Heavy virtualization

UNIVERSITÉ CÔTE D'AZUR

Outline

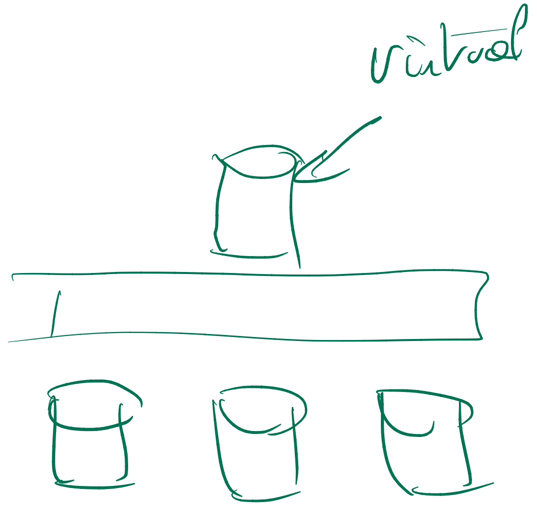
- Virtualization techniques and hypervisor
- Virtualization without architectural support
  - Xen & VMWare (good old days…)
- The Popek/Goldberg Theorem
- Virtualization with architectural support
  -

Introduction - 1

- Virtualizations boils down to separating a service from the underlying layer (hardware)
- Old technique pioneered in the 60s
  - Used in mainframes
  - But also directly in operating systems

Introduction - 2

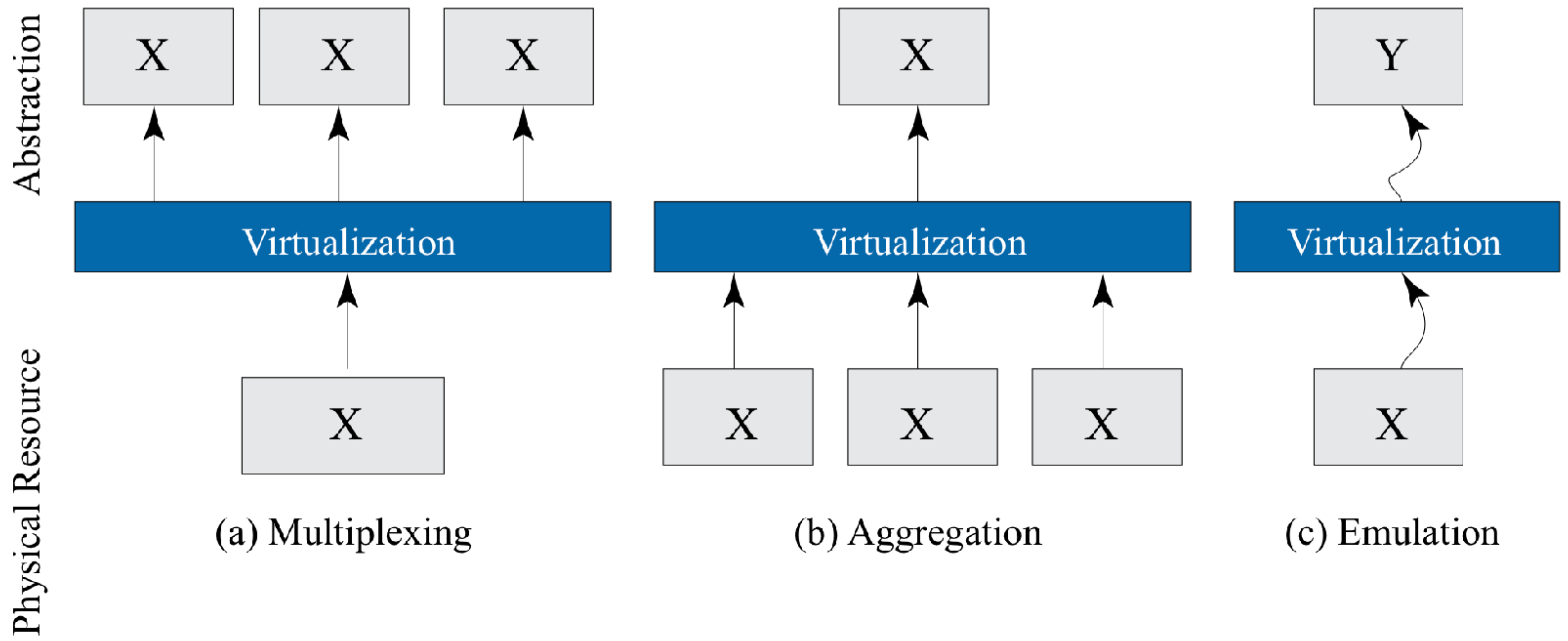
- Two fundamental principles
  - Layering : presentation of a single abstraction
  - Enforced modularity :
    - The exposed higher-level interface is the same as the lower-level one
    - The higher-level software cannot escape the modularity
- Virtualization is not limited to Virtual Machines
- Example 1 : Redundant Array of Inexpensive Disk (RAID)
  - Aggregation of multiple disk as a single (virtual) disk



Introduction- 3

- Example 2 : Memory management in Operating Systems
- Each process has its own (complete) address space
    - Physical memory is shared among them
    - No direct access to RAM, always go through a Memory Management Unit (MMU)
    - More on that later…

Virtualization techniques



(a) Multiplexing (b) Aggregation (c) Emulation

- Multiplexing : OS with physical memory
- Aggregation : RAID
- Emulation : Emulate a disk in RAM (e.g. /proc in Linux)

Hypervisor

Key features (Popek and Goldberg, 1974)

- A software providing virtualization service is called a *Hypervisor*
  - Virtualized machines are sometimes called guests
- Must have 3 key features (Pokep and al.)
  - Equivalent execution
    - Software running in a virtualized environment should have an identical execution to running in a native one, except for resource usage and timing

$\hookrightarrow$ slower
  - Safety
    - Virtual machines are isolated from each other as well as from the hypervisor
  - Performance
    - VM should suffer only from minor performance decrease. Most instructions should be executed directly on the physical CPU.
    - Difference between hypervisors and simulators

UNIVERSITÉ CÔTE D'AZUR

Instructions and CPU

- A program is made of instructions
  - Executed by the CPU
- Not all instructions are equals
  - Some instructions are free to use (computation)
  - Some instructions are only accessible to the Kernel (management of resources)
- 2 set of instructions
  - Unprivileged : can be called by any process
  - Privileged : only callable by the kernel
- Enforced by the CPU itself

UNIVERSITÉ CÔTE D'AZUR

x86 rings

- Standard x86 architecture has 4 privilege levels
  - Rings 0-3
- Each ring restricts callable instructions
  - When calling a forbidden instruction, CPU trap
- Processes are executed in Ring 3
- OS is executed in Ring 0
  - Has access to all instructions
- Others rings are unused.

System calls - 1

- How can an unprivileged process execute privileged code
  - Need a mechanism to temporary change ring level
  - It’s the system calls
- The OS provides a list of system calls
  - Specified with numbers
- Different ways to call them
- Software interrupt (*legacy*)
  - Unprivileged process write the system call number in EAX CPU register
  - Process then call software interrupt 0x80
  - The OS is executed and looks into EAX to continue

System calls - 2

- On x86 in 32/64 bits modes, 0x80 is deprecated.
- 32 bits mode: use of 2 new instructions + EAX
  - Systenter : quickly switch to ring 0
  - Systexit : quickly switch back to ring 3
- In 64 bits mode
  - Syscall
  - Sysret

↳ register

Trap

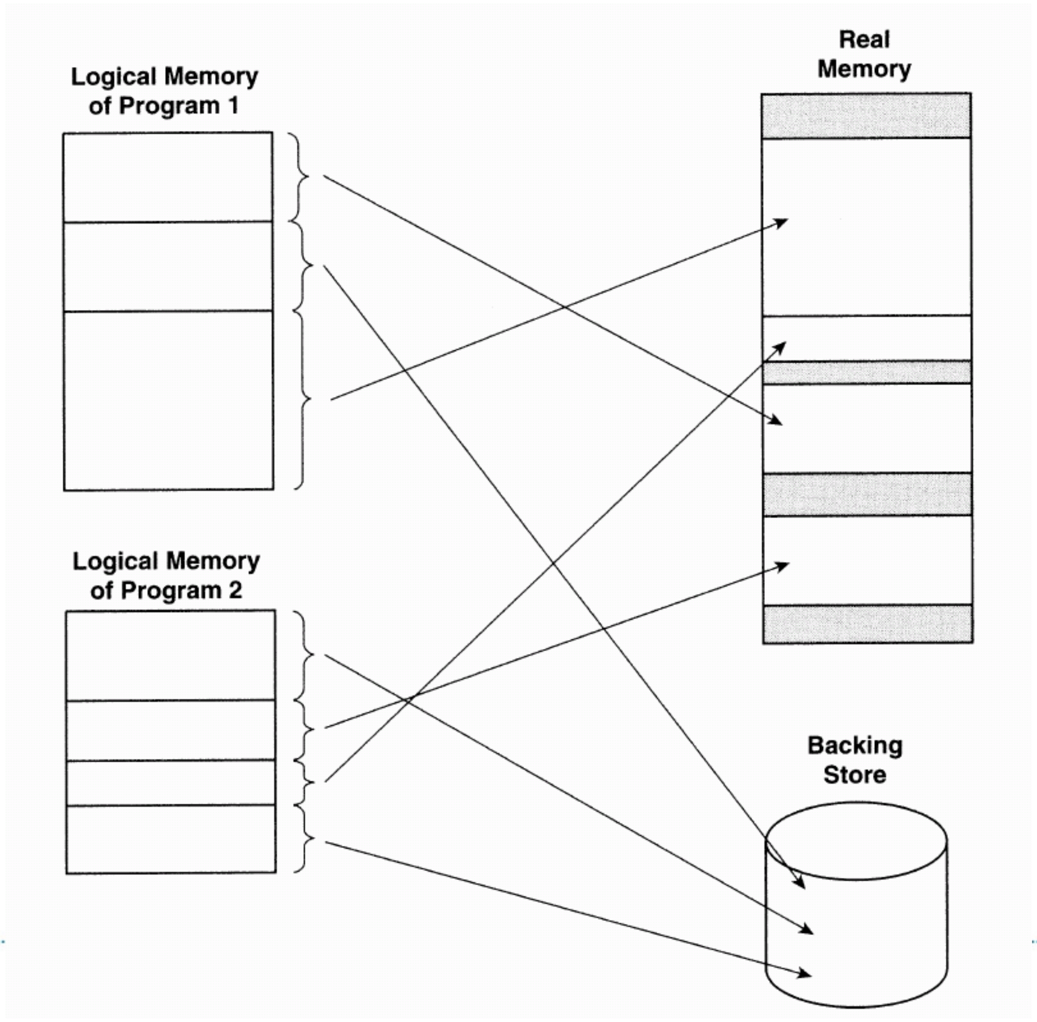
- A form of hardware interrupt
- Raised by the CPU when something is wrong
  - Invalid memory access
- At boot time, the OS registers methods (memory address) to be executed in case of trap
- Allows for handling of errors
  - Ex : invalid memory access by a process => trap to OS => kill the process (segfault anyone?)

Trap-and-emulate virtualization - 1

- Allows for virtualization on “classical” x86 CPUs
- Slightly changes ring levels
  - Hypervisor in Ring 0 (all access)
  - OS moved to less privileged rings (aka Ring deprivileging)
- What happens when guest OS calls a ring 0 instruction ?
  - CPU error and trap
  - Hypervisor is executed and handles the instruction

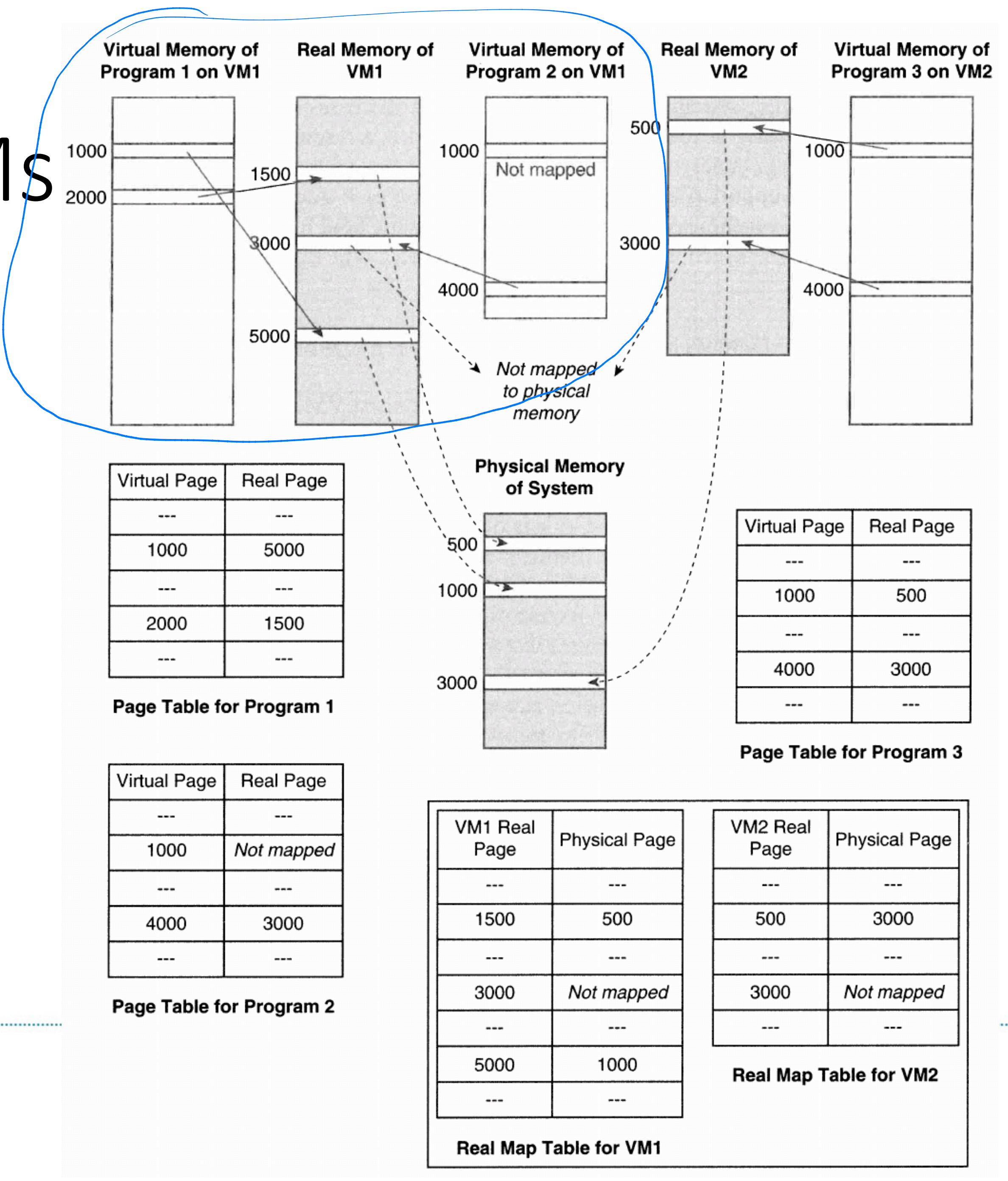
Before	After
3- Apps	3 - Apps
	? OS
0- OS	0 - Hypervisor

Classical Virtual Memory



Virtual Memory with VMs

- 2 indirections level
  - Guest Program -> Guest OS
  - Guest OS -> host



Virtual Memory of Program 1 on VM1
Real Memory of VM1
Virtual Memory of Program 2 on VM1
Real Memory of VM2
Virtual Memory of Program 3 on VM2
1000
2000
1500
3000
5000
1000
Not mapped
4000
500
3000
1000
4000
Not mapped to physical memory
Physical Memory of System
500
1000
3000

Virtual Page	Real Page
---	---
1000	5000
---	---
2000	1500
---	---

Page Table for Program 1

Virtual Page	Real Page
---	---
1000	*Not mapped*
---	---
4000	3000
---	---

Page Table for Program 2

Virtual Page	Real Page
---	---
1000	500
---	---
4000	3000
---	---

Page Table for Program 3

VM1 Real Page	Physical Page
---	---
1500	500
---	---
3000	*Not mapped*
---	---
5000	1000
---	---

Real Map Table for VM1

VM2 Real Page	Physical Page
---	---
500	3000
---	---
3000	*Not mapped*
---	---

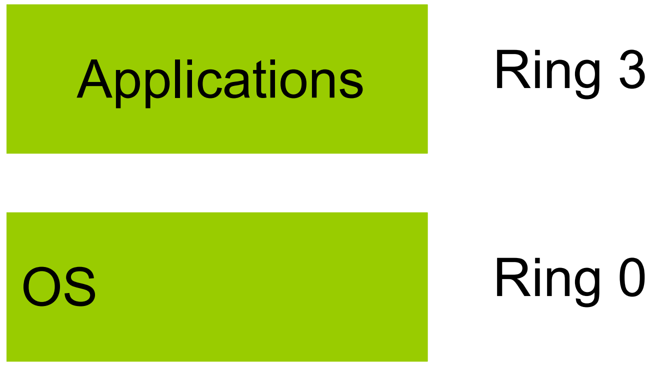
Real Map Table for VM2

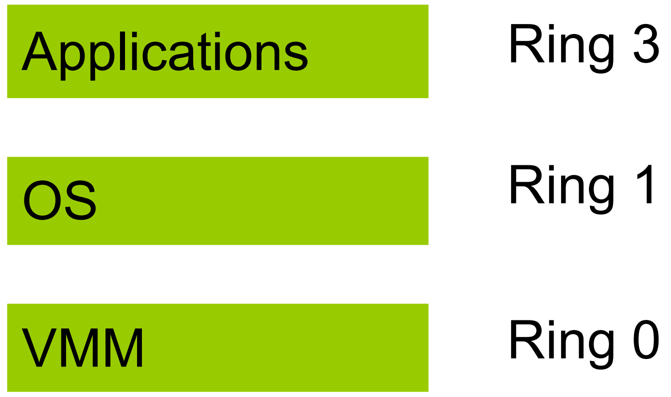
Virtualization of CPU, RAM and I/O

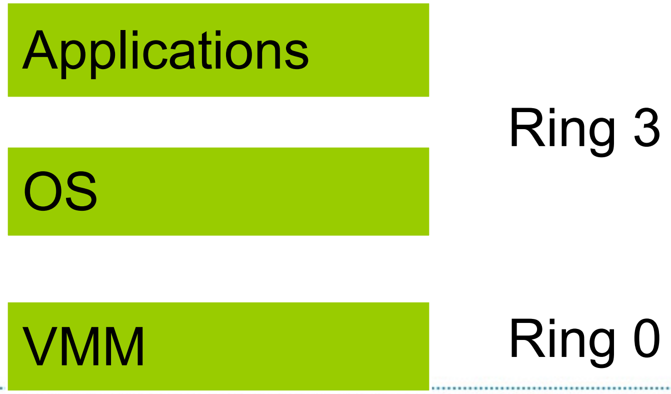
- CPU and RAM virtualization
  - RAM : spatial multiplexing
  - CPU : temporal multiplexing
- I/O virtualization done via emulation
  - Key advantage : portability (guest sees the same virtual hardware on different hosts)
  - Better hardware support now (see later)
  - Why not direct access to device ?

Trap-and-emulate virtualization – 2

- 2 different models







Discussion

- Rings 0/3 :
  - Called Ring Compression
  - OS and user programs are in the same ring level
  - So no protection between them (bad)
- Rings 0/1/3 :
  - OS is protected from user programs
  - Hypervisor is protected from OS
- So it’s a good solution… or not
  - On x86 some instructions fail silently when called from the wrong ring

Getting around x86 limitations without hardware support

- How to force trap on some instructions ?
- Solution 1: paravirtualisation
  - Modify the guest OS
  - Replace non-trappable system calls with calls to hypervisor
  - Not too hard if source code is available
  - Example : Xen on Linux
  - Still exists today (guest additions…)
- Solution 2: binary translation
  - Execute the guest OS on an interpreter
  - Modify it dynamically
  - Dynamic and adaptive translation
  - Example : VMWare

Paravirtualization

Xen

Xen

- Xen is an hypervisor that uses paravirtualization
- Guest OS has to be modified…
  - XenoLinux kernel
- … but not the applications
- Memory management
  - Guest OS are responsible for memory management
  - Xen keeps track of page directoy and page tables
- When a guest OS creates a new Page table
  - Register it with Xen
  - All following updates will be validated by Xen
  - Tables are read only for guests

Xen - 2

- Guest OS runs in ring 1
- Paravirtualization of privileged instructions
- Xen handles interrupt with specific handlers
  - E.g page fault
- I/O
  - Pass interrupts to the corresponding guest
  - Memory transfer between device and guest using shared memory
  - Handled by Xen

Xen – Performance

- Old benchmarks (2003)
- Performance will vary greatly with workload
  - CPU vs Memory vs IO
- Spec CPU
  - Same performance as native
- Linux kernel 2.4 compilation (7% CPU, 93% I/O)
  - 3% overhead
- OS bench
  - Fork : 198 µs (Xen) vs 110 µs (Linux native)
  - Context switch : up to 3 µs overhead

Binary translation

VMware

UNIVERSITÉ CÔTE D'AZUR

VMWare

- Founded in 1998
- Interpretation/compilation of guest OS code
- Idea : the hypervisor keeps track of important register/structures
  - Use of **shadow** structures
- But hard for memory area with sensitive data structures
  - E.g page tables
  - Could be modified by any memory access from guest
  - Need some protection

Binary translation

- Instructions from guest are grouped into Translation Units
- Computational instructions are not translated
- Jump/Branch instruction have to be modified
    - Jump addresses are not valid anymore
- No optimization performed

VMWare Performance

- VMPlayer in software mode
- SPECint 2000
  - Integer computation
  - Between 0 and 9 % overhead
- SPECjbb 2005
  - Server side java
  - 0-2% overhead
- Apache ab benchmark
- HTTP server performance
  - Between 25 and 65% overhead

Table 1.1: Virtual Hardware of early VMware Workstation [45]

	Virtual Hardware (front-end)	Back-end
Multiplexed	1 virtual x86-32 CPU	Scheduled by the host operating system with one or more x86 CPUs
	Up to 512 MB of contiguous DRAM	Allocated and managed by the host OS (page-by-page)
Emulated	PCI Bus	Fully emulated compliant PCI bus with B/D/F addressing for all virtual mother-board and slot devices
	4 x 4IDE disks 7 x Buslogic SCSI Disks	Either virtual disks (stored as files) or direct access to a given raw device
	1 x IDE CD-ROM	ISO image or real CD-ROM
	2 x 1.44 MB floppy drives	Physical floppy or floppy image
	1 x VGA/SVGA graphics card	Appears as a Window or in full-screen mode
	2 x serial ports COM1 and COM2	Connect to Host serial port or a file
	1 x printer (LPT)	Can connect to host LPT port
	1 x keyboard (104-key) and mouse	Fully emulated
	AMD PCnet NIC (AM79C970A)	Via virtual switch of the host

The Popek/Goldberg Theorem

Is an ISA Virtualizable?

- What are the requirements to virtualize an ISA?
  - Necessary and sufficient
- "Formal Requirements for Virtualizable Third-Generation Architectures", Popek and Goldberg, in Communications of the AC,M 1974
- The theorem states that if hypothesis met, an OS running on hardware can also run inside a virtual machine

Model

- One processor with two modes of execution
    - User-level and supervisor
- Virtual memory implemented with segments (not pages)
    - Virtual address x ∊ [0,L] mapped to segment with offset B (i.e. [B,B+L])
- Physical memory is contiguous
- Processor state (called processor status word – psw) consists of
    - Execution level (user, supervisor)
    - Segment register (B, L)
    - Current Program counter

Model - 2

- Trap
  - Saves PSW to a well location
  - Load into PSW values from another well-known location
  - What is the purpose… ? Manage trap then come back to normal flow
- ISA support instructions to
  - Store/Load PSW from arbitrary virtual location
  - What is the purpose… ? $\Rightarrow$ switch between VM

Classification of instruction

- Simple privileged/non-privileged distinction not enough
- Control-sensitive instruction
  - Can update the system state
- Behavior-sensitive instruction
  - its semantics depend on the actual values set in the system state
- Innocuous instruction
  - All the others

Elements of proof

- Hypervisor is only software in supervisor mode
- Each VM has a fixed and contiguous part of physical memory
    - Segment hypothesis
    - Enforce isolation
- Hypervisor has reserved physical memory, separated from VMs
- Hypervisor maintain copies of the psw of each VM

Hardware assisted virtualization

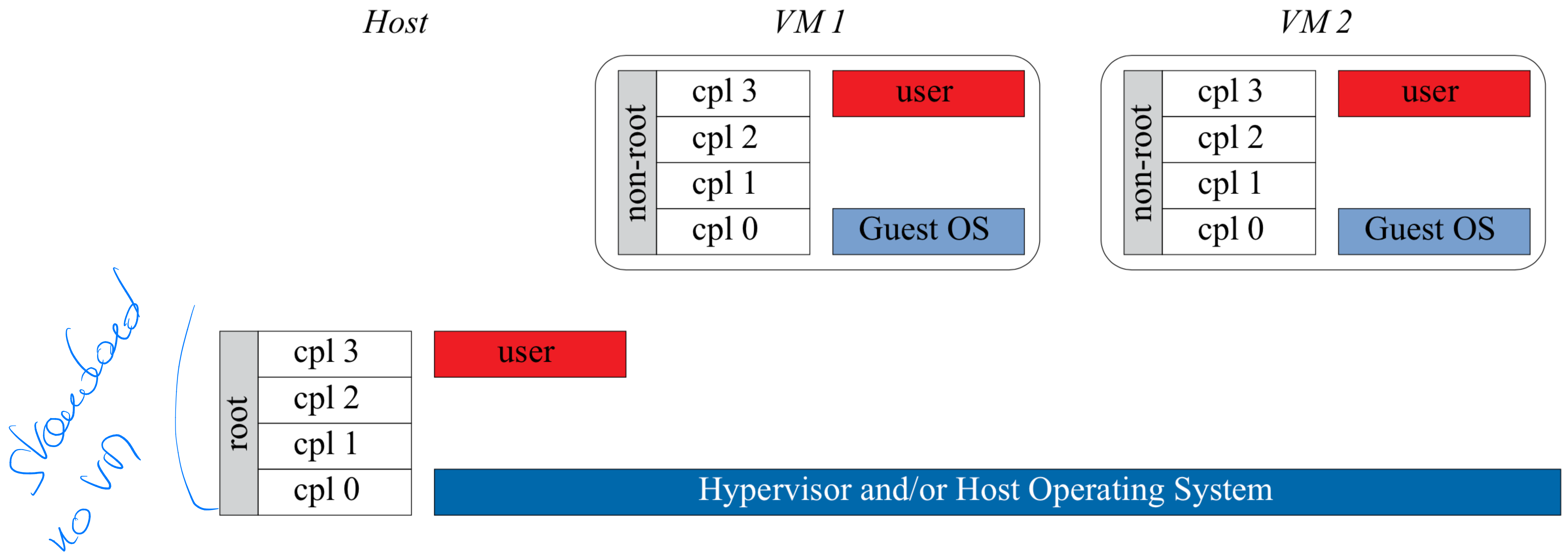
UNIVERSITÉ CÔTE D'AZUR

Intel and AMD

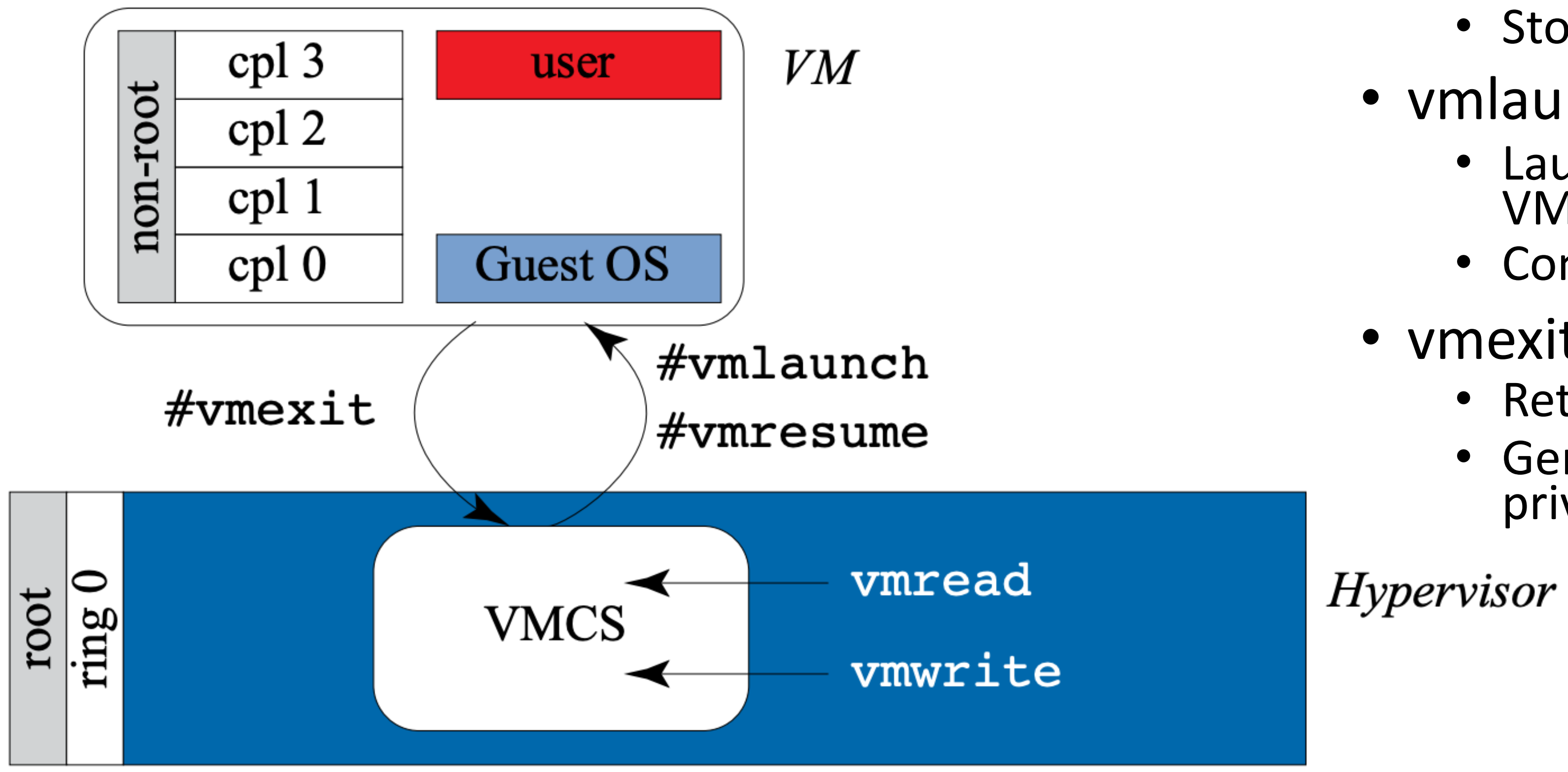
- x86 ISA common to Intel and AMD processors
- Intel Virtualization Technology (Intel VT)
- AMD Virtualization (AMD-V)
- First introduced in 2006
  - New features added regularly
- Objective
  - Eliminate the need for paravirtualization and binary translation

Intel VT-x

- Do not change the semantics of instructions
  - Don’t remove 17 sensitive instructions
- Duplicates the visible state of the processor
  - Add a new mode of execution : root mode
- Hypervisor and OS run in root mode
- VMs run in non-root mode
- Properties
  - Transition between root and non-root mode is atomic
  - Each mode still relies on security rings


Host
VM 1
VM 2
non-root
cpl 3
cpl 2
cpl 1
cpl 0
user
Guest OS
non-root
cpl 3
cpl 2
cpl 1
cpl 0
user
Guest OS
root
cpl 3
cpl 2
cpl 1
cpl 0
user
Hypervisor and/or Host Operating System
Standard
no VM

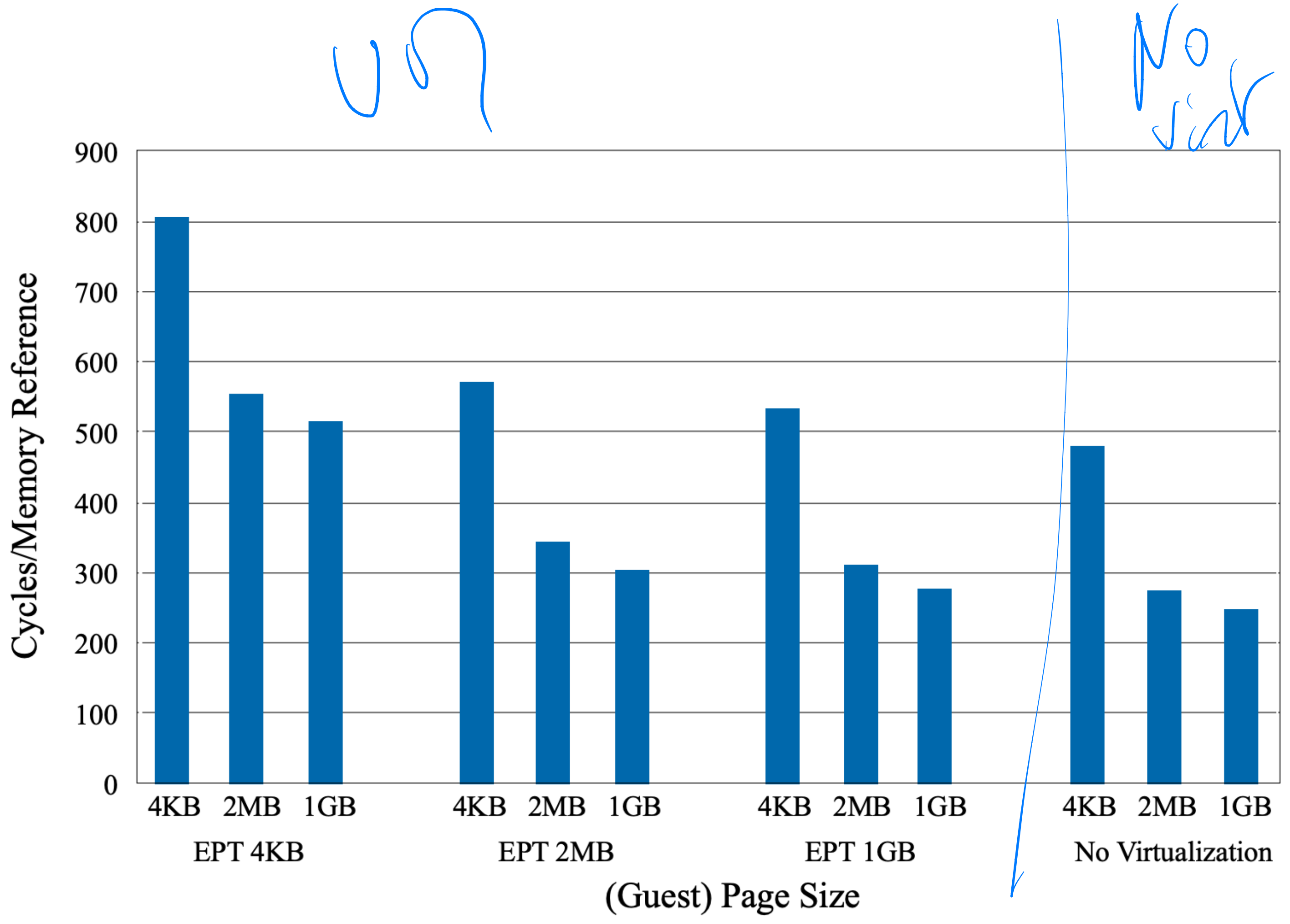
VT-x transitions



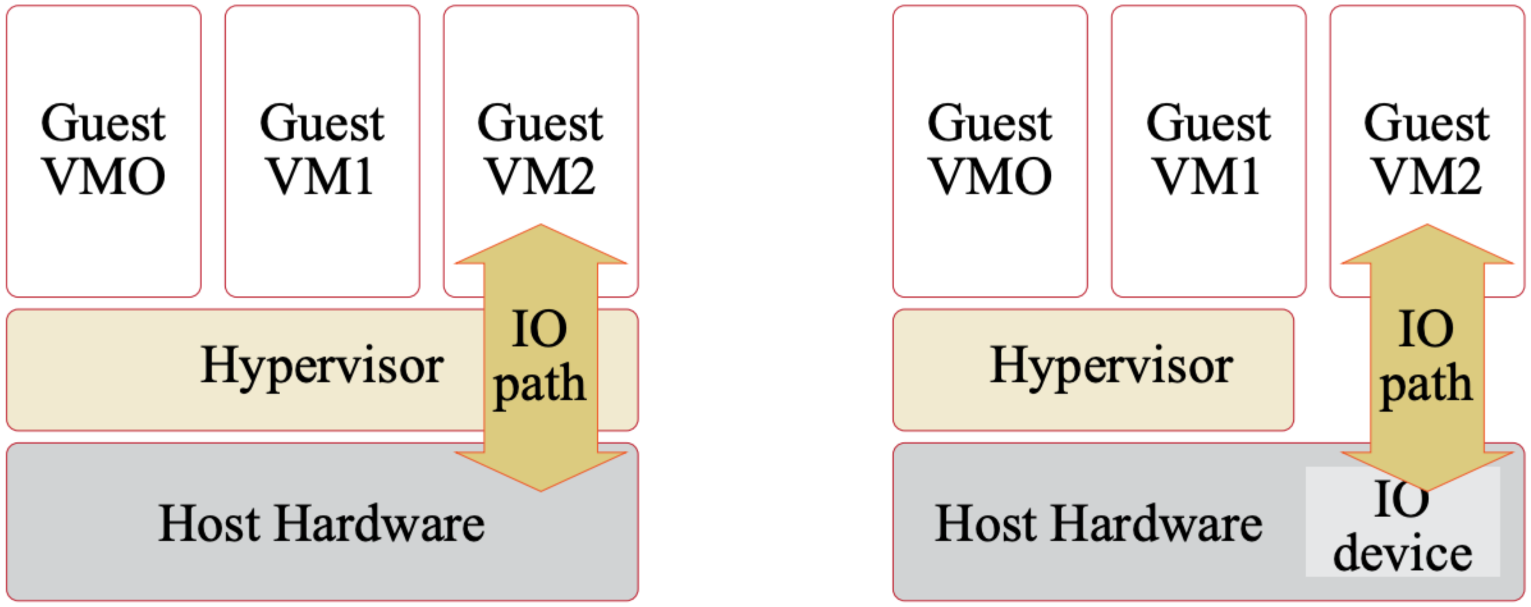
- Virtual Machine Control Structure (VMCS)
  - Stores the state of the VM
- vmlaunch/vmresume
  - Launche/resume a VM managed by the VMCS
  - Control is transferred to VM
- vmexit
  - Returns control to hypervisor
  - Generated when calling root-mode-privileged instructions, interrupts…

MMU Virtualization

- Extended Page Tables (EPT)
  - Available on Intel and AMD
- Add a second page table structure
  - Map guest-physical to host-physical address



Virtual I/O with hardware support



(a) Emulation/paravirtualization (b) Direct device assignment

- Allow guest direct access to device (passthrough)
  - Not scalable
  - Security issue with DMA

Virtual I/O with hardware support

- 2 mechanisms
  - IOMMU (I/O Memory management unit)
  - SRIOV (Single Root I/O Virtualization)
- IOMMU provides security
  - Remapping of DMA
  - Instead of DMA to physical address, DMA to I/O virtual address
- SRIOV extends PCIe
  - Support for devices that self-virtualize
  - An SRIOV-capable I/O device can present multiples instances of itself
  - E.g : a GPU, a NIC