CSE 451: Operating Systems
Winter 2026
Virtual Machines

Jonathan Trinh

Administrivia

- Lab 4 Design Doc due Tomorrow!
- P-Set 3 releases Friday after class
  - Due the week after

Agenda

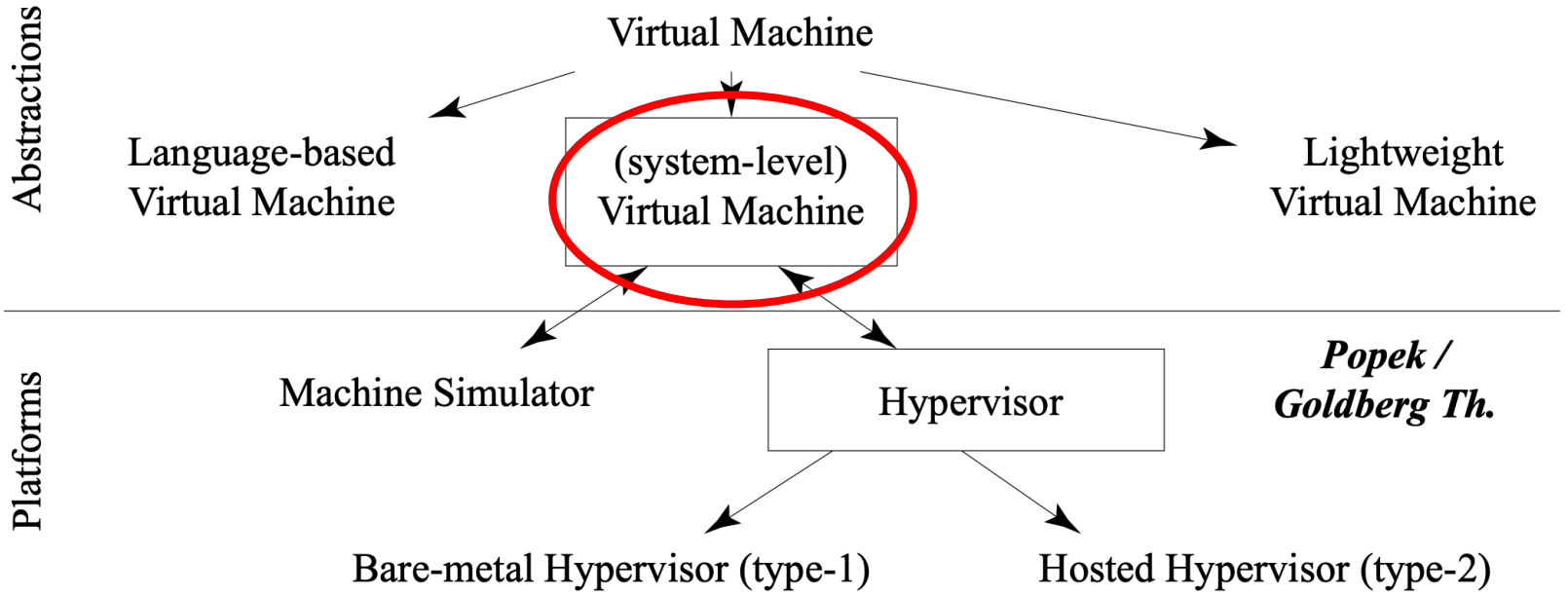
- Overview of Virtual Machines
- General Mechanisms of Virtual Machines
  - System Calls
  - Memory Management
  - I/O

Motivation

- Isolation & Security
  - Developing Operating Systems
    - Isolate system bugs and protect hardware
  - Running and testing dangerous software ("**Sandboxing**")
- Legacy Support for Applications
  - Simulate environments to run legacy programs on modern hardware
- Data Centers and Cloud Computing
  - Cloud Providers own massive data centers with unbelievable amounts of hardware and server racks
  - Providers run many Virtual Machines on top of their hardware to distribute services to their clients

What is a Virtual Machine

- General Definition: A complete compute environment with its own isolated processing capabilities, memory, and communication channels

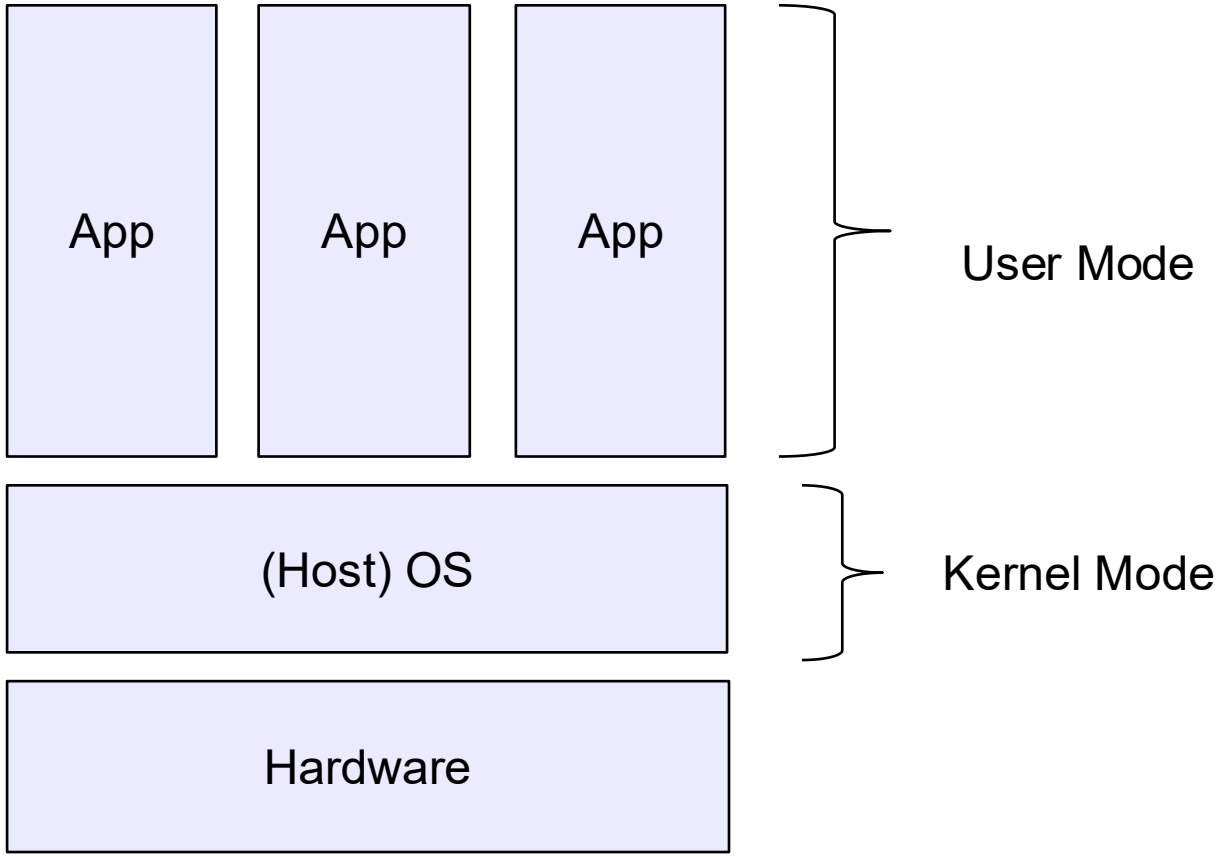


VM Terminology

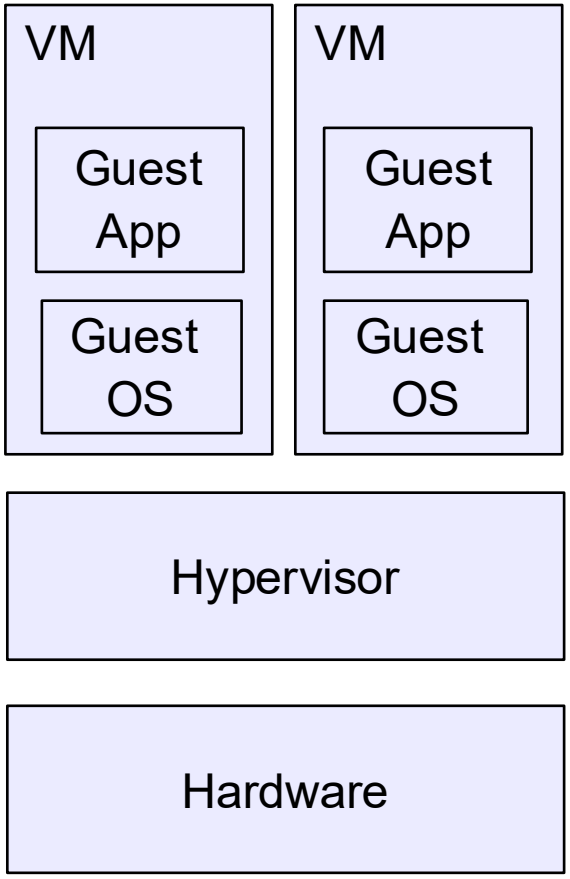
- Host OS: The primary operating system running directly on the physical hardware
- Guest OS: The operating system running within the Virtual Machine using the virtualized hardware

- Hypervisor: Software that creates and runs Virtual Machines with main purpose of virtualizing hardware
  - Type-1 (Bare Metal):
    - Ex: Microsoft Hyper-V, VMware, Xen
  - Type-2 (Hosted):
    - Ex: VirtualBox, VMware Workstation, KVM

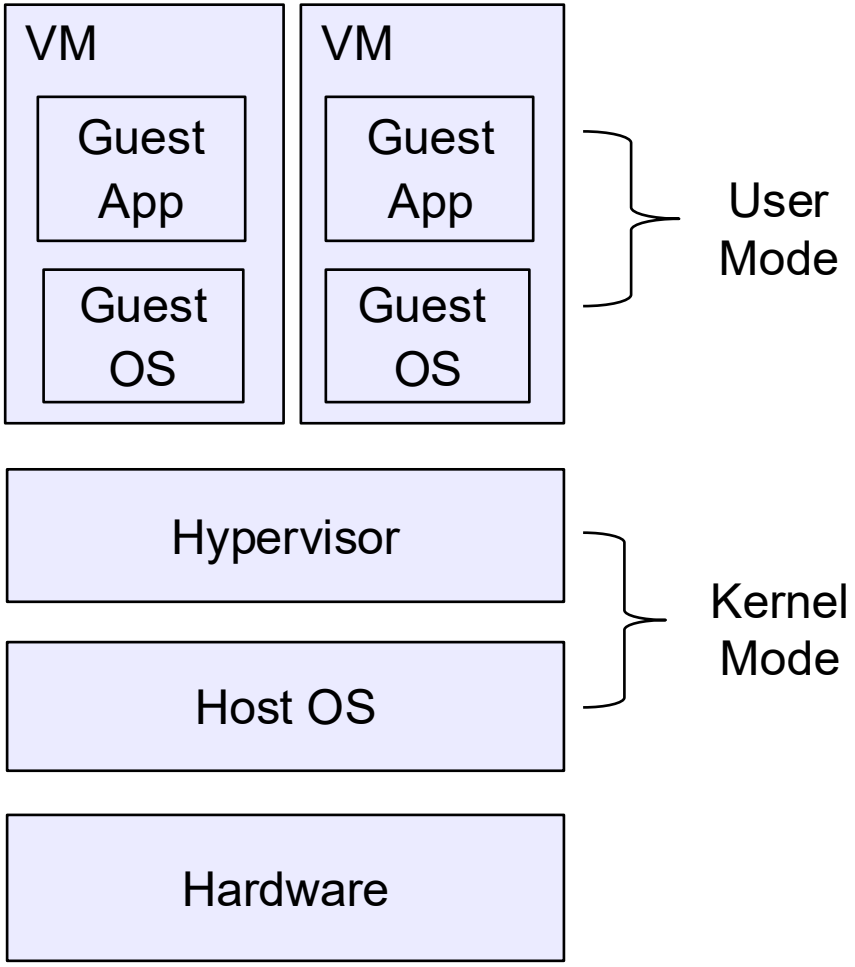
Standard Kernel Structure



Type-1 Hypervisor

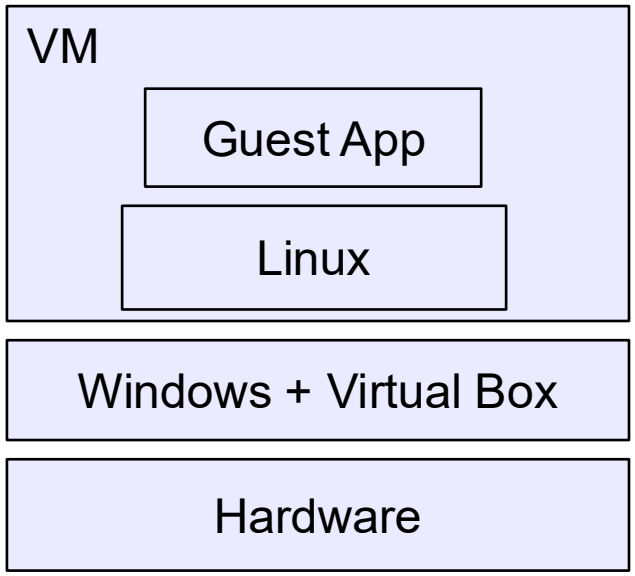


Type-2 Hypervisor



System-Level Virtual Machines

- Owns Virtualized Hardware
- Runs an Operating System of its choice
  - Guest OS runs in user mode*
    - How does Guest OS run privileged instructions?



Virtualized System Calls

- Many solutions!
  - Binary Translation: Hypervisor scans and swaps out dangerous instructions
  - Hardware-Assisted Virtualization: Modern chips (Intel VT-x or AMD SVM) introduce a Non-Root Mode
  - Trap & Emulate: The Guest OS traps into the hypervisor to handle emulating that instruction
  - Paravirtualization: The Guest OS gets “enlightened” and starts using Hypercalls

- With many solutions come many decisions.

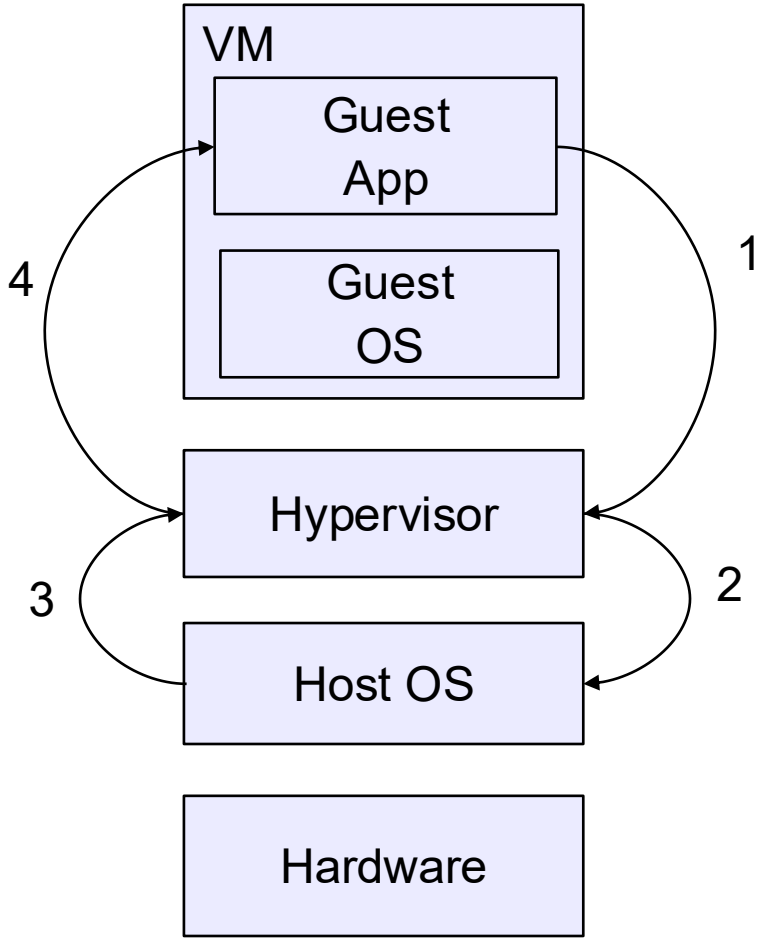
Scenario: Migration!

- A finance firm is migrating its infrastructure to a virtualized environment. They have two distinct workloads:
  - **Workload A:** A proprietary, closed-source risk assessment engine running on an archaic version of Windows. There's no source code available, and the OS is no longer supported by Microsoft.
  - **Workload B:** A high-performance, Linux based trading platform built in-house. The engineering team has full access to the kernel and can modify the OS image as needed.

- What is the optimal approach for each workload?
  - What's possible? What isn't, given the constraints of each workload?

Guess App System Call

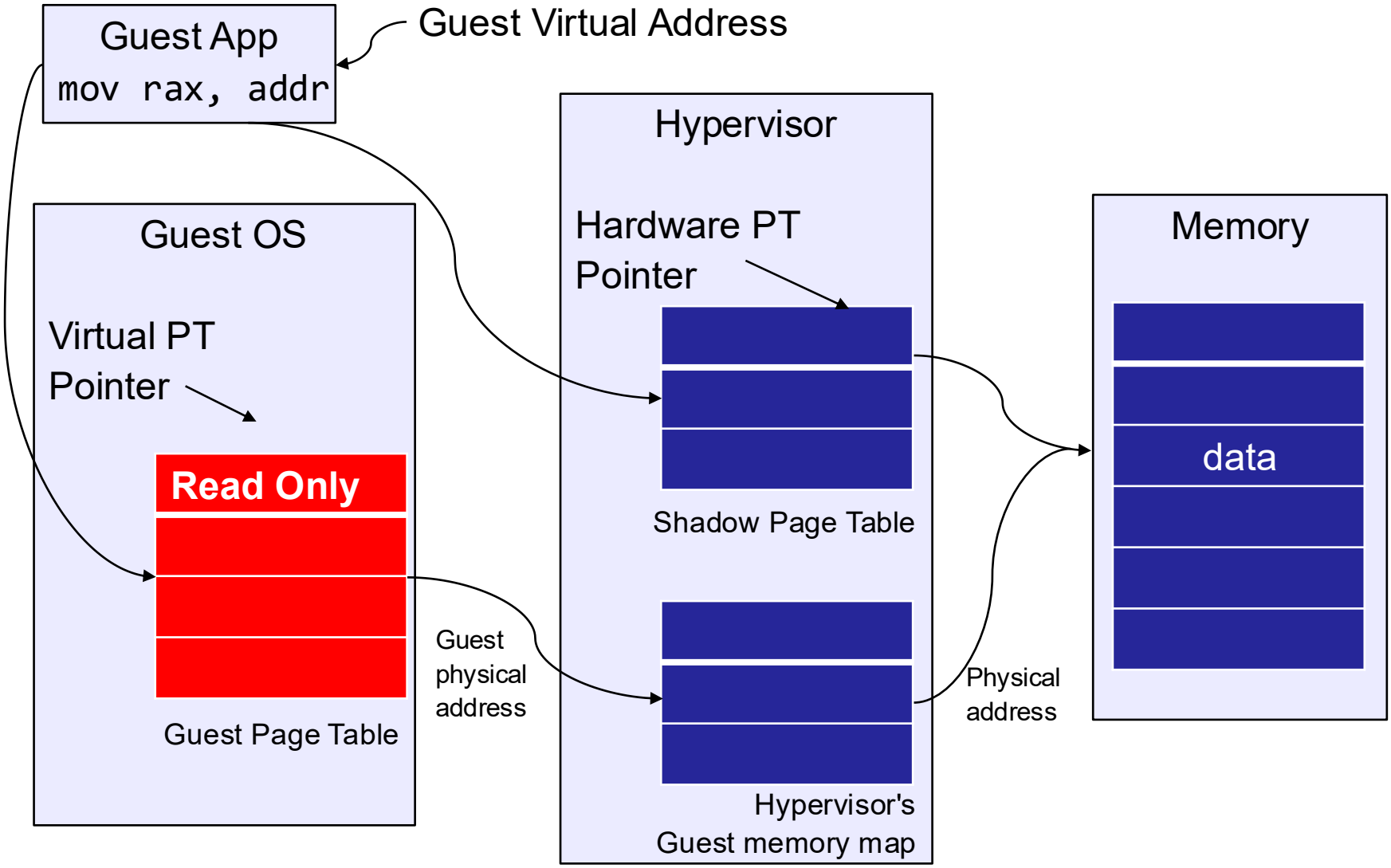
1. The Guest App makes a system call. The hardware (CPU) detects and **traps** to Hyper-V
2. Hyper-V analyzes the instruction and updates its V-Hardware and **traps** to Host OS
3. Host OS **emulates** the system call and returns to the Hyper-V
4. Hypervisor Increments the Guest App's %RIP and calls VM-Entry to restore state and attach emulated result



Memory Virtualization

- Adapting to the MMU
- Shadow Page Tables
- Paravirtualization
- Lots of Optimizations

Shadow Page Table

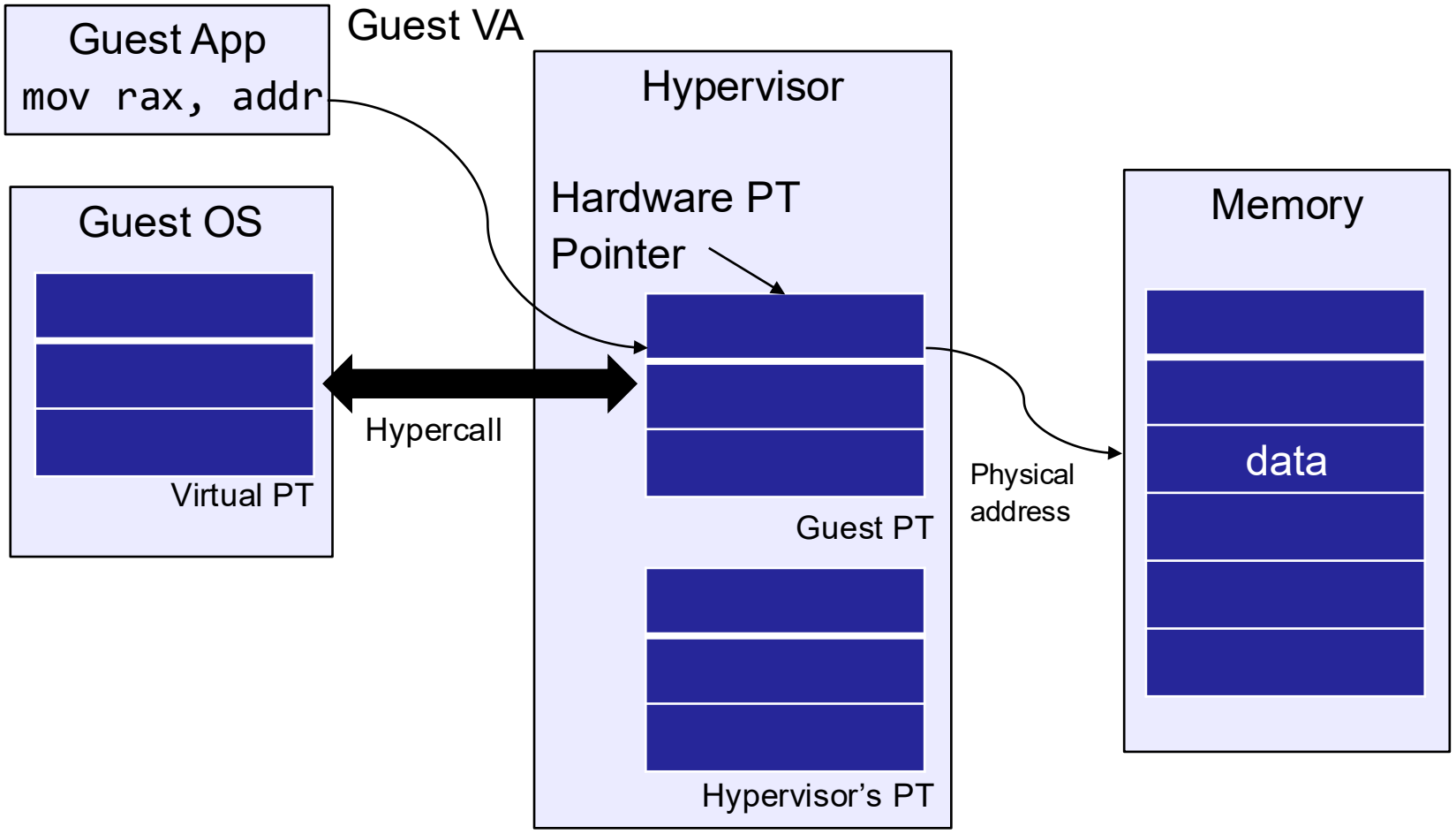


Pros & Cons

- Pros
  - Portable for both the OS and Hardware
  - Very fast once the Shadow PT is populated

- Cons
  - Lots of overhead
    - Trap and Emulate
    - All the Page Tables
  - TLB flushes very often

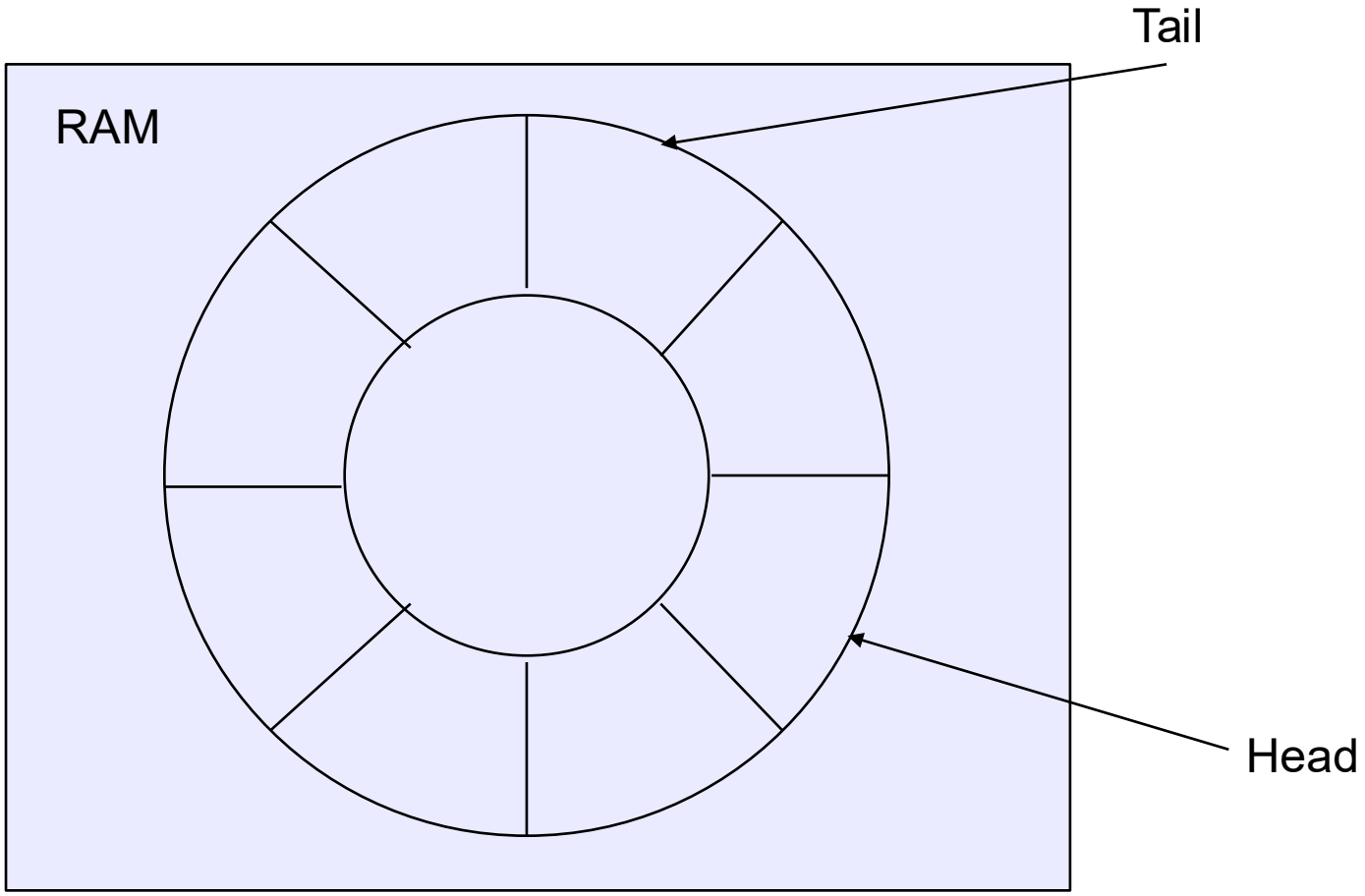
Paravirtualization



I/O Virtualization

- Hyper-V Emulation
  - The Guest OS **Traps** to Hyper-V
  - Hyper-V performs I/O on its behalf
  - Guest OS is interrupt to start its handler
  - Guest OS after fulfilling I/O **Traps** to Hyper-V
  - Two Traps per I/O (very slow)
- VirtIO (Paravirtualization)
  - Use a Virtual Queue to Minimize Traps
  - Guest and Hypervisor agree on a region in Ram for this
  - Requests are batched asynchronously into the hypervisor
  - When done Hyper-V interrupts Guest OS to receive its result

VirtIO's Virtual Queue



Extra: Containers vs Virtual Machines

- Containers
  - Uses the Host OS to spin up the container
  - The Host OS handles isolating the Container which acts as a process with its own name space and resources (cgroups)
- Virtual Machines
  - Uses the Hypervisor to virtualize the hardware
  - Allows the Virtual Machine to run its own Guest OS separate from the Host OS
- Trade-offs
  - Security:
    - VMs are much more secure since they isolate at a higher level
  - Cost:
    - Containers are more scalable and make better use of hardware resources

References

- Operating Systems Principles and Practice 2nd Ed.
- Hardware and Software Support for Virtualization
- Compiler Design: Virtual Machines
- VMScape
  - Blog on breaking the isolation of Virtual Machines to steal information