Fachhochschule Nordwestschweiz
Hochschule für Technik

## Application Performance Management

# Clustering & High Availability

Michael Faes

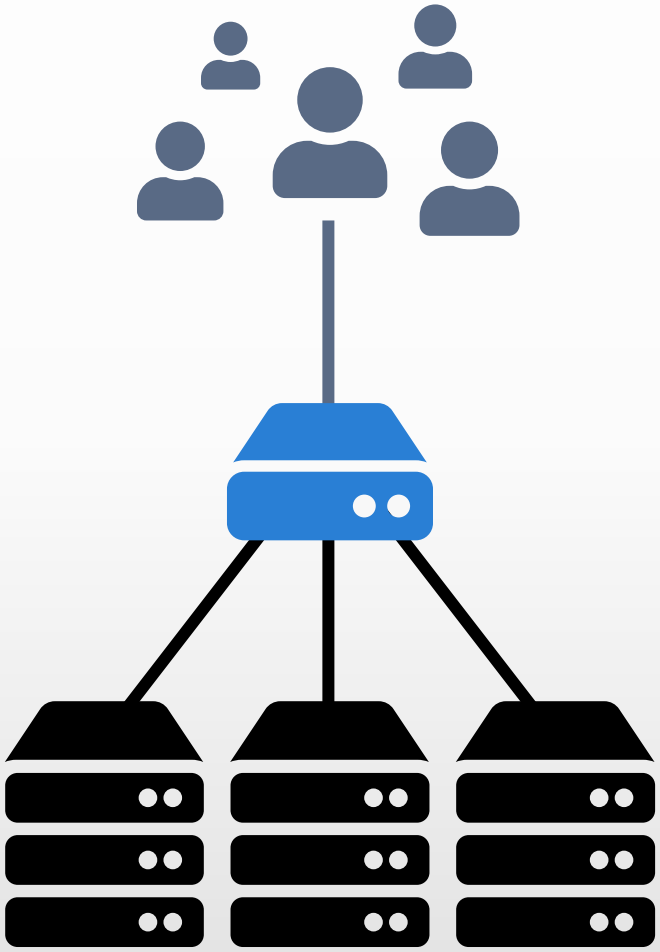
# Rückblick: Load Balancing

Verteilen von Requests auf mehrere Server

Neue Komponente: *Load Balancer*

Herausforderungen:

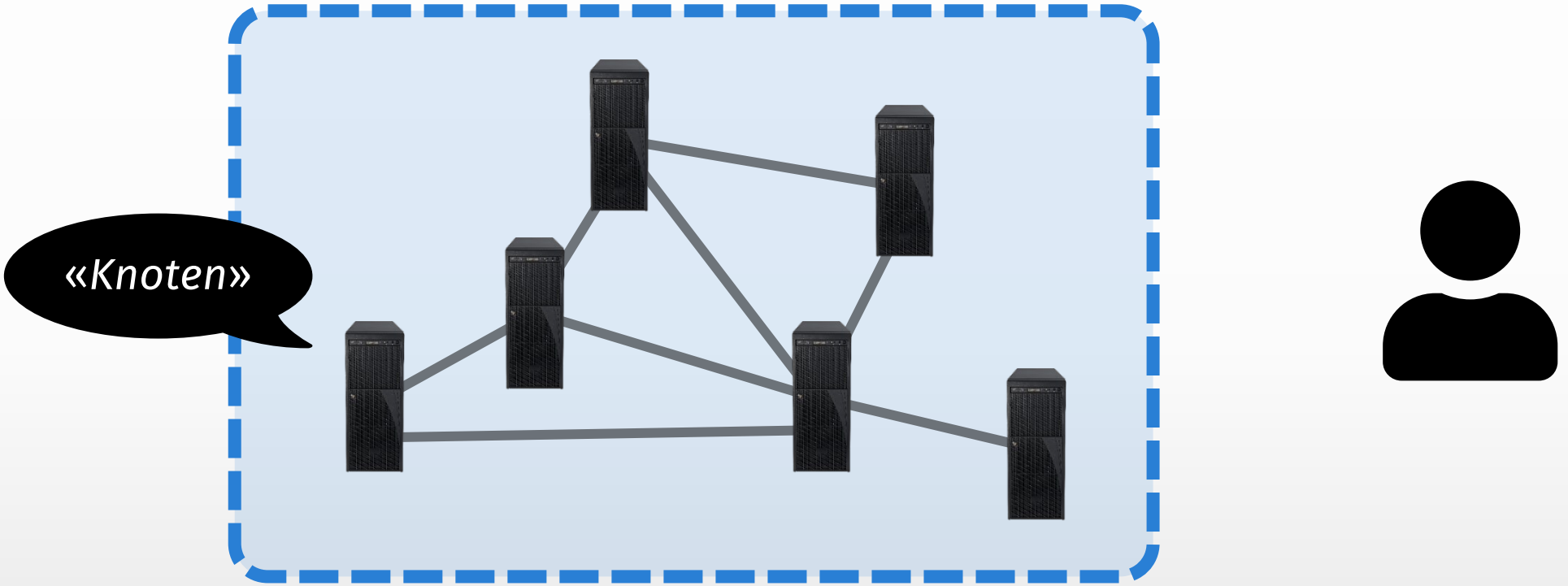
- Monitoring
- Persistenz
- Heute: **Verfügbarkeit**



# Übersicht Woche 10

1. Besprechung Assessment 1
2. Rückblick Load Balancing
3. Clustering & High Availability
   - Failover
   - Anwendungsanforderungen
   - Virtualisierung und Container
   - Cluster-Speichersysteme
4. Übung

# Was ist ein «Cluster»?




***Cluster***: Mehrere vernetzte Rechner (*Knoten*), die zusammen arbeiten, und die man im Prinzip als ein System ansehen kann.

Sicht vom Benutzer (von aussen)

# Arten von Clusters

*Compute Cluster*

- Rechenpower für High-Performance Computing
- Applikationen werden explizit parallelisiert

*Load-Balancing Cluster*

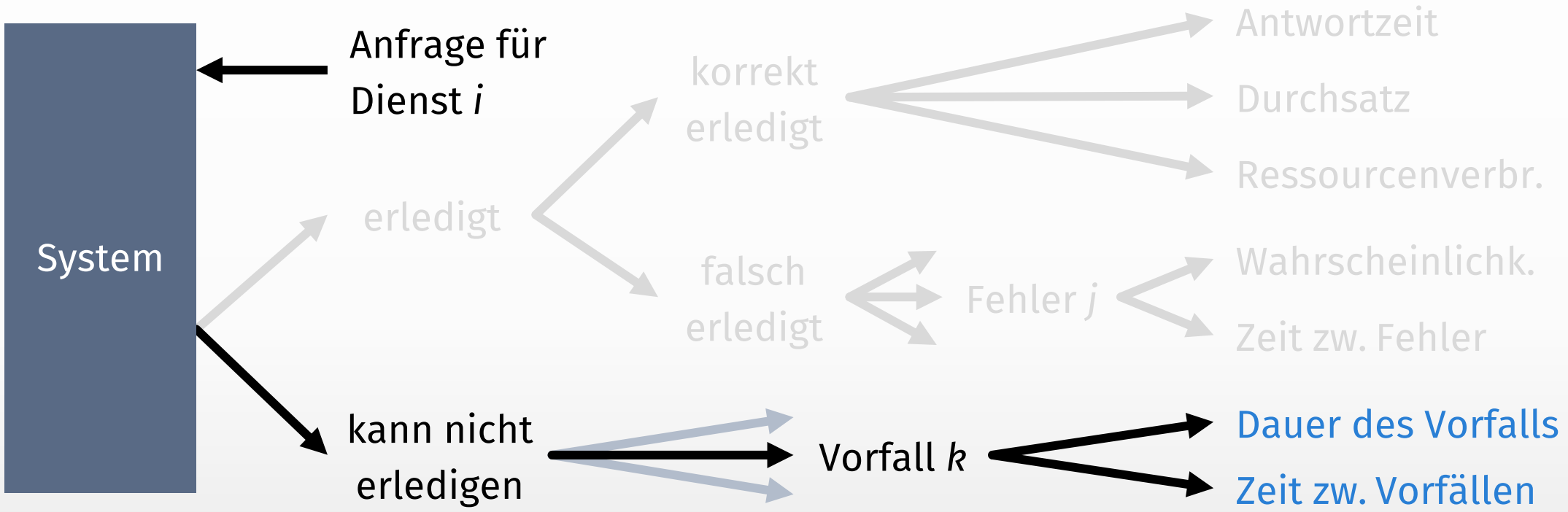
- Verteilen von «gleichartigen» Requests
- In unserem Setup: die Web-Server

***High-availability (HA) Cluster***

- Wenn Knoten ausfällt, übernimmt ein anderer
- In unserem Setup: noch nötig für die Load Balancers

(high availabilty)

# Rückblick: Performance-Metriken

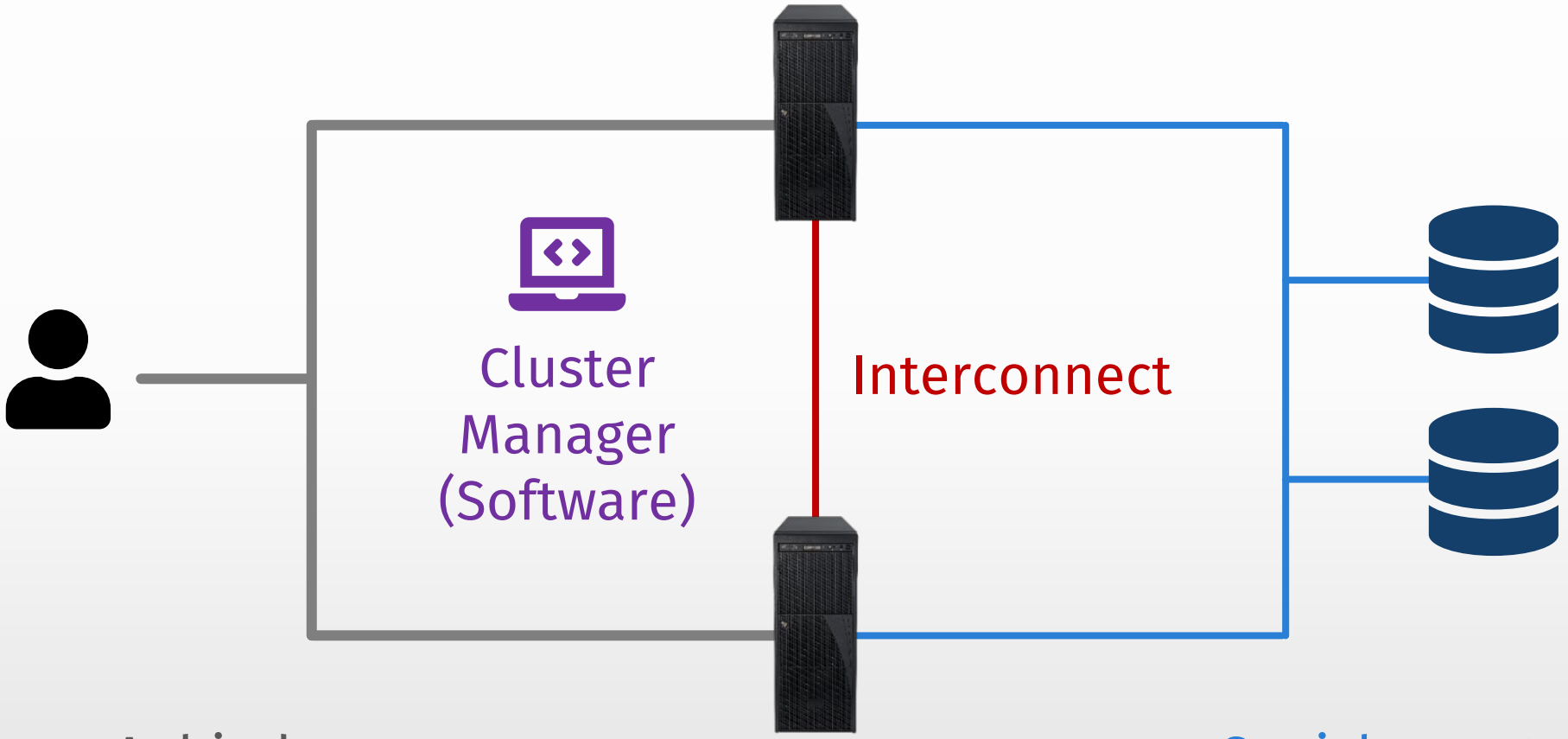



***Verfügbarkeit*** (availability) (allgemein)

*Anteil der Zeit, während der das System Anfragen beantwortet, z. B. 99%*

# Komponenten eines HA-Clusters

mind. 2 Konten, ansonsten kein Cluster




Anbindung
an Internet

Speichersystem

Knoten einer der beiden ist aktiv und der andere passiv (stand-by). Sobald also der aktive Knoten offline geht, wird der Passive aktiv

# Failover

Interconnect wird für *Heartbeat* verwendet: regelmässiges Signal, welches anderen Knoten mitteilt, dass ein Knoten noch verfügbar ist.

- Sobald ein Knoten den Heartbeat eines anderen eine bestimmten Zeitraum nicht mehr erhält, übernimmt er dessen Rolle: *Failover*.

**Beispiel:** Zwei Load Balancers, ein Primär-/Master- und ein Sekundär-/Backup-Load-Balancer. Sobald Master ausfällt, *übernimmt* Backup.

**Herausforderungen**

- Alle Knoten müssen stets aktuellen Zustand haben/synchronisieren
- Andere Komponenten müssen über Failover «informiert» werden
  **Beispiel:** VRRP (siehe Übung)

# Anwendungsanforderung

1. Einfacher Weg, App zu starten, stoppen und Status abzufragen (automatisierbar!)
2. App muss geteilten Speicher verwenden können
3. Zustand der App muss möglichst oft und vollständig auf persistentem Speicher gesichert werden
4. Keine Datenkorruption bei Crash oder Neustart

5. ist die App verfügbar -> muss sichtbar sein

→ **Viele der Anforderungen können durch *Virtualisierung/Containerisierung* minimiert werden**

# Virtualisierung und Container

# Virtualisierung: Prinzip

ohne Virtualisierung

Applikation

Betriebssystem

Maschine

Virtualisierung:

Applikation | ... | ...

Betriebssystem | ... | ... «Guest OS»

Virtuelle Maschine | ... | ... «Guest»

Hypervisor

Reale Maschine «Host»

# Funktionsweise

Einfachste Technik: *Emulation*

- Gast-Instruktionen werden von Software *interpretiert*
- Kann beliebige Hardware virtualisieren (z. B. ARM auf x86-64 CPU)

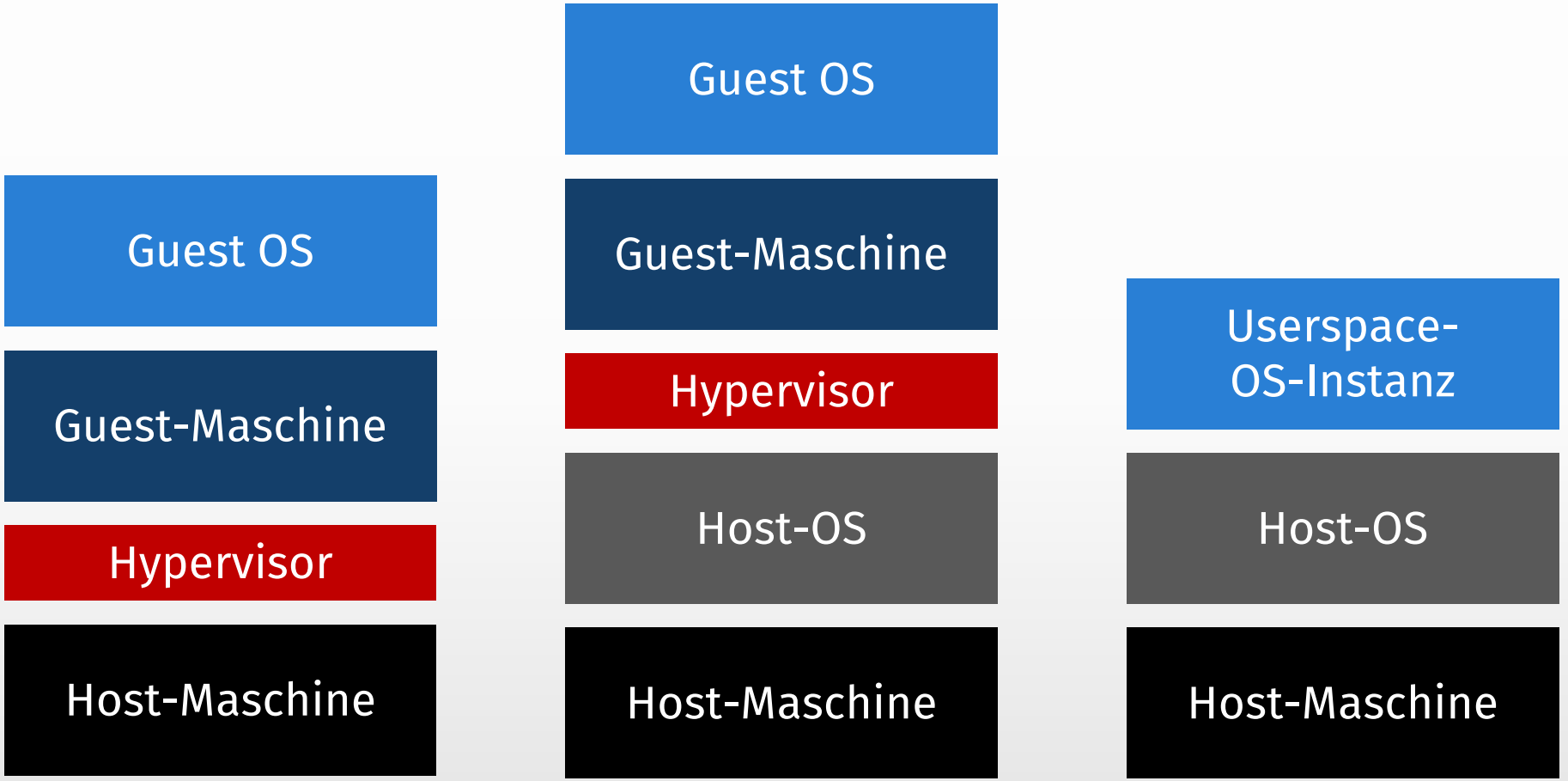
**Langsam!** Bei gleichem *Instruction Set* (z. B. x86-64), können Code-Stücke stattdessen direkt auf Host-CPU ausgeführt werden

- Nicht alle Instruktionen! Gewisse sind Kernel vorbehalten («Ring 0»)

*Hardware-assisted Virtualisation* erlaubt noch effizientere Ausführung

- CPU gaukelt dem Gast vor, er würde in Ring 0 laufen, aber schützt Host-OS von unerwünschten Änderungen
- Beispiele: Intel VT-x, AMD-V

# Arten von Virtualisierung




«Typ-1»
(Bare-Metal) Hypervisor

«Typ-2»
Hypervisor

Containerisierung

z. B. Docker

# Virtualisierung vs. Containerisierung

**Virtualisierung**

- Beliebige Host/Guest-Kombinationen möglich (z. B. x64-Windows / ARM-Mac)
- Images sind portabel
- Images sind schwergewichtig, da vollständiges OS enthalten
- Langsamer Start durch Booten
- Sicherheit durch Guest- & Host-Kernel & Hypervisor

**Containerisierung**

- Guest-Architektur und -Kernel müssen zu Host passen (nur x64-Linux / x64-Linux)
- Images sind Plattform-spez.
- Images sind leicht, da nur App & Userspace-Abhängigk. drin
- Schneller Start
- Sicherheit: abhängig von Kernel und Konfiguration...

# Bedeutung für Clustering/Cloud

**Ressourcen-Pooling**

NIST: «*Computing-Ressourcen werden zusammengelegt, um mehrere Kunden mit denselben physischen Ressourcen zu bedienen.*»

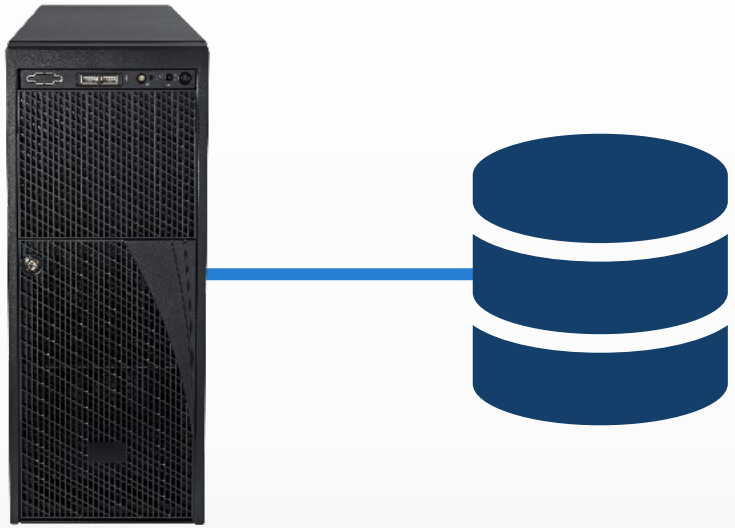
- CPU & RAM: Hypervisor kann Limiten für VMs/Container festlegen
- Speicherplatz: Zentralisierter Speicher für alle Gäste, wird nach Bedarf aufgeteilt
- *Thin Provisioning*: Speicherplatz wird Guest zugeschrieben, aber erst alloziert, wenn er wirklich verwendet wird
- *Deduplication*: Identische Blöcke von VMs werden nur 1x gespeichert

***Live Migration***

RAM-Inhalt und Netzwerk-Verbindungen können beibehalten werden, wenn VM von Host zu Host migriert wird

# Cluster-Speichersysteme

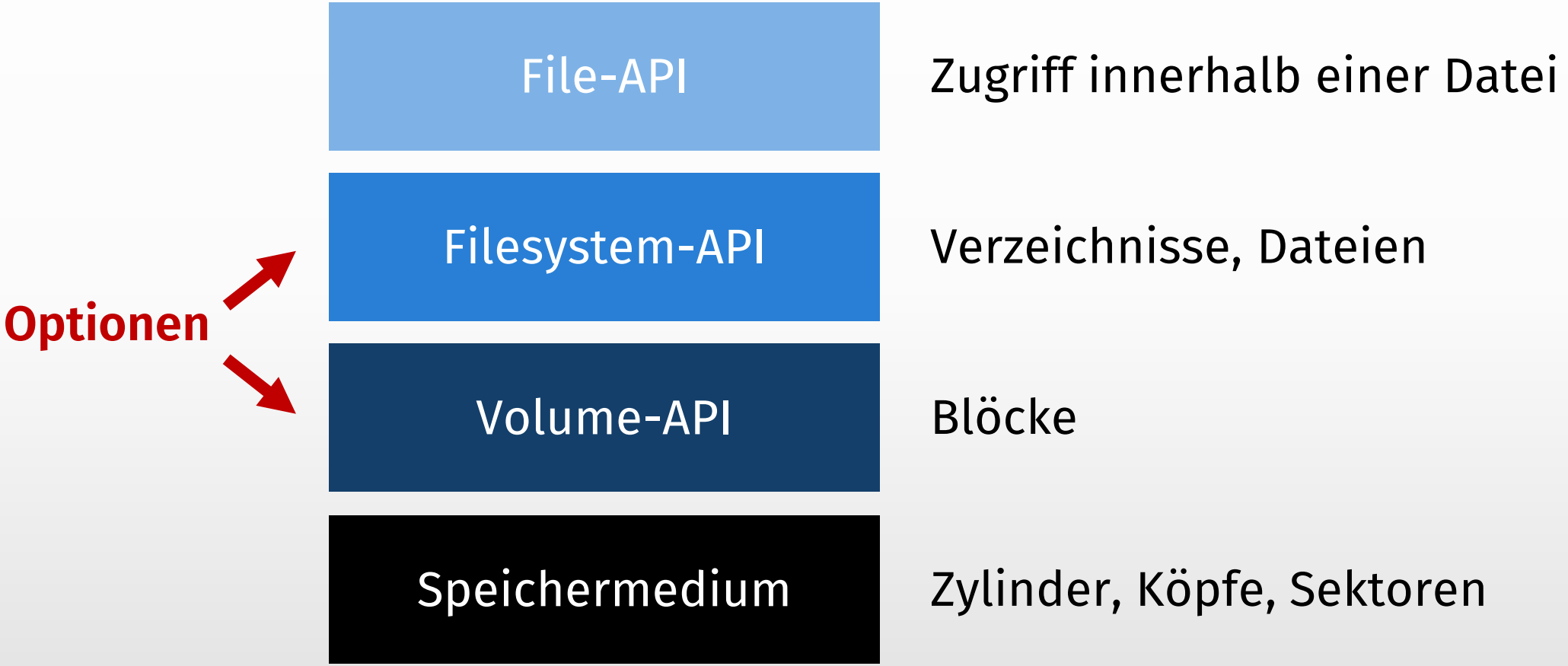
# Direct-Attached Storage (DAS)



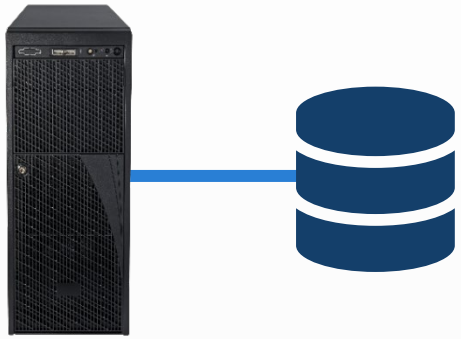
*Direct-Attached Storage*: Fancy Name für Festplatten, die direkt an Host angeschlossen sind

- Schnittstellen: SCSI, SATA, eSATA, USB, …
- *Block-basierte* Schnittstelle

# Speicher-APIs



# DAS: Vor-/Nachteile



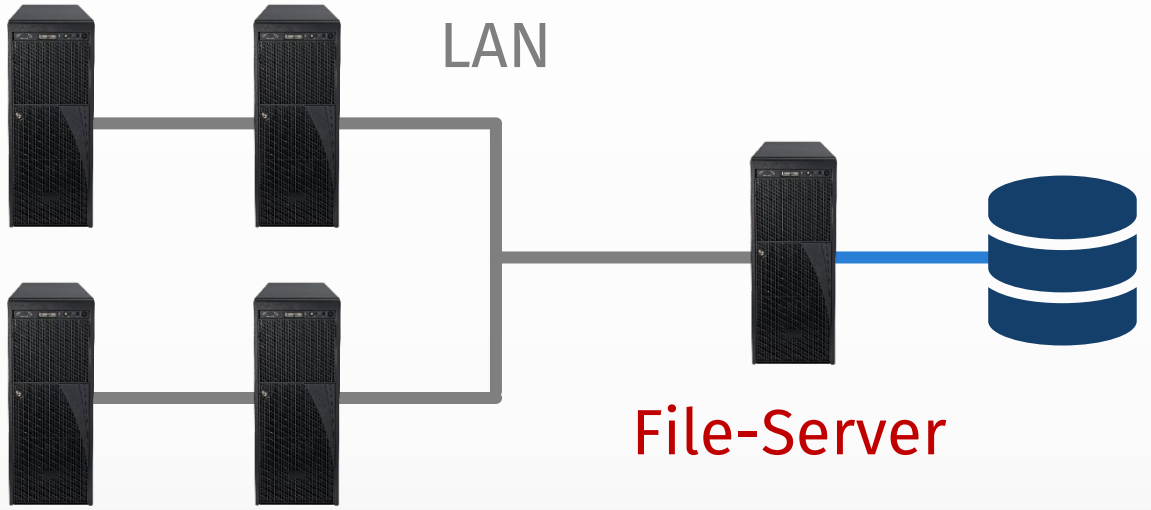
Super-einfach

Direkter Zugriff nur von einem Host möglich

Zugriff von anderen Hosts:

- Protokolle wie FTP
- Netzwerk-Dateisysteme wie NFS oder SMB
- Oder…

# Network-Attached Storage (NAS)




*Network-Attached Storage*: Dedizierter File-Server in LAN, Zugriff von allen anderen Hosts möglich

- Zugriff normalerweise über NFS oder SMB (d. h. über TCP/IP)
- Einfache Lösung, aber: limitierte Bandbreite, belastet LAN

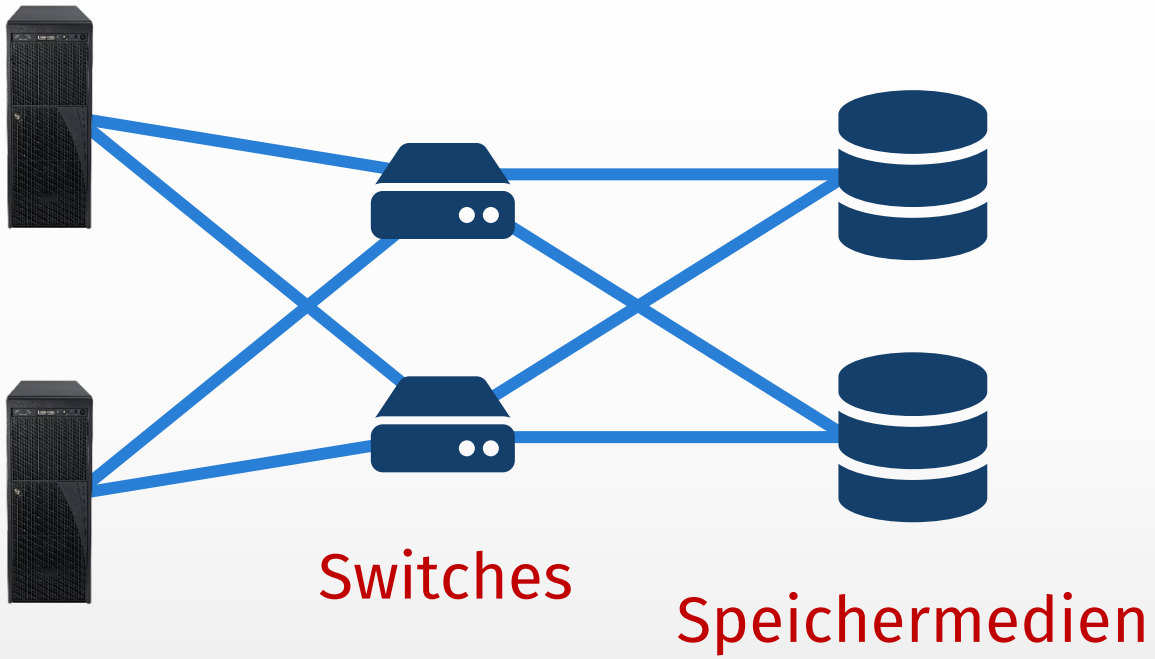
# DAS & NAS: High Availability?

Wenn Speichermedium an *einen* Host angeschlossen ist, **wie verhindern wir Single Point of Failure**?

Ansätze:

- Regelmässiges Spiegeln auf anderen Host?
- Verteiltes Dateisystem?!
- **SAN!**

# Storage Area Network (SAN)




*Storage Area Network*: Dediziertes Netzwerk für Speicher

Extra-Features: Automatisches Backup, Monitoring

# SAN: Eigenschaften

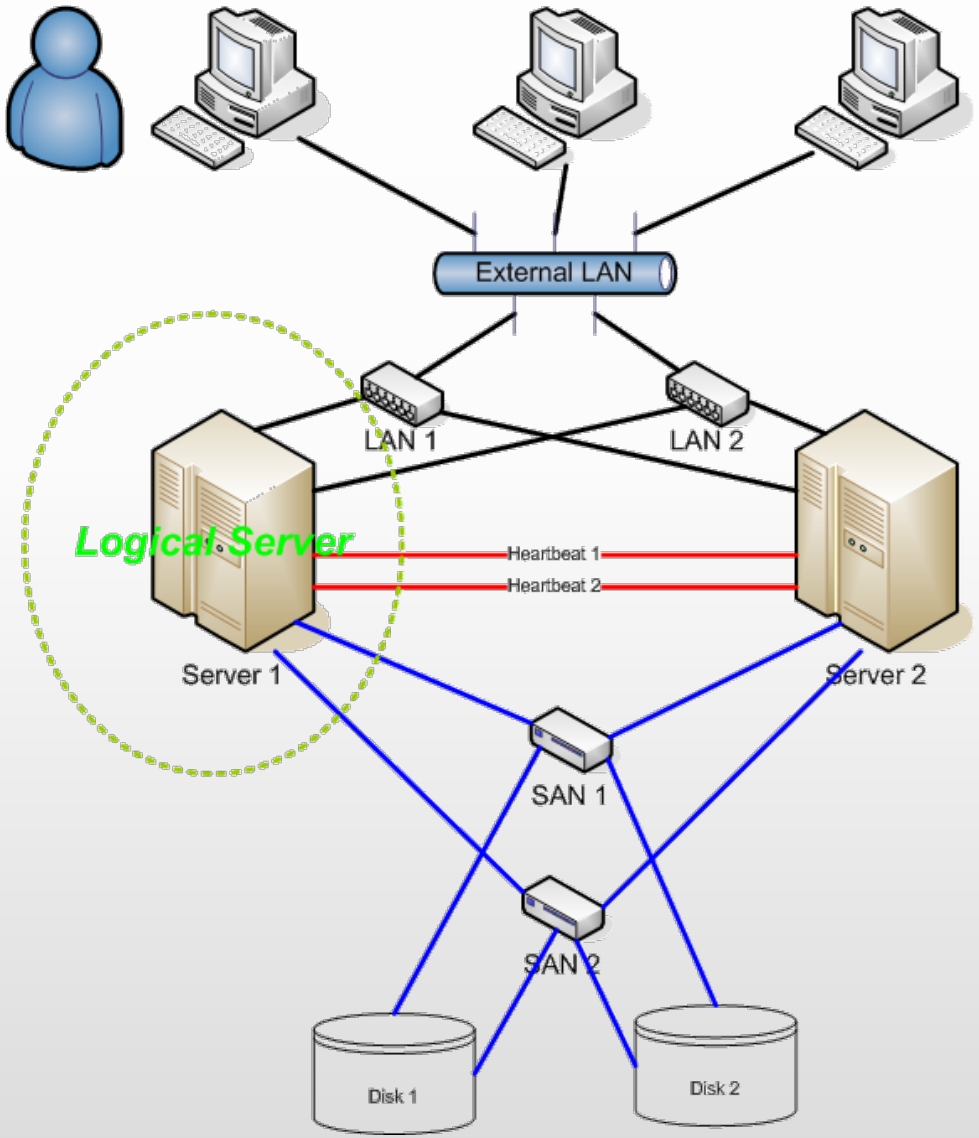
Spezielle Protokolle: *Fibre Channel*, *iSCSI*, *Infiniband*

- Darunter: Kupfer oder Glasfaser, für bis 10 km-Leitungen
- Höhere Bandbreite: z. Z. 50 Gbit/s pro Link, mit 4x, 8x, 12x…

**Zugriff ist block-basiert!** (Wie bei direct-attached storage)

- D. h. Dateisystem wird von Host implementiert
- Braucht spezielles *Shared Disk File System*
- Zugreifende Hosts müssen **Zugriff koordinieren**!

# Echte Redundanz



# SAN: HA-Alternativen

File System

Direct-attached storage mit (echtem) **verteiltem FS**

- Beispiel: Hadoop Distributed File System (Open Source)

Google

Für weniger Daten: **Verteilter In-Memory Store**

bleibt im Hauptspeicher der verschiedener Hosts

- Beispiel: Hazelcast (Open Source)
- Werden wir später einsetzen!

# Probleme mit Shared Storage

Nodes müssen genau wissen, welche anderen Knoten *alive* sind und ebenfalls auf Speicher zugreifen.

- SAN: Block-basierter Zugriff!
- Auch ohne SAN ein Problem: Datenkonsistenz

***Split Brain***

Links zwischen zwei Knoten sind down, aber Anbindung an öffentliches Netz und an Speicher steht noch

*Jeder Knoten meint, er sei der einzige und koordiniert nicht mehr!*

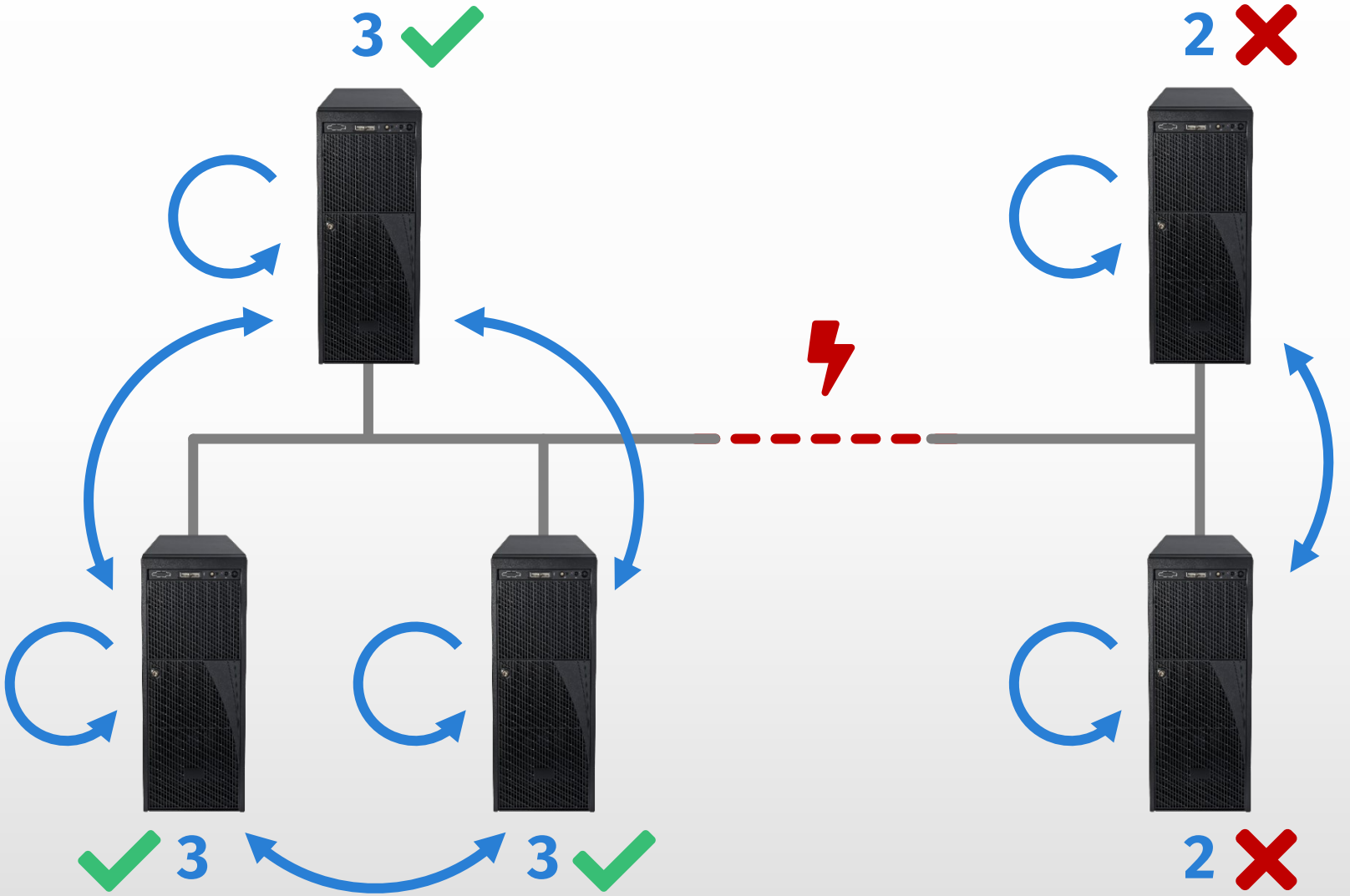
# Quorum

*Quorum*: Methode, um zu bestimmen, welcher Teil eines Clusters weiterläuft, wenn Links ausfallen

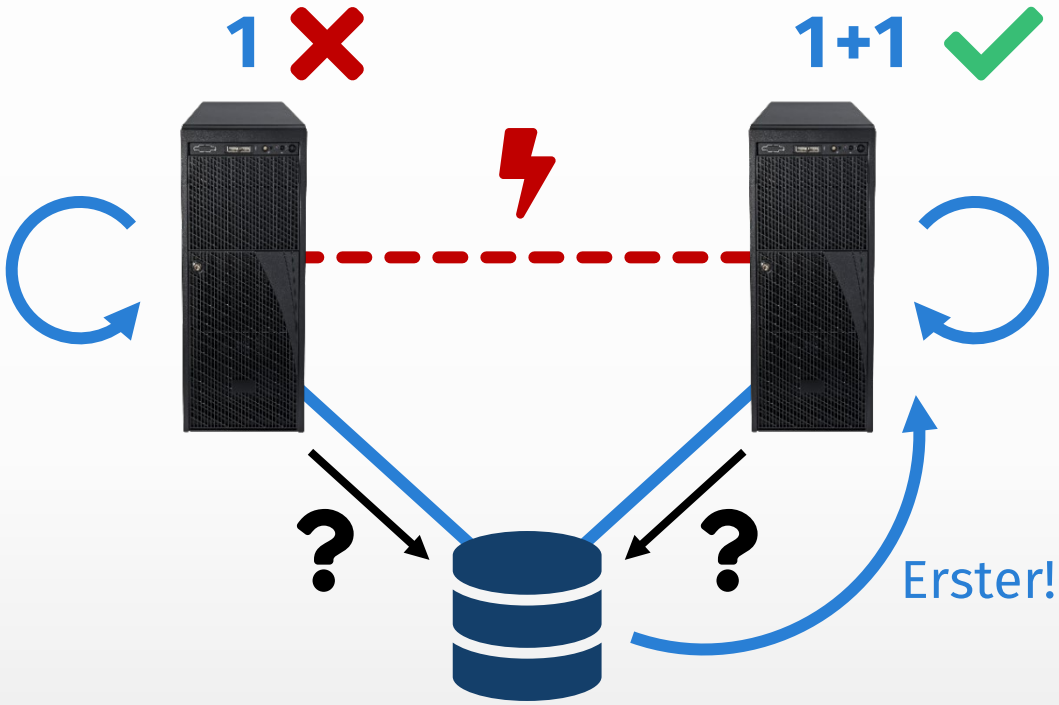
**Prinzip:** Abstimmung

- Jeder Knoten hat eine «Stimme»
- Knoten erhalten für jeden erreichbaren anderen Knoten dessen Stimme
- Jeder Knoten, der >50% der Stimmen hat, läuft weiter
- Alle anderen «schalten sich aus»

# Quorum: Beispiel



# Quorum: Gerade Anzahl?




**Lösung:** Zusätzlicher «Zeuge» (*witness*) mit Extra-Stimme: Wer die Stimme (als erstes) holen kann, gewinnt!

- Beispiele: Voting Disk (Quorum Disk) in SAN, oder File Share

# Fragen?