

# 10. Foliensatz Betriebssysteme

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences  
(1971–2014: Fachhochschule Frankfurt am Main)  
Fachbereich Informatik und Ingenieurwissenschaften  
[christianbaun@fb2.fra-uas.de](mailto:christianbaun@fb2.fra-uas.de)

# Lernziele dieses Foliensatzes

- Am Ende dieses Foliensatzes kennen/verstehen Sie...
  - was **Virtualisierung** ist, ihre Vorteile, Nachteile und Grenzen
  - den Unterschied zwischen **Hardware-Emulation** und **Virtualisierung**
  - verschiedene **Virtualisierungskonzepte**:
    - **Partitionierung**
    - **Anwendungsvirtualisierung**
    - **Vollständige Virtualisierung** (Virtueller Maschinen-Monitor)
    - **Paravirtualisierung** (Hypervisor)
    - **Hardware-Virtualisierung**
    - **Betriebssystem-Virtualisierung bzw. Container bzw. Jails**
    - **Speichervirtualisierung (SAN)**
    - **Netzwerkvirtualisierung (VLAN)**

Übungsblatt 10 wiederholt die für die Lernziele relevanten Inhalte dieses Foliensatzes

# Virtualisierung – Grundlagen

- Durch **Virtualisierung** werden die Ressourcen eines Rechnersystems aufgeteilt und von mehreren unabhängigen Betriebssystem-Instanzen genutzt
- Virtualisierung ist stellvertretend für mehrere grundsätzlich verschiedene Konzepte und Technologien
- Jede **Virtuelle Maschine** (VM)...
  - verhält sich wie ein vollwertiger Computer mit eigenen Komponenten
  - läuft in einer abgeschotteten Umgebung auf einer physischen Maschine
- In einer VM kann ein Betriebssystem mit Anwendungen genau wie auf einem realen Computer installiert werden
  - Die Anwendungen merken nicht, dass sie sich in einer VM befinden
- Anforderungen der Betriebssystem-Instanzen werden von der Virtualisierungssoftware abgefangen und auf die real vorhandene oder emulierte Hardware umgesetzt
  - Die VM selbst bekommt davon auch nichts mit

# Ursprung der Virtualisierung

- Virtualisierung ist kein neues Konzept
  - Einführung in den 1960er-Jahren bei Großrechnern
- 1970/71: IBM stellt die Virtual Machine Facility/370, kurz VM/370 vor
  - Auf dieser Plattform wurde Mehrbenutzerbetrieb realisiert, indem mehrere Einzelbenutzerbetriebinstanzen in virtuellen Maschinen ausführten wurden
  - Jede VM stellte eine vollständige Nachbildung der darunter liegenden, physischen Hardware dar

## Quellen

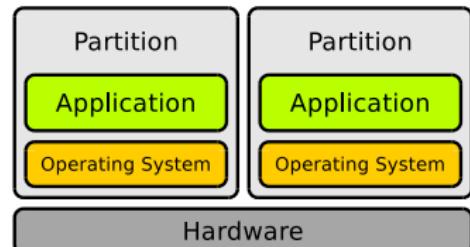
- Creasy RJ. **The origin of the VM/370 time-sharing system.**  
IBM Journal of Research and Development 25 (1981), No. 5, 483–490
- Amit Singh. **An Introduction to Virtualization.** 2004  
<http://www.kernelthread.com/publications/virtualization/>

# Virtualisierungskonzepte

- Es existieren Unterschiedliche Virtualisierungskonzepte:
    - Partitionierung
    - Hardware-Emulation
    - Anwendungsvirtualisierung
    - Vollständige Virtualisierung (Virtueller Maschinen-Monitor)
    - Paravirtualisierung (Hypervisor)
    - Hardware-Virtualisierung
    - Betriebssystem-Virtualisierung bzw. Container bzw. Jails
    - Speichervirtualisierung (SAN)
    - Netzwerkvirtualisierung (VLAN)
    - ...

# Partitionierung

- Bei Partitionierung können auf den Gesamtressourcen eines Computersystems Teilsysteme definiert werden
  - Jedes Teilsystem kann eine lauffähige Betriebssysteminstanz enthalten
  - Jedes Teilsystem ist wie ein eigenständiges Computersystem verwendbar
- Die Ressourcen (Prozessor, Hauptspeicher, Datenspeicher...) werden über die **Firmware** des Rechners verwaltet und den VMs zugeteilt
- Partitionierung kommt z.B. bei IBM Großrechnern (zSerie) oder Midrange-Systemen (pSerie) mit Power5/6/7 Prozessoren zum Einsatz
  - Ressourcenzuteilung ist im laufenden Betrieb ohne Neustart möglich
  - Auf einem aktuellen Großrechner können mehrere hundert bis tausend Linux-Instanzen gleichzeitig laufen
- Aktuelle CPUs unterstützen lediglich die Partitionierung der CPU selbst und nicht des Gesamtsystems (Intel Vanderpool, AMD Pacifica)
  - Partitionierung spielt im Desktop-Umfeld keine Rolle



# Beispiel für Partitionierung – Watson (1/2)

- Sieger beim US-Quiz *Jeopardy Challenge* im Februar 2011 war *Watson*
  - Watson ist ein Cluster aus 90 IBM Power 750 Servern mit 2.880 Power7 CPU-Kernen (je 8 Kerne pro CPU) und 16 TB RAM



Bildquelle (Watson stage replica in Jeopardy! contest, Mountain View, California): Atomic Taco. [flickr.com](https://flickr.com/photos/atomic_taco/3800000000/) (CC-BY-SA-2.0)

Bildquelle (Interns demonstrating Watson capabilities in Jeopardy! exhibition match): Rosemaryetoufee. Wikimedia (CC-BY-SA-4.0)

## Beispiel für Partitionierung – Watson (2/2)

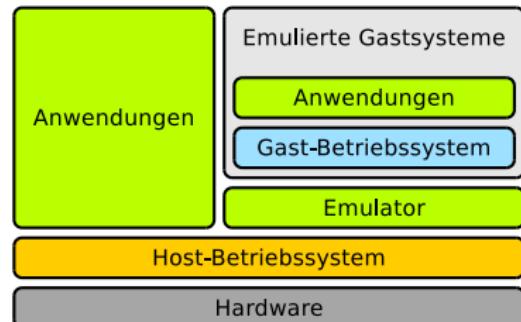
- Auf den 90 Knoten können Partitionen erstellt werden
  - In jeder Partition kann ein AIX, Linux oder IBM i (früher OS/400) laufen
  - Die Partitionen sind unabhängige Installationen
    - In jeder Partition kann ein unterschiedliches Betriebssystem laufen
- Auf jedem Knoten läuft ein *POWER Hypervisor*
  - Er regelt den Hardwarezugriff
- Seit Power6 kann man laufende Partitionen ohne Unterbrechung auf andere physische Server umziehen (⇒ Live Partition Mobility)
- Man kann Partitionen auch erlauben, dass sie sich Hauptspeicher teilen (⇒ Active Memory Sharing)
  - Active Memory Expansion kann Speicherseiten komprimieren
    - Je nach Anwendung geht das schneller als Verschieben oder Auslagern

# Hardware-Emulation

- Emulation bildet die **komplette Hardware** eines Rechnersystems nach, um ein **unverändertes Betriebssystem**, das für eine **andere Hardwarearchitektur** (CPU) ausgelegt ist, zu betreiben

- Ausnahme: Wine
  - Wine emuliert keine Hardware, sondern nur die Schnittstellen des Betriebssystems Windows

- Nachteile der Emulation:
  - Entwicklung ist sehr aufwendig
  - Ausführungsgeschwindigkeit ist geringer als bei Virtualisierung
- Wichtige Unterscheidung: **Emulation  $\neq$  Virtualisierung**
- Einige Emulatoren: Bochs, QEMU, PearPC, Wabi, DOSBox, Microsoft Virtual PC (ist in der Version für MacOS X/PowerPC ein x86-Emulator)



# Auswahl an Emulatoren

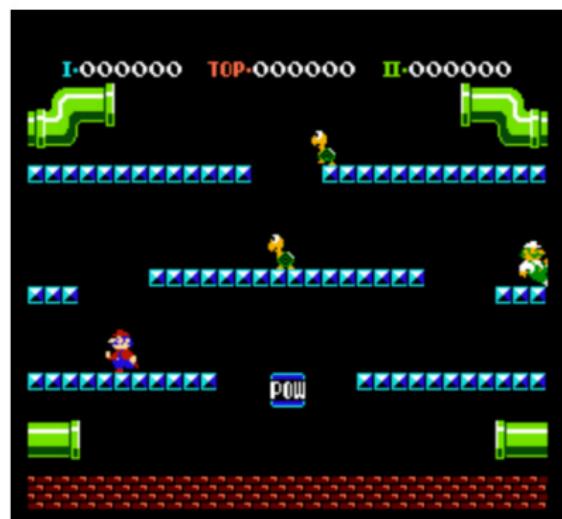
Name	Lizenz	Host	Emulierte Architektur	Gast-System
Bochs v2.3.6	LGPL	Linux, Solaris, MacOS, Windows, IRIX, BeOS	x86, AMD64	Linux, DOS, BSD, Windows, BeOS
QEMU v0.9.0	GPL	Linux, BSD, Solaris, BeOS, MacOS-X	x86, AMD64, PowerPC, ARM, MIPS, Sparc	Linux, MacOS-X, Windows, BSD
DOSBox v0.72	GPL	Linux, Windows, OS/2, BSD, BeOS, MacOS-X	x86	DOS
DOSEMU v1.4.0	GPL	Linux	x86	DOS, Windows bis 3.11
PearPC v0.4.0	GPL	Linux, MacOS-X, Windows	PowerPC	Linux, MacOS-X, BSD
Basilisk II v0.9-1	GPL	Linux, diverse UNIX, Windows NT4, BeOS, Mac OS, Amiga OS	680x0	MacOS $\leq$ 8.1
Wabi v2.2	proprietär	Linux, Solaris	x86	Windows 3.x
MS Virtual PC v7	proprietär	MacOS-X	x86	Windows, (Linux)
M.A.M.E. v0.137	MAME-Lizenz	Linux, Windows, DOS, BeOS, BSD, OS/2	diverse Arcade	diverse Arcade
SheepShaver	GPL	Linux, MacOS-X, BSD, Windows, BeOS	PowerPC, 680x0	MacOS 7.5.2 bis MacOS 9.0.4
Hercules 3.07	QPL	Linux, MacOS-X, BSD, Solaris, Windows	IBM-Großrechner	IBM System/360, 370, 390

- Die Tabelle erhebt keinen Anspruch auf Vollständigkeit!

# Beispiel für einen aktuellen Emulator – JSNES

- JSNES emuliert das Nintendo Entertainment System (NES)
- Der Emulator ist in JavaScript implementiert und läuft im Browser
- <http://fir.sh/projects/jsnes/>
- [github.com/bfirsh/jsnes](https://github.com/bfirsh/jsnes)
- Freie Software (GPLv3)

Ben Firshman  
JSNES



Mario Bros. ▾

pause restart enable sound zoom out

Running: 44.40 FPS

# Aktuellste Entwicklung: Browser emuliert PC – jslinux

c't 13/2011

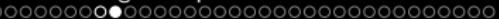
Der PC-Emulator von Fabrice Bellard läuft dank vollständiger Umsetzung in JavaScript in Web-Browsern wie Firefox 4 und Chrome 11. Während des erstaunlich flotten Bootvorgangs startet die Software ein eigens abgespecktes Linux (JS/Linux mit Kernel 2.6.20). (...)

Da das Linux im Emulator in einer RAM-Disk startet, lässt sich auch ausprobieren, was passiert, wenn man beispielsweise das gesamte Dateisystem via `rm -Rv` löscht. Ein Reload der Webseite startet den Emulator anschließend neu und stellt den Ursprungszustand wieder her.

Die als private Studie entstandene Java-Script-Software emuliert weitgehend einen 486er 32-Bit-Prozessor ohne FPU (Floating Point Unit), einen Interrupt-Controller 8259, einen Interrupt-Timer 8254 und eine ungebufferte serielle Schnittstelle vom Typ 16450. Die fehlende FPU kompensiert Linux durch seinen eingebauten FPU-Emulator. Für die Umsetzung in JavaScript nutzt Bellard Typed Arrays, die die JavaScript-Engines von Firefox 4 und Chrome 11 beherrschen. (rek)

```
TCP reno registered
checking if image is initramfs...it isn't (bad gzip magic numbers); looks like a
n initrd
Freeing initrd memory: 2048K freed
Total HugeTLB memory allocated, 0
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered (default)
Real Time Clock Driver v1.12ac
JS clipboard: I/O at 0x03c0
Serial: 8250/16550 driver $Revision: 1.90 $ 4 ports, IRQ sharing disabled
serial1650: ttys1 at I/O 0x3f8 (irq = 4) is a 16450
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
loop: loaded (max 8 devices)
TCP cubic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
Using IPI Shortcut mode
Time: pit clocksource has been installed.
RAMDISK: ext2 filesystem found at block 0
RAMDISK: Loading 2048KiB [1 disk] into ram disk... done.
EXT2-fs warning: maximal mount count reached, running e2fsck is recommended
VFS: Mounted root (ext2 filesystem).
Freeing unused kernel memory: 124k freed
Booted in 8.961 s
Welcome to JS/Linux
- # uname -a
Linux (none) 2.6.20 #2 Mon Aug 8 23:51:02 CEST 2011 i586 GNU/Linux
- #
© 2011 Fabrice Bellard : News · FAQ · Technical notes
```

Image Source: <http://bellard.org/jslinux/>



## JSLinux

Run Linux or other Operating Systems in your browser!

The following emulated systems are available:

CPU	OS	User Interface	VSync access	Startup Link	TEMU Config	Comment
x86	Alpine Linux 3.12.0	Console	Yes	<a href="#">click here</a>	<a href="#">url</a>	
x86	Alpine Linux 3.12.0	X Window	Yes	<a href="#">click here</a>	<a href="#">url</a>	Right mouse button for the menu.
x86	Windows 2000	Graphical	No	<a href="#">click here</a>	<a href="#">url</a>	<a href="#">Dioclites</a> .
x86	FreeDOS	VGA Text	No	<a href="#">click here</a>	<a href="#">url</a>	
riscv64	Buildroot (Linux)	Console	Yes	<a href="#">click here</a>	<a href="#">url</a>	
riscv64	Buildroot (Linux)	X Window	Yes	<a href="#">click here</a>	<a href="#">url</a>	Right mouse button for the menu.
riscv64	Fedora 29 (Linux)	Console	Yes	<a href="#">click here</a>	<a href="#">url</a>	Warning: longer boot time.
riscv64	Fedora 29 (Linux)	X Window	Yes	<a href="#">click here</a>	<a href="#">url</a>	Warning: longer boot time. Right mouse button for the menu.

© 2011-2020 Fabrice Bellard - [News](#) - [VM list](#) - [FAQ](#) - [Technical notes](#)

Loading...

Welcome to JS/Linux (i586)

Use 'vlogin username' to connect to your account.  
 You can create a new account at <https://vfsync.org/signup>.  
 Use 'export file filename' to export a file to your computer.  
 Imported files are written to the home directory.

```
localhost:~# uname -a
Linux localhost 4.12.0-rc6-g48eclf0-dirty #21 Fri Aug 4 21:02:28 CEST 2017 i586
Linux
localhost:~#
```

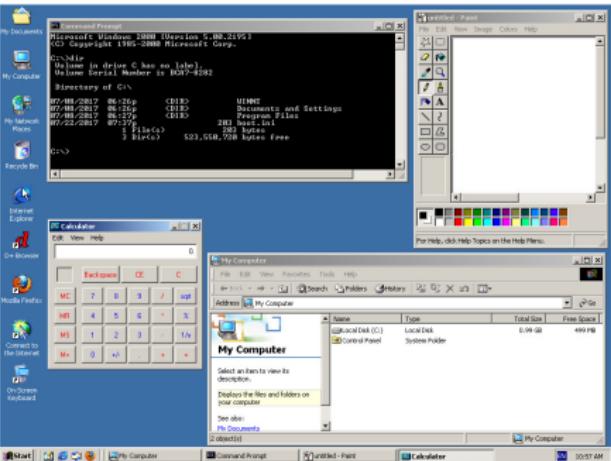
Type HELP to get support on commands and navigation.

Welcome to the **FreeDOS 1.2** operating system (<http://www.freedos.org>)

Use KEYB to set the keyboard mapping (e.g. KEYB FR for a French keyboard)

C:\>mem

Memory Type	Total	Used	Free
conventional	639K	33K	606K
Upper	0K	0K	0K
Reserved	385K	385K	0K
Extended (XMS)	64,512K	328K	64,184K
Total memory	65,536K	746K	64,790K
Total under 1 MB	639K	33K	606K
Total Expanded (EMS)		8,680K (8,896,512 bytes)	
Free Expanded (EMS)		8,192K (8,388,608 bytes)	
Largest executable program size		606K (620,000 bytes)	
FreeDOS is resident in the high memory area.			



- Seit 2011 hat der Autor von JSLinux den Funktionsumfang stark erweitert

Bild oben rechts: FreeDOS 1.2 (x86)

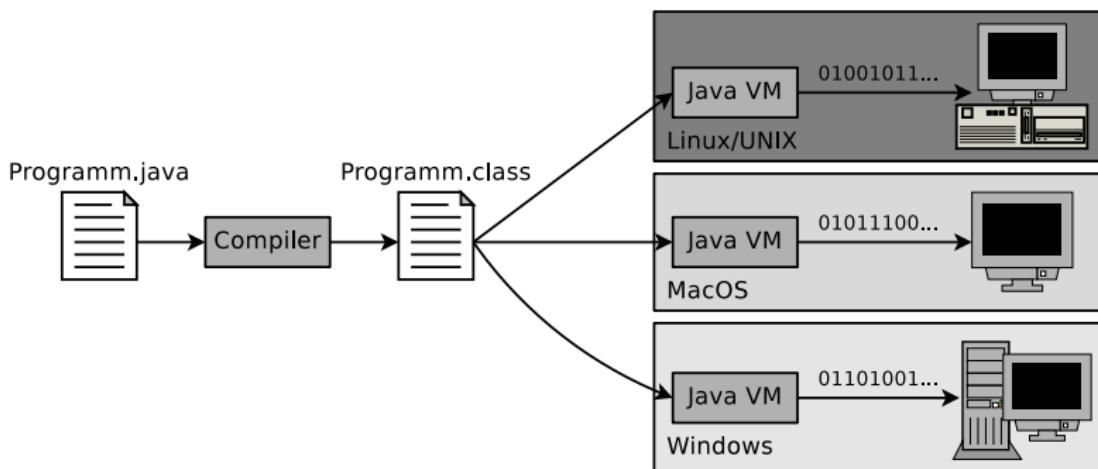
Bild unten links: Alpine Linux 3.12.0 (x86)

Bild unten rechts: Windows 2000 (x86)

# Anwendungsvirtualisierung

- Anwendungen werden unter Verwendung lokaler Ressourcen in einer virtuellen Umgebung ausgeführt, die alle Komponenten bereitstellt, die die Anwendung benötigt
  - VM befindet sich zwischen der auszuführenden Anwendung und dem Betriebssystem
- Populäres Beispiel: Java Virtual Machine (JVM)
  - Die JVM ist der Teil der Java-Laufzeitumgebung (JRE), der für die Ausführung des Java-Bytecodes verantwortlich ist
  - Die JVM ist für Java-Programme die Schnittstelle zum Rechnersystem und dessen Betriebssystem
- Vorteil: Plattformunabhängigkeit
- Nachteil: Geringere Ausführungsgeschwindigkeit gegenüber nativer Programmausführung

# Prinzip der Java Virtual Machine (JVM)



- Der Compiler `javac` übersetzt Quellcode in architektur-unabhängige `.class`-Dateien, die Bytecode enthalten, der in der Java VM lauffähig ist
- Das `java`-Programm startet eine Java-Anwendung in einer Instanz der Java VM

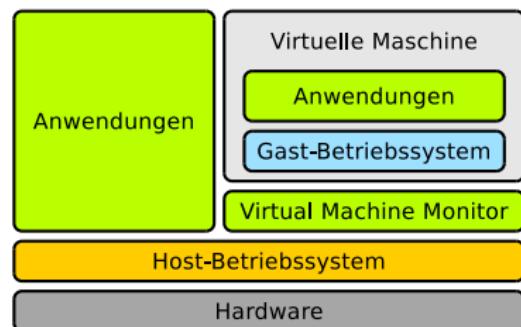
# VMware ThinApp

<http://www.vmware.com/products/thinapp/>

- Weiteres Beispiel für Anwendungsvirtualisierung: VMware ThinApp
  - Bis 2008 unter dem Namen Thinstall bekannt
- Eine Windows-Anwendung wird in eine einzelne .exe-Datei gepackt
- Die Anwendung wird dadurch portabel und kann ohne lokale Installation verwendet werden
  - Die Anwendung kann u.a. auf einem USB-Stick ausgeführt werden
- Es erfolgen keine Einträge in der Windows Registry. Es werden auch keine Umgebungsvariablen und DLL-Dateien auf dem System erstellt
- Benutzereinstellungen und erstellte Dokumente werden in einer eigenen Sandbox gespeichert
- Nachteil: Funktioniert ausschließlich mit Microsoft Windows

# Vollständige Virtualisierung (1/3)

- Vollständige Virtualisierungslösungen bieten einer VM eine vollständige, virtuelle PC-Umgebung inklusive eigenem BIOS
  - Jedes Gastbetriebssystem erhält eine eigene VM mit virtuellen Ressourcen (u.a. CPU, Hauptspeicher, Laufwerken, Netzwerkkarten)
- Es kommt ein **Virtueller Maschinen-Monitor** (VMM) zum Einsatz
  - Der VMM heißt auch **Typ-2-Hypervisor**
  - Der VMM läuft *hosted* als Anwendung im Host-Betriebssystem
  - Der VMM verteilt Hardwareressourcen an VMs
- Teilweise emuliert der VMM Hardware, die nicht für den gleichzeitigen Zugriff mehrerer Betriebssysteme ausgelegt ist
  - Ein Beispiel sind Netzwerkkarten
  - Die Emulation populärer Hardware vermeidet Treiberprobleme



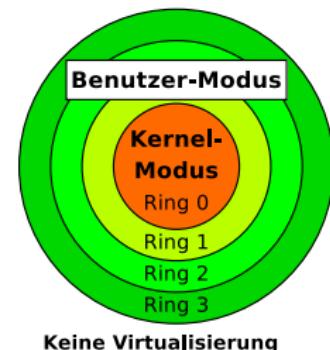
# Virtualisierungsgrundlagen der x86-Architektur (1/2)

- x86-kompatible CPUs enthalten 4 Privilegienstufen
  - Ziel: Stabilität und Sicherheit verbessern
  - Jeder Prozess wird in einem Ring ausgeführt und kann sich nicht selbstständig aus diesem befreien

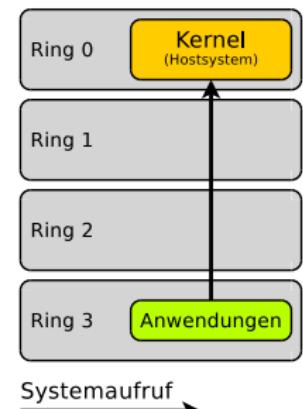
## Realisierung der Privilegienstufen

- Das Register CPL (Current Privilege Level) speichert die aktuelle Privilegienstufe
- Quelle: Intel 80386 Programmer's Reference Manual 1986  
<http://css.csail.mit.edu/6.858/2012/readings/i386.pdf>

- In Ring 0 (= **Kernelmodus**) läuft der Betriebssystemkern
  - Hier haben Prozesse vollen Hardwarezugriff
  - Der Kern kann physischen Speicher ( $\Rightarrow$  Real Mode) adressieren
- In Ring 3 (= **Benutzermodus**) laufen die Anwendungen
  - Hier arbeiten Prozesse nur mit virtuellem Speicher



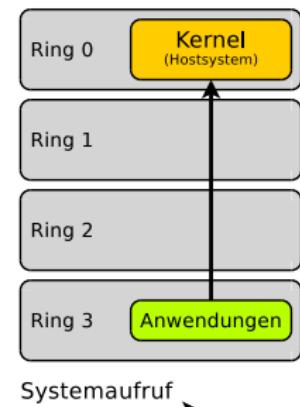
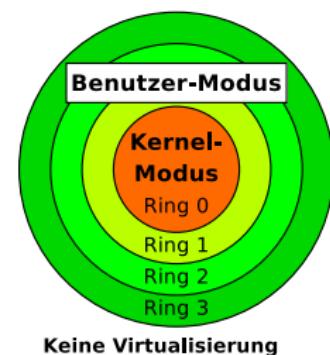
Keine Virtualisierung



# Virtualisierungsgrundlagen der x86-Architektur (2/2)

Moderne Betriebssysteme verwenden nur 2 Privilegienstufen (Ringe)

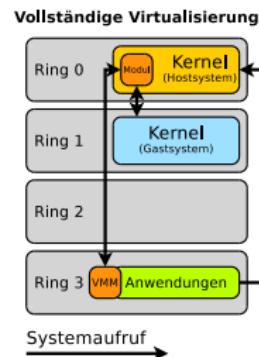
- Grund: Einige Hardware-Architekturen (z.B. Alpha, PowerPC, MIPS) unterstützen nur 2 Stufen
  - Ausnahme: OS/2 nutzt Ring 2 für Anwendungen, die auf Hardware und Eingabe-/Ausgabeschnittstellen zugreifen dürfen (z.B. Grafiktreiber)
- 
- Muss ein Prozess im Benutzermodus eine höher privilegierte Aufgabe erfüllen (z.B. Zugriff auf Hardware), kann er das dem Kernel durch einen **Systemaufruf** (⇒ Foliensatz 7) mitteilen
    - Der Prozess im Benutzermodus erzeugt eine Exception, die in Ring 1 abgefangen und dort behandelt wird



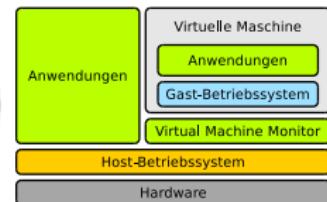
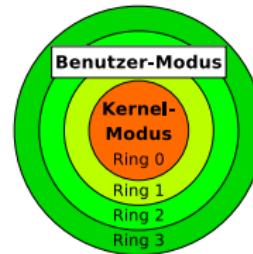
## Vollständige Virtualisierung (2/3)

- Vollständige Virtualisierung nutzt die Tatsache, dass x86-Systeme meist nur 2 Privilegienstufen verwenden

- Der VMM läuft wie die Anwendungen in Ring 3
  - VMs befinden sich im weniger privilegierten Ring 1



- Der VMM enthält für jede Ausnahme eine Behandlung, die privilegierte Operationen der Gastbetriebssysteme abfängt, interpretiert und ausführt
- VMs erhalten nur über den VMM Zugriff auf die Hardware
  - Garantiert kontrollierten Zugriff auf gemeinsam genutzte Systemressourcen



## Vollständige Virtualisierung (3/3)

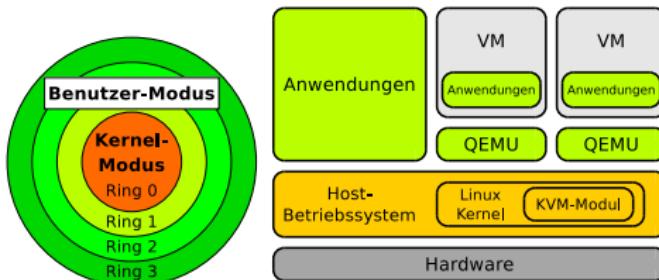
- Vorteile:
  - Kaum Änderungen an Host- und Gast-Betriebssystemen erforderlich
  - Zugriff auf die wichtigsten Ressourcen wird nur durchgereicht
    - ⇒ Fast native Verarbeitungsgeschwindigkeit der Gast-Betriebssysteme
  - Jedes Gast-Betriebssystem hat seinen eigenen Kernel
    - ⇒ Hohe Flexibilität
- Nachteile:
  - Wechsel zwischen den Ringen erfordern einen Prozesswechsel
    - ⇒ Jeder Prozesswechsel verbraucht Rechenzeit
  - Fordert eine Anwendung im Gast-Betriebssystem die Ausführung eines privilegierten Befehls an, liefert der VMM eine Ersatzfunktion und diese weist die Ausführung des Befehls über die Kernel-API des Host-Betriebssystems an
    - ⇒ Geschwindigkeitseinbußen

# Beispiele für Vollständige Virtualisierung

- Beispiele für Virtualisierungslösungen, die auf dem Konzept des VMM basieren:
  - VMware Server, VMware Workstation und VMware Fusion
  - Microsoft Virtual PC (in der Version für x86)
  - Parallels Desktop und Parallels Workstation
  - VirtualBox
  - Kernel-based Virtual Machine (KVM)
  - Mac-on-Linux (MoL)

# Kernel-based Virtual Machine (KVM)

- KVM ist als Modul direkt im Linux-Kernel integriert
  - KVM-Basismodul: kvm.ko
  - Hardwarespezifische Module: kvm-intel.ko und kvm-amd.ko
- Nach dem Laden der Module arbeitet der Kernel selbst als Hypervisor
- KVM kann nur mit CPUs mit Hardwarevirtualisierung arbeiten
  - Dadurch braucht KVM weniger Quellcode als z.B. Xen
- Neben den Kernelmodulen enthält KVM den Emulator QEMU
  - KVM stellt keine virtuelle Hardware zur Verfügung. Das macht QEMU
    - CPU-Virtualisierung stellt der Prozessor bereit (Intel VT oder AMD-V)
    - Hauptspeicher und Blockspeicher wird durch KVM virtualisiert
    - E/A wird durch einen QEMU-Prozess pro Gastsystem virtualisiert

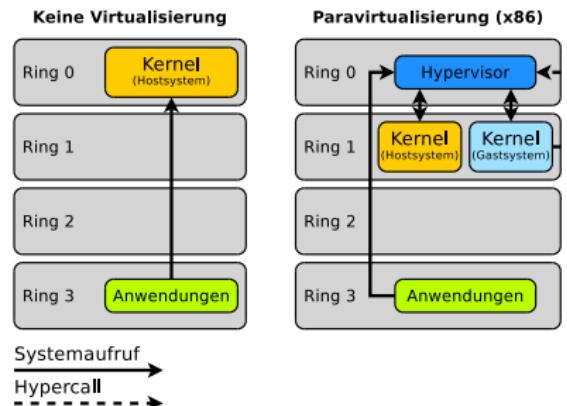


# Paravirtualisierung (1/4)

- Es wird keine Hardware virtualisiert oder emuliert
  - Gast-Betriebssystemen steht keine emulierte Hardwareebene, sondern eine API zur Verfügung
- Gast-Betriebssysteme verwenden eine abstrakte Verwaltungsschicht (⇒ **Hypervisor**), um auf physische Ressourcen zuzugreifen
  - Hypervisor ist ein auf ein Minimum reduziertes **Metabetriebssystem**
    - Der Hypervisor verteilt Hardwareressourcen unter den Gastsystemen, so wie ein Betriebssystem dieses unter den laufenden Prozessen tut
    - Der Hypervisor ist ein **Typ-1-Hypervisor** und läuft *bare metal*
  - Ein Metabetriebssystem ermöglicht den unabhängigen Betrieb unterschiedlicher Anwendungen und Betriebssysteme auf einer CPU
- Der Hypervisor läuft im privilegierten Ring 0
  - Das Host-Betriebssystem läuft im weniger privilegierten Ring 1
    - Ein Host-Betriebssystem ist wegen der Gerätetreiber nötig

## Paravirtualisierung (2/4)

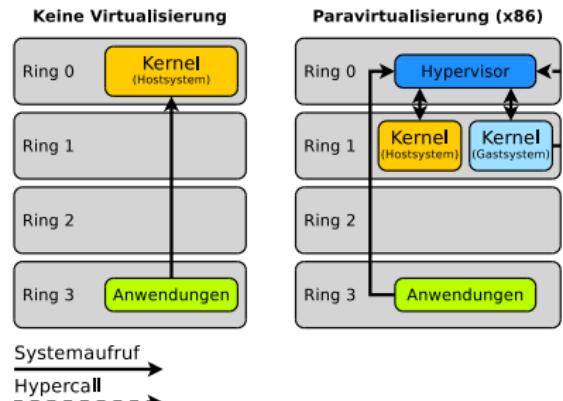
- Das Host-Betriebssystem läuft nicht mehr in Ring 0, sondern in Ring 1
  - Darum kann der Kernel keine privilegierten Anweisungen ausführen
  - Lösung: Der Hypervisor stellt **Hypercalls** zur Verfügung



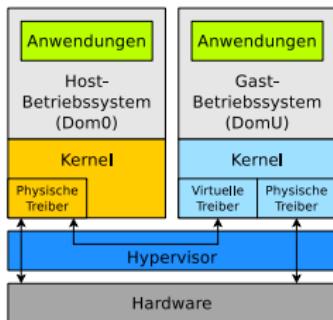
- Hypercalls sind vergleichbar mit Systemaufrufen (System Calls)
  - Die Interrupt-Nummern sind verschieden
  - Fordert eine Anwendung die Ausführung eines Systemaufrufs an, wird eine Ersatzfunktion im Hypervisor aufgerufen
  - Der Hypervisor weist die Ausführung des Systemaufrufs über die Kernel-API des Betriebssystems an

# Paravirtualisierung (3/4)

- In den Betriebssystemkernen der Gast-Betriebssysteme müssen alle Systemaufrufe für Hardware-Zugriffe durch die entsprechenden Hypercall-Aufrufe ersetzt werden
- Das Auffangen und Prüfen der Systemaufrufe durch den Hypervisor führt nur zu geringen Geschwindigkeitseinbußen
- Beispiele: Xen, Citrix Xenserver, Virtual Iron, VMware ESX Server



# Paravirtualisierung (4/4)



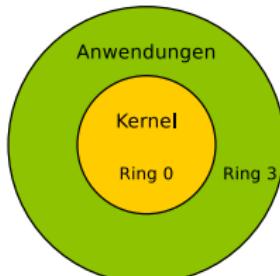
- VMs heißen **unprivilegierte Domain** (DomU)
- Der Hypervisor ersetzt das Host-Betriebssystem
  - Die Entwickler können aber nicht alle Treiber selbst schreiben und pflegen
    - Darum startet der Hypervisor eine (Linux-)Instanz mit ihren Treibern und leihst sich diese Treiber
    - Diese spezielle Instanz heißt Domain0 (Dom0)

- Nachteile:
  - Kernel der Gast-Betriebssysteme müssen für den Betrieb im paravirtualisierten Kontext angepasst sein
    - Rechteinhaber proprietärer Betriebssysteme lehnen eine Anpassung aus strategischen Gründen häufig ab  
⇒ Funktioniert häufig nur mit OpenSource-Betriebssystemen
- Vorteil:
  - Geschwindigkeitseinbußen, die beim VMM entstehen, werden vermieden

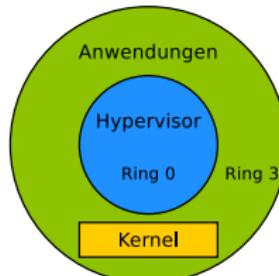
# Problem: x86-64-Architektur

- Die x86-64-Architektur (z.B. IA64) verzichtet auf die Ringe 1 und 2

Keine Virtualisierung



Paravirtualisierung (IA64)

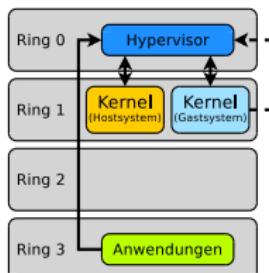


- Der Hypervisor befindet sich wie bei der x86-32-Architektur in Ring 0
- Der Betriebssystemkern wird bei der x86-64-Architektur in Ring 3 zu den Anwendungen verschoben

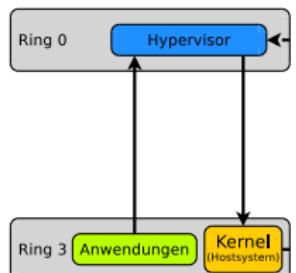
Keine Virtualisierung



Paravirtualisierung (x86)



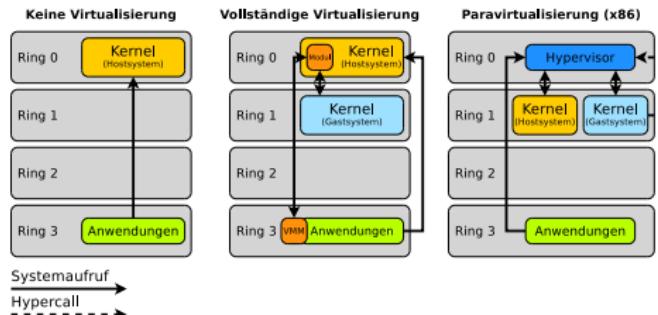
Paravirtualisierung (IA64)



- Der Betrieb der Hardwaredreiber und Anwendungen in einem Ring ist tendenziell unsicher

# Zusammenfassung: Voll- vs. Paravirtualisierung

- **Paravirtualisierung** erfordert angepasste Gastsysteme
  - Typ-1-Hypervisor läuft *bare metal* (= ersetzt das Host-Betriebssystem)
  - Hypervisor läuft in Ring 0 und hat vollen Zugriff auf die Hardware
  - Beispiele: VMware ESX(i), Xen, Microsoft Hyper-V
- **Vollvirtualisierung** ermöglicht unveränderte Gastsysteme
  - VMM (Typ-2-Hypervisor) läuft *hosted* als Anwendung im Host-Betriebssystem
  - VMM läuft in Ring 3 auf der Ebene der Anwendungen
  - Beispiele: VMware Workstation, KVM, Oracle VirtualBox, Parallels

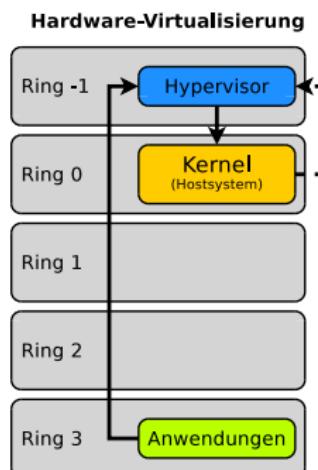


# Hardware-Virtualisierung (1/2)

- Aktuelle Prozessoren von Intel und AMD enthalten Erweiterungen um Hardware-Virtualisierung zu ermöglichen
    - Vorteil: Unveränderte Betriebssysteme können als Gast-Systeme ausgeführt werden
    - Die Lösungen von Intel und AMD sind ähnlich aber inkompatibel
  - Seit 2006 enthalten AMD64 CPUs den Secure-Virtual-Machine-Befehlssatz (**SVM**)
    - Die Lösung heißt **AMD-V** und war vorher als **Pacifica** bekannt
  - Die Lösung von Intel heißt **VT-x** für IA32-CPUs und **VT-i** für Itanium
    - Intels Lösung lief vormals unter dem Stichwort **Vanderpool**
- 
- Xen unterstützt ab Version 3 Hardware-Virtualisierung
  - Auch Windows Server 2008 (Hyper-V) nutzt Hardwarevirtualisierung
  - VirtualBox unterstützt Hardware-Virtualisierung
  - KVM kann nur mit CPUs mit Hardwarevirtualisierung arbeiten

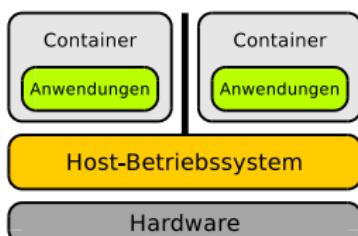
## Hardware-Virtualisierung (2/2)

- Kern der Neuerung ist eine Überarbeitung der Privilegienstruktur
- Ein neuer Ring ( $\Rightarrow$  Ring -1) für den Hypervisor ist hinzugefügt
  - Der Hypervisor bzw. VMM läuft im Ring -1 und besitzt jederzeit die volle Kontrolle über die CPU und die Ressourcen, da damit ein höheres Privileg als Ring 0 implementiert ist
- VMs laufen in Ring 0 und heißen HVM
  - HVM = Hardware Virtual Machine
- Vorteile:
  - Gastbetriebssysteme müssen nicht angepasst sein
    - Auch proprietäre Betriebssysteme (z.B. Windows) laufen als Gastsysteme
  - Der Kernel läuft nicht wie bei Paravirtualisierung (IA64) auf der Privilegienstufe der Anwendungen



# Betriebssystem-Virtualisierung / Container / Jails (1/2)

- Unter ein und demselben Kernel laufen mehrere voneinander abgeschottete identische Systemumgebungen
  - Es wird kein zusätzliches Betriebssystem gestartet
    - Es wird eine isolierte Laufzeitumgebung erzeugt
  - Alle laufenden Anwendungen verwenden denselben Kernel
    - Betriebssystem-Virtualisierung heißt **Container** in SUN/Oracle Solaris
    - Betriebssystem-Virtualisierung heißt **Jails** in BSD

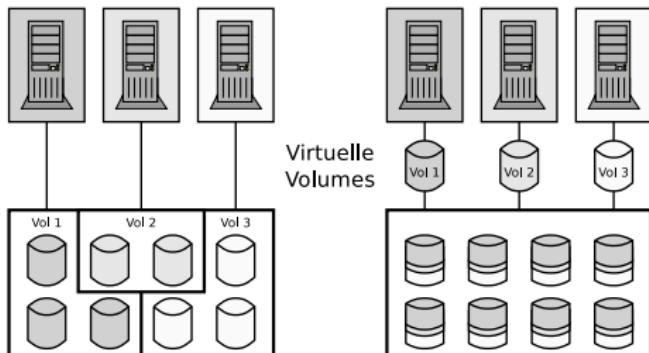


- Anwendungen sehen nur Anwendungen in der gleichen virtuellen Umgebung
  - Ein Vorteil ist der geringe Overhead, da der Kernel in gewohnter Weise die Hardware verwaltet
- Nachteil: Alle virtuellen Umgebungen nutzen den gleichen Kernel
    - Es werden nur unabhängige Instanzen eines Betriebssystems gestartet
    - Verschiedene Betriebssysteme können nicht gleichzeitig verwendet werden

# Betriebssystem-Virtualisierung / Container / Jails (2/2)

- Diese Art der Virtualisierung nutzt man, um Anwendungen in isolierten Umgebungen mit hoher Sicherheit zu betreiben
- Besonders Internet-Service-Provider, die (virtuelle) Root-Server oder Webdienste auf Mehrkernprozessorarchitekturen anbieten, nutzen diese Form der Virtualisierung
  - Wenig Performance-Verlust, hoher Grad an Sicherheit
- Beispiele:
  - SUN/Oracle Solaris (2005)
  - OpenVZ für Linux (2005)
  - Linux-VServer (2001)
  - FreeBSD Jails (1998)
  - Parallels Virtuozzo (2001, kommerzielle Variante von OpenVZ)
  - FreeVPS
  - Docker (2013)
  - chroot (1982)

# Speichervirtualisierung



- Speicher wird in Form virtueller Laufwerke (*Volumes*) den Benutzern zur Verfügung gestellt
- Logischer Speicher wird vom physischen Speicher getrennt

- Vorteile:
  - Nutzer sind nicht an die physischen Grenzen der Laufwerke gebunden
  - Physischen Speicher umstrukturieren/erweitern stört die Nutzer nicht
  - Redundantes Vorhalten erfolgt transparent im Hintergrund
  - Besserer Auslastungsgrad, da der physische Speicher effektiver auf die Benutzer aufgeteilt werden kann
- Nachteil: Professionelle Lösungen sind teuer
- Bekannte Anbieter: EMC, HP, IBM, LSI und SUN/Oracle

# Netzwerkvirtualisierung via Virtual Local Area Networks

- Verteilt aufgestellte Geräte können via VLAN in einem einzigen virtuellen (logischen) Netzwerk zusammengefasst werden
  - VLANs trennen physische Netze in logische Teilnetze (Overlay-Netze)
    - VLAN-fähige Switches leiten Pakete eines VLAN nicht in andere VLANs weiter
    - Ein VLAN ist ein nach außen isoliertes Netz über bestehende Netze
  - Zusammengehörende Geräte und Dienste in eigenen VLANs konsolidieren
    - Vorteil: Andere Netze werden nicht beeinflusst  
⇒ Höhere Sicherheit

## Gute einführende Quellen

Benjamin Benz, Lars Reimann. *Netze schützen mit VLANs*. 11.9.2006

<http://www.heise.de/netze/artikel/VLAN-Virtuelles-LAN-221621.html>

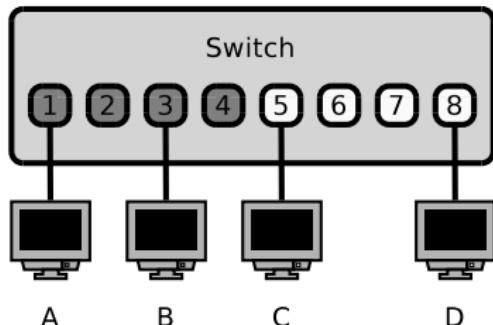
Stephan Mayer, Ernst Ahlers. *Netzsegmentierung per VLAN*. c't 24/2010. S.176-179

# Typen von VLANs

## ① Ältester Standard: **Statisches VLAN**

- Die Anschlüsse eines Switches werden in logische Switches unterteilt
- Jeder Anschluss ist fest einem VLAN zugeordnet oder verbindet unterschiedliche VLANs
- Schlecht automatisierbar

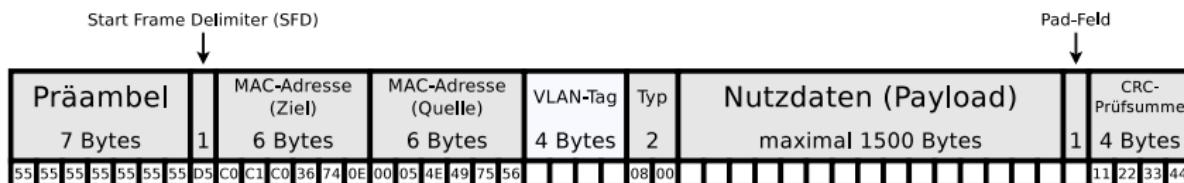
Nur Knoten A und B sowie Knoten C und D können miteinander kommunizieren, obwohl Sie mit dem gleichen Switch verbunden sind



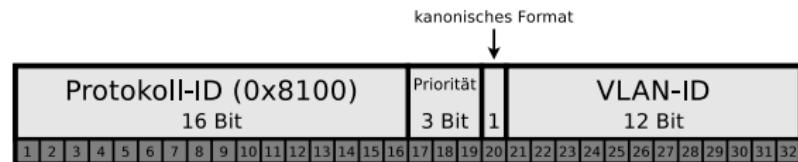
## ② Aktuell: **Paketbasiertes, dynamisches VLAN** nach IEEE 802.1Q

- Pakete der Vermittlungsschicht enthalten eine spezielle VLAN-Markierung (*Tag*)
- Dynamische VLANs können mit Hilfe von Skripten rein softwaremäßig erzeugt, verändert und entfernt werden

# Ethernet-Rahmen mit VLAN-Tag nach IEEE 802.1Q



- Länge der VLAN-Markierung: 32 Bits
- Die Protokoll-ID (16 Bits) hat immer den Wert 0x8100 für IPv4
- 3 Bits repräsentieren die Priorität (QoS)
  - 0 steht für die niedrigste und 7 für die höchste Priorität
  - Damit können bestimmte Daten (z.B. VoIP) priorisiert werden
- Kanonisches Format (1 Bit)  $\Rightarrow$  höchstwertiges Bit der MAC-Adressen
  - 0 = Ethernet, 1 = Token Ring
- 12 Bits enthalten die ID des VLAN, zu dem das Paket im Rahmen gehört



# Beispiele sinnvoller Einsatzgebiete von VLANs

- **Telekom Entertain**

- DSL-Anschluss mit Festnetzanschluss und IPTV ( $\Rightarrow$  *Triple Play*)
- Verwendet 2 VLANs, um den IPTV-Datenverkehr zu bevorzugen
  - „Normales“ Internet via PPPoE über VLAN ID 7
  - IPTV ohne Einwahl via VLAN ID 8

- **Eucalyptus**

- Private Cloud Infrastrukturdienst (IaaS)
- Jede Virtuelle Maschine (Instanz) ist einer Sicherheitsgruppe zugeordnet
  - Jede Sicherheitsgruppe hat eigene Firewall-Regeln
- Eucalyptus kann für jede Sicherheitsgruppe ein eigenes VLAN anlegen
  - Isolation des Datenverkehrs der Instanzen anhand der Sicherheitsgruppen

- **Rechenzentren oder auch Büro zuhause**

- Trennung des Datenverkehrs nach ökonomischen Gesichtspunkten
- Ziel: Absicherung vor Bedienfehlern und fehlerhafter Software
  - Ein VLAN als „Produktionsnetz“ mit den wichtigen Diensten
  - Zusätzliche VLANs für Experimente, Projektarbeit oder Spiele der Kinder

# Gründe für Virtualisierung (1/2)

- Bessere Ausnutzung der Hardware
  - Serverkonsolidierung: Zusammenlegen vieler (virtueller) Server auf weniger physischen Servern
    - Kostensenkung bei Hardware, Verbrauchskosten (Strom, Kühlung), Stellplätze, Administration, usw.
- Vereinfachte Administration
  - Anzahl physischer Server wird reduziert
  - Ausgereifte Managementwerkzeuge existieren
  - VMs können im laufenden Betrieb verschoben werden (Live Migration)
- Vereinfachte Bereitstellung
  - Neue Infrastrukturen und Server können innerhalb von Minuten manuell oder automatisch erzeugt werden

## Gründe für Virtualisierung (2/2)

- Maximale Flexibilität
  - VMs können leicht vervielfältigt und gesichert werden
  - Snapshots vom aktuellen Zustand einer VM können erzeugt und wieder hergestellt werden
- Höhere Sicherheit
  - VMs sind gegenüber anderen VMs und dem Host-System isoliert
  - Unternehmenskritische Anwendungen können in einer VM gekapselt und so in einer sicheren Umgebung laufen
  - Ausfall einer VM tangiert keine anderen VMs oder den Host
- Optimierung von Software-Tests und Software-Entwicklung
  - Gleichzeitiger Betrieb mehrerer Betriebssysteme
  - Testumgebungen können schnell aufgesetzt werden
- Unterstützung alter Anwendungen
  - Legacy-Betriebssysteme oder Legacy-Anwendungen, für die keine Hardware mehr zu bekommen ist, können reanimiert werden

# Nachteile und Grenzen der Virtualisierung

- Leistungsverlust
  - Aktuelle Virtualisierungstechnologien sind so ausgereift, dass sich der Leistungsverlust mit 5-10% nicht sonderlich auswirkt
  - Seit aktuelle Computer-Systeme Mehrkernprozessoren mit Unterstützung für Hardware-Virtualisierung (Intel VT/VT-x und AMD-V) enthalten, spielt der Leistungsverlust eine zunehmend untergeordnete Rolle
- Nicht jede Hardware kann angesprochen oder emuliert werden
  - Kopierschutzstecker (Hardwaredongles) sind ein Problem
  - Beschleunigte Grafik kann nicht immer realisiert werden
- Beim Ausfall eines Hosts würden mehrere virtuelle Server ausfallen
  - Ausfallkonzepte und redundante Installationen sind notwendig
- Virtualisierung ist komplex
  - Zusätzliches Know-how ist notwendig

# Virtualisierung im Cloud Computing

- **Anwendungsvirtualisierung** (JVM) in Plattformen wie der GAE
- **Partitionierung** spielt wegen der hohen Anschaffungskosten nur in Bereichen wie Cloud Gaming eine Rolle
- **Vollständige Virtualisierung** spielt im Dienstangebot von Cloud-Anbietern mit Ausnahme von KVM zurzeit kaum eine Rolle
  - KVM kann von einigen Private Cloud-Lösungen verwendet werden
- Xen (**Paravirtualisierung**) ist die Basis der AWS
  - Viele Private Cloud-Lösungen verwenden Xen
- **Betriebssystem-Virtualisierung** kann in Cloud-Umgebungen helfen, die Hardware effizienter auszunutzen
- **Speichervirtualisierung** ermöglicht in Cloud-Rechenzentren den Speicher zu konsolidieren und besser auszunutzen
- Einige öffentlich zugängliche und private Cloud-Dienste nutzen **VLANs**, um die Netzwerkkommunikation der Instanzen vom Produktionsnetzwerk der physischen Infrastruktur zu trennen