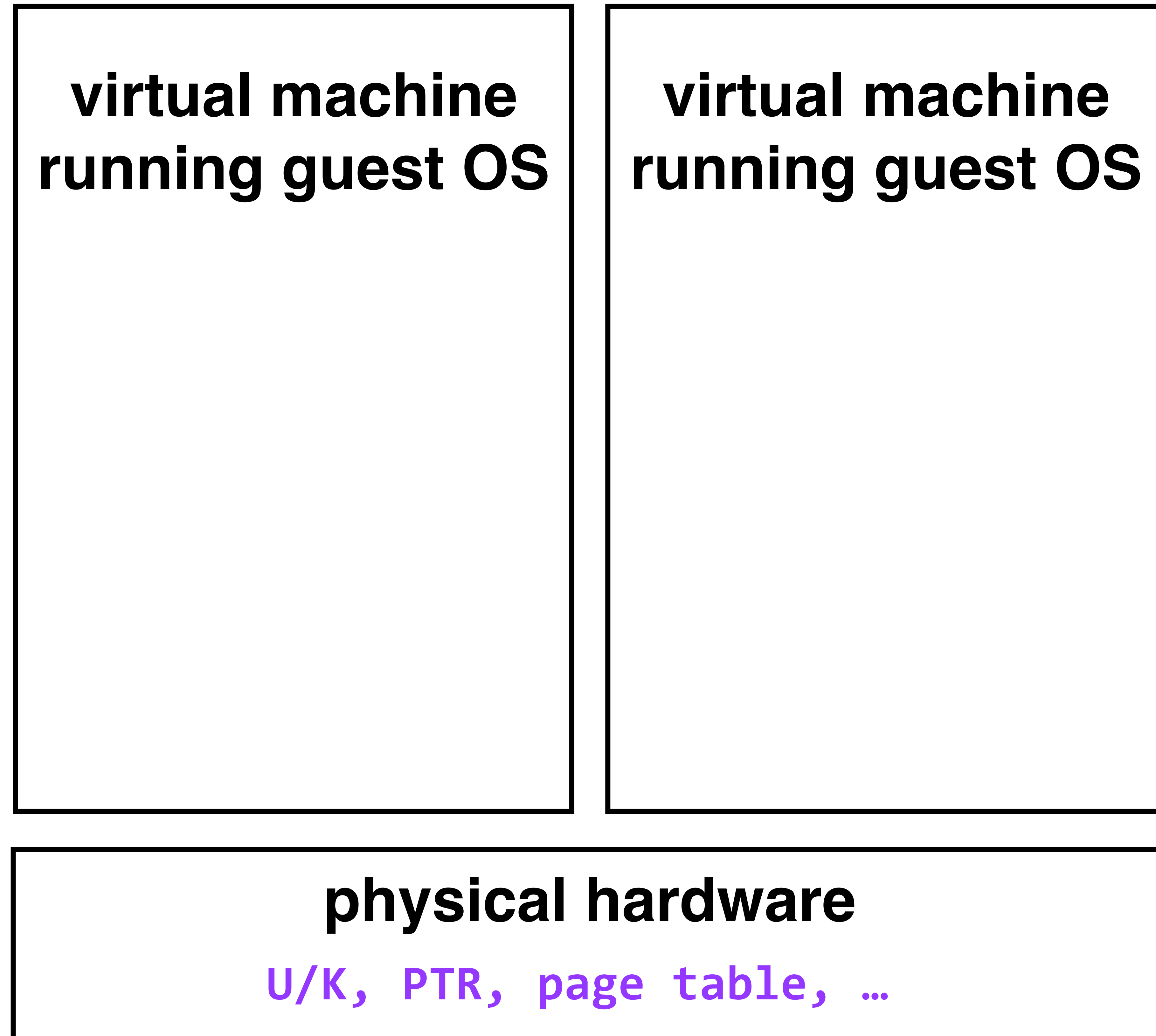
3. programs should be able to **share a CPU** without one program halting the progress of the others → **threads** (virtualize processors)

**today's goal:** run multiple operating systems at once

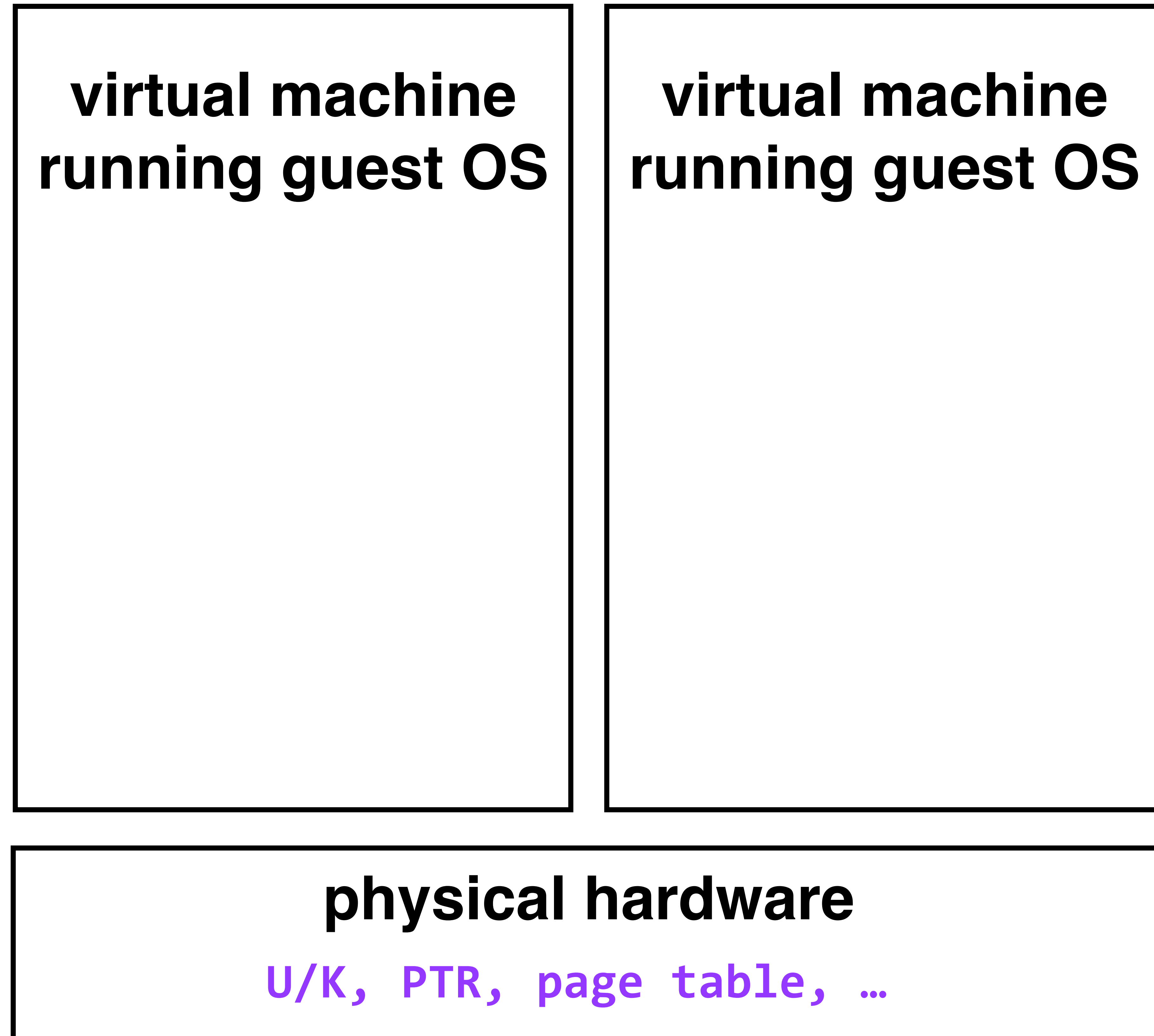
**common computing environment:** running multiple OSes on a single physical machine



**common computing environment:** running multiple OSes on a single physical machine

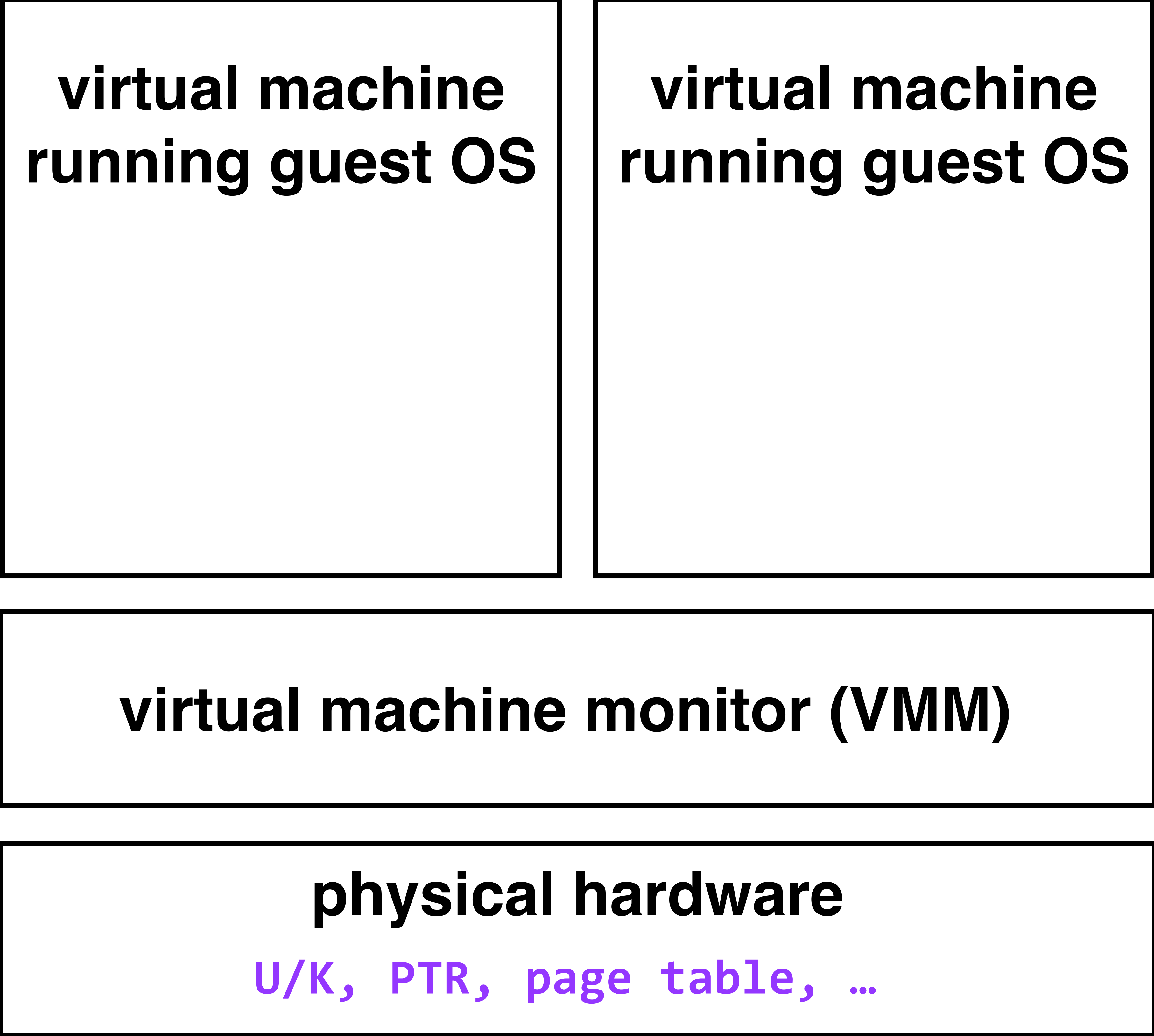


**common computing environment:** running multiple OSes on a single physical machine

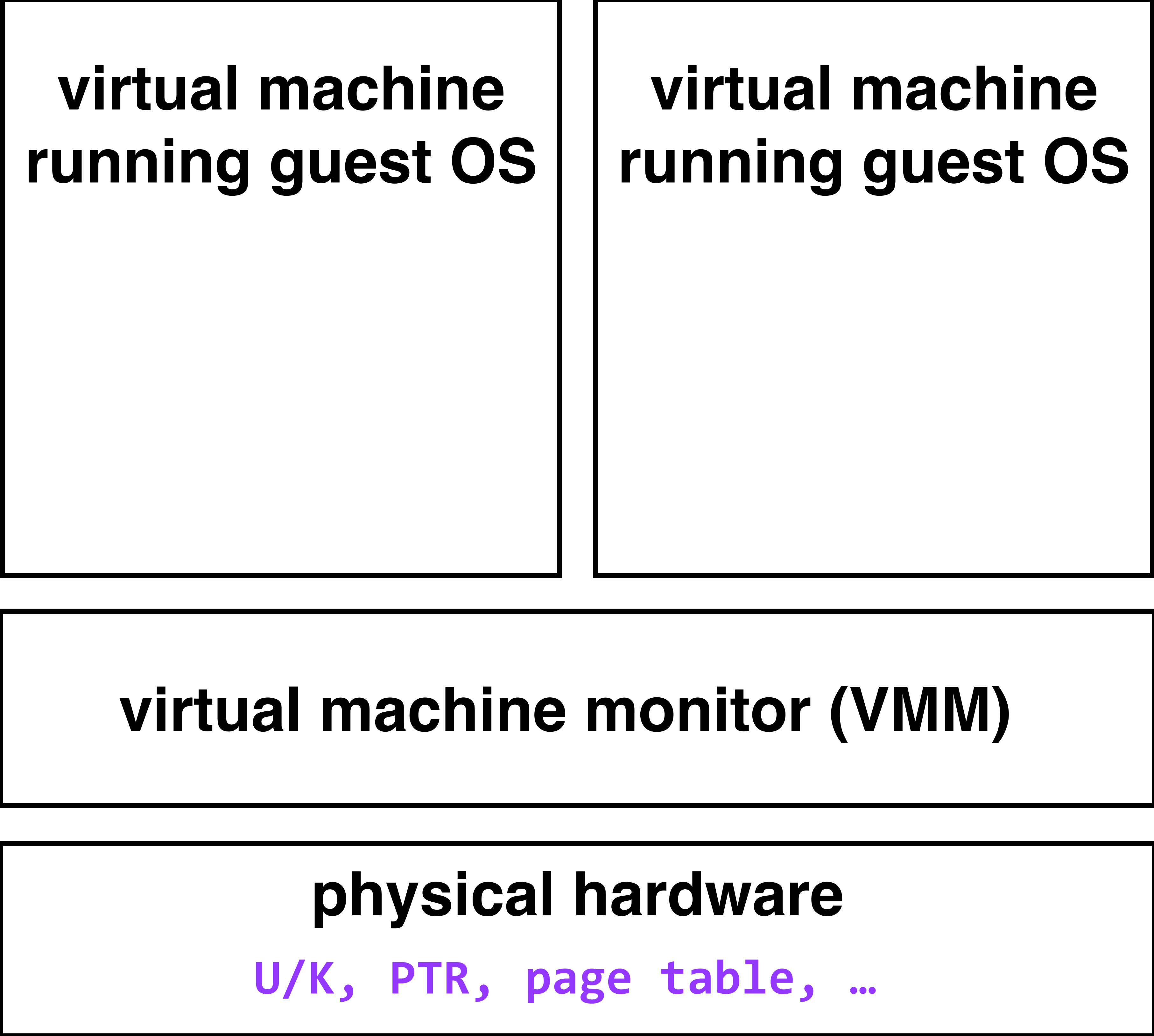



**problem:** how to (safely) share physical hardware?

**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

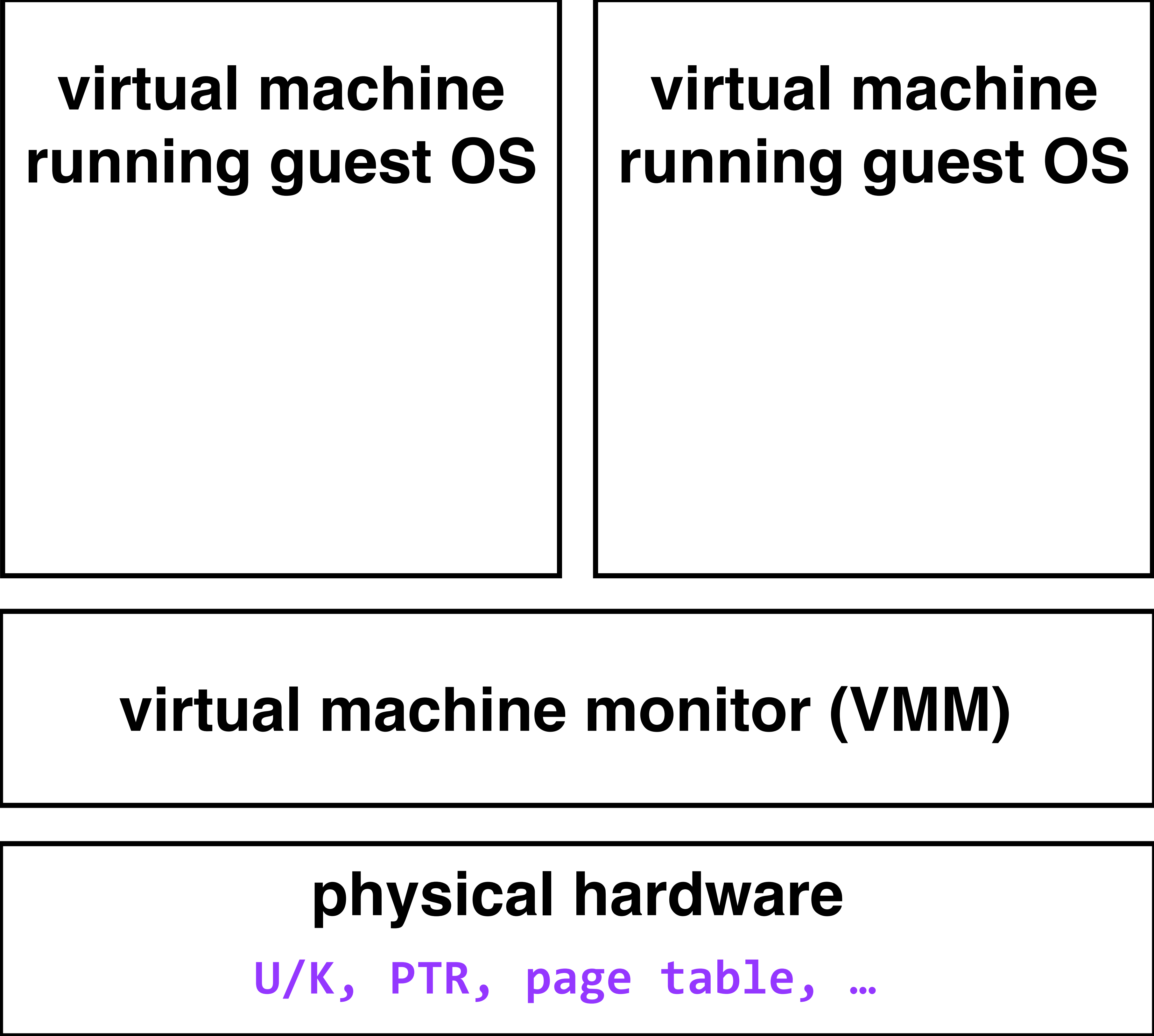


**virtual machine monitor:** virtualizes the physical hardware for the guest OSes




guest OSes run in user mode

# **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

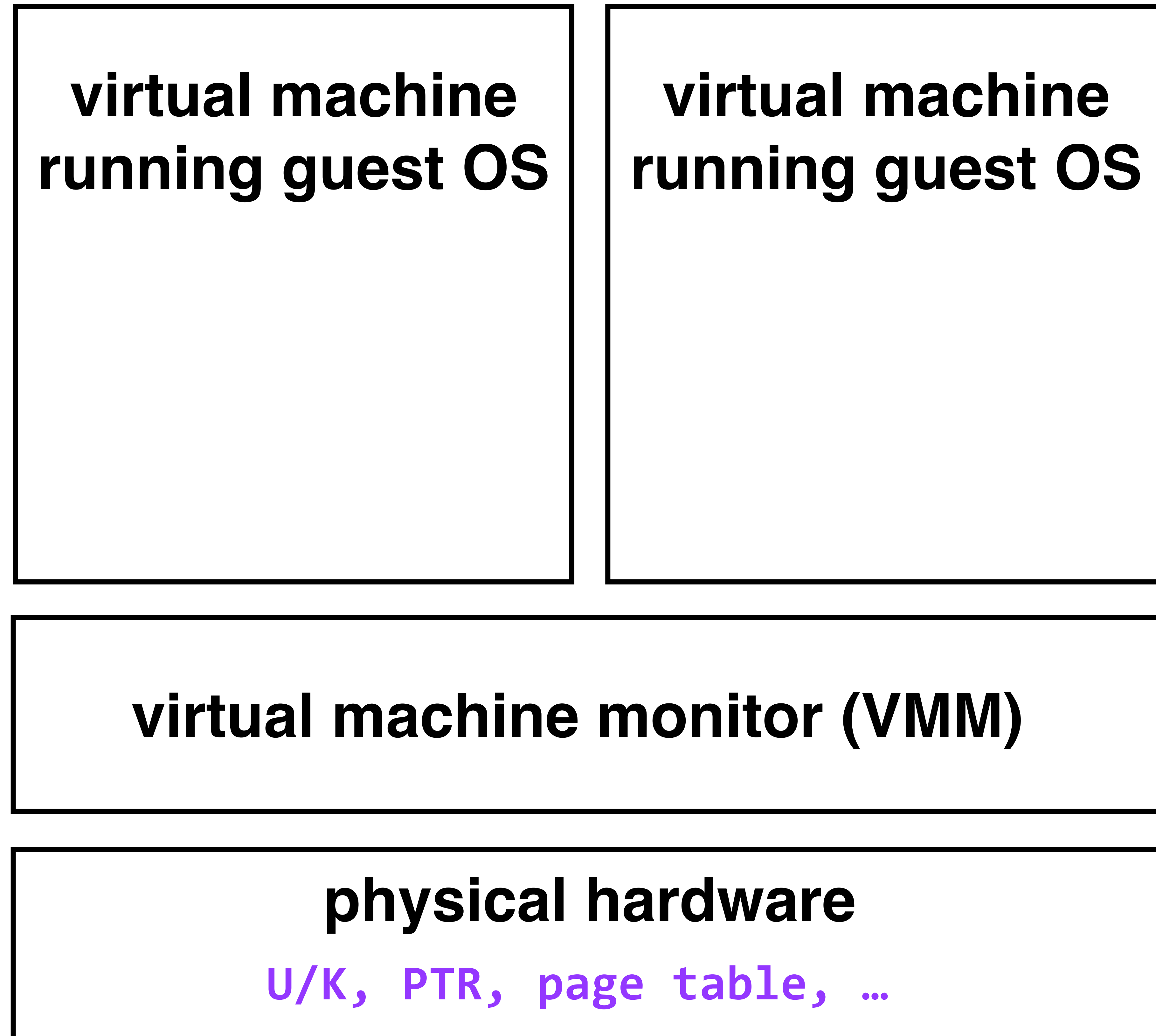



guest OSes run in user mode

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

**virtual machine monitor:** virtualizes the physical hardware for the guest OSes



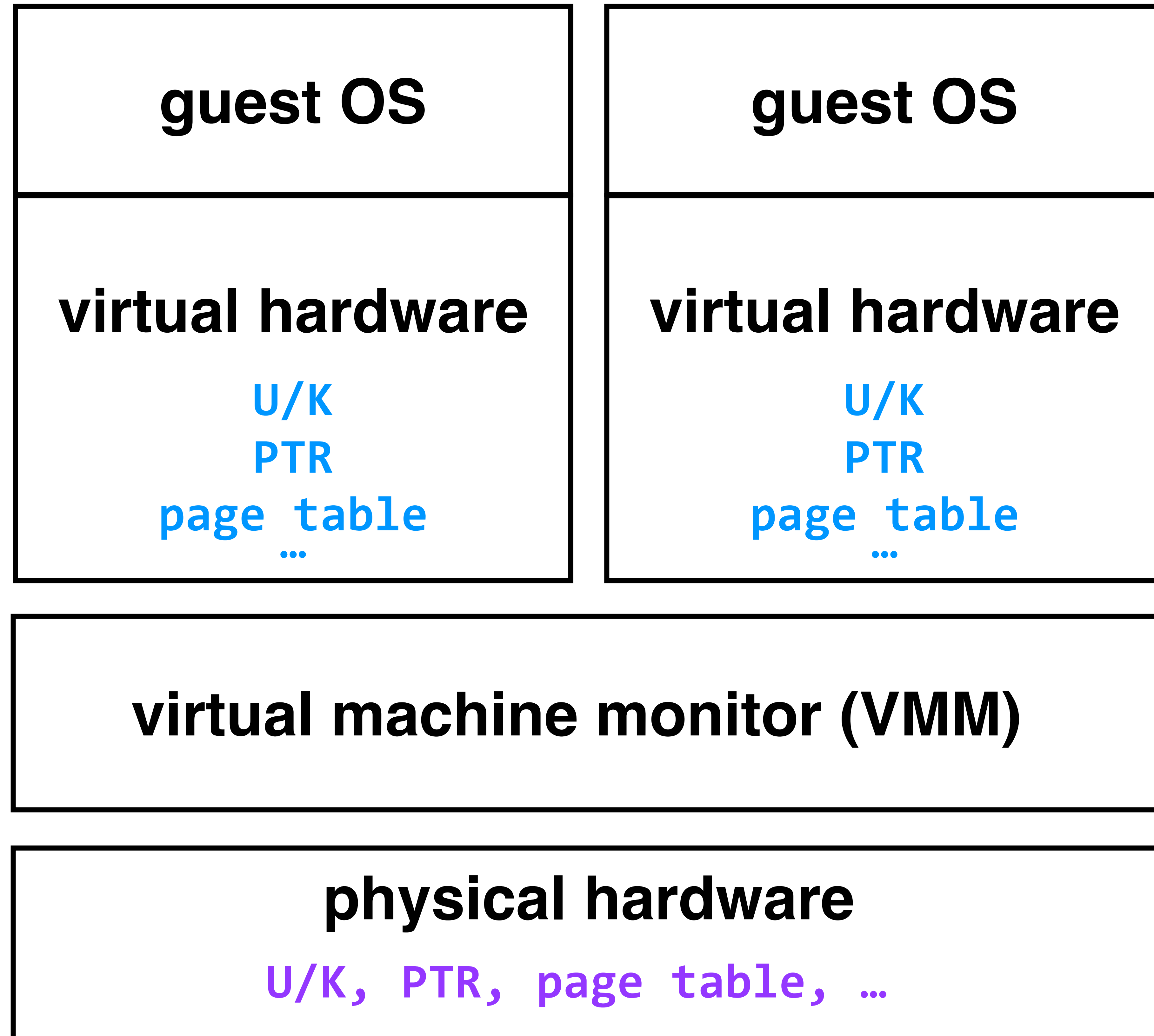

guest OSes run in user mode

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

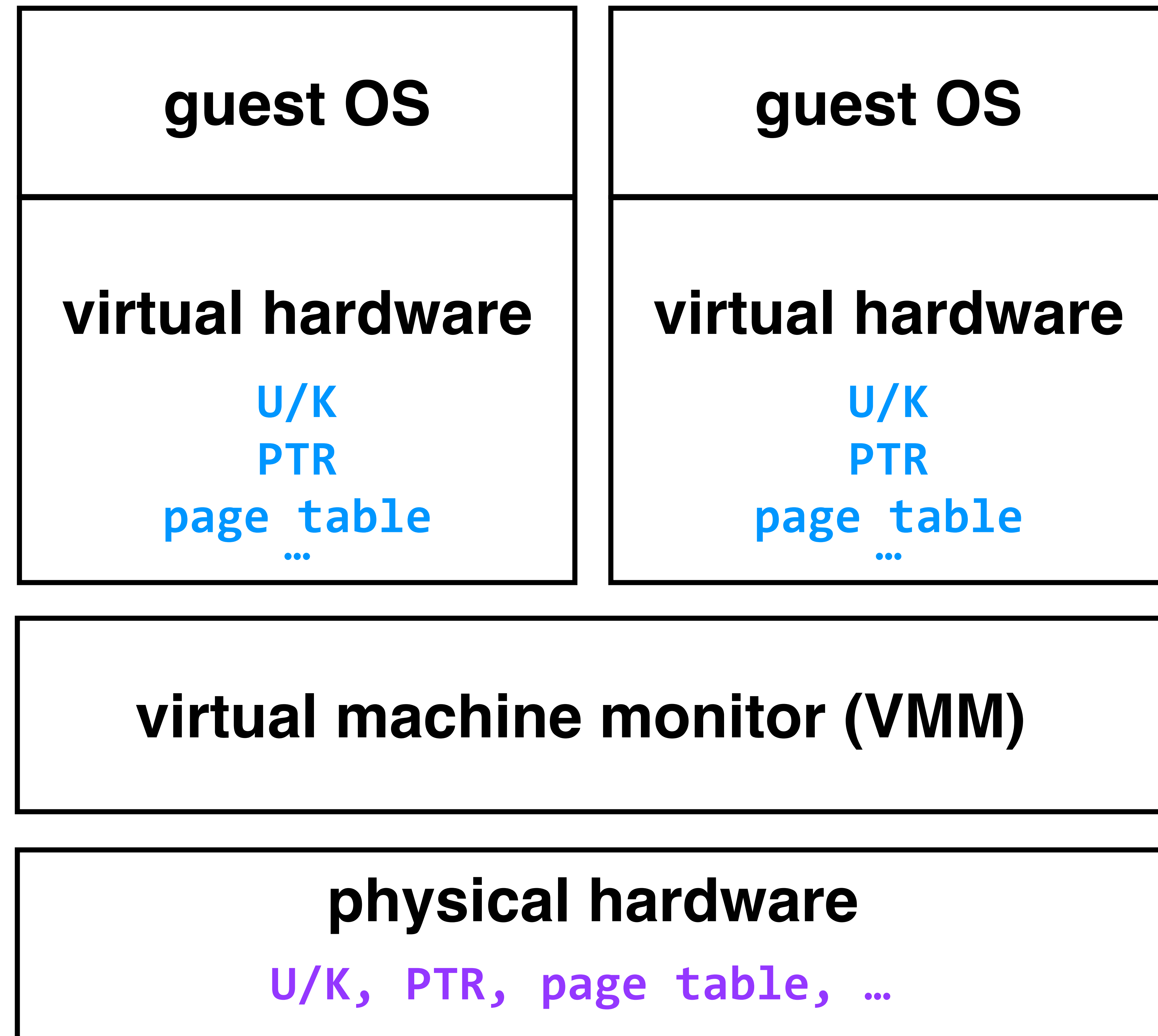
privileged instructions in guest OS will cause an exception, which the VMM will intercept ("**trap**") and **emulate**

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

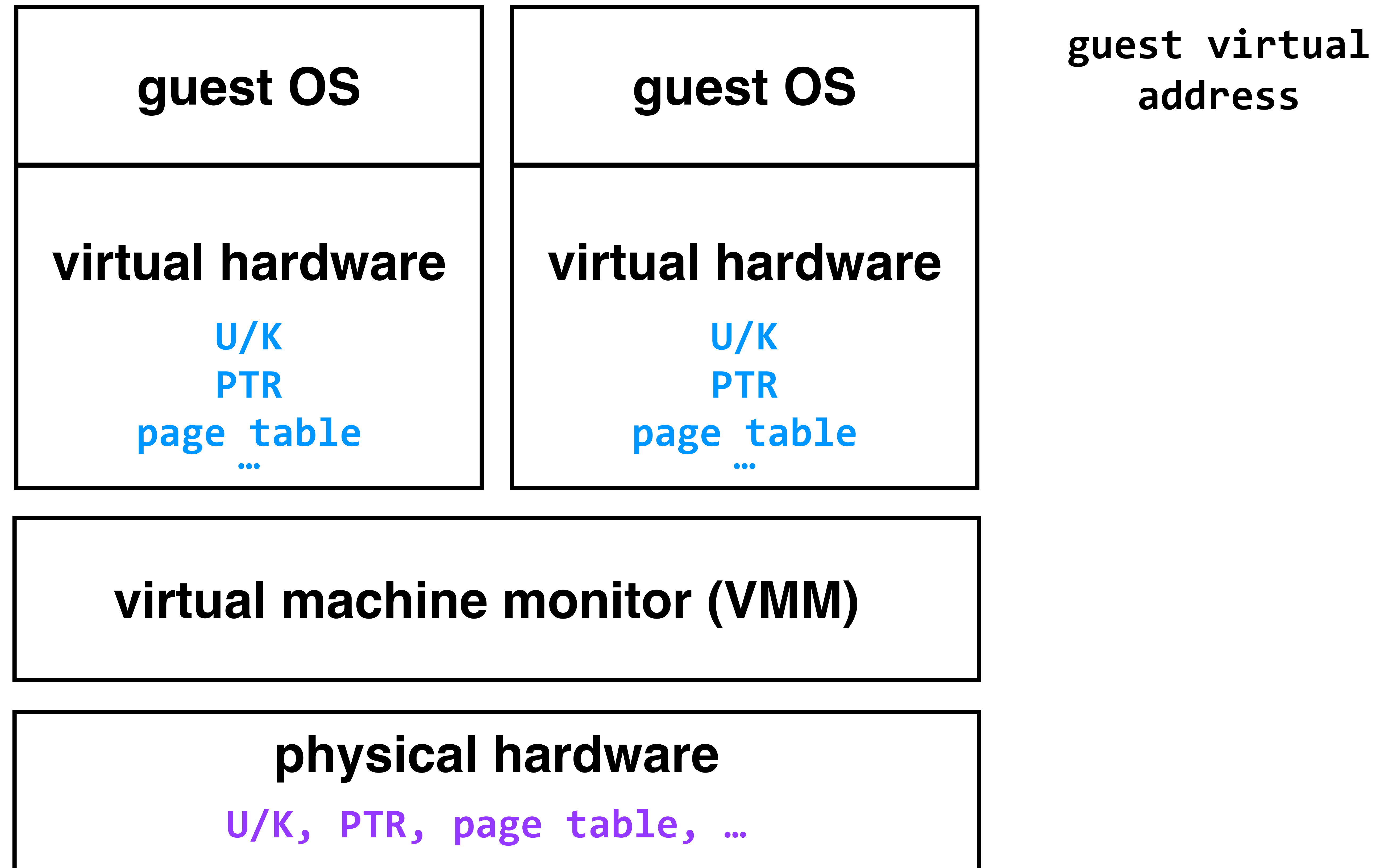
first example: virtualizing memory (again)




**first question:** what does it mean to emulate?

**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)

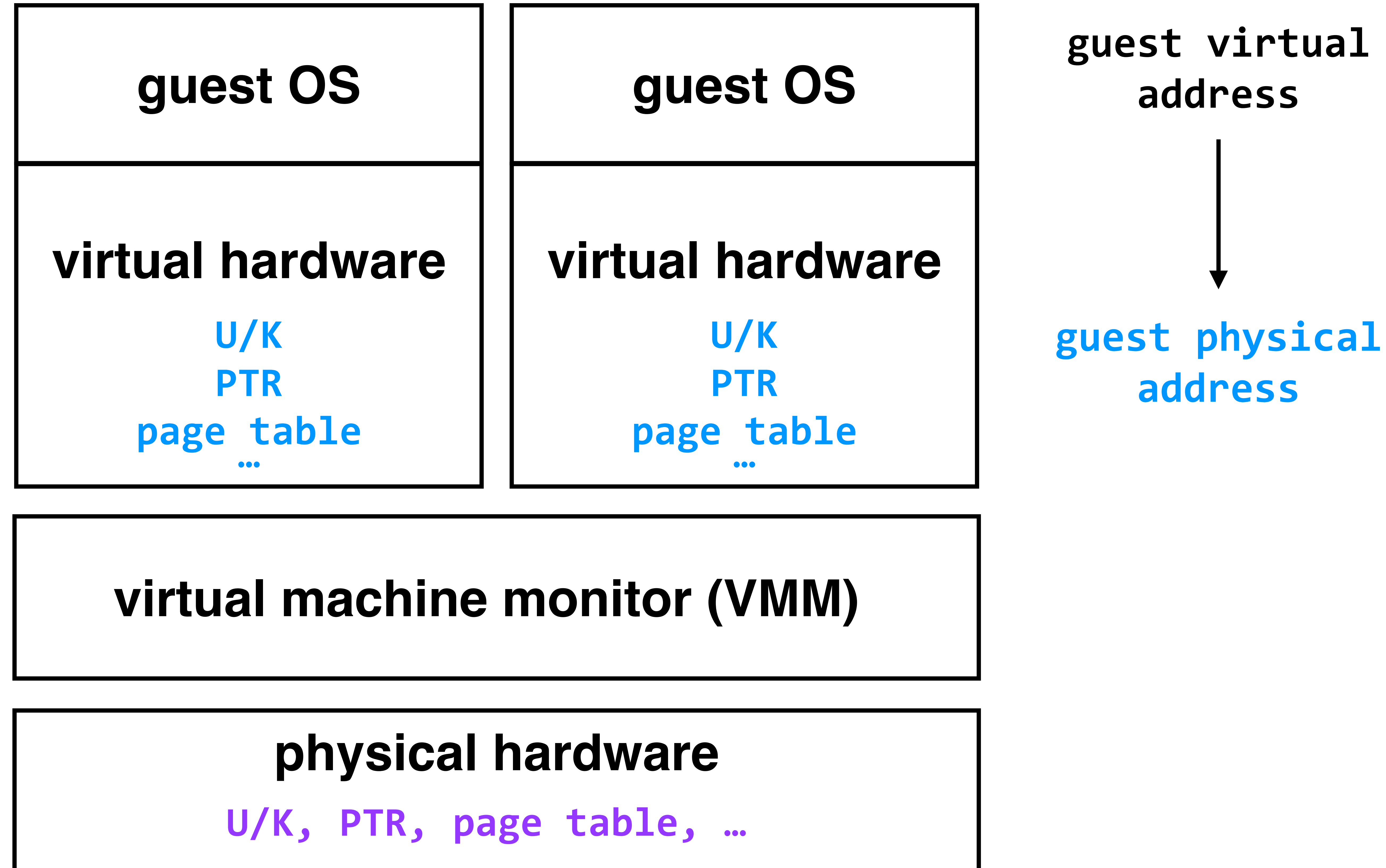



guest virtual address

**first question:** what does it mean to emulate?

**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)




guest virtual address

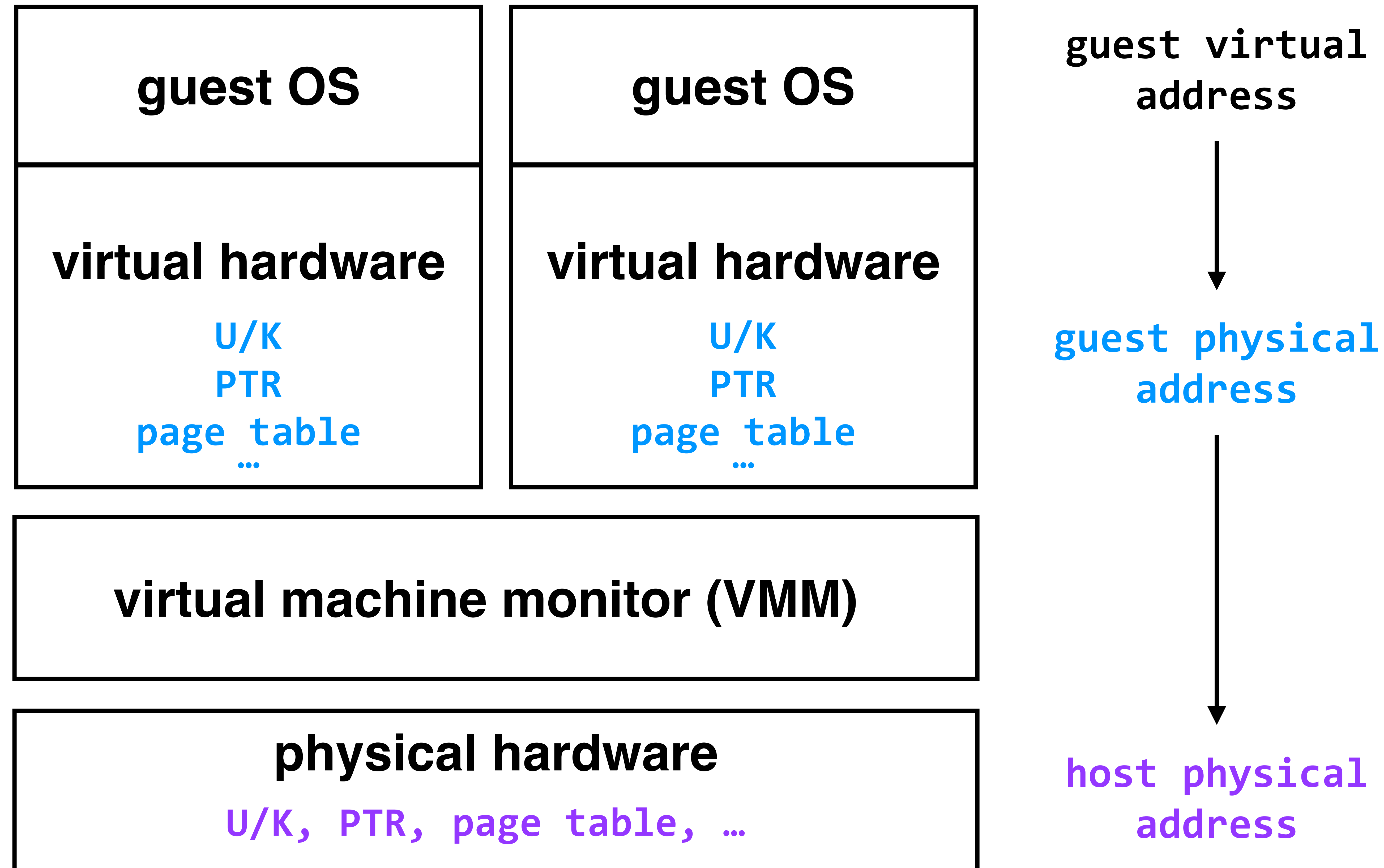
↓

guest physical address

**first question:** what does it mean to emulate?

# **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)




guest virtual address

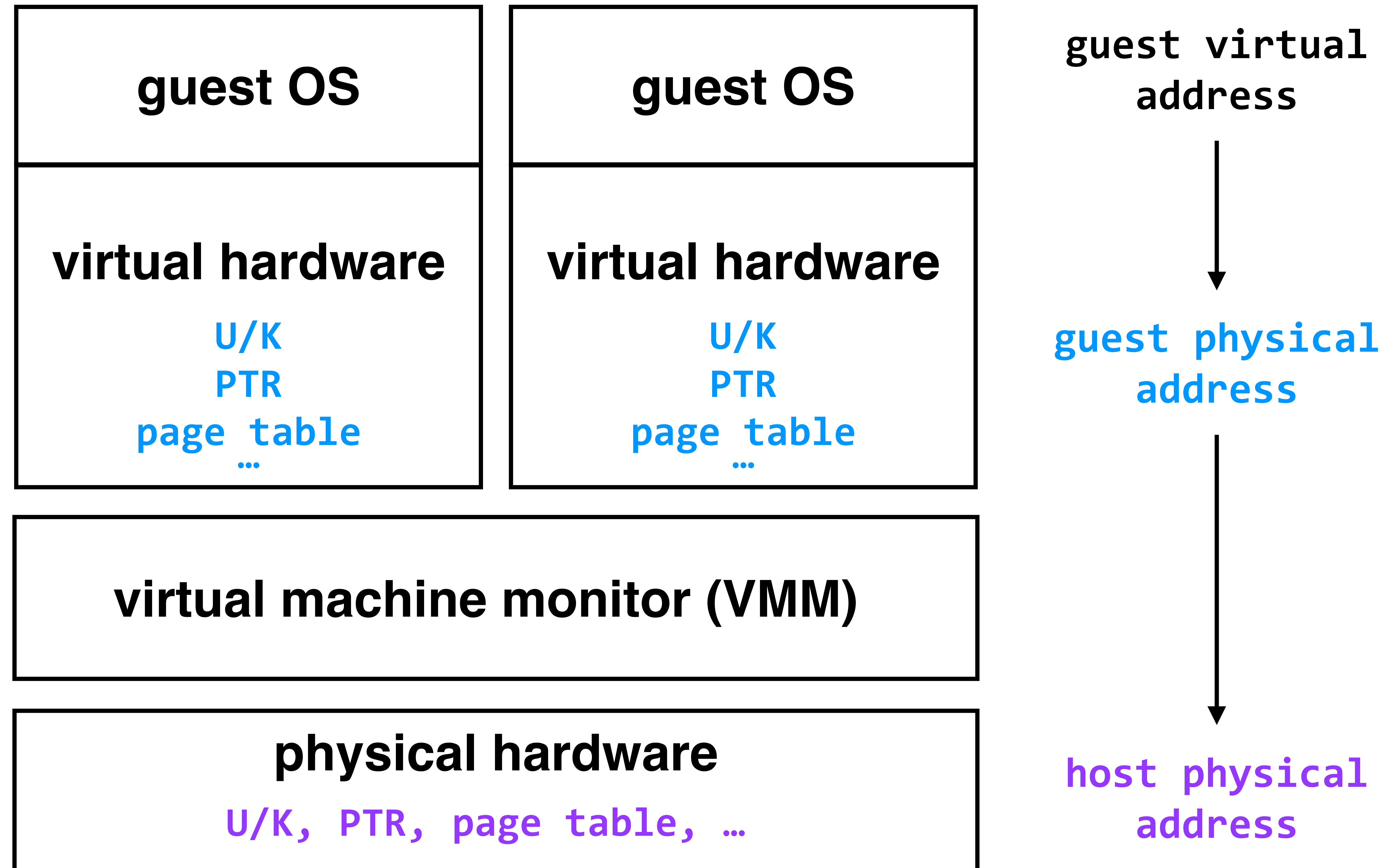
guest physical address

host physical address

**first question:** what does it mean to emulate?

**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)




guest virtual address

guest physical address

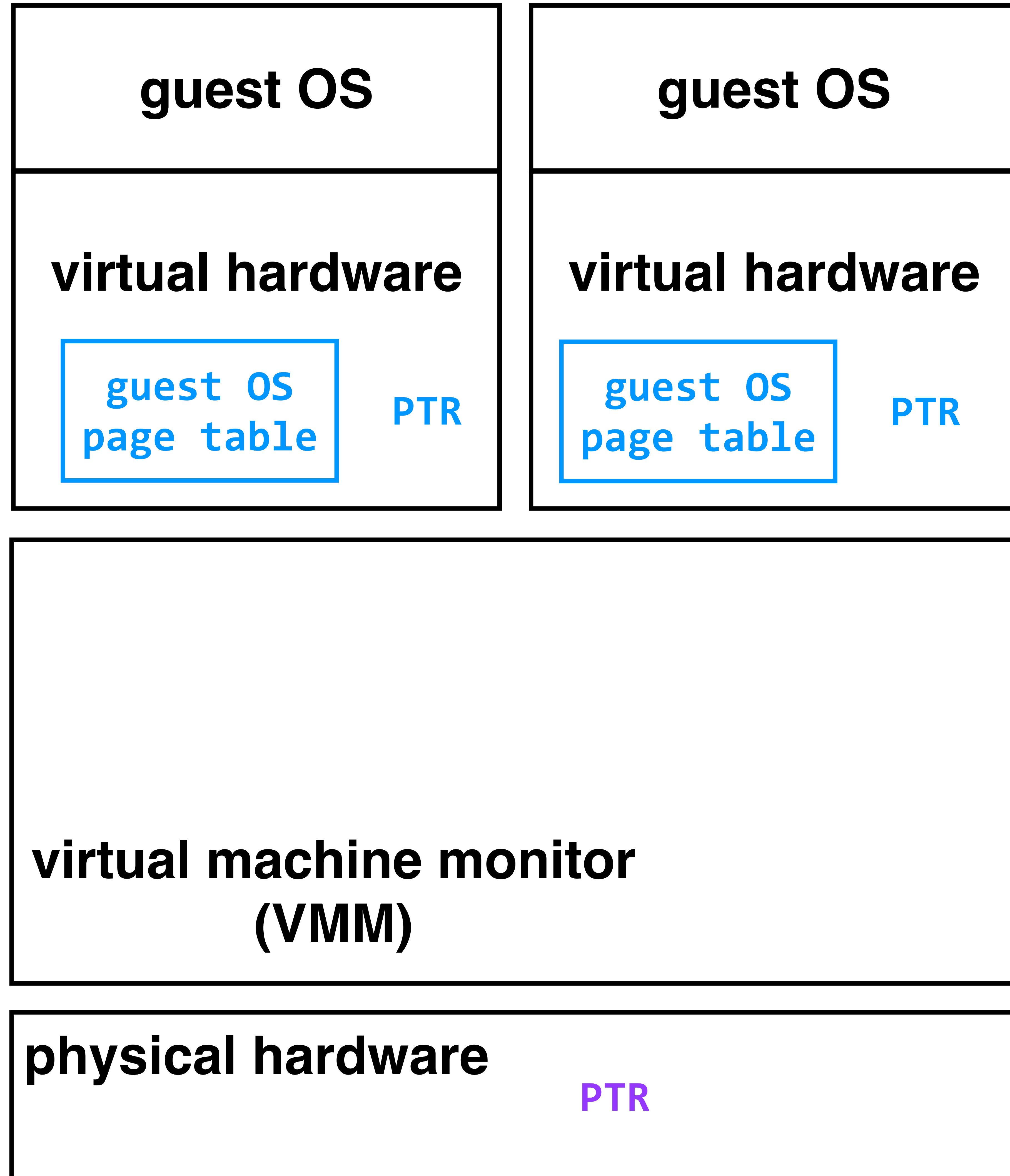
host physical address

**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

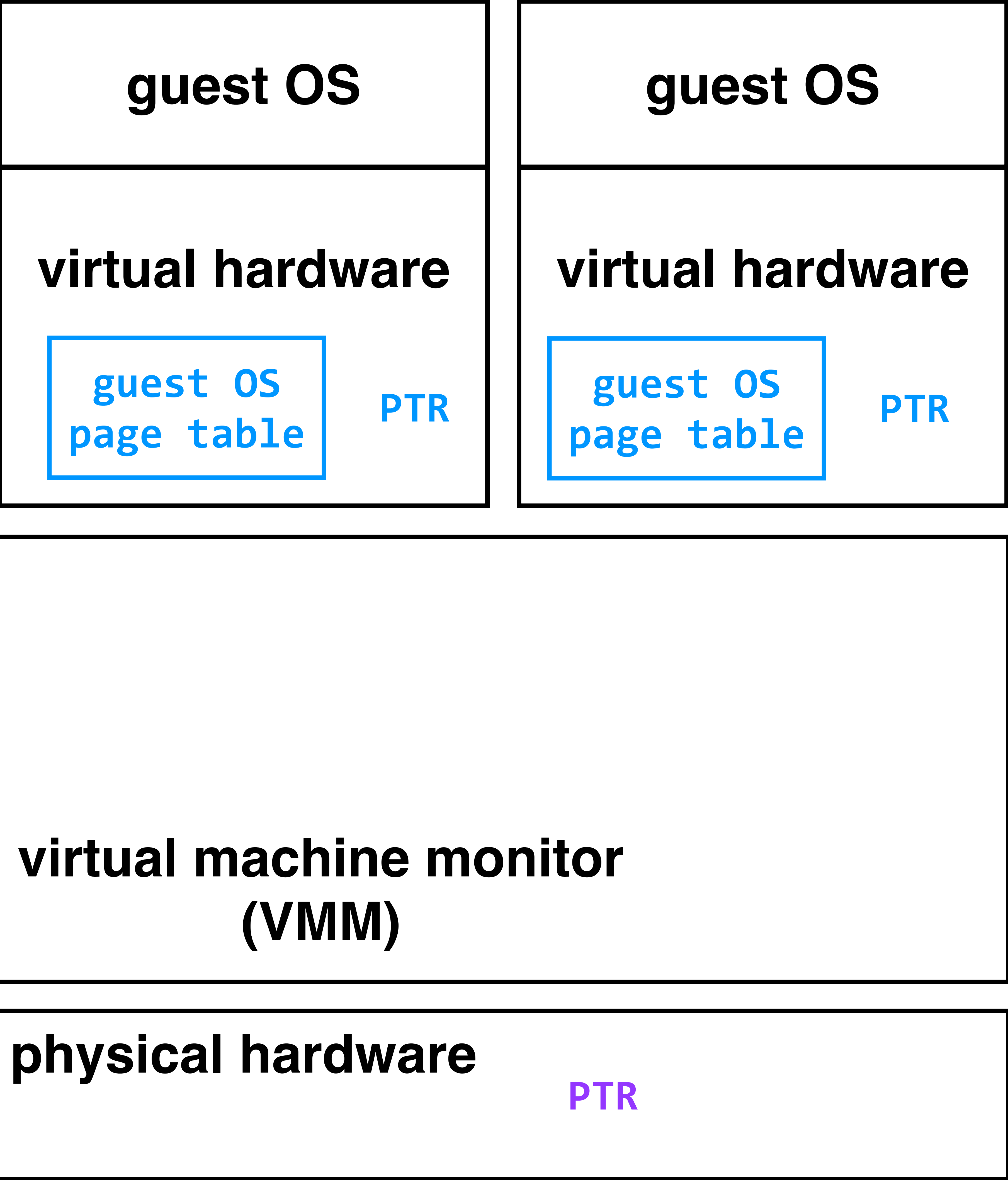
# **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)



## **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

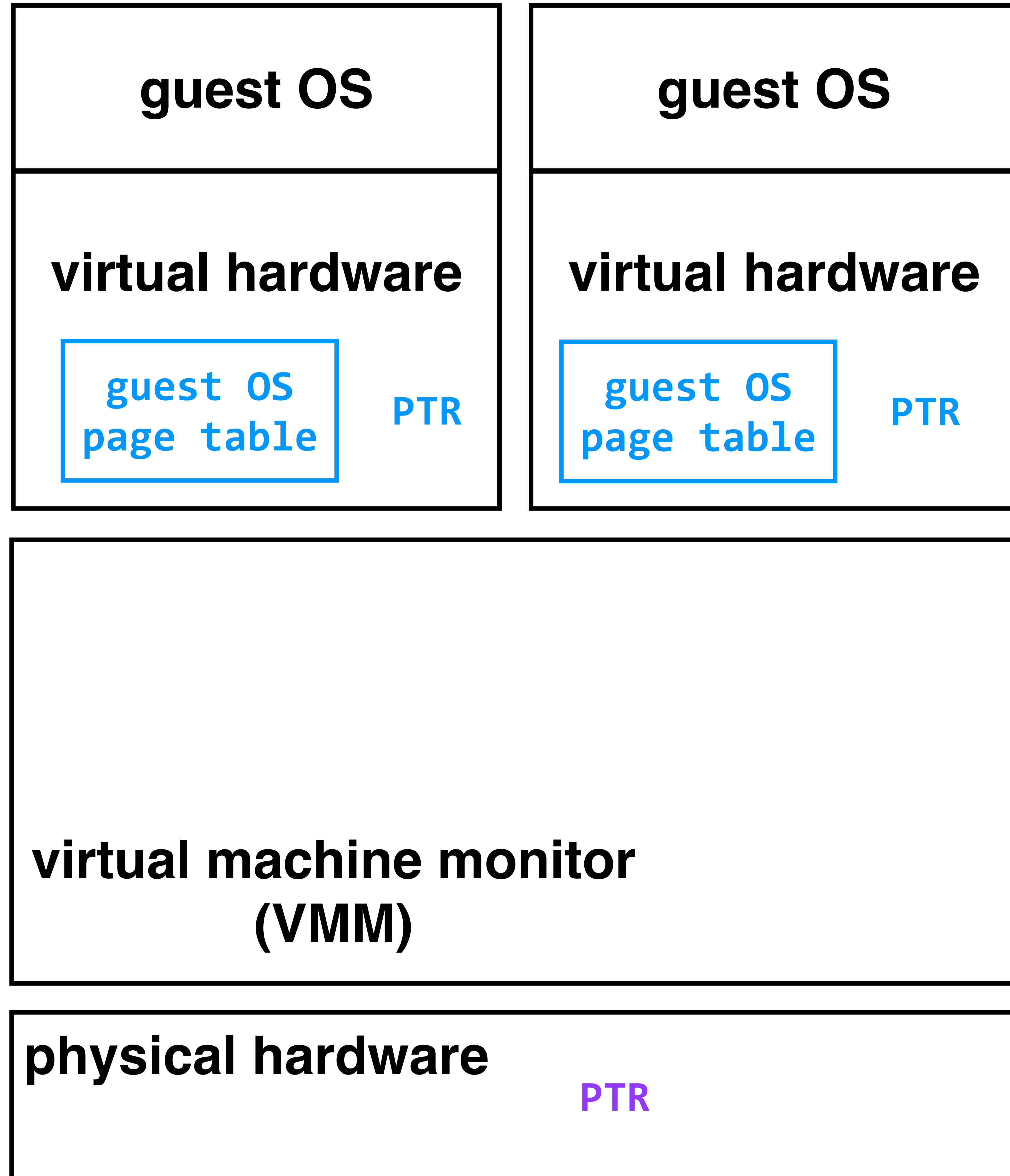
first example: virtualizing memory (again)



guest virtual → guest physical

# **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)

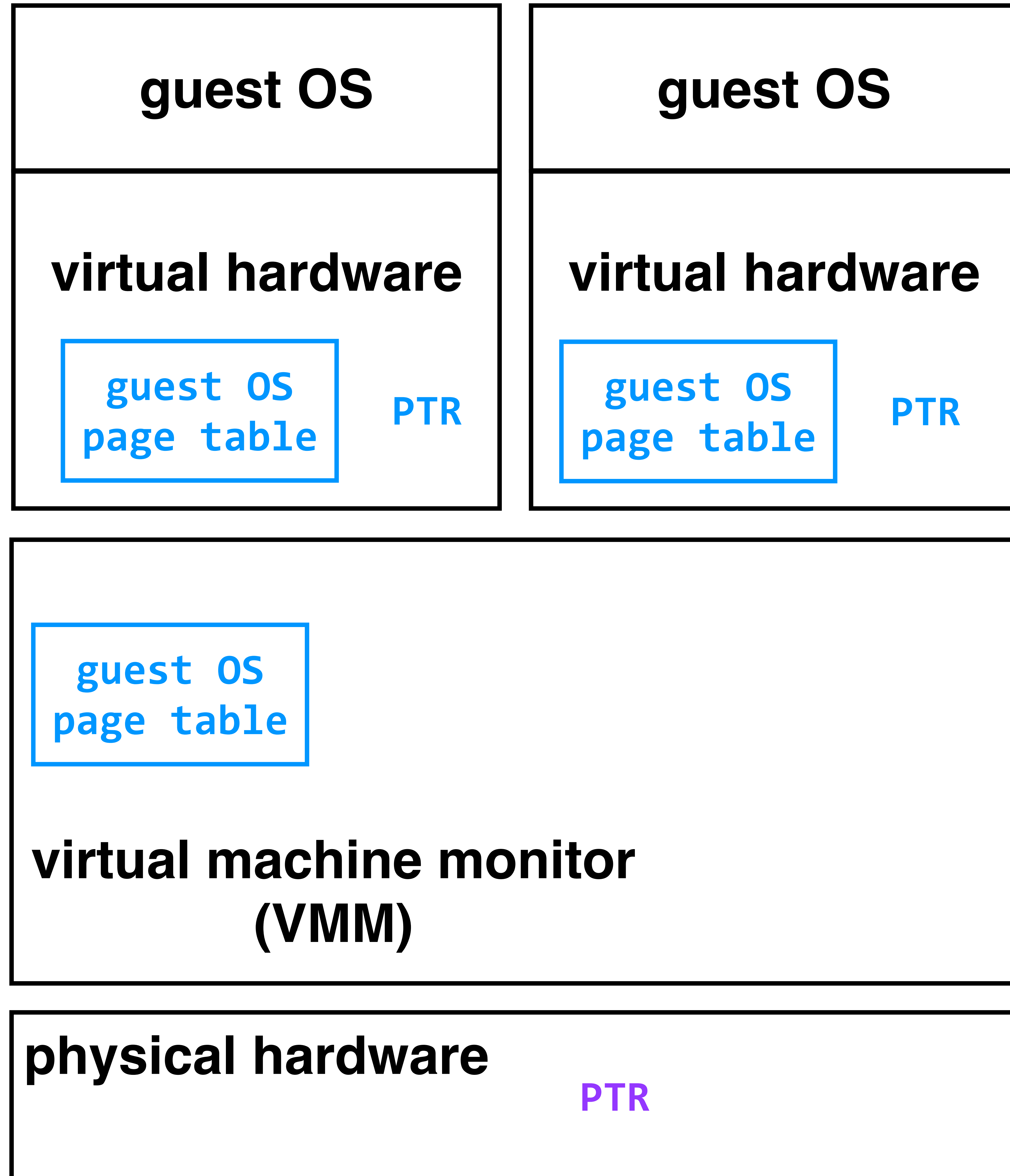



1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

guest virtual → guest physical

# **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

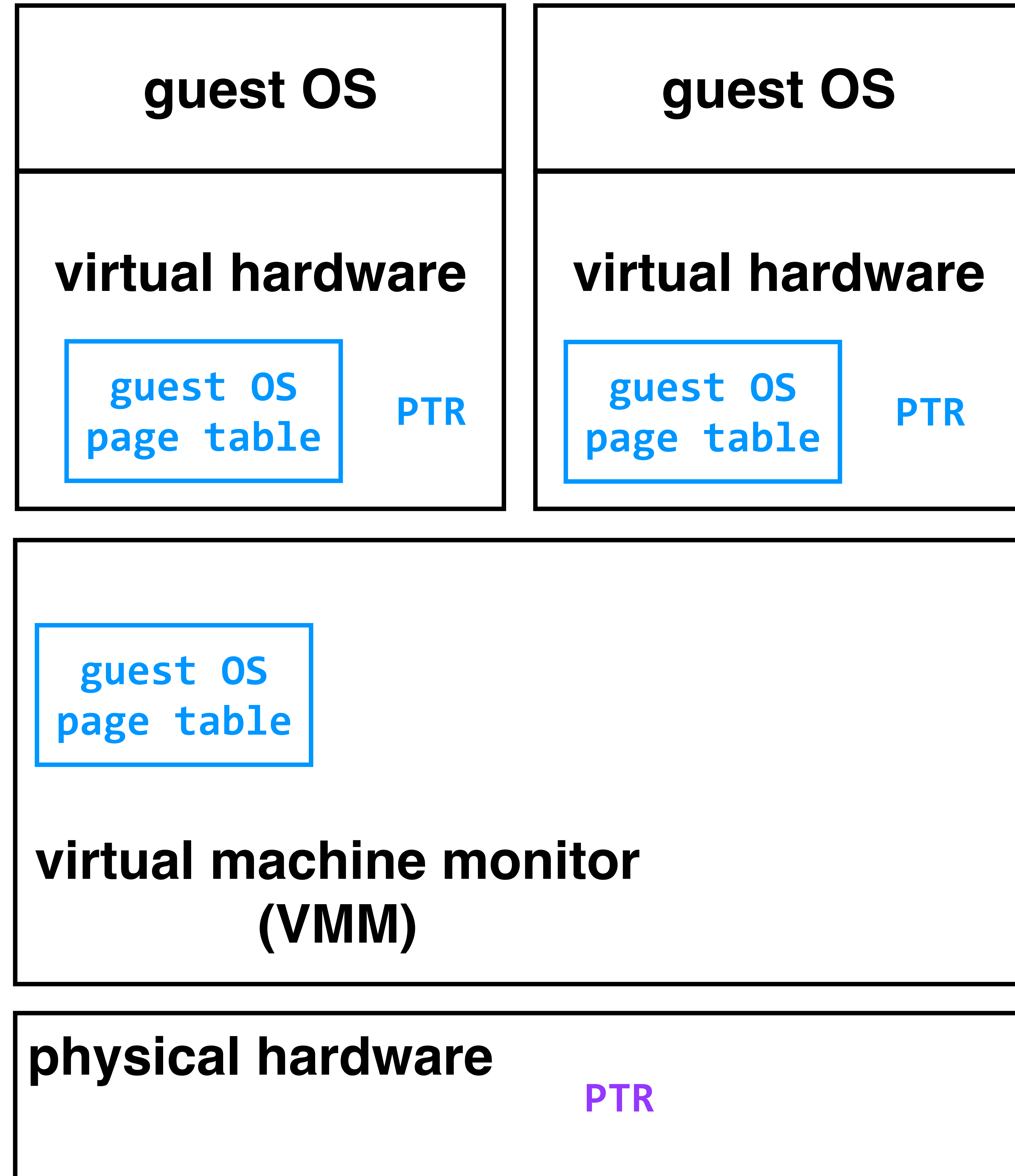
first example: virtualizing memory (again)




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

**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)



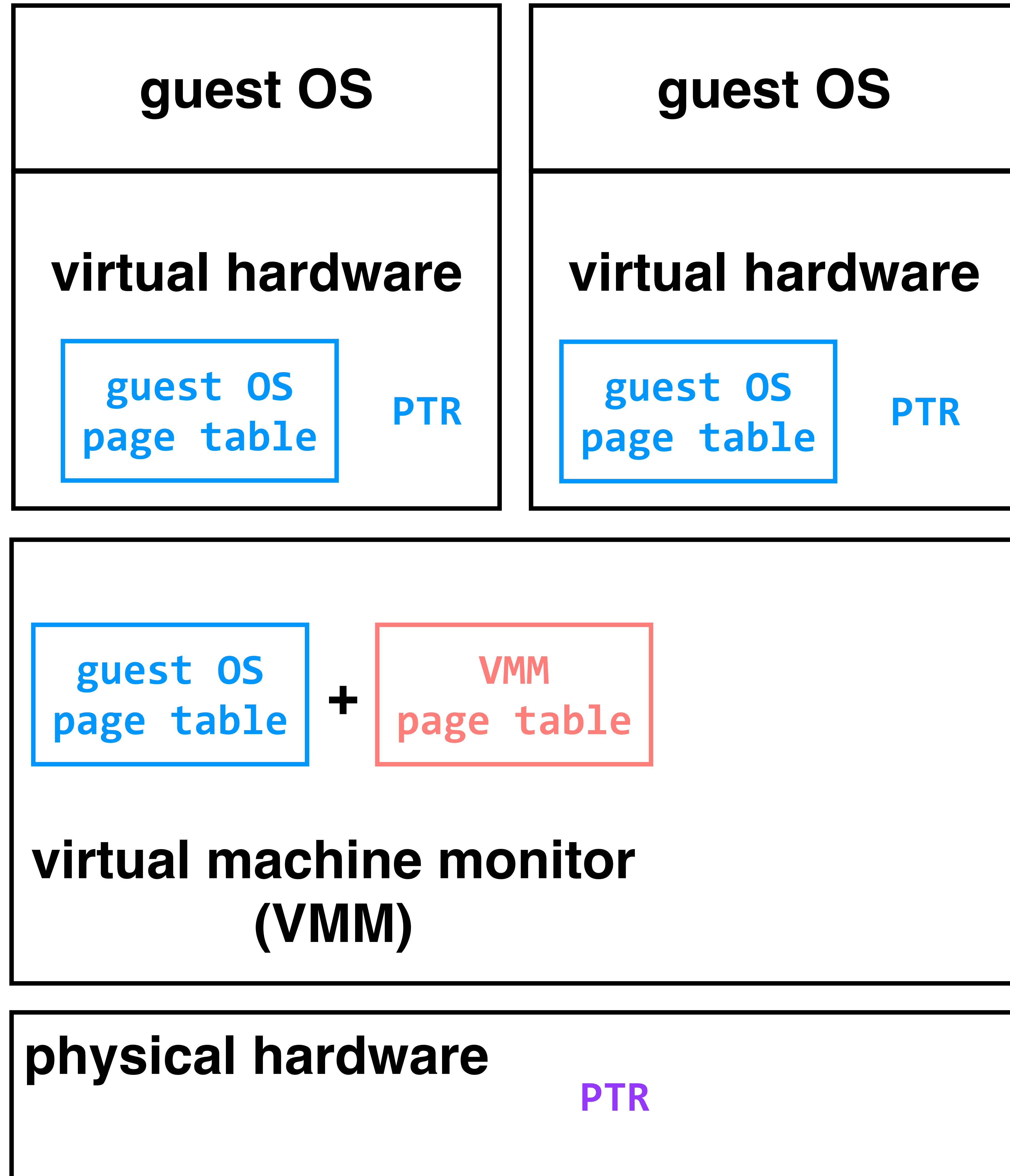

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

2. VMM combines the guest page table with its own page table to create a host page table

guest virtual → guest physical

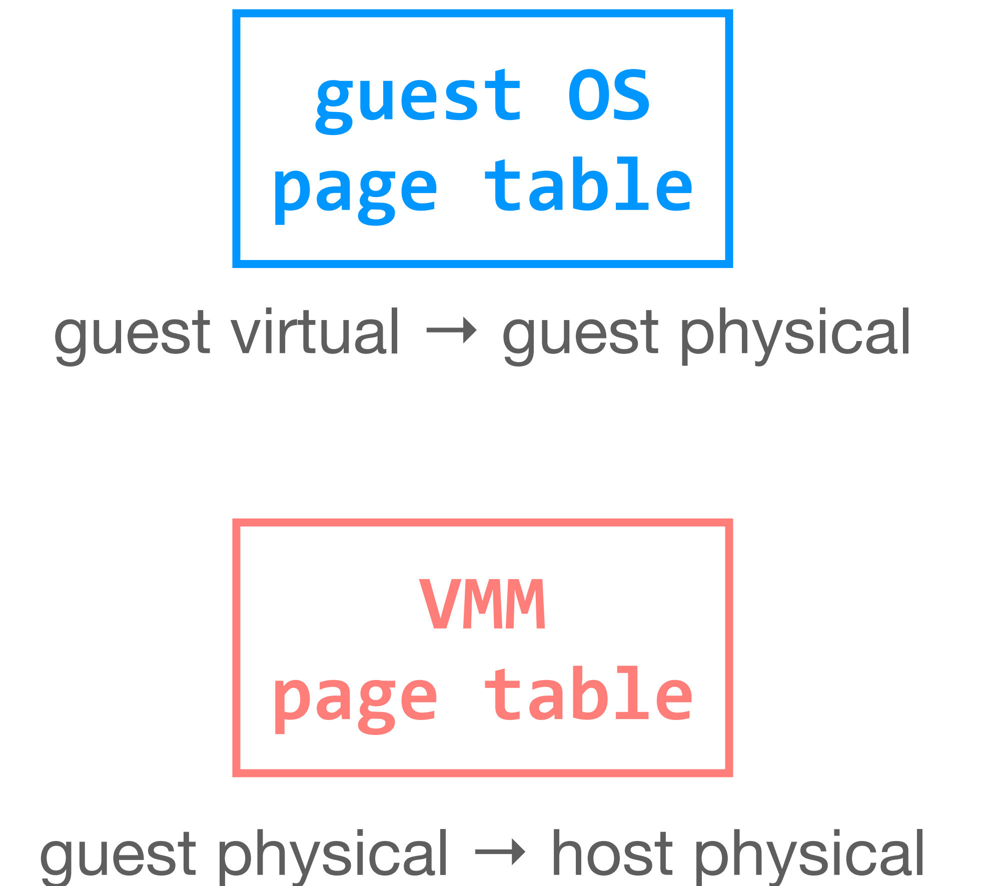
# **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)



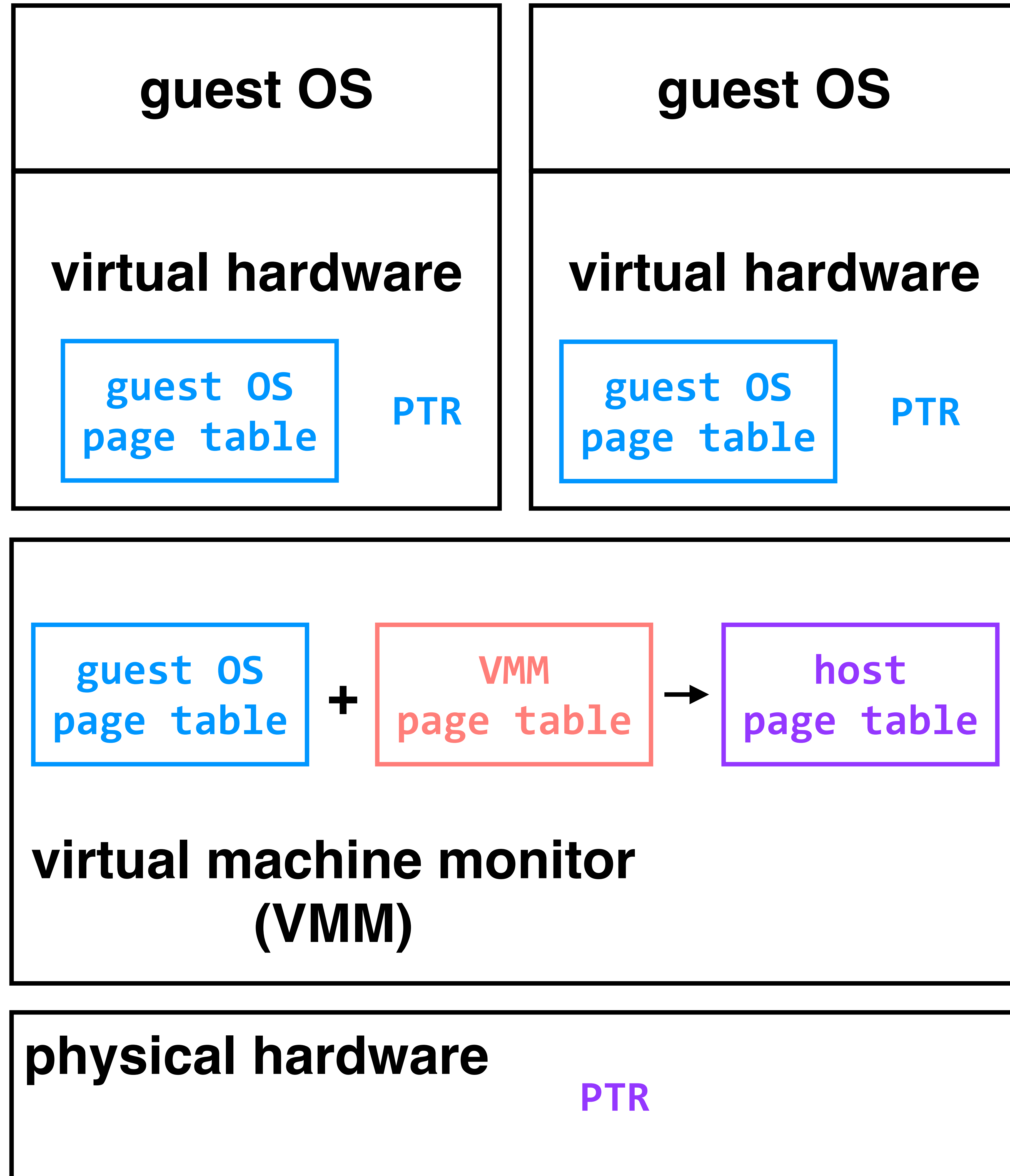

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

2. VMM combines the guest page table with its own page table to create a host page table



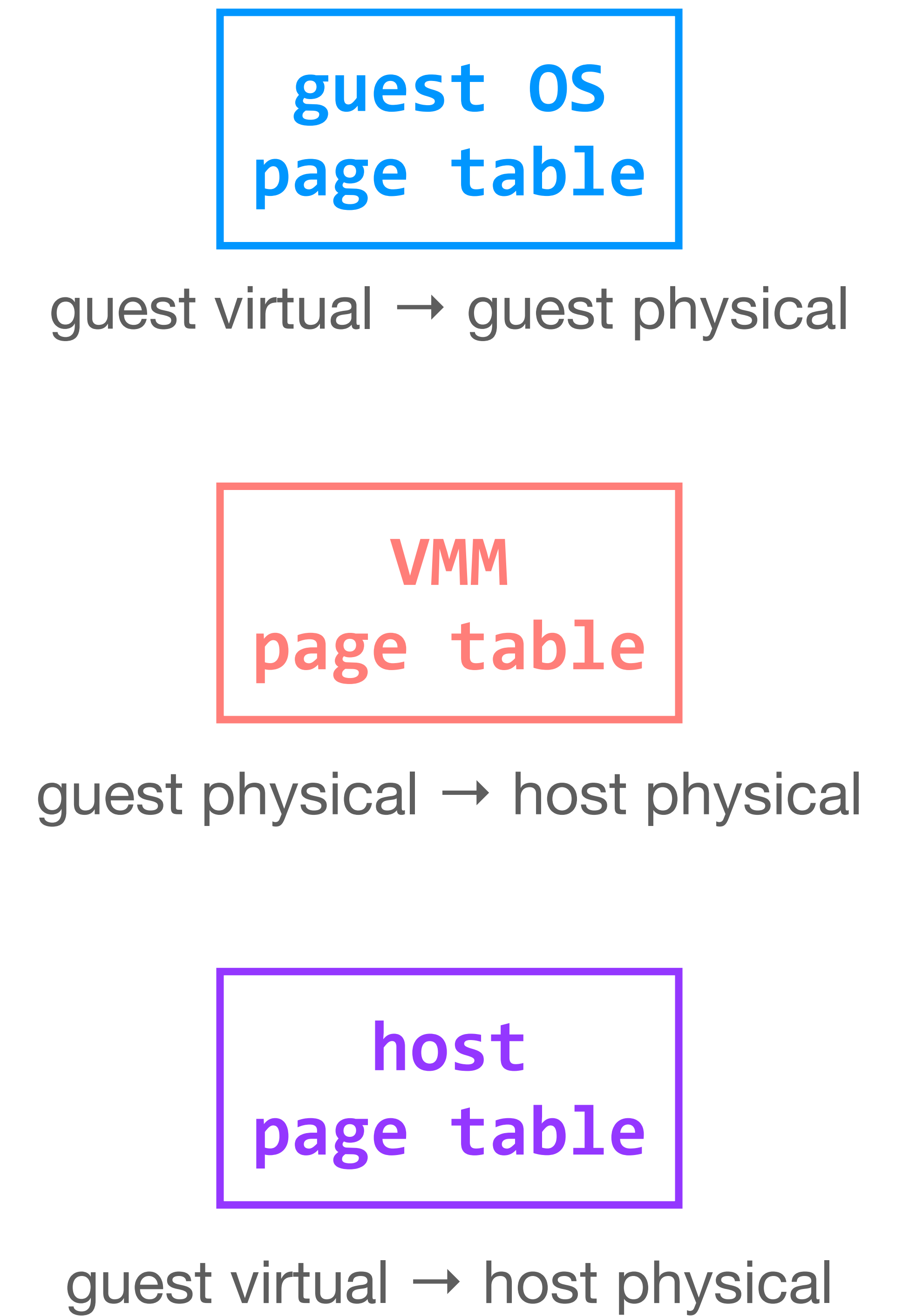
# **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)



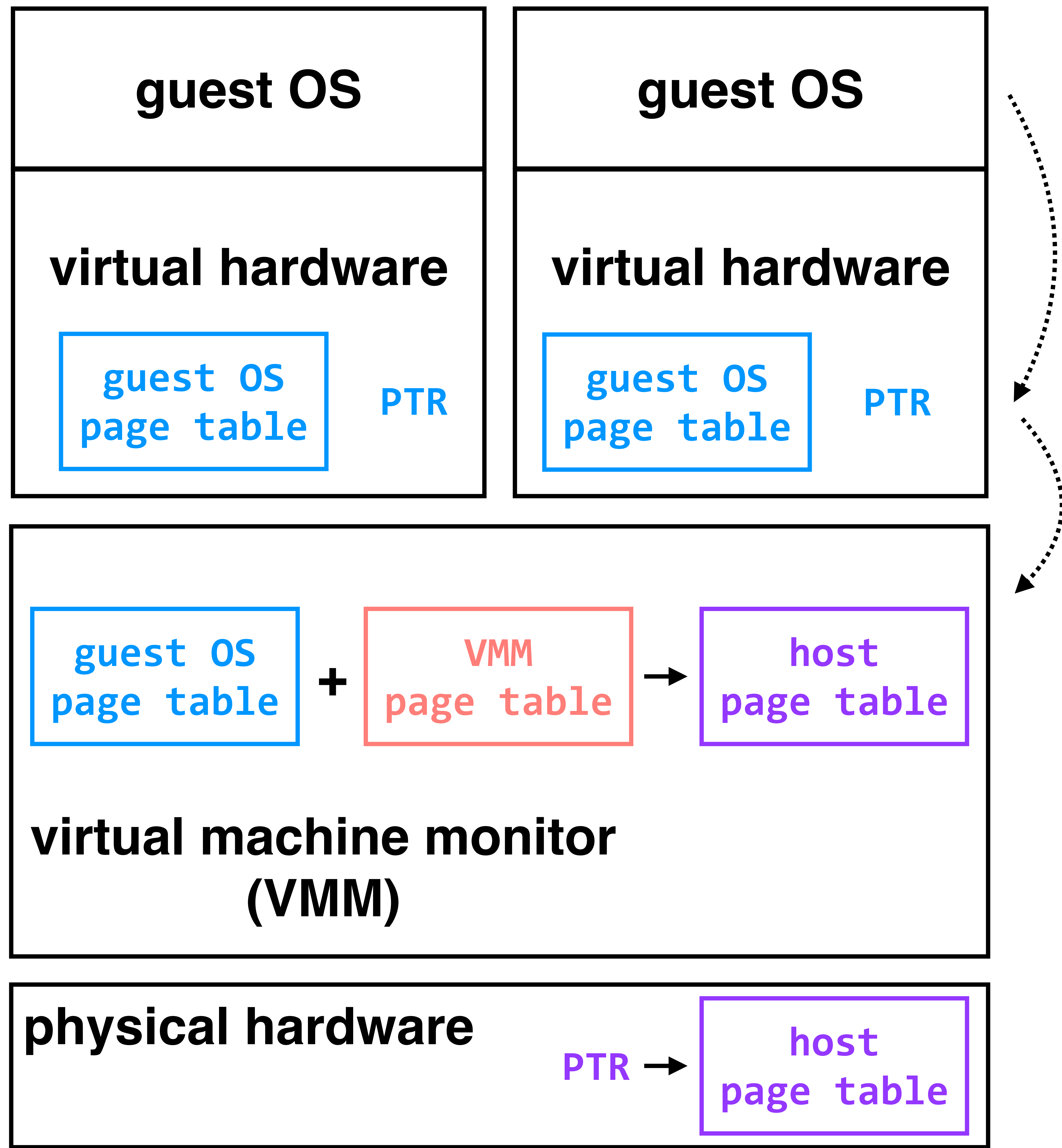

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

2. VMM combines the guest page table with its own page table to create a host page table

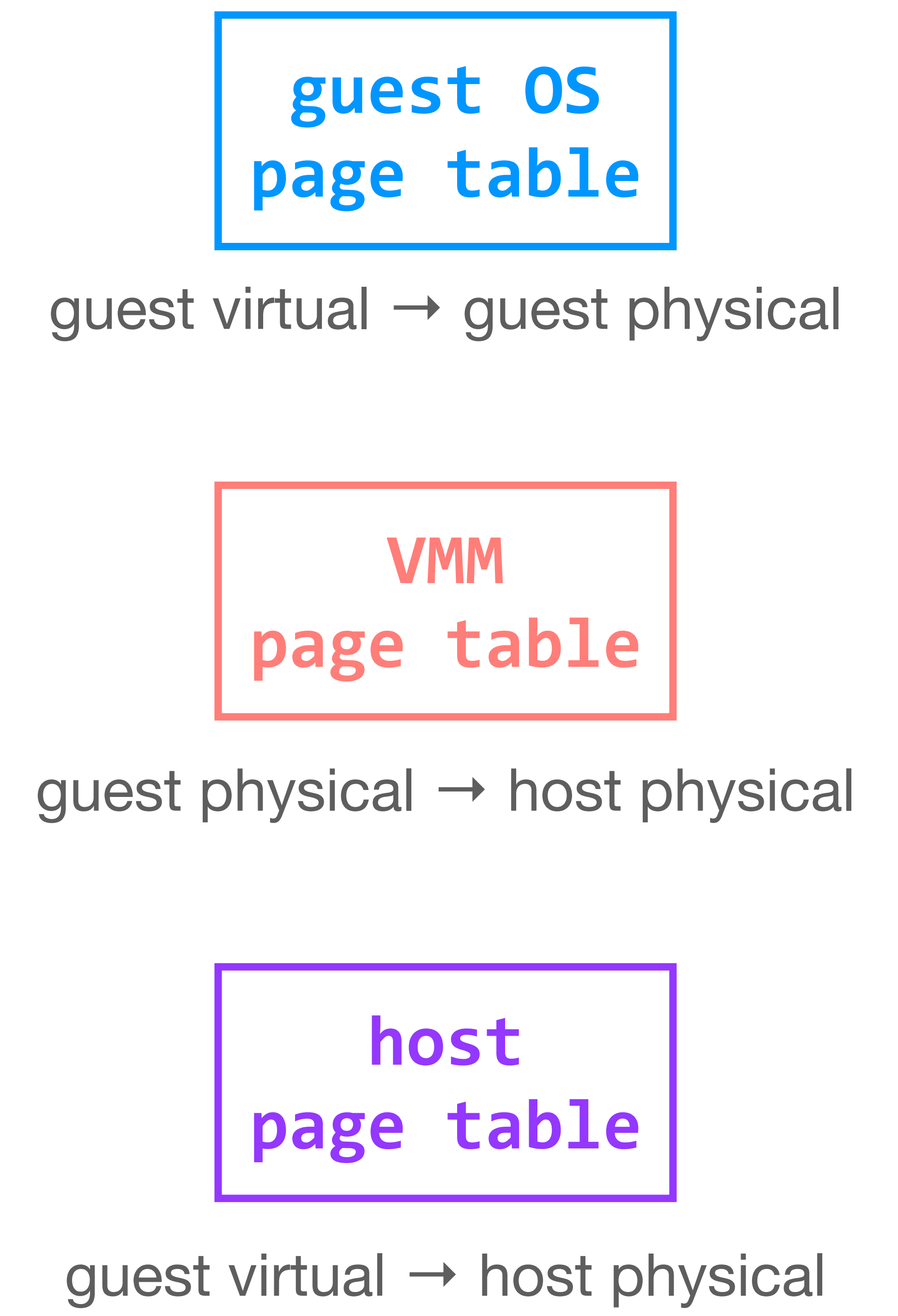


# **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)

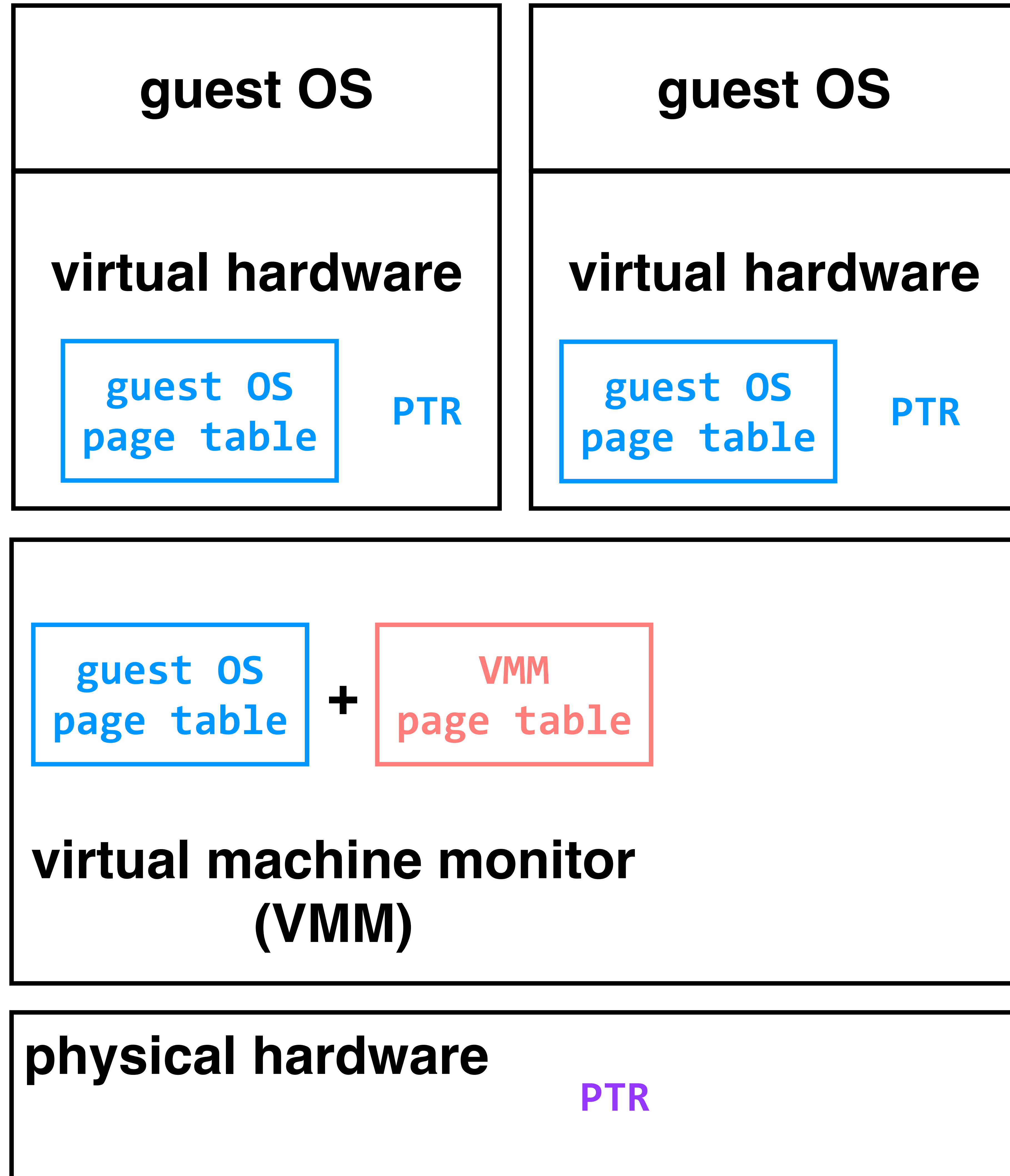



1. guest OS loads its PTR, which triggers an exception; the VMM intercepts
2. VMM combines the guest page table with its own page table to create a host page table
3. physical hardware uses the host page table



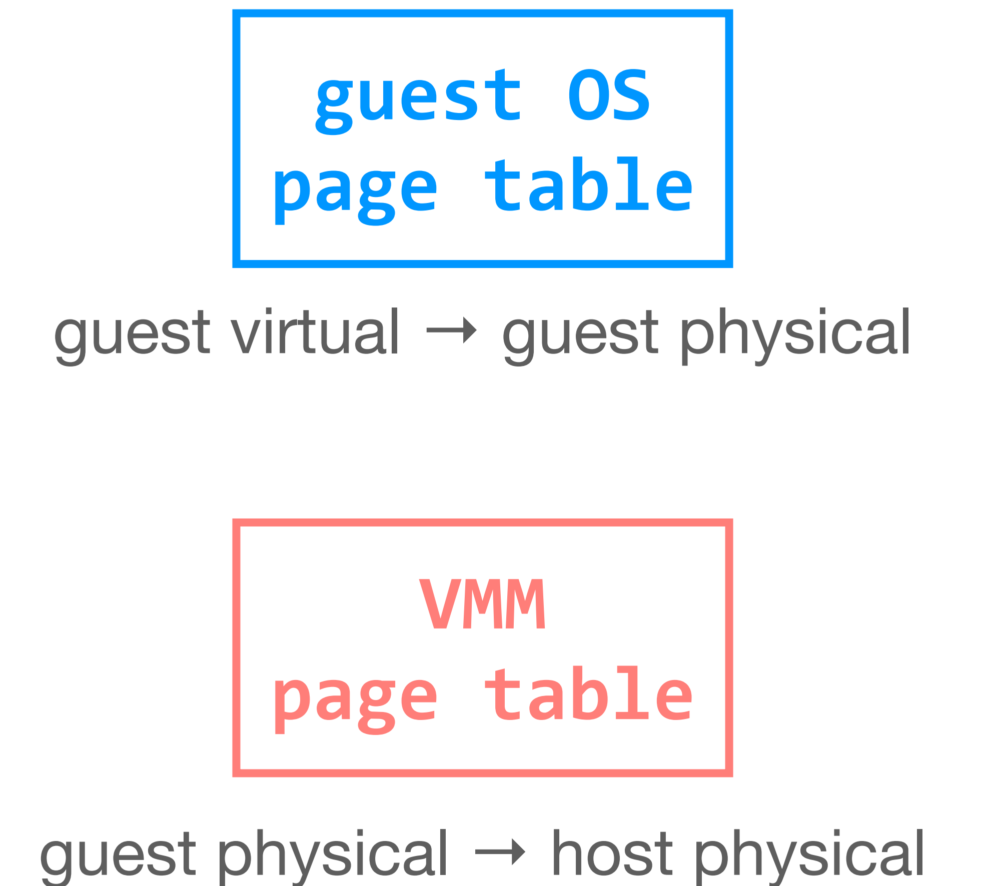
# **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)

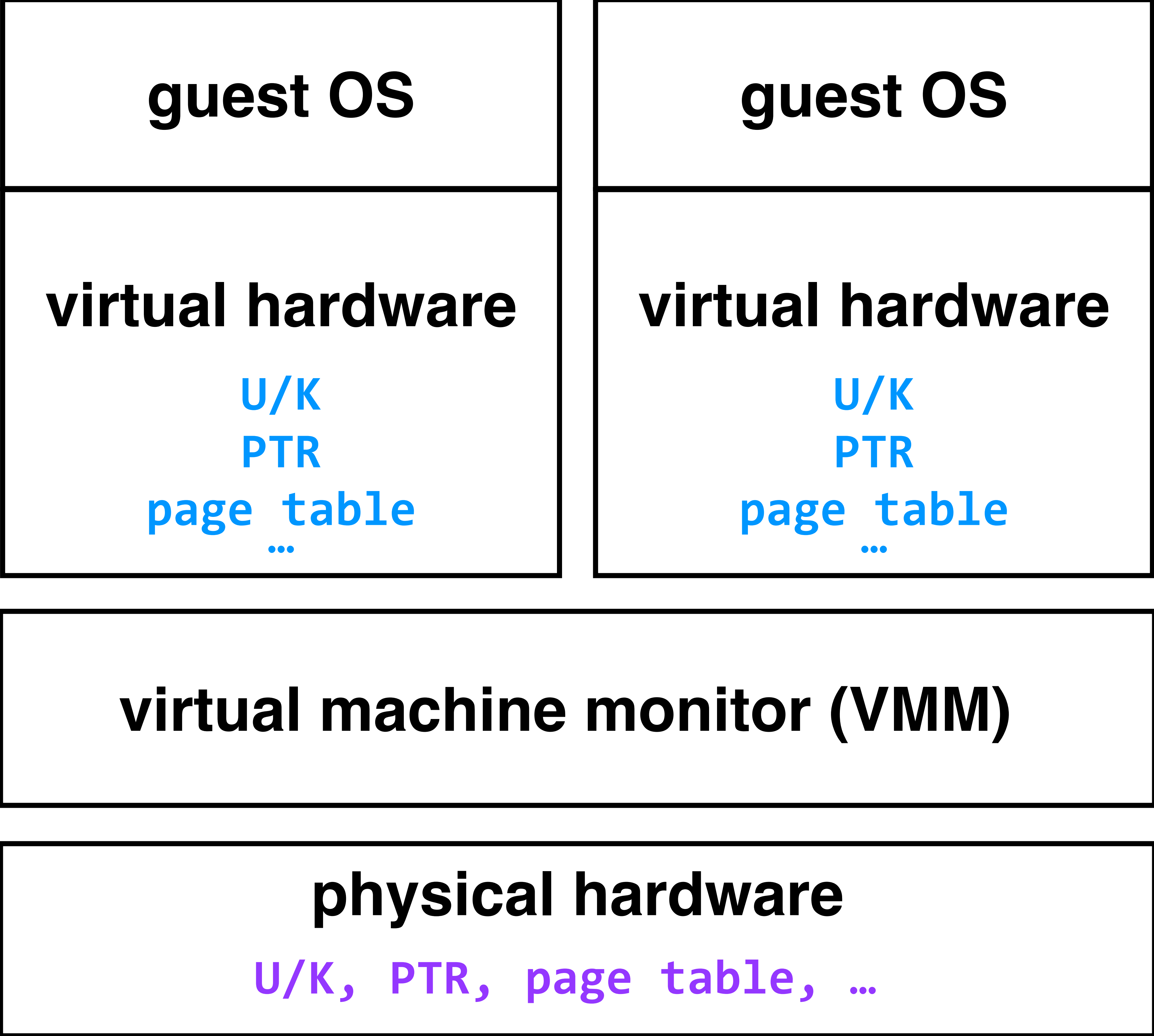



1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

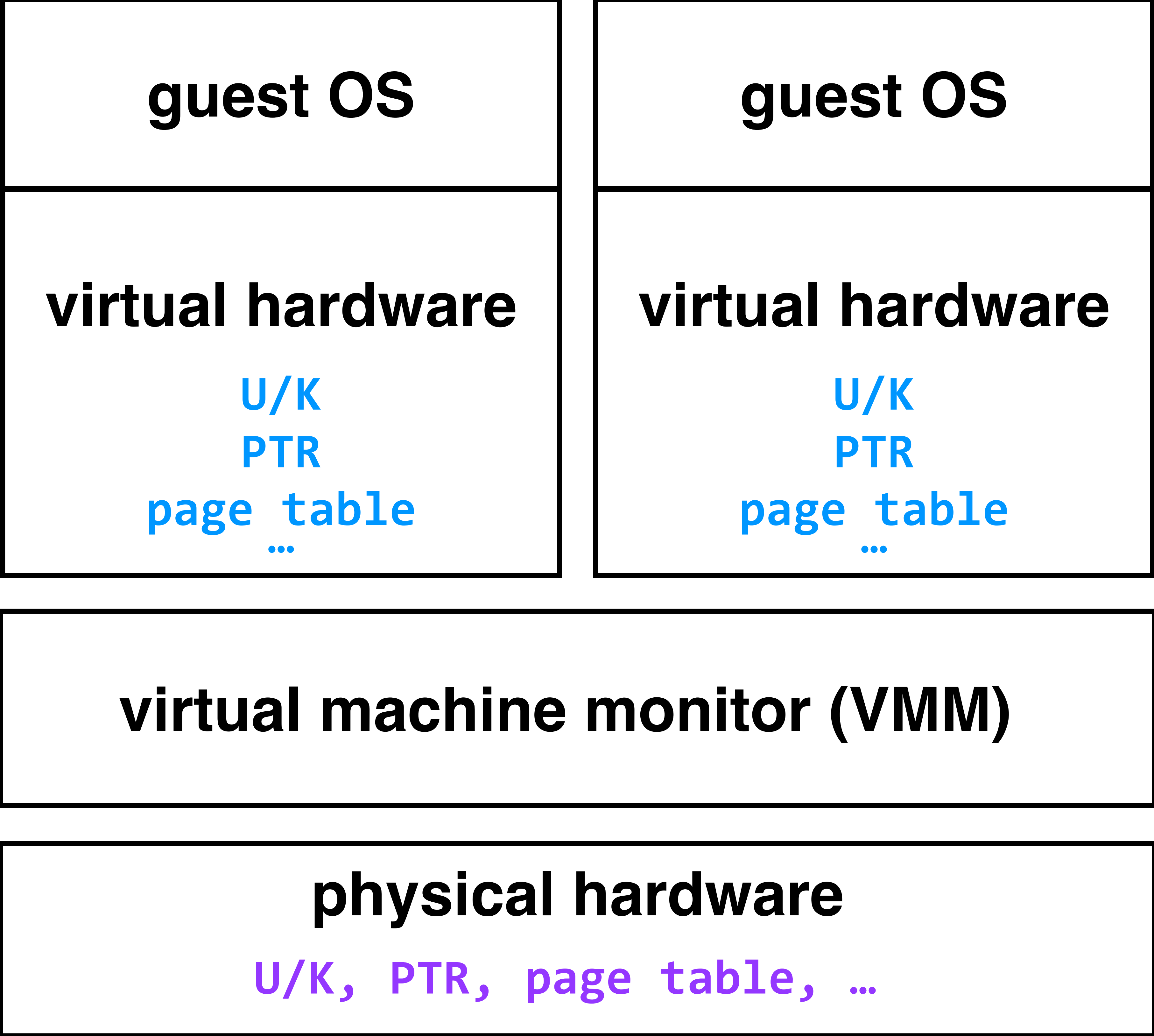
in modern hardware, the physical hardware is aware of both page tables, and performs the translation from guest virtual to host physical itself



**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

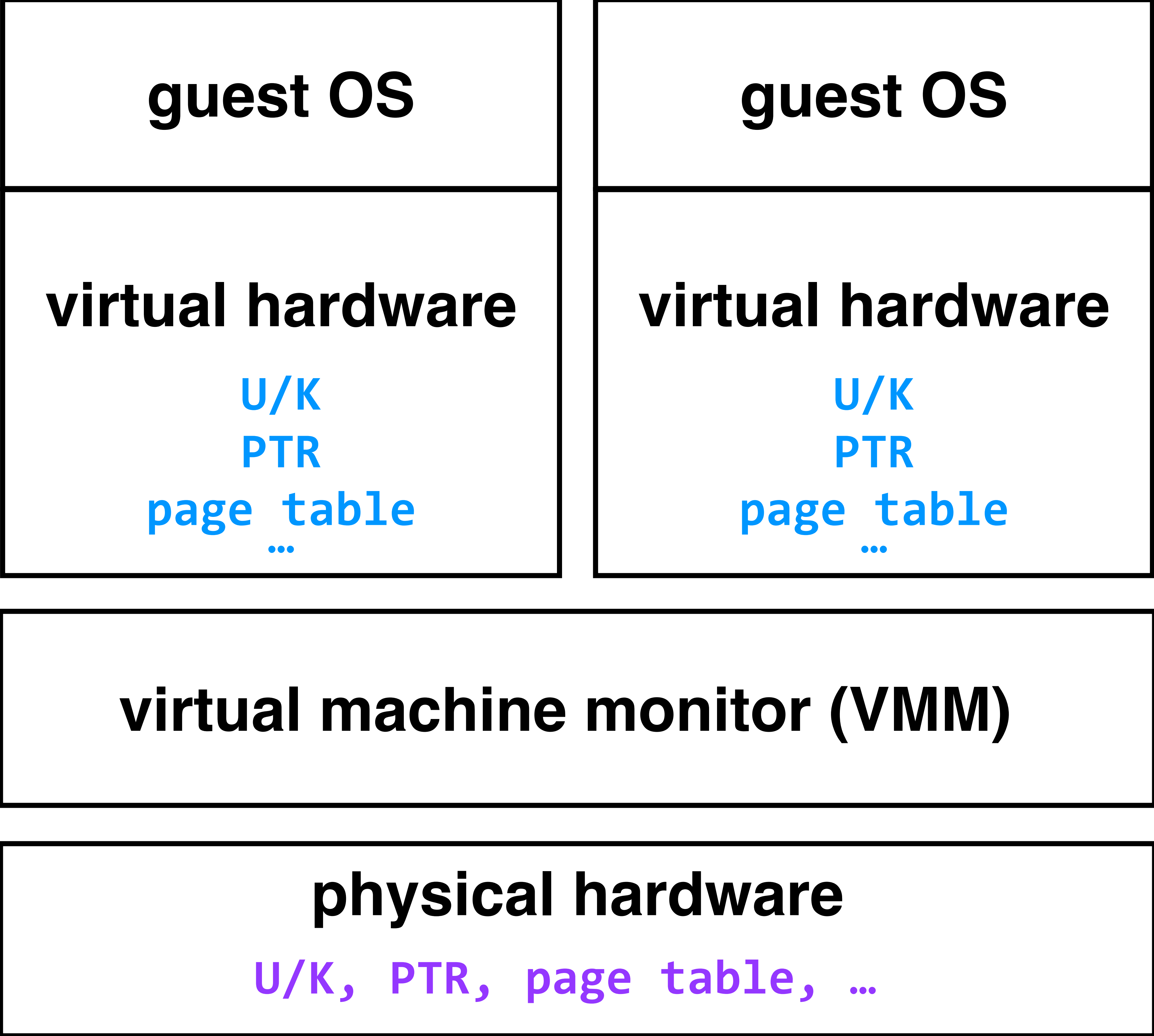


**virtual machine monitor:** virtualizes the physical hardware for the guest OSes




guest OSes run in user mode

**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

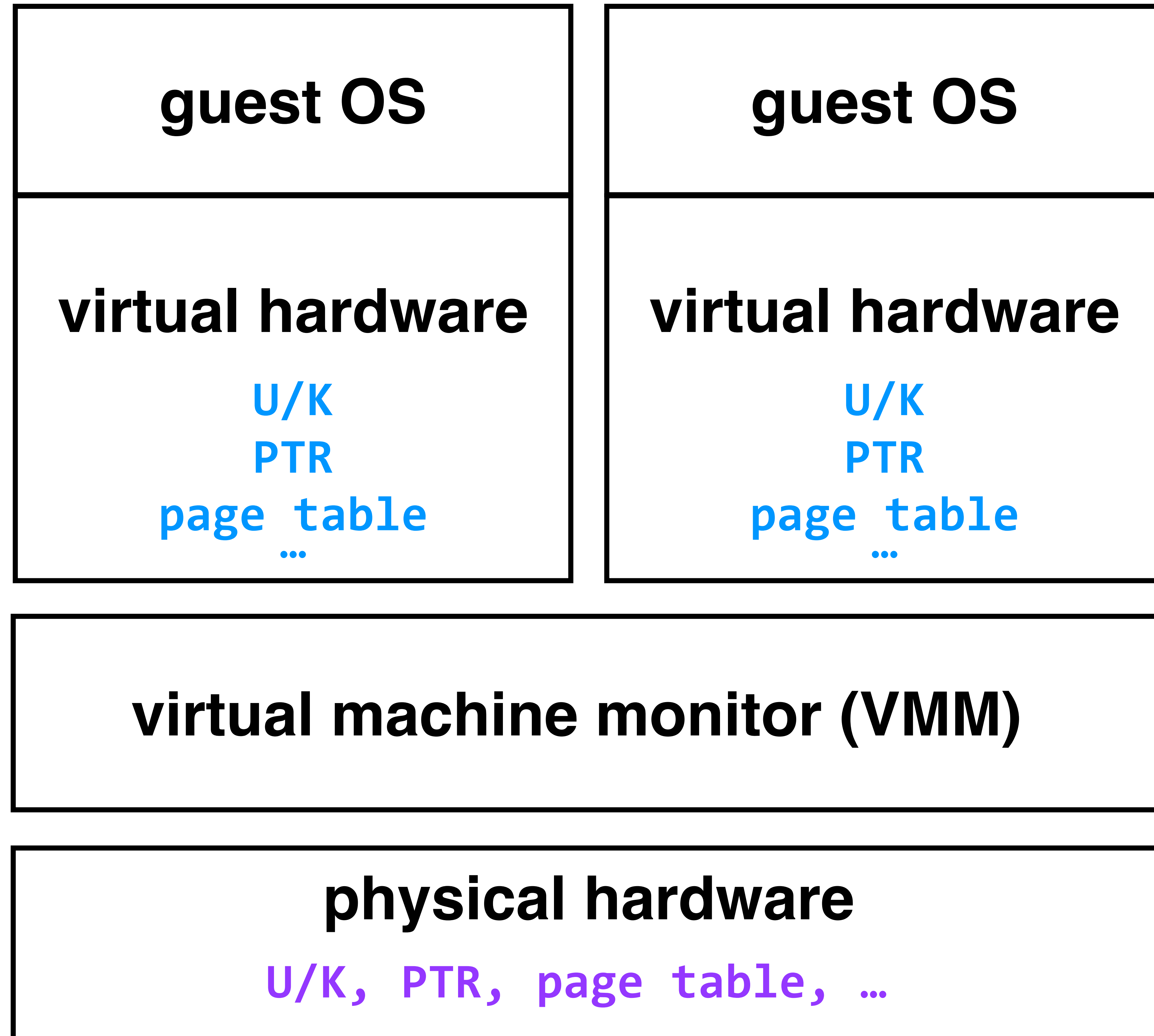



guest OSes run in user mode

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

**virtual machine monitor:** virtualizes the physical hardware for the guest OSes




guest OSes run in user mode

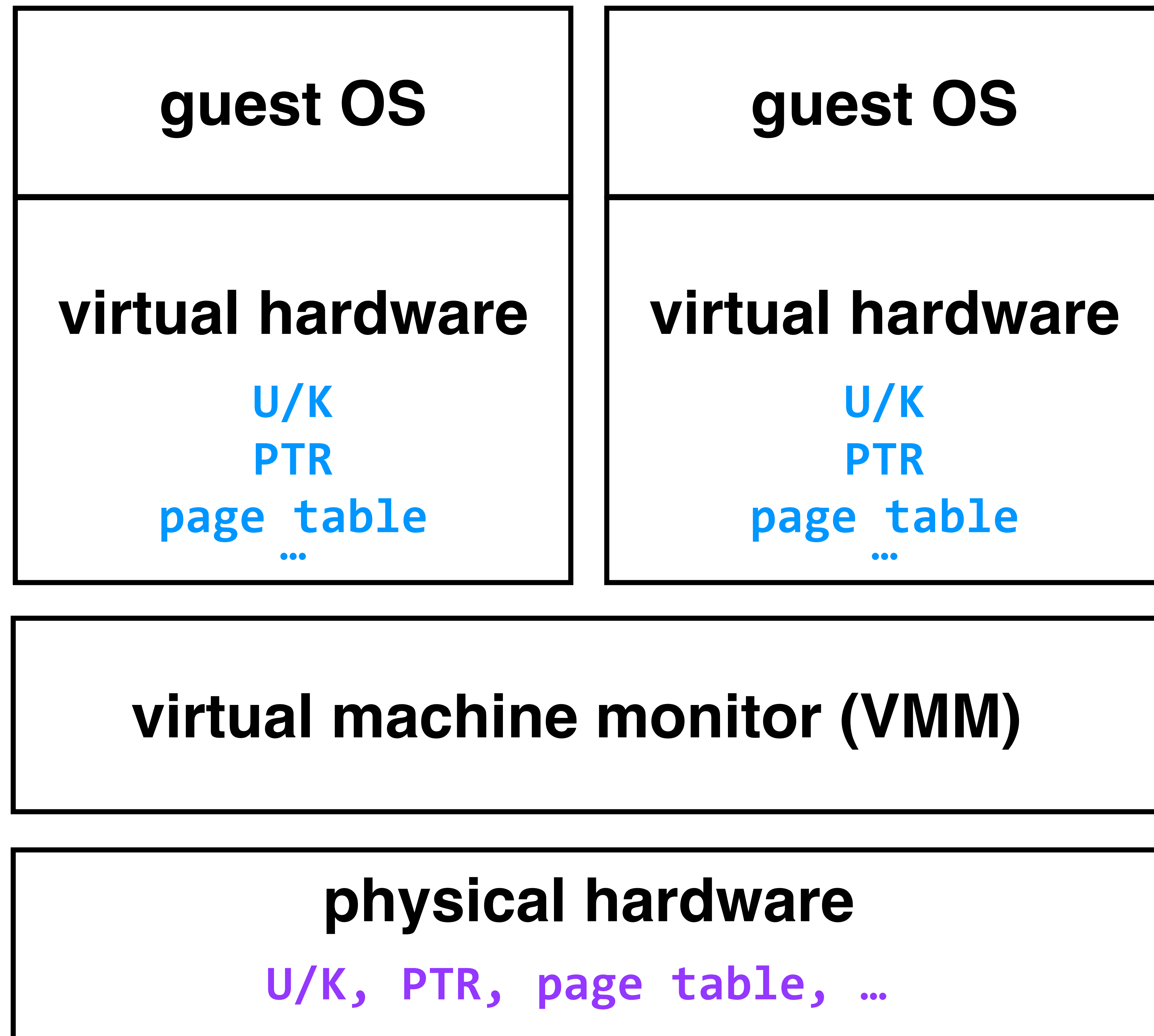
privileged instructions in guest OS will cause an exception, which the VMM will intercept ("**trap**") and **emulate**

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

**second question:** what about when the VMM needs to emulate an instruction that doesn't trigger an exception?

**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

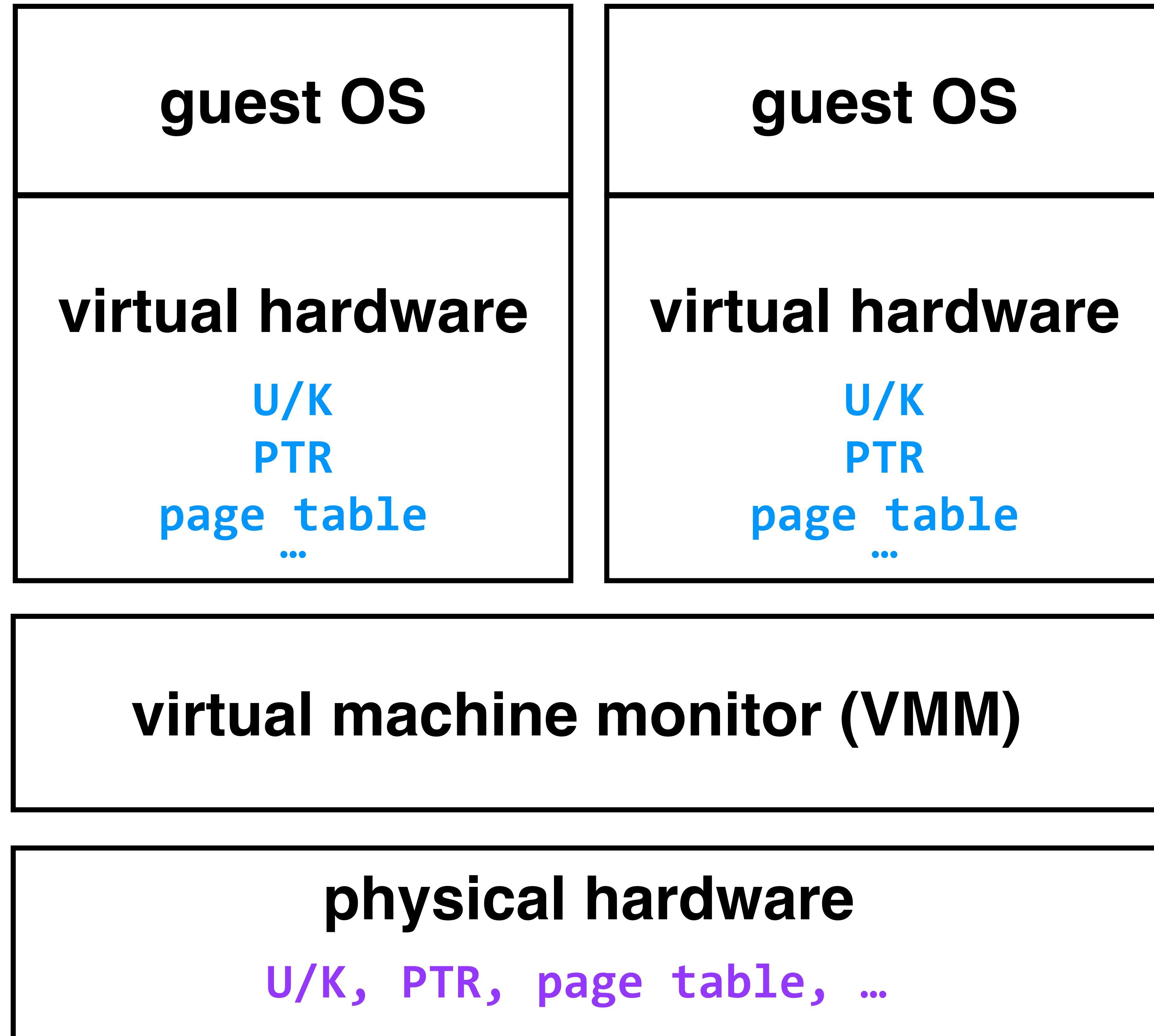
second example: virtualizing the U/K bit




**second question:** what about when the VMM needs to emulate an instruction that doesn't trigger an exception?

**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

second example: virtualizing the U/K bit

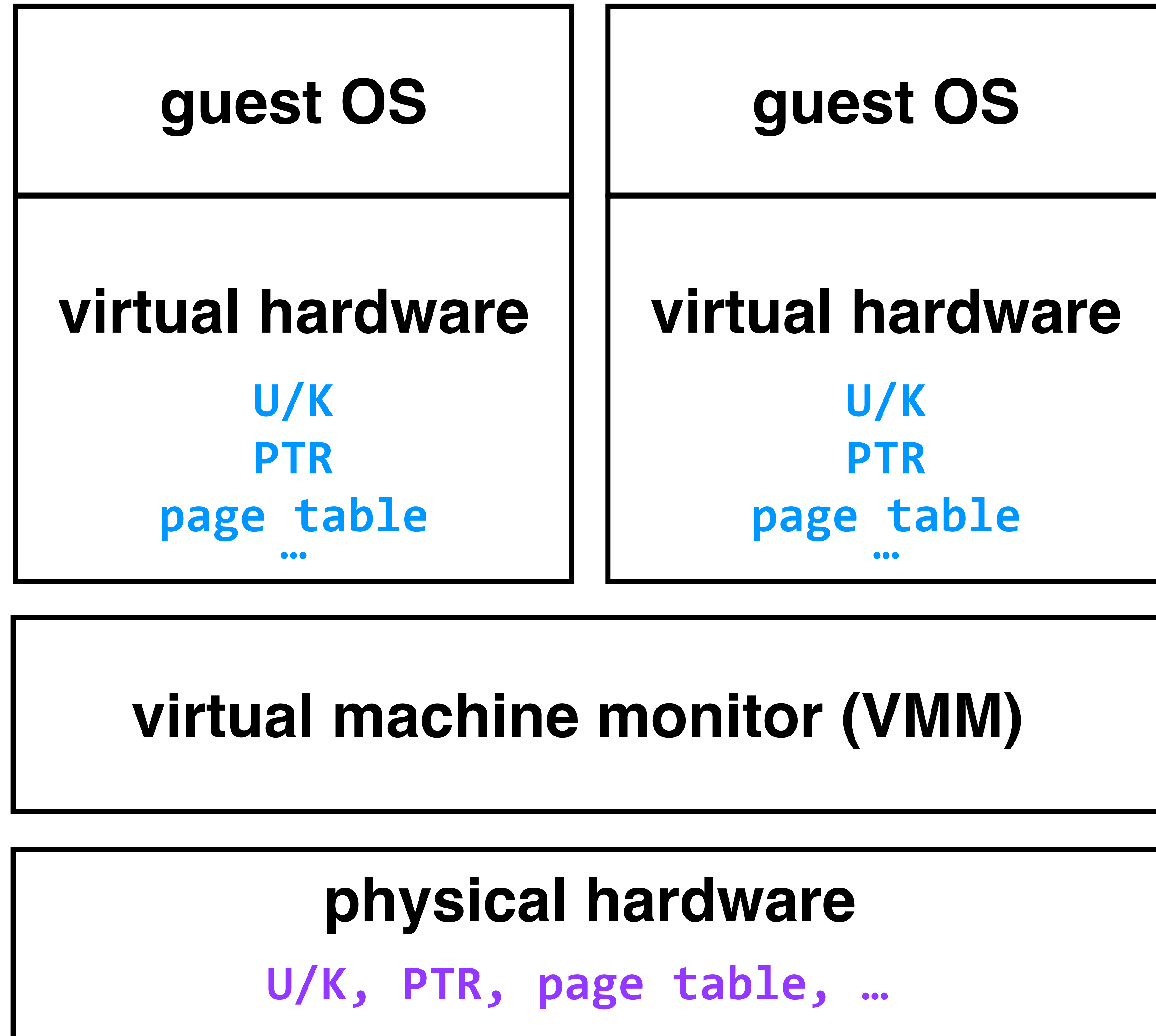



**para-virtualization:** modify guest OS slightly

**second question:** what about when the VMM needs to emulate an instruction that doesn't trigger an exception?

## **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

second example: virtualizing the U/K bit



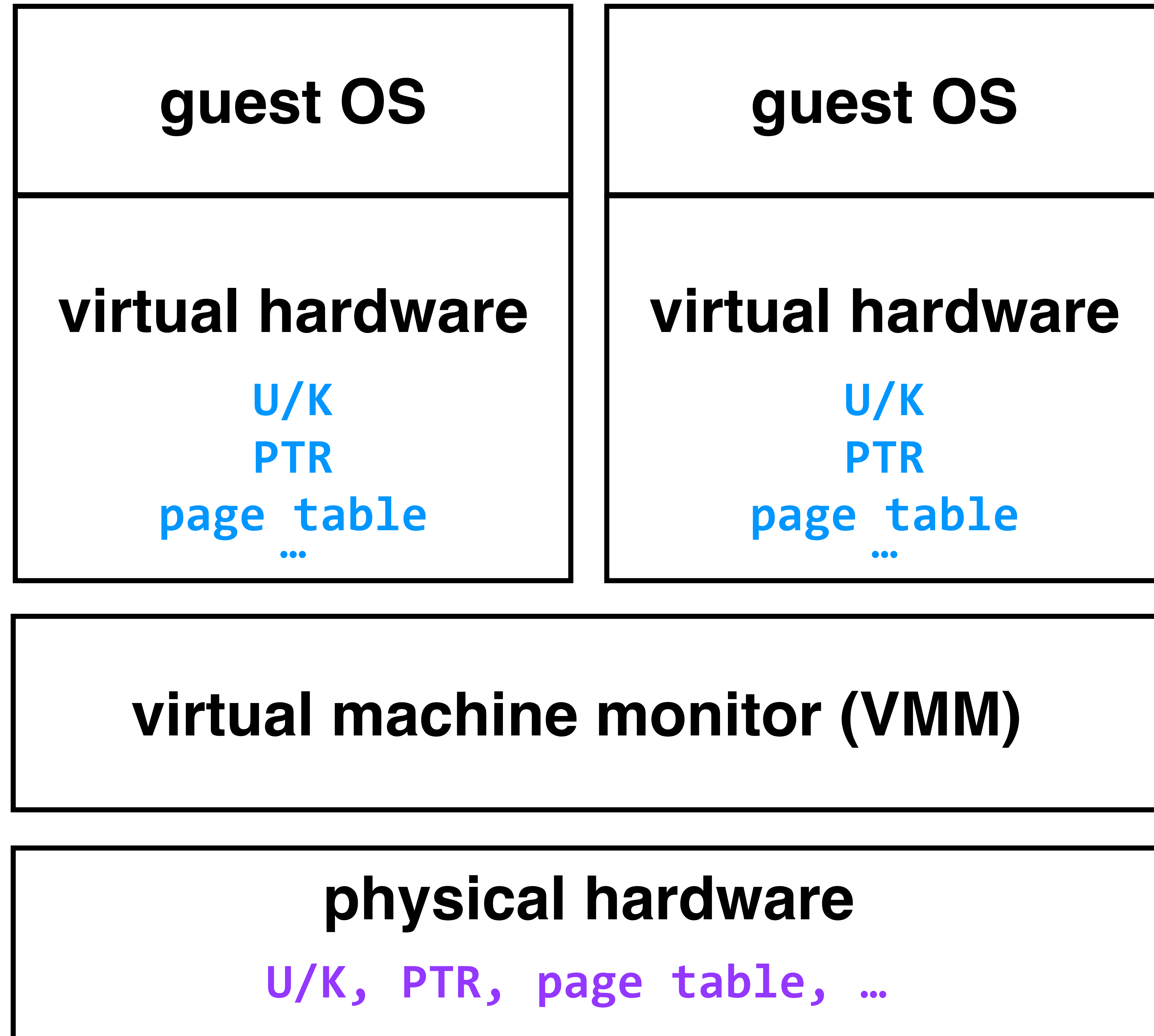

**para-virtualization:** modify guest OS slightly

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

**second question:** what about when the VMM needs to emulate an instruction that doesn't trigger an exception?

## **virtual machine monitor:** virtualizes the physical hardware for the guest OSes

second example: virtualizing the U/K bit



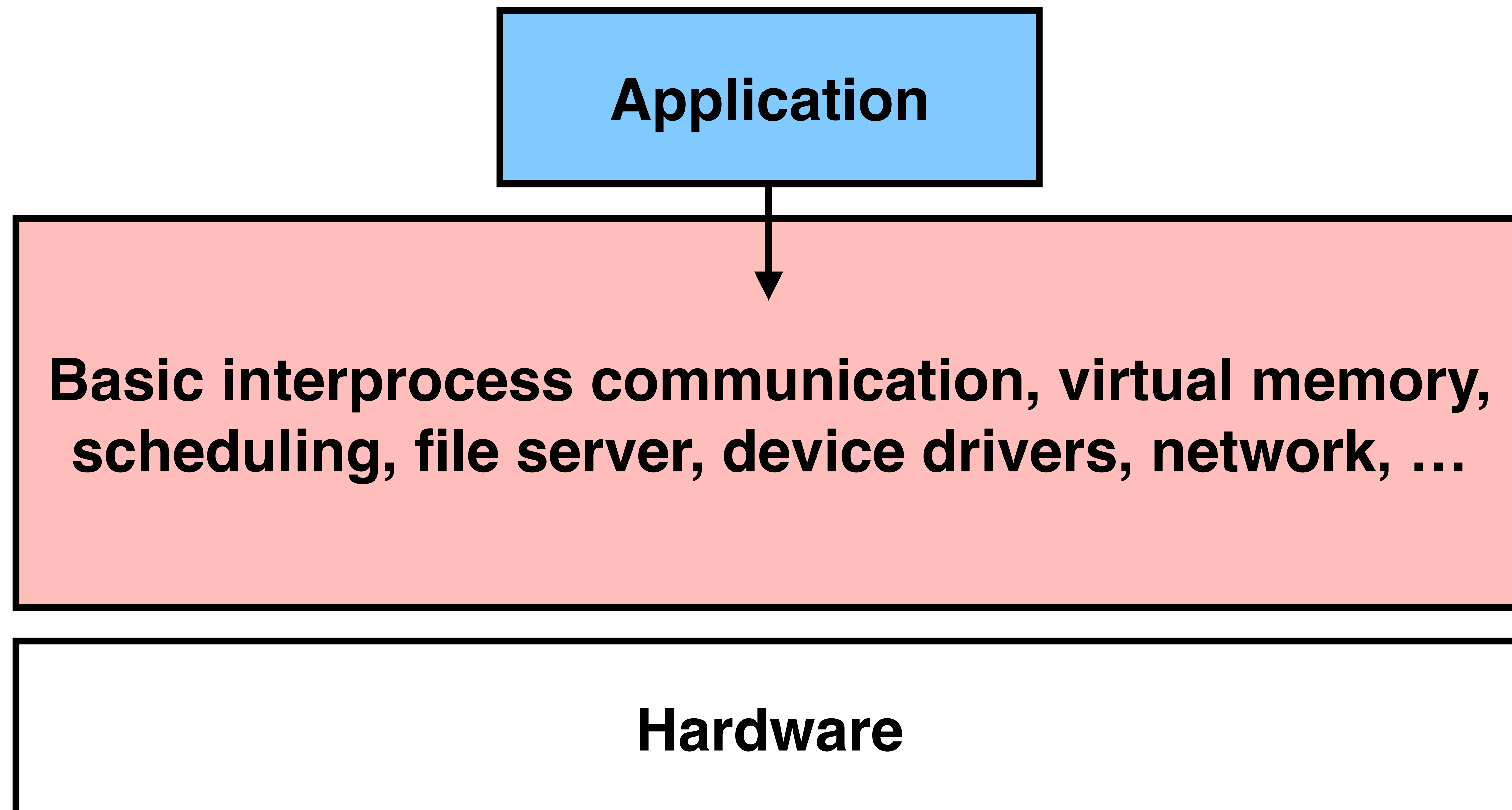

**para-virtualization:** modify guest OS slightly

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

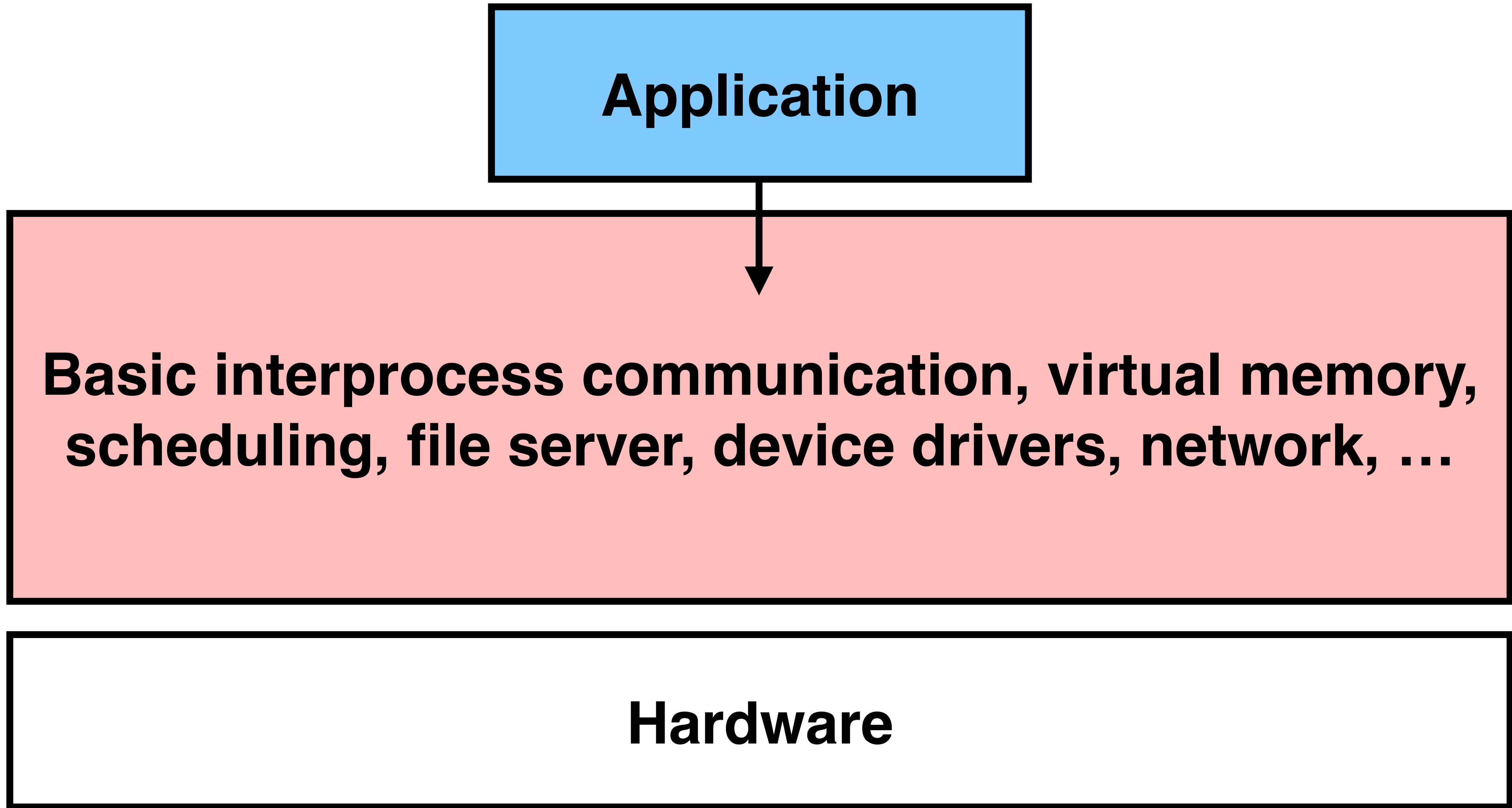
**hardware support:** architecture provides a special operating mode for VMMs in addition to user mode, kernel mode

**second question:** what about when the VMM needs to emulate an instruction that doesn't trigger an exception?

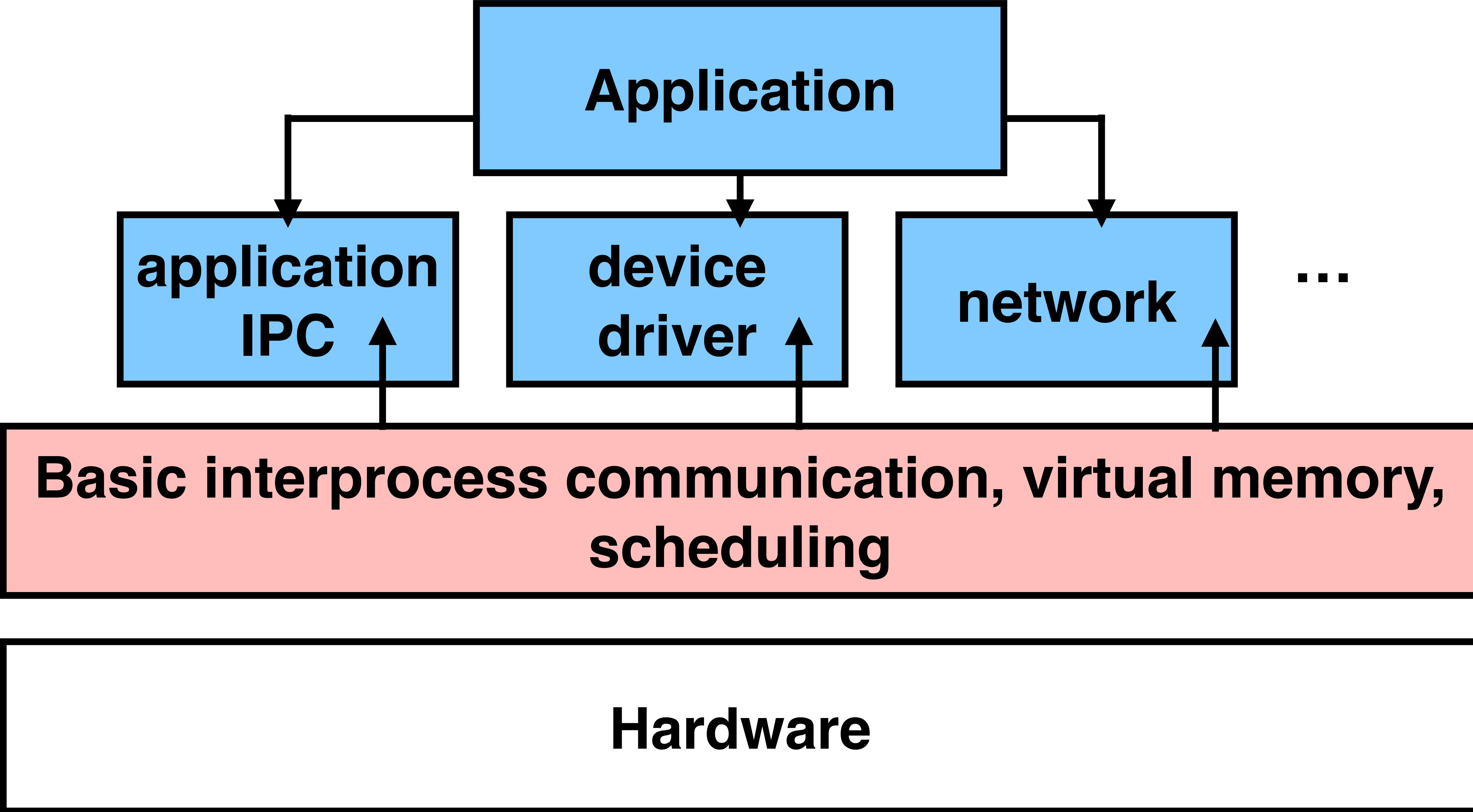
**monolithic kernel:** no enforced modularity within the kernel itself



**monolithic kernel:** no enforced modularity within the kernel itself




**microkernels:** enforce modularity by putting subsystems in user programs



**performance:** performance issues have influenced a lot of the system designs you've seen so far

**performance:** performance issues have influenced a lot of the system designs you've seen so far

**latency:** how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of memory in an OS?

**performance:** performance issues have influenced a lot of the system designs you've seen so far

**latency:** how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of memory in an OS?

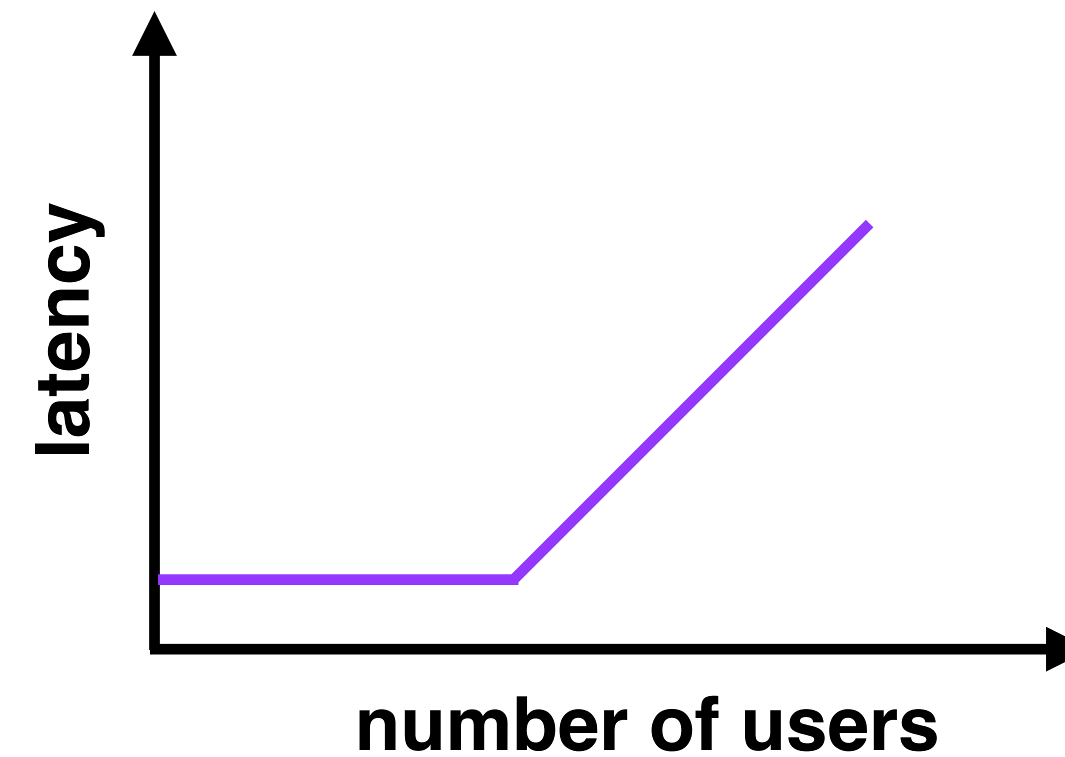
**throughput:** how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

**performance:** performance issues have influenced a lot of the system designs you've seen so far

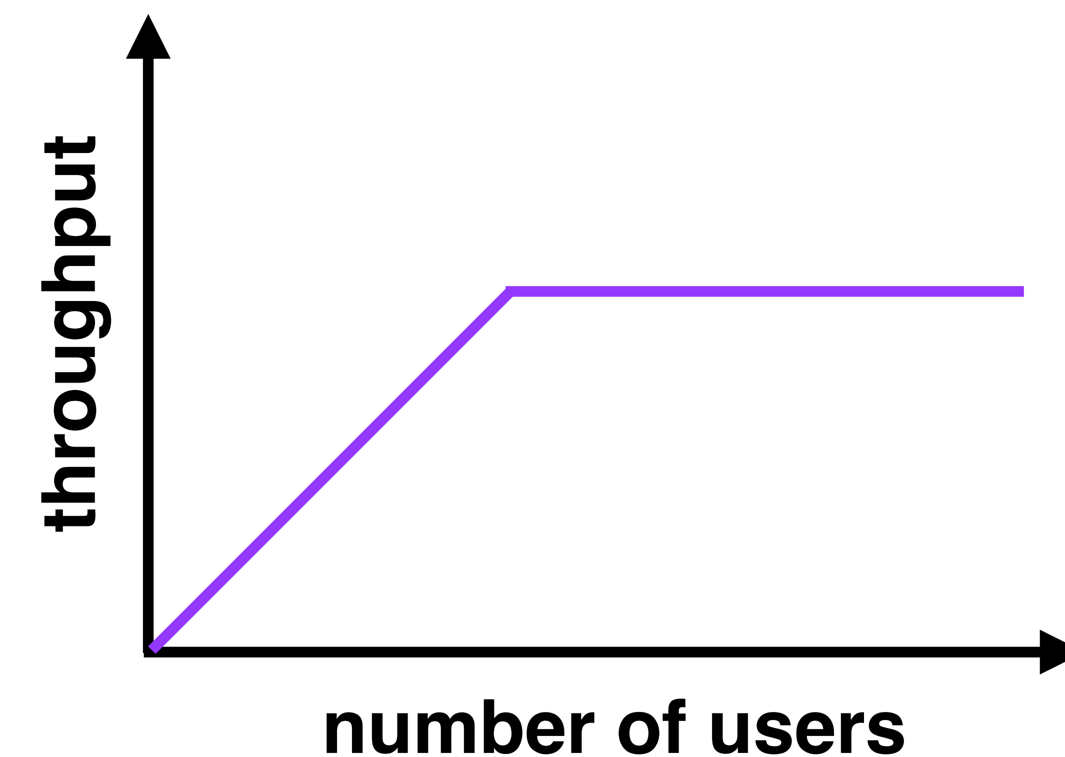
**latency:** how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of memory in an OS?




**throughput:** how many requests per unit of time?

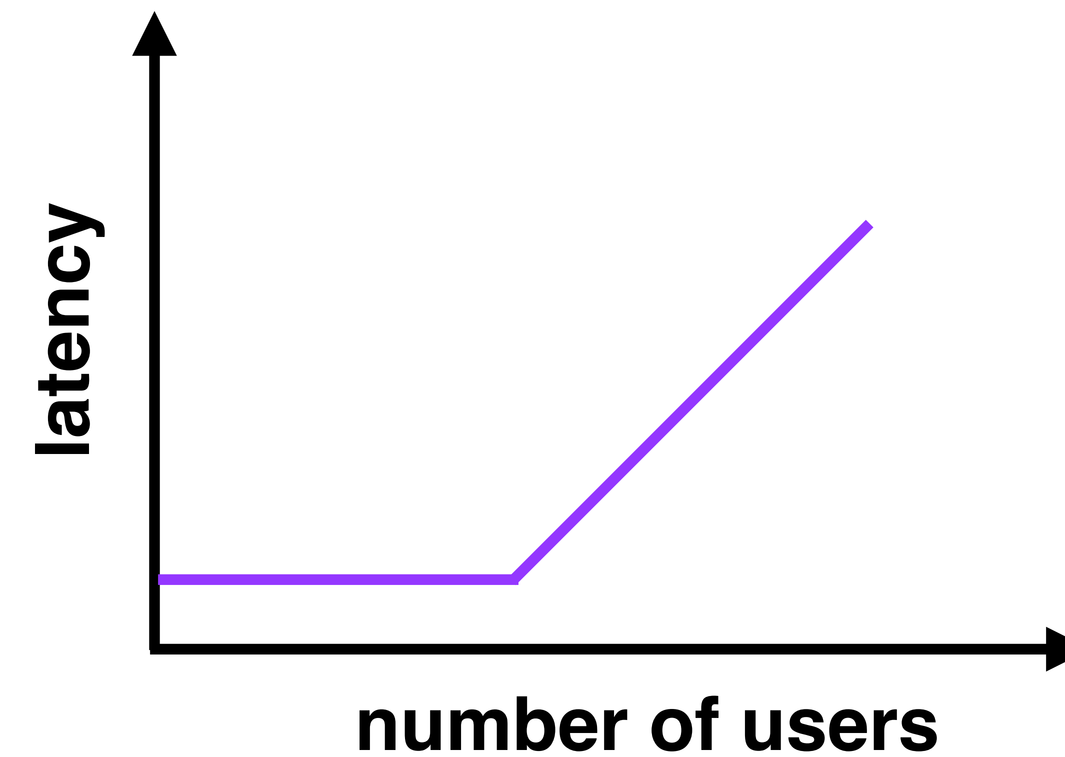
example: how many reads or writes can a system do to a disk at once?



**performance:** performance issues have influenced a lot of the system designs you've seen so far

**latency:** how long does it take to complete a single request?

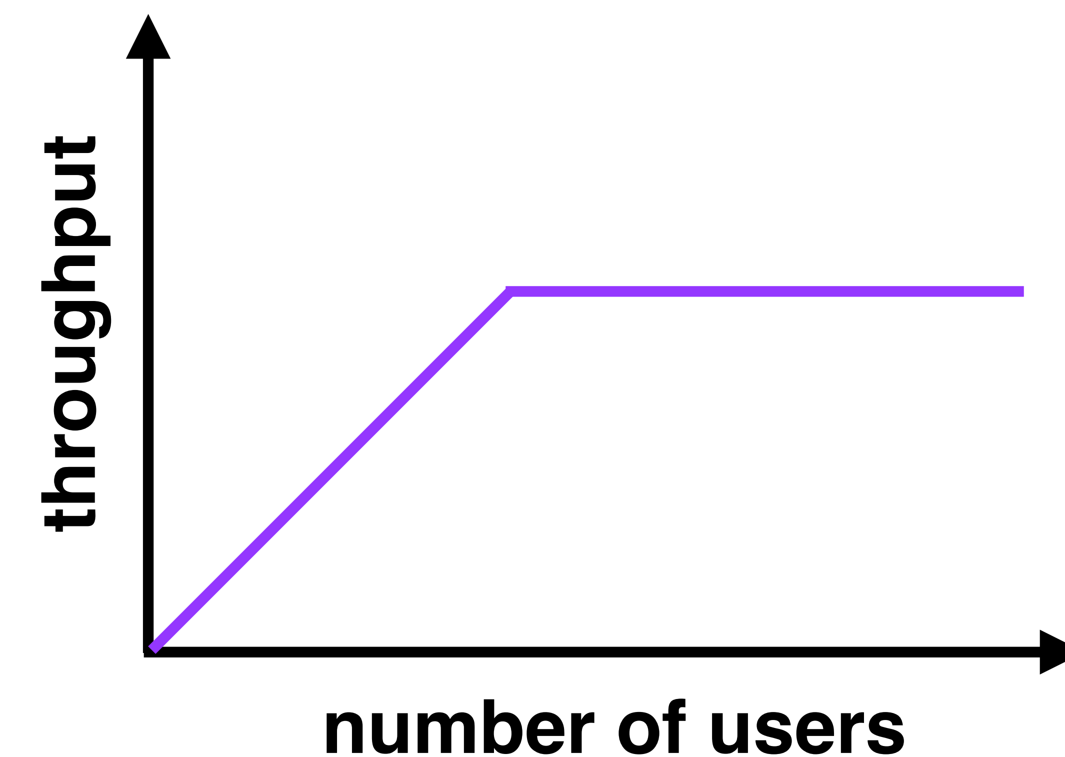
example: how long does it take to retrieve a particular piece of memory in an OS?




often we see latency remain low until the system is heavily utilized, and then it rises.

**throughput:** how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

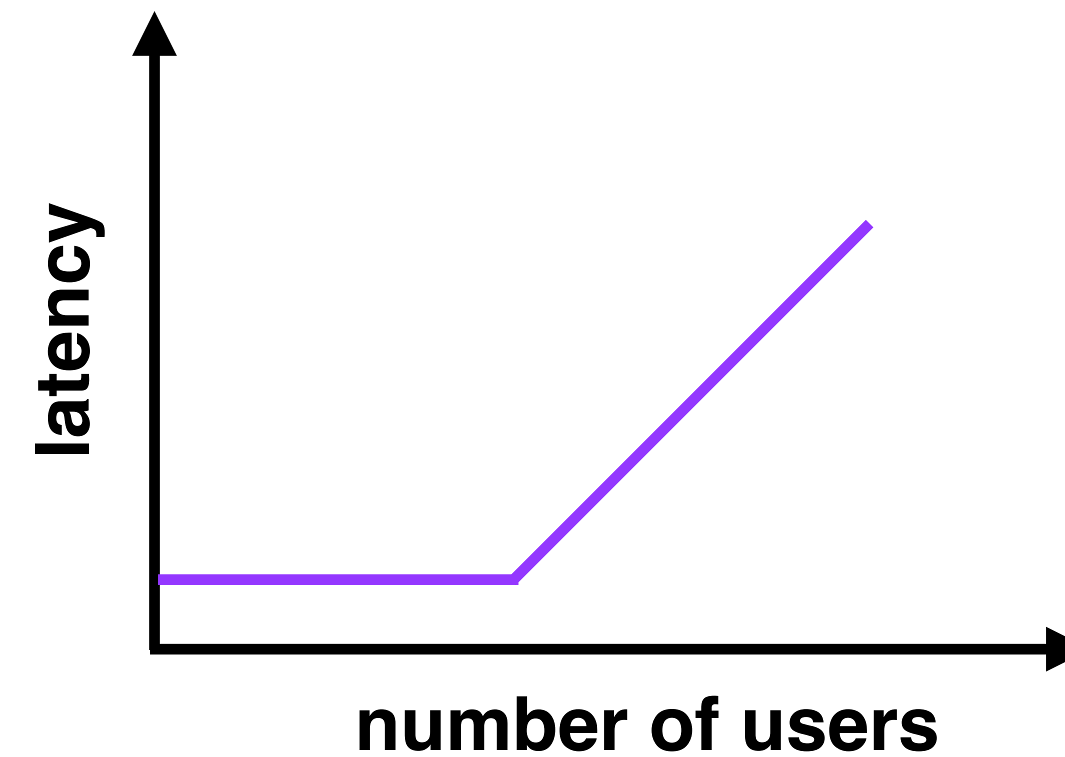



throughput, on the other hand, increases as requests increase, and maxes out when the system is fully utilized

**performance:** performance issues have influenced a lot of the system designs you've seen so far

**latency:** how long does it take to complete a single request?

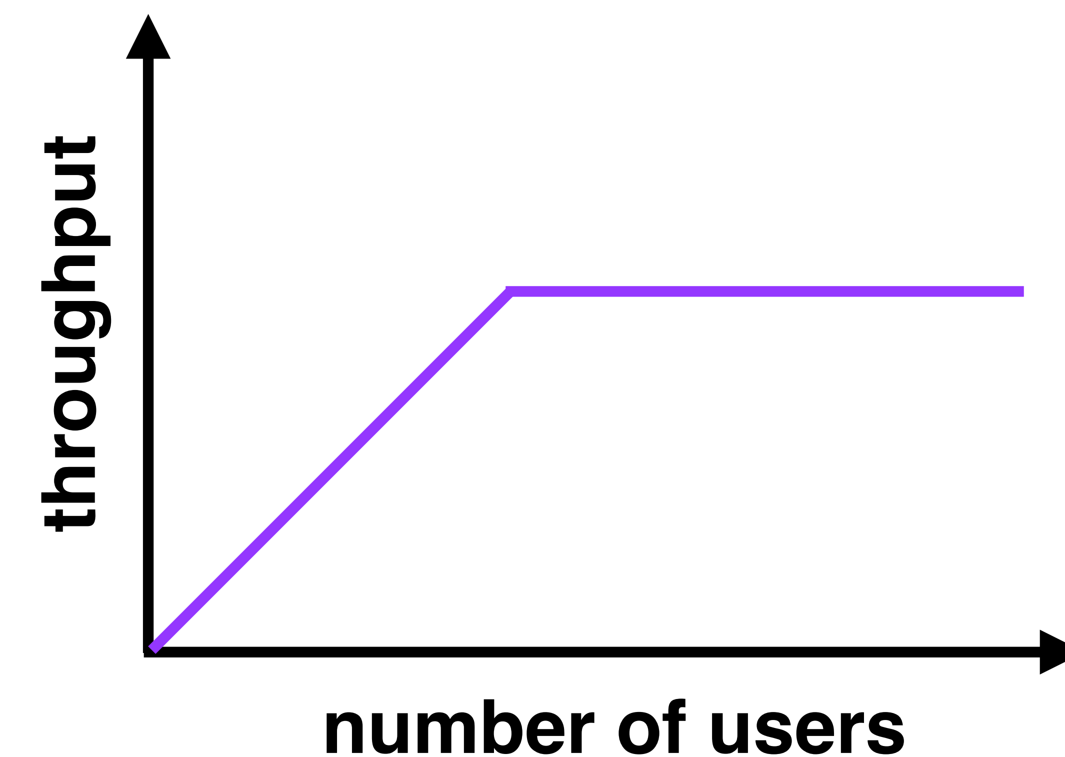
example: how long does it take to retrieve a particular piece of memory in an OS?




often we see latency remain low until the system is heavily utilized, and then it rises.

**throughput:** how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?




throughput, on the other hand, increases as requests increase, and maxes out when the system is fully utilized

**utilization:** what fraction of resources are being utilized? this puts our performance measurements in context

**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.

**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.

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

**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.

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

**performance** affects all aspects of system design. **throughput**, **latency**, and **utilization** are important metrics (but not the only ones), and we have general, systems-level techniques for attempting to improve them (e.g., caching, batching). knowing when to apply them requires a solid understanding of the system itself and how it's used