Hardware Virtualization Principles

What Is Virtualization?

- Virtualization allows multiple isolated operating systems to run on a single physical machine
- Each VM behaves as if it has dedicated hardware
- Enabled by a software layer called a **hypervisor**
- Core goal: isolation + efficient resource sharing

Why Virtualization Matters

- Improves hardware utilization through consolidation
- Enables workload portability across machines and data centers
- Provides fault isolation between workloads
- Foundation for modern cloud computing platforms

The Role of the Hypervisor

- Hypervisor mediates access to CPU, memory, and devices
- Presents virtual hardware abstractions to guest operating systems
- Enforces isolation boundaries between VMs
- Schedules and multiplexes physical resources

Hypervisors vs. Traditional Operating Systems

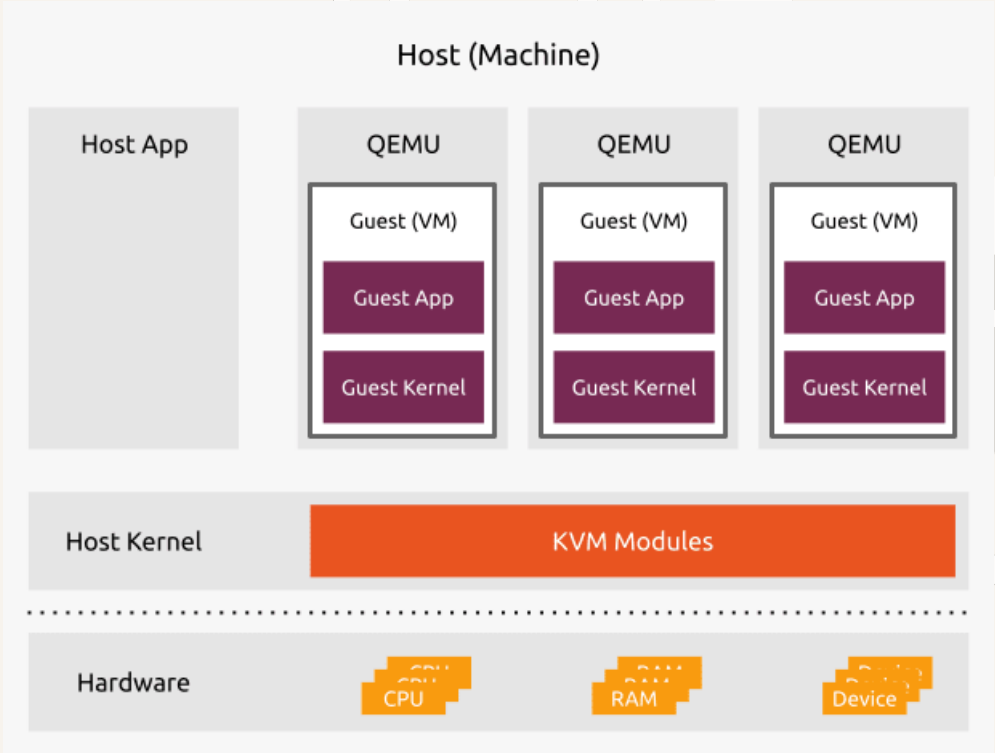
- **Traditional OS:** manages hardware on behalf of user processes
- **Hypervisor:** manages hardware on behalf of entire operating systems
- OS assumes it is the sole owner of the machine
- Hypervisor must safely multiplex ownership across multiple guests
- Design priority shifts from *fair process scheduling* to *strong isolation*
- Minimalism: many hypervisors expose fewer services than general-purpose OSes
- Security model treats guest OSes as untrusted workloads

Common Hypervisor Implementations

- **Closed / Commercial Hypervisors**
  - **VMware ESXi**
    - Widely used in enterprise and private clouds
    - Mature tooling, strong ecosystem, proprietary licensing
  - **Microsoft Hyper-V**
    - Integrated with Windows Server and Azure
  - Common in Microsoft-centric enterprise environments
- **Open-Source Hypervisors**
  - KVM (Kernel-based Virtual Machine)
    - Built directly into the Linux kernel
    - Foundation for many public clouds (via Linux-based stacks)
  - **Xen**
    - Early cloud hypervisor with strong isolation model
    - Historically used by large providers; still relevant in specialized contexts

Example: KVM

- QEMU is an emulator.
- Amazingly, this setup can still lead to near-native performance.



https://ubuntu.com/blog/kvm-hyphervisor

Hardware Requirements for Virtualization

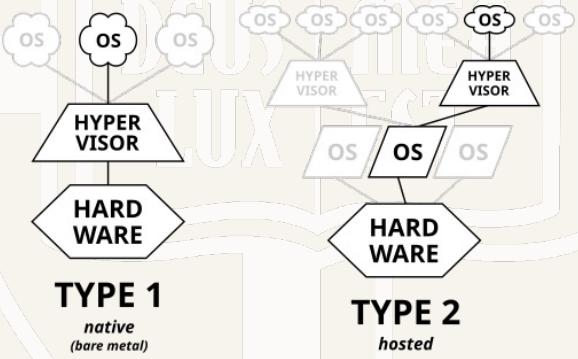
- CPU must explicitly support **guest vs. host execution modes**
- **Intel VT-x (VMX)** and **AMD-V (SVM)** add hardware-managed virtualization states
- Sensitive and privileged instructions must **trap automatically** when executed by a guest
- Hardware must provide **controlled entry and exit** between guest and hypervisor
- Memory subsystem must support **nested address translation**
  - Intel: **Extended Page Tables (EPT)**
  - AMD: **Nested Page Tables (NPT / RVI)**
- Interrupts, timers, and exceptions must be **virtualizable** without guest awareness

Privilege Rings and CPU Modes

- CPUs provide multiple privilege levels (rings)
- Traditional OS runs kernel in highest privilege level
- Virtualization introduces an additional control layer
- Hardware support (e.g., VMX/SVM) enables safe trapping

Where Does the Hypervisor Sit?

- Fundamental design choice: **does the hypervisor run on bare hardware or on top of an OS?**
- Determines how hardware access, drivers, and scheduling are handled
- Affects performance, security boundaries, and system complexity
- Leads to two architectures:
  - **Bare-metal hypervisors** (direct hardware control)
  - **Hosted hypervisors** (hypervisor as an application)



https://en.wikipedia.org/wiki/Hypervisor

Type 1 (Bare-Metal) Hypervisors

- Run directly on physical hardware
- No host operating system underneath
- Examples: enterprise and cloud hypervisors
- Advantages: performance, security, scalability

Type 2 (Hosted) Hypervisors

- Run on top of a general-purpose host OS
- Host OS manages hardware and device drivers
- Common in desktop and development use
- Easier to install, higher overhead

Comparing Type 1 vs. Type 2

- Type 1: production cloud, strong isolation, lower latency
- Type 2: development, testing, convenience-focused
- Difference is architectural, not just performance
- Cloud platforms almost exclusively use Type 1

What Hardware Resources Must Be Virtualized?

- **CPU:** execution state, privilege levels, and scheduling
- **Memory:** address spaces, isolation, and translation
- **Storage:** disks presented as virtual block devices
- **Networking:** virtual NICs, MAC/IP isolation, traffic control
- **Devices:** timers, interrupts, and I/O peripherals
- Hypervisor must virtualize **each resource independently** while preserving the illusion of a full machine
- Weakness in any layer can compromise isolation or performance

Memory Virtualization: The Problem

- Guest OS assumes direct control over physical memory
- Multiple VMs must coexist without interference
- Hypervisor must translate guest memory addresses
- Requires an additional level of indirection

Page Tables: Why They Matter for Virtualization

- **Page tables** map virtual addresses used by software to physical memory locations
- Modern OSes already rely on page tables for **process isolation** and protection
- Virtualization adds an extra layer:
  - Guest OS believes it manages “physical” memory
  - Hypervisor must map guest-physical memory to real machine memory
- This creates a **two-level translation problem**
- Efficient memory virtualization depends on hardware-assisted paging support
- Page tables are the foundation that makes **memory isolation** possible across VMs

Shadow Page Tables and EPT

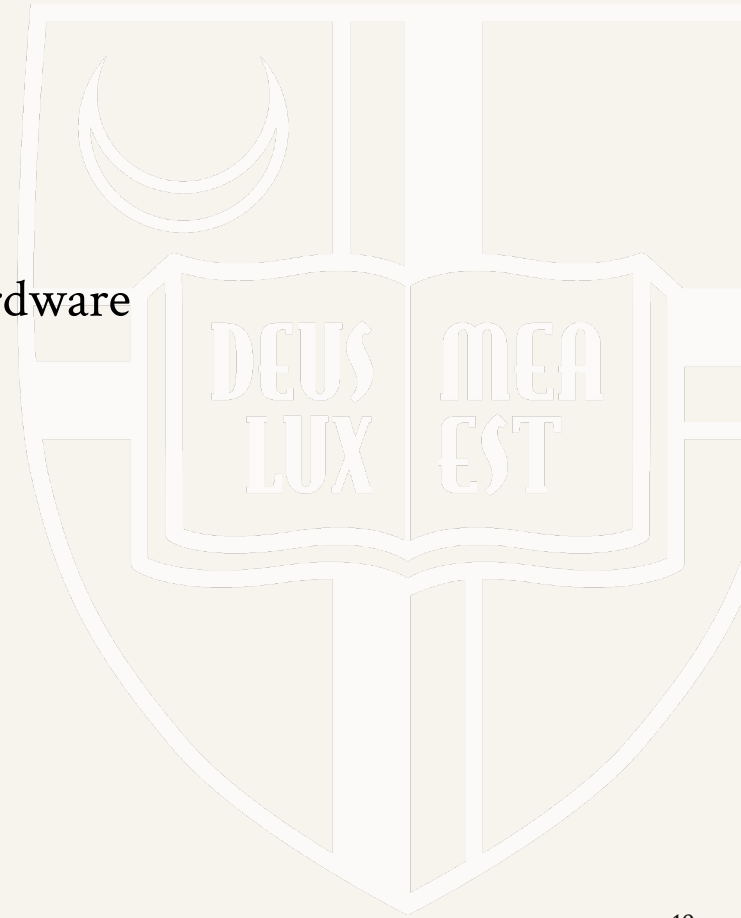
- Early approach: hypervisor-maintained shadow page tables
- High overhead due to frequent updates
- Modern CPUs support nested paging (EPT/NPT)
- Hardware-assisted memory translation reduces overhead

Device Virtualization and I/O

- Physical devices cannot be safely shared directly
- Hypervisor exposes virtual devices to guests
- I/O requests must be mediated or forwarded
- Often a major source of virtualization overhead

Paravirtualized I/O

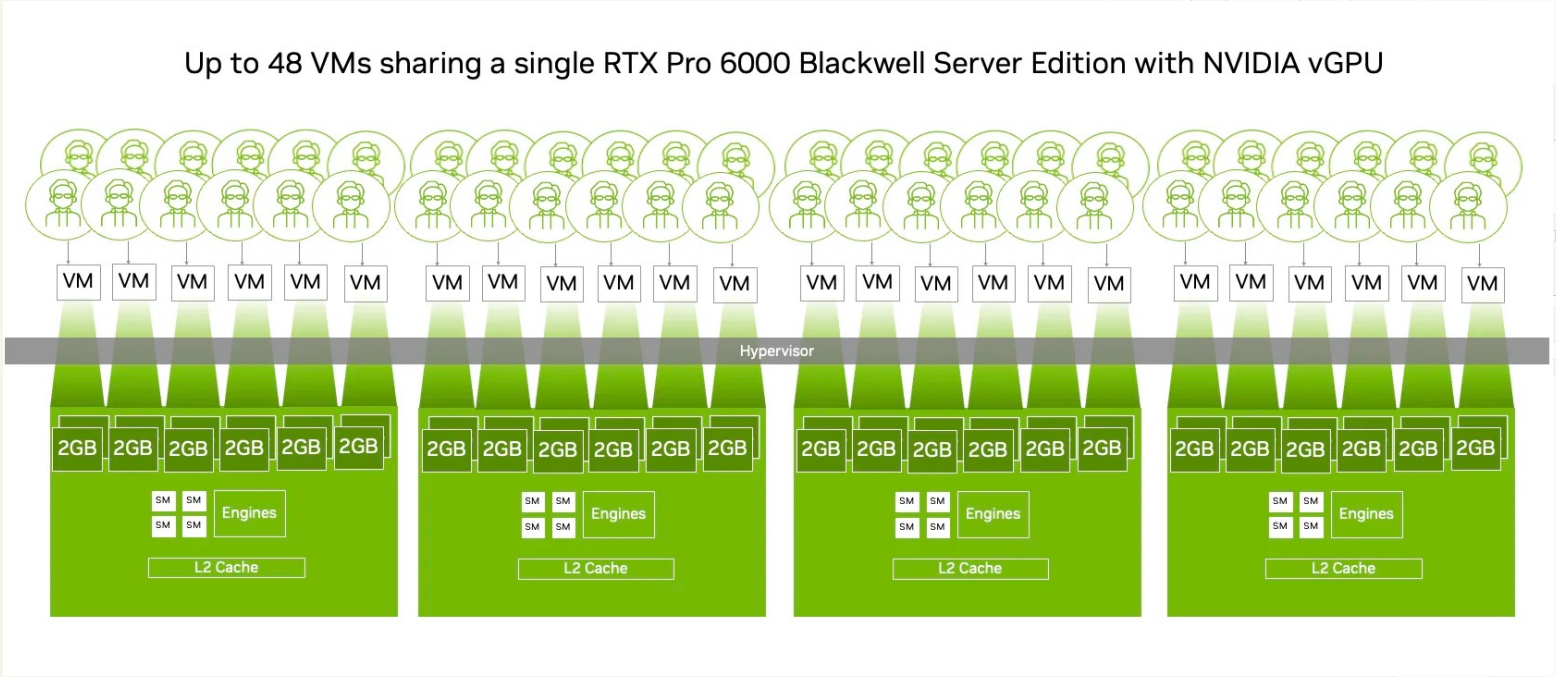
- Guest OS cooperates with the hypervisor
- Uses specialized drivers instead of emulated hardware
- Reduces context switches and emulation cost
- Common for network and storage devices



GPU Virtualization

- GPUs are **stateful, high-throughput devices**, making them harder to virtualize than CPUs
- Hypervisor must control **memory access, command queues, and scheduling** on the GPU
- Common approaches:
  - **Device passthrough:** a VM gets exclusive access to a physical GPU
  - **Mediated / shared GPUs:** hardware and drivers partition GPU resources across VMs
- Modern platforms rely on vendor support:
  - **NVIDIA**: vGPU, MIG (hardware-level partitioning)
  - **AMD**: SR-IOV–based GPU virtualization
- Trade-off space:
  - Passthrough -> near-native performance, weaker consolidation
  - Shared GPUs -> higher utilization, stronger isolation requirements
- GPU virtualization is critical for **AI, ML, and graphics workloads** in the cloud

Nvidia vGPU



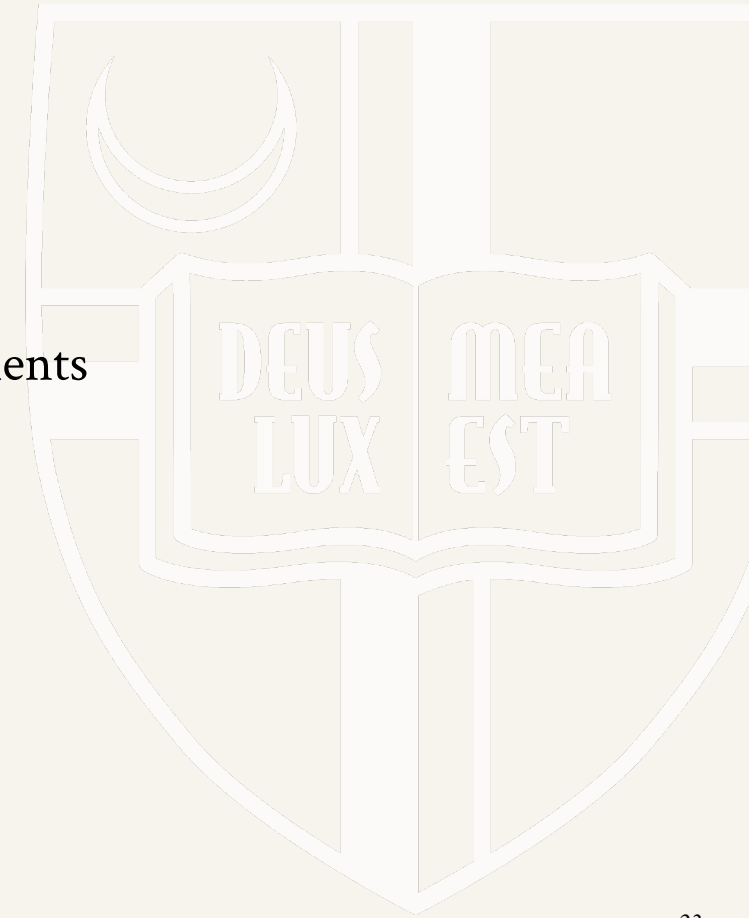
https://developer.nvidia.com/blog/nvidia-vgpu-19-0-enables-graphics-and-ai-virtualization-on-nvidia-blackwell-gpus

Isolation Guarantees in Virtual Machines

- CPU: time-sliced execution with enforced boundaries
- Memory: address space separation via hardware paging
- Devices: controlled access through hypervisor mediation
- Stronger isolation than process-level models

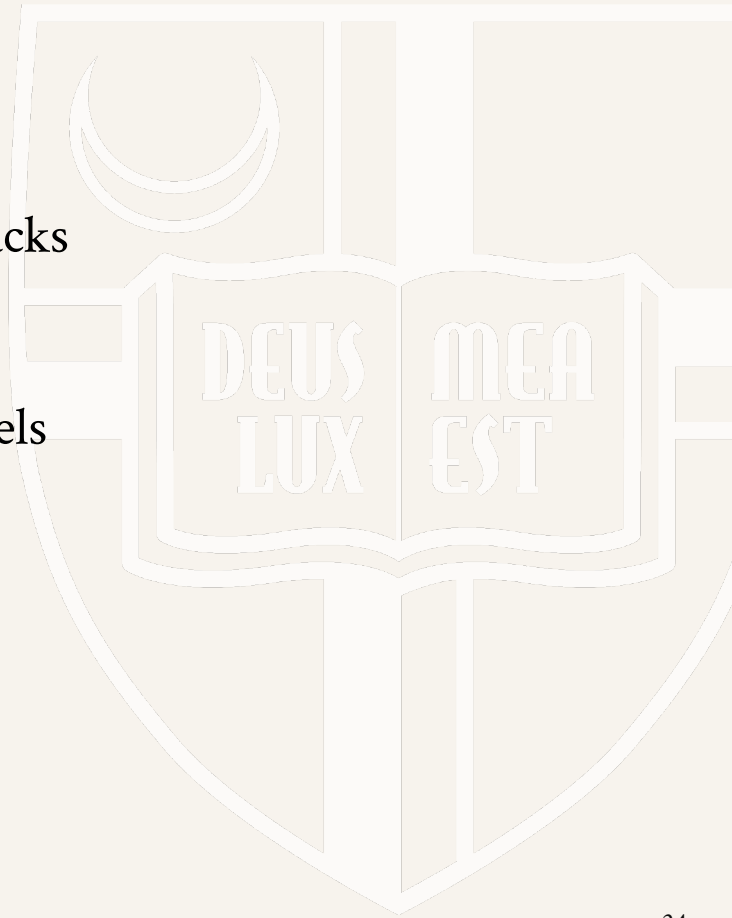
Virtualization and Consolidation

- Multiple workloads share one physical host
- Improves cost efficiency and energy utilization
- Enables elastic provisioning in cloud environments
- Supports overcommitment with managed risk



Virtualization as a Cloud Enabler

- VMs are portable across identical hypervisor stacks
- Failure of one VM does not affect others
- Enables rapid provisioning and teardown
- Sets the stage for containers and serverless models



Preview: Beyond Basic Virtualization

- VM management and deployment workflows
- Performance and isolation trade-offs
- Comparison with containers and other models
- Virtualization as one layer in a broader compute stack