UNIVERSIDADE
BEIRA INTERIOR

Virtualization of CPU, Memory and I/O Devices

Tecnologias de Virtualização e Centros de Dados
Mestrado em Engenharia Informática

Mário M. Freire
Departamento de Informática

Ano Letivo 2023/2024

Credits

- These slides are based on the following book:
- Distributed and Cloud Computing: From Parallel Processing to the Internet of Things, Kai Hwang, Jack Dongarra, Geoffrey C. Fox (Authors), Morgan Kaufmann, 1st edition, 2011, ISBN-13: 978-0123858801, 672 pages.

Agenda

- Hardware Support for Virtualization
- CPU Virtualization
- Memory Virtualization
- I/O Virtualization

Hardware Support for Virtualization

- To support virtualization, processors such as the x86 employ a **special running mode and instructions**, known as **hardware-assisted virtualization**.
- The VMM and guest OS run in different modes and **all sensitive instructions** of the guest OS and its applications are **trapped in the VMM**.
- To save processor states, **mode switching** is completed by **hardware**. For the x86 architecture, **Intel and AMD have proprietary** technologies for **hardware-assisted virtualization**.

Hardware Support for Virtualization

- **Modern operating systems** and processors permit **multiple processes** to run **simultaneously**.
- **Without protection mechanism** in a processor, all **instructions from different processes** will access the hardware **directly** and cause a system crash.
- All processors have at least **two modes**, **user mode** and **supervisor mode**, to ensure controlled access of critical hardware.
- Instructions running in **supervisor mode** are called **privileged instructions**.
- **Other instructions** are **unprivileged instructions**.

Hardware Support for Virtualization

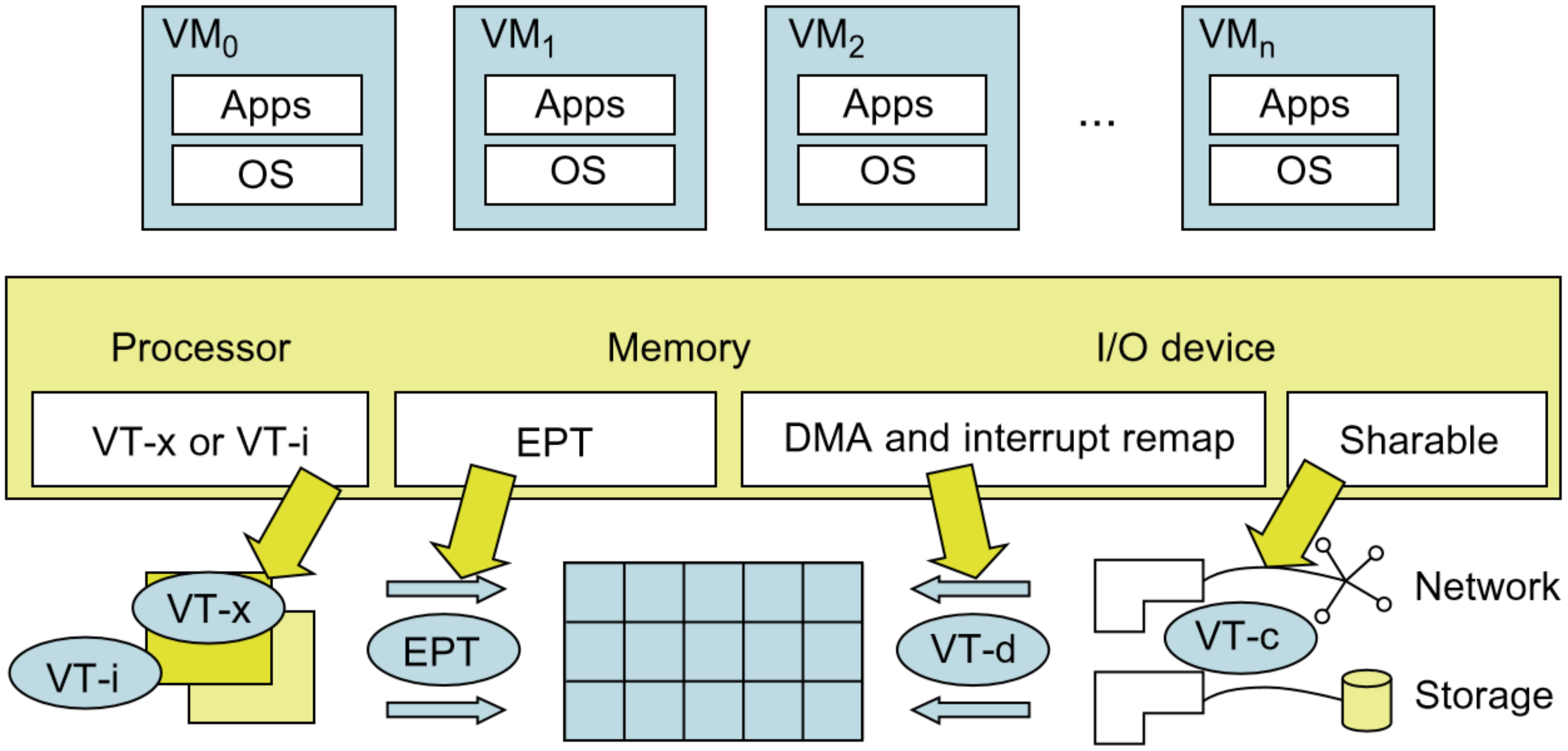
- Software-based virtualization (late 1990s)
- Hardware-assisted virtualization (Started ~2006)
  - **CPU**: Virtual 8086 mode, AMD virtualization (AMD-V), Intel virtualization (VT-x, VT-i), VIA virtualization (VIA VT), Interrupt virtualization (AMD AVIC and Intel APICv)
  - **GPU**: Graphics Virtualization Technology (Intel GVT-d, GVT-g and GVT-s)
  - **Chipset**
    - I/O Memory Management Unit (MMU) virtualization (AMD-Vi and Intel VT-d)
    - Network virtualization (VT-c)
    - PCI-SIG Single Root I/O Virtualization (SR-IOV)

Hardware Support for Virtualization

- One or more guest OS can run on top of the hypervisor.
- **KVM** (Kernel-based Virtual Machine) is a Linux kernel virtualization infrastructure. KVM can support **hardware-assisted virtualization** and **paravirtualization** by using the Intel VT-x or AMD-v and VirtIO framework, respectively.
- **VirtIO** framework includes a **paravirtual Ethernet card**, a **disk I/O controller**, a balloon device for adjusting guest **memory usage**, and a **VGA graphics** interface using VMware drivers.
- Xen is a hypervisor for use in IA-32, x86-64, Itanium, and PowerPC 970 hosts.

Hardware Support for Virtualization

Intel hardware support for virtualization of processor, memory, and I/O devices.



CPU Virtualization

- **Unprivileged instructions** of VMs **run directly** on the host machine for higher efficiency.
- **Critical instructions** are divided into three categories: privileged **instructions**, **control-sensitive instructions**, and **behavior-sensitive instructions**.
- **Privileged instructions** execute in **a privileged mode** and will be **trapped** if executed **outside this mode**.
- **Control-sensitive instructions** attempt to change the **configuration** of used resources.
- **Behavior-sensitive instructions** have **different behaviors** depending on the **configuration of resources**, including the load and store operations over the virtual memory.

CPU Virtualization

- ***CPU Virtualization***
- A CPU architecture is **virtualizable** if it supports the ability to run the **VM's privileged and unprivileged instructions in the CPU's user mode** while the **VMM runs in supervisor mode**.
- When the **privileged instructions** including **control- and behavior-sensitive instructions** of a VM are executed, they are **trapped in the VMM**.
- The VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system.

CPU Virtualization

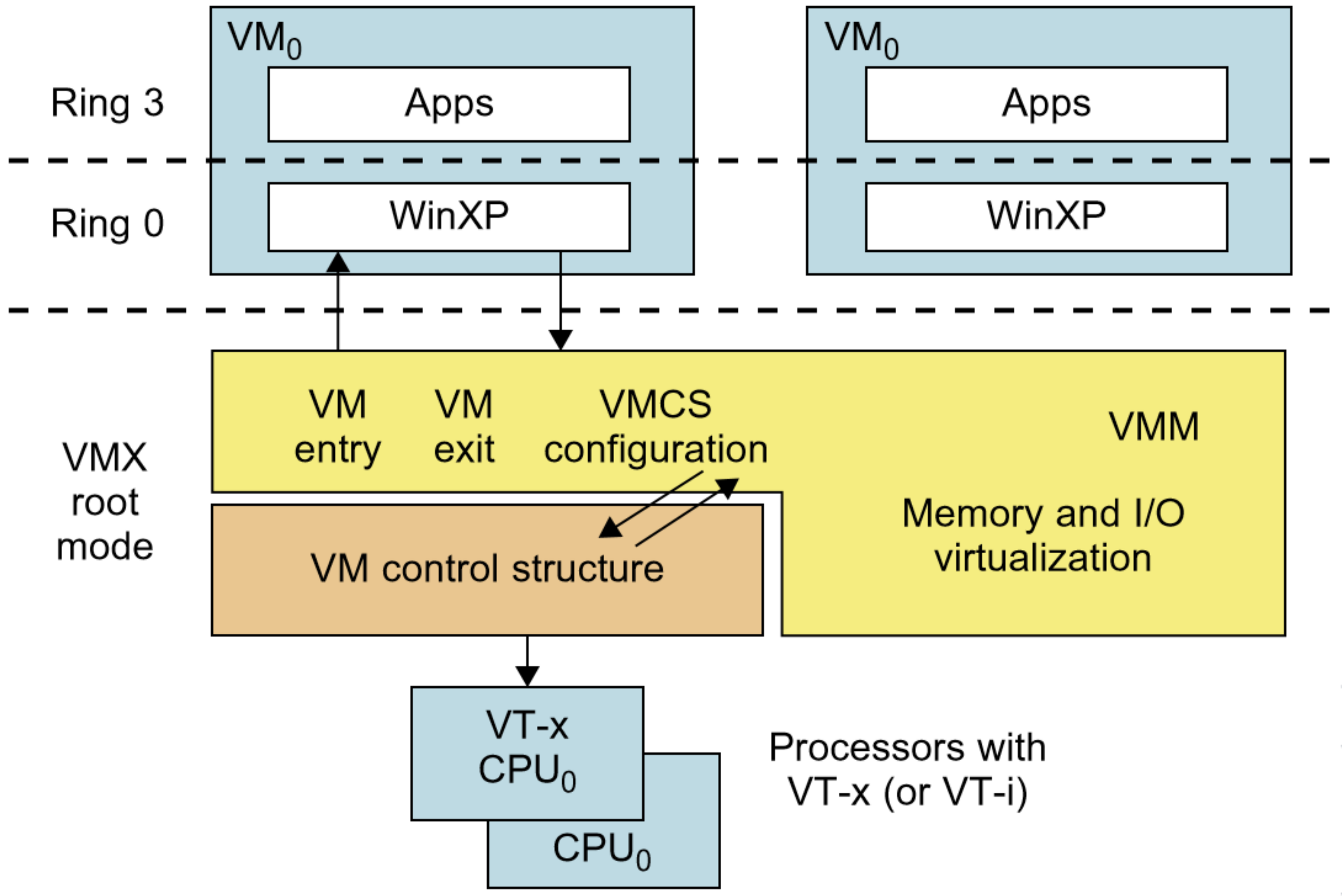
- **Not all CPU** architectures are **virtualizable**.
- **RISC CPU** architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions.
- On the contrary, **x86 CPU architectures are not** primarily **designed to support virtualization**.
- This is because about 10 sensitive instructions, such as SGDT (Store Global Descriptor Table Register) and SMSW (Store Machine Status Word), are not privileged instructions. When these instructions execute in virtualization, they cannot be trapped in the VMM.

CPU Virtualization

- ***Hardware-Assisted CPU Virtualization***
- This technique attempts to **simplify virtualization** because **full or paravirtualization is complicated**.
- **Intel and AMD** add an additional mode called **privilege mode level** (AKA Ring -1) to x86 processors.
- Operating systems can still run at Ring 0 and the hypervisor can run at Ring -1.
- All the **privileged** and **sensitive instructions** are **trapped** in the **hypervisor** automatically.
- This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification.

CPU Virtualization

Intel Hardware-Assisted CPU Virtualization.



Memory Virtualization

- **Virtual memory virtualization** is similar to the **virtual memory support** provided by modern operating systems.
- A **traditional operating system** maintains **mappings of virtual memory** to **machine memory** using **page tables**, which is a one-stage mapping from virtual memory to machine memory.
- All modern x86 CPUs include a **memory management unit** (MMU) and a **translation lookaside buffer** (TLB) to **optimize virtual memory** performance.
- However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs.

Memory Virtualization

- That means a **two-stage mapping process** should be maintained by the guest OS and the VMM, respectively: **virtual memory to physical memory** and **physical memory to machine memory**.
- **MMU virtualization should be supported**, which is transparent to the guest OS. The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs. But the guest OS cannot directly access the actual machine memory.
- The VMM is responsible for mapping the guest physical memory to the actual machine memory.

Memory Virtualization

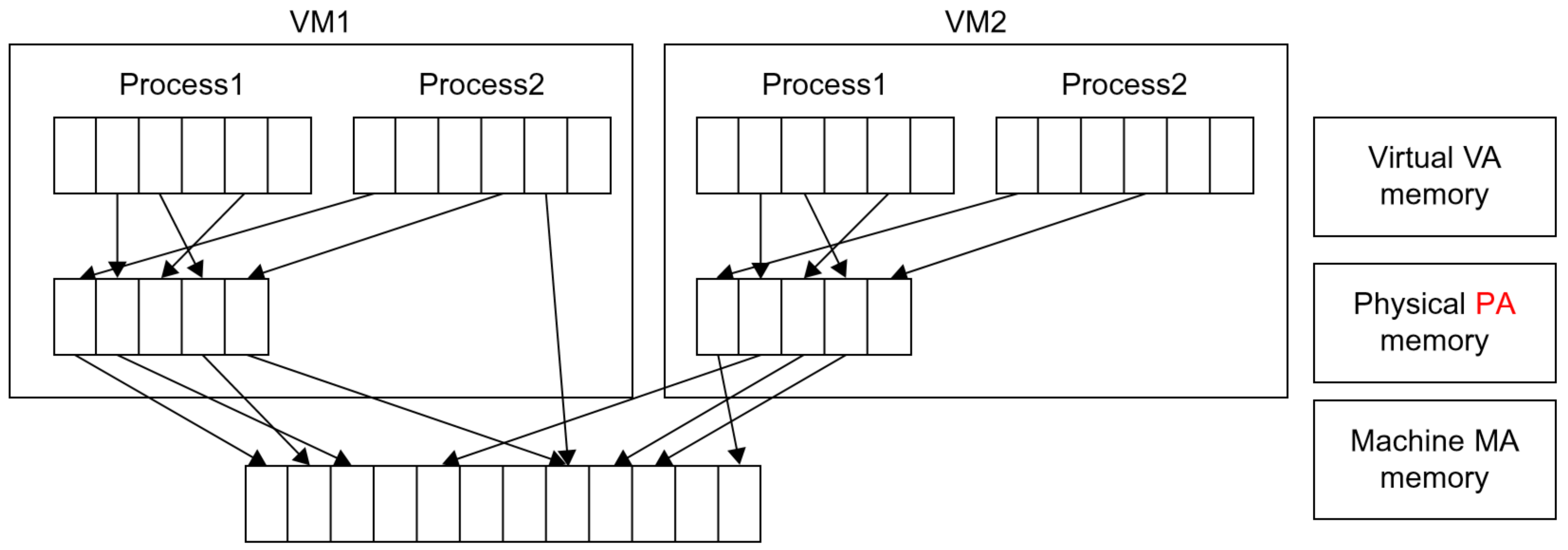
- Since each page table of the guest OSes has a separate page table in the VMM corresponding to it, the **VMM page table** is called the **shadow page table**.
- Nested page tables add another layer of indirection to virtual memory. The MMU already handles virtual-to-physical translations as defined by the OS.
- **The physical memory addresses** are translated to **machine addresses** using another set of page tables defined by the hypervisor.
- Since modern operating systems **maintain a set of page tables** for **every process**, the **shadow page tables** will **get flooded**. Consequently, the **performance overhead** and **cost of memory** will be **very high**.

Memory Virtualization

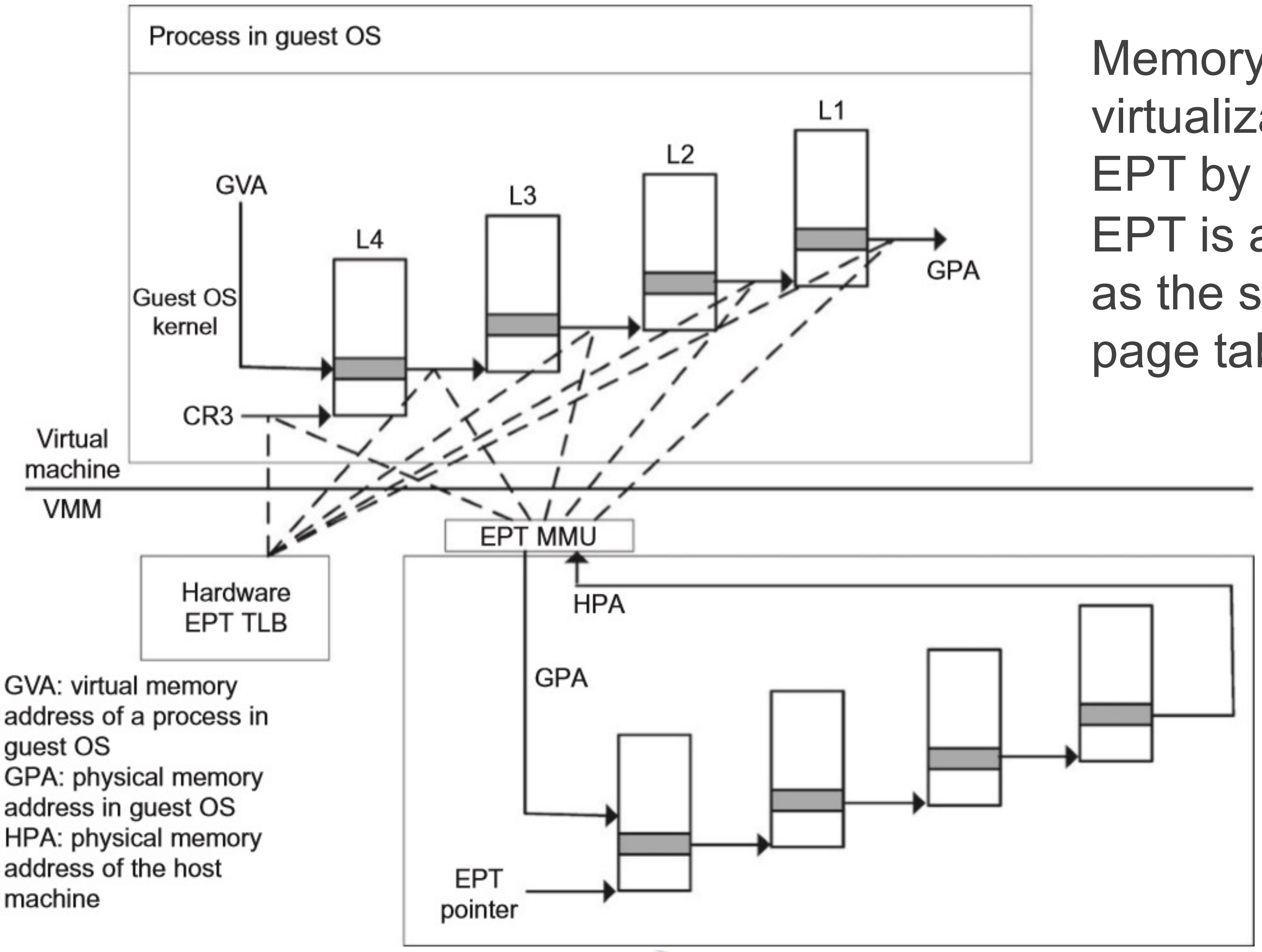
- **VMware** uses **shadow page tables** to perform v**irtual-memory-to-machine-memory address** translation.
- Processors use **TLB hardware** to map the **virtual memory directly** to the **machine memory** to avoid the two levels of translation on every access.
- When the guest OS changes the **virtual memory** to a **physical memory mapping**, the VMM updates the **shadow page tables** to enable a **direct lookup**.
- The AMD Barcelona processor has featured hardware-assisted memory virtualization since 2007. It provides hardware assistance to the two-stage address translation in a virtual execution environment by using a technology called nested paging.

Memory Virtualization

Two-level memory mapping procedure.



Memory Virtualization



GVA: virtual memory address of a process in guest OS
GPA: physical memory address in guest OS
HPA: physical memory address of the host machine

Memory virtualization using EPT by Intel (the EPT is also known as the shadow page table.

Input/Output Virtualization

- **I/O virtualization** involves managing the **routing** of **I/O requests** between **virtual devices** and the **shared physical hardware**.

- There are **three ways** to implement **I/O virtualization**:
  - **Full device emulation**;
  - **Para-virtualization**;
  - **Direct I/O**.

Input/Output Virtualization

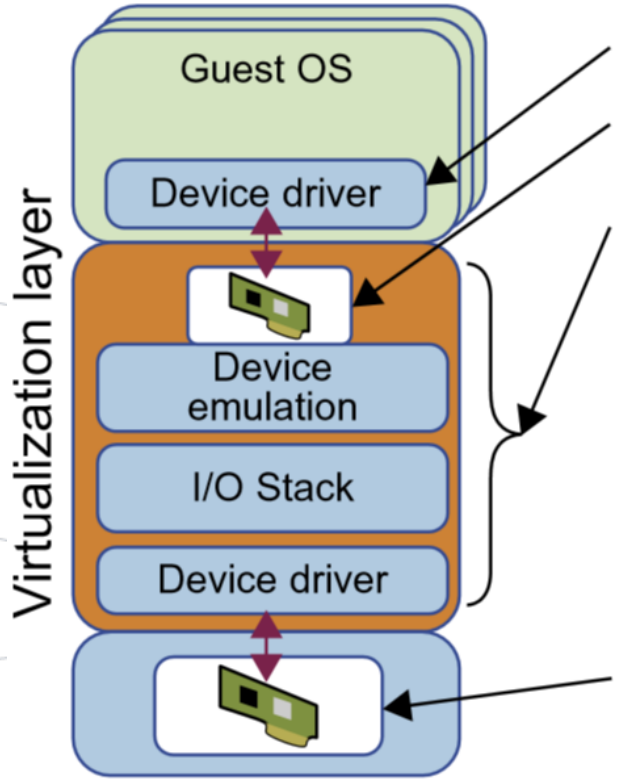
- **Full device emulation** is the first approach for I/O virtualization. Generally, this approach emulates well-known, real-world devices.

- All the **functions** of a device or bus infrastructure, such as **device enumeration**, **identification**, **interrupts**, and **DMA**, are replicated in software. This software is located in the VMM and acts as a **virtual device**.

- The **I/O access requests** of the guest OS are **trapped in the VMM** which interacts with the **I/O devices**.

Input/Output Virtualization

Full device emulation approach: Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use.



- Guest device driver
- Virtual device
- Virtualization layer
  - emulates the virtual device
  - remaps guest and real I/O addresses
  - multiplexes and drives the physical device
  - I/O features. e.g., COW disks
- Real device
  - may be different from virtual device

Input/Output Virtualization

- A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates.
- The **para-virtualization method** of I/O virtualization is typically used in Xen.
- It is also known as the **split driver model** consisting of a **frontend driver** and a **backend driver**.
- The **frontend driver** is running in **Domain U** and the **backend driver** is running in **Domain 0**. They interact with each other via a block of shared memory.
- The **frontend driver** manages the **I/O requests** of the **guest OSes** and the **backend driver** is responsible for managing the **real I/O devices** and **multiplexing the I/O data** of different VMs.
- **Para-I/O-virtualization** achieves better device performance than **full device emulation**, but it comes with a **higher CPU** overhead.

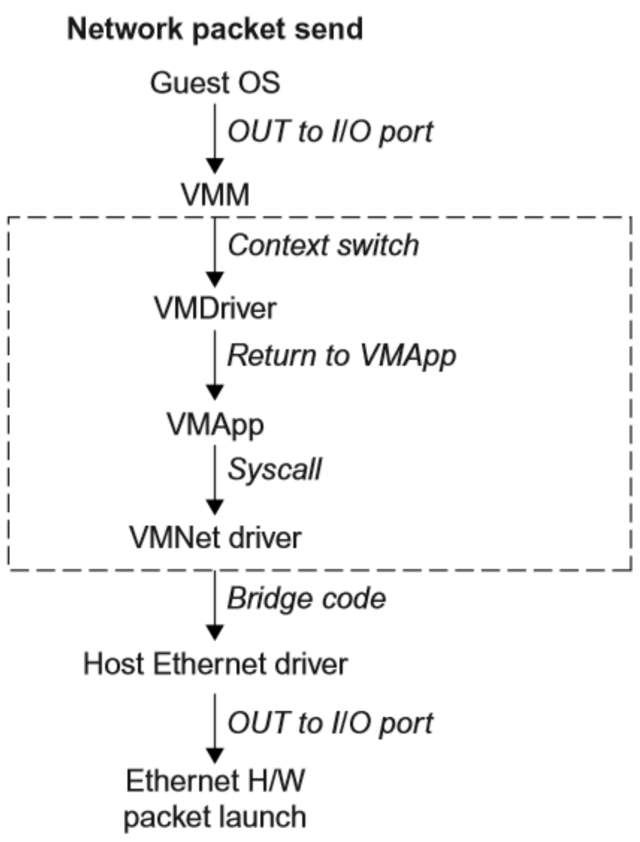
Input/Output Virtualization

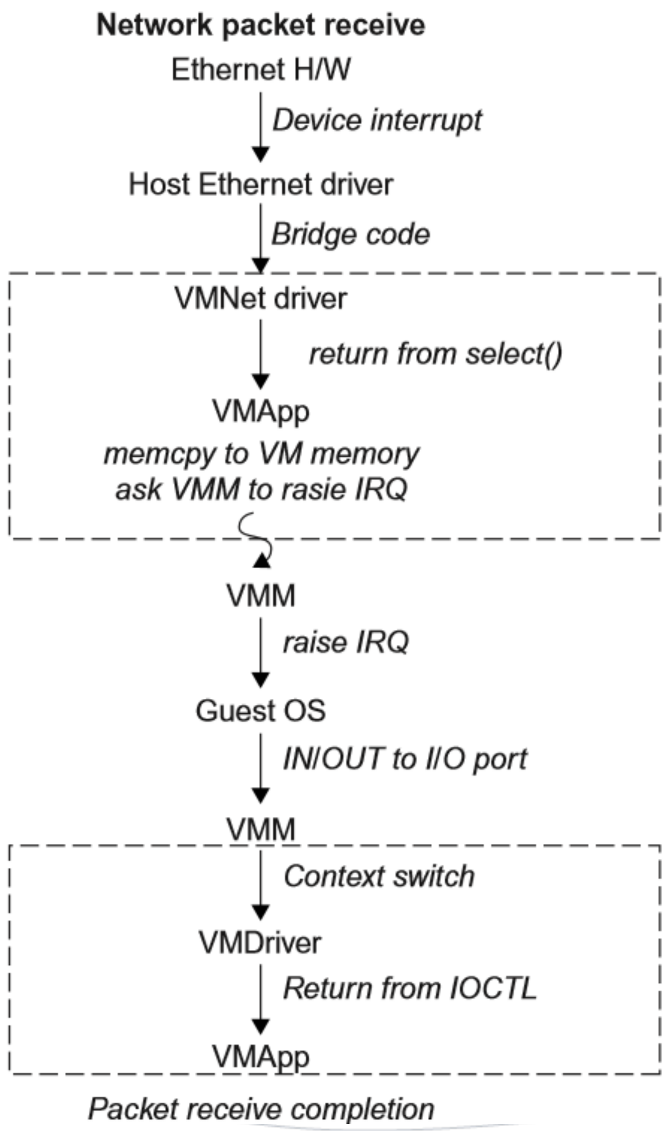
- **Direct I/O virtualization** lets the VM **access devices directly**.
- It can achieve **close-to-native performance** without **high CPU costs**. However, current direct I/O virtualization implementations focus on networking for mainframes.
- When a physical device is reclaimed (required by workload migration) for later reassignment, it may have been set to an arbitrary state (e.g., DMA to some arbitrary memory locations) that can function incorrectly or even crash the whole system.
- Since **software-based I/O virtualization requires** a very high overhead of device emulation**, hardware-assisted I/O virtualization is critical.**
- **Intel VT-d** supports the remapping of I/O DMA transfers and device-generated interrupts. The architecture of VT-d provides the flexibility to support multiple usage models that may run unmodified, special-purpose, or virtualization-aware guest Oses.

Input/Output Virtualization

- Another way to help I/O virtualization is via **self-virtualized I/O** (SV-IO). The **key idea** of SV-IO is to harness the **rich resources** of a **multicore processor**.
- **All tasks** associated with virtualizing an I/O device are **encapsulated** in SV-IO. It provides **virtual devices** and an **associated access API** to VMs and a **management API** to the VMM.
- SV-IO defines **one virtual interface** (VIF) for every kind of **virtualized I/O device**, such as virtual network interfaces, virtual block devices (disk), virtual camera devices, and others
- The guest OS interacts with the VIFs via VIF device drivers. **Each VIF** consists of **two message queues**. One is for **outgoing messages** to the devices and the other is for **incoming messages** from the devices. In addition, each VIF has a unique ID for identifying it in SV-IO.

Input/Output Virtualization





VMware Workstation for I/O Virtualization (full device emulation)

Functional blocks involved in sending and receiving network packets.

Virtualization in Multi-core Processors

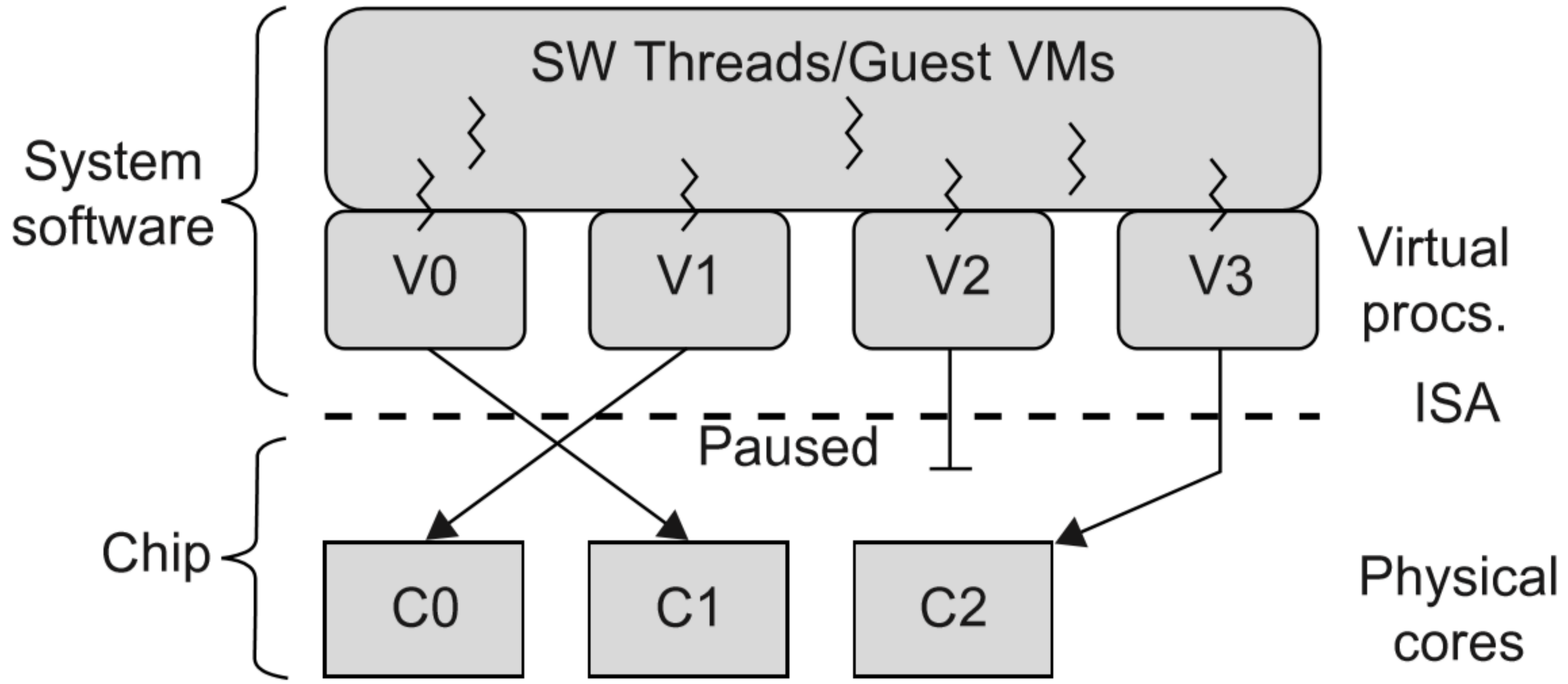
- Multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip.
- **Virtualizing** a **multi-core processor** is relatively more **complicated** than virtualizing a uni-core processor.
- There are mainly **two difficulties for multi-core virtualization**:
  1. **Application programs** must be **parallelized** to use **all cores fully**.
  2. **Software must explicitly assign tasks** to the **cores**, which is a very complex problem.

Virtualization in Multi-core Processors

- **Physical versus Virtual Processor Cores**
- Wells et al. proposed a **multicore virtualization method** to allow hardware designers to get an **abstraction** of the **low-level details** of the **processor cores**.
- This technique **alleviates** the **burden** and **inefficiency** of **managing hardware resources** by **software**. It is located under the ISA and remains unmodified by the operating system or VMM (hypervisor).
- Next figure illustrates the technique of a **software-visible VCPU** moving from one core to another and temporarily suspending execution of a VCPU when there are no appropriate cores on which it can run.

Virtualization in Multi-core Processors

Multicore virtualization method that exposes four VCPUs to the software, when only three cores are actually present.



Virtualization in Multi-core Processors

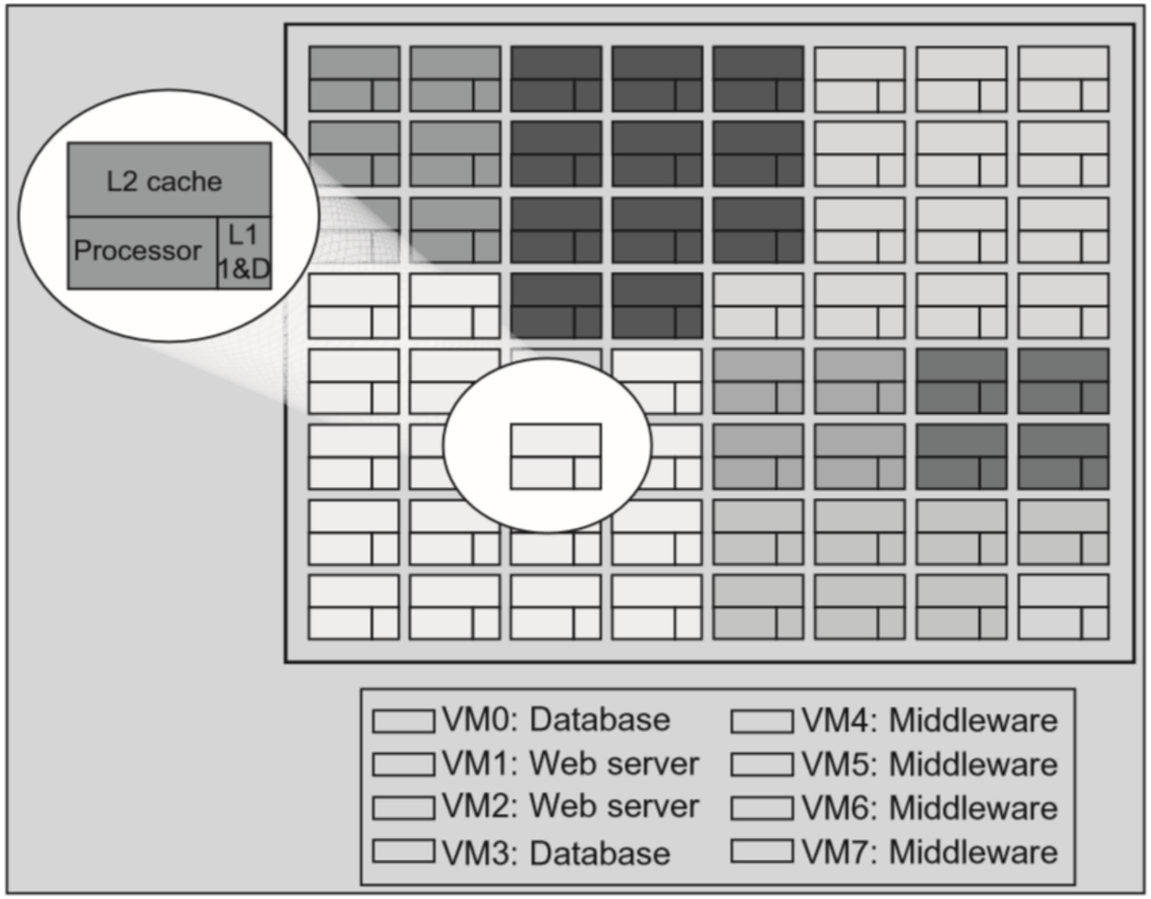
- **Virtual Hierarchy (1)**
- Instead of supporting **time-sharing jobs** on one or a few cores, we can use the abundant cores in a **space-sharing**, where **single-threaded** or **multithreaded jobs** are **simultaneously assigned** to **separate** groups of **cores** for long time intervals.
- This idea was originally suggested by Marty and Hill. To **optimize** for **space-shared workloads**, they propose using **virtual hierarchies** to overlay a **coherence and caching hierarchy** onto a physical processor.
- Unlike a fixed physical hierarchy, a **virtual hierarchy** can adapt to **fit** how the **work** is **space shared** for **improved performance** and **performance isolation**.

Virtualization in Multi-core Processors

- **Virtual Hierarchy (2)**
- Today’s many-core CMPs (chip multiprocessors) use a **physical hierarchy** of **two or more cache levels** that statically determine the cache **allocation and mapping**.
- A virtual hierarchy is a cache hierarchy that can adapt to fit the workload or mix of workloads.
- The hierarchy’s **first level** locates **data blocks close** to the **cores** needing them for **faster access**, establishes a **shared-cache domain**, and establishes a **point of coherence** for **faster communication**.
- When a **miss** leaves a tile, it first attempts to **locate** the block (or sharers) within the **first level**. The first level can also provide isolation between independent workloads. A **miss** at the **L1 cache** can **invoke** the **L2 access**.

Virtualization in Multi-core Processors

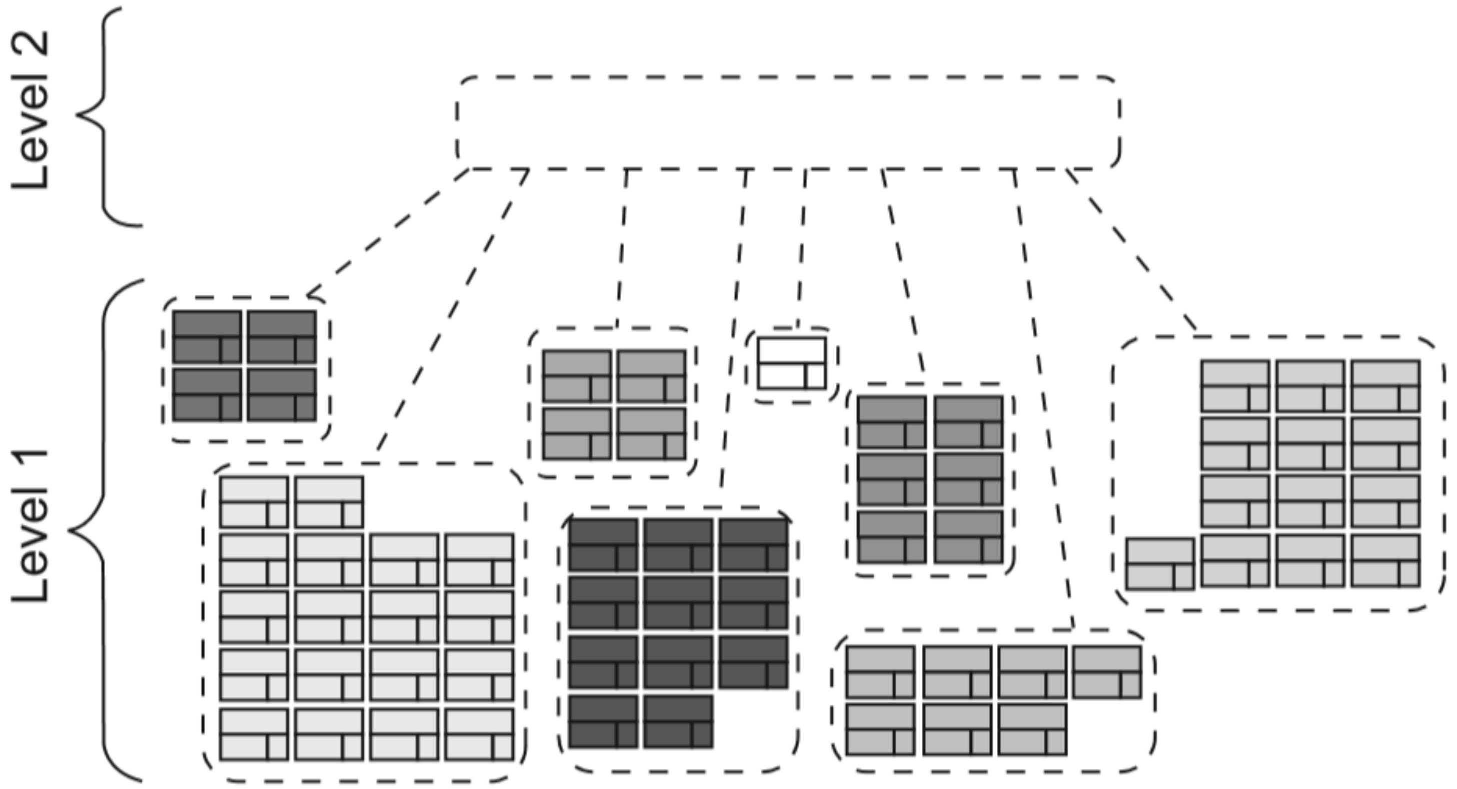
CMP server consolidation by space-sharing of VMs into many cores forming multiple virtual clusters to execute various workloads.



(a) Mapping of VMs into adjacent cores

Virtualization in Multi-core Processors

CMP server consolidation by space-sharing of VMs into many cores forming multiple virtual clusters to execute various workloads.



(b) Multiple virtual clusters assigned to various workloads