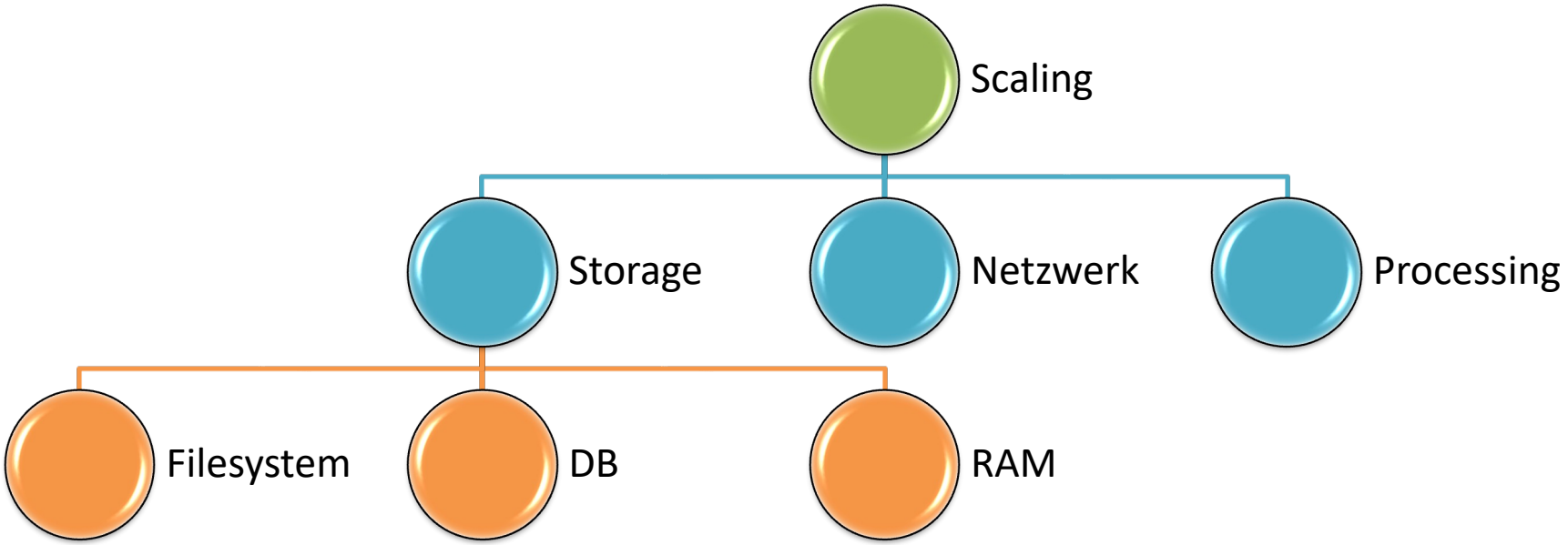
Scaling Dimensionen



Wegweiser

Storage Scaling:

Verteilte Dateisysteme statt scale up

Dateisysteme mit Cloud-Bezug

- Synergie: HPC
  - Parallel: Skalierbare IO Leistung
- Verteilt, shared, vgl. Shared Disk…
- Fehlertolerant, Synchronisation/Replikation, RAID
- Oft auf SAN/NAS/iSCSI aufgesetzt
- Einige wichtige Vertreter:
  - *CEPH*
  - *Glusterfs*
  - Lustre
  - XtreemFS
  - Google File System
  - *GPFS*
  - Hadoop Distributed File System
  - FhGFS

Filesystem Scaling: Auswahl?

Quelle:
Gartner

Eigenschaften zur Bewertung von verteilten Dateisystemen

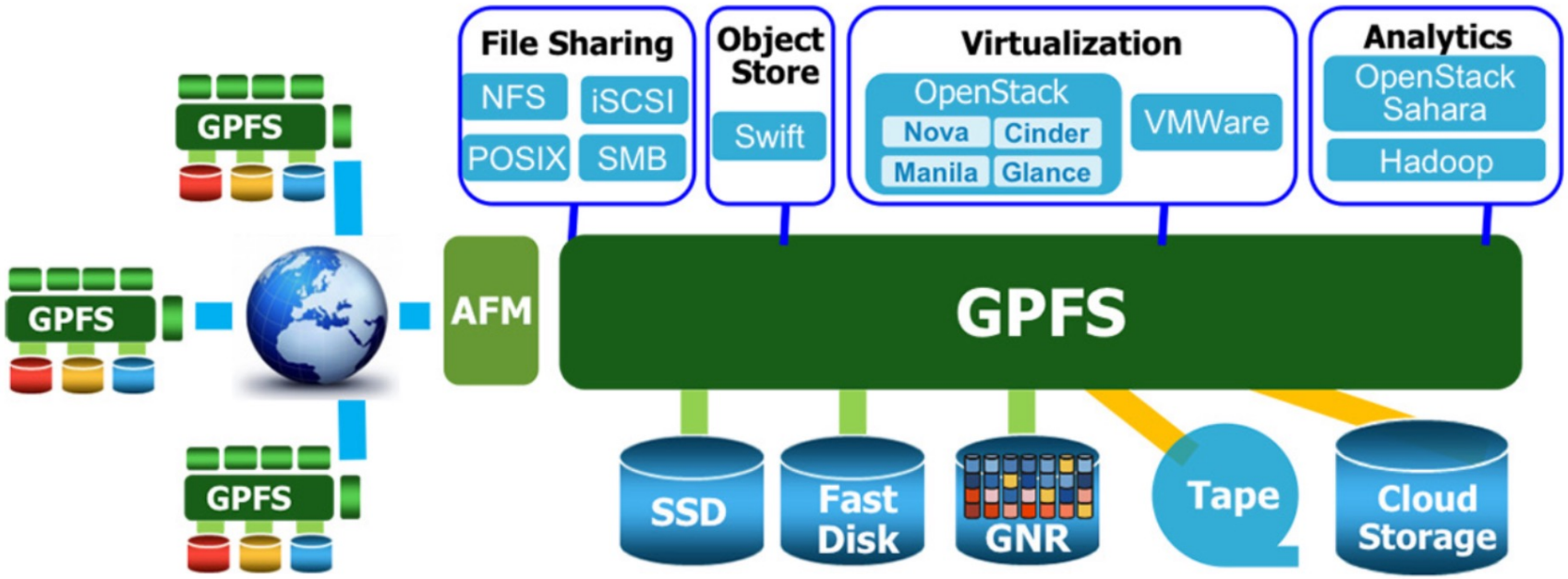
- Zugriffsinterfaces
- Konsistenzmodell
- Optimierungsstrategie
  - Redundanz / Points of Failure
  - Resilienz / Quorum
  - Performance / Wiederherstellungsperformance
  - Security
  - Verteilbarkeit / Skalierbarkeit
  - IO Bottlenecks
- Features
  - Umgang mit Metadaten
  - Umgang mit Zeitbasis
  - Versionierung / Immutability / Journaling
- Systemsupport/Systemabhängigkeiten/Storagesysteme

Vergleich einiger verteilter Dateisysteme

	GPFS	glusterfs	CEPH
Zugriffsinterfaces	Posix , SMB, NFS, OpenStack REST (S3, Cinder, Swift)	Posix, REST (S3, Swift), NFS, HDFS, SMB, libvirt	Rados API, REST (S3, Cinder, Swift), Posix (fuse), Block device
Konsistenzmodell	Diverse, lokal synchrone und global asynchrone Replikation	Client quorum (read/write), Server quorum, Replica levels (sync/async), Arbiter	Write Quorum, Synchron
Optimierung	GNR, Striped, File placement, Storage Pooling, div. andere	Distributed, Replicated, Striped, Distributed Replicated, Distributed Striped	Skaliert linear mit OSDs, tuneable replication, CRUSH
Metadaten	Verteilt	Verteilt	Separate Services
Systemsupport	AIX, Linux, Windows	Linux, BSD, MacOS X, Solaris	Linux
Management	SNMP, DMAPI, REST, div. Managementapps	Cli, storaged, ansible, REST, Python/Go bindings	Various Web, REST, Cli
Lizenz	Kommerziell	GPLv3 + kommerzieller Support	LGPL 2.1 + kommerzieller Support

Beispiel IBM GPFS(General Parallel File System) aka „Spectrum Scale“

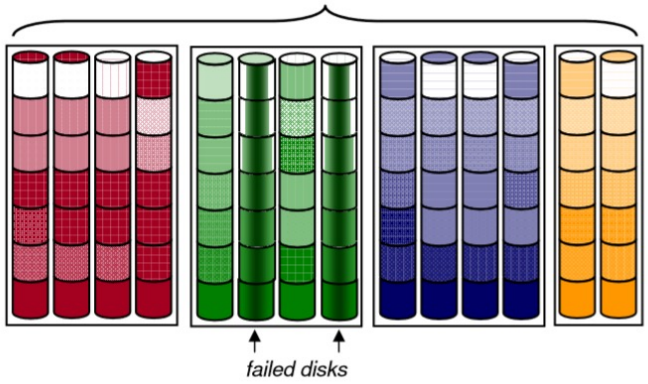
- IBM, seit 1991. Aktuell Version 4.2, bekannt für Supercomputer Einsatz
- Verteiltes Locking, Rolling upgrades, Encryption, Compression, Storage Pools, Placement Groups, Quotas, Transport Encryption, Zertifizierungen,…



GNR und Declustering

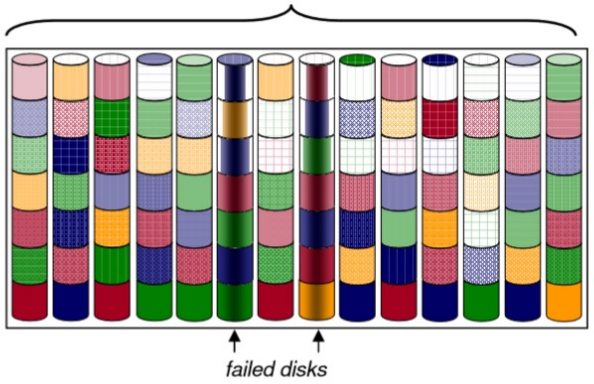
Declustered RAID6 Example

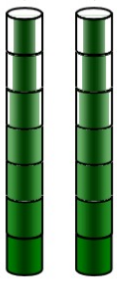
14 physical disks / 3 traditional RAID6 arrays / 2 spares



Decluster data, parity and spare

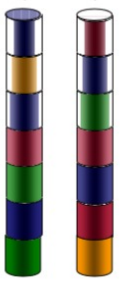
14 physical disks / 1 declustered RAID6 array / 2 spares





Number of faults per stripe		
Red	Green	Blue
0	2	0
0	2	0
0	2	0
0	2	0
0	2	0
0	2	0
0	2	0

Number of stripes with 2 faults = 7



Number of faults per stripe		
Red	Green	Blue
1	0	1
0	0	1
0	1	1
2	0	0
0	1	1
1	0	1
0	1	0

Number of stripes with 2 faults = 1

Quelle: https://www.usenix.org/legacy/events/lisa11/tech/slides/deenadhayalan.pdf

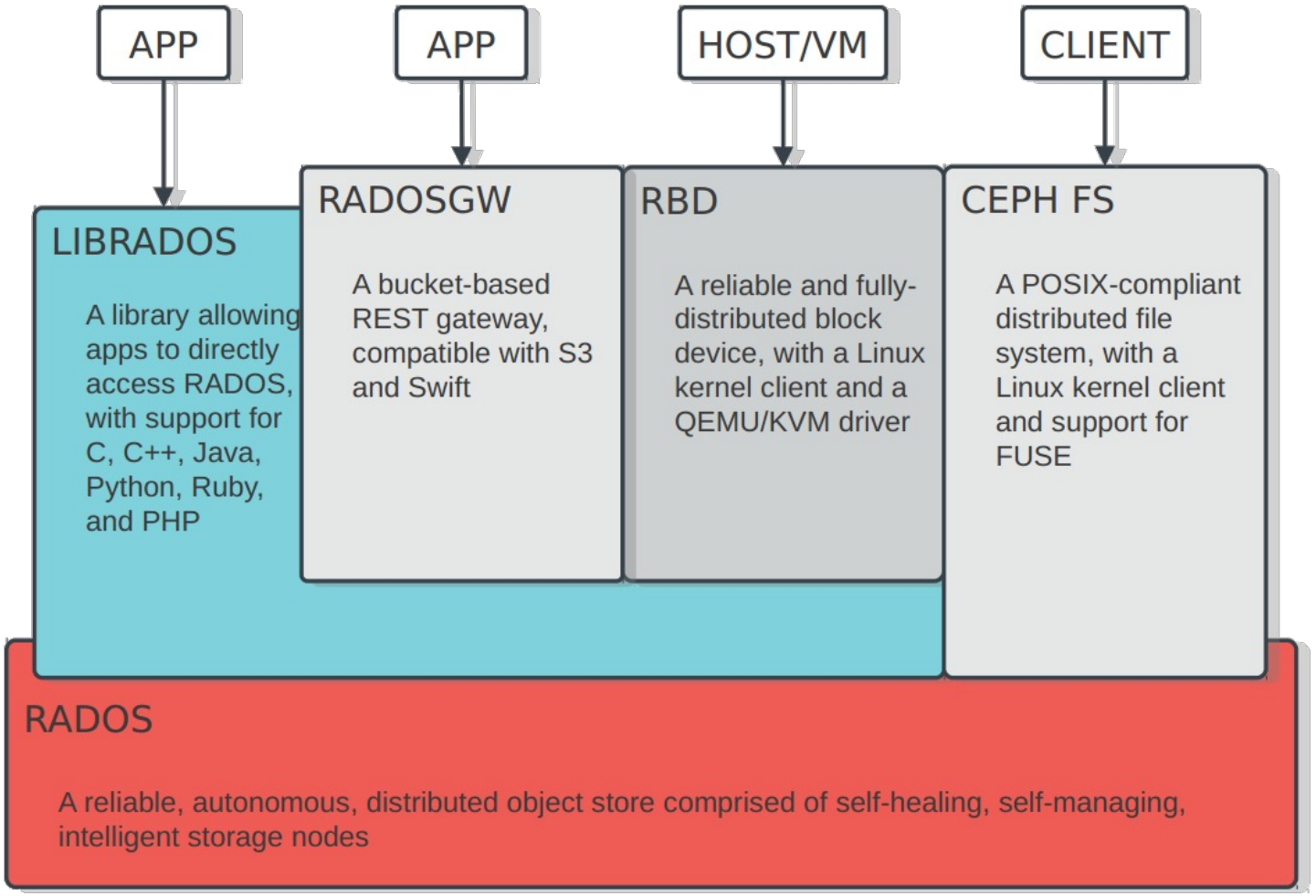
Beispiel glusterfs

- `http://www.gluster.org`
- Von Red Hat, Open Source
- Spezielle Features:
  - Globaler Namespace
  - Elastischer Hash Algorithmus & Volume Manager
  - User-space
  - Keine Blocksize-Größenbeschränkungen
  - Heterogene Storage-Systeme (Bricks)

Bepispiel CEPH

- Von InkTank, gehört nun Red Hat
- Features:
  - Metadaten getrennt von Daten
  - Hoher Autonomiegrad, kein Single Point of Failure, CRUSH als Algorithmus für Zugriffszielauswahl
  - FUSE, VFS, S3 kompatibel oder Ceph Library Zugriff
  - Diverse Dateisysteme als Grundlage auf Storage Nodes, Performance-Unterschiede…
- Nachteil: nur für Linux verfügbar

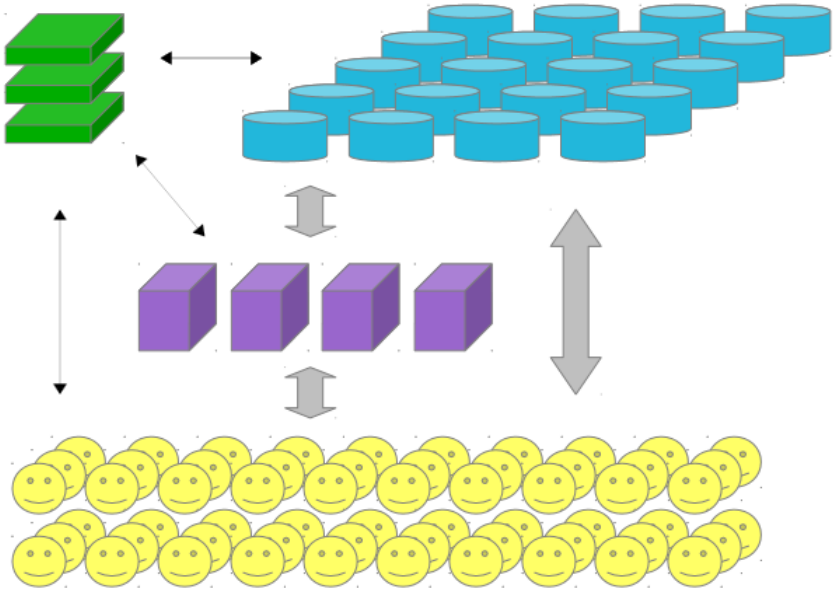
CEPH Layering



Quelle: Red Hat

Beispiel CEPH

- monitors (ceph-mon)
  - 1s-10s, paxos
  - lightweight process
  - authentication, cluster membership, critical cluster state
- object storage daemons (ceph-osd)
  - 1s-10,000s
  - smart, coordinate with peers
- clients (librados, librbd)
  - zillions
  - authenticate with monitors, talk directly to ceph-osds
- metadata servers (ceph-mds)
  - 1s-10s
  - build POSIX file system on top of objects



Quelle: ceph.com

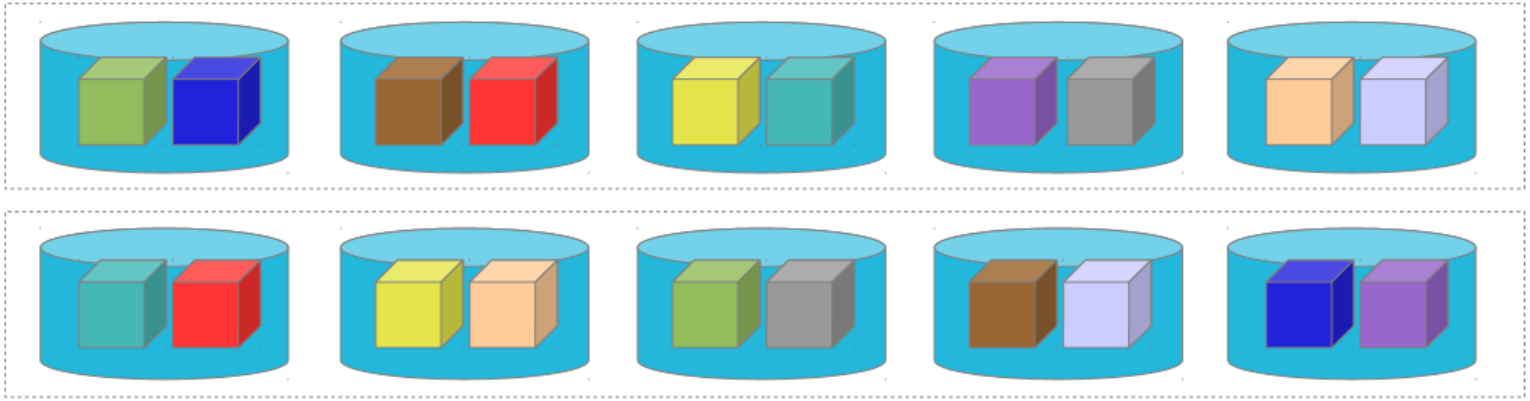
CEPH CRUSH

pool

hash(object name) % num_pg = pg

placement group (PG)

CRUSH(pg, cluster state, rule) = [A, B]



Weitere aktuell wichtige verteilte Dateisysteme

- **Google File System (GFS)**: http://research.google.com/archive/gfs.html
- **HDFS Implementierung**: http://hadoop.apache.org/docs/stable/hdfs_design.html
- **Colossus (GFS2)**: http://www.fclose.com/b/cloud-computing/3202/colossus-successor-to-google-file-system-gfs/
- **BigTable**: http://research.google.com/archive/bigtable.html
- **Megastore**: http://research.google.com/pubs/pub36971.html
- **Spanner**: http://research.google.com/archive/spanner.html
- **Dynamo**: http://dl.acm.org/citation.cfm?id=1294281
- **RAMCloud**: http://dl.acm.org/citation.cfm?id=1965751 und http://dl.acm.org/citation.cfm?id=2043560

Damit wird auch klar, was die Großen (vor allem Google) machen!

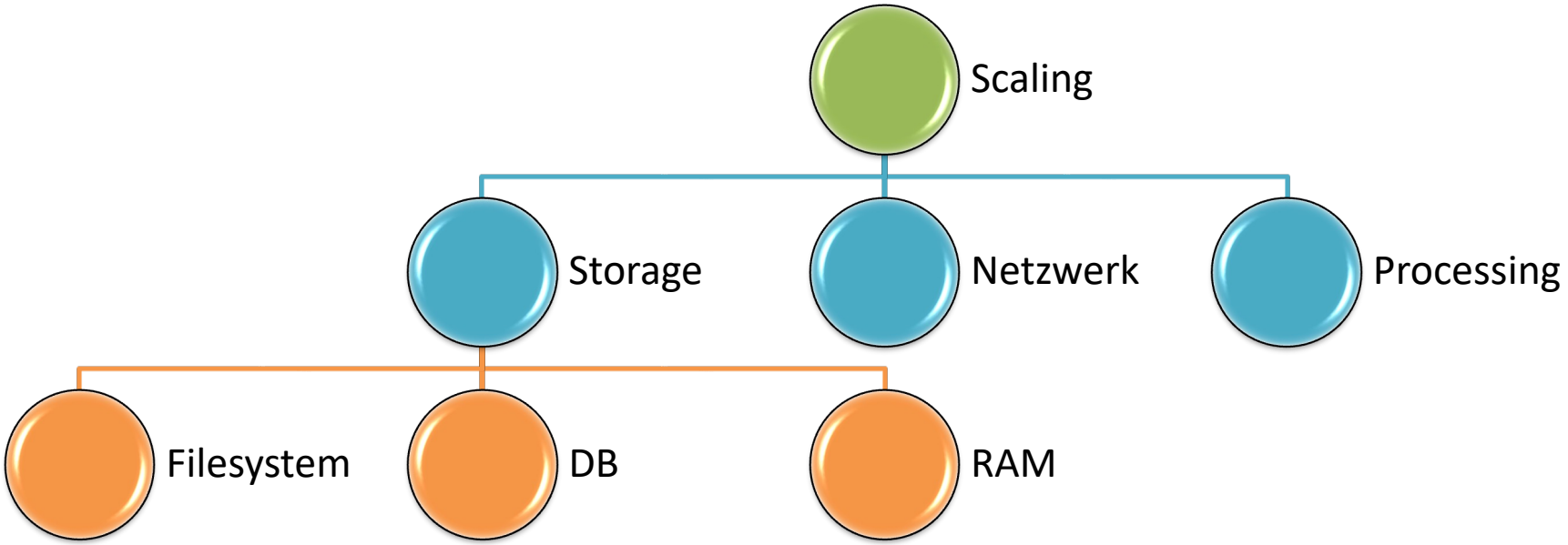
http://arstechnica.com/business/2012/01/the-big-disk-drive-in-the-sky-how-the-giants-of-the-web-store-big-data/

http://static.googleusercontent.com/media/research.google.com/de//university/relations/facultysummit2010/storage_architecture_and_challenges.pdf

Nicht vergessen: CAP gilt. Siehe z.B.

http://cloudscaling.com/wp-content/themes/cloudscaling/assets/downloads/cloudscaling_whitepaper_converged_storage.pdf

Scaling Dimensionen



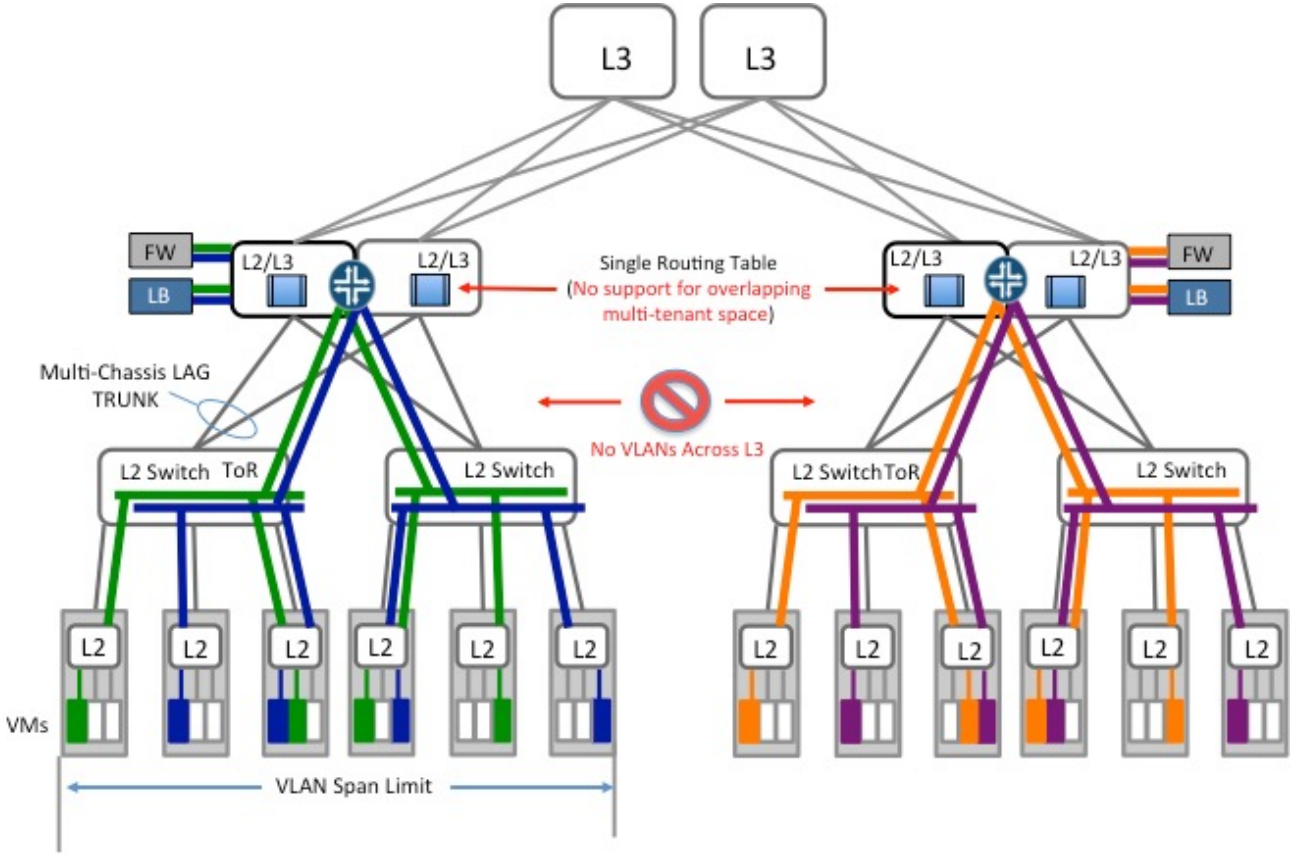
Wegweiser

Network Scaling

Das Netzwerk in IaaS und PaaS

- Große, sehr schnell veränderliche Mengen von virtuellen Netzen
- Internationale Verteilung
- Themenschwerpunkte:
  - Netzwerkvirtualisierung
  - Software Defined Networks (SDN)
  - Open Networking Foundation-> OpenFlow
  - Open Hardware & Software
  - P4 als DSL für Packet Processing

Netzwerkvirtualisierung (1) klassisches Datacenter mit VLANs

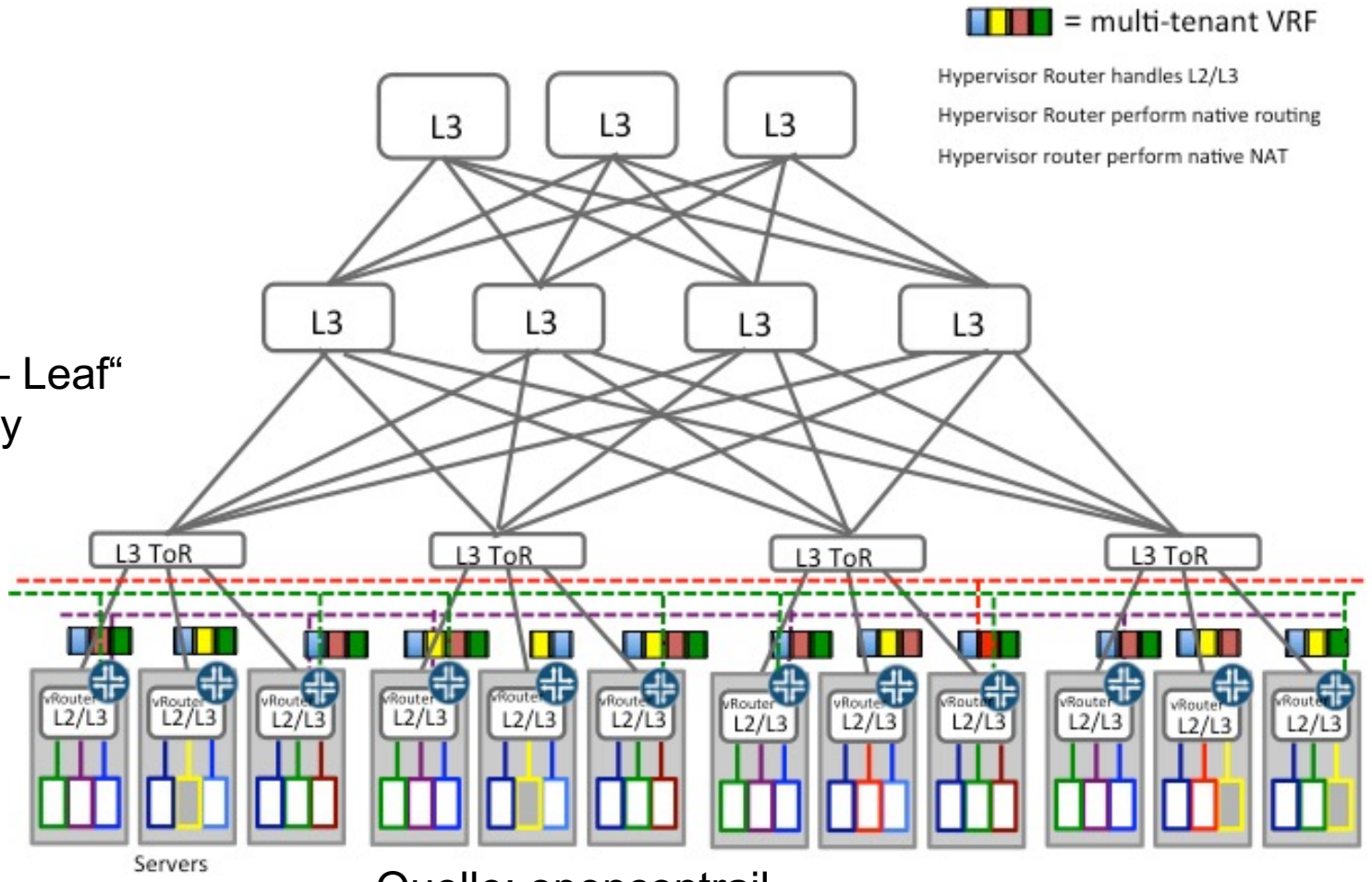


Quelle: opencontrail

Netzwerkvirtualisierung (2)

Cloud Datacenter mit virtueller Routingfunktion für viele Mandanten

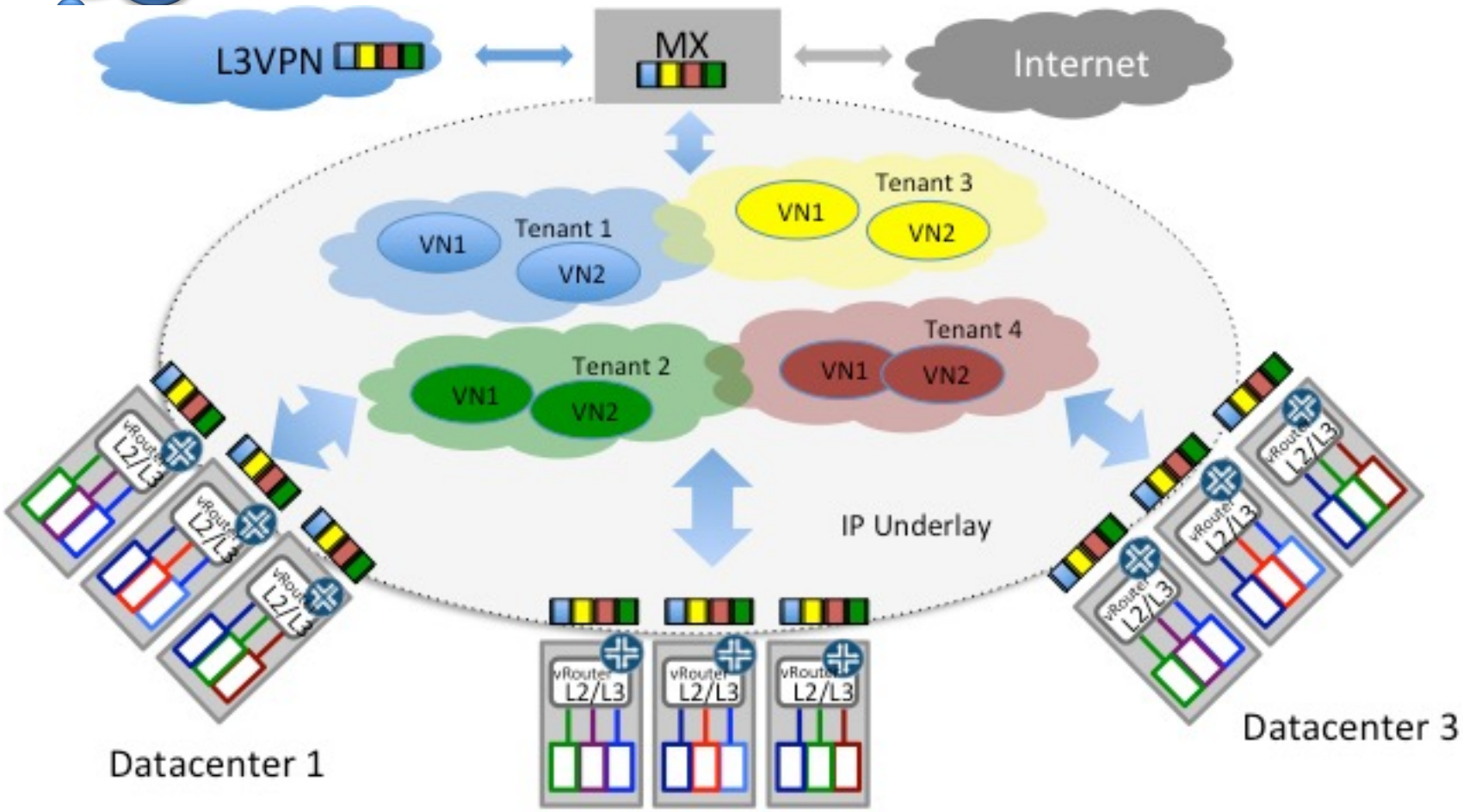
„Spine – Leaf“ Topology



Quelle: opencontrail

Netzwerkvirtualisierung (3)

Vielbenutzerbetrieb über mehrere Datacenter



Quelle: opencontrail

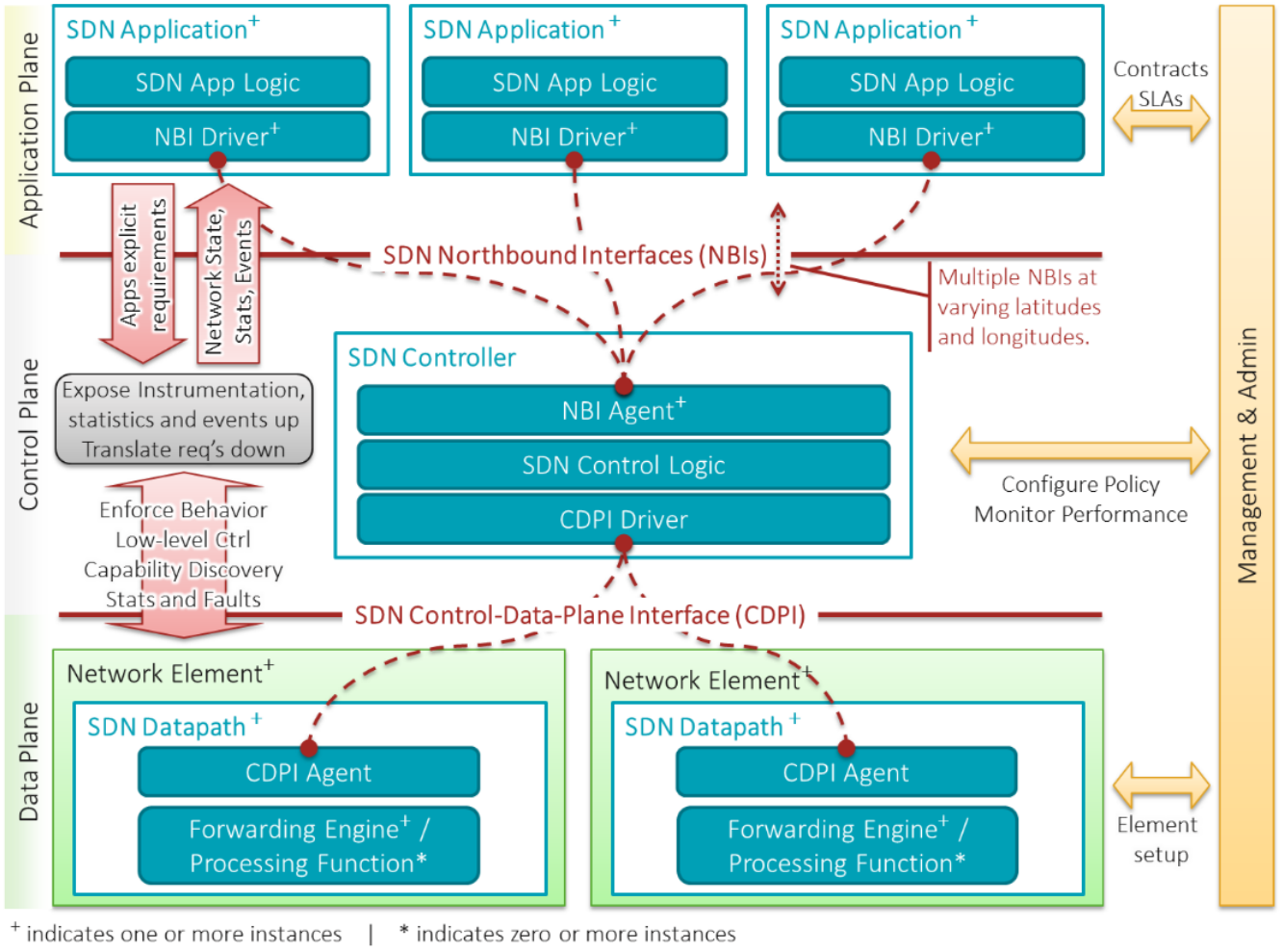
Software Defined Networks

- Scale-Out durch
  - Verschieben der Netzwerkvirtualisierung in L3 Soft-Switches
  - Nutzung von
    - L3 Dynamik: OSPF, BGP, Netconf, MP-BGP
    - L2 Tunneln o.ä.: MPLS, NVGRE, VXLAN, L2/L3 VPN, SPB, TRILL, LISP, STT
    - Offenen Plattformen auf Netzwerkelementen, z.B. Cumulus
  - protokollbasierte Kontrolle der Netzwerkkomponenten im Transport
  - Orchestrierung der Dienstleistungen (z.B. per XMPP)
- Dadurch
  - Mehrmandantenfähigkeit
  - Entkopplung des physikalischen Netzes
    - von der logischen Sicht
    - von Netzwerkelement-Migration und Lastverteilung
    - In Bezug auf die Bedeutung von Fehlern: Isolation!
- Fortschreitende Standardisierung

SDN Prinzipien

- Abstraktion
  - Kleine Menge von Zugriffsmethoden
  - Protokoll-unabhängig, in Hardware gemacht
  - Policy unabhängig
- Offene APIs und Programmierbarkeit
  - Forwarding / Kontrolle entkoppelt, OpenFlow
  - Neue Features leicht einzuführen
- Globale Netzwerksicht
  - Provisionierung vereinfacht
  - Problemlösungen beschleunigt
  - Virtualisierte Netzwerksicht

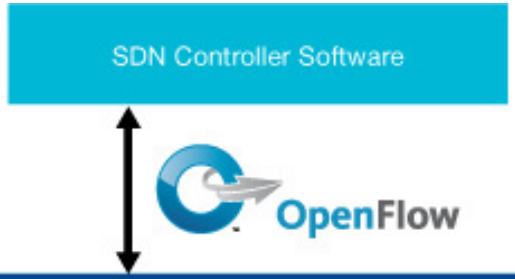
SDN Architektur

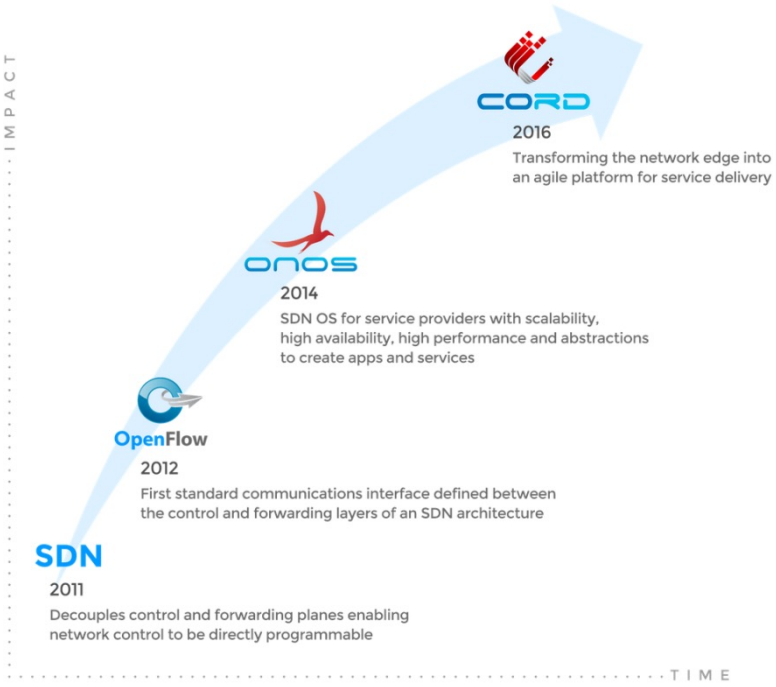


Quelle: Open Network Foundation

OpenFlow

Zentrales SDN Protokoll der Open Networking Foundation





OpenFlow-enabled Network Device

Flow Table comparable to an instruction set

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Quelle: https://www.opennetworking.org/sdn-resources/sdn-library/whitepapers

P4 als SDN DSL

- Packet Processing Domain Specific Language
  - Targets: Netzwerkelemente aller Art (ASIC bis Software)
  - Protokollunabhängig
  - C-Style

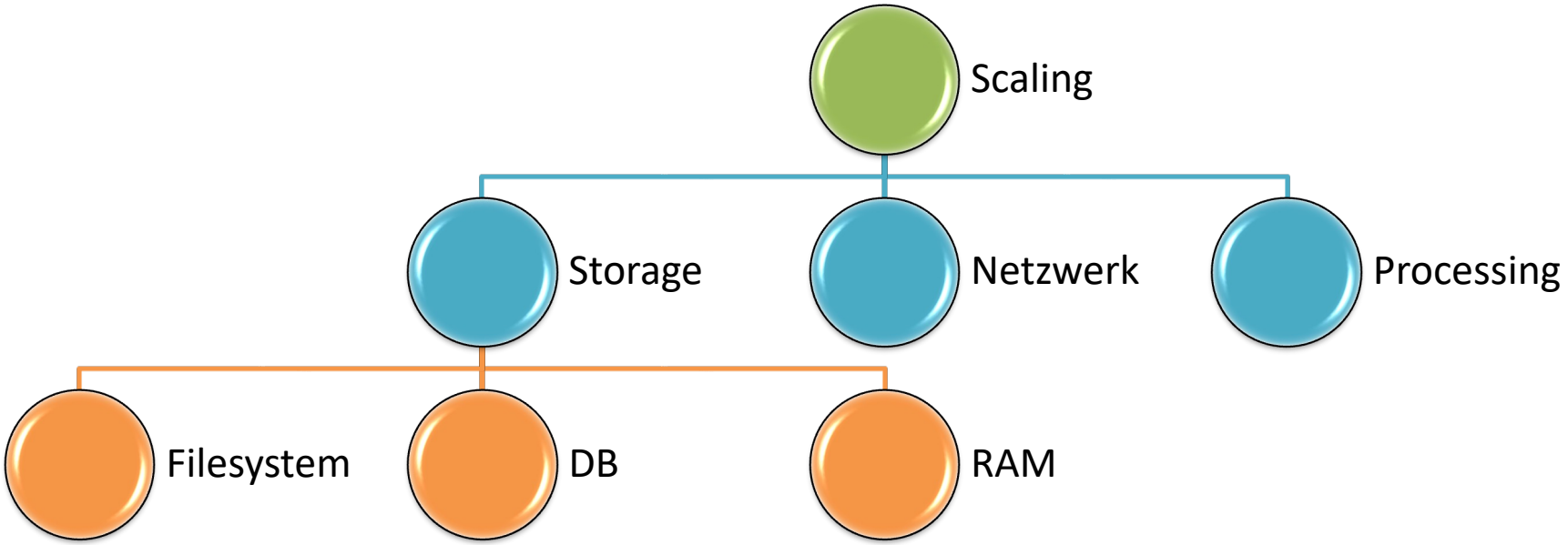
```
control ingress {
    apply(port);
    apply(host_ip) {
        miss {
            apply(lpm_ip);
        }
    }
    if (valid(vlan_tag[0])) {
        apply(port_vlan) {
            hit { apply(remap_vni); }
        }
    }
    apply (bridge_domain);
    if (valid(mpls_bos)) {
        apply(mpls_label);
    }
    retrieve_tunnel_vni();
    if (valid(vxlan) or valid(nvgre)) {
        apply(dest_vtep);
        apply(src_vtep);
    }
    . . . .
}
```

Weiterführende Quellen

- Beispiel: Google
  - https://www.wired.com/2012/04/going-with-the-flow-google/
  - http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf
  - http://www.ietf.org/proceedings/84/slides/slides-84-sdnrg-4.pdf
  - https://sdn-lab.com/2017/08/16/espresso-more-insights-into-googles-sdn/

- Try this: https://www.youtube.com/watch?v=l-DcbQhFAQs

Scaling Dimensionen



Wegweiser

CPU Scaling:

Virtualisierung

Motivation

- Kosten (Auslastung Optimieren)
  - Viele „alte, kleine“ Dienste werden parallel auf modernen HW zusammengefasst
- Ressourcen kontrollieren, flexibel nutzen
- Verfügbarkeit erhöhen ohne SW Änderung
- Snapshots
- Umzug

Eine Virtualisierung kommt selten allein…

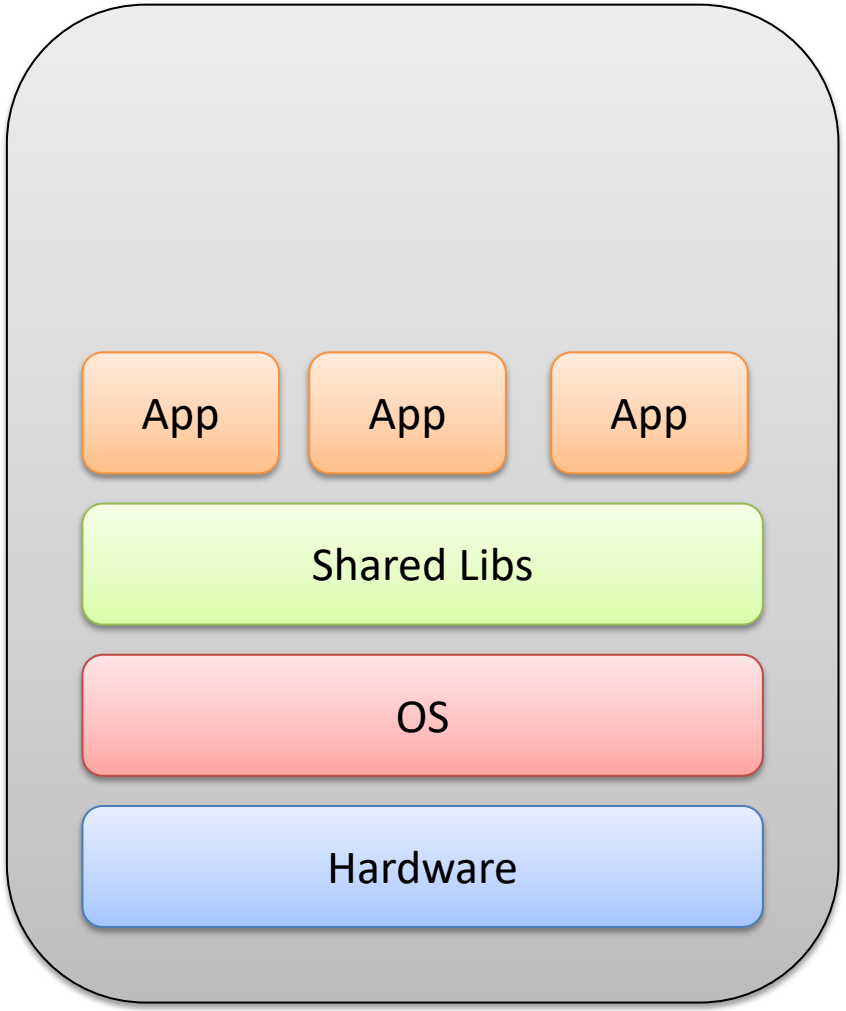
- Lösungen zur CPU Virtualisierung virtualisieren auch:
  - Speicher
  - Festplatten
  - Netzwerk
  - Grafik
  - Audio
  - USB
  - …
- … um eine vollständige Umgebung zur Verfügung zu stellen
- … und das für Systeme oder auch nur für einzelne Prozesse

Definition

- Host:
  - Die Betriebssystem Instanz welche direkt auf der HW läuft
  - Stellt alle HW Ressourcen zur Verfügung
  - Hat die „Kontrolle“ über die HW Ressourcen
- Gast / Gäste:
  - Die Virtualisierte (Laufzeit-)Umgebung welche über eine Abstraktion die HW-Ressourcen nutzt
  - Unterschiedliche Ebenen von Abstraktion – abhängig davon sind eventuell Änderungen am Gast notwendig um in der virtualisierten Umgebung zu laufen.

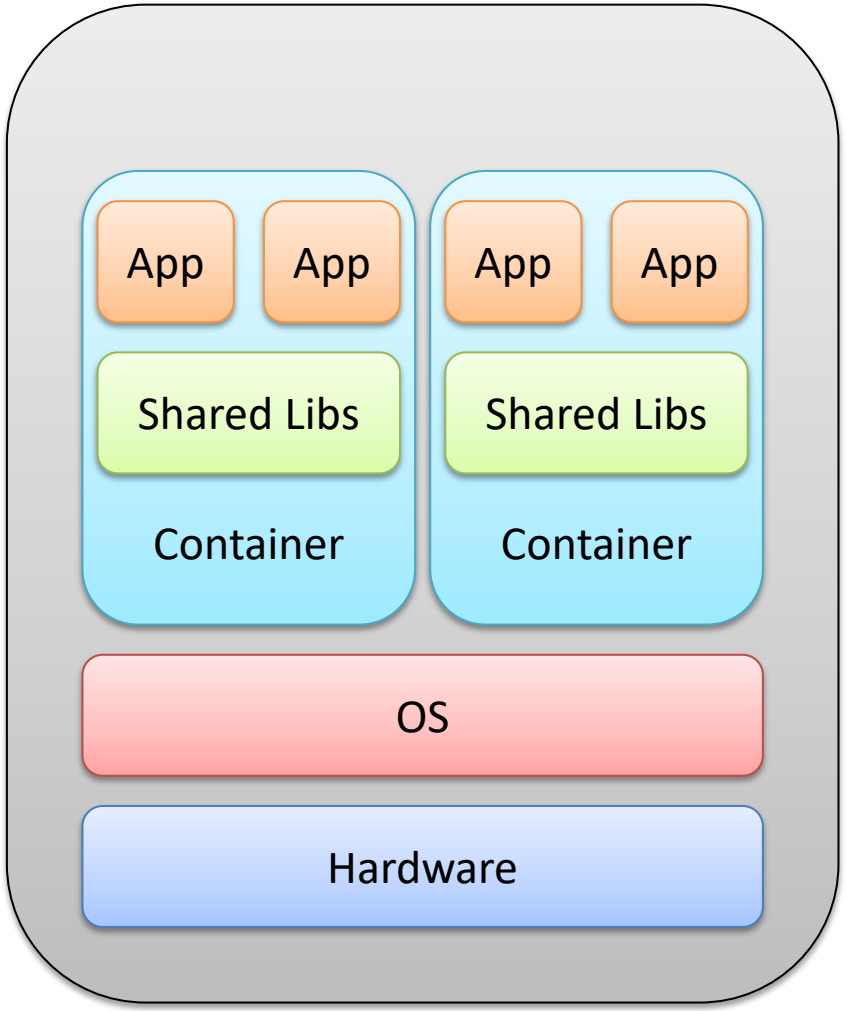
Was bisher geschah:

- Anwendungen laufen parallel auf einem OS, in getrennten Speicherräumen
- Geteilt: Libraries, OS, Treiber, (CPU)
- Getrennt: Register Sets, Speicher / Adressraum, Umgebungsvariabeln…
- Technologien: Prozess Scheduler, Virtueller Speicher, CPU Support für Kontextwechsel
- Bsp: Moderne Betriebsysteme



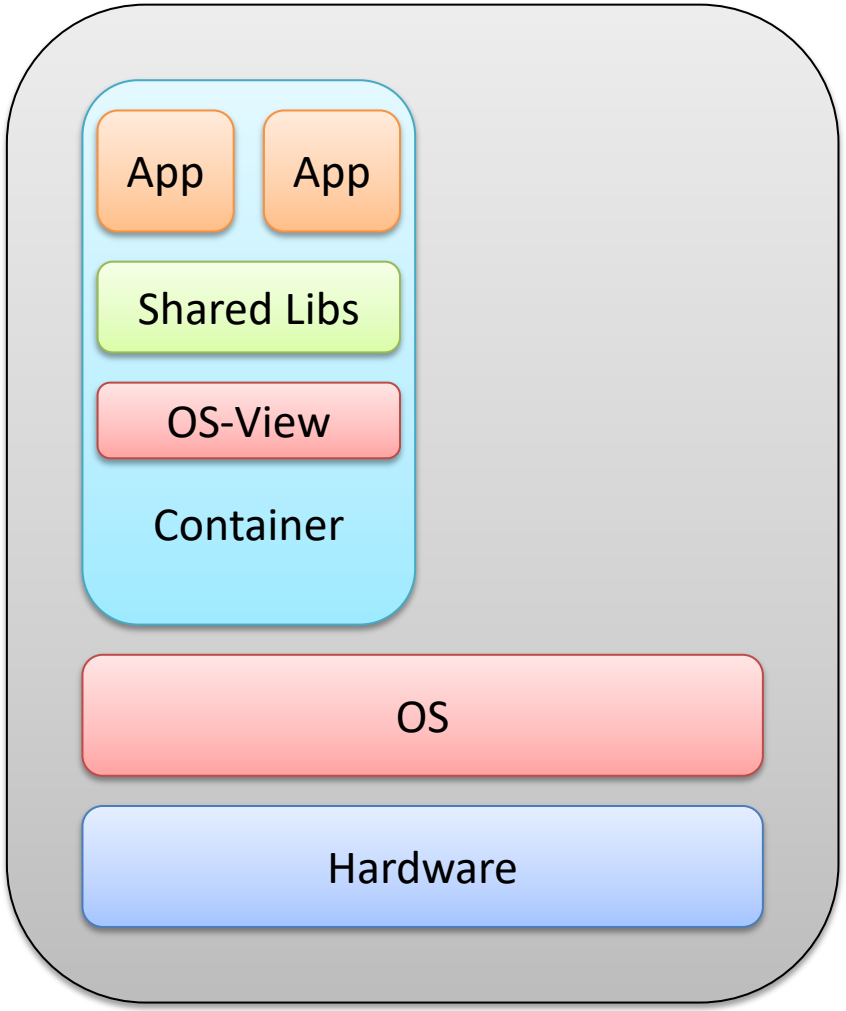
Applikations Virtualisierung

- Anwendungen laufen parallel auf einem OS – in einer eigenen Laufzeitumgebung (Jails, Container)
- Geteilt: OS, Treiber
- Getrennt: Libraries
- Technologien: JIT-Compiler, JVM,
- Bsp: chroot, Tomcat



Container Virtualisierung

- Wie Applikationsvirtualisierung aber stärkere Isolierung der Container:
  - Namespace Isolation: Prozess Ids, User Ids, Netzwerk Karten, Mounts, Interprozess Kommunikation, etc.
  - Ressource Management für CPU, RAM, I/O: Limitierung, Priorisierung, Accounting
- Viele Instanzen gleichartiger Systeme (Root Server, Virtuelle Desktops / Thin Clients)
- Linux: LXC, OpenVZ (Linux Containers) Cgroups, Docker

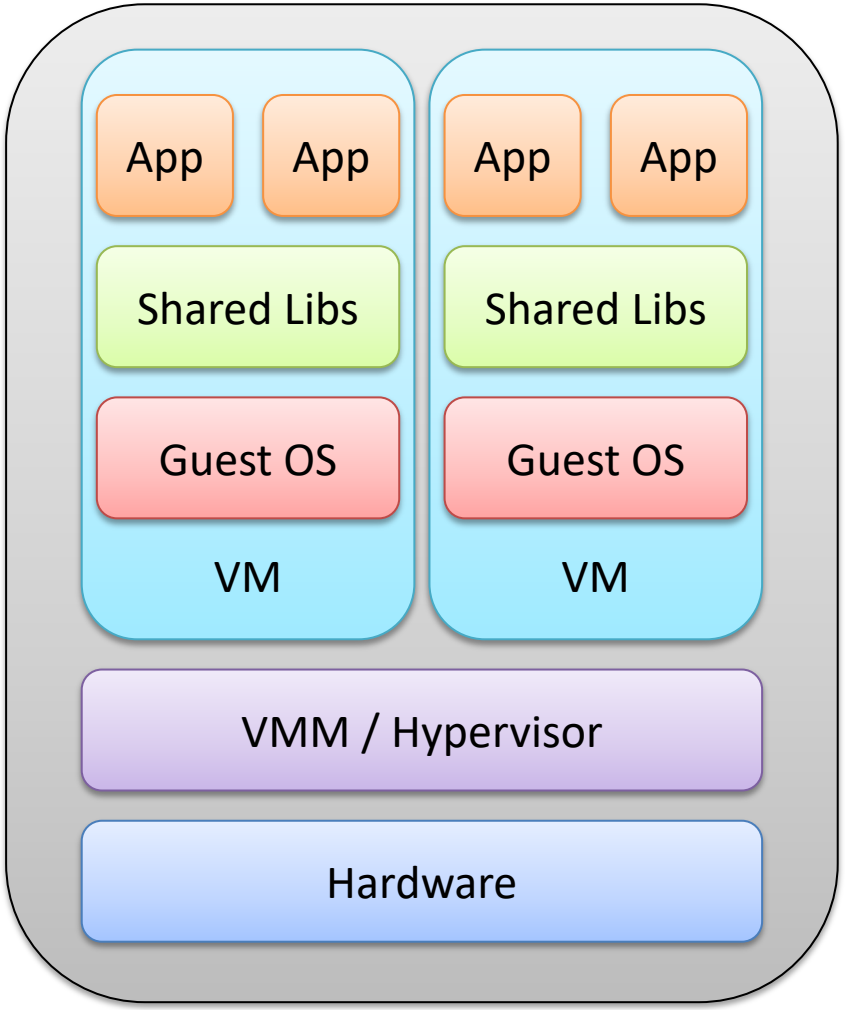


HW Virtualisierung: Virtual Machine Monitor (VMM)/ Hypervisor

- Aufgaben:
  - Abstraktion der HW
  - Verwaltung der VMs (anlegen, start, stop, snapshot, ressource Zuweisung, …)
- Position des VMM:
  - „Bare Metal“
  - Hosted
- Implementierungsvarianten / Grad der Virtualisierung:
  - HW-Emulation
  - Voll Virtualisierung
  - Paravirtualisierung
  - Virtualisierungstechniken werden oft gemischt z.B. HW Virtualisierung mit Paravirtualisierten Treibern

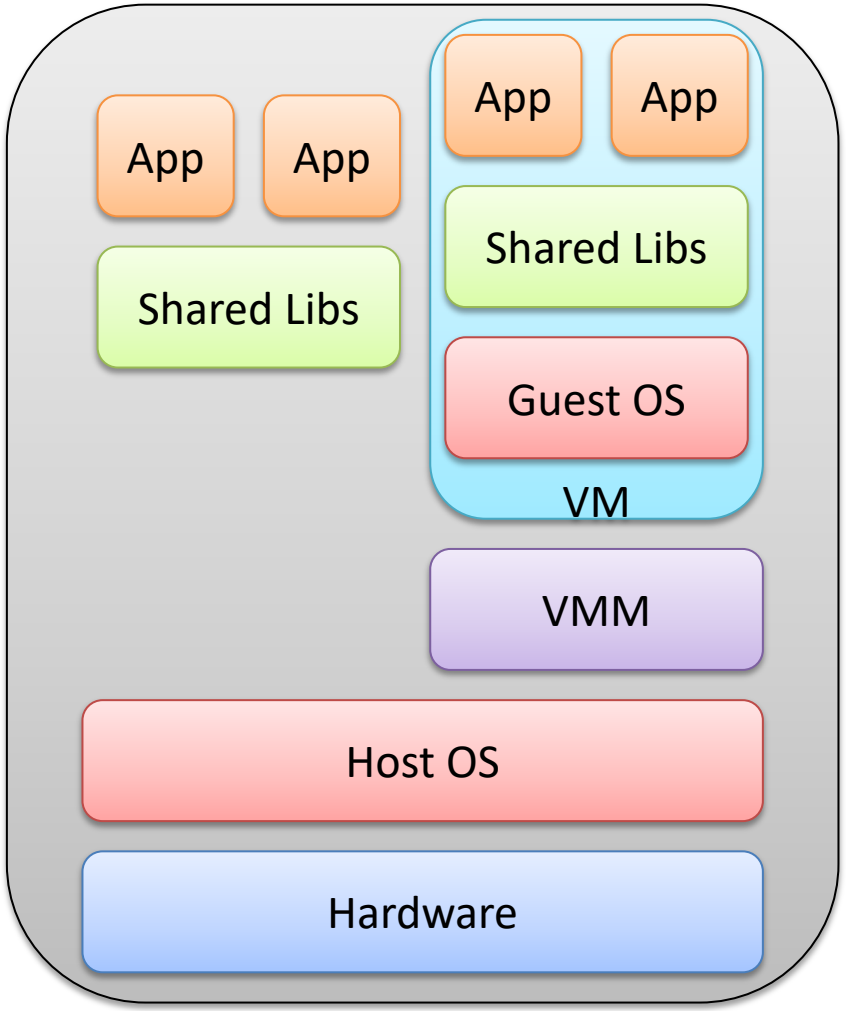
HW Virtualisierung: Bare Metal (Typ 1)

- VMM / Hypervisor verwaltet die HW direkt – kein weiteres OS
- Vorteile:
  - Bessere I/O Leistung
  - Mögliche Unterstützung für Echtzeit Betriebsysteme
- Nachteile
  - Benötigt eigenen Treiber für alle HW Komponenten die den VMs angeboten werden => limitierte HW Auswahl
  - Installationsaufwand
- Beispiele:
  - VM-Ware ESXi, MS Hyper-V



HW Virtualisierung: Hosted (Typ2)

- VMM / Hypervisor läuft auf Basis von Host Betriebsystem
- Benutzt Treiber des Host OS
- Beispiele:
  - Virtual Box, VMware Workstation, Virtual PC (ab Windows 7 enthalten)
- „Grauzone“:
  - KVM (Kernel Based VM)



X86 Architektur: „Protected Mode“

- 4 Sicherheitsstufen (Ringe 0-3)
- Privilegien nehmen nach „außen“ ab
  - Ring 0 (Kernel Mode): darf „alles“, Speicher, Befehle
  - Ring 1,2: unbesetzt
  - Ring 3 (User Mode): darf System Befehle nur über SYSCALL/SYSENTER, INT 80 ausführen

X86 Architektur: Weitere Herausforderungen:

- Virtuelles Memory Management
- Interrupts
- Zeit
- Weitere Geräte (Netzwerk, HD, etc)

Emulation

- Komplettes Abbild der HW Umgebung in SW
- Jeder Befehl der VM wird in SW übersetzt
- Eine oder verschiedene HW Architekturen können parallel emuliert werden – z.B. PPC/ARM/x86 …
- Nicht sehr performant, JIT möglich
- Beispiel: QEMU, MAME

Vollvirtualisierung

- Wie Emulation
- Optimierung bei gleicher HW-Architektur GAST/HOST => nur die privilegierten Befehle werden zur Laufzeit „übersetzt“ => „Binary Translation“ / „Trap and emulate“
- VMM Läuft auf:
  - Ring 0 (TYP1) und somit alle Gast-OS auf Ring 1
    Privilegierte Befehle per „Trap and emulate“
  - Ring3 (Typ2) und somit der Gast auch in Ring 3
    Privilegierte Befehle per „Binary Translation“
- Keine Änderung an den Gast-OS
- VM Performance ca. <=25% geringer als Host-native

Paravirtualisierung

- Hypervisor stellt API für „Systemcalls“ => Hypercalls zur Verfügung
- Gast OS wird modifiziert - privilegierte Befehle werden in „Hypercalls“ verwandelt
- Spezielle Treiber im Gast OS
- Beispiel: Xen

Beispiele für HW unterstützte Virtualisierung

- VMM in Ring -1:
  - Intel: Vanderpool, VT-x
  - AMD: Pacifica, AMD-v
- PCI-Paththrough, I/O Memory Mapping:
  - Intel VT-d bzw. VT-c
  - AMD Vi
- Grafik
  - Intel GVT-d, -g, -s
- Hardware Assisted Paging (HAP) / Nested Page Table (NTP):
  - Intel Extended Page Table
  - AMD Rapid Virtualization Indexing