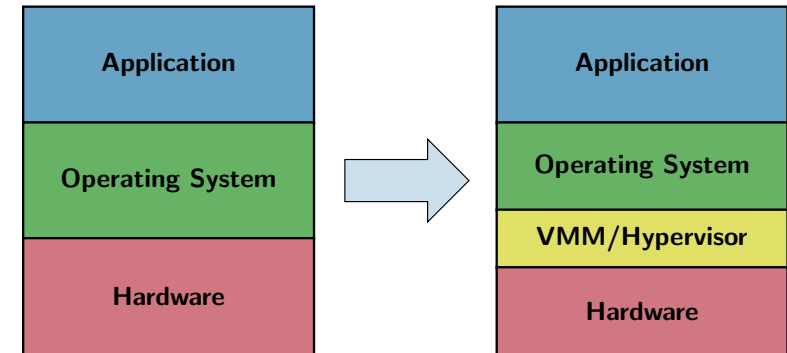


## Virtualisierung

Motivation  
Grundlagen  
Xen  
Linux-VServer  
Zusammenfassung



- Einführung eines *Virtual Machine Monitor (VMM)*
  - Hier als Synonym verwendet: *Hypervisor*
  - Zusätzliche Indirektionsstufe zwischen
    - Betriebssystem und Anwendung → Prozessvirtualisierung
    - Hardware und Betriebssystem → Systemvirtualisierung

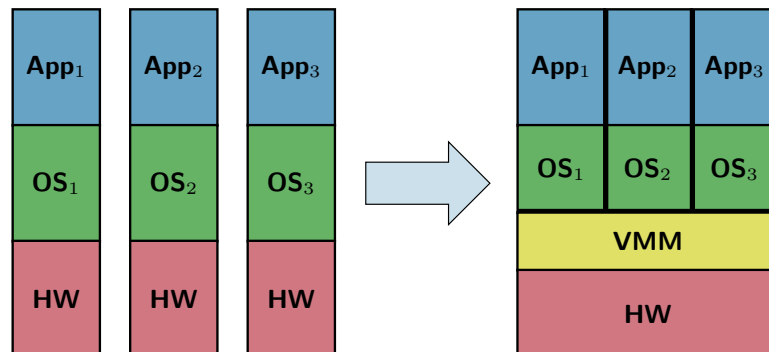


- *Viruelle Maschine (VM)*: Vom VMM bereitgestellte Umgebung



## Einsatzbereiche (Beispiele)

- Ausnutzung der Hardware-/Plattform-Unabhängigkeit
- Unterstützung von Legacy-Anwendungen
- Bereitstellung von Fehlertoleranzmechanismen
- Zentrale Technik zur Server-Konsolidierung
  - Zusammenlegung von schwach ausgelasteten Rechnern
  - Parallelbetrieb verschiedener Anwendungen und Betriebssysteme möglich



## Anforderungen an ein virtualisiertes System

„A **virtual machine** is taken to be an **efficient, isolated duplicate of the real machine.**”  
[Popek et al.]

- Eigenschaften nach [Popek et al.]
  - Äquivalenz
    - Identisches Verhalten im Vergleich zur nichtvirtualisierten Ausführung
    - Erlaubte Ausnahmen
      - \* In der geringeren Verfügbarkeit von Ressourcen bedingte Auswirkungen
      - \* Abweichendes zeitliches Verhalten
  - Ressourcenkontrolle
    - VMM hat die komplette Kontrolle über alle System-Ressourcen
    - VMM teilt VM Ressourcen zu, kann ihr diese aber auch wieder entziehen
  - Effizienz: Ein Großteil aller Instruktionen soll direkt von der Hardware, also ohne Umweg über den VMM, ausgeführt werden

### Literatur



Gerald J. Popek and Robert P. Goldberg  
**Formal requirements for virtualizable third generation architectures**  
*Communications of the ACM*, 17(7):412-421, 1974.



## Virtualisierbarkeit nach [Popek et al.]

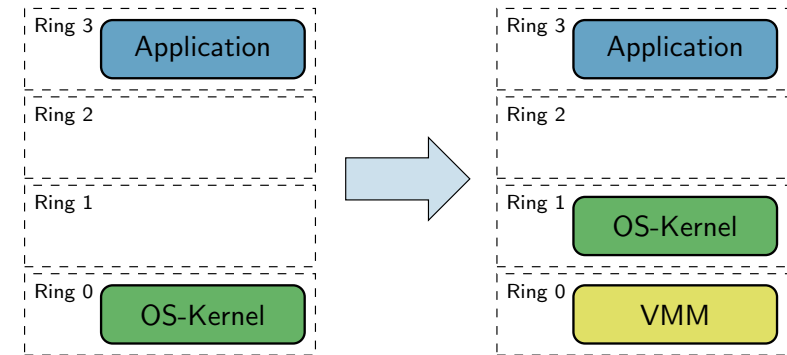
- Existenz (mindestens) zweier Betriebsmodi
  - Uneingeschränkter Modus (*Supervisor Mode*)
  - Eingeschränkter bzw. Nutzer-Modus (*User Mode*)
- Kategorisierung von Instruktionen
  - *Privilegierte vs. nichtprivilegierte* Instruktionen
    - Privilegierte Instruktionen: *Trap* bei Aufruf im Nutzer-Modus
    - Nichtprivilegierte Instruktion: Kein *Trap* bei Aufruf im Nutzer-Modus
  - *Sensitive vs. „harmlose“ (innocuous)* Instruktionen
    - Sensitive Instruktionen können
      - \* Zustände außerhalb des Isolationsbereichs des Aufrufers beeinflussen
      - \* durch externe Zustände beeinflusst werden
    - Harmlose Instruktionen: alle nichtsensitiven Instruktionen
- Kriterium für Virtualisierbarkeit

**Die Menge der sensitiven Instruktionen muss eine Teilmenge der Menge der privilegierten Instruktionen sein**



## Virtualisierung mittels *Trap-and-Emulate*

- Reduzierung der Privilegien des in der VM laufenden Betriebssystems



Beispiel: Privilegierungsstufen der x86-Architektur, Ring 0: Supervisor Mode

- Aufgaben des Virtual Machine Monitor
  - Verwaltung von *Schattendatenstrukturen* (z. B. Register) für VM
  - Abfangen (→ *Trap*) der von der VM initiierten privilegierten Instruktion
  - Emulation des von der VM erwarteten Verhaltens einer Instruktion



## Fallbeispiel: x86-Architektur

- Untersuchte Prozessoren: Pentium, Pentium II, Pentium III,...
- Beispiel: Zugriffe auf das Code-Segment-Register (CS)
  - In Teilen des Registers ist der aktuelle Betriebsmodus codiert
  - *PUSH*-Instruktion
    - Kopieren von Registerinhalten auf den Stack
    - Nichtprivilegierte Instruktion
  - Problematischer Zugriff in virtualisierter Umgebung
    - Ein in einer virtuellen Maschine im vermeintlichen Ring 0 ausgeführter Prozess liest per *PUSH* den Inhalt des CS-Registers aus
    - CS-Registerinhalt offenbart Betriebsmodus mit geringeren Privilegien
- Ergebnis der Studie von [Robin et al.]
  - Insgesamt 17 von ~250 Instruktionen problematisch
  - Prozessoren im Sinne der Definition von [Popek et al.] nicht virtualisierbar
- Literatur



John Scott Robin and Cynthia E. Irvine  
**Analysis of the Intel Pentium's ability to support a secure virtual machine monitor**  
*Proceedings of the 9th USENIX Security Symposium, pages 129-144, 2000.*



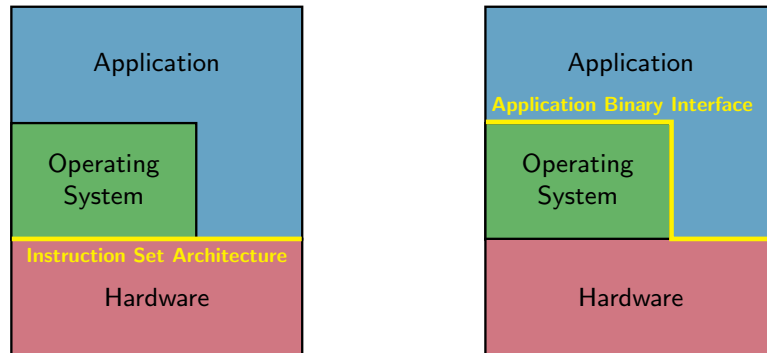
## Virtualisierung mittels *Binary Translation*

- Einordnung
  - Alternative zum *Trap-and-Emulate*-Ansatz
  - Verwandtes Prinzip: Ausführung von Byte-Code in Java
- Einsatz eines *Interpreters*
  - Virtuelle Maschine hat keinen direkten Zugriff auf CPU
  - Übersetzung von VM-Instruktionen auf Instruktionen der Zielplattform
  - Üblicherweise Basis-Blöcke als Übersetzungseinheit
- Vorgehen bei identischen Instruktionssätzen von VM und Hardware
  - 1:1-Abbildung aller nichtsensitiven Instruktionen
  - Anpassung der sensitiven Instruktionen durch den VMM
    - Illusion eines Betriebsmodus mit höheren Privilegien
    - Übersetzung von Speicheradressen
    - ...
- Anwendungsbeispiel für x86-Architektur: VMware Workstation



## Virtualisierungsebenen

- Systemvirtualisierung
  - Virtualisierung der *Instruction Set Architecture (ISA)*
  - Beispiele
    - Hardware-unterstützte Virtualisierung
    - Paravirtualisierung
- Prozessvirtualisierung
  - Virtualisierung des *Application Binary Interface (ABI)*
  - Beispiel: Betriebssystemvirtualisierung



## Hardware-unterstützte Virtualisierung

- Kommerzielle Umsetzungen: Intel VT-x, AMD-V
- Hardware-Erweiterungen für Virtualisierung
  - Zusätzlicher Ausführungsmodus (*Gastmodus*)
    - Direkte Ausführung von {nicht,}privilegiertem VM-Code
    - Instruktionen für Wechsel in den bzw. aus dem Gastmodus
  - Datenstruktur zur Verwaltung von VMs (**V**irtual **M**achine **C**ontrol **B**lock)
- Funktionsweise
  - VM-Code wird im Gastmodus ohne Einbeziehung des VMM ausgeführt
  - VMM legt per VMCB Kriterien für Austritt aus dem Gastmodus fest
  - Hardware übergibt bei Eintreffen eines dieser Kriterien Kontrolle an VMM

→ Ziel: Minimierung der Wechsel zwischen den Ausführungsmodi

### Literatur



Keith Adams and Ole Agesen

**A comparison of software and hardware techniques for x86 virtualization**  
*Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '06), pages 2-13, 2006.*

## Paravirtualisierung

- Probleme
  - Vollständige Virtualisierung der ISA oftmals aufwendig bzw. ineffizient
  - Nicht alle Prozessoren ermöglichen Hardware-unterstützte Virtualisierung
- Lösungsansatz
  - Verzicht auf Einhaltung der Äquivalenz-Bedingung von [Popek et al.]
  - Bereitstellung einer der ISA „ähnlichen“ Schnittstelle
  - Erweiterung des VMM um zusätzliche Methoden, die vom Betriebssystem einer virtuellen Maschine direkt aufgerufen werden können → *Hypercalls*
  - Sicherstellung der Isolation durch den VMM
- Konsequenzen
  - Höhere Effizienz durch Kooperation zwischen VM und VMM
  - Vereinfachte Implementierung des VMM
  - Um in einer paravirtualisierten Umgebung laufen zu können, ist eine **Portierung des (Gast-)Betriebssystems** erforderlich
- Beispiele: Xen, VMware ESX Server

## Betriebssystemvirtualisierung

- Ausgangspunkt
  - Nicht immer ist es erforderlich, virtuelle Maschinen mit heterogenen Betriebssystemen auf dem selben physischen Rechner auszuführen
  - Optimierungsmöglichkeiten durch Festlegung auf ein Betriebssystem
- Ansatz
  - Verlagerung der Virtualisierung auf ABI-Ebene
  - Identischer Betriebssystem-Kernel für alle virtuellen Maschinen
  - Instanziierung des Betriebssystems
  - Virtuelle Maschinen im User-Space
  - Ausnutzung von existierenden Mechanismen zur Isolation von Prozessen
- Beispiele
  - Linux-VServer
  - FreeBSD Jail
  - Solaris Containers

# Xen

- Zielsetzungen
  - Gleichzeitiger Betrieb von bis zu 100 VMs auf dem selben Rechner
  - Identische Performanz im Vergleich zur nichtvirtualisierten Ausführung
  - Einsatz heterogener Betriebssysteme in VMs
- Xen-Hypervisor
  - Virtual Machine Monitor für die x86-Architektur
  - Paravirtualisierung → Portierung der Gastbetriebssysteme nötig
  - Keine Modifikation der Anwendungen erforderlich

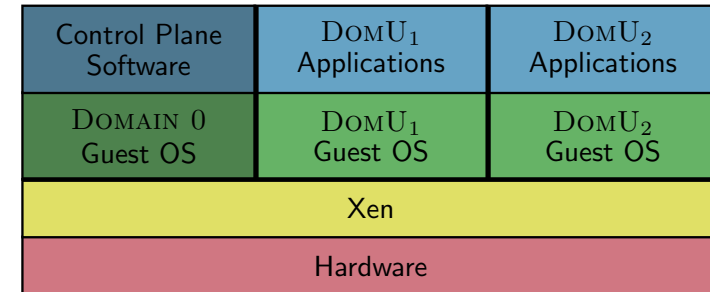
## Literatur

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris et al.  
**Xen and the art of virtualization**  
*Proc. of the 19th Symposium on Operating Systems Principles (SOSP '03)*,  
S. 164–177, 2003.

David Chisnall  
**The definitive guide to the Xen hypervisor**  
Prentice Hall, 2007.

# Architektur

- Privilegierte Domäne (*Domain 0, Dom0*)
  - Beim Systemstart von Xen automatisch erzeugt
  - Zugriff auf die Kontrollschnittstelle zur Verwaltung von Gastdomänen
    - Starten und Stoppen
    - Konfiguration von VM-Scheduling, Speicherzuteilung, Netzwerkzugriff,...
- Gastdomänen (*DomU\**)
  - Nichtprivilegierte virtuelle Maschinen mit eigenem Betriebssystem
  - Ausführung von Nutzeranwendungen



# Kommunikation zwischen Xen und einer Domäne

- Domäne → Xen
  - Anwendung: Systemaufrufe
    1. Aufrufparameter per PUSH auf den Stack legen
    2. Interrupt 80h auslösen
    3. Behandlung des Interrupt erfolgt in Xen
    4. Xen leitet den Aufruf zur Bearbeitung an das Gastbetriebssystem weiter
  - Gastbetriebssystem: Hypercalls
    1. Aufrufparameter in dedizierten Registern ablegen
    2. Interrupt 82h auslösen
    3. Behandlung des Interrupt erfolgt in Xen
    4. Bearbeitung des Hypercall

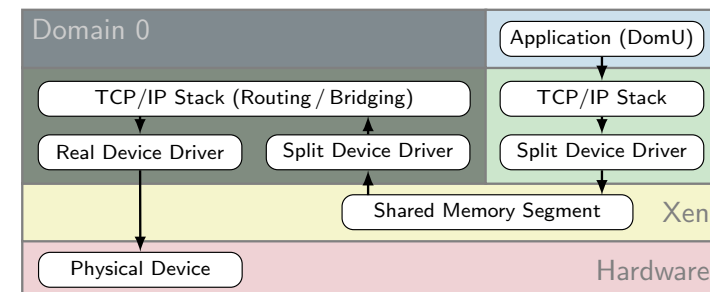
[Hinweis: Es existieren darüber hinaus weitere Varianten für die Implementierung von Hypercalls.]

## Xen → Domäne

- Setzen eines Flag in einer Ereignis-Bitmaske der Domäne
- Aufruf eines von der Domäne zuvor registrierten Event-Handler
- Beispiele: Geräte-Interrupts, Aufforderung an Domäne sich zu beenden

# Verwaltung von Geräten

- Ansatz
  - Xen stellt von sich aus keine Treiber bereit
  - Einsatz der Treiber des Betriebssystems der Domain 0
  - Indirekter Hardware-Zugriff über privilegierte Domäne
- Beispiel: Senden eines Netzwerkpakets aus einer Gastapplikation
  - Datenaustausch zwischen Domänen erfolgt per Shared Memory
  - Aufgaben der Domain 0
    - Multiplexen der Hardware für Zugriff mehrerer Gastdomänen
    - Anwendung von Firewall-Regeln



- Scheduling
  - Bereitstellung von *virtuellen Prozessoren (VCPUs)* für Gastdomänen
  - Domäneninterne Ablaufplanung durch Scheduler des Gastbetriebssystems
  - Xen-Scheduler: Dynamische Abbildung von VCPUs auf reale Prozessoren
  - Ziel: Performanzisolation zwischen virtuellen Maschinen
  - Beispiel: Credit Scheduler
    - *Weight*: Relative Gewichte für Gastdomänen → Anteil an CPU-Zeit
    - *Cap*: Maximaler Anteil einer Gastdomäne an der verfügbaren CPU-Zeit
- Timer
  - *Realzeit*: Zeit seit dem Systemstart
  - *Virtuelle Zeit*: Schreitet nur voran wenn eine Domäne ausgeführt wird
  - *"Wall-Clock"-Zeit*: An die Realzeit gekoppelte Uhr einer Gastdomäne
- Festplattenzugriff
  - Zugriff über *Virtual Block Devices*
  - Datenaustausch per Shared Memory

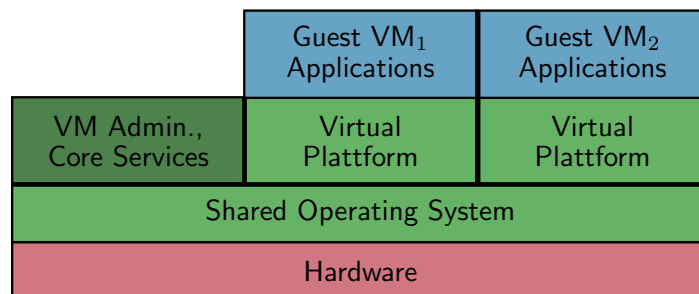


- Zielsetzungen
  - Virtualisierung für Szenarien mit abgeschwächten Isolationsanforderungen
  - Reduzierte Flexibilität zugunsten erhöhter Effizienz
- Container-basiertes Betriebssystem
  - Virtualisierung auf Ebene des *Application Binary Interface (ABI)*
  - Alle virtuellen Maschinen nutzen den selben Betriebssystemkern
  - Standard-Linux mit Erweiterungen
- Einsatz (Beispiele)
  - PlanetLab
  - High-Performance-Cluster
- Literatur
  - Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier et al. **Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors**  
*Proc. of the 2nd European Conference on Computer Systems (EuroSys '07)*, pages 275–287, 2007.



## Architektur

- Partitionierung von Ressourcen
  - Jede Partition stellt einen eigenen *Security Context* dar
  - VM: Gruppe von Prozessen, die dem selben Kontext zugeordnet sind
  - Isolation verschiedener Kontexte voneinander
- Virtuelle Maschinen
  - VM für Administrations- und Verwaltungsaufgaben (*Host VM*)
  - VMs für Ausführung von Nutzerapplikationen (*Guest VMs*)



## Nutzung bereits in Linux bestehender Mechanismen

- *Linux Capabilities*
  - Prozessspezifische, feingranulare Rechteverwaltung auf Operationsebene
  - Beispiele
    - `CAP_SYS_MODULE`: Hinzufügen und Entfernen von Kernel-Modulen
    - `CAP_SYS_NICE`: Modifikation der Prioritäten anderer Prozesse
- Ressourcenlimits
  - Festlegung von *Hard-* und *Soft-Limits* pro Prozess
  - Beispiel: Maximale Dauer, die ein Prozess die CPU nutzen darf
- Erweiterte Dateiattribute (z. B. `IMMUTABLE`: Schutz vor Modifikationen)
- `chroot`: Ändern des Wurzelverzeichnis eines Dateisystems
- Literatur
  - Herbert Pötzl **Linux-VServer Paper**  
<http://linux-vserver.org/Paper>



- Prozessverwaltung
  - Global eindeutige Prozess-IDs (PIDs)
  - Erweiterung von Kernel-Strukturen für Zuordnung von PIDs zu VMs
  - Filter zur Trennung von Prozessen verschiedener VMs
  - Pseudo-init-Prozesse mit jeweils PID 1
- Prozesseinplanung
  - Standard-Linux-Scheduler kombiniert mit *Token Bucket Filter*
  - Funktionsweise
    - Jeder virtuellen Maschine wird ein Token-Bucket zugeordnet
    - Jeder Token-Bucket wird mit einer individuellen Rate befüllt
    - Das Token-Kontingent einer aktiven VM wird schrittweise reduziert
    - Solange Tokens verfügbar sind, ist die korrespondierende VM lauffähig
  - Erweiterungen
    - *Reservations*: Garantierter CPU-Anteil
    - *Shares*: Anteil an der nichtreservierten CPU-Zeit



- Netzwerk
  - Netzwerkzugriffe
    - Anhängen der VM-Kontext-ID an die Netzwerkpakete einer VM
      - Zuordnung zu virtuellen Maschinen möglich
    - Einplanung ähnlich wie bei Prozessen (→ Reservierungen und Anteile)
  - Gemeinsame Nutzung von Routing-Tabellen
  - Zuweisung von Netzwerkadressen zu VMs
  - Spezielle Behandlung der Adresse localhost erforderlich
- Festplattenzugriff
  - Einplanung von Zugriffen
    - Einsatz des Standard-Linux-I/O-Scheduler (*Completely Fair Queuing*)
    - Gleichmäßige Aufteilung der Datenrate auf alle zugreifenden VMs
  - Festlegung einer maximalen Anzahl von Blöcken bzw. Inodes pro VM
- Dateisystem
  - Geteiltes Dateisystem für sich selten ändernde Dateien (z. B. Bibliotheken)
  - Copy-on-Write-Ansatz bei Modifikation: Erzeugung einer privaten Kopie



- Fehlerisolation
  - Fähigkeit den Einfluss einer fehlerhaften VM zu beschränken
  - Beide: Isolation zwischen VMs durch getrennte Adressräume
  - Von VMs geteilte Komponenten
    - Xen: Virtual Machine Monitor und Betriebssystem der Domain 0
    - Linux-VServer: Betriebssystem
  - Schnittstellen zwischen VMM und VMs
    - Xen: Zugriff auf virtuelle Geräte, Signalisierung von Ereignissen
    - Linux-VServer: Bereitstellung einer vollständigen ABI
- Ressourcenisolation
  - Fähigkeit den Ressourcenverbrauch einer VM zu überwachen/beschränken
  - Beide: Mechanismen für Reservierung und Zuteilung von Ressourcen
- Sicherheitsisolation
  - Umfang der Schutzmechanismen für vertrauliche Informationen (z. B. virtuelle Speicheradressen, Ports, Nutzer- und Prozess-IDs)
  - Beide: Abhängig vom Grad der zwischen VMs geteilten Komponenten



- Anforderungen an ein virtualisiertes System
  - Äquivalenz
  - Ressourcenkontrolle
  - Effizienz
- Virtualisierungstechniken
  - Trap-and-Emulate
  - Binary Translation
- Virtualisierungsebenen
  - Systemvirtualisierung
    - Hardware-unterstützte Virtualisierung
    - Paravirtualisierung
    - Beispiel: Xen
  - Prozessvirtualisierung
    - Betriebssystemvirtualisierung
    - Beispiel: Linux-VServer

