Methods of Cloud Computing

Chapter 2: Virtual Resources

Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

Challenges for IaaS Provider

- Rapid provisioning
  - Resources must be available to the consumer quickly
  - No human interaction during provisioning
- Elasticity
  - Create illusion of infinite resources
  - Yet, manage data center in a cost-efficient manner
- Isolation of different consumers
  - Users must not interfere with each other
- Performance
  - Maintain good performance despite other challenges

Approach: Virtualization

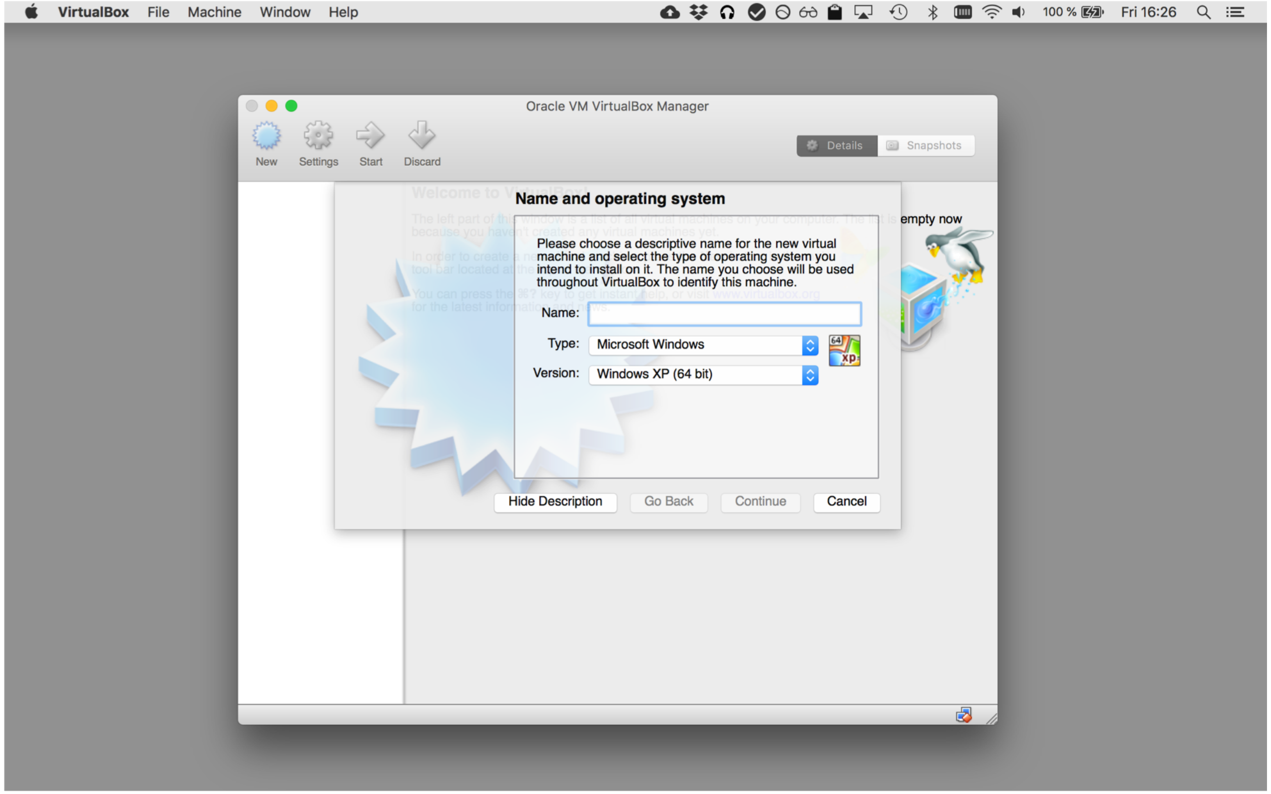
- Very popular idea on IaaS-level
  - Provide resources in the form of virtual machines (VMs)
  - Different types of VMs available
    - Different hardware characteristics (CPU, memory, disk)
    - Different images available (e.g. Windows with Microsoft SQL)
    - Additional storage can be integrated into VMs
  - Providers charge depending on VM type and usage time
    - Predominant model: “pay by the hour” of VM instances
    - Also: pay-per-use of specific resources (e.g. data downloaded from VM instances)

Virtual Machines

- Virtual machine (VM): hardware that is used by an operating system and running processes is not real, but instead virtual => OS and processes run on software emulating hardware
- Use cases:
  - Run a different operating system than installed on host
  - Run applications in isolation
  - Run multiple virtual machines on a single physical host
- VMs often provided by Infrastructure-as-a-Service clouds, e.g. public clouds like Amazon’s EC2

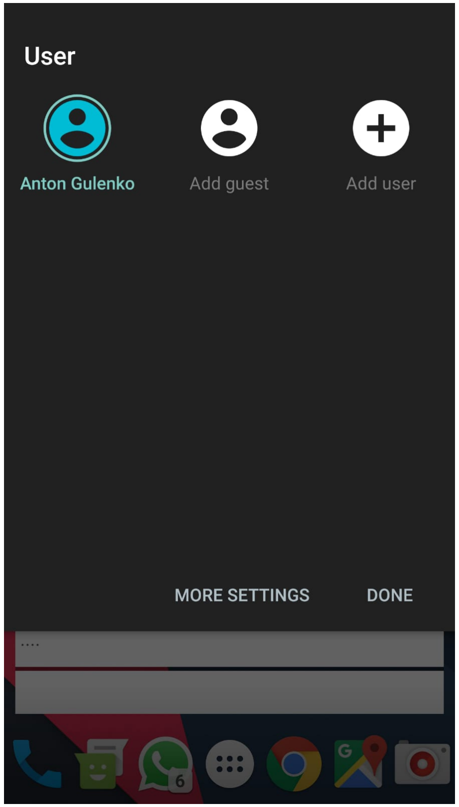
Use Cases (1/3)

- Run a different operating system than your host operating system



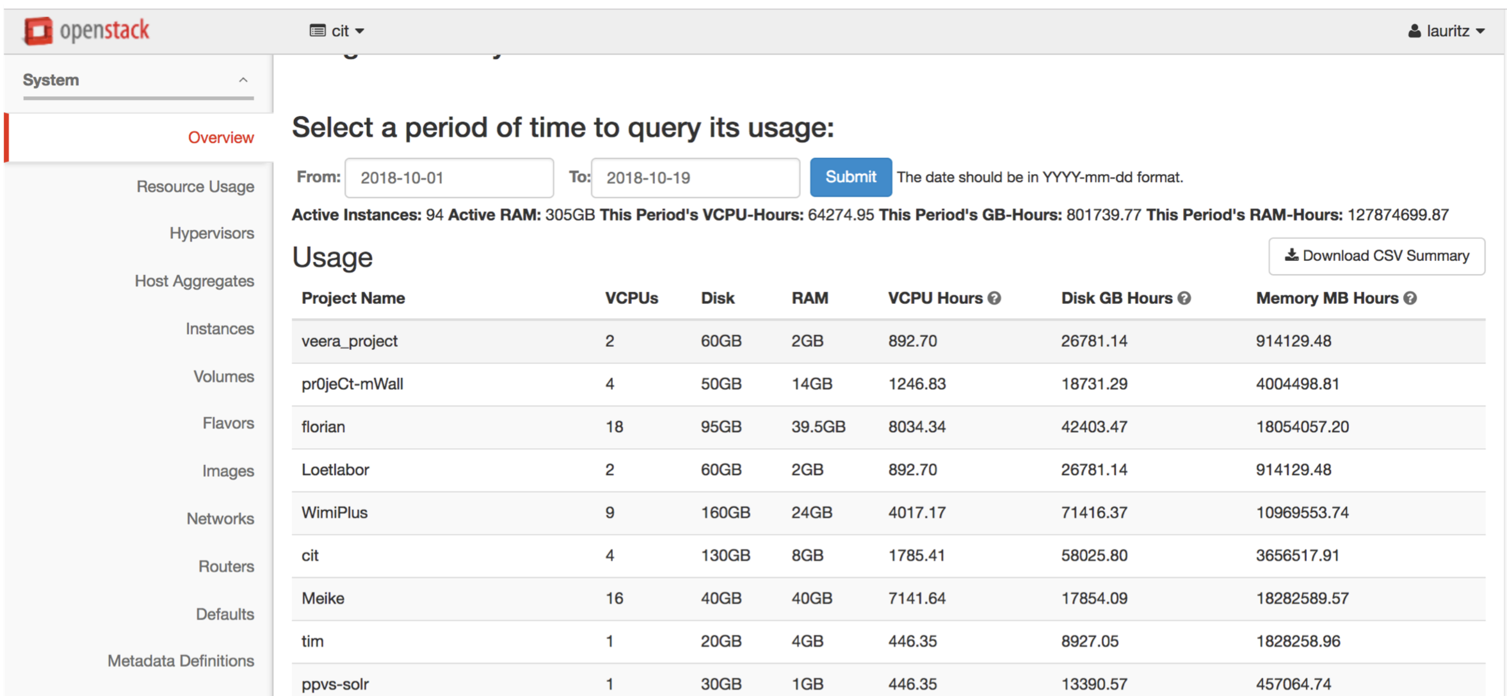
Use Cases (2/3)

- Operating multiple isolated environments on a Smartphone (e.g. for business and private)



Use Cases (3/3)

- Pool of resources shared by multiple users and applications



openstack   cit   lauritz

System

Overview
Resource Usage
Hypervisors
Host Aggregates
Instances
Volumes
Flavors
Images
Networks
Routers
Defaults
Metadata Definitions

Select a period of time to query its usage:

From: 2018-10-01 To: 2018-10-19 Submit The date should be in YYYY-mm-dd format.

Active Instances: 94 **Active RAM:** 305GB **This Period's VCPU-Hours:** 64274.95 **This Period's GB-Hours:** 801739.77 **This Period's RAM-Hours:** 127874699.87

Usage   Download CSV Summary

Project Name	VCPUs	Disk	RAM	VCPU Hours	Disk GB Hours	Memory MB Hours
veera_project	2	60GB	2GB	892.70	26781.14	914129.48
pr0jeCt-mWall	4	50GB	14GB	1246.83	18731.29	4004498.81
florian	18	95GB	39.5GB	8034.34	42403.47	18054057.20
Loetlabor	2	60GB	2GB	892.70	26781.14	914129.48
WimiPlus	9	160GB	24GB	4017.17	71416.37	10969553.74
cit	4	130GB	8GB	1785.41	58025.80	3656517.91
Meike	16	40GB	40GB	7141.64	17854.09	18282589.57
tim	1	20GB	4GB	446.35	8927.05	1828258.96
ppvs-solr	1	30GB	1GB	446.35	13390.57	457064.74

Example: EC2 VM Instances

- Pay by the hour by image, region, and size of VM:

Linux | RHEL | SLES | Windows | Windows with SQL Standard | Windows with SQL Web | Windows with SQL Enterprise | Linux with SQL Standard | Linux with SQL Web | Linux with SQL Enterprise

Region: EU (Frankfurt)

	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
t3.nano	2	Variable	0.5 GiB	EBS Only	$0.006 per Hour
t3.micro	2	Variable	1 GiB	EBS Only	$0.012 per Hour
t3.small	2	Variable	2 GiB	EBS Only	$0.024 per Hour
t3.medium	2	Variable	4 GiB	EBS Only	$0.048 per Hour
t3.large	2	Variable	8 GiB	EBS Only	$0.096 per Hour
t3.xlarge	4	Variable	16 GiB	EBS Only	$0.192 per Hour
t3.2xlarge	8	Variable	32 GiB	EBS Only	$0.384 per Hour
t2.nano	1	Variable	0.5 GiB	EBS Only	$0.0067 per Hour

What is Virtualization?

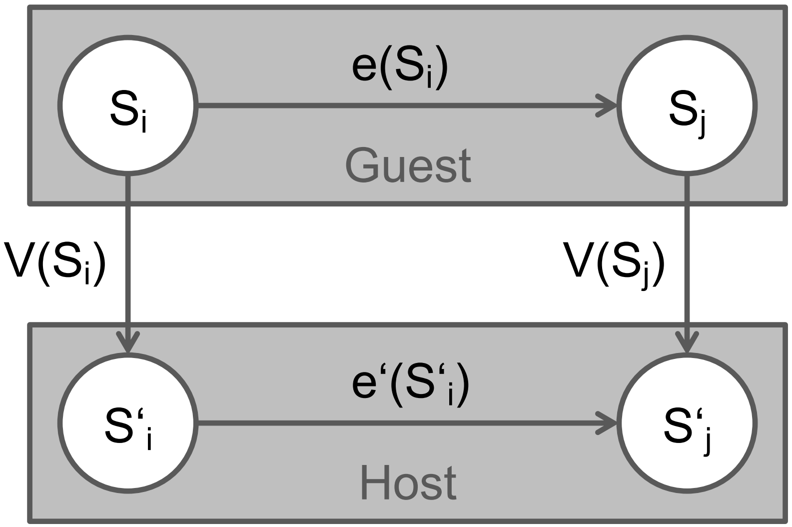
- Definition of virtualization according to NIST:

> "Virtualization is the simulation of the software and/or hardware upon which other software runs. This simulated environment is called a virtual machine (VM)."

- Virtualization can transform a real system so
  - it looks like a different virtual system
  - multiple virtual systems

- **Real system** is often referred to as **host (system)**
- **Virtual system** is often referred to as **guest (system)**

Formal Definition of Virtualization



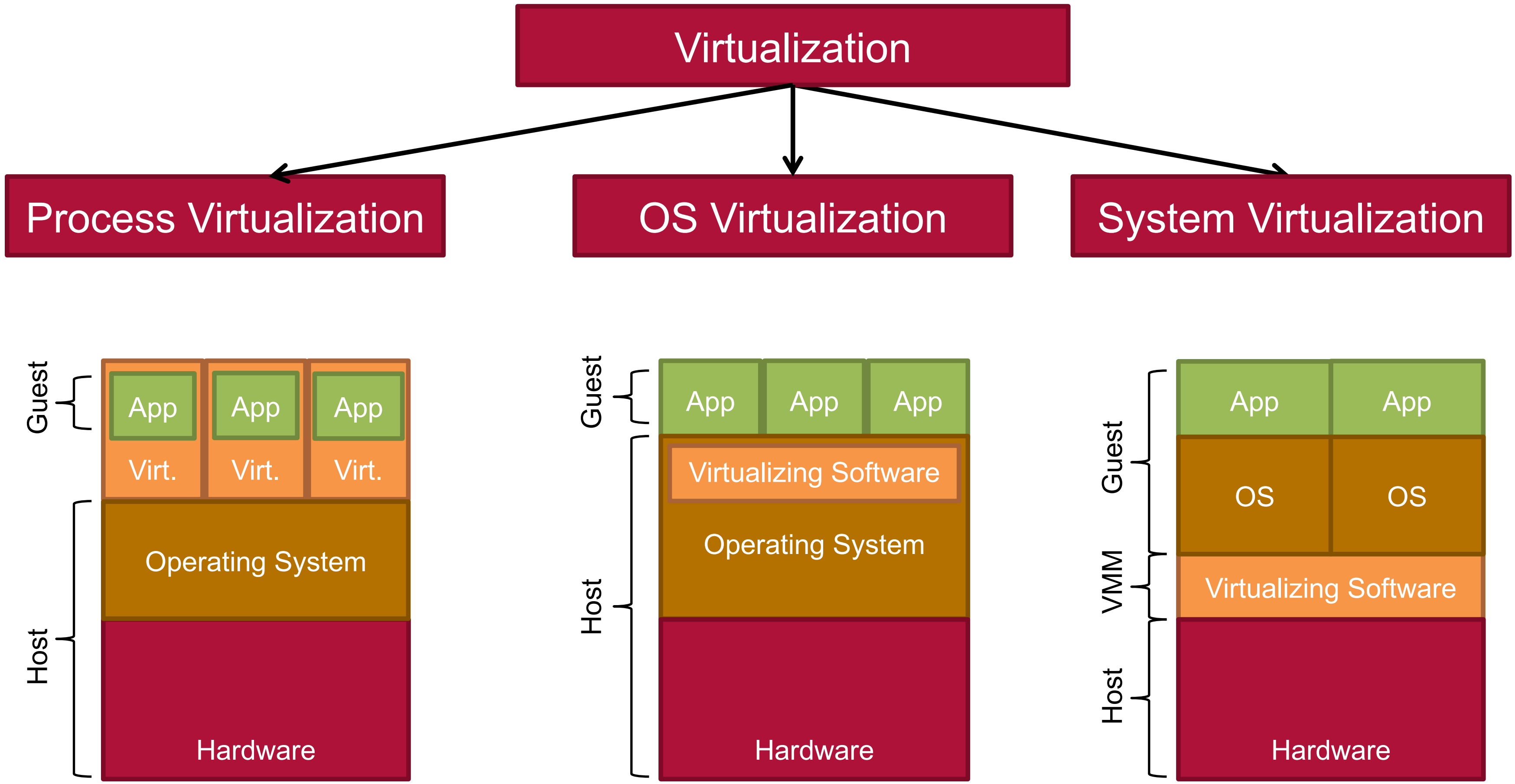
- Isomorphism V:
  - Si, Sj: States of machine
  - e: Sequence of operations

- Isomorphism V maps guest state to host state such that
  - for e that modifies the guest's state from S_i to S_j
  - there exists a corresponding sequence of operations e' that performs an equivalent modification between host's states (S'_i to S'_j)

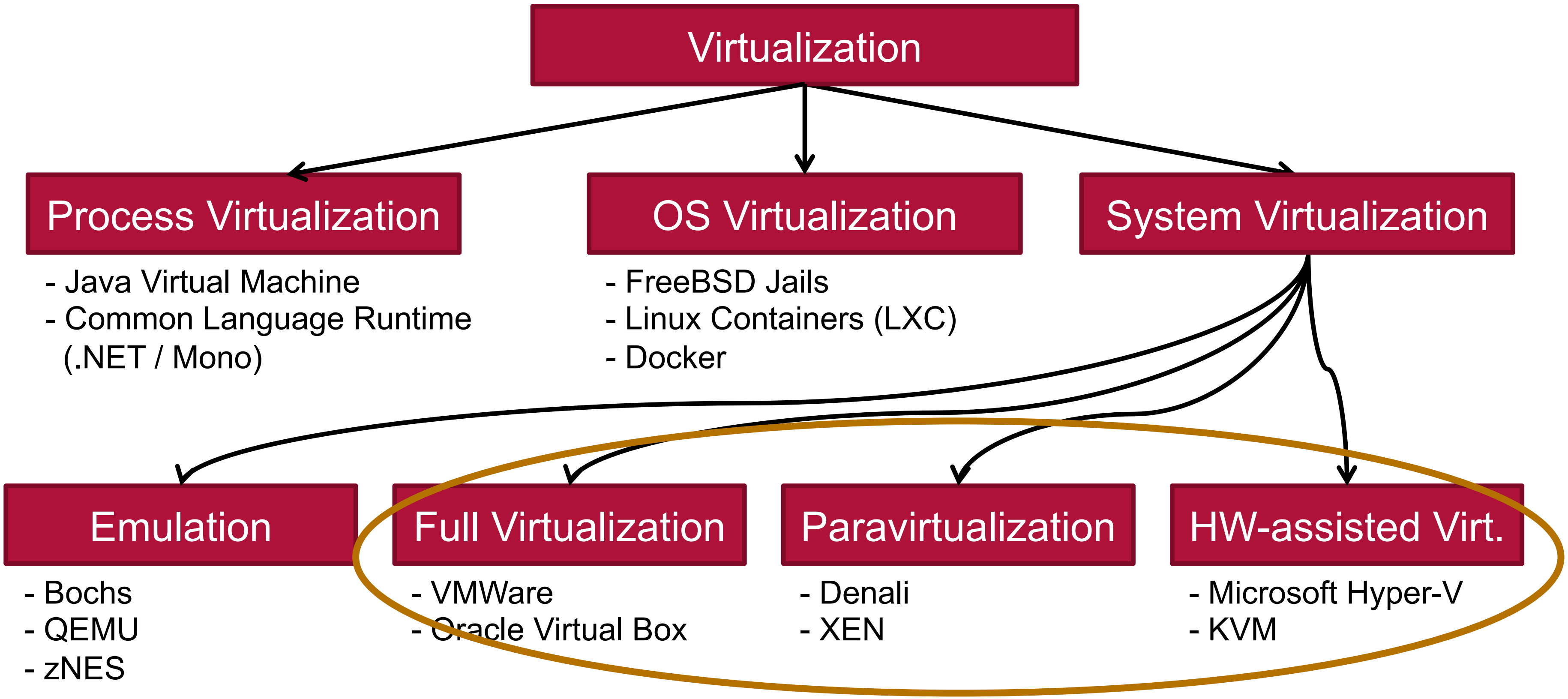
Alternative to VMs: Containers

- Lightweight “OS-level virtualization”: isolated environments for single applications within an OS
  - No virtual hardware: cannot run an application build for a different architecture or operating system kernel
  - Reduced scope: single application in pre-build environments
  - Reduced isolation: containerized applications share the same kernel, but are isolated on process-level
  - Live migration is more difficult
- Yet: Smaller container images, faster container startup, and reduced overhead

Taxonomy of Virtualization (1/2)



Taxonomy of Virtualization (2/2)



Relevant techniques for Infrastructure as a Services
(Also referred to as Hardware Virtualization)

Overview

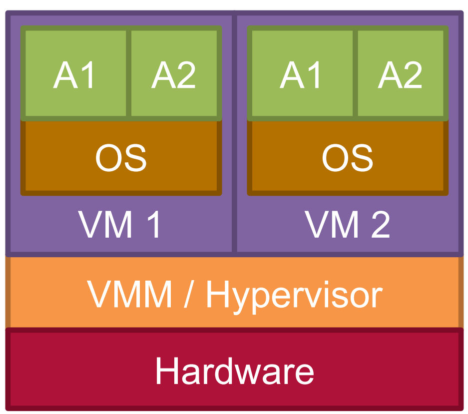
- Virtual Resources and Infrastructure-as-a-Service
- **Hardware Virtualization**
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

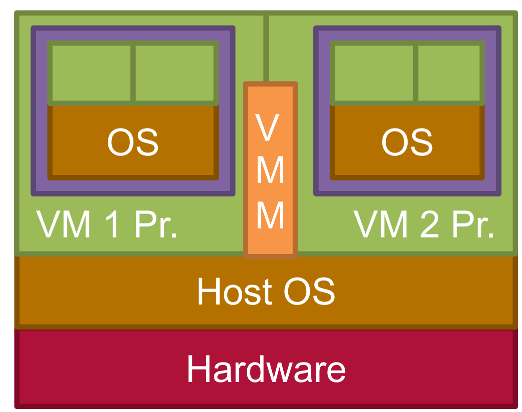
Basic Designs for Hardware Virtualization

- VMM Type I
  - Directly on hardware
  - Basic OS to run VMs
  - Pro: More efficient
  - Con: Requires special device drivers

- VMM Type II
  - VMM as host OS proc.
  - VMs run as processes, supported by VMM
  - Pro: No special drivers
  - Con: More overhead

Also known as bare-metal / hypervisor virtualization





Also known a[s] hosted virtualization

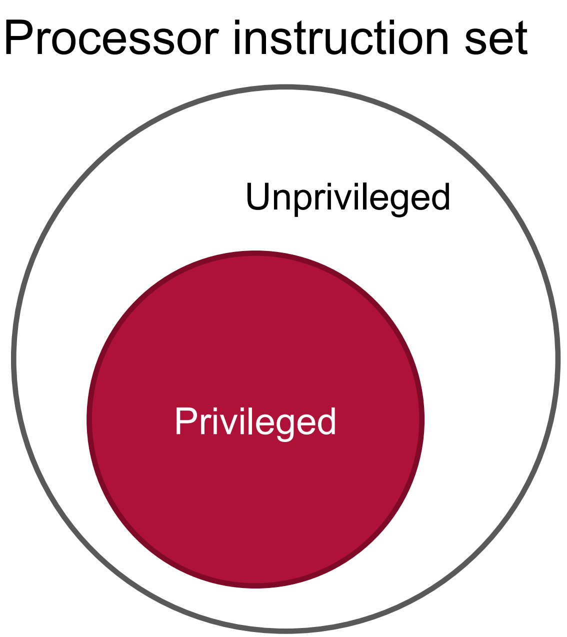
Conditions for ISA Virtualizability (VMM Type I)

- Fundamental problem for hardware virtualization:
  - VMM must have ultimate control over hardware
  - Guest operating system must be disempowered without noticing

- Four assumptions in analysis of Popek and Goldberg[3]
  1. One processor and uniformly addressable memory
  2. Two processor modes: system and user mode
  3. Subset of instruction set only available in system mode
  4. Memory addressing is relative to relocation register

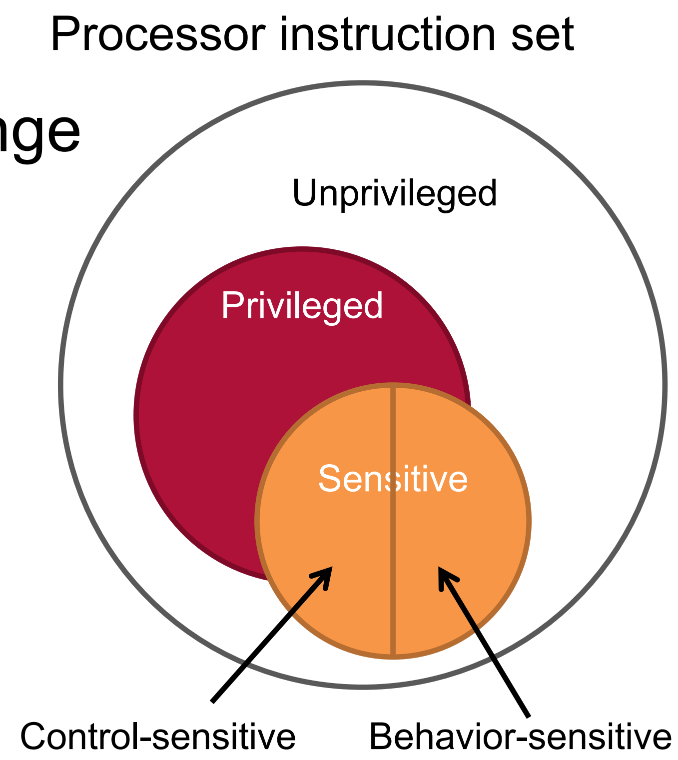
Categories of Processor Instructions (1/2)

- Privileged instructions
  - Can only be executed in system mode
  - Trap when processor is in user mode
- Examples
  - Load PSW (S/370)
    - One bit to indicate system mode
    - Malicious program could modify bit
  - Set CPU Timer (S/370)
    - Defines when user code loses CPU



Categories of Processor Instructions (2/2)

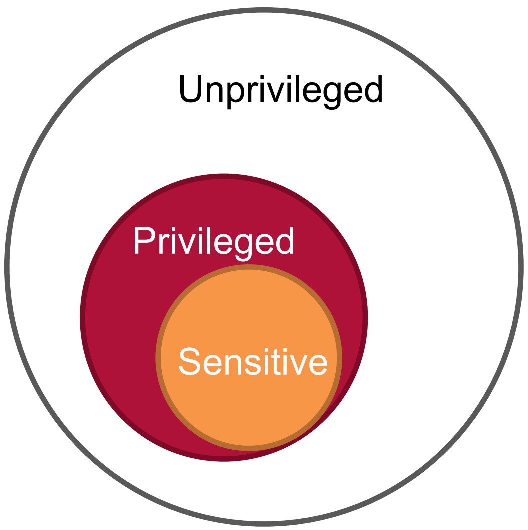
- Sensitive instructions
  - Control-sensitive instructions: Change configuration of resource
  - Behavior-sensitive instructions: Behave different depending on configuration of resource
- Examples
  - Load Real Address (S/370)
  - Pop Stack into Flags Register (IA-32)



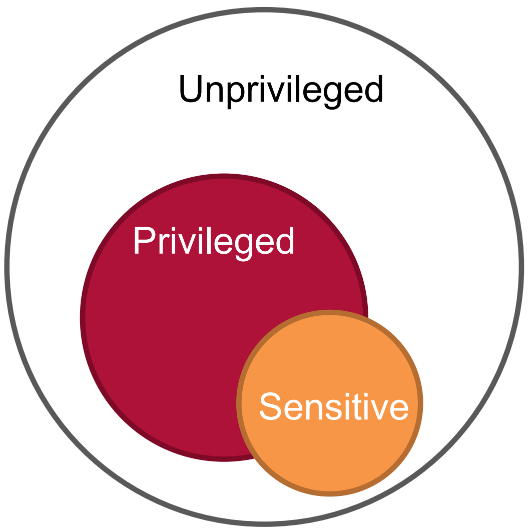
Popek and Goldberg‘s Theorem

- Basic condition for the construction of *efficient* VMMs

> “For any conventional third generation computer, a virtual machine monitor may be constructed if the set of **sensitive instructions** for that computer is a **subset** of the set of **privileged instructions**.”



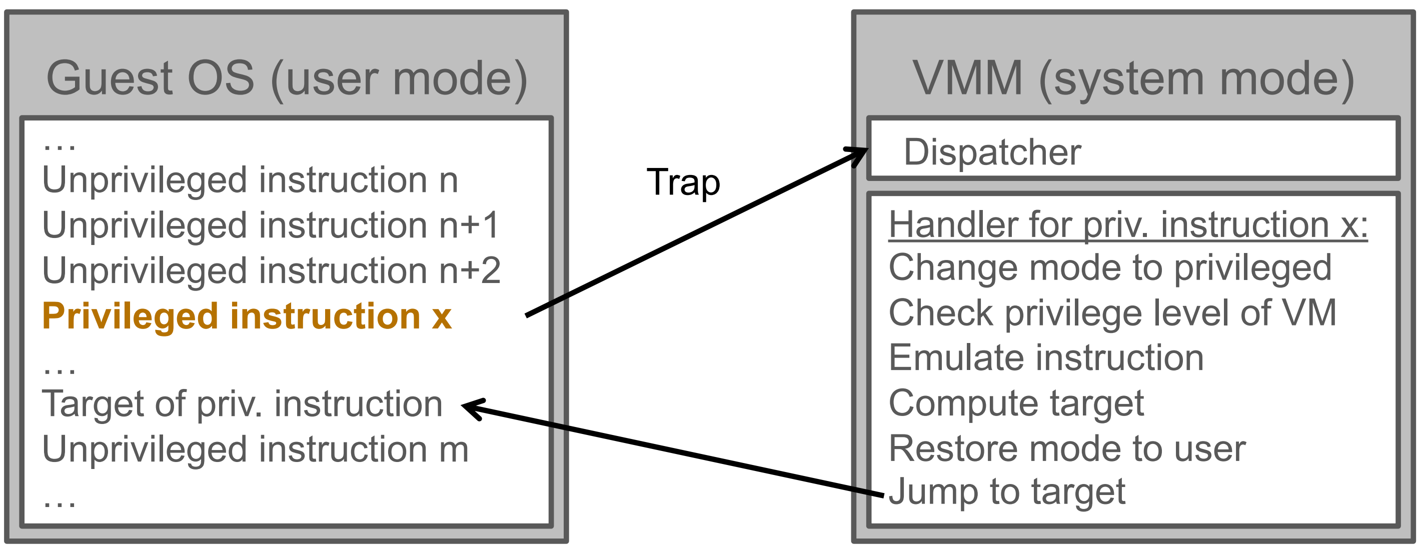
Condition satisfied



Condition unsatisfied

Implications of Popek and Goldberg‘s Theorem

- Efficient VMM: All non-sensitive instructions run natively on processor
- Trap and emulate: Guest OS calls sensitive instruction
  - Instructions traps
  - VMM emulates instruction operation



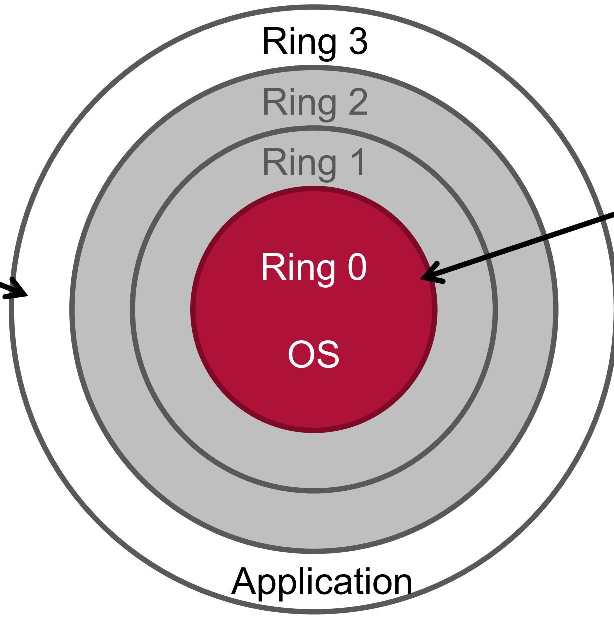
Popek and Goldberg‘s Requirements in Practice

- Which ISAs satisfy Popek and Goldberg’s requirement?
- IBM Power ✓
- Sun Sparc ✓
- Intel IA-32 ✗
  - ~17 critical instructions (= sensitive but not privileged)[4]
  - Critical instructions do not trap, but have different semantics if not executed in system mode
- Apparently, virtualization on IA-32 is possible. So, how can it be done?

Virtualization of IA-32 Architectures (1/2)

- IA-32 uses rings to manage privileges
  - Four different code privileges possible
  - Designed as generalization of two processor modes
  - To simplify portability, only ring 0 and 3 used in practice

User space
- Runs applications
- Execution of privileged instruction requires syscall



Supervisor mode
- Full control over CPU
- OS runs in this ring
- Equiv. to system mode

Virtualization of IA-32 Architectures (2/2)

1. Full Virtualization using Binary Translation

2. OS-assisted virtualization (Paravirtualization)

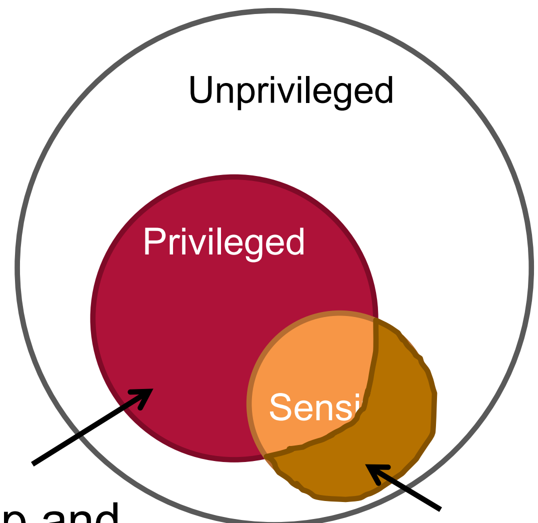
3. Hardware-assisted virtualization

Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - **Binary Translation**, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

Full Virtualization using Binary Translation

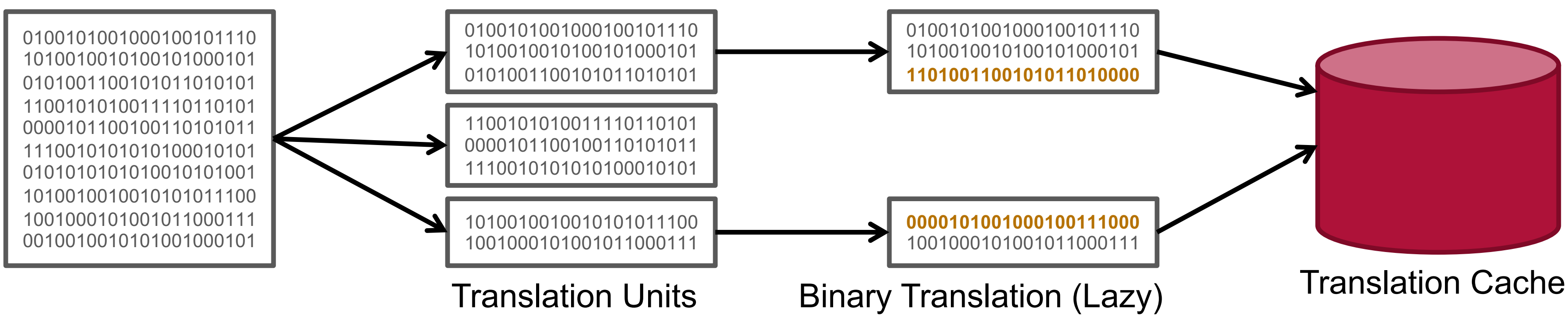
- Translating a book word for word => inefficient
- Idea: Find critical instructions and replace them
  1. Run unprivileged instructions directly on CPU
  2. Trap and emulate privileged and sensitive instructions
  3. Find critical instructions and replace with exception



- Problem: Differentiation if critical or regular depends in some cases on the parameters used (e.g. LOAD-command)
  - ➔Replacement must be done at runtime

Basic Approach for Binary Translation

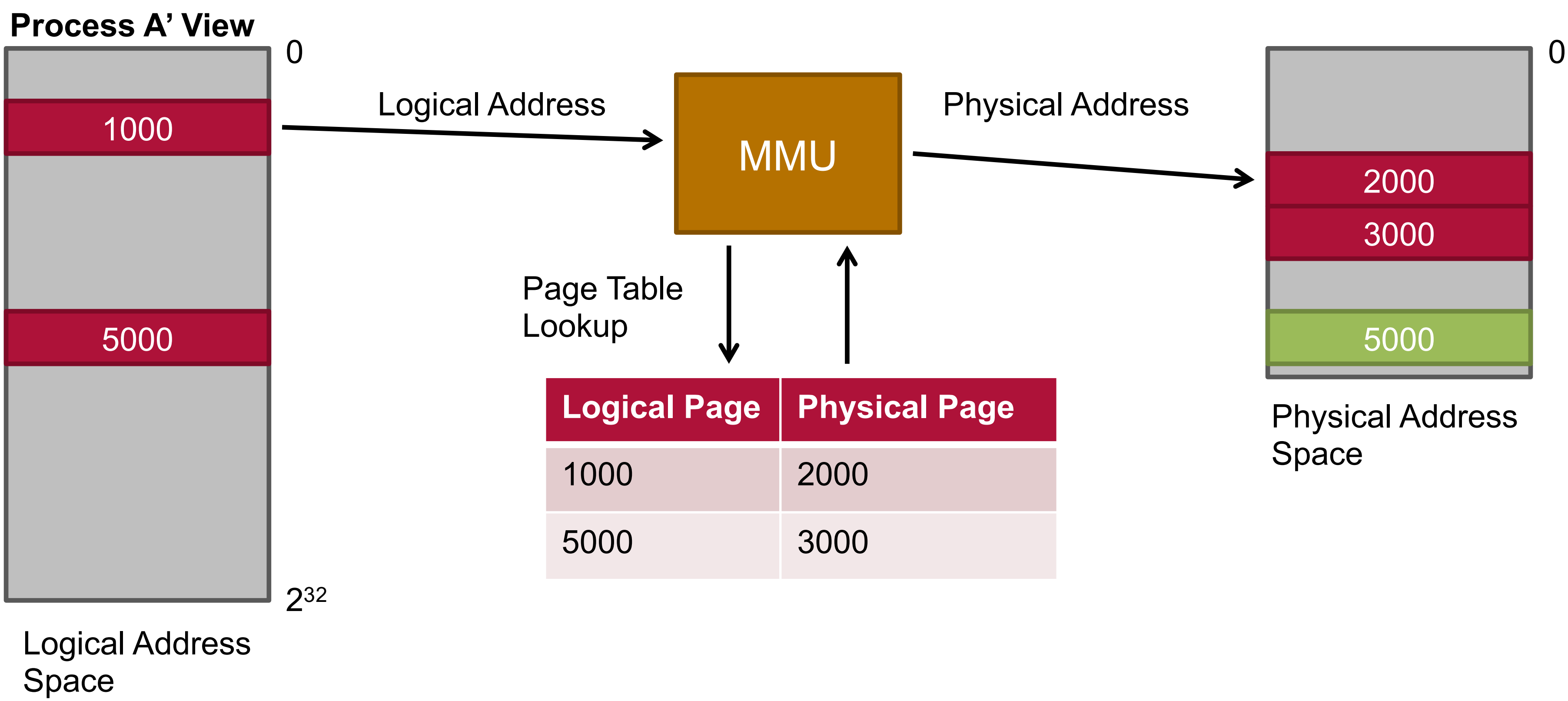
1. Separate instruction sequence in translation units
2. Check unit for critical instructions and modify code
3. Modified code is stored in translation cache



- Translation is done lazily
  - Some units may be never translated (exception handling)
  - Frequently used units benefit from translation cache

Memory Management on IA-32 (Recap)

- MMU translates logical to physical memory addresses



Logical Page	Physical Page
1000	2000
5000	3000

Memory Management on IA-32 Architectures (Recap)

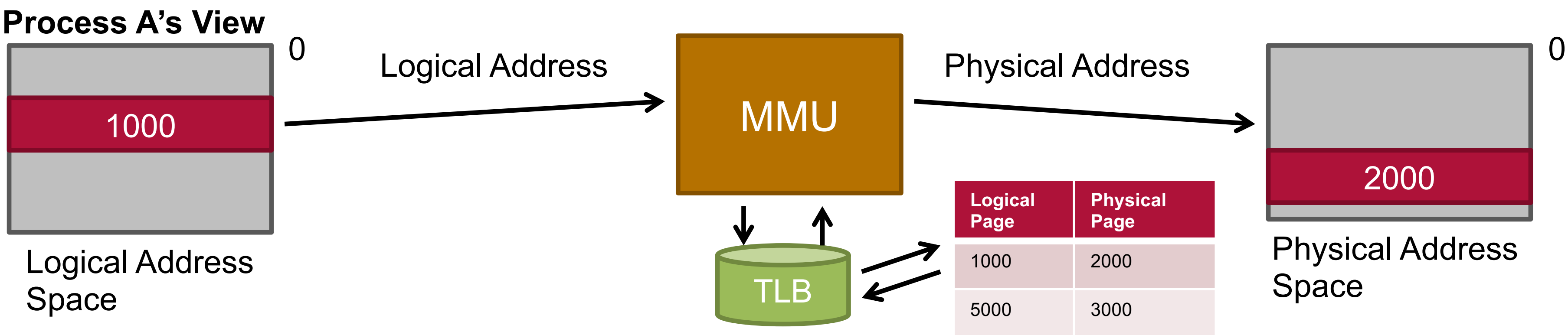
- Page tables are architected on IA-32
  - Hardware knows layout of page table
  - OS can modify the page table, lookup happens transparently

- Page tables reside in main memory themselves → Overhead of memory access essentially doubles

- Idea: Introduce special hardware-accelerated cache to remember recent address translations
  → Translation Lookaside Buffer (TLB)

Translation Lookaside Buffer (Recap)

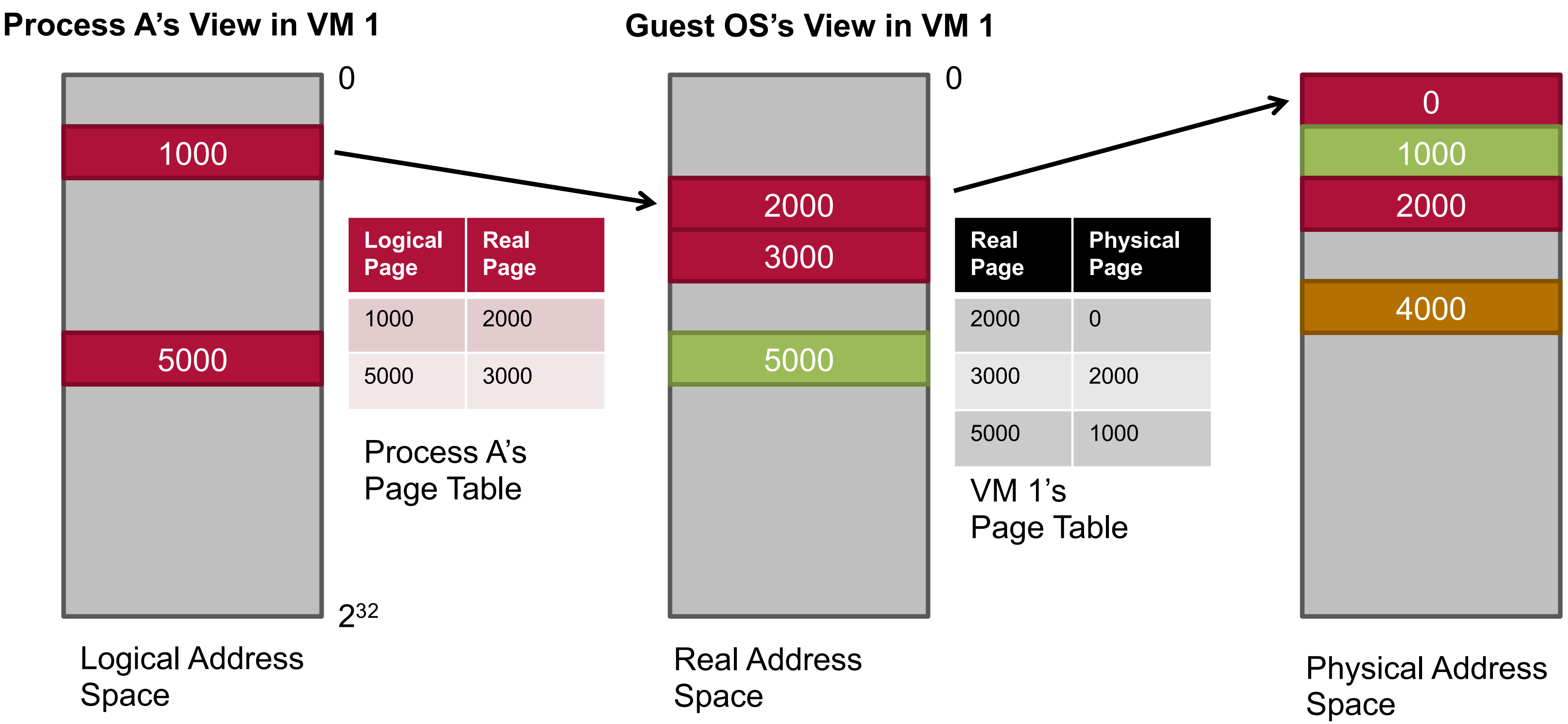
- TLB acts as cache of the MMU
  - Typically really fast (~1 cycle hit time)
  - Typically really good hit rate (> 99%)



- On IA-32, the TLB is invisible to the operating system
  - Is updated by hardware on every page table lookup
  - Must be flushed on every context switch

Memory Management and Full Virtualization (1/2)

- General idea: Add another level of indirection



Logical Page	Real Page
1000	2000
5000	3000

Process A's Page Table

Real Page	Physical Page
2000	0
3000	2000
5000	1000

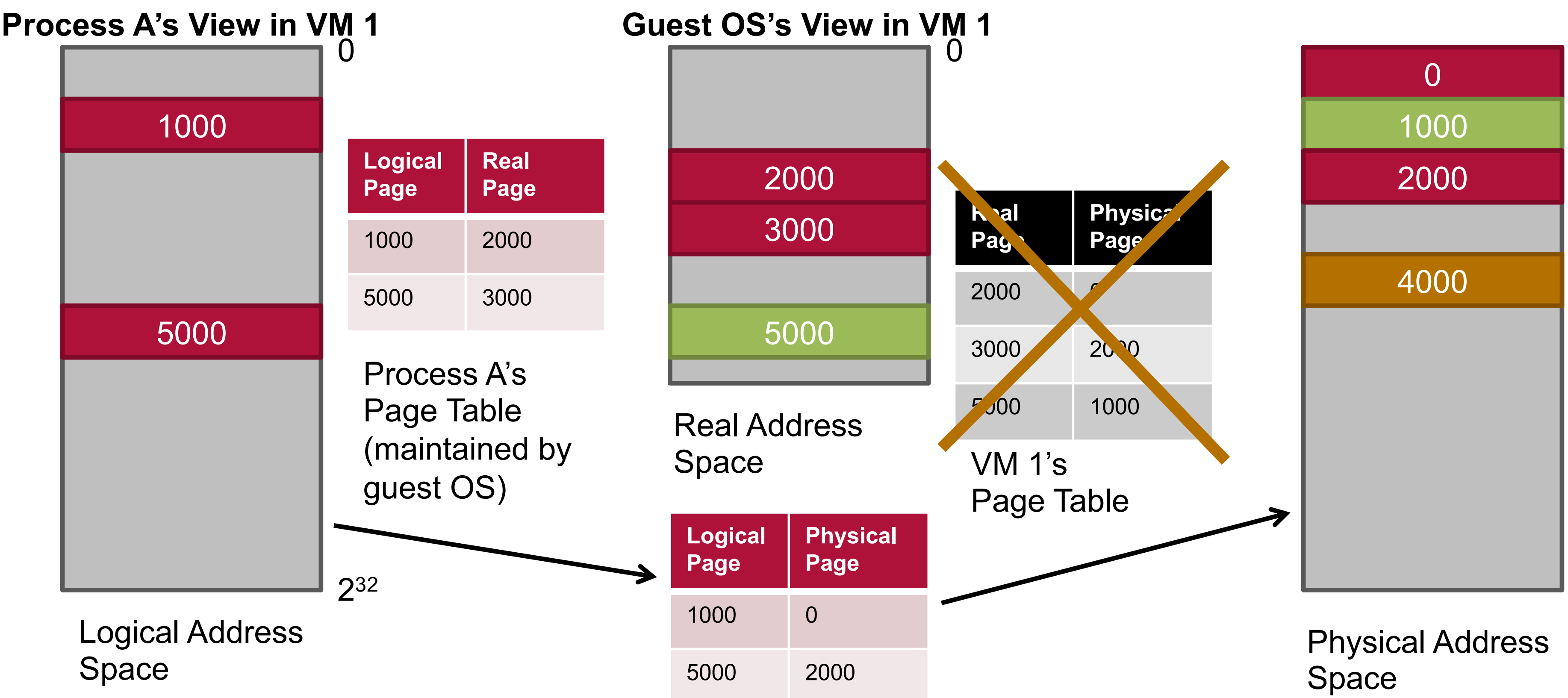
VM 1's Page Table

Memory Management and Full Virtualization (2/2)

- Problem: Additional memory access required to resolve address → significant performance decrease

- Practical implementation: *Shadow page tables*
  - Guest OSs maintain own page tables (for compatibility)
  - But modifications to guest's page table trap and entries are copied to the VMM's shadow page table
  - Shadow page table is actually used by hardware
    - Keeps TLB up-to-date
    - Works through virtualization of page table pointer

Memory Virtualization with Shadow Page Tables



Logical Page	Real Page
1000	2000
5000	3000

Process A's Page Table (maintained by guest OS)

Real Page	Physical Page
2000	0
3000	2000
5000	1000

VM 1's Page Table

Logical Page	Physical Page
1000	0
5000	2000

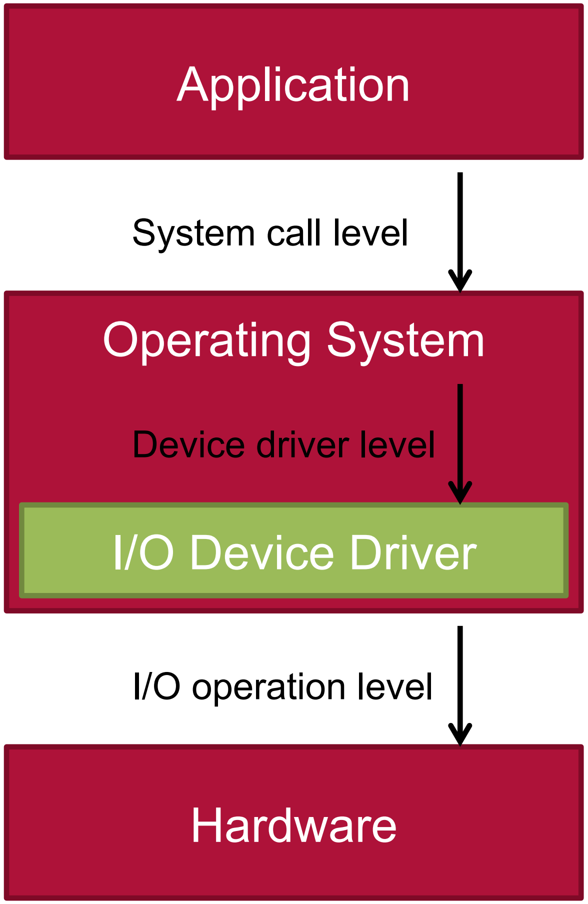
Process A's Shadow Page Table (maintained by VMM)

Full Virtualization and I/O (1/2)

- I/O devices can be categorized in five classes
  1. Dedicated devices (e.g. display, keyboard, mouse, …): Not shared among VMs on a very long time scale
  2. Partitioned devices (e.g. disks): Partitions made available to VMs as dedicated devices
  3. Shared devices (e.g. network adapters): Shared among VMs on very fine-grained time scale
  4. Spooled devices (e.g. printers): Shared among VMs but with time higher granularity
  5. Nonexistent physical devices (e.g. virtual NICs): Virtual devices without physical counterpart

Full Virtualization and I/O (2/2)

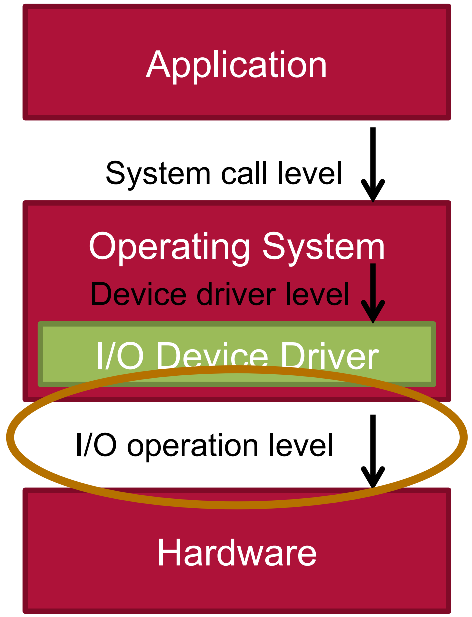
- Different levels of I/O virtualization possible
  1. At system call level
  2. At device driver level
  3. At I/O operation level



I/O Virtualization at I/O Operation Level

- IA-32 provides special privileged instructions to talk to I/O devices
- Pro: All I/O instructions trap
  - Easy for VMM to intercept them
- Con: Instructions are very low-level
  - Example: Read/write byte to I/O port
  - Higher-level I/O operation consist of several of those instructions
  - Hard for VMM to determine concrete I/O operation, “reverse engineering” required

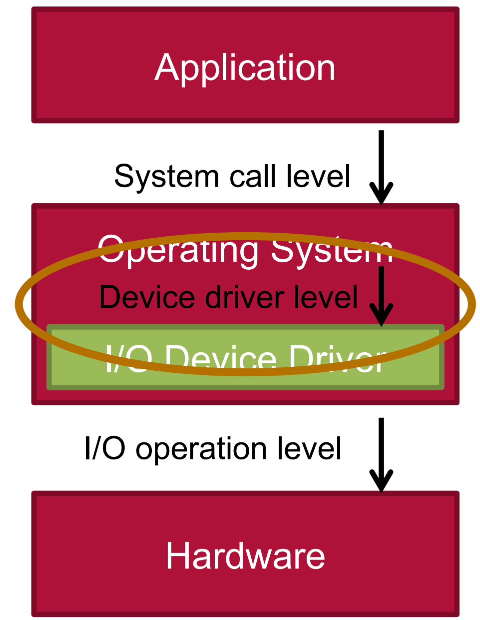
→ Difficult for arbitrary devices



I/O Virtualization at Device Driver Level

- VMM intercepts calls to virt. device driver
  - Converts virtual device information to corresponding physical device
  - Redirects calls to physical device's driver program
- Pro: Natural point for virtualization
  - No "reverse engineering" required
- Con: Requires knowledge of guest's device driver interface

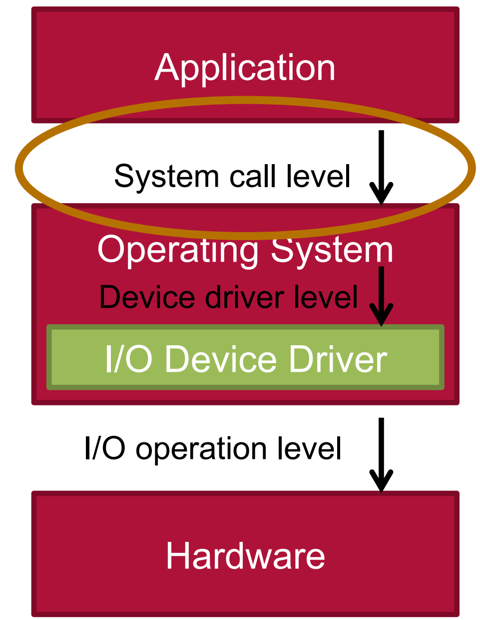
→ Not generally applicable, OK for many practical purposes (e.g. Windows, Linux)



I/O Virtualization at System Call Level

- VMM intercepts system call at OS interface
- Pro: VMM handles the entire I/O operation
- Con: VMM must shadow OS routines available to the user
  - Virtualization must be transparent to the guest
  - Requires broad knowledge of the guest OS's internals

→ Very complicated, hardly seen in practice



Summary Full Virtualization with Binary Translation

- Requires modified guest OS? **NO**
- Requires hardware support? **NO**
- Performance
  - Good approach for compute-intensive applications
    - Unprivileged instructions run directly on CPU
  - Degraded performance for data-intensive applications
    - I/O requires syscalls → privileged instructions
    - “trap and emulate” often requires context switches
    - Context switches lead to complete flush of TLB

VMWare Adaptive Binary Translation

- Modern CPUs are deeply pipelined
- Trapping privileged instructions can be too expensive
- Example: **`rdtsc`** (read time-stamp counter), Pentium 4[6]
  - Trap-and-emulate: 2030 cycles
  - Callout-and-emulate: 1254 cycles (*Callout method replaces traps with stored emulation functions*)
  - In-Translation Cache (In-TC) emulation: 216 cycles
- VMWare feature: Adaptive Binary Translation
  - Monitor frequency and costs of traps
  - Adaptively switch between different execution strategies at runtime

Limitations of Adaptive Binary Translation

- Adaptive Binary Translation improves speed over simple “trap and emulate” approach
  - Replaces most traps with faster callouts
  - Some instructions rewritten to run w/o VMM intervention

- However, some limitations remain
  - System calls always require VMM intervention
    - Native system call is ~200 cycles, VMM adds ~2000 cycles
  - Many traps due to shadow table page mechanism
  - Instructions for I/O usually trap, context switch for VMM type II required

Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, **OS-Assisted Virtualization**, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

OS-Assisted Virtualization (Paravirtualization)

- Idea of OS-assisted virtualization
  - Make guest OS aware that it is running in a VM
  - Modify the guest source code so that it avoids assistance of the VMM as far as possible
- Denali project also coined term paravirtualization[7]
- Requirements for pure OS-assisted approach
  - Source code of guest operating system is available
  - Modified guest OS maintains application binary interface

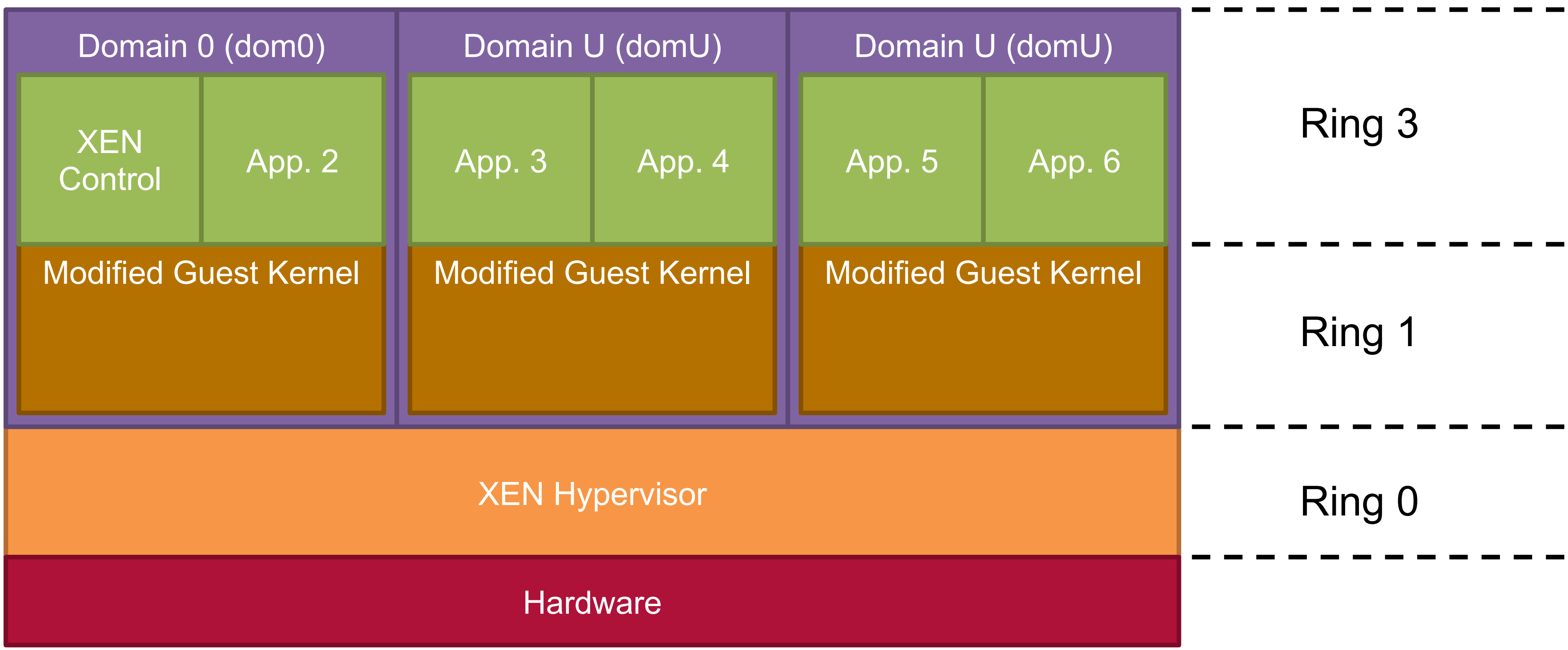
OS-Assisted Virtualization in Practice

- Today, most virtualization platforms use OS-assisted virtualization for their device drivers

- Classic representative for paravirtualization: XEN[8]
  - Type I Hypervisor
  - Available as open-source software
  - Originally developed at University of Cambridge, UK, in collaboration with Microsoft Research Cambridge
  - Presented at SOSP'03

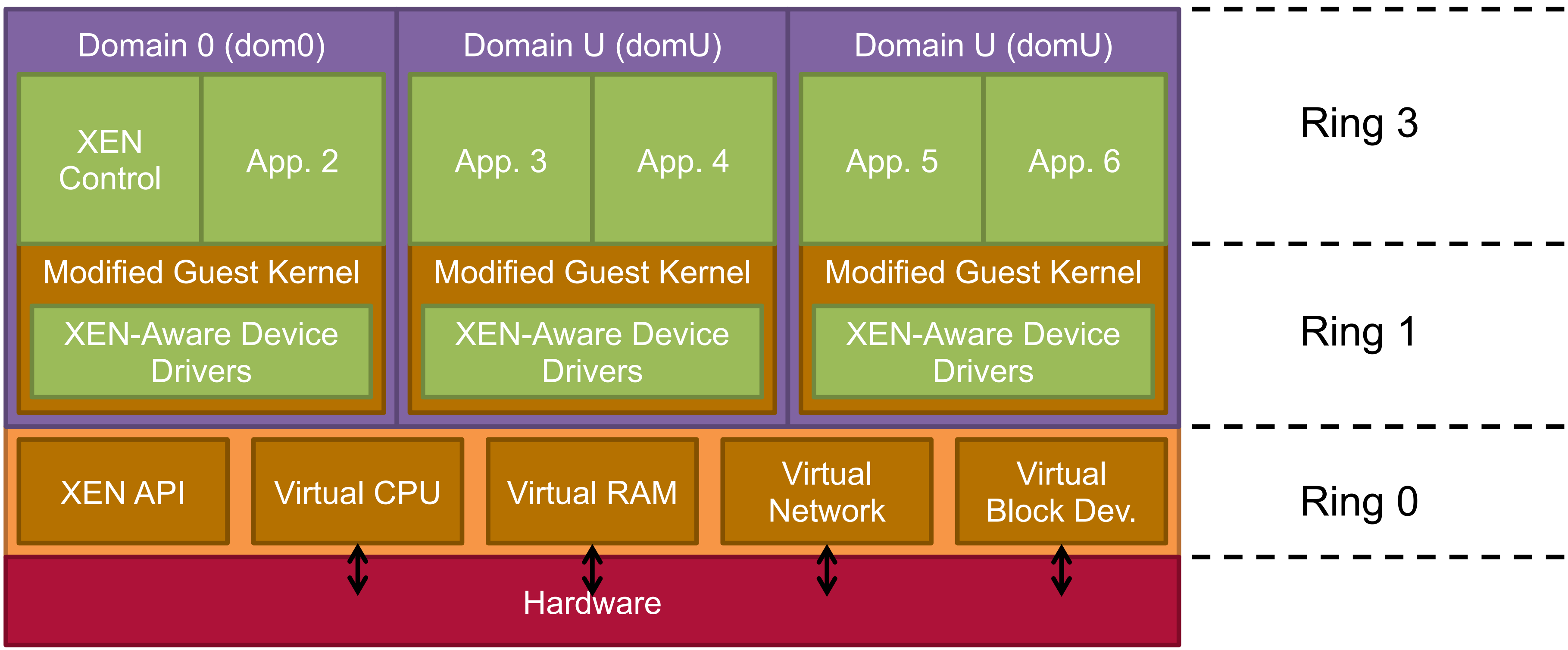
XEN Architecture and Domains

- Domain 0: Privileged guest for control/management
- Domain U: Guest with XEN-enabled OS



Interfaces and Driver Concept (XEN 1.0)

- Communication between XEN and domains:
  - Hypercall: Synchronous call from to domain to XEN
  - Event: Asynchronous notification from XEN to domain



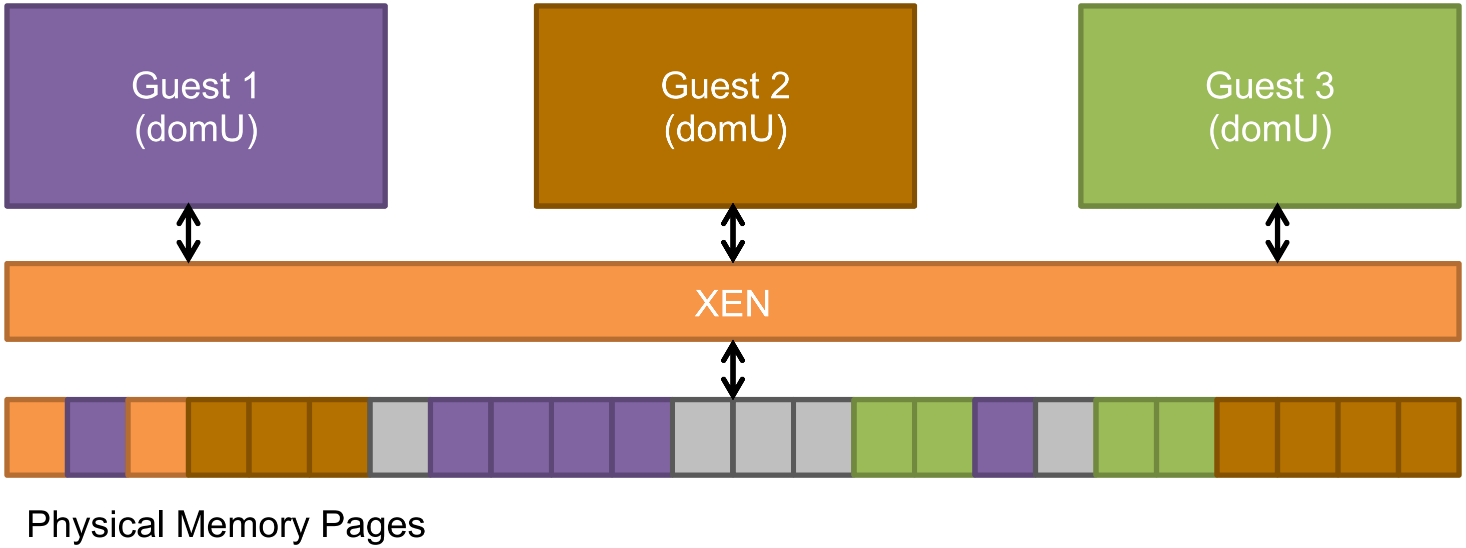
How Does XEN Tackle Full Virtualization Problems?

- Critical instructions do not trap on IA-32
  - Guest OS is aware of virtualization → Critical instructions can be avoided

- Frequent intervention of the hypervisor required
  - Most common reason for required intervention
    - Page table updates
    - System call
  - XEN cannot get rid of those interventions either
    - Guest domains run in Ring 1
    - However, XEN plays some tricks to decrease frequency

XEN and Physical Memory

- Domain gets fraction of phys. memory at creation time
  - Static partitioning among domains
  - No guarantee partition is contiguous
  - Hypervisor knows which domain „owns“ which pages

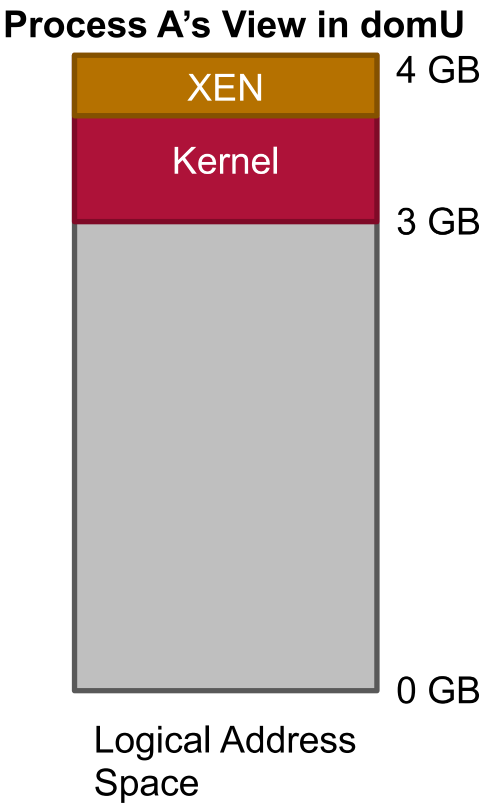


XEN and Memory Virtualization (1/3)

- XEN lets guests maintain their own page tables
  - Guest page tables are visible to the MMU
    - Prerequisite: Guest OS knows its fraction of phys. memory
  - → No need for hypervisor intervention on read requests
  - → XEN must only validate write requests to ensure isolation

- Procedure for writes
  1. Guest requests page table update via hypercall
  2. XEN checks if mapping address belongs to domain
  3. If ok, allows update to page table

XEN and Memory Virtualization (2/3)



- XEN exists in top 64 MB of every logical address space
  - Kernel can access hypervisor without context switch
  - → No TLB flush
- Address region not used by any common x86 ABI → does not break compatibility with applications

XEN and Memory Virtualization (3/3)

- General XEN trick: command batching
  - Decreases number of required hypervisor entries/exits
- Example:

```
void *ptr = malloc(4 * 1024 * 1024); // 1024 4K pages
```

  - Translates to 1024 necessary page table updates
  - Full virtualization: 1024 entries and exits to the hypervisor
  - XEN: Requests collected, submitted with one hypercall
    - Requests are not immediately processed
    - XEN ensures correctness despite delay

    → Only one entry/exit to hypervisor required

XEN and System Calls

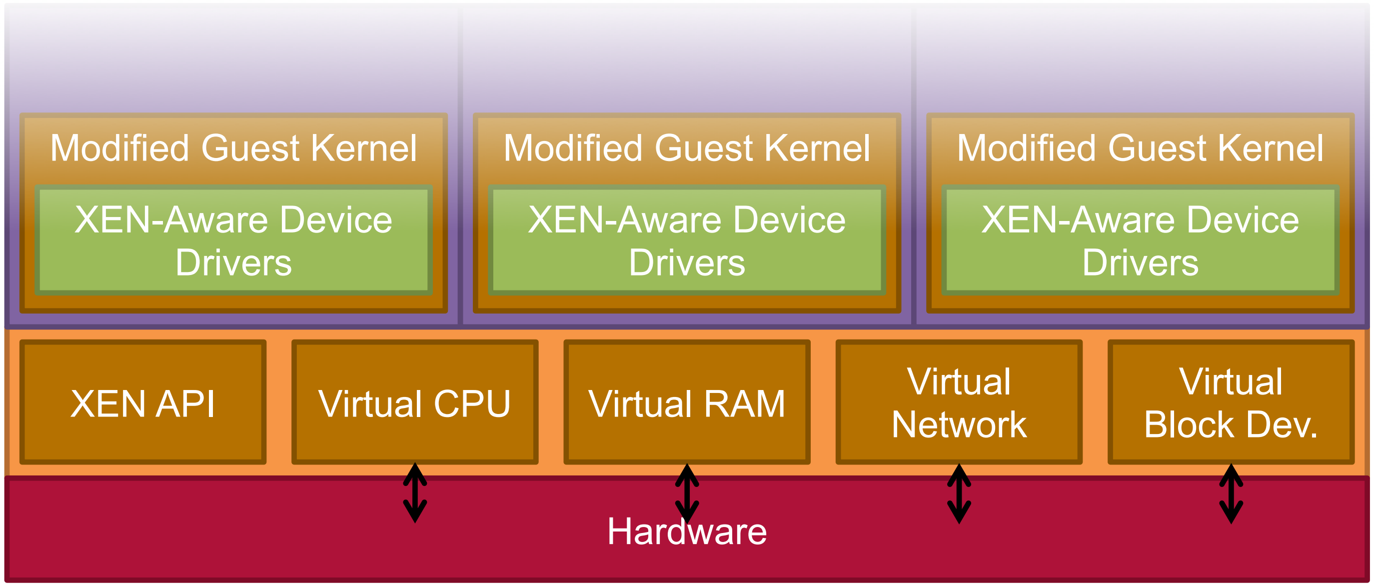
- Major source for VMM intervention with full virtualization

- Syscalls implemented through software exceptions
  - Upon exception, hardware consults hardware exception table to find code to handle exception
  - XEN allows guest to install “fast” exception handler in the hardware exception table (automatic forwarding!)
  - XEN validates the handlers before installing them

  → Application can call into guest OS without indirection through VMM (Ring 0) on each call

XEN and I/O Virtualization

- XEN presents “idealized” hardware abstraction
  - XEN itself contains specific device drivers (XEN 1.0)
  - Domains must only implement lightweight frontend driver
    - Hypervisor and domains cooperate
    - Communication through hypercalls and events



Cost of Porting an OS to XEN

OS Subsection	# Lines Linux	# Lines XP
Architecture-independent	78	1299
Virtual network driver	484	-
Virtual block-device driver	1070	-
Xen-specific (non-driver)	1363	3321
Total	**2995**	**4620**
% of tot. x86 code base	**1.36 %**	**0.04 %**

- No virtual I/O drivers available for XP at that time
- Cost of porting device drivers not considered here

Summary OS-Assisted Virtualization

- Requires modified guest OS? **YES**
- Requires hardware support? **NO**
- Pros:
  - Better performance through cooperation between hypervisor and guest OS
- Cons:
  - Limited compatibility, not generally applicable
  - Increased management overhead for data center operator, different version of OS must be maintained

OS-Assisted Virtualization and Cloud Environment

- OS-Assisted Virtualization is de-facto standard for I/O virtualization at the moment
- All major virtualization solutions provide special drivers
  - VMWare, XEN, KVM (virtio project)

- XEN currently enjoys big support by the community
  - Commercial: Amazon EC2, Rackspace, …
  - Academia: Eucalyptus, OpenStack, …
  - OS Distributors: openSuSE, Debian, Ubuntu, NetBSD, …

- HW-assisted virt. plays increasingly important role

Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, **Hardware-Assisted Virtualization**
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

Hardware-Assisted Virtualization

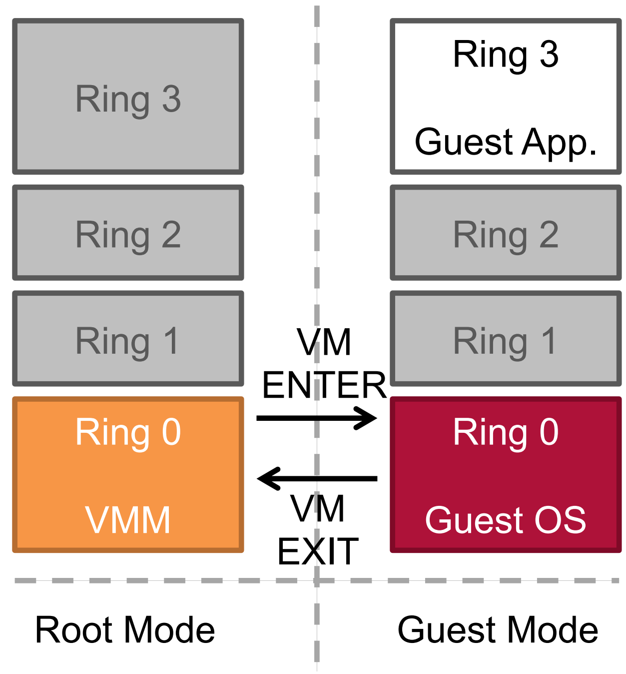
- Most virtualization difficulties caused by IA-32 design
  - Sensitive instructions do not always trap in rings > 0
  - Guests can observe they are not running in ring 0
- Success of VMWare has demonstrated demand for virtualization
- Idea: Extend IA-32 architecture to circumvent virtualization obstacles on the hardware level
  - Independent developments by Intel and AMD
  - Yet, developments do share same basic ideas

Incremental Hardware Support for Virtualization

	intel	AMD
2005/2006 Extension for CPU virtualization	Intel VT-x (Vanderpool)	AMD SVM, AMD Virtualization, AMD-V (Pacifica)
2007/2008 Extension for MMU virtualization	Extended Page Tables (EPT)	Rapid Virtualization Indexing, Nested Page Tables (NPT)
2009/2010 Extension for I/O virtualization	Intel VT-d	AMD IOMMU

First Generation Support for Virtualization (VT-x, AMD-V)

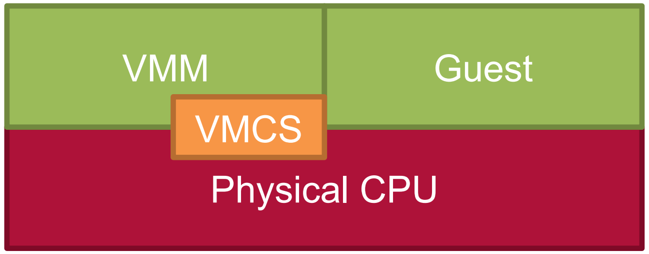
- Two new CPU modes: root mode vs. guest mode
  - VMM runs in root mode
  - Guest OS in guest mode



- VMM and guest run as “co-routines”
  - VMM can give CPU to guest OS (VM ENTER)
  - VMM can define conditions when to regain CPU (VM EXIT)

VMM Control Structures[11]

- VMM controls guest through HW-defined structure
  - Intel: VMCS (virtual machine control structure)
  - AMD: VMCB (virtual machine control block)



- VMCS/VMCB contains
  - Guest state
  - Control bits defining criteria for VM EXIT
    - Exit on IN, OUT, CPUID, …
    - Exit on write to page table register, …
    - Exit on page fault, interrupt, …
  - VMM uses control bits to “confine” and observe guest

Benefits of 1st Generation Hardware Extension

- VMM controls guest through VMCS in fine-grained way
  - Guest OS continues to run in Ring 0
  - →Not all privileged instructions necessarily trap
  - →VMM has flexibility to decide which instructions guest is allowed to handle itself

- HW extension eliminates many reasons for VMM intervention compared to classic HW environment, such as syscalls

Limitations of 1st Generation HW Extension

- VMM intervention is still required on several occasions
  - Page table updates (read/write)
  - Context switches
  - I/O
  - Interrupts

Benefits/Limitations Illustrated (1/3)

```
uint64_t i, s = 0;

for (i = 0; i < 1000000000; i++) {
  s = s + i;
}

printf("s= %ld, c = %ld\n",s, i * (i -1) / 2);
```

Source: [11]

	Full Virtualization with BT	HW-Assisted Virtualization (1st Gen)
Performance compared to native	95%	95%
Explanation	No VMM intervention, almost all code runs natively	

Only qualitative comparison, don‘t take numbers too serious

Benefits/Limitations Illustrated (2/3)

```
uint64_t i;

for (i = 0; i < 1000000000; i++) {
  getppid();
}
```

Source: [11]

	Full Virtualization with BT	HW-Assisted Virtualization (1st Gen)
Performance compared to native	25%	95%
Explanation	System call executed in ring ≠ 0, VMM intervention required	System call executed in Ring 0, code continues to run natively

Only qualitative comparison, don‘t take numbers too serious

Benefits/Limitations Illustrated (3/3)

```
uint64_t i;

for (i = 0; i < 1000000000; i++) {
  if(fork() == 0) return;
}
```

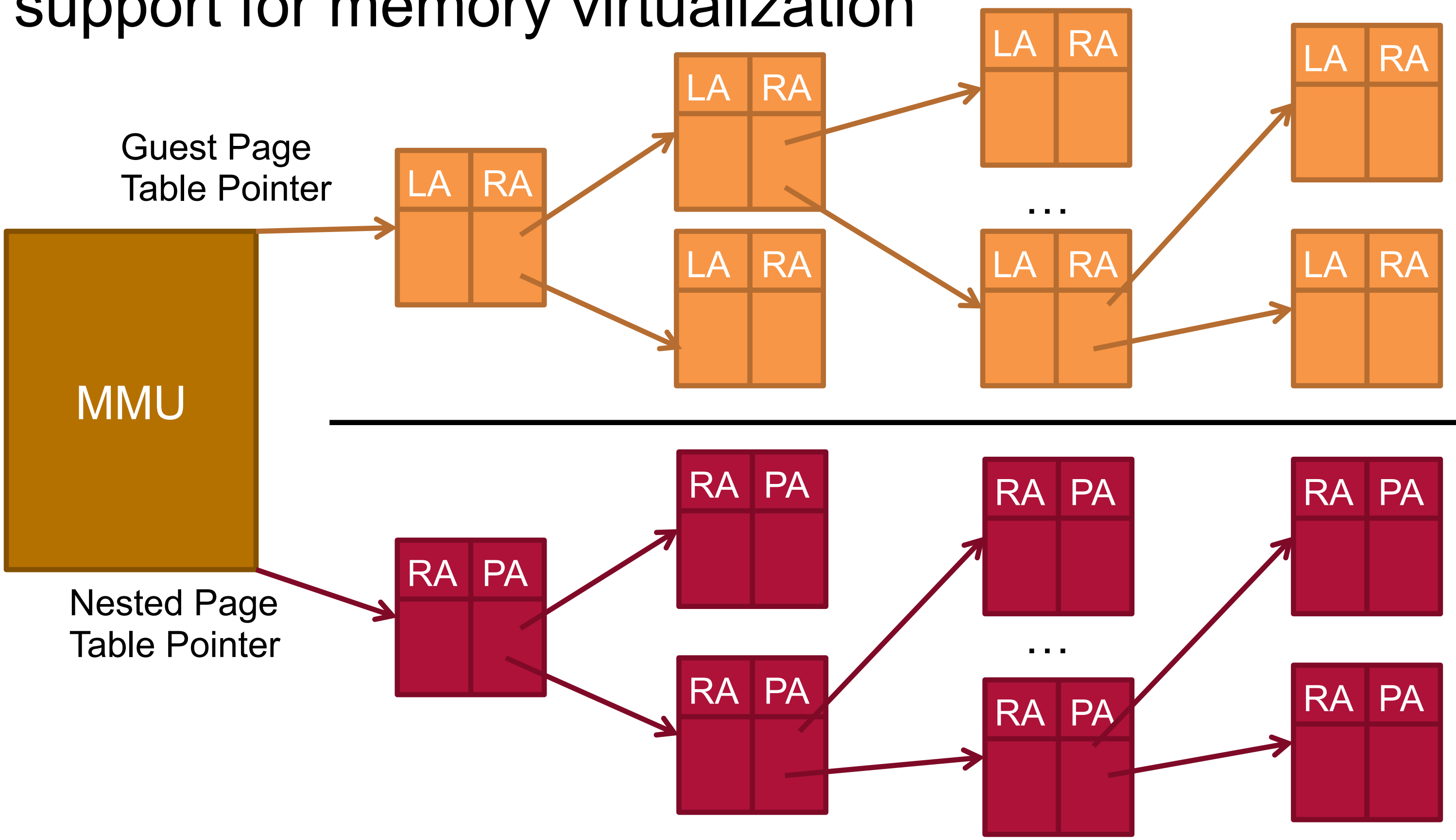
Source: [11]

	Full Virtualization with BT	HW-Assisted Virtualization (1st Gen)
Performance compared to native	15%	5%
Explanation	Frequent creation of page tables, VMM intervention required due to shadow page tables	

Only qualitative comparison, don‘t take numbers too serious

Second Generation Support for Virtualization

- Extended Page Tables/Nested Page Tables introduce HW support for memory virtualization



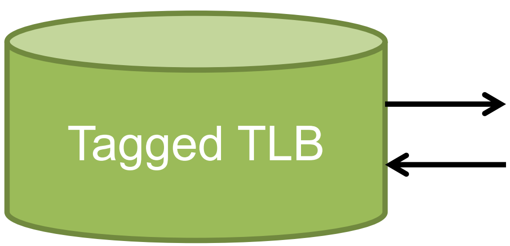
LA: Logical Address, RA: Real Address, PA: Physical Address

Tagged Translation Lookaside Buffer

- Translation lookaside buffer continues to cache LA → PA address translation

- Both Intel and AMD introduced tagged TLBs
  - Every TLB entry associated with address space tag

  → Only some entries are invalid on context switch



Address Space	Logical Page	Physical Page
A	1000	2000
A	5000	3000
B	1000	4000

Analysis of EPT/NPT Hardware Extension[11]

- MMU composes LA → RA and RA → PA mapping at TLB fill time
- Benefits
  - Significantly less VMM intervention required
  - No shadow page table memory overhead
  - Better scalability on multi-core CPUs
- Costs
  - High cost for TLB misses: $O(n^2)$, n = page table depth

Limitations of 2nd Gen. HW-Support Illustrated

```
#define S (8192 * 4096)
volatile char large[S];

for (unsigned i = 0; i < 10 * S; i++) {
  large[(4096 * i + i) % S] = 1 + large[i% S];
}
```

Source: [11]

	Full Virtualization with BT	HW-Assisted Virtualization (2nd Gen)
Performance compared to native	85%	40%
Explanation	Code violates locality assumption, lots of TLB misses, O(n) lookup cost	

Only qualitative comparison, don‘t take numbers too serious

Third Generation Support for Virtualization

- Third generation support for virtualization focuses on I/O

- Paravirtualization already decreased CPU overhead for I/O and increased data throughput
  - Cooperation between virtualized device driver and VMM
  - Idealized interface reduced number of VMM interventions
  - However, overhead still too high for high-performance apps

- Goal of hardware support:
  - High-performance data transfer between device and guest
  - Isolation between guests

Design Focus of HW-Support: Direct Assignment

- Direct assignment: Guest VM owns a physical device
  - No sharing of device between several VMs
  - Guest VMs run the unmodified device drivers
  - Goal: Efficient I/O without VMM intervention
  - Challenge: VMM must still ensure correctness & isolation

Full Virtualization
- VMM emulates device
- Intervention on every IN or OUT instruction
- Physical device can be shared across VMs

OS-Assisted Virtualization
- VMM provides idealized HW interface
- Guest VM implements driver to idealized interface
- Physical device can be shared across VMs

HW-Assisted Virtualization
- Guest runs native I/O driver, “owns” physical device
- Can communicate with device as in native case
- VMM only ensures correctness & isolation

Summary HW-Assisted Virtualization

- Requires modified guest OS? **NO**
- Requires hardware support? **YES**
- Pros:
  - Improved performance even for unmodified guest OSs
  - Good adaption of 1st generation HW-support by VMMs
  - 2nd generation VMM support increasingly deployed
- Cons:
  - Reduced flexibility due to hardware constraints (especially for 3rd generation HW support)

Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - **Virtual Machine Migration**
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

Virtual Machine Migration

- Migration: Move VM from one physical host to another
- Motivation:
  - Fault mgmt.: Host reports HW errors, must be shut down
  - Maintenance: Update of BIOS, hypervisor, …
  - Load balancing: Move workload to another physical host

- Desired property: Live migration (i.e., without downtime)
  - No shutdown of the virtual machine
  - No disruption of the service
  - Minimal impact for the user

→ Minimize *downtime* and *total migration time*

Concerns for Live Migration

- Memory migration
  - Ensure consistency between the memory state of source and destination VM
- Local resources
  - Network resources
    - Maintain all open network connections
    - Do not rely on forwarding of the source host
  - Storage resources
    - Storage must be accessible both at the source and destination VM

Strategies for Memory Migration

1. Push
   - Source VM continues running, sends pages to destination
   - Memory must potentially be sent multiple times

   → Minimum downtime, potentially long migration time
2. Stop-and-copy
   - Source VM stopped, pages copied to destination VM
   - Destination VM is started after having received all pages

   → Short overall migration time, long downtime
3. Pull
   - Execute new VM, pull accessed pages from source

   → Performance depends on number of page faults

Strategy for Memory Migration in XEN[15]

- XEN pursues *pre-copy* strategy for memory migration
  - Usage of a push and a stop-and-copy phase
  - Balances short downtime with short total migration time

- Iterative approach: Multiple rounds of push phase, then short stop-and-copy phase in the end

- Similar approach in VMWare vMotion: vMotion also capable of slowing down source VM in case memory changes too fast for network transfer

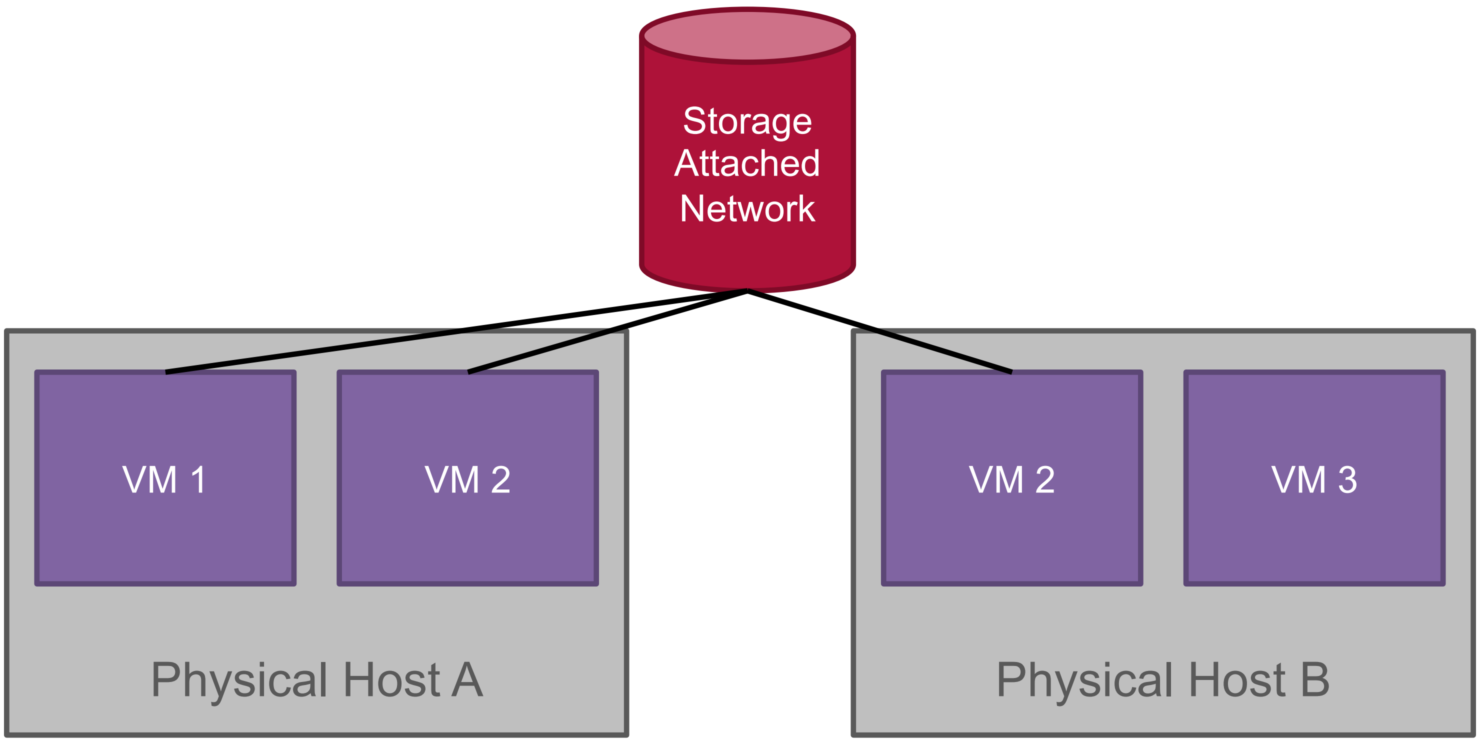
Strategy for Network Migration in XEN[15]

- Assumption: Source and destination VM are on same IP subnet (no IP-level routers involved)

- Approach for migration:
  - Destination VM will have new MAC but old IP address
  - After memory transfer, source host sends unsolicited ARP reply
    - Broadcast message to all hosts on the same network
    - Hosts will remove IP ←→ MAC mapping from caches
  - Upon new ARP request, destination VM will return new MAC address

Strategy for Storage Migration in XEN[15]

- XEN assumes VMs to reside on storage network
  - Migration by rerouting network traffic
  - Similar to network migration



XEN Migration Timeline[15]

VM running normally on Host A

Stage 0: Pre-Migration
- Active VM on Host A
- Select target host B

Stage 1: Reservation
- Initialize a container on the target host

Overhead due to copying

Stage 2: Iterative Pre-copy
- Copy dirty pages in rounds

Downtime (VM out of service)

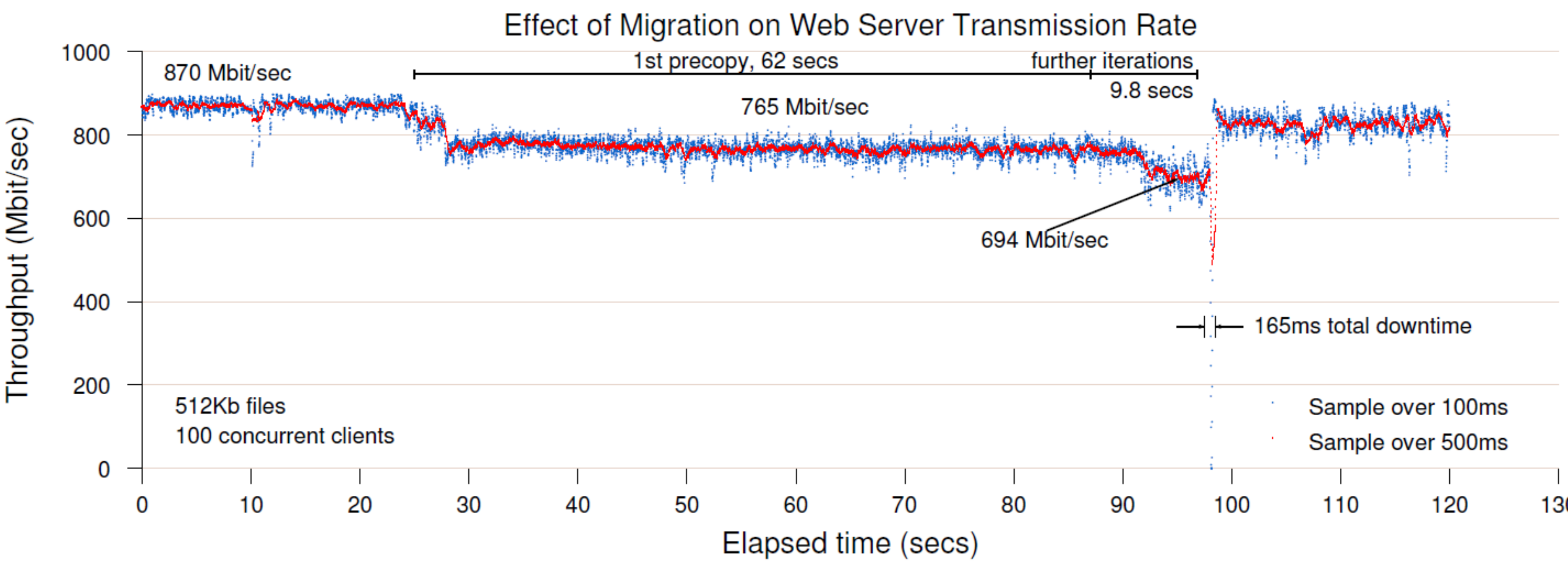
Stage 3: Stop and copy
- Suspend VM on Host A
- Generate ARP, synchronize state to Host B

Stage 4: Commitment
- Release VM state on Host A

VM running normally on Host B

Stage 5: Activation
- Start VM on Host B, connect to local devices
- Resume normal operation

XEN Migration Performance Figures



- Migration of a running web server VM (800 MB RAM)
  - Web server continuously serves 512 KB file to 100 clients

Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - **Resource Isolation and Performance Implication**
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

Resource Fairness & Performance Implications

- So far, we have more or less assumed one virtual machine per physical host
- In commercial IaaS clouds, many VMs often run on the same physical hardware

- Main questions:
  - What notion of fairness do cloud operators provide?
  - How is fairness enforced?
  - What are the implications of resource sharing?

Resource Distribution among VMs

- Storage space: statically partitioned
  - Each VM typically receives predefined fraction of disk
- Main memory: statically partitioned
  - Each VM typically receives predefined fraction of RAM
- CPU: Different methods possible
  - Pinning: Each VM is statically assigned CPU (cores)
  - Scheduling: VMM dynamically assigns time slots to VMs
- I/O Access: Typically FCFS
  - More sophisticated methods subject to research!

CPU Scheduling Algorithms in XEN

- Goals of the schedulers
  - Each VMs supposed to receive “fair” share of the CPU
  - High CPU utilization
  - Low response times

- Available algorithms
  - Credit Scheduler: general purpose, weighted fair share
  - Credit2 Scheduler: better with latency sensitive jobs
  - RTDS: real-time scheduler (Embedded, mobile, automotive, Graphics & Gaming)
  - ARINC 653: hard real-time (Avionics, Drones, Medical)

CPU Scheduling and Shared I/O

- Currently, VM scheduling focuses on CPU alone
  - Results in good fairness/response times for compute-intensive applications

- However, processing I/O requests by VMM also consumes CPU time
  - In particular when I/O is bursty
  - Guest OSs unaware of that source of processing delay
  - Perceived as high delay variations by the guests and negatively impacts I/O performance

Overview

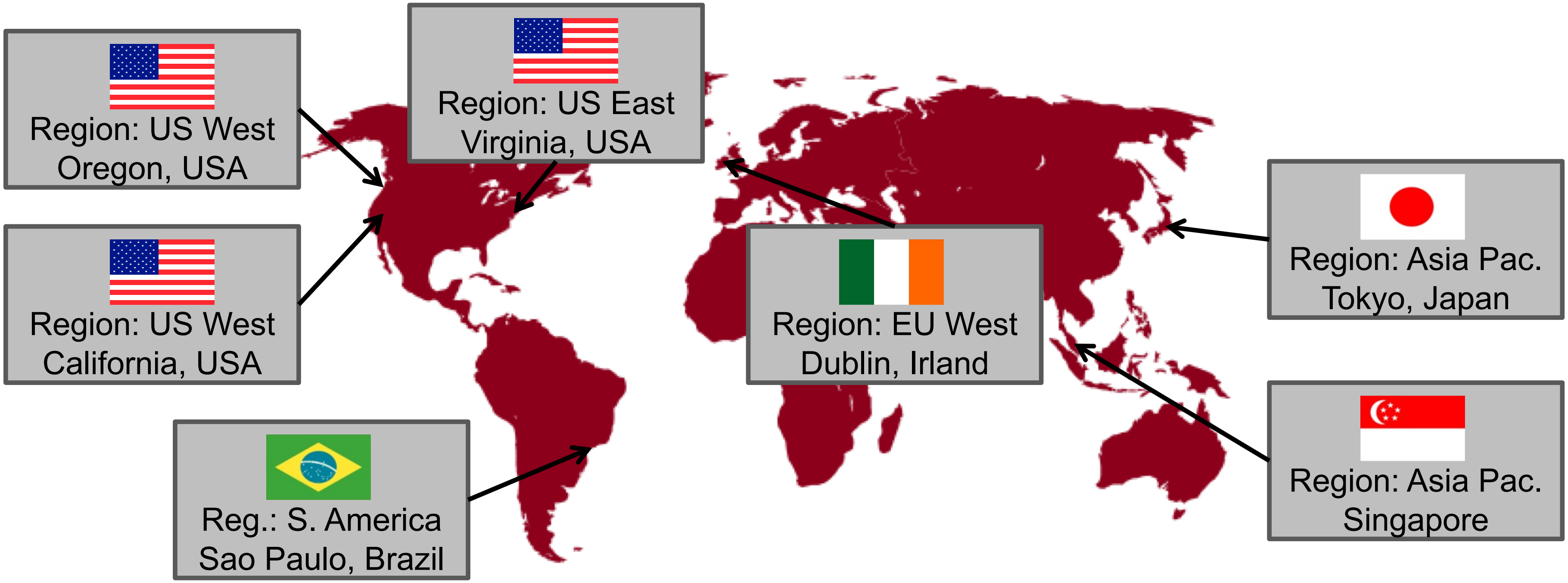
- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - **Case Study: Amazon EC2**
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

Amazon Elastic Compute Cloud (EC2)

- Public IaaS cloud by Amazon Web Services (AWS)
  - Subsidiary of Amazon.com, Inc.
  - Launched in 2006
  - Major cloud platform today

- AWS encompasses increasing number of service
  - Computing: EC2, Elastic MapReduce
  - Storage: Simple Storage Service, Elastic Block Storage
  - Databases: DynamoDB, RDS
  - Analytics: ElasticSearch, Lambda, SageMaker

Geographic Distribution of EC2 Data Centers



- Amazon calls each geographic location a *region*
  - Regions are subdivided into *availability zones*
    - Intra-availability zone traffic is free of charge
    - Per-GB fee for inter-availability zone/inter-region traffic

EC2 Per-Hour Pricing Model

- Per-hour pricing model (hence the term “Elastic”)
  - Amazon charges fee for each started hour of VM usage
  - Customer can shutdown VM at anytime

  → No long-term obligations, reduced risk of over-/under-provisioning

- Concrete per-hour cost depends on several factors
  - Region
  - Virtual machine type (EC2 calls those instance types)
  - Operating system, image (possible license costs)
  - Usage of external services (EBS, Internet traffic, …)

EC2 Instance Types (1/2)

- Instance types define VM classes with particular hardware characteristics

	Small Instance	Medium Instance	Large Instance	Extra Large Instance
Compute power	1 virtual core, 1 comp. unit	1 virtual core, 2 comp. units	2 virtual cores, 4 comp. units	4 virtual cores, 8 comp. Units
Main memory	1.7 GB	3.75 GB	7.5 GB	15 GB
Local storage	160 GB	410 GB	850 GB	1690 GB
Platform	32/64-Bit	32/64-Bit	64-Bit	64-Bit
Price	USD 0.06	USD 0.12	USD 0.24	USD 0.48

Example from region US East (Virginia), Linux/UNIX usage

EC2 Instance Types (2/2)

- "EC2 Compute Unit": Abstract unit for compute power
  - One compute unit corresponds to a 1.0-1.2 GHz AMD Opteron or Intel Xeon of 2007
  - Introduced to improve consistency and predictability among different generations of HW inside data center

- Schad et al. examined performance variations on EC2[18]
  - Instances of same type may be hosted on different generations of hardware

  → Significant performance variations across different instances of same type possible

References

- Andrew S. Tanenbaum, Herbert Bos: Modern Operating Systems, Pearson, 2015
- William Stallings: Operating Systems – Internals and Design Principles, 2015

[3] G.J. Popek and R.P. Goldberg: “Formal Requirements for Virtualizable Third Generation Architectures”, Communications of the ACM, 17 (7), 1974

[4] J.S. Robin, C.E. Irvine: “Analysis of the Intel Pentium’s Ability to Support a Secure Virtual Machine Monitor”, Proc. of the 9th Conference on USENIX Security Symposium, 2000

[6] K. Adams, O. Agesen: “A Comparison of Software and Hardware Techniques for x86 Virtualization”, Proc. of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, 2006

[7] A. Whitaker , M. Shaw , S.D. Gribble: “Denali: Lightweight Virtual Machines for Distributed and Networked Applications”, Proc. of the 2002 USENIX Annual Technical Conference, 2002

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield: “Xen and the Art of Virtualization”, Proc. of the 19th ACM Symposium on Operating Systems principles, 2003

[11] O. Agesen: “Performance Aspects of x86 Virtualization”, VMWORLD 2007

[13] G. Neiger, A. Santoni, F. Leung, D. Rodgers, R. Uhlig: “Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization”, Intel Technology Journal, 10 (3), 2006

[15] C. Clark, K. Fraser, S. Hand, J.G. Hanseny, E. July, C. Limpach, I. Pratt, A. Wareld: “Live Migration of Virtual Machines”, Proc. of the 2nd conference on Symposium on Networked Systems Design & Implementation, 2005

[18] J. Schad, J. Dittrich, J.-A. Quiané-Ruiz: “Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance”, Proc. of the VLDB Endowment, 3 (1-2), 2010