SUPSI

# Virtualization

Operating Systems

Amos Brocco, Lecturer & Researcher

14.09.18

# Objectives

- Understand why we want to virtualize
- Learn the different types of virtualization
- Study the main problems related to virtualization
- Study how memory virtualization works

**Browsing**
- Get a rapid overview.

**Reading**
- Read it and try to understand the concepts.

**Studying**
- Read in depth, understand the concepts as well as the principles behind the concepts.

*You are also encouraged to try out (compile and run) code examples!*



# System virtual machine: motivation

- ...everything started with the general availability of multi-processor and multi-core systems
  - Performance increased
    - ...But machines are not always fully exploited
    - Users want on-demand performance
  - Prices dropped
    - More performance at a lower cost

- Virtualization enables a more efficent use of a physical machine by creating and running more virtual systems depending on the users' need



# Types of virtualization

- **System Level Virtualization**
  - emulates a complete architecture which allows, for example, the execution of different operating systems on the same physical machine (Virtualbox, XEN, Vmware, KVM,...)
- **Application Level Virtualization**
  - abstraction which implements a new architecture in software, to allow portable software (Java Virtual Machine, Common Language Runtime)
- **Operating System Level Virtualization**
  - abstraction of the operating system which allows isolation and the creation of independent “sub-systems” (chroot, OpenVZ, BSD jails, Solaris Containers,...)



# Benefits of virtualization

- **Run multiple operating systems** at the same time
- **Isolation**
  - Systems with different requirements on the same physical machine
  - Security
- **Consolidation**: provide multiple services on a single system
  - More efficient usage of available resources
  - Reduction of hardware maintenance costs
- Keep running "**legacy**" applications
- **On-demand computing**
  - Dynamic resource allocation
  - Load balancing



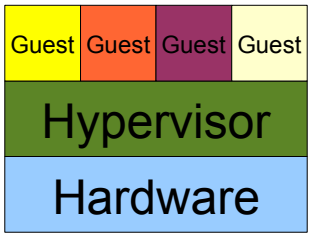
# Virtualization requirements (Popek and Goldberg 1974)

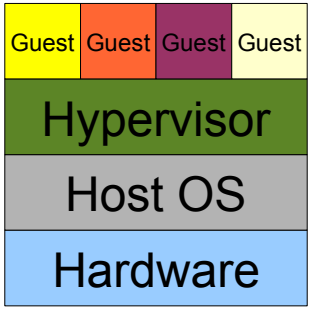
- **The efficiency property**
  - "All innocuous instructions are executed by the hardware directly, with no intervention at all on the part of the control program."
- **The resource control property**
  - "It must be impossible for that arbitrary program to affect the system resources, i.e. memory, available to it; the allocator of the control program is to be invoked upon any attempt."
- **The equivalence property**
  - "Any program K executing with a control program resident, with two possible exception [timing/resource availability], performs in a manner indistinguishable from the case when the control program did not exist and K had whatever freedom to access to privileged instructions that the programmer had intended."

Popek, G. J.; Goldberg, R. P. (July 1974). "Formal requirements for virtualizable third generation architectures". Communications of the ACM 17 (7): 412–421

# System level virtualization

- Hypervisor/Virtual Machine Monitor (VMM)
  - Software which manages access to the hardware
    - "Operating system for operating systems"
- **Type 1 "bare metal" (VMWare ESX, Citrix XenServer,...)**
  - The hypervisor runs directly on the hardware and has control over its resources
  - There is no "host" operating system
- **Type 2 "hosted" (VirtualBox, KVM, XEN,...)**
  - The hypervisor runs on top of the "host" operating system
  - All other operating systems (called "guest") are managed by the hypervisor





# Full virtualization: handling privileged instructions

- With **full virtualization** we can run a complete and unmodified operating system on the virtual machine
  - The guest operating system doesn't need to care that it's running on a virtual machine

- Some instructions, especially those run by the kernel of an operating system, require the CPU to be running in **privileged mode** (**kernel mode**), otherwise they generate an exception
  - ... can we just trap those exceptions and handle them in the VMM ?



# Full virtualization: technical problems with privileged instructions

- The x86 architecture was not designed from the ground up with virtualization in mind...
  - Some privileged instructions do not generate a trap when invoked from an unprivileged context
    - We get a **general protection fault** which kills the calling process (the VMM cannot do anything about it)



# Solutions

- **Full software emulation** (i.e. this not virtualization anymore) [Bochs]
  - Recompile/emulate every instruction (privileged/unprivileged)
  - SLOW!

- **Paravirtualization** [Xen]
  - Unprivileged instructions are executed natively on the CPU
  - Use modified guest operating systems which call the hypervisor instead of doing privileged calls
  - Good performance, especially for **drivers** which can be optimized for a virtualized hardware

- **Dynamic translation** [VirtualBox, VMWare]
  - Unprivileged instructions are executed natively on the CPU
  - Detect when code switches to privileged mode, analyze instructions, and rewrite privileged instructions (**trap-and-emulate**)

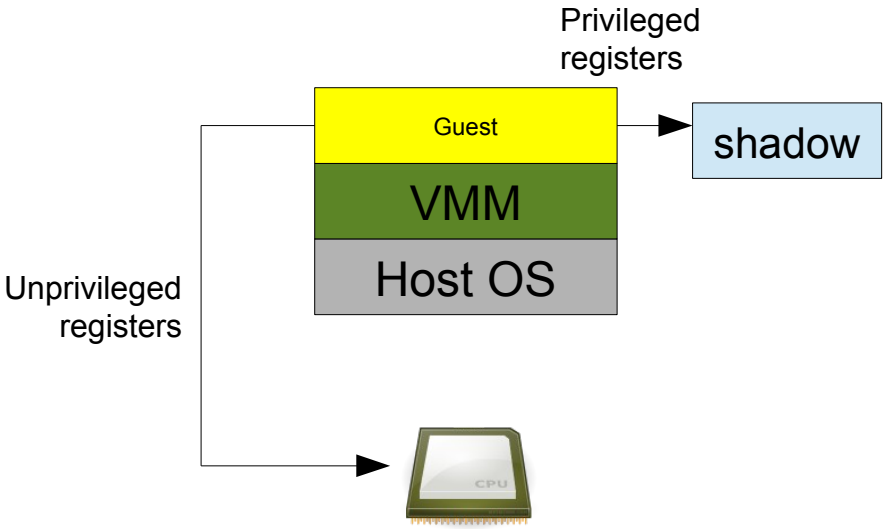


# Trap and emulate

- Trap and emulate privileged instructions is not enough to provide full virtualization, examples:
  - The guest operating can detect that it is not running in privileged mode just by reading a register (*code segment selector*), and can refuse to continue execution
  - Access to some **CPU registers** can be problematic: for example the page table base register or the processor status register
  - Access to some **in-memory data structures** can also be problematic: changes to the page table
  - Access to hardware devices needs to be controlled by the VMM

# Shadow registers

- The VMM can manage a ***shadow copy*** of “privileged” registers in order to provide the guest with a coerent view of the virtualized system
  - access to privileged registers is managed through dynamic translation
  - unprivileged registers can be natively used by the guest code

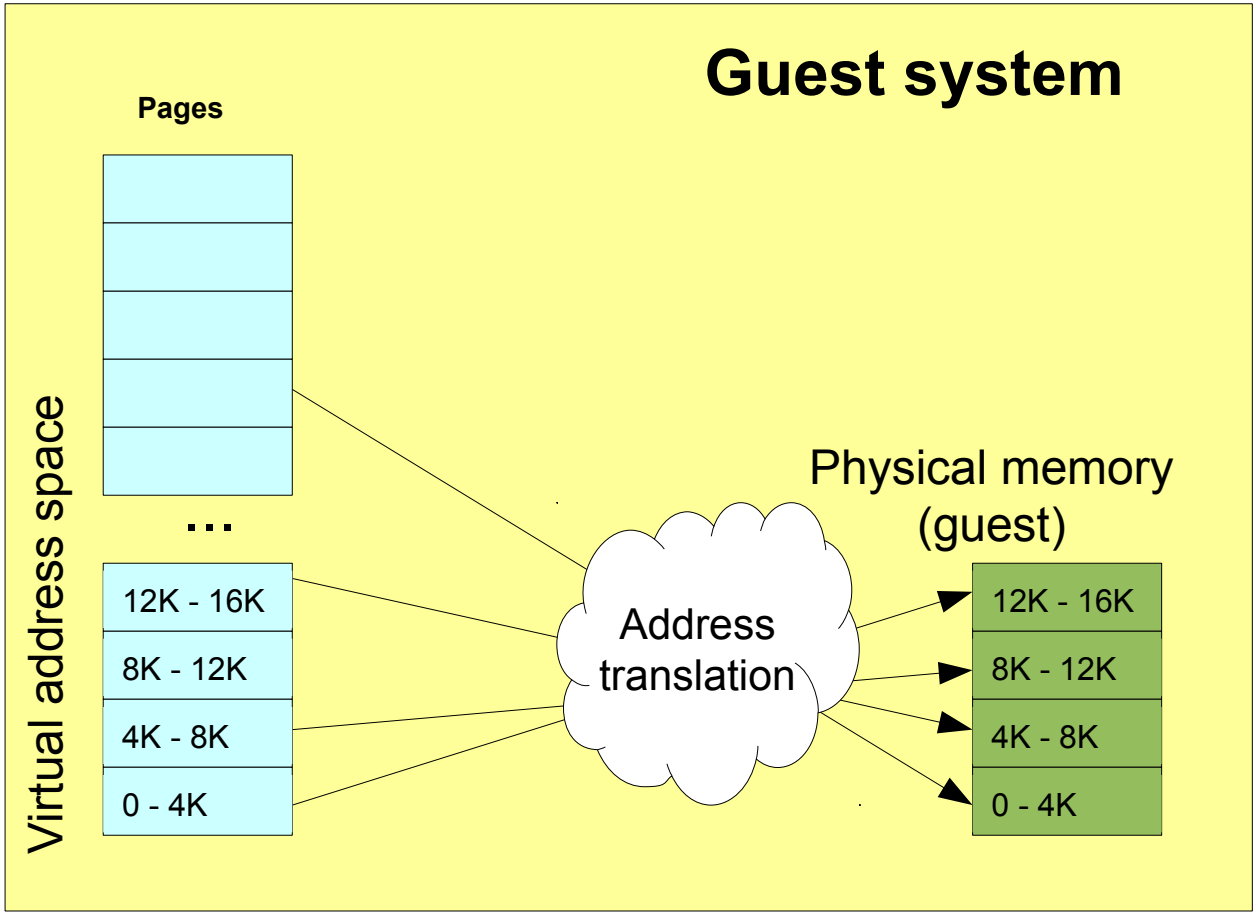




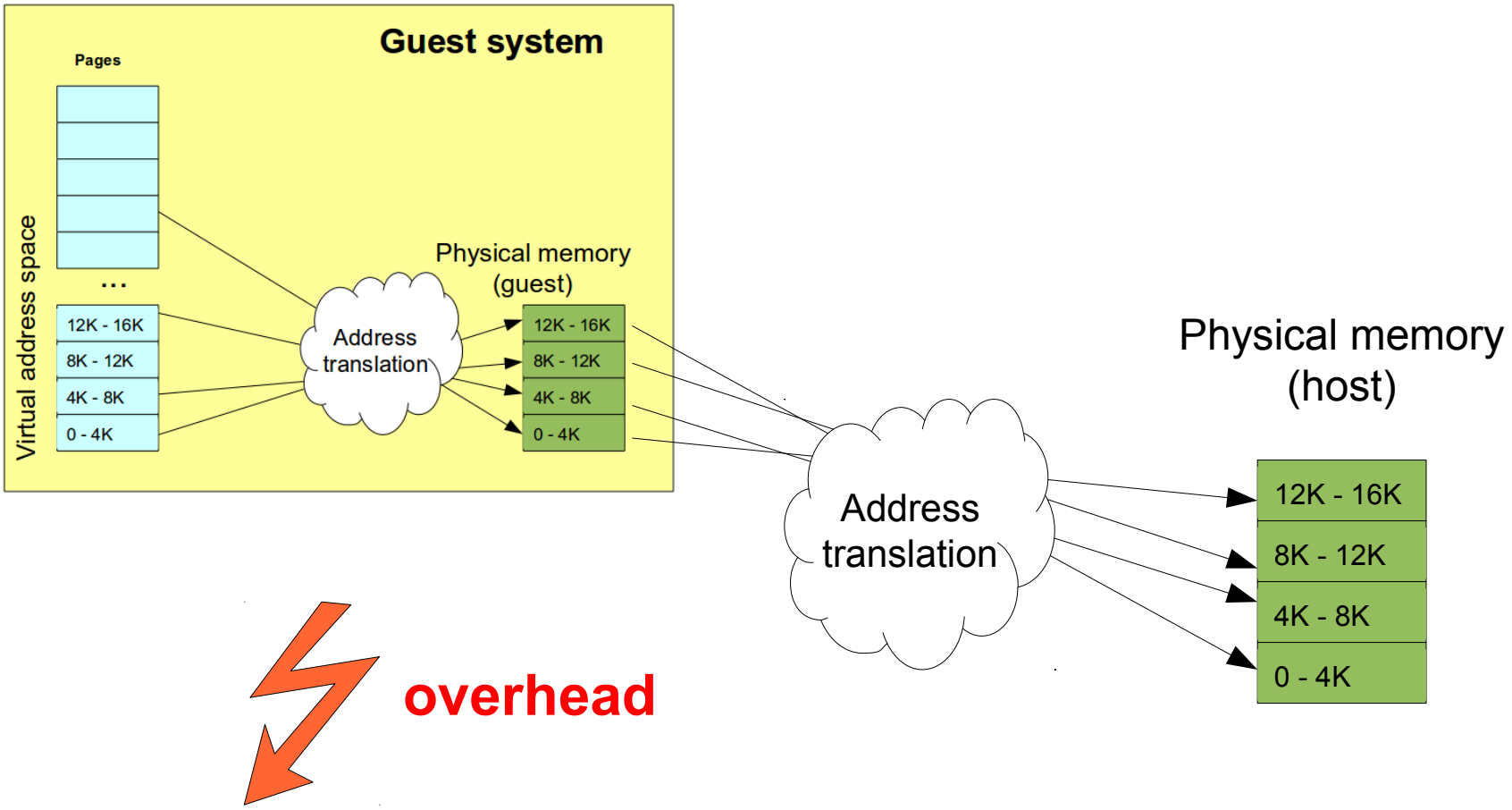
# In-memory data structures

- Each memory access can potentially read or modify privileged data structures
    - Intercepting every memory access is not convenient
    - Solution: **memory tracing**
        - Protect privileged memory areas (for example by setting them as read-only) and intercept (and manage) exceptions caused by the guest trying to write those regions
- ... and what about paging?

# Virtual memory: what the guest OS is seeing

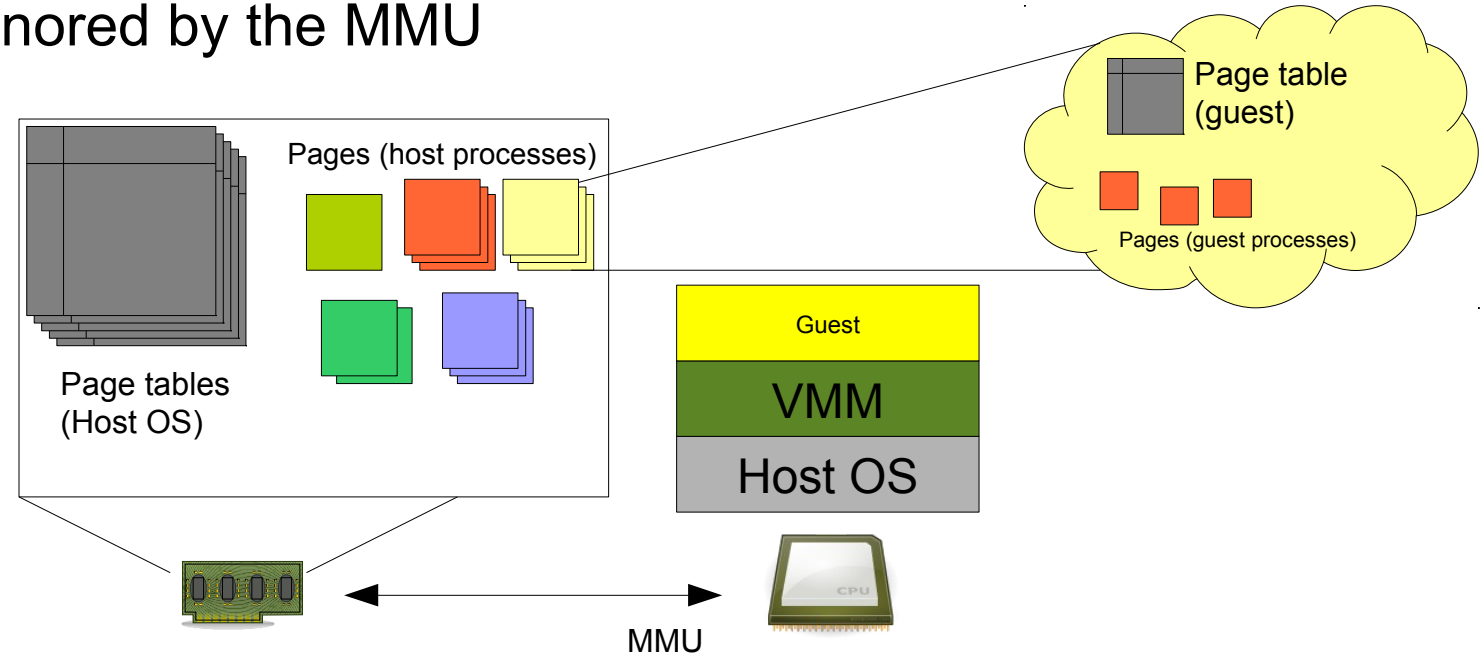


# Virtual memory: in reality



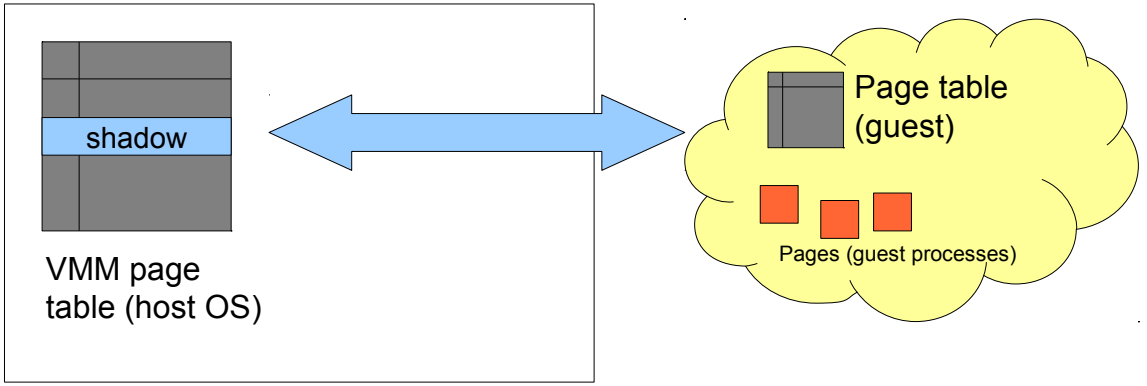
# Handling page tables (x86)

- The guest operating system manages page tables which are ignored by the MMU



# Handling page tables (x86)

- The VMM, which typically runs in privileged mode, can use a **shadow page table** which maps virtual addresses generated inside the guest into physical addresses




- When the guest is running the hardware MMU can access the shadow page table, and the TLB can speed-up translation
- Since the current page table is referenced using the CR3 register when the guest updates the register the VMM intercepts the instruction and updates the shadow table



# Handling page tables (x86): keep tables in sync

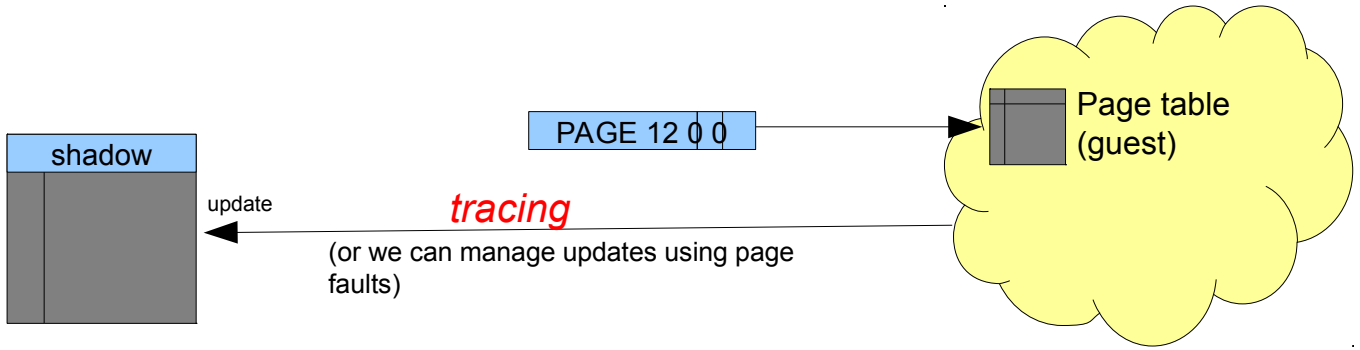
- To keep both tables synchronized we can use **memory tracing**

- Alternatively we can **exploit page faults**:
  - When the guest tries to access a virtual address corresponding to a page which is not yet mapped we get a page fault:
    - the VMM handles the page fault by looking into the guest page table:
      - if the requested page is also missing in the guest page table it forwards the fault to the guest operating system
      - otherwise, it updates the shadow table

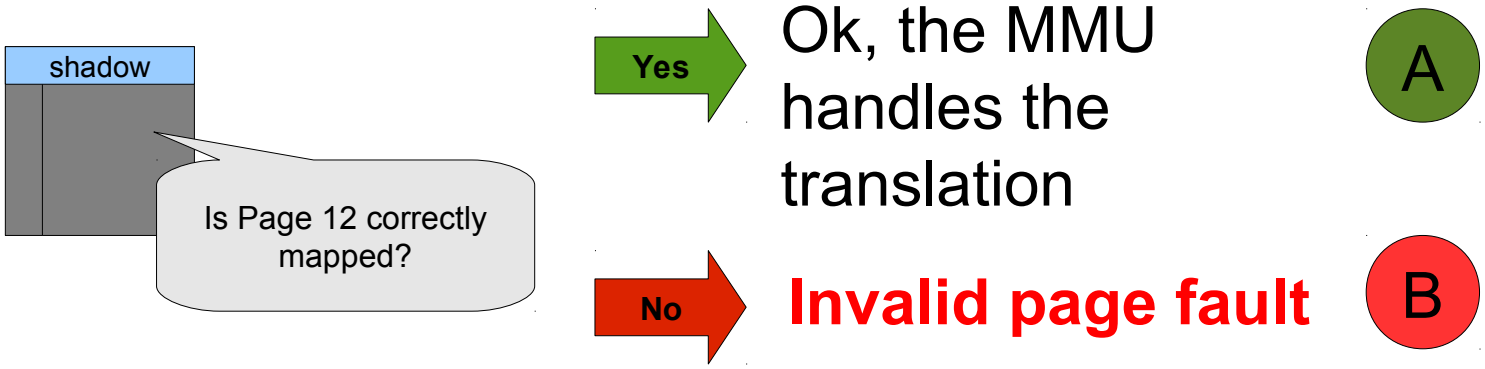
# Handling page tables (x86)

- Example:

  **Using memory tracing:** The guest creates a new entry in its page table




  **Exploiting page faults:** A guest process tries to access page 12



# Handling page tables (x86)

A

Is Page 12 in memory?

Yes → Translate and access

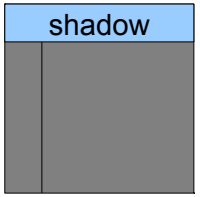
No → Major page fault → Load the page from disk

B **Invalid page fault**

Page table (guest)

Is Page 12 correctly mapped?

Yes → Create a new shadow entry




No → Invalid Page fault sent to the guest



# How to cleanup the shadow page table

- If we do not use memory tracing to update the shadow page table up we need a way to intercept changes that remove elements from the guest page table
  - on current systems when a page is removed (unmapped) from the page table the system also executes a TLB flush to invalidate the TLB.
  - The VMM can intercept such an instruction to synchronize the shadow page table



# Hardware support

- Recent CPUs implement hardware support for virtualization
  - The CPU can generate traps for privileged instructions (to avoid the need for frequent dynamic translation)
  - To speed-up address translation the CPU supports guest page tables (**nested page tables**) and can walk into guest page tables using **two dimensional page table walks**:
    - beside the CR3 register another register which points to the nested page table is available:
      - When there's a TLB miss, the MMU can walk into both tables to perform address translation

(http://developer.amd.com/wordpress/media/2012/10/NPT-WP-1%201-final-TM.pdf)

# Hardware devices

- To deal with hardware devices the main problem concerns the DMA (Direct Memory Access)
  - Since the guest operating system has no direct access to the hardware it cannot use DMA, because physical addresses (as seen by the guest) do not match with the real ones
- A recent solution is the introduction of a MMU for I/O devices, called **IOMMU**, which offers an address translation similar to the MMU but for I/O devices.