# NoSQL Datenbanken
## Vorlesung – Hochschule Mannheim

# NoSQL Basics

# Inhaltsverzeichnis

# Motivation: Big Data

Datenmengen, die zu groß oder zu komplex sind oder sich zu schnell ändern, um sie mit händischen und klassischen Methoden der Datenverarbeitung auszuwerten

(Wikipedia)

- NoSQL Datenbanken werden genutzt um Big Data zu speichern
- 2012 zum Trend geworden (durch BITKOM)
- 2014 Leitthema der CeBIT
- Big Data wird durch vier “V“s beschrieben

# Big Data

- **Volume**

Mehrere Tera bis Exabytes an Daten zur Verarbeitung und Analyse

- **Velocity**

Verarbeitung und Ausgabe von Daten in Echtzeit

(Echtzeitstreaming von Finanzdaten)

# Big Data

- **Variety**

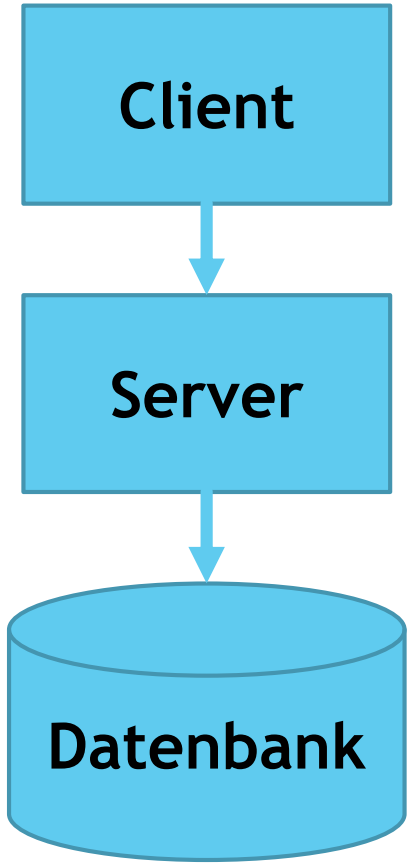
Daten liegen teilweise strukturierte Weise aber teilweise auch unstrukturiert vor

- **Veracity**

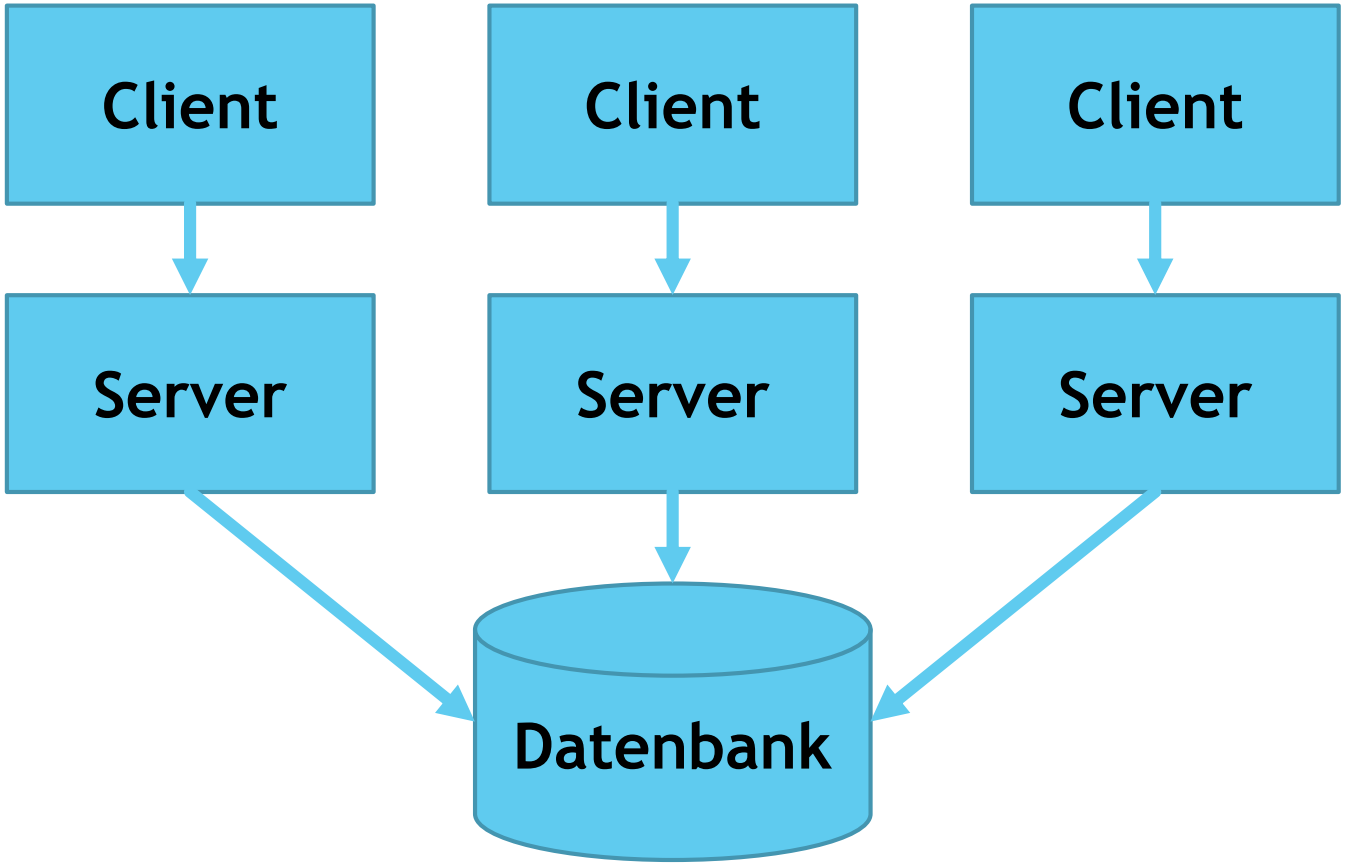
Daten liegen nicht vollständig vor:

- inkonsistent
- interpolierte Daten

# Bisher



# Verteilte Architektur



# Skalierungsarchitekturen

# Scale up vs Scale out

## Scale up:

Wenige große Server mit hoher Rechenleistung

## Scale out:

Viele kleine (günstige) Server

# Scale up vs Scale out

**Scale up**

Vorteile:

- transparent für DBMS

- Administrationsaufwand konstant

Nachteile:

- Hardware-Kosten

- Skalierung nur in größeren Stufen möglich

höhere Kosten und ungenutzte Leistung

**Scale out**

Vorteile:

- Kostengünstigere Hardware

- Skalierung in kleineren Stufen möglich

Nachteile:

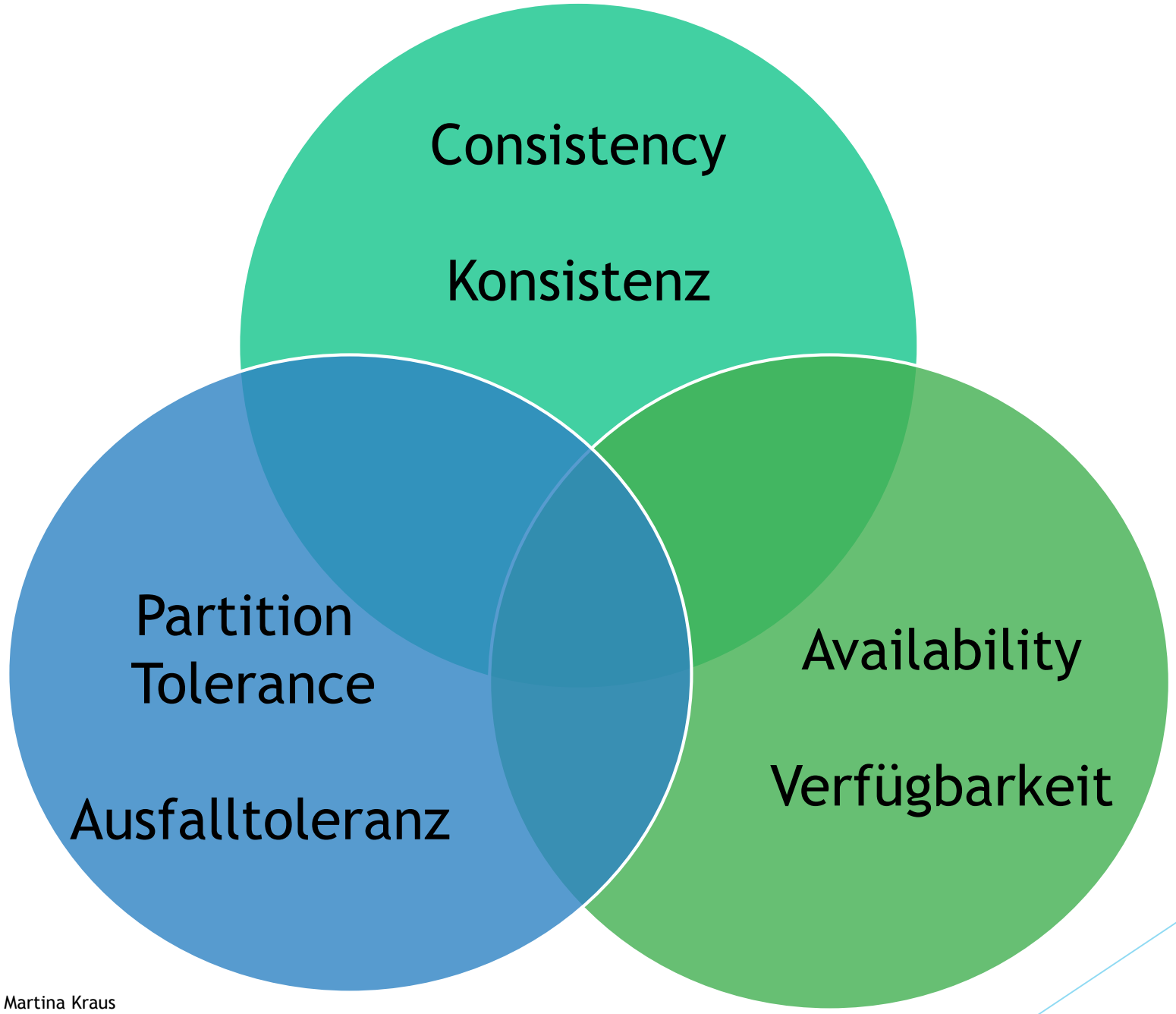
- Last- und Datenverteilungnotwendig

- Ggf. verteilte Protokolle (Replikation)

- Erhöhte Fehlerrate (mehr und einfachere Hardware)

- Erhöhter Administrationsaufwand

# Anforderungen an ein verteiltes System



# Consistency

- Vor und nach einer Transaktion ist der Datenbestand konsistent
- Alle Clients sehen **jederzeit denselben** Datenbestand
- Nicht zu verwechseln mit der Konsistenz bei ACID
  - betrifft nur den Datenbestand einer relationalen Datenbank

# Availability

- Jede Anfrage eines Clients wird zu jederzeit beantwortet
- Fällt ein Netzknoten aus, schaltet sich sofort ein anderer ein
- System ist immer „up and running“

# Partition Tolerance

System arbeitet auch bei …

- Verlust von Daten
- Netzwerkunterbrechung
- Ausfall von Knoten

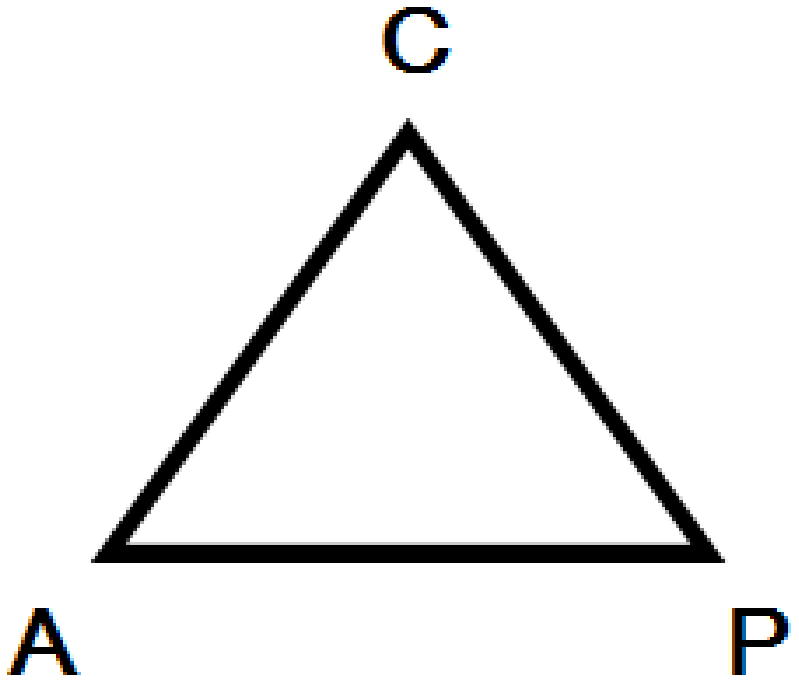
ohne Probleme weiter

# CAP – Theorem

“You can satisfy
at most 2

out of the 3 requirements”

- 2000: E. Brewer, N. Lynch



# CA -> RDBMS

- Konsistenz hat höchste Priorität bei relationalen Datenbanken
- hochverfügbare Netzwerke und Server
- System stoppt bei Ausfall eines Knoten
- horizontale Skalierung basiert allerdings auf Datenverteilung und Redundanz

# CP -> Banking Anwendungen

- Finanzanwendungen (Geldautomaten) sind verteilte Anwendungen
- Konsistenz hat auch hier höchste Priorität
- Geld soll auch bei Systemausfall ankommen
- Verfügbarkeit? Unwichtig, ein Automat kann auch mal einfach ausfallen.

# AP -> Cloud Computing

- NoSQL Datenbanken und Cloud Plattformen setzen auf horizontale Skalierung
- billige Hardware (ausfallanfällig)
- Web-Anwendungen welche eine strenge Konsistenz nicht benötigen

# BASE – Konzept

**B**asically **A**vailable, **S**oft State, **E**ventually Consistent

- In NoSQL Systemen verbreiteter Ansatz
- Gegenkonzept zu ACID
- verzichtet auf “strenge Konsistenz”
- Konsistenz der Daten werden als Zustand betrachtet
- Skalierbarkeit und Verfügbarkeit haben höchste Priorität

Konsistenz bei relationalen DB:

Bezogen auf die Integrität der Daten

Konsistenz bei NoSQL DB:

Bezogen auf den Inhalt der Daten

# Basically Available

- Daten sind immer Verfügbar
  - Egal in welchem Konsistenzzustand
- Durch erstellen von Duplikaten garantiert
- Keine Sperrung bei gleichzeitigen Lese und Schreibzugriffen

# Soft State

- Datenmengen erreichen nur periodisch ihren eigentlichen Endzustand
- Auch ohne Input werden Daten permanent geändert

# Eventually Consistent

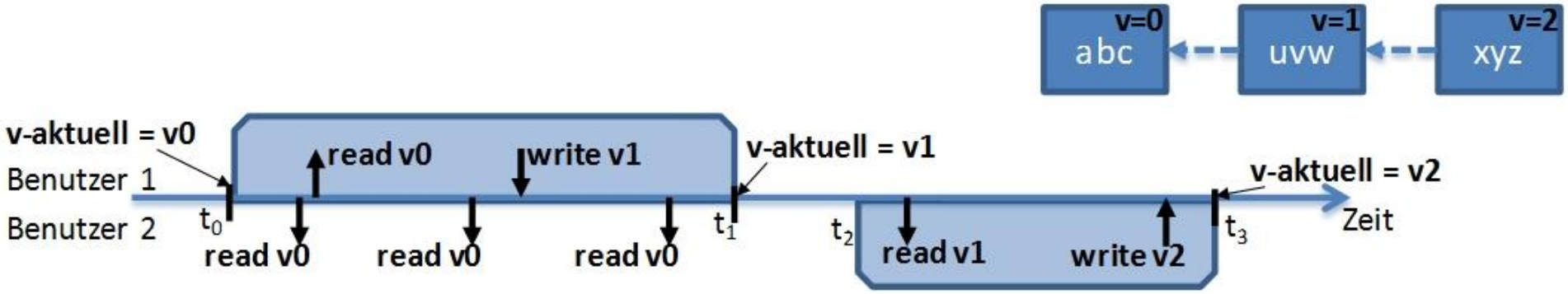
- „auf lange Sicht“, „schließlich“ konsistent
- stellt Abkehr vom strikten Konsistenzbegriff dar
- „Konsistenz als Zustandsübergang, der irgendwann erreicht wird“
- Resultat der Web 2.0 Bewegung (und die daraus resultierende Skalierung)

# Multiversion Concurrency Control

- Verfahren zur Manipulation von Daten ohne Sperrung
- kontrolliert CRUD-Befehle auf einen Datenbestand
- Für jeden manipulierenden Zugriff wird eine neue Version des Datensatz erstellt (via Zeitstempel oder TransaktionsID)
- Jede Version verweist auf ihren Vorgänger
- Keine Sperrung notwendig es steht immer eine Version zur Verfügung

# Lesender Zugriff

- Entkoppelt von manipulierenden Zugriffen
- Zur Datenanalyse könne ältere Versionen gelesen werden
- Es wird solange die ältere Version gelesen bis die manipulative Transaktion beendet ist.



# Schreibender Zugriff

- Jeder manipulative Zugriff erstellt eine neue Version

- Beim Transaktionsende wird die Vorgänger-Versionsnummer des aktuell in dieser Transaktion geänderten Datensatzes mit seiner aktuellen Versionsnummer verglichen

**v_vorgänger = v_aktuelle**

Geänderter Datensatz kann gespeichert werden, v_neu = v_aktuelle

# Schreibender Zugriff

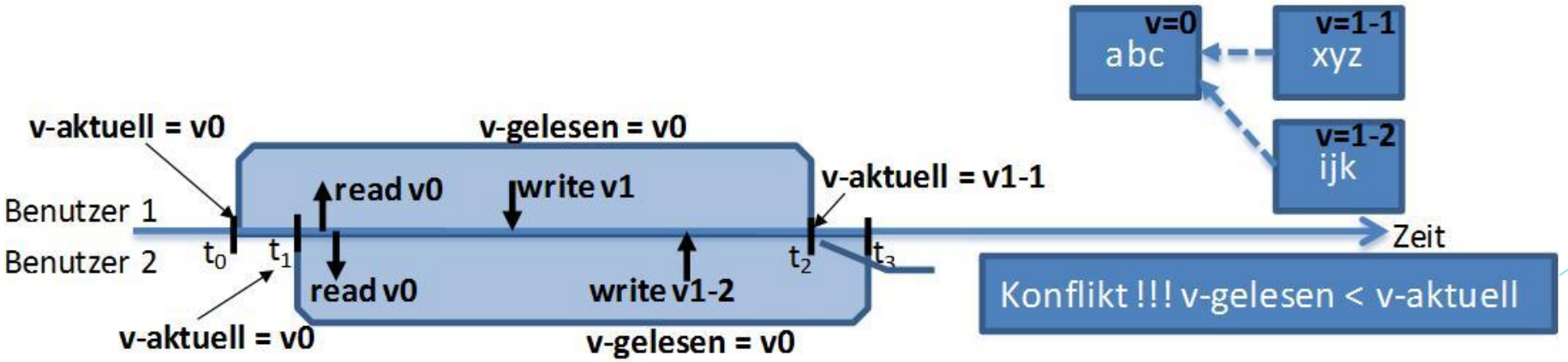
**v_vorgänger < v_aktuelle**

- es wurden unterschiedliche Attributwerte geändert:

neue Version wird aus den neuen Attributwerten beide Transaktionen zusammengesetzt

- Es wurden dieselben Attributwerte geändert:

Konflikt an den User und Abbruch der Transaktion

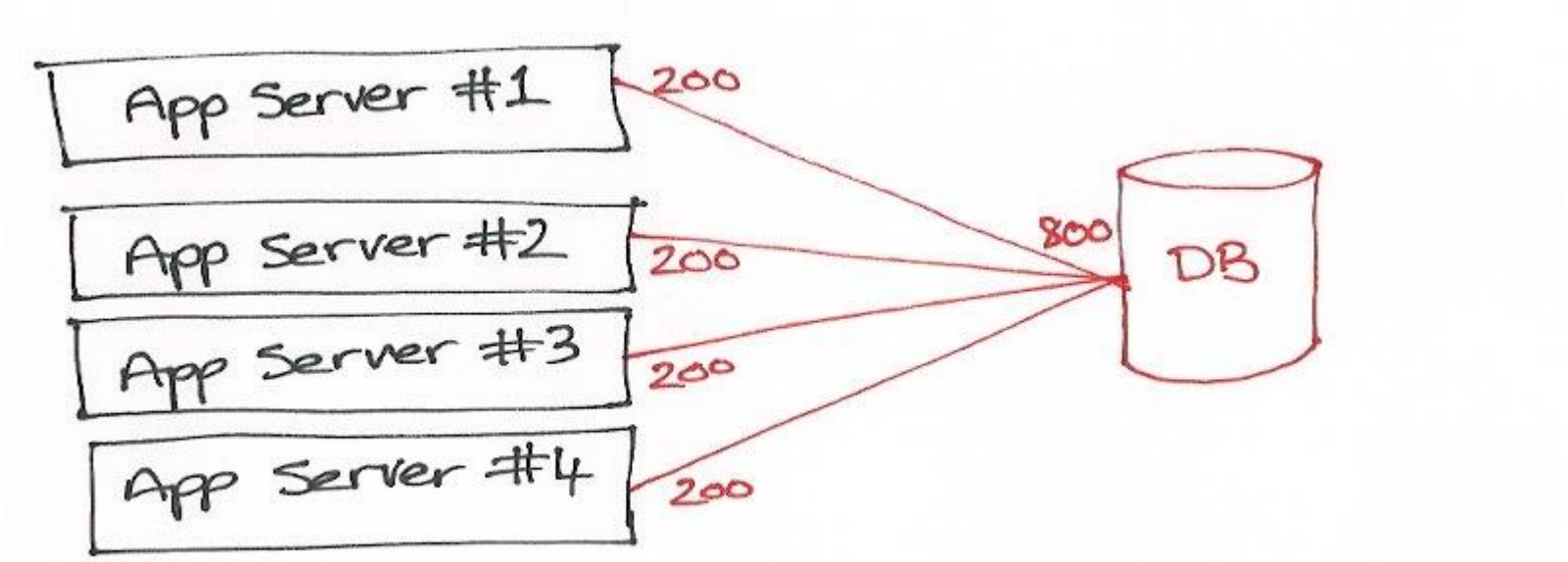


# Nachteile

- Ältere Daten-Versionen müssen aufgeräumt werden
- Einarbeitung / Perspektivwechsel bzgl verschieden starken Konsistenzeigenschaften
- Konsistenz dauert etwas
- Speichern der unterschiedlichen Versionen

# Sharding

# Bisher

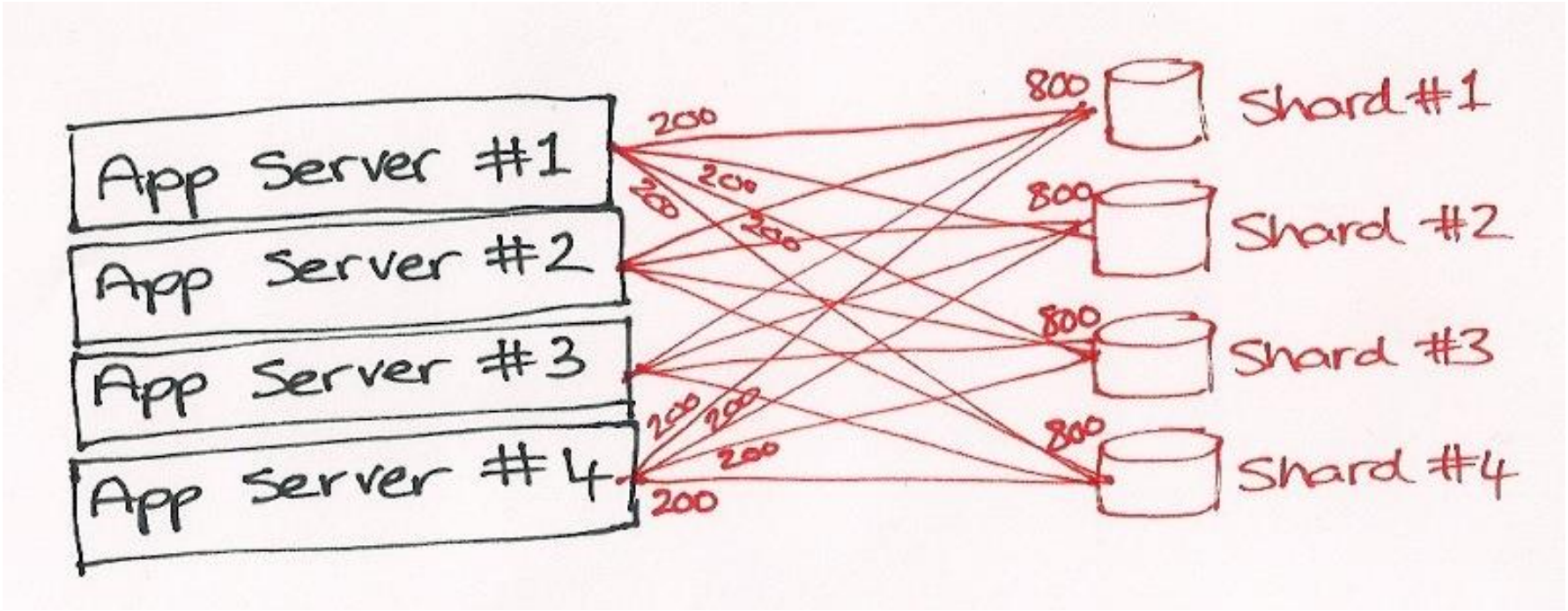



horizonale Skalierung?

# Verteilung der Daten

- Ausfallsicherheit
- Speicherauslastung (zuviele Daten)
- CPU Auslastung zu hoch (zuviele Anfragen)

# Sharding (heute)



# Sharding

- beschreibt die horizontale Aufteilung einer Datenbank

- Datensätze werden mithilfe eines **Verteilungsschlüssels** auf mehrere Rechner aufgeteilt und erhalten ein zusätzliches Attribut (**Sharding Key**)

- Datenmodell und Zugriffpfäde müssen so designed werden, dass keine Joins über eine Instanz hinweg statt finden

- NoSQL Systeme unterstützen keine Joins

# Sharding

Beispiel:

- Speicherung von Personen anhand ihres Anfangsbuchstaben
  - DB1: A-F
  - DB2: G-O
  - DB3: P-Z
- Vor der Abfrage wird die Datenbank bestimmt an die der Request weitergeleitet wird (DB-Proxy)

# Sharding

- Zu jedem Datensatz wird hierbei ein Shardingkey gespeichert
- Shardingkeys (Beispiel): A-F, G-O, P-Z

| s_key | Name |
|---|---|
| A1 | Andreas |
| A2 | Alexander |
| F1 | Felix |

| s_key | Name |
|---|---|
| P1 | Patrick |
| R1 | Robert |
| T1 | Thomas |

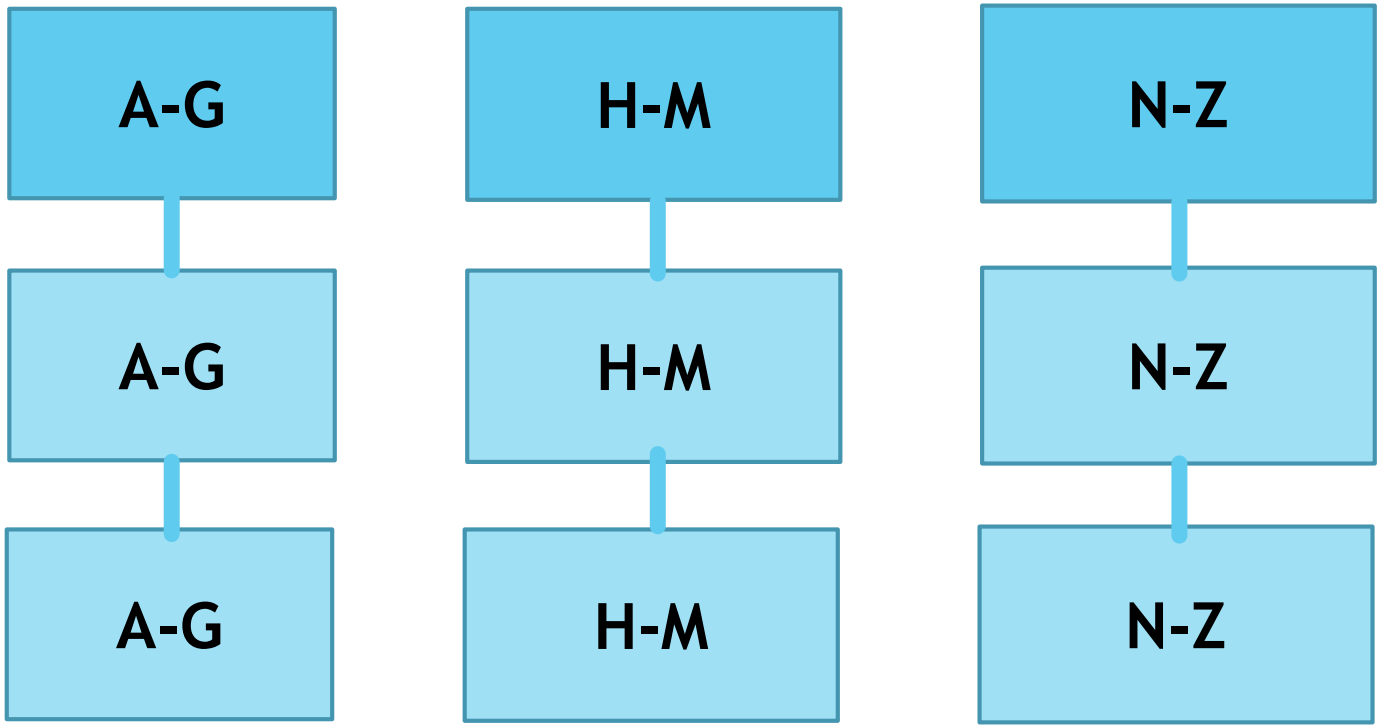
| s_key | Name |
|---|---|
| J1 | Jascha |
| M1 | Martina |
| M2 | Michael |

# Range-Based Sharding

- Verteilung der Datensätze nach einem sortierbaren Attribut
- Jeder Shard besitzt eine Range/ Intervall der Attributwerte
  - A-G, H-O,P-Z
  - 1-10, 11-20, 21-30 …
- Neuverteilung notwendig bei
  - Hinzufügen von Datensätzen
  - Hinzufügen eines Rechners

# Range-Based Sharding

- Ausfall eines Rechners

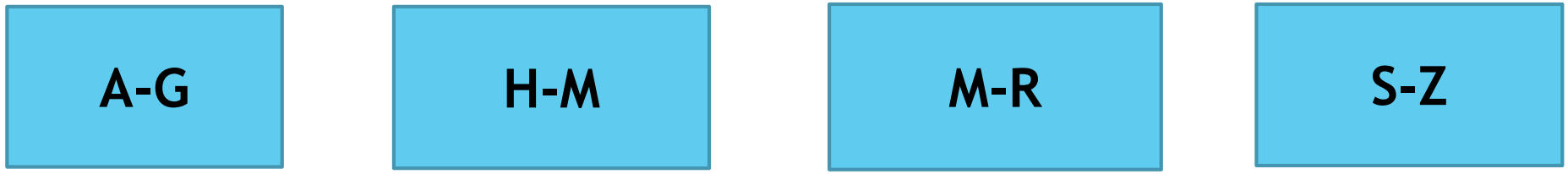


# Range-Based Sharding

- Ausfall eines Rechners abhängig von der Datenbankimplementierung

  (Meist unter Zuhilfenahme von Replikas)

- Non-unique Problematisch bei zuvielen Datensätzen in derselben Range



# Date-Based Sharding

- Spezialfall von Range-Based
- Sharding Attribut: Date Timestamp

- Neue Datensätze erhalten den Attributwert ***NOW***
- Hinzufügen eines neuen Rechners unproblematisch
  - nächster Rechner erhält aktuellere Datensätze
- Nur geringe Datensätze können non-unique sein

# Hash-Based Sharding

- Weiteres Attribut als Hash Key
  - Eigener Algorithmus oder
  - Hashwert eines Attributwerts (z.B. einer ID)

- Dann Range-Based Sharding auf Hash Keys

+ Gleiche Verteilung über den Datenbestand

- Keine sinnvolle Range-Based Abfrage

# Consistent Hashing

- Basiert auf Hash-Based Sharding
- Wertebereich wird als Ring verstanden
- Server werden anhand eines Hashwertes von ihrer z.B.  IP auf dem Adressring zugeordnet
- Datensätze werden anhand íhres Hashwertes angeordnet

# Consistent Hashing

- Fügt man einen neuen Knoten hinzu werden die ihm im Adressring näheren Datensätze rüberkopiert

- Entfernt man ein Knoten werden dessen Datensätze auf den nächsten Rechner (ihm Uhrzeigersinn) des Adressrings gespeichert

- Jeder Rechner kann je nach Leistung virtuelle Server erzeugen

# Sharding - Definitonen

- **Shard**

logischer Container partitionierter Daten, der in einer (und nur in einer einzigen) physischen Datenbank gehostet wird

- **Physische Datenbank**

Datenbankinstanzen, die Teile einer Sharddatenbank hosten

(Azure SQL-Datenbank, MongoDB, Elasticsearch, MySQL)

- **Mandat**

Verwaltungsentität welcher Teile einer Shardingdatenbank besitzen

- **Shardlet**

eine Gruppe von Datensätzen, die den gleichen Shardingschlüssel gemeinsam verwendet

# Sharding - Definitonen

- **Hot Shard**

Shard mit aktuellsten oder anspruchsvollsten Daten

- **Verteilungsschlüssel**

ein deterministischer Algorithmus für die gleichmäßige Verteilung der Daten.

Meist eine Gleichung, Modulo oder komplexer kryptographischer Prozess

# Sharding - Nachteile

- Komplexitätssteigerung
- Single point of failure
- Backups der Daten sind komplizierter
- Schema Modifikationen sind komplex

# Replikate

# Replikationen

- alternative zum Sharding
- angewandt in verteilten Systemen
- Daten werden nicht nur auf einem Rechner, sondern auf mehreren mehrfach gespeichert
- Gängige Replikationsrate: 3-5 Replikate

# Replikationen

- Inkonsistenz vermeiden durch Sperrung bei Kopierungsprozess
- ständige synchronisierung notwendig
- Ausfallsicher