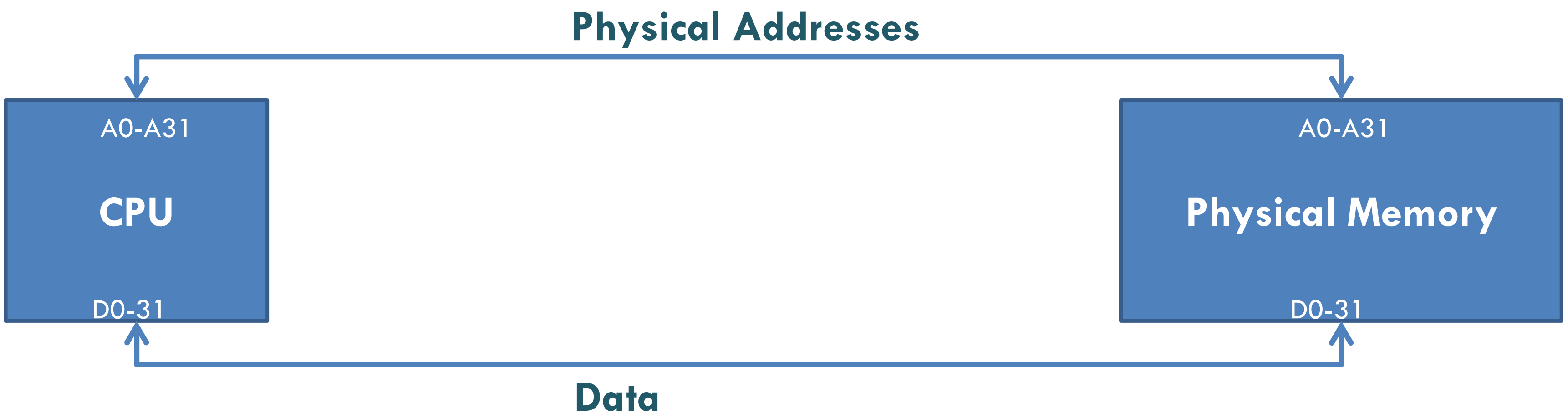
# COMPUTER ARCHITECTURE

Virtualization and Memory Hierarchy

# Contents

# Limits of Physical Addressing

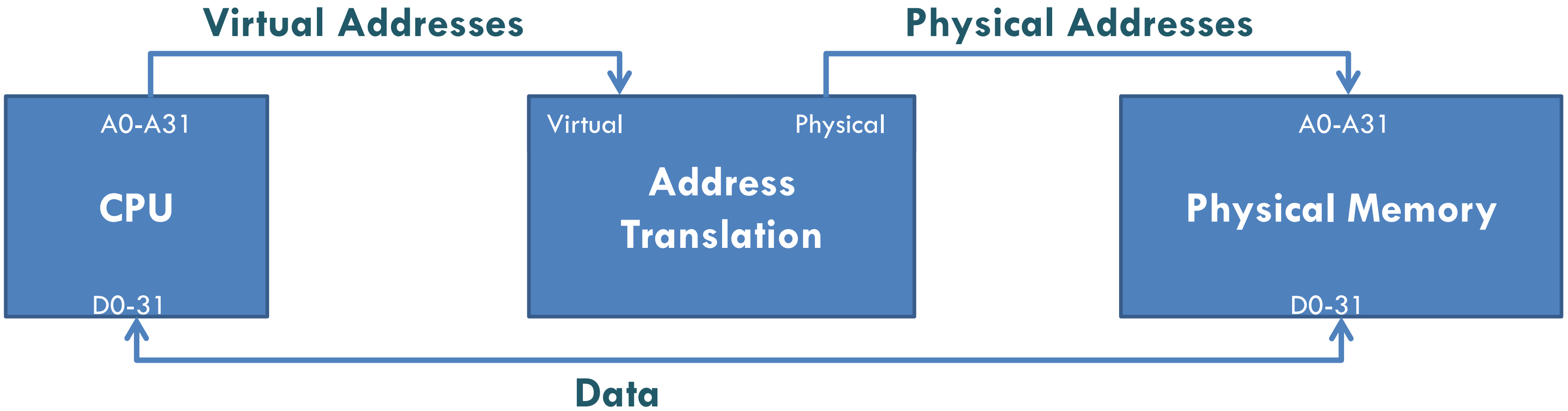





- All programs share a single address space.
  - Physical address space.

- There is no way to prevent a program to get access to a resource.

# Limits of Physical Addressing




- Programs executed in a normalized virtual address space.
- **Address Translation:**
  - Performed by hardware.
  - Managed by OS.
- **Supported features:**
  - Protection.
  - Translation.
  - Sharing.

# Advantages of virtual memory



- **Translation.**
  - Programs may have a consistent view of memory.
  - Reduced cost of multi-thread applications.
  - Only the working-set is needed in main memory.
  - Dynamic structures only use physical memory they really need (e.g. stack).
- **Protection.**
  - Allows to protect a process from others.
  - Attributes can be set at page-level.
    - Read-only, execution, …
  - Kernel data protected from programs.
  - Improves protection against malware.
- **Sharing.**
  - A page can be mapped to several processes.

# Differences with cache



- **Replacement:**
  - Cache: Hardware controlled.
  - VM: Software controlled.
- **Size:**
  - Cache size independent of address length.
  - VM size dependent of address length.

| Parameter | L1 cache | Virtual memory |
|---|---|---|
| Block size | 16 – 128 bytes | 4096 – 65,536 bytes |
| Hit time | 1 – 3 cycles | 100-200 cycles |
| Miss penalty | 8 – 200 cycles | $10^6 - 10^7$ cycles |
| Access time | 6 – 160 cycles | $8 \cdot 10^5 - 8 \cdot 10^6$ cycles |
| Transfer time | 2 – 40 cycles | $2 \cdot 10^5 - 2 \cdot 10^6$ cycles |
| Miss rate | 0.1 – 10 % | 0.00001 – 0.001% |
| Address mapping | 25 – 45 bits physical<br>14 – 20 bits cache | 32 – 64 bits virtual addr.<br>24 – 45 bits physical addr. |

# Contents



- Virtual memory.
- **Policies and strategies.**
- Page tables.
- Virtual machines.
- Requirements of virtual machines and ISA support.
- Virtual machines: Memory and I/O.
- Use case: Xen.
- Use case: Intel VT.

# Four question on memory hierarchy



- **Q1**: Where can a block be placed in the upper level?
  - Block placement.
- **Q2**: How is a block found in the upper level?
  - Block identification.
- **Q3**: Which block should be replaced on a miss?
  - Block replacement.
- **Q4**: What happens on a write?
  - Write strategy.

# Four question on memory hierarchy



- **Q1**: Where is a ***page*** placed in ***main memory***?
  - **Page** placement.
- **Q2**: How is a **page** found in **main memory**?
  - **Page** identification.
- **Q3**: Which **page** should be **replaced on a miss**?
  - **Page** replacement.
- **Q4**: What happens on a **write**?
  - Write strategy.

# Q1: Where is a page placed in main memory?



- A **page** can be placed in any **page frame** in main memory.
  - Fully associative mapping.
- Managed by **operating system**.
- **Goal**: Minimize miss rate.
  - Cannot do much with miss penalty.
  - Very high penalty due to slow magnetic disks.

# Q2: How is a page found in main memory?



- Keep in main memory a **page table** for every process.
  - Mapping table between page identifier and frame identifier.

- **Translation time reduction.**
  - **TLB**: *Translation Lookaside Buffer*.
  - Avoid accesses to page table in main memory.

# Q3: Which page should be replaced on a miss?



- **Replacement policy** defined by OS.
  - Typically LRU (Least-recently used).
- Architecture must supply support to OS.
  - **Use bit**: Activated when page is accessed.
    - Actually only on TLB miss (to reduce work).
  - Operating system periodically zeroes this bit.
    - Records values later.
    - Allows to determine pages that have been touched within an interval.

# Q4: What happens on a write?



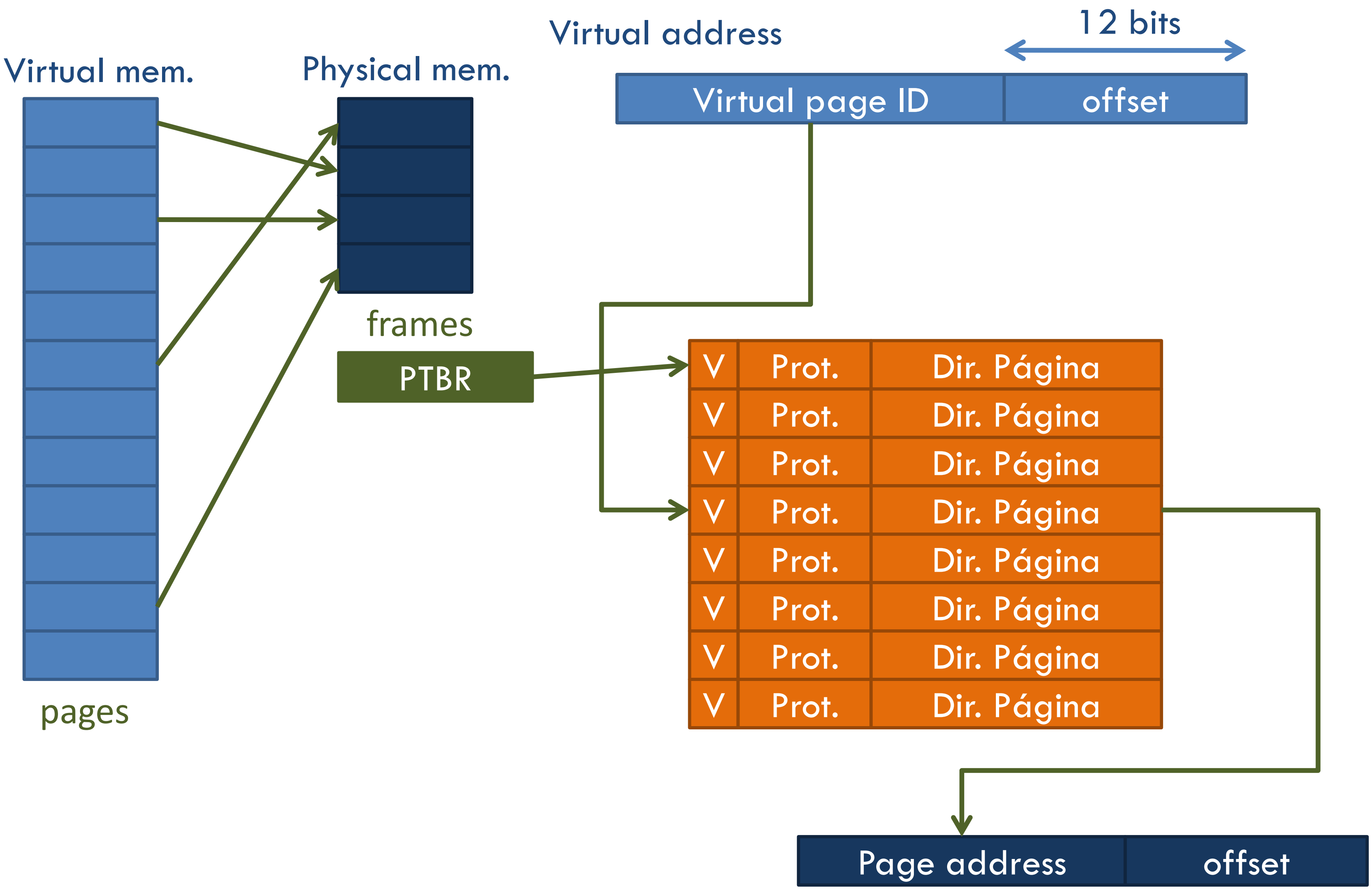
- Write policy is always **write-back**.
- No VM system with write-through ever built.
  - Don't be tempted!
- Disk write costs extremely high.
  - Disk writes minimization.
  - *Dirty bit* used to signal when a page has been modified.

# Contents

- Virtual memory.
- Policies and strategies.
- **Page tables.**
- Virtual machines.
- Requirements of virtual machines and ISA support.
- Virtual machines: Memory and I/O.
- Use case: Xen.
- Use case: Intel VT.

# Page table




Virtual mem.
Physical mem.
frames
pages
PTBR
Virtual address
12 bits
Virtual page ID
offset
V
Prot.
Dir. Página
V
Prot.
Dir. Página
V
Prot.
Dir. Página
V
Prot.
Dir. Página
V
Prot.
Dir. Página
V
Prot.
Dir. Página
V
Prot.
Dir. Página
V
Prot.
Dir. Página
Page address
offset

# Page table size

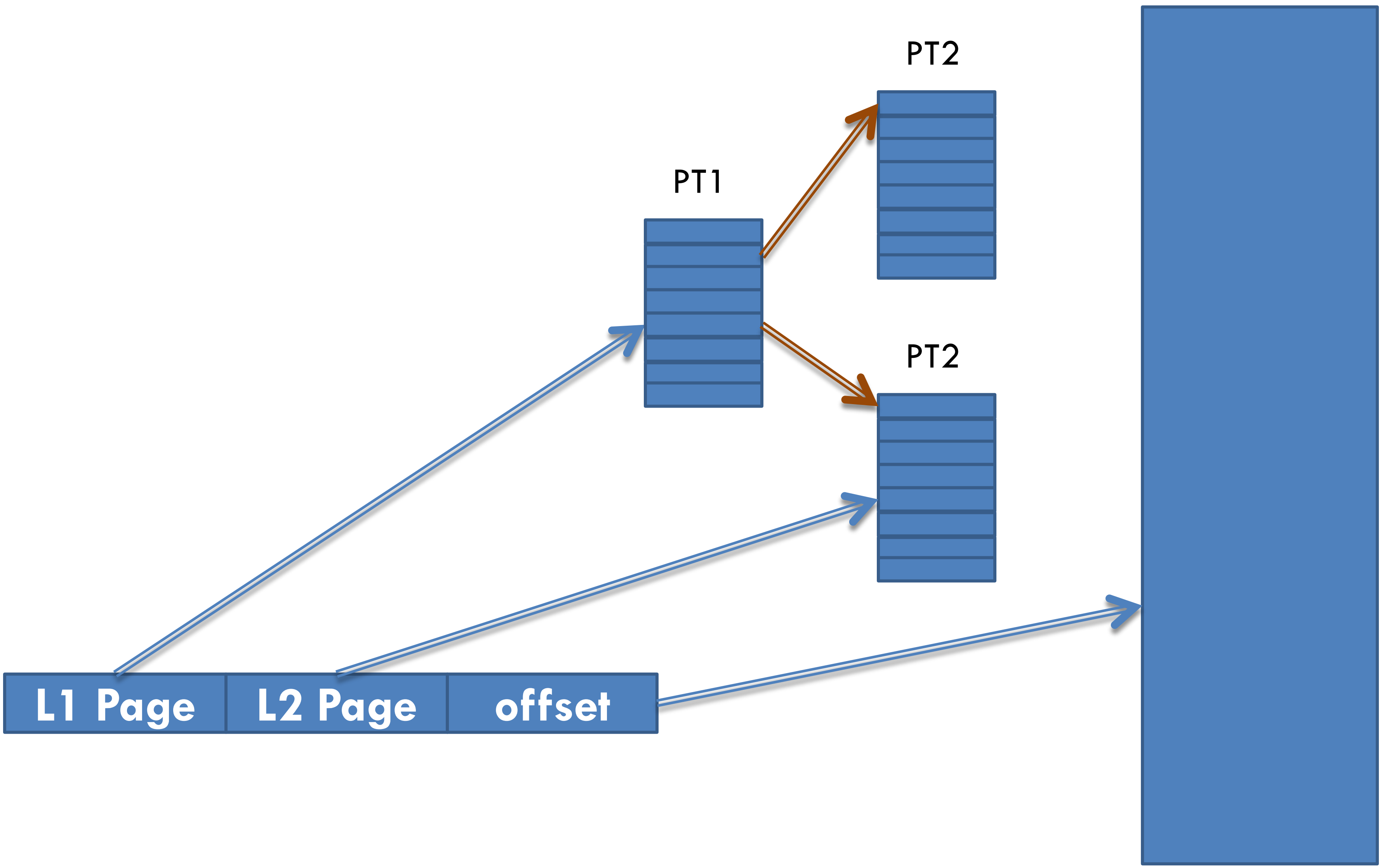


- Assuming 32 bits virtual addresses, 4 KB pages and 4 bytes per table-entry:
  - Table size: $(2^{32} / 2^{12}) \cdot 2^{2} = 2^{22} = 4$ MB

- **Alternatives**:
  - Multi-level page tables.
  - Inverted page tables.

- **Example**: IA-64
  - Offers both alternatives to OS developer.

# Multi-level page table



# TLB

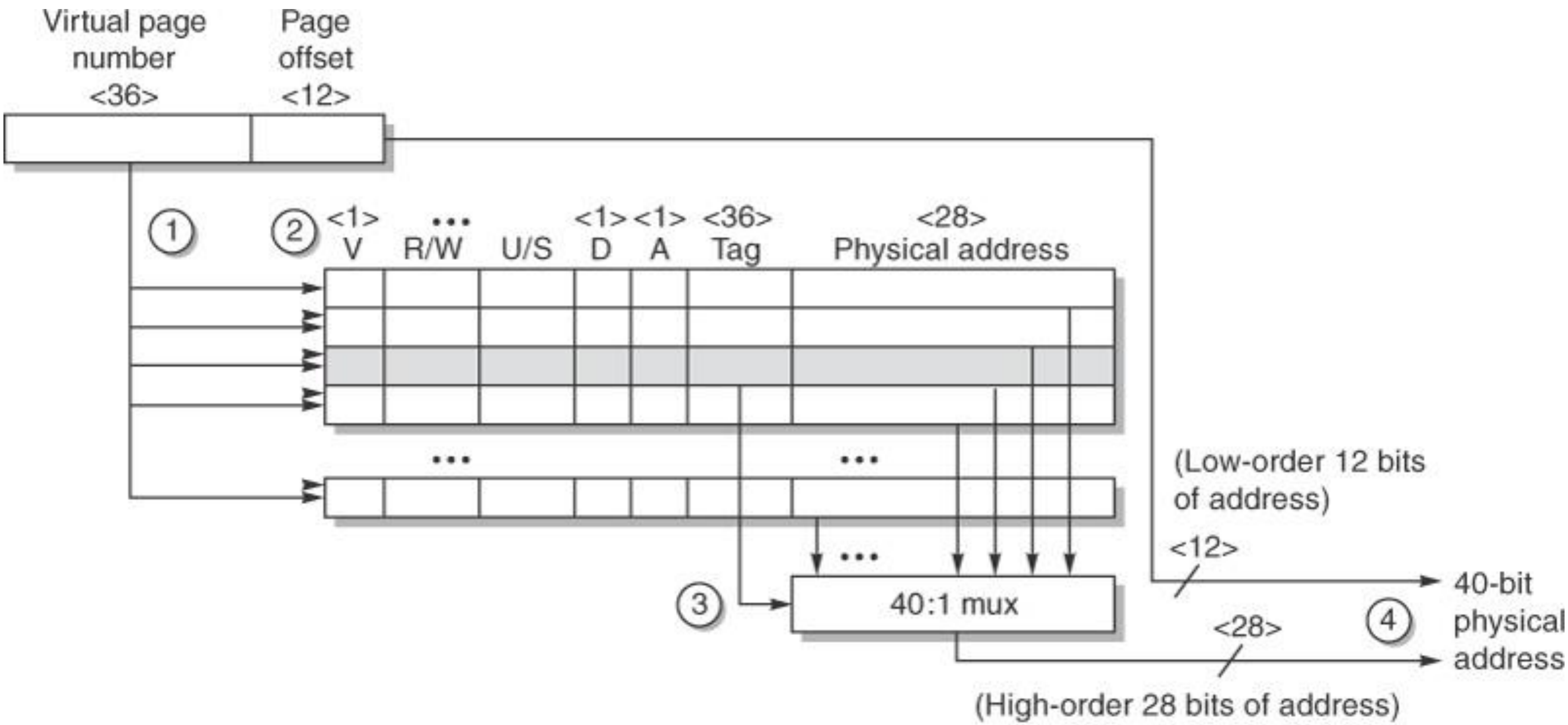


- Ideal case (no miss):
  - Every memory access requires two accesses.
    - Access to page table.
    - Access to memory.
  - Worse scenario in case of multi-level page tables.

- **Solution:**
  - Use translation cache to avoid page table accesses.
  - **Tag**: Portion of virtual address.
  - **Data**: Frame number, protection bits, validity bits and dirty bit.

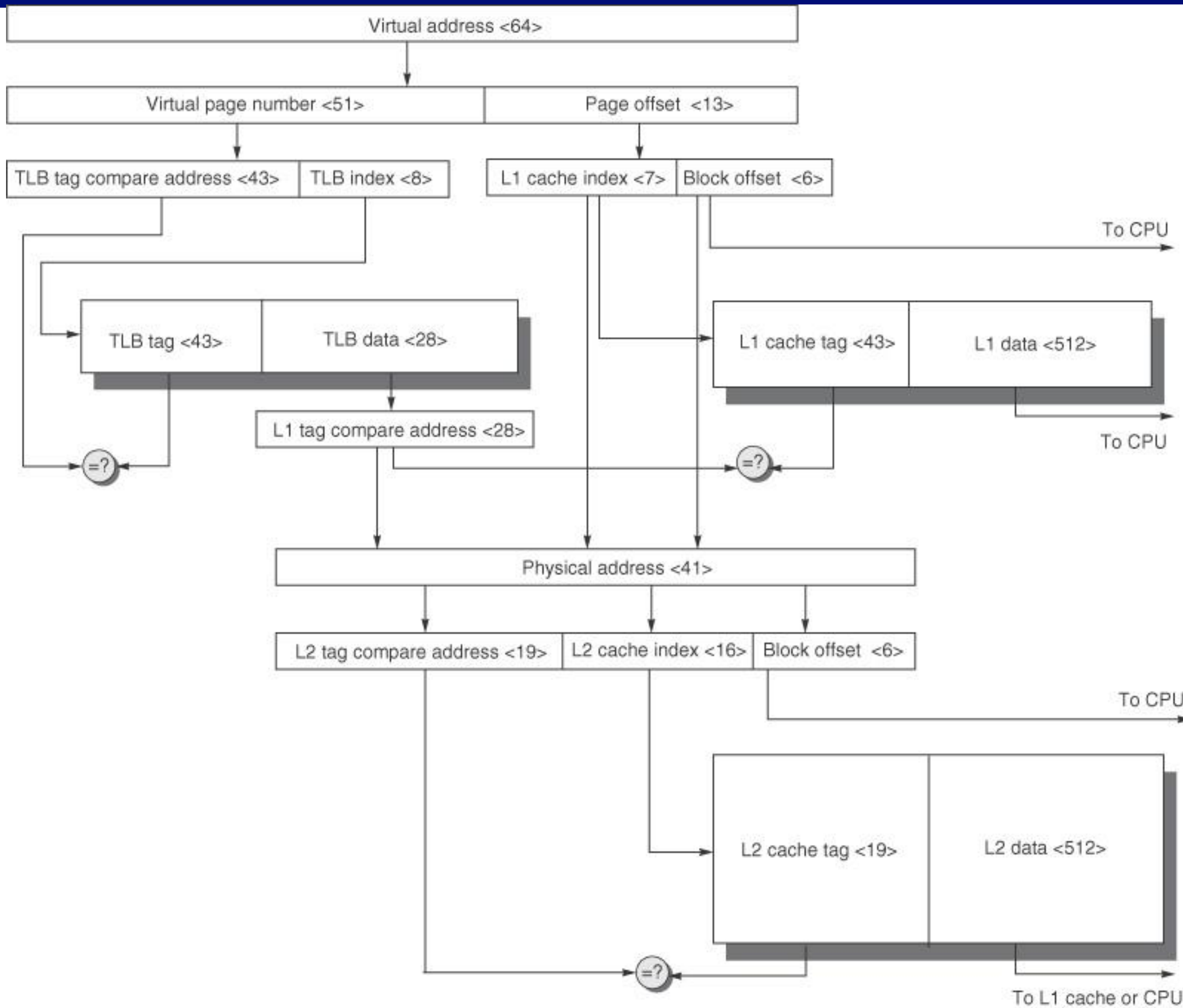
# Example: TLB Opteron





# A global view





# Contents



- Virtual memory.
- Policies and strategies.
- **Page tables.**
- Virtual machines.
- Requirements of virtual machines and ISA support.
- Virtual machines: Memory and I/O.
- Use case: Xen.
- Use case: Intel VT.

# Virtual machines

- Developed in late 60's.
  - Used since then in mainframe environments.
  - Ignored in single-user machines until late 90's.

- Popularity recovered due to:
  - Increasing importance of isolation and security in modern systems.
  - Security failures and reliability requirements in operating systems.
  - Sharing of a single computer by several unrelated users.
    - Datacenter, cloud, …
  - Dramatic increase in processors performance.
    - VMM'S overhead now acceptable.

Universidad Carlos III de Madrid
www.uc3m.es

# What is a virtual machine?

ARCOS

- *A virtual machine is taken to be an efficient, isolated duplicate of the real machine. We explain these notions through the ida of a virtual machine monitor (VMM)…*
- *… a VMM has threee essential characteristics.*
  - First, the VMM provides an environment for programs which is essentially identical with the original machine,
  - second, programs run in this environment show at worst only minor decreases in speed;
  - and last, the VMM is in complete control of system resources.

**Popek, G. y Goldberg, R.**
**Formal requirements for virtualizable third generation architectures.**
**Communications of the ACM, July 1974**

# What is a virtual machine?



- **General definition**: Any emulation method offering a standard software interface for the physical machine.
  - JVM? .NET?

- **System level virtual machines**: Offer a complete system environment at binary ISA level.
  - Usually assuming that VM ISA and hardware ISA are identical.
  - Examples:
    - IBM VM/370.
    - VMWare ESX Server.
    - Xen.

# What is a virtual machine?



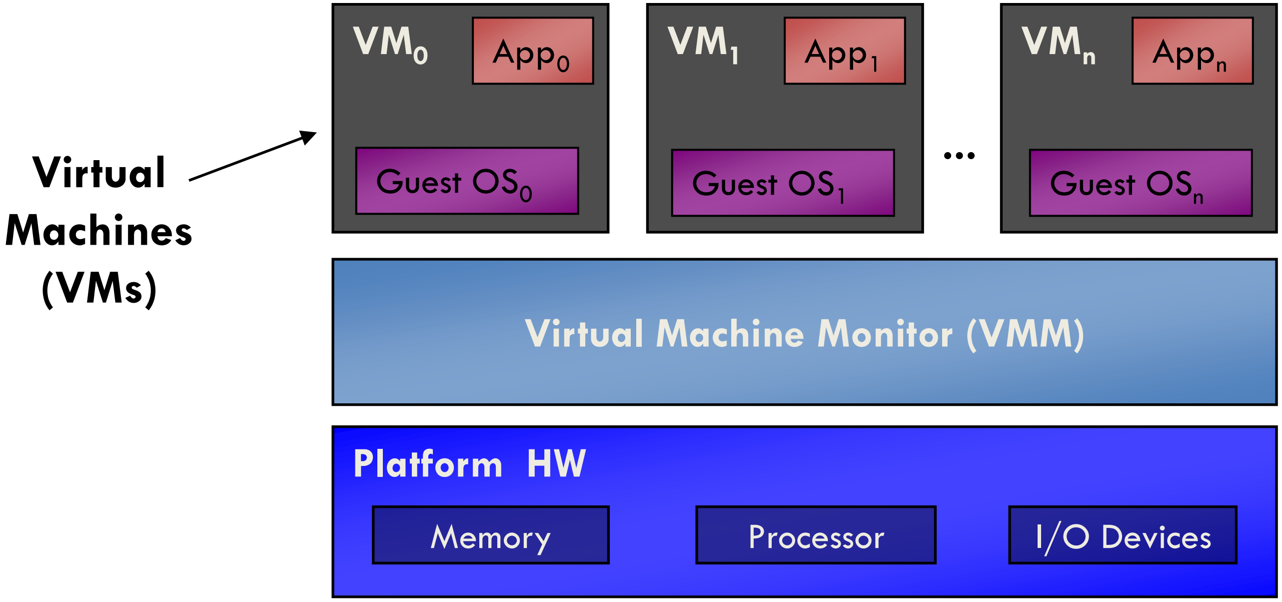
- Offer to users the illusion that they have a complete computer to use.
  - Including their own copy of OS.

- A computer runs several **virtual machines**.
  - May support several operating systems.
  - All OS's sharing the hardware.

- Terminology:
  - **Host**: Underlying hardware platform.
  - **Guest**: Virtual machines sharing resources.

# VM y VMM: Layers






- **VMM** → System Software Layer.
  - Allows running several VM on a single hardware.
  - Allows running unmodified applications.

# VMM: Virtual Machine Monitors



- Software supporting virtual machines
  - Virtual machine monitor or Hypervisor.
- VMM determines **mapping** between virtual and physical resources.
- **Alternatives** for physical resources sharing:
  - Time sharing.
  - Partitioning.
  - Software emulation.
- A **VMM** is smaller than a traditional OS.

# VMM overhead



- **Workload dependent.**

- User-level processor-bound programs:
  - Examples: SPEC.
  - Overhead: 0.
  - Seldom invocations to OS.

- I/O intensive programs → OS intensive:
  - Many system calls → Privileged instructions.
  - May lead to much virtualization overhead.

- I/O intensive and I/O bound programs:
  - Low processor utilization.
  - Virtualization may be hidden.
  - Low virtualization overhead.

# Other uses of VMs (besides protection)



- Software management.
  - VM offers an abstraction allowing to run a complete software stack.
    - Old operating systems (DOS? Windows XP?, ...?)
  - Typical deployment:
    - VM running legacy OS + stable OS + testing new OS.
- Hardware management.
  - VM allows to run separate software stacks on top a a single hardware platform.
    - Server consolidation.
    - Independence → Higher reliability.
  - Migrating running VMs.
    - Load balancing..
    - Hardware evacuation due to failures.

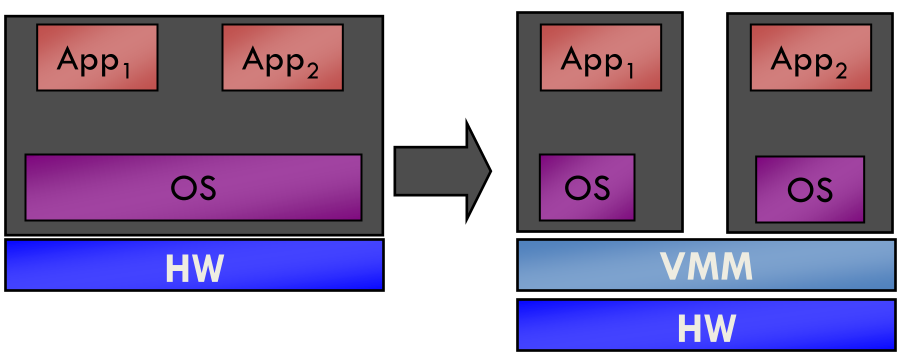
Cloud-based servers usually supported by virtualization

e.g. Amazon

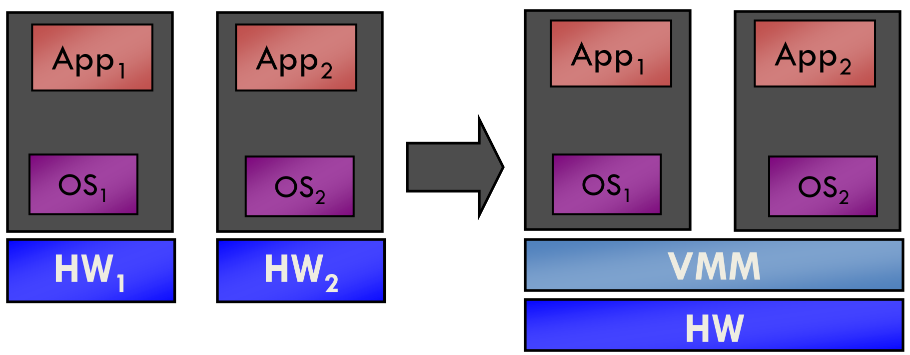
# Uses of virtualization



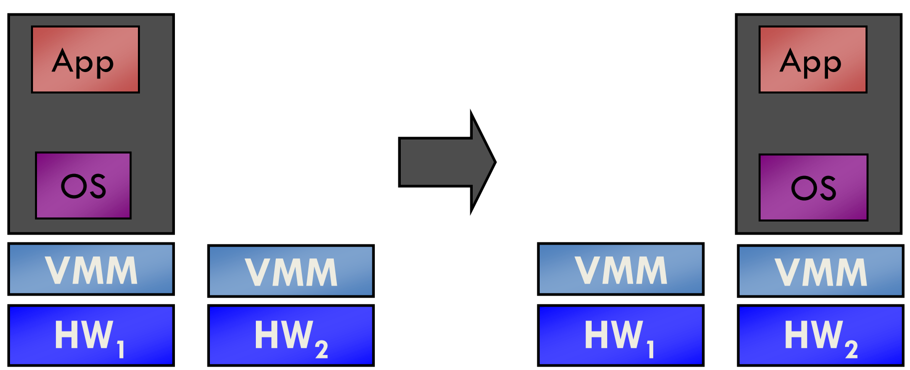
## Isolation




## Consolidation




## Migration



# Contents



- Virtual memory.
- Policies and strategies.
- **Page tables.**
- Virtual machines.
- Requirements of virtual machines and ISA support.
- Virtual machines: Memory and I/O.
- Use case: Xen.
- Use case: Intel VT.

# VMM requirements (I)



- A **VMM**:
  - Offers a software interface to guest software.
  - Isolates a guest state from the rest.
  - Protects itself from guests.

- **Guest software** should behave as if there was no VMM, except for
  - Performance dependent behavior.
  - Fixed resources limitations which are shared among several VMMs.

# VMM requirements (and II)



- **Guest software must not be able to modify directly real resources allocation.**

- VMM must control everything, even if it is used by guests.
  - Access to privleged state, address translation, I/O, exceptions, interruptions, …

- VMM must run at a higher privileged level than guests.
  - Execution of any privileged instruction by VMM.

- Requirements of VMM (*equivalent to requirements for virtual memory*)
  - A minimum of two processor modes.
  - Privileged instruction subset only in privileged mode.
    - Trap is executed in user mode.

# ISA support for Virtual Machines



- If VM considered in ISA design, it is easy to reduce instructions to be run by VMM and emulation time.
  - Most desktop ISA designed before VM emergence.
- VMM must ensure that guest only interacts with virtual resources.
  - Guest OS run in user mode.
  - HW access tries lead to trap.
- If ISA is not VM aware, then VMM must intercept problematic instructions.
  - Introduction of virtual resources.

# Contents

# Impact on virtual memory

- Every guest manages **virtual memory**.
  - Virtual memory virtualization?

- VMM makes distinction between **real memory** and **physical memory**.
  - **Real memory**: Intermediate layer between virtual memory and physical memory.
  - **Guest**: Maps virtual to real memory.
  - **VMM**: Maps real memory to physical memory.

- To reduce indirection levels, VMM keeps a **shadow page table**.
  - Mapping from virtual to physical memory.
  - VMM must capture changes in page table and pointer to page table.

# ISA support for virtual machines and virtual memory



- IBM 370 (1970's) additional level of indirection managed by VMM.
  - Eliminates need for a shadow page table.

- **TLB virtualization**:
  - VMM manages TLB and keeps copies of each guest TLB.
  - TLB access generate traps.
  - TLB with process identifiers simplify management allowing entries from multiple VMMs at the same time.

# Impact of I/O on virtual machines



- Most complex part of **virtualization**.
  - Increasing number of I/O devices.
  - Increasing diversity of I/O devices.
  - Sharing devices among VMs.
  - Support of great variety of drivers.
- General part of driver left in guest.
  - Specific part in VMM.
- Device dependent method.
  - **Disks**: Partitioned by VMM to create virtual disks.
  - **Network interfaces**: Multiplexed over time.
    - VMM manages virtual network addresses.

# Contents

# Impure virtualization



- Solution for non virtualizable architectures and to reduce performance problems.

- **Approaches:**
  - **Paravirtualization**: *Port* guest OS code to modified ISA.
    - Development effort.
    - Need to be repeated for every OS.
    - Source code availability.
  - **Binary translation**: *Replace* non-virtualizable instructions by emulation code or call to VMM.
    - Does not require source code.
    - Some emulations possible at user space.

# Example: XEN

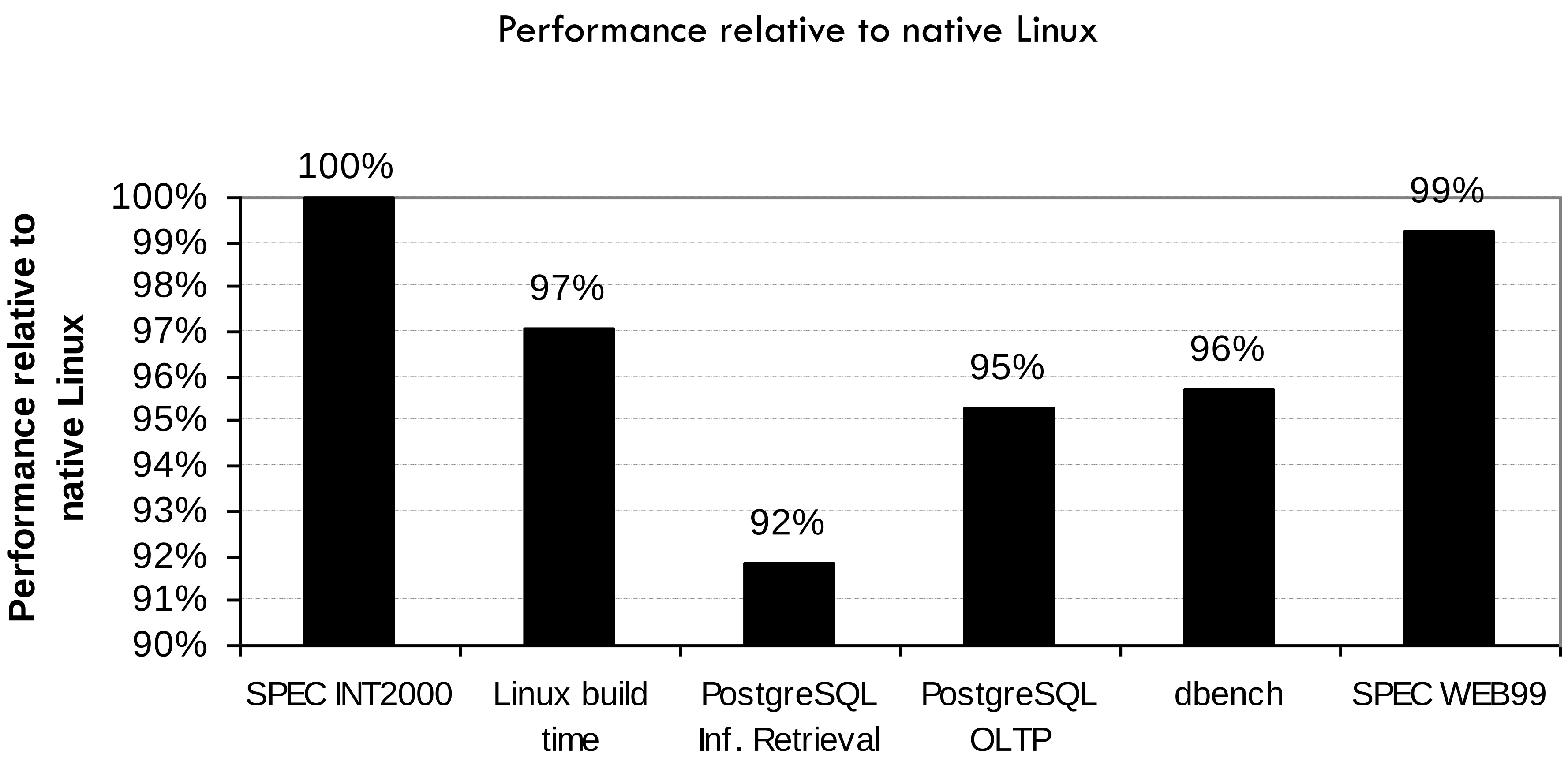
- **Xen**: Open source VMM for x86.
- **Strategy**: Paravirtualization.
  - Small modifications to OS
- Examples paravirtualization:
  - Avoid TLB flush when VMM invoked.
    - Xen mapped into upper 64 MB in every VM.
  - Allow guests to allocate pages.
    - Check protection restrictions are not violated.
  - Protection between programs and guest OS.
    - Use protection levels from **x86**:
      - Xen (0), Guest (1), Programs (3).

# Changes in Xen



- Changes in Linux → 3,000 LOC.
  - 1% of the x86 specific code.

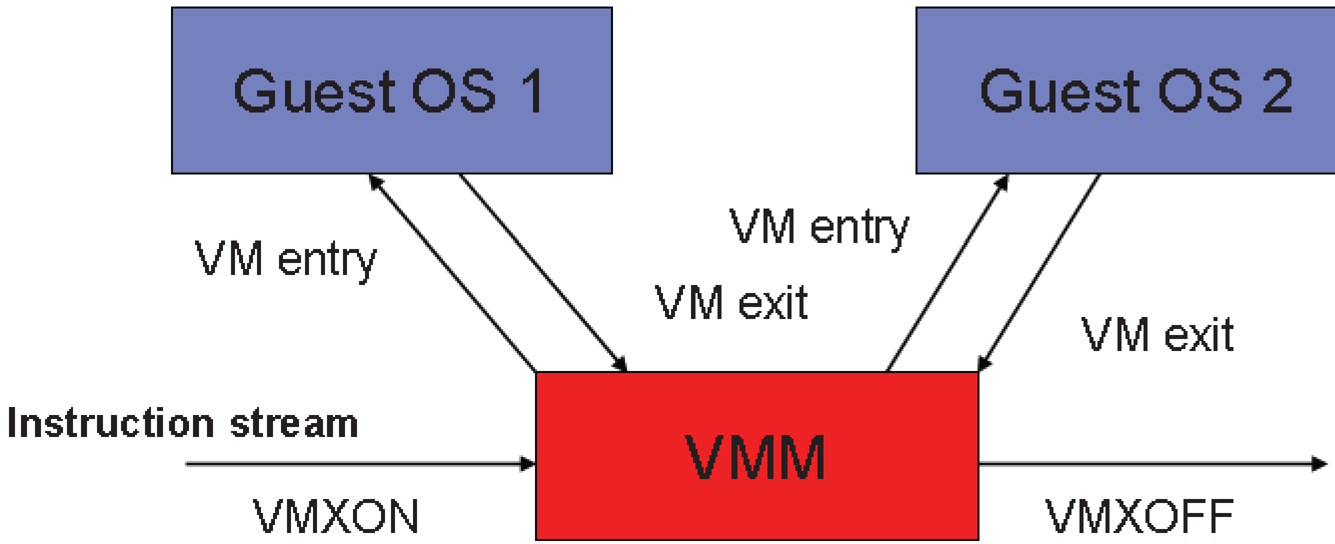
# Xen: Performance



# Contents

# Intel Virtualization Technology

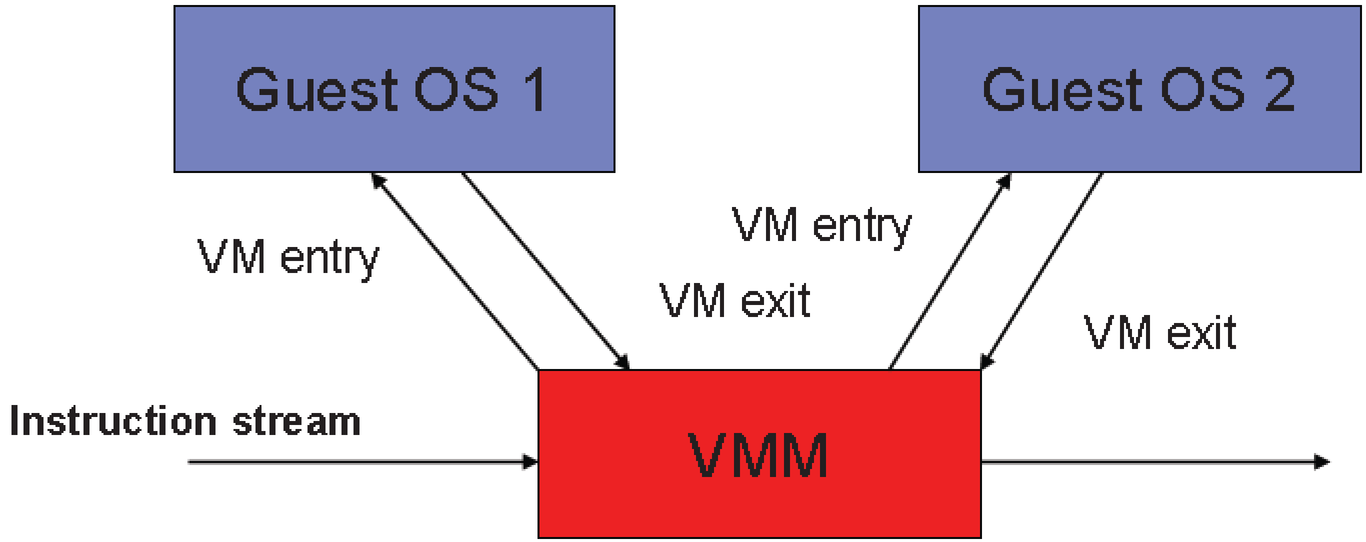
- Adds new instructions: VMXON, VMXOFF, VMLAUNCH, VMRESUME, …



# AMD Secure Virtual Machine



- Adds new instructions: VMRUN, VMCALL, …



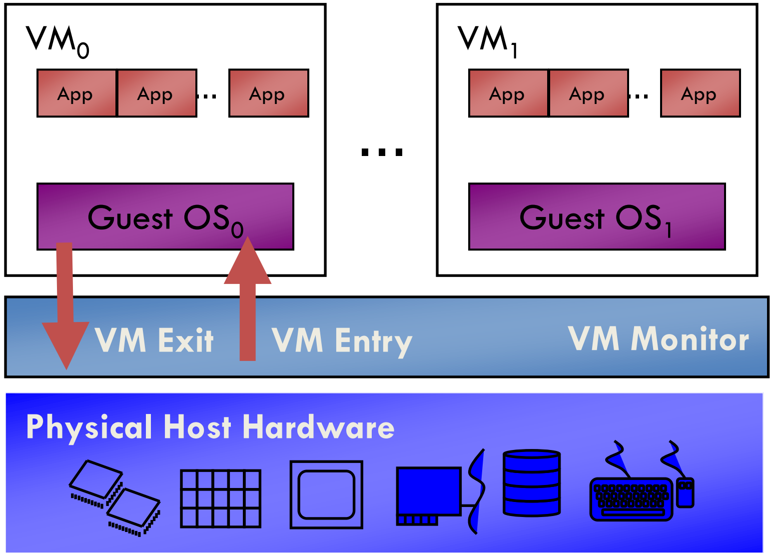
# Operation modes

- **VMX root:**
  - **Fully privileged.**
  - Designed to be used by VMMs.
- **VMX non-root:**
  - **Non privileged.**
  - Designed to by used by guest software.

# Entering and exiting virtual machines

- **VM Entry**
  - Transition from VMM to Guest.
  - Entry to non-root mode.
  - Loads guest mode.
  - **VMLAUNCH** instruction used for initial entry.
  - **VMRESUME** instruction used for subsequent entries.

- **VM Exit**
  - Enter to root mode.
  - Saves guest state.
  - Loads state of VMM.
  - **VMEXIT** instruction used to transition to VMM.
  - Additional instructions and events may cause **VMEXIT**.



# Benefits



- Reduces OS **dependency**.
  - Eliminates need for binary translation.
  - Eases support for legacy OSs.
- Improves **robustness**.
  - Eliminates the need for complex techniques.
  - VMM smaller and simpler.
- Improves **performance**.
  - Less transitions to VMM.

# Reference



- **Computer Architecture. A Quantitative Approach. Fifth Edition.**
  Hennessy y Patterson.
  Sections: B.4, 2.4

- Exercises: B.12, B.13, B.14, 2.20, 2.21, 2.22, 2.23