

Heavy and Light Virtualization

Guillaume Urvoy-Keller
UCA/I3S

Outline

- Part I
 - ➔ What is virtualization : a first definition
 - ➔ Why virtualizing?
 - ➔ The new features offered by virtualization
 - ➔ The various virtualization solutions
- Part II : Nuts and bolts of Hypervisors
 - ➔ Refresher on computer architecture and the hardware/OS/Application interfaces
 - ➔ Popek and Golberg criteria for virtualizing an ISA
 - ➔ Details on Vmware player
 - ➔ Hardware support: Intel-VT and AMD-V
- Part III : Nuts and bolts of Containers
 - ➔ Namespaces
 - ➔ Docker

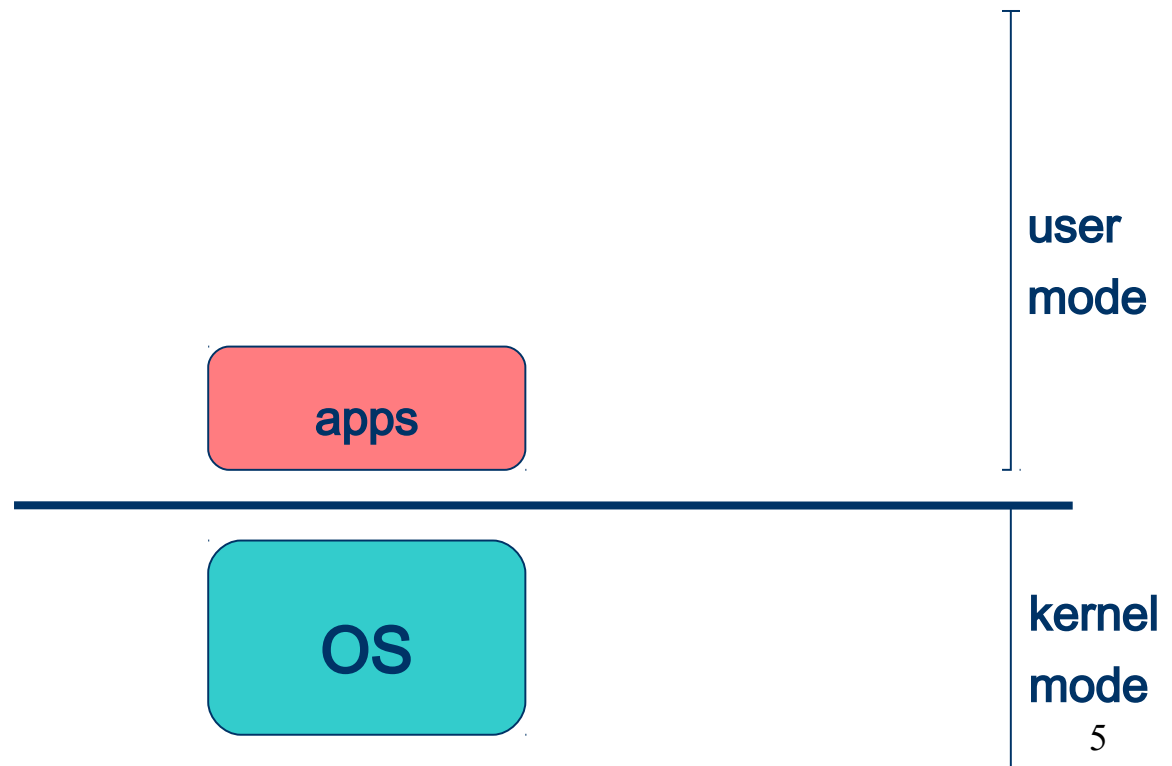
Part I: Introduction

Introduction

- Set of techniques to enable several OS to run simultaneously on a physical machine/server (**≠ multi-boot**)
- **Virtualization entails a specific layer called a hypervisor, a.k.a, virtual machine monitor (VMM)**
 - ➔ **Mediator between physical resources and the OSes that are virtualized**

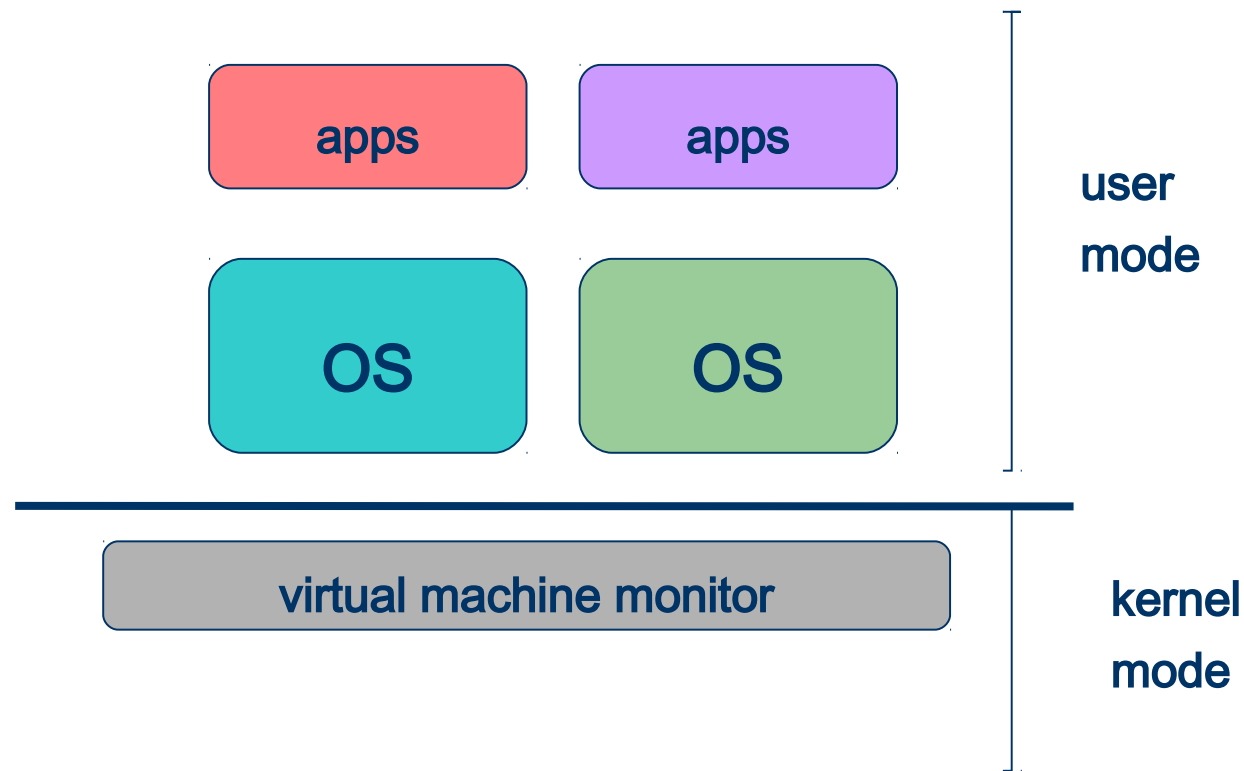
Virtualization = unpriviledging an OS

- Processors features two modes : user and kernel



Unpriviledging an OS

OS in user mode is called Guest OS



Hypervisors Zoo

➤ For servers :

- ➔ VmWare Vsphere ~ 60% market share
- ➔ Microsoft Hyper-V ~ 20%
- ➔ Citrix XenServer ~ 4% - Amazon Web Services
- ➔ QEMU/KVM – default openstack deployments

➤ For clients

- ➔ Oracle Virtualbox
- ➔ Vmware player

➤ Source :

<http://www.nasdaq.com/article/growing-competition-for-vmware-in-virtualization-market-cm316783>

Why virtualizing?

- In the 90s, cost of servers decreased gradually
- Software editors (Microsoft, distribution Linux) advocate one application/service per server →
 - One DNS server
 - One mail server
 - One NFS server
- Each server with specific OS version and libraries
- Servers isolation

Why virtualizing

- Net result:
 - A host of servers in servers rooms (data centers)
 - 80% have an average utilization below 10%
 - Maintenance/operational costs increase with the number of servers
 - Server rooms are not infinitely extensible → lack of place
 - Non negligible electricity/air conditioning cost

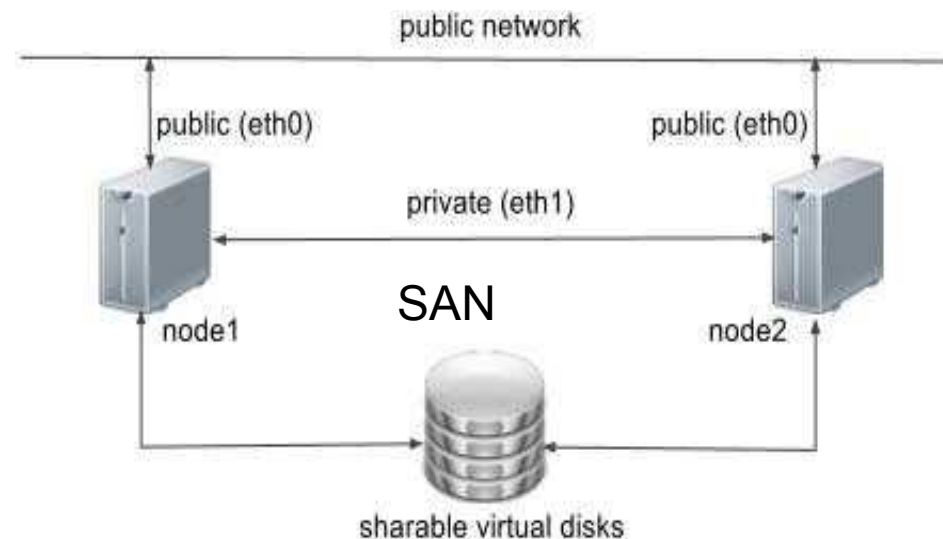
Why virtualizing

- Servers are less expensive but also more powerful
 - ➔ 64 bits multi-core with tons of RAMs are the norm
 - ➔ One server in 2009 is roughly estimated to be one order of magnitude more powerful than a server in 2004
- Replacing servers on a one to one basis is not an option any more
- Still, one wants to ensure service isolation

Advantages of virtualization

- Consider a company that has virtualized its IT
- Typical set-up for an SME:
 - ➔ Two high-end servers, a SAN (Storage Area Network) to share/consolidate storage between the servers
 - ➔ Licences, e.g., Vmware
 - ➔ Training cost for IT service

Source : <http://oracle-base.com>

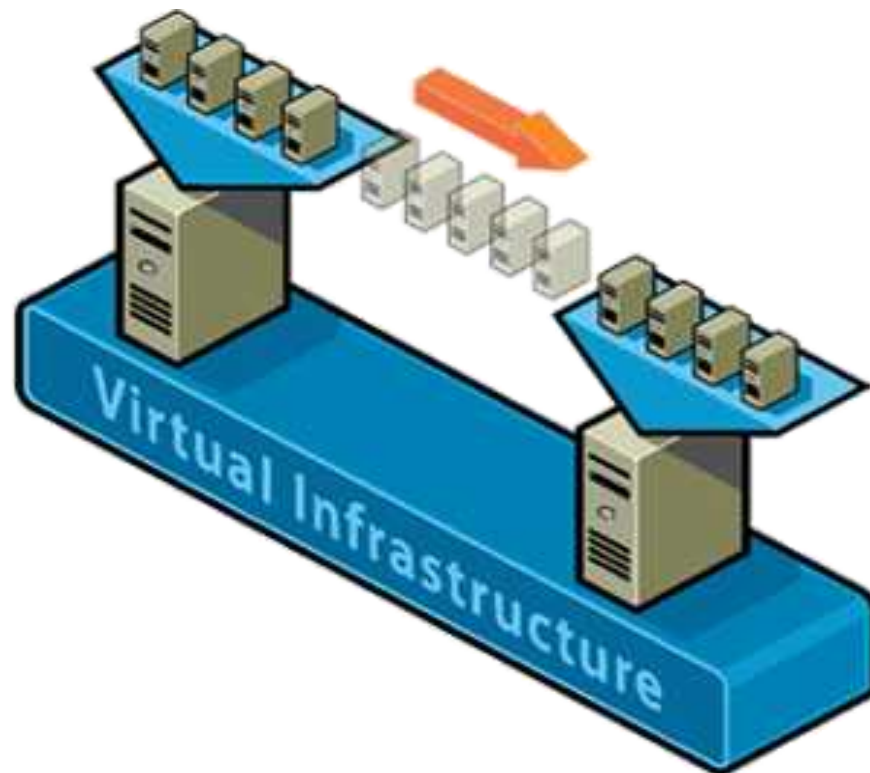


Advantages of virtualization

- Cost reduction
 - ➔ 20 to 40% (even if you had a a few tens of servers before and just 2 after)
- More space in the server room
- New functionalities

Virtualization: New features

- Migration of VM from one physical server to the other one
 - ➔ In case of failure (rare) → higher availability
 - ➔ Maintenance of physical servers (more often)



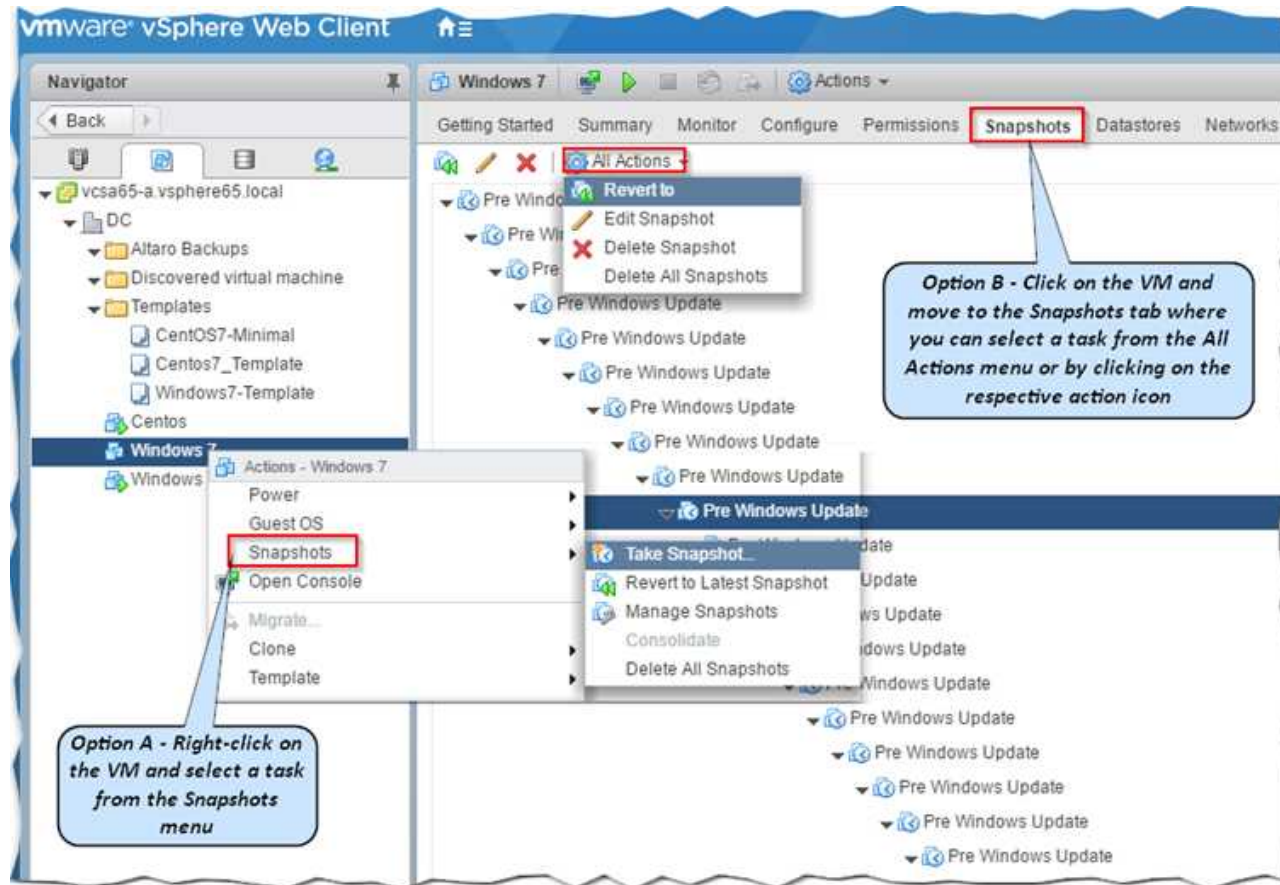
Virtualization: New features

- Fast start-up of new machines through a GUI:
 - ➔ Pick number of CPU, amount of memory, disk type and size, network access, OS
 - ➔ Indicate where the ISO image of OS
 - ➔ Start installation
- Using ISO takes time
 - ➔ Use of templates(one for Linux Debian, etc) that are duplicated (aka clone in the virtualization world) on the fly
 - ➔ Called images in Virtualization world
 - × Vagrant images, AMI (Amazon Images), Docker images
- Current trend : automatic preparation of images out of ISO
 - ➔ See Packer <https://packer.io/>

Virtualization: New features

- Snapshots of VM stage
- Example :
 - ➔ You want to update a library/software but you are unaware of final outcome
 - ➔ You can :
 - × Make a snapshot
 - × Install the library/software
 - × Roll-back time if it is not satisfactory
 - ➔ Also used by hypervisor when freezing (stopping/saving) a virtual machine

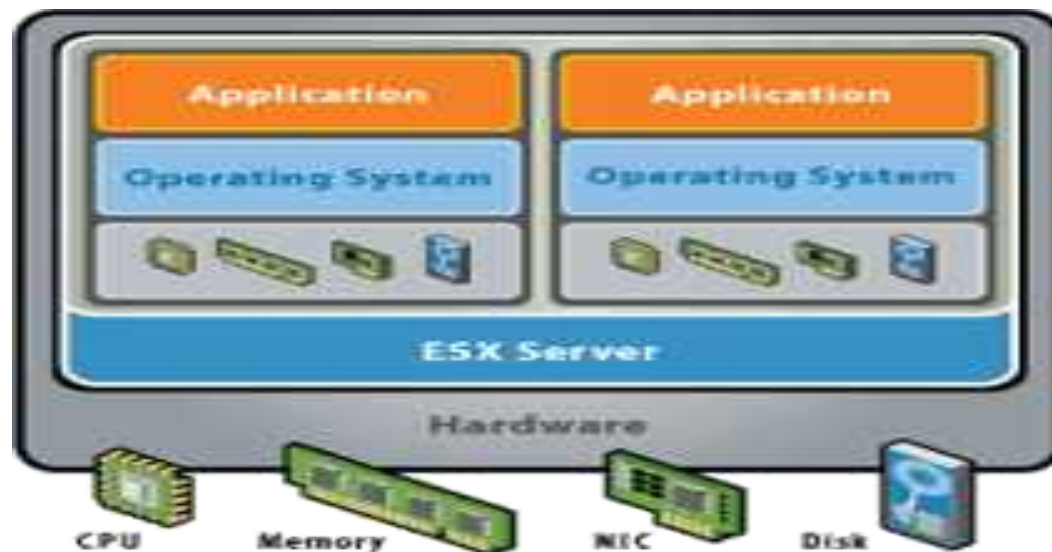
Snapshots



Source : https://www.altaro.com/vmware/wp-content/uploads/2017/02/022817_1130_Workingwith1.png

Virtualization: New features

- Isolation
 - ➔ Virtualization still enforces the one server per machine rule
 - ➔ If one VM is compromised, the other services remain safe
- On the fly reconfiguration of VMs → more CPU, more memory, new virtual disks
- Easier configuration of VM as hypervisor always displays the same interface to the VM
 - ➔ This is the hypervisor that handles the gory details, e.g., supports of SSD drive while VMs are exposed always an SCSI drive → no need to install driver in VMs!!!!



Various types of virtualization

- Different types:
 - ➔ Bare-metal (native) versus host-based ..versus KVM/QEMU
 - ➔ Virtual versus para-virtual.. versus hardware assisted
 - ➔ Container-based versus hypervisor-based

Bare metal vs. host-based

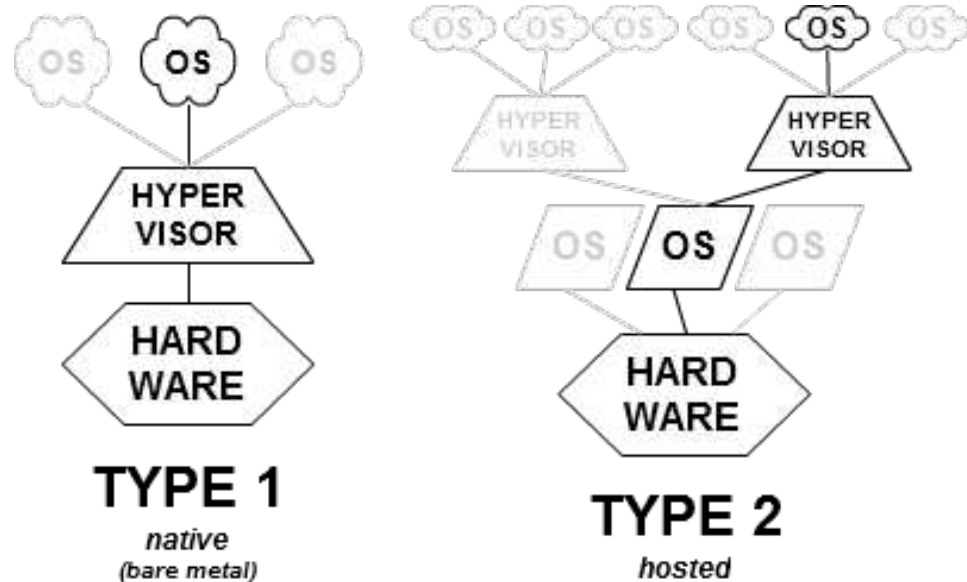
➤ Bare-metal :

- ➔ Layer 1
- ➔ Production servers, data centers
- ➔ Hypervisor seats directly on top of hardware
 - × Machine boots on hypervisor directly
 - × Installed as an OS
- ➔ Examples :
 - × VMware VSphere Hypervisor
 - × Microsoft Hyper-V
 - × Citrix XenServer

Bare metal vs. host-based

➤ Host-based

- ➔ Hypervisor is an application running in an existing OS
- ➔ Layer 2 virtualization
- ➔ Typically deployed on end-user machines
 - × VMware player
 - × Virtualbox



Bare metal vs. host-based

- KVM – Kernel Virtual Machine
 - ➔ Support of virtualization in Linux kernel
 - ➔ Not an hypervisor per se
- Need QEMU – Quick emulator
- Layer 1 in terms of perf
- Layer 2 in style...
- Default in Openstack

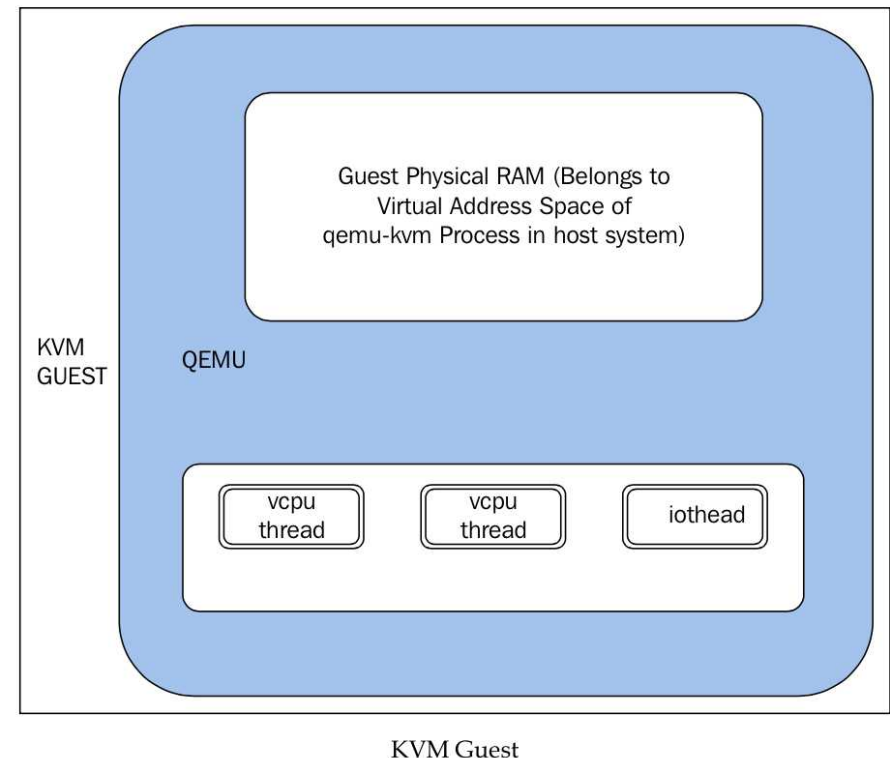
Source :

Mastering KVM Virtualization

Dive in to the cutting edge techniques of Linux KVM virtualization, and build the virtualization solutions your datacentre demands

Humble Devassy Chirammal
Prasad Mukhedkar Anil Vettathu

[PACKT] open source
PUBLISHING community experience distilled

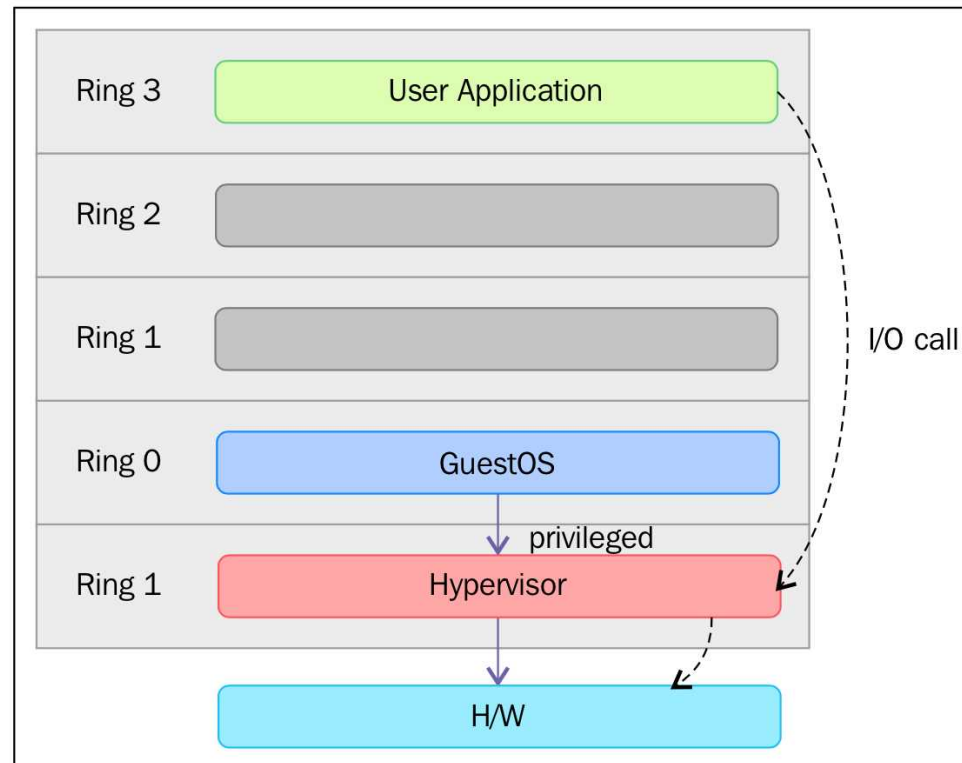


Full vs. Para-virtualization

- Key question : is there a need to patch the guest OS?
 - ➔ No → full virtualization
 - × Direct support by hypervisor
 - × VMware approach
 - ➔ Yes → para-virtualization
 - × A (typically small) part of the kernel is patched to interact with hypervisor
 - × Better performance
 - × Used by Xen initially
- Current trend : no patch but installation of guest additions inside OS

Hardware assisted Virtualization

- Intel and AMD propose various solutions to support virtualization at hardware level → ease of hypervisor task
- Intel VT-x, AMD-V
- Enable OS of virtual machines to do more actions natively → addition of a duplicated structures within processor and a new control level called (-1) for hypervisor



Container-based vs. Hypervisor-based

- Rather than using an hypervisor, the container approach shares kernel among VM
- On a typical server :
 - ➔ 10-100 virtual machines
 - ➔ 100-1000 containers
- A container is a group of processes on a Linux host, grouped together in an isolated environment.
 - ➔ Use of **namespaces** so as to assign to a set of processes : isolation, their own network stack (interfaces, sockets, routing), volumes
 - ➔ Use of **cgroups** to assign resources to processes, eg., CPU share, memory limit

Container-based vs. Hypervisor-based

- Inside the box, it looks like a VM.
- Outside the box, it looks like normal processes.
- A container might not contain a full VM, it might contain simply process, eg. Apache or MySQL server.
- Container engines:
 - ➔ LXC (LinuX Containers – August 2008)
 - ➔ Docker (started in March 2013)
 - ➔ Openvz (started in 2005)

Container-based vs. Hypervisor-based

- Typical arguments for container approach

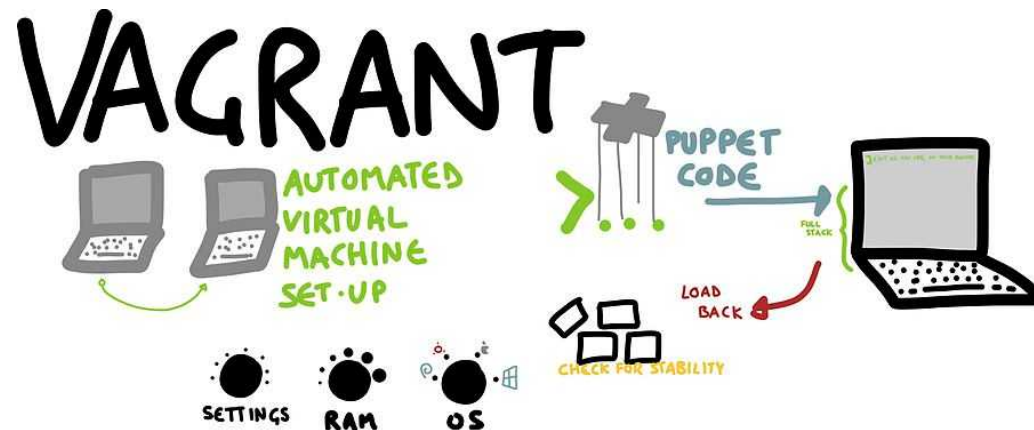
➔ Source : <http://goo.gl/bFHSh>

	Ships within ...	Manual deployment takes ...	Automated deployment takes ...	Boots in ...
No virtualisation	days	hours	minutes	minutes
Virtualization	minutes	minutes	seconds	less than a minute
Lightweight Virtualization	seconds	minutes	seconds	seconds

Around virtualization: management of VMs, containers, virtual network

➤ Management of VMs

- ➔ Vmware Vsphere, Citrix Xen are hypervisors and can offer management of a handful nodes of the same type (ESX servers only or Citrix Server only)
- ➔ Vagrant: Management of VMs a hypervisor independent approach
 - × Notion of images (boxes in Vagrant)
 - × Provisioning of VM: Puppet, Chef, Ansible to configure automatically the VMs
 - × A single file that includes everything



Vagrantfile (excerpt)

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure(2) do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
  # boxes at https://atlas.hashicorp.com/search.
  config.vm.box = "ubuntu/vivid64"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

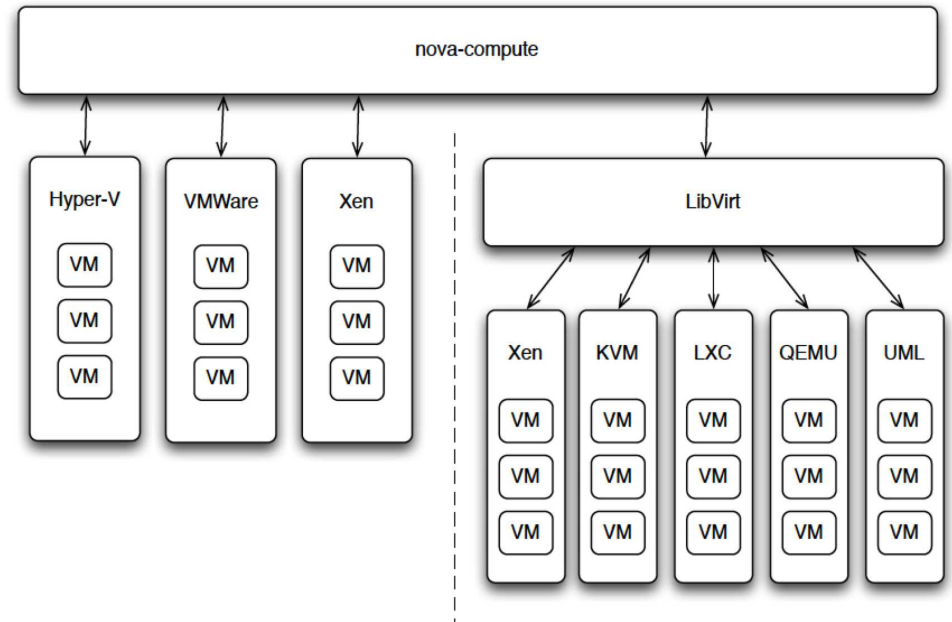
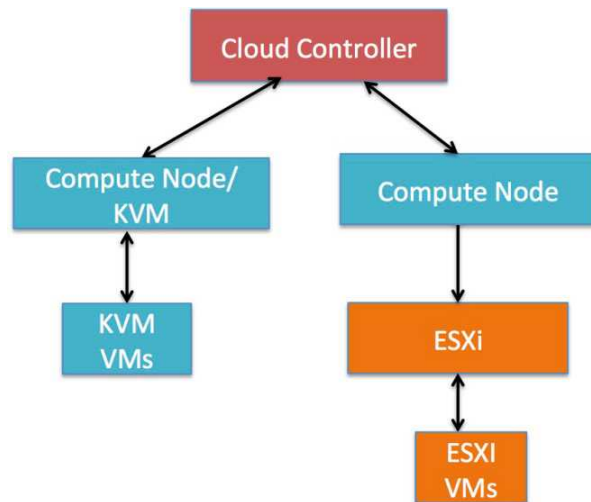
  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  config.vm.network "forwarded_port", guest: 5001, host: 5001

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  # config.vm.network "private_network", ip: "192.168.33.10"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  config.vm.network "public_network"
```

Around virtualization: management of VMs, containers, virtual network

- ➔ Cloud platforms to orchestrate at a larger scale, with possibly different hypervisors
 - × Openstack
 - × Each function (management of VM, of network, of volumes, of identities) is a component
 - × Nova: compute nodes (hypervisors)
 - × Cinder : volumes
 - × Neutron : network
 - × Components interact through a REST API
 - × Compute nodes (physical servers) might run different hypervisors: KVM, Xen, Citrix, etc



Around virtualization: management of VMs, containers, virtual network

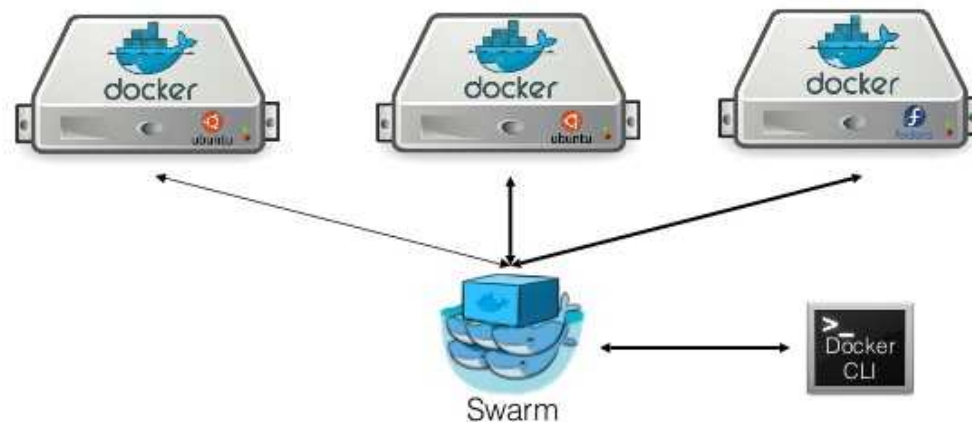
➤ Orchestration of containers

- ➔ Single server level: Docker, LXC
- ➔ Several servers level: Docker Swarm

➤ Advanced orchestration: Kubernetes



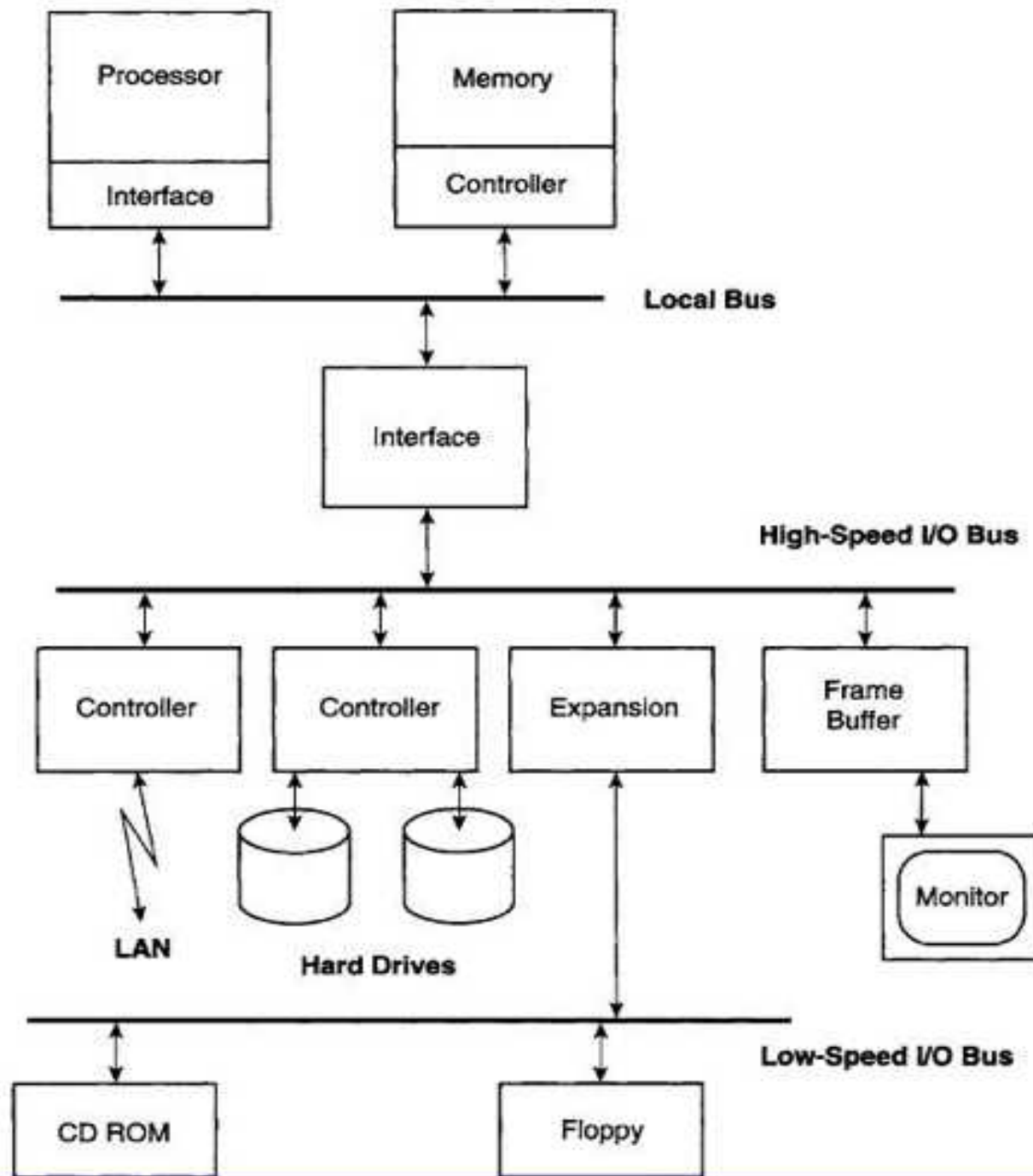
With Docker Swarm



Part II: the nuts and bolts of (hypervisor-based) virtualization

Refresher on computer architecture and OS

- Computer Components
- ISA (Instruction Set Architecture)
- Operating System



Processor

- Nowadays processors
 - Several cores
 - Multithreading → 2 processes sharing the same core
- Consequence: if you have p processors with c core and multithreading, VMware, Xen and others will expose $2 * p * c$ vCPU (virtual CPUs)
- We talk hereafter about processor by referring to a basic computation unit (a processor or a core or half a core..)
- Important : at a given time instant, a processor is servicing a single program:
 - Either a user program (processor in user mode)
 - Or the OS (processor in kernel mode)

ISA

- Architecture of a processor defines
 - ➔ A set of resources : memory, registers, ...
 - ➔ A set of instructions that operate on registers and memory
- Definition of storage resources + instructions → Instruction Set Architectures (ISA)
- One distinguishes :
 - ➔ User instructions for programs → computations
 - ➔ System instructions for OS → management of resources

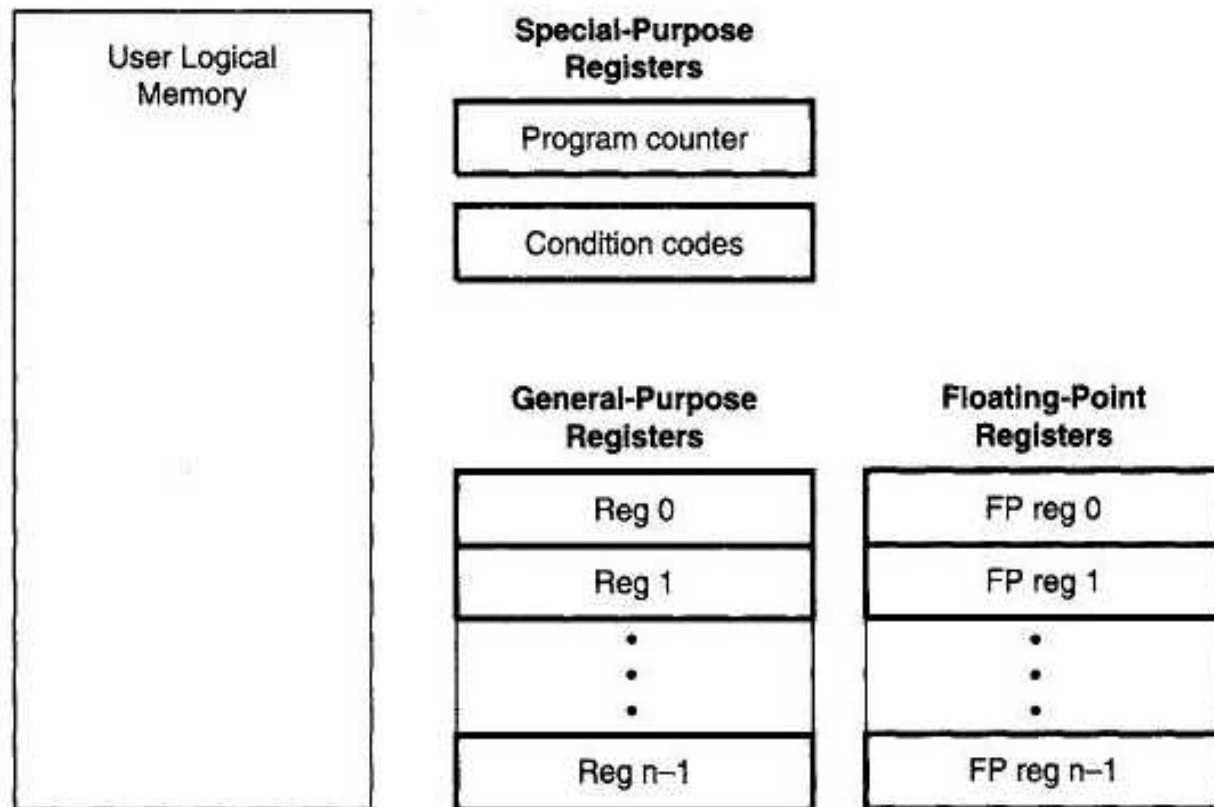


Figure A.6 Key Architected User State of a Typical ISA. *This includes a logical (virtual) memory space, a special-purpose register, general-purpose registers, and floating-point registers.*

Registers

- Generic(aka working) registers
 - Can host different types of values
- Typed registers: for specific operands
 - ISA dependent
 - ex: pointers towards memory segments in Intel IA-32 ISA
- Specific registers:
 - Program counters: index of current instruction
 - Condition

User ISA

- Four main categories
 - Load/store memory $\leftrightarrow$ registers
 - Integral/logical operations and shifting
 - Floating point operations
 - Branching and jumps

System ISA

- Processors features several execution modes :
 - ➔ In general 4, from 0 to 3
 - ➔ Windows and Unix use levels 0 and 3
 - ➔ Level 0 : system mod for OS to control and share fairly resources among programs
 - ➔ Level 3 : user programs
- Execution mode is stored in a specific register

System registers

- Time Register
- Traps and interruptions registers
- Traps and interruptions masking registers
- Page table pointer
 - ➔ Mapping between logical and physical spaces. Kept in RAM
 - ➔ “Page table pointer” points to the memory location of this table

Traps and interruptions

- Traps and Interruptions lead to a transfer of control of processor to OS
- Interruption: request from an I/O device to the OS
- Trap:
 - ➔ An error during the execution of an instructions(page fault , division by 0, ..)

or

- ➔ An explicit demand from a program

System ISA : management of processor

- OS must be able to hand over control of the processor to a user program
 - ➔ Jump to an instruction of a user program
 - ➔ Modification of execution state register
- ... and must be able to gain control later again
 - ➔ Thanks to a timer that will generate an interruption

System ISA : I/O management

- Can be specific instructions
- Can be specific addresses translated as instructions to be executed by memory controller
- Wide variety of I/O devices → ISA offers in general only a few basic instructions
 - OS in charge of communication with devices through the device driver

Refresher on Operating system

OS tasks

- Manage resources on behalf of user programs
 - ➔ Time multiplexing of processor
 - ➔ Management of physical memory via page table and TLB
 - × When page error, OS takes over control of CPU
 - ➔ I/O management:
 - ➔ Processes perform requests to OS via system calls (that result in traps)
 - ➔ OS uses device drivers (that are added to OS) to control devices

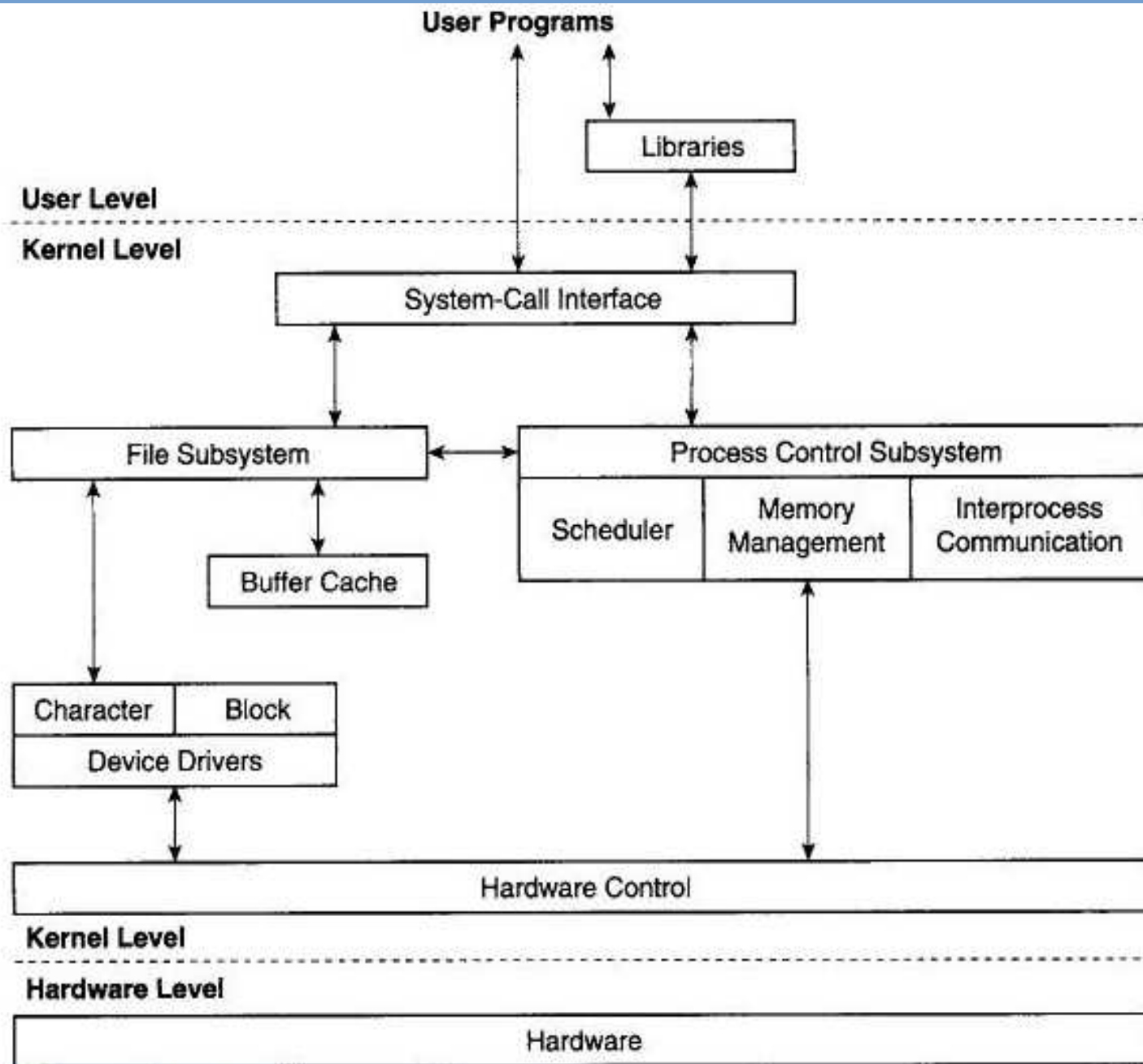


Figure A.12 Linux Architecture.

Interface with OS

- User mode ISA directly accessible to user programs
- ABI (Application Binary Interface) is the interface with OS
- API: high-level libraries (as compared to ABI)
- System calls:
 - Process management, ex : `fork()`
 - Memory management, ex : `malloc()`
 - I/O, ex : `. read()`
- Abstraction of traps and interruptions at ABI = signals

ISA, ABI, API

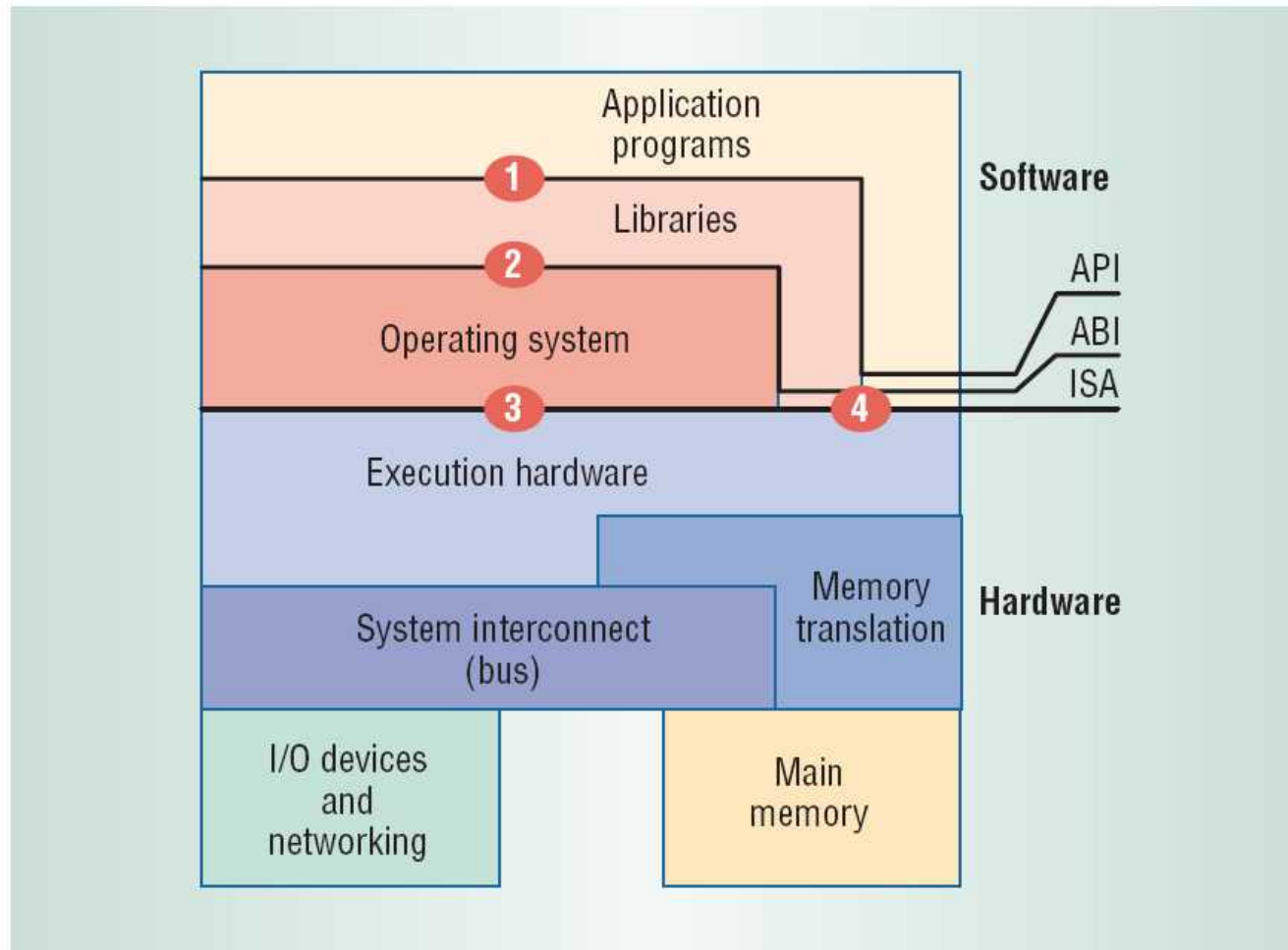
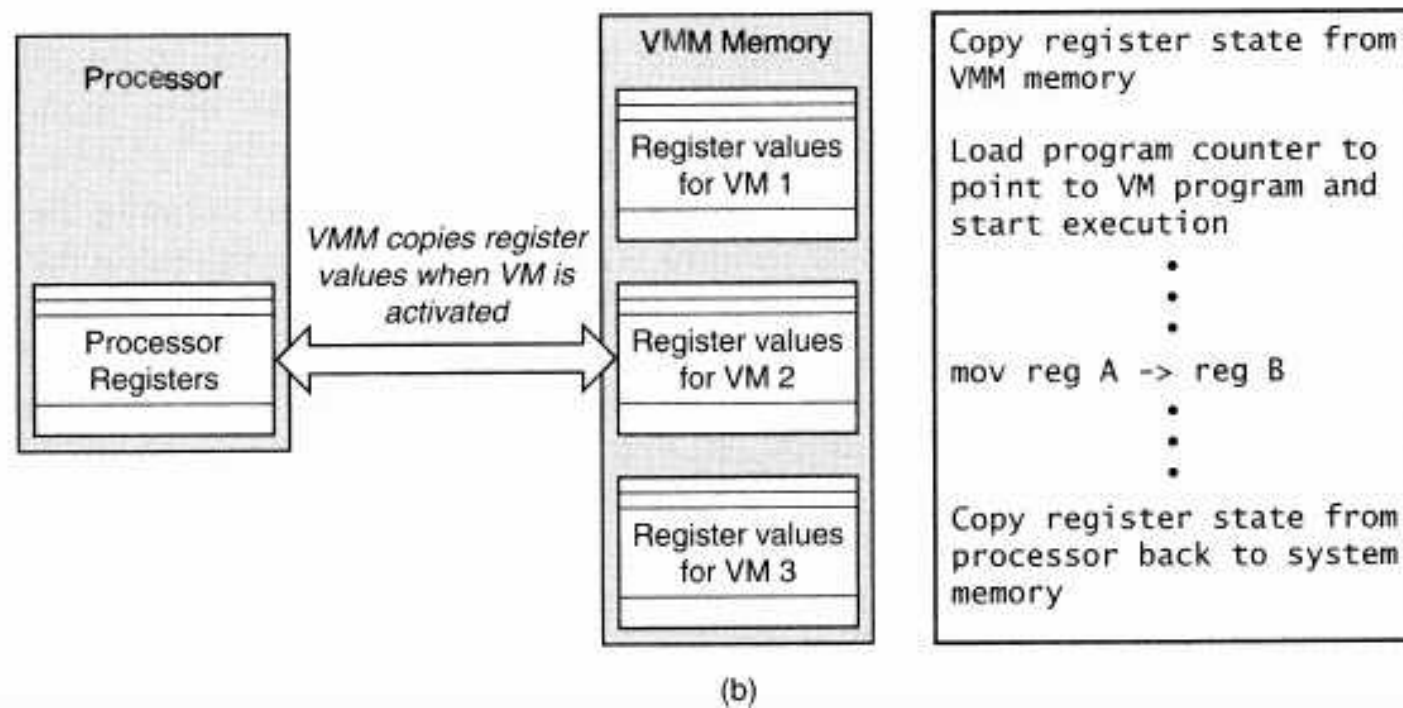
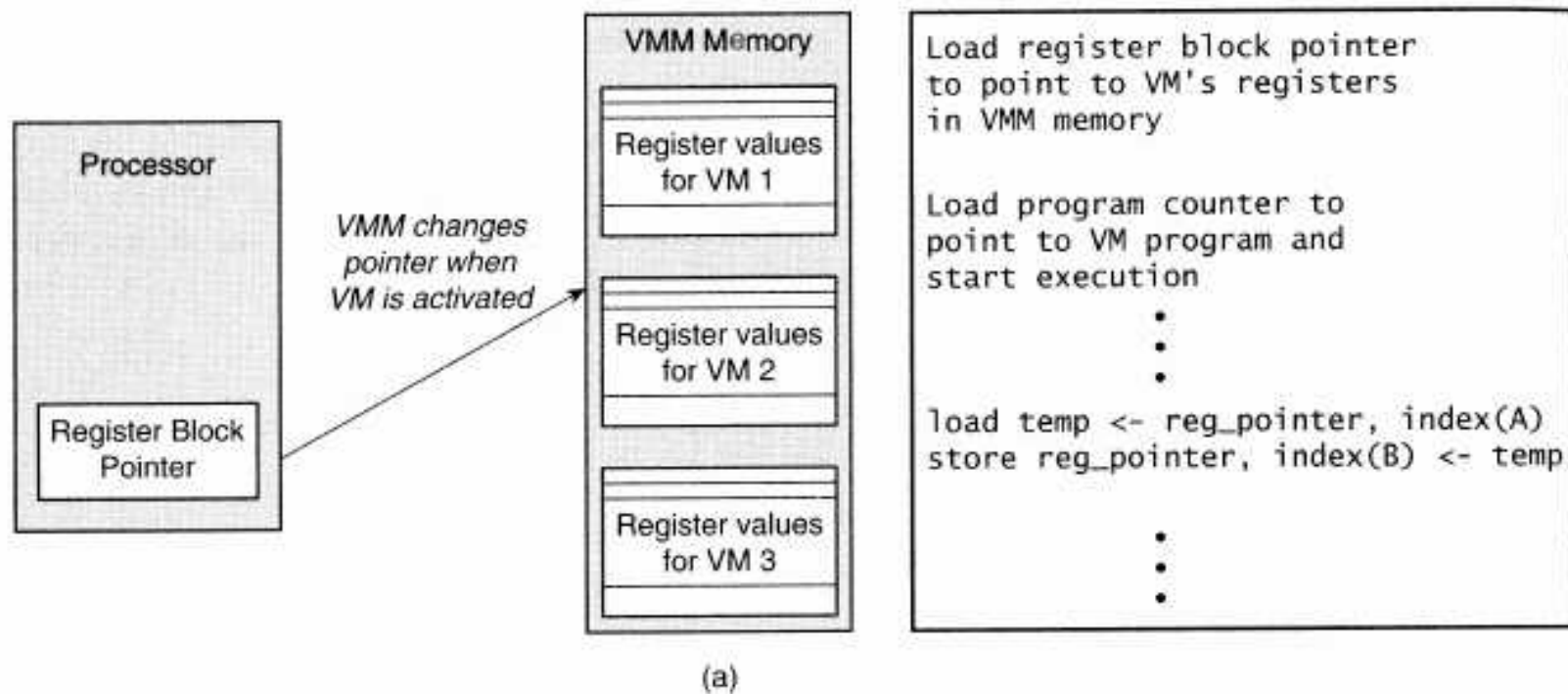


Figure 2. Computer system architecture. Key implementation layers communicate vertically via the instruction set architecture (ISA), application binary interface (ABI), and application programming interface (API).

L1 and L2 hypervisors from the inside

Management of VM states

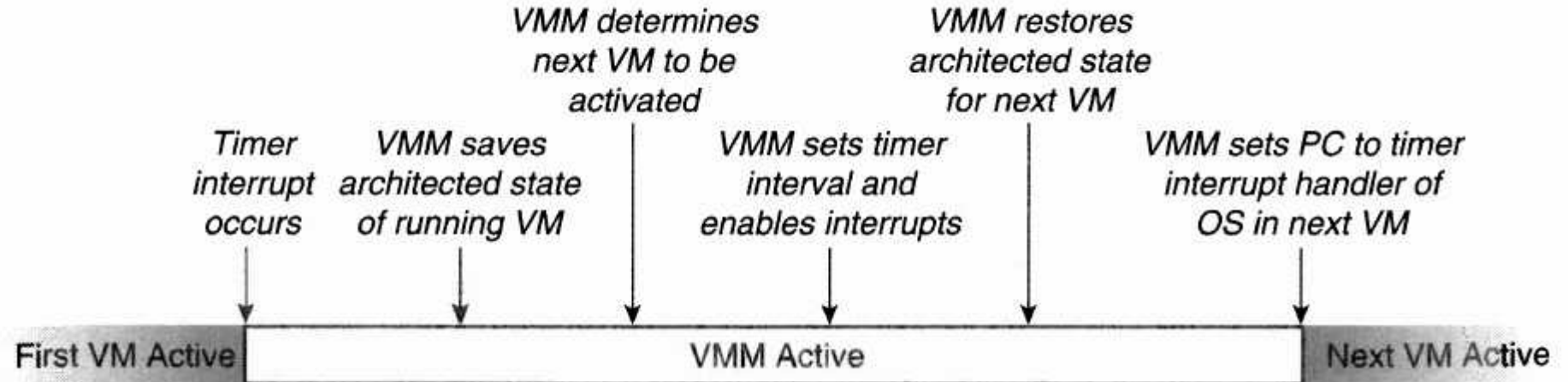
- Hypervisor must store complete VM state when switching from one VM to the next VM :
 - ➔ Registers
 - ➔ Page table (of guest OS)
 - ➔ I/O
- Two main options for context switching:
 - ➔ Indirection – inefficient because operation between register imposes several memory accesses
 - ➔ Copy – possible if same ISA between guest and host (the



Global control by Hypervisor

- VMM (=hypervisor) must be able to get back control of processor when VMs
 - ➔ Equivalent to time multiplexing of OS (dealing with user programs)
 - ➔ Same objective: fairness
- How? VMM manages the time timer
- As guest OSes need also this timer, one needs emulate this timer for guest OSes

Global control by Hypervisor



Virtualization of resources – processor

- Two main approaches
- Emulation
 - ➔ Mandatory if guest and host (physical) ISA are different
 - × Ex: you want to test an Android program on a different architecture
- Direct execution (same ISA) :
 - ➔ Faster than emulation
 - ➔ Some instruction must be emulated anyway:
 - × Instructions sent by user program can be executed as is
 - × Instruction sent by guest OS must often be emulated

Conditions for an ISA to be virtualizable

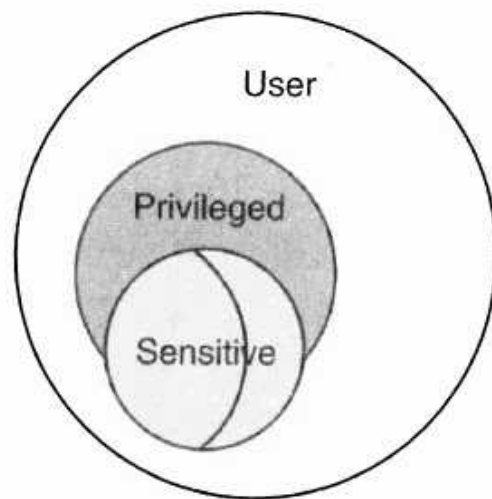
- Article Popek and Goldberg 1974
 - ➔ Virtualization dates back to the mainframe age
- Hypotheses (short version) :
 - ➔ Hardware = 1 (unique) processor
 - ➔ Processor has 2 modes : system and user
 - ➔ Some instructions can only be executed in kernel mode
- I/O non considered in article, but results can be extended to this case

Sensitive vs. non sensitive instructions

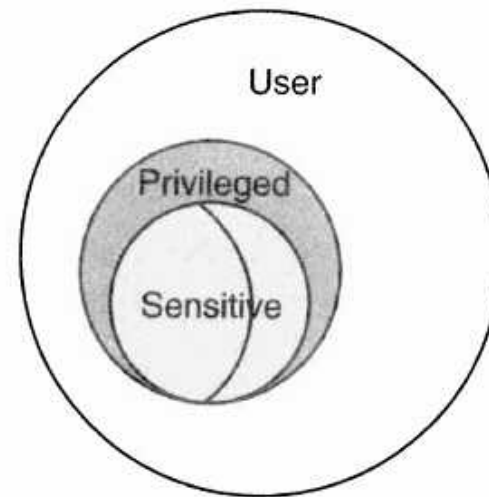
- Sensitive instructions:
 - × Modify the configuration of resources, e.g. memory or execution mode (control type)
 - × Ex : SPT - Set CPU timer (IBM/370)
 - × Whose result depend on resource state (behavioral type)
 - × Ex : Load Real Address (IBM/370)
 - ×
- Instructions that are neither of control nor behavioral type are called benign (non sensitive)

Popek and Goldberg

Property : sensitive instructions must form a subset of privileged instructions



(a) Does not satisfy condition



(b) Satisfies condition —
efficiently virtualizable

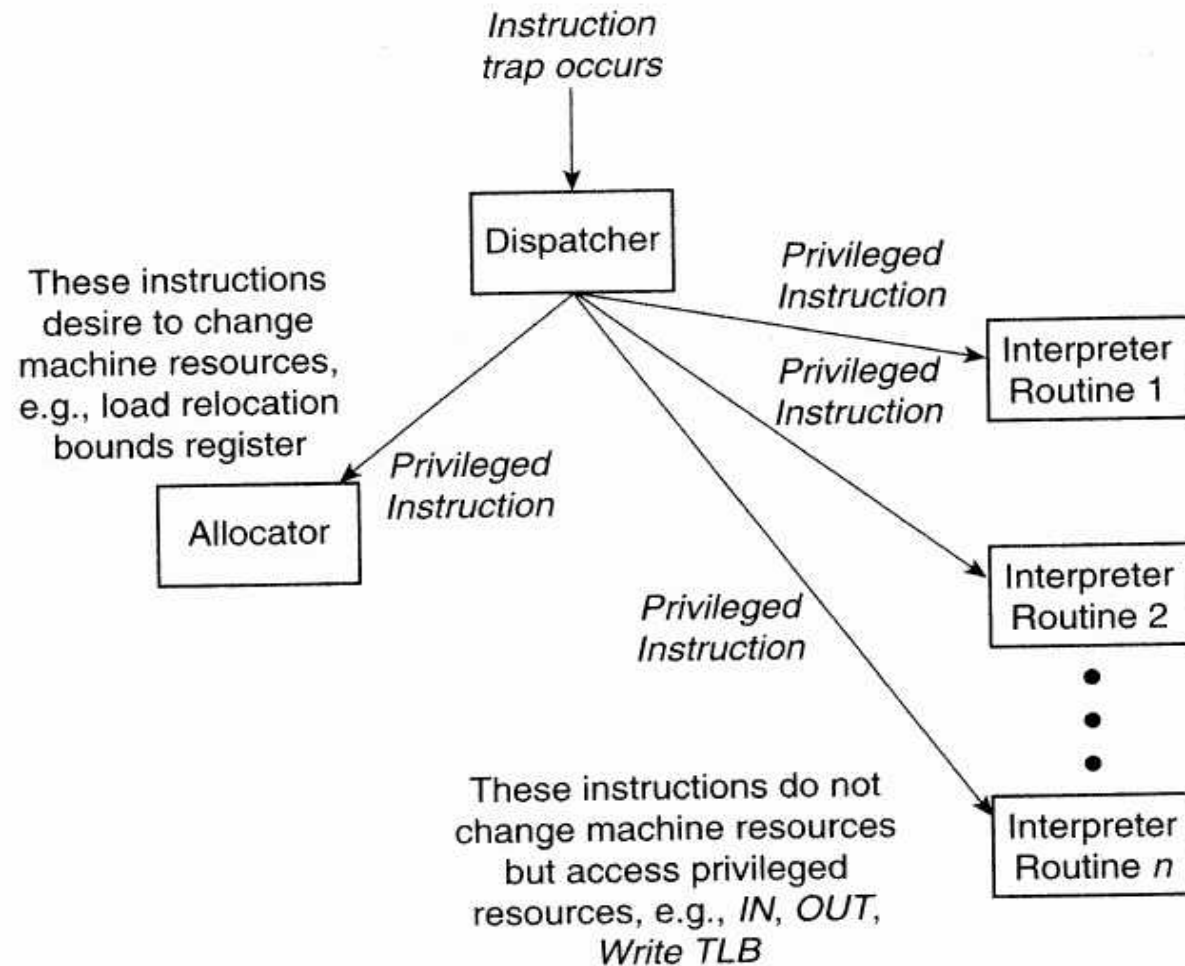
Job of hypervisor if Popek and Goldberg conditions are met

- Non sensitive instructions can be executed natively (no interference of hypervisor) → speed
- Hypervisor keeps control of resources : a user program or the guest OS cannot modify system resources (as processor has to be in kernel mode)

Job of hypervisor if Popek and Goldberg conditions are met

- If the set of sensitive instructions is a subset of privileged ones, the job of hypervisor is easy: wait for traps (errors from guest OS or guest programs) and patch:
 - ➔ If an error by guest OS, do the job and give back hand to the guest OS
 - ➔ If an error by guest program, give back hand to guest OS

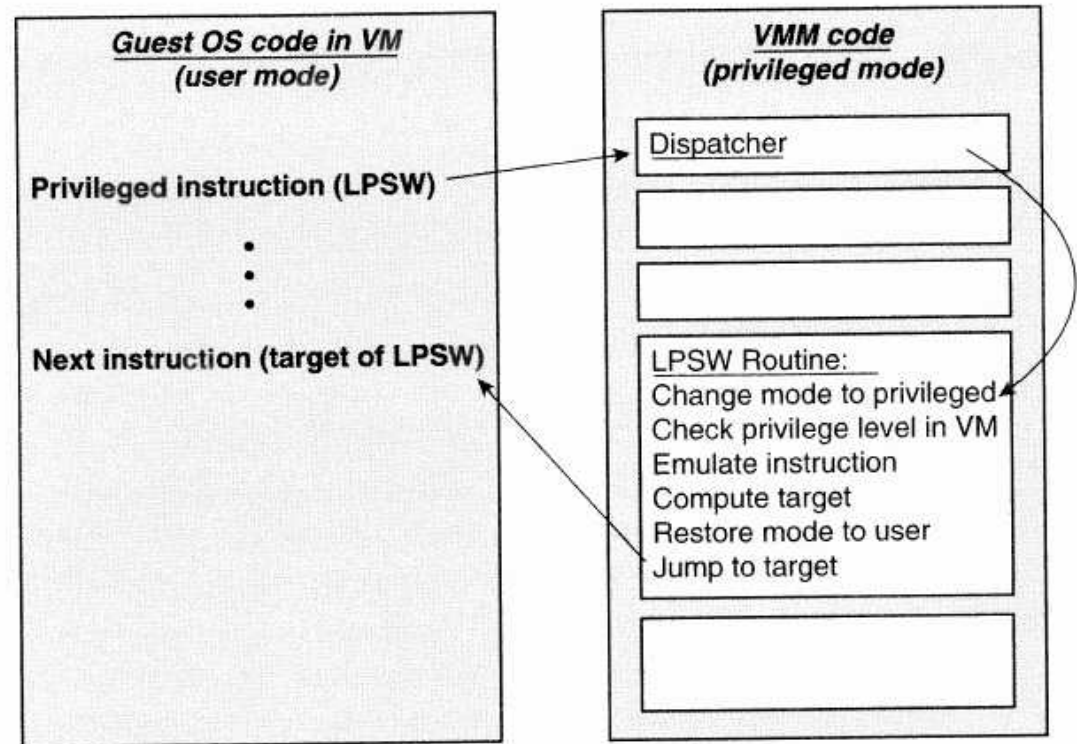
Job of hypervisor if Popek and Goldberg conditions are met: Wait for traps!!!



Intel ISA x86 case

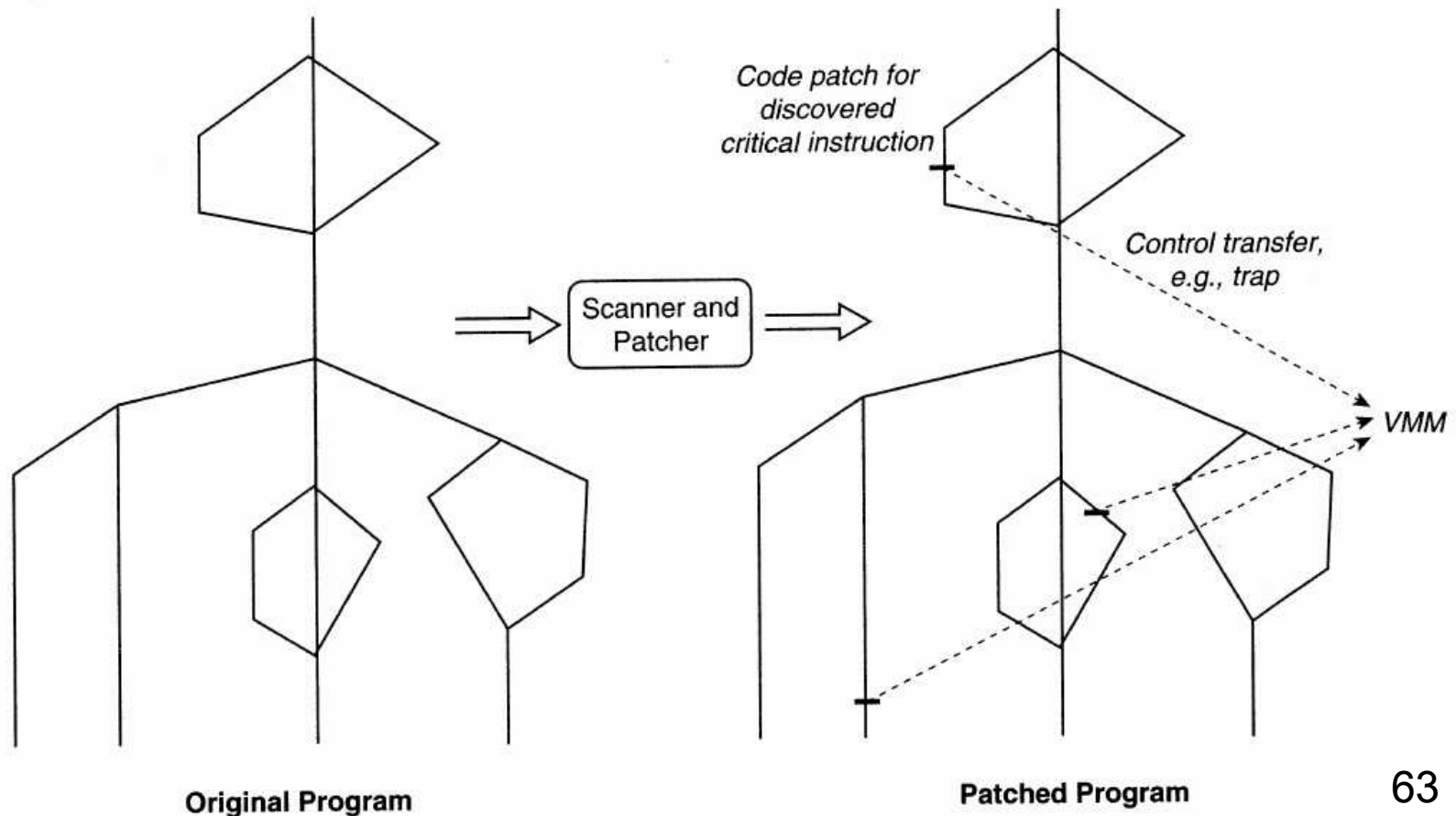
- “Analysis of the Intel Pentiums Ability to Support a Secure Virtual Machine Monitor” USENIX 2000, J. Robin, C. Irvine
- Over the 250 instructions, 17 are sensitive but not privileged (they are called critical)
 - ➔ Example: popf
 - × Popf in user mode: change ALU flag
 - × Popf in kernel mode: change ALU *and* system flag
 - × *No trap in user mode*

- ▶ Consequence : if popf executed by guest OS, it will not be trapped but system flag won't be modified → guest OS will be in inconsistent state!!!
- ▶ Similar problem in other ISAs.
- ▶ Solution: VMM intercepts and “patches”



Hypervisor job: track critical instructions

- Hypervisor must scan code and patch instructions before execution
- Challenge : code can branch and it is difficult to know which branch will be used in advance



Processor Virtualization: summary

- Popek and Goldberg conditions not met by ISA in general
- Hypervisor must scan code
 - Only when guest OS is in function
- It must patch
- Above operation is called Binary Translation
 - VMware is an expert in this area

I/O Virtualization

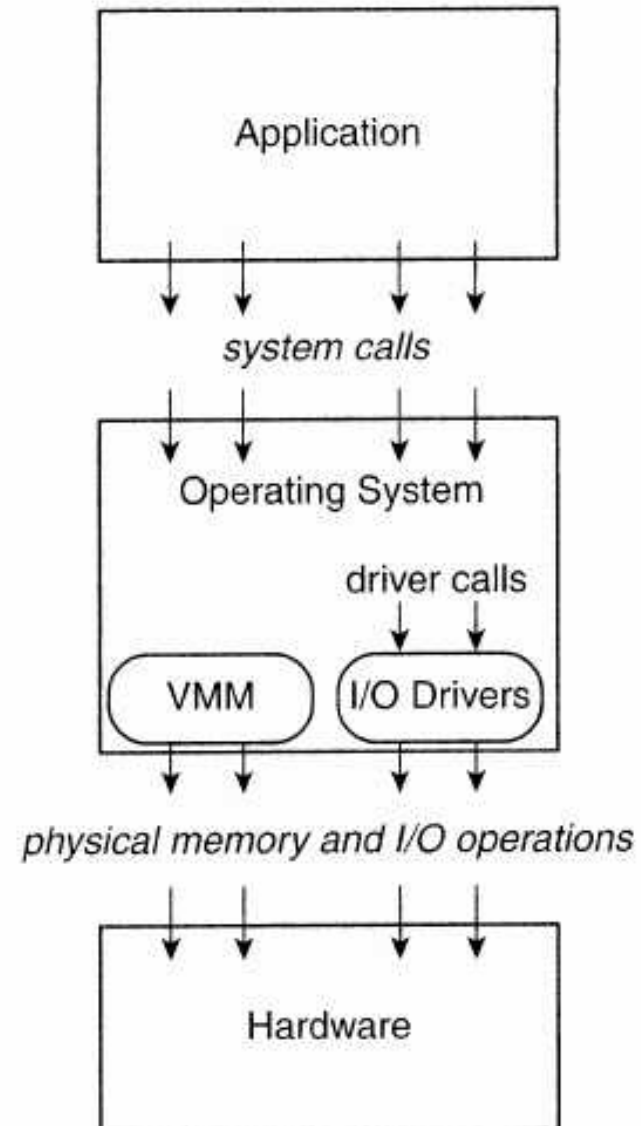
- ▶ One of the most complex tasks in virtualization because there is a LOT of I/O devices
- ▶ General approach: offer a clean interface to guest OS

I/O Virtualization

- Typology of devices:
 - Dedicated. Ex: screen, keyboard.
 - Partitioned. Ex : disks
 - Shared. Ex: NIC
 - Spooled. Ex : printer. Shared, but at a high level of granularity(files in spooler). In practice, printers often accessed through network
 - Non existing. Ex : virtual switch to interconnect VM in the same physical machine

Where can we virtualize I/O?

- ▶ Options:
 - System calls?
 - Drivers?
 - I/O operations?
- ▶ Solution used: driver level



Driver level

- Call to driver via system call
- For hypervisor to intercept this call, it must know details of the guest OS and of driver → too complex
- Alternative approach: offer a normalized interface to device(e.g., always the same Intel Ethernet card, irrespectively of physical one) and provide a specific driver to guest OS to intercept and translate requests.
- This entails that hypervisor has drivers for real device

Virtualization at driver level : the case of Xen

- Dom U: VMs
- Dom 0 : drivers

