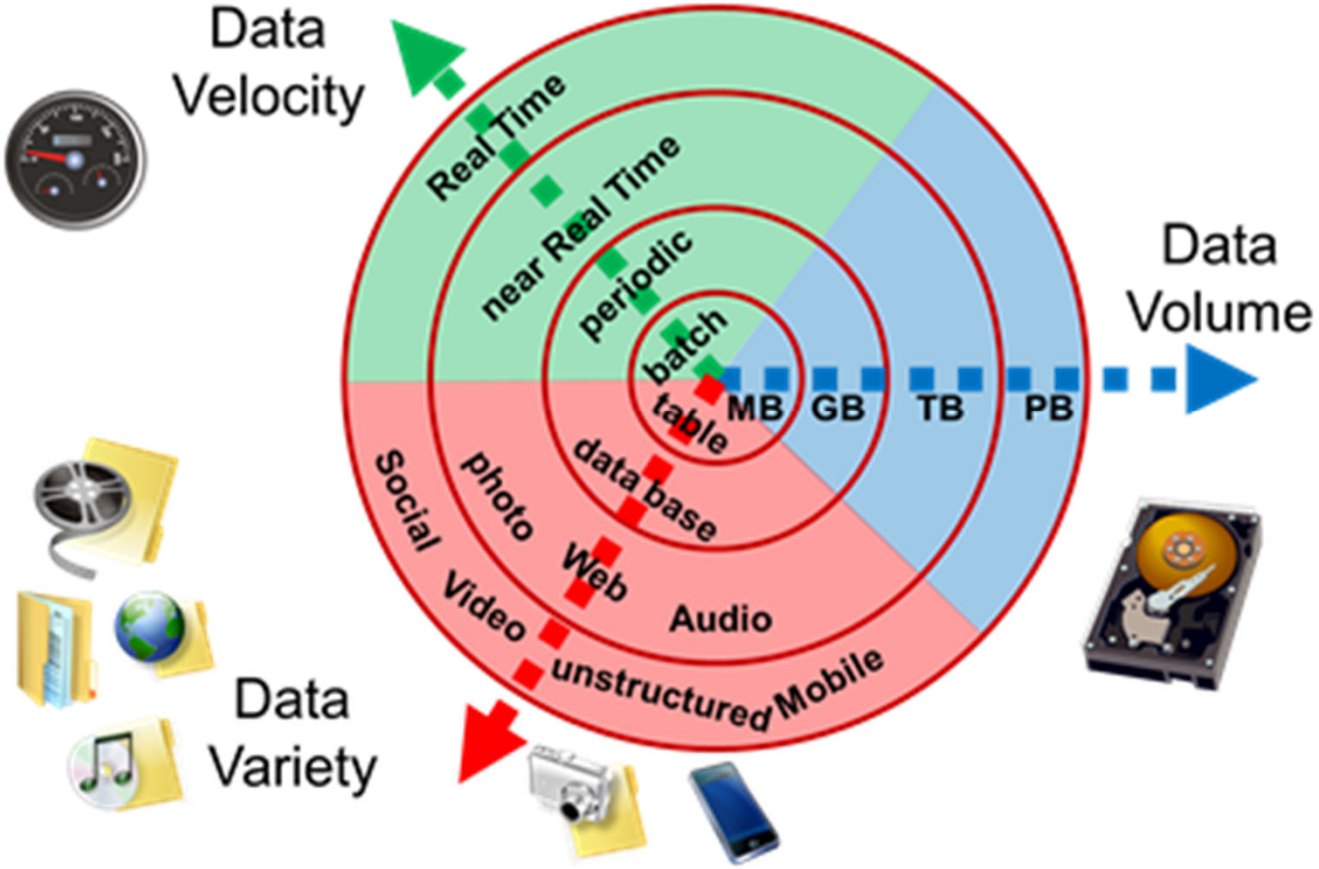
10 Datenbanken

10 Lernziele

- Sie wissen, was der Begriff „Big Data“ bedeutet und welche Herausforderungen sich damit ergeben.
- Sie wissen, wie verteilte Datenbanken funktionieren und welche Vorteile diese haben.
- Sie kennen die Probleme, die sich beim Betrieb verteilter Datenbanken ergeben und wie sie sich lösen lassen.

Aspekte von Big Data

- **Volume**: Große Menge der zu speichernden Daten
- **Velocity**: Hohe Geschwindigkeit der Erhebung neuer Daten
- **Variety**: Starke Variation in der Struktur der Daten



10 Globales Datenaufkommen & Future Skills

→**Im Jahre 2027:**
wird das Datenvolumen voraussichtlich auf über 284 Zettabytes ($285*10^{21}$ Bytes) ansteigen.

https://de.statista.com/statistik/daten/studie/267974/umfrage/prognose-zum-weltweit-generierten-datenvolumen/#professional
Abruf 01.10.23

McKinsey - Studie: zusätzlich benötigte Mitarbeiter bis 2023 im D.
Platz 1: Komplexe Datenanalyse: 455 Tausend
Platz 2: Nutzerzentriertes Designen (UX): 79 Tausend
…
https://www.stifterverband.org/future-skills/bedarf-bis-2023

Future Job Skills: Datenkompetenz bis 2030 gefragteste Fähigkeit
https://www.digital-chiefs.de/datenkompetenz-job-skill-bis-2030-faehigkeit/ **Meldung vom Juli 2022**

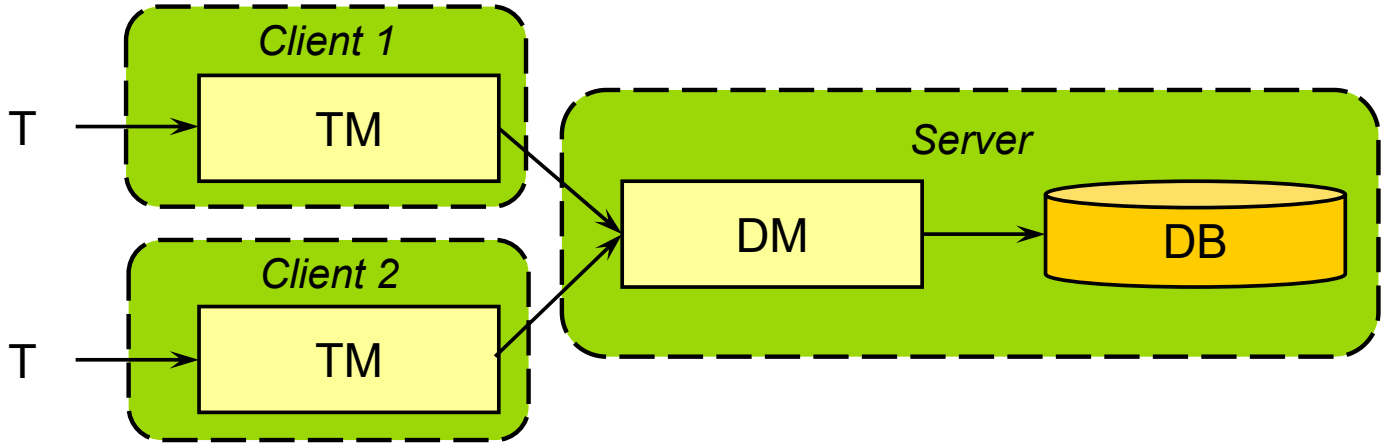
10 Big Data – Herausforderungen

- Große Datenmengen stellen Unternehmen vor neue Herausforderungen im Umgang mit Big Data
- Mobile Endgeräte
  - Zahl mobiler Endgeräte wächst zweistellig
  - Viele Produkte im Bereich der personalisierten Medizin (Smartwatches, Wearables)
- Internet of Things (IoT)
  - Generierung massiver Datenmengen
- Alle diese Anwendungen benötigen hochperformante, verteilte und robuste Datenbanksysteme

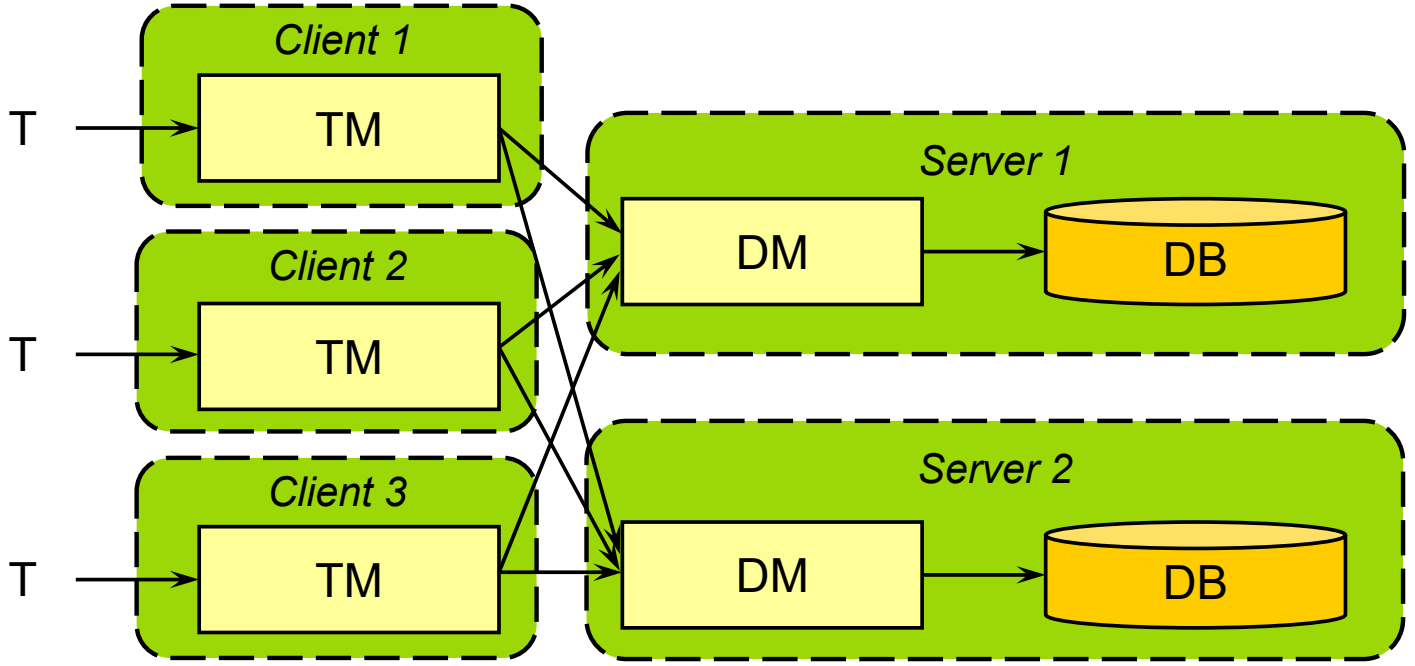
Big Data – Skalierung der Hardware

- **Scaling Up (Vertikale Skalierung)**
  - Alle Server erfahren das gleiche Upgrade (schnellere CPU, mehr RAM,...)
  - Mehr Ressourcen pro Server
- **Scaling Out (Horizontale Skalierung)**
  - Verteilung der Last auf mehrere Server (Clustering)
  - 1000 x 1TB Festplatten sind kostengünstiger als eine PB Festplatte
  - Mehr Server im System

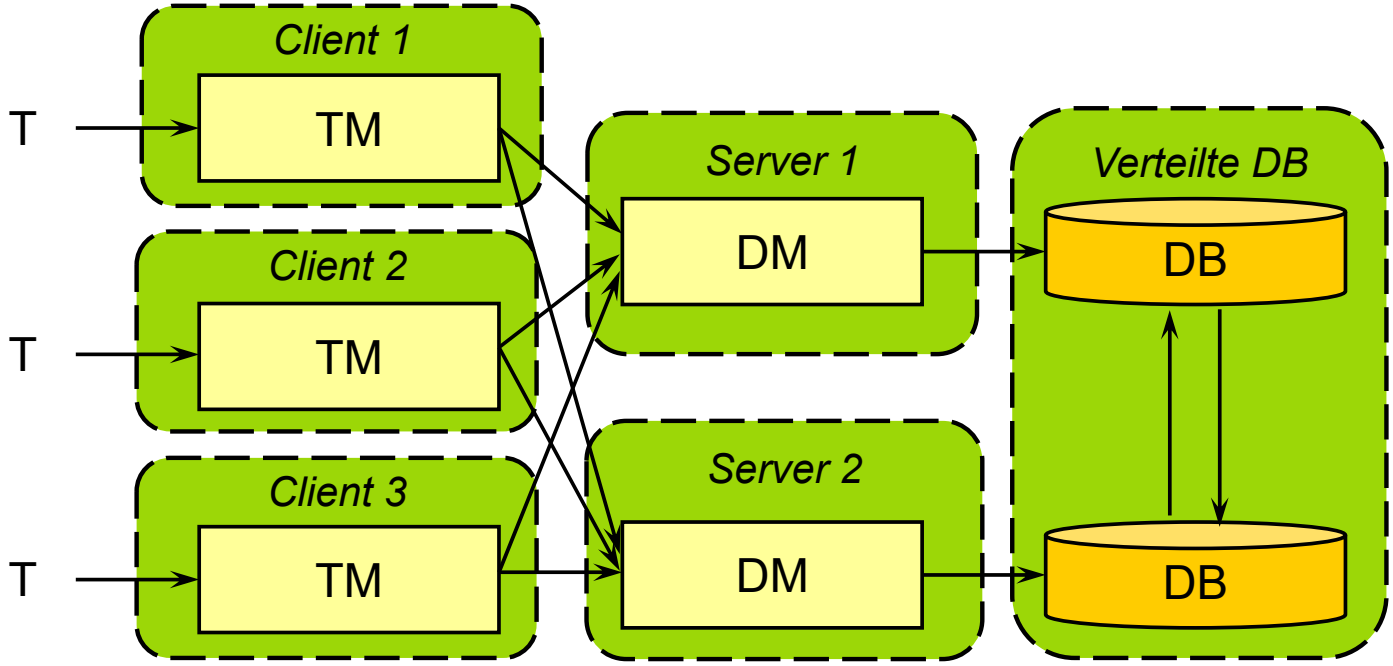
Single-Site Data (Client-Server-Architektur)



Multiple-Site Data (Verteilte Architektur)

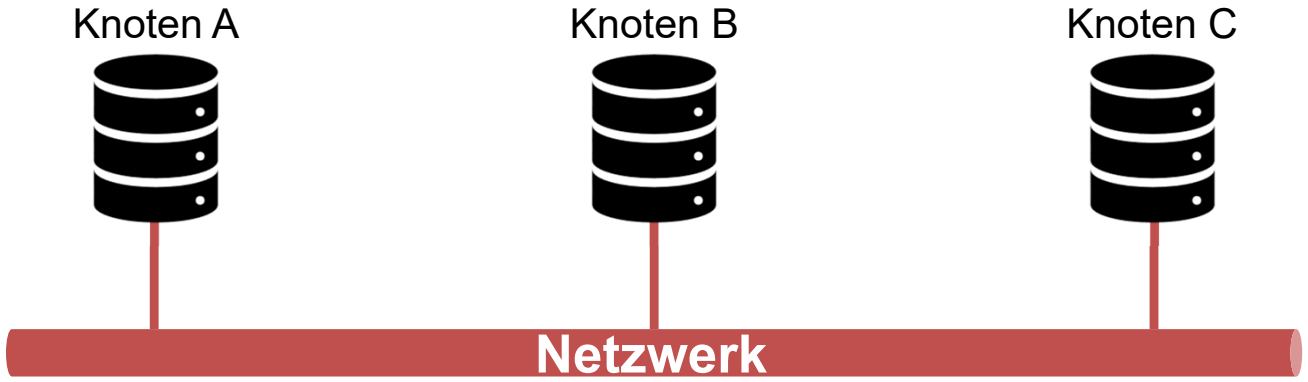


Multiple-Site Data (Verteilte Architektur)



10 Eigenschaften verteilter Datenbanken

- Eine verteilte Datenbank besteht aus mehreren Knoten.
- Diese Knoten…
  - Sind über ein Netzwerk verbunden
  - Agieren als eigenständige Datenbanken
  - Arbeiten zusammen, um Anfragen an die **verteilte Datenbasis** zu erfüllen
- Der Benutzer merkt nicht, dass die Datenbank verteilt ist. (Transparenz)
  - Es ist kein Wissen des Anwenders über die Verteilung erforderlich



10 Verteiltes Datenbanksystem

Ein **verteiltes Datenbanksystem** besteht aus mehreren über ein Datennetz verbundenen Knoten, für die gilt:

- Jeder Knoten stellt eine eigenständige Datenbank dar, aber
- die Knoten arbeiten bei Bedarf zusammen, so dass jeder Benutzer an jedem Knoten auf jedes Datum, das im Netzwerk existiert, so Zugriff hat, als ob das Datum am Ort des Benutzers gespeichert wäre.

10 Verteilte Datenbasis

Eine **verteilte Datenbasis** ist eine Sammlung von Daten,

- die physisch über mehrere, autonom agierende, über ein Netzwerk verbundene Rechner verteilt sind;
- die durch die überwiegende Zahl von Anwendungen rechnerlokal bearbeitet werden;
- deren Zusammenschluss eine einheitliche, logische Gesamtsicht auf die Daten darstellt;
- die so verwaltet werden, dass jedes beliebige Anwendungsprogramm den Eindruck hat, die Daten würden auf dem Rechner gehalten, auf dem das jeweilige Anwendungsprogramm abläuft.
- auf die in ihrer Gesamtheit von jedem Rechner des Netzwerks aus zugegriffen werden kann.

Weitere Infos: Mehrrechner-Datenbanksysteme - Grundlagen der verteilten und parallelen Datenbankverarbeitung, Erhard Rahm, 1994
https://www.researchgate.net/publication/220688530

10 Motivation für verteilte Datenbanken

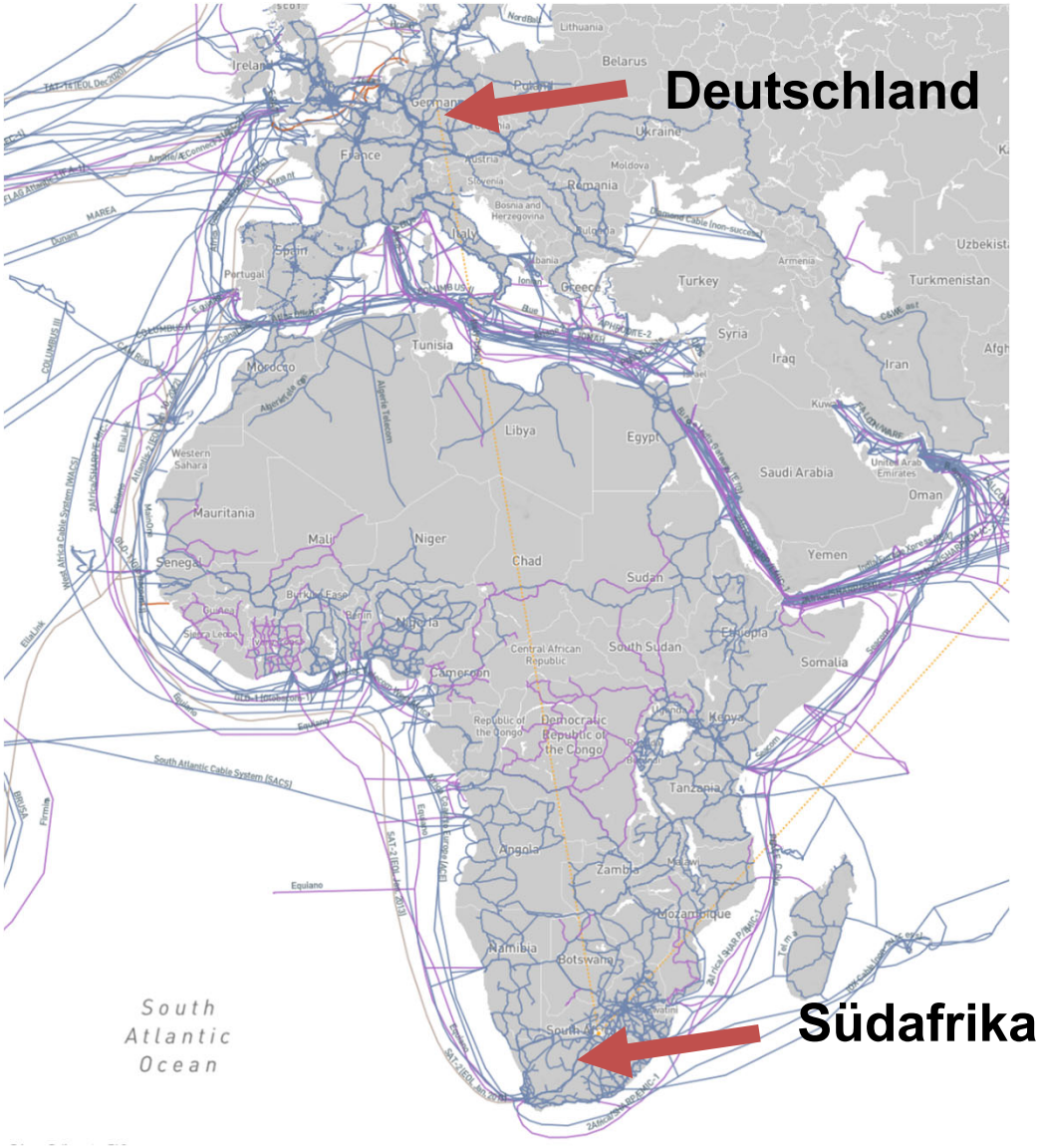
- Duplizieren von Daten (ggf. ohne verteilte Datenbasis) ➔ Replication
  - Sehr viele Zugriffe / Viele komplexe Zugriffe
  - Redundanz bei Serverausfällen / Netzwerkproblemen
- Aufteilen von Daten ➔ Sharding
  - Daten werden geografisch aufgeteilt ➔ Datenlokalität
  - Langsames Netzwerk
  - Kapazität eines Rechners nicht ausreichend
- Ggf. eine Kombination aus beidem

10 Geografische Verteilung (Extrembeispiel)

Daten …

- brauchen Zeit (Latenz, Bandbreite)
- gehen verloren (Paketverlust)
- gehen durch viele Knoten („Hops“)
- nutzen „langsame“ Protokolle (TCP)

➔ Datenlokalität ist sinnvoll



Karte: https://www.infrapedia.com/

10 Pfadverfolgung nach Südafrika

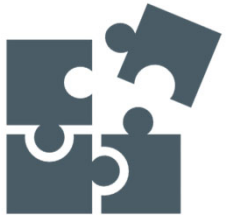
```
→ tracert www.gov.za

Routenverfolgung zu www.gov.za [164.151.129.20]
über maximal 30 Hops:

  1    <1 ms    <1 ms    <1 ms  WILHELM [192.168.169.1]
  2    23 ms    19 ms    12 ms  i5E86C64C.versanet.de [94.134.198.76]
  3    21 ms     7 ms     7 ms  i59F44416.versanet.de [89.244.68.22]
  4     7 ms     7 ms     7 ms  62.214.32.40
  5     8 ms     8 ms     8 ms  ae-22.r00.frnkge13.de.bb.gin.ntt.net [80.81.192.46]
  6    10 ms     7 ms     8 ms  ae-2.r20.frnkge13.de.bb.gin.ntt.net [129.250.6.13]
  7    18 ms    18 ms    18 ms  ae-14.r21.londen12.uk.bb.gin.ntt.net [129.250.3.12]
  8    20 ms    20 ms    33 ms  ae-1.a02.londen12.uk.bb.gin.ntt.net [129.250.3.215]
  9    22 ms    25 ms    21 ms  dimensiondata-0.r02.londen03.uk.bb.gin.ntt.net [83.231.235.222]
 10   186 ms   183 ms   187 ms  core1b-pkl-ten-ge-0-6-0-1.ip.ddii.network [168.209.100.213]
 11   183 ms   183 ms   183 ms  za-gp-pkl-hpe-1-fo-0-1-0-1-0sub102.ip.ddii.network [168.209.1.200]
 12   190 ms   190 ms   190 ms  za-gp-pkl-p-1-be-111.ip.ddii.network [168.209.129.136]
 13   183 ms   183 ms   183 ms  za-gp-bry-p-1-hu-0-0-0-11.ip.ddii.network [168.209.90.7]
 14   183 ms   183 ms   183 ms  za-gp-bry-pe-2-be-102.ip.ddii.network [168.209.131.135]
 15   188 ms   188 ms   188 ms  197.97.72.54
 16     *        *        *     Zeitüberschreitung der Anforderung.
 17     *        *     183 ms  dmz1.gcis.gov.za [164.151.129.20]
```

Websiteaufruf von Deutschland: 1MB Daten – 16 Sekunden

10 Replication und Sharding

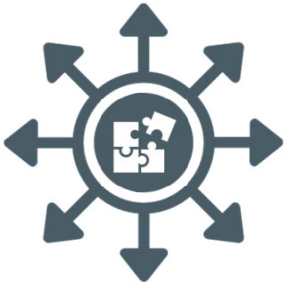
Replication / Replikation

Duplizierung von Daten über mehrere Knoten der verteilten Datenbank.
Funktion: Backup, horizontale Skalierung, Datenlokalität



Fragmentierung

Aufteilung von großen Tabellen anhand von Attributen oder Spalten.
Funktion: Steigerung der Zugriffsgeschwindigkeit, „Archivierung“



Sharding / Aufteilung / Allokation

Aufteilung der ggf. vorher fragmentierten Datenbasis über mehrere Knoten.
Funktion: Datenlokalität bei weniger Redundanz, Lastverringerung

10 Anforderungen an eine verteilte Datenbank

Übersicht:

- Generell: Logische Transparenz aus Benutzersicht
  („Fundamentalprinzip“)
- Spezielle Regeln für verteilte Datenbanken:
  - Lokale Autonomie
  - Unabhängigkeit lokaler Komponenten
  - Dauerbetrieb / hohe Verfügbarkeit
  - Lokale Transparenz
  - Fragmentierungstransparenz
  - Replizierungstransparenz
  - Verteilte Bearbeitung von Datenbankoperationen
  - Verteiltes Transaktionsmanagement
  - Hardware-, Betriebssystem-, Netzwerk- und DBMS-Transparenz

Quelle: Hermann Kudlich: Verteilte Datenbanken, 1992

10 Anforderungen an eine verteilte Datenbank

- Lokale Autonomie:
  - Jeder Standort kontrolliert und verwaltet autonom die am Standort gespeicherten Daten.
  - Kein Standort A ist in dem Maße von einem anderen Standort B abhängig, dass A gar nicht funktionieren würde, wenn ein Zugriff auf B nicht möglich ist.
- Unabhängigkeit lokaler Komponenten
  - Alle Standorte sind gleichberechtigt.
  - Keine zentrale Instanz.
- Dauerbetrieb / hohe Verfügbarkeit
  - Für Updateprozesse oder Hinzufügen neuer Standorte sollte der Betrieb an den vorhandenen Standorten nicht eingeschränkt werden.
  - Unanfälligkeit gegenüber Fehlern wie z.B. Ausfall eines Knotens.

10 Anforderungen an eine verteilte Datenbank

- Lokale Transparenz
  - Ein Benutzer muss nicht wissen, an welchem Standort die Daten gespeichert sind.
  - Die Daten müssen so erscheinen, als seien sie alle am lokalen Ort gespeichert.
- Fragmentierungstransparenz
  - Für den Benutzer einer verteilten DB soll die Fragmentierung unsichtbar (transparent) sein.
    Beispiel: Eine Tabelle „Kunde“ ist fragmentiert.
    Die Abfrage kann trotzdem so formuliert sein:

```
select * from Kunde
where name = 'Müller';
```

- Replizierungstransparenz
  - Für den Benutzer soll die Replikation unsichtbar sein.
  - ➔ Die Daten auf allen Replikationen müssen gleich sein.
  - ➔ Die Replikation wird „automatisch“ gewählt.

10 Anforderungen an eine verteilte Datenbank

- Verteilte Bearbeitung von Datenbankoperationen
  - Benötigte Daten können an unterschiedlichen Orten liegen.
  - Es sind Techniken für die Optimierung der Datenabfragen notwendig, um die Netzlast zu verringern (sog. Query Optimizer).
- Hardwaretransparenz
  - Das verteilte Datenbankmanagementsystem (VDBMS) soll unabhängig von der zugrundeliegenden Hardware arbeiten.
  - Das VDBMS soll sich auf jeder Hardware gleich verhalten.
- Betriebssystemtransparenz
  - Das VDBMS soll unabhängig vom verwendeten Betriebssystem sein und sich auf verschiedenen Systemen gleich verhalten.
- Netzwerktransparenz
  - Das VDBMS soll unterschiedliche Netzwerktypen unterstützen.
- DBMS-Transparenz
  - Unterschiedliche SQL-Datenbankimplementierungen sollen in einem Netzwerk aufgrund des SQL-Standards zusammenarbeiten können.

10 Vorteile verteilter Datenbanken

- Datenlokalität
  - Die Daten liegen an dem Ort, an dem sie am meisten benötigt werden
- Zugriffsgeschwindigkeit
  - Schnellerer Zugriff auf die Daten, da meist nur eine Teilmenge der Gesamtdaten benötigt werden
- Bearbeitungseffizienz
  - Effizientere Datenbearbeitung, da dezentral
- Ausbaukosten
  - Hinzufügen neuer Knoten ist günstiger als Ausbau des eines Servers.
- Ausfallsicherheit
  - Durch Verteilung der Daten höhere Ausfallsicherheit
- Erweiterbarkeit
  - Neue Standorte können in das Netzwerk integriert werden, ohne dass andere Standorte davon beeinträchtigt werden

10 Historie der verteilten Datenbanken

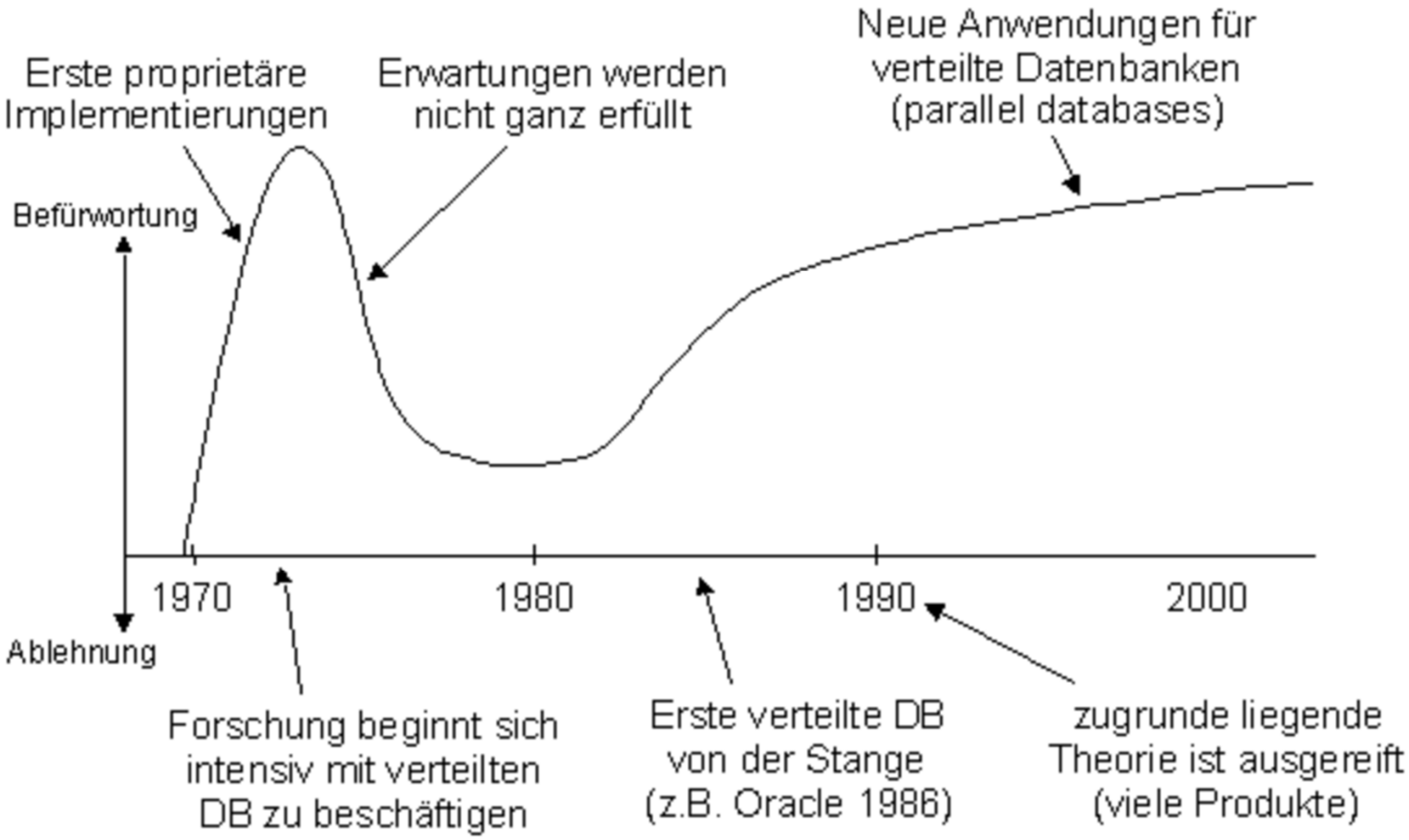


Bild: https://igw.tuwien.ac.at/fit/fit08/team3/distributed.html

10 Nachteile verteilter Datenbanken

- Management und Kontrolle
  - Weitaus komplexer als bei einer zentralen Datenbasis
- Sicherheit
  - Mehr „Angriffsfläche“ führt zu erhöhten Sicherheitsmaßnahmen
- Unterstützung heterogener Systeme
  - Mangelnde Einhaltung von Standards und/oder unterschiedliche Versionen erschweren den Zusammenschluss vorhandener DBMS zu einem verteilten DBMS
- Alle Probleme eines verteilten Systems

Vorgehen in **drei Phasen**

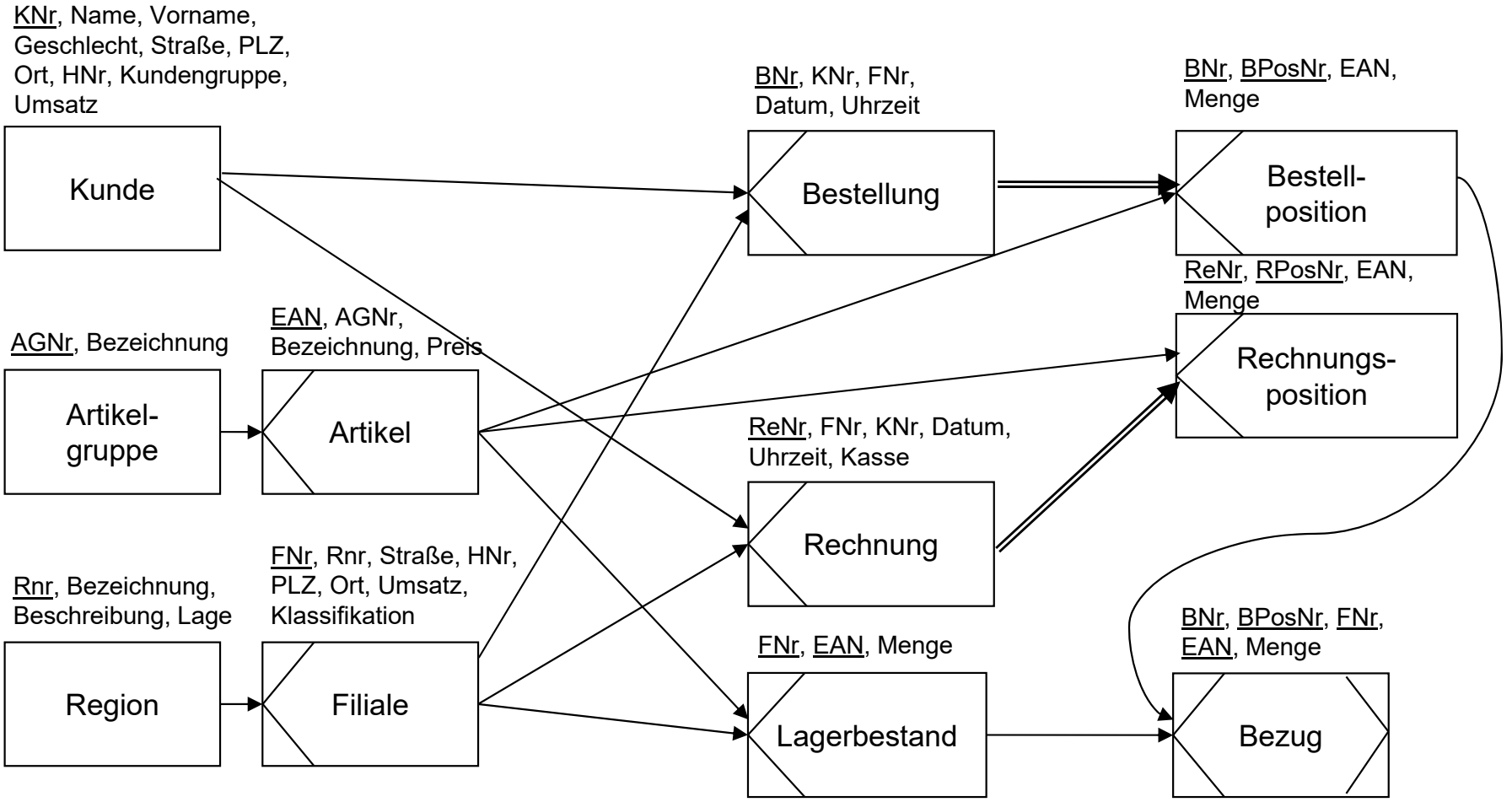
1. Entwurf des globalen Modells
2. Entwurf der Fragmentierung
3. Replizieren und Ortzuweisung

10 Beispiel

*Ein Unternehmen verfügt über mehrere **Filialen in verschiedenen Regionen**, deren aktueller **Artikelbestand** registriert wird. **Artikel sind in Gruppen zusammengefasst.** Verwenden Kunden beim Einkauf eine **Kundenkarte**, wird erfasst, wann sie **in welcher Filiale welchen Artikel** in welcher Menge bezogen haben. Darüber hinaus können auch **Artikel** bestellt werden, die nicht mehr in der Filiale vorhanden, jedoch noch **in anderen Filialen** in ausreichender Menge vorrätig sind. Nach der Bestellung erfolgt eine **Rechnung**.*

- *Regionale Filialen mit Artikelbestand*
- *Artikel sind gruppiert*
- *Kundenkarte erfasst Einkäufe*
- *Artikel können aus anderen Filialen bestellt werden*
- *Zu jeder Bestellung gibt es eine Rechnung*

10 Beispiel: Entwurf eines globalen Modells



10 Beispiel

Kunde									
KNr	Name	Vorname	Geschlecht	Strasse	PLZ	Ort	HNr	Kundengruppe	Umsatz

Region			
RNr	Bezeichnung	Beschreibung	Lage

Artikelgruppe	
AGNr	Bezeichnung

Filiale							
FNr	RNr	Strasse	HNr	PLZ	Ort	Umsatz	Klassifikation

Artikel			
EAN	AGNr	Bezeichnung	Preis

Bestand		
FNr	EAN	Menge

Rechnung					
ReNr	FNr	KNr	Datum	Uhrzeit	Kasse

10 Beispiel

Rechnungsposition			
ReNr	RPosNr	EAN	Menge

Bestellung				
BNr	KNr	FNr	Datum	Uhrzeit

Bezug				
BNr	BPosNr	FNr	EAN	Menge

Bestellposition			
BNr	BPosNr	EAN	Menge

10 Beispiel: Horizontale Fragmentierung

Mitarbeiter				
MNr	Name	Vorname	Ausbildung	Gehalt
⋮				

Mitarbeiter_Abteilung_1				
MNr	Name	Vorname	Ausbildung	Gehalt

…

Mitarbeiter_Abteilung_X				
MNr	Name	Vorname	Ausbildung	Gehalt

10 Beispiel: Vertikale Fragmentierung

Kunde									
KNr	Name	Vorname	PLZ	Ort	Straße	HNr	Geb.Dat.	Geschl.	Familienstand

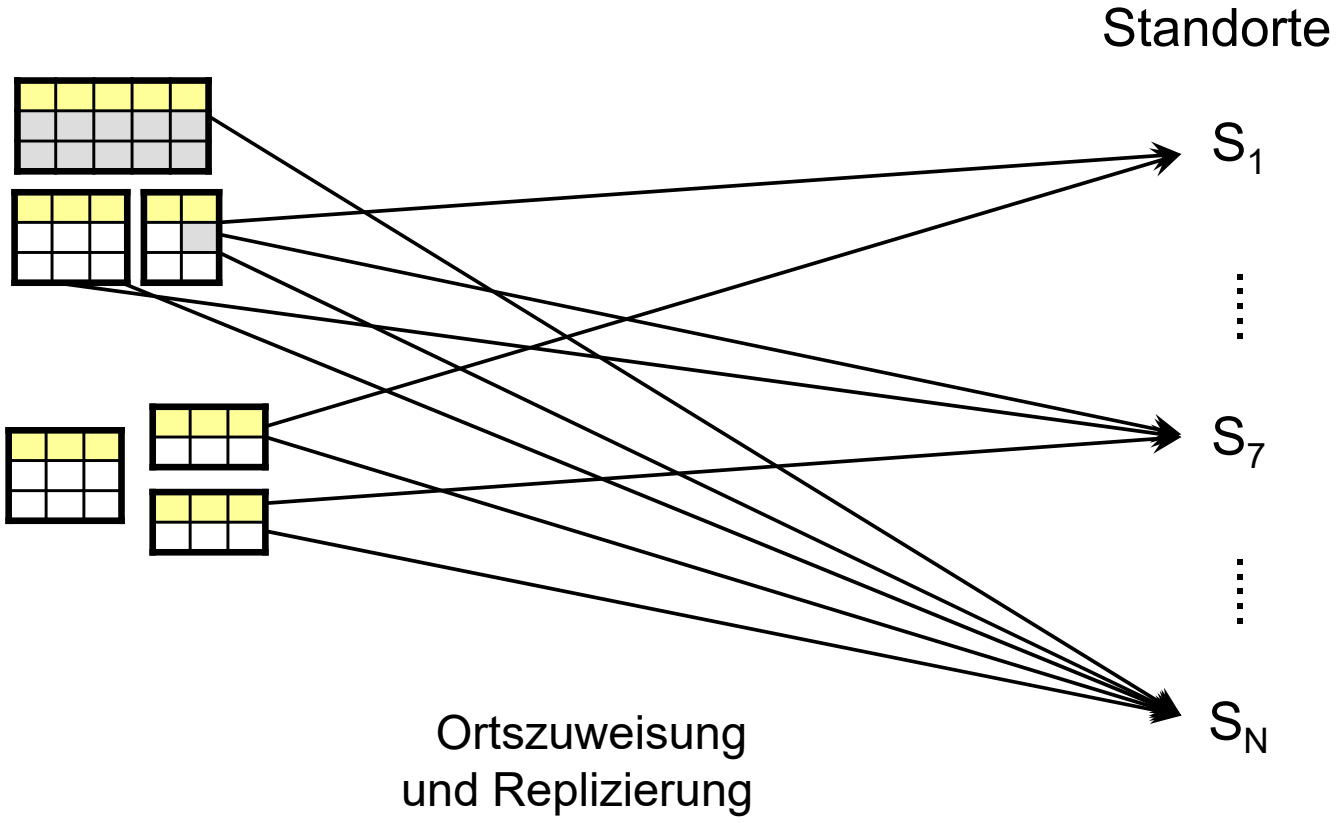
Kunde_Versand						
KNr	Name	Vorname	PLZ	Ort	Straße	HNr

Kunde_Marketing			
KNr	Geb.Dat.	Geschl.	Familienstand

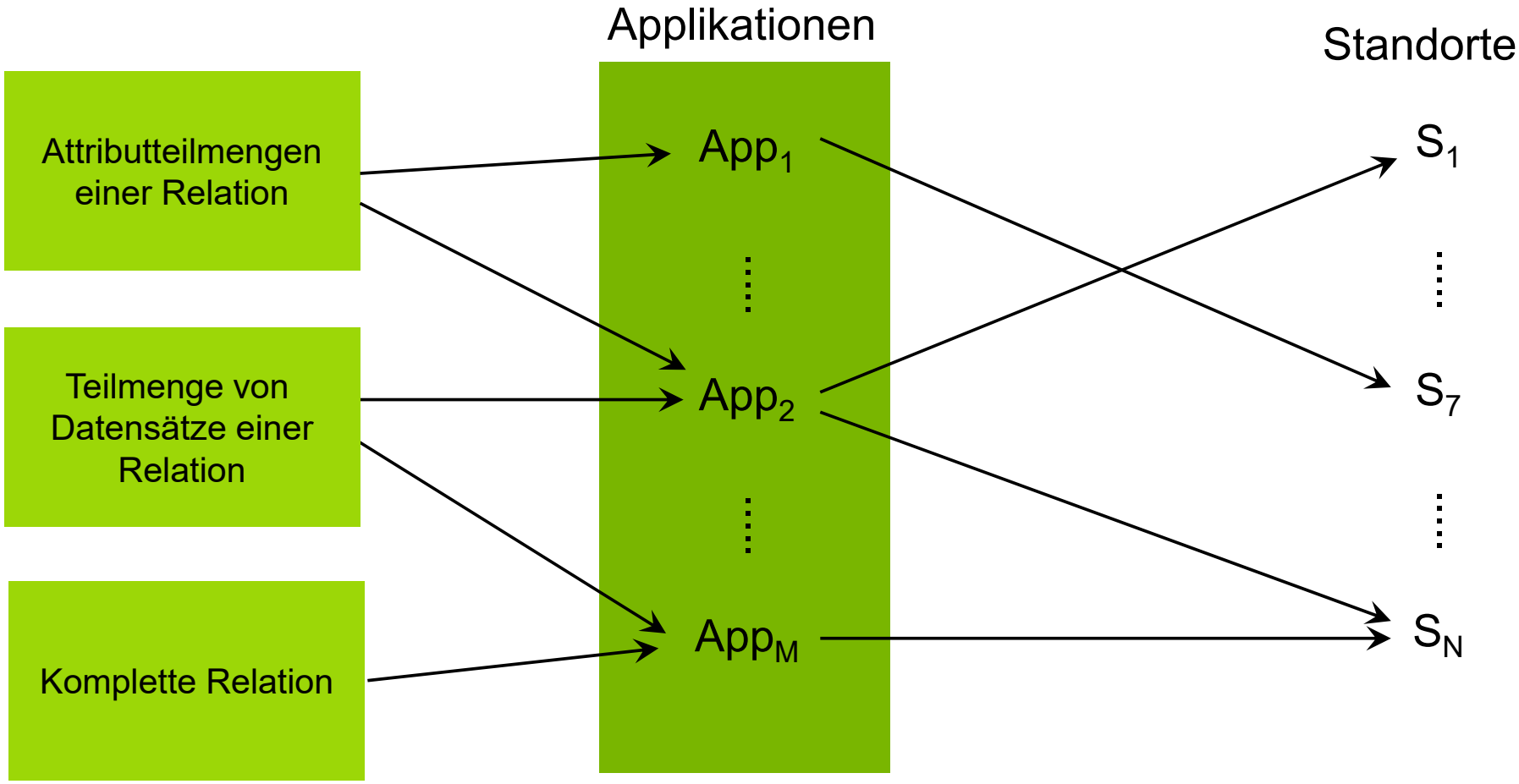
Fragmentierung, Ortzuweisung und Replizierung

Vertikale Fragmentierung
Attributteilmengen einer Relation

Horizontale Fragmentierung
Datensätze einer Relation



10 Beispiel: Replizierung und Ortzuweisung



10 Betrieb verteilter Datenbanken

Betriebsaufgaben:

1. Finden der relevanten Daten:
   Katalogmanagement
2. Sicherstellung der Konsistenz:
   Verteiltes Transaktionsmanagement

10 Katalogmanagement (Katalog = Schema)

Aufgabe: Finden des Standortes von Relationen

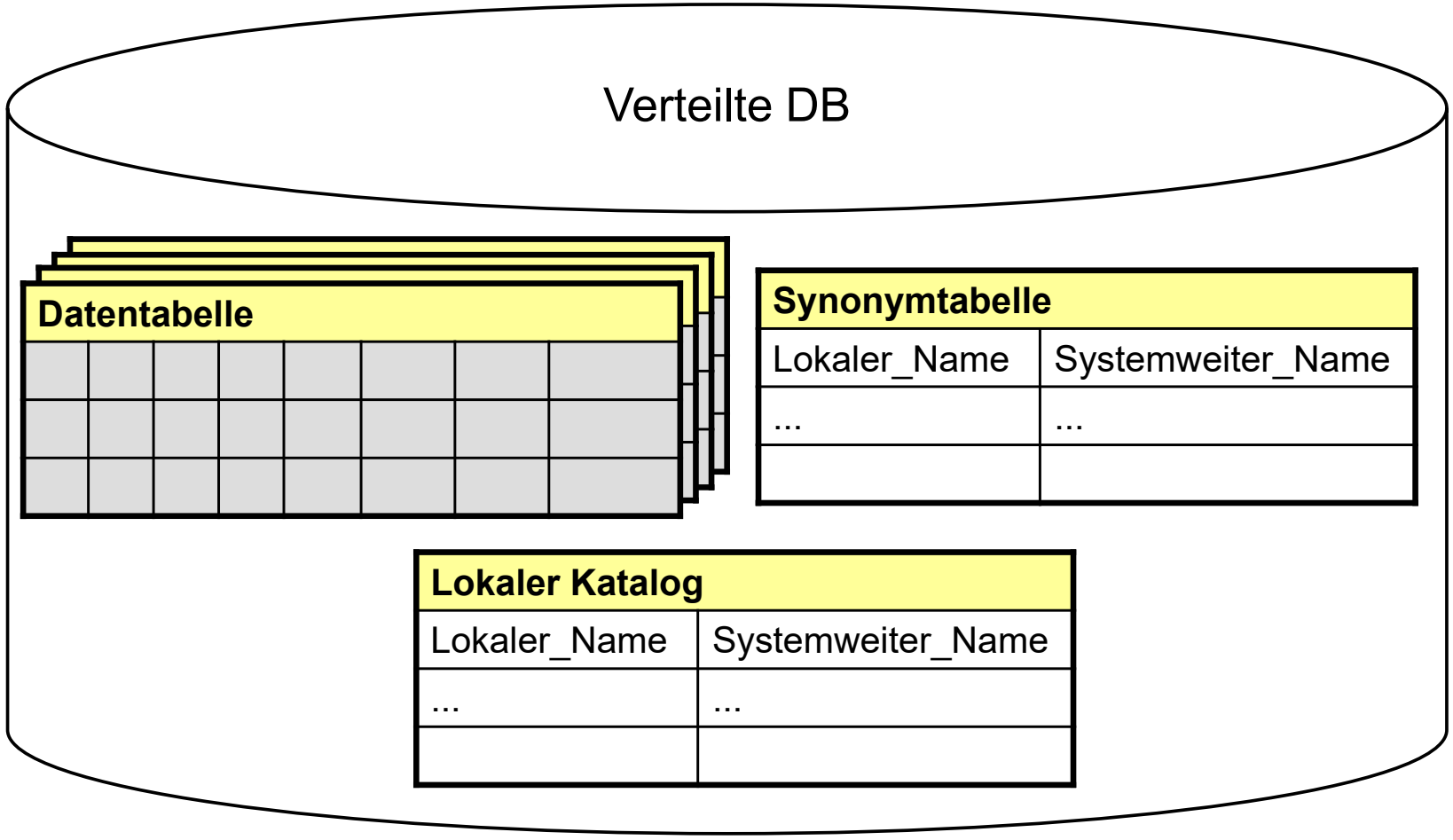
Lösungsmöglichkeiten:

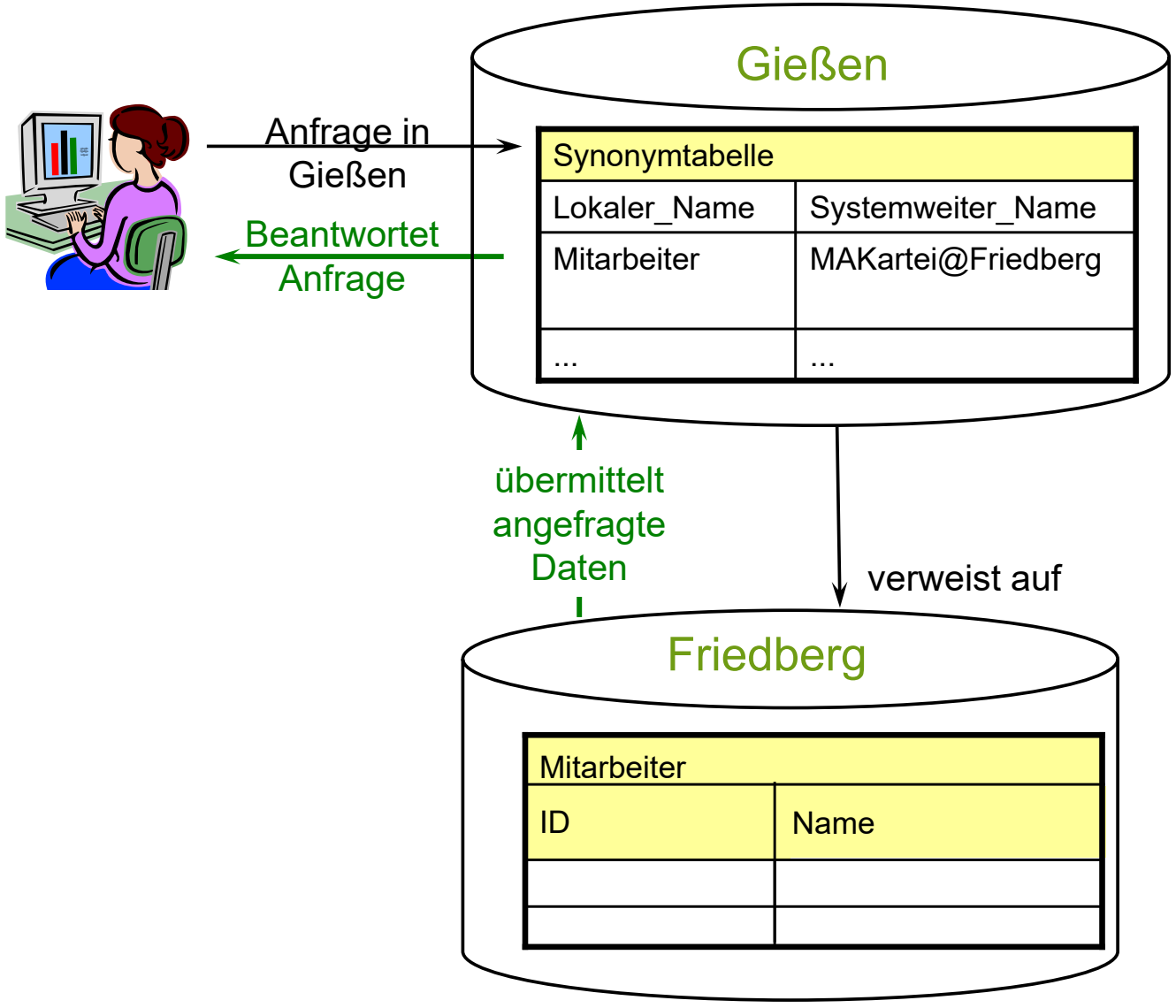
1. Zentralspeicherung des globalen Katalogs
2. Komplettkatalog an jedem Knoten
3. Kompromiss: Cluster aus mehreren Knoten haben globalen Katalog
4. Verteilte Katalog- und Katalogverweisverwaltung

→ Systemweiter Name:

→ Synonym erstellen:

```
CREATE SYNONYM Mitarbeiter
FOR MAKartei@Friedberg
```


Verteilte DB
Datentabelle
Synonymtabelle
Lokaler_Name
Systemweiter_Name
...
...
Lokaler Katalog
Lokaler_Name
Systemweiter_Name
...
...


Gießen
Anfrage in Gießen
Beantwortet Anfrage
Synonymtabelle
Lokaler_Name
Systemweiter_Name
Mitarbeiter
MAKartei@Friedberg
...
...
übermittelt angefragte Daten
verweist auf
Friedberg
Mitarbeiter
ID
Name

Transaktionsmanagement

Ziel: Sicherstellung der Datenintegrität

Lösungsmöglichkeiten für Sperroperationen:

- Zentraler Knoten
  - Unabhängigkeit lokaler Komponenten nicht mehr gegeben
- Schreiben sperrt alles – Lesen sperrt eins
  Write locks all – Read locks one
  - Lokale Leseoperationen möglich, falls Relation vorhanden
  - Keine Schreibsperren bei ausgefallenem Knoten möglich
- Majoritätssperren
  - Lese-/Schreibsperre gilt als gesetzt, wenn Mehrheit der Relationen gesperrt werden kann
- Verwendung von Primär- und Sekundärkopien
  - Schreiben in Primärkopie, Lesen aus Sekundärkopie

Lösungsmöglichkeiten für Deadlockbehandlung:

- **Vermeidung** durch Reihenfolgespezifikation der Datenobjekte
  - T1 und T2 möchten gleichzeitig **A_Tabelle** und **B_Tabelle** sperren
  - T1 sperrt zunächst **A_Tabelle**
  - Es würde zu einem Deadlock kommen, wenn T2 nun zuerst **B_Tabelle** sperrt
  - Dies ist jedoch aufgrund der geforderten (alphabetischen) Reihenfolge nicht gestattet!

Lösungsmöglichkeiten für Deadlockbehandlung:

- **Entdeckung und Behebung**:
  - **Timeout**

- Analyse von **Abhängigkeitsgraphen**
  - Standorte tauschen Informationen bzgl. wartender Transaktionen aus
  - Zyklen weisen auf Deadlocks hin

Teuer! - daher so gut wie nicht praktiziert

10 Transaktionsmanagement: Commit

- ACID setzt voraus, dass Transaktionen atomar sind

Also entweder

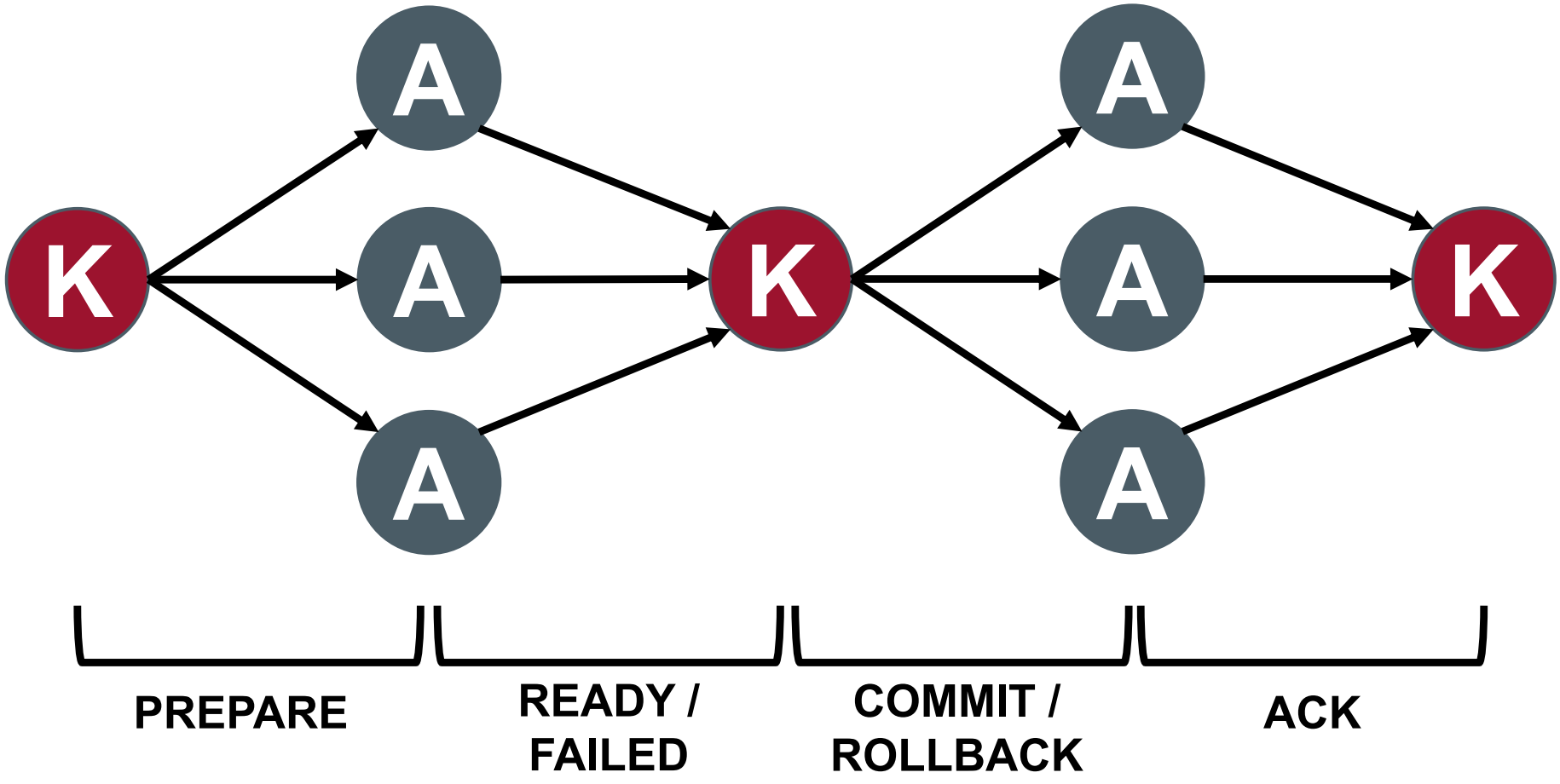
COMMIT ➔ Die verteilte Transaktion wird von allen beteiligten Knoten abgeschlossen.

oder

ROLLBACK ➔ Die Transaktion wird von allen Knoten abgebrochen

Problem: in einem verteilten System
können alle Knoten unabhängig voneinander abstürzen.

10 2-Phase-Commit (2PC)



10 2-Phase-Commit (2PC)

Vorbereitung:

- Transaktionskoordinator sendet PREPARE TO COMMIT an alle beteiligten Stationen
- Empfänger: Logfile schreiben
- Rückmeldung: READY oder FAILED (oder Timeout)
- Koordinator: Fortsetzung oder Abbruch

Finales COMMIT:

- COMMIT an alle Stationen
- Empfänger: Änderung in Datenbank
- Rückmeldung: ACK.

Was wenn hier ein Teilnehmer ausfällt?

10 Das CAP-Theorem

- CAP ist ein Akronym für drei Eigenschaften:
  - Consistency
  - Availability
  - Partition Tolerance
- Kernaussage: In einem verteilten System ist es nicht möglich diese drei Eigenschaften gleichzeitig zu garantieren.
- Die Größen sind graduell, es gibt also kein „Ja oder Nein“.
- Im Jahr 2000 von Eric Brewer vermutet, 2002 axiomatisch bewiesen.
- Gilt nur im Fall von verteilten Systemen!

Weitere Infos: Understanding the CAP Theorem, Akhil Mehra, 2019
https://dzone.com/articles/understanding-the-cap-theorem

10 Die CAP Eigenschaften

Einheitlichkeit (Consistency) der Daten aller Knoten

- Immer konsistent (ACID)
- Schlussendlich konsistent (eventual consistency)

Verfügbarkeit (Availability) des Gesamtsystems

- Anfragen werden immer beantwortet
- Anfragen werden manchmal nicht beantwortet

Partitionstoleranz (Partition tolerance) der Knoten im Gesamtsystem

- Das System kann mit Partitionierung umgehen
- Das System versagt bei Partitionierung ganz oder teilweise

Der Umgang mit einer Netzwerkpartitionierung erfordert Abstriche in Einheitlichkeit oder Verfügbarkeit!

10 Verteilte DB – Eine Demo