

Virtualization

Guillaume Urvoy-Keller
UNS/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
 - ➔ 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 : virtual switches
 - ➔ Open Vswitch design and performance

References

- Smith and Nair - Elsevier



References

The Architecture of Virtual Machines



A virtual machine can support individual processes or a complete system depending on the abstraction level where virtualization occurs. Some VMs support flexible hardware usage and software isolation, while others translate from one instruction set to another.

James E. Smith
University of Wisconsin-Madison

Ravi Nair
IBM T.J. Watson Research Center

Virtualization has become an important tool in computer system design, and virtual machines are used in a number of subdisciplines ranging from operating systems to programming languages to processor architectures. By freeing developers and users from traditional interface and resource constraints, VMs enhance software interoperability,

A computer's instruction set architecture (ISA) clearly exemplifies the advantages of well-defined interfaces. Well-defined interfaces permit development of interacting computer subsystems not only in different organizations but also at different times, sometimes years apart. For example, Intel and AMD designers develop microprocessors that implement the Intel IA-32 (x86) instruction set,

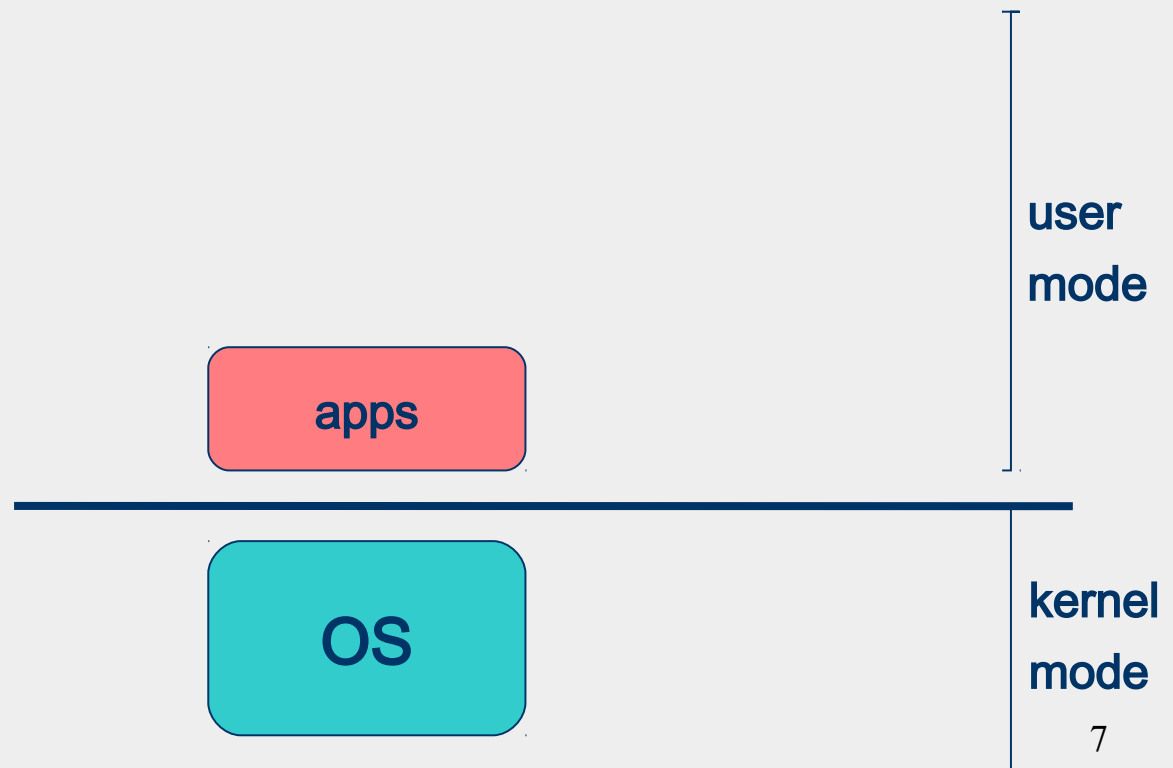
Part I: Introduction

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**

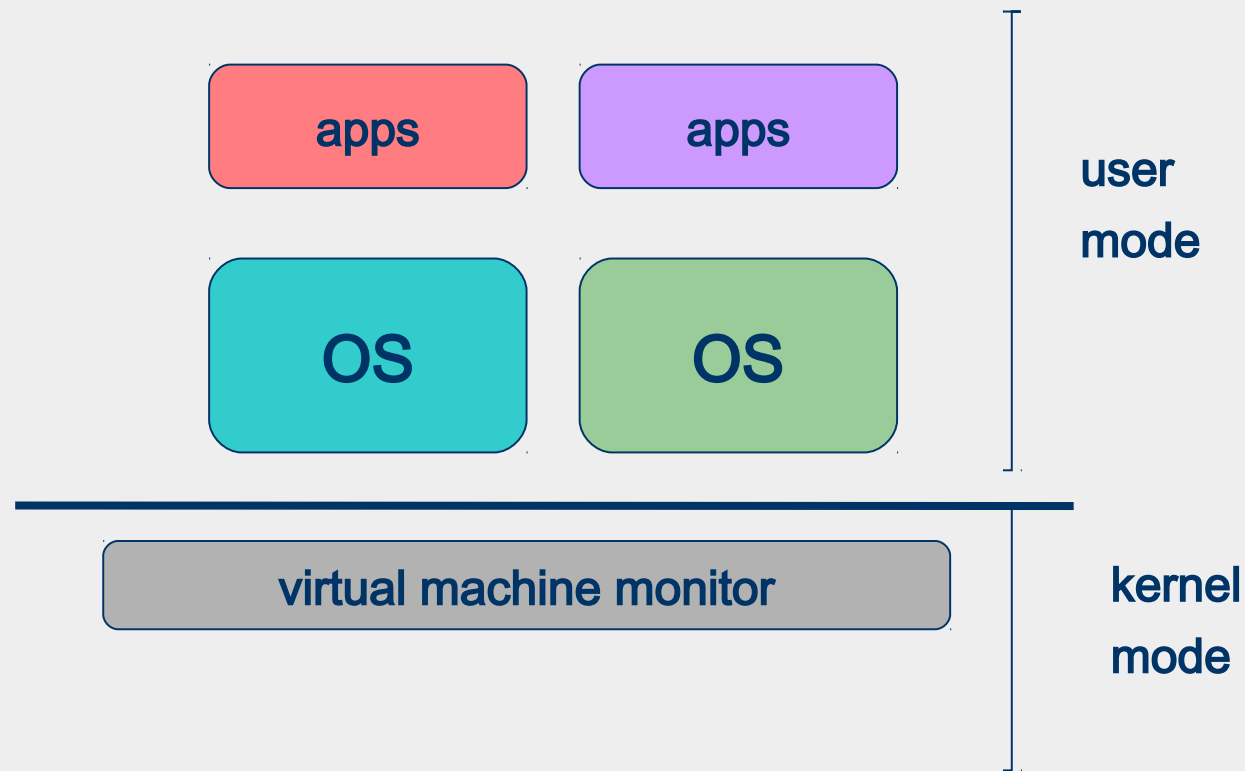
Virtualization = unpriviledging an OS

- 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%

➤ For clients

- Oracle Virtualbox
- QEMU/KVM, etc.

➤ 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

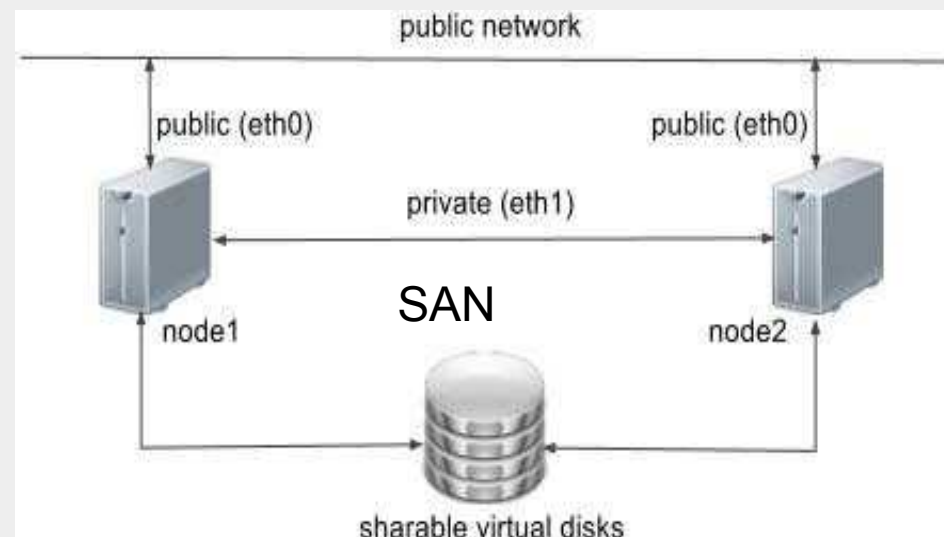
Why virtualizing

- Servers are less expensive but also more powerful
 - ➔ 64 bits multi-core with tons of RAMs are the norm
 - ➔ On 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>

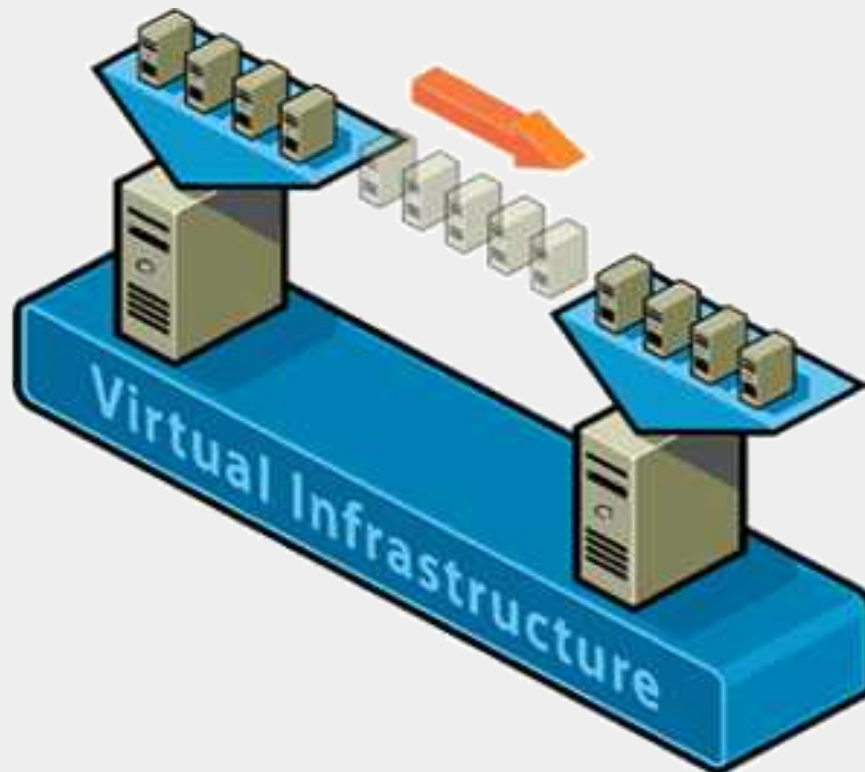


Advantages of virtualization

- 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/>

Example : VMware Vcenter GUI

The screenshot shows the VMware vSphere Client interface for a vSphere Client named 'VCENTER51-3'. The main view is 'Production' with the 'VSA Manager' tab selected. The left sidebar shows a tree view with 'VSA HA Cluster' and its members: 'VSA-0', 'VSA-1', and 'VSA-2'. The main content area displays the 'VSA Cluster Properties' section, which includes a 'VSA Cluster Status' (Online), a 'VSA Cluster Network' (IP: 172.30.0.190, Netmask: 255.255.255.0, Gateway: 172.30.0.1), and a 'Capacity' section (Physical: 36.00 GB, Storage: 18.00 GB, Time since last data update: 00:03). Below this is a 'View' section with 'Datastores', 'Appliances', and 'Map' tabs. The 'Datastores' tab is active, showing a table of VSADs. The table has columns: Name, Status, Capacity, Free, Used, Exported By, Datastore Address, and Datastore Netmask. The table lists three VSADs: VSADs-0, VSADs-1, and VSADs-2, all with a status of 'Online'. Below the table is the 'Datastore Properties' section, which includes 'Name' (VSADs-0), 'Status' (Online), 'Exported By' (VSA-2), 'Datastore Path' (//172.30.0.189/exports/53d7260a-d425-4204-8c63-f752b1edcffe), 'Datastore Network' (IP: 172.30.0.189, Netmask: 255.255.255.0, Gateway: 172.30.0.1, VLAN ID: 0), and 'Capacity' (Total: 5.99 GB, Used: 139.99 MB, Free: 5.86 GB). A circular progress indicator is shown next to the capacity information. The bottom right corner of the interface shows the version 'vSphere Storage Appliance 5.1.0.0'.

VSA Cluster Properties

VSA Cluster Status

Name: vStorage Cluster
Status: ✔ Online

VSA Cluster Network

IP Address: 172.30.0.190
Netmask: 255.255.255.0
Gateway: 172.30.0.1

Capacity

Physical Capacity: 36.00 GB
Storage Capacity: 18.00 GB
Time since last data update: 00:03

View: Datastores Appliances Map

Name	Status	Capacity	Free	Used	Exported By	Datastore Address	Datastore Netmask
VSADs-0	✔ Online	5.99 GB	5.86 GB	139.99 MB	VSA-2	172.30.0.189	255.255.255.0
VSADs-1	✔ Online	5.99 GB	5.86 GB	139.99 MB	VSA-1	172.30.0.187	255.255.255.0
VSADs-2	✔ Online	5.99 GB	5.86 GB	139.99 MB	VSA-0	172.30.0.185	255.255.255.0

Datastore Properties

Name: VSADs-0
Status: ✔ Online
Exported By: VSA-2
Datastore Path: //172.30.0.189/exports/53d7260a-d425-4204-8c63-f752b1edcffe

Datastore Network

IP Address: 172.30.0.189
Netmask: 255.255.255.0
Gateway: 172.30.0.1
VLAN ID: 0

Capacity

Total: 5.99 GB
Used: 139.99 MB
Free: 5.86 GB

VSADs-0

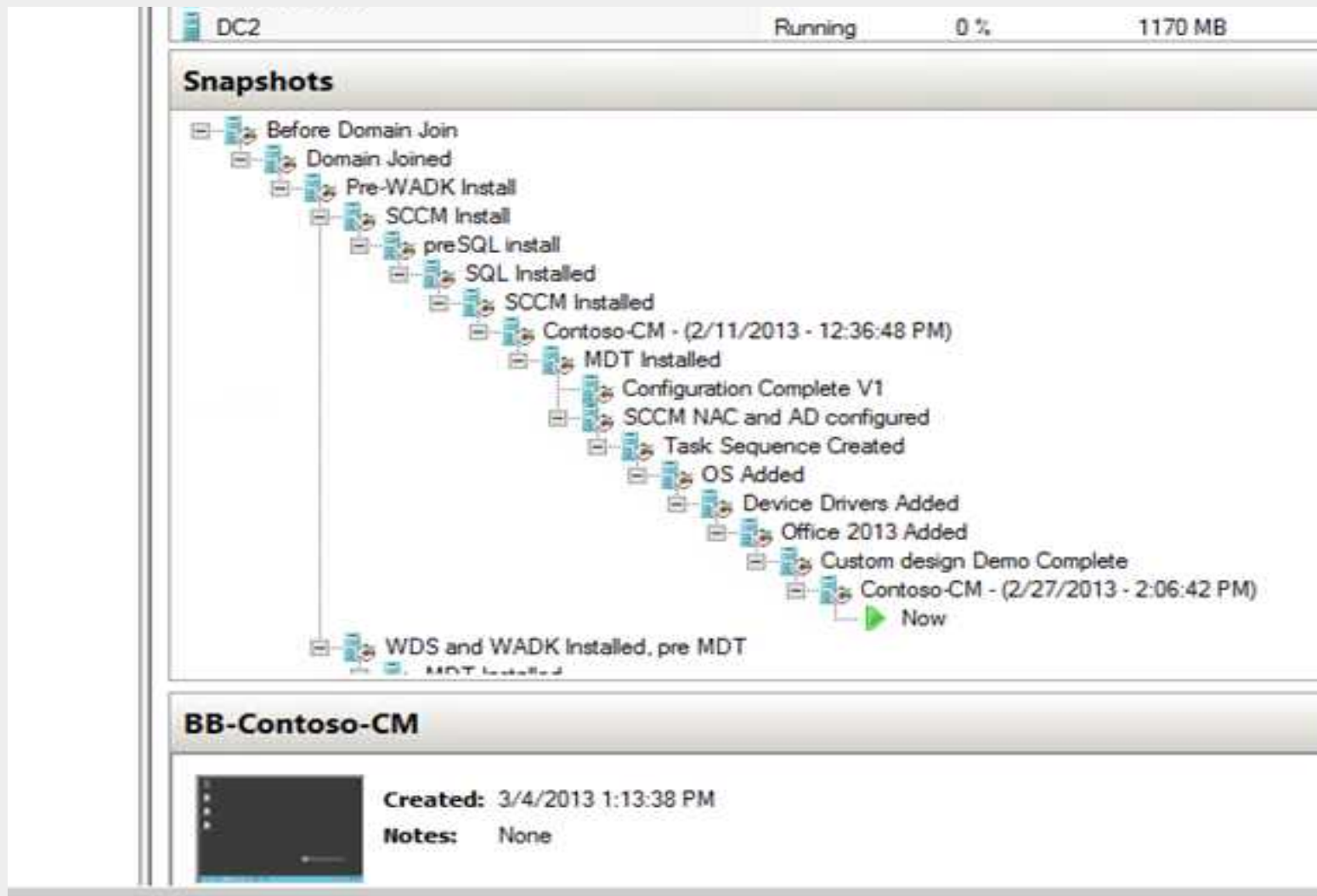
- VSA-2 ✔ Online
- Replica ✔ Online
- VSA-1 ✔ Online
- Replica ✔ Online

vSphere Storage Appliance 5.1.0.0

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



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
 - ➔ Virtual versus para-virtual
 - ➔ Container-based of hypervisor-based

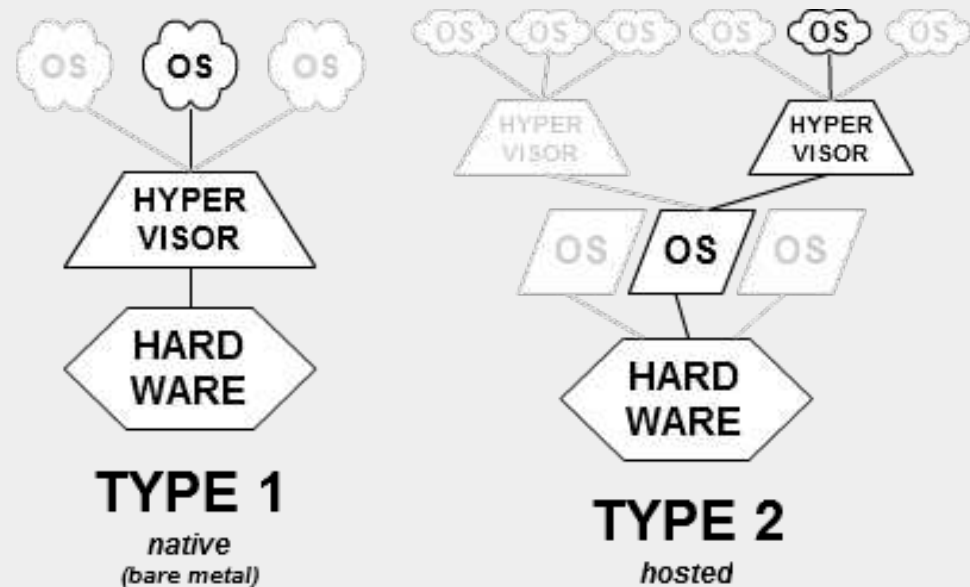
Bare metal vs. host-based

- 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



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

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 box, put 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

- Manufacturers (IBM, HP, DELL, NEC, CISCO,...) indicate which L1 virtualization solution their server support

[Overview](#) [Tech Specs](#) [Services](#) [Solutions](#)

Processor

Intel® Xeon® processor E5-2400 and E5-2400 v2 product families

Processor sockets:
2

Internal interconnect:
Intel QuickPath Interconnect (QPI) link: 6.4GT/s; 7.2GT/s; 8.0 GT/s

Cache:
2.5MB per core; core options: 4, 6, 8, 10

Operating System

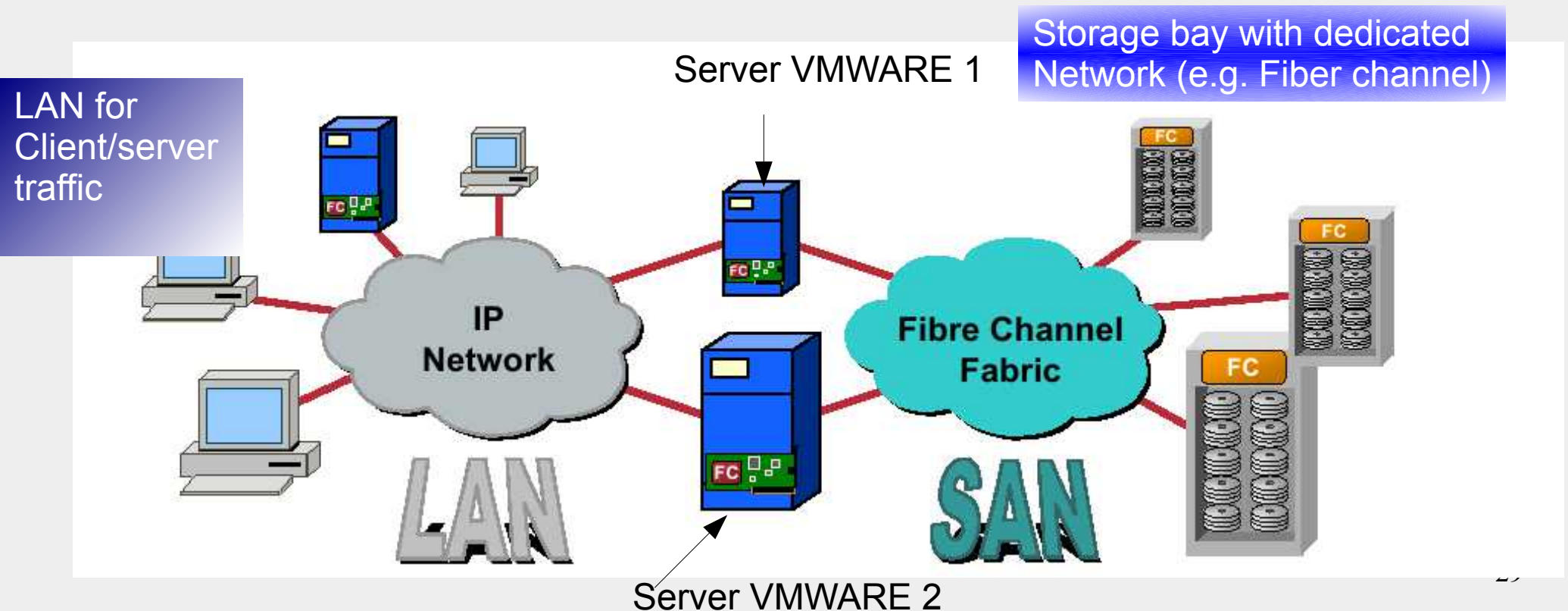
Microsoft® Windows Server® 2012
Microsoft Windows Server 2008 R2 SP1, x64 (includes Hyper-V®)
Novell® SUSE® Linux Enterprise Server
Red Hat® Enterprise Linux®

Virtualization options:
Citrix® XenServer®
VMware® vSphere® ESXi™
Red Hat Enterprise Virtualization®



Around virtualization

- Storage:
 - ➔ Virtualization → consolidation of servers
 - ➔ Often associated with storage consolidation stockage



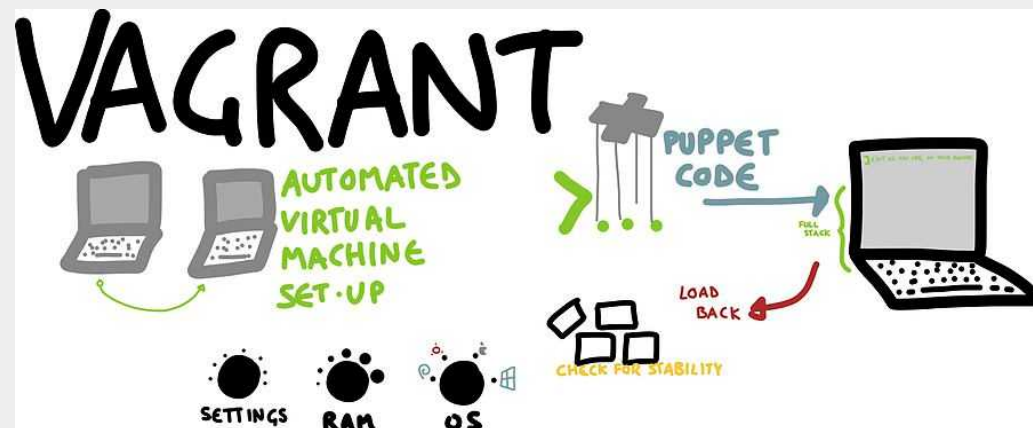
Around virtualization: hardware manufacturers

- Intel and AMD propose various solutions to support virtualization at hardware level
- Intel VT-x, AMD-V
- More detail at the end of part II

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

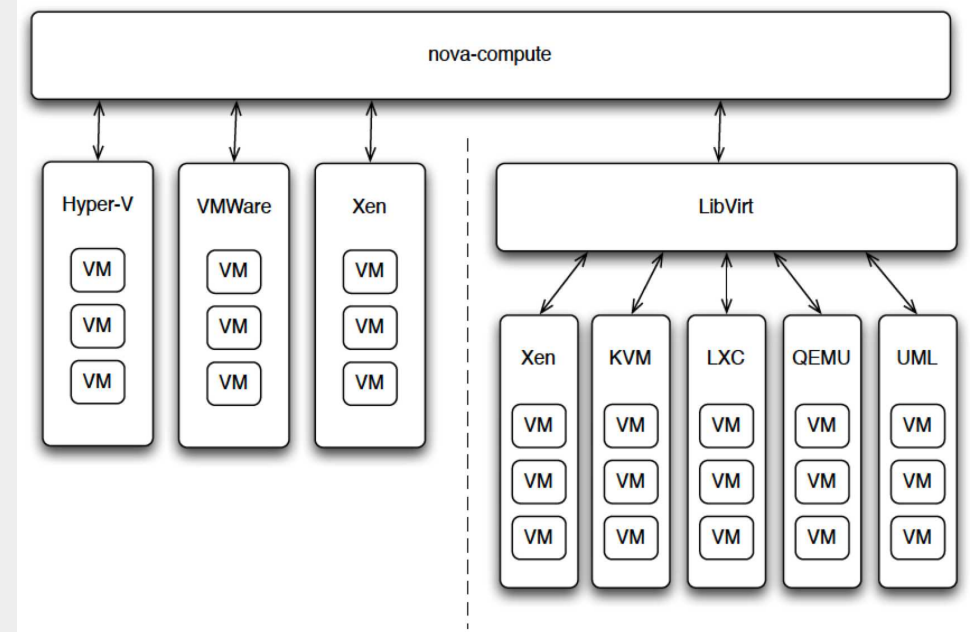
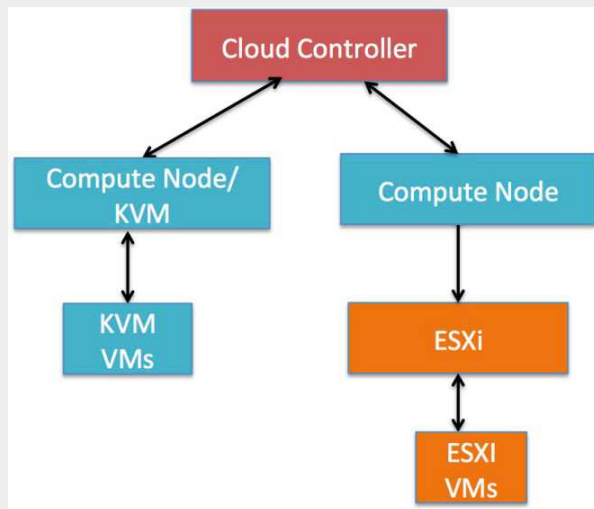
  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  config.vm.network "forwarded_port", guest: 5001, host: 5001

  # 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

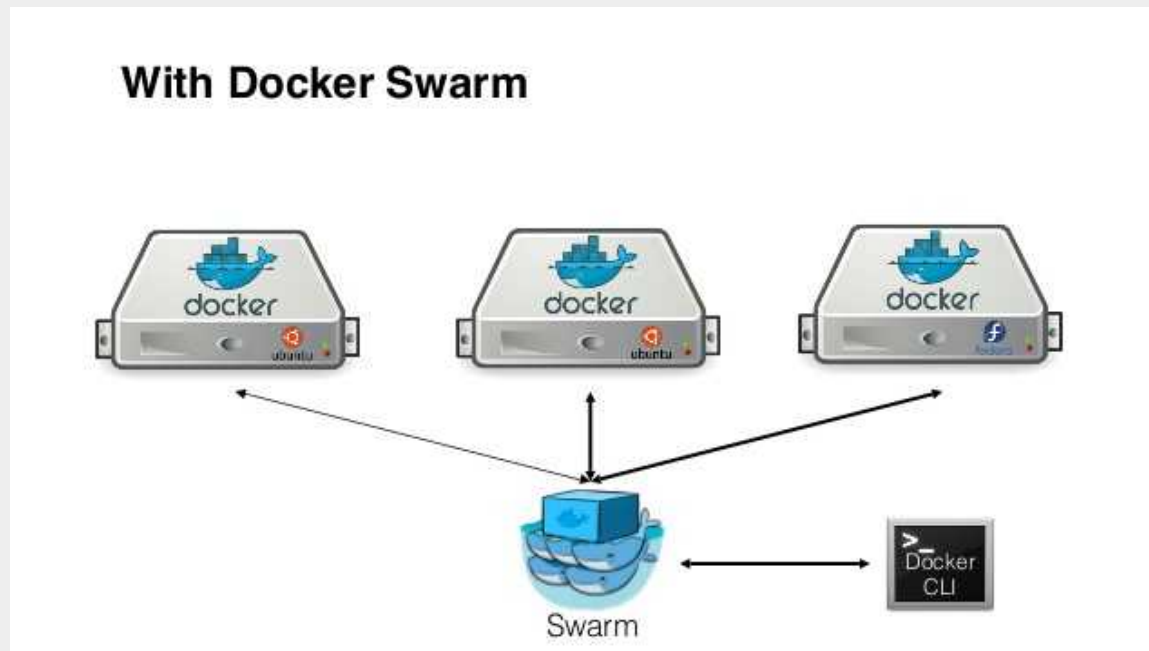
➤ Orchestration of containers



- ➔ Single server level: Docker, Vsphere, LXC

- ➔ Several servers level: Docker Swarm

➤ Advanced orchestration: Kubernetes, Swarm, Mesos



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

System VM vs. Process VM

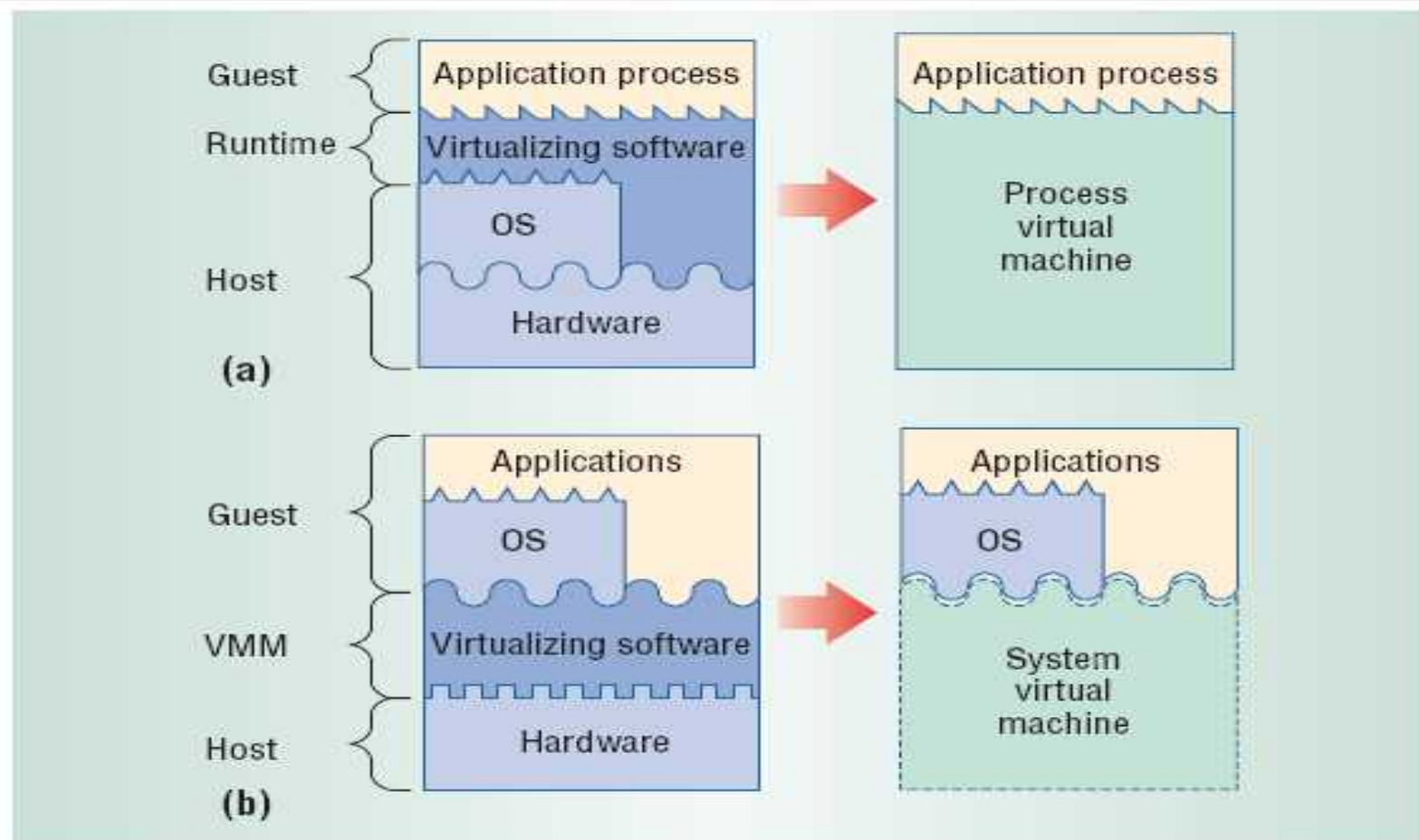
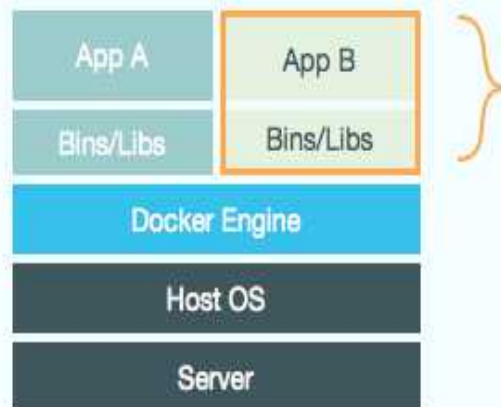


Figure 3. Process and system VMs. (a) In a process VM, virtualizing software translates a set of OS and user-level instructions composing one platform to those of another. (b) In a system VM, virtualizing software translates the ISA used by one hardware platform to that of another.

Process VM

- OS can be seen as VM process
 - ➔ Indeed, they provide each process the “illusion” to control the whole hardware resources
 - ➔ → OS enables time-sharing of resources and provides a process VM to each application
- Container-based approaches are process VMs



Docker

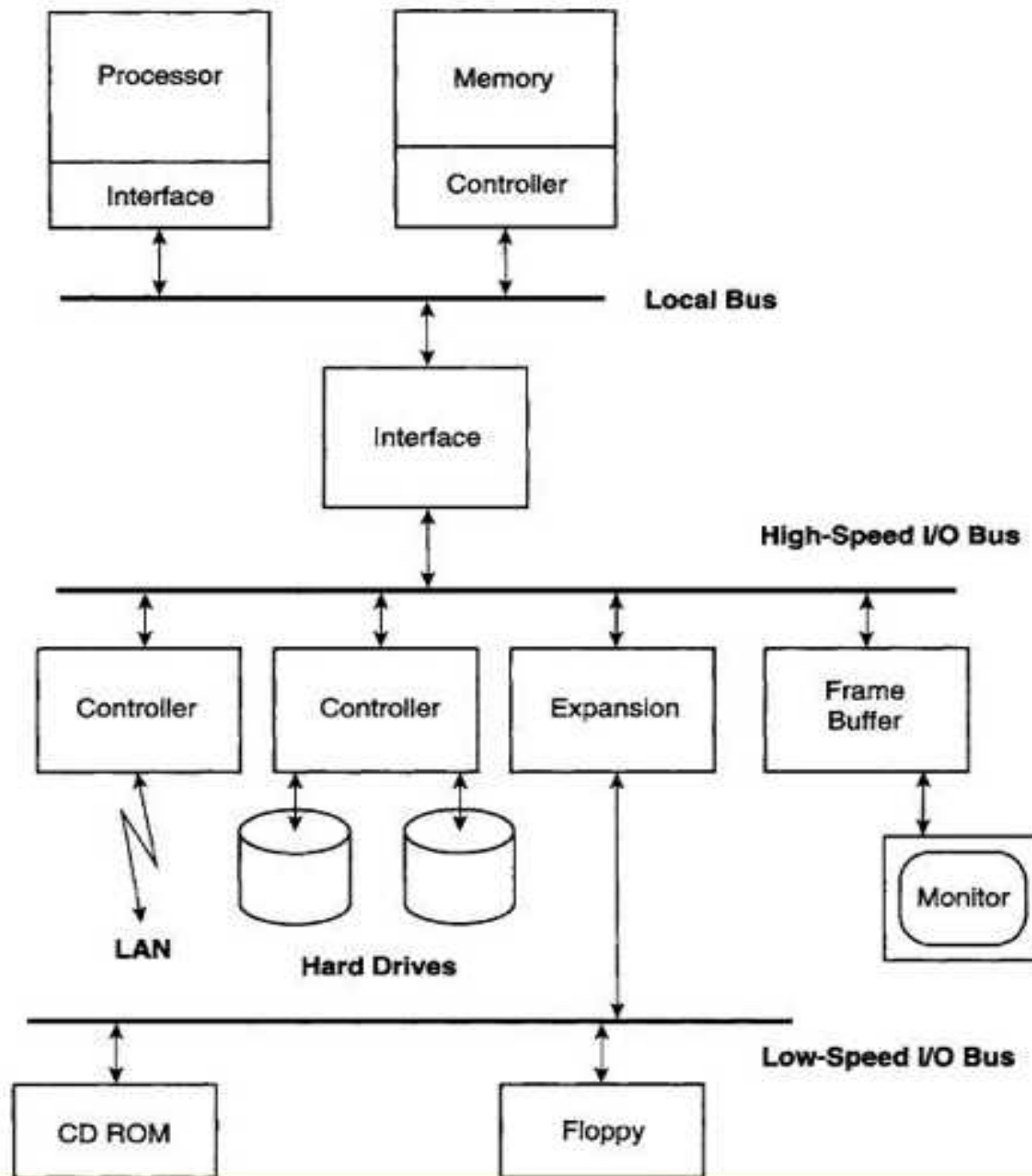
The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

System level VM

- L1 and L2 hypervisors
- Guest OS runs in unprivileged mode
→ root of the problem (to be solved by hypervisors)
- Need a refresher on computer architecture and OS to understand the challenges faced by hypervisors

Refresher on computer architecture and OS

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



Processor

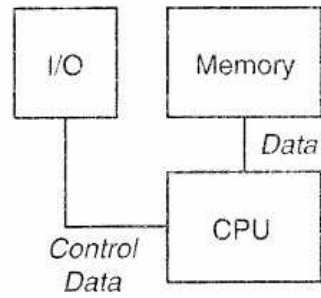
- Fetches instruction in RAMs and execute them
- Each instruction has to go through a set of execution steps:
 - Reading from RAM
 - Decoding
 - Loading instruction and operands to registers
 - Execution
 - Storing result into RAM again

Processor

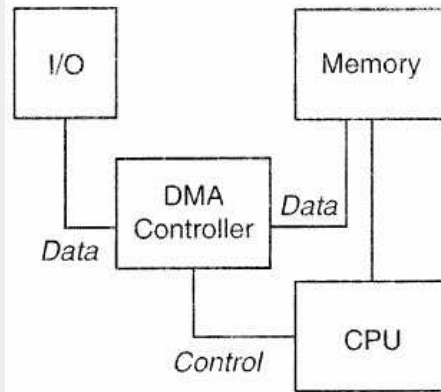
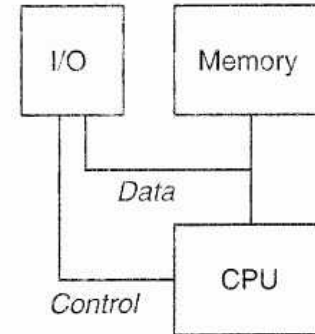
- Set of steps → instructions are pipelined
- Pipeline enables continuous operation of processor even if one instruction stalls, e.g., waiting for memory read
- Stalled events still occur
 - Ex: one instruction in the pipe waiting for the result of another instruction
- This is the job of the compiler to decide about the best possible ordering of instructions

Inputs/Outputs

- Interruption based I/O: I/O controller tells OS when data is ready
 - × CPU can continue computations...
 - × But must move data from controller to RAM
- DMA I/O (Direct Memory Access)
 - ➔ Controller copies data to RAM directly
 - ➔ Very common nowadays



(a)



(c)

I
7

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

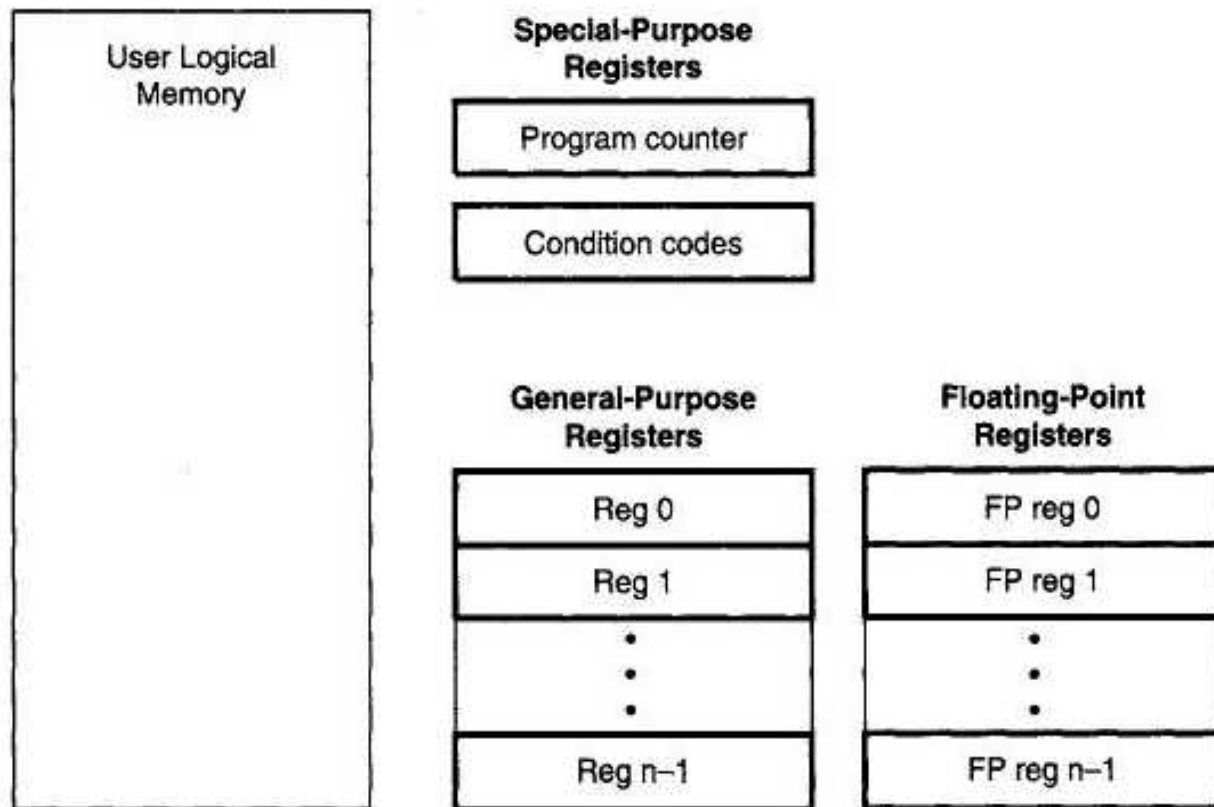


Figure A.6 Key Architected User State of a Typical ISA. *This includes a logical (virtual) memory space, a special-purpose register, general-purpose registers, and floating-point registers.*

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

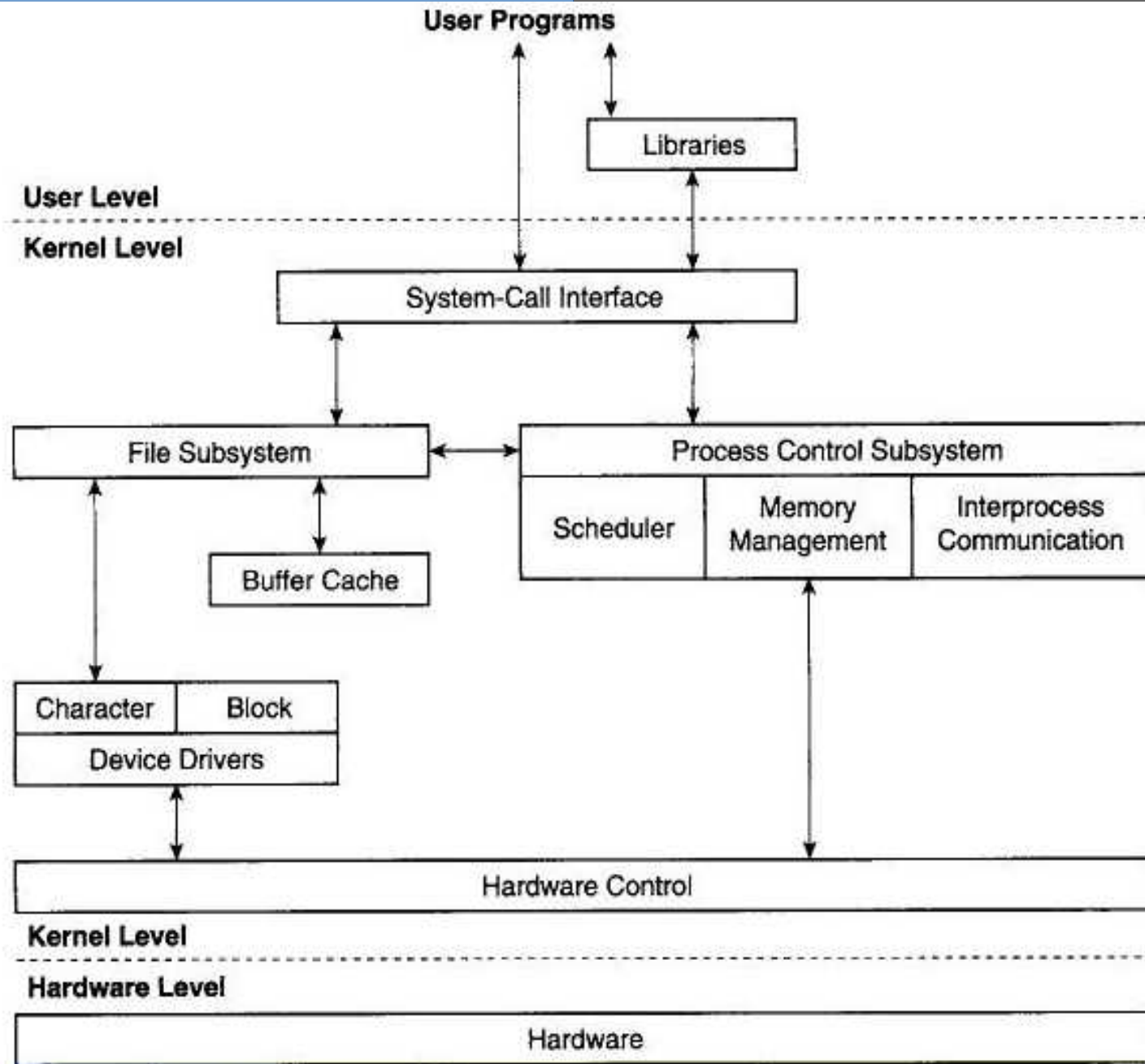


Figure A.12 Linux Architecture.

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

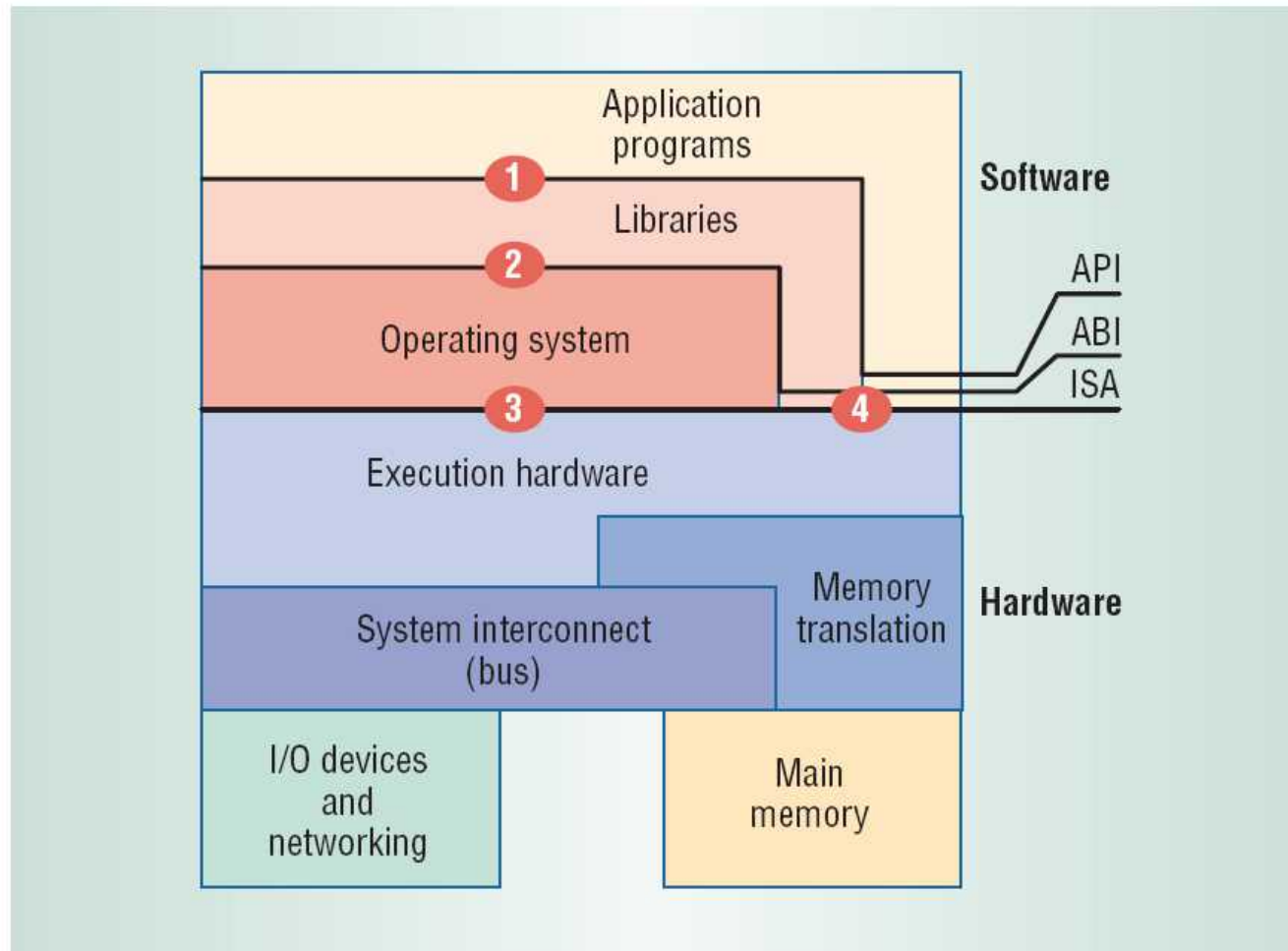
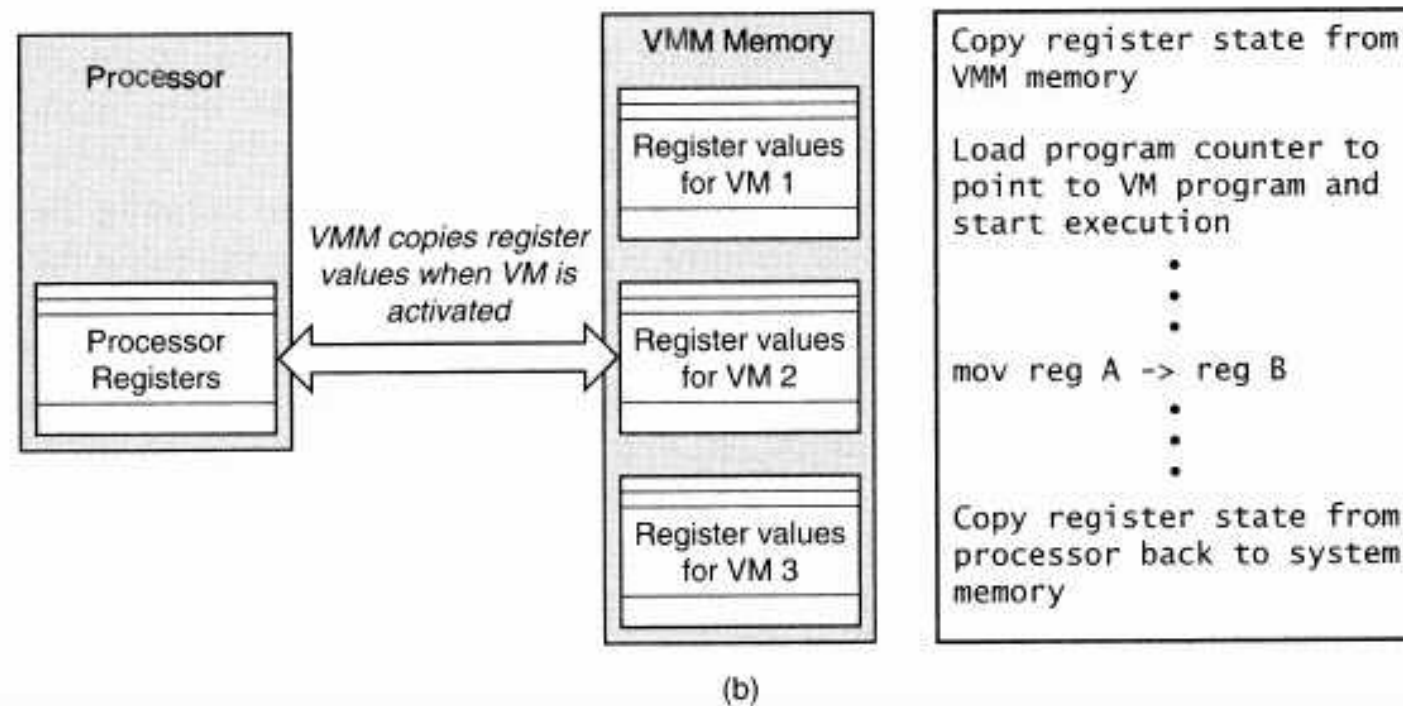
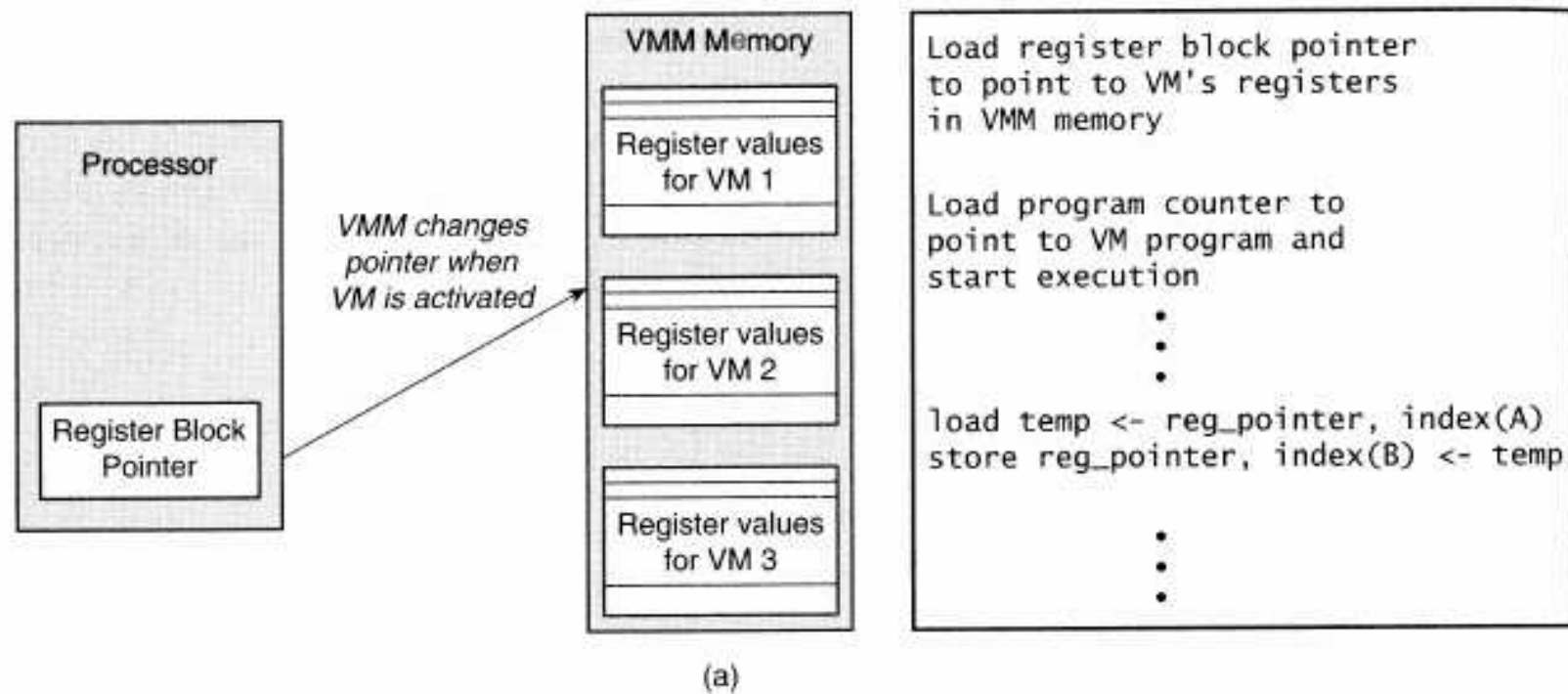


Figure 2. Computer system architecture. Key implementation layers communicate vertically via the instruction set architecture (ISA), application binary interface (ABI), and application programming interface (API).

L1 and L2 hypervisors from the inside

Management of VM states

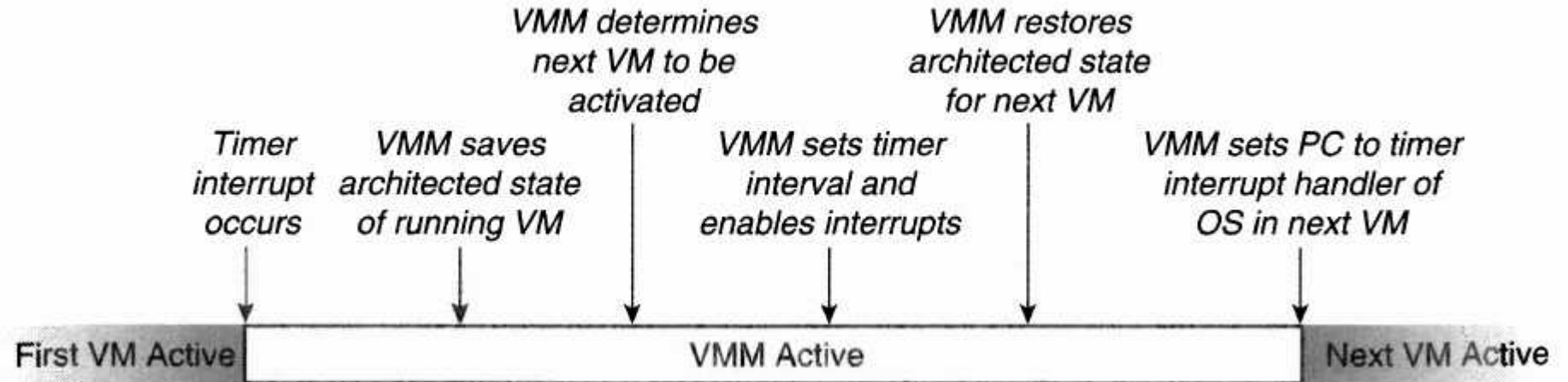
- Hypervisor must store complete VM state when switching from one VM to the next VM :
 - ➔ Registers
 - ➔ Page table (of guest OS)
 - ➔ I/O
- Two main options for context switching:
 - ➔ Indirection – inefficient because operation between register imposes several memory accesses
 - ➔ Copy – possible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

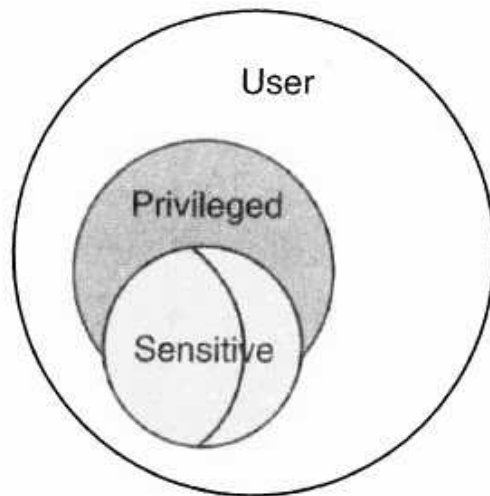
- Article Popek and Goldberg 1974
 - ➔ Virtualization dates back to the mainframe age
- Hypotheses (short version) :
 - ➔ Hardware = 1 (unique) processor
 - ➔ Processor has 2 modes : system and user
 - ➔ Some instructions can only be executed in kernel mode
- I/O non considered in article, but results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

Sensitive vs. non sensitive instructions

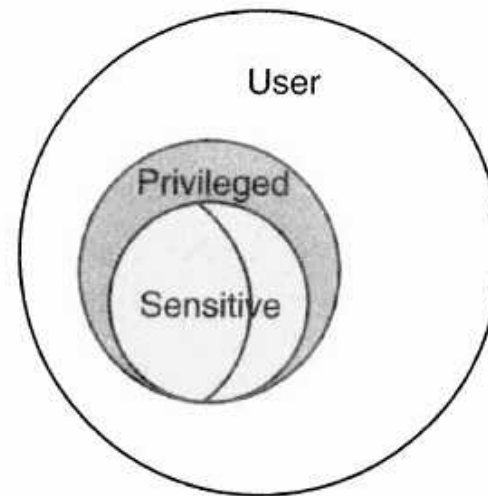
- Sensitive instructions:
 - × Modify the configuration of resources, e.g. memory or execution mode (control type)
 - × Ex : SPT - Set CPU timer (IBM/370)
 - × Whose result depend on resource state (behavioral type)
 - × Ex : Load Real Address (IBM/370)
 - ×
- Instructions that are neither of control 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

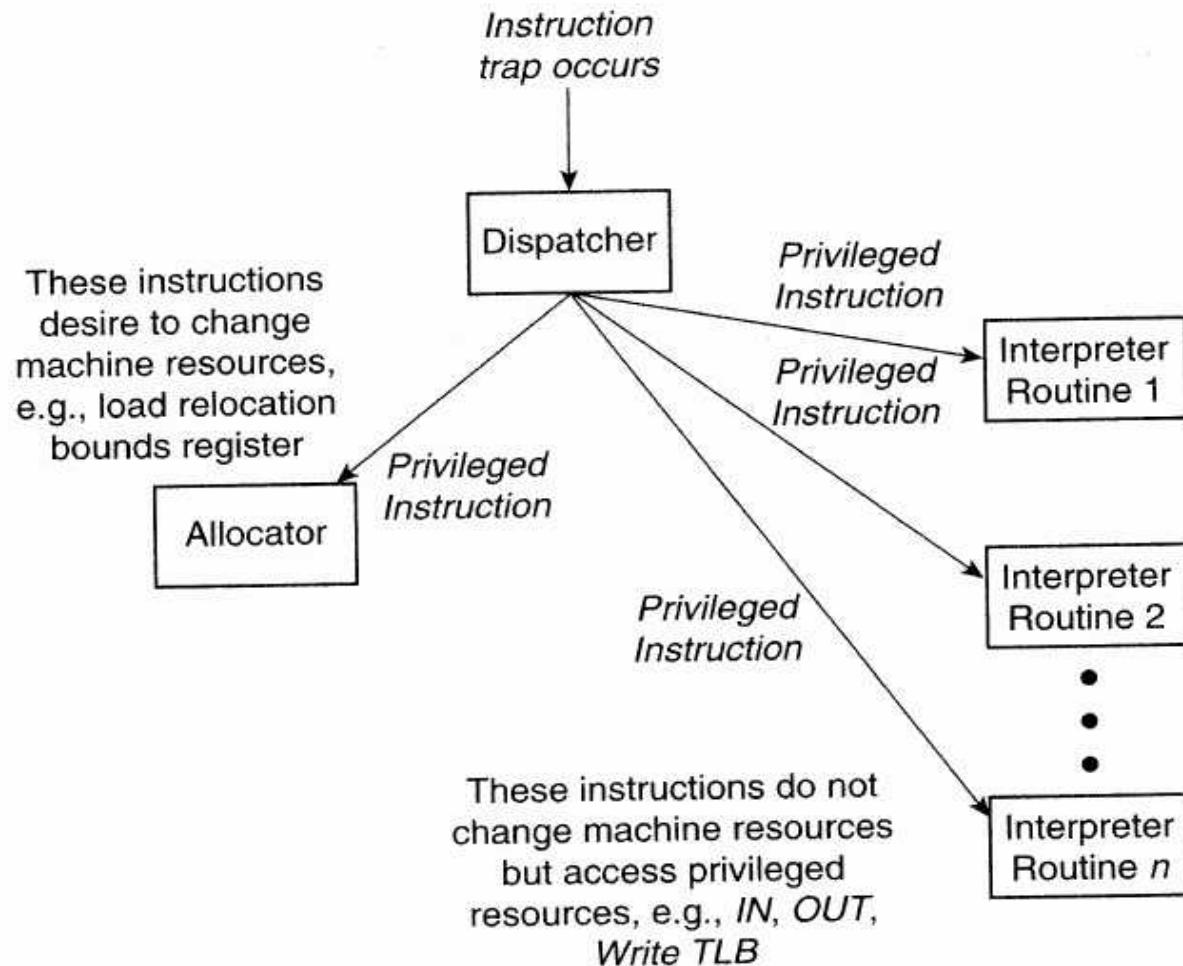


(b) Satisfies condition —
efficiently virtualizable

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)
- This ensures equivalence : a program has a similar execution in real or virtual mode

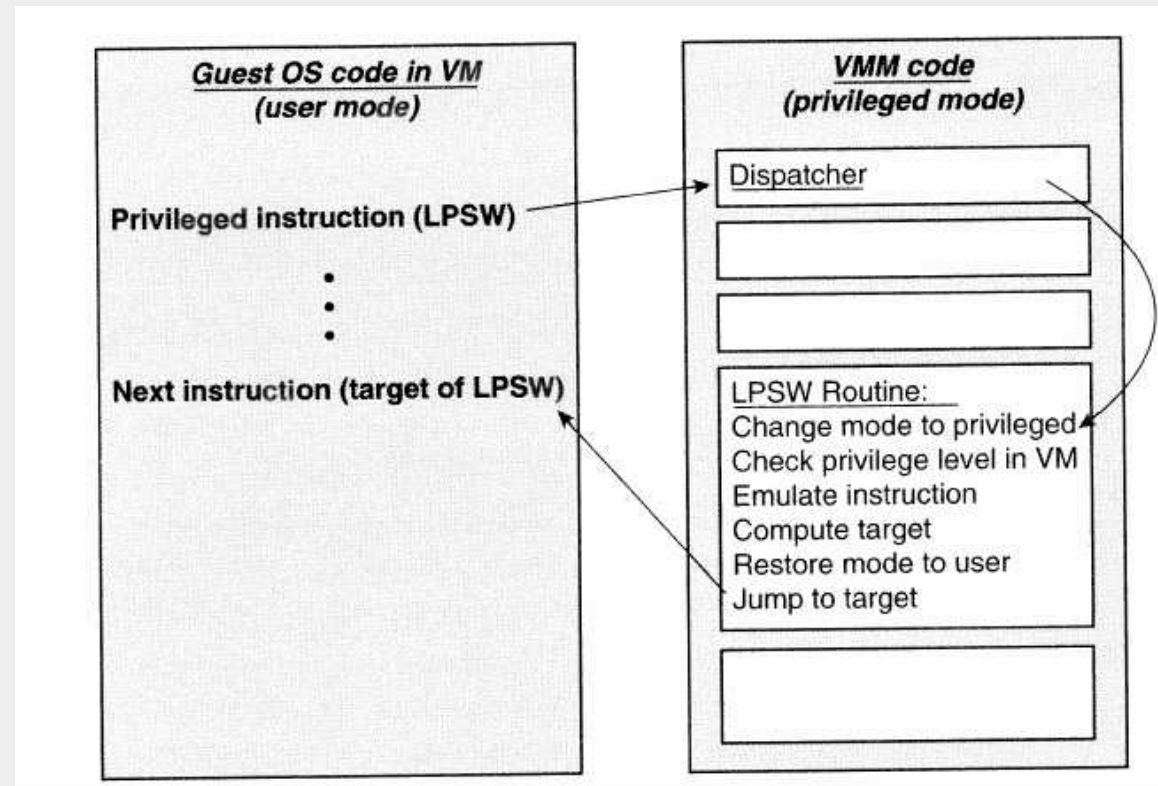
Job of hypervisor if Popek and Goldberg conditions are met: Wait for traps!!!



Intel ISA x86 case

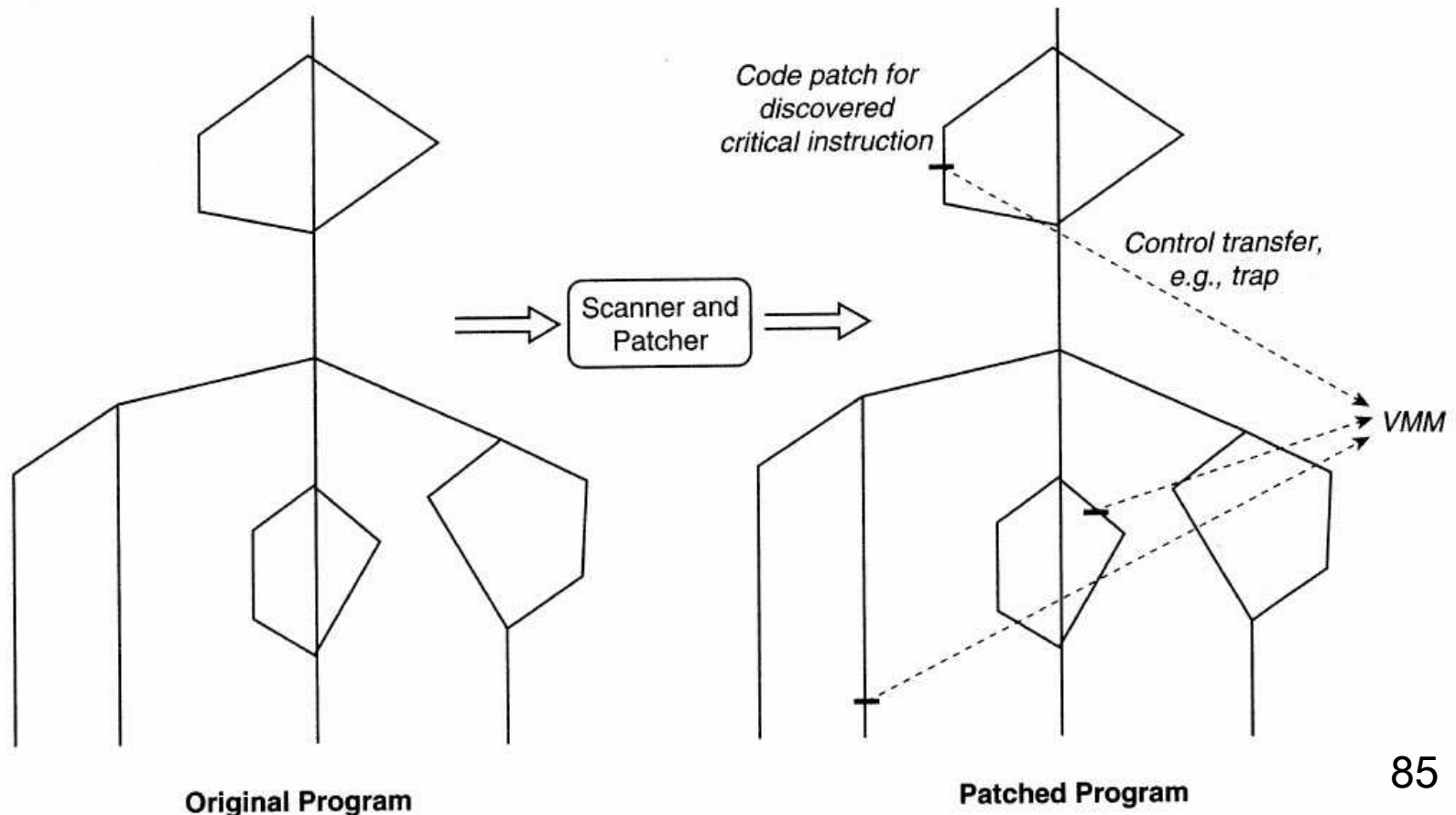
- “Analysis of the Intel Pentiums Ability to Support a Secure Virtual Machine Monitor” USENIX 2000, J. Robin, C. Irvine
- Over the 250 instructions, 17 are sensitive but not privileged (they are called critical)
 - ➔ Example: popf
 - × Popf in user mode: change ALU flag
 - × Popf in kernel mode: change 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”



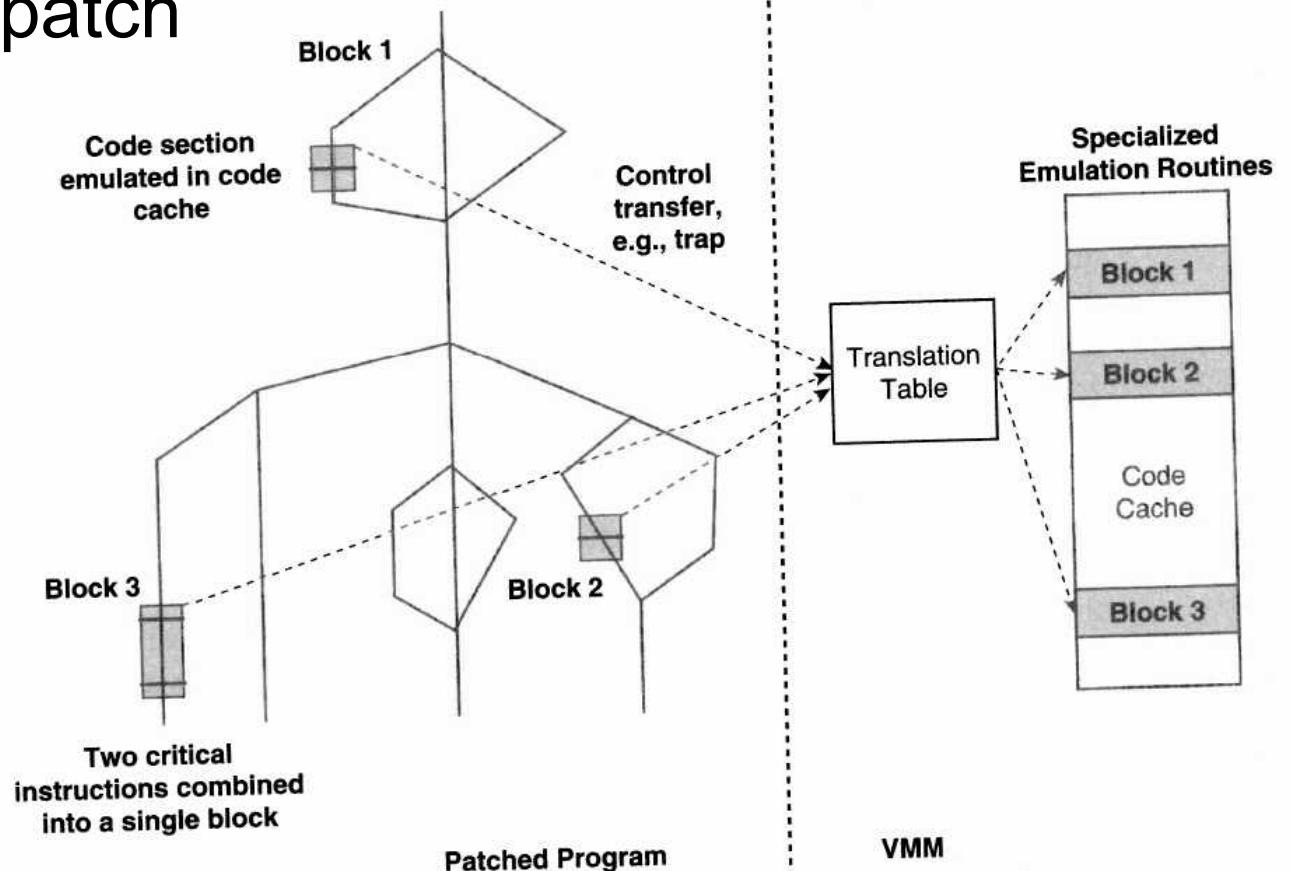
Hypervisor job: track critical instructions

- 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



Patching Optimization

- ▶ Caching of patched code
- ▶ Cache locality should help: code will need several times the same patch



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
 - Caching of patches might help
- Above operation is called Binary Translation
 - VMware is an expert in this area

I/O Virtualization

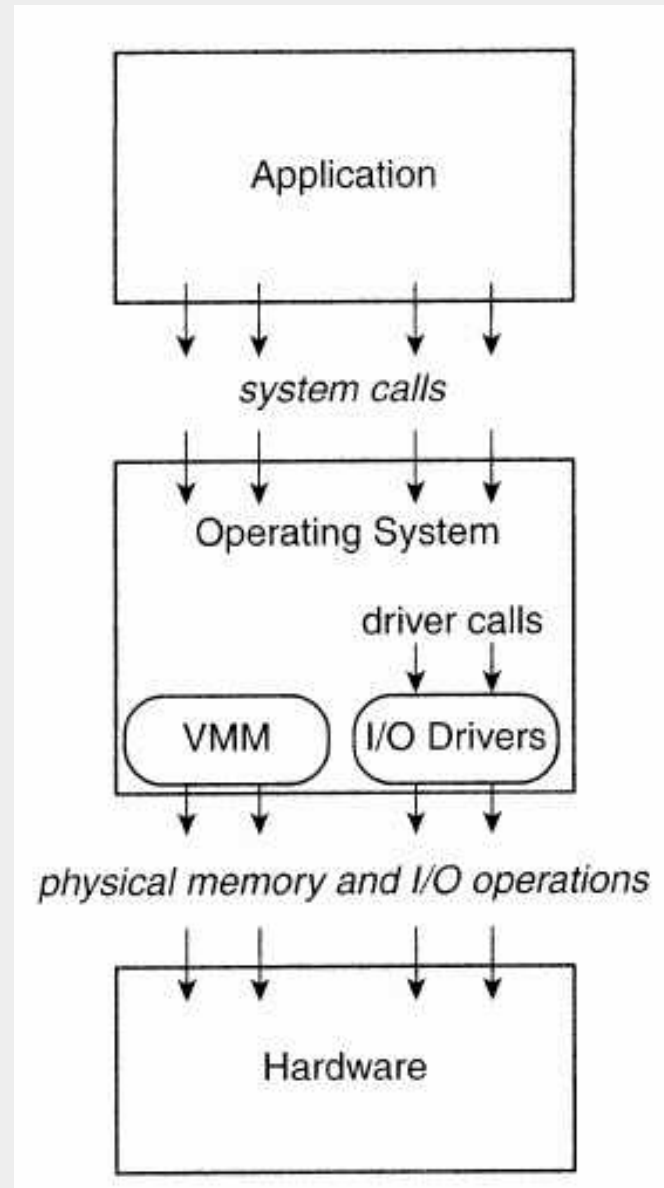
- ▶ One of the most complex tasks in virtualization because there is a LOT of I/O devices
- ▶ General approach: offer a clean interface to guest OS

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

Example of a layer 2 hypervisor:
VMware player

Introduction

- IA-32 architecture was initially dedicated to low-end micro-processor
 - ➔ No support for multi-user in mind
 - ➔ What is more, PC architecture was open → a host of peripherals!
- In contrast, the IBM team that was virtualizing the System/370 mainframe was controlling both the hardware and software !

Introduction

- VMware had many challenges:
 - No control on hardware development
 - No control on software development (Windows, Linux)
- Starting with a L2 hypervisor was a WISE strategy

Hosted Architecture

➤ 3 components :

➔ VMM-n

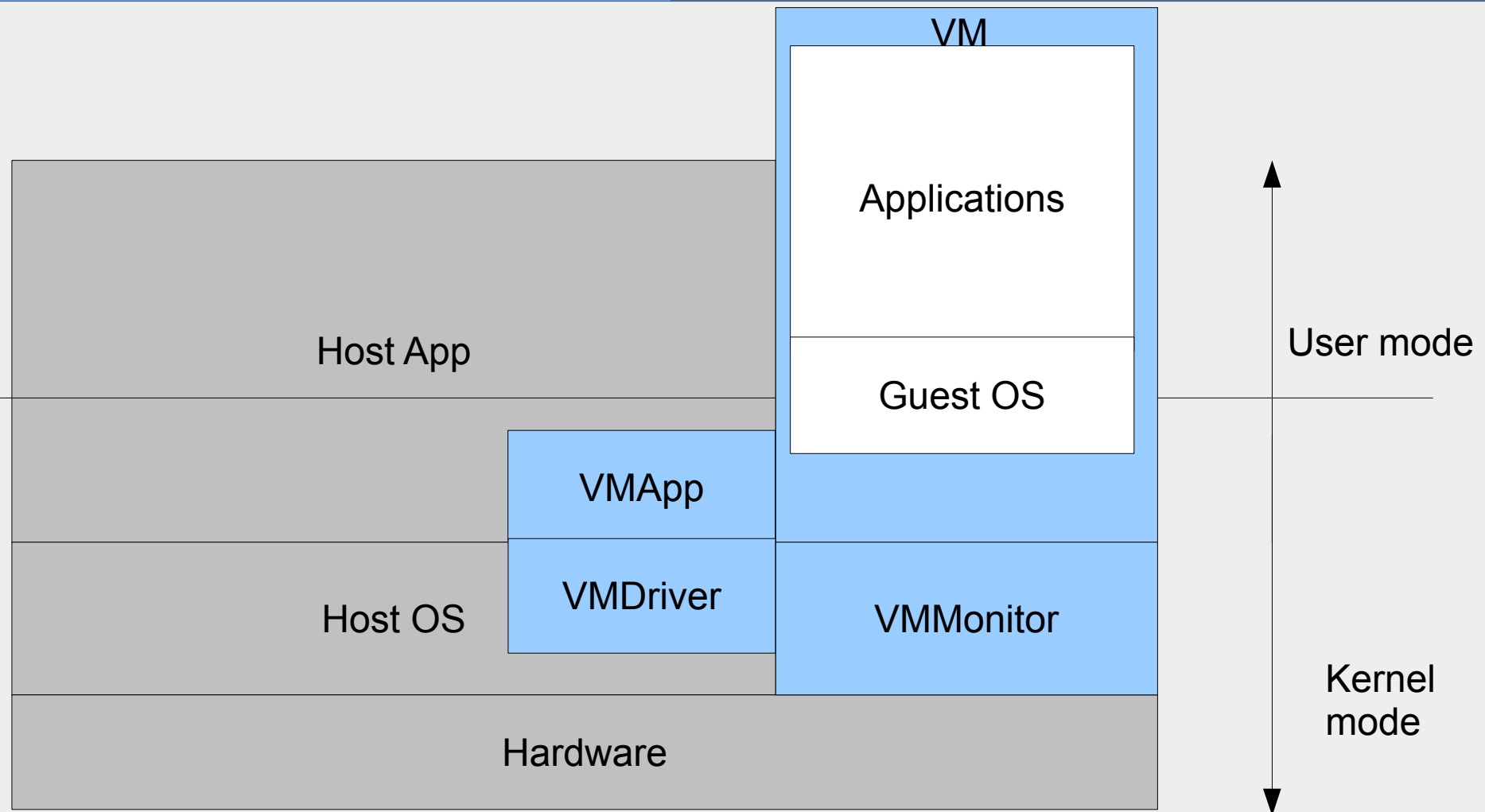
- × n = native
- × This is the hypervisor
- × Kernel module, operates in kernel mode

➔ VMM-u

- × u = user
- × This is the application you launch

➔ VMM-d

- × d= driver
- × Enable communication between the two wolrd



- VMMonitor can “morph” from guest OS to an application that requests the OS to send a command to a peripheral thanks to VMdriver⁹⁸

Processor support for virtualization

Processor Virtualization

- Article Robin and Irvine (2000) → 17 critical instructions in Intel IA-32 ISA
- Example : `mov ax, cs`
 - ➔ `cs` register contains the privilege level (bits 0 and 1 – 4 levels to encode in)
 - ➔ If kernel mode, `ax` copied to `cs`
 - ➔ Otherwise, it is interpreted as a no-op (hence no trap!!!)
 - ➔ VMware must scan code for this kind of instructions

Processor Virtualization

- An alternative approach would be to have hardware support
 - no change to ISA since Windows and Linux are still there
- Intel and AMD both offer such technologies
 - Intel VT-x (Vanderpool)
 - AMD-v
- Current processors feature this technology

Intel VT-x

- New functionality : VMX mode
 - ➔ Non VMX mode : for non virtualized P. Use of the 4 usual levels 0(OS),1,2,3 (Appli)
 - ➔ VMX mode
 - × VMX root mode for hypervisor
 - × Non-root mode for guest OS/user application
 - × In both modes, 4 levels: 0,1,2,3

Intel VT-x

- VMX Mode = Non VMX Mode + specific instructions for virtualization support
- New instructions
 - ➔ vmxon and vmxoff to start/stop VMX mode
 - ➔ vmlaunch and vmresume for VMs start-up/resume

Intel VT-x

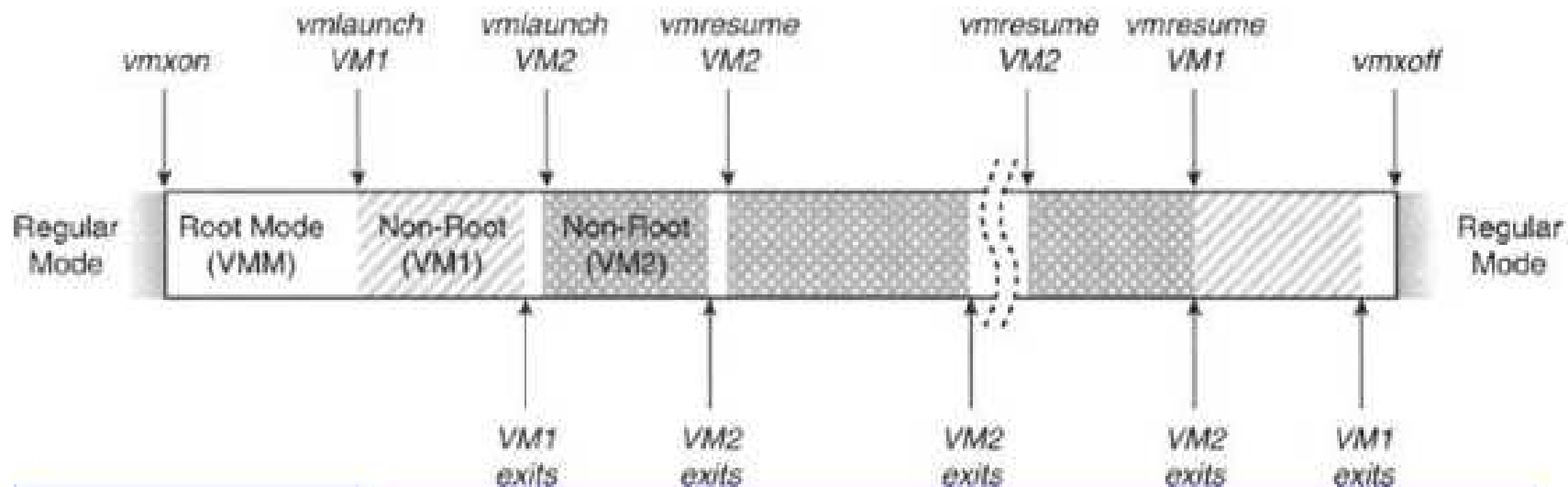


Figure 8.26 Transition Between Operation Phases with the Intel VT-x Technology. *White areas represent the operation in root mode of the virtual machine monitor (VMM). The cross-hatched and dotted areas represent the regions in time when the two active virtual machines are executing.*

Intel VT-x

- Guest OS runs in VMX non-root with privilege level 0
 - ➔ Enables more native operations, esp. on registers
 - × Due to hardware offering VMCS (Virtual Machine Control Structure) with per VM register state
 - × Enables guest OS to work natively with registers
 - × Hardware dumps states at each VM switching
 - ➔ There still remain operations, esp I/O, where hypervisor must take over

Part III: OpenVswitch

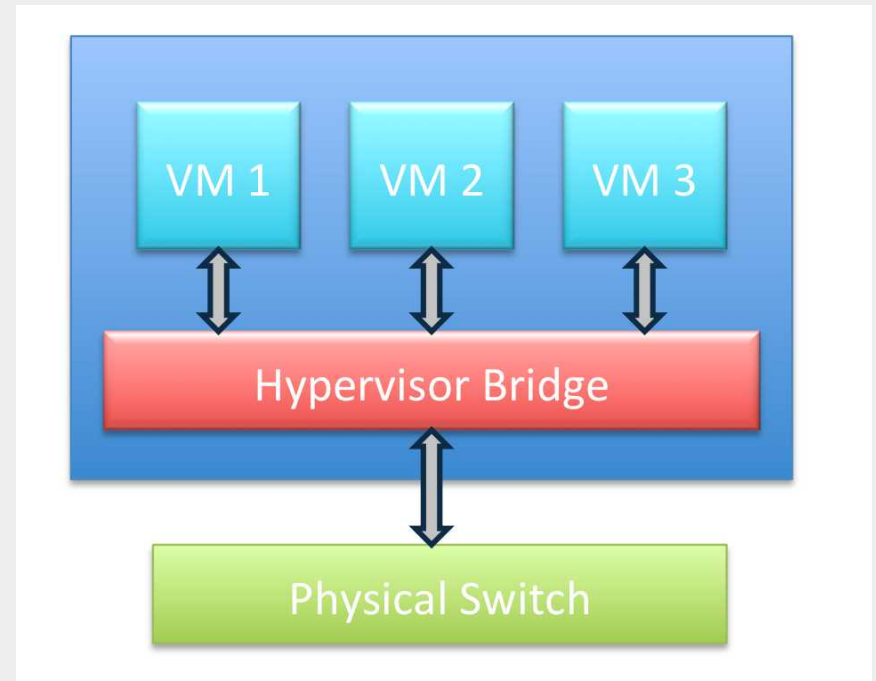
sources :

<http://openvswitch.org/slides/LinuxSummit-2011.pdf>

<http://openvswitch.org/slides/OpenStack-131107.pdf>

Introduction

- Multiple VMs on same host
 - ➔ Sharing of physical interfaces
- Works on Linux-based hypervisors: Xen, KVM, VirtualBox
- Open source, started in 2009



List of contributors



Packaging

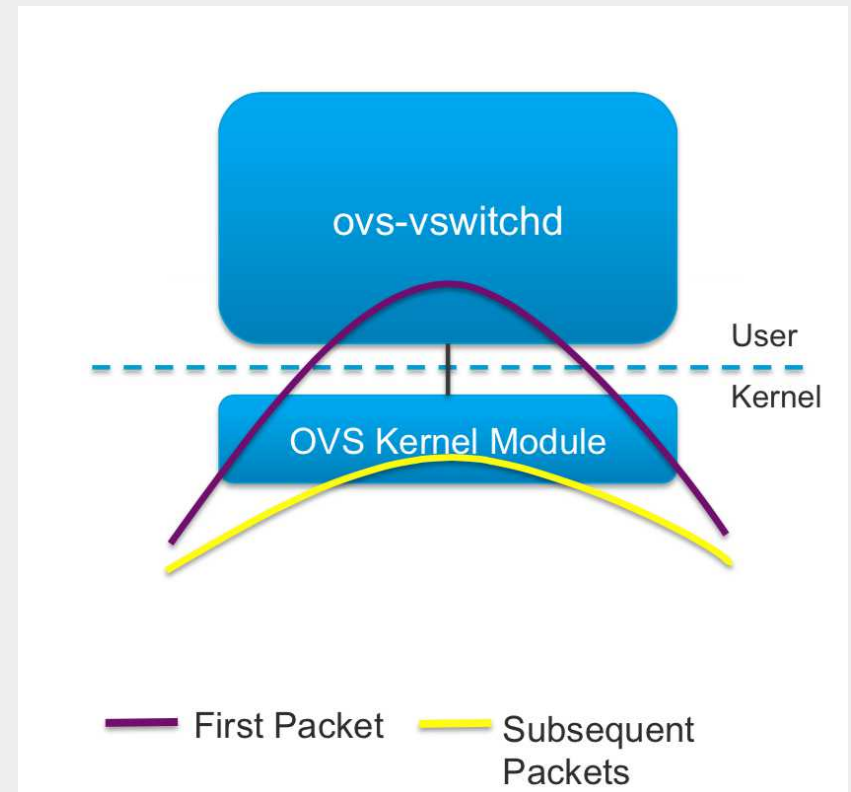
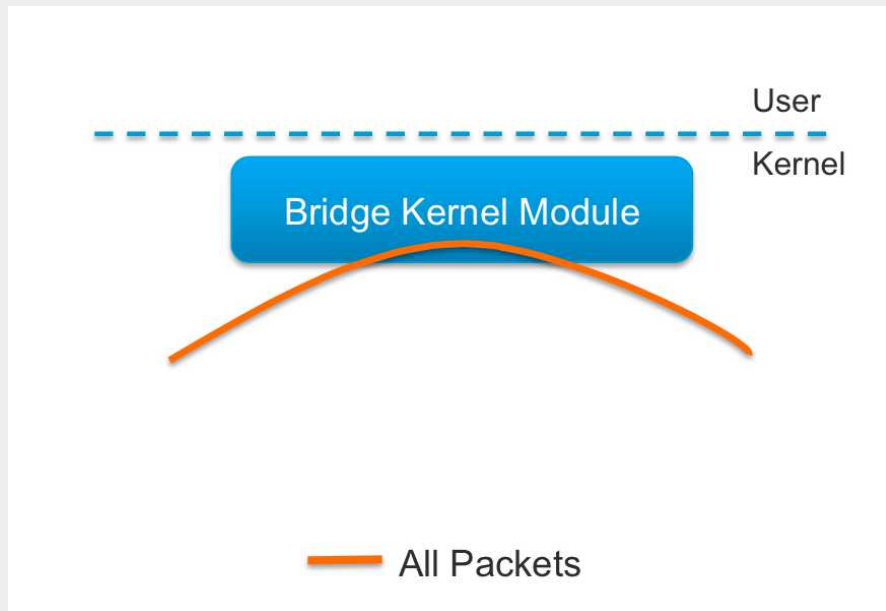
- Default networking stack for Xen Cloud Platform (XCP)
- Ships with Citrix XenServer and basis for their Distributed Virtual Switch (DVS)
- Distribution packaging
 - Debian
 - Ubuntu – SUSE
 - Red Hat

Functionalities

- Network protocols:
 - ➔ Openflow: centralized controller that controls several switches and installs flow rules (matching L2 to L4 fields)
 - ➔ Alternatively can work as normal L2 switch
 - ➔ LACP (NICs teaming)
 - ➔ VXLAN:
 - × VLANs at large scale in data centers
 - × Based on UDP tunnels with smart end-points
 - × Promoted by CISCO and VMware
- Reporting:
 - ➔ Netflow, sFlow (sampled netflow)

L2 vs Openflow

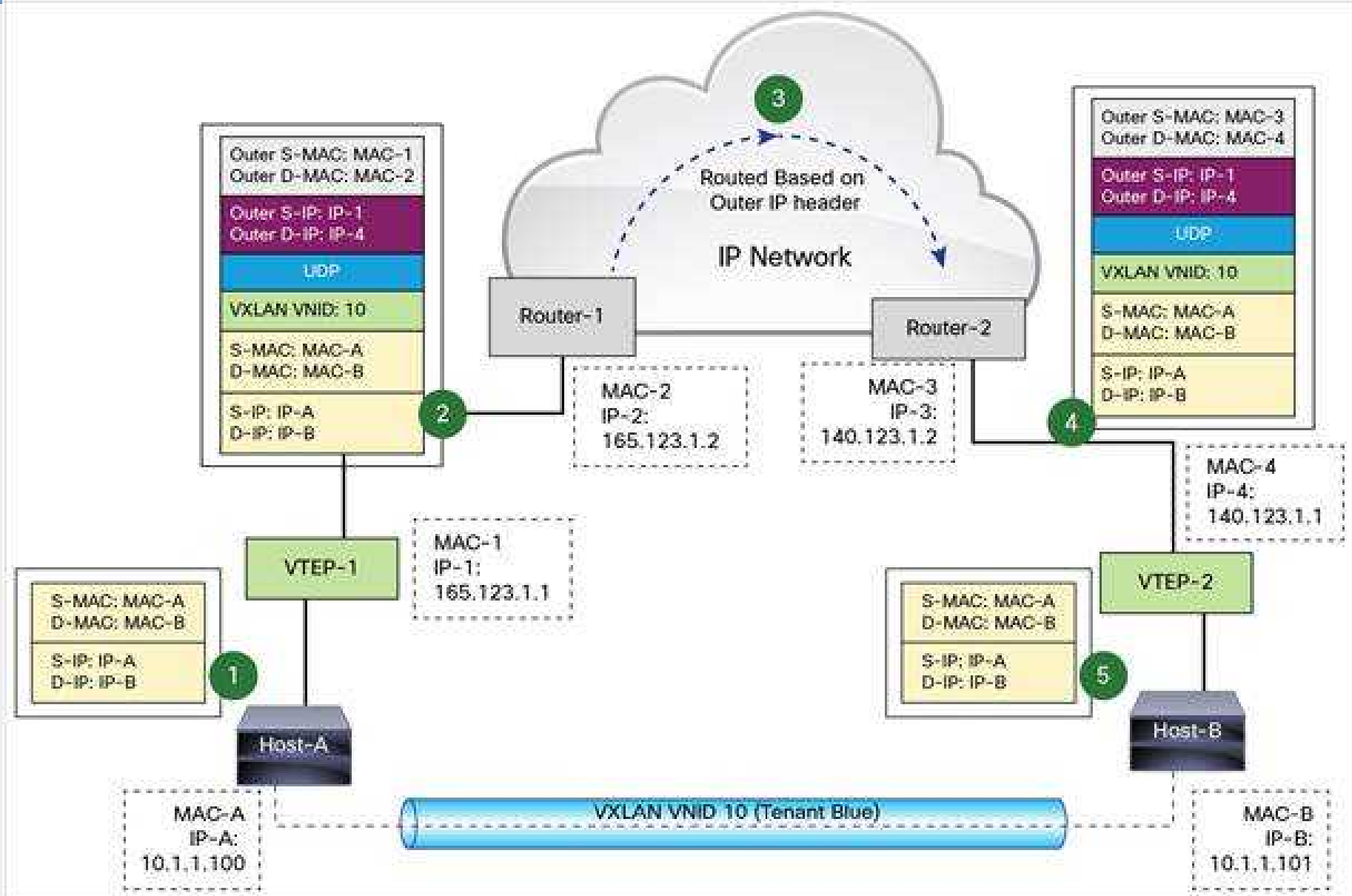
➤ L2 legacy forwarding



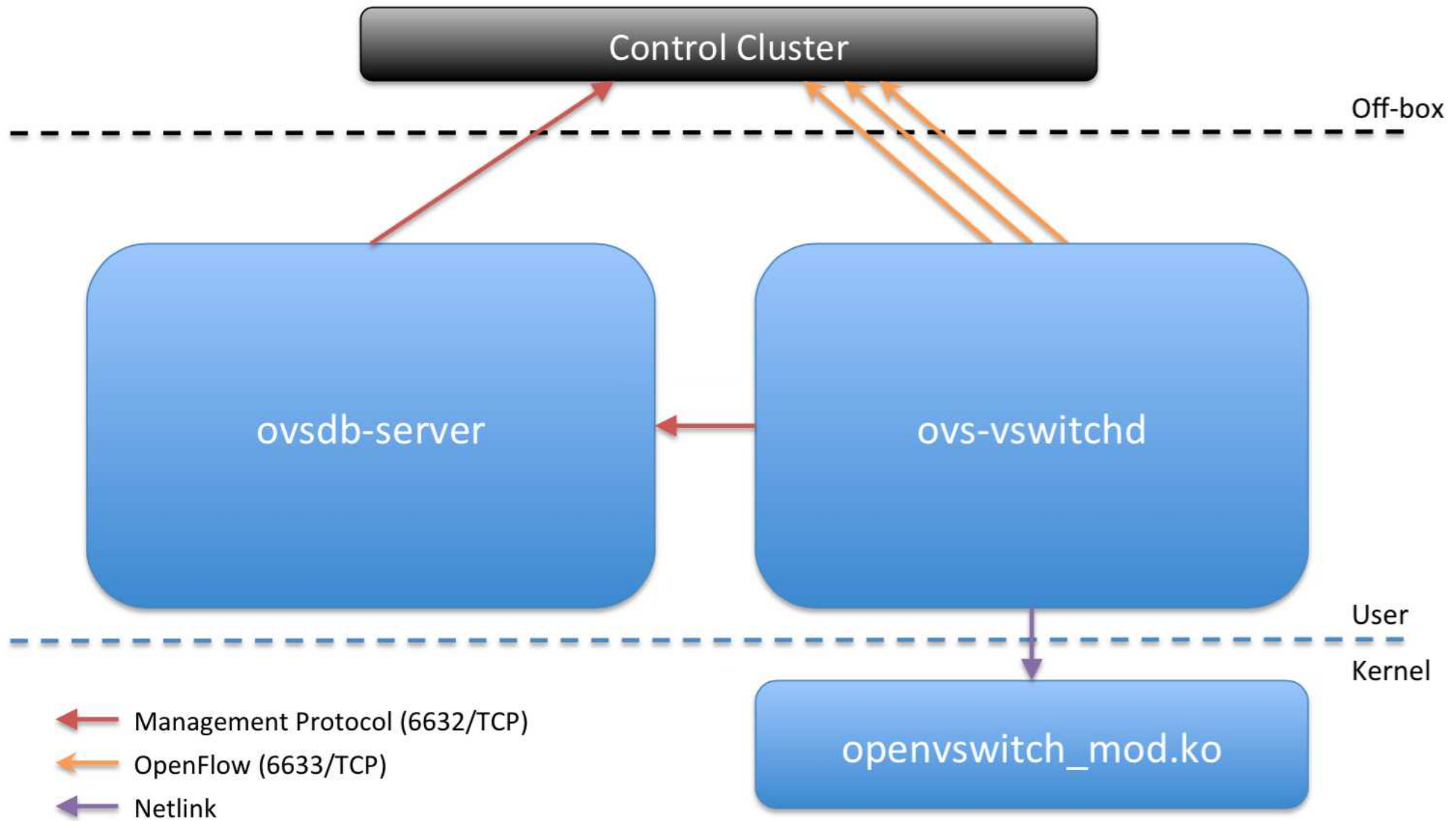
OpenFlow :

- ➔ first packet need to install rules → contact controller.
- ➔ Subsequent in fast path

VXLAN



Main components



Forrwarding components

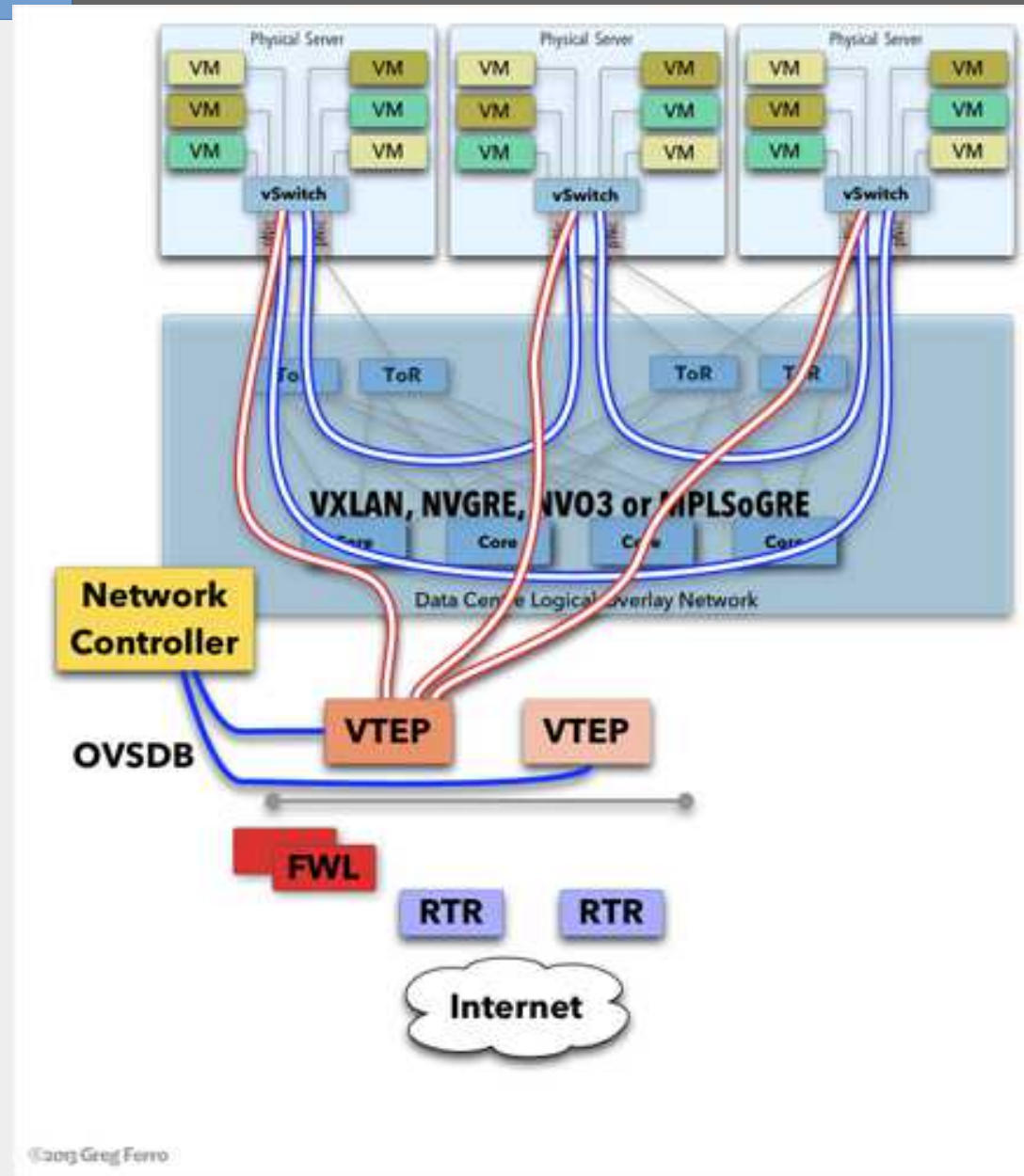
- Ovs-vsitchd (Slow Path)
 - ➔ Forwarding logic (learning, mirroring, VLANs, and bonding)
 - ➔ Remote configuration and visibility
- Openvswitch_mod.ko (Fast Path)
 - ➔ Linux kernel module
 - ➔ Packet lookup, modification, and forwarding
 - ➔ Tunnel encapsulation/decapsulation

Control plane

- Database that holds switch-level configuration: Bridge, interface, tunnel definitions
- OVSDB and OpenFlow controller addresses
- Configuration is stored on disk and survives a reboot

Typical deployment with e.g. Openstack

- Tunnels (VXLAN, GRE) between virtual switches
- Centralized management of switches
- VTEP : virtual tunnel endpoint
- Minimal support of physical switches that simply interconnect OVS switches
- Other options with clever



Performance

- A key factor in production environments

Accepted at 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)

©2014 IEEE

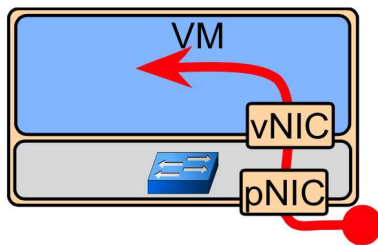
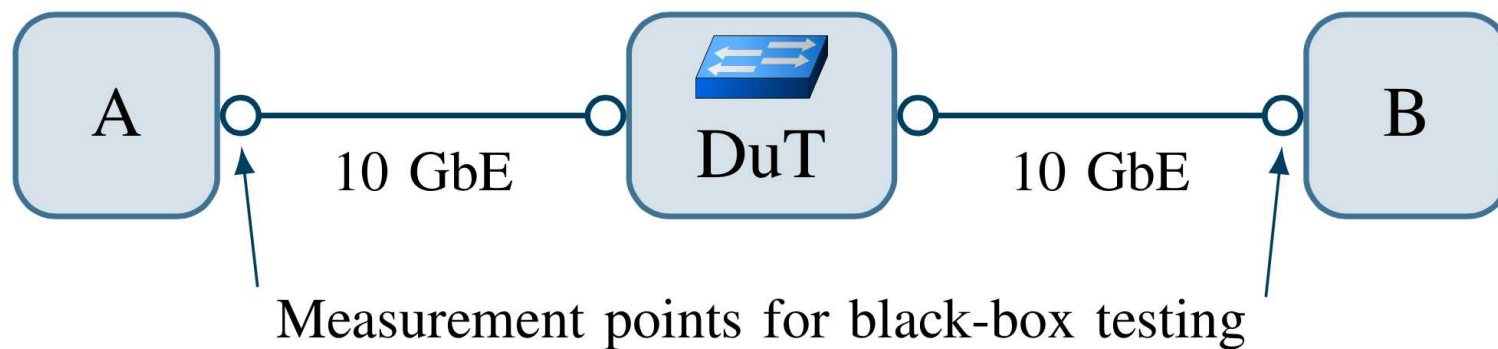
Performance Characteristics of Virtual Switching

Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle

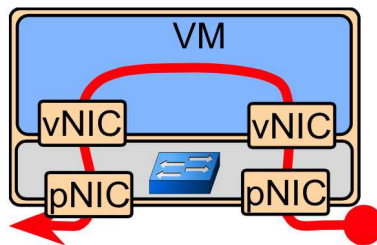
Technische Universität München, Department of Computer Science, Network Architectures and Services
{emmericp|raumer|wohlfart|carle}@net.in.tum.de

Performance

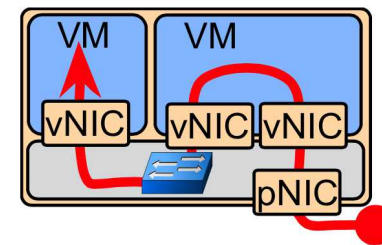
- Test set up: DUT = Device Under Test
- p/vNIC = physical/virtual NIC



(a)



(b)



(c)

Performance

- Switches: Linux bridge, OVS, IP forwarding (routing between IP by kernel)
- Results in pps (packet per s): this is what matters. Small packets → stress on I/O

TABLE I. SINGLE CORE DATA PLANE PERFORMANCE COMPARISON

Application	pNIC-pNIC [Mpps]	pNIC-vNIC [Mpps]	pNIC-vNIC-pNIC [Mpps]	pNIC-vNIC-vNIC [Mpps]
Open vSwitch	1.88	0.85	0.3	0.27
IP forwarding	1.58	0.78	0.19	0.16
Linux bridge	1.11	0.74	0.2	0.19

Performance

- Switch sends packets over different CPU: scale until reaching the maximum number of cores
- Visible when close to saturation → when the packet size becomes small

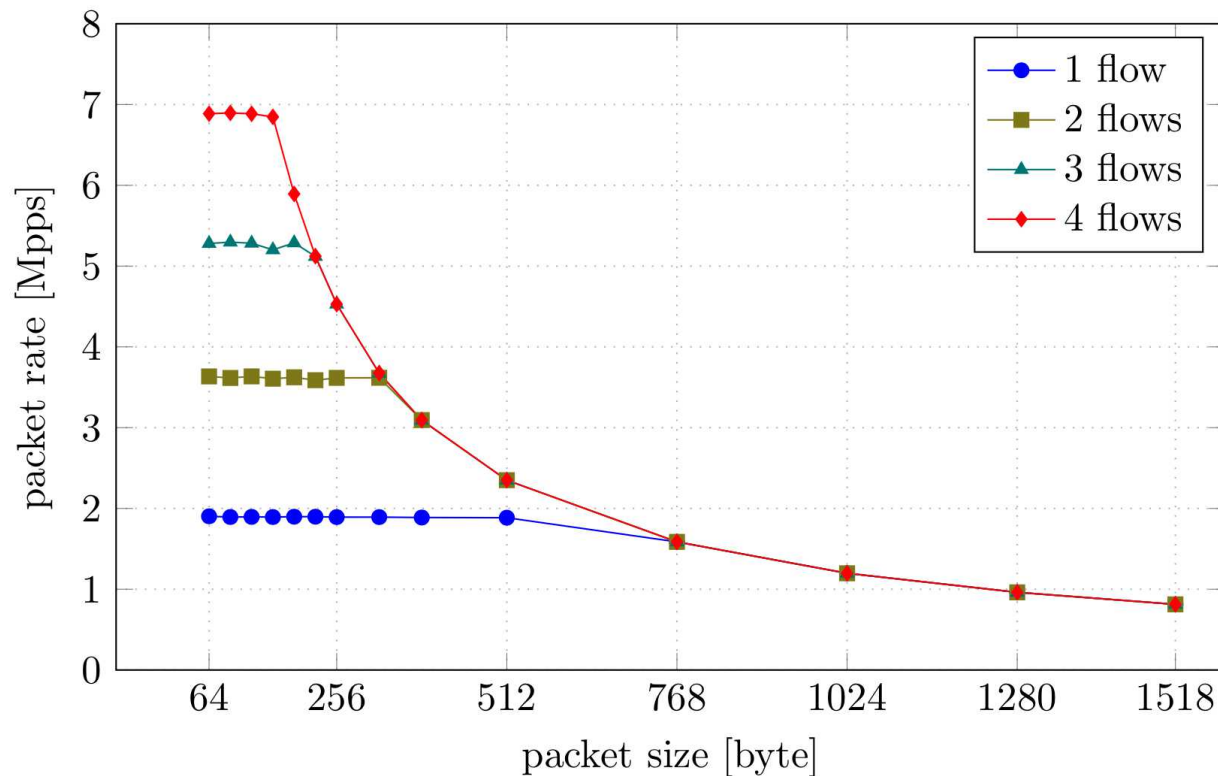


Fig. 5. Throughput with various packet sizes and flows

Performance

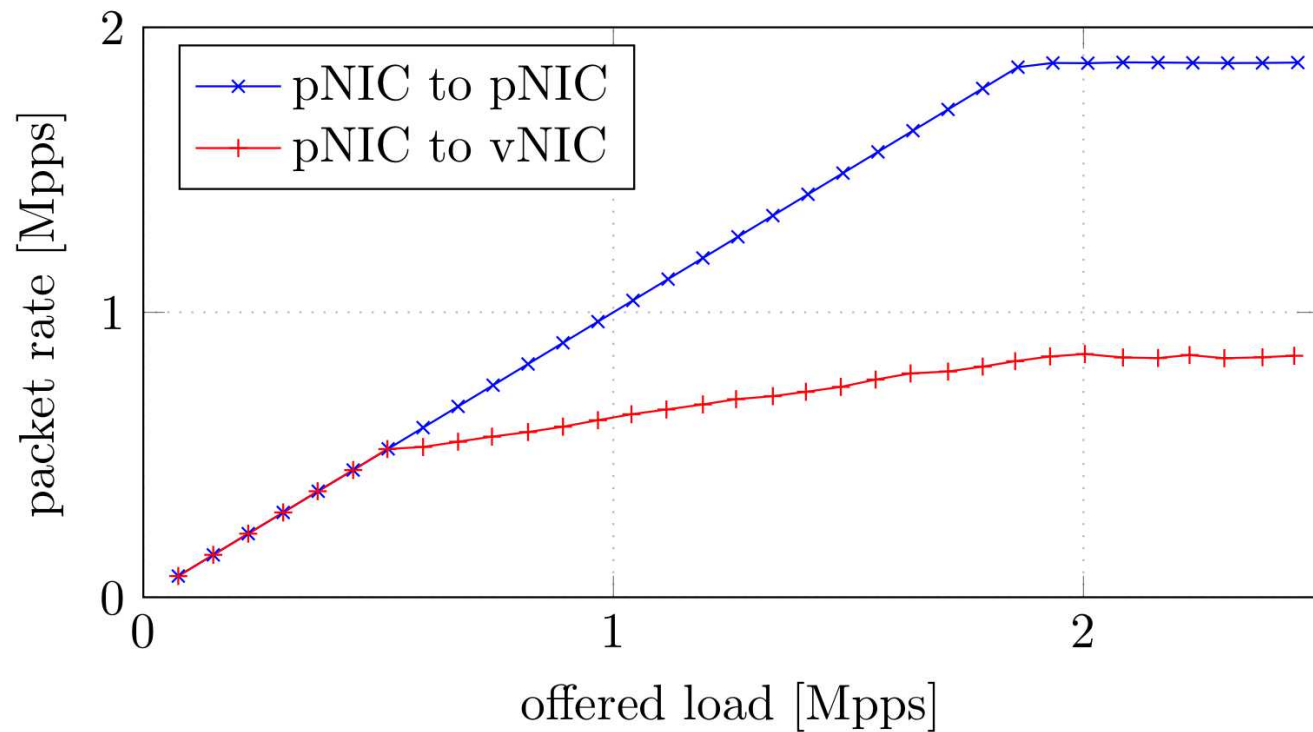


Fig. 9. Offered load vs. throughput with pNICs and vNICs

Performance

- Moving from pNIC to vNIC means moving to a different VM → context switching from ovs process to the VM process by the hypervisor

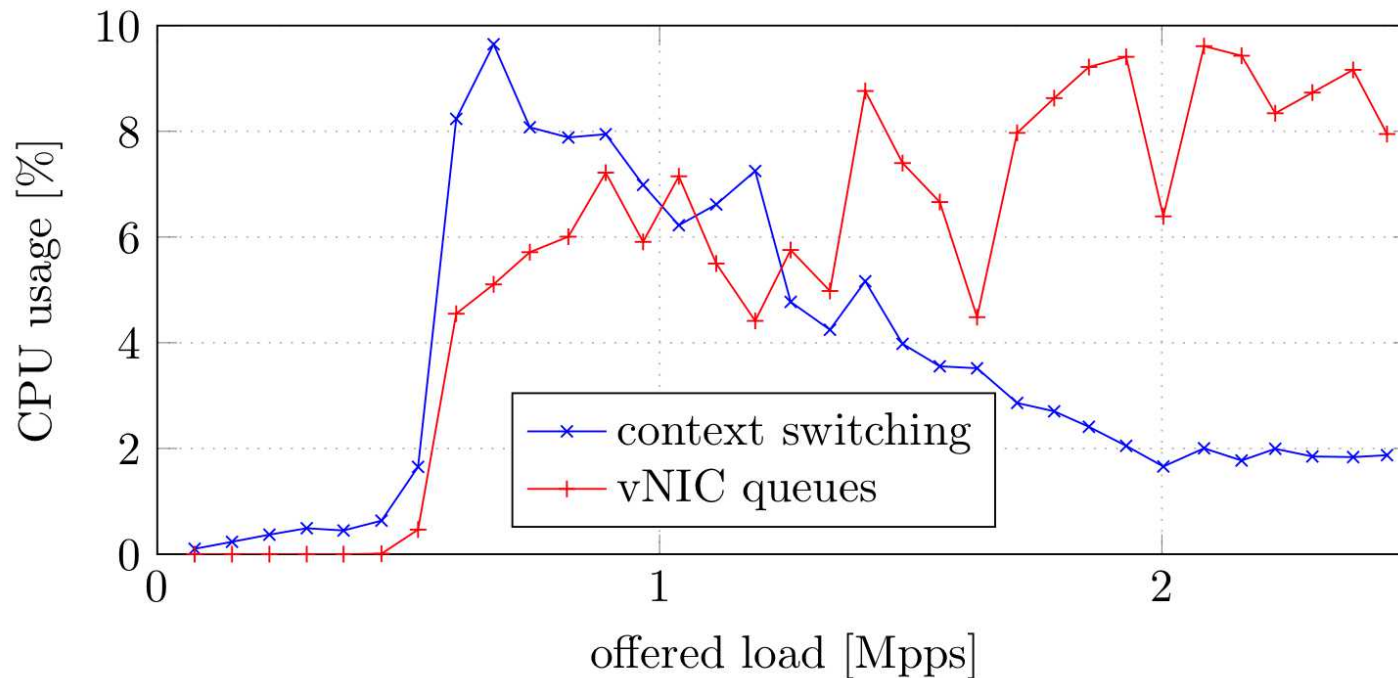


Fig. 10. CPU load of context switching and vNIC queuing