UNIVERSITY of VIRGINIA

# CS4740
# CLOUD COMPUTING

## Virtualization

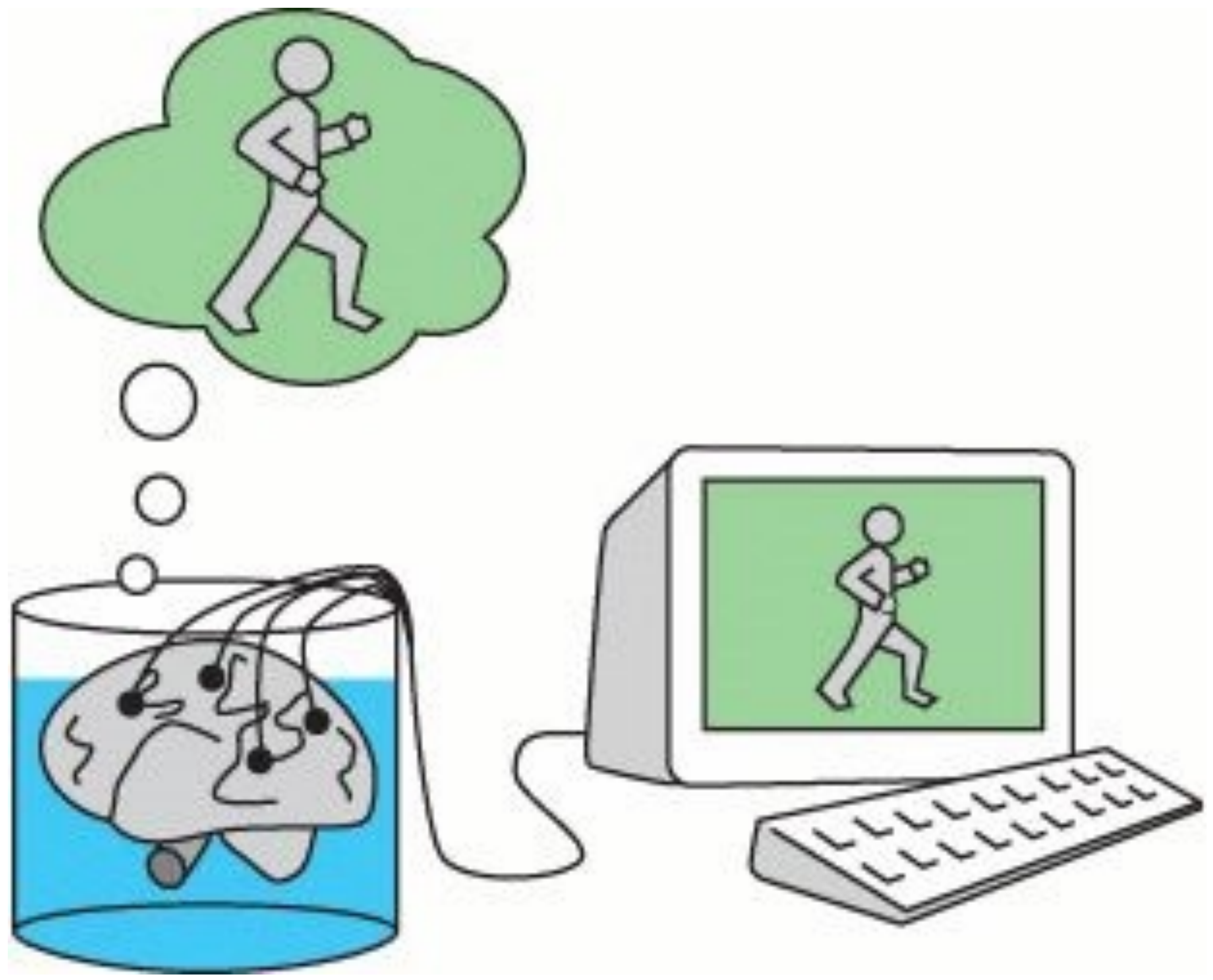
Prof. Chang Lou, UVA CS, Spring 2024

# Are we living in the real world, or a simulation?

# ARE WE LIVING IN A COMPUTER SIMULATION?

By Nick Bostrom

*I argue that at least one of the following propositions is true: (1) the human species is very likely to become extinct before reaching a 'posthuman' stage; (2) any posthuman civilization is extremely unlikely to run a significant number of simulations of its evolutionary history (or variations thereof); (3) we are almost certainly living in a computer simulation. It follows that the belief that there is a significant chance that we shall one day become posthumans who run ancestor-simulations is false, unless we are currently living in a simulation. I discuss some consequences of this result.*



# INTRODUCTION

– If the answer is yes, is there a way to tell?

– Hope you can get some insights from today's lecture :)

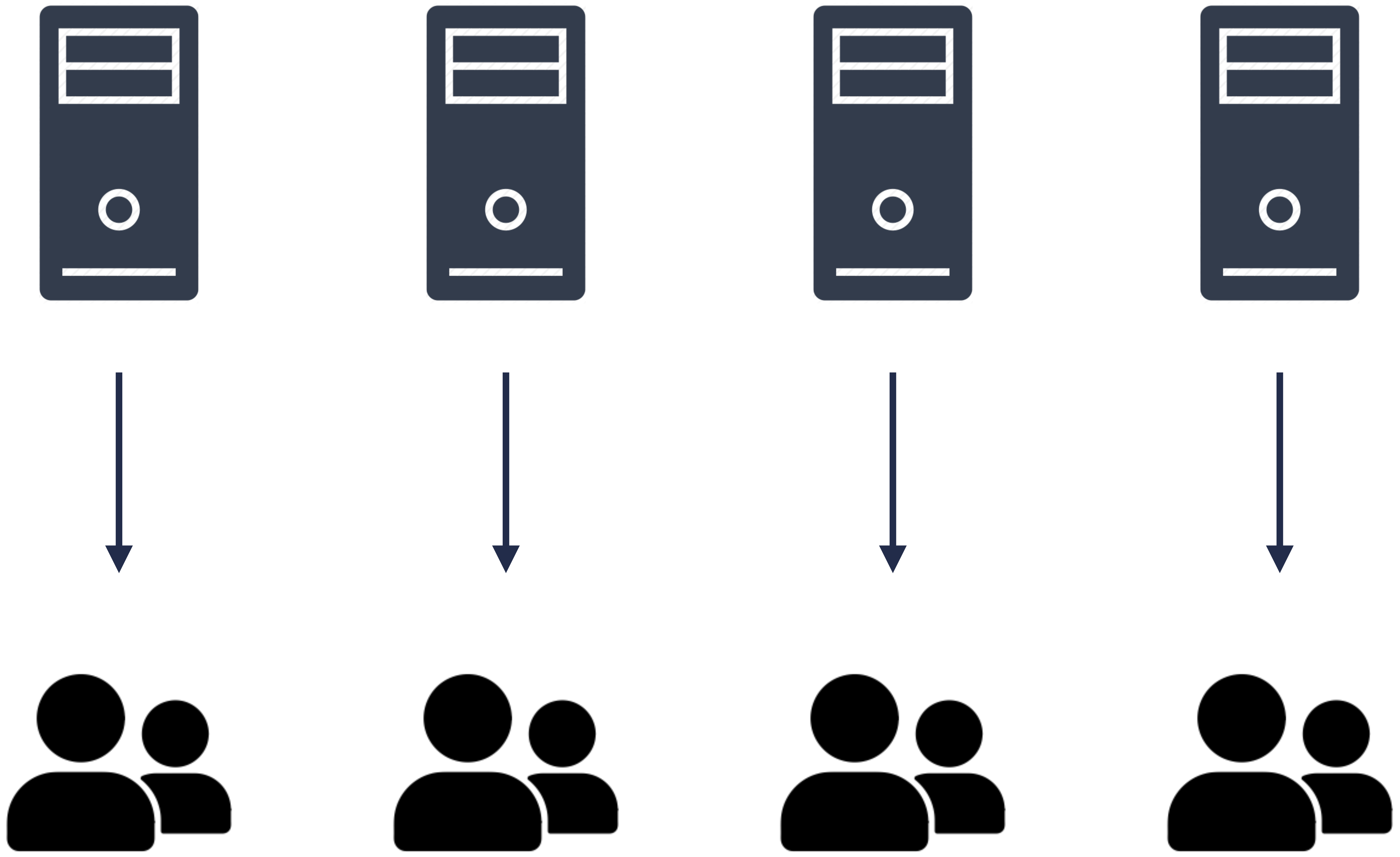
# AGENDA

– Motivation

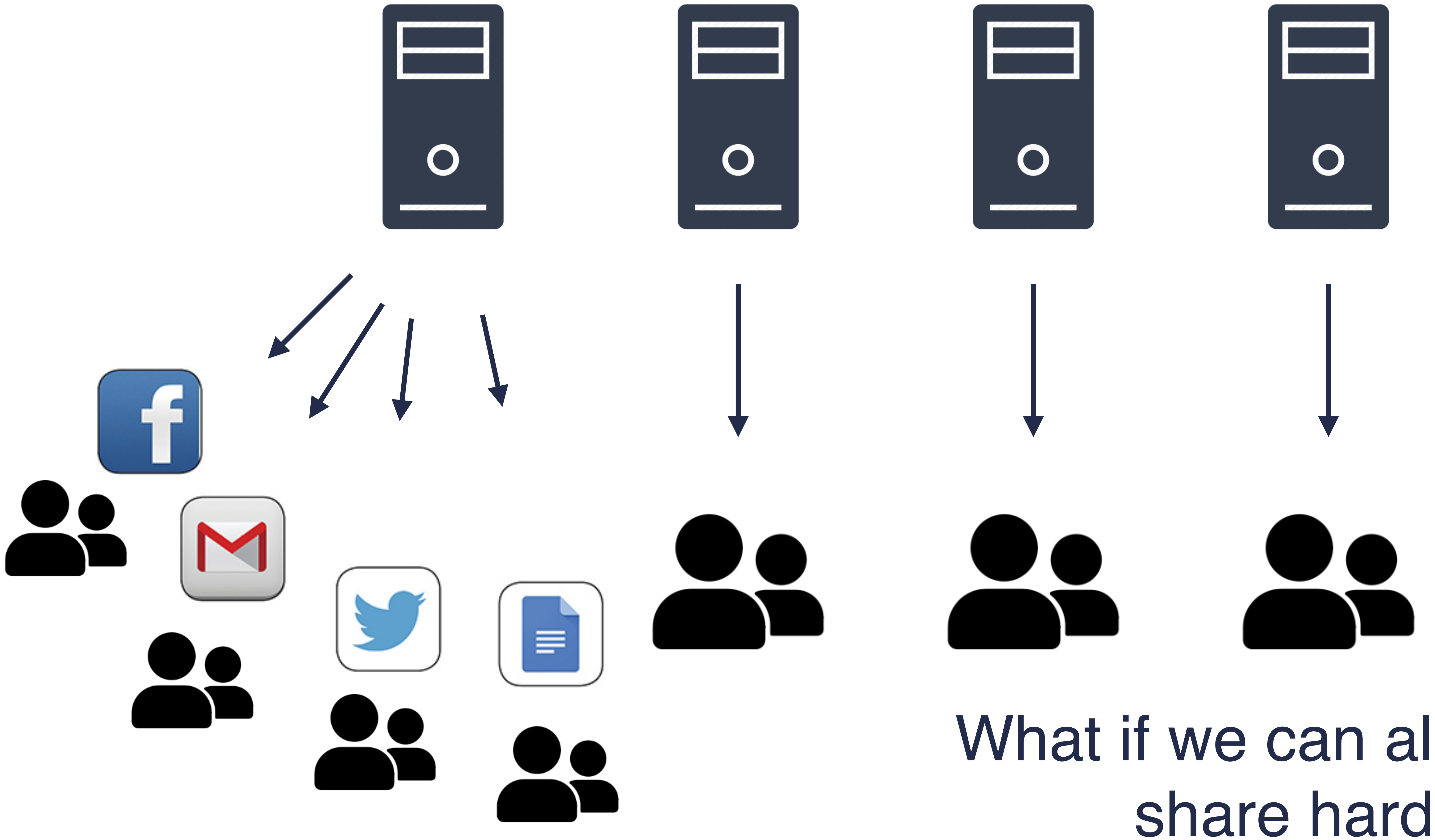
– Techniques
  – Virtual machines
  – Containers

*This lecture's slides are heavily based on Prof. Ryan Huang's OS course at UMich

# Why cloud computing needs virtualization?



Efficiency?



What if we can allow different customers share hardware resources?

# VIRTUALIZATION: BENEFITS

- Resource efficiency
  - Maximum use of the physical hardware’s computing capacity.
- Easier management
  - Automated IT service management workflows.
- Minimal downtime
  - Failover and migration.
- Faster provisioning

# VIRTUALIZATION: BENEFITS (CONTD.)

- Software compatibility
  - VMMs can run pretty much all software
- Isolation
  - Seemingly total data isolation between virtual machines
  - Leverage hardware memory protection mechanisms
- Encapsulation
  - Virtual machines are not tied to physical machines
  - Checkpoint/migration
- Many other cool applications
  - Debugging, emulation, security, speculation, fault tolerance…

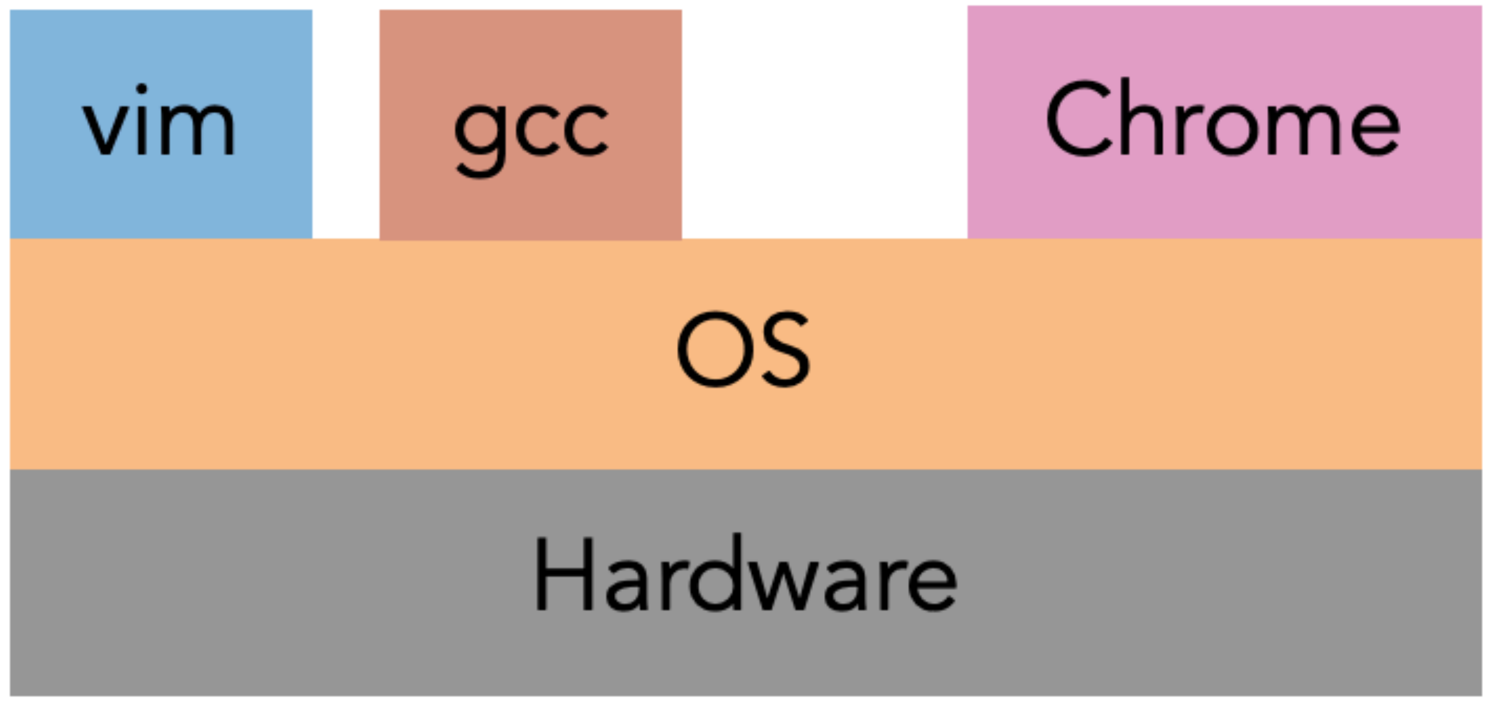
# OLD IDEA FROM THE 1970S

- IBM VM/370 – A VMM for IBM mainframe
  - Multiplex multiple OS environments on expensive hardware
  - Desirable when few machines around
- Interest died out in the 1980s and 1990s
  - Hardware got cheap
  - Compare Windows NT vs. N DOS machines
- Revived by the Disco [SOSP '97] work
  - Led by Mendel Rosenblum, later lead to the foundation of VMware
- Another important work Xen [SOSP '03]

# How to implement virtualization?

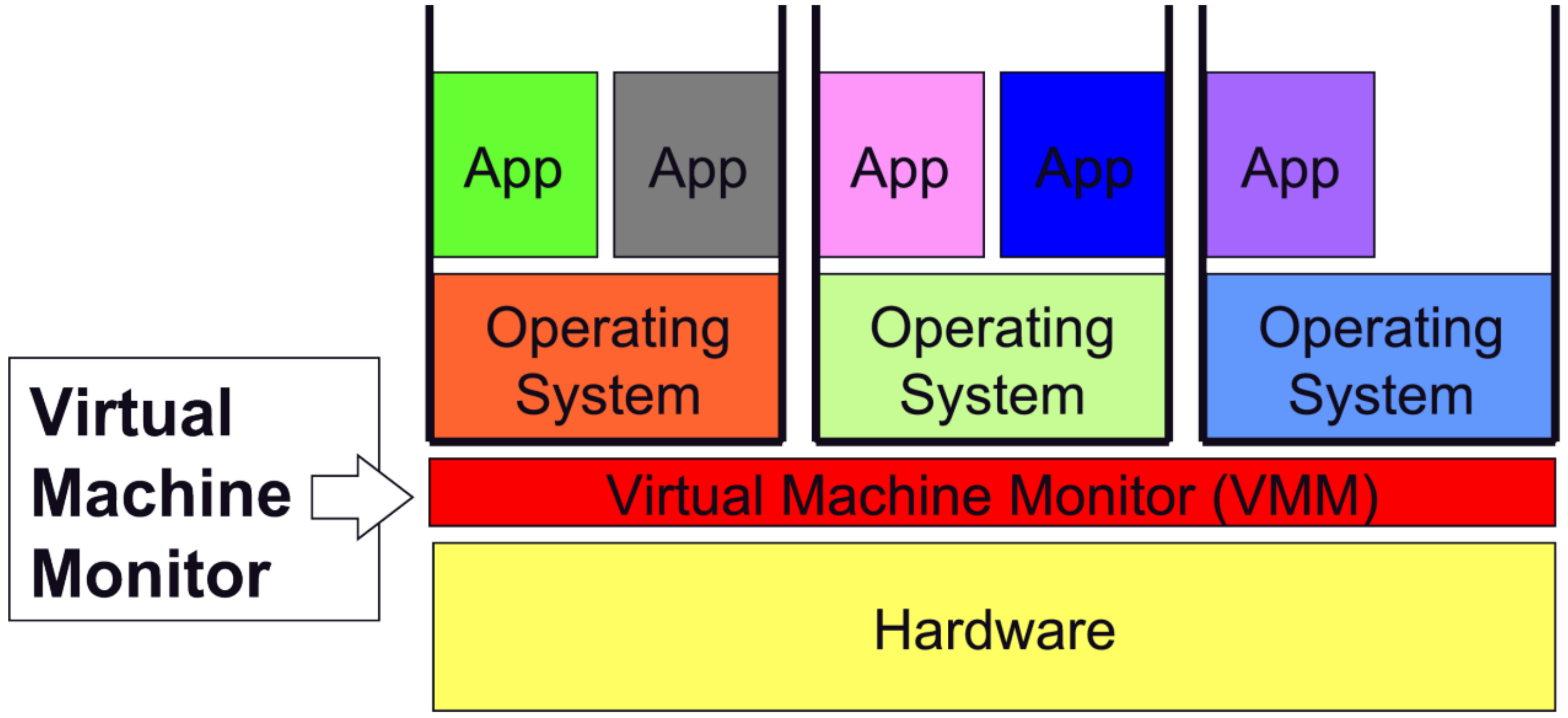
# BACKGROUND: OS

- OS is software between applications and hardware
  - Abstracts hardware to makes applications portable
  - Makes finite resources (memory, # CPU cores) appear much larger
  - Protects processes and users from one another



# VIRTUAL MACHINE

- Thin layer of software that virtualizes the hardware
  - Exports a virtual machine abstraction that looks like the hardware
  - Provides the illusion that software has full control over the hardware
  - Run multiple instances of an OS or different OSes simultaneously on the same physical machine



# VMMS TODAY

- VMs are used everywhere
  - Popularized by cloud computing
  - Used to solve different problems
- VMMs are a hot topic in industry and academia
  - Industry commitment
    - Software: VMware, Xen,…
    - Hardware: Intel VT, AMD-V
  - Academia: lots of related projects and papers

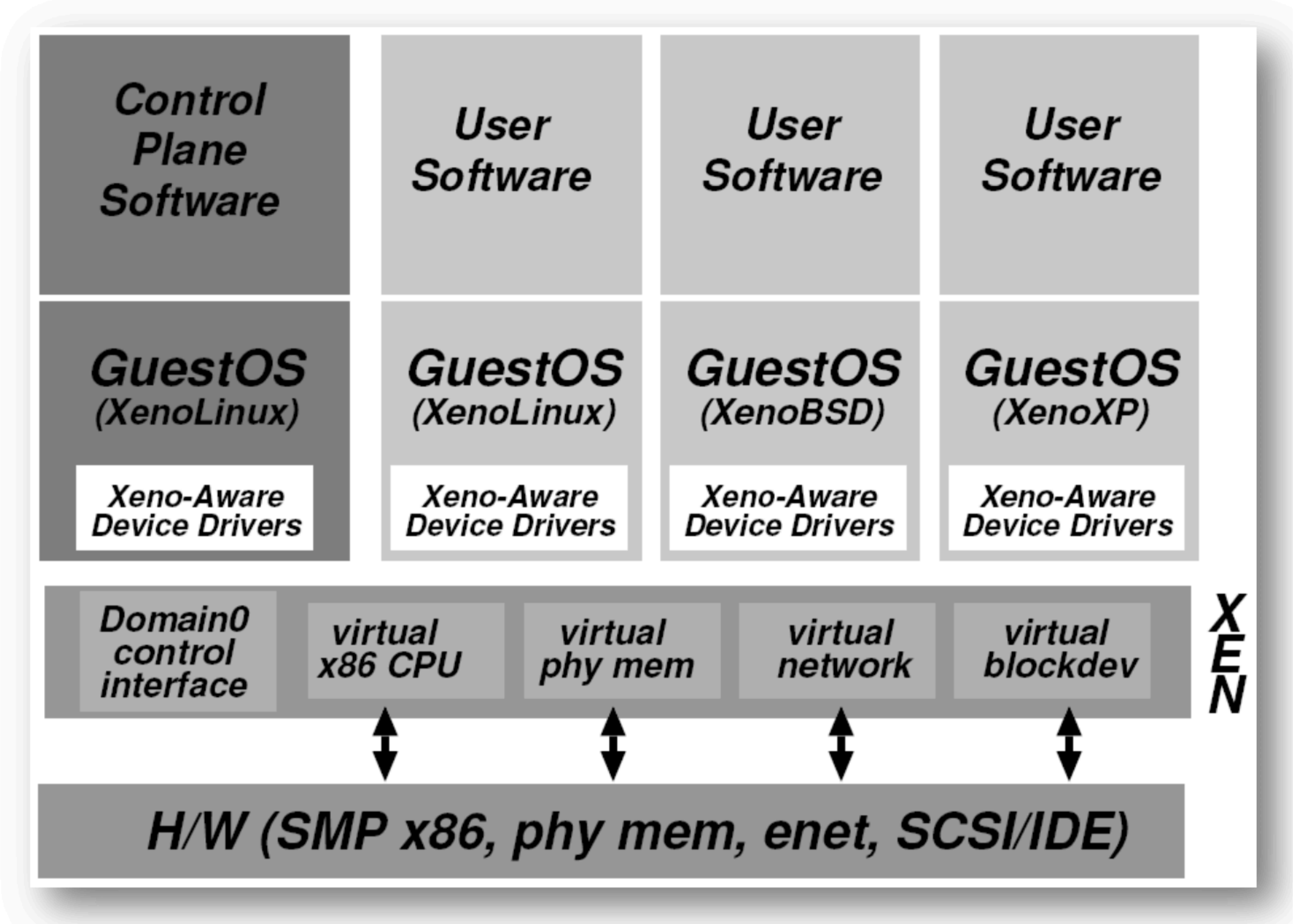
# IMPLEMENTING VMMS - REQUIREMENTS

- Fidelity
  - OSes and applications work the same without modification
  - (although we may modify the OS a bit)
- Isolation
  - VMM protects resources and VMs from each other
- Performance
  - VMM is another layer of software…and therefore overhead
  - As with OS, want to minimize this overhead
  - CPU-intensive apps: 2-10% overhead (early)
  - I/O-intensive apps: 25-60% overhead (much better today)

# VMM CASE STUDY 1: XEN

– Early versions use "paravirtualization"
  – Fancy word for "we have to modify & recompile the OS"
  – Since you're modifying the OS, make life easy for yourself
  – Create a VMM interface to minimize porting and overhead
– Xen hypervisor (VMM) implements interface
  – VMM runs at privilege, VMs (domains) run unprivileged
  – Trusted OS (Linux) runs in own domain (Domain0)
– Most recent version of Xen does not require OS mods
  – Because of Intel/AMD hardware support
– Commercialized via XenSource, but also open source

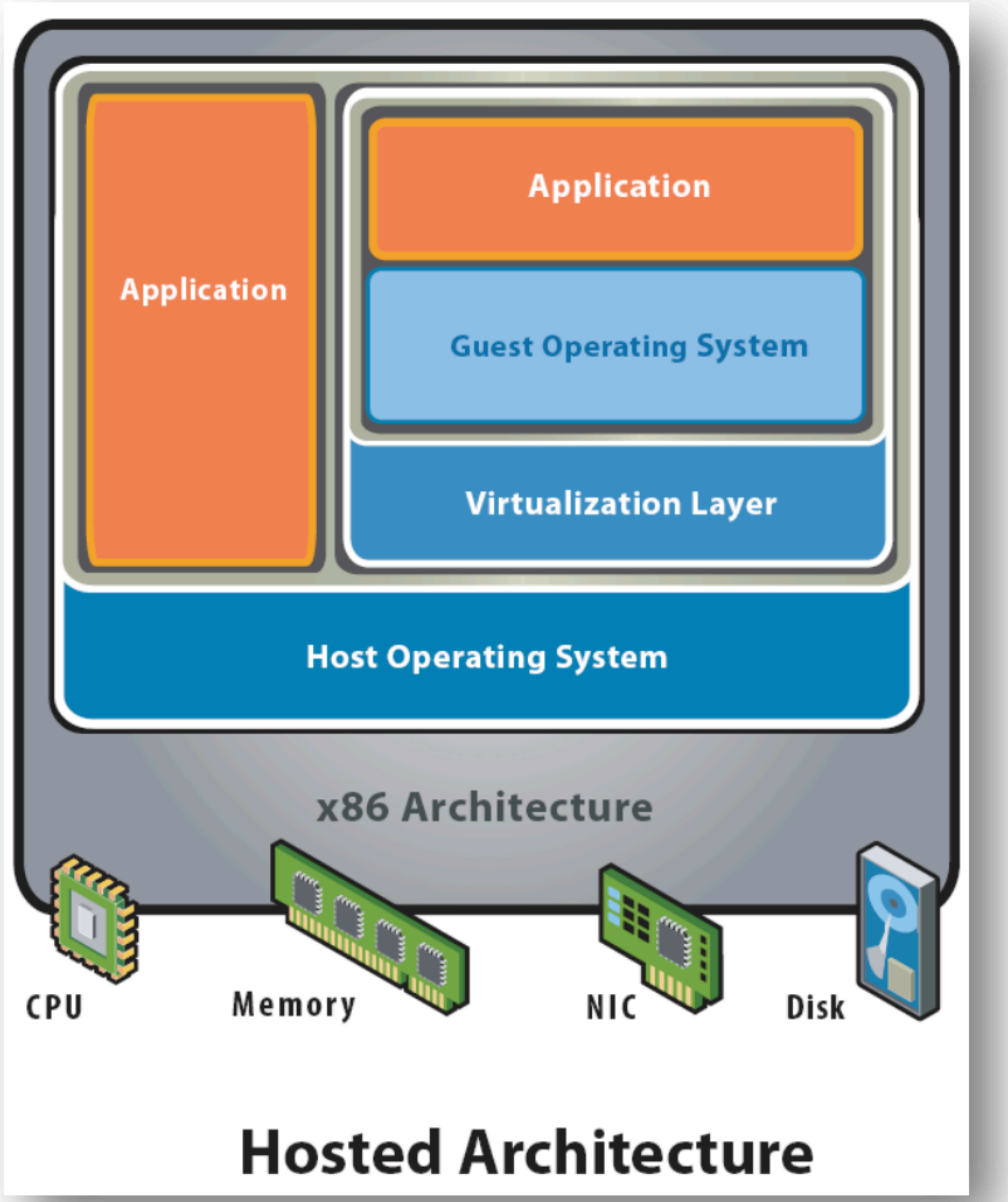
# XEN ARCHITECTURE



# VMM CASE STUDY 2: VMWARE

- VMware workstation uses hosted model
  - VMM runs unprivileged, installed on base OS (+ driver)
  - Relies upon base OS for device functionality
- VMware ESX server uses hypervisor model
  - Similar to Xen, but no guest domain/OS
- VMware uses software virtualization
  - Dynamic binary rewriting translates code executed in VM
  - Think JIT compilation for JVM, but full binary x86 -> IR code -> safe subset of x86
  - Incurs overhead, but can be well-tuned (small % hit)

# VMWARE HOSTED ARCHITECTURE



# WHAT NEEDS TO BE VIRTUALIZED?

- Exactly what you would expect
  - CPU, Memory, I/O devices, Events (exceptions and interrupts)

- How to do it?

# APPROACH 1: COMPLETE MACHINE SIMULATION

- Simplest VMM approach, used by bochs
- Run the VMM as a regular user application atop a host OS
- Application simulates all the hardware (i.e., a simulator)
  - CPU – A loop that fetches each instruction, decodes it, simulates its effect

```
while (1) {
  curr_instr = fetch(virtHw.PC); virtHw.PC += 4;
  switch (curr_instr) {
    case ADD:
      int sum = virtHw.regs[curr_instr.reg0] +
        virtHw.regs[curr_instr.reg1];
    virtHw.regs[curr_instr.reg0] = sum;
    break;
    case SUB: //...
```

  - Memory – Memory is just an array, simulate the MMU on all memory accesses
  - I/O – Simulate I/O devices, programmed I/O, DMA, interrupts

# APPROACH 1: COMPLETE MACHINE SIMULATION

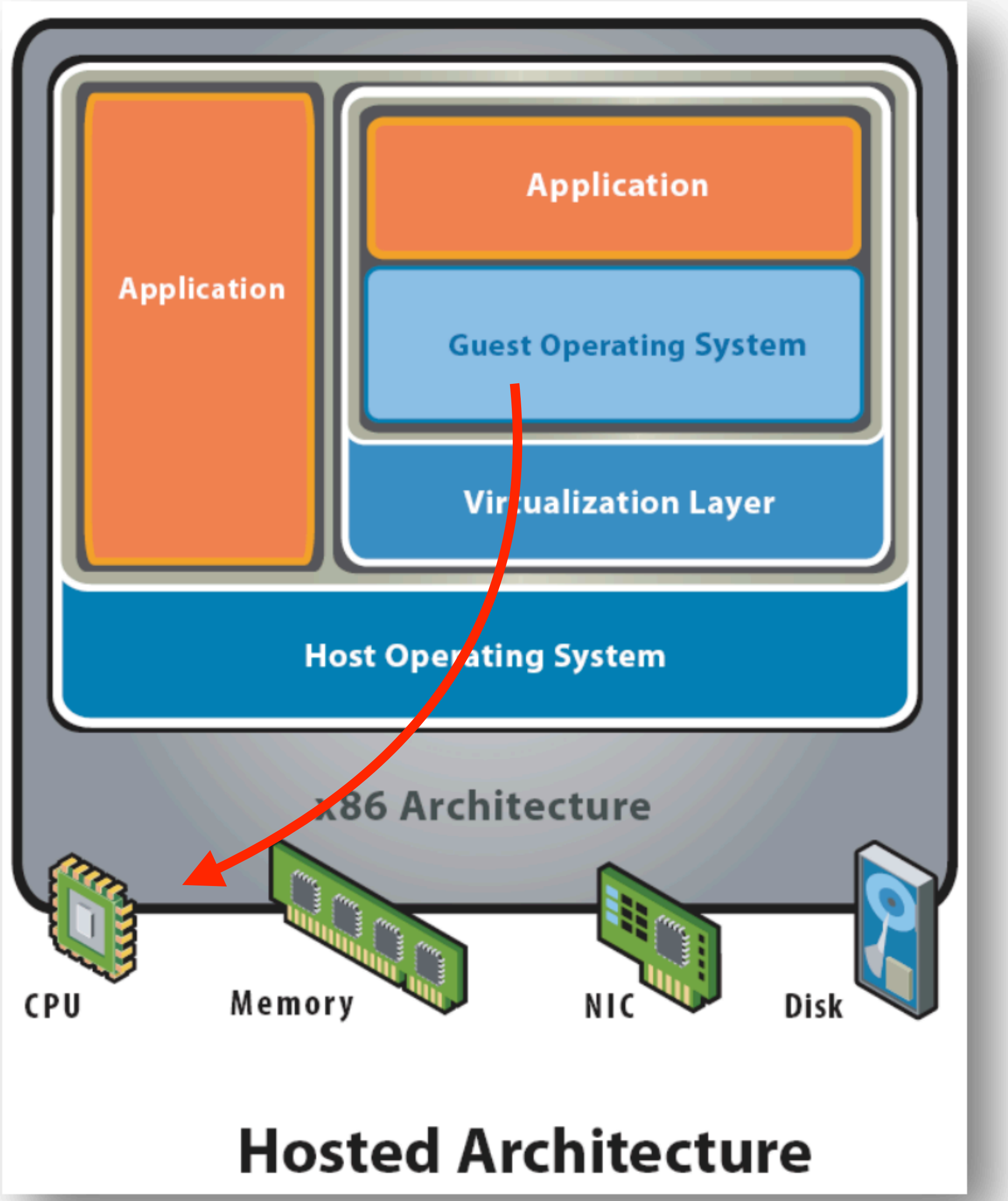

This 8-bit processor built in Minecraft can run its own games I PCWorld

# APPROACH 1: COMPLETE MACHINE SIMULATION

- Simplest VMM approach, used by bochs
- Run the VMM as a regular user application atop a host OS
- Application simulates all the hardware (i.e., a simulator)
- Problem?
- Too slow!
  - CPU/Memory – 100x CPU/MMU simulation
  - I/O Device – < 2× slowdown.
  - 100× slowdown makes it not too useful
- Need faster ways of emulating CPU/MMU

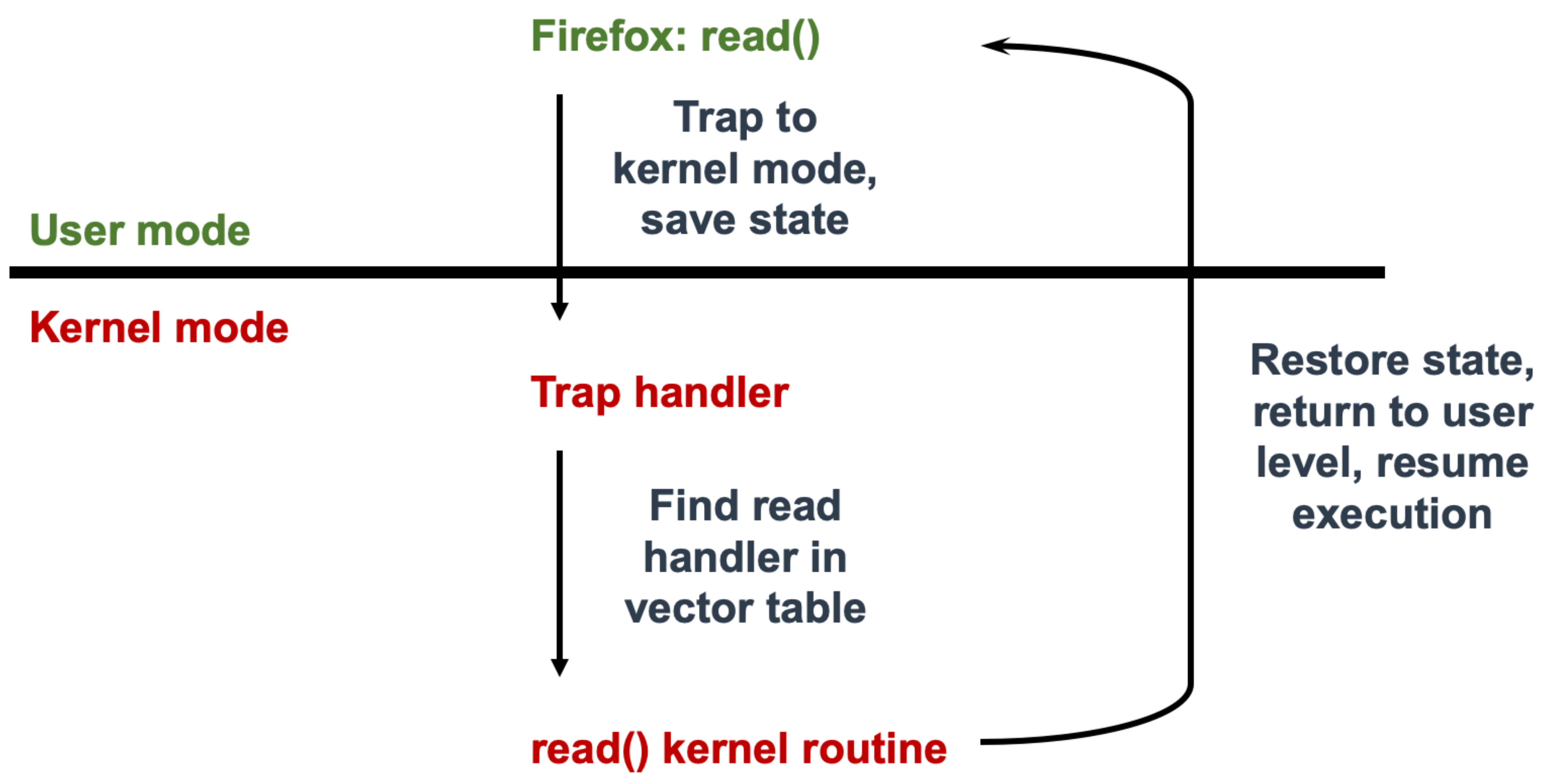
# APPROACH 2: DIRECT EXECUTION W/ TRAP & EMULATE

- Observations: Most instructions are the same regardless of processor privileged level
  - Example: incl %eax
- Why not just give instructions to CPU to execute?
  - One issue: Safety – How to get the CPU back? Or stop it from stepping on us? How about cli/halt?
  - Solution: Use protection mechanisms already in CPU



# BACKGROUND: DUAL-MODE OPERATION IN CPU

- User mode
  - Limited privileges
  - Only those granted by the operating system kernel
- Kernel mode
  - Execution with the full privileges of the hardware
  - Read/write to any memory, access I/O device, read/write disk sector, send/read packet



# APPROACH 2: DIRECT EXECUTION W/ TRAP & EMULATE

- Run virtual machine's OS directly on CPU in unprivileged user mode
  - "Trap and emulate" approach
  - Most instructions just work
  - Privileged instructions trap into monitor and run simulator on instruction

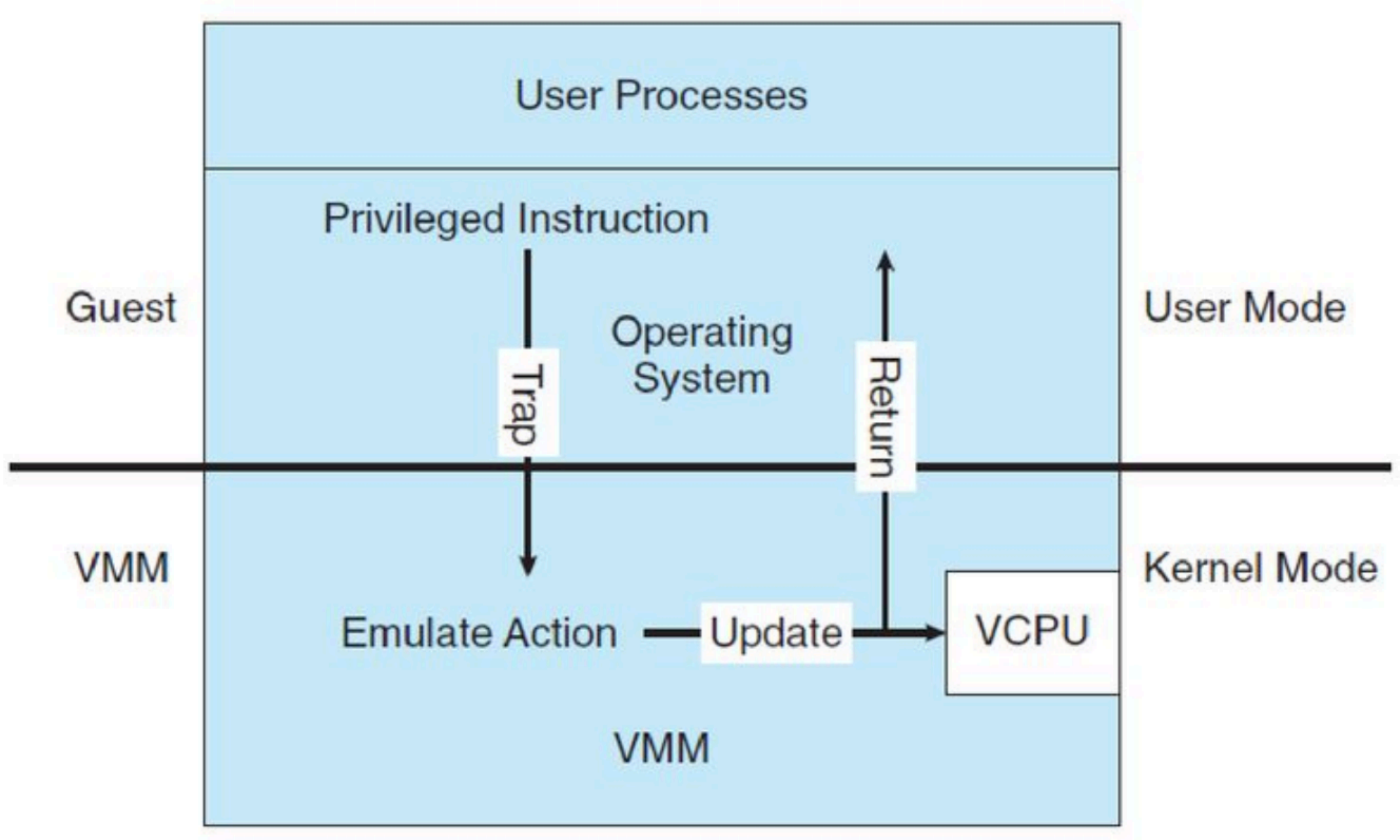
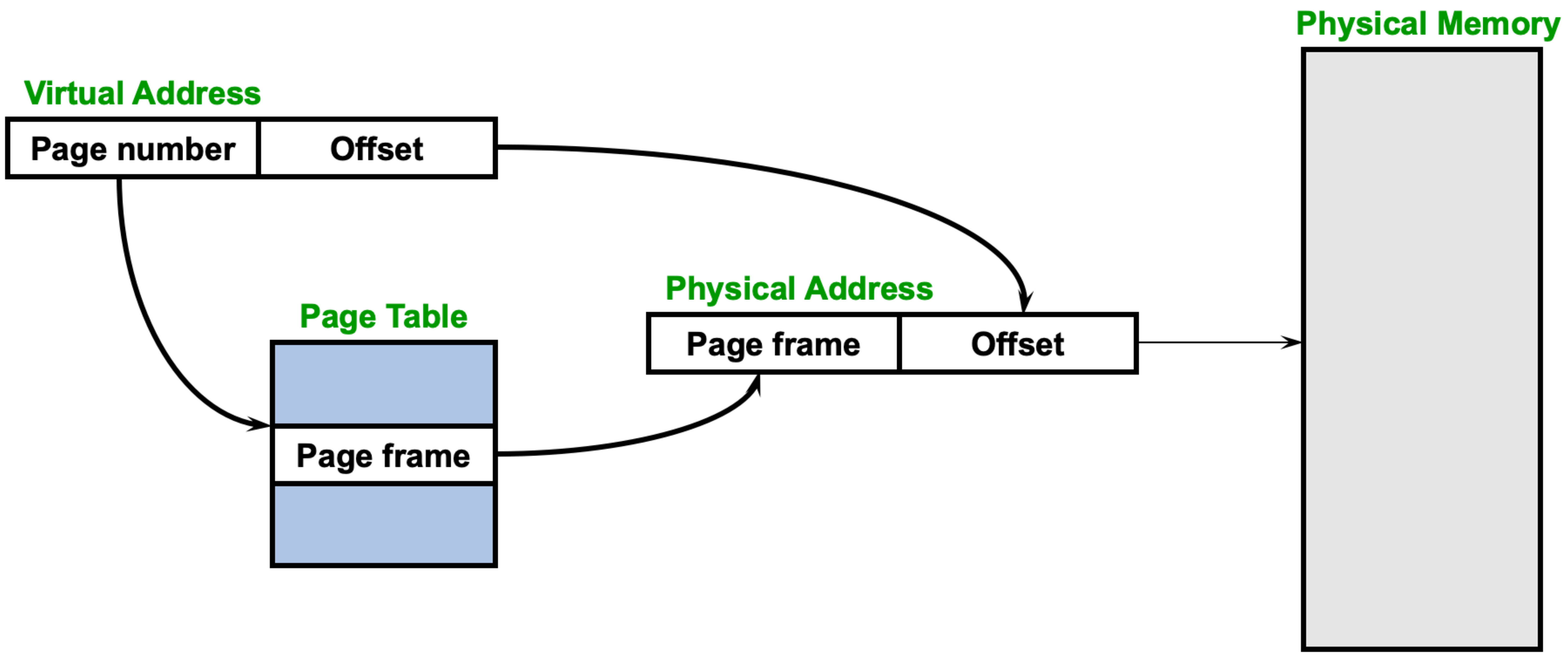
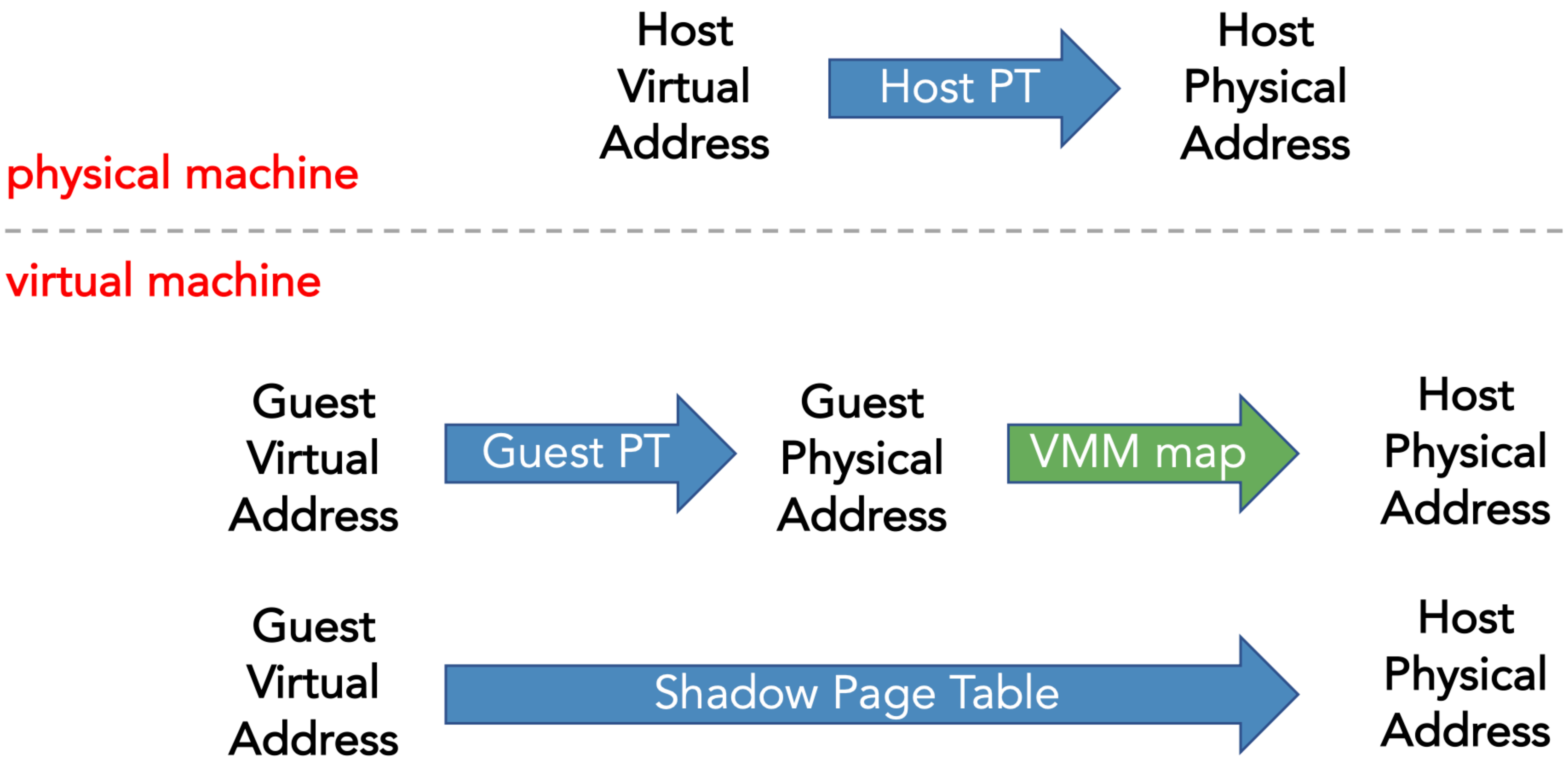



**Figure 16.2** Trap-and-emulate virtualization implementation.

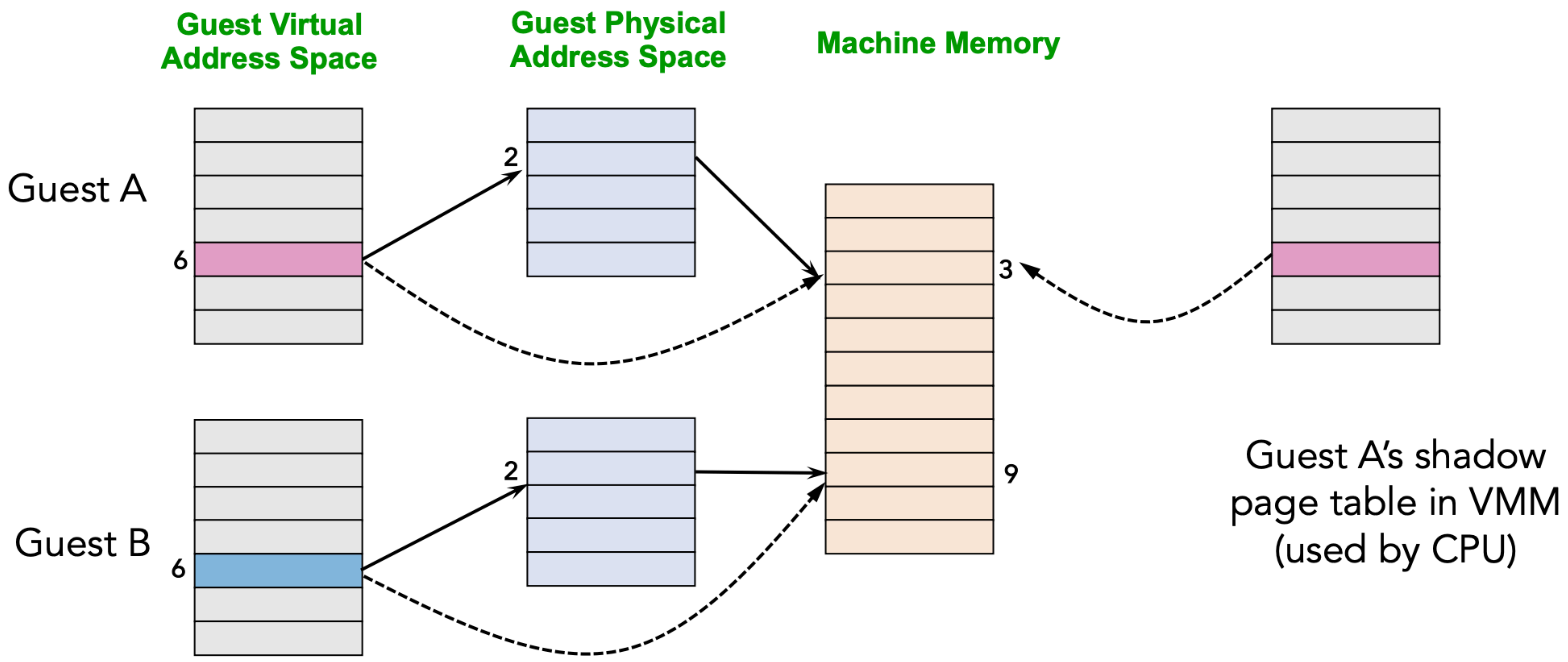
# BACKGROUND: PAGE TABLE



# VIRTUALIZING MEMORY



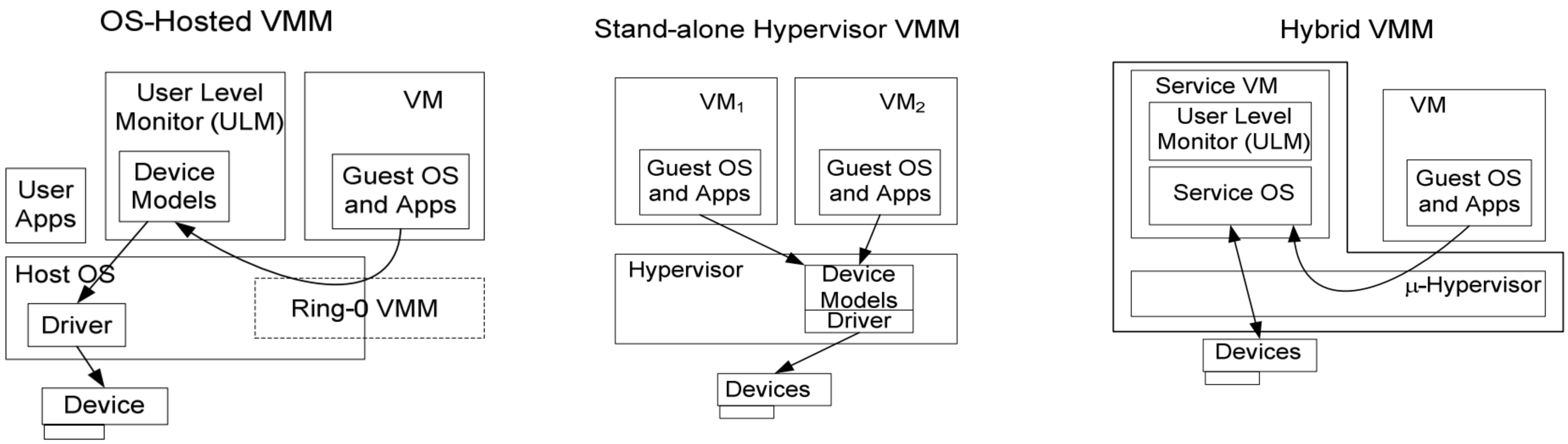
# VIRTUALIZING MEMORY



# VIRTUALIZING I/O: THREE MODES

- Xen: modify OS to use low-level I/O interface (**hybrid**)
  - Define generic devices with simple interface: Virtual disk, virtual NIC, etc.
  - Ring buffer of control descriptors, pass pages back and forth
  - Handoff to trusted domain running OS with real drivers
- VMware: VMM supports generic devices (**hosted**)
  - E.g., AMD Lance chipset/PCNet Ethernet device
  - Load driver into OS in VM, OS uses it normally
  - Driver knows about VMM, cooperates to pass the buck to a real device driver (e.g., on underlying host OS)
- VMware ESX Server: drivers run in VMM (**hypervisor**)

# VIRTUALIZING I/O: THREE MODES

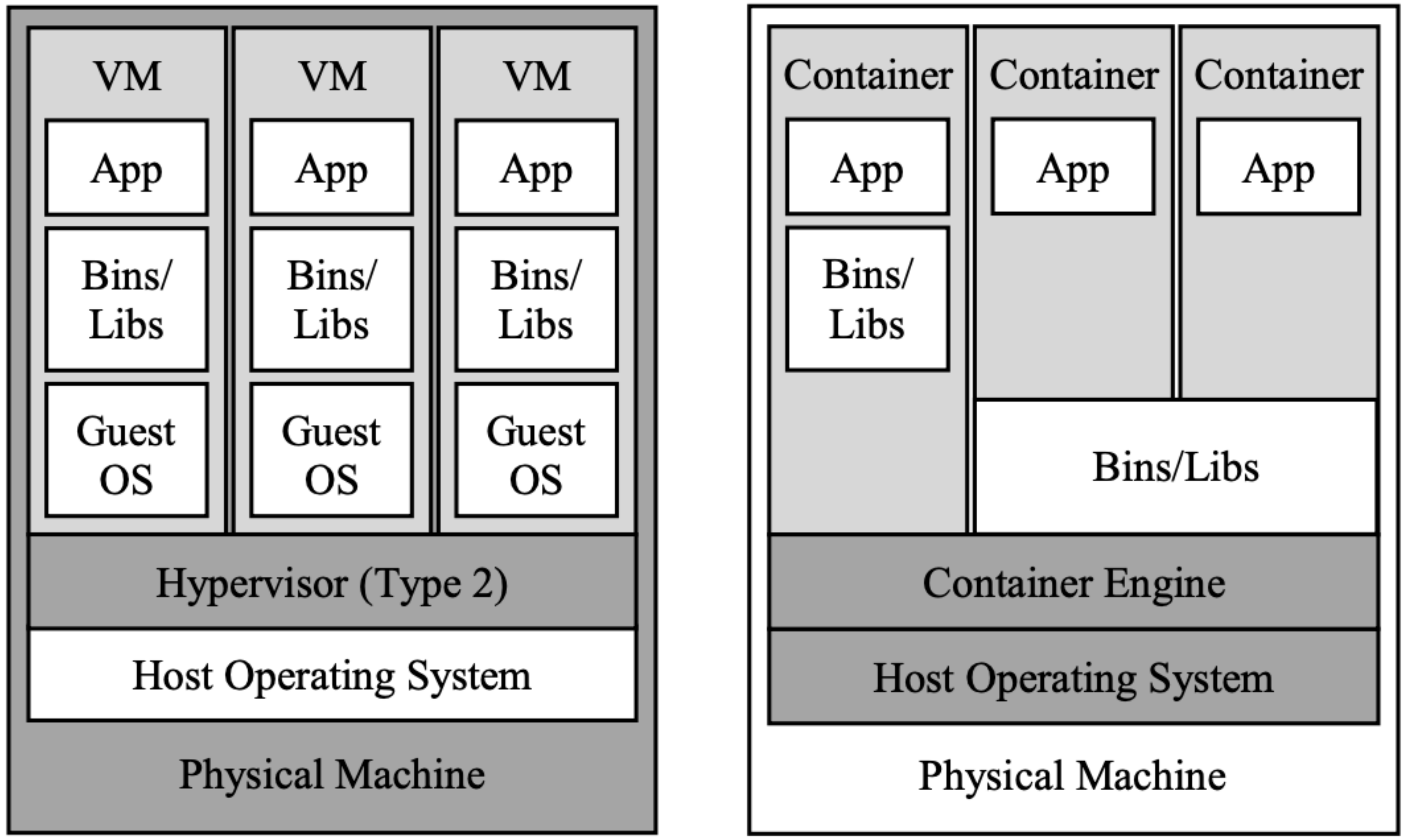



Abramson et al., "Intel Virtualization Technology for Directed I/O", Intel Technology Journal, 10(3) 2006

# Containers

# CONTAINERS

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7921010




(a) Hypervisor-based virtual service.

(b) Container-based virtual service.

# DIFFERENCES BETWEEN VMMS AND CONTAINERS

"Virtualization vs Containerization to support PaaS", VMware

| Parameter |
|---|
| Guest OS |
| Communi-cation |
| Security |
| Performance |
| Isolation |
| Startup time |
| Storage |

# DIFFERENCES BETWEEN VMMS AND CONTAINERS

"Virtualization vs Containerization to support PaaS", VMware

| Parameter | Virtual Machines | Containers |
|---|---|---|
| Guest OS | Each VM runs on virtual hardware and Kernel is loaded into in its own memory region | All the guests share same OS and Kernel. Kernel image is loaded into the physical memory |
| Communication | Will be through Ethernet Devices | Standard IPC mechanisms like Signals, pipes, sockets etc. |
| Security | Depends on the implementation of Hypervisor | Mandatory access control can be leveraged |
| Performance | Virtual Machines suffer from a small overhead as the Machine instructions are translated from Guest to Host OS. | Containers provide near native performance as compared to the underlying Host OS. |
| Isolation | Sharing libraries, files etc between guests and between guests hosts not possible. | Subdirectories can be transparently mounted and can be shared. |
| Startup time | VMs take a few mins to boot up | Containers can be booted up in a few secs as compared to VMs. |
| Storage | VMs take much more storage as the whole OS kernel and its associated programs have to be installed and run | Containers take lower amount of storage as the base OS is shared |

# DOCKER

- A software platform that allows you to build, test, and deploy applications quickly.
- Docker packages software into standardized units
  - including libraries, system tools, code, and runtime.
- Quickly deploy and scale applications into any environment

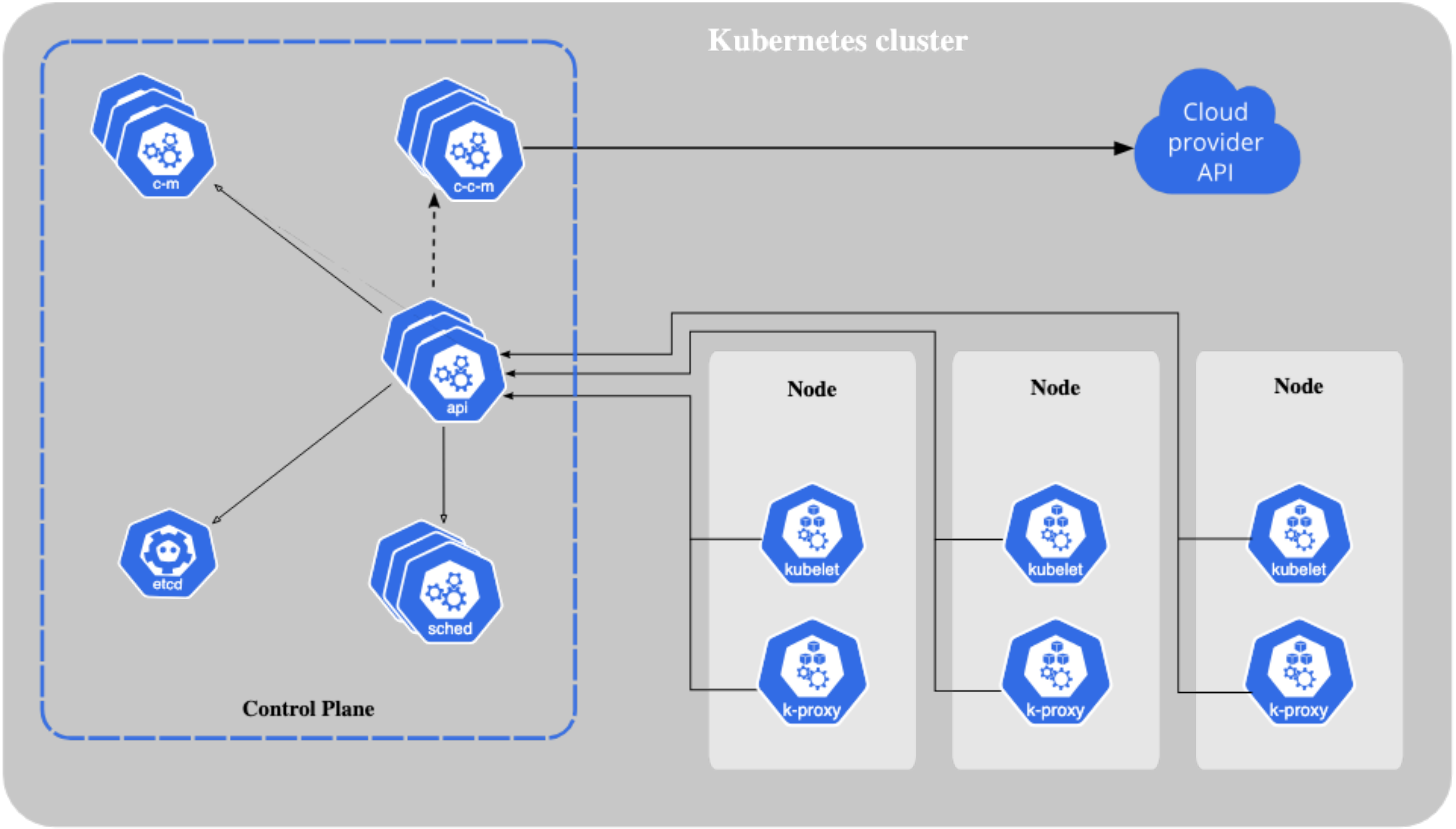
```
ARG CODE_VERSION=latest
FROM ubuntu:${CODE_VERSION}
COPY ./examplefile.txt /examplefile.txt
ENV MY_ENV_VARIABLE="example_value"
RUN apt-get update

# Mount a directory from the Docker volume
# Note: This is usually specified in the 'docker run' command.
VOLUME ["/myvolume"]

# Expose a port (22 for SSH)
EXPOSE 22
```

Example Docker file

# KUBERNETES (K8S)

- An open-source container orchestration system
- Inspired by Google's Borg cluster manager
- Widely deployed software systems in the world
- Features
  - Service discovery and load balancing
  - Storage orchestration
  - Automated rollouts and rollbacks
  - Self-healing
  - ...

# KUBERNETES (K8S)



## IS THERE A WAY TO TELL LIVING IN VIRTUALIZATION OR REALITY?

- For humans, maybe not
- For software, especially malware, yes
  - Scan registry entries specific to virtualization software
  - Monitor for specific processes or files that are typically part of virtualization software
  - Check firmware: known MAC addresses associated with virtual machines, check BIOS serial numbers
  - Run specialized instruction sets
- Some malware try to escape from VMs (*run, Neo!*)

# TAKEAWAYS

- VMMs multiplex virtual machines on hardware
- Implementing VMMs by virtualizing CPU, Memory, I/O
- Lesson: Never underestimate the power of **indirection**
- Next class: **Hacker Day**
  - fix bugs in lab 2a, 2b and implement lab 2c!

UNIVERSITY of VIRGINIA

# ACKNOWLEDGEMENT

THIS COURSE IS DEVELOPED HEAVILY BASED ON COURSE MATERIALS SHARED BY PROF. INDRANIL GUPTA, PROF. ROBERT MORRIS, PROF. MICHAEL FREEDMAN, PROF. KYLE JAMIESON, PROF. WYATT LLOYD AND PROF. ROXANA GEAMBASU. MANY APPRECIATIONS FOR GENEROUSLY SHARING THEIR MATERIALS AND TEACHING INSIGHTS.

THIS LECTURE INCORPORATES MATERIALS FROM PROF. RYAN HUANG'S OS COURSE AND PROF. ZAOXING (ALAN) LIU'S CLOUD COMPUTING COURSE.