UNIVERSIDADE
BEIRA INTERIOR

Virtualization of Memory and I/O Devices (Complementos)

Tecnologias de Virtualização e Centros de Dados

Mestrado em Engenharia Informática

Alexandre Fonte (Prof. Convidado)
Departamento de Informática

Ano Letivo 2023/2024

Credits

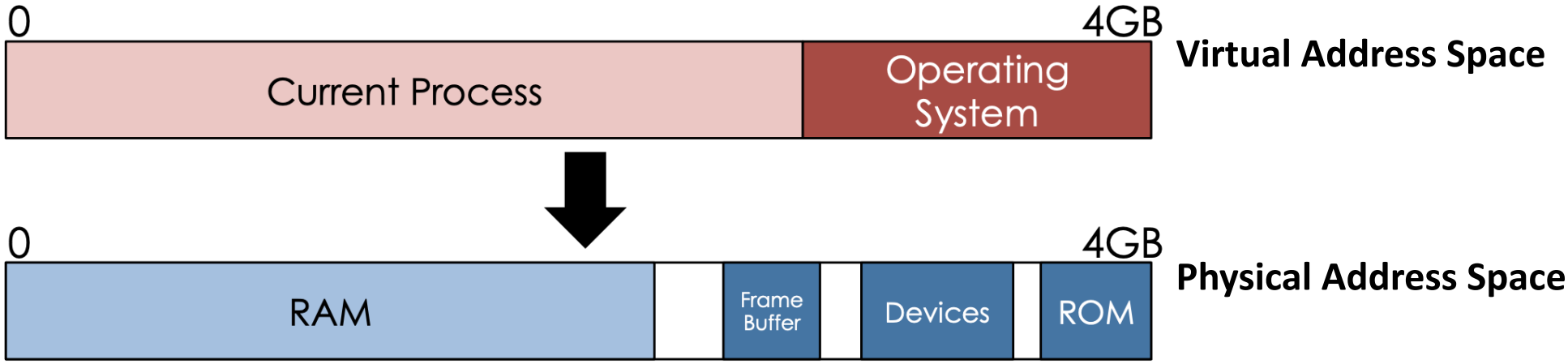
- These slides are based on the following set of slides and books:
- Prof. Mário Freire, Bloco de Slides “Virtualization of CPU, Memory and I/O Devices”, UBI, ano letivo 2020/2021
- Scott Devine, Blocos de Slides “Memory Virtualization”, VMWare
- Distributed and Cloud Computing: From Parallel Processing to the Internet of Things, Kai Hwang, Jack Dongarra, Geoffrey C. Fox (Authors), Morgan Kaufmann, 1st edition, 2011, ISBN-13: 978-0123858801, 672 pages.
- James E. Smith, Ravi Nair, Virtual Machines, Morgan Kaufmann; 1st edition (June 17, 2005), 2005, ISSN 15459888, ISBN 9781558609105, https://doi.org/10.1016/B978-155860910-5/50001-X.

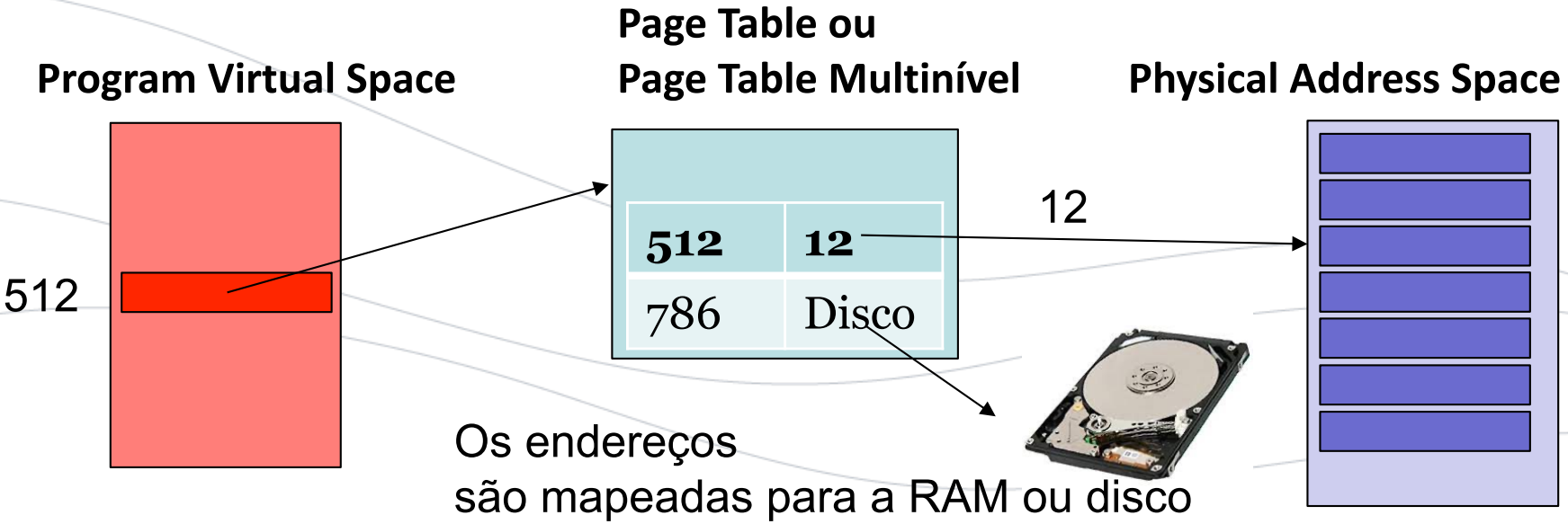
Agenda

- Memory Virtualization
- I/O Virtualization

Espaços de Endereços *Tradicionais*

- Memória Virtual é uma camada de indireção (indirection)

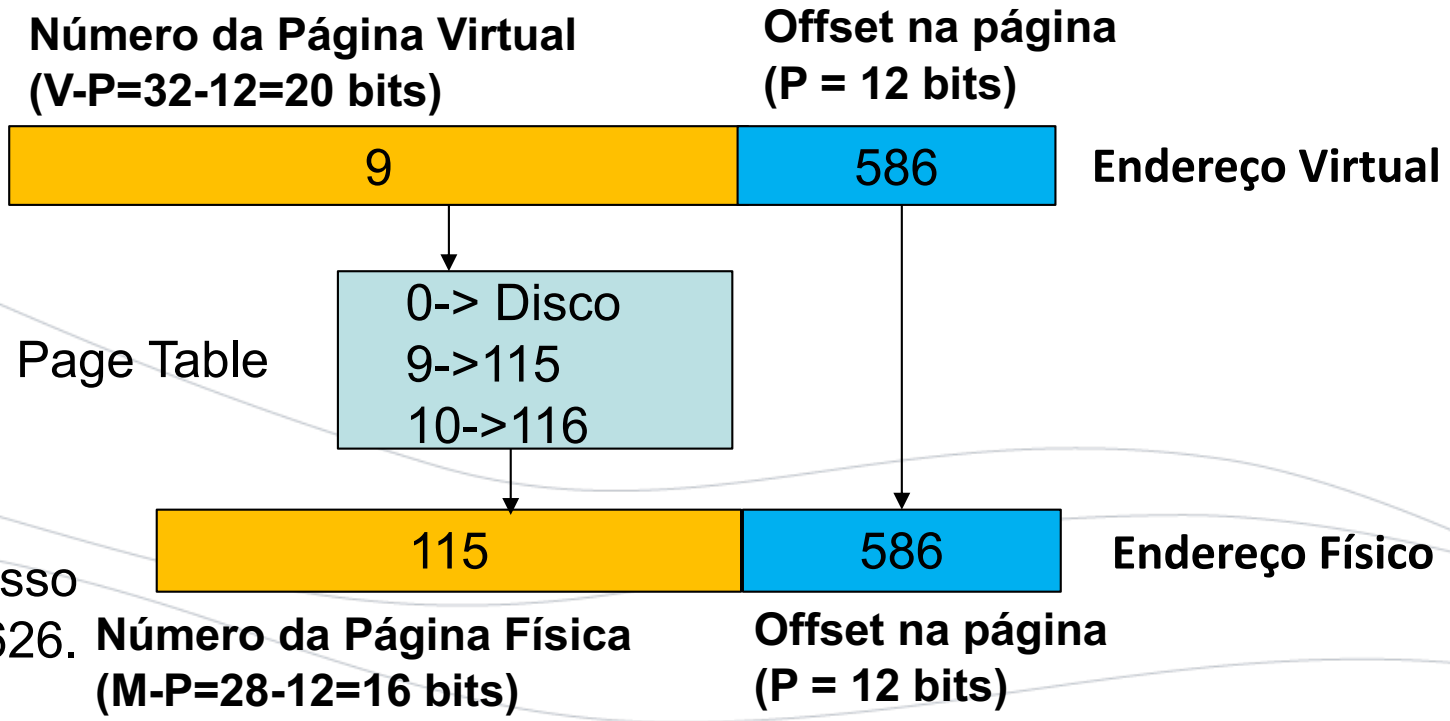




Revisão Exemplo tradução: Endereço Virtual para Endereço Físico

- Considerações:
  - Memória Virtual com endereços de 32 bits => V = 32 bits
  - 256 MB de RAM => 2^28 => M = 28 bits
  - Páginas de 4KB => 2^12 => P = 12bits

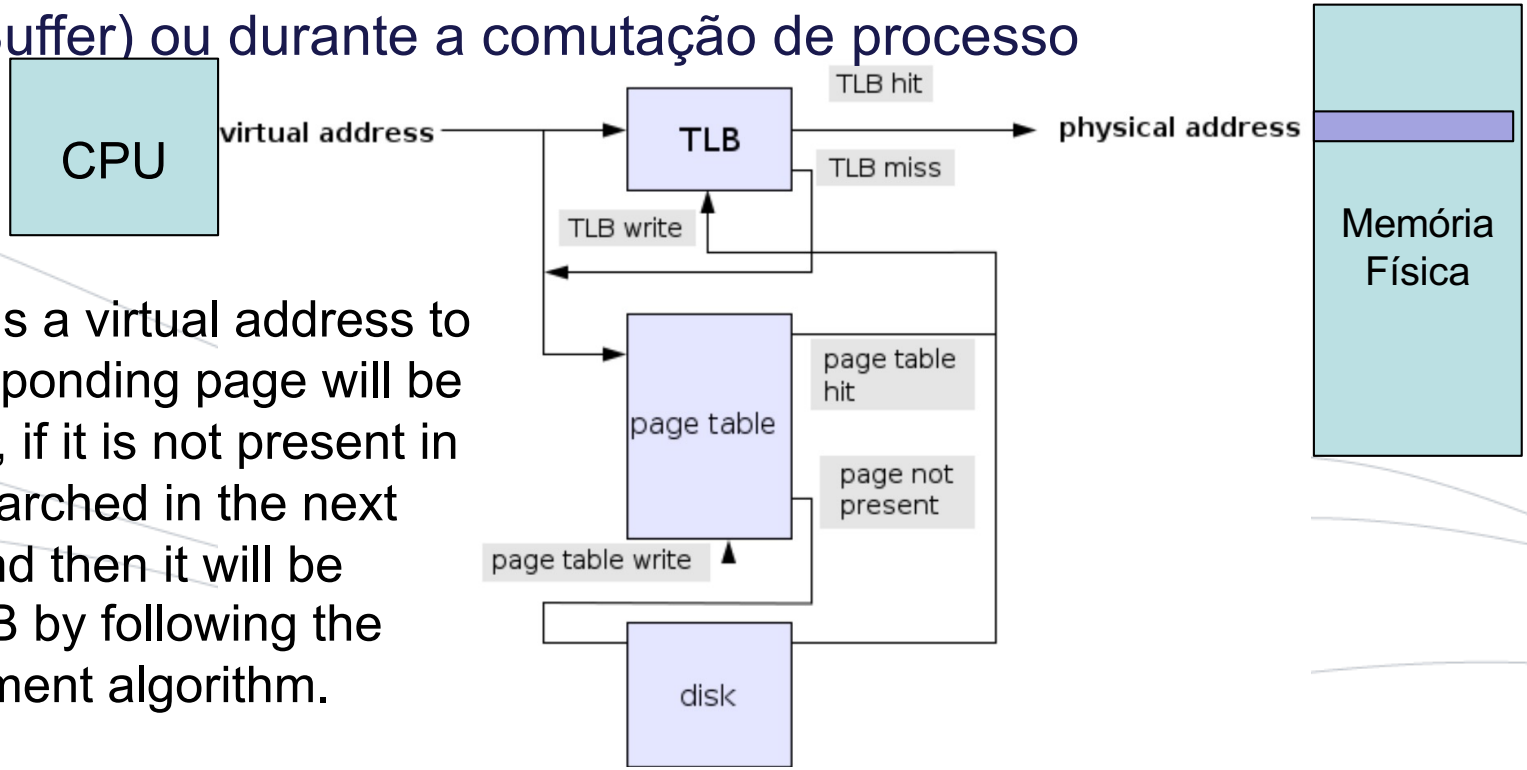
A MMU mapeia Endereço Lógico 37450 que um processo pretende aceder, para a página 9 e offset (resto) 586



Finalmente, a MMU Realiza o pedido de acesso à localização física 471626.

Tradução Tradicional

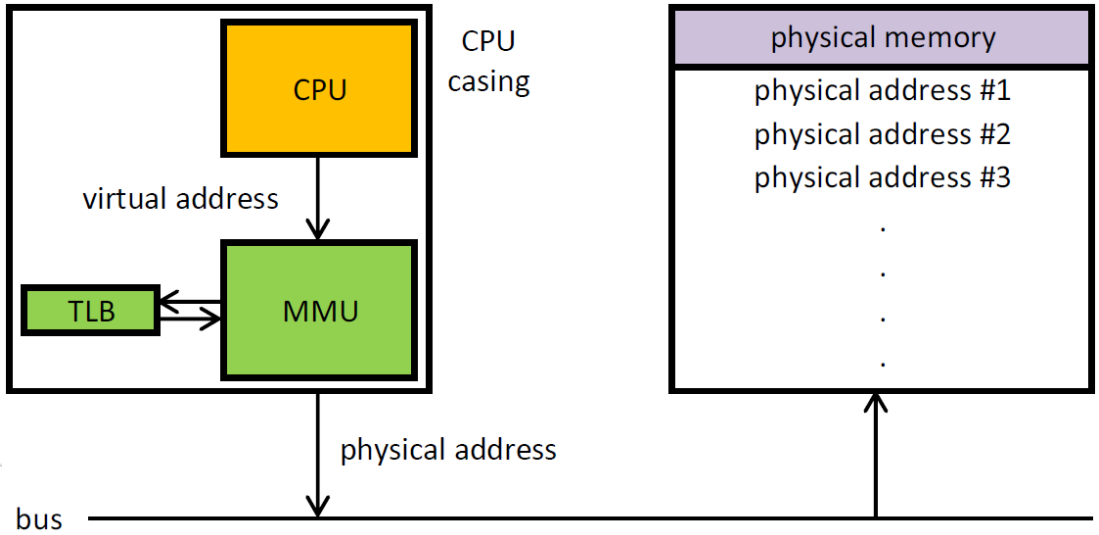
- As traduções são guardadas nas Page Tables. As mais recentes são *cached* no TLB (**translation lookaside buffer)**
- Um TLB hit elimina um acesso à memória/cache
- O endereço da Page Table raiz de um novo processo escalonado (atual) é guardada no registo de controlo CR3 (em x86).
- O TLB é flushed usando a instrução invlpg (Invalidate Translation Lookaside Buffer) ou durante a comutação de processo



When CPU sends a virtual address to MMU, the corresponding page will be searched in TLB, if it is not present in TLB, it will be searched in the next level memory, and then it will be placed in the TLB by following the suitable replacement algorithm.

Tradução Tradicional

- Note that switching between threads of one process does not involve reloading CR3 or flushing caches, and thus is significantly more lightweight.

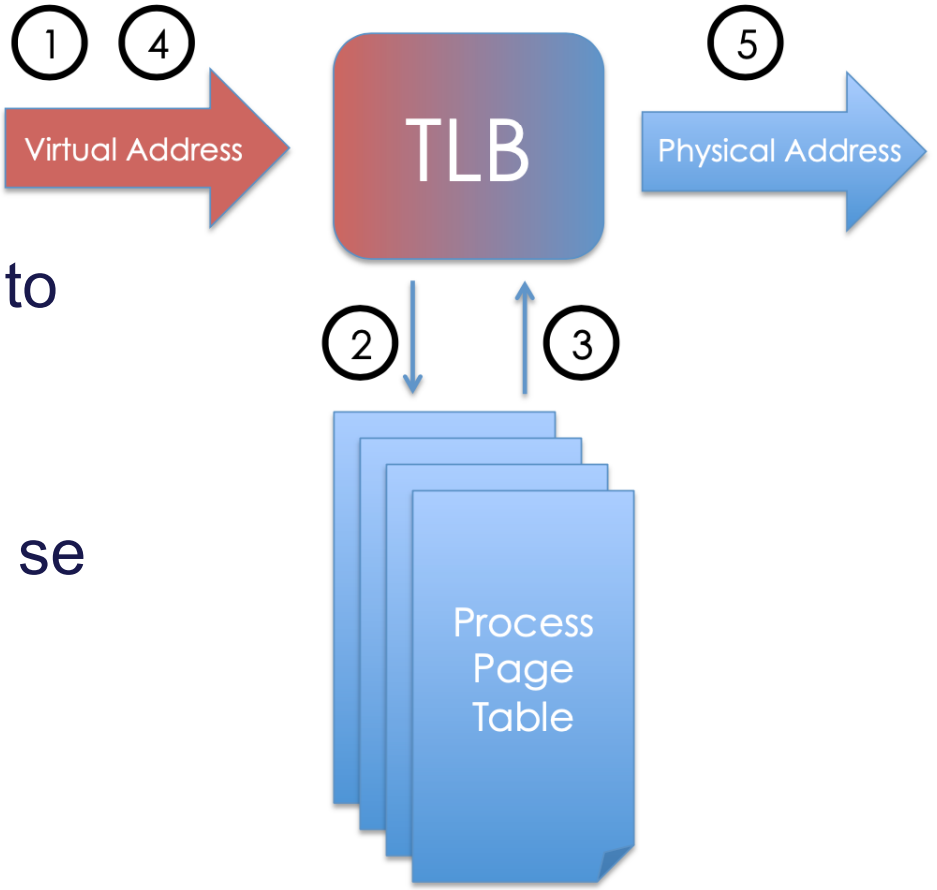


CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

Tradução Tradicional

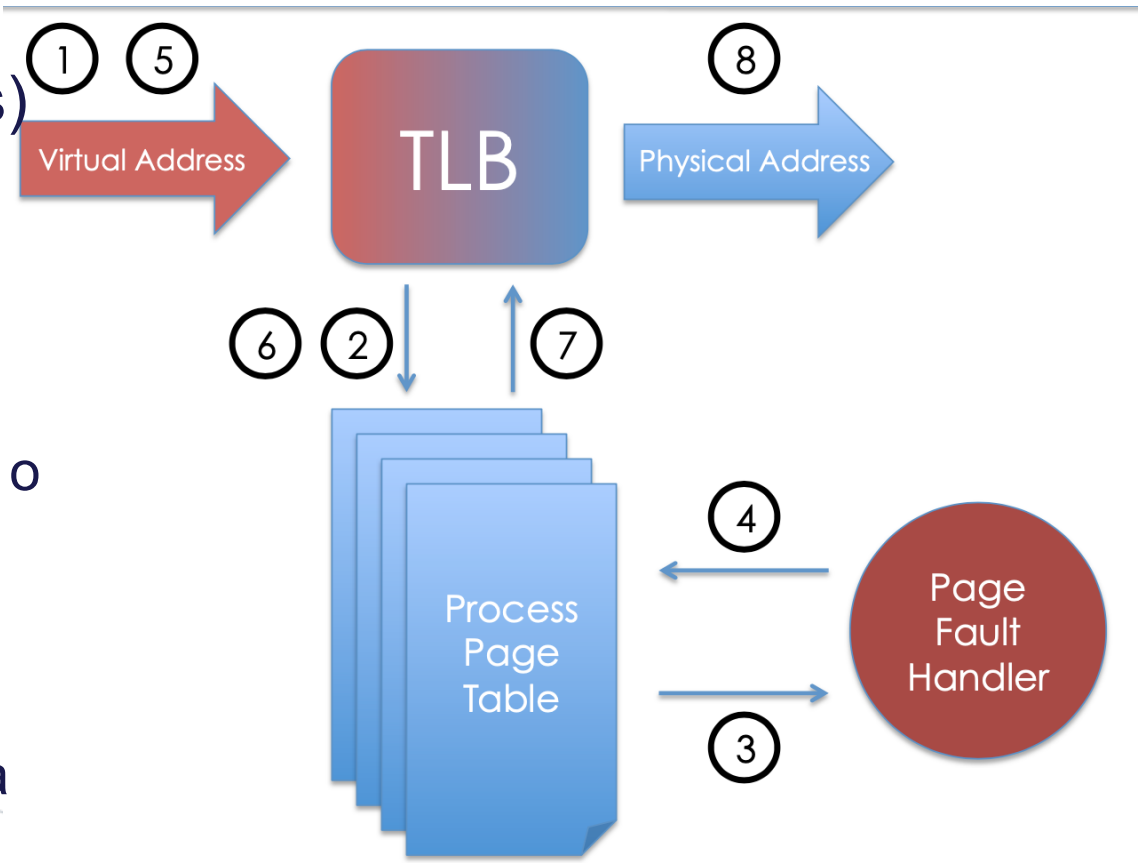
- Caso 1: Endereço disponível no TLB (TLB hit)

- Caso 2: Endereço não disponível no TLB (TLB miss), mas o mapeamento está disponível na Page Table.
- O HW copia para o TLB, se necessário apaga uma entrada no TLB com um mapeamento antigo.



Tradução Tradicional

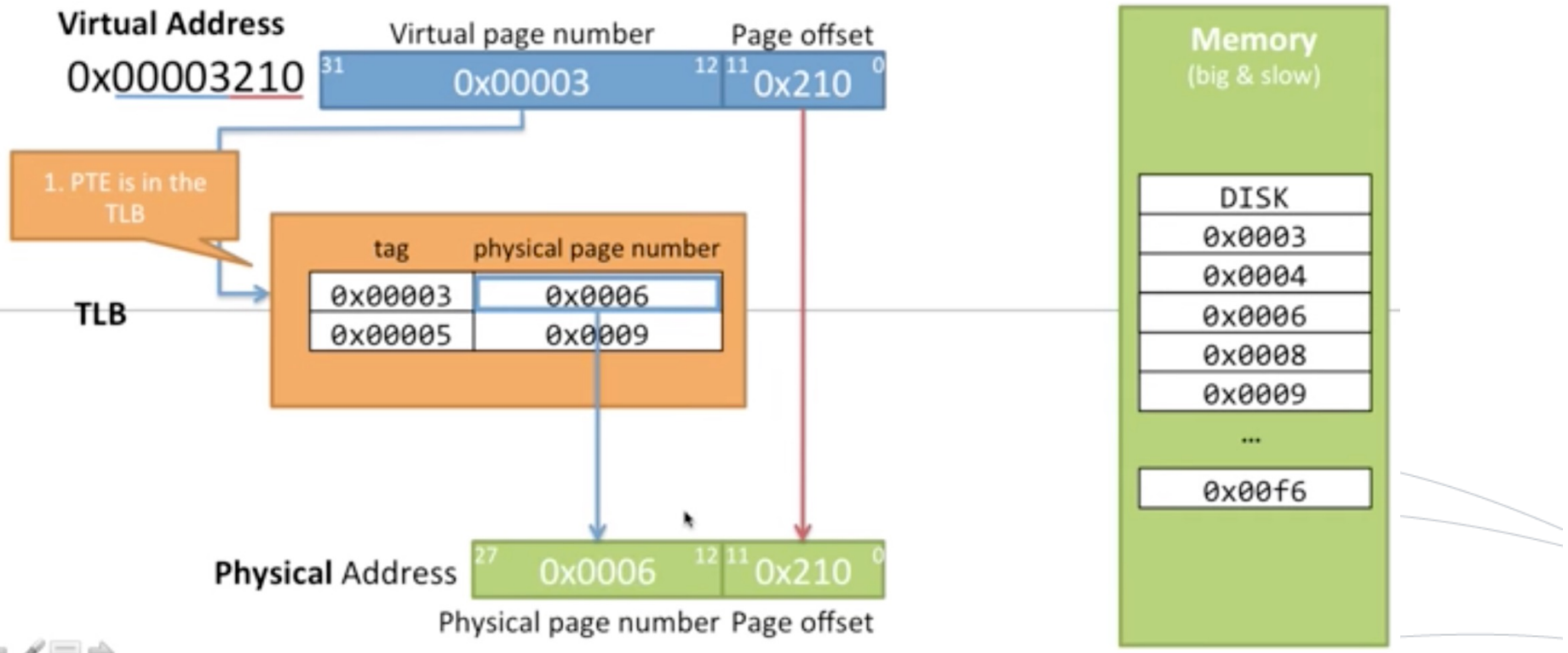
- Caso 3: Endereço não disponível no TLB (TLB miss) e a Page não está presente na RAM.



- A MMU causa uma trap e coloca o endereço da página em falta no registo de control CR2 do processador.
- O SO corrige a entrada na tabela de páginas
- A instrução que desencadeou a falta é reexecutada (passo 5)

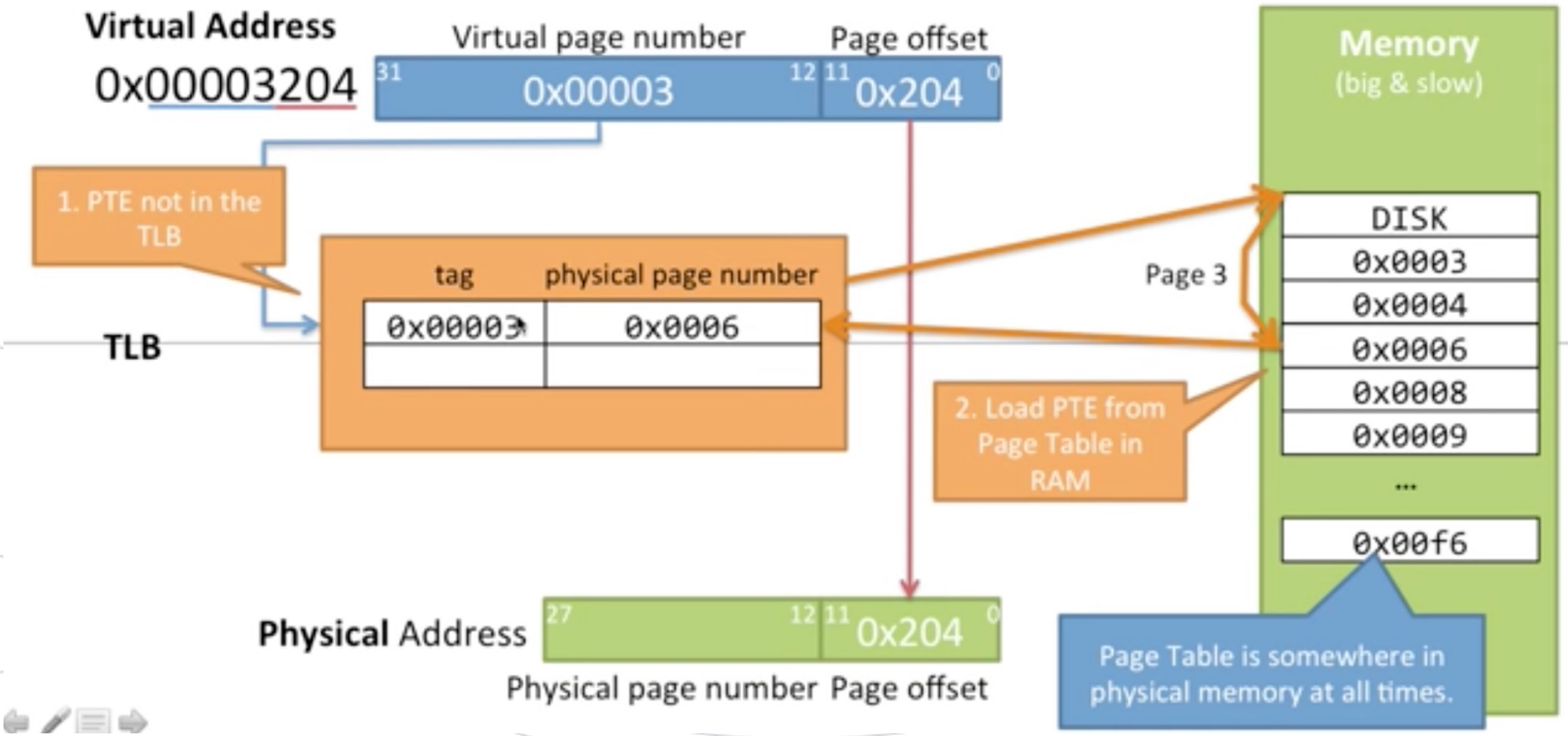
Tradução Tradicional

- Translation caches in TLB (**translation lookaside buffer)** caches – Exemplo o *TLB hit, assumindo que é possível o tipo de acesso solicitado; caso não seja permitido é gerada uma exceção e é causada uma trap para o SO)*



Tradução Tradicional

- Translation caches in TLB (**translation lookaside buffer)** caches – Exemplo o *TLB miss*



Nota: Na figura já se mostra a atualização da PTE, inicialmente a entrada está vazia.

Virtualização de Memória

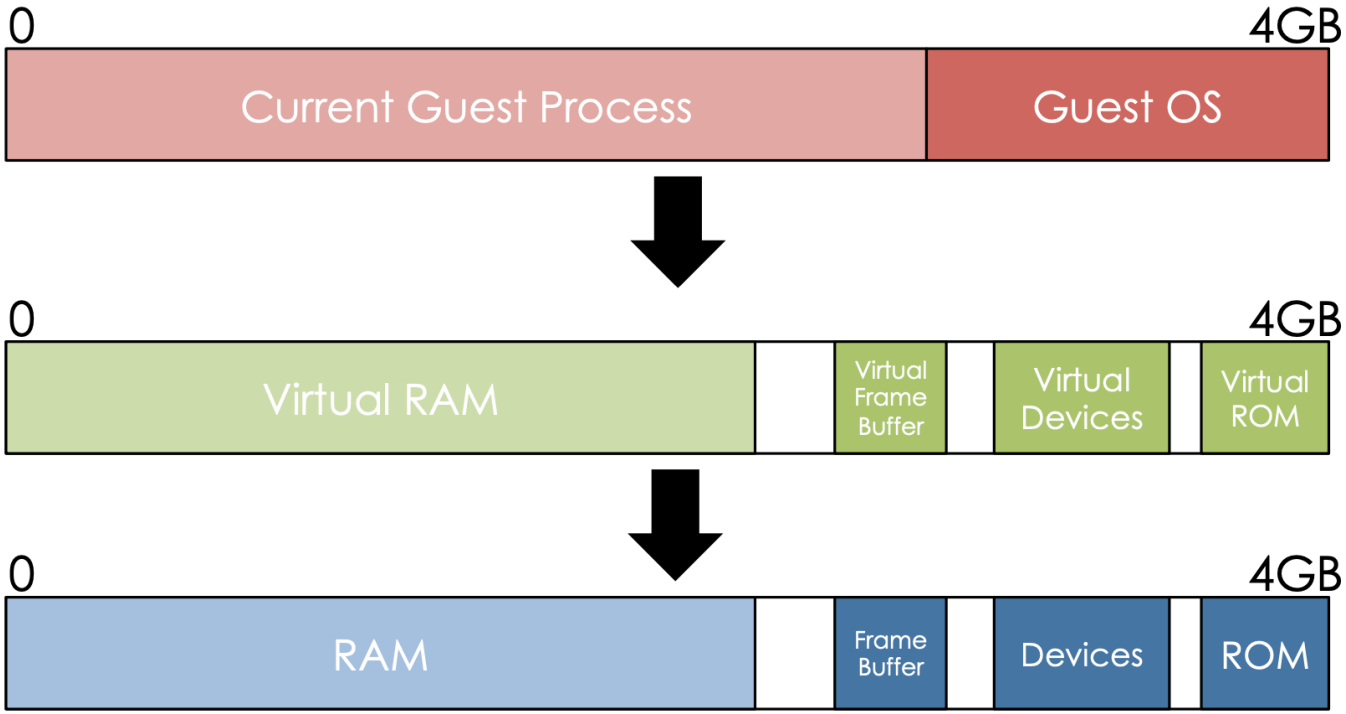
- **Virtual memory virtualization** generalizes the concept of virtual memory.
- It's similar to the **virtual memory support** provided by modern operating systems.
  - Application provided a logical view of memory
  - OS manages actual real memory
  - OS maintains **mappings of virtual memory** to **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.

Requisitos para a Virtualização de Memória

- Virtual memory virtualization **involves sharing the physical system** memory in RAM **and** dynamically **allocating it to the physical memory of the VMs.**
- Three abstractions of memory
  - Machine: actual hardware memory
    - 2GB of DRAM
  - Physical: abstraction of hardware memory managed by OS
    - If a VMM allocates 512 MB to a VM, the OS thinks the computer has 512 MB of contiguous physical memory
    - (Underlying machine memory may be discontiguous)
  - Virtual: virtual address spaces you know and love
    - Standard 2^32 address space

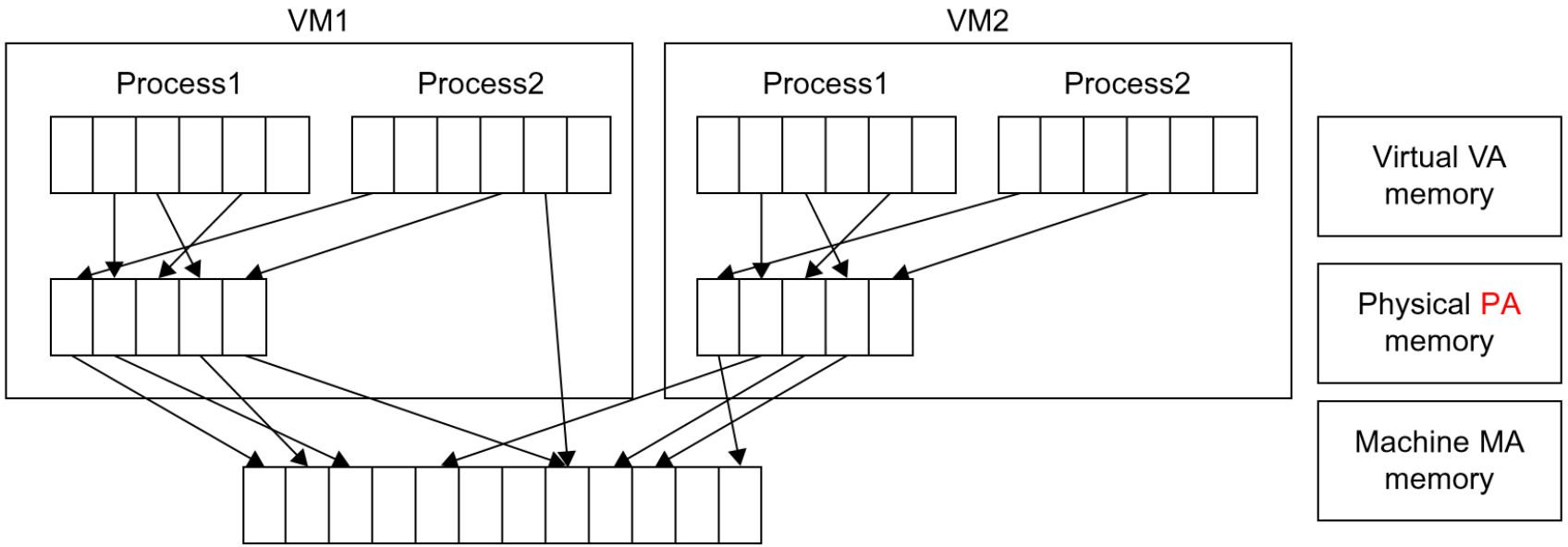
Requisitos para a Virtualização de Memória

- A **two-stage mapping process** should be maintained by the guest OS and the VMM, respectively:
  - **1. Guest virtual memory to Guest physical memory**
  - **2. Guest physical memory to Host machine memory**.

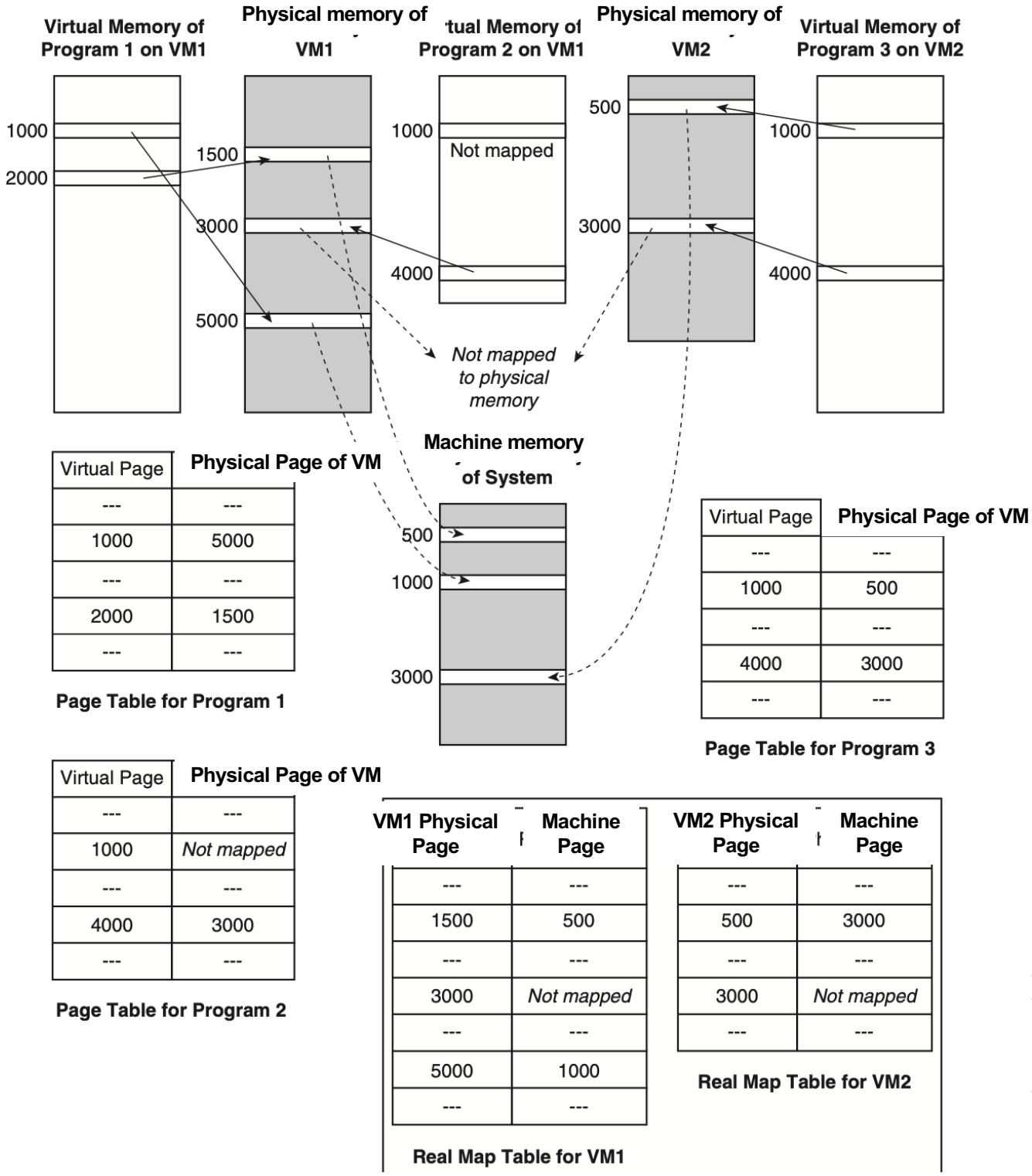


Requisitos Memory Virtualization

Two-level memory mapping procedure.



Requisitos Memory Virtualization



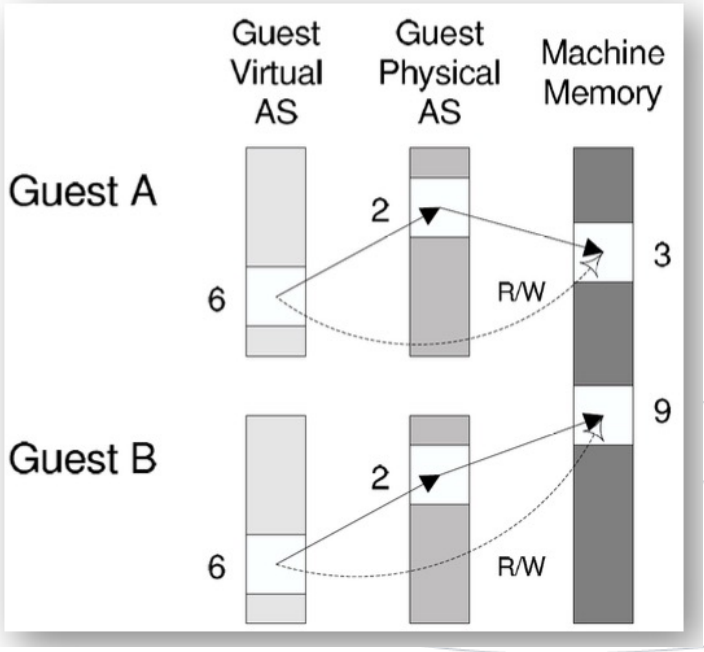
Requisitos Memory Virtualization

- **MMU virtualization should be supported**, which is transparent to the guest OS. In each VM, guest OS continues to create and manage page tables for its virtual address spaces; **but these page tables are not used by the MMU hardware**.

- **We can store only one translation in the TLB** – e.g., Guest Virtual Address -> Host Physical Address

Memory Virtualization – Método 1
Software-assisted Paging (Shadow Page Tables)

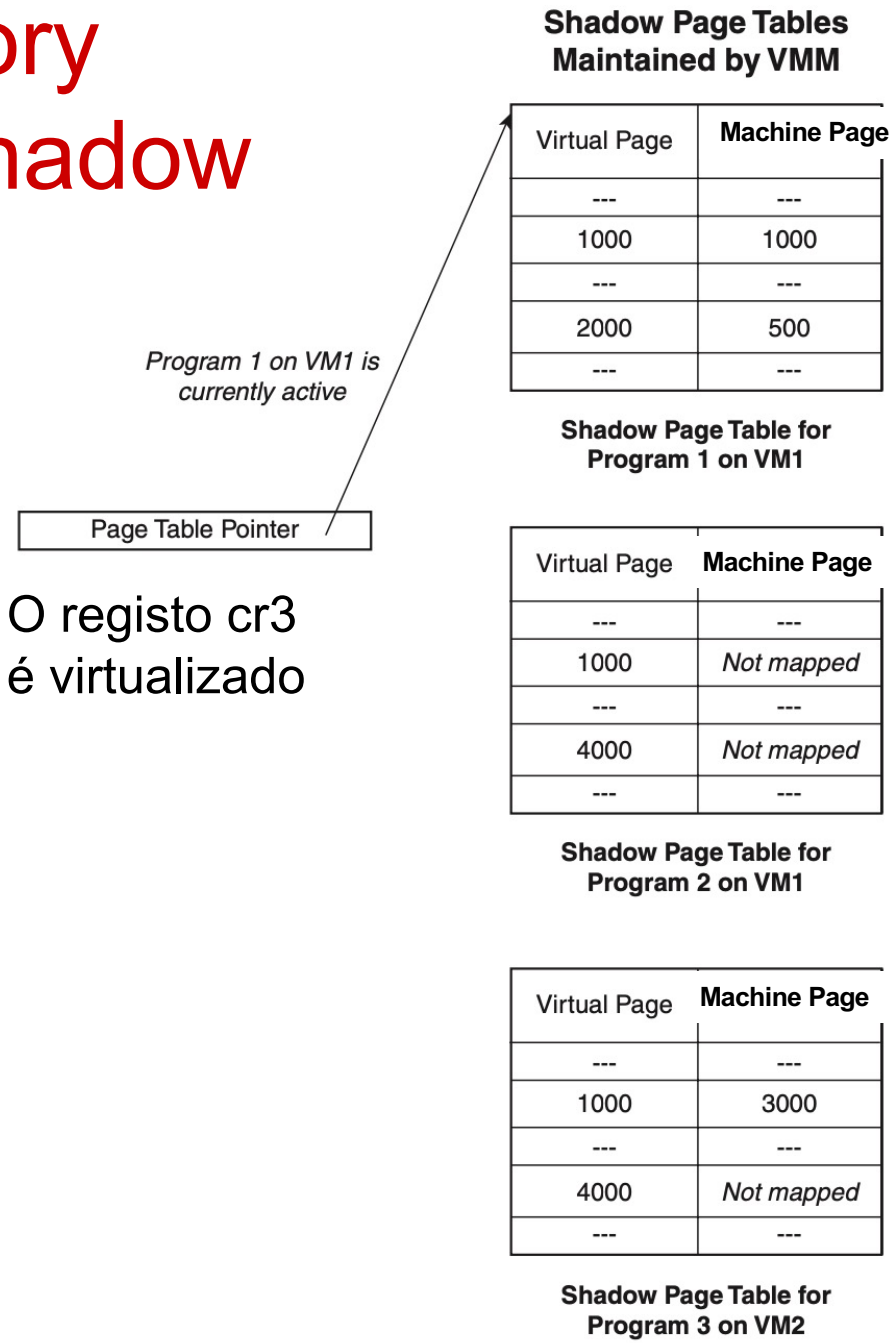
- VMM creates and manages another set of page tables that **map virtual pages directly to machine pages**.
  - **A VMM page table** is called **shadow page table**.
  - **Shadow Page Tables are used** by the HW to translate virtual addresses and **to keep TLB up-to-date.**
- VMM needs to keep its V→M tables consistent with changes made by OS to its V→P tables



Memory Virtualization – Shadow Page Tables

- **To make this method work, the page table pointer register is virtualized.**
- The VMM manages the real page table pointer and has access to the virtual version of the register associated with each guest VM.
- At the time the VMM activates a guest VM, it updates the page table pointer so that it indicates the correct shadow version of the guest VM current page table.
- **If the guest VM attempts to access the virtualized page table pointer, either to read it or write it, the read or write instruction traps to the VMM (**because these instructions are privileged):
  - If is a Read attempt: The VMM returns the guest's virtual page table pointer;
  - If is a Write attempt: The VMM updates the virtual version and then sets the real table pointer to the corresponding shadow table pointer.

Requisitos Memory Virtualization –Shadow Page Tables



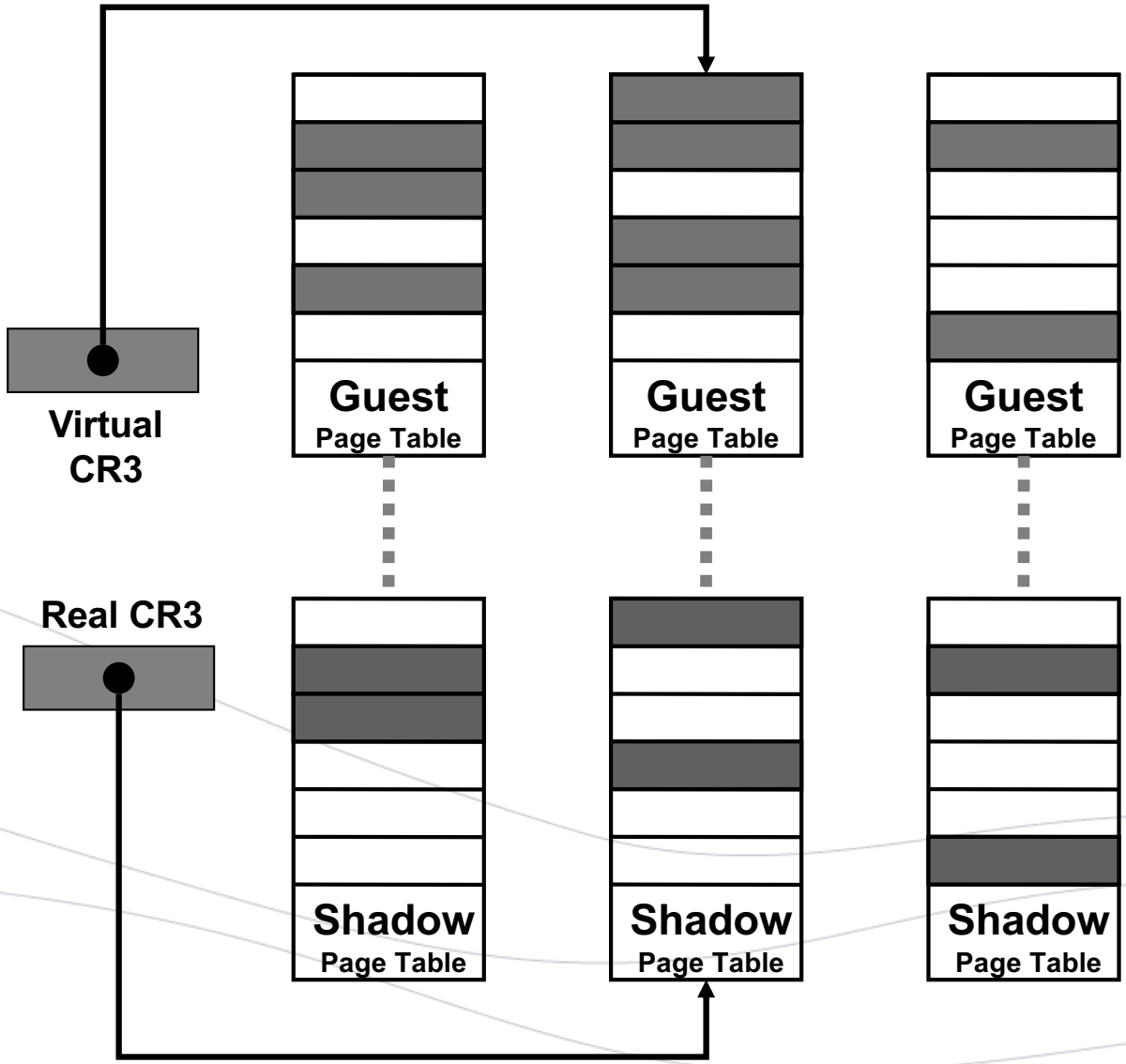
O registo cr3 é virtualizado

The virtual-to-physical mapping is kept by the VMM in *shadow page tables*, one for each of the guest VMs:

- Index: guest virtual address
- Output: host physical address or machine address page

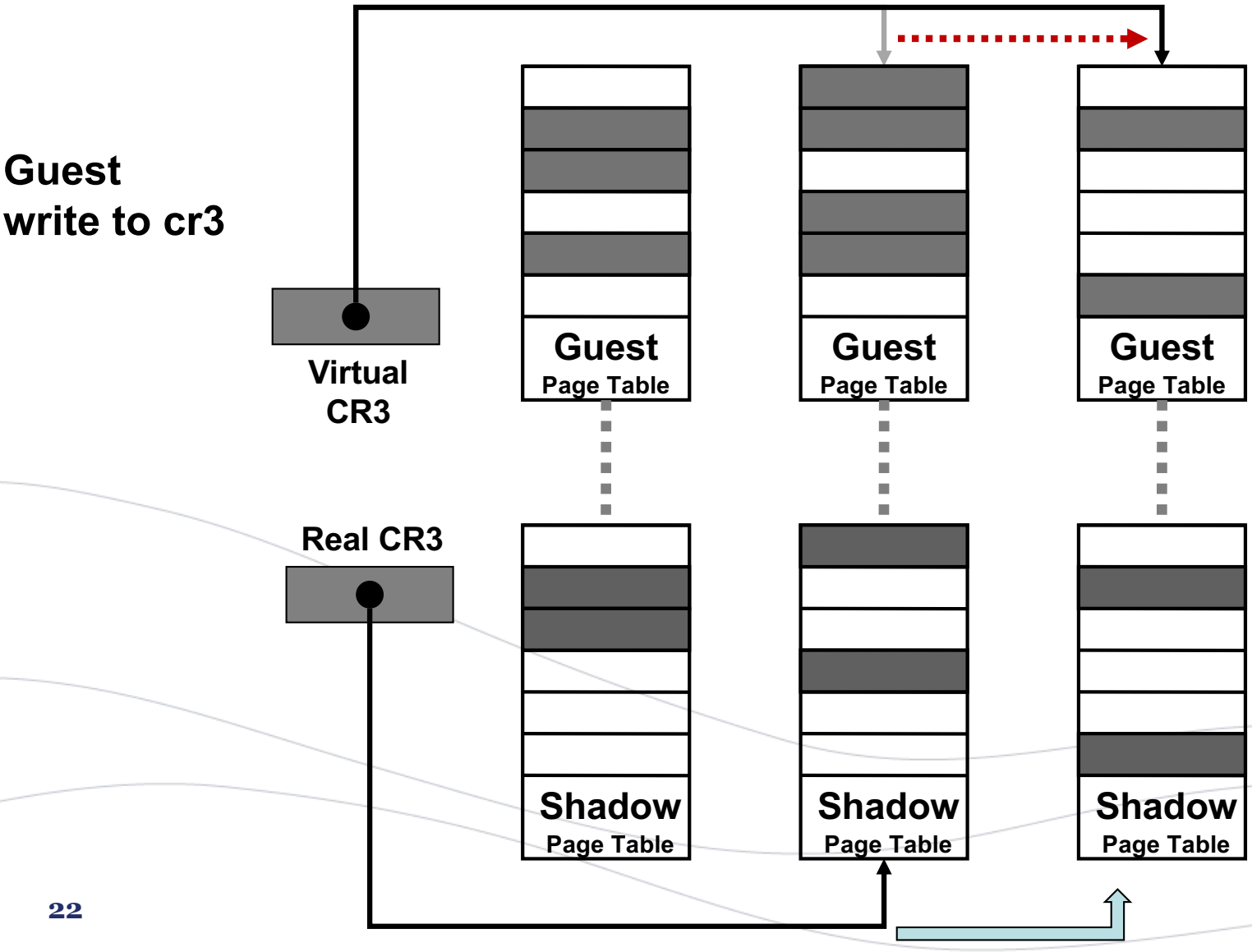
Figure 8.15 Shadow Page Tables for Memory Mapping Illustrated in Figure 8.14. *The shadow tables ar used by hardware in performing address translation. The VMM manages the shadow tables and i responsible for setting the page table pointer register.*

Requisitos Memory Virtualization –Shadow Page Tables



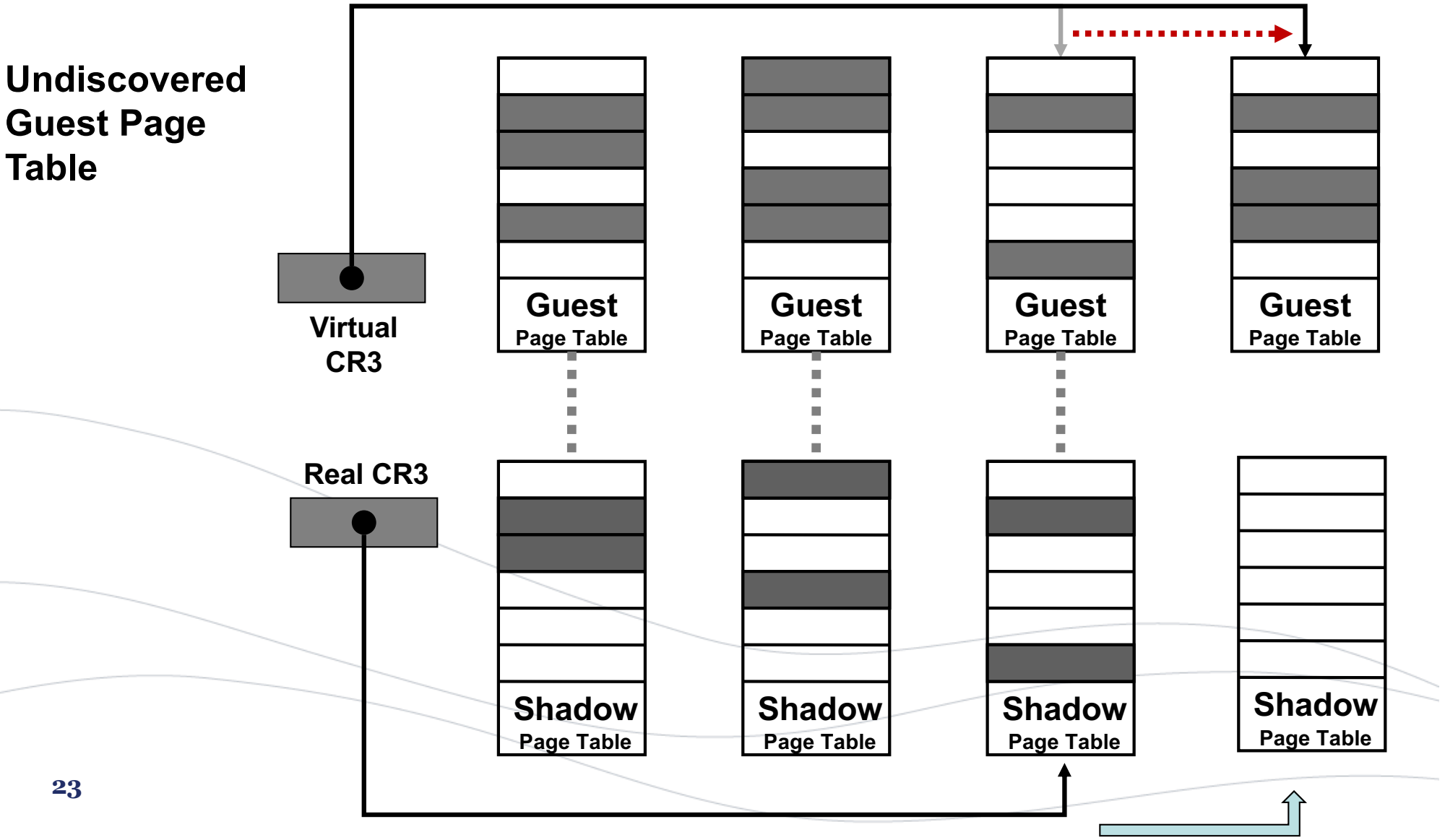
O SO na VM não pode ter acesso ao CR3 real. É de acesso previligiado.
O hypervisor captura todas as tentativas de acesso ao CR3 virtual e redirecciona para uma tabela de páginas especifica.

Requisitos Memory Virtualization –Shadow Page Tables

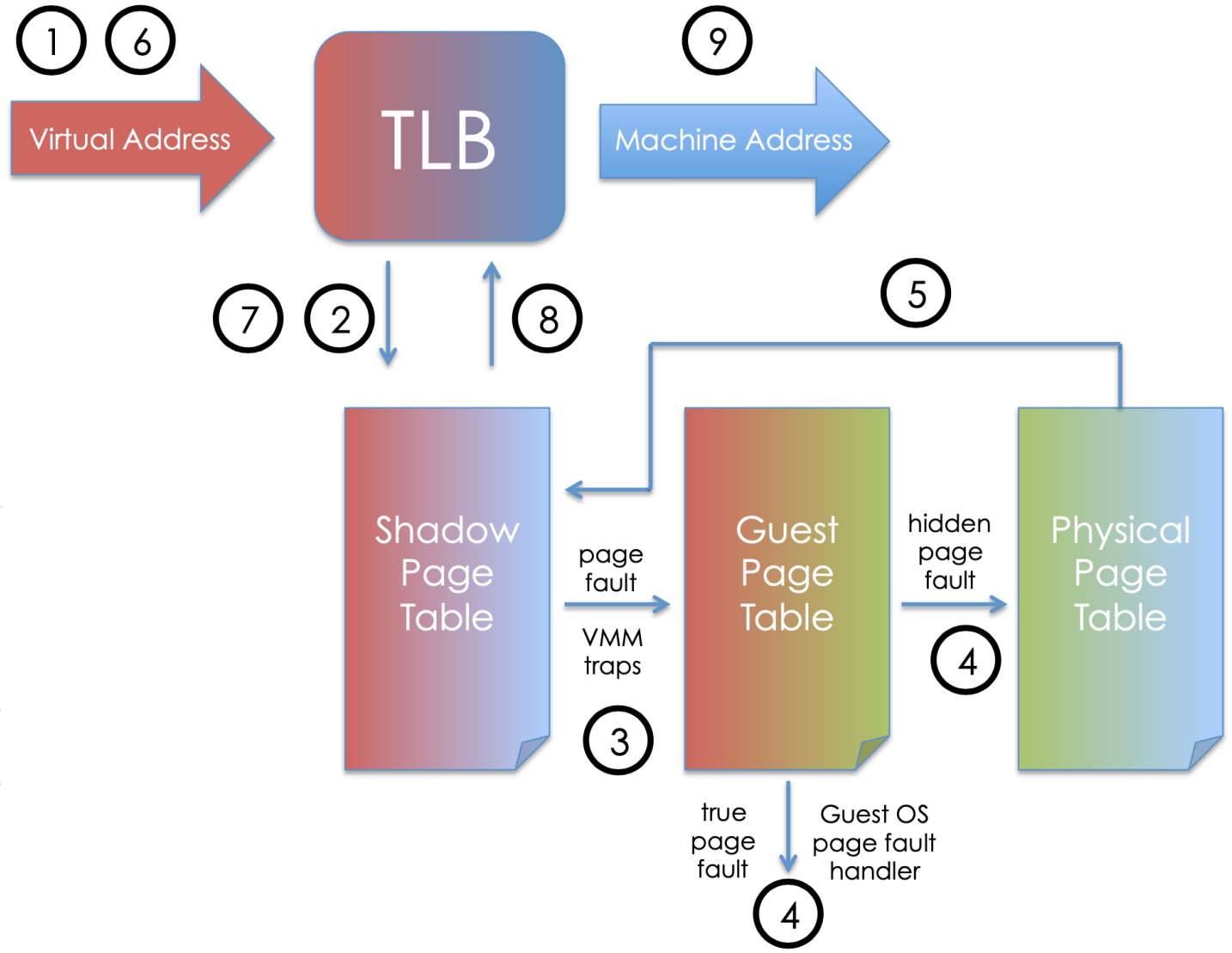


The VMM updates the virtual version and then updates the real hw table pointer to the corresponding shadow table pointer.

Requisitos Memory Virtualization –Shadow Page Tables



Requisitos Memory Virtualization –Shadow Page Tables

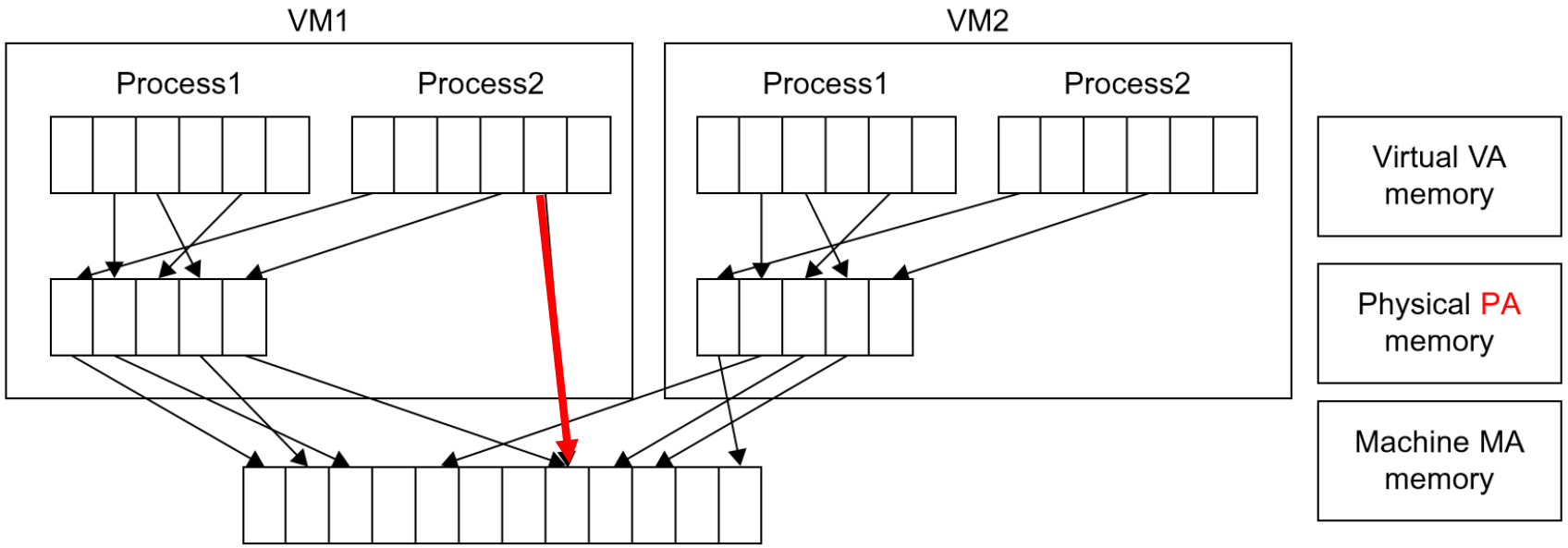


Requisitos Memory Virtualization – Shadow Page Tables (Page Faults)

- **Gestão dos Mapeamentos**
- **The true mapping of virtual to machine pages may differ from the virtual to physical view that the guest operating systems have, and page fault handling must take this into account.**
- **Hidden Page Fault:** If it is mapped in the guest, this page fault should be handled entirely by the VMM. This is a case where the VMM has moved the accessed physical page of VM to swap space. Consequently, the VMM brings the physical page back into physical memory and then updates the physical map table and the affected shadow table(s) appropriately to reflect the new mapping. The guest OS is not informed of the page fault because such a page fault would not have occurred if the guest OS were running natively.
- **Page Fault:** if the page is not mapped in the guest, the VMM transfers control to the trap handler of the guest, indicating a page fault. The guest OS then issues I/O requests to effect a page-in operation (possibly with a swap out of a dirty page). The guest OS then issues instructions to modify its page table. These requests are intercepted by the VMM, either because they are privileged instructions or because the VMM write-protects the area of memory holding the page table of the guest. At that point the VMM updates the page table and also updates the mapping in the appropriate shadow page table before returning control back to the guest virtual machine.

Requisitos Memory Virtualization –Shadow Page Tables

Two-level memory mapping procedure.



Memory Virtualization – Shadow Page Tables

- **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**.

Memory Virtualization – Shadow Page Tables

- 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**.

Requisitos Memory Virtualization – Método 2 (SLAT)

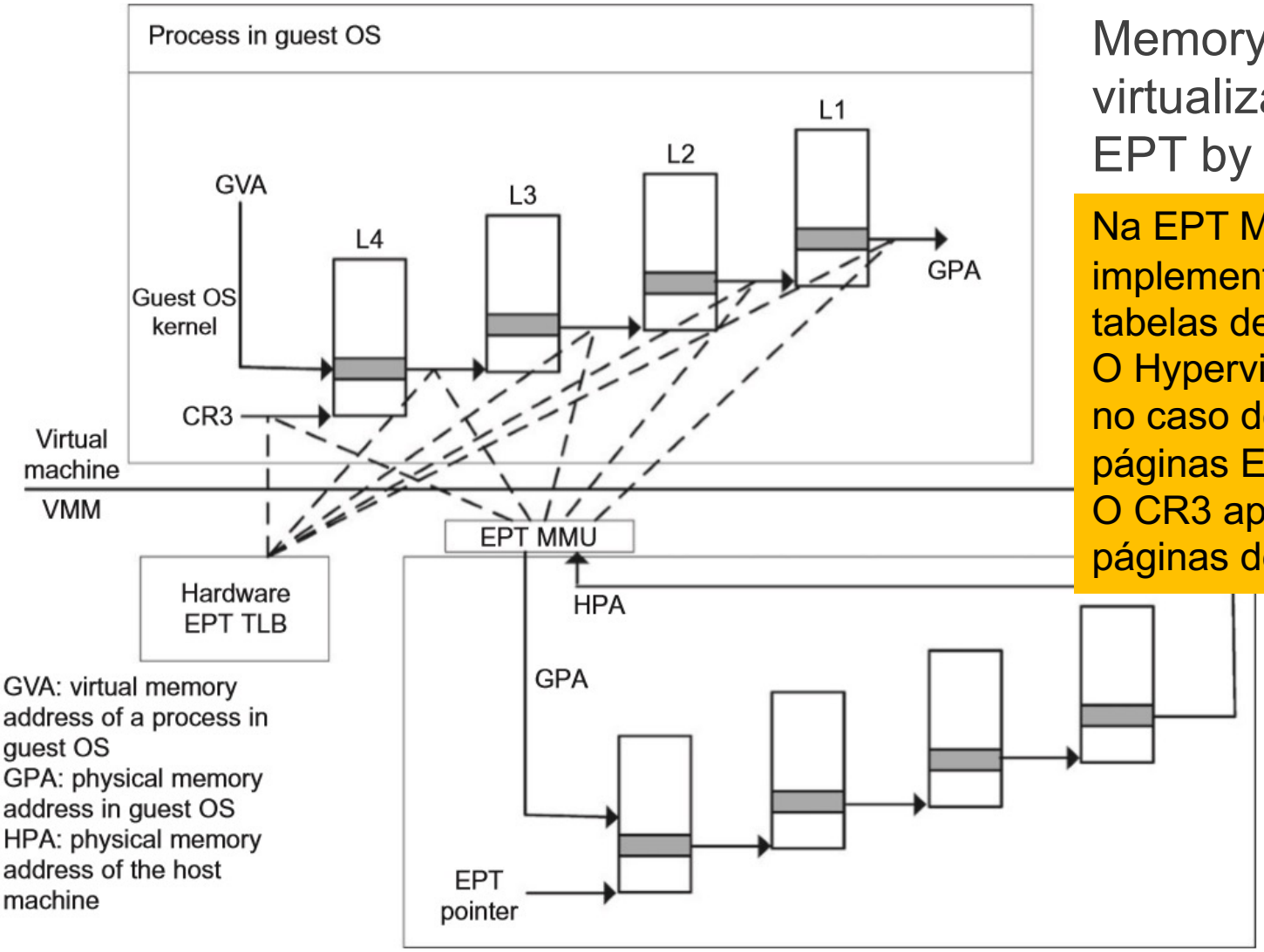
- In order to make this translation more efficient, processor vendors implemented technologies commonly called SLAT (Second Level Address Translation (SLAT)), also known as nested paging.
- By treating each **guest-physical address** as a **host-virtual address**, a slight extension of the hardware used to walk a non-virtualized page table (now the guest page table) can walk the host page table.
- With multilevel page tables the host page table can be viewed conceptually as nested within the guest page table. A hardware page table walker can treat the additional translation layer almost like adding levels to the page table.

Extended Page Tables (EPT)
É a implementação SLAT da Intel.

Rapid Virtualization Indexing (RVI) ou NPT (Nested Page Tables)
É a implementação SLAT da AMD.

Memory Virtualization HW-Assisted

Extended Page Tables (EPT)
É a implementação SLAT da Intel.



Memory virtualization using EPT by Intel.

Na EPT MMU são implementadas tabelas de páginas EPT.
O Hypervisor é envolvido no caso de falhas de páginas EPT.
O CR3 aponta para as páginas de nível mais alto.

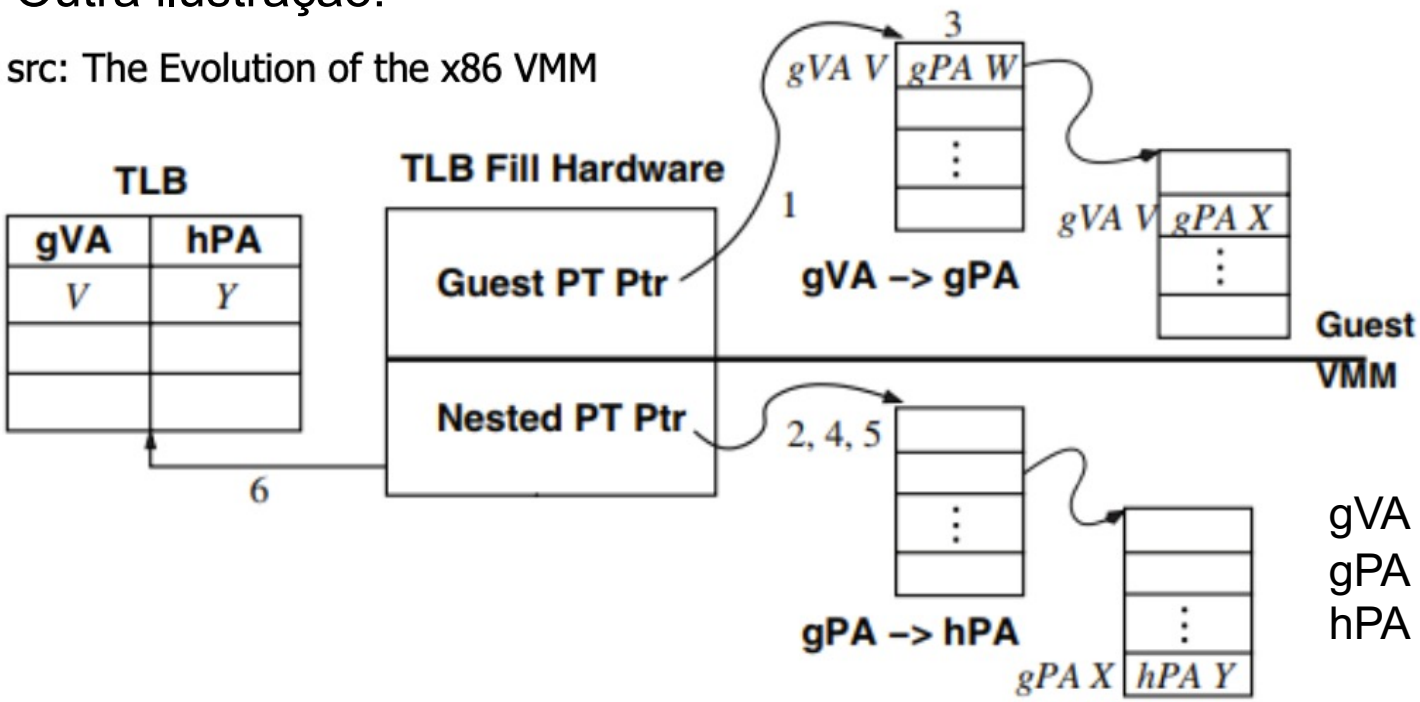
Novo EPT base pointer

Memory Virtualization HW-Assisted

Extended Page Tables (EPT)
É a implementação SLAT da Intel.

Outra ilustração:

src: The Evolution of the x86 VMM



Memory virtualization using EPT by Intel.

gVA = Guest Virtual Address
gPA = Guest Physical Address
hPA = Host Physical Address

- No need for hypervisor involvement
- HW will walk guest page table, then host page table, and install entry in the TLB

This composite translation eliminates the need to maintain shadow page tables and synchronize them with the guest page tables.

Input/Output Virtualization

- Virtualization of the input/output subsystem is one of the more difficult parts of implementing a system virtual machine environment.
- Each I/O device type has its own characteristics and needs to be controlled in its own special way.
- The difficulty of virtualizing I/O devices is compounded by the fact that the number of devices of a given type is often very large and continues to grow.
- **Virtualization of such a large number of devices is a challenge for any new VMM.**

Input/Output Virtualization

- The **virtualization strategy** for a given I/O device type **consists of constructing a virtual version of the device and then virtualizing the I/O activity** directed at the device.
- A virtual device given to a guest VM is typically (but not necessarily) supported by a similar, underlying physical device.
- When a guest VM makes a request to use the virtual device, the request is intercepted by the VMM. The VMM converts the request to the equivalent request for the underlying physical device and it is carried out.
- The next two subsections discuss techniques for virtualizing devices and carrying out virtual I/O activity.

Input/Output Virtualization
Tipos de Dispositivos

- **Dedicated I/O devices**
  - Examples: display, keyboard, mouse etc.
  - **No device virtualization necessary**
  - The request are routed through the VMM because the guest OS is running in the nonprivileged user mode
  - **Interrupts from the device are first handled by the VMM**, which determines that it came from a device dedicated to a particular guest VM and queues up the interrupt to be handled by the VM when it is activated by the VMM.

Input/Output Virtualization
Tipos de Dispositivos

- **Partitioned devices**
  - A disk, for example, can be partitioned into several smaller virtual disks that are then made available to the VM. A virtual disk is treated by a VM exactly as a physical disk.
  - To emulate an I/O request for the virtual device such as a disk, **the VMM must translate the parameters (e.g., the track and sector locations) into corresponding parameters for the underlying physical device(s),** using a map maintained for this purpose, and then reissue the request to the physical device (e.g., the disk controller).
  - Similarly, status information coming back from the physical device is used to update status information in the map and is transformed into appropriate parameters for the virtual device before being delivered to the guest VM.

Input/Output Virtualization
Tipos de Dispositivos

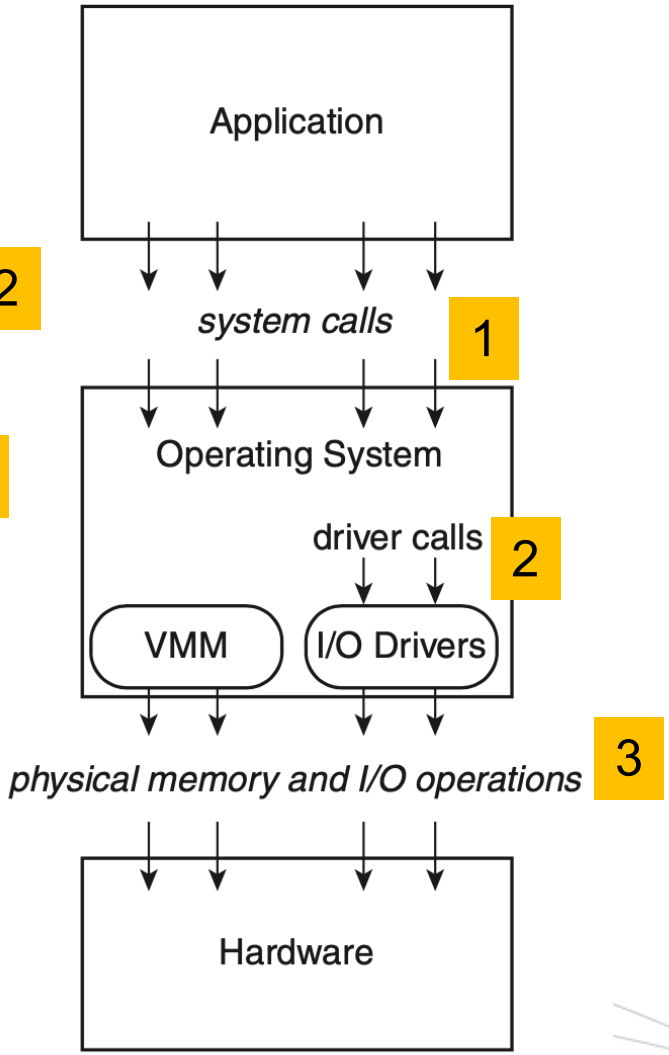
- **Shared devices**
  - Devices, such as a network adapter, **can be shared** among a number of guest VMs at a **fine time granularity**.
  - **Each guest may have its own virtual state related to usage of the device**, e.g., a virtual network address. This state information is maintained by the VMM for each guest VM.
  - A request by a guest VM to use the device is translated by the VMM to a request for the physical device through a virtual device driver.
  - For a network, the VMM's virtualization routine will translate a request from some virtual machine to a request on a network port using its own physical network address. Similarly, the incoming requests through various ports will be translated into requests for virtual network addresses associated with the different virtual machines.

Input/Output Virtualization
Tipos de Dispositivos

- **Spooled devices**
  - Spooled device is also shared but at **higher granularity**, e.g., printer
  - A printer is solely under the control of the program that is printing until the completion of the printing task.
  - Two-level spool tables:
    - first level is in the OS
    - and the second level is in the VMM
  - **An operating system's request to print a spool buffer is intercepted by the VMM**, which copies the buffer contents into one of its own spool buffers. This allows VMM to schedule requests from different VMs on the same printer.

Virtualizing I/O Activity

- An operating system abstracts most of the peculiarities of hardware devices and makes these I/O devices accessible through the system call interface and the device driver interface. 1 2
- At a level even lower is the operation-level interface, where individual instructions interact with the I/O system, typically by placing device-specific addresses and data on an I/O bus. 3



Exemplo: open() ou read().
São chamadas independentes do HW.
São convertidas em chamadas
sobre as rotinas do driver.

Figure 8.18 Major Interfaces in Performing an Input/Output Action.

Virtualizing I/O Activity

- When an I/O action is to be performed, part of the action is carried out by guest software and part by the VMM.
- The VMM can intercept a guest's I/O action and convert it from a virtual device action to a real device action at any of the three interfaces:
  - **the system call interface,**
  - **the device driver interface,**
  - **or the operation-level interface.**
- Note that it is the VMM that finally interacts directly with the hardware device. Hence the VMM that either invokes the device driver or controls access to the device driver.

Virtualizing I/O Activity

- **Virtualizing at the I/O Operation Level**
  - A natureza privilegiada (ou protegida) das operações I/O facilitam a intercepção pelo VMM, uma vez causam uma Trap em modo de utilizador.
  - No entanto, uma vez interceptadas, pode ser difícil para o VMM determinar exactamente que acção I/O está a ser solicitada.
  - Uma acção I/O completa (por exemplo, uma leitura do disco) envolve normalmente a emissão pelo driver do dispositivo de várias instruções I/O de pequena granularidade.
  - O VMM precisa de deduzir qual a acção I/O de nível superior a ser executada, para isso deve ser capaz de ”reverter" as operações individuais. Isto pode ser extremamente difícil na prática.

Virtualizing I/O Activity

- **Virtualizing at the Device Driver Level**
  - Uma chamada de sistema tal como read() é convertida pelo SO noutra chamada (ou chamadas) para a interface do driver do dispositivo.
  - O VMM intercepta uma chamada para o driver do dispositivo virtual, e converte a informação do dispositivo virtual para o dispositivo físico correspondente e redirecciona a chamada para o driver do dispositivo físico.
  - **Este esquema é simples e permite a virtualização de forma natural**, mas exige que o developer do VMM tenha algum conhecimento do SO convidado e das interfaces internas das implementações dos seus drivers de dispositivos.
  - **Precisa de drivers virtuais (um por dispositivo e por SO convidado).**
    - **Exige o desenvolvimento de virtual device drivers (one per device type) can be developed for each of the guest operating systems. Estes drivers are part of the overall VMM package distributed to users. Then when a guest OS is installed, the virtualized drivers are installed at the same time.**
    - **VMM precisa dos *real device drivers* de cada dispositivo. O problema pode ser simplicado pedindo emprestados ("borrowing") os drivers e a driver interface de um SO existente (e.g. Linux). Num ambiente de virtualização alojada, são usados os drivers existentes para o host OS.**

Virtualizing I/O Activity

- **Virtualizing at the System Call Level**
  - Forma mais eficiente é interceptar o pedido I/O inicial sobre interface ABI (application binary interface) do OS. Depois, toda a acção I/O poderá ser realizada pelo VMM.
  - Para o conseguir, contudo, o VMM precisa de rotinas ABI “sombra” das rotinas ABI disponíveis para o utilizador.
  - A escrita das rotinas de emulação da ABI terá de fazer parte do desenvolvimento do VMM e exige um conhecimento dos OS convidados muito mais profundo do que a escrita de drivers (como na subsecção anterior).
  - Também, todas as interacções entre as rotinas de emulação ABI e outras partes do SO convidado têm que ser fielmente emuladas.

Input/Output Virtualization

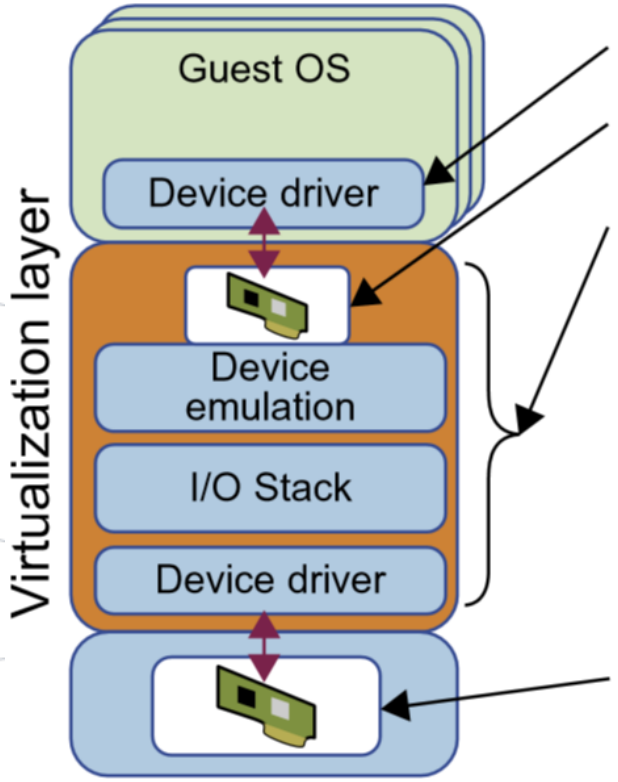
- **I/O virtualization** involves managing the **routing** of **I/O requests** between **virtual devices** and the **shared physical hardware**.
- There are **three ways** to implement **I/O virtualization**:
  - **Full device emulation**;
  - **Para-virtualization**;
  - **Direct I/O**.

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.



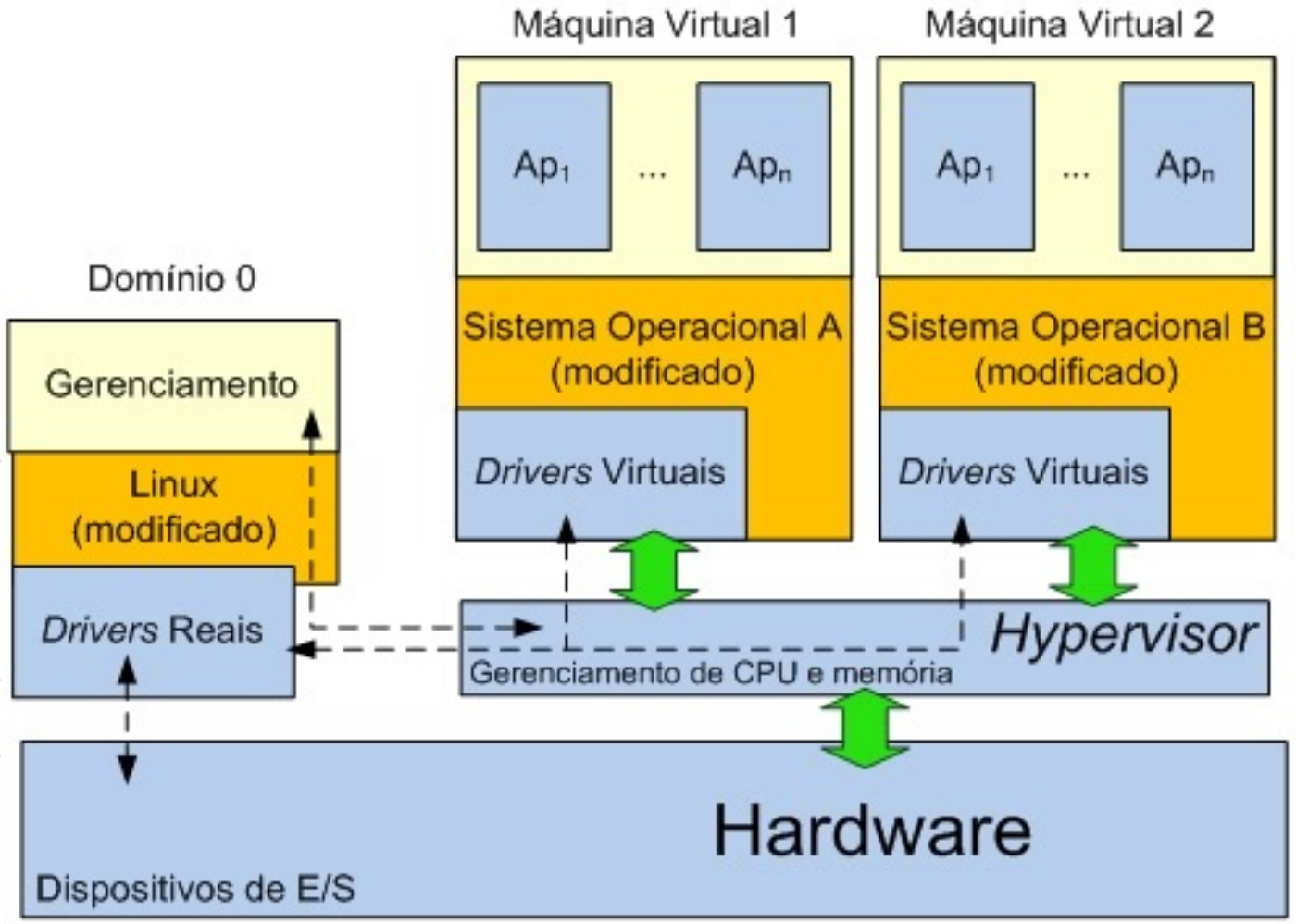
- Guest device driver
- Virtual device
- Virtualization layer
  - emulates the virtual device
  - remaps guest and real I/O addresses
  - multiplexes and drives the physical device
  - I/O features. e.g., COW disks
- Real device
  - may be different from virtual device

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

- The **para-virtualization method** of I/O virtualization is typically used in Xen.



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.

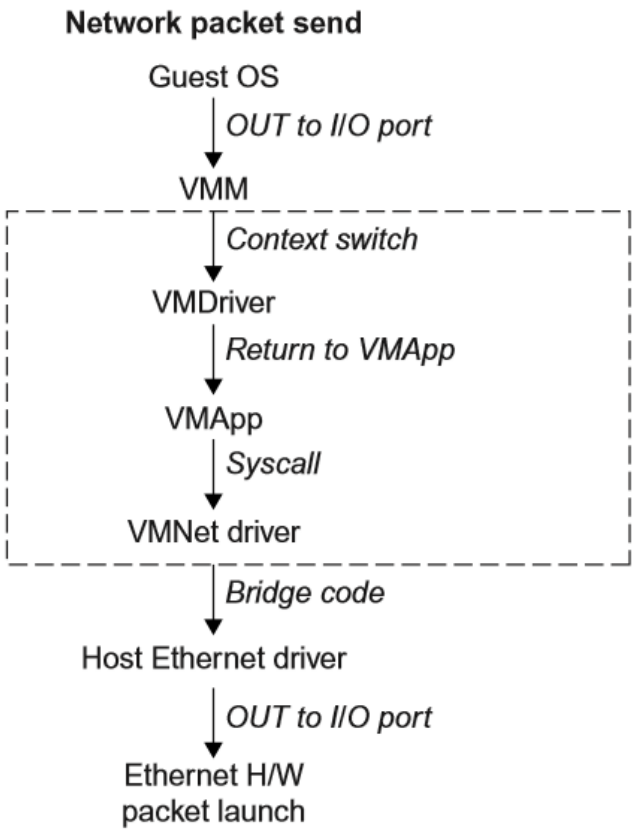
Input/Output Virtualization

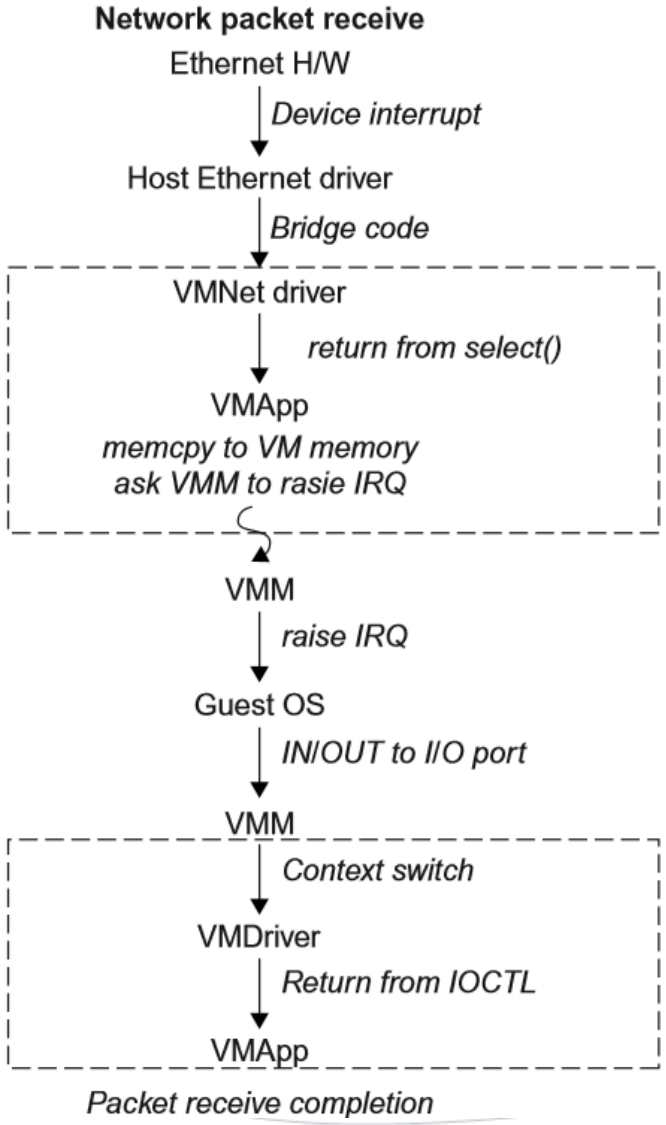
- **Intel VT-d (Virtualization Technology for Directed I/O)** makes direct access to a PCI device possible for guest systems with the help of the Input/Output Memory Management Unit (IOMMU) provided.
- A LAN card to be dedicated to a guest system, which makes attainment of increased network performance beyond that of an emulated LAN card possible. However, live migration of the guest system is no longer possible.
- VMware can be configured for use with an activated Intel VT-d system using VMware VMDirectPath for direct access to PCI cards.
- Pre-requisites for Intel VT-d
  - The chip set as well as the BIOS used must support Intel VT-d.
  - Requires the Intel Nehalem or later micro-architecture, such as that found in Xeon 34xx, 55xx and more recent, as well as other, desktop CPUs.

Input/Output Virtualization

- Another way to help I/O virtualization is via **self-virtualized I/O** (SV-IO).
- A self-virtualized I/O device is a peripheral with additional computational resources *close* to the physical device.
- **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, which exports a well-defined interface to the guest OS, such as ethernet or SCS.
- **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.