Cloud Systems

Chapter 2: Virtual Machines

Dr Lauritz Thamsen

lauritz.thamsen@glasgow.ac.uk

School of Computing Science

University of Glasgow

First Half: Lecture Chapters

Cloud Resource Management:

1. Cloud Computing Intro
2. **Virtual Machines**
3. Containers
4. Cloud Infrastructure Management
5. Cloud Sustainability

Outline of Chapter 2: Virtual Machines

2.1 Virtualizability

2.2 Full Virtualization (with Binary Translation)

– short break (10-15 minutes) –

2.3 OS- and Hardware-Assisted Virtualization

2.4 Resource Isolation and Performance Implications

2.5 Case Study: Amazon EC2 / AWS

Outline of Chapter 2: Virtual Machines

2.1 Virtualizability

2.2 Full Virtualization (with Binary Translation)

– short break (10-15 minutes) –

2.3 OS- and Hardware-Assisted Virtualization

2.4 Resource Isolation and Performance Implications

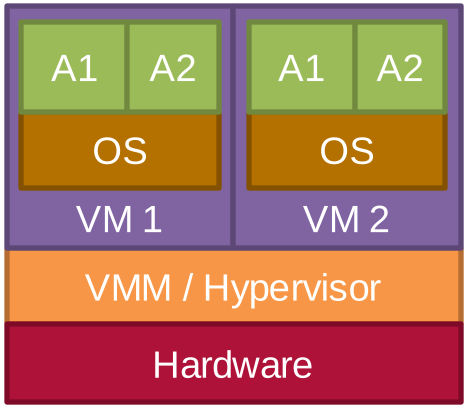
2.5 Case Study: Amazon EC2 / AWS

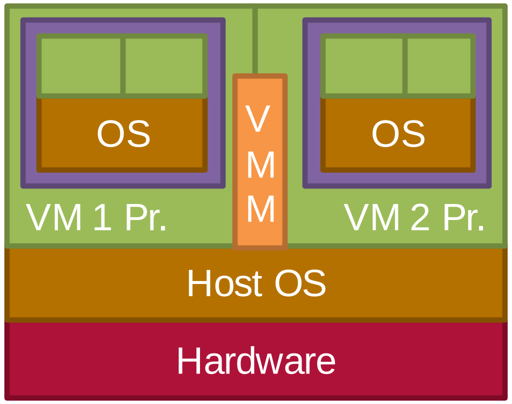
Basic Designs for System Virtualization

- 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 / native hypervisor virtualization





Also known as hosted virtualization

Virtualizability (VMM Type I)

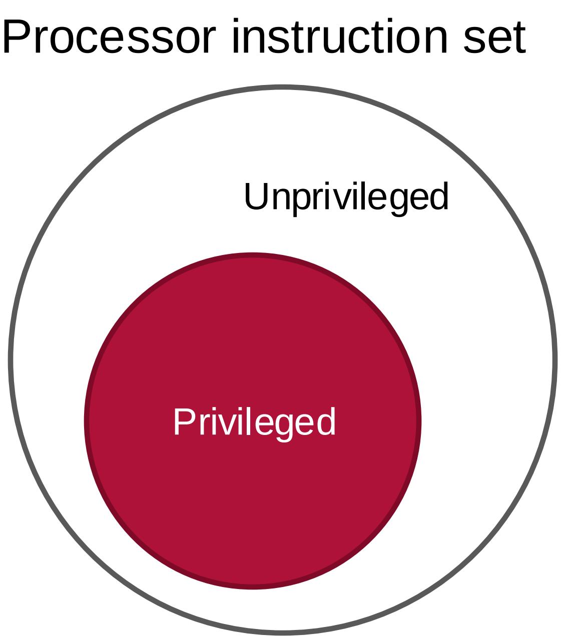
- Fundamental problem for system virtualization:
  - VMM must have ultimate control over hardware
  - Guest operating system must be disempowered without noticing

- 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

Recap: Categories of Processor Instructions (1/2)

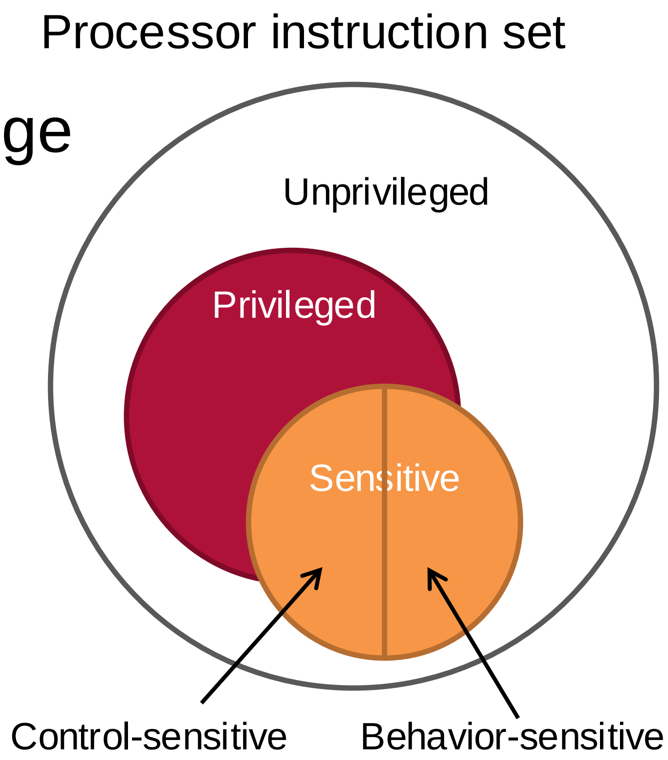
- Privileged instructions
  - Can only be executed in system mode
  - Trap when processor is in user mode

- Examples of privileged instruct.
  - 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



Recap: Categories of Processor Instructions (2/2)

- Sensitive instructions
  - Control-sensitive instructions: Change configuration of resources
  - Behavior-sensitive instructions: Behave different depending on configuration of resource

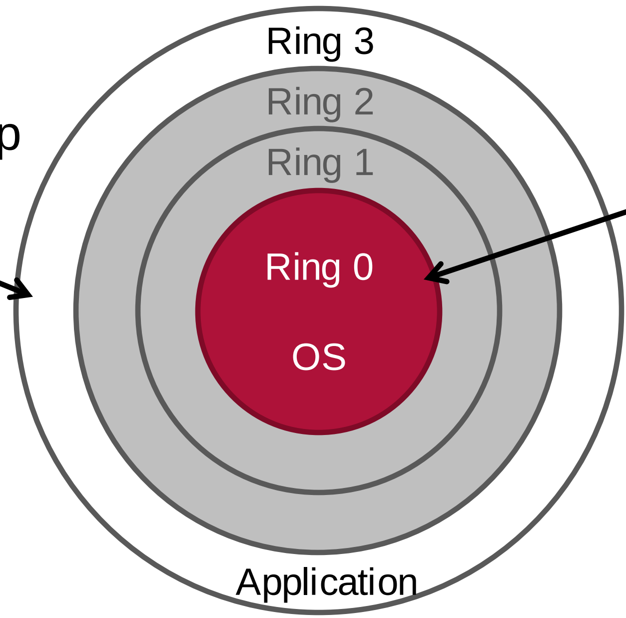


- Examples
  - Load Real Address (S/370)
  - Pop Stack into Flags Register (IA-32)

Recap: IA-32 Architectures

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

User space
- Runs applications
- Privileged instructions trap
- Execution of privileged instruction requires syscall

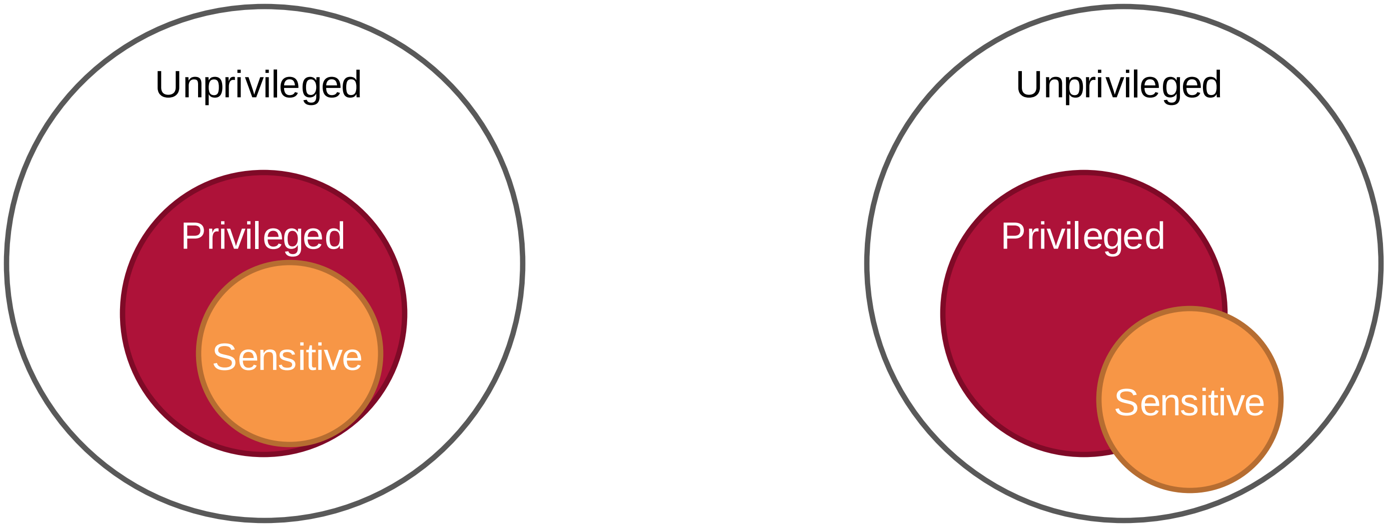


Supervisor mode
- Full control over CPU
- OS runs in this ring
- Equiv. to system mode

Popek and Goldberg‘s Theorem

- Basic condition for the construction of *efficient* VMMs

> “… 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**.”

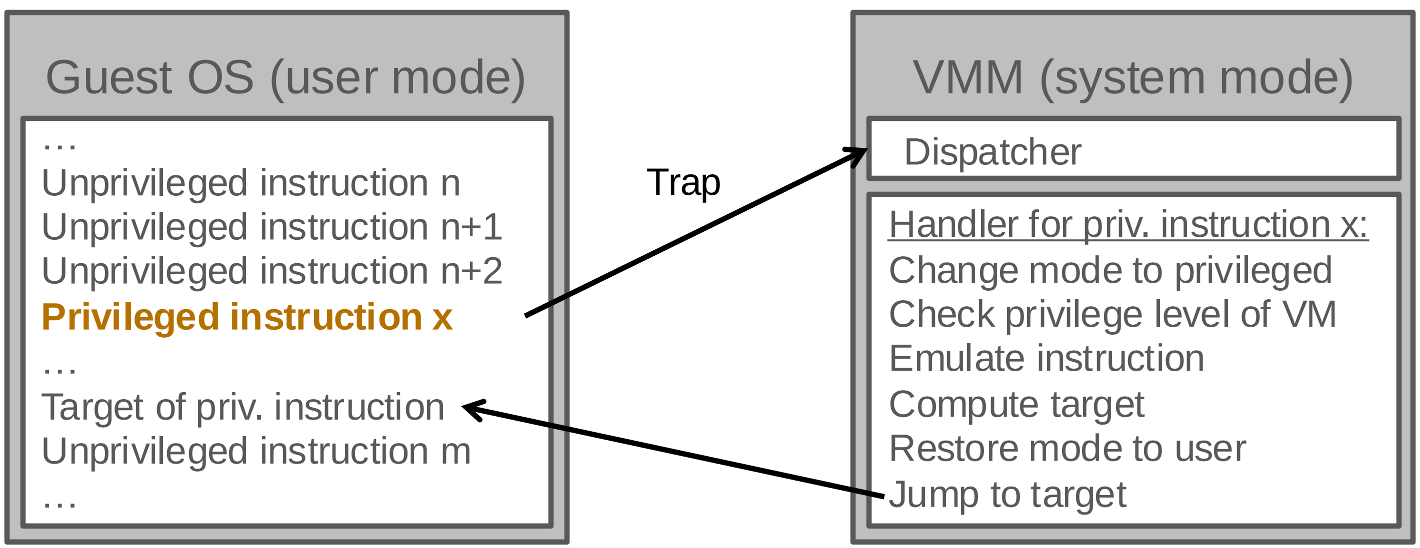


Condition satisfied

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
  - Instruction 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
  - Arm ✗
- Apparently, virtualization on IA-32 and ARM is possible. So, how?

Virtualization of IA-32 Architectures

1. Full Virtualization using Binary Translation

2. OS-Assisted Virtualization

3. Hardware-Assisted Virtualization

Outline of Chapter 2: Virtual Machines

2.1 Virtualizability

2.2 Full Virtualization (with Binary Translation)

– short break (10-15 minutes) –

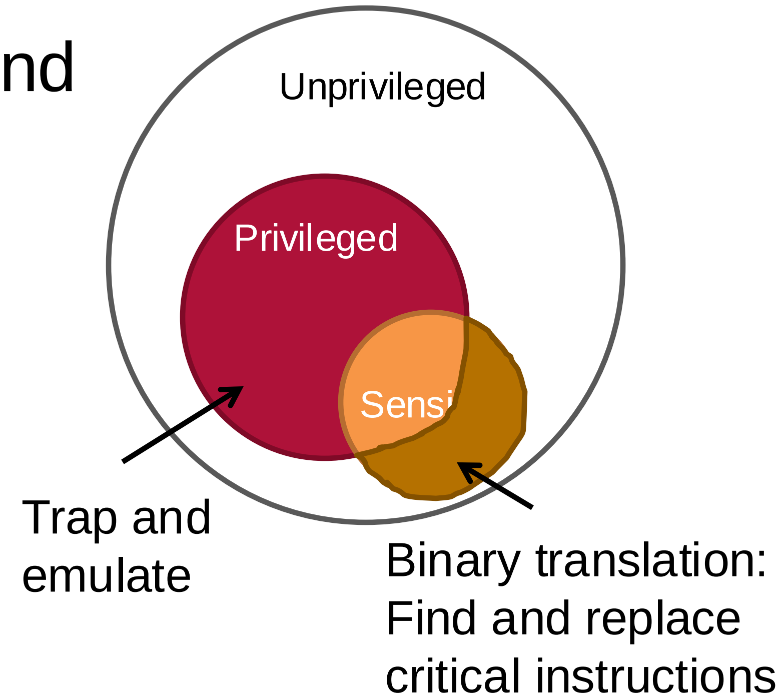
2.3 OS- and Hardware-Assisted Virtualization

2.4 Resource Isolation and Performance Implications

2.5 Case Study: Amazon EC2 / AWS

Full Virtualization using Binary Translation

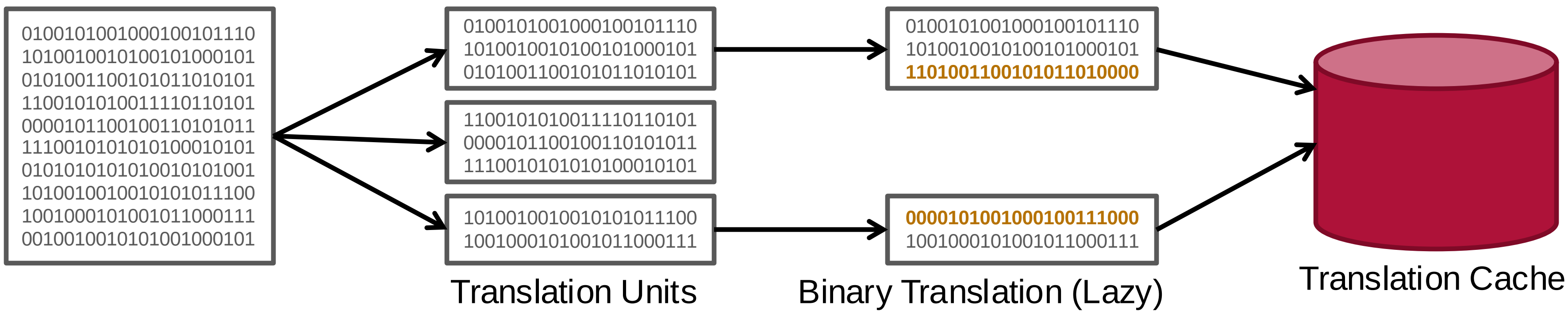
- Idea: Find critical instructions and replace them
  1. Run *unprivileged instructions* directly on CPU
  2. Trap and emulate *privileged instructions*
  3. Find *critical instructions* and replace with exception



- Whether an instruction is critical or not can depend on parameters used (e.g. LOAD instruction)
  - Replacement must be done at runtime
- Translating a book word for word is inefficient

Basic Approach for Binary Translation

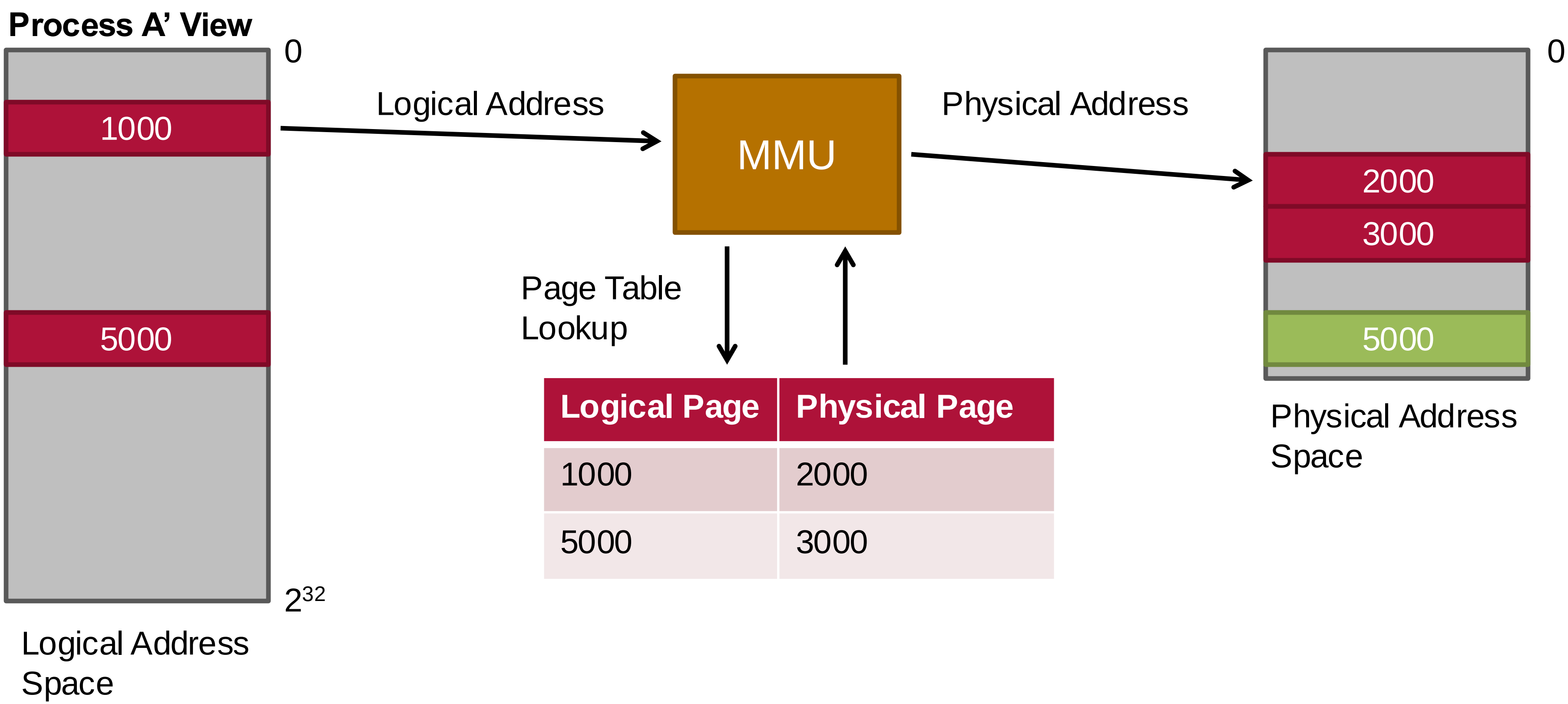
1. Separate instruction sequence into translation units
2. Check units for critical instructions and modify code
3. Modified code is stored in translation cache



- Translation is done lazily
  - Some units may be never translated (e.g. except. handl.)
  - Frequently used units benefit from translation cache

Recap: Memory Management on IA-32 Architectures (1/2)

- MMU translates logical to physical memory addresses



Logical Page	Physical Page
1000	2000
5000	3000

Recap: Memory Management on IA-32 Architectures (2/2)

- Page tables are part of IA-32 architecture
  - Hardware knows the layout of page tables
  - OS modifies the page table, lookup happens transparently

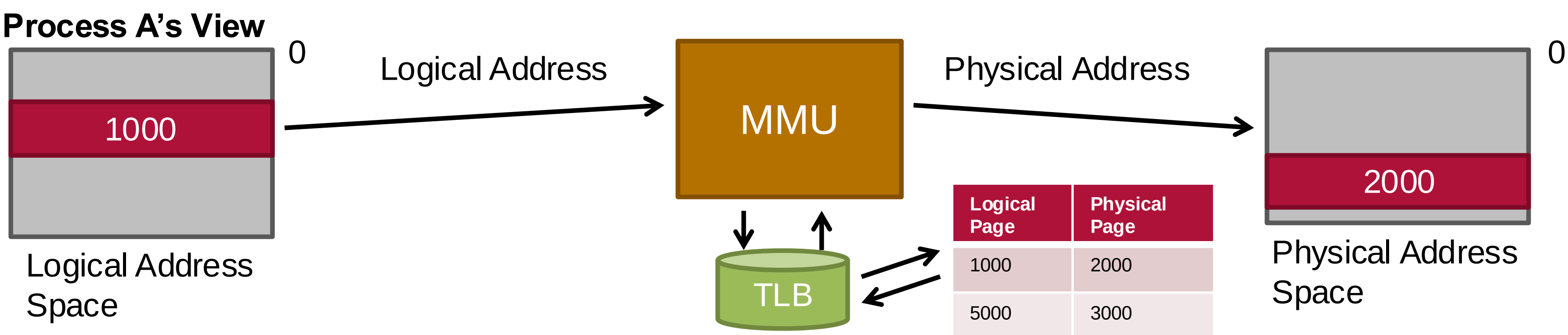
- Page tables reside in main memory themselves → Overhead of memory access essentially doubles

- Idea: Introduce special hardware-accelerated cache to remember recent address translations

  → Translation Lookaside Buffer (TLB)

Recap: Translation Lookaside Buffer (TLB)

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

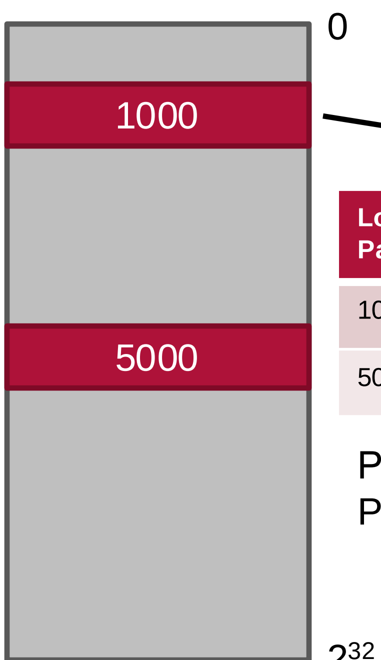


- On IA-32, the TLB is invisible to the operating system
  - Is updated by hardware on every page table lookup
  - It is flushed on every context switch

Memory Management and Full Virtualization (1/2)

- General idea: Add another level of indirection

Process A's View in VM 1

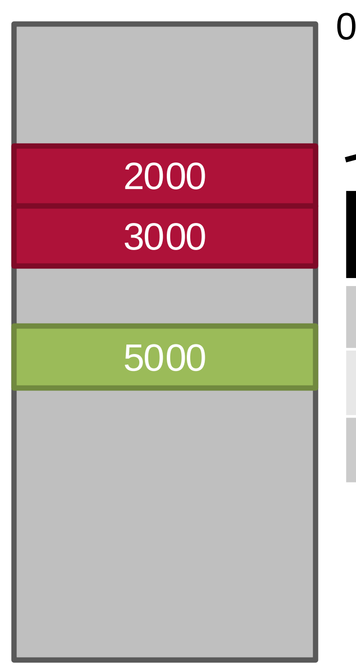


Logical Address Space

Logical Page	Real Page
1000	2000
5000	3000

Process A's Page Table

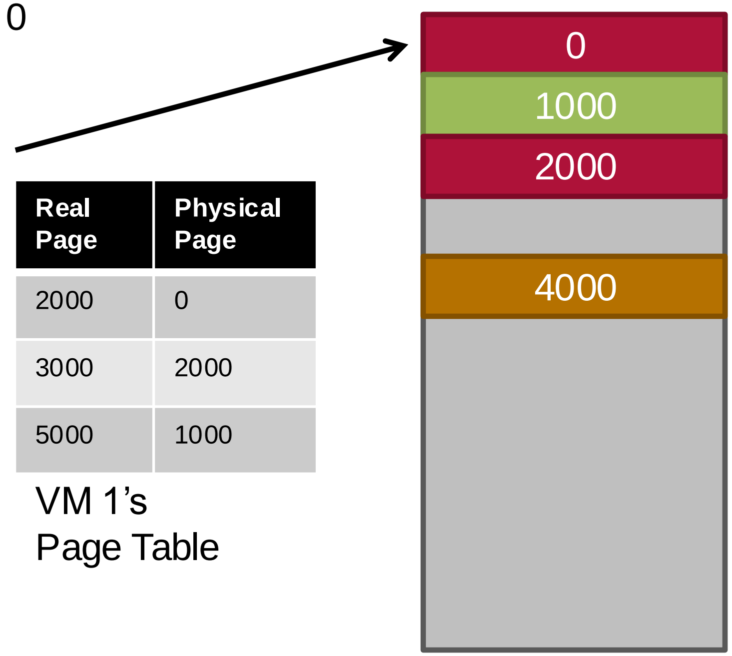
Guest OS's View in VM 1



Real Address Space

Real Page	Physical Page
2000	0
3000	2000
5000	1000

VM 1's Page Table



Physical Address Space

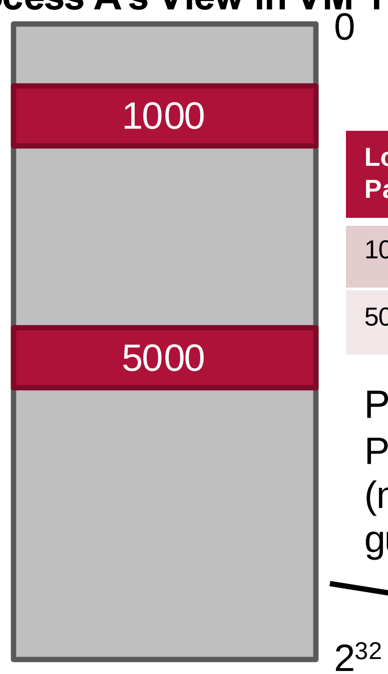
Memory Management and Full Virtualization (2/2)

- 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 then used by hardware
    - Keeps TLB up-to-date
    - Works through virtualization of page table pointer

Memory Virtualization with Shadow Page Tables

Process A's View in VM 1

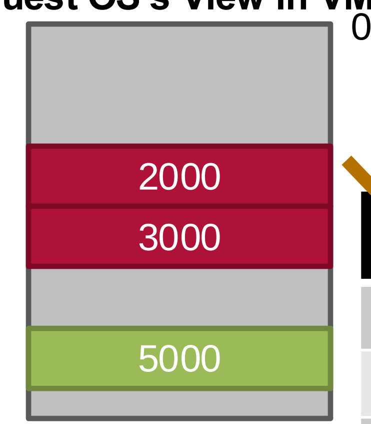


Logical Address Space

Logical Page	Real Page
1000	2000
5000	3000

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

Guest OS's View in VM 1



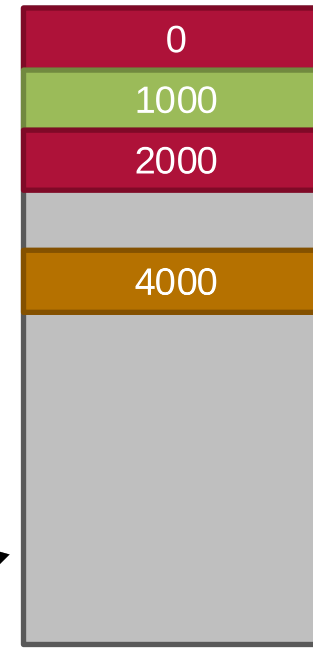
Real Address Space

Real Page	Physical Page
2000	0
3000	2000
5000	1000

VM 1's Page Table

Logical Page	Physical Page
1000	0
5000	2000

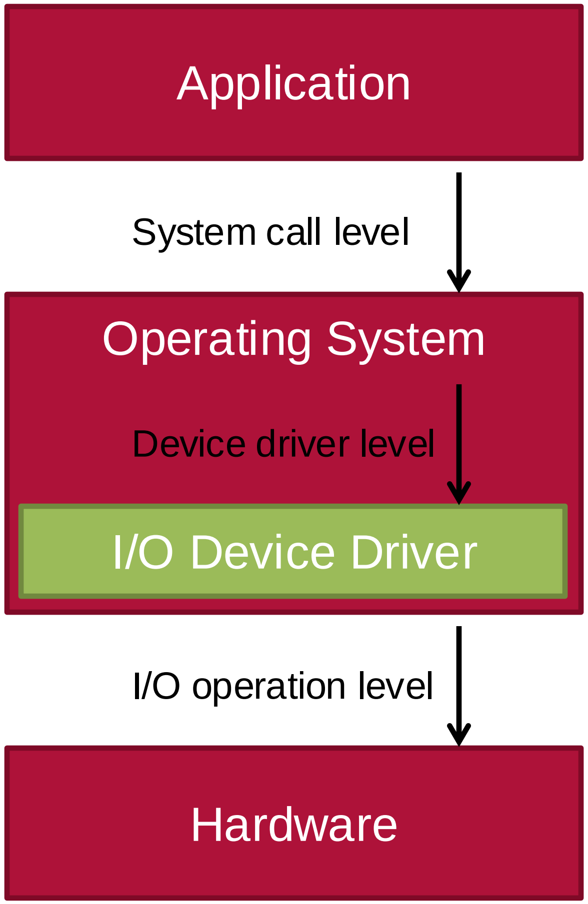
Process A of VM 1's Shadow Page Table (maintained by VMM)



Physical Address Space

Full Virtualization and I/O

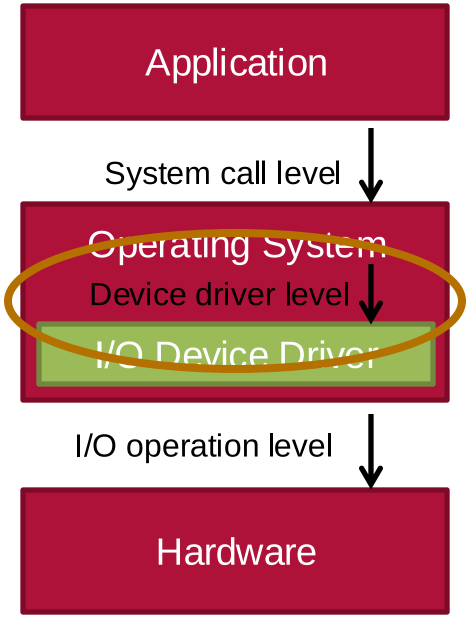
- Different levels of I/O virtualization possible
  1. At system call level
  2. At device driver level
  3. At I/O operation level



I/O Virtualization at 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

→ Works in practice (e.g. for Linux and Windows guests)



Summary Full Virtualization with Binary Translation

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

Outline of Chapter 2: Virtual Machines

2.1 Virtualizability

2.2 Full Virtualization (with Binary Translation)

– short break (10-15 minutes) –

2.3 OS- and Hardware-Assisted Virtualization

2.4 Resource Isolation and Performance Implications

2.5 Case Study: Amazon EC2 / AWS

Outline of Chapter 2: Virtual Machines

2.1 Virtualizability

2.2 Full Virtualization (with Binary Translation)

– short break (10-15 minutes) –

2.3 OS- and Hardware-Assisted Virtualization

2.4 Resource Isolation and Performance Implications

2.5 Case Study: Amazon EC2 / AWS

OS-Assisted Virtualization

- 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]
- Today, most virtualization platforms are aware of using virtual devices (→ OS-assisted virtualization for I/O)
- Requirements for (pure) OS-assisted approach
  - Source code of guest operating system is available
  - Modified guest OS maintains application binary interface

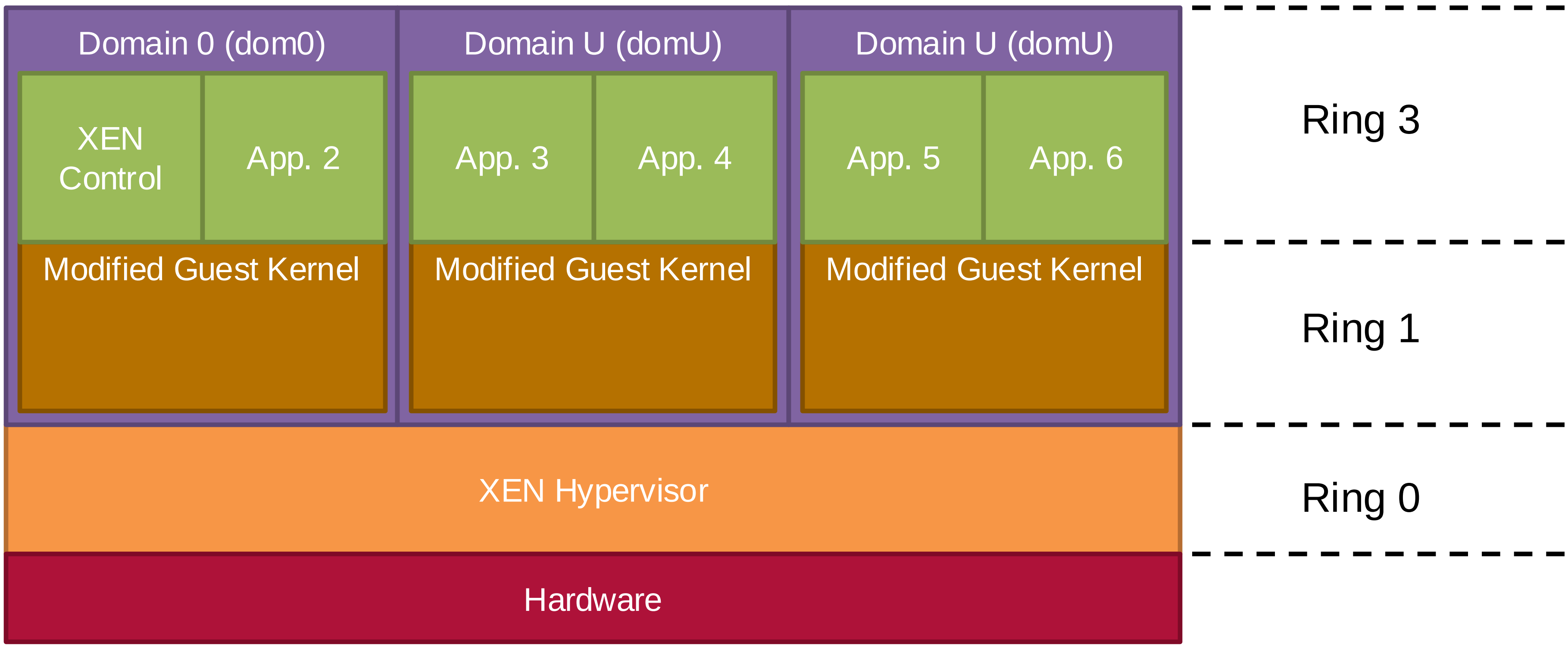
OS-Assisted Virtualization in Practice: The Example of Xen

- Classic representative for OS-assisted virtualization: XEN [8]
  - Type I Hypervisor
  - Available as open-source software
  - Originally developed at the University of Cambridge, UK, in collaboration with Microsoft Research Cambridge

- Prime example for paravirtualization, but can also be used to run unmodified guest kernels using HW-assisted virtualization

XEN Architecture and Domains

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

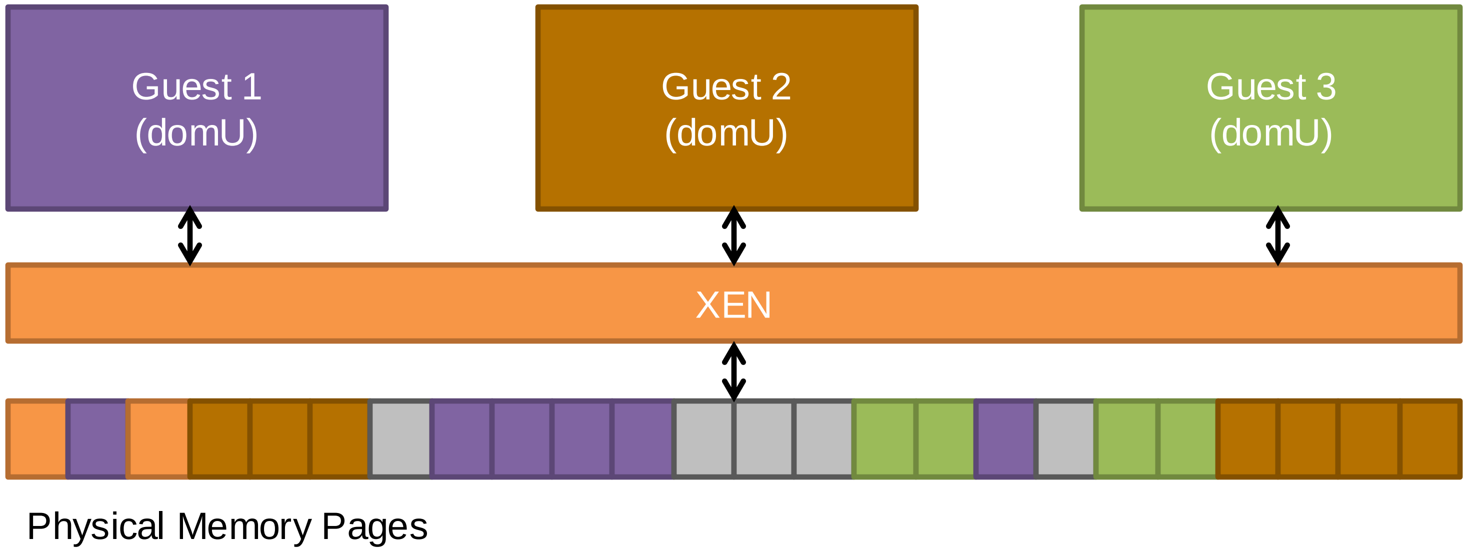


XEN and CPU Virtualization

- Critical instructions do not trap on IA-32
  - Guest OS is aware of virtualization → Critical instructions can be avoided
- Still, frequent VMM intervention required, e.g. for system calls and page table updates
- However, XEN minimizes frequencies and costs
  - Some system calls to guest OS (Ring 1) without VMM intervention (registering guest OS handlers with Xen)
  - Mapping hypervisor code into process address spaces (so hypercalls do not require context switches)
  - Command batching (e.g. subsequent page table updates)

XEN and Physical Memory

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



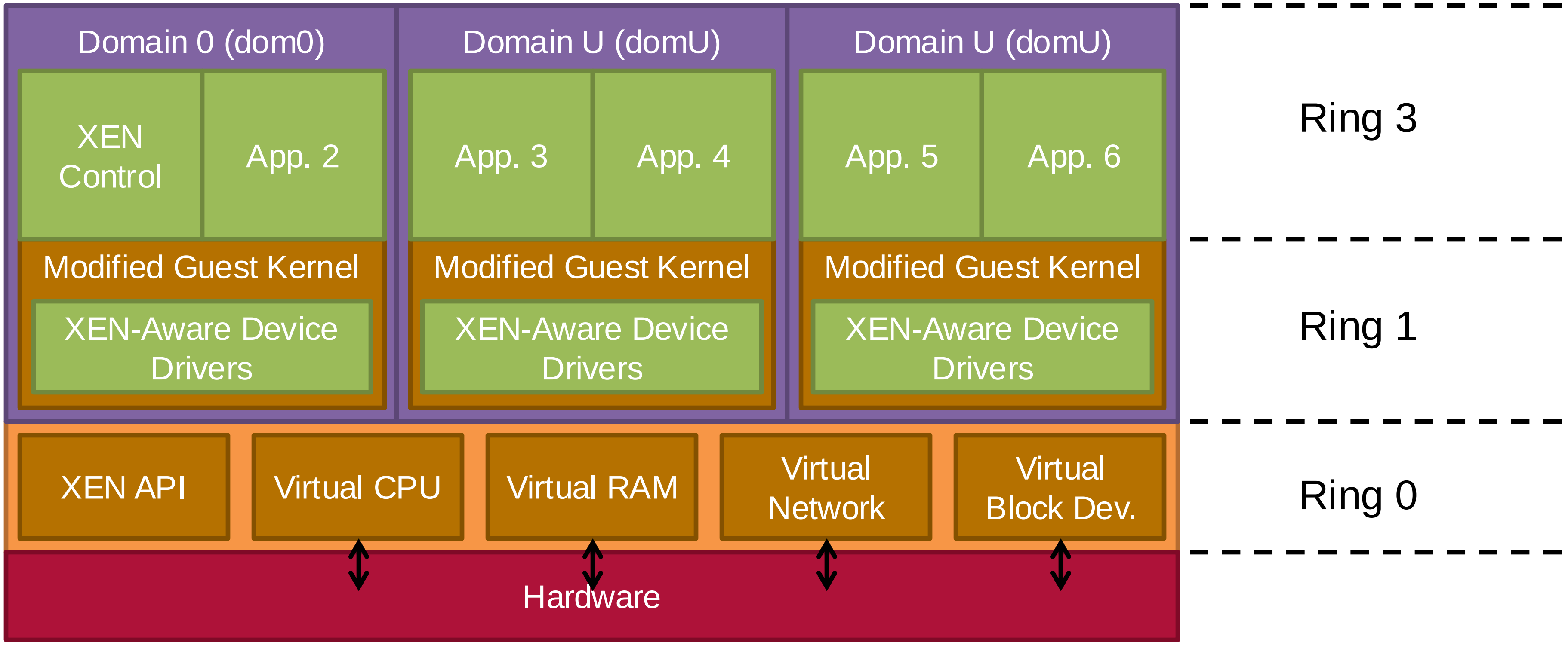
XEN and Memory Virtualization

- XEN lets guests maintain their own page tables
  - Prerequisite: Guest OS knows its fraction of physical memory
  - Guest page tables are used by the guest’s MMU
  - XEN validates page table updates to ensure isolation
  - No need for hypervisor intervention on reads

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

XEN and I/O Virtualization

- XEN presents virtual devices, domains use lightweight virtual device drivers, and physical device drivers reside in dom0
- Communication between XEN and domains:
  - Hypercall: Synchronous call from domain to XEN
  - Event: Asynchronous notification from XEN to domain



Summary OS-Assisted Virtualization

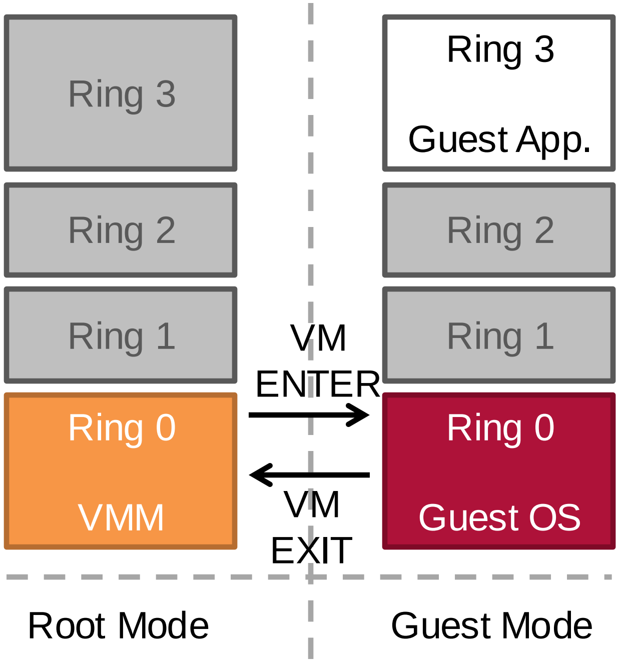
- Requires modified guest OS? **YES**
- Requires hardware support? **NO**
- Pros:
  - Better performance through cooperation between hypervisor and guest OS
- Cons:
  - Limited compatibility, not generally applicable
  - Increased management overhead for data center operator as different version of OS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

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 share same basic ideas

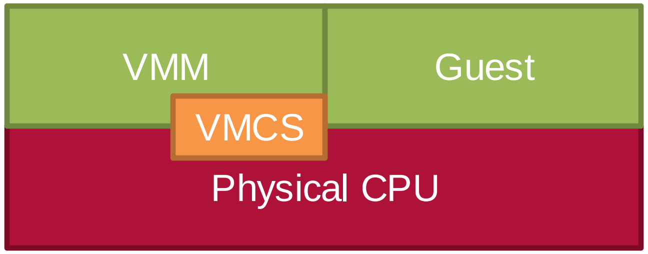
CPU Virtualization Extensions

- 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 [13]

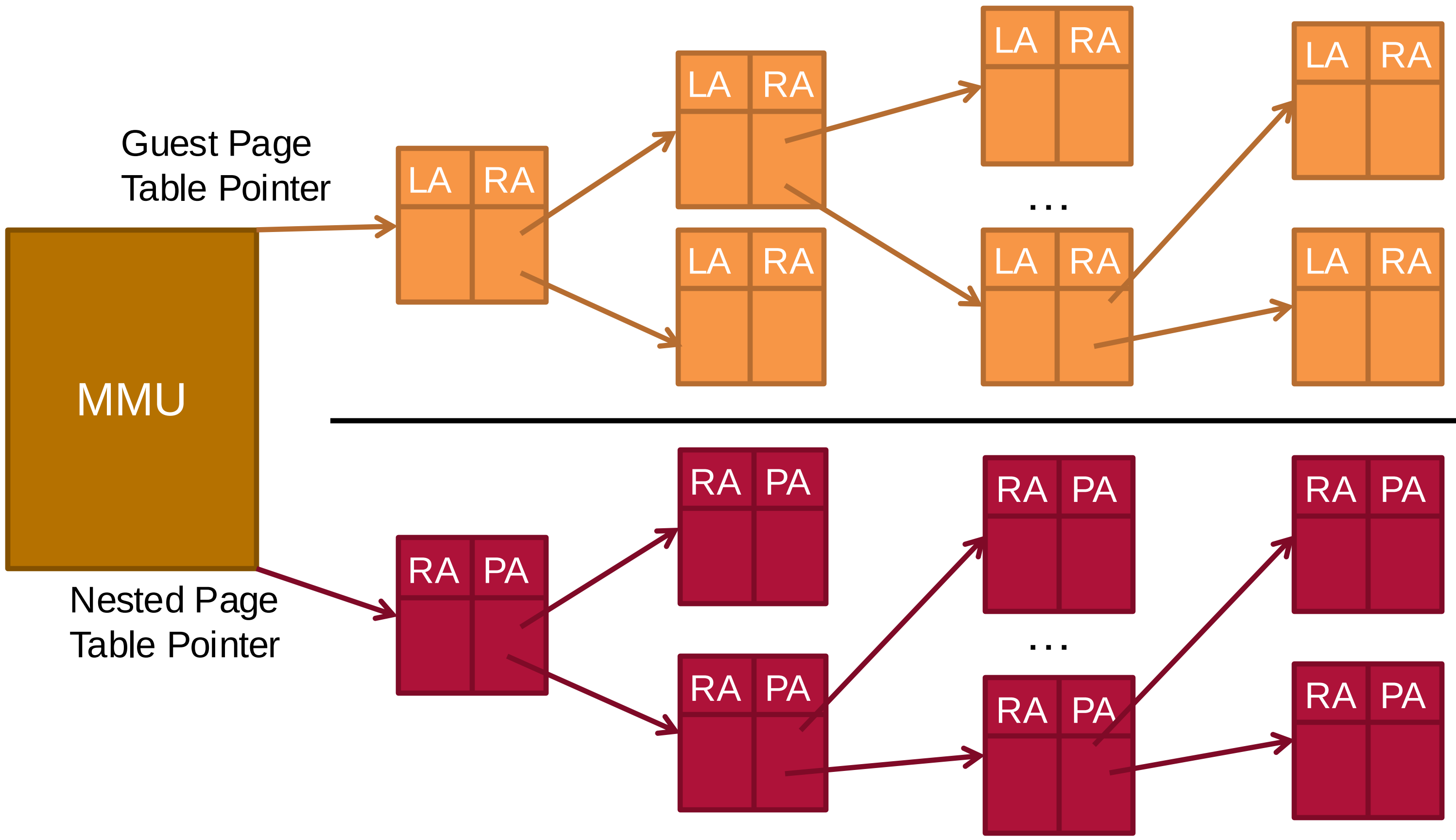
- VMM controls guest through HW-defined structure
  - Intel: VMCS (virtual machine control structure)
  - AMD: VMCB (virtual machine control block)



- 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

Memory Virtualization Extensions

- Extended Page Tables/Nested Page Tables



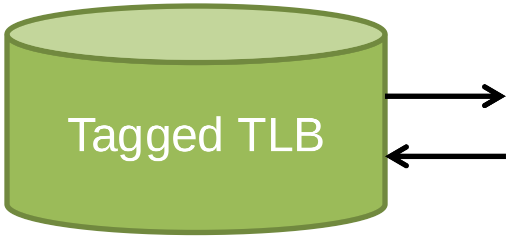
LA: Logical Address, RA: Real Address, PA: Physical Address

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

Focus of HW-Support for I/O: 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
- VMM intervention on every IN or OUT instruction
- Physical devices 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
- Cons:
  - Specialized hardware required and reduced flexibility due to hardware constraints

Outline of Chapter 2: Virtual Machines

2.1 Virtualizability

2.2 Full Virtualization (with Binary Translation)

– short break (10-15 minutes) –

2.3 OS- and Hardware-Assisted Virtualization

2.4 Resource Isolation and Performance Implications

2.5 Case Study: Amazon EC2 / AWS

Resource Sharing & Performance Implications

- In commercial IaaS clouds, many VMs often run on the same physical hardware

- Main questions:
  - How are resources of a physical node shared?
  - 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

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

- Different algorithms available for e.g. XEN, including
  - a general-purpose scheduler
  - a scheduler for latency-sensitive jobs
  - several experimental real-time schedulers
  - …

CPU Scheduling and Shared I/O

- Currently, VM scheduling focuses on CPU
- Results in good fairness/response times for compute-intensive applications
- However, processing of 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 performance

Outline of Chapter 2: Virtual Machines

2.1 Virtualizability

2.2 Full Virtualization (with Binary Translation)

– short break (10-15 minutes) –

2.3 OS- and Hardware-Assisted Virtualization

2.4 Resource Isolation and Performance Implications

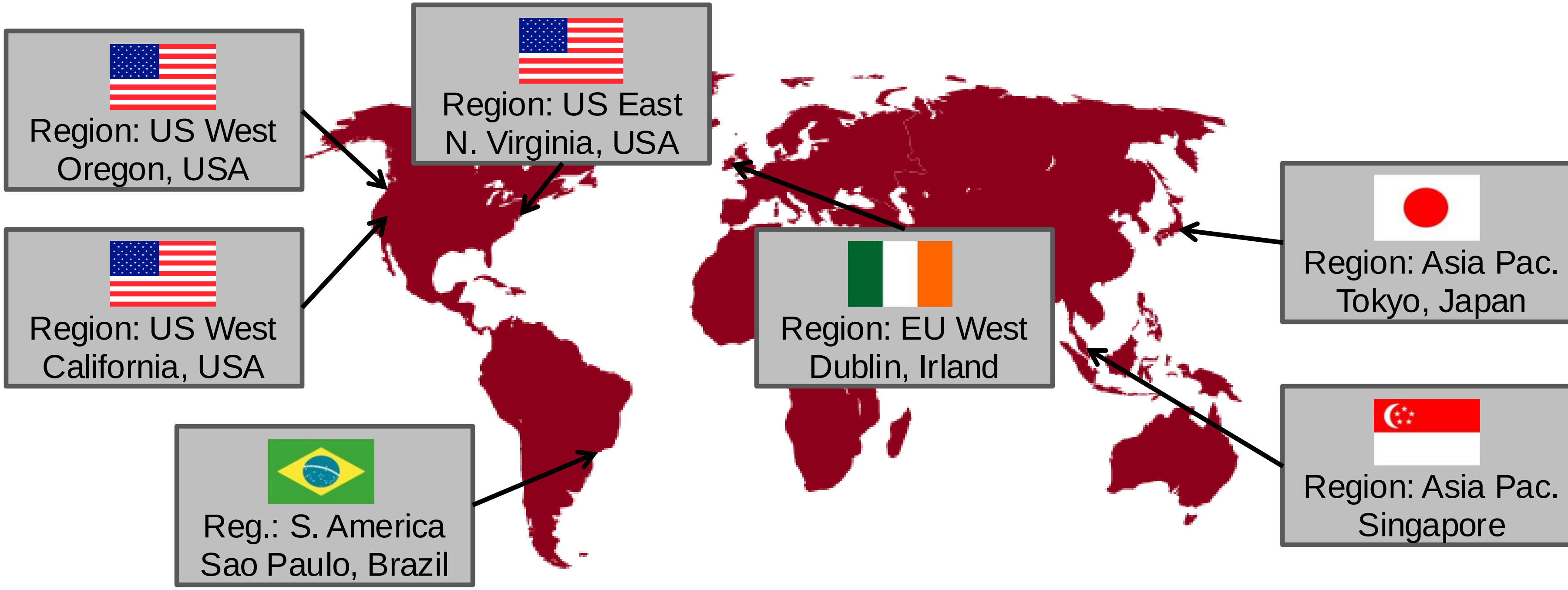
2.5 Case Study: Amazon EC2 / AWS

Amazon Elastic Compute Cloud (EC2)

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

- AWS encompasses large number of service, e.g.
  - Computing: EC2, Lambda
  - Storage: S3, ECB
  - Databases: RDS, DynamoDB
  - Analytics: EMR, SageMaker

Geographic Distribution of EC2 Data Centers



- Amazon calls each geographic location a *region*
  - Regions are again divided 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-time pricing model (hence the term “Elastic”)
  - Amazon charges fee for each started hour of VM usage
  - Customer can shutdown VMs 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

Instance	t2 Small	t2 Medium	t2 Large	t2 xLarge
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	2 GiB	4 GiB	8 GiB	16 GiB
Storage	via EBS			
Platform	32/64-Bit	32/64-Bit	64-Bit	64-Bit
Price	USD 0.02	USD 0.05	USD 0.09	USD 0.19

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

EC2 Instance Types (2/2)

- "EC2 Compute Unit": Abstract unit for compute power
  - One compute unit corresponded to a 1.0-1.2 GHz AMD Opteron or Intel Xeon of 2007
  - Introduced as a reference measure, allowing predictability, with different generations of HW inside data centers

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

  → Significant performance variations across different instances of the same type are possible!

Outline of Chapter 2: Virtual Machines

2.1 Virtualizability

2.2 Full Virtualization (with Binary Translation)

– short break (10-15 minutes) –

2.3 OS- and Hardware-Assisted Virtualization

2.4 Resource Isolation and Performance Implications

2.5 Case Study: Amazon EC2 / AWS

Summary: Virtual Machines

- IaaS clouds let consumers rent basic IT resources in form of virtual machine
  - Customers have full control over OS and deployed applications in VMs

- System virtualization as the enabling method
  - Several customers can share physical infrastructure
  - Different techniques to realize system virtualization
  - Performance overhead depends on specific virtualization technique, application, and hardware

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

[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