# Virtualization Technologies

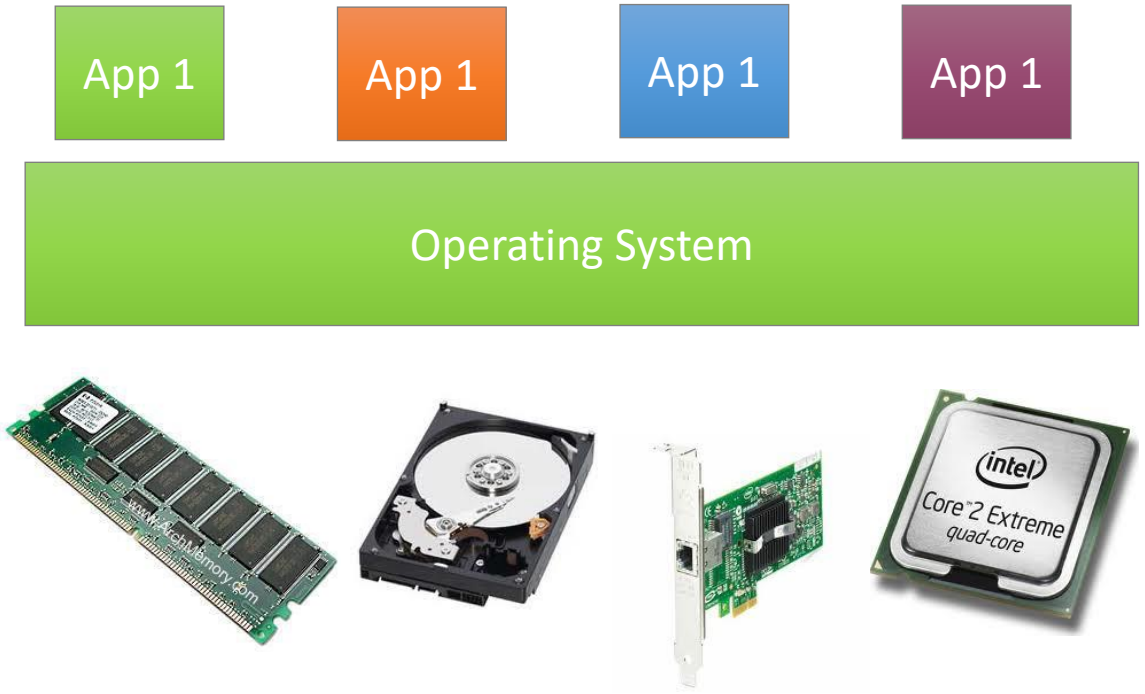
## Gestão de Infraestruturas de Computação

João Paulo Barraca

deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

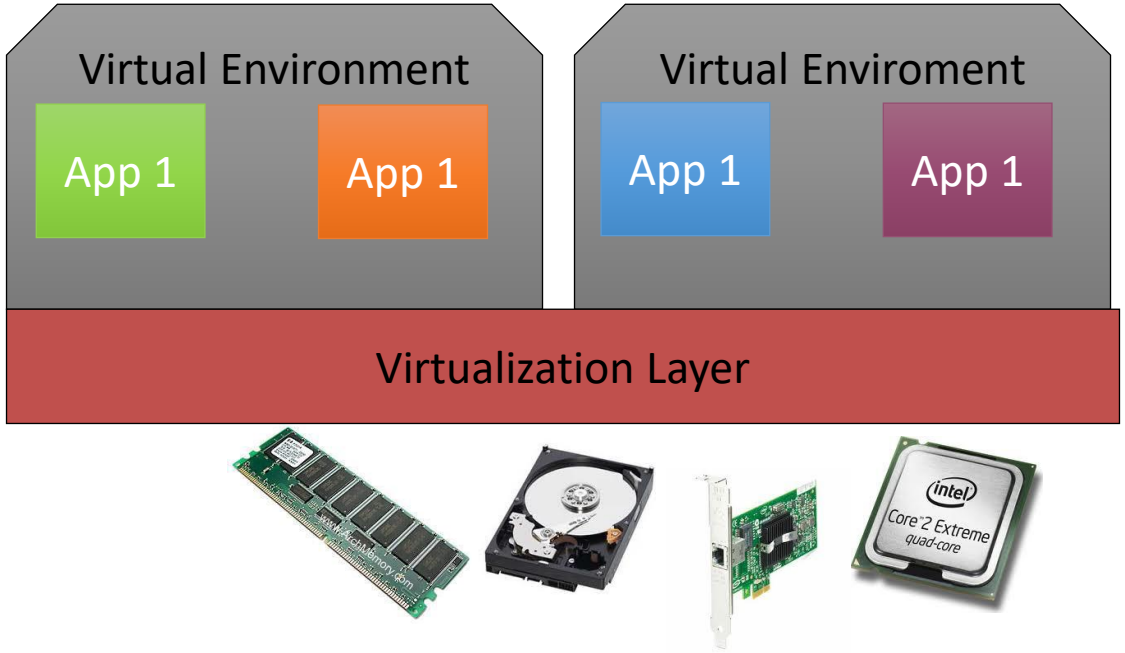
# Traditional Computing Model

- One single OS controls all applications and devices



# Virtualized System

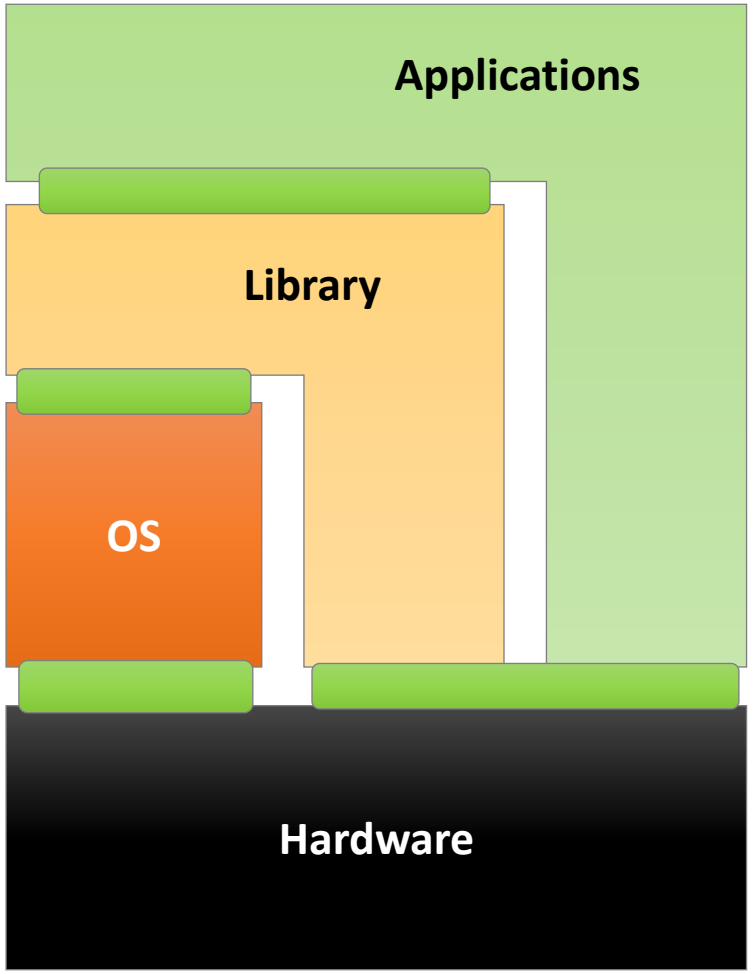
- **Multiple environments over the same hardware**
- **Guest access is multiplexed to hardware**
  - May be virtual or real



# Virtualization

## Virtualization can occur at different levels

- **Library**
  - Same OS/HW, individual libs for each environment
- **Operating System**
  - Same HW, individual OS view for each environment
- **Hardware calls**
  - Individual hardware for each environment



# Virtualization

- **Very abstract concept...**
  - Virtual Memory, Virtual File System, Virtual Networks (VPN's), Namespaces

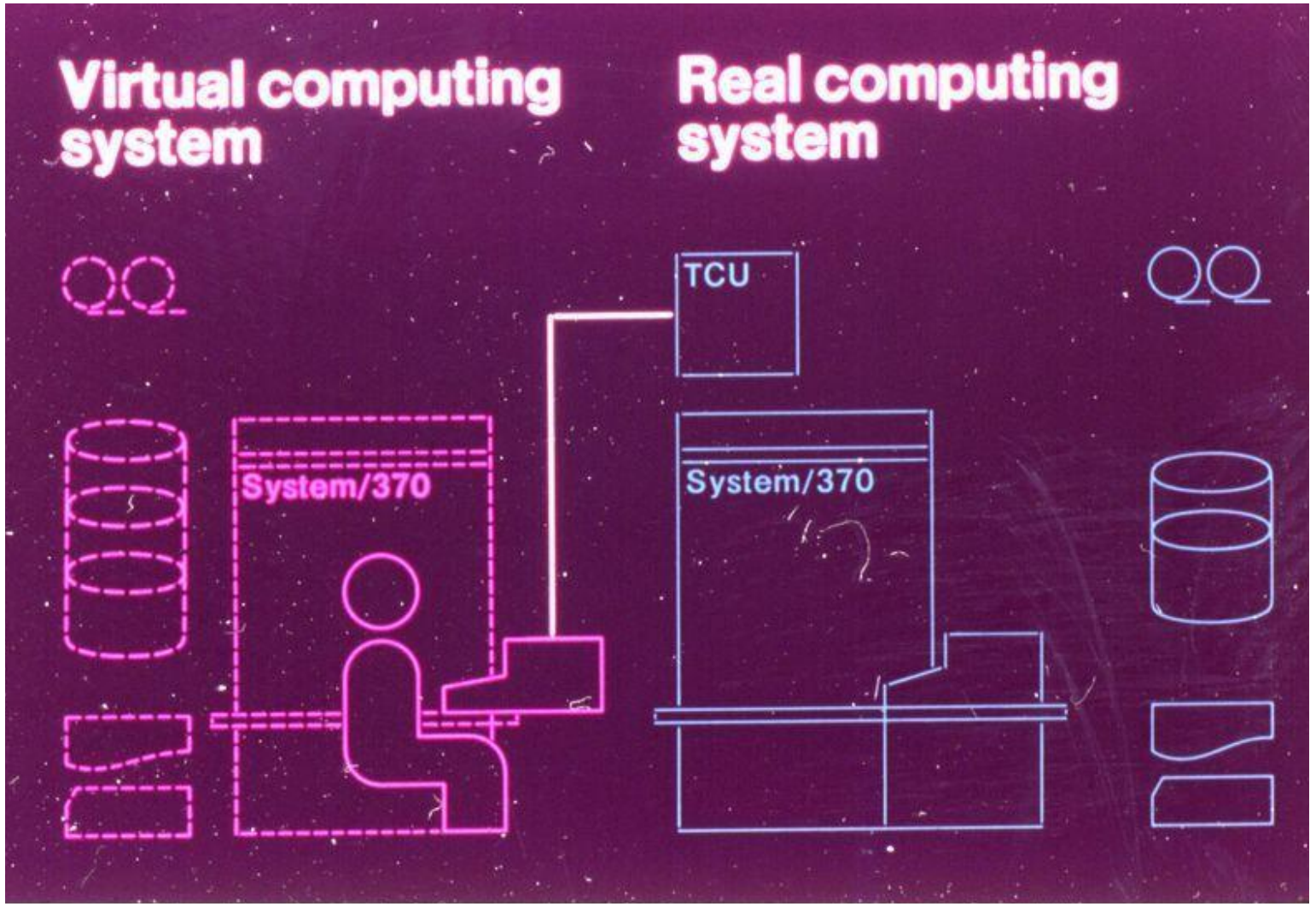
- **Virtualization allows a single computer play the role of several computers, through sharing hardware resources to multiple environments.**

- **Virtualization is a tecnique allowing to run several different systems in the same hardware, in a totaly separated manner.**

# Virtualization history

- **Is it a new technology? No!**
- **IBM S/360 mainframes allows application timesharing through resource partitioning**
  - Multiple single-user OS instances
  - Guest directly on hardware
  - Circa 1960s, concepts still present in z/VM (2000-2009)
- **IBM also provided Logical resource Partitioning (LPAR)**
  - Creates subsets of the host hardware
  - Multiple Operating systems
  - DLPAR allows dynamic modification of partitions
  - Circa 1970

# IBM S/360, ca 1972



# Virtualization history (Software)

- **Interest was lost in 80s-90s in favor of multi-user OS**
  - Hardware also lost most virtualization features
- **Bochs software provides emulated environments in 1990s**
  - Mostly for debugging
  - Complete environment with emulated hardware
- **VMware is founded in 1998 from ideas existing at Stanford**
  - Part of EMC
  - Provided Hypervisors creating fully isolated containers
- **XEN sprouts from Cambridge in 2003**
  - Proposes para-virtualization (host assisted)

# Virtualization history (Software)

## Every company jumps to virtualization

- QEMU in 2005 – allows multiple architectures
  - Was used for Android emulation
  - Is extensively used for embedded development/security
- OpenVZ in 2005
- Parallels Workstation in 2006
- Linux KVM in 2008
- Oracle Virtualbox in 2007
- Microsoft Hyper-V in 2008
- LinuX Containers in 2008
- Microsoft Virtual PC in 2009
- Docker in 2013

# Virtualization history (Hardware)

- Intel VT-x/AMD-V: Hardware accelerated Ring-0 management
- Intel VT-d/AMD-Vi: Virtualized Direct IO
- Intel VMCS/AMD-V: Support for nested virtualization
- Intel APICv/AMD AVIC: Virtual Interrupt controllers
- Intel GVT-d/g/s: Partitioning of GPU to VMs
- Intel VT-c: Network devices

# Virtualization History

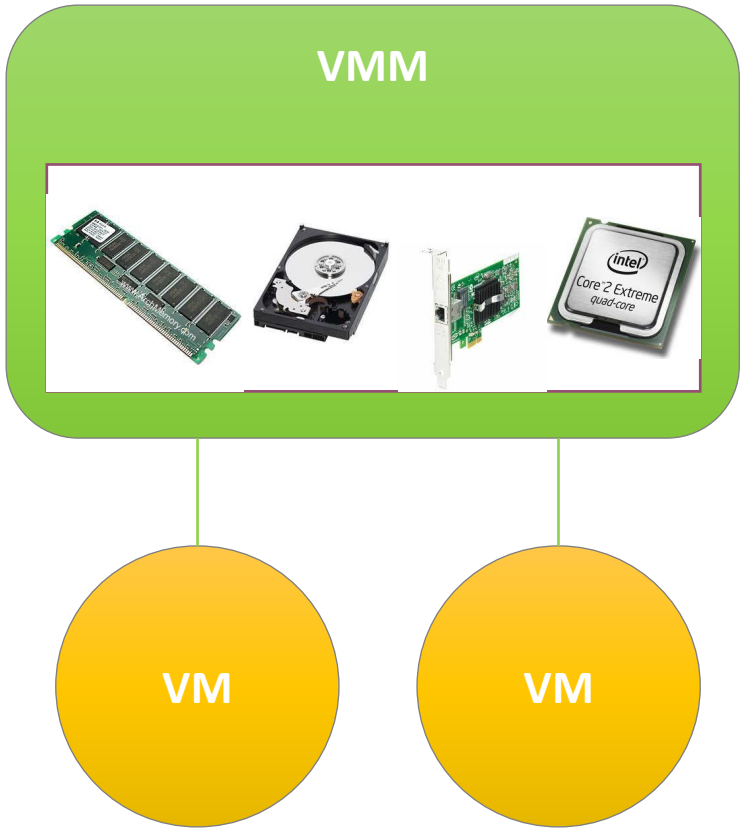
## Virtualization enables the flexible cloud model we have

- **Datacenters are now completelly virtual**
  - Every aspect is **Software Defined**
  - Infrastructures are becoming programable (later)

- **Services in Virtual Instances in bare metal**

# Virtual Machine Monitor (Hypervisor)

- **Runs on bare metal**
  - Direct access to hardware
- **Partition resources through different Guest hosts**
  - May provide not-existing hardware
  - May overcommit resource allocation
- **Each guest OS uses emulated resources as if they were real**
  - Some hardware may be real (passthrough/partitioned)



# Virtualization Methods/Approaches

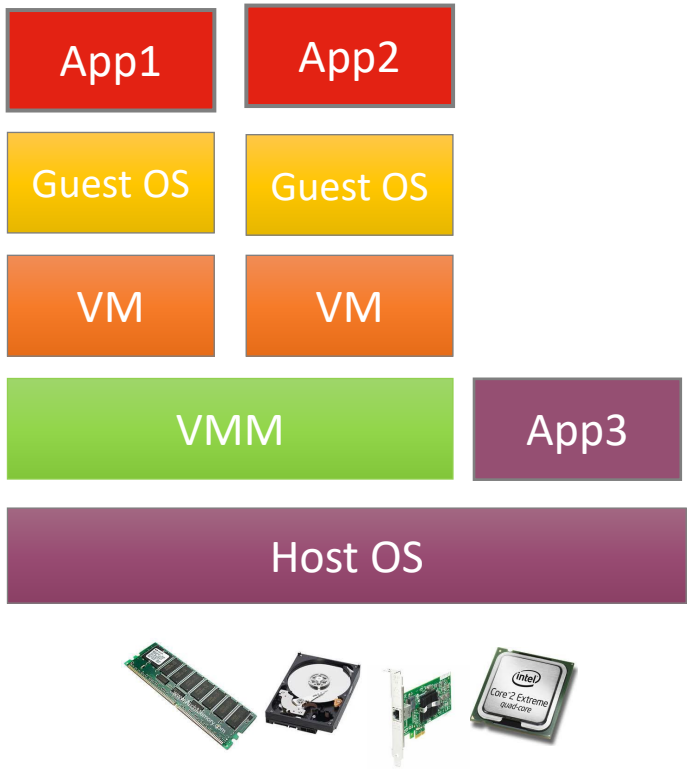
- Hosted Virtualization
- Popek and Goldberg VMs
- Para-Virtualization
- Hardware Assisted Virtualization

# Hosted Virtualization (interpretation)

- **VMM is a standard application running on a common OS**
  - Keeps track of a set of virtual hardware
    - Emulates generic devices or specific devices
  - Each guest sees individual hardware
  - Hardware support varies with host OS
- **Application code is interpreted step by step**
  - Code is usually native instructions
- **Most basic virtualization method (70s)**
  - x86 and other architectures actually had no virtualization support until 2005

# Hosted Virtualization (interpretation)

- **Advantages**
  - Compatibility as instructions are emulated
  - **Can emulate diferent architectures!**
  - Isolation as environments are virtual
  - Unmodified OS (instruction emulation)
  - No real instruction is executed on the host!




- **Disadvantages**
  - More resources required
  - Lower performance due to step by step execution
    - Full OS running, all instructions are emulated
  - Only usable for simple processors

# Popek and Goldberg VM

- Assuming that guest and host architectures are the same, virtualization becomes easier (faster, simpler)

+ Most instructions can be executed directly (not emulated)

+ Performance is almost native

- Virtual hardware must be emulated

- Some instructions must be emulated to ensure isolation

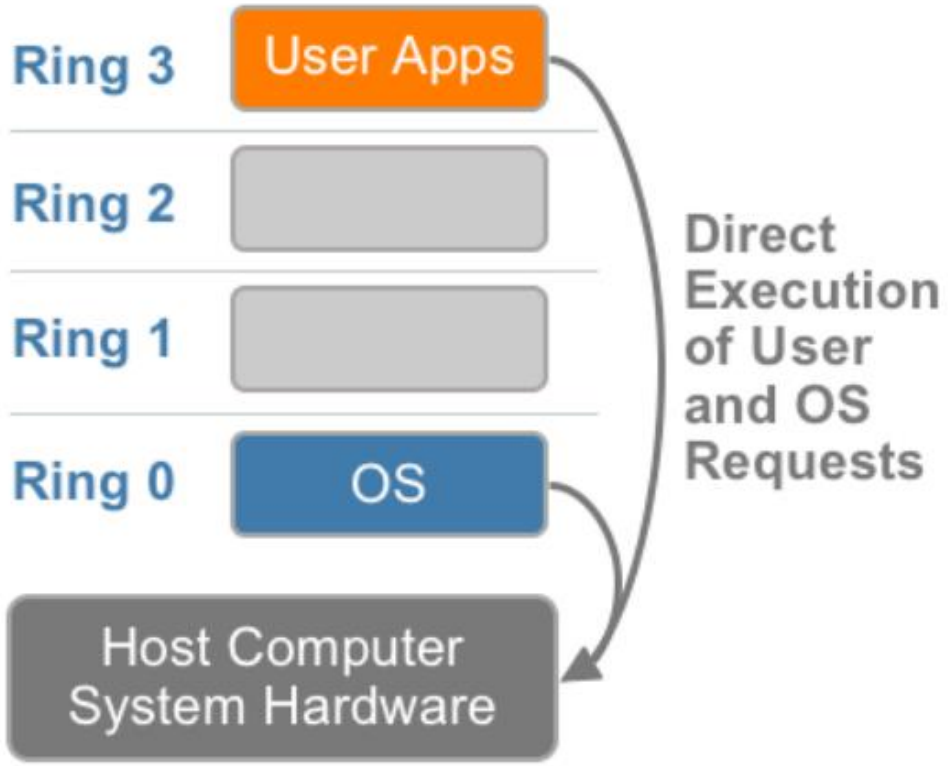
# Popek and Goldberg VM

- **Also known as: Trap and emulate**

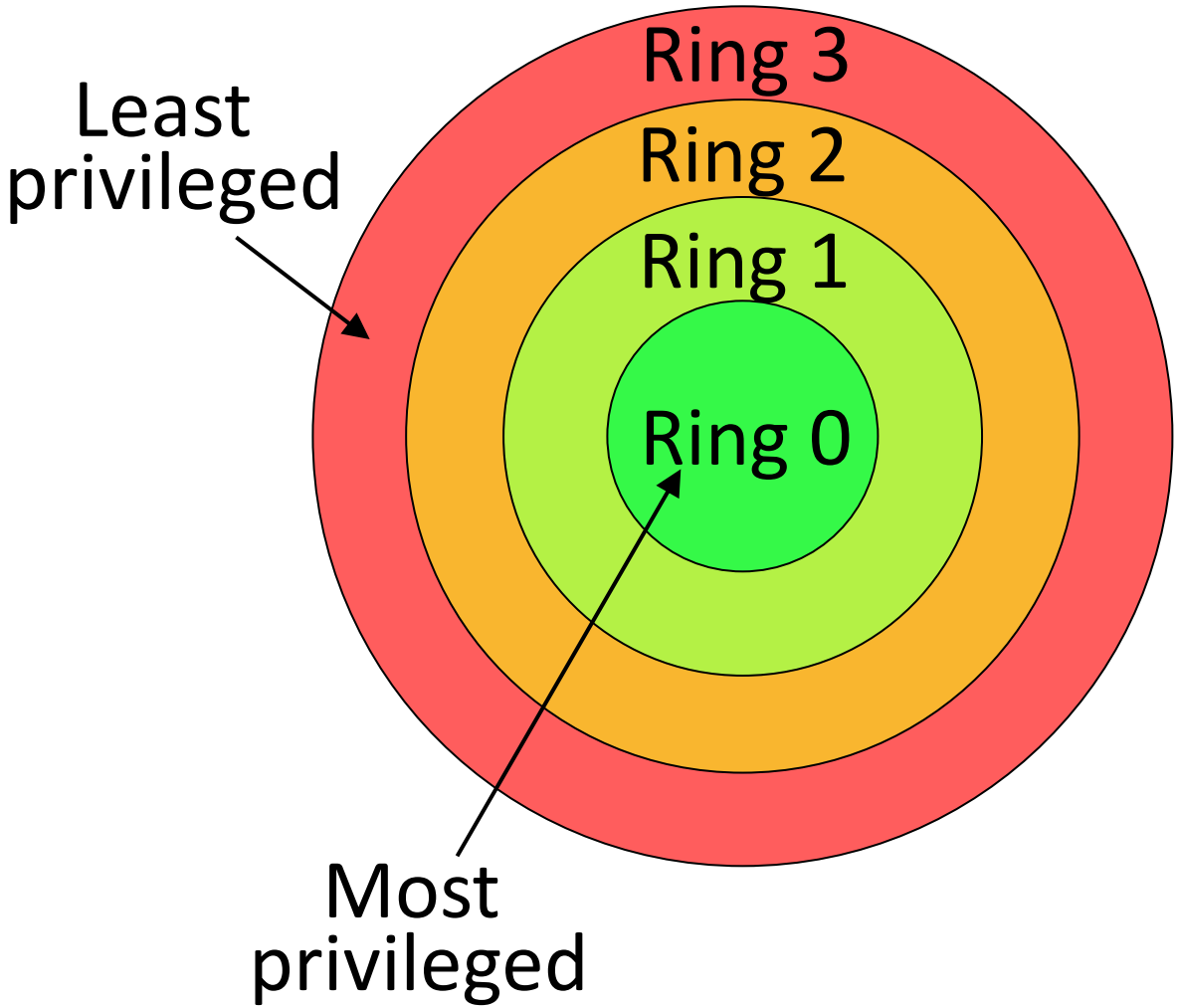
- **Instruction types:**
  - **Sensitive Instruction**: has low level access and to machine (IO, page tables) and must be handled by the VMM
  - **Safe Instruction**: no global impact and can be executed without issues
  - **Privileged Instruction**: will generate a trap when executed in user mode

- **Guest code is executed as a standard application**
  - Sensitive instructions must be caught

- **VMM catches Trap and emulates behavior for guest VM**



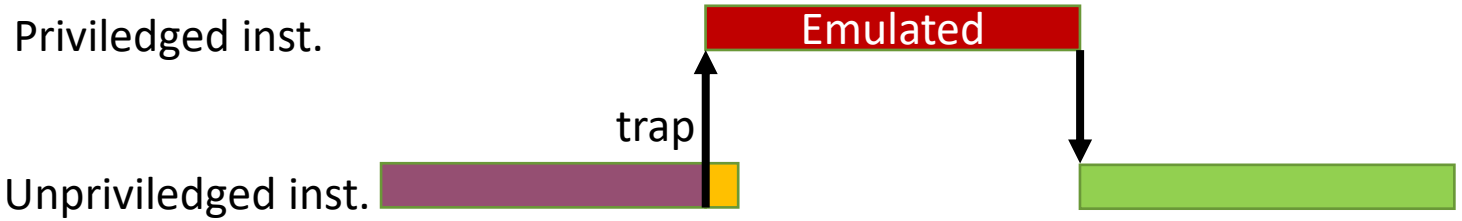

VMWare, Understanding Full Virtualization, Paravirtualization and Hardware Assist, 2007



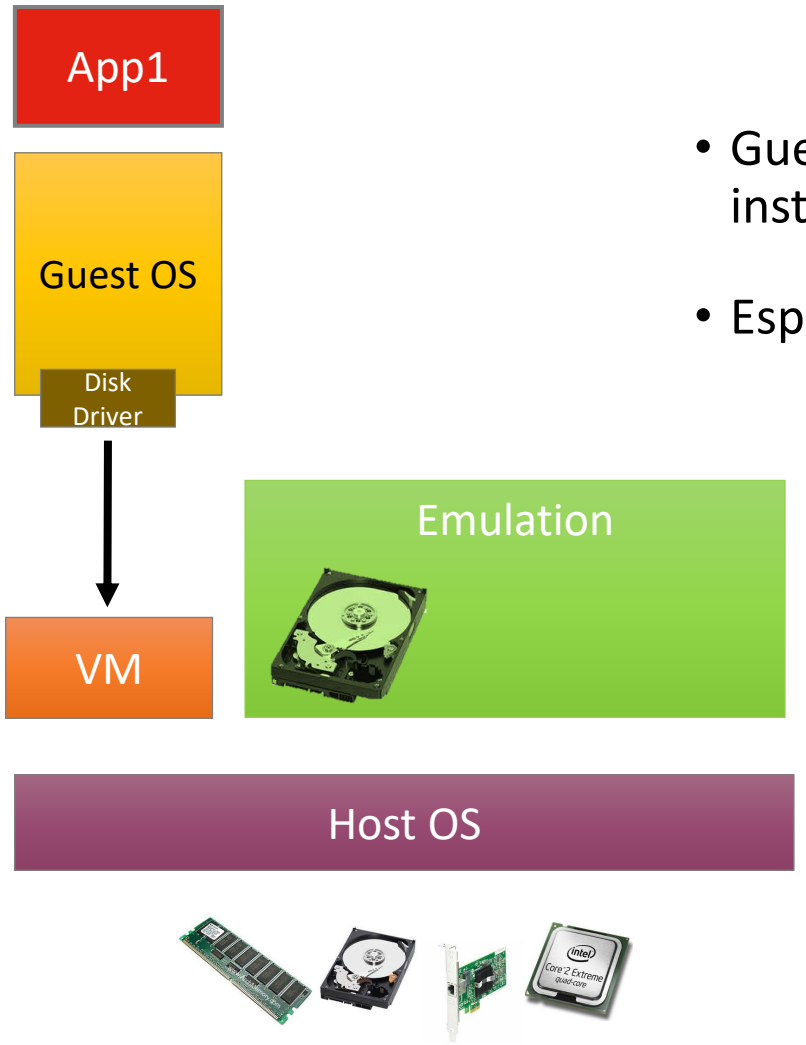

**x86 Rings**

- **x86 Page Table Entry has 1 bit to state if page is user accessible**
  - 0 = Ring 0
  - 1 = Ring 3
- **Ring 0 can execute everything!**
- **Ring 3 can only execute safe instructions**
  - Privileged instructions generate a **trap**
- **Ring 1 and 2 are legacy for drivers (not used)**

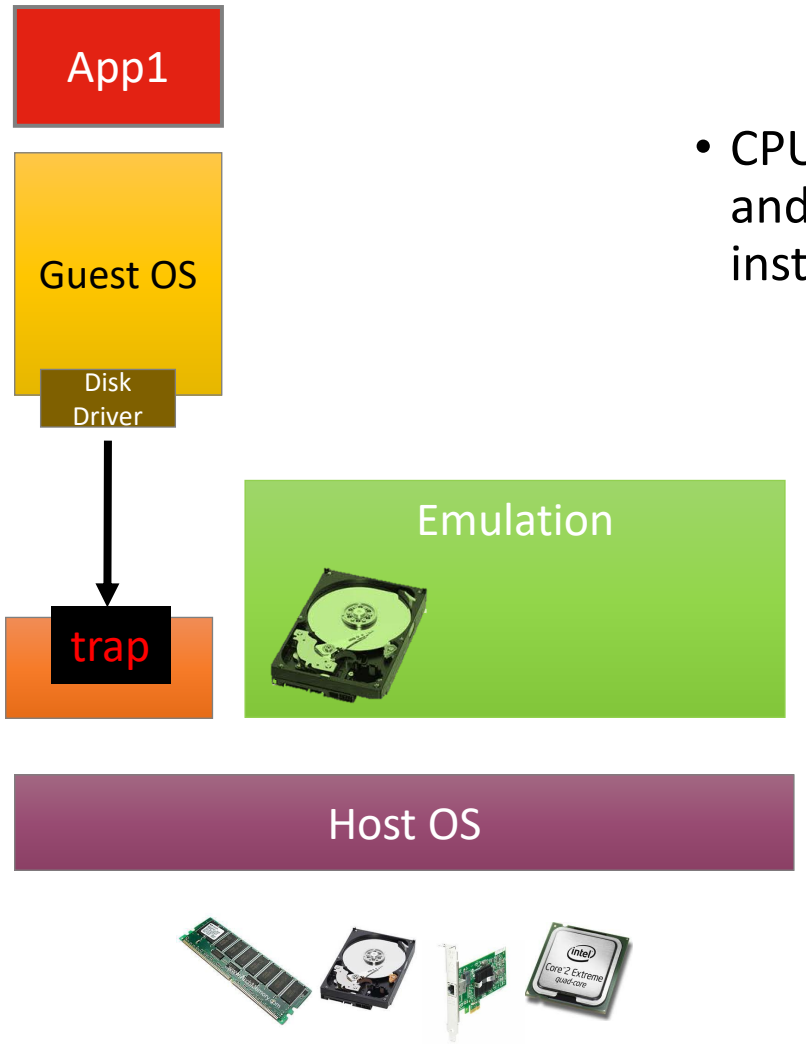
# Popek and Goldberg VM



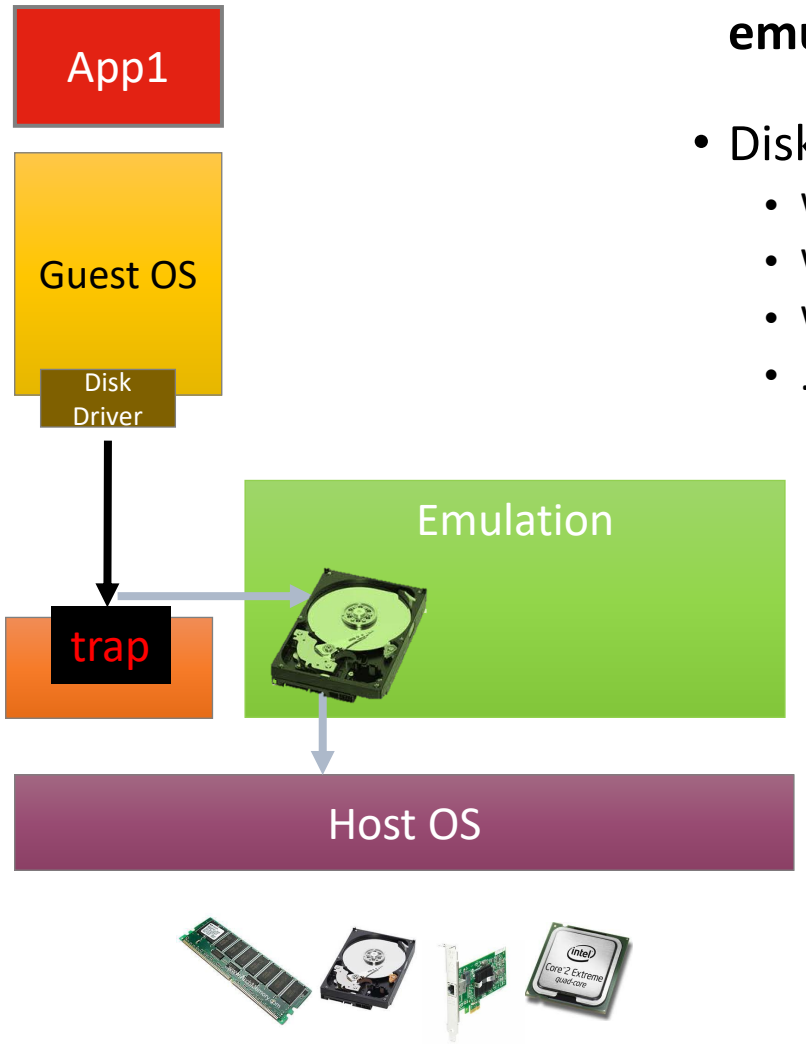

- Applied to multiple subsystems
  - CPU and Memory: registers and memory state
- Memory Management Unit
  - Page tables, segments
- Platform
  - Interrupt controller, timers, buses
- Peripheral devices
  - Disks, network cards, GPUs, serial ports




- Guest may issue priviledged instructions
- Especially if it is a kernel




- CPU will generate a trap and suspend that instruction




- VMM catches trap and **emulates** instruction
- Disk write:
  - Write to physical device
  - Write to file
  - Write to RAM
  - …

# Popek and Goldberg VM

- **According to Popek and Goldberg: Sensitive instructions must be priviledged**
- **Some architectures do not generate traps for all sensitive instructions!**
  - E.g.: x86

| Group | Instructions |
|---|---|
| Access to interrupt flag | `pushf, popf, iret` |
| Visibility into segment descriptors | `lar, verr, verw, lsl` |
| Segment manipulation instructions | `pop <seg>, push <seg>, mov <seg>` |
| Read-only access to privileged state | `sgdt, sldt, sidt, smsw` |
| Interrupt and gate instructions | `fcall, longjump, retfar, str, int <n>` |

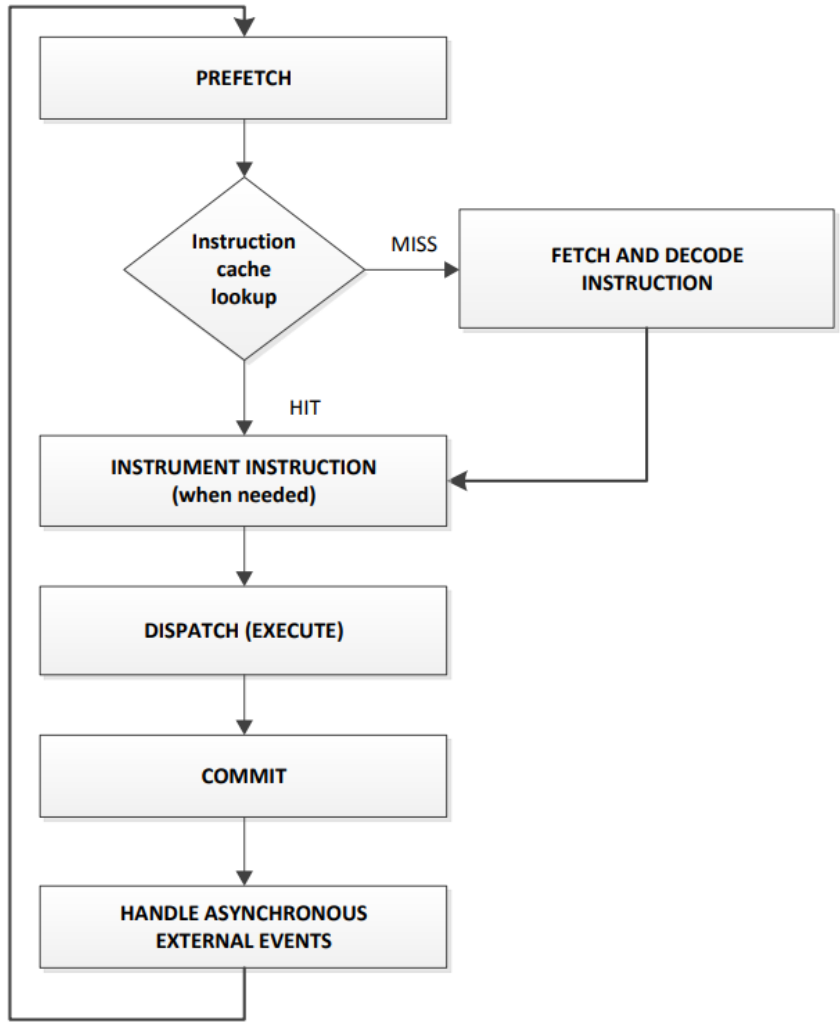
John Scott Robin and Cynthia E. Irvine (2000). "Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor". *Proc. 9th USENIX Security Symposium*.

# Alternatives

- **Process guest code and detect all sensitive instructions before execution**
  - Interpretation (BOCHS, JSLinux)
  - Binary Translation (QEMU, VMWare)
- **Use modified guest operating systems**
  - Paravirtualization (XEN, L4, Hyper-V)
- **Really fix the issue and make all sensitive instructions privileged**
  - Hardware Support Virtualization (most solutions)
    - Intel VT-x, AMD SVM

# Interpretation

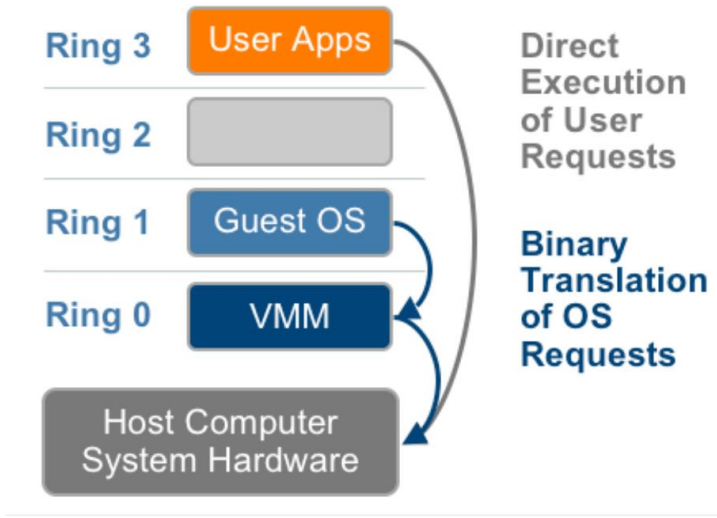
- Simple pipeline fetching one instruction at a time
- Instruction is decoded and cached
- Instruction is then emulated in the host (execution)
- Every guest CPU instruction results in many other host instructions




Stanislav Shwartsman and Darek Mihoka, How Bochs works under the hood, 2nd edition, 2012

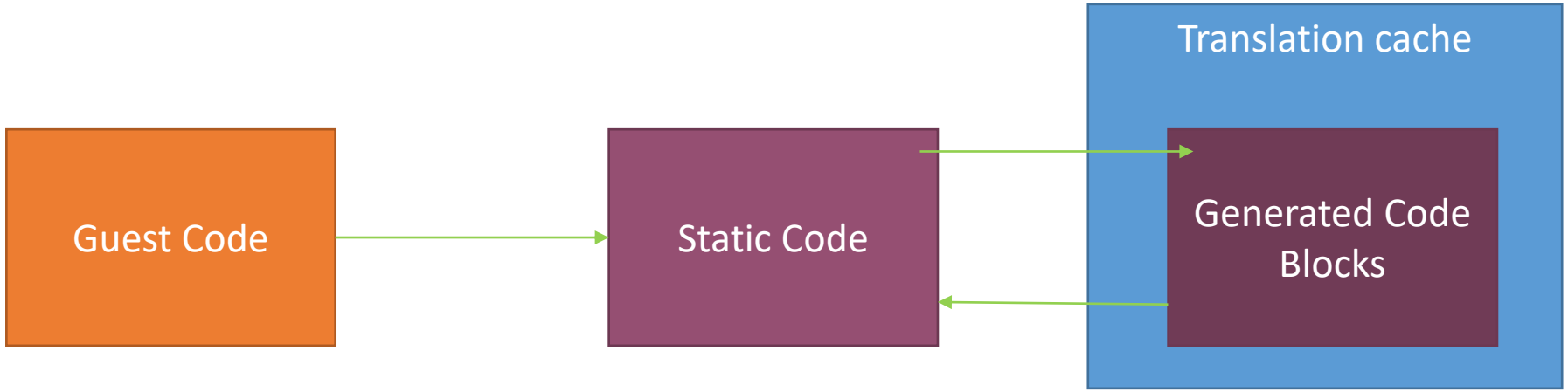
# Binary translation

- **A faster approach is to translate sensitive Ring 0 code**
  - Between architectures (more complex)
  - Allows to avoid issues with sensitive instructions **avoiding or forcing trap-emulate for sensitive instructions**



# Binary translation

- **Translated code can execute mostly as real code in the host architecture**
  - Some prologue-epilogue may be required
  - Translation is made in **basic blocks** to deal with branches
  - Caches can reduce need for repeated translation



```
mov 32(%rbp), %rbx
sub %rbx, %rcx
push %rcx
popf //Sensitive instruction,
     //but not privileged!
     //Won't trap when called
     //in Ring 3.
```

**Original guest OS code**

**Translated guest OS code**

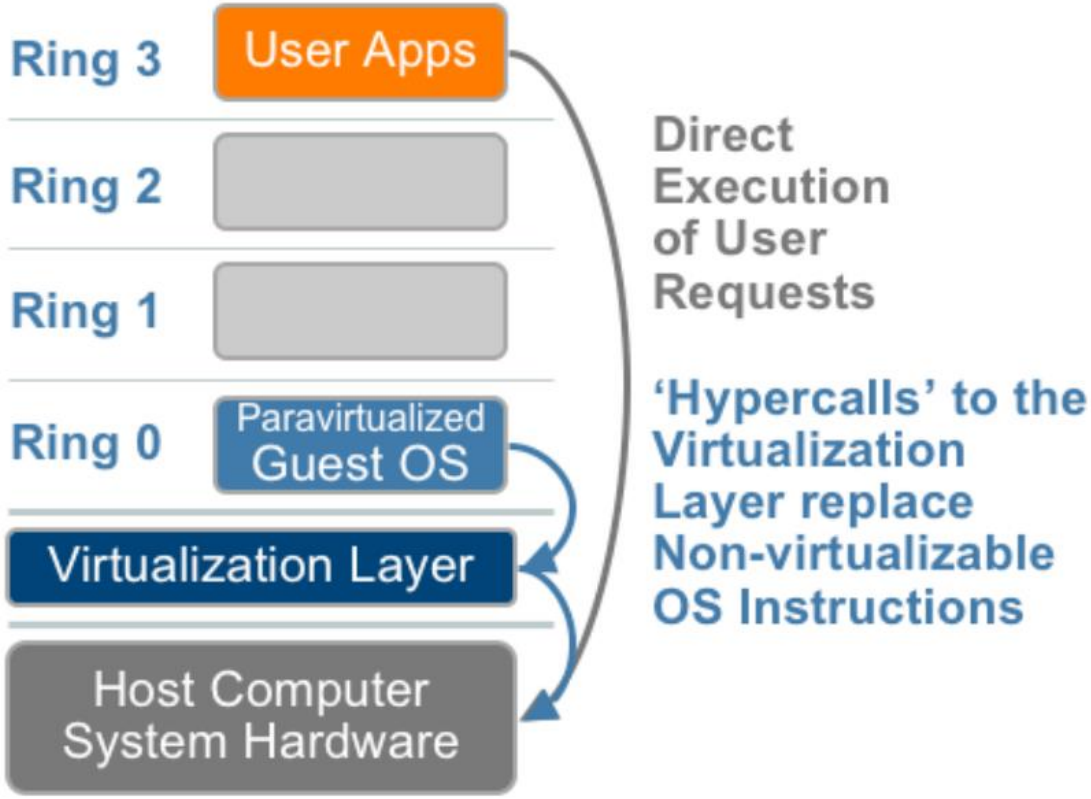
```
mov 32(%rbp), %rbx
sub %rbx, %rcx
push %rcx
int3
    //A special one-byte
    //instruction that traps.
    //Originally intended for
    //use by debuggers to set
    //breakpoints.
```

# Paravirtualization

- **What if the guest has no sensitive instructions?**
  - All instructions can be executed without delay!

- **Approach**
  - Consider that the guest can be modified
    - Change code base and recompile
  - Produce a guest OS that has no privileged instructions
  - Instead, guest OS will use **Hypercalls**
    - **Hypercall**: communicate directly with the hypervisor to execute otherwise sensitive operations

# Paravirtualization



# Paravirtualization

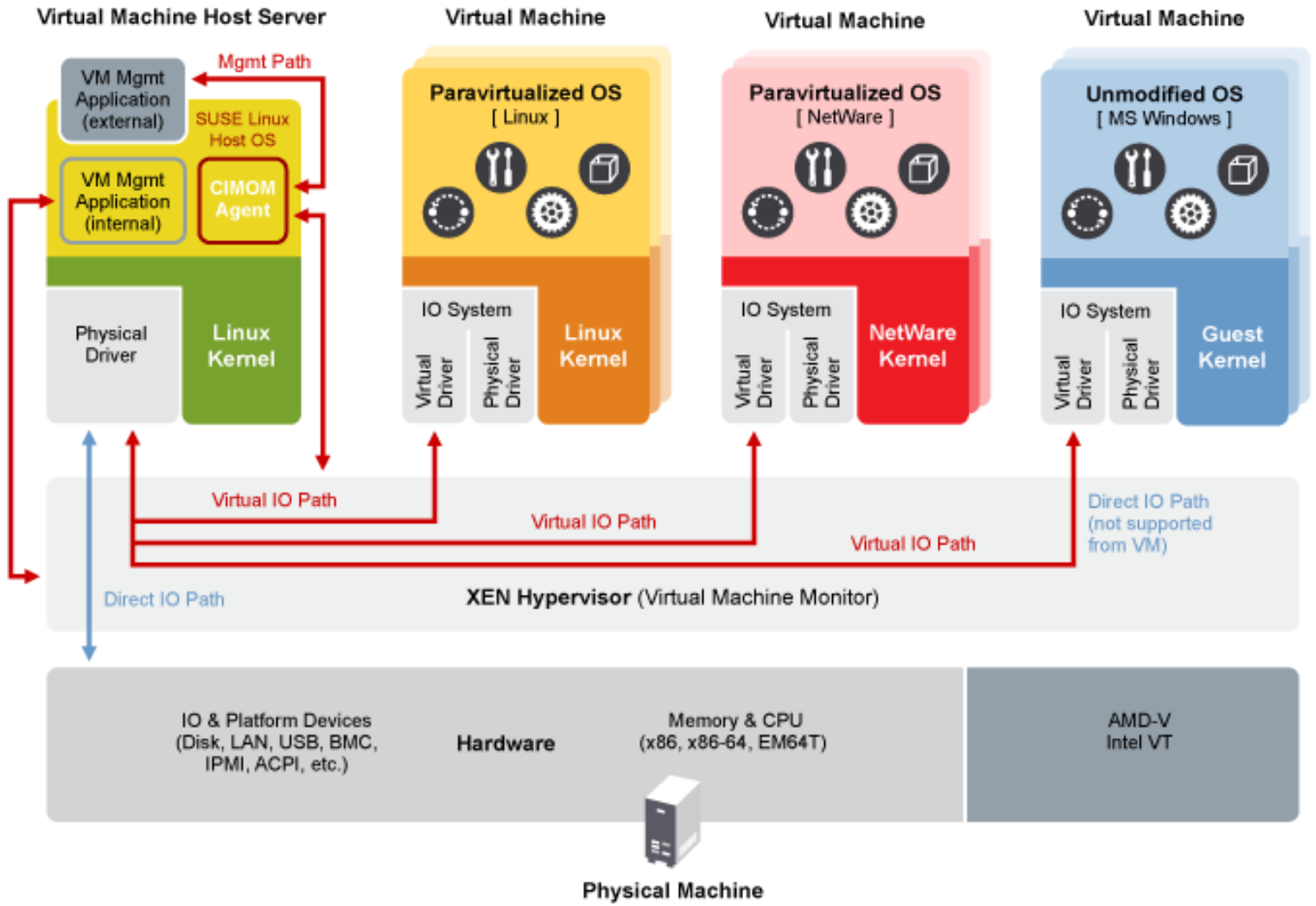
- **Support for paravirtualization is implemented in kernel + drivers**
- **Kernel for basic interaction with guest hardware**
- **Drivers mostly for paravirtualized IO**
  - Disk
  - Network
  - GPUs
- **Drivers can also provide additional services to host**
  - Performance metrics
  - Information (IP address)
  - Folder sharing

```
[    0.000000] NX (Execute Disable) protection: active
[    0.000000] SMBIOS 2.5 present.
[    0.000000] DMI: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/01/2006
[    0.000000] Hypervisor detected: KVM
[    0.000000] kvm-clock: Using msrs 4b564d01 and 4b564d00
[    0.000000] kvm-clock: cpu 0, msr 20e01001, primary cpu clock
[    0.000000] kvm-clock: using sched offset of 4709579586 cycles
[    0.000002] clocksource: kvm-clock: mask: 0xffffffffffffffff max_cycles: 0x1cd42e4dffb,
```

```
[    0.061823] Booting paravirtualized kernel on KVM
[    0.061826] clocksource: refined-jiffies: mask: 0xffffffff max_cycles: 0xffffffff,
[    0.061831] setup_percpu: NR_CPUS:8192 nr_cpumask_bits:1 nr_cpu_ids:1 nr_node_ids:1
[    0.062124] percpu: Embedded 55 pages/cpu s188416 r8192 d28672 u2097152
[    0.062137] pcpu-alloc: s188416 r8192 d28672 u2097152 alloc=1*2097152
[    0.062138] pcpu-alloc: [0] 0
```

```
user@vm:~$ lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:02.0 VGA compatible controller: VMware SVGA II Adapter
00:03.0 Ethernet controller: Red Hat, Inc. Virtio network device
00:04.0 System peripheral: InnoTek Systemberatung GmbH VirtualBox Guest Service
00:05.0 Multimedia audio controller: Intel Corporation 82801AA AC'97 Audio Controller (rev 01)
00:07.0 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:0c.0 USB controller: Intel Corporation 7 Series/C210 Series Chipset Family USB xHCI Host Controller
00:0f.0 SCSI storage controller: Red Hat, Inc. Virtio SCSI (rev 01)
```

# Paravirtualization with XEN




https://doc.opensuse.org/documentation/leap/archive/42.2/virtualization/html/book.virt/cha.xen.basics.html

# Paravirtualization

+ Completely avoids the need for Trap-emulate issues
+ Instructions run at full speed
+ Sensitive instructions are replaced by hypercalls
+ VMM doesn’t need to emulate virtual hardware
  – VMM provides API to Guests!
  – Guests know that they are being virtualized

- Guest must be developed for a specific VMM
  – Solution is not compatible with closed source software

# Hardware Assisted Virtualization

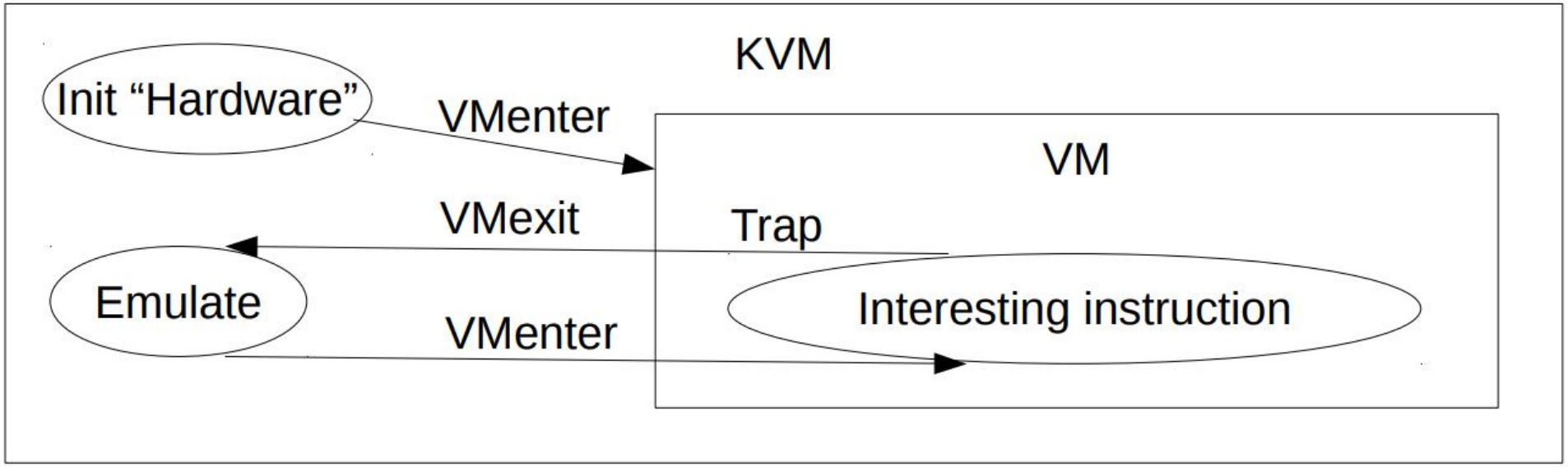
- **What if the hardware actually helped to virtualize guests?**

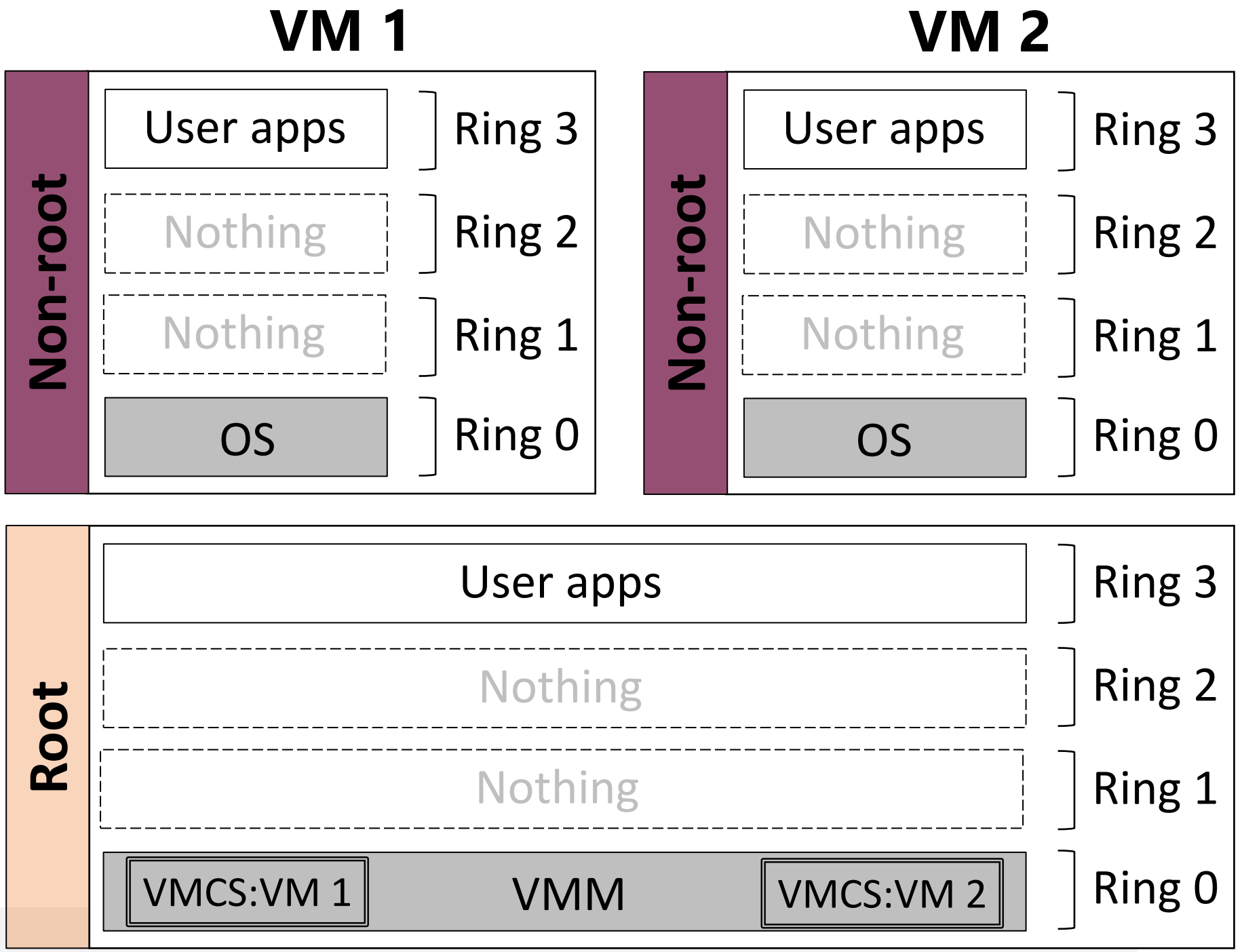
- **How:**
  - Creating more rings below ring 0 (where the kernel executes)
    - There are not actual rings, but they represent contexts with higher access than Ring 0
  - Allowing for multiple contexts (Virtual Devices)
  - Allowing for fast context switch between guests

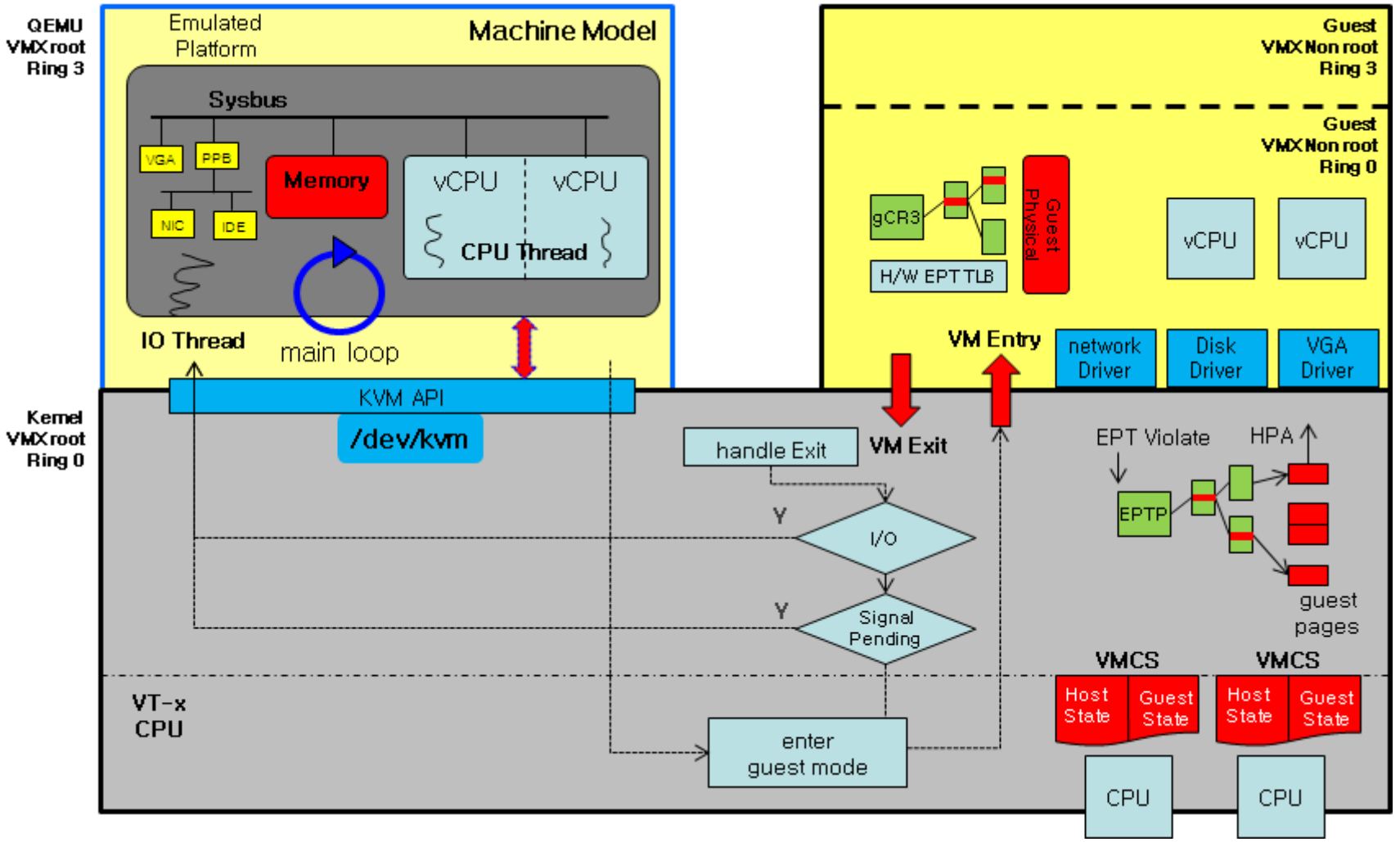
# Hardware Assisted Virtualization

- **Modern CPUs consider two modes of operation**
  - VMX Root: 4 rings, and standard operation
    - Host kernel runs here
    - VMM (Hypervisor) also runs here
  - VMX Non Root: 4 rings, but a diferent operation model
    - Guests run here
    - Some instructions are not allowed
    - Includes a new concept: Virtual Machine Control Structure

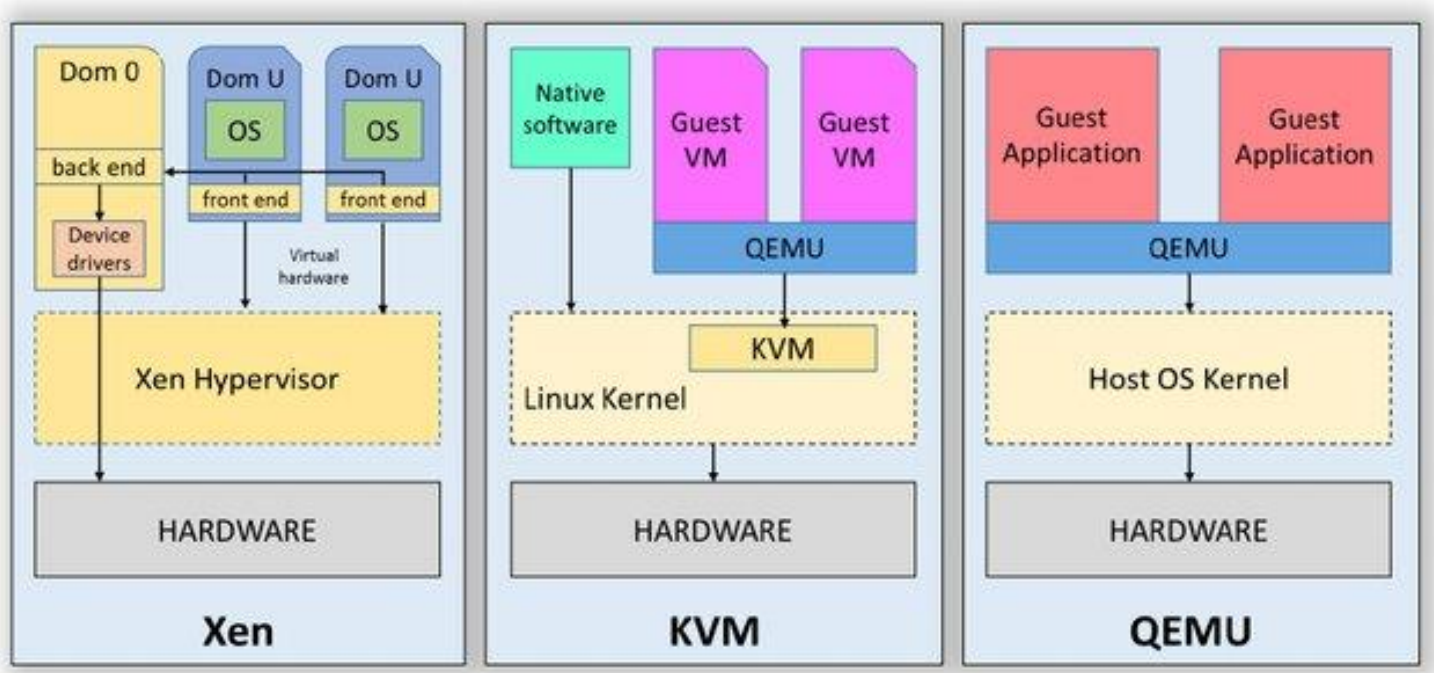
- **VMCS allows quick context switch and isolation**
  - Save/restore host/guest state
  - Fine grained control over the instructions executed in the guest
    - Guests execute without changes until they trigger an VM Exit
    - Hypervisors catch the VM Exit and decide what to do (e.g. emulate)
  - Isolated memory spaces
  - Guests run in ring 0 (of the VMX Non root)


KVM
Init “Hardware”
VMenter
VM
VMexit
Trap
Emulate
Interesting instruction
VMenter

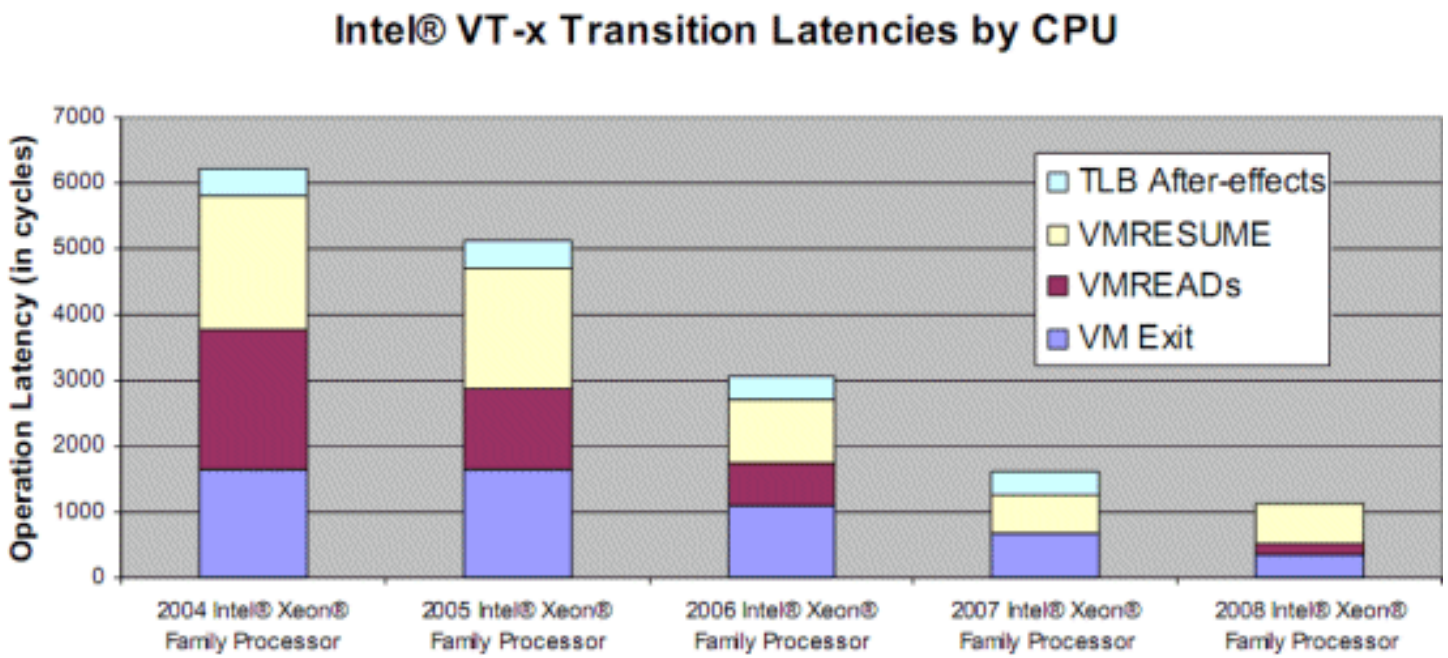

VM 1
Non-root
User apps
Ring 3
Nothing
Ring 2
Nothing
Ring 1
OS
Ring 0
VM 2
Non-root
User apps
Ring 3
Nothing
Ring 2
Nothing
Ring 1
OS
Ring 0
Root
User apps
Ring 3
Nothing
Ring 2
Nothing
Ring 1
VMCS:VM 1
VMM
VMCS:VM 2
Ring 0


QEMU
VMX root
Ring 3
Emulated
Platform
Machine Model
Sysbus
VGA
PPB
NIC
IDE
Memory
vCPU
vCPU
CPU Thread
IO Thread
main loop
Guest
VMX Non root
Ring 3
Guest
VMX Non root
Ring 0
gCR3
H/W EPT TLB
Guest Physical
vCPU
vCPU
VM Entry
network Driver
Disk Driver
VGA Driver
Kernel
VMX root
Ring 0
KVM API
/dev/kvm
handle Exit
VM Exit
EPT Violate
HPA
EPTP
Y
I/O
Y
Signal Pending
guest pages
VMCS
VMCS
Host State
Guest State
Host State
Guest State
VT-x
CPU
enter
guest mode
CPU
CPU

# Comparing VMMs

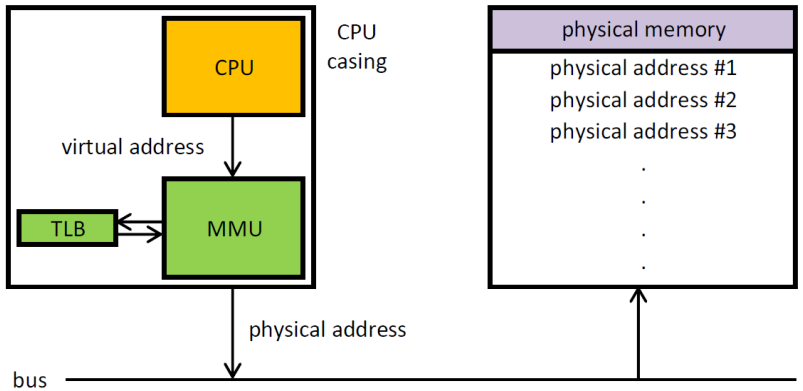



Song, Yang & Wang, Haoliang & Soyata, Tolga. (2015). Hardware and Software Aspects of VM-Based Mobile-Cloud Offloading. 10.4018/978-1-4666-8662-5.ch008.

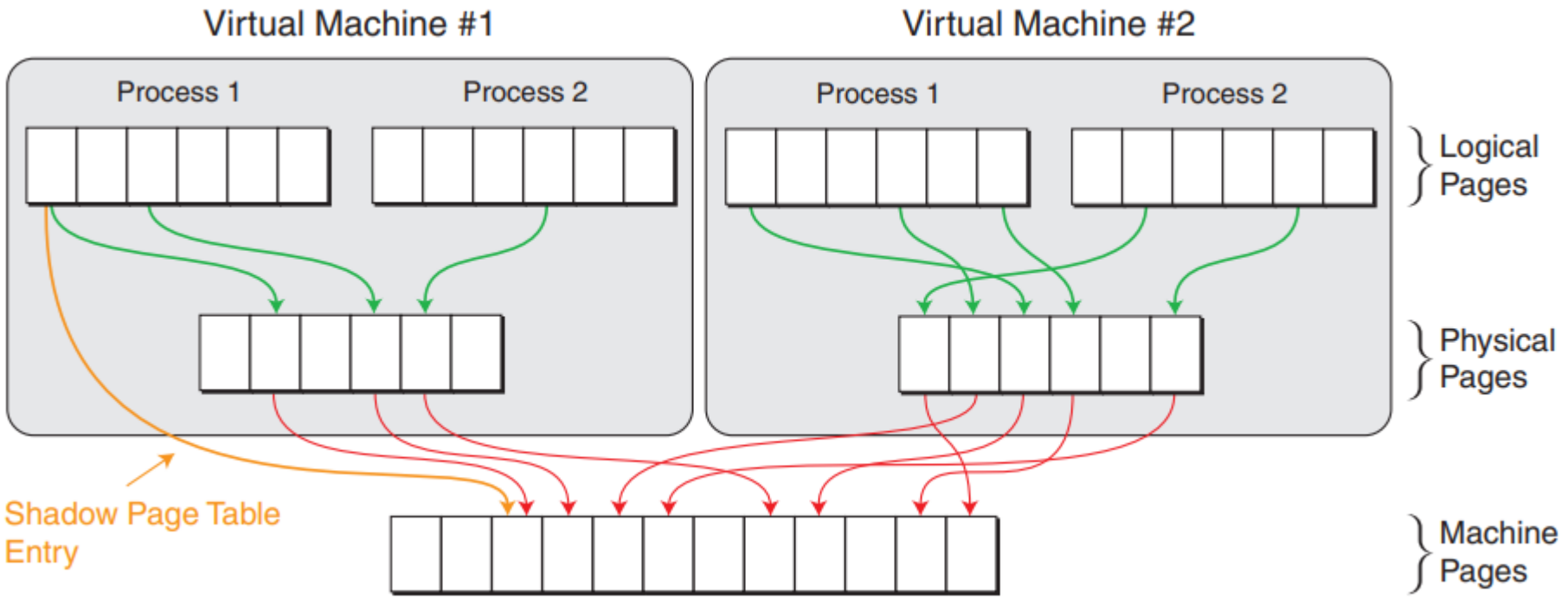

Intel® VT-x Transition Latencies by CPU
Operation Latency (in cycles)
7000
6000
5000
4000
3000
2000
1000
0
TLB After-effects
VMRESUME
VMREADs
VM Exit
2004 Intel® Xeon® Family Processor
2005 Intel® Xeon® Family Processor
2006 Intel® Xeon® Family Processor
2007 Intel® Xeon® Family Processor
2008 Intel® Xeon® Family Processor

# Kernel Samepage Merging

- **Guests will see a memory area as provided by the VMM with pages**
  - Looks like standard RAM
- **Pages from the VM address space are mapped into pages in the host**
  - VMM Address space if using hosted virtualization/binary translation
  - Specific Address space if using hardware assisted virtualization
- **An MMU performs virtual memory management**
  - memory protection
  - cache control
  - bus arbitration



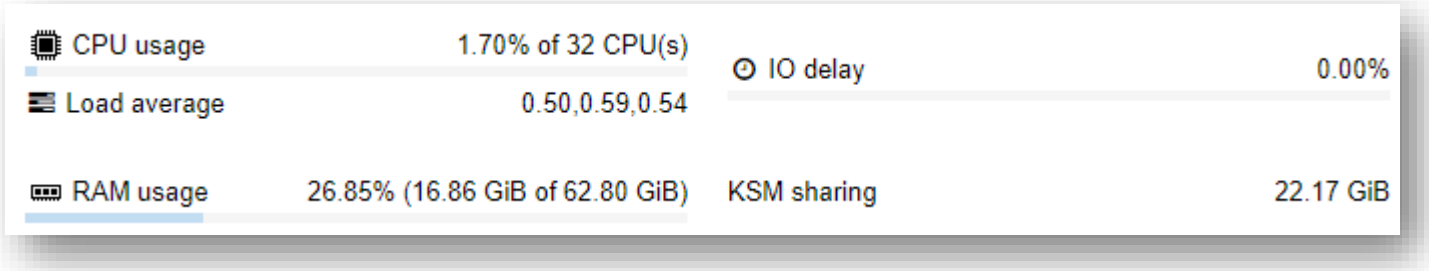
# Kernel Samepage Merging



# Kernel Samepage Merging

- **If VMM has several similar guests, RAM will also be similar**
  - Binaries, shared libraries, (data?)

- **Virtual memory pages can be merged to a single real memory page**
  - Copy on write: if guest modifies a shared memory page, a copy is created in the host



# Ballooning

- **An OS will always use most RAM available**
  - Kernel
  - Frame buffers
  - Mapped areas
  - Shared Libraries in use
  - Active applications
  - But also: cache and IO buffers
- **Some may not be actually needed: Page cache**

**In the example: 5851MB total, 1841MB used but only 1730MB free**

```
               total        used        free      shared  buff/cache   available
Mem:            5851        1841        1730         143        2279        3583
Swap:              0           0           0
```

# Ballooning

- **VMM will is not able to identify how much memory is actually being used**
  - Guests will always have full memory allocated

- **Ballooning: Driver existing inside the guest**
  - Communicates with Hypervisor through API
  - Can be inflated by allocating memory
  - Memory allocated is identified
  - Guest OS purges page cache

| | total | used | free | shared | buff/cache | available |
|---|---|---|---|---|---|---|
| Mem: | 5851 | 1800 | 3581 | 143 | 469 | 3627 |
| Swap: | 0 | 0 | 0 | | | |

# I/O Virtualization

- **I/O is one of the most importante bottlenecks of virtualization**
  - **Algorithms mostly use safe instructions**: performance is high
  - **I/O uses communication with devices**: may require privileged instructions, therefore, performance is much slower
  - Higher I/O load scenarios will behave diferently than pure compute scenarios

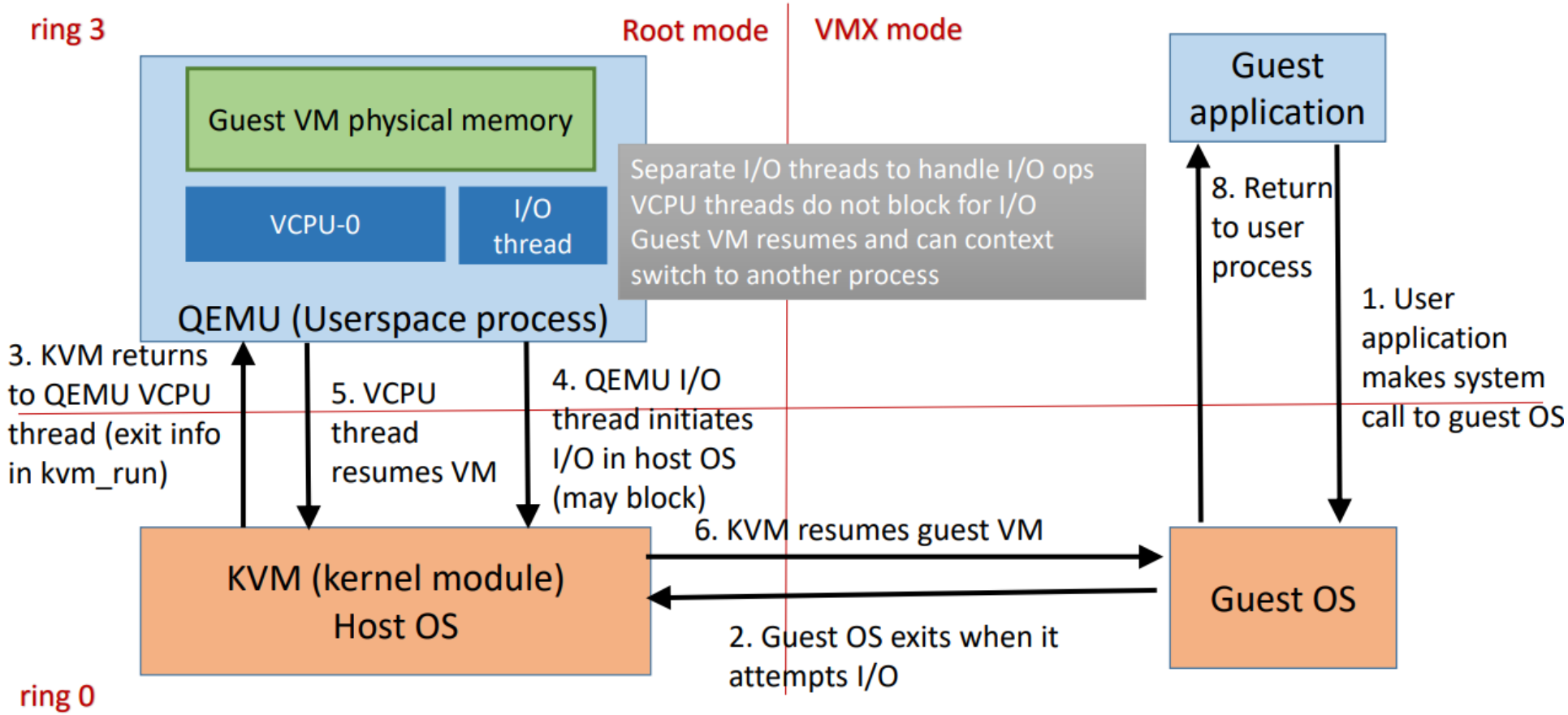
- **I/O virtualization strategies:**
  - **Emulation**: I/O triggers trap and VMM emulates I/O
  - **Direct** I/O or Passthrough: device or a slice of it is passed to the Guest

- **Specific drivers and VMM solutions will provide optimizations for this.**

# I/O Virtualization

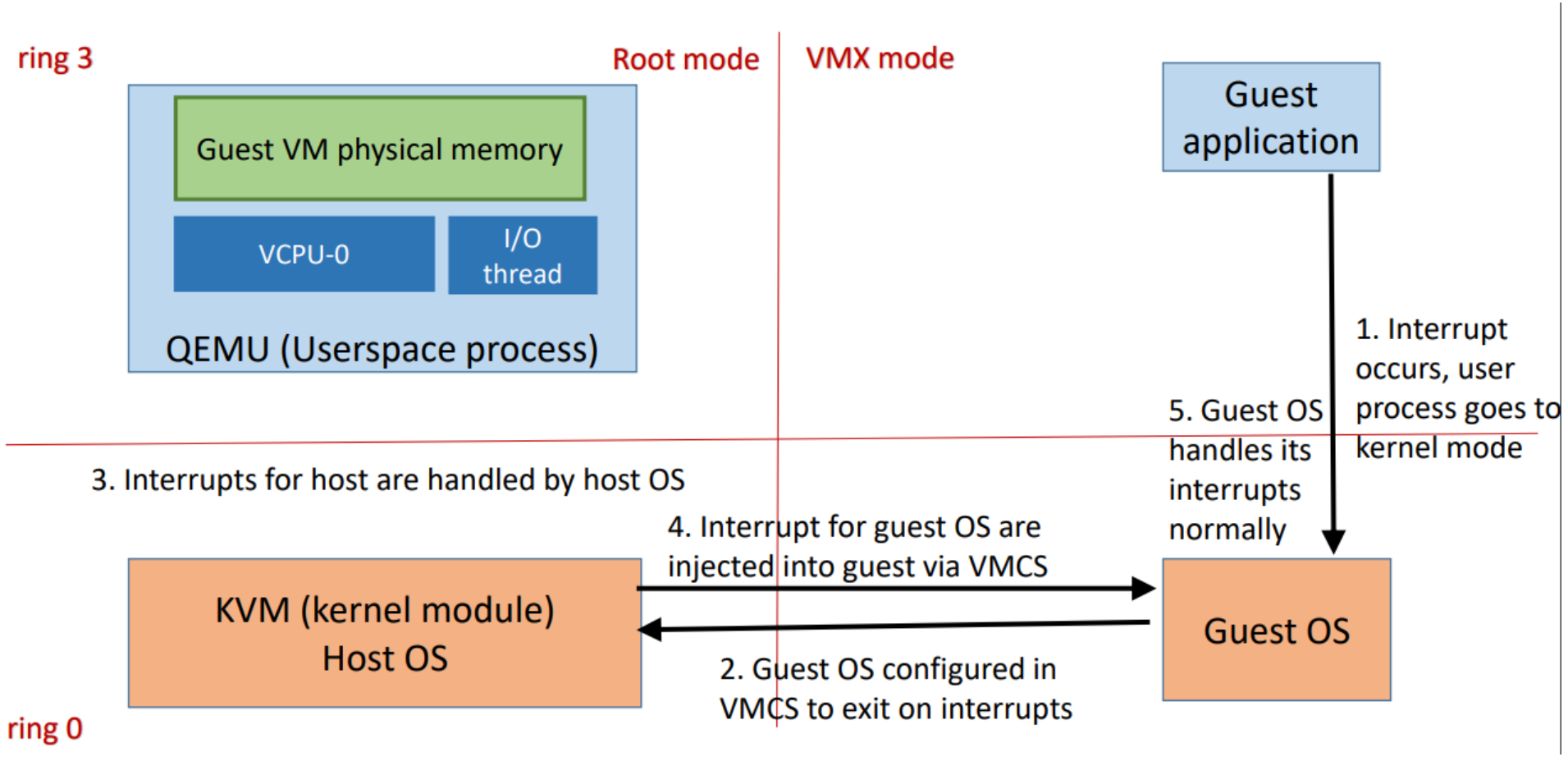
- **Device memory is exposed as registers (command, status, data etc.)**
  - I/O happens by reading/writing this memory
  - E.g., write command into device register to begin I/O
- **OS can read/write device registers in two ways:**
  - **Explicit I/O**: in/out instructions in x86 can write to device memory
  - **Memory mapped I/O**: Some memory addresses are assigned to device memory and are not to RAM. I/O happens by reading/writing this memory.
- **Accessing device memory (via explicit I/O or memory mapped I/O) can be configured to trap to VMM**
- **Device raises interrupt when I/O completes (alternative to polling)**
  - Modern I/O devices perform DMA (Direct Memory Access) and copy data from device memory to RAM before raising interrupt
  - Device driver provides physical address of DMA buffers to device

# QEMU/KVM IO



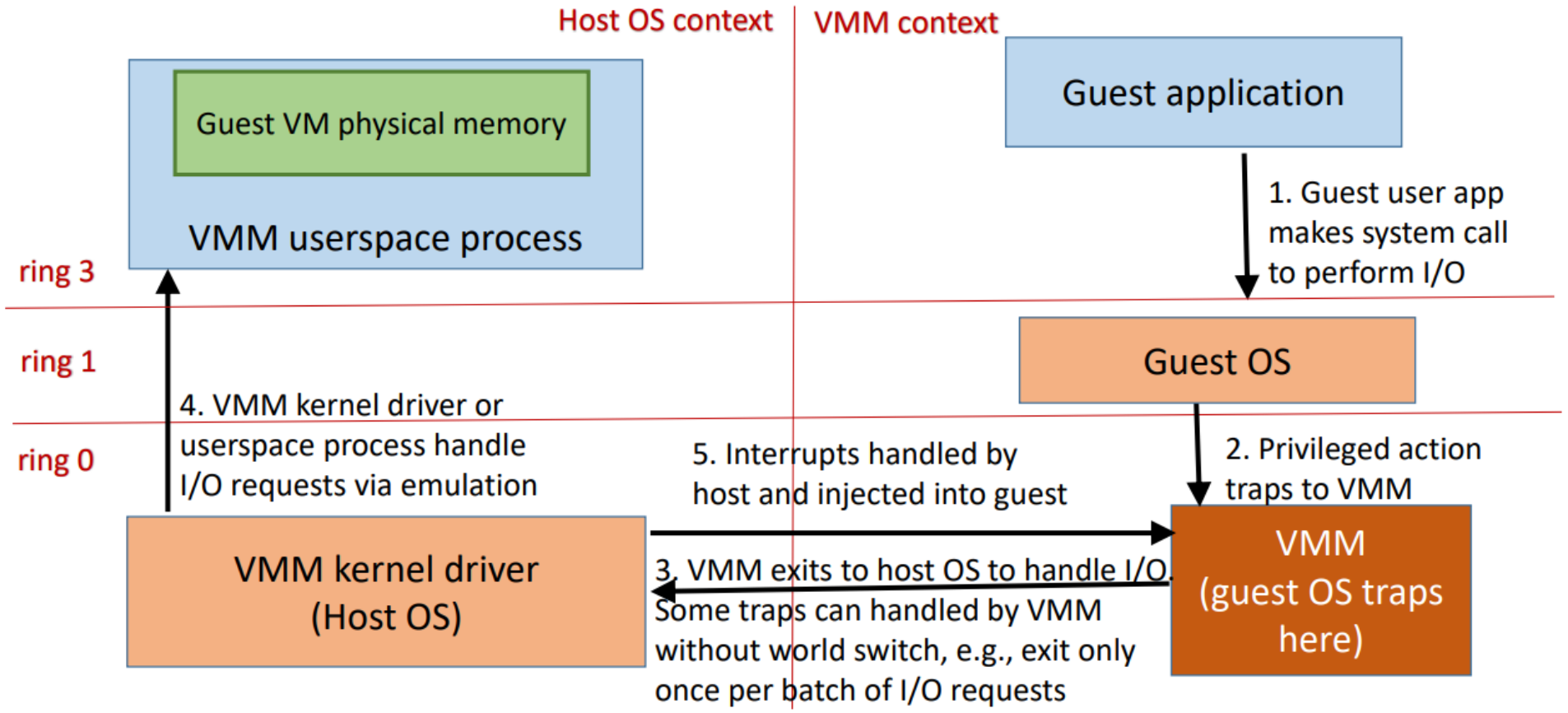

Source: https://www.cse.iitb.ac.in/~mythili/virtcc/

# QEMU/KVM Interrupt handling



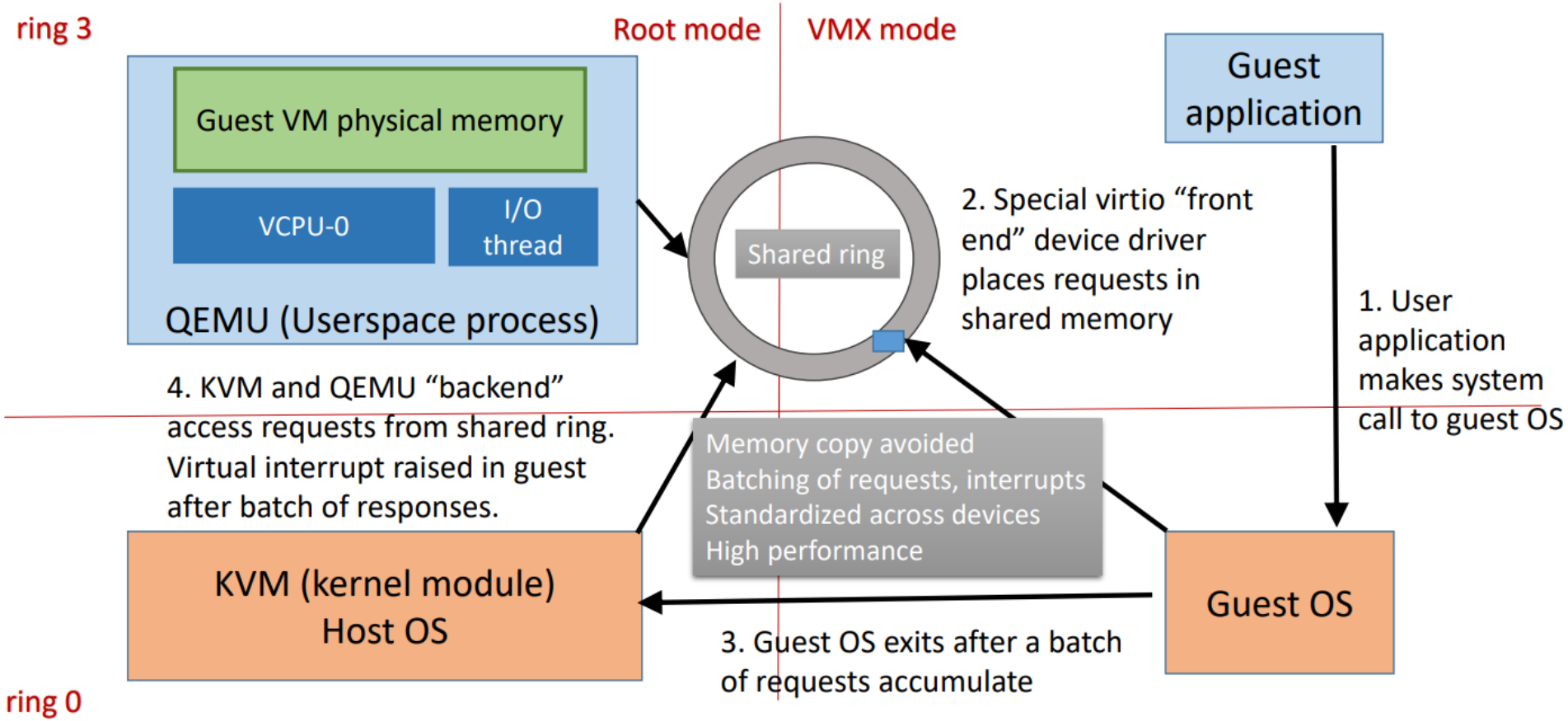

Source: https://www.cse.iitb.ac.in/~mythili/virtcc/

# Full virtualization




Source: https://www.cse.iitb.ac.in/~mythili/virtcc/

# QEMU VirtIO



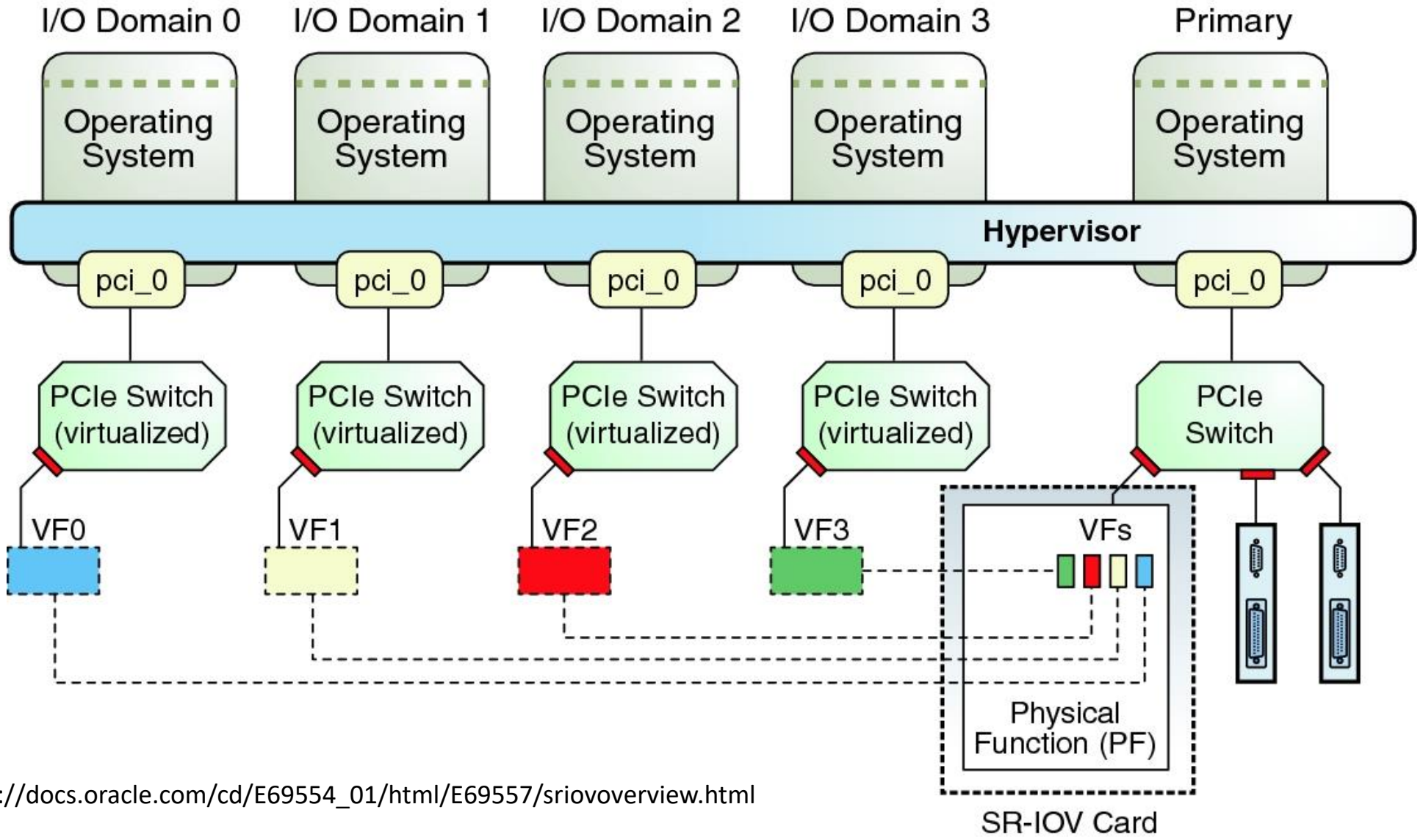

Source: https://www.cse.iitb.ac.in/~mythili/virtcc/

# Direct I/O

- **Device is directly mapped to guest**
  - Host may lose access to mapped device
    - E.g. USB or PCI devices
- **Much more eficient than emulated access as guest OS will use native driver to access mapped device**
- **Approach is mandatory for several use cases**
  - Legacy hardware without support for virtualization
  - When high performance is required
    - Low latency, low jitter networking: Industrial machines, Smart Grids
    - High bandwidth use cases: GPUs and other accelerators

# SR-IOV

- **Single Root  IO Virtualization**
  - Extension to the PCI Express specification
  - Network cards, RAID controllers, GPUS…
- **Device presents itself as:**
  - one **physical function (PF)**: managed by the host and representing the core of the device
  - multiple **virtual functions (VF)**: managed by guests and providing a specific, contextualized functionality
- **Guests use VFs directly without host intervention**
  - Cards have internal IOMMU mapping Guest Physical Addresses to internal buffers
  - Multiple data paths, one per guest




https://docs.oracle.com/cd/E69554_01/html/E69557/sriovoverview.html

# Live Migration

- **Essential feature for operations continuity in virtualized environments**
- **Scheduled maintenance: Guests can be moved to alternative hosts, allowing upgrades of original host**
- **Failure recovery: If host starts malfunctioning, some VMs may be recovered before they fail**
- **Infrastructure optimization: keep hosts optimally loaded**
  - Reduce number of VMs in overloaded hosts
  - Shutdown empty hosts
  - Equalize guests across infrastructure

# Live Migration

- **Guests are files + configuration + active context**
  - Files: Persistence
    - May be globally available (not requiring any copy)
  - Configuration: Network addresses, VM description
  - Context: RAM, CPU context

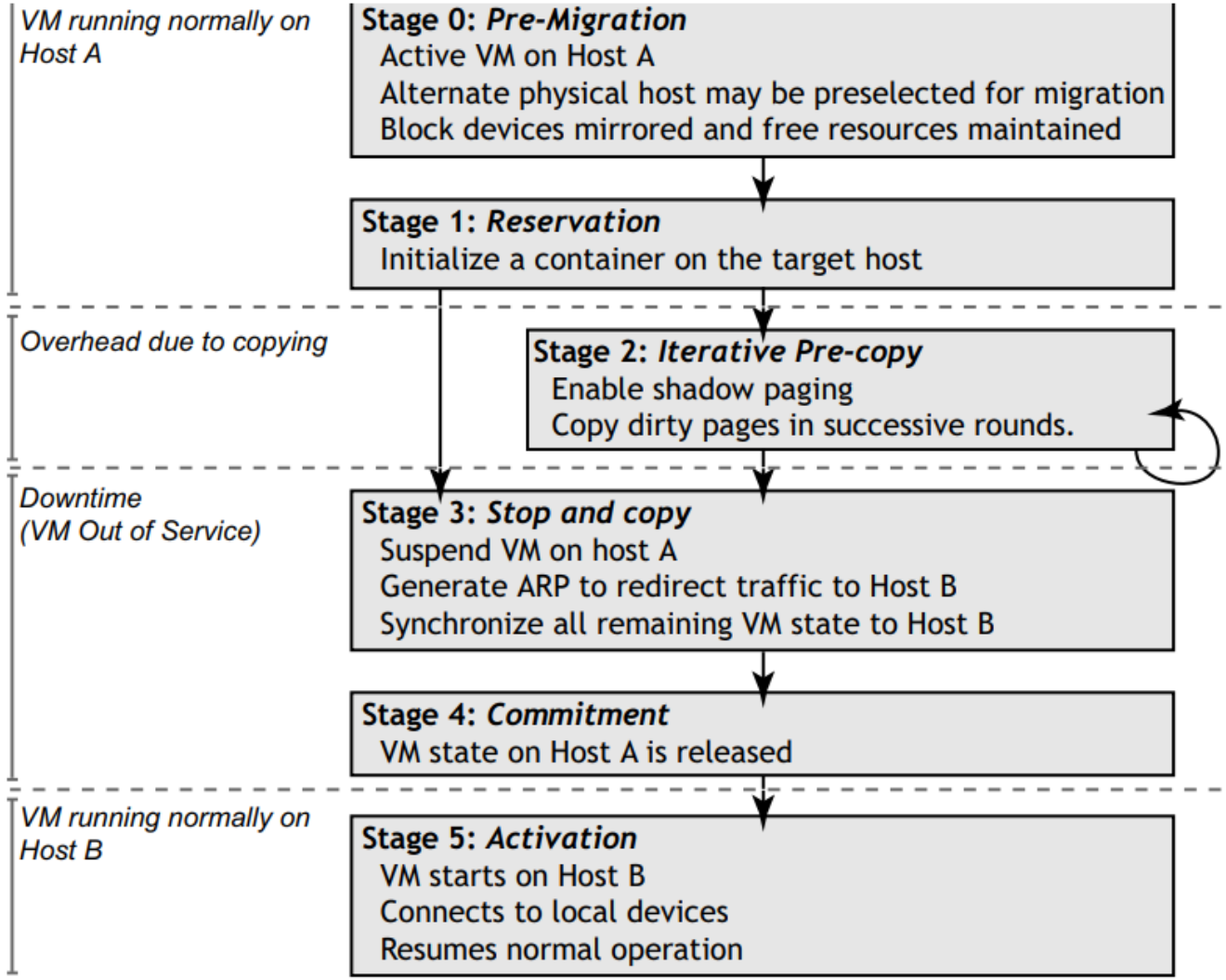
- **Guests can be migrated between hardware hosts**
  - In same Datacenter, or in different Datacenters
  - Across the world?

# Live Migration Phases

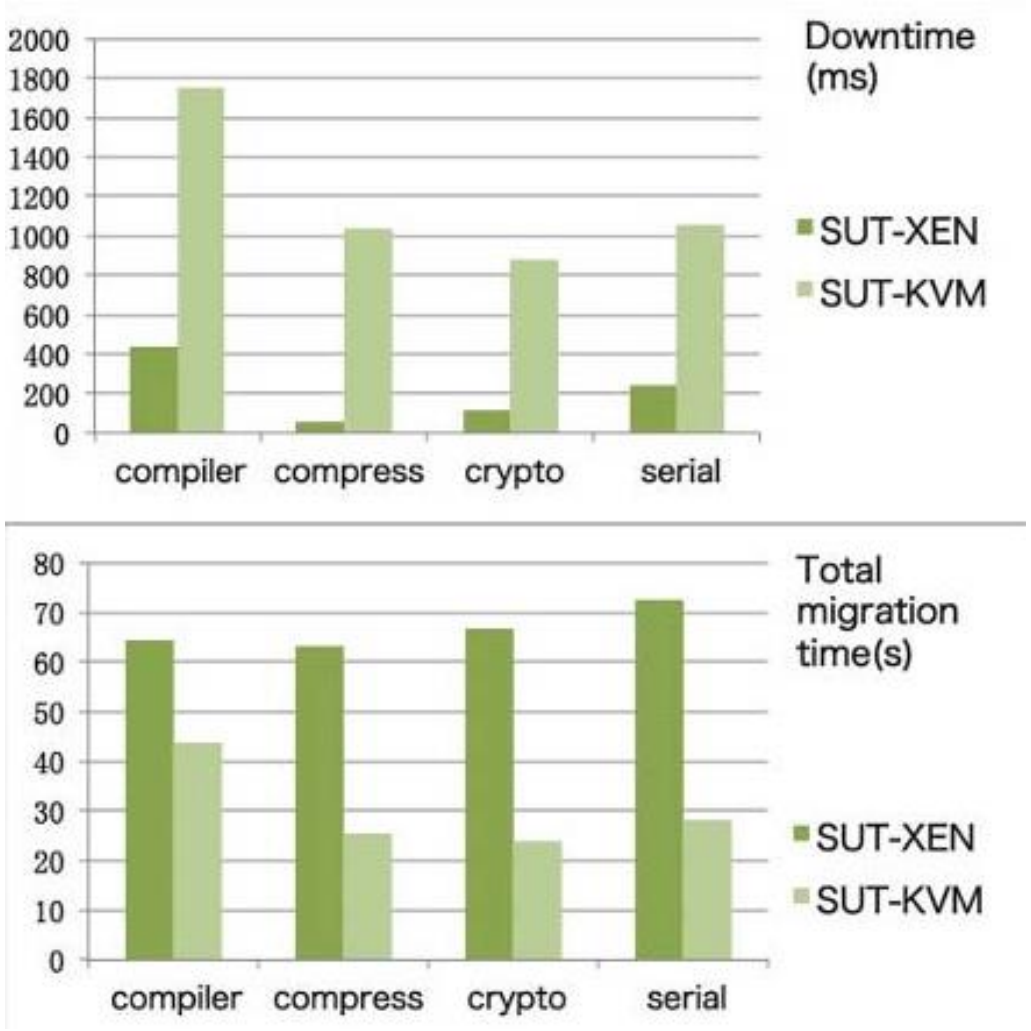
- **Setup phase**
  - Destination host allocates resources
- **Push phase**
  - Sends data from running VM to destination host
- **Stop-and-copy phase**
  - VM context is migrated, network is reconfigured
  - Some downtime may happen during this phase
- **Pull phase**
  - VM is started, missing data can be fetched

# Live Migration Approaches

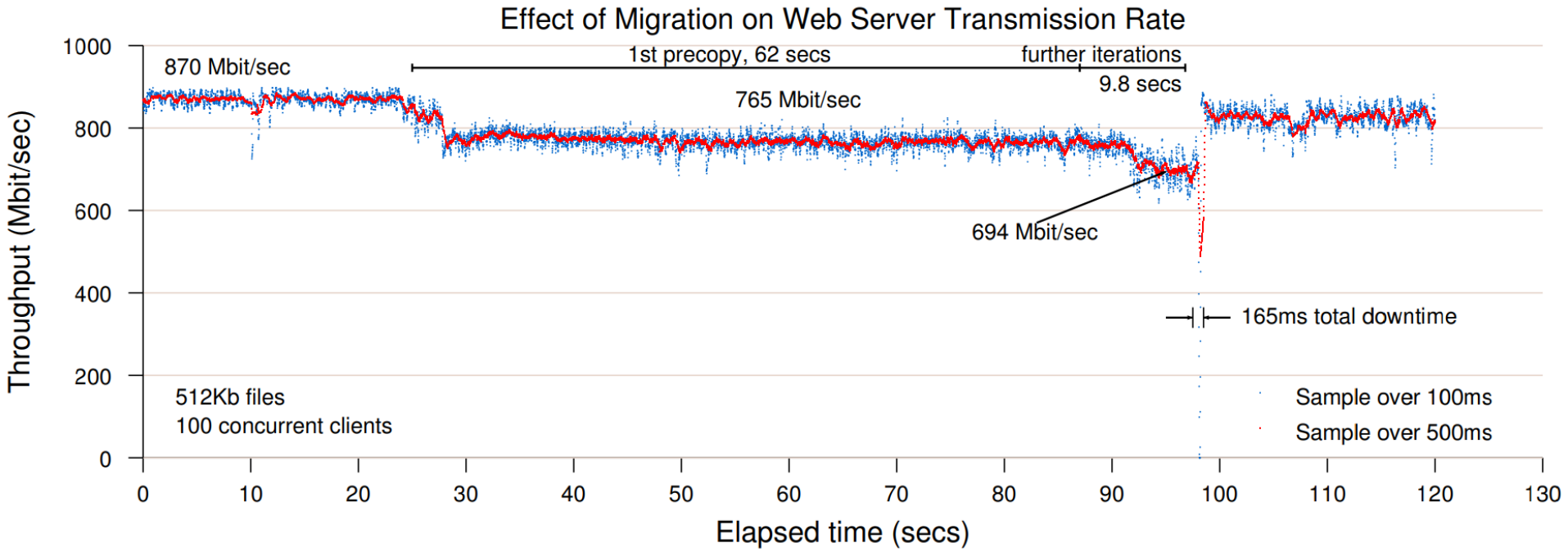
- **Stop and Copy**
  - VM Stopped, Data is copied, VM started on destination
  - Very slow, high downtime
- **Pre-Copy Migration**
  - Data is copied, VM is paused, and VM is resumed on the destination
- **Post-Copy Migration**
  - Minimal data is copied, VM is replicated to destination, storage handles are updated, missing pages are copied on demand
- **Hybrid Migration**
  - Mix of previous. Some data is copied, VM is briefly stopped and started, remaining data is pulled



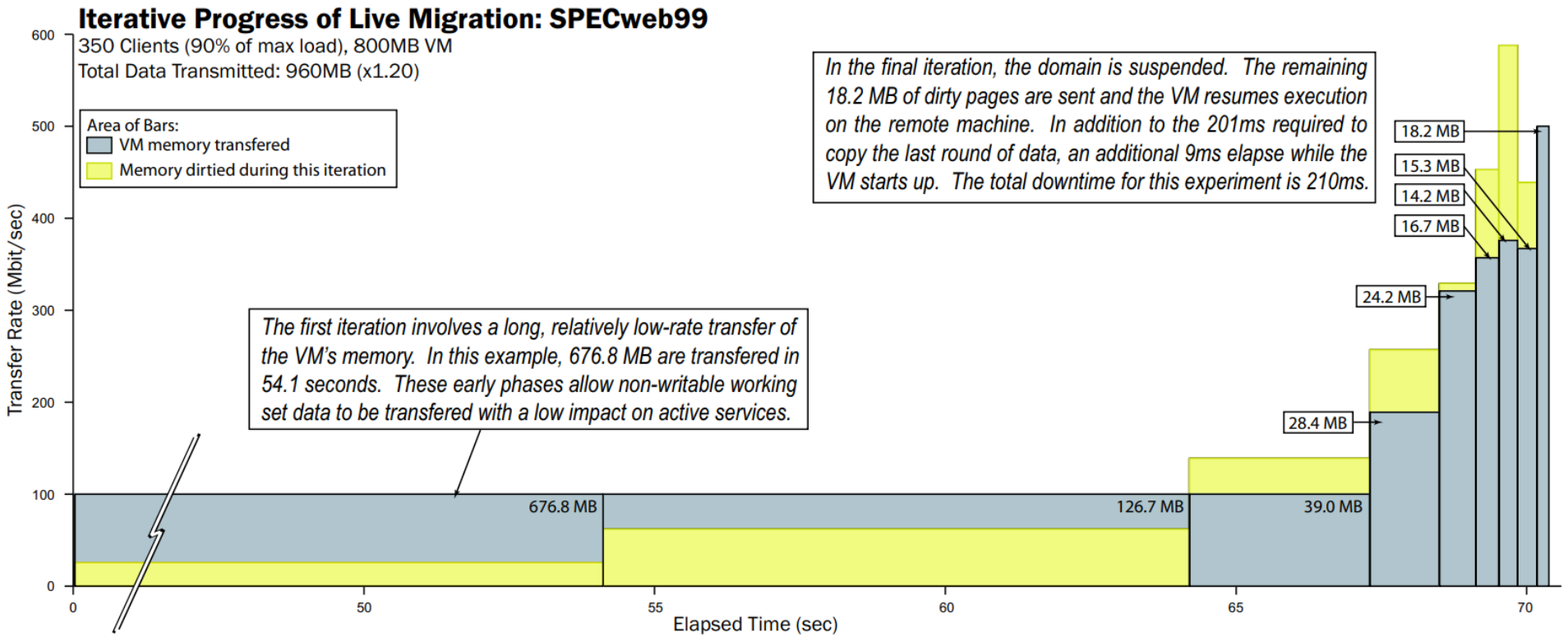

*Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live migration of virtual machines. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05). USENIX Association, USA, 273–286.*



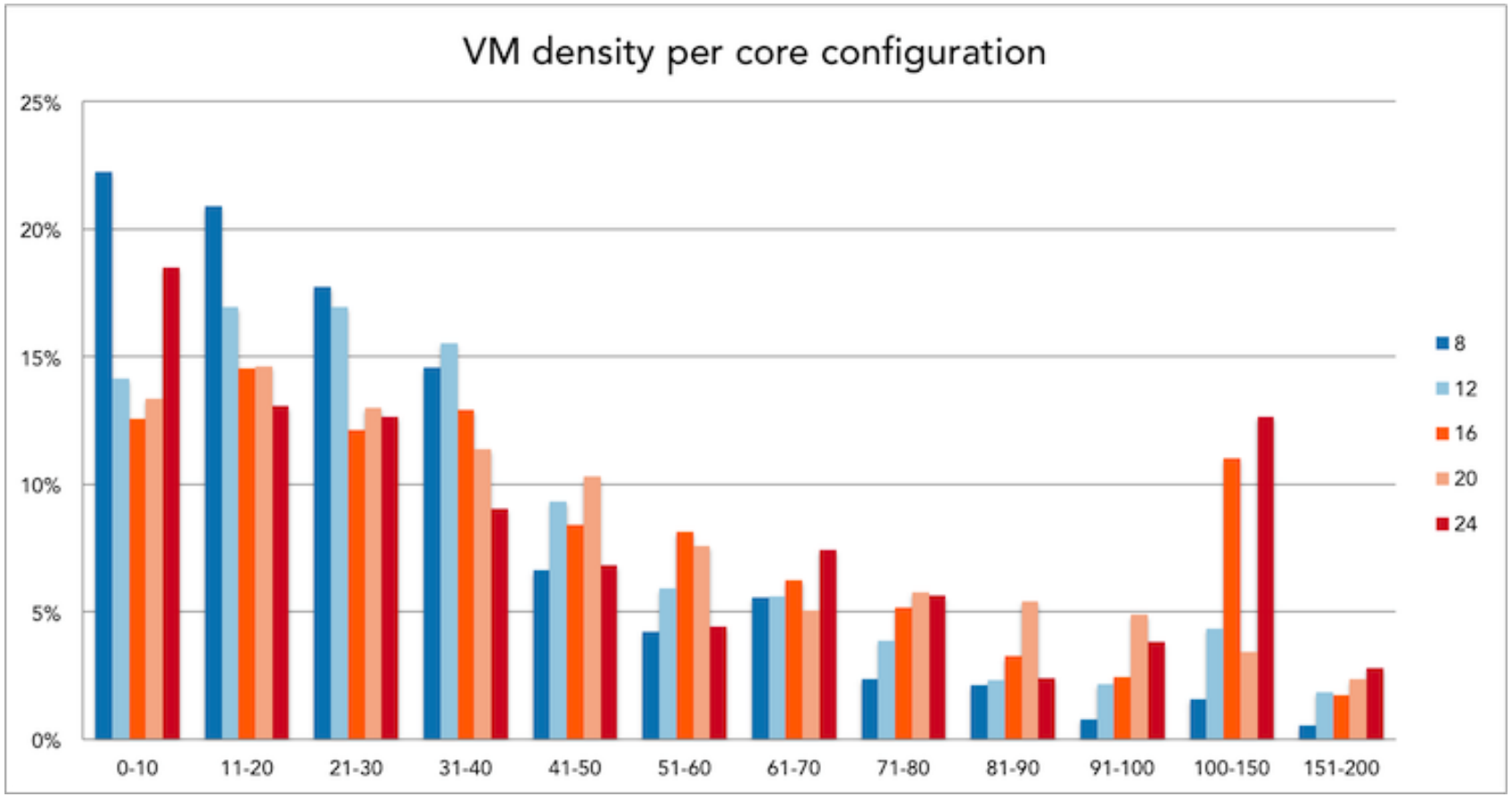

Dawei Huang, Deshi Ye, Qinming He, Jianhai Chen, Kejiang Ye, “Virt-LM: A Benchmark for Live Migration of Virtual Machine”, Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering, 2011




Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live migration of virtual machines. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05). USENIX Association, USA, 273–286.


Iterative Progress of Live Migration: SPECweb99
350 Clients (90% of max load), 800MB VM
Total Data Transmitted: 960MB (x1.20)
Area of Bars:
VM memory transfered
Memory dirtied during this iteration
In the final iteration, the domain is suspended. The remaining 18.2 MB of dirty pages are sent and the VM resumes execution on the remote machine. In addition to the 201ms required to copy the last round of data, an additional 9ms elapse while the VM starts up. The total downtime for this experiment is 210ms.
The first iteration involves a long, relatively low-rate transfer of the VM's memory. In this example, 676.8 MB are transfered in 54.1 seconds. These early phases allow non-writable working set data to be transfered with a low impact on active services.
Transfer Rate (Mbit/sec)
Elapsed Time (sec)
676.8 MB
126.7 MB
39.0 MB
28.4 MB
24.2 MB
18.2 MB
15.3 MB
14.2 MB
16.7 MB

# Infrastructure optimization




Frank Denneman, “Insights into VM density”, http://frankdenneman.nl/2016/02/15/insights-into-vm-density/

# Checkpointing and cloning

- **VM Checkpointing**
  - **VMM can create snapshots at specific instants or periodically**
  - Mostly useful for high availability as VMM can rapidly recover the guest to the last checkpoint

- **VM Cloning**
  - **VMM can fork many guests with same base image**
  - Mostly useful for parallel execution and elasticity
  - VMM can adjust the number of guests to meet some performance metric

Remus: High Availability via Asynchronous Virtual Machine Replication Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield

SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing H. Andrés Lagar-Cavilla, Joseph A. Whitney, Adin Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno, M. Satyanarayanan

# How to virtualize – Storage (more later)

- **Hard disks**
  - Real device
    - Maximum performance, lower flexibility
  - Fixed size: pre-allocated with maximum size
    - Higher performance, wasted storage capacity
  - Dinamic Size: allocated as needed
    - Higher efficiency, lower performance
  - Remote Disk: Stored in SAN/NAS
    - Maximum flexibility, lower performance
- **Optical drives**
  - Existing image
  - Passthrough real device

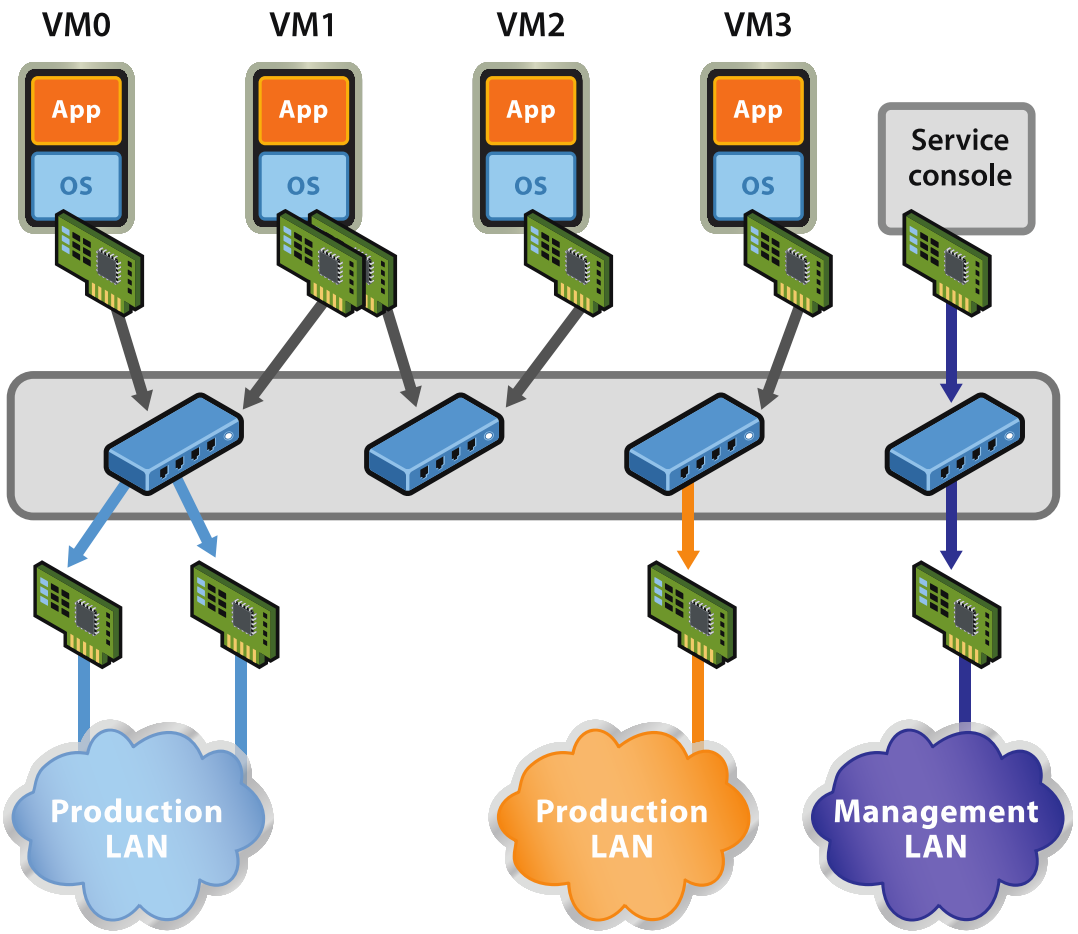
# How to virtualize - Network

- **Host Only: A TAP interface connects host to guest**
- **Internal Network: Communication with a virtual network just for VMs**

- **NAT: guest is behind NAT**
  - Can access the outside, but cannot receive connections
  - VMM implements Router + NAPT services

- **Bridged: A “real” network device is connected to the guest through a virtual switch**
  - Variations: OpenVSwitch, MACVTAP
  - Can be SDN enabled
  - Network Device is not a Network Interface Card!

# How to virtualize - Network




Virtual Interfaces

Virtual Switch

Real Interfaces
802.1Q Tagging

# Network Isolation

- **There must be a method to isolate multiple tenants**
  - Limit horizontal attacks at the infrastructure of other tenants
  - Limit visibility to own scope

- **There must be a method to isolate guests from infrastructure network**
  - Limit attacks at the infrastructure

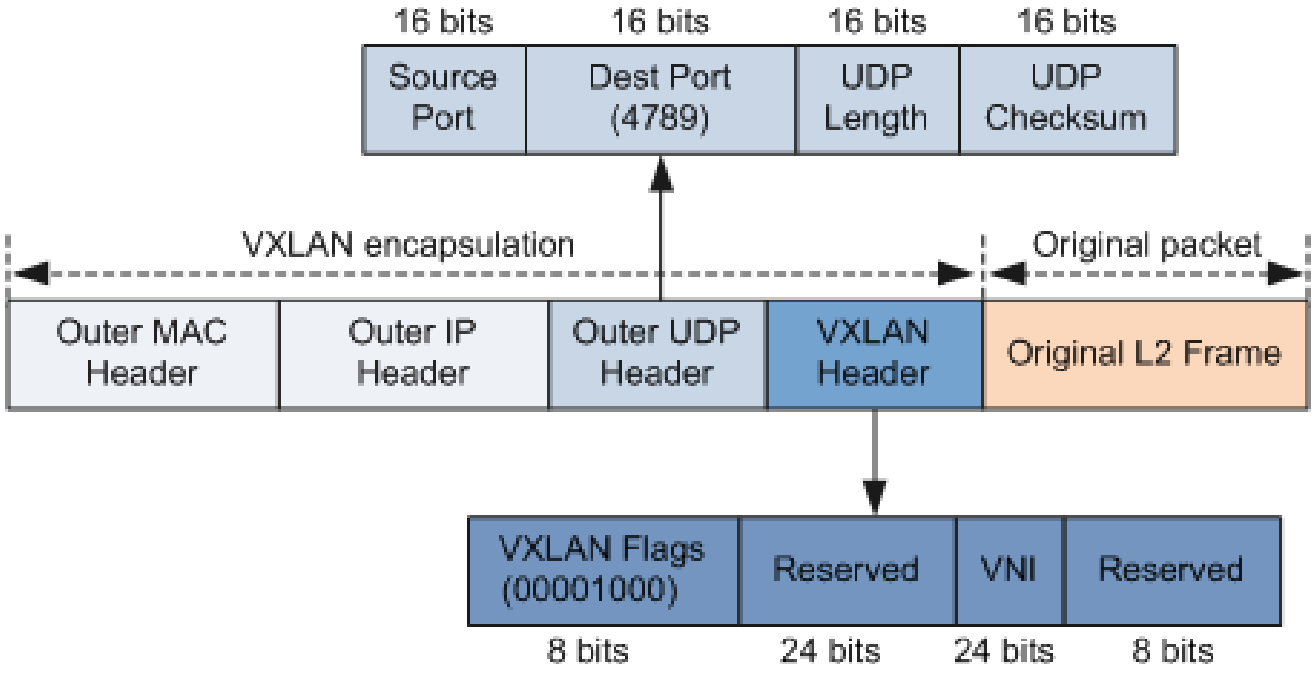
# Network Isolation

- **Traditional solution: 802.1Q VLAN**
  - Each domain (costumer) in its VLAN
  - 4096 IDs available
  - Routers filter access between VLANs
  - L3 Routing between VLANs

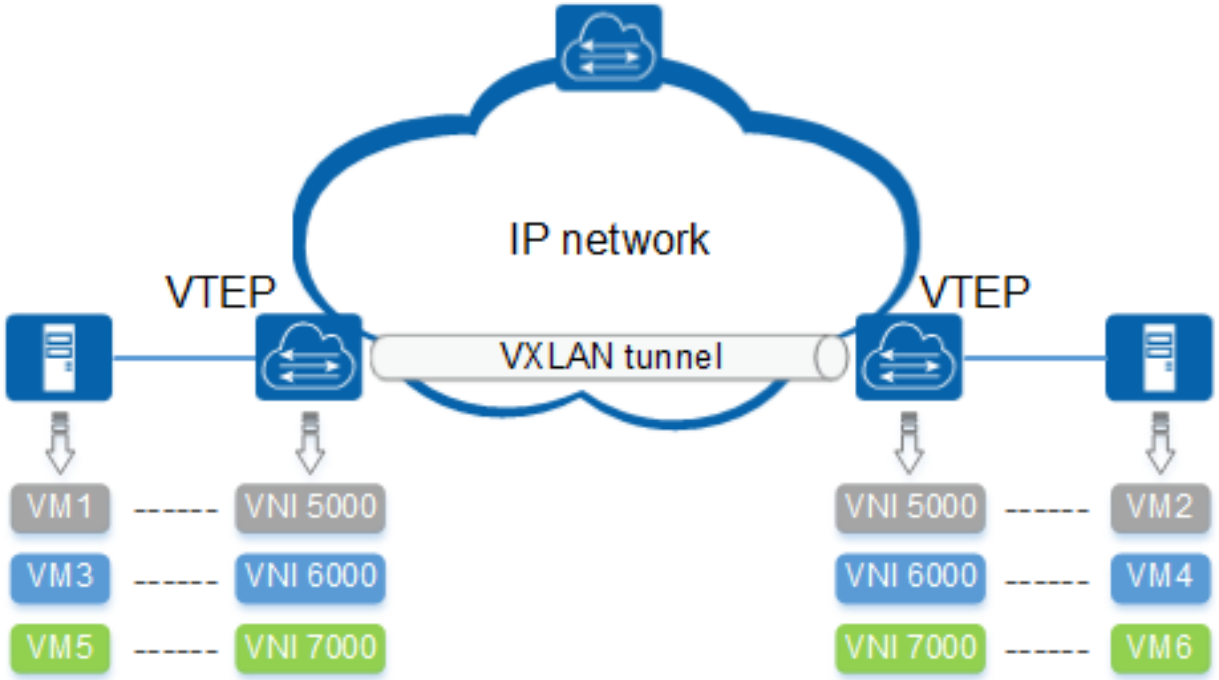
- **Problems: numbers grow rapidly**
  - Some switches are severely limited <150 VLANs
  - Hypervisor must support 802.1Q mapping
  - Burden to manage across a wide campus
    - Requires centralized repository of VLANs
  - Top of Rack switches must handle MACs and VLANs for all VMs
    - Tens of servers, 100s VMs per server

# How to virtualize - Network

- **VXLAN: Ethernet over UDP**
  - Outer Addresses are from host
  - Inner Address are from guest
  - VXLAN header provides 24bit ID
- **Advantages:**
  - Same L2 Network over and IP Network (Internet)
- **Problems:**
  - Hardware support is limited to high end equipment
  - Overhead is much higher, benefiting >1500 MTU

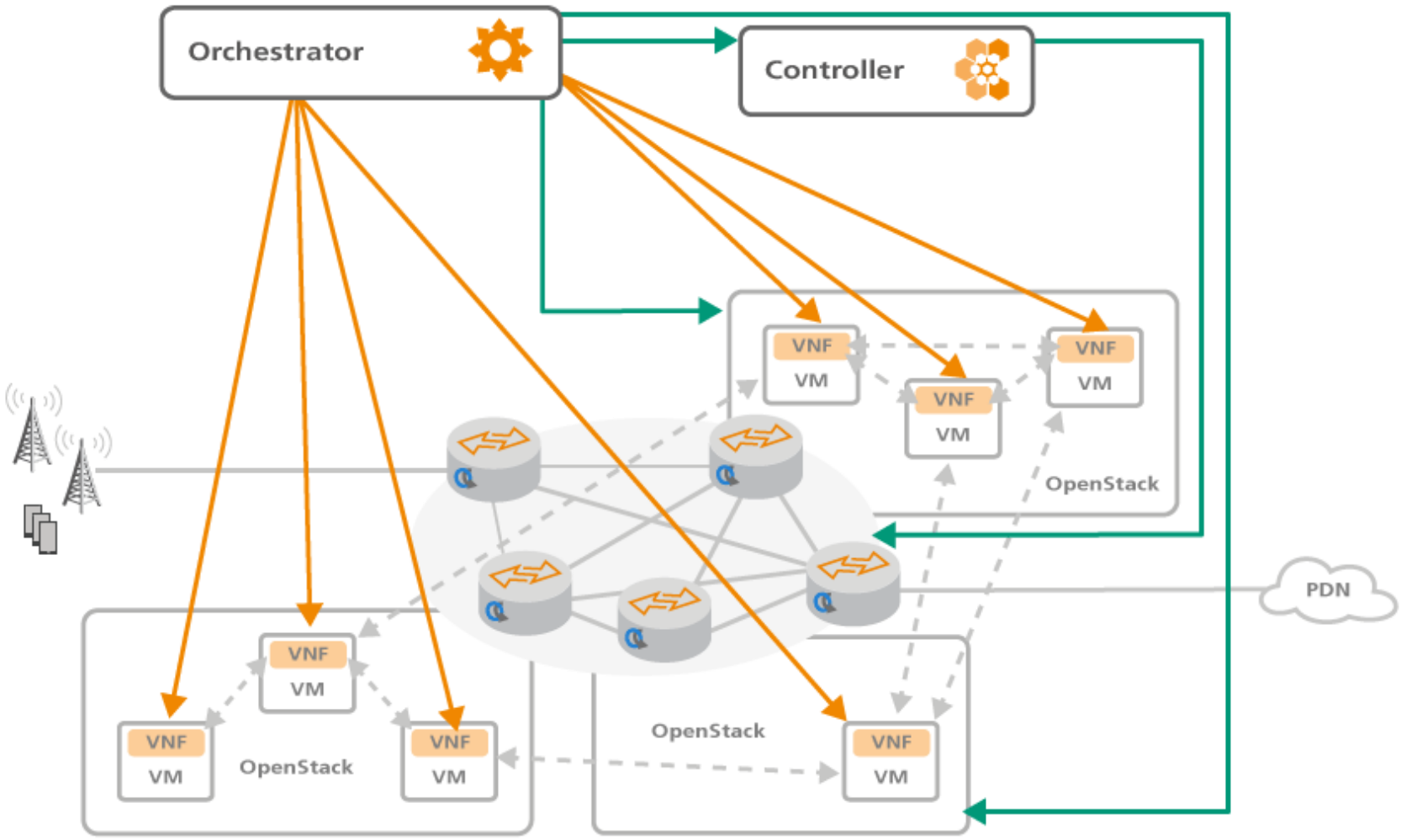

16 bits
16 bits
16 bits
16 bits
Source Port
Dest Port (4789)
UDP Length
UDP Checksum
VXLAN encapsulation
Original packet
Outer MAC Header
Outer IP Header
Outer UDP Header
VXLAN Header
Original L2 Frame
VXLAN Flags (00001000)
Reserved
VNI
Reserved
8 bits
24 bits
24 bits
8 bits

- **VTEP: Virtual Tunnel EndPoint**
- **VNI: Virtual Network Interface**



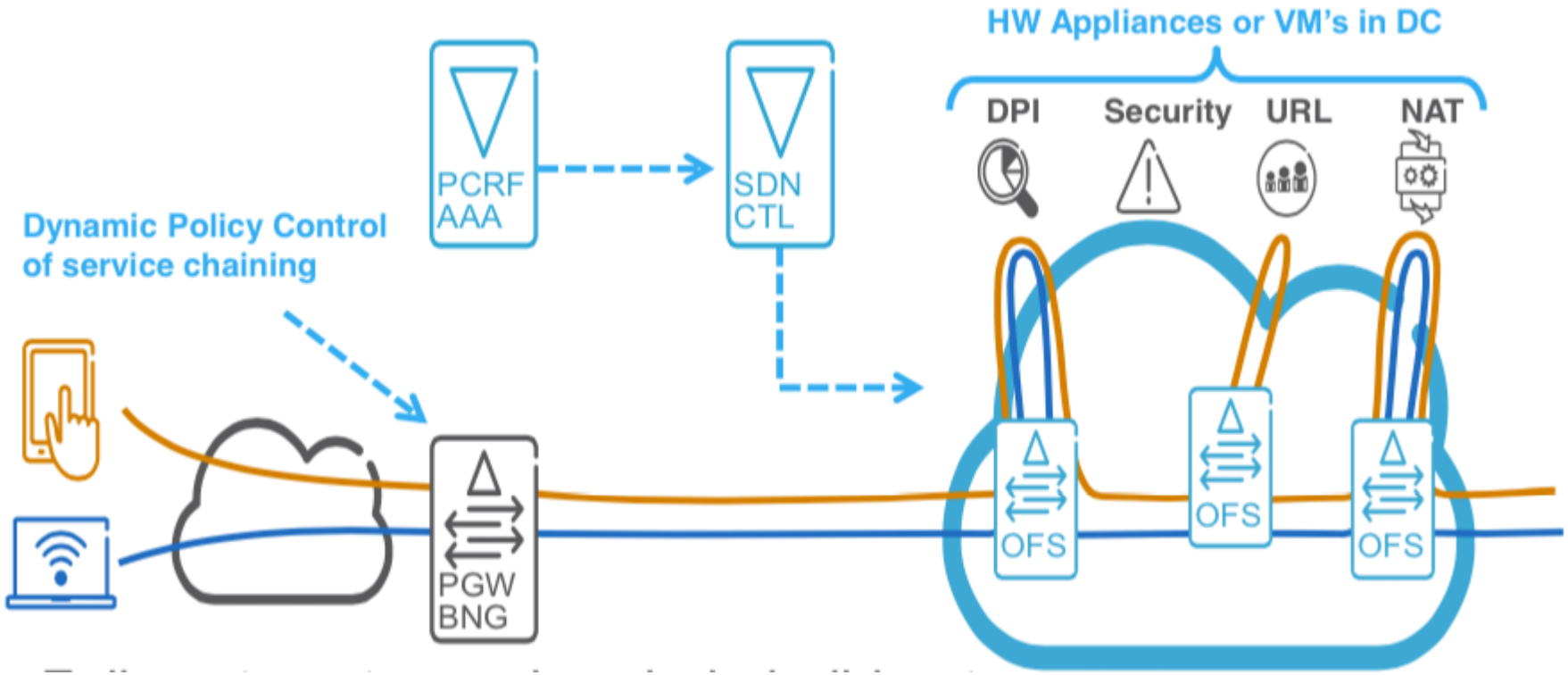
# How to virtualize - Network

- **Software Defined Networks**
  - Integrated with virtualization platform
    - OpenStack, OpenDayLight, VMWare
  - Departure from VLAN/Spanning Tree concepts
- **OpenFlow based switching**
  - Central controller for all L2 infrastructure
  - Integrated with VM orchestartor
- **Model being adoped by DC and Telco**




https://www.opensdncore.org/

# Virtual CPE


HW Appliances or VM's in DC
DPI
Security
URL
NAT
PCRF
AAA
SDN
CTL
Dynamic Policy Control
of service chaining
PGW
BNG
OFS
OFS
OFS