Software Architectures for Enterprises SA4E

Peter Sturm
Universität Trier
Winter 2024

8. Scale

Zehn Hoch

4

5

IBM z16

- maximal 57 mal 5.2 GHz Telum CPU
  - CISC CPU
  - 8 Cores pro CPU
  - 200 Cores verfügbar
- maximal 40 TB Speicher
- 5-10 Millionen Dollar

6

Sortieren

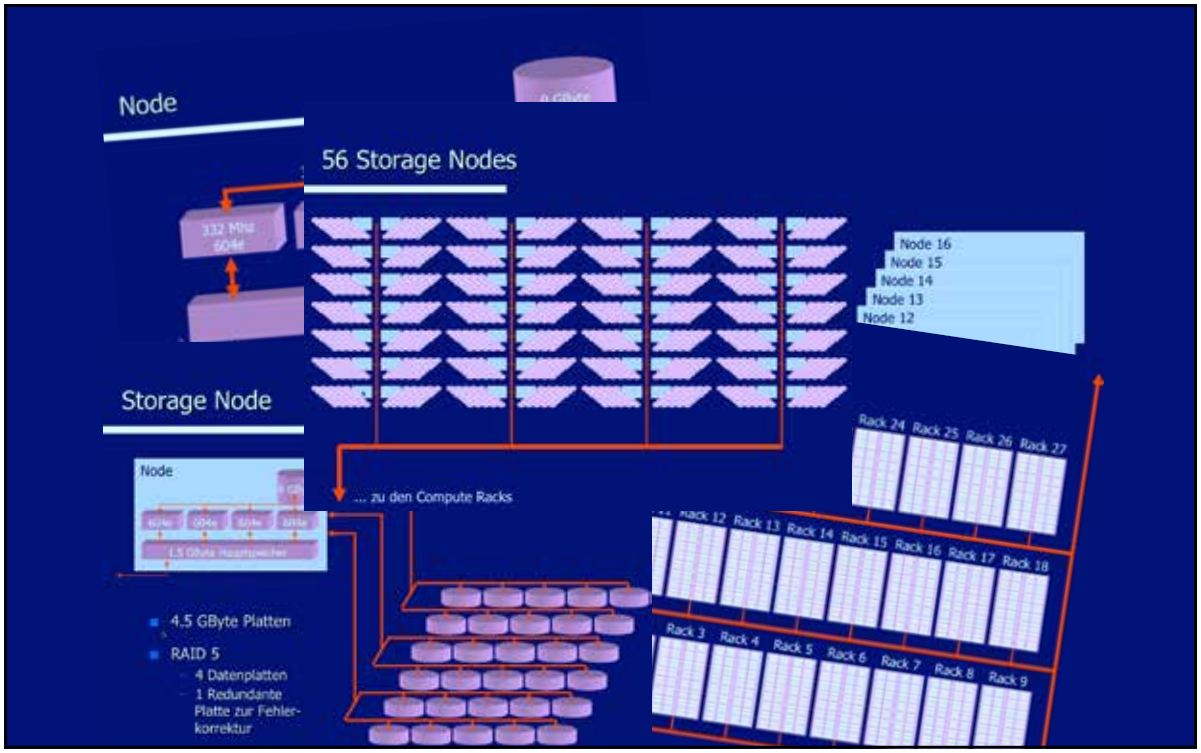
109.951.162.776 Bytes in 1057 Sekunden sortiert

http://www.austin.ibm.com/

Ordnung auch in der größten Unordnung.

Die Aufgabe

- 10 Milliarden Datensätze
- Datensatz
  - 10 Byte Schlüssel (zufällig verteilt)
  - 90 Byte Zusatzdaten (Erhöhen E/A-Anforderung)
- Gesamtgröße: 1 Terabyte (1613 randvolle CD-ROM)
- Eingabe und Ergebnis auf Externspeicher
- Zeit inkl. Programmstart etc.



In Zahlen

- Prozessoren
  - 27 mal 16 mal 4 = 1728 Arbeitsprozessoren
  - 56 mal 4 = 224 E/A-Prozessoren
  - 1952 Prozessoren insgesamt
- Hauptspeicher
  - (27 mal 16 + 56) mal 1.5 GByte = 732 GByte
  - 1.7 Millionen DM bei 300 DM pro 128 MByte
- Lokale Platten
  - (27 mal 16 + 56) mal 9 GByte = 4.2 TByte
  - 122000 DM bei 250 DM pro 9 GByte Platte
- RAID-System
  - 56 mal 6 mal 5 Platten = 1680 Platten
  - 4.5 GByte pro Platte = 7.38 TByte

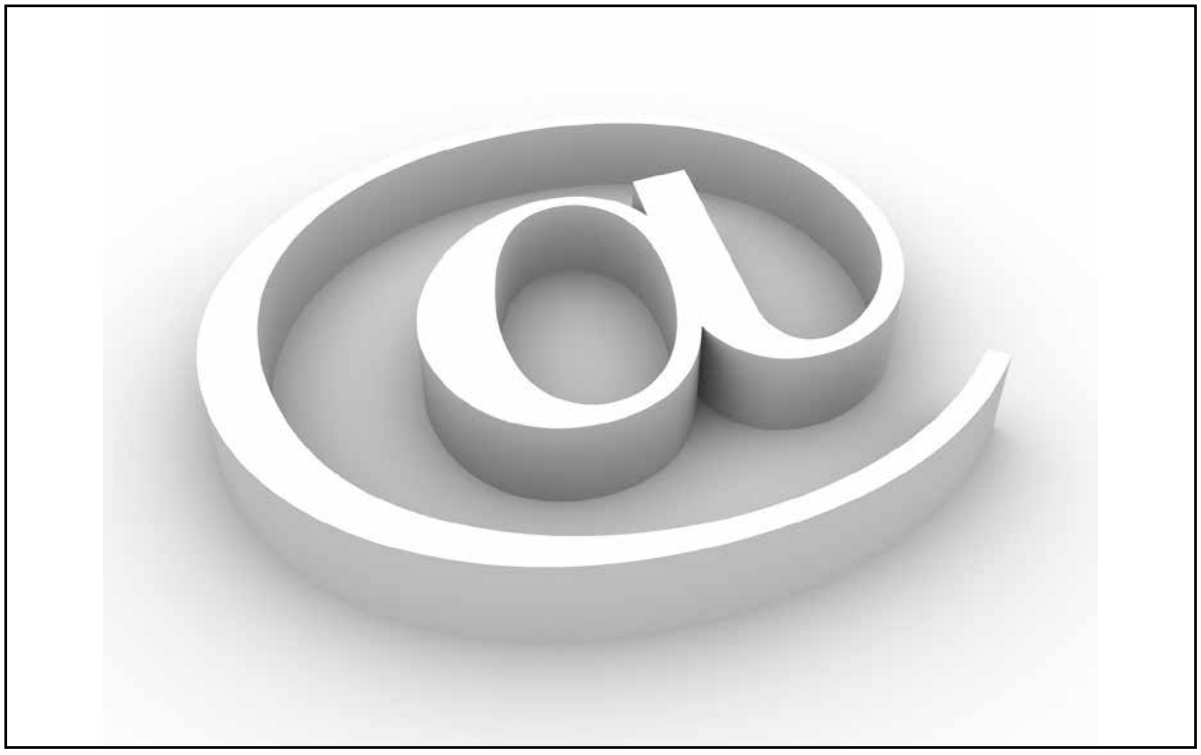
sortbenchmark.org

	Daytona	Indy
Gray	2016, 44.8 TB/min **Tencent Sort** 100 TB in 134 Seconds 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub	2016, 60.7 TB/min **Tencent Sort** 100 TB in 98.8 Seconds 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub
Cloud	2016, $1.44 / TB **NADSort** 100 TB for $144 394 Alibaba Cloud ECS ecs.n1.large nodes x (Haswell E5-2680 v3, 8 GB memory, 40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk) Qian Wang, Rong Gu, Yihua Huang Nanjing University Reynold Xin Databricks Inc. Wei Wu, Jun Song, Junluan Xia Alibaba Group Inc.	2022, $0.97 / TB **Exoshuffle-CloudSort** 100 TB for $97 40 Amazon EC2 i4i.4xlarge nodes 1 Amazon EC2 r6i.2xlarge node Amazon S3 storage Frank Sifei Luan UC Berkeley Stephanie Wang UC Berkeley and Anyscale Samyukta Yagati, Sean Kim, Kenneth Lien, Isaac Ong, Tony Hong UC Berkeley SangBin Cho, Eric Liang Anyscale Ion Stoica UC Berkeley and Anyscale
Minute	2016, 37 TB **Tencent Sort** 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub	2016, 55 TB **Tencent Sort** 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub
Joule 10^{10} recs	2023, 59.3 KJoules **MendSort** 169 K records sorted / joule AMD Ryzen 7900, 96 GB RAM, Nsort, Rocky Linux 9.3, 5 WD SN850X 2 TB SSDs, 3 WD SN850X 1 TB SSDs Igor Mendelev, Levi Mendelev, Yonathan Mendelev mendsort@gmail.com	2023, 52.7 KJoules **AMR5** 190 K records sorted / joule AMD Ryzen 5800U, 64 GB RAM, Nsort, Ubuntu Server 23.04, 2 Samsung 970 EVO Plus 2 TB SSDs, Samsung BAR Plus 64 GB USB Flash Drive Phillip Griffith

Neue Kriterien

- GraySort
  - TBs pro Minute bei sehr großen Datenmengen (> 100 TB)
- CloudSort
  - Sortieren auf öffentlicher Cloud
- PennySort
  - Sortierbare Datenmenge für 1 Penny Systemzeit
- MinuteSort
  - In 1 Minute sortierbare Datenmenge
- JouleSort
  - Benötigte Ernergiemenge (10^8, 10^9, 10^{10} Datensätze)


Scale Out
15



Größe einer Email
im Durchschnitt?

75 KB

(ohne Multimedia)


Emails pro Tag weltweit?


376.400.000.000 pro Tag (2024)

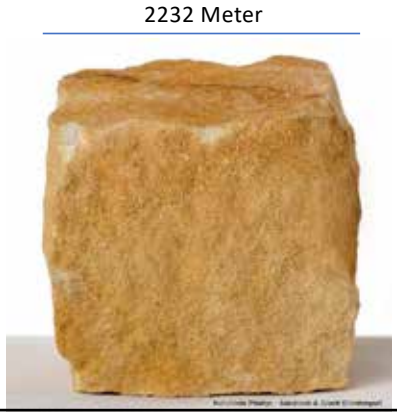
Email

- 376.4 Milliarden Emails pro Tag (2024)
- 75 KB pro Email

25.6 PByte / Tag

In Smarties?

- 1 Byte = 1 Gramm
- 28,907,520,000 Tonnen



Verarbeiten

- Mindestens jedes Zeichen sehen
  - Schlüsselwörter
- Einzelrechner (400 MB/s)
  - Daten sind auf Platte
  - 105 Tage pro Tag

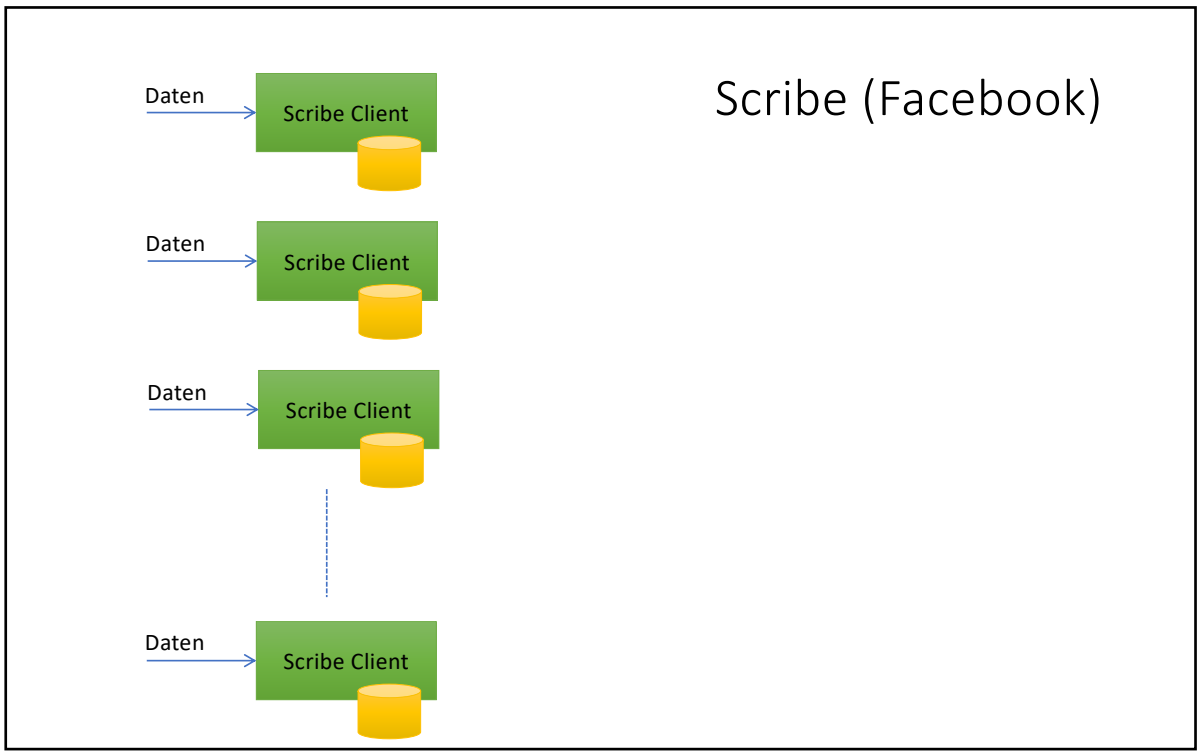
Daten erst mal einfangen!

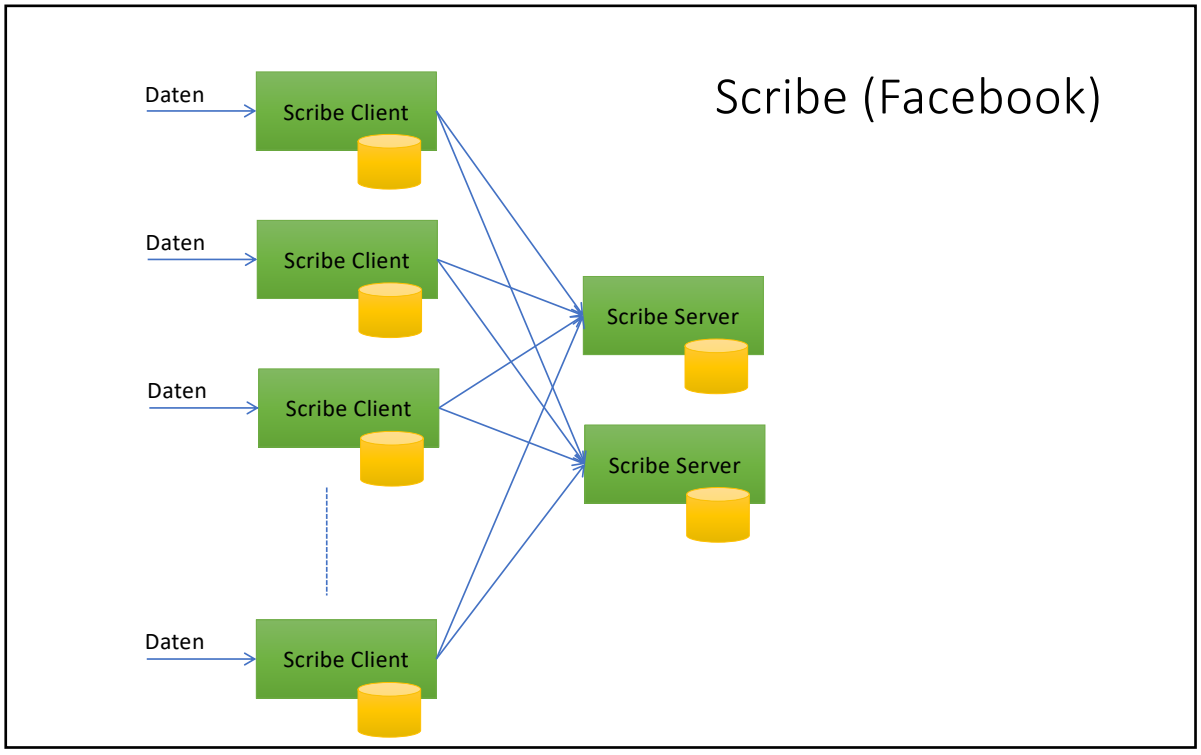
- 42.3 GByte / s = 338 GBit / s
- Unzählige Quellen

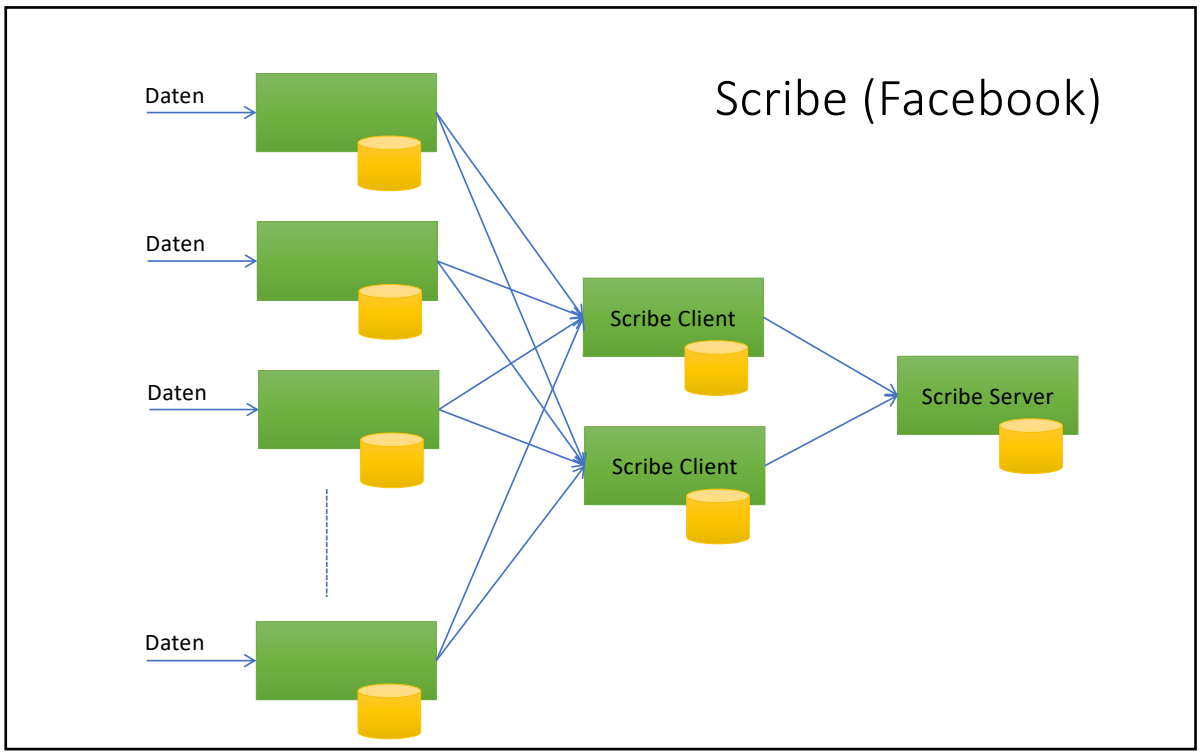
Facebook

- 3.7 Milliarden monatlich aktive Nutzer (Ende 2024)
- 2.11 Milliarden täglich aktive Nutzer
- Data Warehouse ~ 300PB
- Täglich ~ 4PB neue Daten:
  - 3.63 Milliarden Posts
  - 350 Millionen Fotos
  - 108 Milliarden Kommentare
  - 4 Millionen Likes pro Minute

28


Scribe (Facebook)
Daten
Scribe Client
Daten
Scribe Client
Daten
Scribe Client
Daten
Scribe Client

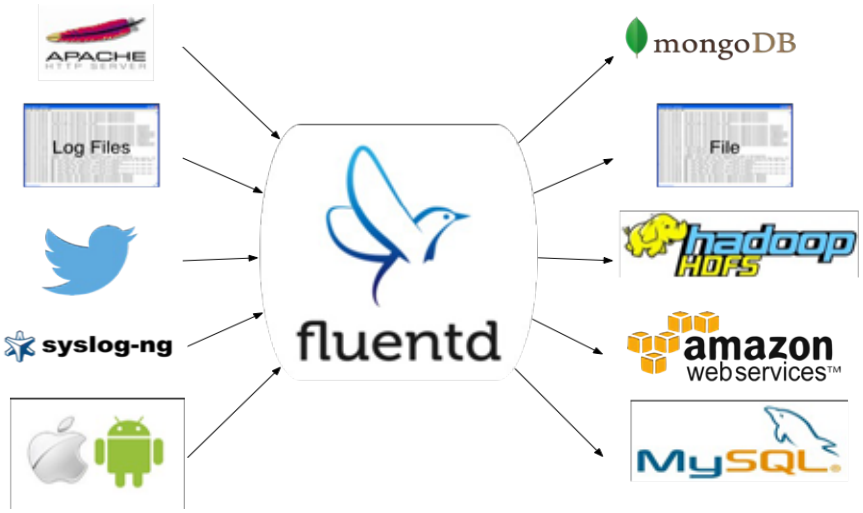

Scribe (Facebook)
Daten
Scribe Client
Daten
Scribe Client
Daten
Scribe Client
Daten
Scribe Client
Scribe Server
Scribe Server



Scribe (Facebook)

- 2 Tupel (category:string, message:string)
- Keine transaktionale Garantien
- C++ Implementierung
- Apache Thrift (RPC)

fluentd



twitter

Mehr als 400 Millionen Tweets pro Tag (2012)

Juni 2011

2200 TPS (Tweets per Second)
= 190 Millionen pro Tag

18000 QPS (Queries per Second)
= 1.5 Milliarden pro Tag

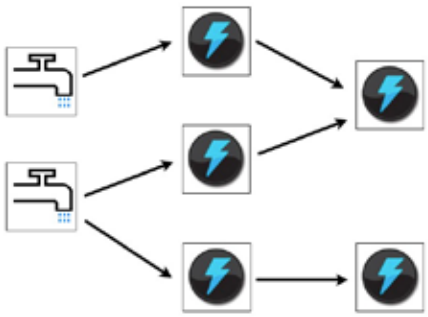
Twitter

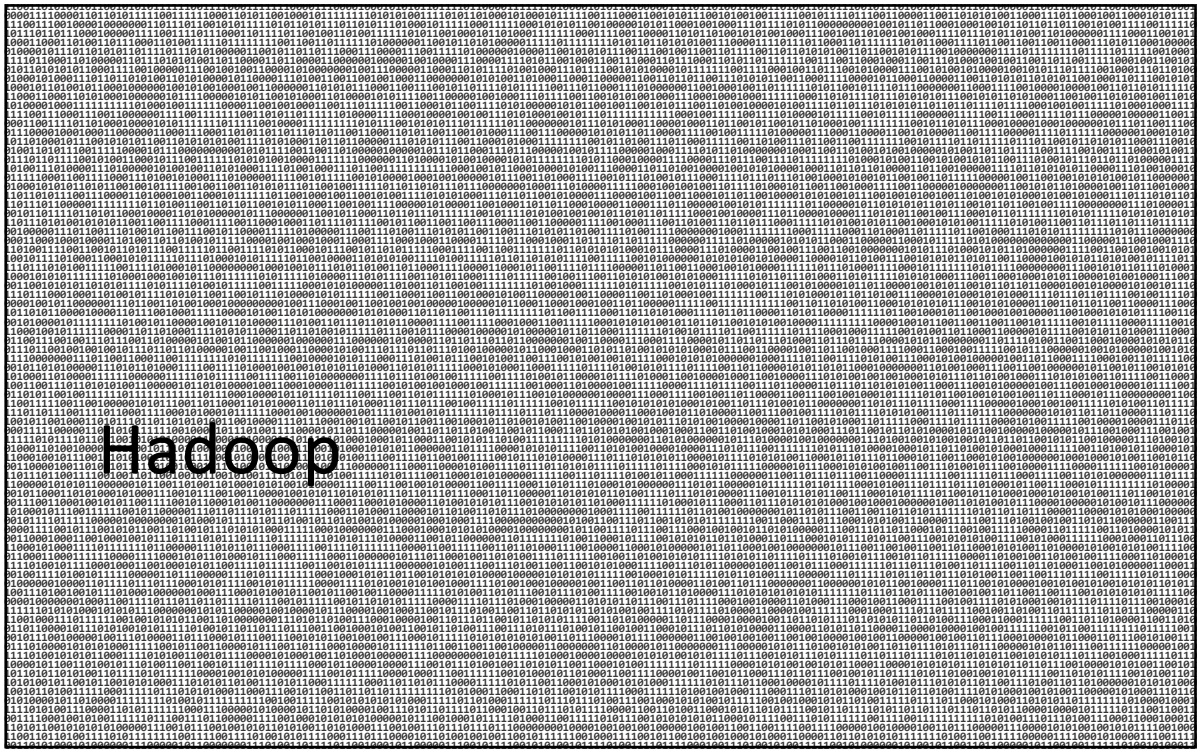
- Timeline Cache
  - Redis (In-Memory, Key/Value, Snapshots & Logs)
- Real-Time Search Index
  - Earlybird (Invertierter Index)

- Fast schon langweilig? Zu klein?
  - 50-60 GB Daten pro Tag
  - rund 2 Milliarden Queries pro Tag (23000/s)

Apache Storm

- Realtime processing of streams
  - Hadoop for batch processing
- Interoperable
  - Thrift
- Trident
  - Höhere Semantiken

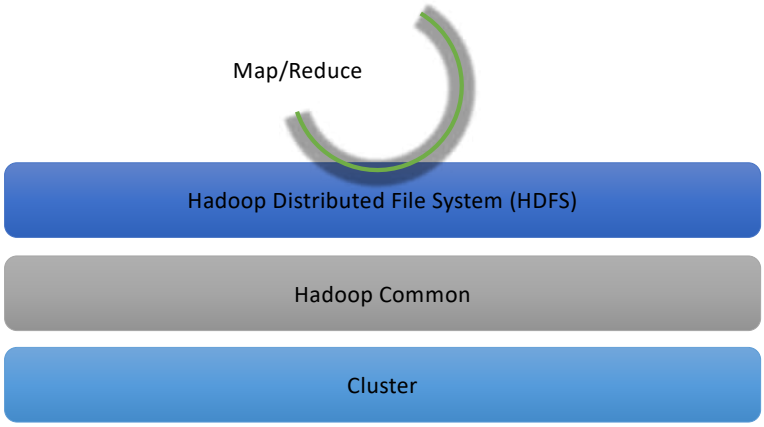




Ziele

- Framework for
  - reliable
  - scalable
  - distributed
- computation of large data sets

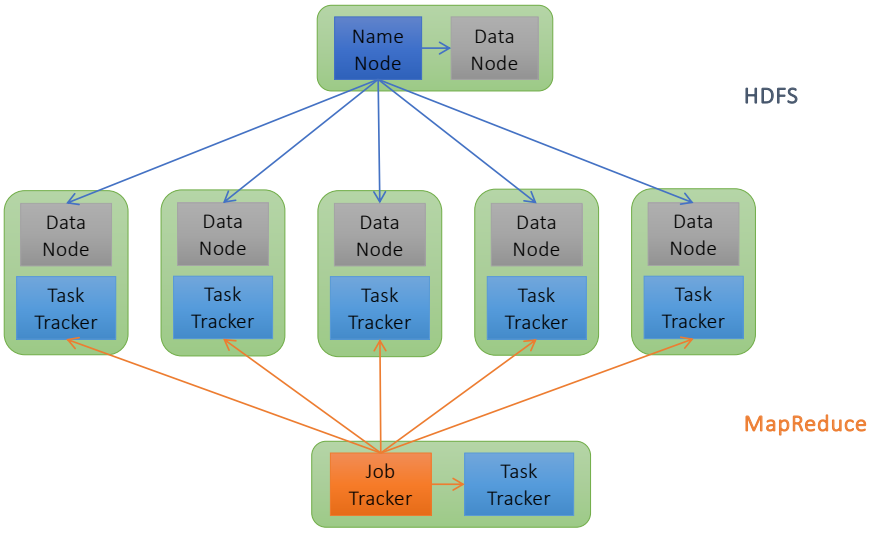
Aufbau



Nobody is perfect!

- Cluster besteht aus Massen-PCs
- Einzelnes Gerät ist nicht zuverlässig
- „Keep it simple“
  - Lieber mehr Muskeln als mehr Hirn
- Heterogene Komponenten

Hadoop Cluster Setup



Dateien

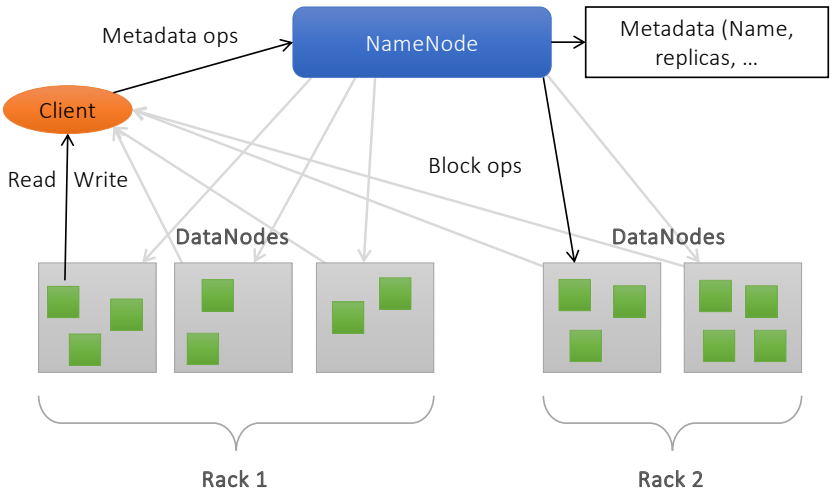
- Große Dateien (mehr als 1 GB)
- Klassische Hierarchie
- Datei besteht aus Blöcken
- Große Blöcke (64 MB und mehr)
- Steuerbarer Replikationsgrad (Blockebene)

HDFS Architektur (1)

- Master/Slave
  - Rack aware
- Name Node (Master):
  - Verwaltet Dateisystem
  - Beantwortet Zugriffswünsche
  - Verteilt die Blöcke und Replikate über Data Nodes
- Name Node ist redundant
  - Manuelles Handover

HDFS Architektur (2)

- Data Nodes (Slaves):
  - Speichern die Datenblöcke
  - Bedienen Lese- und Schreibzugriffe
  - Senden periodisches Heartbeat an Name Node
  - Setzen Aufträge der Name Node um (Create, Delete, ...)
- Zugriffsmodell
  - WORM (Write once, read many)
  - File Streaming zu den Clients

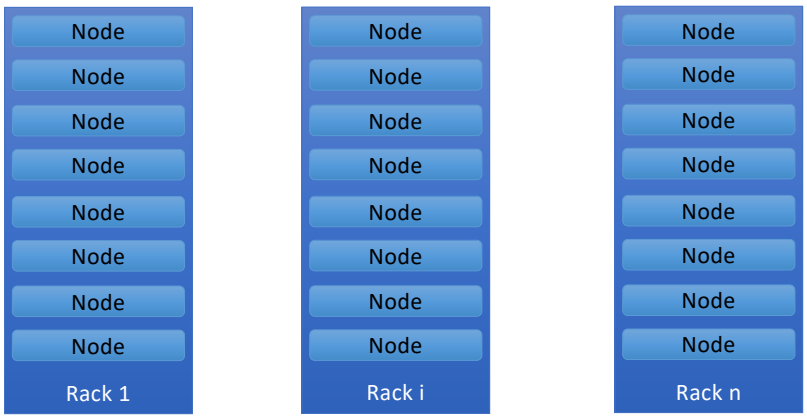


Replikation

- Replikationsfaktor
  - Konfigurierbar pro Datei
  - Jederzeit änderbar
- Name Node organisiert Replikation
  - Legt Default-Faktor fest
  - Entscheidet über die Blockverteilung
  - Reagiert auf ausfallende Replikate

Replikationsstrategie

- Default-Faktor = 3



Locality Awareness

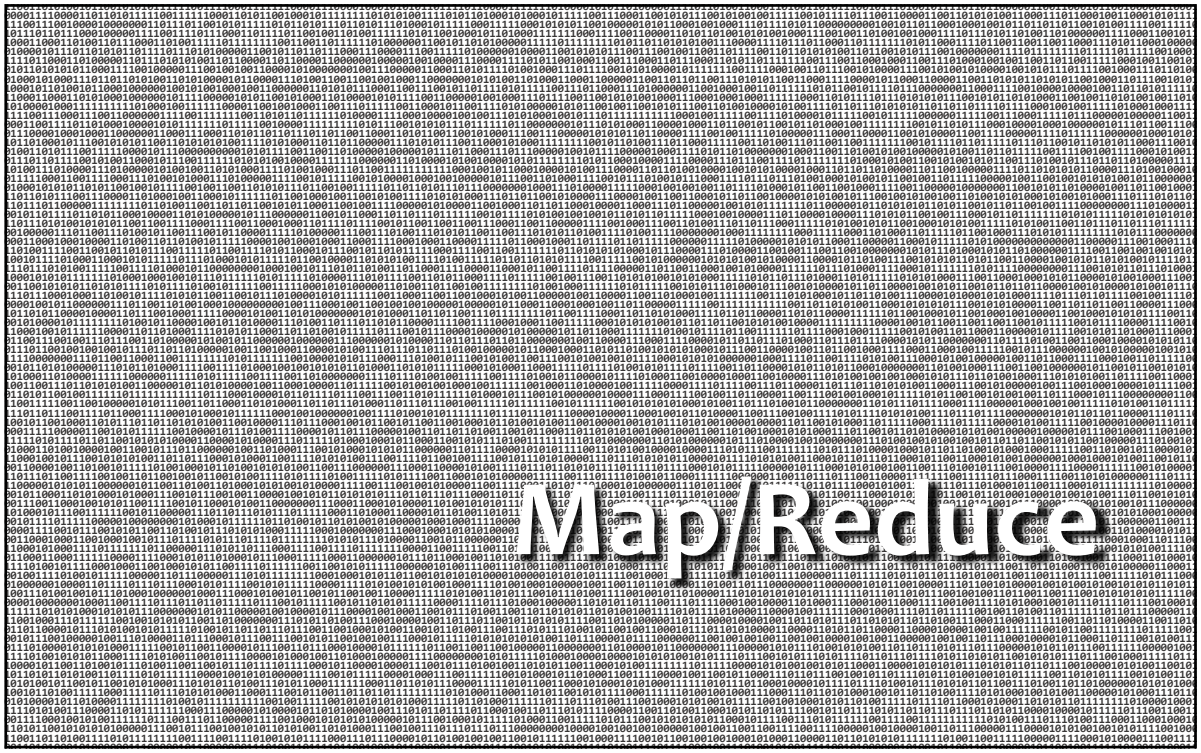
- Leseauftrag des Clients
- HDFS wählt die “nächste” Data Node
- Ideal, wenn Client auf gleichem Rechner

Zuverlässigkeit der Metadaten

- Atomares Metadaten-Log (*EditLog*)
- Vollständiger Namensraum des Dateisystems in einer Datei *FsImage*
- Kopie von FsImage im Hauptspeicher
  - Checkpointing
    - EditLog wird in FSImage eingepflegt

EditLog- und FsImage-Duplikate

- Snapshots
  - Für die Zukunft geplant
  - Kopie des Dateisystemzustand zum Zeitpunkt t
  - Roll Backs
- Secondary Name Node

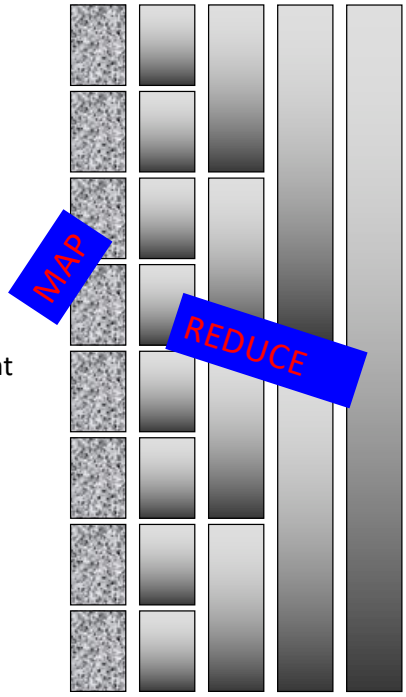


Idee

- Verfahren soll möglichst lange mit der Anzahl der eingesetzten Rechner skalieren
- Analoges Problem zum Sortieren
- Dynamik/Elastizität wünschenswert
  - Neue Rechner „schnell“ hinzufügen
  - Trading von Ressourcen bei langen Tasks

Analogie

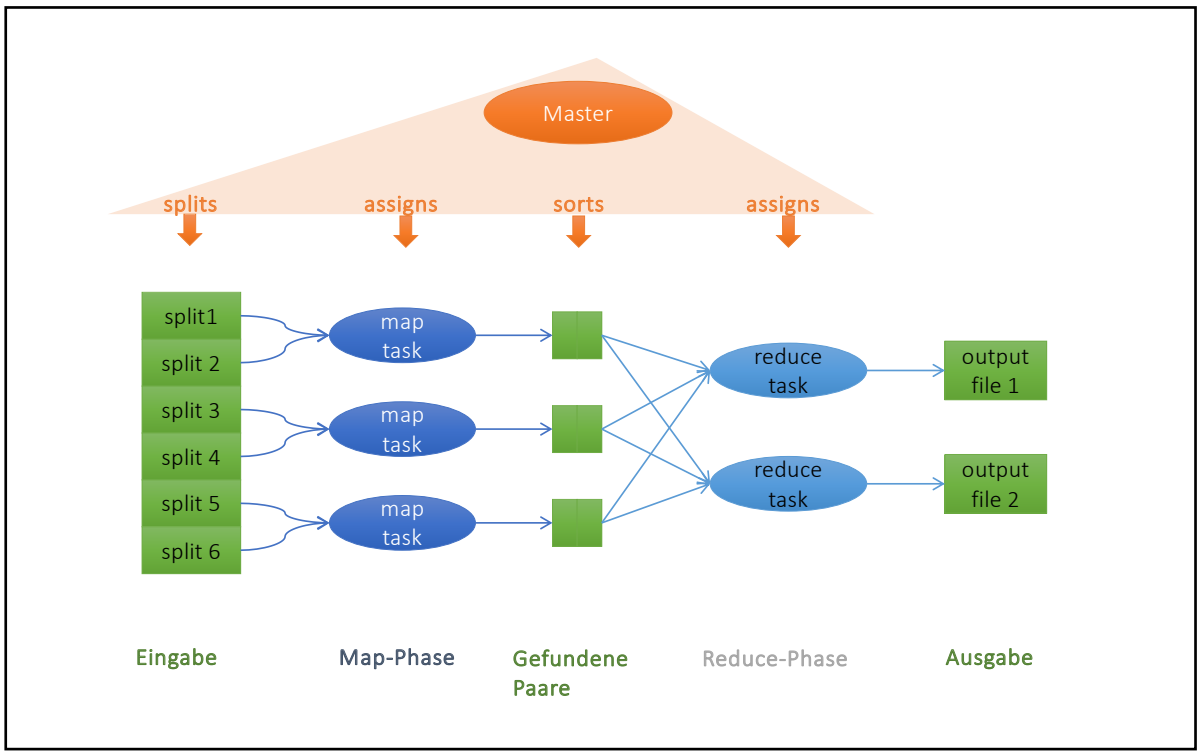
- Mergesort
  - Parallelitätsgrad sinkt gegen Ende
  - Wäre für den Weltrekord zu schlecht
- Ansatz reicht aber hier
  - Phase 1 komplexer
  - Ideal nur noch Phase 2



Map/Reduce

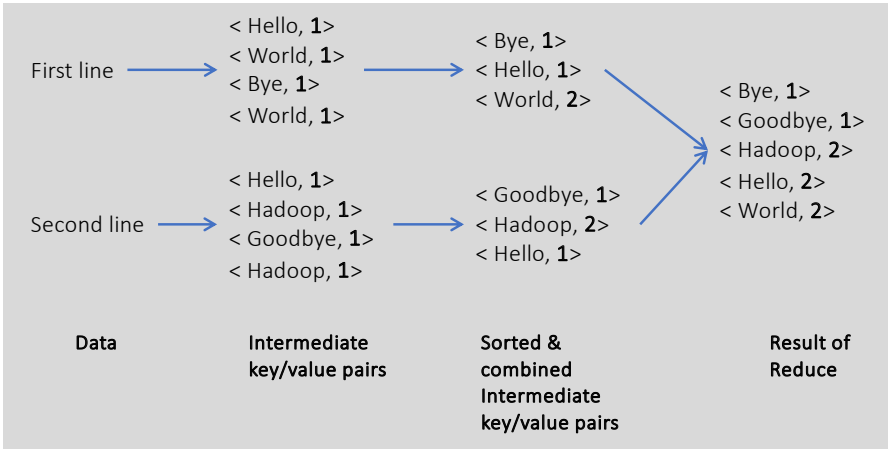
- Pattern “Divide & Conquer”
- Funktioniert besonders gut bei semistrukturierten Daten
  - Key/Value-Paare
- Map-Phase
  - Filter für gesuchte Paare
- Reduce-Phase
  - Verarbeiten der gefundenen Paare

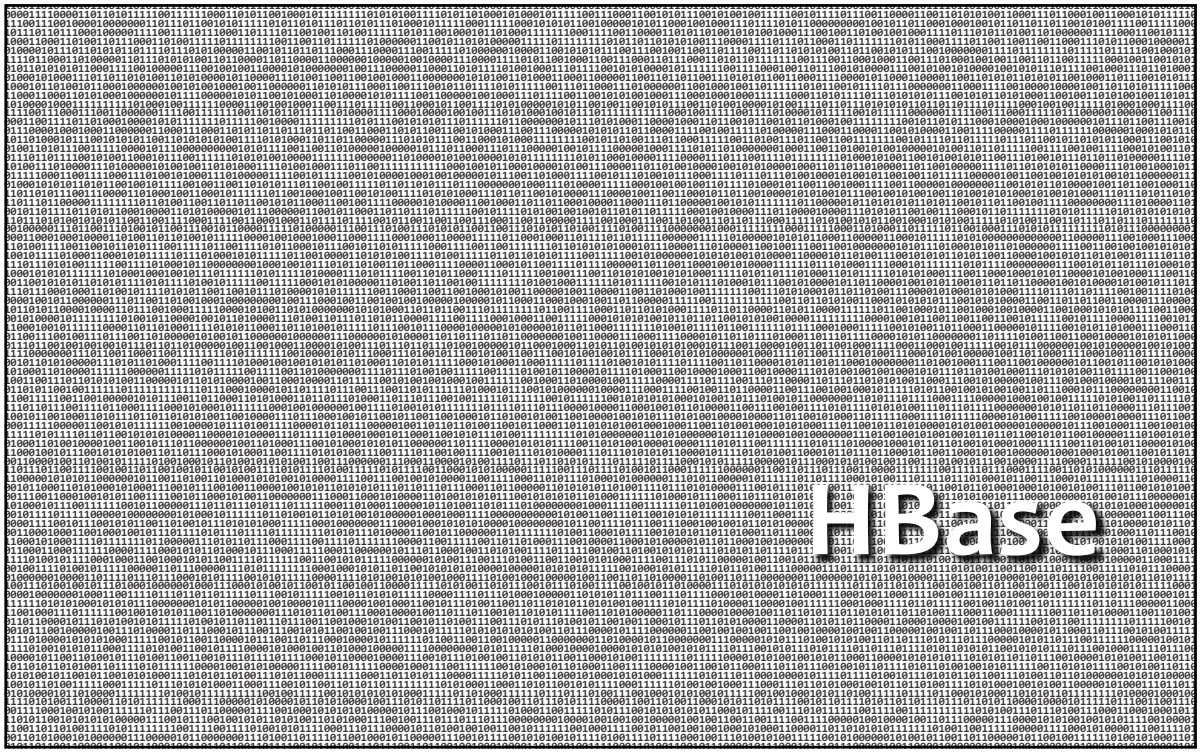
„Simplified data processing on large clusters
Jeffrey Dean, Sanjay Ghemawat in Communications of the ACM (2008)“



Wordcount

- Datei 1: „Hello World Bye World“
- Datei 2: „Hello Hadoop Goodbye Hadoop“





HBase

- Datenbank
  - Setzt auf Hadoop HDFS auf
- Basiert auf Google BigTable
- Erlaubt den schnellen Zugriff auf einzelne Records
  - HDFS kann das nicht

Datenmodell

- Matrix
  - Zeilen: Sortierter Schlüssel (primary key)
  - Spalten: Sortiert nach Arten
- Lexikograpische Ordnung auf “Bytes”
- Matrixzellen
  - enthalten beliebige Inhalte
  - Mehrfachinhalte (Zeitstempel)

Table: webtable

Row key	Time Stamp	ColumFamily contents	ColumnFamily anchor
“com.cnn.www“	T9		Anchor:cnnsi.com=“CNN”
“com.cnn.www“	T8		Anchor:my.look.ca=“CNN.com”
“com.cnn.www“	T6	Contents:html=“<html>…“	
“com.cnn.www“	T5	Contents:html=“<html>…“	
“com.cnn.www“	t3	Contents:html=“<html>…“	

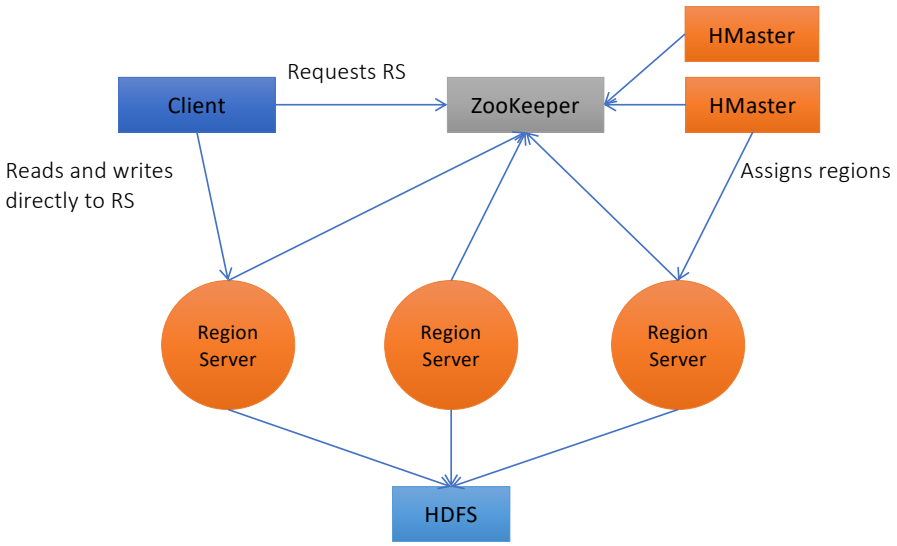
http://hbase.apache.org/book/datamodel.html

- Request values for all columns of row **“com.cnn.www“**

  => Contents:html=“<html>…” (at T6)
  Anchor:cnnsi.com=“CNN” (at T9)
  Anchor:my.look.ca=“CNN.com” (at T8)

Google Bigtable

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes



„Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it ...“

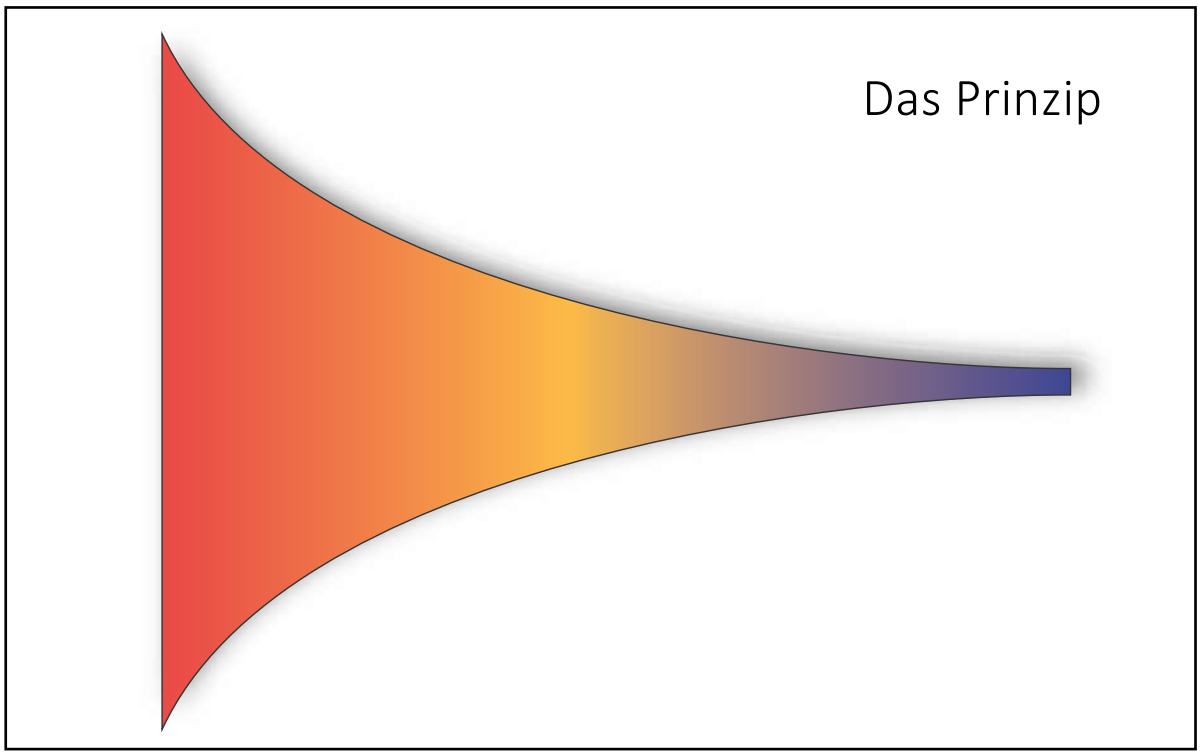
Dan Ariely (Duke University)

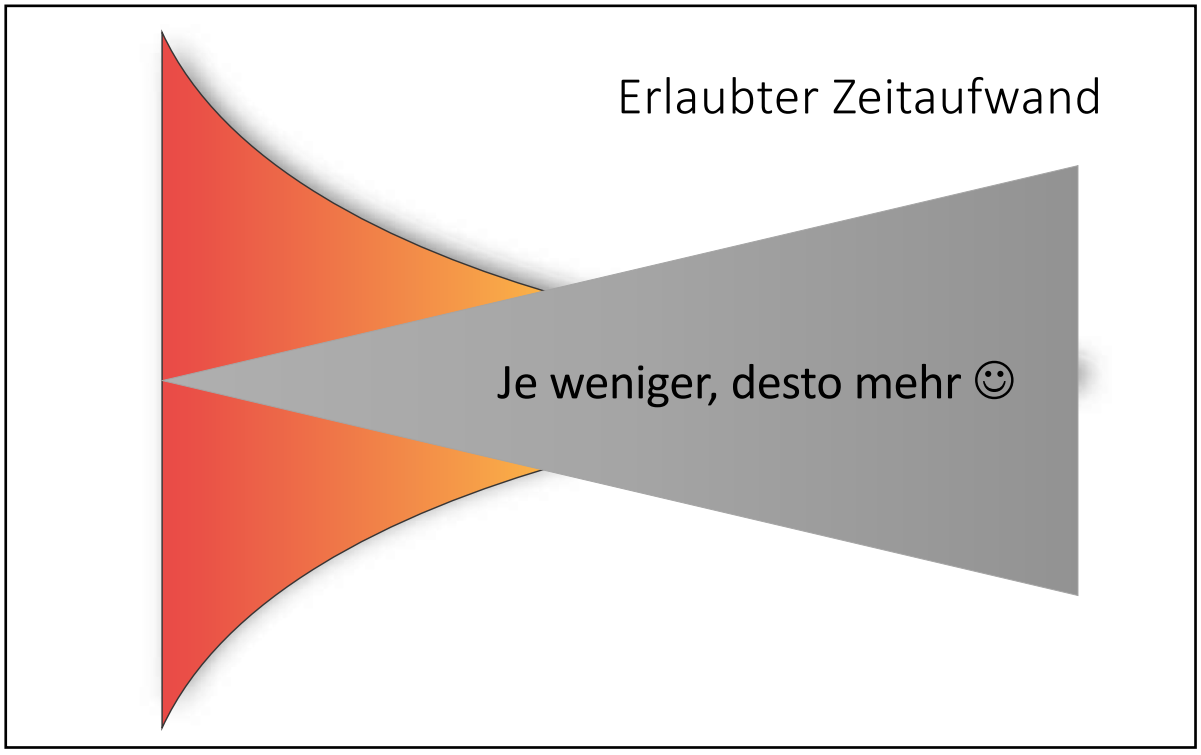
The Four V's

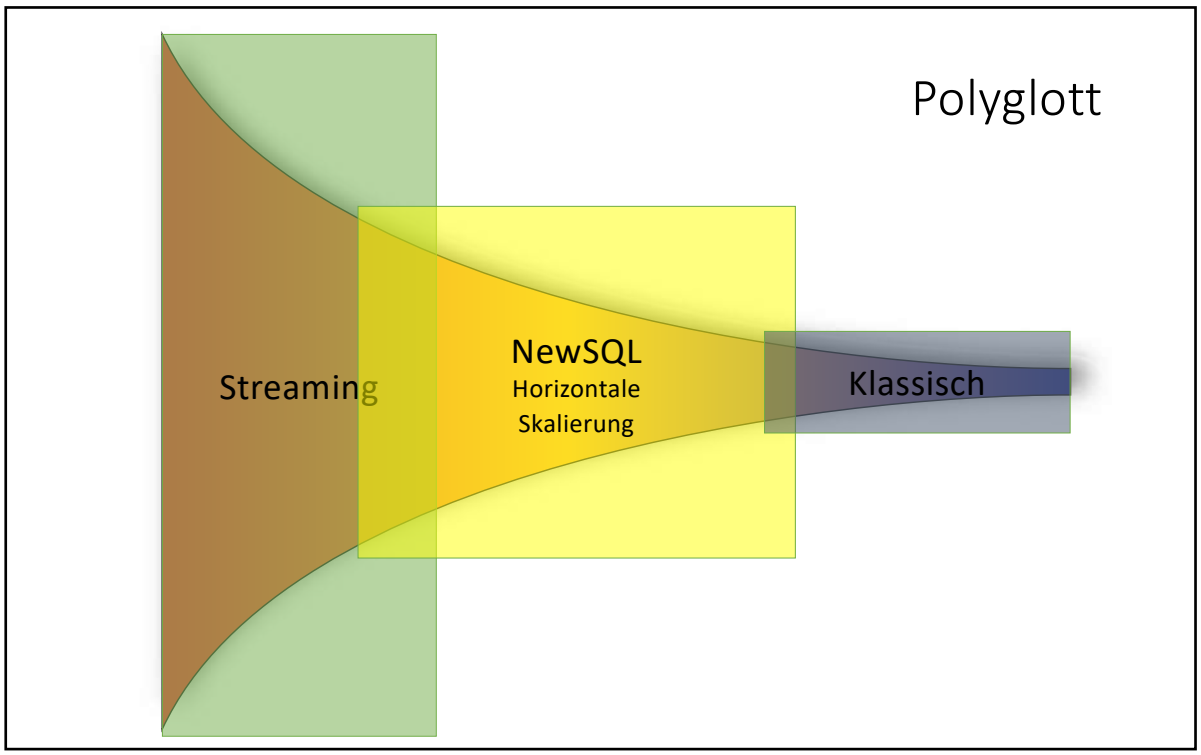
Volume
Variety
Velocity
Veracity

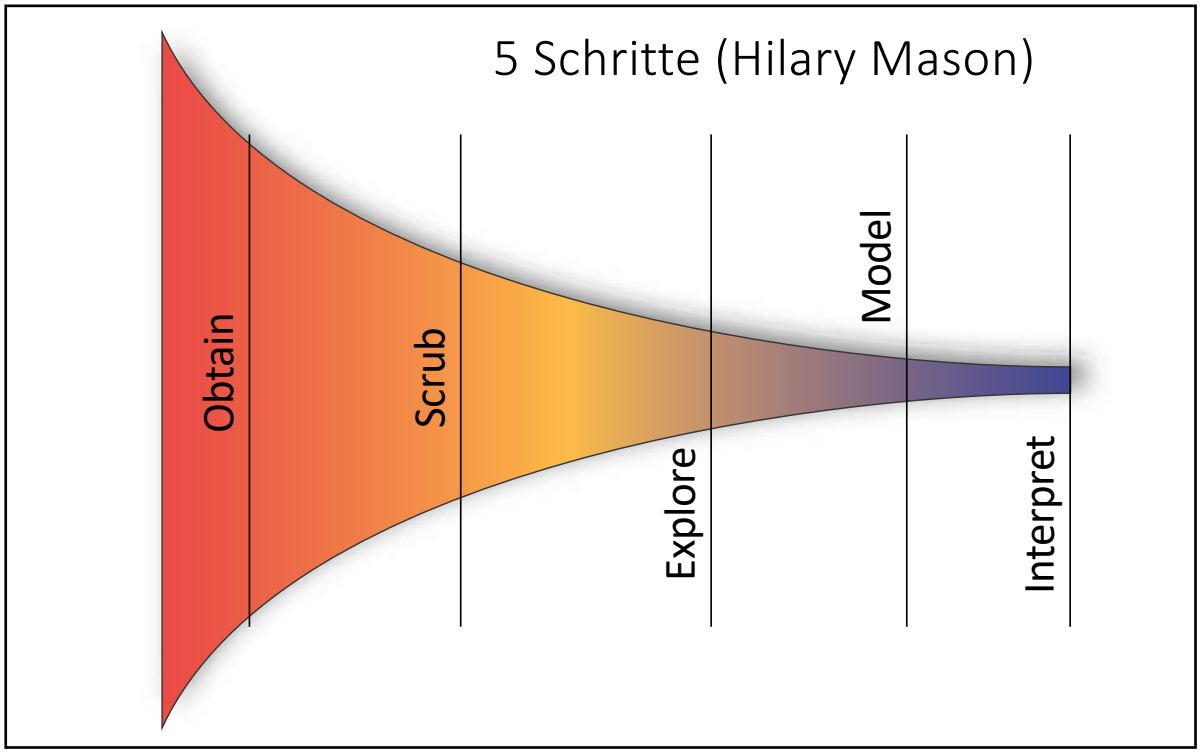
„Cheaper to keep data than to throw it away!“

Hilary Mason (ehemals bitly jetzt „Fast Forward Lab“)


Das Prinzip


Erlaubter Zeitaufwand
Je weniger, desto mehr ☺


Polyglott
Streaming
NewSQL
Horizontale
Skalierung
Klassisch


5 Schritte (Hilary Mason)
Obtain
Scrub
Explore
Model
Interpret

1. Obtain

- Dran kommen ist manchmal schwer
  - Abhören von Unterwasserkabeln
- Handwerkliches Geschick
- Datenmenge
- Lastspitzen

2. Scrub

- Balanceakt
  - Speicherlatenzen vs. CPU-Zeit
- Häufig unabhängige Ereignisse
  - Verteilung
- Filtertechniken
- „False Positives“ bevorzugt

3. Explore

- Informationssichtung
- Datenqualität bewerten
- Analyse- und Visualisierungswerkzeuge
- Hier beginnt „Business Intelligence“

4. Model

- Verstehen und Vorhersagen
- Validierung gegen Testmengen

5. Interpret

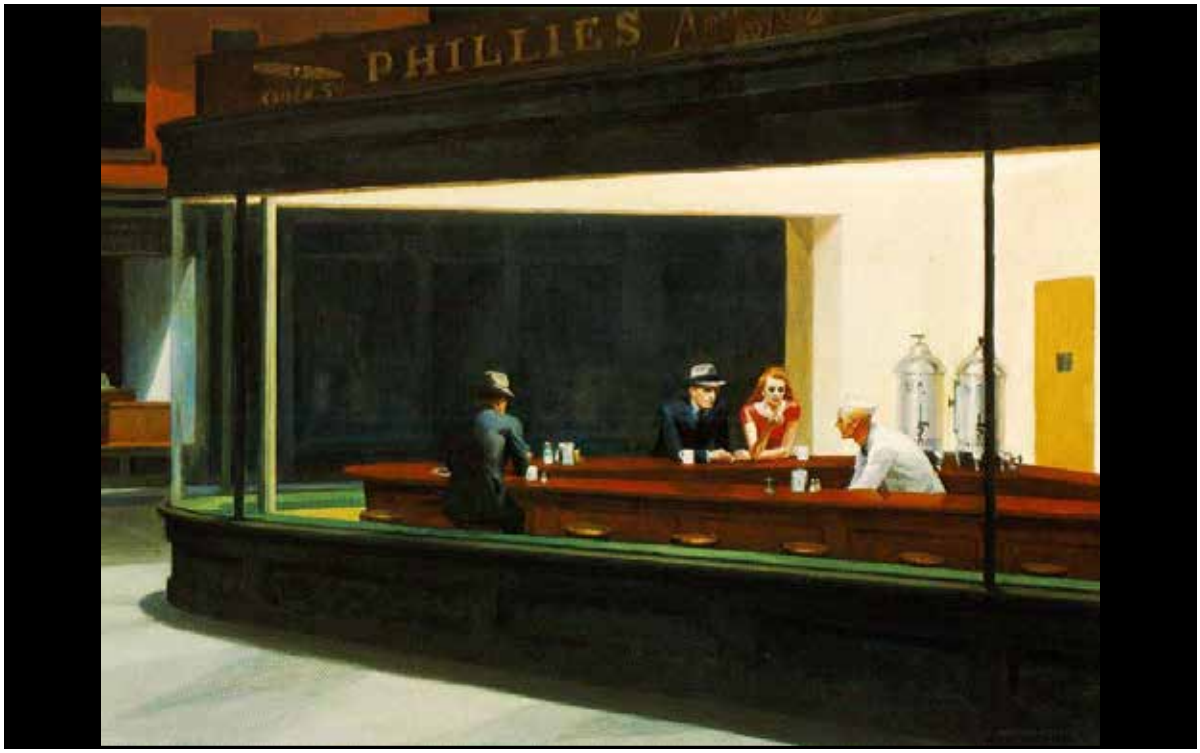
- Beherrschungsstufe
- Generalisierungen
- Transformieren

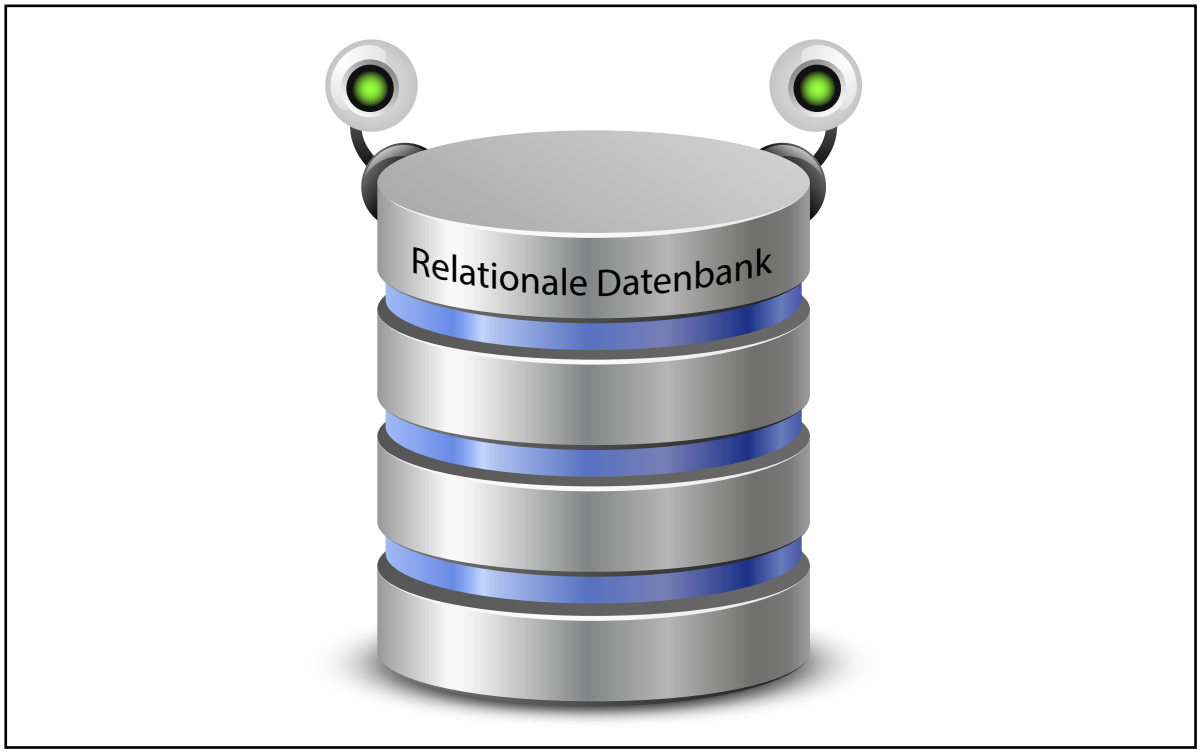
„OSEMN“

- Taxonomie für „Data Science“
- Interdisziplinarität
- Neue Berufsfelder
  - Informatik
  - Statistik
  - Mathematik
  - Anwendungsdomäne
- 4th Paradigm

NewSQL

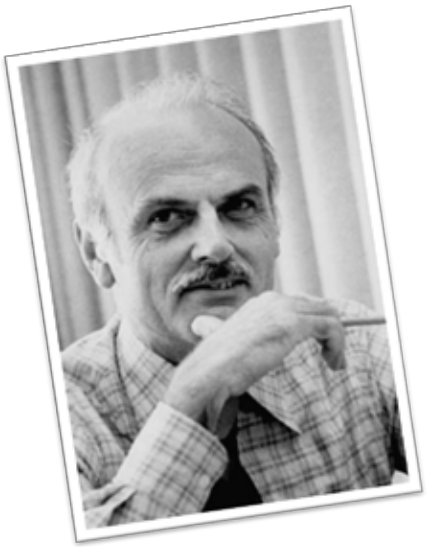
83


PHILLIES


Relationale Datenbank

RDBMs

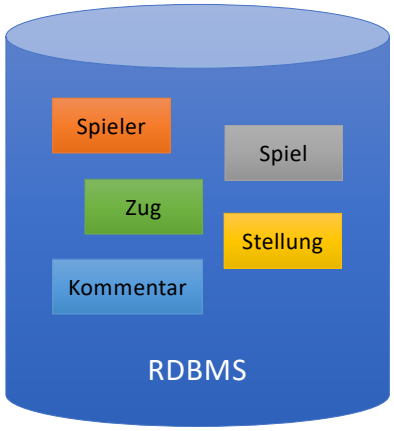
- 1970, IBM
- Edgar Codd
- Persistente Datenspeicherung
- Formal fundierte Methode
- Transaktionen
- Retrievalsprache



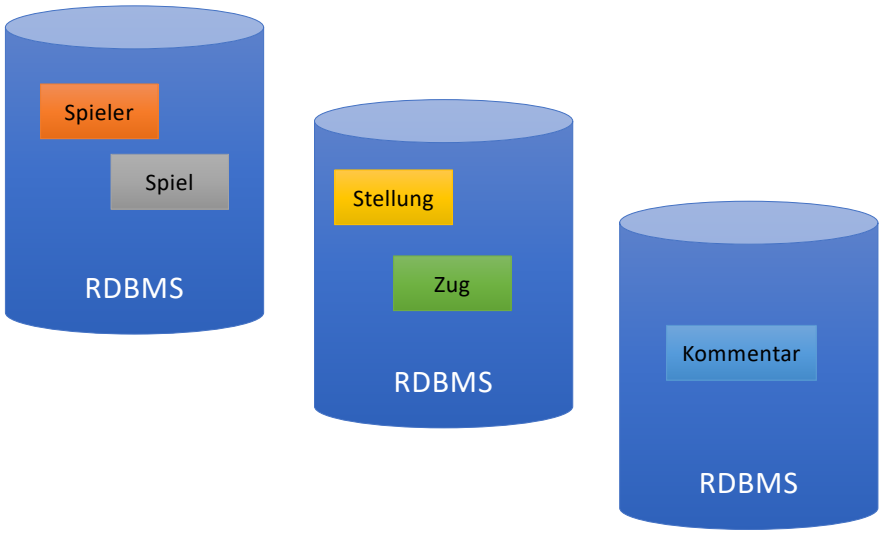
Kleinere Wehwehchen ☺

- Impedance Mismatch
  - Objektorientiert vs. Relational
- Shared Database Integration
  - Wachsende Komplexität
- Application Databases
  - Integration über Services
  - Service-Orientation

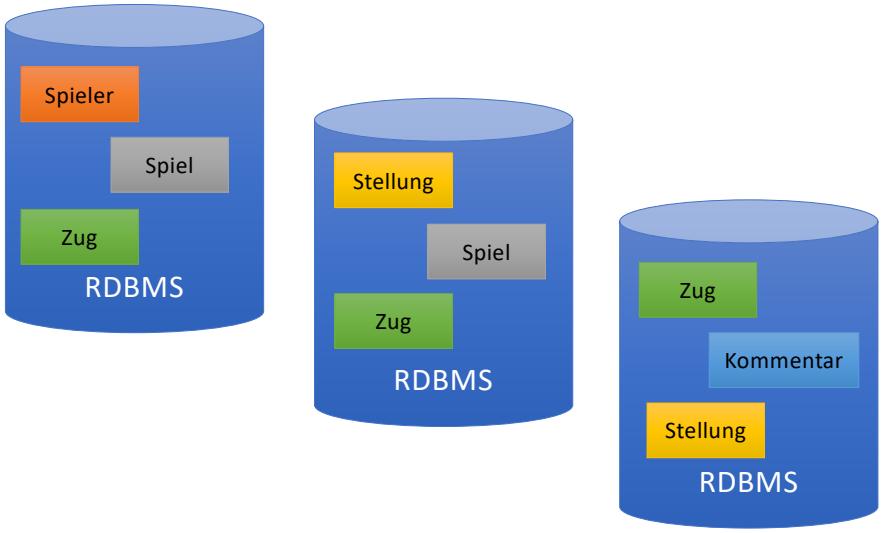
Ein Server (RDBMS)



Mehrere Server (Distribution)



Mehrere Server (Partitioning)



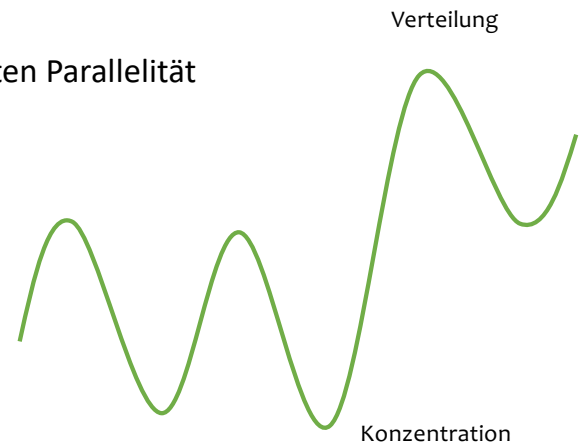
Replikation

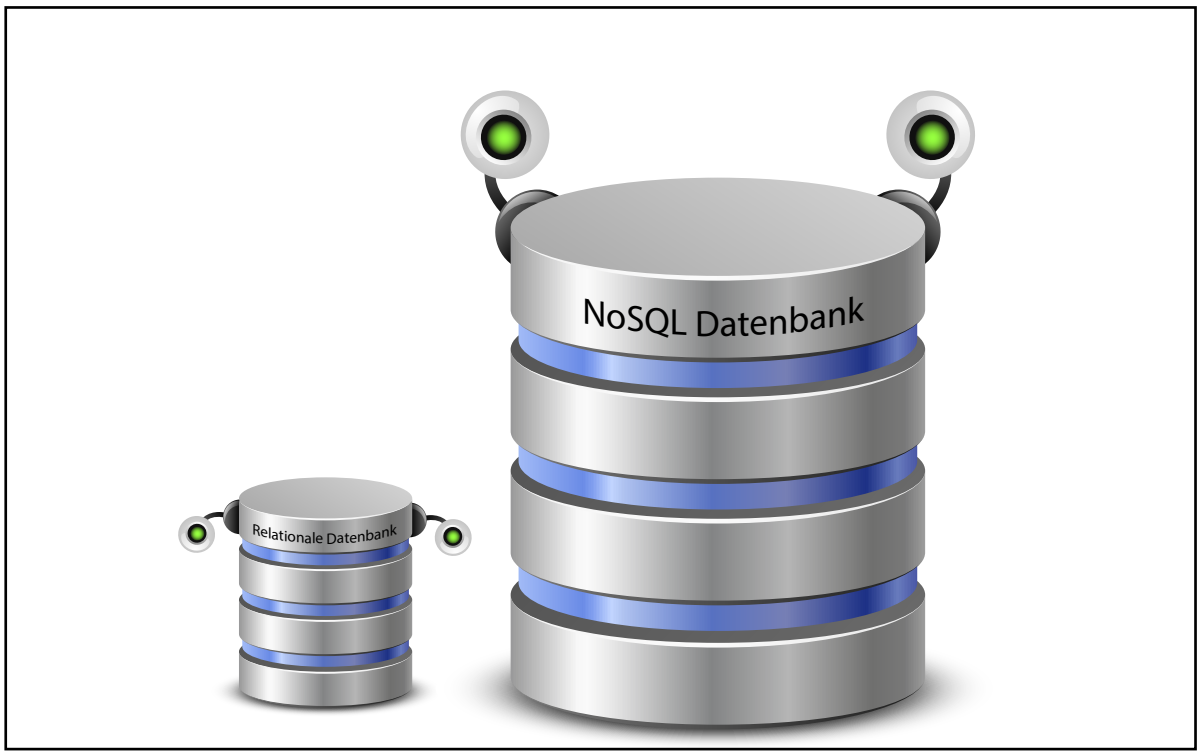
- Synchron
  - 2PC/3PC, Zeitaufwendig
- Asynchron
  - Store and Forward
    - Periodisch
    - Aperiodisch

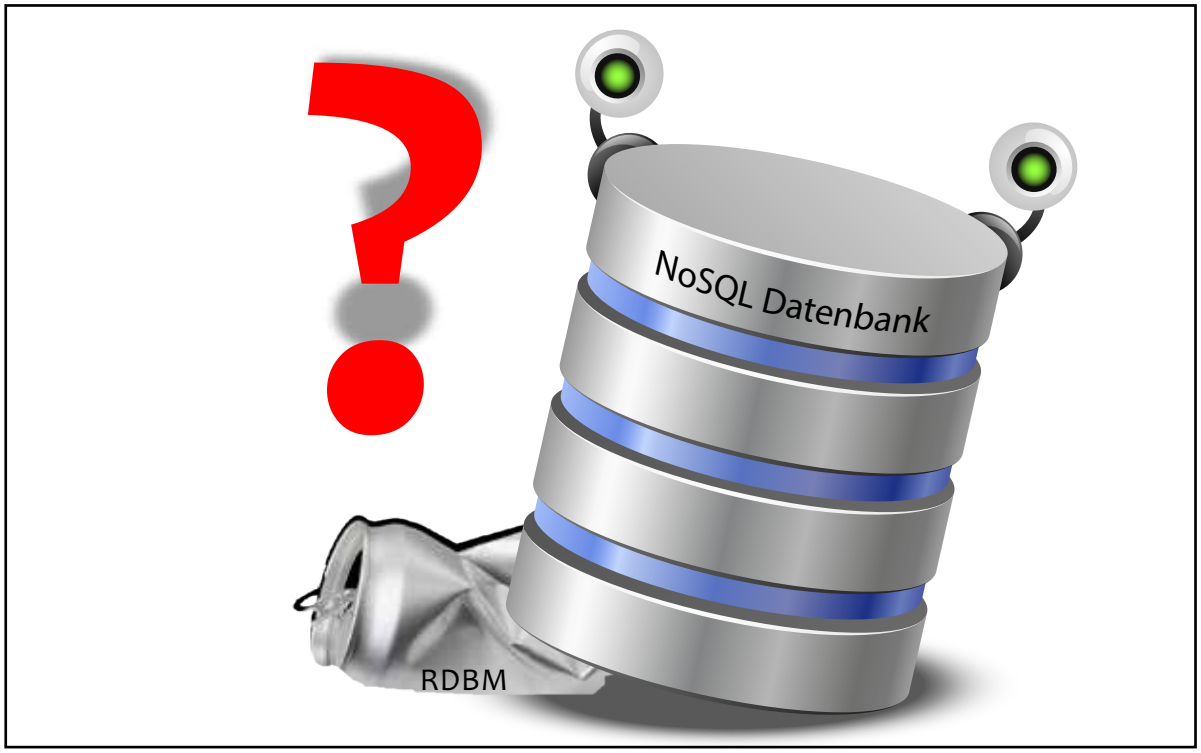
- Snapshot Replication
- Transactional Replication
- Merge Replication
- Statement-based Replication
  - SQL Statements

Zäsur

- Skalierbarkeit bei RDBMS hat Grenzen
- Datenvolumen
- Ausschöpfen der inhärenten Parallelität
  - Echtzeit-Internet
- Zuverlässigkeit
- Verfügbarkeit
- Neue Denkmodelle




NoSQL Datenbank
Relationale Datenbank



Warum?

- Skalierbarkeit
  - Riesige Datenmengen
- Performanz und Elastizität
  - Auslastung großer Cluster
- Agile Modelle
  - „Erst schießen dann fragen“

Konkreter

- Aus Spaß an Neuem
- Hohes Datenwachstum
- Hohe Latenzen
- Online statt Offline

Konsequenz

- Viele Server (10^4 und mehr)
  - Netzwerkpartitionierungen sind normal
- Mehr Freiheitsgrade
  - Datenkonsistenz
  - Latenz
- Nicht alle Anwendungen mögen das

```
int x = 17
if (x ==17) {
      x = 42;
}
```

x = 42

Themengebiete

- Schema-less statt Schema
- BASE statt ACID
- Horizontale statt vertikale Skalierung
- CAP
  - Irgendwann konsistent, garantiert!

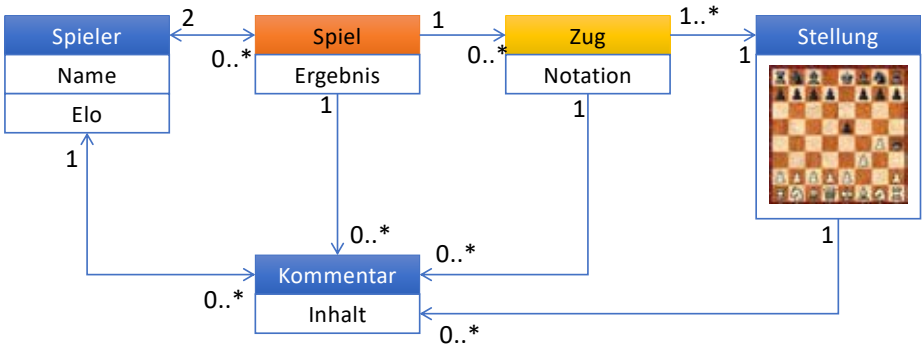
Mit oder ohne Schema

SQL

Schema

- A priori erstelltes Datenmodell
  - Objekte
  - Objektbeziehungen inkl. Kardinalitäten
- Erweitertes Entity-Relationship-Modell
- Abbildung auf Relationen
  - Normalisierung
  - Vermeidung von Datenredundanzen
  - Werzeugunterstützt

Schema



Relationen

Spieler

PKey	Name	Elo
1	Carlsen	2868
2	Aronian	2813
3	Kramnik	2811
4	Anand	2783
5	Sturm	42

Spiel

GKey	Weiss	Schwarz	Resultat
10	1	2	1:0
11	5	1	0:1

Zug

GKey	Lfd	PosKey	Notation
11	1	1234	f3
11	2	1235	e5
11	3	1243	g4
11	4	4242	Qh4#
10	1	1542	e4

Stellung 4242 (Fool‘s Mate)

SQL

- Structured Query Language
  - Definition
  - Manipulation
  - Control
  - Retrieval
- 1979 entstanden, 1986 ANSI, 1987 ISO
- 2011: SQL/XML

Join

- Tabellenverknüpfung
  - Primär- und Fremdschlüssel

Spieler

PKey	Name	Elo
1	Carlsen	2868
2	Aronian	2813
3	Kramnik	2811
4	Anand	2783
5	Sturm	42

Spiel

GKey	Weiss	Schwarz	Resultat
10	1	2	1:0
11	5	1	0:1

```
SELECT Spieler.Name
FROM Spieler INNER JOIN Spiel
ON Spieler.PKey = Spiel.Weiss
WHERE Spiel.Resultat = "1:0"
```

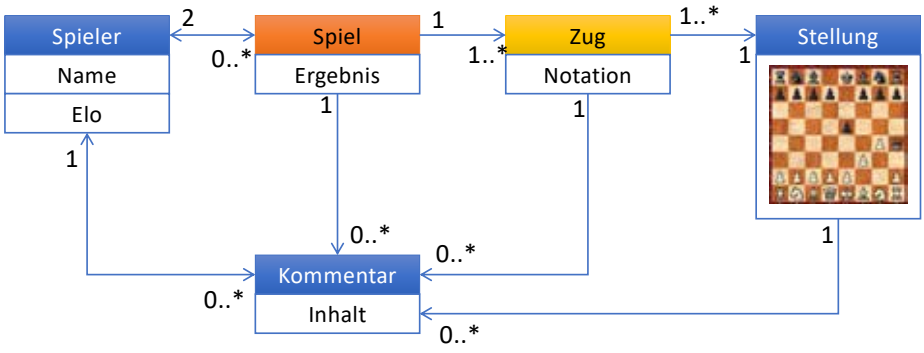
NoSQL

Schemaless

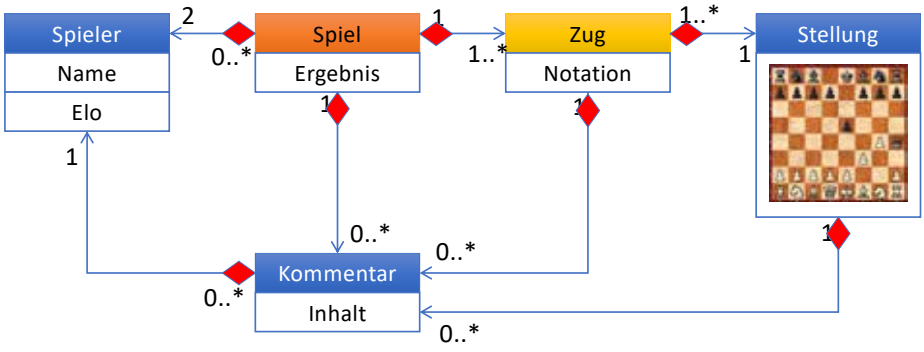
- Unbekanntes Terrain
- „Moving Target“
- Unstrukturierte /Semi-strukturierte Daten
- Manchmal Strukturiert -> Semistrukturiert

- Höhere Performanz durch Datenaggregation

Schema



Aggregation



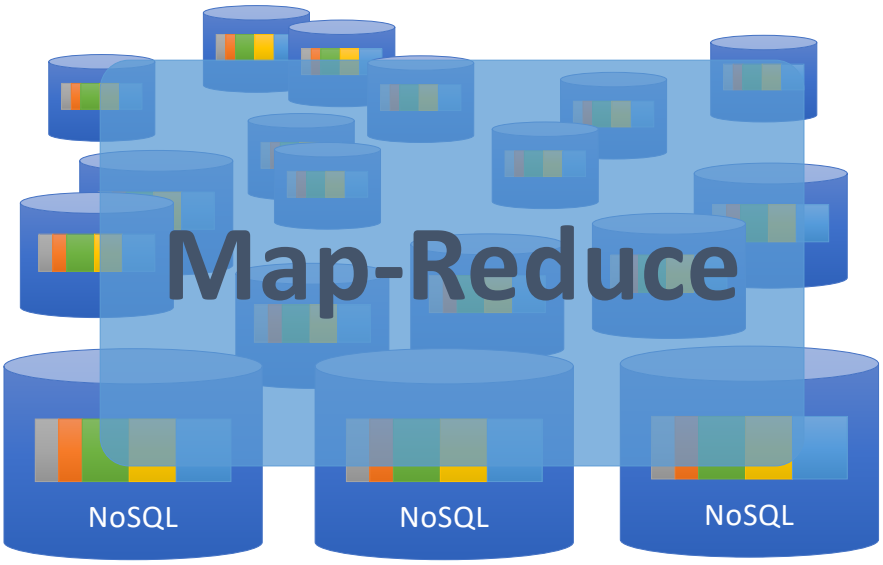
Beispiel JSON

```
{
    "white": { "name": "Sturm", "elo": 42 },
    "black": { "name": "Carlsen", "elo": 2868 },
    "zuege": [
        { "zug": "f3",
          "stellung":
            { "fen": "rnbqkbnr/pppppppp/8/8/8/5P2/PPPPP1PP/RNBQKBNR b KQkq - 0 1" },
          "kommentar":
            { "inhalt": "Nicht gerade der sinnvollste erste Zug!",
              "kommentator": { "name": "Carlsen", "elo": 2868 }
            }
        }
        { "zug": "e5",
          "stellung":
            { "fen": "rnbqkbnr/pppp1ppp/8/4p3/8/5P2/PPPPP1PP/RNBQKBNR w KQkq - 0 2" }
        }
        { "zug": "g4",
          "stellung":
            { "fen": "rnbqkbnr/pppp1ppp/8/4p3/6P1/5P2/PPPPP2P/RNBQKBNR b KQkq - 0 2" }
        }
        { "zug": "Qh4#",
          "stellung":
            { "fen": "rnb1kbnr/pppp1ppp/8/4p3/6Pq/5P2/PPPPP2P/RNBQKBNR w KQkq - 1 3" },
          "kommentar":
            { "inhalt": "Strafe muss sein!",
              "kommentator": { "name": "Stegmann", "elo": 2912 }
            }
        }
    ]
}
```

Bemerkungen

- Aggregate in sich abgeschlossen
- Groß
- Datenredundanzen
- Vielfältige Aggregationsmöglichkeiten
- Nicht wirklich Schemalos
  - Implizit im Programmcode

NoSQL (Aggregates)



Muskeln

statt Hirn

Sauer oder seifig?

SQL

Atomicity

Consistency

Isolation

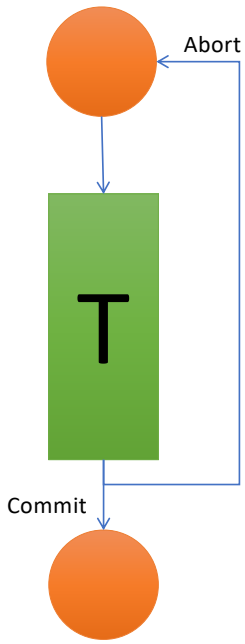
Durability

- Mehrere Operationen als Einheit auffassen
  - BOT
    ...
    ...
    EOT
- Datenkonsistenz

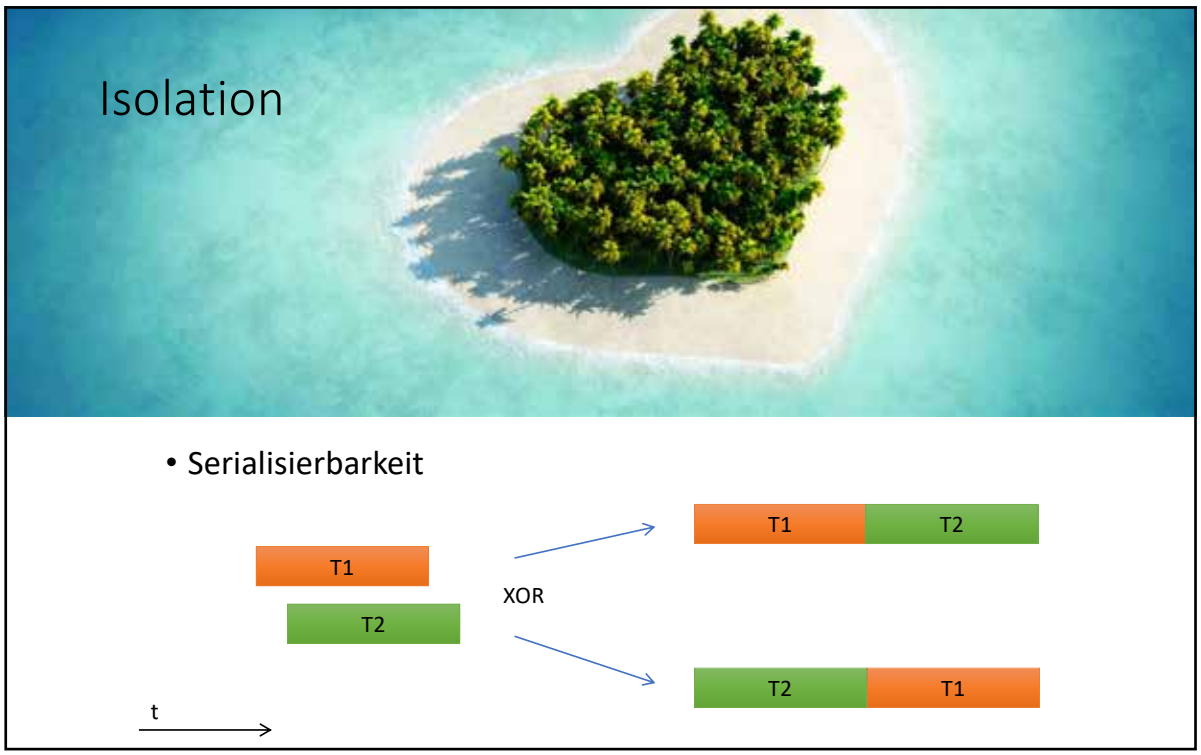
Atomicity

- Unteilbarkeit
- „Alles oder Nichts“
- EOT: Abort
- EOT: Commit

Consistency



- Anwendungsdefiniert
  - BOT ... EOT
- Datenkonsistenz


Isolation
• Serialisierbarkeit
T1
T2
XOR
T1
T2
T2
T1
t


Durability

NoSQL

B asically

A available

S oft state

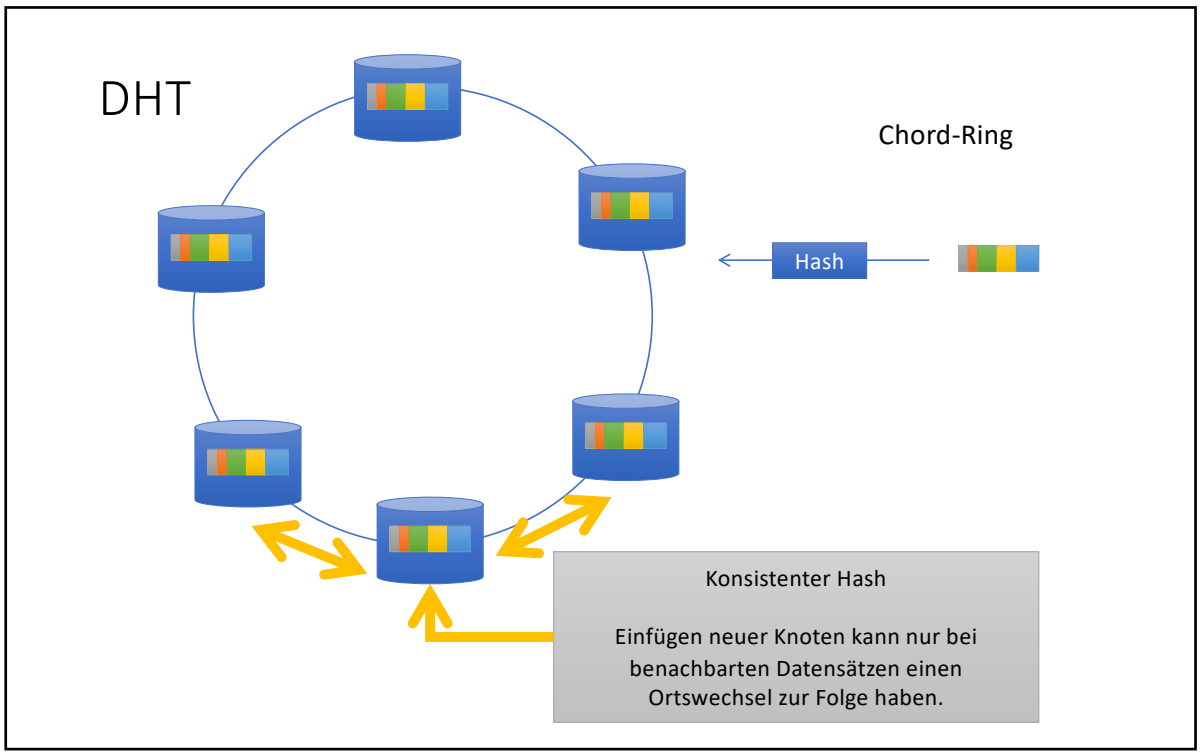
E eventual consistency

BASE

- Verfügbarkeit
- Wiederherstellbarer Zustand
  - Näherungen
  - Teilsichten
  - Latenztoleranz
  - Schwächt D in ACID ab
- Schwache Konsistenz (Eventual Consistency)
  - Kopien „letztendlich“ identisch

Skalierungsvarianten

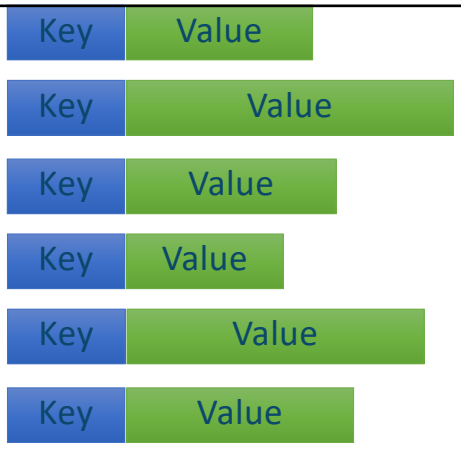
- Sharding (= Partitionierung)
  - Verteilen der Daten über alle Knoten
- Replikation
  - Master (RW) / Worker (R)
  - Peer-2-Peer (RW)
- Schnelles Finden
  - Distributed Hash Table (DHT)


DHT
Chord-Ring
Hash
Konsistenter Hash
Einfügen neuer Knoten kann nur bei benachbarten Datensätzen einen Ortswechsel zur Folge haben.


Exkurs: Small Worlds (Skalenfreie Netze)
132

Datenmodelle

- Schema
  - Relationen
  - Objekt-relational (UML)
- Schemaless
  - Key-Value
  - Document
  - Column-Family
  - Graph



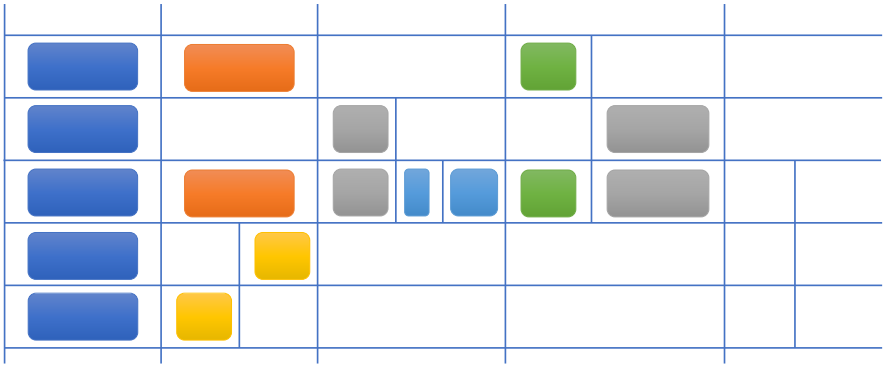
- Größte Gruppe
- „Value opaque“
- Beispiele
  - Cassandra, Dynamo(DB), Voldemort, Redis (In-Memory), MongoDB, FoundationDB (ACID), InfinityDB, ...

Key

Document

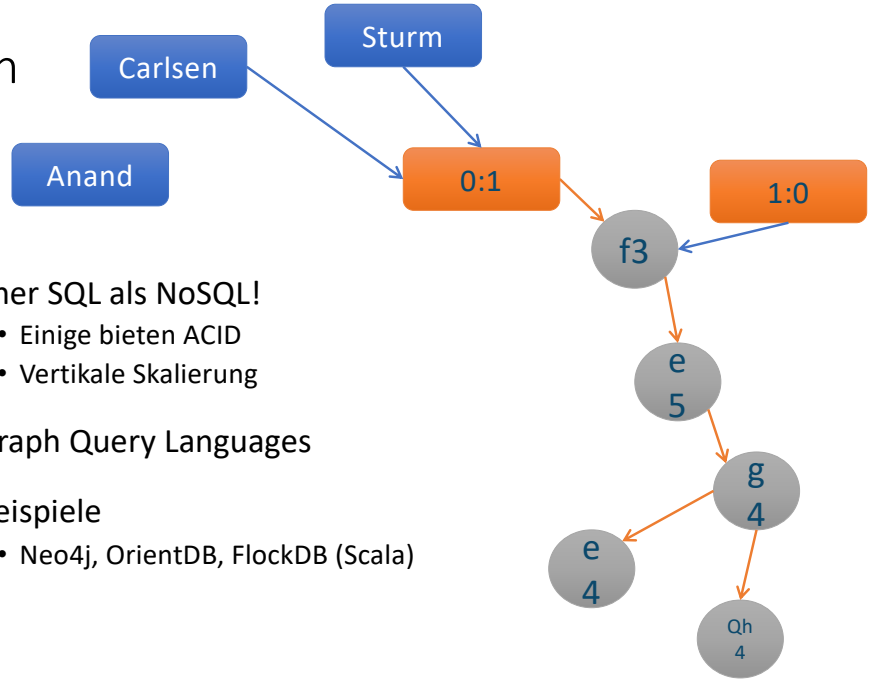
- Value zu einem gewissen Grad strukturiert
- Beispiele
  - CouchDB (JSON), MongoDB (BSON), OrientDB, SimpleDB

Column-Family



- Beispiele
  - BigTable, Accumulo, HBase, ...

Graph



- Eher SQL als NoSQL!
  - Einige bieten ACID
  - Vertikale Skalierung
- Graph Query Languages
- Beispiele
  - Neo4j, OrientDB, FlockDB (Scala)

Polyglot Persistence

- Wechselseitige Synergieeffekte
- Innovative Ansätze finden Einzug in RDBMS
- ACID (in Teilen) und SQL für NewSQL
- Nutzung mehrerer DBMS
  - Application DB
  - Service-Orientierung


Relationale Datenbank
NoSQL Datenbank


NewSQL Datenbank
NewSQL Datenbank
NewSQL Datenbank
NewSQL Datenbank

Granularität

- Eigene Server und/oder Datacenter
- Ein- und ausgehende Daten (Data Ingress and Egress)
  - Räumliche Verteilung
- Fremde Server und/oder Datacenter
  - Cloud-Lösungen

142

* as a service (aaS) - Horizontal

- Infrastructure as a service (IaaS)
  - Wir verwalten (virtuelle) Maschinen etc. in der Cloud
- Platform as a service (PaaS)
  - Wir organisieren Anwendungen und Daten
- Function as a service (FaaS), Serverless
  - Wir implementieren Funktionen und Services
- Software as a Service (SaaS)
  - Fullstack Provider-seitig, wir verwenden die Anwendungen

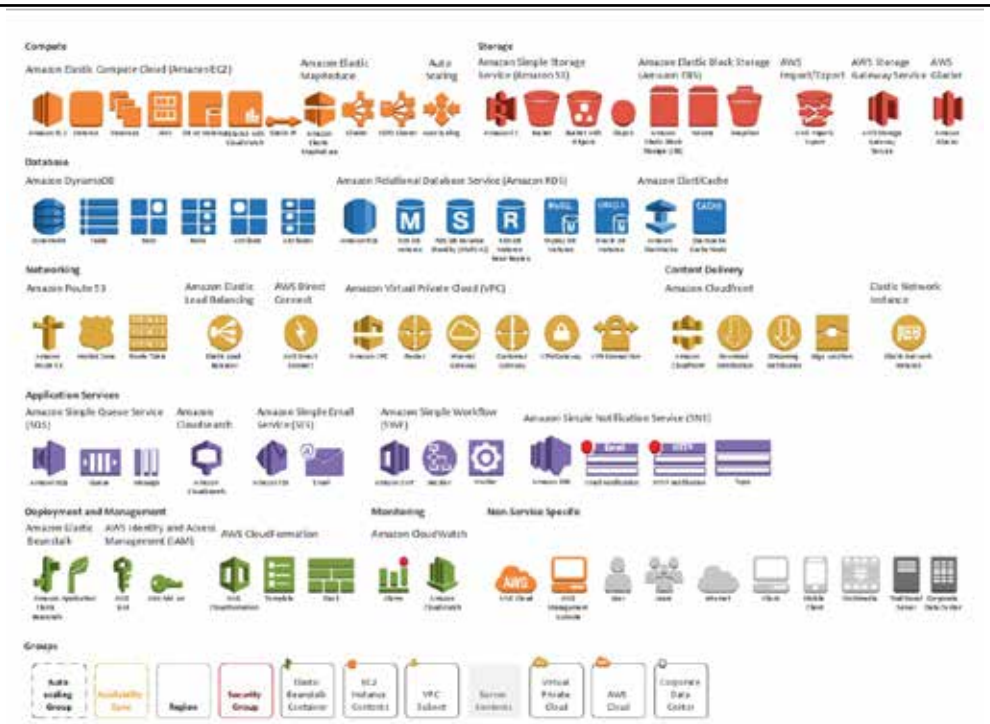
143

* as a service (aaS) – Vertikal (Tiers)

- Storage as a service (STaaS)
- Database as a service (DBaaS)
- Container as a service (CaaS)
- Backend as a service (BaaS)
- Networking as a service (NaaS)

144

Beispiel Amazon AWS



145

Microsoft Azure

146

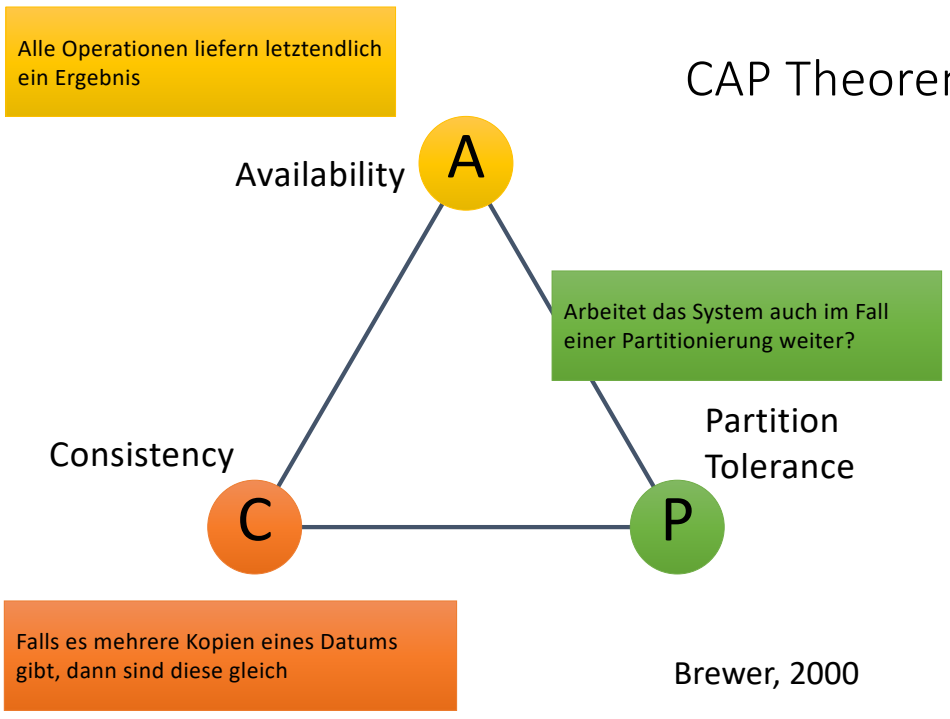
Google Cloud Platform (GCP)

Profane Aspekte

- Latenz
- Administration und Wartung
- Dynamik
  - Churn-In und Churn-Out bzgl. Server

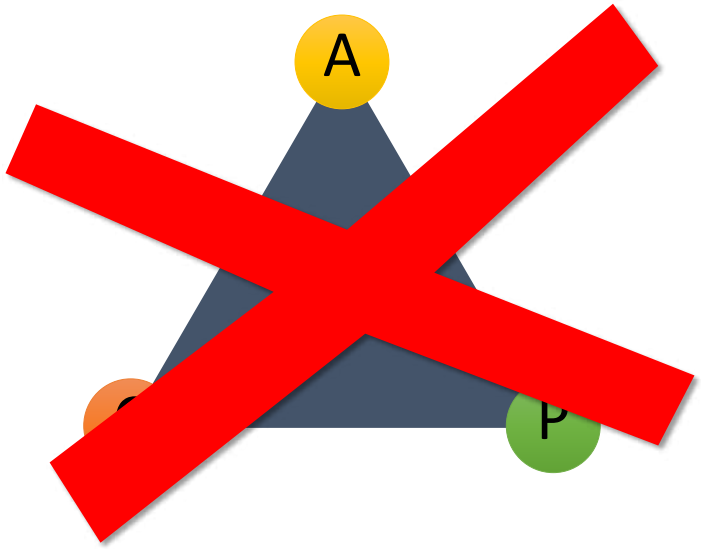
149

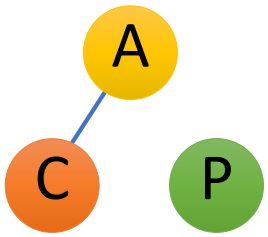
CAP Theorem



Brewer, 2000

There is no free lunch!

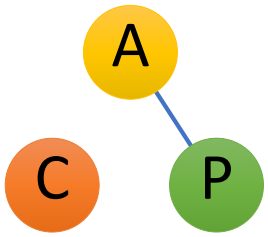




CA

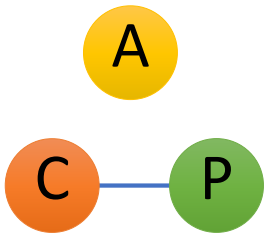
- Vertikale Skalierung
- Keine Antwort bei erkannter Partitionierung
- Beispiele
  - Klassische Datenbanksysteme

AP



- Hauptsache Verfügbar
- Hochgradig replizierte Daten
- Inkonsistenzen tolerieren (Eventually)
- Beispiele
  - DNS, NoSQL, ...

CP



- Hauptsache konsistent
  - Auf alle Knoten warten
- Beispiele
  - Geldautomat, ...