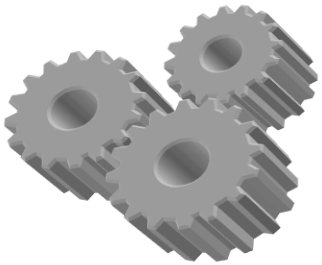
Dr. Stefan Fütterling – The Open Group Distinguished Architect – stefan.fuetterling@capgemini.com
Martin Bachmaier – IT Versatilist – mbachmaier@lenovo.com
Daniel Amor - The Open Group Master Architect - danny@dxc.com

Duale Hochschule Baden-Württemberg Stuttgart
IT Architekturen, 2025-10

Server Virtualisierung

Gesamtsicht der Vorlesung

- **Einführung**
  - 1.1 Einführung in IT Architektur
  - 1.2 Dynamische IT Infrastrukturen
  - 1.3 Cloud Computing (mit Übungsaufgabe)
- **Server Virtualisierung**
  - 2.1 Einführung in die Server Konsolidierung und Virtualisierung
  - 2.2 Virtuelle Maschinen (VMs) am Beispiel VMware vSphere (ESXi) (mit Übungsaufgabe)
  - 2.3 OS Containers am Beispiel Linux LXC und Docker (mit Übungsaufgabe)
  - 2.4 Deep Dive x86 Virtualisierung (Typ 1 Hypervisor)
- **Zentralisierter Storage**
  - 3.1 Storage Area Networks (SAN) und Network Attached Storage (NAS)
  - 3.2 RAID Levels
  - 3.3 Disksysteme und Hyperconverged Infrastructure
- **Clusterarchitekturen**
  - 4.1 Einführung in Clusterarchitekturen (LB-Cluster, HPC Cluster, HA Cluster)
  - 4.2 Scale Out Data Center
  - 4.3 Clustersoftware am Beispiel parallele Datenbanksysteme und Big Data Analysis Cluster (Hadoop)
- **IT Betrieb**
  - 5.1 Überblick DevOps, Application Management und Systems Management
  - 5.2 IT Service Management (ITIL)

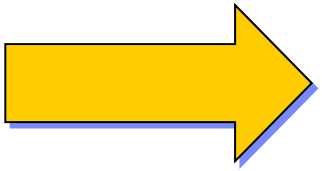


2.1 Einführung in die Server Konsolidierung und Virtualisierung

Ziele der Server Konsolidierung und Virtualisierung

- Reduktion der Anzahl der physischen Server
- Bessere Auslastung der Server Hardware, geringere Hardwarekosten
- Einfachere Skalierbarkeit von virtuellen Servern
- Vereinfachung der Systemlandschaft, verbesserte Wartbarkeit, geringere Betriebsaufwände
- Kurze Bereitstellungszeiten für virtuelle Server
- Erhöhung der Verfügbarkeit (High Availability) und Wiederherstellbarkeit (Disaster Recovery), z.B. durch Verlagerung oder Neustart virtueller Server auf einem anderen physischen Host

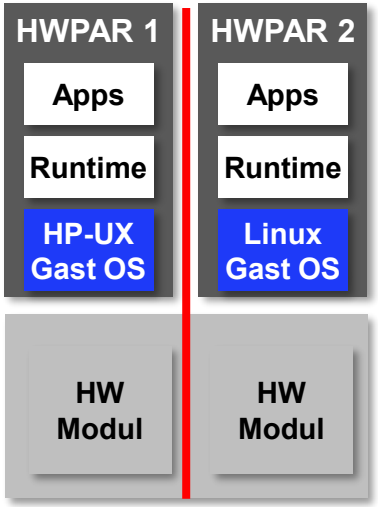
viele kleine Systeme, nicht virtualisiert



wenige große Systeme, virtualisiert

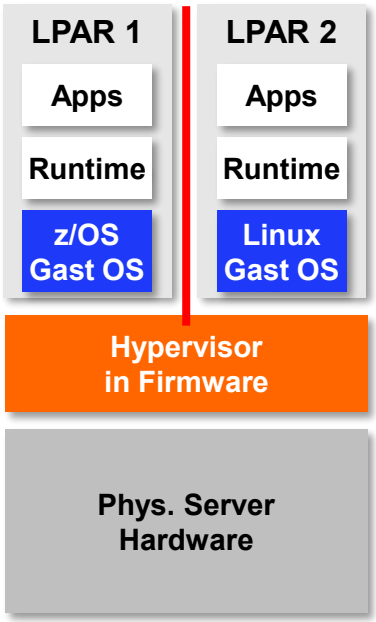
4 grundlegende Arten der Server Partitionierung und Virtualisierung

1. Hardware Partitioning



- HP Integrity nPar

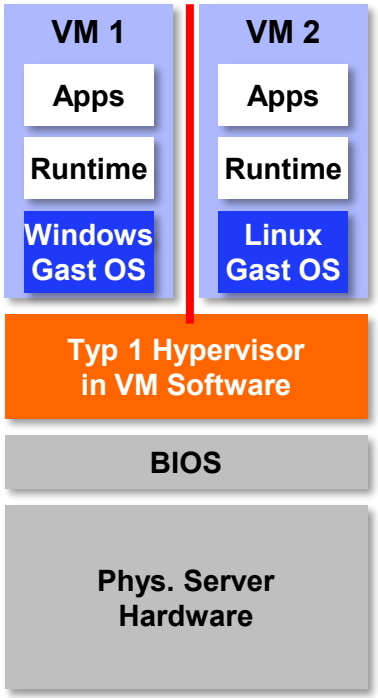
2. Logical Partitioning (LPAR)



- IBM System p
- IBM System z (Mainframe)

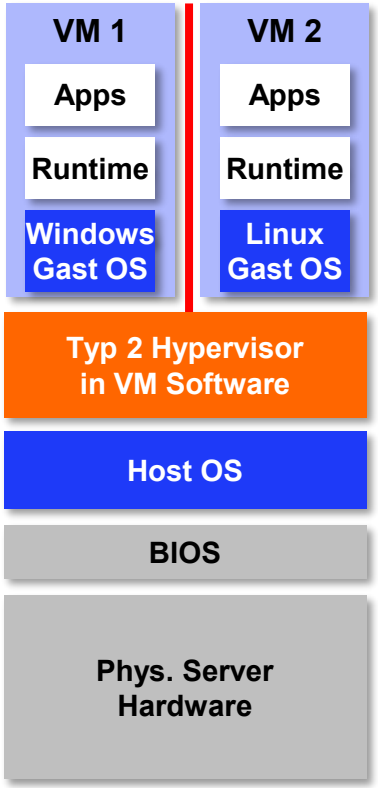
3. Software Partitioning (VM)

Typ 1 Hypervisor



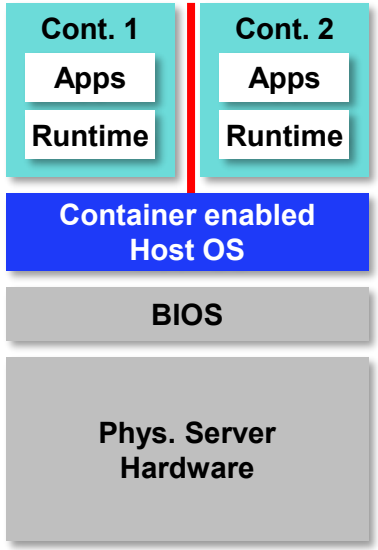
- VMware vSphere (ESXi)
- KVM
- IBM System z/VM
- Microsoft Hyper-V (Standalone)

Typ 2 Hypervisor



- VMware Workstation
- VirtualBox

4. OS Containers



- HP-UX Resource Partitions
- Solaris Zones
- Linux Containers (LXC)

4 grundlegende Arten der Server Partitionierung und Virtualisierung

Art der Partitionierung	Technische Umsetzung	Eigenschaften der Partitionen	Beispiel
Hardware Partitionierung	▪ Elektrische Isolation von Hardware Modulen	▪ Große starre Partitionen ▪ Ein Gast-OS pro Partition oder weitere Partitionierung ▪ Multi-OS möglich	▪ HP Integrity nPar
Logische Partitionierung (LPARs)	▪ Hypervisor in der Firmware	▪ Feingranular, kleine LPARs möglich ▪ Dynamische Änderung der LPARs möglich ▪ Ein Gast-OS pro LPAR ▪ Multi-OS möglich	▪ IBM System p ▪ IBM System Z (Mainframe)

Software Partitionierung (VMs)	▪ Hypervisor in einer Virtualisierungssoftware
	  ▪ Typ 1: Hypervisor läuft direkt auf der Firmware / auf dem BIOS
	  ▪ Typ 2: Hypervisor läuft auf einem Host-OS
		▪ Feingranular, kleine VMs möglich
		▪ Dynamische Änderung der VMs möglich
		▪ Ein Gast-OS pro VM
		▪ Multi-OS möglich
			▪ VMware vSphere (ESXi)
			▪ HP Integrity vPar
			▪ IBM System Z/VM
			▪ KVM
			▪ Microsoft Hyper-V

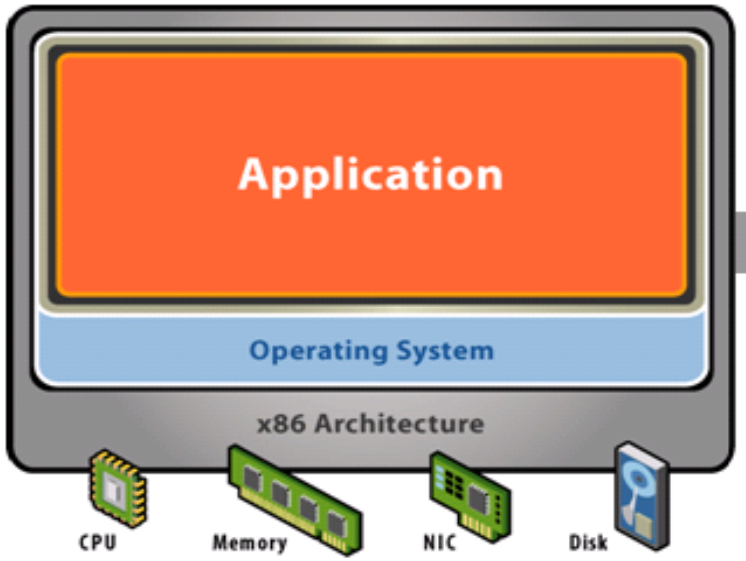
OS Containers	▪ Mehrere getrennte Bereiche für Anwendungen (Containers) auf einem gemeinsamen Kernel	▪ Sehr feingranular, kleine Container möglich ▪ Geringer Overhead im Vergleich zu VMs: ▪ Kein zusätzliches Virtualisierungsprodukt nötig ▪ Kein vollständiges Gast-Betriebssystem im Container ▪ Dynamische Änderung der Container möglich ▪ Nur ein OS (kein Multi-OS möglich)	▪ Solaris Zones ▪ HP-UX Resource Partitions ▪ Linux Containers

DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart

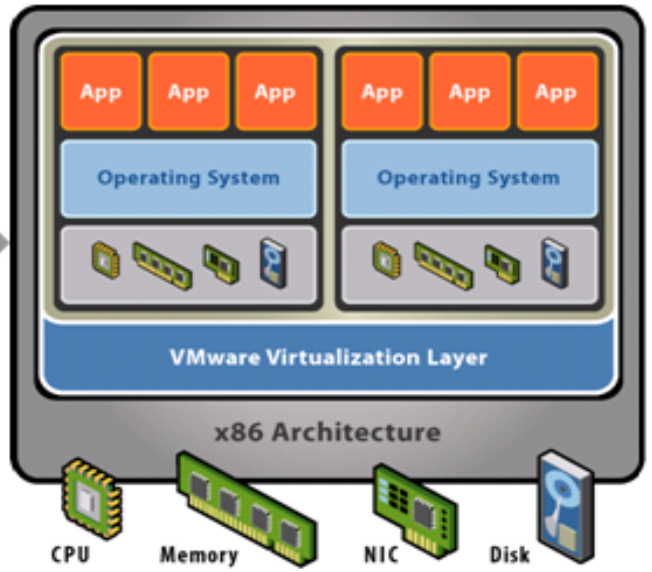
2.2 Virtuelle Maschinen (VMs) am Beispiel VMware vSphere (ESXi)

Was ist eine virtuelle Maschine?

- Ein physischer x86 Server kann mittels VMware vSphere in mehrere virtuelle Maschinen (VMs) unterteilt werden.
- Jede VM verfügt über **virtuelle Devices**.
- Jede VM enthält ein **vollständiges Gast-Betriebssystem** (z.B. Linux, Windows).
- VMware vSphere führt eine **vollständige Virtualisierung** durch, d.h.:
  - jede VM verhält sich wie ein physischer Server
  - ein Gast-Betriebssystem weiß nicht, ob es in einer VM oder auf einem physischen Server läuft.



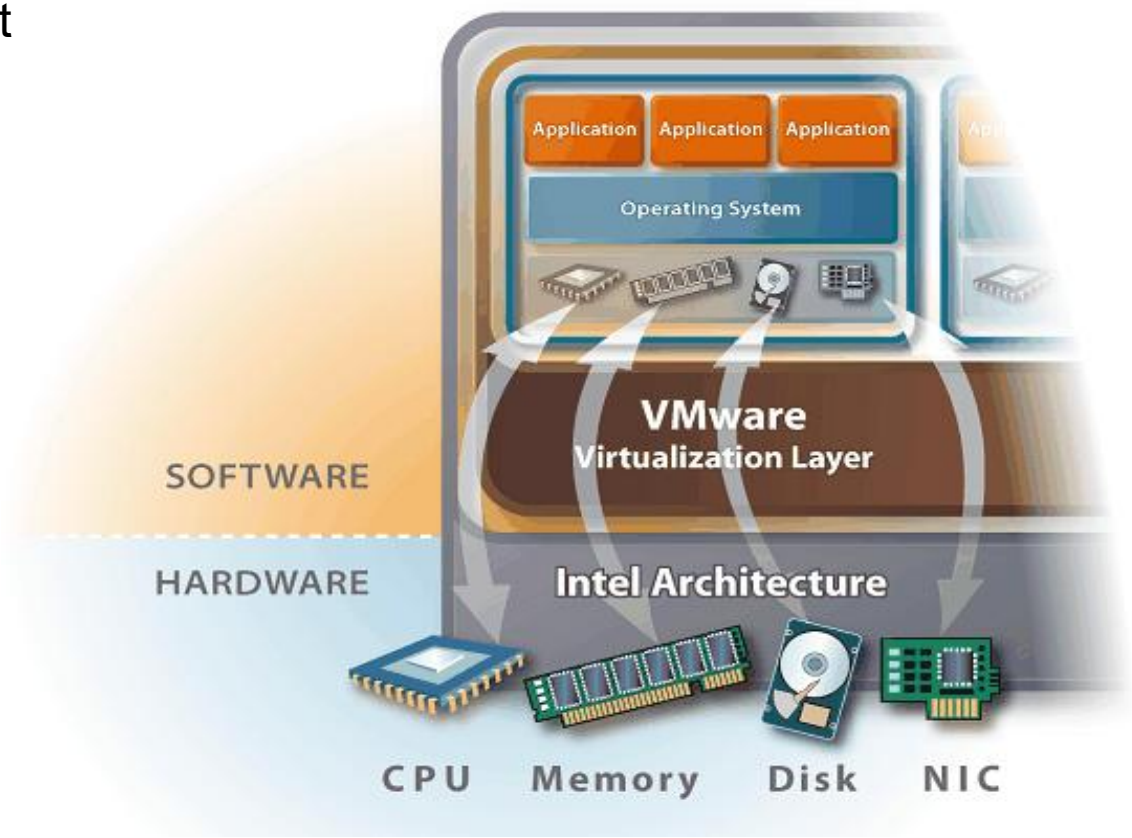
ohne Virtualisierung



mit Virtualisierung

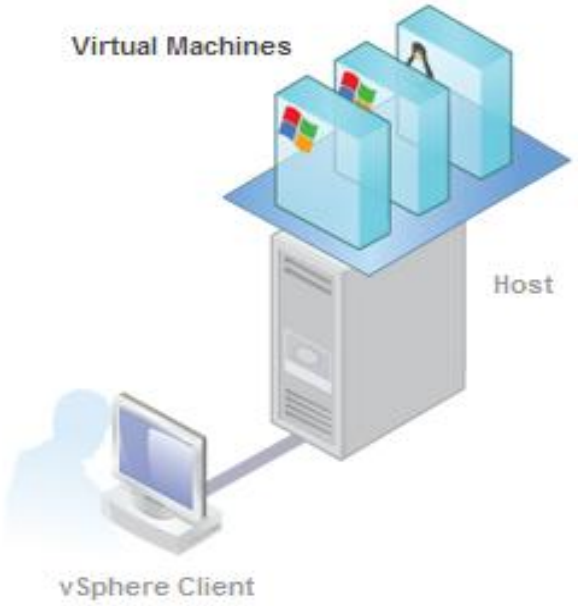
Partitionierung durch die VMware vSphere Software

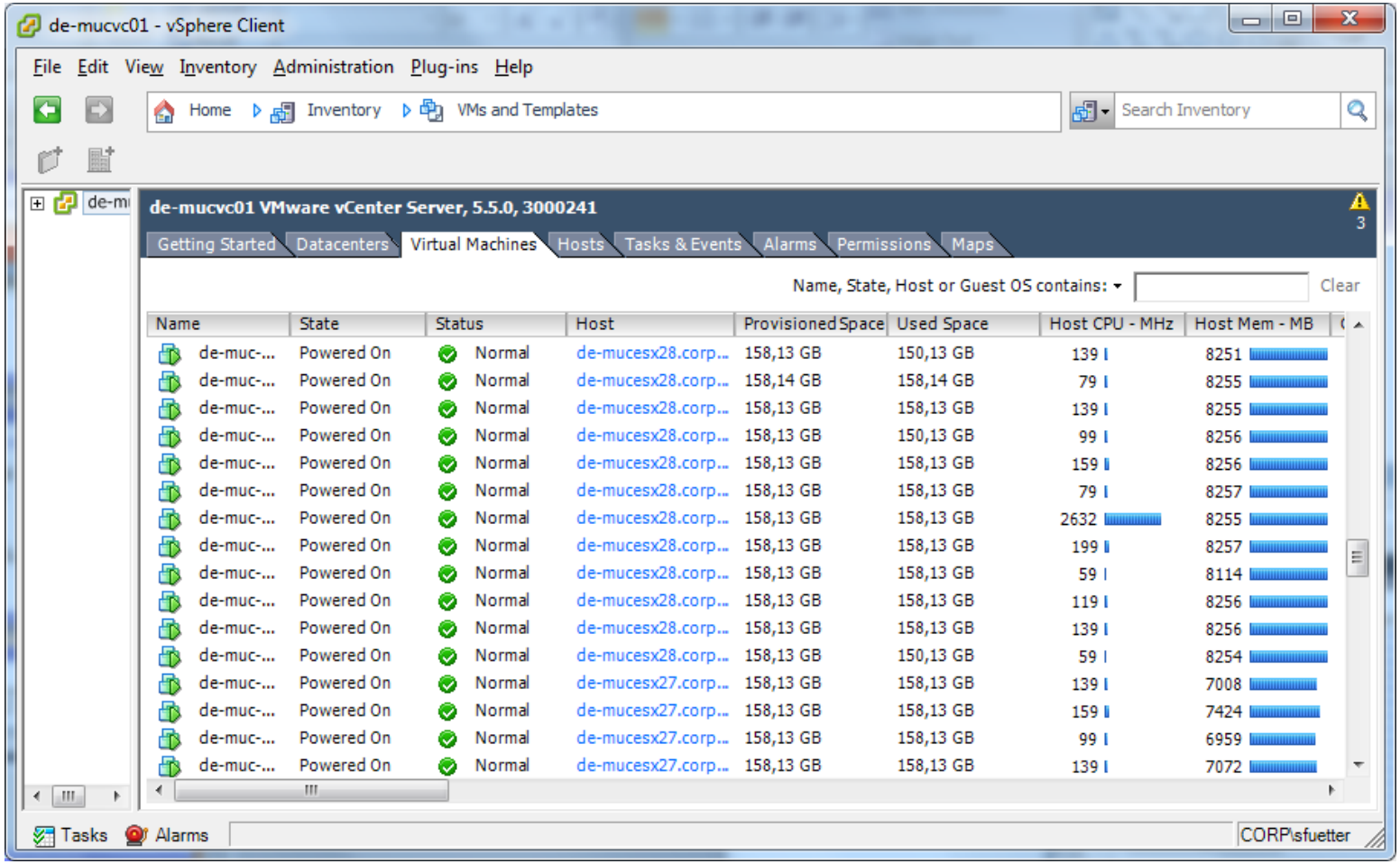
- Die Virtualisierungs-Software erlaubt die Nutzung der physischen Hardware durch mehrere VMs.
- Jeder VM werden **virtuelle Devices** zugewiesen:
  - Virtuelle CPU(s)
  - Virtuelles Memory (RAM)
  - Virtuelle Disk(s)
  - Virtuelle Ethernet Adapter (NIC)
- Die Gast-Betriebssysteme benutzen die virtuellen Devices wie physische Devices
- Die Gast-Betriebssysteme brauchen nur die **Gerätetreiber** für die virtuellen Devices



vSphere Client

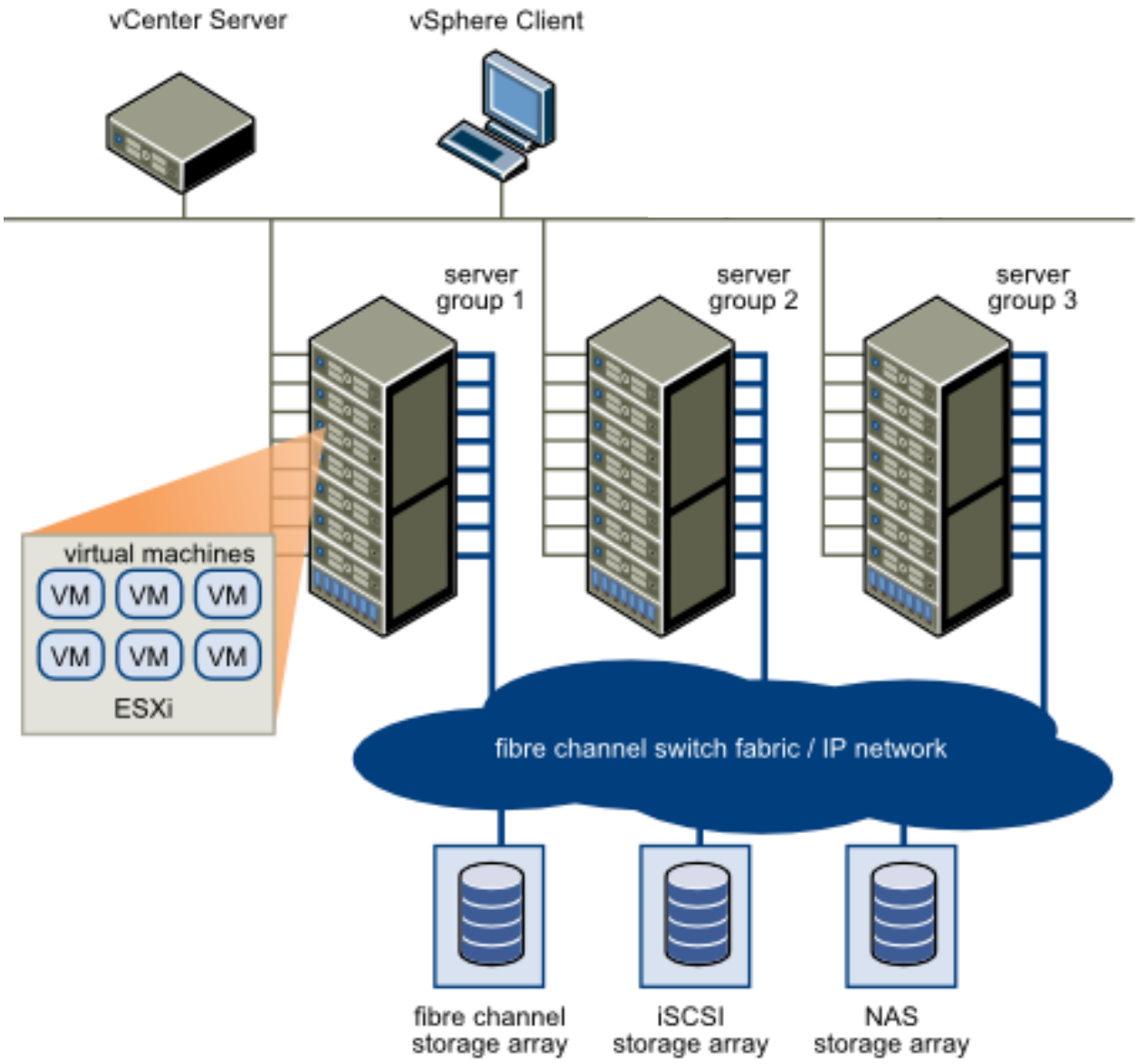
- Administrations-Interface zum vSphere Host / Cluster
- Anlegen, starten und stoppen von VMs
- Setzen von Konfigurationseinstellungen für VMs





VMware Cluster im Datacenter

- Infrastruktur
  - Server mit vSphere Hypervisor (ESXi)
  - Speichersysteme
- vCenter Server
  - Kontrollpunkt für die Administration der gesamten vSphere Umgebung
  - Zugriffsteuerung
  - Anlegen, Konfiguration und Steuerung von VMs
  - Zuweisung von Ressourcen an VMs
  - Leistungsüberwachung
  - Zuweisung der VMs auf vSphere Hosts
- vSphere Client
  - Administrationskonsole für die vSphere Administratoren



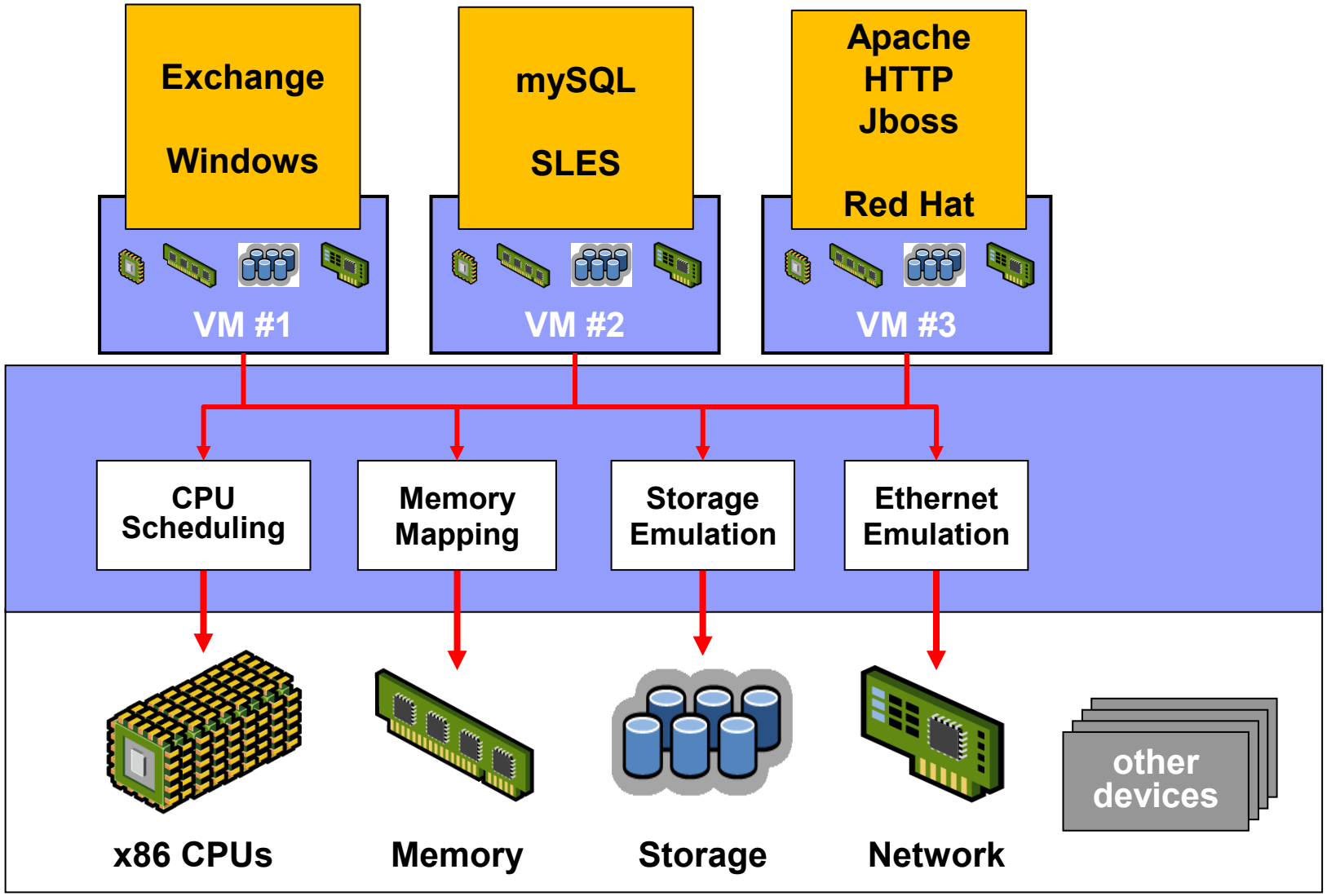
vSphere Hypervisor Architektur

Gast-Betriebs-systeme in virtuellen Maschinen

Virtuelle Devices der virtuellen Maschinen

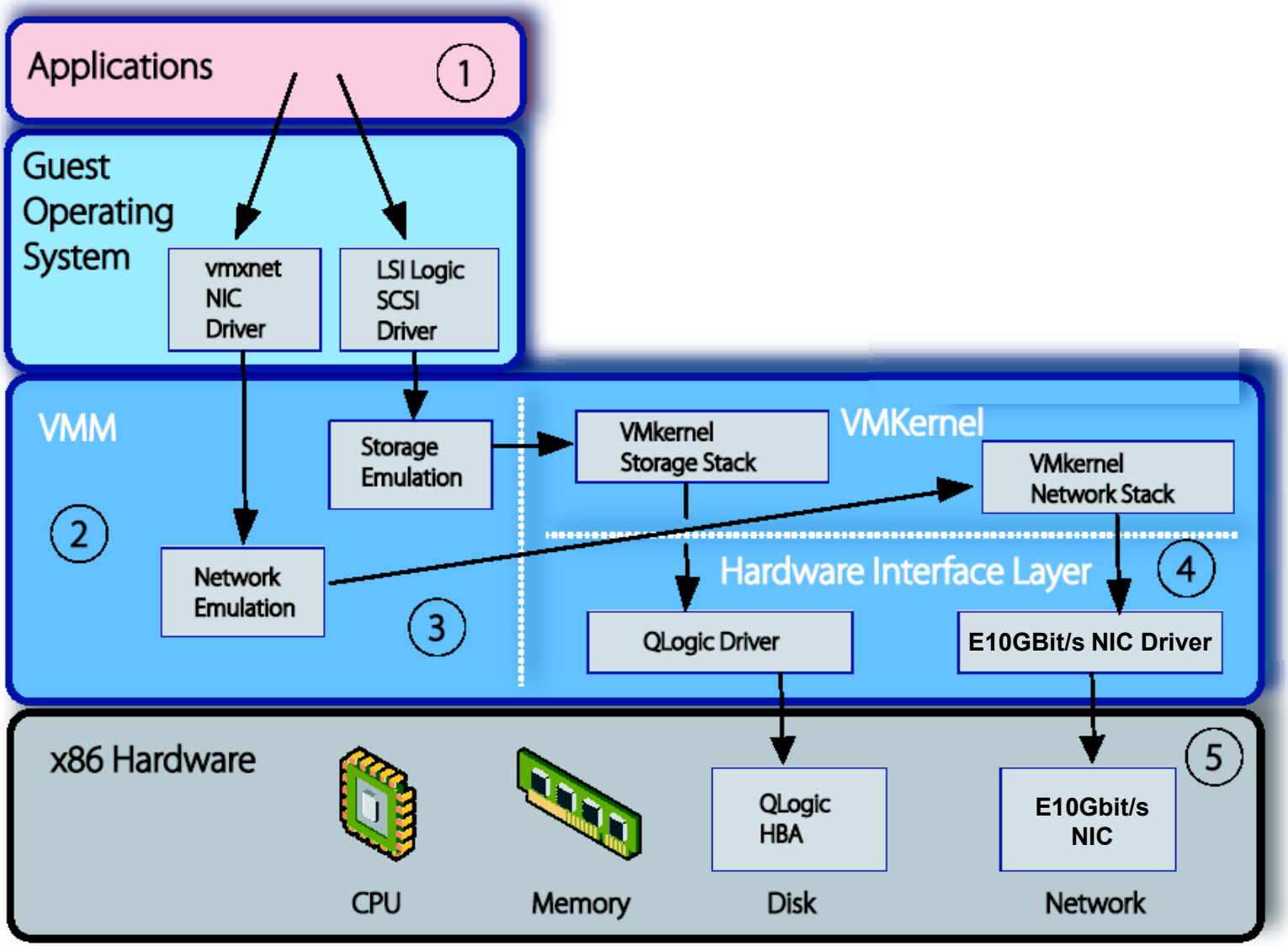
vSphere Kernel (ESXi)

Physischer Server mit physischen Devices



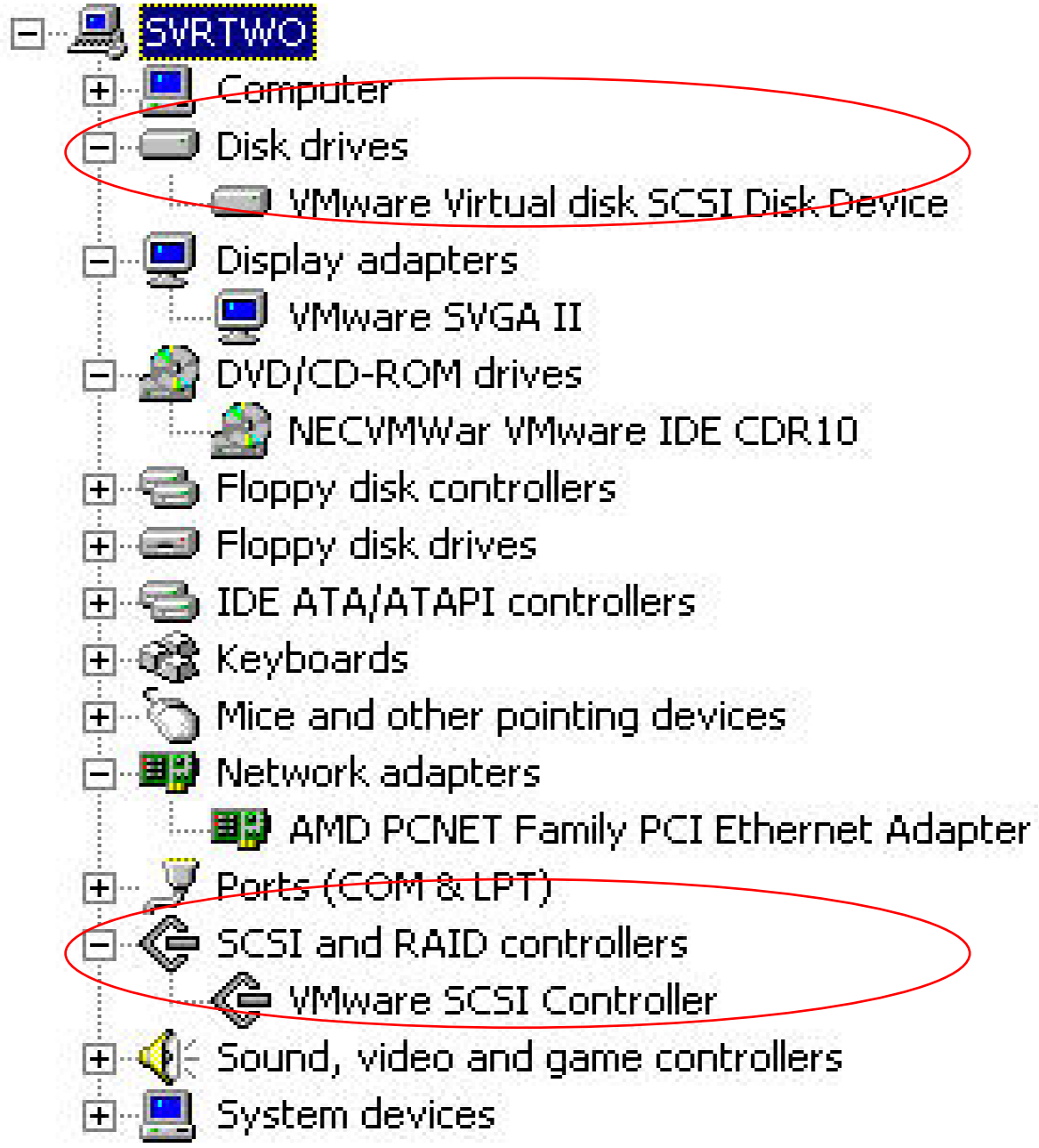
Storage und Netzwerk I/O Umsetzung

- Der Hypervisor bietet dem Gastbetriebssystem in der VM virtuelle I/O Adapter für
  - Ethernet
  - SCSI
- Die Gast-Betriebssysteme brauchen nur den **Treiber für die virtuellen I/O Adapter**
- Die Netzwerk und Storage Zugriffe werden im VMkernel **auf die physischen Adapter umgesetzt**



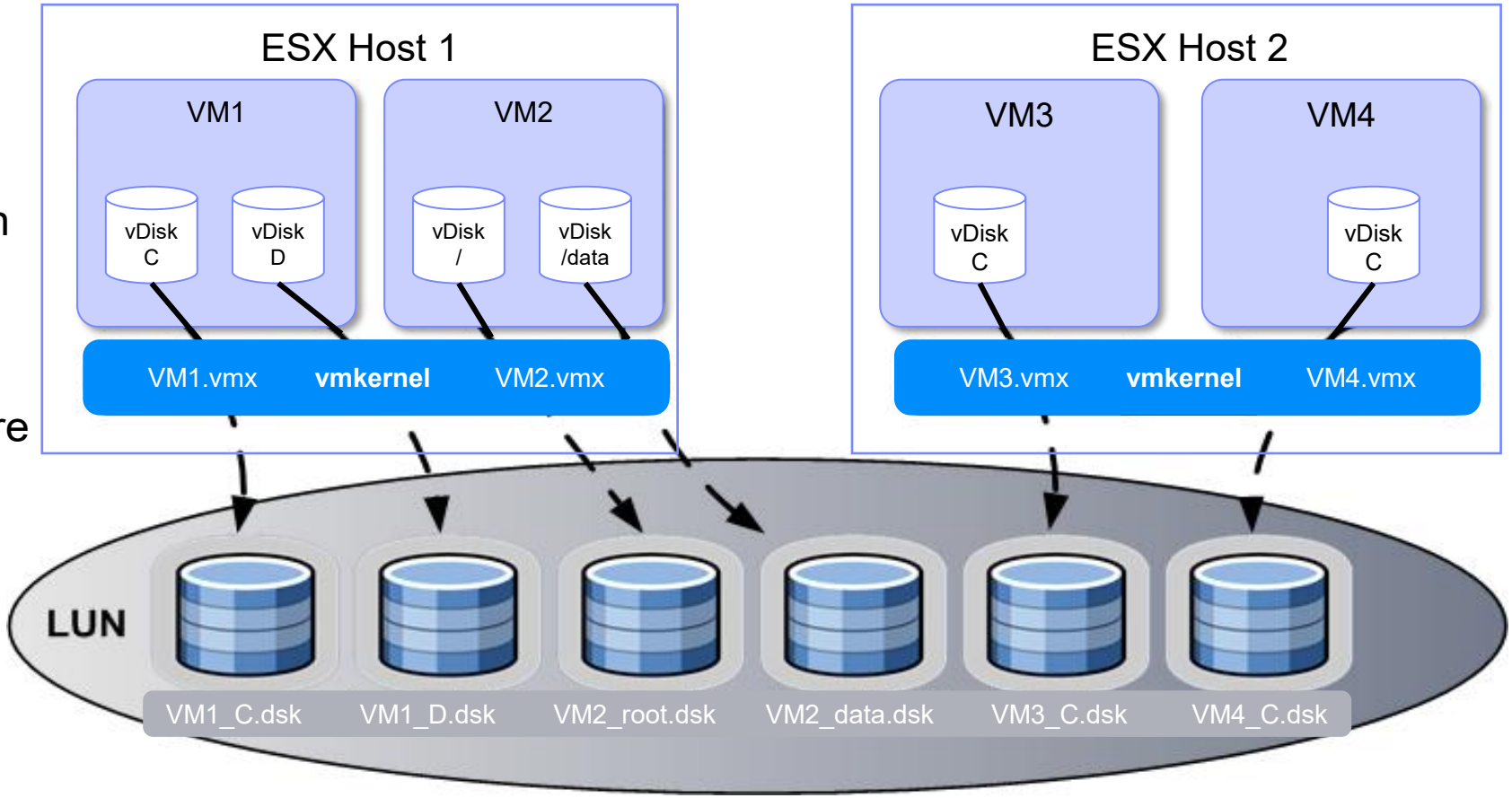
Virtuelle Disks

- Die virtuellen Maschinen haben mindestens eine virtuelle Disk
- Das Gast-Betriebssystem sieht immer **virtuelle SCSI Disks** an einem virtuellen SCSI Controller
- Die virtuellen Disks liegen als **Diskfiles** (Dateien) im **Datastore** des vSphere Servers, z.B.:
  - auf lokalen Festplatten im vSphere Host
  - auf einem SAN Storage, der an den vSphere Host angebunden ist
- Beispiel: Device Manager eines Windows Gast-Betriebssystems



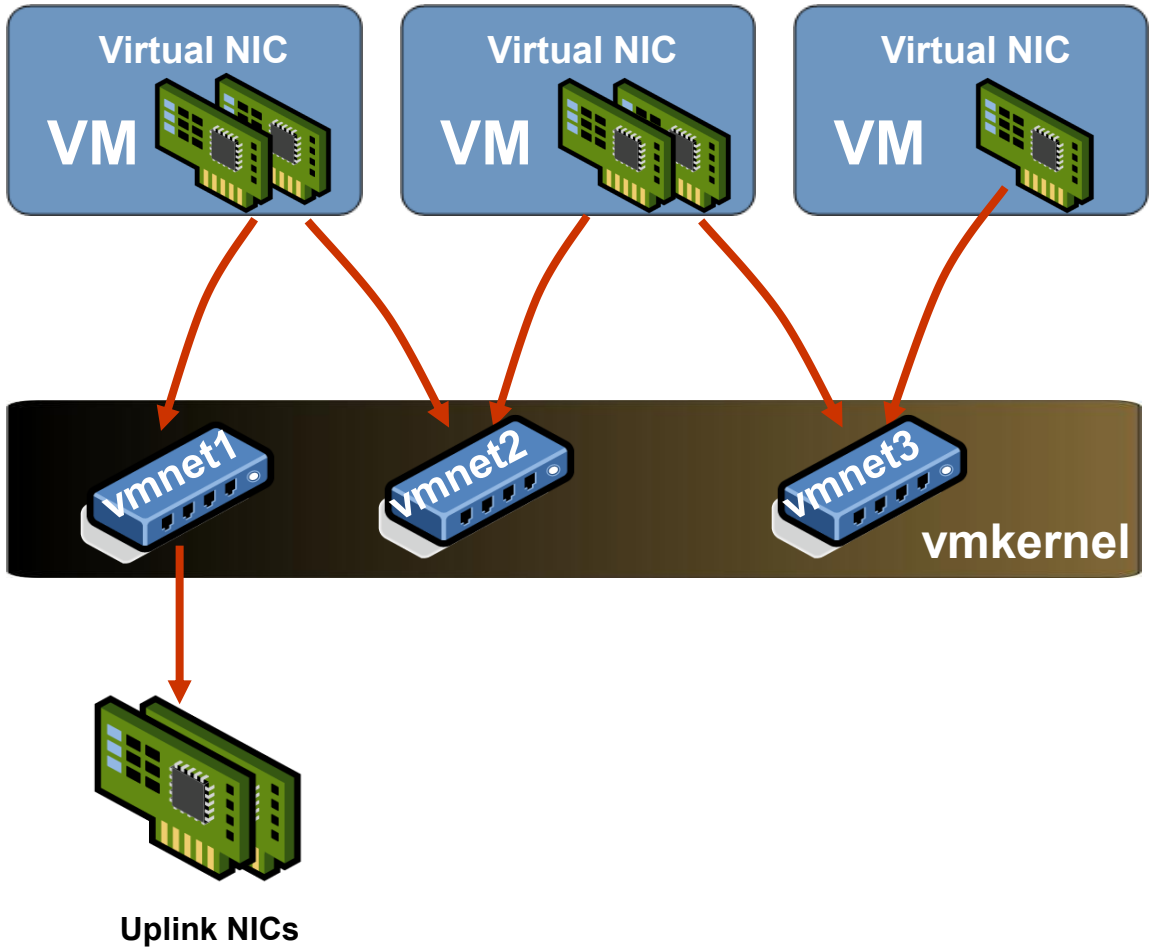
vSphere Hosts mit zentralem SAN Storage

- Mehrere vSphere Hosts teilen sich einen **Datastore** (z.B. SAN LUN)
- Im Datastore liegt ein **VMFS Dateisystem**
- VMFS regelt den gleichzeitigen Zugriff mehrerer vSphere Hosts auf den Datastore
- Der vmkernel verwaltet FibreChannel Multipathing und Failover

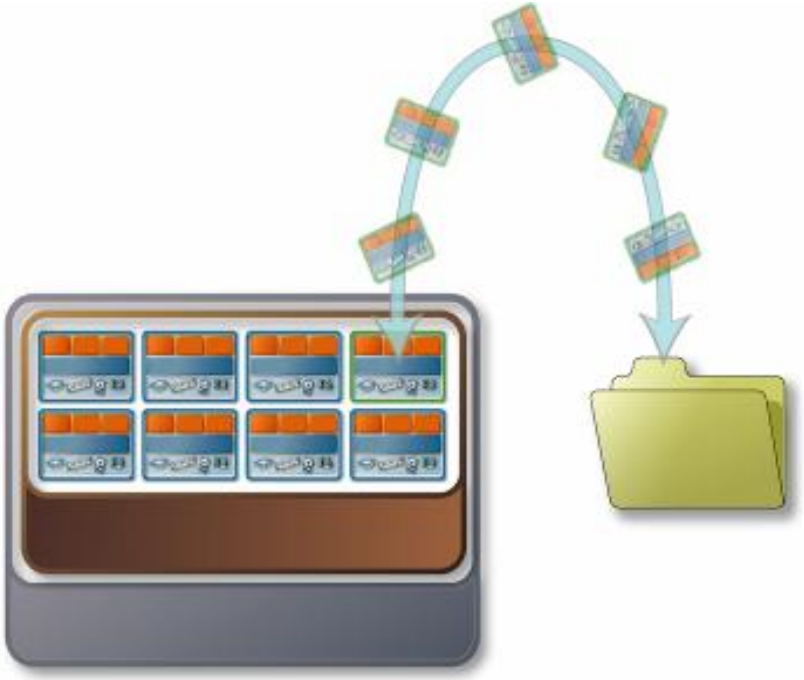


Virtuelle Netzwerke

- Jede VM hat mindestens einen **virtuellen Ethernet Adapter (NIC)**
- Jede virtuelle NIC hat eine eigene MAC und IP Adresse
- Die virtuellen NICs sind mit **virtuellen Ethernet Switches (vmnet)** im vmkernel verbunden
- Die virtuellen Ethernet Switches
  - Stellen die Verbindung zu den physischen Ethernet Adaptern her
  - Ermöglichen die Abbildung ganzer Netzwerk Topologien innerhalb eines physischen Servers
  - Ermöglichen die interne Kommunikation unter den VMs



Konfigurationsdateien einer VM



Kapselung der VM in drei Dateien:

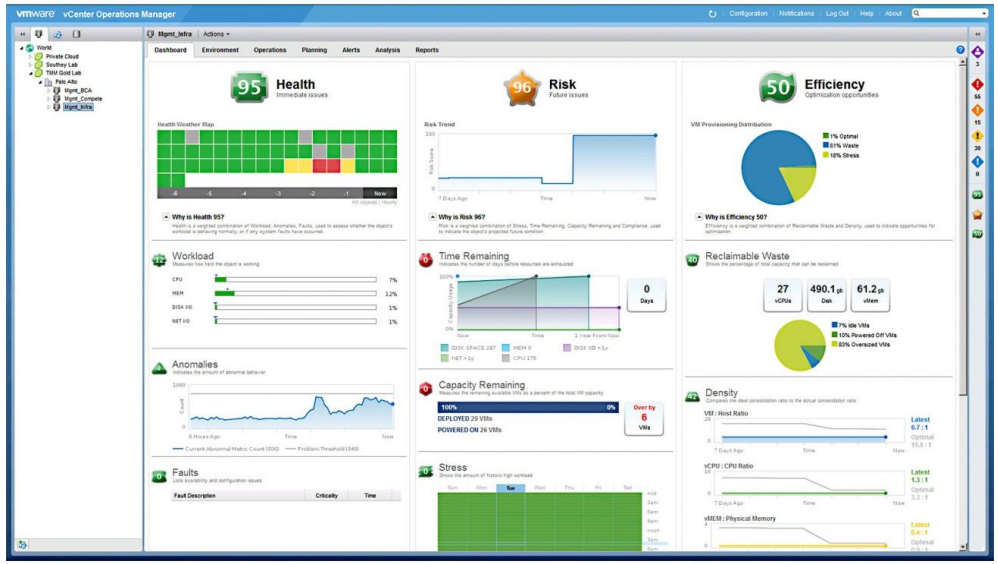
- Konfigurationsdatei für Hardware Attribute (*.vmx)
- Datei mit “Virtual BIOS” Einstellungen (*.nvram)
- Diskdateien (*.vmdk, *.dsk)

Die Kapselung erlaubt:

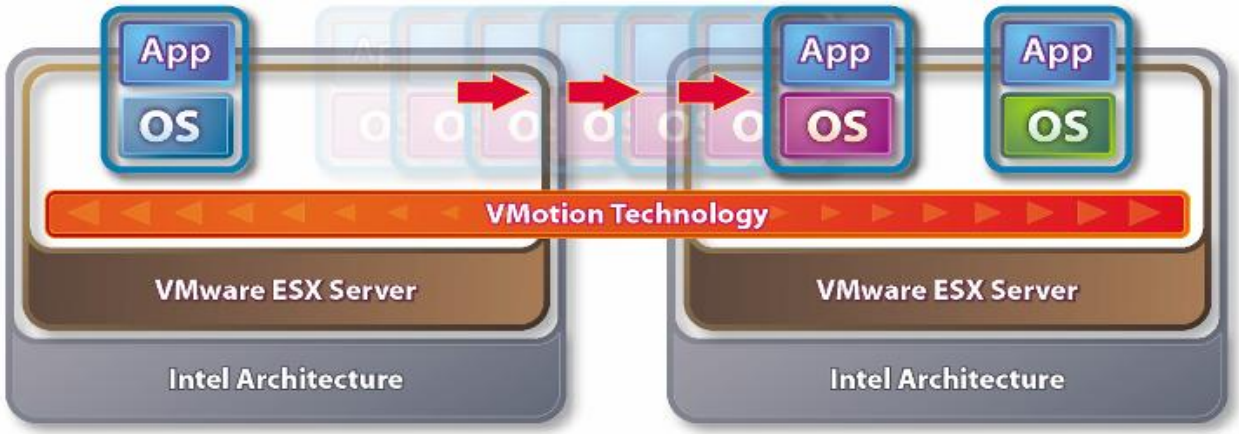
- einfaches Verschieben einer inaktiven VM auf einen anderen Host
- einfaches Klonen einer VM

VMware Zusatzprodukte

- **vSphere Operations Management**
  - Erweitertes Kapazitäts- und Performance Management (über die Funktionen des vCenter Servers hinaus)
  - Vmotion: Verschieben aktiver VMs zwischen vSphere Hosts
  - Load Balancing über die vSphere Hosts im Cluster
  - Hochverfügbarkeit: Restart von VMs nach Ausfall eines vSphere Hosts im Cluster



- **vRealize Suite (Provisioning Tool)**
  - Automation von Deployment- und Bereitstellungsprozessen (*vRealize Automation*)
  - Orchestration und Workload Management der VMs (*vRealize Operations*)
  - Accounting der verbrauchten Leistung (*vRealize Business*)



Vergleich gängiger VM-Lösungen

	VMware vSphere	Proxmox VE	OpenStack
Hersteller	▪ Broadcom	▪ Proxmox Server Solutions (Austria)	▪ Open Source ▪ Verschiedene Distributionen, z.B. Redhat oder Ubuntu/Canonical
Zielgruppe	▪ Große Unternehmen mit komplexen Anforderungen und bestehender VMware-Infrastruktur	▪ Kleine und mittlere Unternehmen (KMU), Heimanwender, Unternehmen die Open Source bevorzugen	▪ Große Cloud Provider, Unternehmen mit sehr großen und skalierbaren Infrastrukturen, die hohe Flexibilität benötigen.
Hypervisor	▪ ESXi	▪ KVM	▪ KVM
Unterstützte Gast OS	▪ Linux ▪ Windows	▪ Linux ▪ Windows	▪ Linux ▪ Windows
Unterstützter Storage	▪ FC, FCoE, iSCSI, NFS, local HCI (vSAN)	▪ FC, FCoE, iSCSI, NFS, local storage, u.a.	▪ Distributionsspezifisch (local und remote)
Zentrales Management	▪ vCenter mit Monitoring, Accounting, Load Balancing, Automation/Provisioning	▪ Web-based Management Interface für das Monitoring und Mgmt von VMs, Storage und Network Konfiguration	▪ Ja, distributionsspezifisch
HA-Funktion	▪ VMware HA	▪ Proxmox HA Clustering	▪ Ja, distributionsspezifisch
Live Migration	▪ Vmotion	▪ ja	▪ Ja, distributionsspezifisch
Implementierungs-aufwand	▪ Medium	▪ Medium	▪ Sehr hoch (Open Source) ▪ Hoch (Distribution)
Kosten	▪ Sehr hohe Lizenzgebühren	▪ Medium	▪ Low (Open Source) ▪ Medium (Distribution)
Stärken & Schwächen	▪ **Stärken:** Umfangreiche Features, ausgereift, Enterprise-Klasse. ▪ **Schwächen:** Hohe Kosten, Vendor Lock-in.	▪ **Stärken:** Open Source, integriert, benutzerfreundlich, gutes Preis-Leistungs-Verhältnis. ▪ **Schwächen:** Weniger Enterprise-Features als vSphere	▪ **Stärken:** Hochgradig skalierbar, flexibel, Open Source. ▪ **Schwächen:** Komplexe Implementierung und Management, hoher Expertenbedarf.

Übungsaufgabe Virtualisierung

Problemstellung 1: Sie müssen die Web-UI einer Applikation öffnen, die nur Internet Explorer 11 und Windows 10 erlaubt.

Lösung: Sie holen sich eine virtualisierte Windows 10 Umgebung mit Internet Explorer

Problemstellung 2: Sie müssen eine Applikation benutzen, die nur auf Fedora funktioniert.

Lösung: Sie holen sich eine virtualisierte Fedora Umgebung.

Laden Sie eine Virtualisierungssoftware herunter und installieren Sie diese auf Ihrem Rechner

VirtualBox/Vagrant/HyperV/VMware Workstation Player/Parallels

Laden Sie das Virtual Image von Microsoft herunter, um eine alte Version von Internet Explorer zu starten:

https://www.microsoft.com/en-us/software-download/windows10ISO

Installieren Sie das Virtual Image in der Virtualisierungssoftware und starten Sie das Windows darin. Rufen Sie eine beliebige Website mit Internet Explorer 11 auf.

Laden Sie das Virtual Image von Fedora:

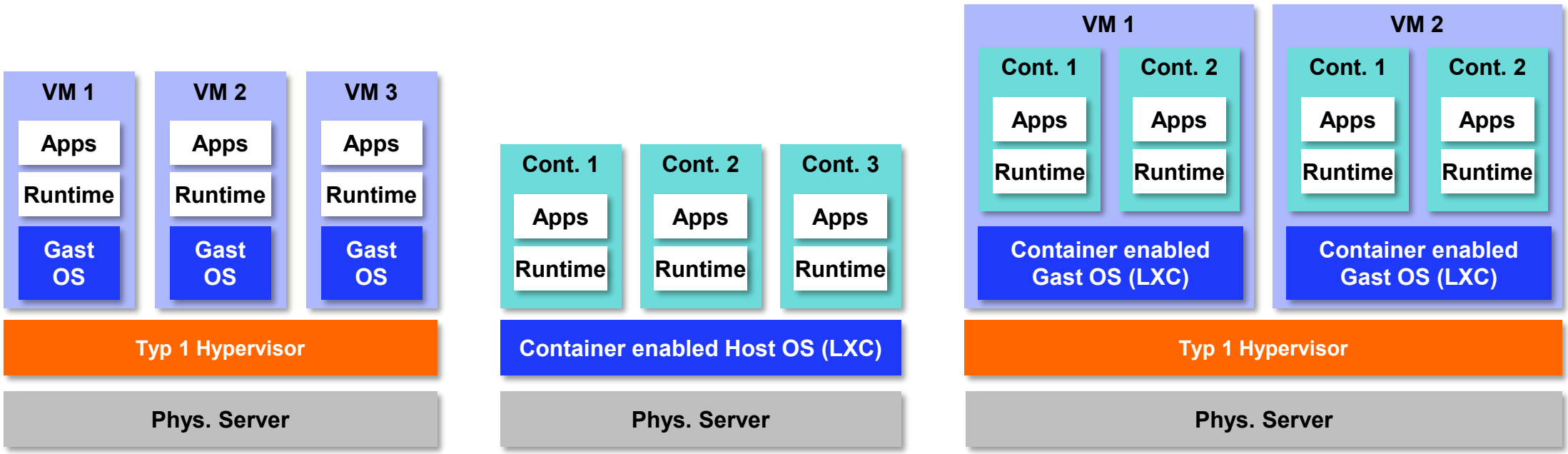
https://www.fedoraproject.org

Installieren Sie das Virtual Image in der Virtualisierungssoftware und starten Sie das Linux darin. Installieren Sie die notwendige Software.

Diese Übungsaufgabe ist freiwillig und wird nicht benotet. Viel Spaß!

2.3 OS Containers am Beispiel Linux LXC und Docker (mit Übungsaufgabe)

VMs und Linux Containers



Virtuelle Maschinen (VMs):

- VMs sind voneinander unabhängig
- Multi-OS
- VMs sind schwergewichtig durch vollständiges Gast-OS

Linux Containers (LXC):

- Gemeinsames Host-OS (kein Multi-OS möglich)
- Container enabled Host OS mit **LXC Kernel Erweiterungen** wie Cgroups, Namespaces, Mandatory Access Control
- Containers sind leichtgewichtig

Linux Containers in VMs:

- Schachtelung von Software Virtualisierung (VM) und OS Containers
- Nutzt alle Vorteile von VMs, wie z.B. einfaches Deployment und einfache Administration
- Gast-OS ist Container-enabled
- Heute oft eingesetzte Form der Virtualisierung

LXC cgroups for Resource Control

cgroups Features

- Resource limiting; memory, CPU, device accessibility, block IO, etc.
- Prioritization; who gets more of the CPU, memory, etc.
- Accounting; how much resources is this group using
- Control; freezing and check pointing

Subsystem	Tunable Parameters
blkio	- Weighted proportional block I/O access. Group wide or per device. - Per device hard limits on block I/O read/write specified as bytes per second or IOPS per second.
cpu	- Time period (microseconds per second) a group should have CPU access. - Group wide upper limit on CPU time per second. - Weighted proportional value of relative CPU time for a group.
cpuset	- CPUs (cores) the group can access. - Memory nodes the group can access and migrate ability. - Memory hardwall, pressure, spread, etc.
devices	- Define which devices and access type a group can use.
freezer	- Suspend/resume group tasks.
memory	- Max memory limits for the group (in bytes). - Memory swappiness, OOM control, hierarchy, etc..

LXC Namespaces for Isolation

Function

- Linux kernel support to partition subsystems
- Multi-tenancy for the Linux kernel

Types of namespaces

- pid (process)
- net (NICs, routing, etc.)
- ipc (System V IPC)
- mnt (mount points, file systems, etc.)
- uts (hostname)
- user (UIDs)

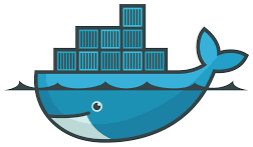
Conceptual examples

- PID of 111 in namespace ‘A’ is not the same process as PID 111 in namespace ‘B’
- A network interface in namespace ‘A’ cannot be seen outside of ‘A’
- File systems mounted in namespace ‘A’ are only visible to ‘A’
- Namespace ‘A’ can have a different hostname than namespace ‘B’

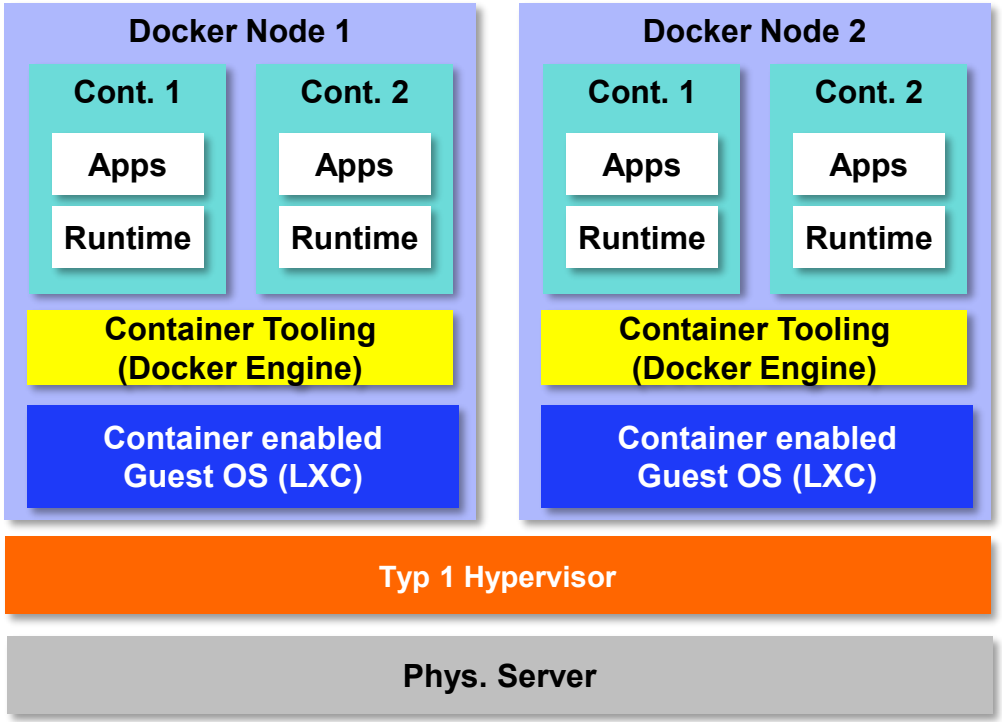
LXC Security

- All containers share the same copy of the Linux Kernel
- Use **Mandatory Access Control (MAC)** vs **Discretionary Access Control (DAC)** approach
- Must leverage one of the existing Linux Security Modules (LSM, all MAC below)
  - AppArmor (ships with Ubuntu, SuSE)
  - SELinux (ships with RHEL, Fedora)
  - TOMOYO
  - GRSEC
- Linux capabilities used to limit privileges
- Reduce shared FS access using RO bind mounts
- Keep kernel current
- User namespaces in Linux Kernel 3.8+:
  - Allow to run containers as non-root user
  - Allow to Map UID/GID inside / outside of containers (e.g. root inside the container will not be root outside)

Docker



- Docker ist ein **Container Tooling**, das die LXC Kernel Erweiterungen nutzt
- Docker macht Container **für den Entwickler leicht nutzbar** (z.B. Build, Deploy, Start, Stop)
- Standardisiertes **Image Format**
- Docker Container haben eine **hohe Portabilität** über verschiedene (Linux) Umgebungen
- Viele **fertige Images** nutzbar, z.B. von Docker Hub
- **Layering** im Image
- **Versionierung** der Images

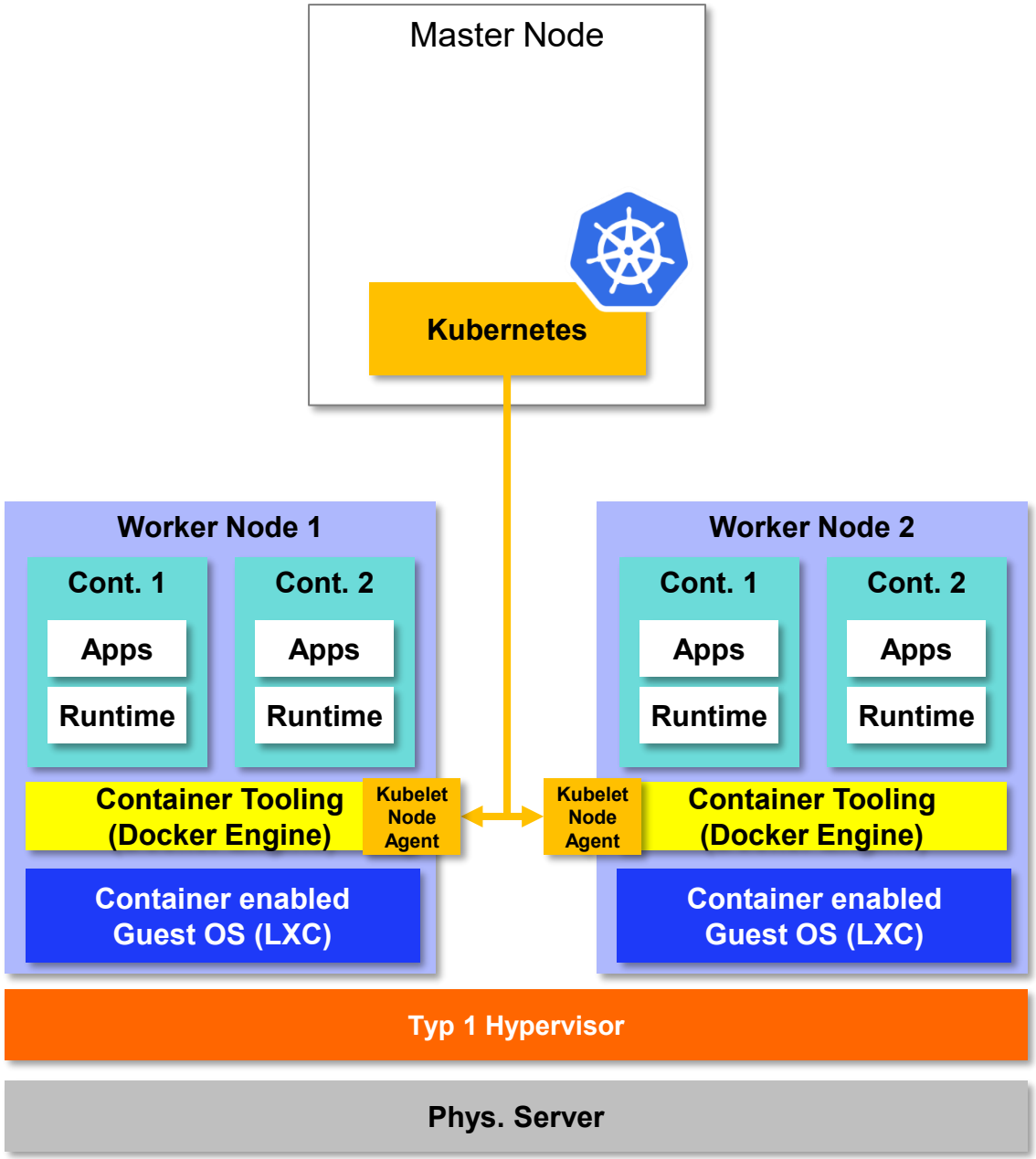


Container Orchestrierung

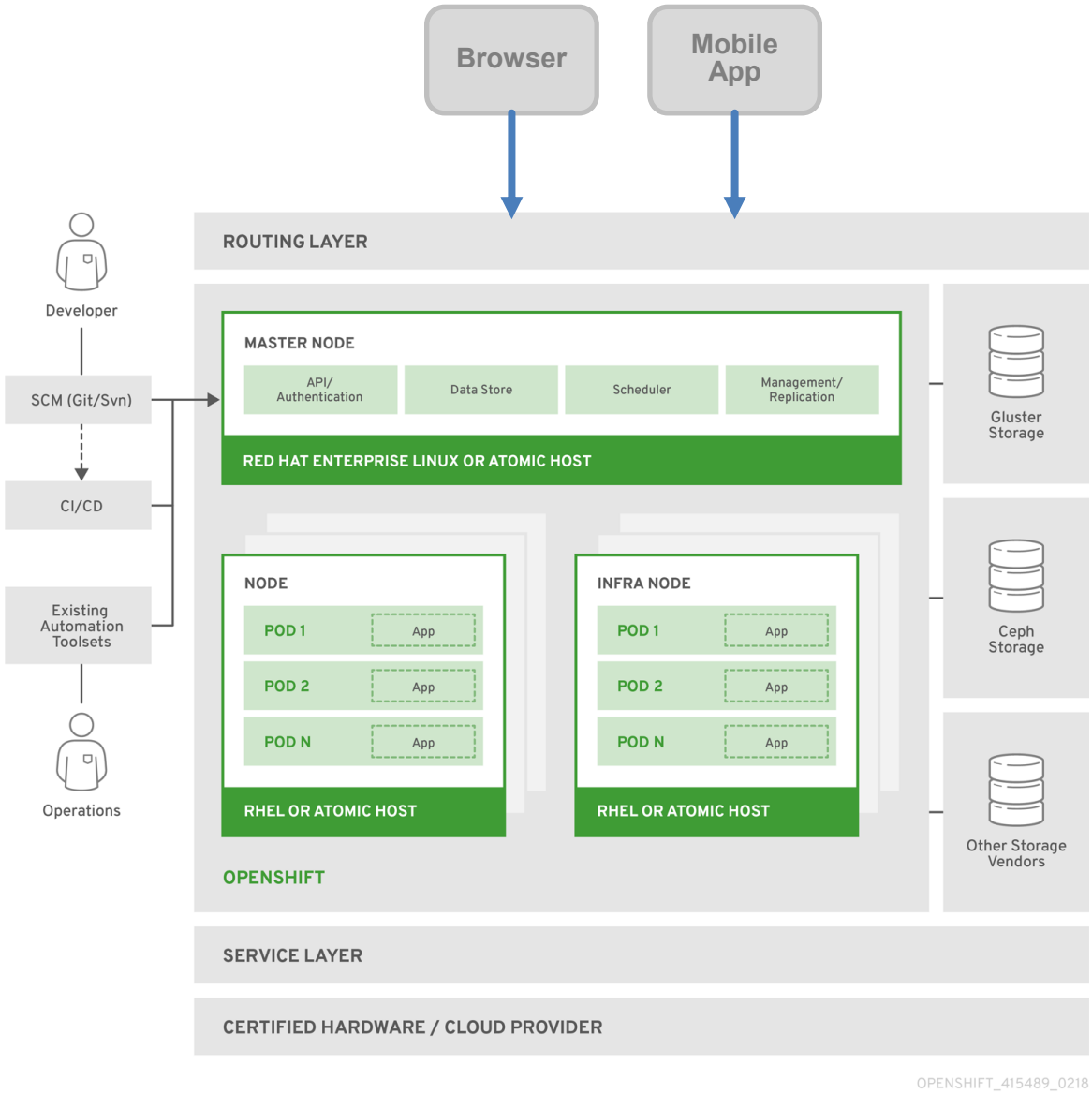
- **Docker** fokussiert auf das Bauen und Management eines einzelnen Containers
- Eine **Container Orchestrierung** unterstützt das Management einer Vielzahl von Containern (z.B. für Microservice-basierte Anwendungslandschaften)
- Beispiele:
  - Docker Swarm
  - Kubernetes
  - Redhat Openshift
  - SuSE Rancher
  - Pivotal Cloud Foundry

Kubernetes

- Clustering von mehreren Nodes zu **CaaS Umgebungen** (ein Master und mehrere Worker Nodes)
- Logische Gruppierung von Projekten und Containern (in **Namespaces** und **Pods**)
- Netzwerk Management, z.B. Routing und Subnetze für verschiedene Namespaces
- **Storage** Management: Zuordnung von Storage Volumes zu den Containern
- **Security** Management: Namespaces und Rollenkonzept für Admins
- Capacity Management, z.B. **Load Balancing** und **Autoscaling**
- Administration über kubectl oder Web GUI



Beispiel für eine CaaS Umgebung: Redhat Openshift



Source: https://docs.openshift.com/container-platform/3.11/architecture/index.html

- Openshift ist eine **Laufzeitumgebung** für Container-basierte Anwendungen (**Container as a Service CaaS**)
- Openshift basiert auf **Docker** und **Kubernetes**
- Anwendungen werden z.B. in Javascript oder in Java (Springboot, JBoss) erstellt
- Openshift kann **On-Premises** (im eigenen Datacenter) oder **Off-Premises** (in einer Public Cloud wie z.B. AWS, Azure oder IBM Cloud) aufgesetzt werden
- Vorteile:
  - Developer bekommen abgegrenzte Projekte (Kubernetes Namespaces)
  - Schnelles/kontinuierliches Deployment von neuen Anwendungen und Releases
  - Einsatz agiler Entwicklungsmethoden und DevOps
  - Openshift bietet Plattform Funktionen wie Monitoring, Auto-Scaling, Load Balancing und Green/Blue Deployments

2.4 Deep Dive
x86 Virtualisierung
(Typ 1 Hypervisor)

Virtualization recap

Popek & Goldberg – 1974
"Formal requirements for virtualizable third generation architecture"

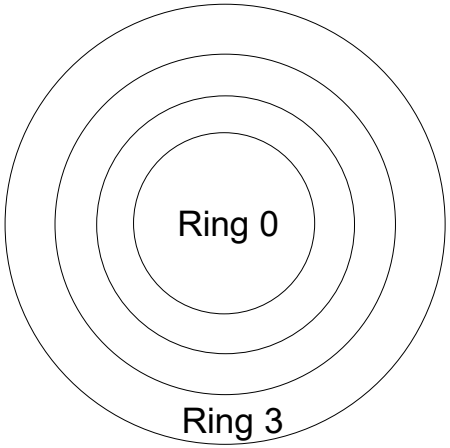
Introduces the concept of a VMM (hypervisor)

a. controls and manages the bare metal hardware resources
b. exports resources to a VM

Popek & Goldberg define three properties for any VMM:

1. Resource Control "*The VMM must be in complete control of the virtualized resources*"
2. Efficiency *"A statistically dominant fraction of machine instructions must be executed without VMM intervention."*
3. Equivalency *"A program running under the VMM should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly."*

Intro – x86 architecture until 2006



Ring 0
All HW can be accessed
All CPU registers can be changed
→ OS kernel runs here

Ring 3
Very limited HW access
→ Applications run here

Linux: kernel mode vs user mode

Intro – x86 virtualization – why it is a problem

According to Popek & Goldberg, all CPU instructions (ISA – instruction set architecture) fall into three categories:

1. **Privileged**: those that trap if the processor is in user mode and do not trap if the processor is in kernel mode
2. **Control sensitive**: those that attempt to change the configuration of resources in the system
3. **Behavior sensitive**: those whose behavior or result depends on the configuration of resources in the system

"For any conventional third generation computer, a VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions"

In 2000, Robin & Irvine analysed the x86 architecture:

→ revealed 17 sensitive but unprivileged instructions (meaning they don't trap, can't be handled by VMM as required by Goldberg)

Like:

PUSH or POP from stack

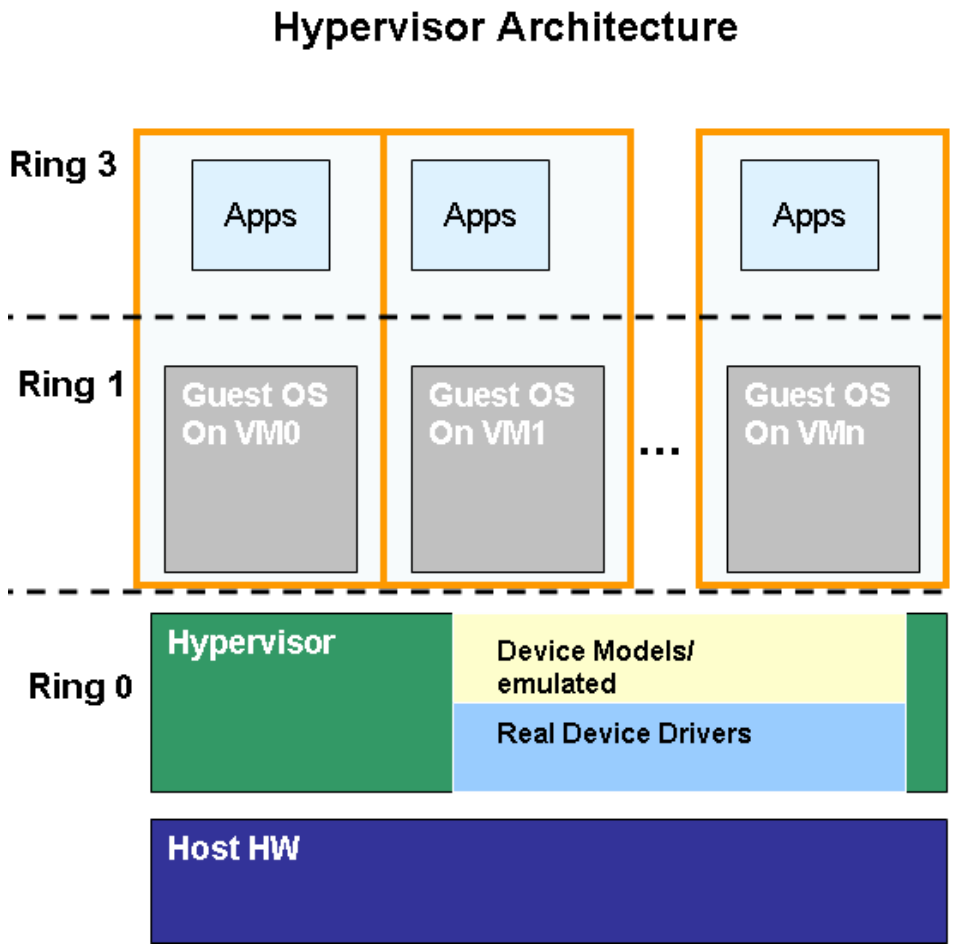
POPF: disable or enable interrupts

Example: What if VM1 in ring1 executes POPF

This is ignored because it does not have the privilege because it's ring 1 - but it also does not trap!

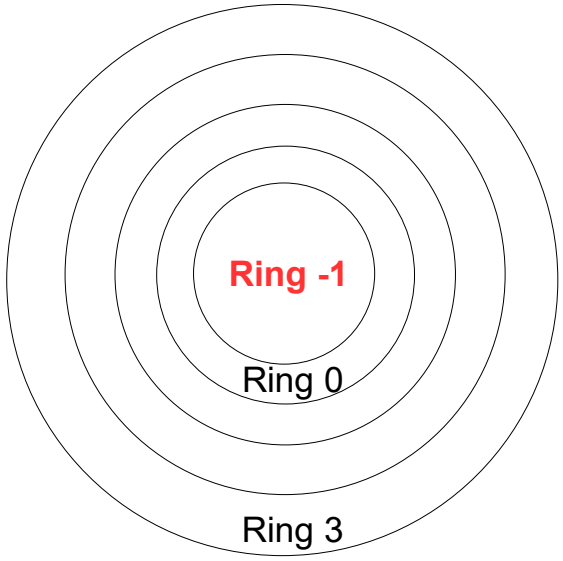
Intro – x86 virtualization – how to circumvent it

Yes, one can modify the guest OS to squeeze it into ring1, but... it's ugly. Modifications, low performance, not always feasible



"Ring deprivileging"

Intro – x86 virtualization – how to really fix it!



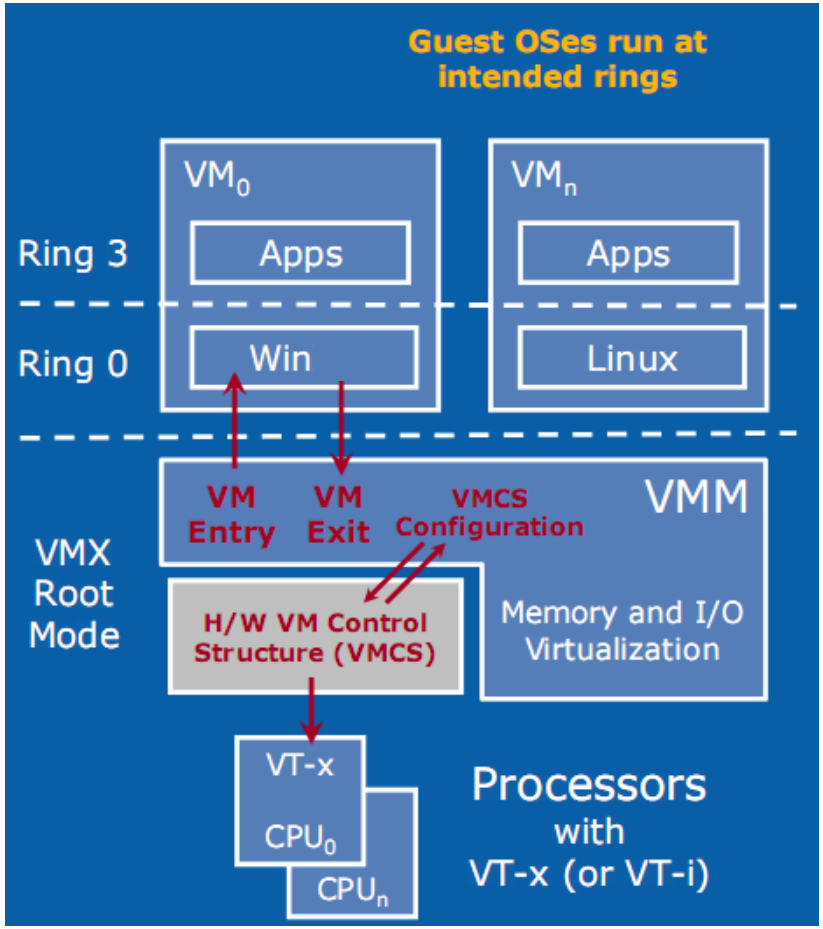
Intel VT-x
Virtualization Technology for x86
Since ~2009, new VMX instructions

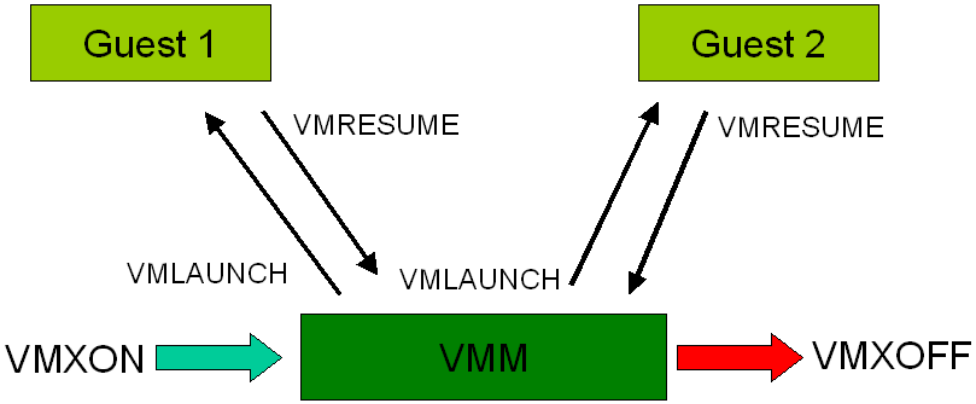
Ring -1
VMX root mode
→ VMM runs here

Ring 0, Ring 3
unchanged

VMM code always remains in full control of the HW

Intro – x86 virtualization – how ring -1 works

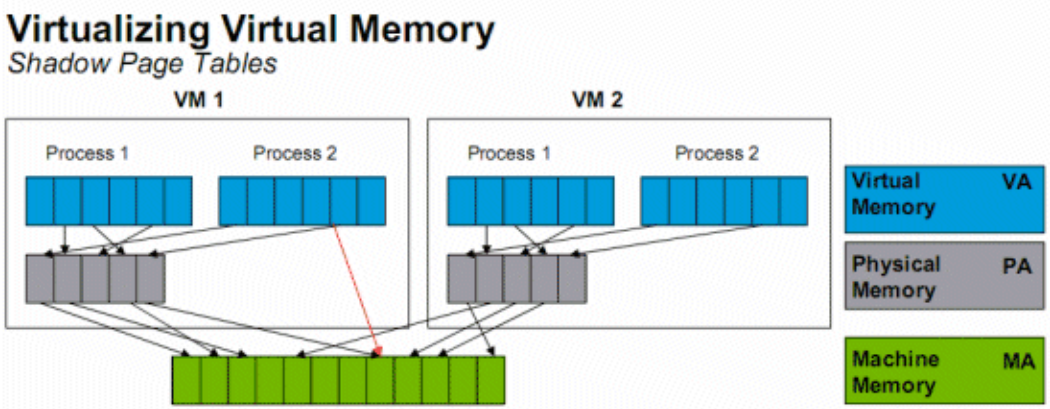




VM entry – VM exit
VMCS: VM control structure (CPU state, register content...)

Intro – x86 virtualization – memory virtualization

Step 1
VMM does all the work



Step 2
Support in Hardware