Vorlesung

Webbasierte Informationssysteme

(CS4130)

Einstieg in Cloud Computing und Hadoop

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe

Chronologische Übersicht über die Themen

Cloud Computing – Eigenschaften (1/2)

- Cloud Dienste
  - Computing-Ressourcen werden den Kunden als Dienste zur Verfügung gestellt
- Ressourcenzugriff
  - via Web Services
- Skalierbare, on-demand Services
  - Ermöglicht automatisches und bedarfsorientertes Mieten/Bereitstellen von Ressourcen
  - Illusion “unendlicher” verfügbarer Ressourcen

Cloud Computing – Eigenschaften (2/2)

- Virtualisierung der Ressourcen
  - bietet eine logische anstelle einer physischen Sicht auf Ressourcen
  - Nutzer hat im Allgemeinen kein Wissen über exakten Ort der genutzten Ressourcen
- Elastizität
  - Dynamische Anpassung der Ressourcen nach Bedarf („Hinzuschalten“ weiterer Rechner)
  - Lösung zum effizienten Einsatzes von Ressourcen
- Geschäftsmodell
  - folgt Pay-by-use mit einer nutzungsabhängigen Abrechnung der Cloud-Dienste

Cloud Computing - Dienstmodelle

Grad der Abstraktion

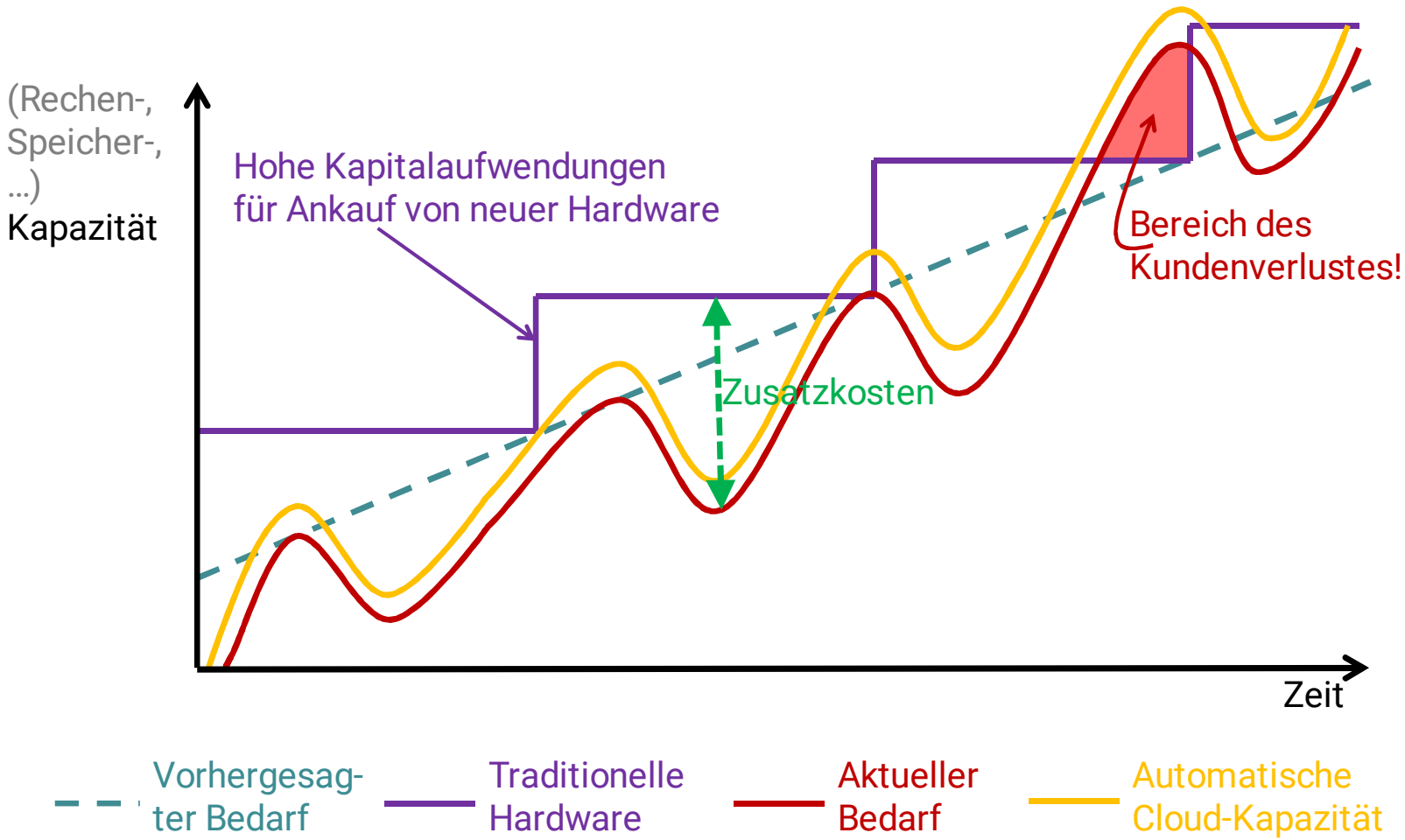
- Software as a Service (SaaS)
  - Bereitstellung von (Web)-Applikationen zur sofortigen Nutzung
  - Standardisierte Software (z.B. Office-Produkte)
  - Beispiele: Google Apps (Dokumentenerstellung (Docs), Tabellenkalkulation (Sheets), Email,...)
- Platform as a Service (PaaS)
  - Framework zur Entwicklung und Ausführung von Applikationen:
    - Programmierumgebung
    - Ausführumgebung
  - Beispiele: Google MapReduce, Google App Engine
- Infrastructure as a Service (IaaS)
  - Mieten von Rechenkapazitäten (CPU), Speicher, Netzwerkressourcen
  - Beispiele: Amazon Elastic Compute Cloud, Amazon Simple Storage Service

Cloud Computing - Vorteile

- Kostenersparnis
  - Kunden müssen keine oder weniger Hardware betreiben
  - zahlen nur für verwendete Ressourcen
- Einheitlicher Ressourcenzugriff
  - durch Web Services für heterogene Clients
- Skalierbarkeit und Elastizität
  - Kunden können jederzeit zusätzliche Ressourcen mieten und wieder freigeben

  ➡ elastische und skalierbare Dienste
- Geringe Einarbeitungszeit
  - Wegfall von Installation und Administration
  - Befreien von der Wahl der Betriebssysteme, der Anwendungen und der Firewallregeln

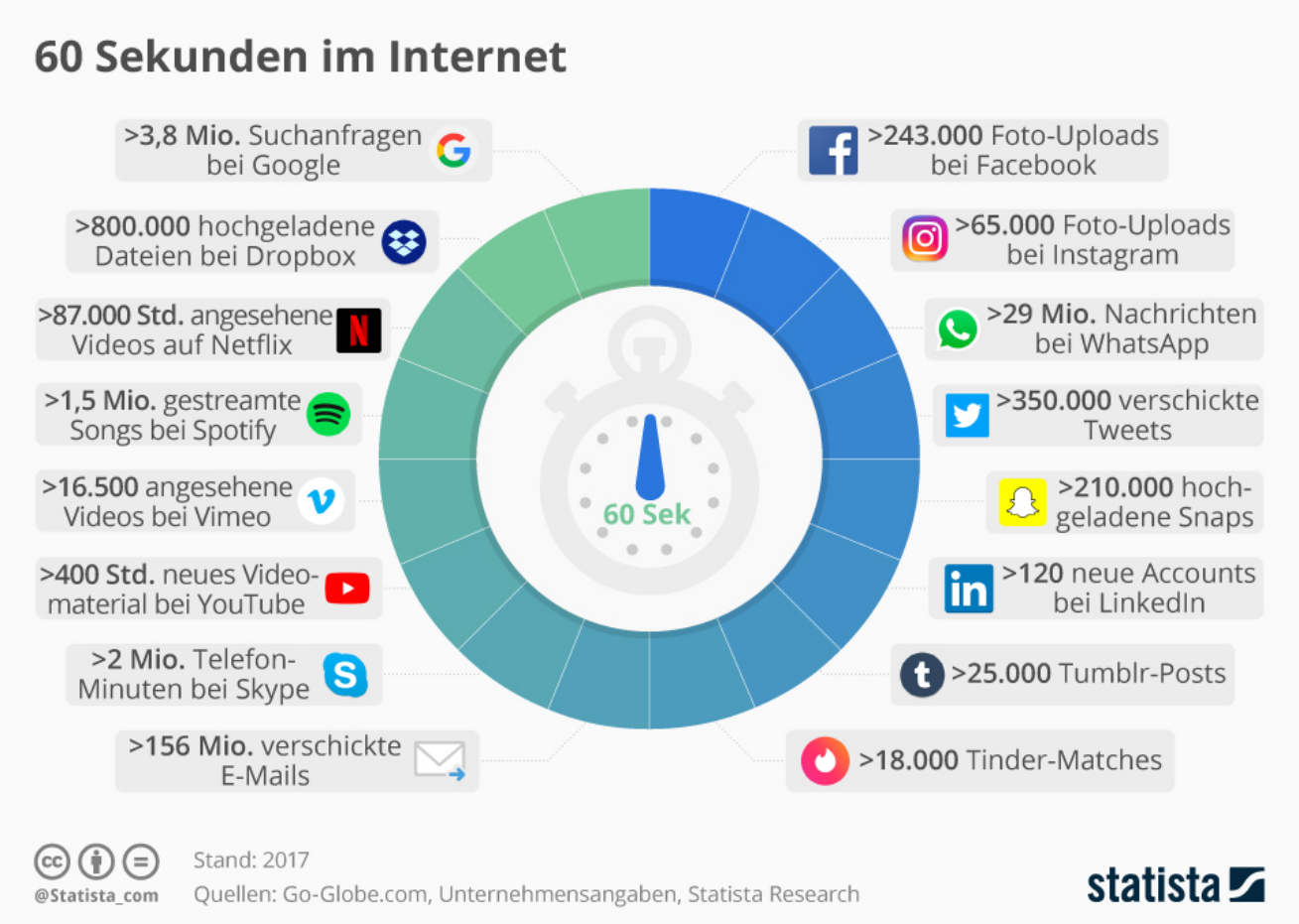
Kapazitäts-Kosten-Performance



Big Data - Beispiele 1/2

- Facebook (2012)
  - $>$ 500 Terabytes/Tag neue Daten:
    2,5 Milliarden Inhalte, 2,7 Milliarden Likes und 300 Millionen Fotos
  - Mehr als 100 Petabytes in einem einzelnen Hadoop Cluster
  - Quelle: Link
- Ebay (2014)
  - 50 Terabytes/Tag neue Daten
  - Insgesamt 100 Petabytes
  - Quelle: Link
- Google (2012)
  - Täglich indexierte Seiten: 20 Milliarden
  - Feb. 2012 in den USA: 11,7 Milliarden Suchanfragen
    - 2,3 Milliarden Suchanfragen direkt durch Knowledge Graph beantwortet
  - Quelle: Link

Big Data - Beispiele 2/2

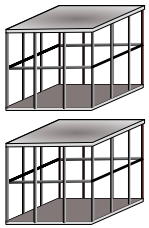


Cloud Frameworks für Big Data

- Viele aufeinander aufbauende und untereinander agierende Frameworks
  - Namen oder Logos meist Tiere oder aus dem Zoo-Kontext
  - Unten sind einige wichtige (aber nicht alle) zu finden

Willkommen im **Hadoop**-Zoo!

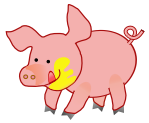
Ich bin **Hadoop**, der starke Elephant für die Verarbeitung!



Wir sind die **HDFS**-Käfige für Big Data...

HBase - die smarte Cloud-Datenbank.

Ich bin der Zoo-Wärter (**Zoo-Keeper**) und organisiere das Zusammenspiel der Tiere



Ich bin **Pig**. Spiele mit mir und die HDFS-Käfige! Ich spreche die relationale Algebra!

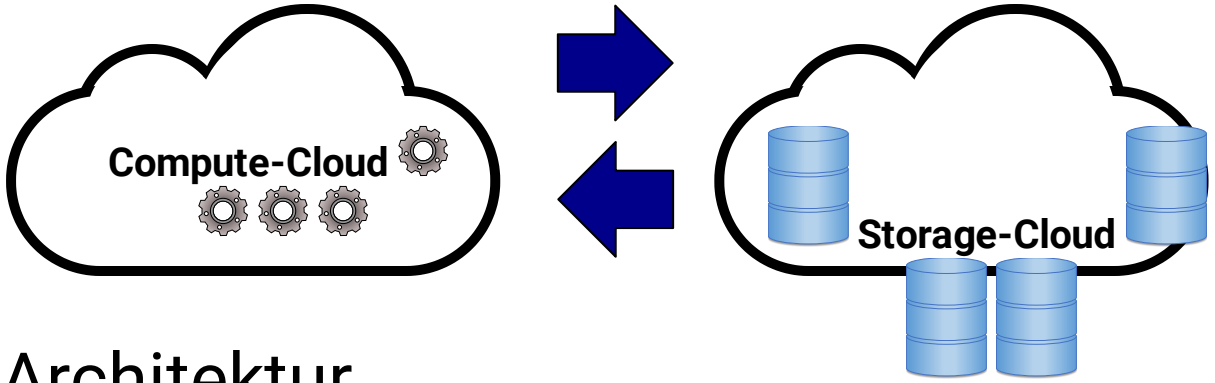
Ich bin **Hive**. Greife auf die HDFS-Käfige zu und spreche SQL mit mir!

Ich bin Apache **Spark** und bin nahezu "real-time" durch iterative Stromverarbeitung.

Ich bin Apache **Flink** und kann Ströme nativ iterativ verarbeiten.

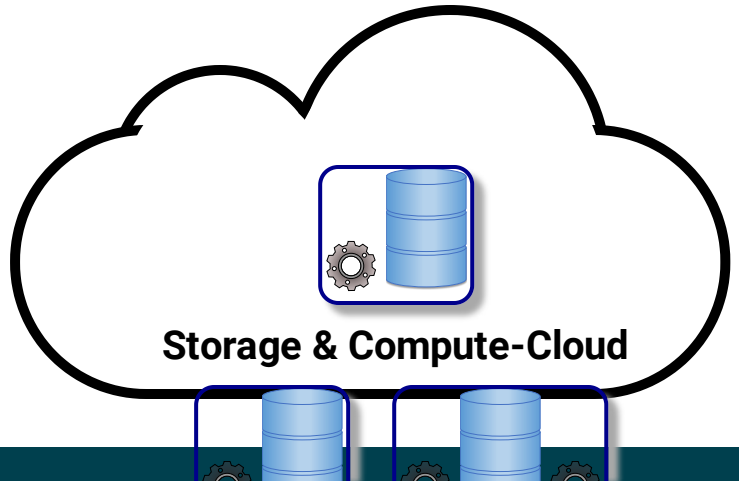
Big Data - Architekturunterschiede

1. Architektur



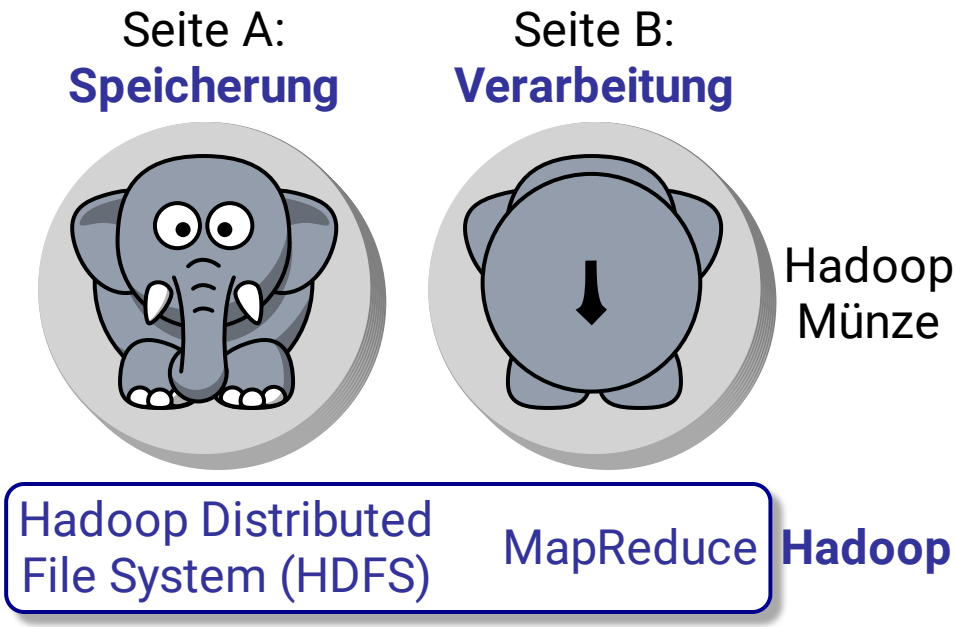
Zu langsam: viele Kopier-operationen!

2. Architektur



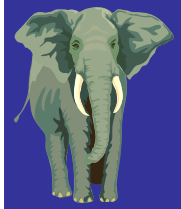
- Daten für Verarbeitung schneller zugreifbar
- Berechnungen können Datenlokalität nutzen

Hadoop

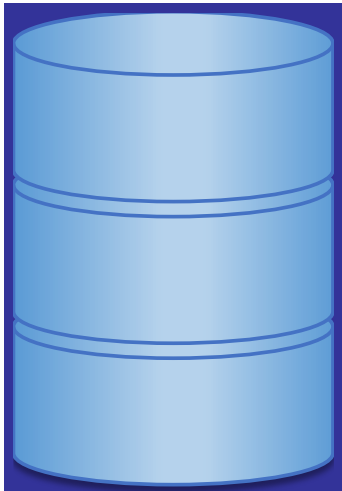


Hadoop ist ein Software Framework für das verteilte Verarbeiten von großen Datenmengen auf großen Clustern von Computern

- Open Source
- Basiert auf den einfachen Programmiermodell MapReduce



Hadoop

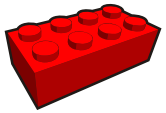


Traditionelle Datenbank

Skalierbarkeit
(insbesondere bei Updates)
- Petabytes an Daten
- Tausende von Rechnern

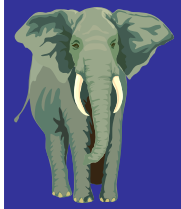
Hohe Performanz
- bei (nur lesenden) Anfragen

Relativ langsam bei Updates

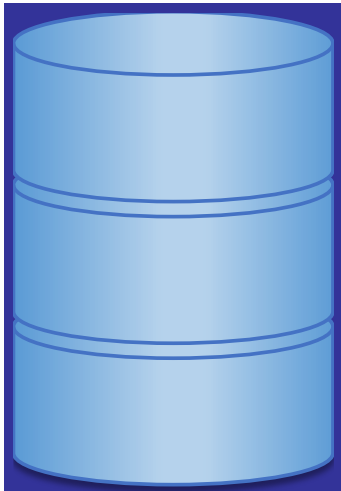


Flexibilität
- Verarbeitung jedweden Datenformates
- schemalos

Einheitliches Datenformat
- Trennung von Schema und Inhalt

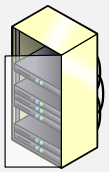


Hadoop



Traditionelle Datenbank

(Relativ) preiswerte (Massenartikel-) Hardware



Wenige High-End Server

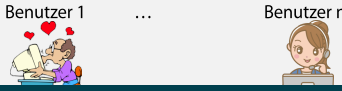
- Wenige Hardwareausfälle

Effiziente und simple fehlertolerante Mechanismen

- Umgang mit häufigen Fehlern (Hardware/Kommunikation)

Transaktionen: Garantie der AtomicityConsistentIsolationDurable Eigenschaften

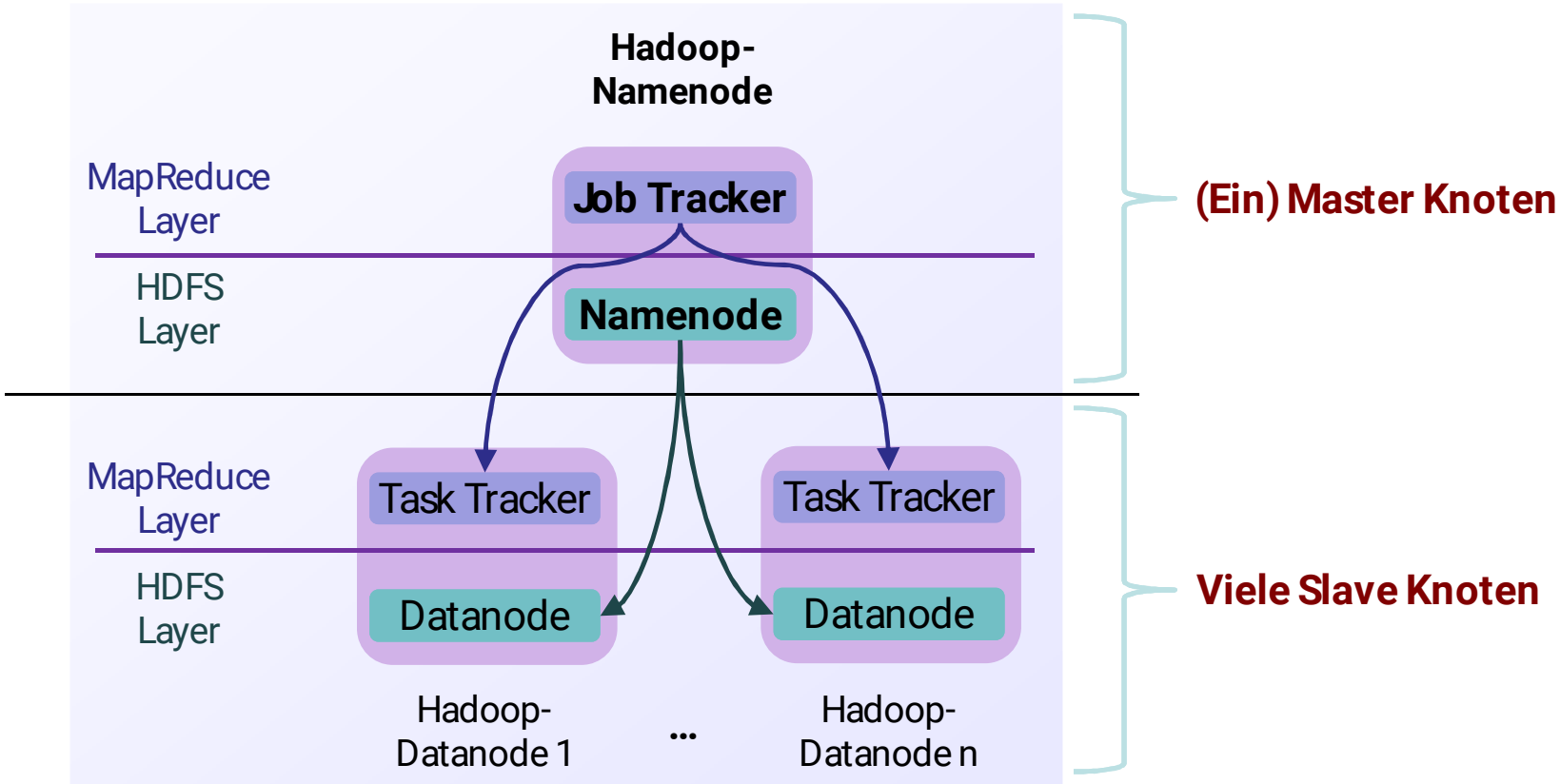
- Annahme: Fehlerfall ist selten



Design Prinzipien von Hadoop

- Automatische Parallelisierung und Verteilung
  - Für den Benutzer „unsichtbar“ im Hintergrund
- Fehlertolerant und automatische Wiederherstellung
  - Automatische Wiederherstellung fehlerhafter Knoten und automatische Wiederholung missglückter Jobs
- Saubere und einfache Programmabstraktion
  - Benutzer stellen nur 2 Pogrammfunktionen (**Map** und **Reduce**) zur Verfügung

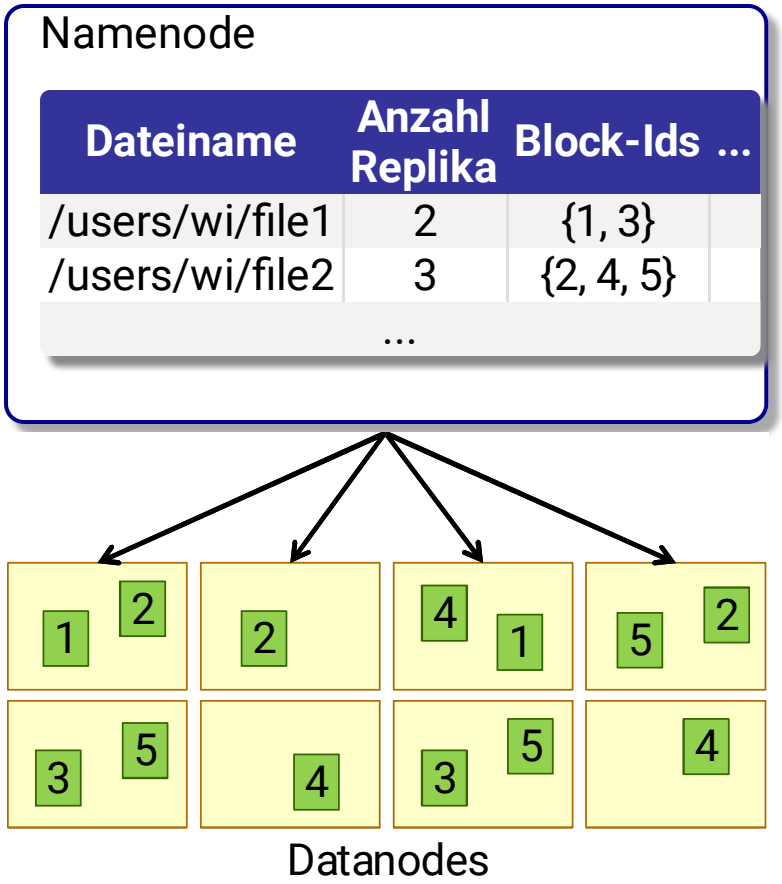
Master-Slave Shared-Nothing Architektur von Hadoop



Haupteigenschaften von Hadoop Distributed File System (HDFS)

- Große Cluster
  - HDFS-Instanz mit bis zu Tausenden von Server-Maschinen
  - jede Server-Maschine speichert einen Teil des Dateisystems
- Replikation
  - Jeder Datenblock ist viele Mal repliziert (Default 3-mal)
- Fehler
  - Fehler sind bei solchen Größenordnungen die Regel und *nicht* die Ausnahme
- Fehlertoleranz
  - Architektonische Ziele von HDFS: Fehlererkennung und schnelle und einfache Wiederherstellung
    - Namenode überprüft ständig Datanodes (Heartbeat)

Hadoop Distributed File System (HDFS)



Zentralisierter Namenode

- verwaltet Metadaten über die Dateien

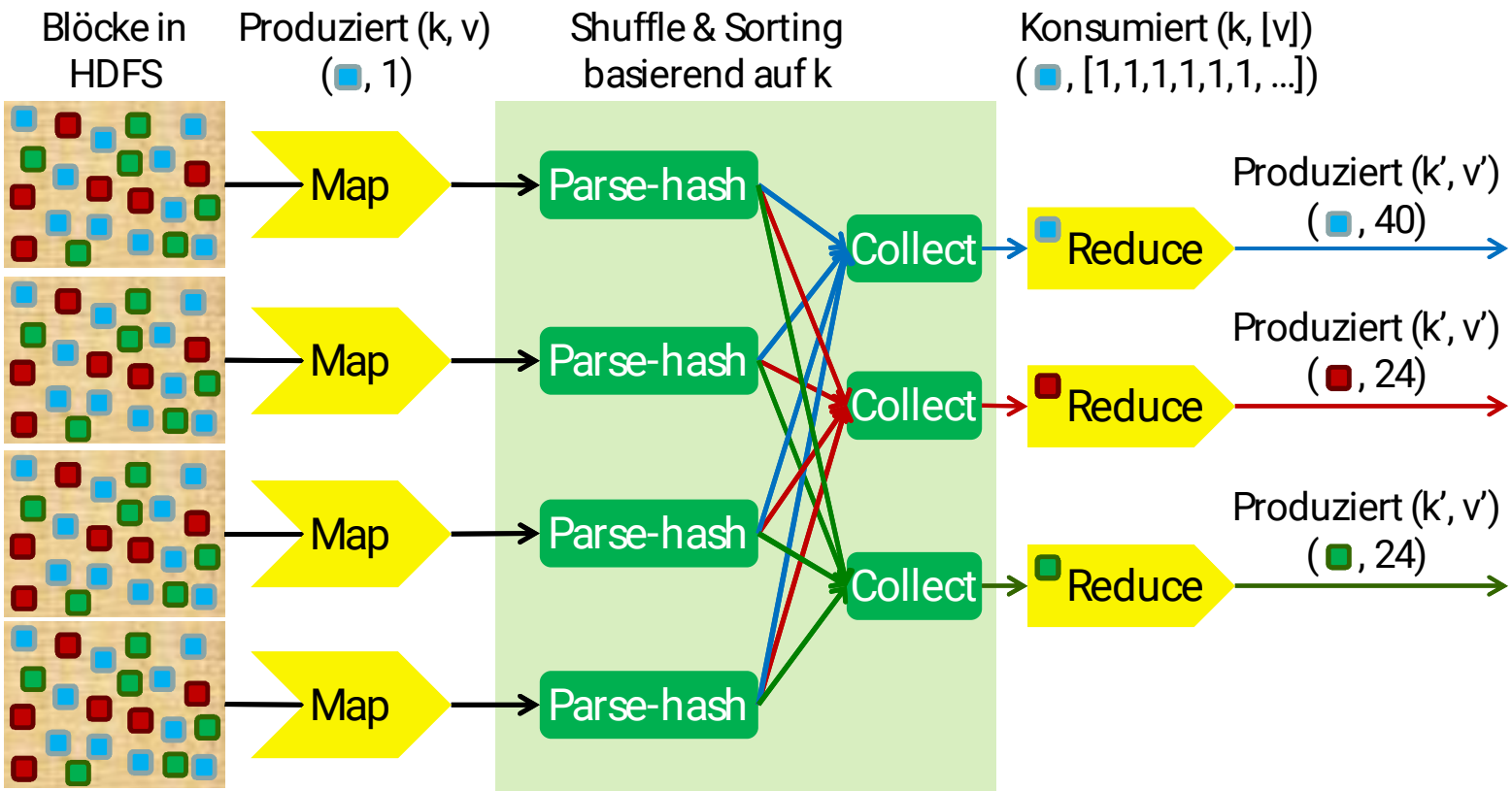
Datei F | 1 | 2 | 3 | 4 | 5 |

Blöcke (zu je 64 MB)

Viele Datanodes (Tausende)

- speichern die Daten
- Dateien sind in Blöcke aufgeteilt
- Jeder Block ist n-mal repliziert (Default $n = 3$)

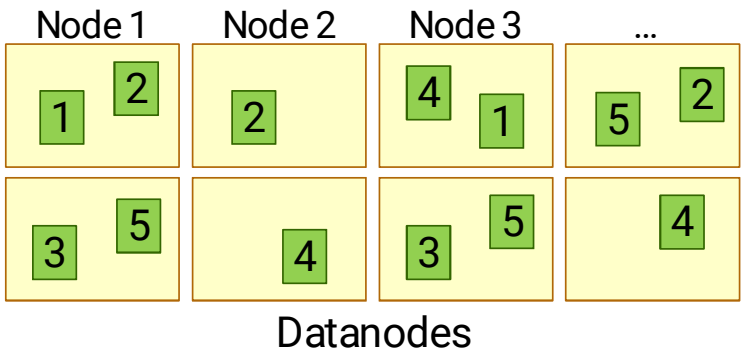
Ausführungsengine von MapReduce (Beispiel: Zählen von Farben)



- Benutzer stellt nur Map und Reduce Funktionen zur Verfügung

Eigenschaften der MapReduce Engine

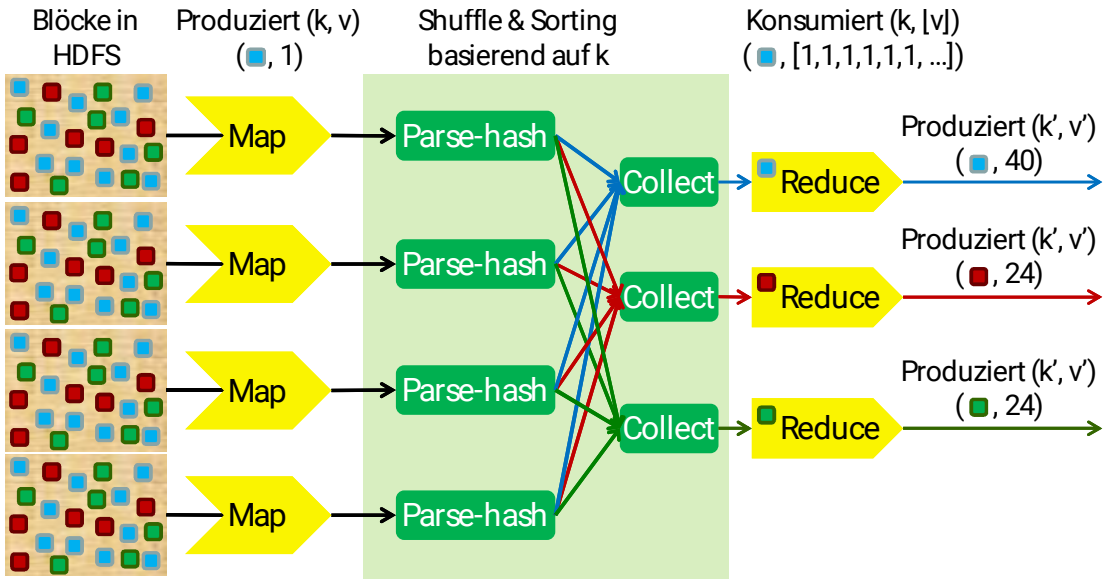
- Masterknoten ist Job Tracker
  - Empfängt den Job vom Benutzer
  - Entscheidet
    - wie viele Tasks laufen werden (Anzahl Mappers/Reducers)
    - wo die Mapper/Reducer laufen werden



- Zu lesende Datei hat 5 Blöcke
  ➡ lasse 5 Map Tasks laufen
- Wo soll der Task laufen, der Block 1 liest?
  - Versuche ihn auf Knoten 1 oder alternativ (z.B. im Fehlerfall) auf Knoten 3 laufen zu lassen

Eigenschaften der MapReduce Engine

- Slave Knoten (auf jeden Datenknoten) ist Task Tracker
  - Empfängt Task vom Job Tracker (auf dem Masterknoten)
  - Führt den (Map- oder Reduce-) Task (komplett) aus
  - Kommuniziert ständig mit dem Job Tracker über den Fortschritt



1 MapReduce Job besteht in diesem Beispiel aus

- 4 Map Tasks und
- 3 Reduce Tasks

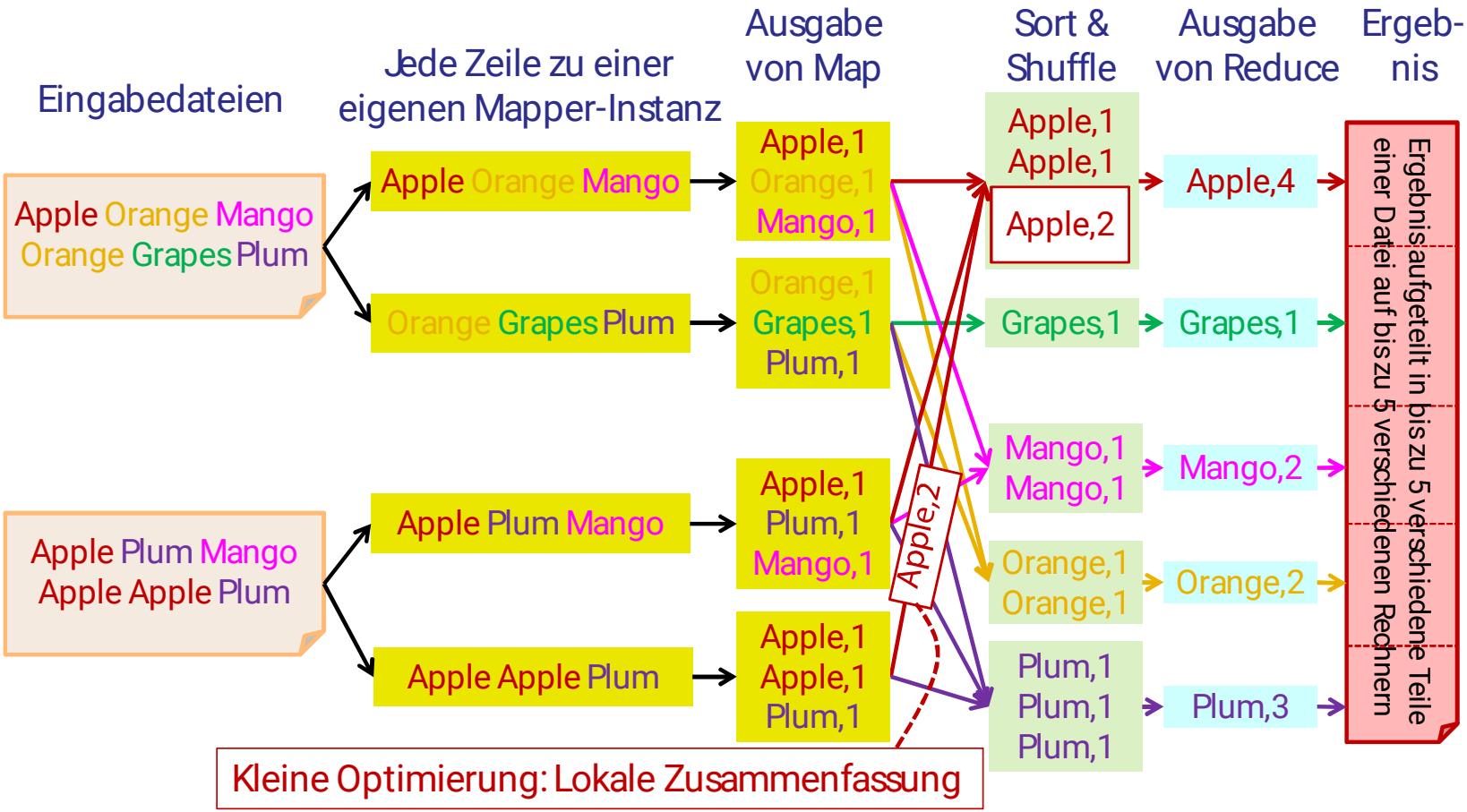
Schlüssel-Wert-Paare

- **Code des Benutzers** für Mapper und Reducer
  - **Grundlegende Schnittstelle** der empfangenen und versendeten Daten der Mapper und Reducer: **Schlüssel-Wert-Paar**
  - *In der Verantwortung des Benutzers*, was Schlüssel und was Wert ist

Funktion	Eingabe	Ausgabe	Bemerkung
Map	(K, V)	Sequenz von (K', V')	Benutzerdefinierte Funktion
Shuffle & Sorting	Sequenz von (K', V')	$(K'$, Iterator über $V')$ für jedes vorkommende K'	Interne Funktion, Bindeglied zwischen Map und Reduce
Reduce	$(K'$, Iterator über $V')$	Sequenz von (K'', V'')	Benutzerdefinierte Funktion

K, K', K'': Datentypen der Schlüssel V, V', V'': Datentypen der Werte

Bsp.: WordCount (Zählen von Wörtern)



Java-Code für WordCount

```
import …

public class WordCount {
  public static class TokenizerMapper { … }
  public static class IntSumReducer { … }

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

⬅ Lokale Zusammenfassung

Map für WordCount

```
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context)
                                          throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

Reducer für WordCount

```
public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {

   private IntWritable result = new IntWritable();

   public void reduce(Text key, Iterable<IntWritable> values, Context context)
                                                throws IOException, InterruptedException {
     int sum = 0;
     for (IntWritable val : values) {
       sum += val.get();
     }
     result.set(sum);
     context.write(key, result);
   }
 }
```

Map und Reduce ist optional/ Hintereinanderschaltung von Jobs

- Map- oder Reduce-Phase kann fehlen
  - Beispiel: Filter nach dem Wort „Orange“ mit nur einer Map-Phase
- Es kann beliebig viele Map- und Reduce-Phasen (in mehreren Jobs) geben
  - Beispiel: Verarbeitung einer komplexen Datenbankanfrage
    - Mit Selektion, mehreren Joins, Projektion, …

      ➡ jede Operation kann durch eine Map-, eine Reduce- oder eine Map- und Reduce-Phase durchgeführt werden

SPARQL-Anfrage in der Cloud

- Welche Map- und Reduce-Phasen sind notwendig zum Verarbeiten folgender SPARQL-Anfrage?

```
PREFIX …
SELECT ?person ?name
WHERE {
  ?article rdf:type bench:Article .
  ?article dc:creator ?person .
  ?person  foaf:name  ?name .
}
ORDER BY ?name
```

Annahme:
Durch geschickte Verwendung von Indexen können die Ergebnisse der Tripelmuster mit einem Indexzugriff erfolgen.

HBase

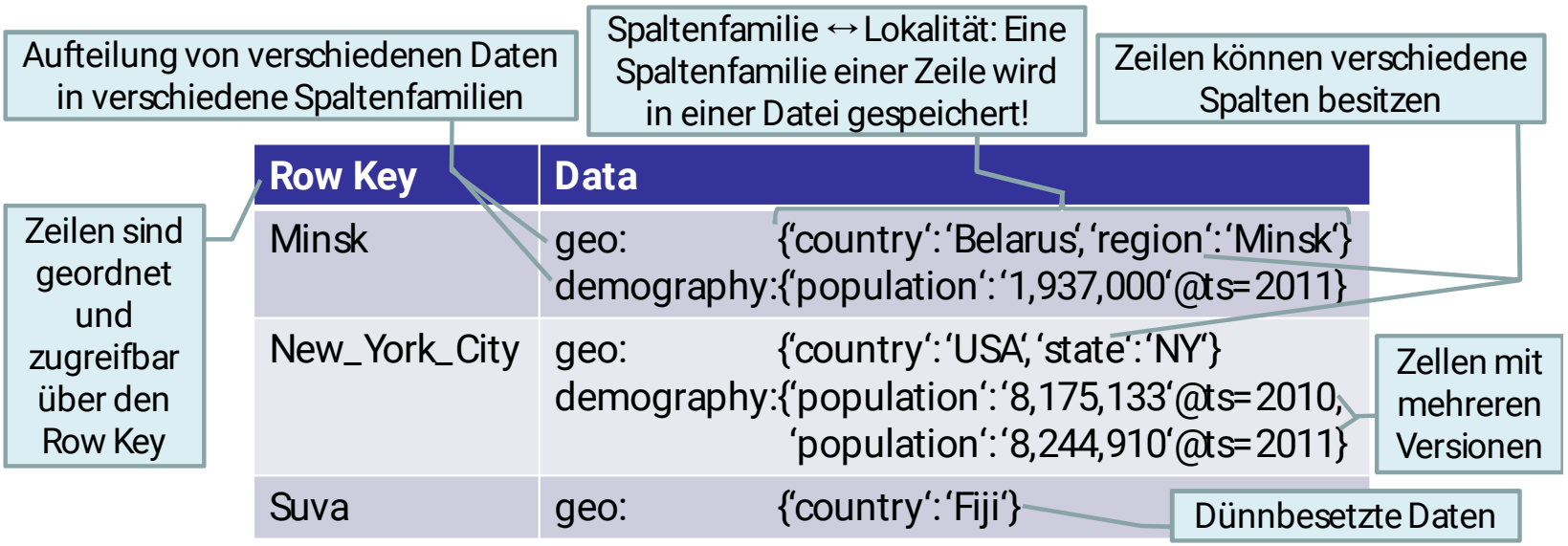
- Open Source nicht-relationale, verteilte, spaltenorientierte Datenbank
  - Lineare Skalierbarkeit durch interne Nutzung von HDFS
    - Verwaltung von Hunderten von Terabytes an Daten
    - Automatisches und konfigurierbares Aufteilen von Tabellen
    - Bulk Import von großen Datenmengen
  - Autom. Ausfallsicherung/hohe Verfügbarkeit durch Replikation
  - Streng konsistente Lese- und Schreiboperationen einer Zeile
  - Volle Integration mit Hadoop MapReduce (als Quelle & als Ziel)
  - **Keine** Unterstützung von: Transaktionen, Joins, SQL (indirekt durch Hive, …), Sekundärindexierung, relationales Datenmodell

Einsatzszenarien von HBase

- Applikationen mit vielen Schreibzugriffen
  - Hinzufügen und Überschreiben von Daten
  - Weniger geeignet für *Lese-Modifiziere-Schreibe*
- Tracking von Benutzeraktivitäten
  - Typische Anfragen:
    - Was waren die letzten 10 Aktionen eines Besuchers?
    - Welche Benutzer haben sich gestern eingeloggt?
- Temporale Datenbank-Applikationen
- Monitoring
- Nachrichten-Systeme (z.B. Twitter, Facebook-Nachrichten)
- ...

Speicherformat von HBase

- Zeile besteht aus Schlüssel-Wert-Zellen:
  (row key, column family, column, timestamp) → value
  - Row Keys: Nicht interpretierte Bytearrays
  - Column Families: Statische Gruppierung der Spalten
  - Zellen: nicht interpretierte Bytearrays mit Zeitstempel
    (Default: Speicherung der letzten 3 Versionen)



Row Key	Data
Minsk	geo: {'country':'Belarus','region':'Minsk'} demography:{'population':'1,937,000'@ts=2011}
New_York_City	geo: {'country':'USA','state':'NY'} demography:{'population':'8,175,133'@ts=2010, 'population':'8,244,910'@ts=2011}
Suva	geo: {'country':'Fiji'}

Hive

- Entwicklung von Facebook, nun Open Source
  - zur Vereinfachung von Hadoop & internen Einsatzes
- Hive Query Language (HQL) als Variante von SQL
  - Ad-Hoc Anfragen ⇝ Analyse von Big Data
  - Einbindung von HBase-Tabellen als externe Tabellen möglich
  - Standardmäßige Speicherung der Tabellen auf HDFS als flache Dateien
- Data Warehouse Infrastruktur basierend auf Hadoop
  - Einfache Datenaggregation
  - Erweiterbar durch benutzerdefinierte MapReduce-Jobs
  - Akzeptable, aber nicht optimale Latenz für interaktives Daten-Browsing, Anfragen über kleine Datensätze oder Testanfragen
    - Latenz von Minuten bei kleinen Datensätzen von ein paar Hundert Megabytes (z.Vgl.: Oracle mit wenigen Millisekunden)
    - **nicht** entwickelt für Online Transaction Processing/Real-Time Anfragen
  - Am besten für Batch-Jobs geeignet über große Mengen an unveränderlichen Daten (wie etwa Web Logs)

Hive: WordCount-Beispiel

```
CREATE TABLE wordcount (token STRING);

LOAD DATA LOCAL INPATH 'wordcount/words'
OVERWRITE INTO TABLE wordcount;

SELECT count(*) FROM wordcount GROUP BY token;
```

- Grundlegende Syntax:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[ORDER BY col_list]
[LIMIT number] ;
```

Zusammenfassung

- Hervorragende Eigenschaft des Cloud Computing: Elastizität
- Hadoop-Framework
  - zum verteilten Speichern von großen Datenmengen (HDFS)
  - zum verteilten Verarbeiten von großen Datenmengen (MapReduce)
  - mit einfachem Programmiermodell mit nur zwei benutzerdefinierten Funktionen Map und Reduce
  - Komplexes, fehlertolerantes, skalierbares und transparentes System auf großen Clustern (Tausende an Servern) für Batch-Verarbeitung
- HBase
  - Effiziente, skalierbare Datenbank nur für einfachste Operationen wie Hinzufügen und Lesen
- Hive
  - Unterstützung eines SQL-Dialektes als Anfragesprache
  - Hohe Latenz: Ideal für Ad-Hoc Anfragen und komplexe Analysen, aber nicht für Real-Time-Anfragen