



# **Big Data & Data Science**

## **Big Data Analytics**

WS 2025/26

**Prof. Dr. Klemens Waldhör**

## Hadoop

- HDFS
- Map Reduce

## Apache Spark

## Docker

## Kubernetes



## Big Data

- Bezieht sich auf die Nutzung großer Datenmengen aus unterschiedlichen Quellen mit hoher Verarbeitungsgeschwindigkeit und unterschiedlicher Datenqualität, um wirtschaftlichen Nutzen zu erzeugen. Der Begriff „Big Data“ bezeichnet dabei sowohl die zugrunde liegenden Daten als auch die Technologien und Methoden zu deren Erfassung, Verarbeitung, Übertragung und Speicherung.
- Zur Identifikation eines Big-Data-Szenarios hat sich die Betrachtung der fünf Dimensionen von Big Data bewährt (Five V's bzw. 5V-Modell).
- Volume – Velocity – Variety – Veracity - Value

## Big Data Analytics

- Bezieht sich auf den komplexen Prozess der Analyse großer und vielfältiger Datenmengen („Big Data“), um Erkenntnisse wie verborgene Muster, unbekannte Zusammenhänge, Markttrends, Kundenpräferenzen und andere geschäftlich relevante Informationen aufzudecken.
- Das Hauptziel von Big-Data-Analytics besteht darin, Organisationen dabei zu unterstützen, fundiertere Geschäftsentscheidungen zu treffen, indem Data Scientists, Modellierer prädiktiver Modelle und andere Analysefachkräfte in die Lage versetzt werden, große Mengen an Transaktionsdaten sowie weitere Datenarten zu analysieren, die von herkömmlichen Business-Intelligence-(BI)-Programmen häufig nicht erschlossen werden.

## Frage: Warum reicht SQL nicht aus?

- Datenbanken wurden doch erfunden, SQL-basierte Datenbanken, um schnell auf Daten zuzugreifen...



## Beispiele für mögliche Geschwindigkeitsunterschiede:

- **Kleine, komplexe Transaktionen**

Ein Online-Shop aktualisiert Lagerbestände  
SQL-Datenbank wahrscheinlich schneller und zuverlässiger.

- **Große Datenmengen analysieren**

Eine Analyse von Social-Media-Daten über mehrere Jahre  
Big Data-Technologie wie Hadoop oder Spark wahrscheinlich effizienter.

## Szenario 1: Bestellverwaltung in einem Online-Shop (SQL-Datenbank)

**Situation:** Ein Online-Shop verwendet eine SQL-Datenbank, um Bestellungen zu verwalten.

Jede Bestellung enthält Informationen wie Kunden-ID, Produkt-ID, Menge, Preis und Datum.

### Typische Aufgaben:

- Einfügen neuer Bestellungen in die Datenbank.
- Aktualisieren des Lagerbestands, wenn eine Bestellung aufgegeben wird.
- Abfragen von Bestelldetails für einen bestimmten Kunden.

### Performance-Aspekte:

- **Schnelligkeit bei Transaktionen:** Das Einfügen und Aktualisieren von Daten erfolgt schnell, da die Datenbank für solche Operationen optimiert ist.
- **Komplexe Abfragen:** Die Datenbank kann komplexe Abfragen effizient verarbeiten, z.B. die Berechnung des Gesamtumsatzes eines Kunden über einen bestimmten Zeitraum.

## Szenario 2: Analyse von Social-Media-Daten (Big Data-Technologie)

**Situation:** Ein Unternehmen analysiert Social-Media-Daten, um Trends und Muster zu erkennen. Die Datenmenge ist riesig und beinhaltet Posts, Likes, Kommentare und Shares über mehrere Jahre.

### Typische Aufgaben:

- Verarbeitung und Speicherung von riesigen Datenmengen.
- Durchführung von Datenanalysen, um Trends oder Muster zu erkennen.
- Erstellung von Berichten über die Aktivität bestimmter Themen oder Hashtags über die Zeit.

### Performance-Aspekte:

- **Verarbeitung großer Datenmengen:** Big Data-Systeme wie Hadoop oder Spark sind dafür ausgelegt, große Datenmengen effizient zu verarbeiten.
- **Skalierbarkeit:** Bei zunehmendem Datenvolumen können diese Systeme leicht skaliert werden, indem man mehr Ressourcen hinzufügt.
- **Flexibilität bei unstrukturierten Daten:** Diese Technologien können mit einer Vielzahl von Datenformaten umgehen, was für die Analyse von Social-Media-Daten wichtig ist.

- Moderne Systeme (BigQuery, Snowflake, Redshift, Spark SQL) sind SQL-basiert
- SQL ist eine Abfragesprache – nicht gleichbedeutend mit einer einzelnen Datenbanktechnologie.

### Nicht wirklich zutreffend

-  *SQL = klein, langsam*
-  *Big Data = schnell, modern*

## Google BigQuery

- Serverloses Cloud Data Warehouse (GCP)
- Automatische Skalierung, spaltenorientiert
- Abrechnung nach gescanntem Datenvolumen
- Ideal für Ad-hoc-Analysen & große Datenmengen

## Amazon Redshift

- AWS-basiertes Data Warehouse
- Cluster-orientiertes MPP-System
- Spaltenorientierte Speicherung
- Geeignet für klassische DWH-Workloads

## Snowflake

- Cloud-agnostisches Data Warehouse (AWS, Azure, GCP)
- Trennung von Storage und Compute
- Sehr gut für BI & parallele Nutzer
- Abrechnung nach Compute-Zeit und Storage

## Spark SQL

- SQL-Modul von Apache Spark
- Verteilte In-Memory-Verarbeitung
- Ideal für ETL, Data Engineering & ML
- Kein klassisches Data Warehouse

## Ausgangsszenario

Unternehmen betreibt Online-Plattform (z. B. Hotel, Ticketing, E-Commerce)

- Datenquellen: Klicklogs, Buchungen, Kundendaten
- Ziel: Analyse von Auslastung, Umsatz, Kundenverhalten

## Schritt 1: Datenaufbereitung mit Spark SQL

Rohdaten aus Logs (CSV/JSON) liegen in Data Lake (S3 / GCS / HDFS)

- Spark SQL bereinigt, filtert und aggregiert Daten
- Beispiel: Sessions → Buchungen → Umsatz pro Tag
- Ergebnis: strukturierte Fakt- und Dimensionstabellen

## Schritt 2a: Analyse mit BigQuery

Geladene aggregierte Daten im BigQuery Warehouse

- Ad-hoc-SQL-Abfragen durch Analysten
- Beispiel: Umsatz nach Region, Saison, Kanal
- Ideal für große Datenmengen & spontane Fragestellungen

## Schritt 2b: Analyse mit Snowflake

Zentrale Datenplattform für Controlling & BI

- Mehrere virtuelle Warehouses für parallele Teams
- Beispiel: Marketing, Management, Data Science
- Hohe Performance bei vielen gleichzeitigen Nutzern

## Schritt 2c: Analyse mit Redshift

Datenhaltung im AWS-Ökosystem

- Integration mit S3, Glue, QuickSight
- Beispiel: Standardreports & Dashboards
- Gut geeignet für klassische DWH-Strukturen

### Spark SQL: Datenverarbeitung & Transformation (ETL)

BigQuery / Snowflake / Redshift: Analyse & Reporting

Trennung von Engineering und Analytics

Skalierbare End-to-End-Datenpipeline

### Offizielle Produkt- & Doku-Seiten

Google BigQuery

- <https://cloud.google.com/bigquery>

Snowflake

- <https://www.snowflake.com>

Amazon Redshift

- <https://aws.amazon.com/redshift>

Apache Spark SQL

- <https://spark.apache.org/sql>

## Praxisbeispiel: Analyse von Nutzungs- und Buchungsdaten

### Ausgangsszenario

Ein mittelständisches Unternehmen betreibt eine Online-Plattform (z. B. Hotelbuchungen, Ticketverkauf oder E-Commerce).

Täglich fallen an:

- Klick- und Eventdaten (Logs)
- Buchungs- bzw. Transaktionsdaten
- Kundendaten

Das Unternehmen möchte:

- Berichte und Dashboards für Management und Controlling
- flexible Ad-hoc-Analysen durch Analysten
- perspektivisch Machine-Learning-Modelle einsetzen

## Praxisbeispiel: Analyse von Nutzungs- und Buchungsdaten

### Aufgabe 1: Systemrecherche

Recherchieren Sie zu den folgenden Systemen:

- Google BigQuery
- Snowflake
- Amazon Redshift
- Apache Spark SQL

Nutzen Sie dafür offizielle Herstellerseiten und Dokumentationen.

Recherchieren Sie mindestens:

- Art des Systems (Data Warehouse, Verarbeitungssystem, ...)
- Typische Einsatzszenarien
- Architektur (z. B. serverlos, Cluster, Trennung von Storage/Compute)
- Skalierungskonzept
- Abrechnungsmodell (grob)
- Rolle von SQL im System

## Praxisbeispiel: Analyse von Nutzungs- und Buchungsdaten

### Aufgabe 2: Vergleichstabelle

Kriterium	BigQuery	Snowflake	Redshift	Spark SQL
Systemtyp				
Cloud / On-Prem				
Skalierung				
SQL-Nutzung				
Typische Use Cases				
Stärken				
Einschränkungen				

## Praxisbeispiel: Analyse von Nutzungs- und Buchungsdaten

### Aufgabe 3: Einsatzentscheidung (Transferaufgabe)

Beantworten Sie die folgenden Fragen mit kurzer Begründung:

Welches System eignet sich am besten für:

- a) spontane Ad-hoc-Analysen großer Datenmengen?
- b) parallele BI-Nutzung durch viele Abteilungen?
- c) komplexe Datenaufbereitung und Transformation?

Warum ist Spark SQL kein klassisches Data Warehouse, obwohl SQL verwendet wird?

Welche Kombination von Systemen würden Sie für das Ausgangsszenario vorschlagen?

## Praxisbeispiel: Analyse von Nutzungs- und Buchungsdaten

### Aufgabe 3: Einsatzentscheidung (Transferaufgabe)

Beantworten Sie die folgenden Fragen mit kurzer Begründung:

Welches System eignet sich am besten für:

- a) spontane Ad-hoc-Analysen großer Datenmengen?
- b) parallele BI-Nutzung durch viele Abteilungen?
- c) komplexe Datenaufbereitung und Transformation?

Warum ist Spark SQL kein klassisches Data Warehouse, obwohl SQL verwendet wird?

Welche Kombination von Systemen würden Sie für das Ausgangsszenario vorschlagen?



# Big Data Analytics Hadoop Kurzzusammenfassung

Apache Hadoop ist ein Open-Source-Framework zur verteilten Speicherung und Verarbeitung sehr großer Datenmengen.

Zentrale Ziele:

- Skalierbarkeit
- Fehlertoleranz
- Verarbeitung großer Datenmengen auf Clustern

- Verteiltes Dateisystem für große Dateien
- Speicherung in großen Blöcken (z. B. 128/256 MB)
- Replikation der Daten für Ausfallsicherheit
- Optimiert für hohen Durchsatz, nicht für niedrige Latenz
- Mehrere Terabyte pro Stunde lesen oder schreiben
- Verarbeitung großer Dateien in Batch-Jobs
- Nicht geeignet für:
  - viele kleine Dateien
  - zufällige, häufige Einzelzugriffe
  - klassische transaktionale Workloads (OLTP)

**HDFS verarbeitet viel auf einmal – aber nicht schnell einzeln.**

## Architektur

- MapReduce ist ein Programmiermodell zur verteilten Batch-Verarbeitung sehr großer Datenmengen.
- Die Verarbeitung erfolgt nahe an den Daten (Data Locality).

## Grundidee

- Zerlege ein Problem in viele Teilaufgaben (Map)
- Fasse Teilergebnisse zusammen (Reduce)
- Verarbeitung erfolgt parallel auf vielen Knoten
- Berechnung möglichst nahe an den Daten (Data Locality)

## MapReduce – Ablauf

---

1. Input-Daten werden in Splits aufgeteilt

2. Map-Phase:

- Transformation in Key-Value-Paare

3. Shuffle & Sort:

- Gruppierung aller Werte pro Key

4. Reduce-Phase:

- Aggregation der Werte pro Key

5. Schreiben der Ergebnisse nach HDFS

### Vorteile

Einfaches, robustes Modell

Hohe Skalierbarkeit und Fehlertoleranz

### Grenzen

Hohe Latenz (Batch-orientiert)

Umständlich für iterative Algorithmen

### Heute

Meist ersetzt durch Apache Spark

Konzeptionell weiterhin wichtig

Yet Another Resource Negotiator (ab Hadoop 2.x)

Verwaltung von CPU- und Speicherressourcen

Trennung von Ressourcenmanagement und Jobsteuerung

Grundlage für Spark & weitere Frameworks

YARN (Yet Another Resource Negotiator) wurde mit Hadoop 2.x eingeführt, um die Einschränkungen des monolithischen MapReduce-Ansatzes zu überwinden.

Ziele von YARN:

- Trennung von Ressourcenmanagement und Jobsteuerung
- Bessere Skalierbarkeit des Clusters
- Unterstützung mehrerer Verarbeitungssysteme (Multi-Tenant)

### ResourceManager (RM):

- Zentrale Instanz für clusterweite Ressourcenverwaltung
- Besteht aus Scheduler und ApplicationManager

### NodeManager (NM):

- Agent auf jedem Knoten
- Überwacht Container und Ressourcennutzung

### ApplicationMaster (AM):

- Anwendungsspezifische Steuerung
- Aushandlung von Ressourcen und Überwachung der Tasks

Apache Spark ist ein schnelles, verteiltes In-Memory-Framework zur Verarbeitung großer Datenmengen.

- Open Source
- Skalierbar von Laptop bis Cluster
- Entwickelt als Alternative zu MapReduce

- Batch-Verarbeitung großer Datenmengen
- Streaming-Datenverarbeitung (Structured Streaming)
- SQL-Analytics (Spark SQL)
- Machine Learning (MLlib)
- Unterstützung mehrerer Sprachen (Python, Scala, Java, R)

- Häufig kombiniert mit Hadoop (HDFS + YARN)
- Verarbeitung auf Data Lakes (On-Prem & Cloud)
- Deutlich schneller als MapReduce durch In-Memory-Verarbeitung

**Spark ersetzt MapReduce – nicht den Data Lake**

## YARN – Vorteile und Bedeutung

---

- Dynamische Ressourcenzuweisung statt statischer Slots
- Höhere Clusterauslastung
- Parallel Ausführung von MapReduce, Spark, Flink, etc.
- Grundlage moderner Big-Data-Plattformen

**YARN macht Hadoop zu einer universellen Ressourcenplattform**

- Basis vieler Big-Data-Architekturen
- HDFS als Data Lake oder Storage-Basis
- MapReduce meist durch Spark ersetzt
- Konzepte bleiben zentral für Big Data



# Big Data Analytics

## Hadoop - Einführung

## Hadoop - <https://hadoop.apache.org/>

- “The Apache® Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.”



## Hadoop

- Open-Source-Apache-Framework zur Speicherung und Verarbeitung großer Datenmengen auf Computerclustern.
- Vereinfacht die Entwicklung verteilter Anwendungen durch die Bereitstellung einfacher Programmiermodelle.
- Hadoop-Anwendungen können von einem einzelnen Rechner bis hin zu mehreren tausend Servern skaliert werden.
- In Java implementiert.
- Ursprünglich im Jahr 2005 als Teil des Apache-Nutch-Projekts entwickelt.
- Seit 2006 ein eigenständiges Projekt, seit 2008 ein Top-Level-Projekt der Apache Software Foundation.
- Die Kernfunktionalitäten (MapReduce und HDFS) wurden 2006 von Yahoo und Google entwickelt.

## Betriebssysteme

- Hauptsächlich Linux
- „GNU/Linux wird als Entwicklungs- und Produktionsplattform unterstützt. Hadoop wurde auf GNU/Linux-Clustern mit 2000 Knoten demonstriert.“ (Quelle: Hadoop-Dokumentation)
- macOS
- Windows
- → Linux ist die bevorzugte Umgebung.
- → Hadoop ist für Cluster- Computing konzipiert.

## Versionen

- Hadoop 1.x: Einführung von HDFS und MapReduce
- Hadoop 2.x: Einführung von YARN als Ressourcen- und Cluster-Management-Werkzeug sowie Verbesserungen an HDFS
- Hadoop 3.x: Verbesserte Skalierbarkeit, höhere Performance, effizientere Ressourcennutzung, robuste High Availability, GPU-Unterstützung sowie Erasure Coding
- **Apache Hadoop 3.4.2** – dies ist das neueste stabile Release der Hadoop-3.4.x-Linie.

Version	Einführung	Zentrale Merkmale	Einordnung heute
Hadoop 1.x	ca. 2009	<ul style="list-style-type: none"> <li>• HDFS</li> <li>• MapReduce (JobTracker / TaskTracker)</li> <li>• Kein echtes Ressourcenmanagement</li> </ul>	<span style="color: red;">✗</span> Veraltet (Legacy)
Hadoop 2.x	2013	<ul style="list-style-type: none"> <li>• <b>YARN</b> (Ressourcenmanagement)</li> <li>• Verbesserte HDFS-Skalierbarkeit</li> <li>• HA NameNode</li> </ul>	<span style="color: orange;">⚠</span> Wartung / Bestandssysteme
Hadoop 3.0	2017	<ul style="list-style-type: none"> <li>• Bessere Skalierbarkeit</li> <li>• Erasure Coding</li> <li>• Mehrere NameNodes</li> </ul>	<span style="color: orange;">⚠</span> Übergangsversion
Hadoop 3.1–3.3	2018–2023	<ul style="list-style-type: none"> <li>• Performance-Optimierungen</li> <li>• Stabilisierung</li> <li>• Cloud- &amp; Container-Verbesserungen</li> </ul>	<span style="color: orange;">⚠</span> Weit verbreitet
Hadoop 3.4.x	2024–2025	<ul style="list-style-type: none"> <li>• Weitere Performance-Verbesserungen</li> <li>• Robustere High Availability</li> <li>• Bessere Ressourcennutzung</li> <li>• GPU-Support</li> </ul>	<span style="color: green;">✓</span> Aktueller Standard
Aktuell	2025	<b>Apache Hadoop 3.4.2</b>	<span style="color: green;">✓</span> Empfohlen

## Facebook (2020, historisch)

- HDFS-Cluster mit einer Speicherkapazität von 21 PB
- 2000 Maschinen (1200 Maschinen mit jeweils 8 Kernen und 800 Maschinen mit jeweils 16 Kernen)
- 12 TB Speicher pro Maschine und 32 GB RAM pro Maschine
- 15 MapReduce-Tasks pro Maschine
- Die Zahlen zeigen die Größenordnung klassischer Hadoop-Cluster – moderne Systeme sind deutlich leistungsfähiger.

## Heute (2024/2025 – moderne Data-Plattformen):

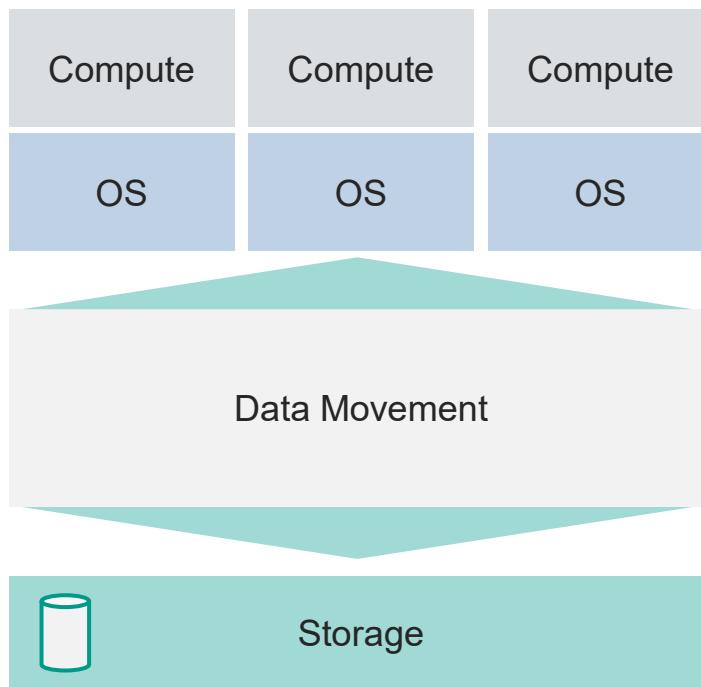
- Multi-Petabyte- bis Exabyte-Skala
- 32–128+ CPU-Cores pro Node
- 128–512+ GB RAM pro Node
- NVMe / SSD / Objekt-Storage
- Data Lakes & hybride Storage-Systeme
- Spark, Flink, Trino, AI/ML-Workloads (GPU-Unterstützung)

- <https://www.linkedin.com/pulse/hadoop-2.0-vs-3.0-unleashing-power-next-generation-enagandula/>

Comparison	Hadoop 2.0	Hadoop 3.0
License	Apache 2.0 Open source	Apache 3.0 Open source
Minimum supported JAVA version	Java 7	Java 8
fault Tolerance	handled by replication	Handled by Erasure coding
Storage scheme	3X replication	Erasure coding in HDFS
Storage overhead	200% overhead Example: If there 6 blocks so there will be 18 blocks occupied(replication)	50% overhead Example: If there 6 blocks so there will be 9 blocks occupied (3 blocks for parity)
Yarn Timeline service	old timeline service	Timeline service V2 and improves Scalability & Reliability
Compatibility File System	HDFS(Default), FTP file system S3, Windows Azure Storage Blobs(WASB)	Supports all file system

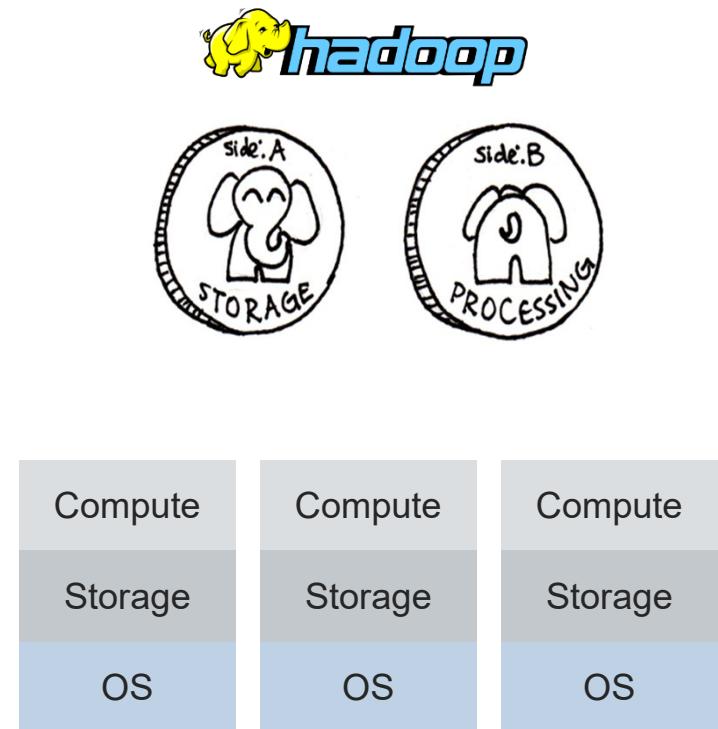
# Das Hadoop Prinzip

- Traditionelle Architektur mit getrennten Speicher- und Rechensystemen

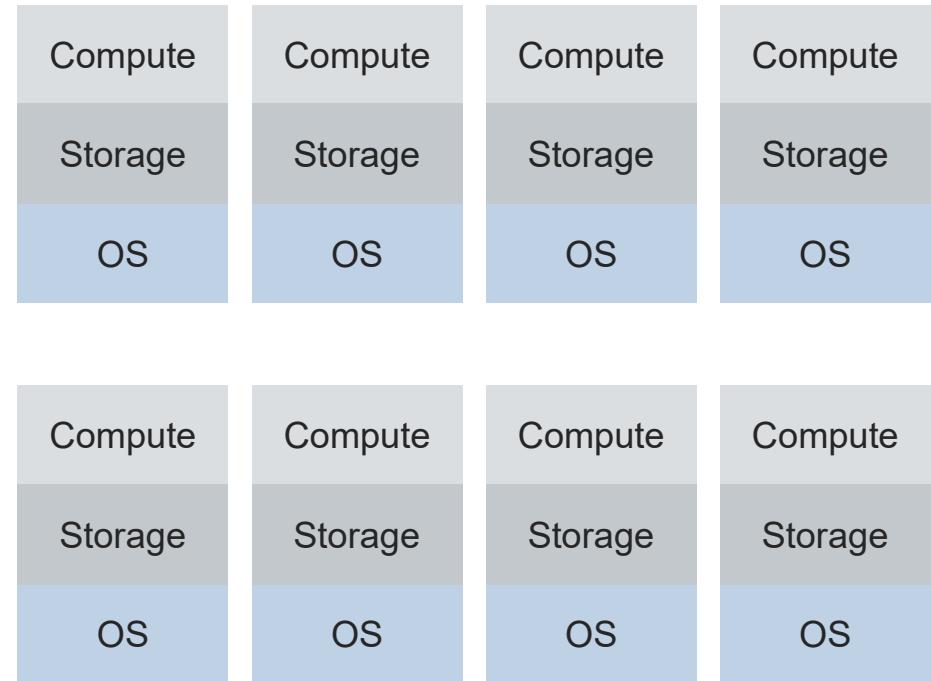


By Apache Software Foundation [Apache License 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>)],  
via Wikimedia Commons

- Hadoop – distributed data storage



- Anstatt einen einzelnen großen Speicher zu verwenden, werden die Daten auf eine große Anzahl von Knoten verteilt.
- Die gleichen Daten werden auf mehrere Knoten repliziert.
- Dies erhöht die Zuverlässigkeit und vermeidet einen Single Point of Failure.
- Knoten können jederzeit hinzugefügt oder entfernt werden.
- Hohe Skalierbarkeit.
- Der Datentransfer wird minimiert, da die Daten dort verarbeitet werden, wo sie gespeichert sind.



## NameNode:

- Der Master-Server, der den Namespace des Dateisystems verwaltet und den Zugriff der Clients auf Dateien steuert.

## Secondary NameNode

- Führt Wartungsaufgaben für den NameNode aus, insbesondere das periodische Zusammenführen von Namespace-Informationen (FsImage) und Transaktionsprotokollen (Edit Logs).  
(Kein Backup- oder Ersatz-NameNode.)

## DataNodes

- Arbeitsknoten, die Datenblöcke speichern und auf Anweisung von Clients oder dem NameNode lesen bzw. schreiben und dem NameNode regelmäßig melden, welche Blöcke sie gespeichert haben.

## MapReduce Engine:

JobTracker:

- Der Master-Knoten, der die Verteilung und Überwachung der MapReduce-Tasks auf die Worker-Knoten koordiniert.

TaskTracker:

- Worker-Knoten, die die vom JobTracker zugewiesenen Tasks ausführen und diesem fortlaufend Statusinformationen melden.

Ab Hadoop 2.x:

- Diese Architektur wurde durch YARN (Yet Another Resource Negotiator) ersetzt, das das Ressourcenmanagement vom Verarbeitungsmodell trennt und mehrere Verarbeitungssysteme (z. B. MapReduce, Spark) parallel unterstützt.

## Scaling Reliability

- Ausfälle sind keine Ausnahme, sondern die Regel!
- 1000 Knoten, mittlere Zeit zwischen Ausfällen (MTBF) < 1 Tag  
4000 Festplatten, 8000 CPU-Kerne, 25 Switches, 1000 Netzwerkkarten, 2000 DIMMs (insgesamt ca. 16 TB RAM)
- Erfordert einen fehlertoleranten Speicher mit angemessenen Verfügbarkeitsgarantien
- Hardwarefehler müssen transparent behandelt werden, ohne Eingriff durch Anwendungen oder Benutzer
- In großen Clustern ist nicht die Frage, ob etwas ausfällt, sondern wann.



[https://cdn.techterms.com/img/lq/bandwidth\\_11.jpg](https://cdn.techterms.com/img/lq/bandwidth_11.jpg)

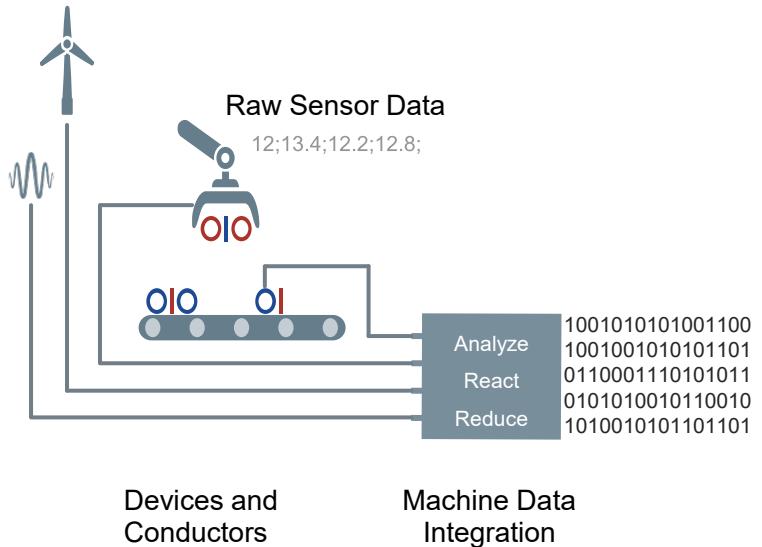
# Relationale Datenbanken vs. Hadoop

Relationale Datenbanken	Kriterium	Hadoop (klassisch)	Cloud Data Lake / Spark SQL
Schema-on-Write, strikt, statisch	<b>Schema</b>	Schema-on-Read, flexibel	Schema-on-Read / Schema-Evolution
Sehr schnelles Lesen & Schreiben (transaktional)	<b>Performance-Fokus</b>	Optimiert für Batch-Schreiben	Optimiert für Analytics (In-Memory, Caching)
Strukturiert	<b>Datenstruktur</b>	Semi- / unstrukturiert	Strukturiert, semi- & unstrukturiert
GB – TB	<b>Datenvolumen</b>	PB	PB – EB
Sehr hoch (ACID)	<b>Datenintegrität</b>	Eingeschränkt	Hoch (ACID via Delta Lake / Iceberg / Hudi)
Verarbeitung meist außerhalb (ETL, DW)	<b>Datenverarbeitung</b>	Verarbeitung nahe am Speicher (Data Locality)	Compute & Storage entkoppelt, elastisch
Begrenzt, vertikal / begrenzt horizontal	<b>Skalierbarkeit</b>	Hoch, nahezu linear	Sehr hoch, elastisch (Cloud-native)
SQL (klassisch)	<b>Abfragesprache</b>	MapReduce, Hive	<b>SQL (Spark SQL, Trino, Presto)</b>
OLTP, klassische BI, Reporting	<b>Anwendungsfälle</b>	Batch-Processing, Offline-Analytics	Moderne Analytics, ML, BI, Streaming
On-Premises	<b>Deployment</b>	On-Premises	Cloud / Hybrid
Relationale DB (Postgres, Oracle, MySQL)	<b>Typische Systeme</b>	HDFS + MapReduce	S3 / ADLS + Spark / Trino

[https://cdn.techterms.com/img/bandwidth\\_11.jpg](https://cdn.techterms.com/img/bandwidth_11.jpg)

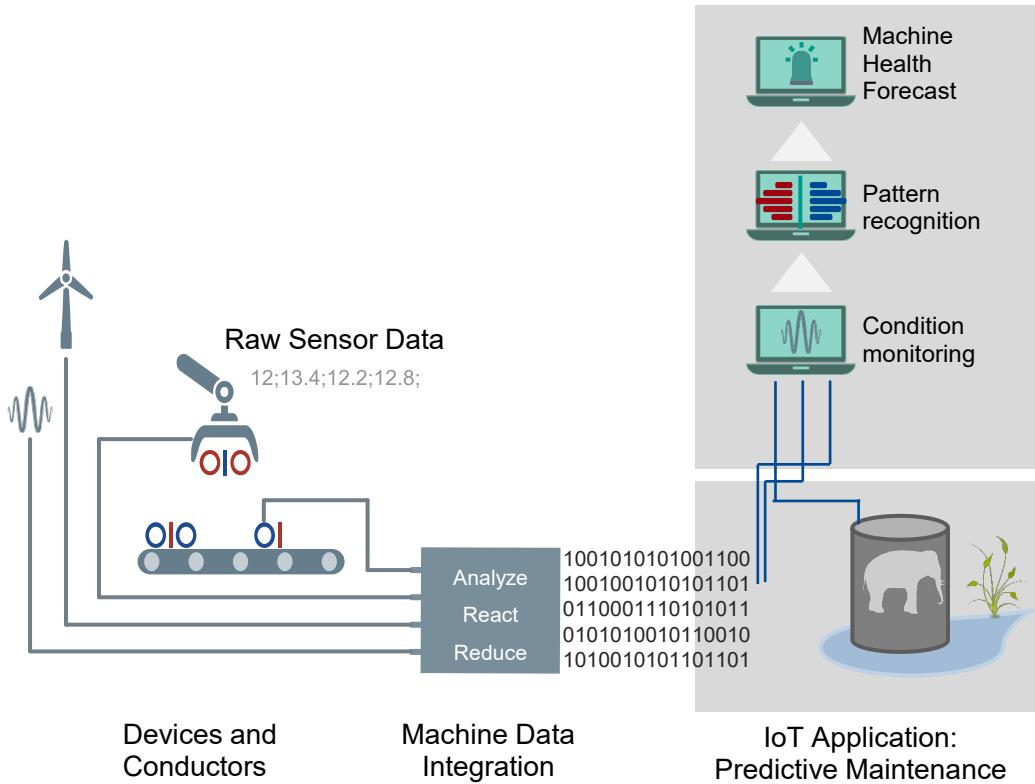
# Beispiel: Predictive Maintenance

Predictive Maintenance:  
The Schema

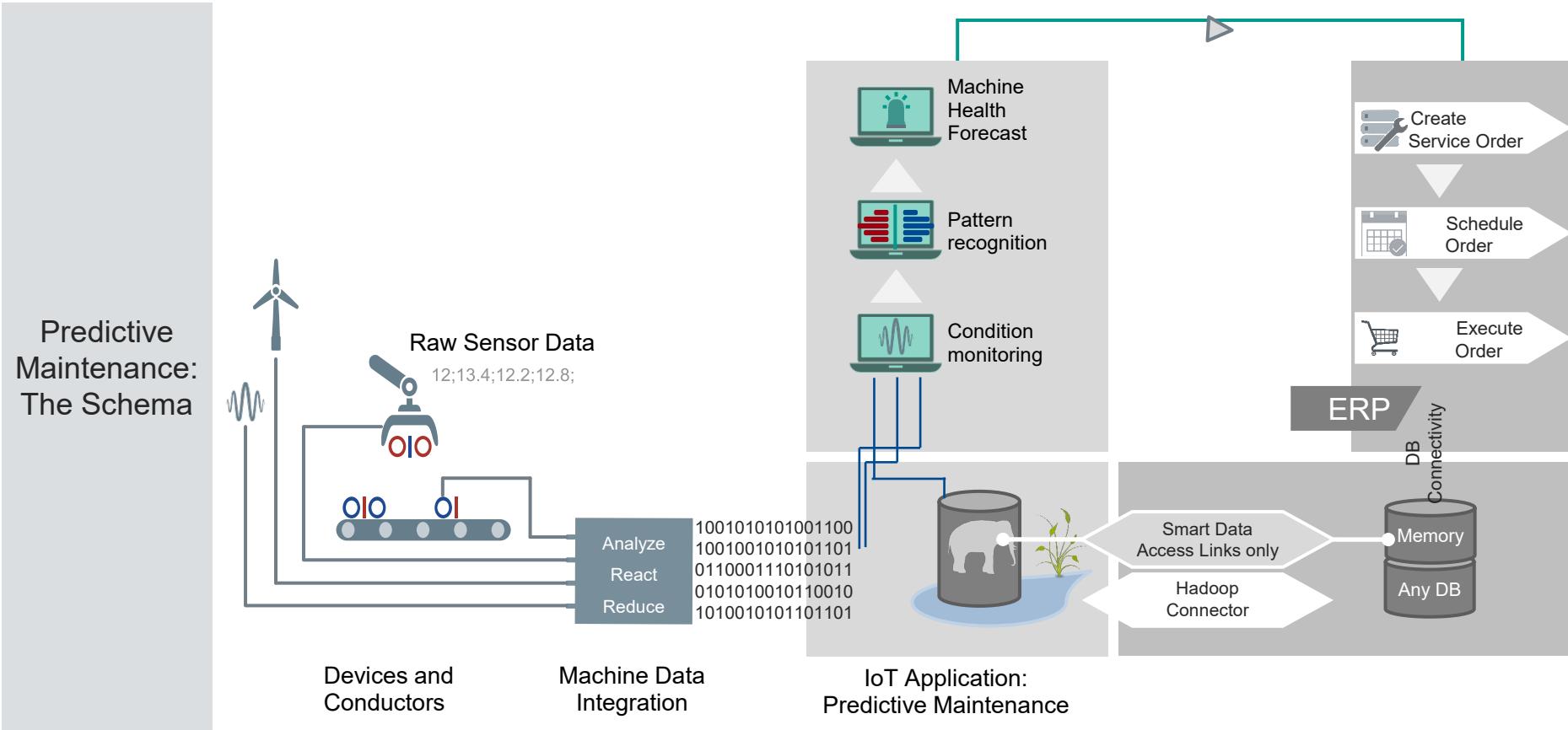


# Beispiel: Predictive Maintenance

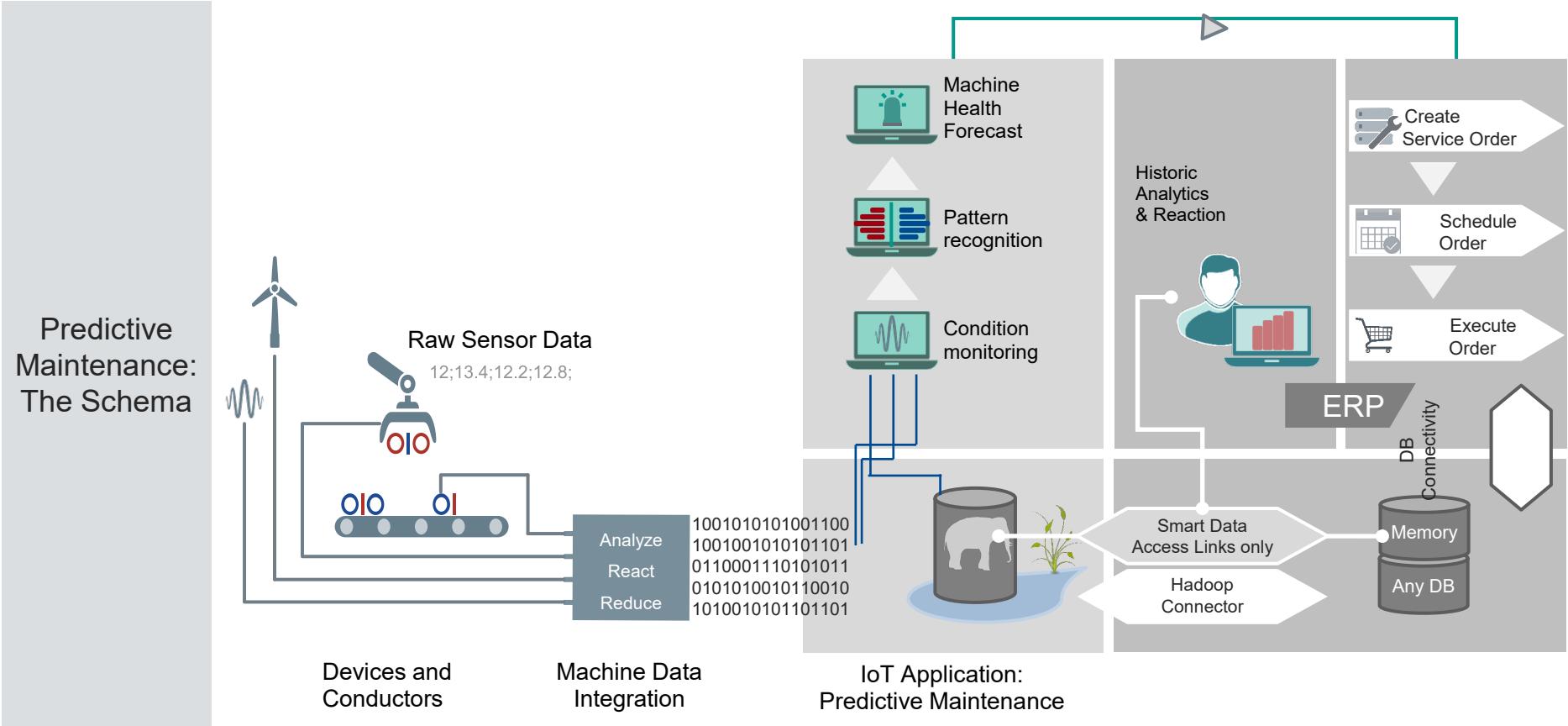
Predictive Maintenance:  
The Schema

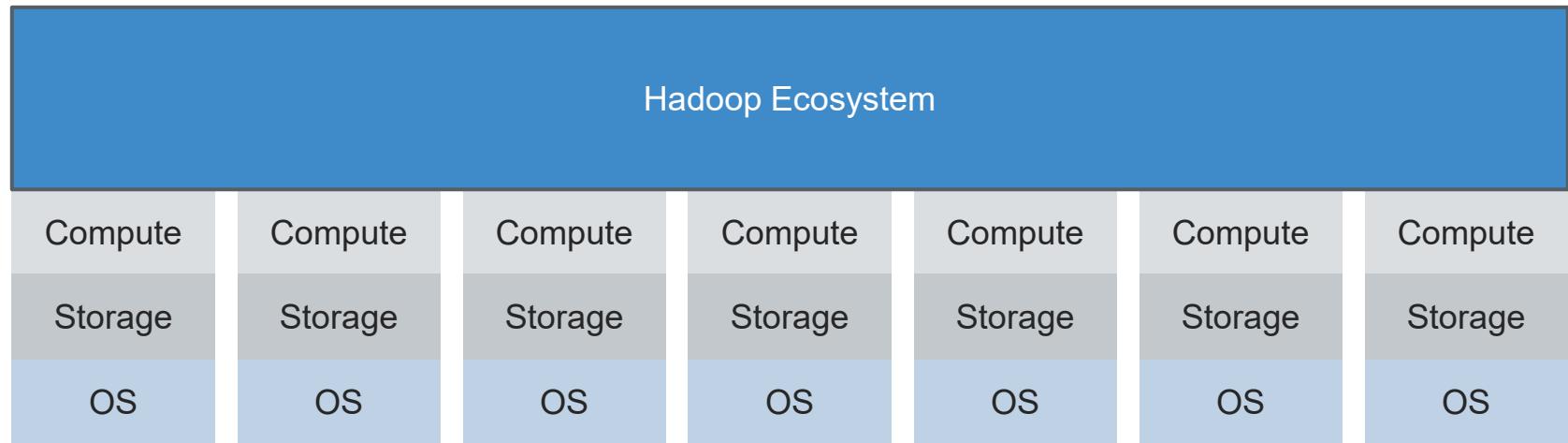


# Beispiel: Predictive Maintenance



# Beispiel: Predictive Maintenance





# **Big Data Analytics**

## **Hadoop - HDFS**

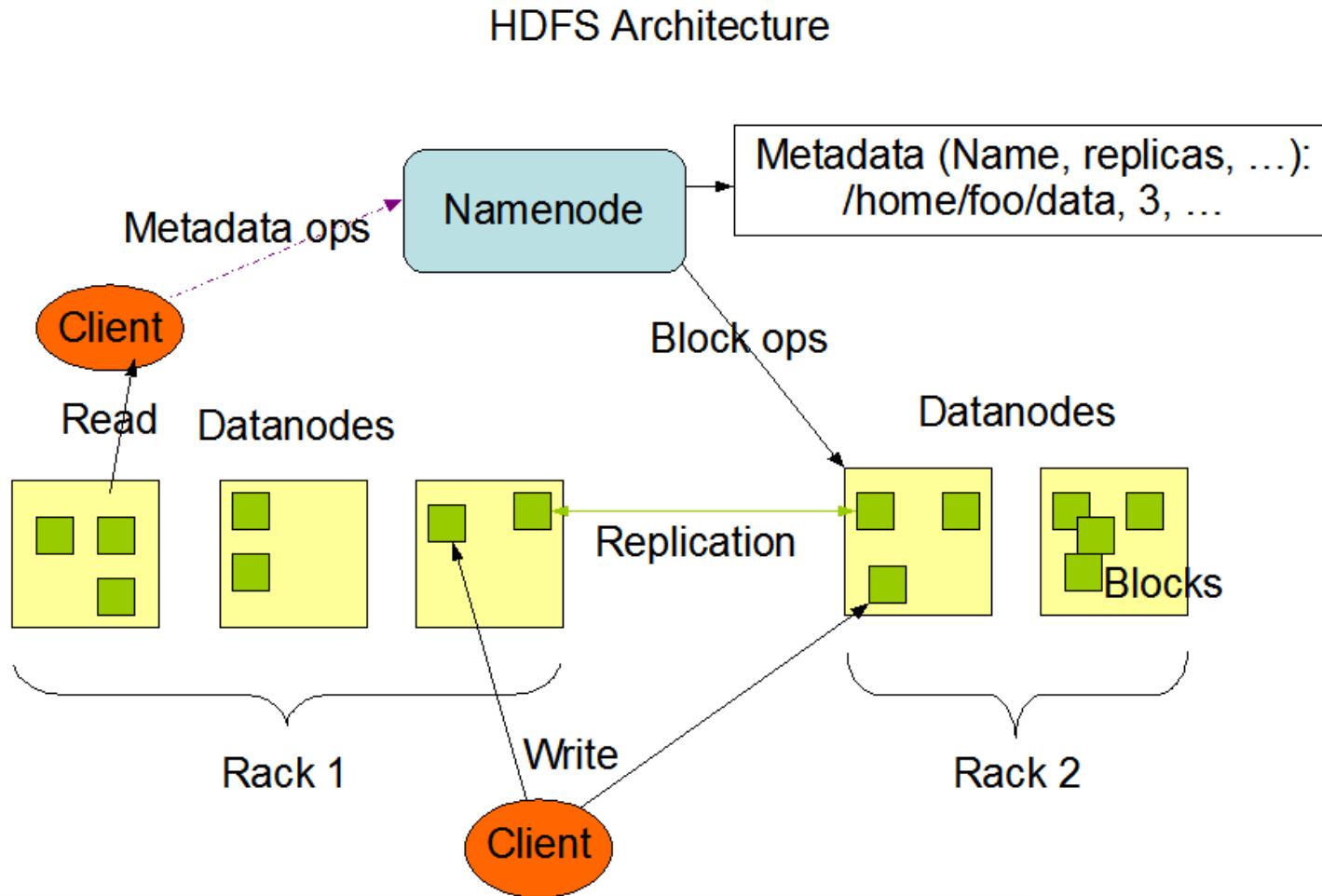
## Hadoop Distributed File System (HDFS)



- Dateisystem eines Hadoop-Clusters
- Optimiert für die verteilte, fehlertolerante Speicherung großer Datenmengen
- Dateien werden in große Blöcke aufgeteilt (typischerweise 128 MB oder 256 MB)
- Jeder Block wird standardmäßig auf mindestens drei Knoten repliziert (zur Erhöhung von Verfügbarkeit und Fehlertoleranz)
- Fällt ein Knoten aus, werden dessen Blöcke automatisch auf andere Knoten neu repliziert
- Hauptanwendungsfall sind große Dateien; viele kleine Dateien werden ineffizient unterstützt und sollten z. B. über Hadoop Archives (HAR), HBase oder Containerformate gebündelt werden
- HDFS unterstützt Rack-Awareness
- Das System kennt die Topologie des Clusters (Position der Knoten und deren Netzwerkzuordnung)
- Datenblöcke werden so über die Knoten verteilt, dass Ausfallsicherheit und Zugriffperformance optimiert werden (z. B. Vermeidung mehrerer Replikate im selben Rack)

**HDFS ist für Durchsatz und Skalierbarkeit optimiert – nicht für niedrige Latenz oder viele kleine Dateien.**

## HDFS Architektur



<https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>

- “The **NameNode** and **DataNode** are pieces of software designed to run on commodity machines. These machines typically run a **GNU/Linux operating system** (OS). HDFS is built using the **Java** language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has **a dedicated machine that runs only the NameNode** software. Each of the **other machines** in the cluster runs one instance of the **DataNode** software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.”
- “The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user **data never flows through the NameNode**.”

<https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>

## Job Execution:

- Ein Client übermittelt einen MapReduce-Job an den JobTracker, der verfügbare TaskTracker-Knoten ermittelt und die Tasks zuweist. (Gilt für Hadoop 1.x; ab Hadoop 2.x übernimmt YARN diese Aufgaben.)
- Map-Phase: Die Eingabedaten werden verarbeitet und in Schlüssel-Wert-Paare (Key-Value Pairs) transformiert. Die Ergebnisse werden anschließend geshuffelt und sortiert, um sie für die Reduce-Phase vorzubereiten.
- Reduce-Phase: Der Reducer verarbeitet die sortierten Ergebnisse der Map-Phase und erzeugt das endgültige Ergebnis.

## Fault Tolerance:

- HDFS ist fehlertolerant ausgelegt, indem Datenblöcke automatisch auf mehrere Knoten repliziert werden.  
Fällt ein DataNode aus, steht eine weitere Kopie des Blocks auf einem anderen Knoten zur Verfügung.
- Der NameNode verwaltet die Metadaten des Dateisystems im Arbeitsspeicher und speichert sie dauerhaft auf der Festplatte: Änderungen werden in einem Transaktionsprotokoll (EditLog) festgehalten. Der vollständige Zustand des Dateisystem-Namespace wird in der Datei FsImage serialisiert gespeichert.

## Skalierbarkeit

- Neue Knoten können dem Cluster hinzugefügt werden, ohne Datenformate, Ladevorgänge, Job-Implementierungen oder darauf aufbauende Anwendungen anpassen zu müssen.
- Die Skalierung erfolgt horizontal durch Hinzufügen weiterer Knoten.

## Cluster-Management

- Hadoop-Cluster werden häufig mit Management- und Monitoring-Werkzeugen wie Apache Ambari verwaltet.
- Apache Ambari stellt:
  - eine **benutzerfreundliche Web-Oberfläche**
  - sowie eine **REST-API** zur Clusterverwaltung bereit.
- Ambari bietet ein Dashboard zur Überwachung von Zustand und Status des Hadoop-Clusters.
- Für die Erfassung von Leistungskennzahlen nutzt Ambari das Ambari Metrics System.
- Für Systemmeldungen und Warnungen verwendet Ambari das Ambari Alert Framework, z. B.:
  - Ausfall eines Knotens
  - geringer verbleibender Speicherplatz
  - andere kritische Systemzustände

Apache Ambari wird weiterhin eingesetzt, wird jedoch in modernen Cloud-Umgebungen häufig durch cloud-native Monitoring- und Managementlösungen ersetzt (z. B. Kubernetes, Cloud Monitoring Services).

## Resource Management (YARN):

- YARN (Yet Another Resource Negotiator) ist eine zentrale Komponente des Hadoop-Ökosystems, die mit Hadoop 2.x eingeführt wurde, um die Einschränkungen des ursprünglichen MapReduce-Programmiermodells zu überwinden.
- YARN ist eine Plattform zur Ressourcenverwaltung, die für die Verwaltung der Rechenressourcen in Clustern und deren Nutzung zur Planung und Ausführung von Anwendungen verantwortlich ist.
- Trennung von Ressourcenmanagement und Job-Steuerung/-Überwachung:  
YARN trennt die Funktionen des Ressourcenmanagements und der Job-Planung/-Überwachung in separate Daemons.  
Diese Trennung verbessert die Skalierbarkeit und Auslastung des Clusters, da mehrere Datenverarbeitungs-Engines wie MapReduce, Apache Tez und Apache Spark parallel auf demselben Hadoop-Cluster betrieben werden können.
- Verbesserte Clusterauslastung: YARN ermöglicht eine dynamische Zuweisung von Ressourcen entsprechend den Anforderungen der Anwendungen.  
Dies führt zu einer besseren Ressourcennutzung im Vergleich zum statischen Slot-basierten Modell des ursprünglichen MapReduce-Frameworks.
- Unterstützung verschiedener Workloads: YARN unterstützt Batch-Verarbeitung, interaktive Abfragen und Streaming-/Near-Realtime-Workloads, die im ursprünglichen Hadoop-1.x-MapReduce-Modell nicht oder nur eingeschränkt möglich waren.

**YARN macht Hadoop von einem reinen MapReduce-System zu einer allgemeinen Plattform für verteilte Datenverarbeitung.**

## ResourceManager (RM):

- Der ResourceManager ist die zentrale Master-Komponente von YARN, die über alle verfügbaren Cluster-Ressourcen (CPU, Speicher usw.) entscheidet.
- Er besteht aus zwei Hauptkomponenten:
  - dem Scheduler
  - dem ApplicationManager
- Scheduler  
Der Scheduler ist für die Zuweisung von Ressourcen an laufende Anwendungen verantwortlich. Dabei berücksichtigt er Kapazitäten, Warteschlangen, Prioritäten und Policies (z. B. Capacity Scheduler, Fair Scheduler).  
Der Scheduler überwacht keine Anwendungen und startet keine Tasks.
- ApplicationManager ist zuständig für:
  - die Annahme von Job-Einreichungen,
  - die Aushandlung des ersten Containers zur Ausführung des anwendungsspezifischen ApplicationMaster,
  - das Neustarten des ApplicationMaster-Containers im Fehlerfall.

### NodeManager

Der NodeManager ist ein Agent auf jedem Cluster-Knoten.

Er ist verantwortlich für:

- die Verwaltung der Container auf dem jeweiligen Knoten,
- die Überwachung der Ressourcennutzung( CPU, Speicher, Festplatte, Netzwerk),
- die Rückmeldung von Status- und Ressourcendaten an den ResourceManager.

Der ResourceManager verteilt Ressourcen global, der NodeManager verwaltet sie lokal auf jedem Knoten.

### ApplicationMaster (AM)

Der ApplicationMaster ist eine instanzierte, frameworkspezifische Komponente (z. B. für MapReduce, Spark, Tez).

Er ist verantwortlich für:

- die Aushandlung geeigneter Ressourcen-Container mit dem Scheduler (über den ResourceManager),
- die Überwachung des Status der zugewiesenen Container,
- das Monitoring des Fortschritts der Anwendung,
- sowie das Handling von Fehlern innerhalb der Anwendung.

Für jede Anwendung existiert genau ein eigener ApplicationMaster.

## Container

Wird eine Anwendung bei YARN eingereicht, weist der ResourceManager einen oder mehrere Container zu.

Ein Container beschreibt eine logische Bündelung von Ressourcen auf einem einzelnen Knoten, z. B.:

- Arbeitsspeicher (RAM)
- CPU-Kerne
- (optional) weitere Ressourcen wie lokale Festplatte oder Netzwerk

Container sind isolierte Ausführungseinheiten, jedoch keine Virtualisierung im Sinne von VMs oder Containern wie Docker.

Ein YARN-Container ist **kein Betriebssystem-Container**, sondern eine **Ressourcenzuteilung**, die von NodeManagern überwacht wird.

## Funktionsweise von YARN

- Bei der Einreichung einer Anwendung weist der ResourceManager zunächst einen Container zu, um den ApplicationMaster der Anwendung zu starten.
- Der ApplicationMaster verhandelt anschließend die benötigten Ressourcen mit dem ResourceManager und arbeitet mit den NodeManagern zusammen, um die Tasks auszuführen und deren Fortschritt zu überwachen.
- Der ResourceManager ist für die clusterweite Ressourcenplanung zuständig, während die NodeManager die Ausführung auf den einzelnen Rechenknoten übernehmen.

## Bedeutung von YARN

- YARN hat Hadoop zu einer mandantenfähigen (Multi-Tenant) Datenverarbeitungsplattform weiterentwickelt.
- Unterschiedliche Datenverarbeitungs-Engines (z. B. MapReduce, Spark, Tez, Flink) können parallel auf denselben Daten in HDFS arbeiten.
- Dies stellt einen wesentlichen Fortschritt gegenüber dem ursprünglichen Hadoop-MapReduce-Paradigma dar, das primär für Batch-Verarbeitung mit MapReduce konzipiert war.

YARN entkoppelt Ressourcenmanagement von der Datenverarbeitung und macht Hadoop zu einer universellen Plattform.

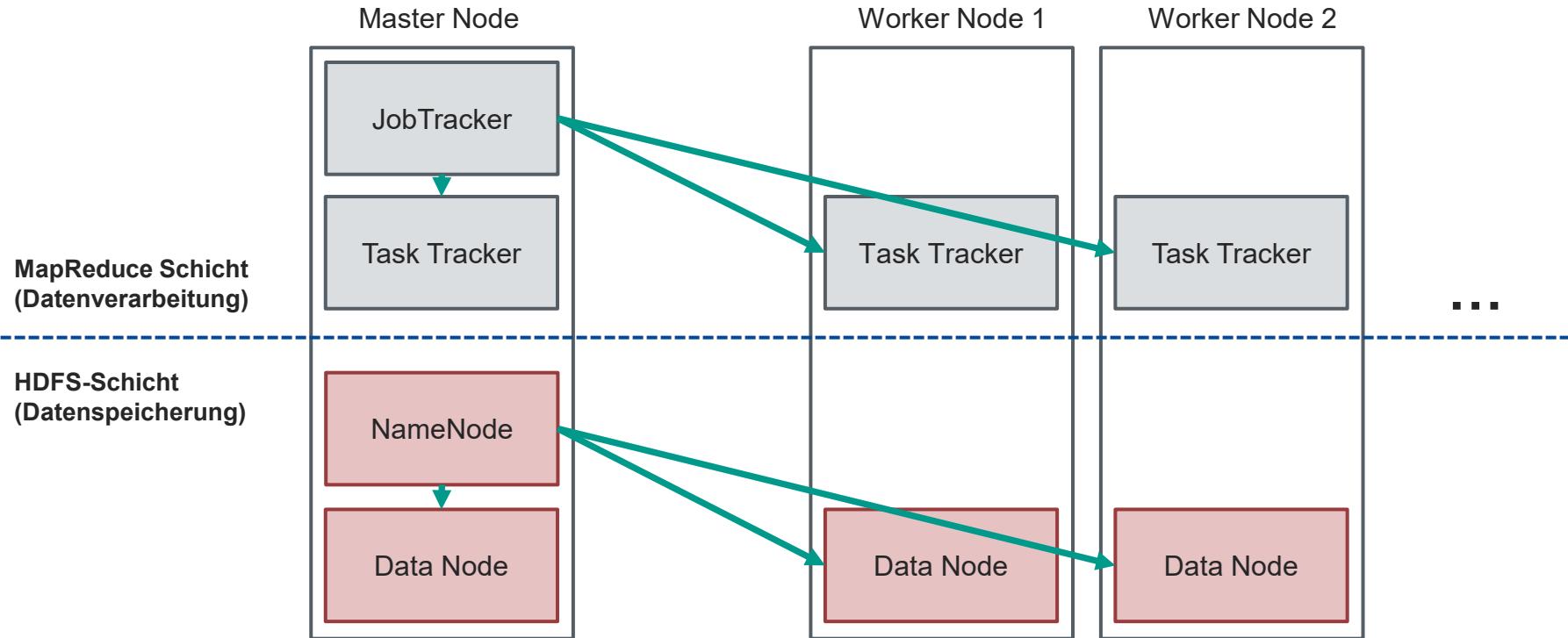
Der JobTracker ist eine zentrale Komponente des MapReduce-Frameworks (Hadoop 1.x) und verantwortlich für:

- Ressourcenverwaltung
- Task-Planung
- Überwachung von Jobs
- Fehlerbehandlung und Wiederherstellung

In neueren Hadoop-Versionen (ab Hadoop 2.x) wurden der JobTracker und der TaskTracker durch YARN (Yet Another Resource Negotiator) ersetzt. YARN teilt die Aufgaben des JobTrackers auf:

- den ResourceManager (clusterweites Ressourcenmanagement)
- den ApplicationMaster (anwendungsspezifische Jobsteuerung)  
Dies führt zu besserer Skalierbarkeit und höherer Clusterauslastung.

Der TaskTracker ist in Hadoop 1.x für die Ausführung der Tasks zuständig und kommuniziert regelmäßig mit dem JobTracker über Fortschritt und Status der Tasks.



Das Bild zeigt die klassische Hadoop-1.x-Architektur mit einem Master Node und mehreren Worker Nodes, unterteilt in eine MapReduce-Schicht (Datenverarbeitung) und eine HDFS-Schicht (Datenspeicherung). Der JobTracker auf dem Master steuert die Ausführung der Jobs und verteilt Tasks an die TaskTracker auf den Worker Nodes, während der NameNode die Metadaten des Dateisystems verwaltet und die DataNodes die eigentlichen Datenblöcke speichern. Die Verarbeitung erfolgt möglichst datenlokal auf den Knoten, auf denen die Daten liegen, um Netzwerkübertragungen zu minimieren.

## NameNode

- Verwaltet den Dateisystem-Namespace und die Metadaten (Zuordnung von Dateien zu Datenblöcken).
- Koordiniert Lese- und Schreibzugriffe der Clients auf die DataNodes.
- Steuert die Replikation der Datenblöcke über die DataNodes hinweg.
- Stellt einen Single Point of Failure dar, sofern kein Hadoop-HA-Modus eingesetzt wird.

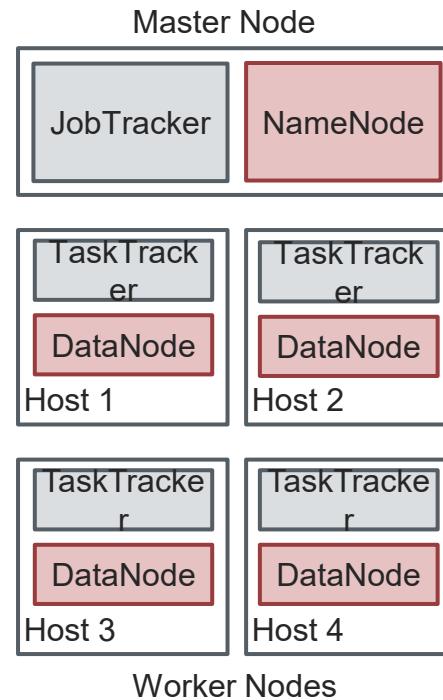
## DataNode

- Speichert die eigentlichen Datenblöcke.
- Ermöglicht Clients den direkten Zugriff auf konkrete Datenblöcke (Lesen/Schreiben).

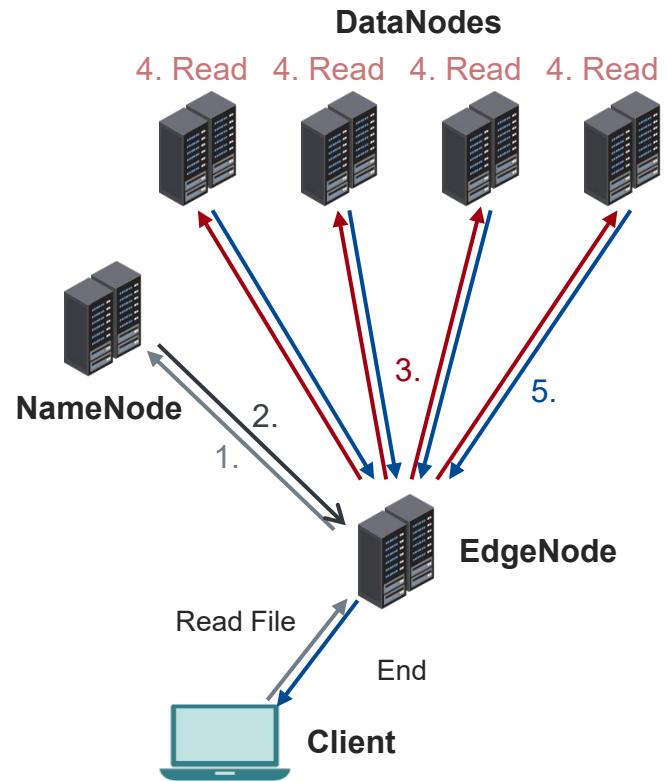
*Der NameNode weiß, wo die Daten liegen – die DataNodes speichern die Daten.*

## EdgeNode

- Dient als Zugangspunkt (Gateway) zum Hadoop-Cluster für Datenübertragung und Client-Zugriffe.
- Führt Client-Anwendungen (z. B. Hive, Spark-Clients) sowie Werkzeuge zur Clusterverwaltung aus.
- Der EdgeNode ist kein Teil der Datenverarbeitung, sondern die Schnittstelle zwischen Nutzern und Cluster.

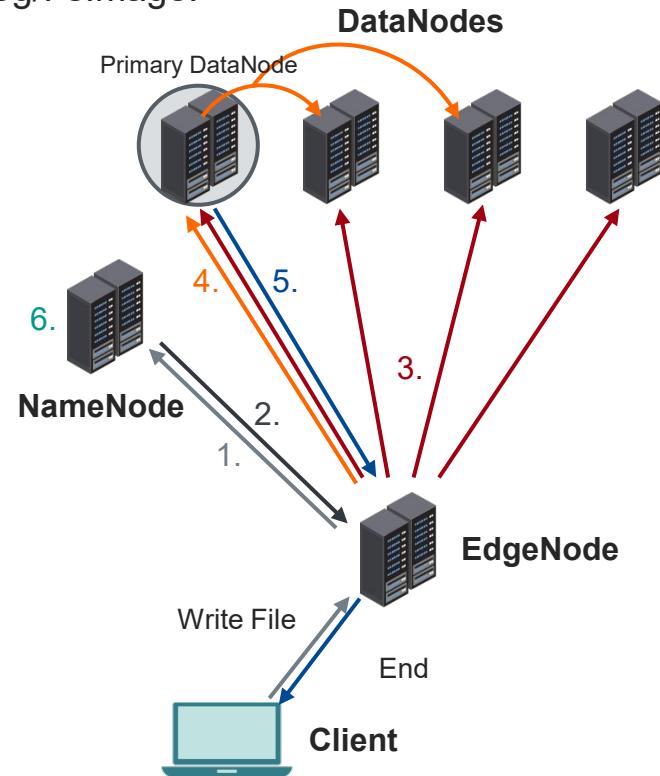


1. Die Client-API berechnet anhand des Dateizeigers (Offset), welcher Datenblock benötigt wird, und sendet eine Anfrage an den NameNode.
2. Der NameNode antwortet mit der Information, welche DataNodes eine Kopie des angeforderten Blocks besitzen.
3. Der Client kontaktiert die DataNodes anschließend direkt, ohne den NameNode weiter einzubeziehen.
4. Die DataNodes lesen die angeforderten Datenblöcke von ihrem lokalen Speicher.
5. Die DataNodes senden die Daten bzw. eine Erfolgsrückmeldung direkt an den Client.



1. Der Client kontaktiert den NameNode, der anhand der Replikationsstrategie einen Primary DataNode sowie die Secondary DataNodes für den neuen Datenblock bestimmt.
2. Der NameNode antwortet dem Client und teilt ihm mit, welcher DataNode der Primary und welche die Secondary-Replikate sind.
3. Der Client streamt die Daten an den Primary DataNode, der die Daten pipeline-artig an die weiteren Secondary DataNodes weiterleitet; die Daten werden zunächst gepuffert.
4. Nach dem vollständigen Schreiben sendet der Client einen Commit an den Primary DataNode, der die Reihenfolge der Bestätigung festlegt und an die Secondary DataNodes weitergibt.

5. Nachdem alle Secondary DataNodes den erfolgreichen Schreibvorgang bestätigt haben, sendet der Primary DataNode eine Erfolgsmeldung an den Client.
6. Der NameNode aktualisiert die Metadaten, d. h. Blockverteilung und Replikationsinformationen, und schreibt diese Änderungen persistent in das EditLog/FsImage.





## Big Data Analytics Hadoop – Fallstudie Übersetzung

## Hintergrund

- Ein Übersetzungsunternehmen verarbeitet Millionen von Wörtern in verschiedenen Sprachen. Dazu pflegt es ein Translation Memory (TM), also eine Datenbank mit bereits übersetzten Sätzen, Phrasen und Absätzen.
- Zusätzlich kann ein Parallelkorpus vorliegen – eine Sammlung von Originaltexten und deren Übersetzungen, die zur Analyse sowie zum Training maschinellder Übersetzungssysteme verwendet wird.

## Datenspeicherung

- Das Hadoop Distributed File System (HDFS) speichert umfangreiche Parallelkorpora und Translation-Memory-(TM)-Datenbanken.
- Diese Datensätze sind häufig zu groß, als dass sie von klassischen relationalen Datenbanksystemen (RDBMS) effizient verarbeitet werden könnten – insbesondere bei unstrukturierten Textdaten.

## Datenverarbeitung

- Das MapReduce-Programmiermodell verarbeitet diese großen Datenmengen, um Übereinstimmungen und Muster zu identifizieren.
- Beispielsweise kann Hadoop beim Start eines neuen Projekts eingesetzt werden, um die Translation Memory schnell nach passenden früheren Übersetzungen zum neuen Inhalt zu durchsuchen.

## Datenanalyse

- Hadoop kann zur Sprachanalyse der gespeicherten Daten verwendet werden, um häufige Phrasen und Terminologie zu identifizieren.
- Dies verbessert die Konsistenz der Übersetzungen über verschiedene Übersetzer und Projekte hinweg.

## Maschinelles Lernen

- Durch die Integration von Machine-Learning-Bibliotheken wie Apache Mahout oder Spark MLlib unterstützt Hadoop das Training maschineller Übersetzungsmodelle auf Parallelkorpora.
- Dadurch kann das Unternehmen eigene Machine-Translation-Modelle entwickeln und kontinuierlich verbessern.

Hadoop dient hier als skalierbare Plattform für Speicherung, Analyse und ML auf großen Textdatenbeständen.

## Skalierbarkeit

- Mit dem Wachstum der unternehmensweiten Übersetzungsdatenbank ermöglicht Hadoop eine einfache Skalierung.
- Zusätzliche Knoten können dem Cluster hinzugefügt werden, um steigende Datenmengen zu verarbeiten, ohne dass eine grundlegende Umstrukturierung der Infrastruktur erforderlich ist.

## Kosteneffizienz

- Im Vergleich zu klassischen relationalen Datenbanksystemen (RDBMS), die bei wachsenden Datenvolumina schnell sehr kostspielig werden können, bietet Hadoop durch seine verteilte Architektur eine kosteneffizientere Lösung für Speicherung und Verarbeitung großer Datenmengen.

## Ergebnis (Outcome)

- Durch den Einsatz von Hadoop kann das Übersetzungsunternehmen seine operative Effizienz steigern, indem es schnelle, verteilte Datenverarbeitung nutzt.
- Dies verbessert die Fähigkeit, präzise und konsistente Übersetzungen bereitzustellen, unterstützt die Entwicklung proprietärer Übersetzungstechnologien und verschafft dem Unternehmen letztlich einen Wettbewerbsvorteil im Übersetzungsmarkt.

## Modernisierte Architektur

- mit Data Lake und Apache Spark

## Fokus

- Skalierbarkeit, Analytics und Machine Learning

## Datenspeicherung

- Speicherung von Parallelkorpora und Translation Memories (TM)
- Data Lake auf HDFS oder Object Storage (S3 / ADLS)
- Geeignet für strukturierte und unstrukturierte Textdaten
- Hohe Skalierbarkeit gegenüber klassischen RDBMS

## Datenverarbeitung

- Verarbeitung großer Textmengen mit Apache Spark
- In-Memory-Verarbeitung für schnelle Analysen
- Suche in Translation Memories
- Erkennung von Mustern und Ähnlichkeiten

## Datenanalyse

- SQL-ähnliche Abfragen mit Spark SQL
- Analyse häufiger Phrasen und Terminologie
- Identifikation von Inkonsistenzen
- Verbesserung der Übersetzungsqualität

## Maschinelles Lernen

- Training von NLP- und Übersetzungsmodellen
- Nutzung von Spark MLlib oder angebundenen ML-Frameworks
- Verarbeitung großer Parallelkorpora
- Kontinuierliche Modellverbesserung

## Skalierbarkeit

- Horizontale Skalierung von Speicher und Rechenleistung
- Trennung von Compute und Storage
- Flexible Nutzung von Cloud-Ressourcen

## Kosteneffizienz

- Bedarfsgerechte Ressourcennutzung
- Geringere Kosten als klassische RDBMS bei großen Datenmengen
- Besonders geeignet für stark wachsende Textdaten

## Ergebnis (Outcome)

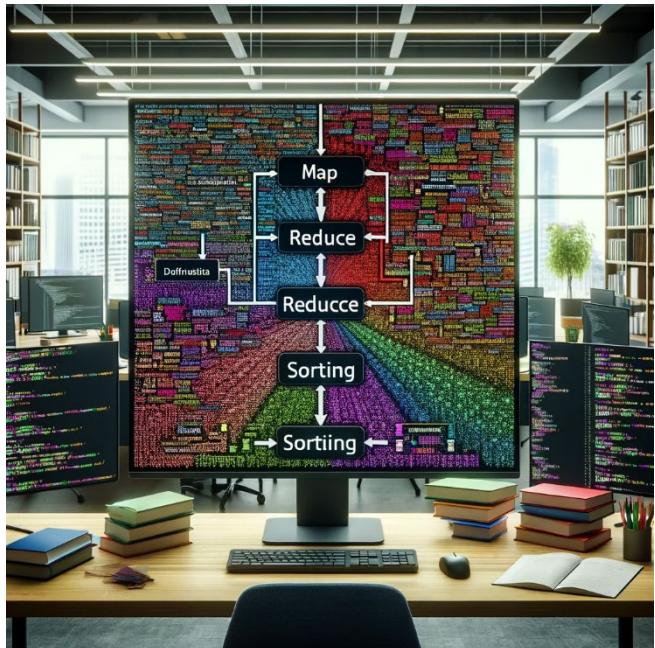
- Höhere operative Effizienz
- Konsistentere und qualitativ bessere Übersetzungen
- Schnellere Analyse- und Trainingszyklen
- Nachhaltiger Wettbewerbsvorteil

## Zusammenfassung

- Hadoop bildet den Data Lake,
- Spark liefert die Intelligenz darüber.

Aspekt	Klassischer Ansatz	Modernisierter Ansatz (Data Lake & Spark)
Datenspeicherung	Relationale Datenbank (RDBMS) für TM	<b>Data Lake</b> (HDFS / Object Storage) für TM & Parallelkorpora
Datenstruktur	Stark strukturiert	<b>Strukturiert &amp; unstrukturiert</b> (Text, Metadaten)
Skalierbarkeit	Begrenzt, teuer bei Wachstum	<b>Horizontale Skalierung</b> durch Hinzufügen von Knoten
Datenvolumen	GB–TB	<b>TB–PB</b>
Suche in TM	SQL-Abfragen	<b>Spark / verteilte Textsuche</b>
Analyse	Begrenzte Textanalysen	<b>Skalierbare Sprach- und Textanalysen</b>
Maschinelles Lernen	Externe Systeme notwendig	<b>Integriertes ML</b> (Spark MLlib, NLP-Pipelines)
Performance	Gut bei kleinen Datenmengen	<b>Hohe Performance</b> bei großen Textkorpora
Kosten	Hohe Lizenz- und Skalierungskosten	<b>Kosteneffizient</b> durch Commodity-Hardware / Cloud
Flexibilität	Schema-on-write	<b>Schema-on-read</b>
Echtzeitfähigkeit	Eingeschränkt	<b>Batch + (Near-)Realtime</b>
Zielsetzung	Verwaltung bestehender Übersetzungen	<b>Datengetriebene Optimierung &amp; KI-gestützte Übersetzung</b>

Finde einen Anwendungsfall, in dem Hadoop in einem Unternehmen eine relationale Datenbank ersetzen kann.



# Big Data Analytics

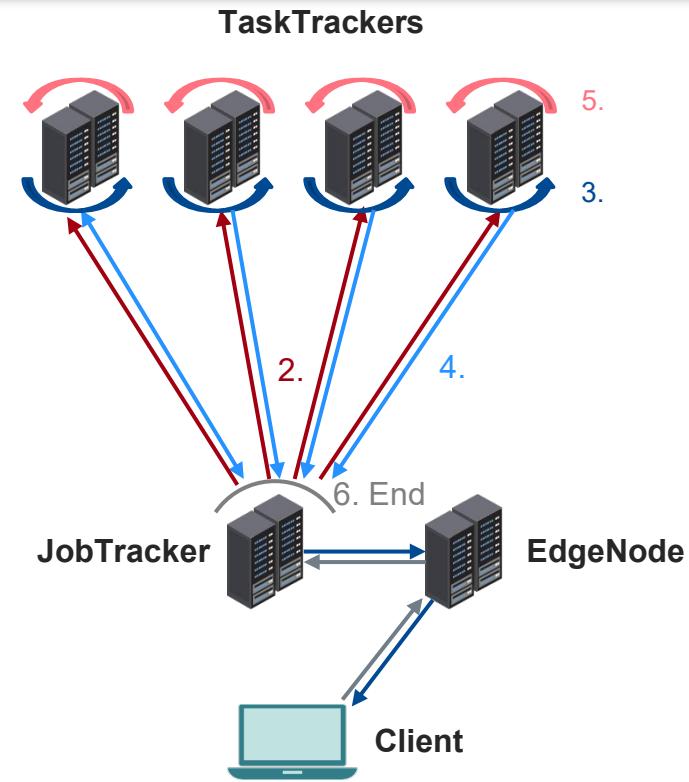
## Hadoop – Map Reduce

- Software-Framework zur verteilten Verarbeitung großer Datenmengen
- Paralleler, verteilter Algorithmus, der auf einem Cluster ausgeführt wird
- Ursprünglich von Google als Werkzeug zur Big-Data-Analyse entwickelt
- Die Datenverarbeitung erfolgt direkt auf den Clusterknoten, auf denen die Daten auch gespeichert sind (Datenlokalität)
- MapReduce-Tasks werden auf mehrere Clusterknoten verteilt, um die Vorteile der parallelen Verarbeitung optimal zu nutzen

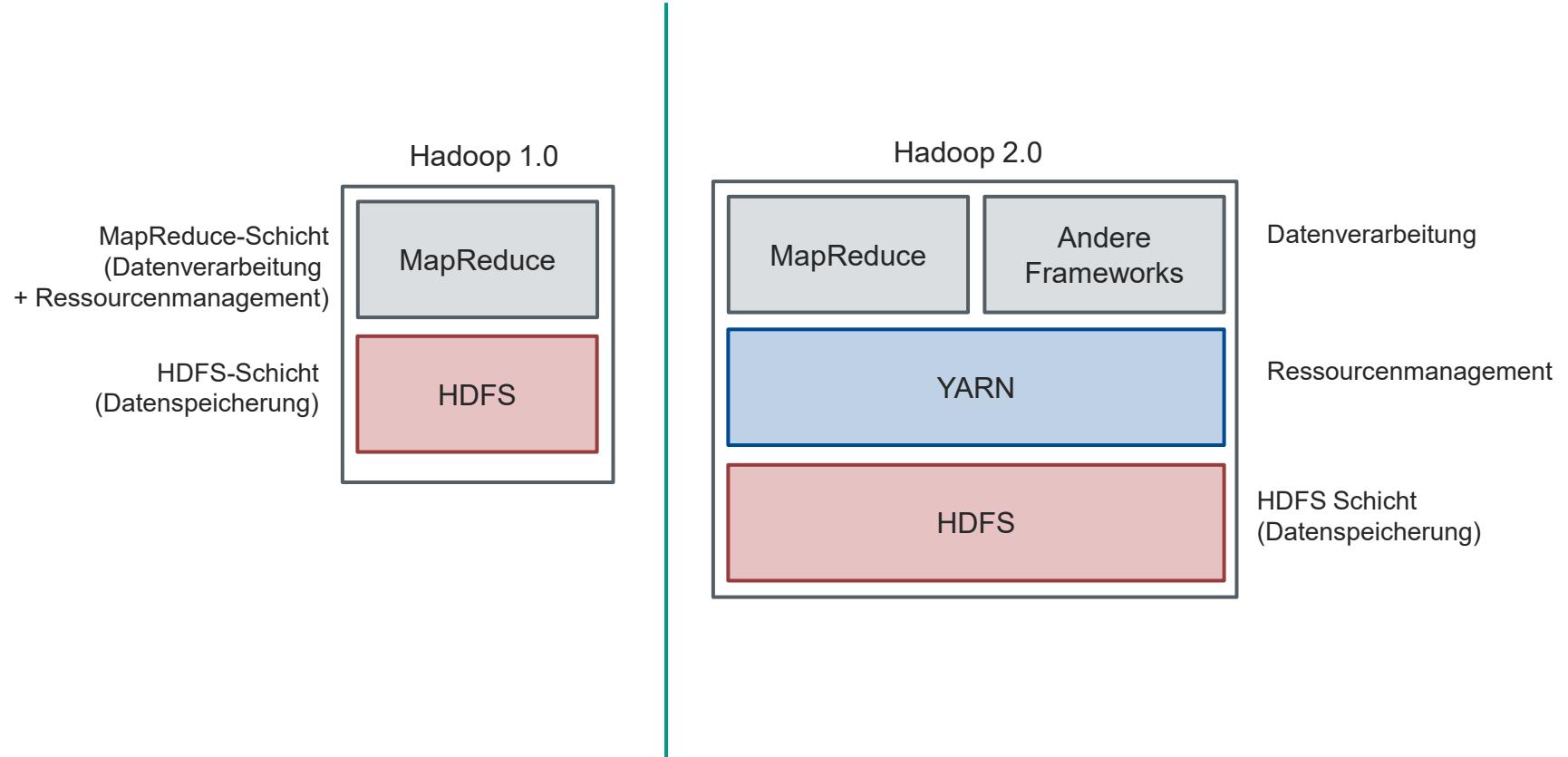
## Hadoop Ecosystem – MapReduce Execution

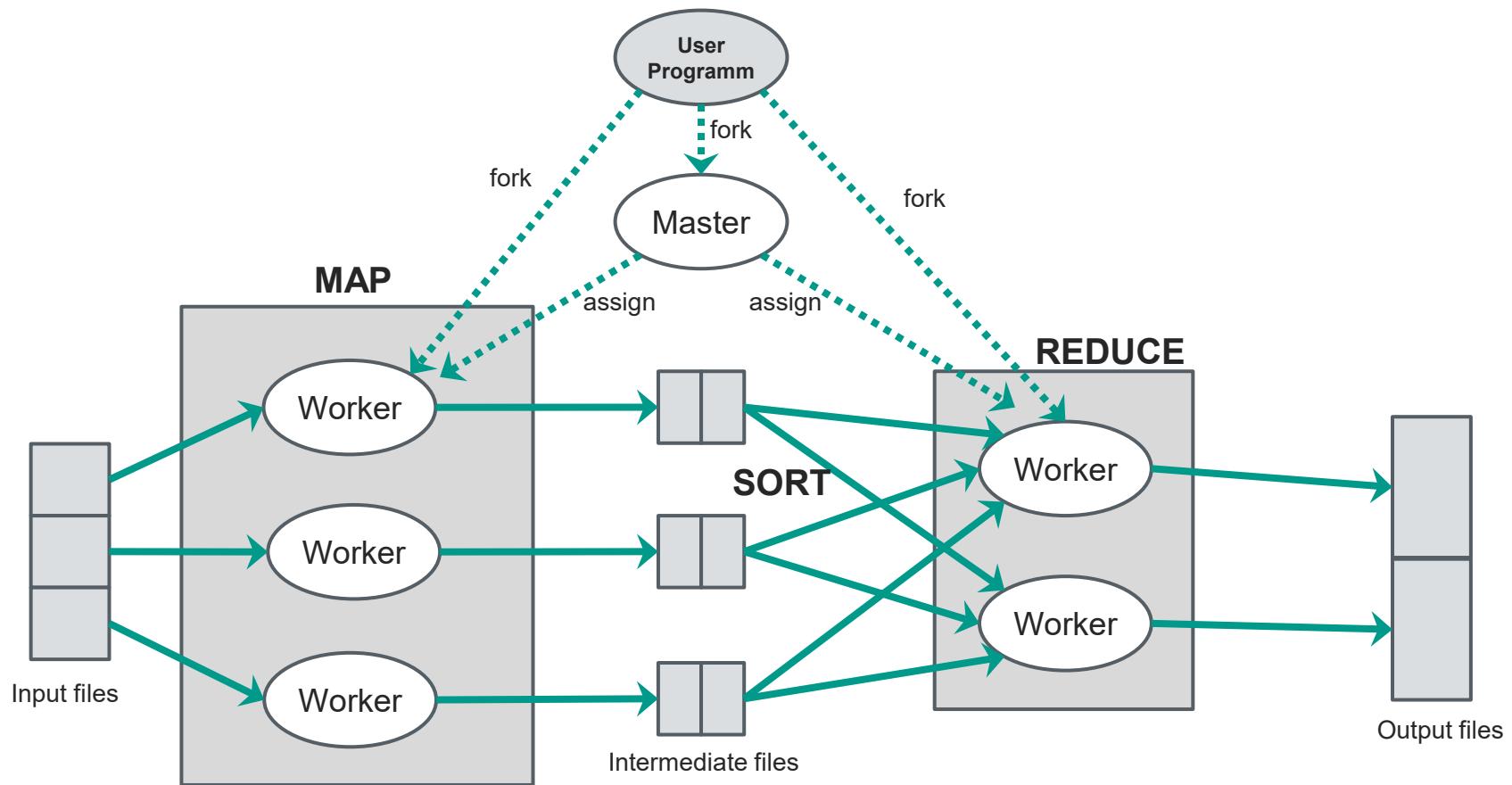
1. Der JobTracker bestimmt anhand des Eingabepfads die Anzahl der Splits und wählt geeignete TaskTracker aus – bevorzugt solche mit Netzwerknähe zu den Datenquellen (Datenlokalität).
2. Der JobTracker sendet die Task-Anfragen an die ausgewählten TaskTracker.
3. Jeder TaskTracker startet die Map-Phase, indem er die Eingabedaten aus den zugewiesenen Splits liest und verarbeitet. Nach Abschluss der Map-Tasks informieren die TaskTracker den JobTracker. Sobald alle Map-Tasks abgeschlossen sind, weist der JobTracker die ausgewählten TaskTracker an, mit der Reduce-Phase zu beginnen.
4. Jeder Reduce-Task liest die Zwischenergebnisse (Partitionen) remote, führt die Reduce-Funktion aus und schreibt die aggregierten Key-Value-Ergebnisse in die Ausgabedatei(eine Ausgabedatei pro Reducer).
5. Nach Abschluss von Map- und Reduce-Phase gibt der JobTracker das Client-Programm frei und signalisiert den erfolgreichen Abschluss des Jobs.

Diese Ablaufbeschreibung bezieht sich auf **Hadoop 1.x** mit JobTracker/TaskTracker. In **Hadoop ≥ 2.x** wird die Ausführung über **YARN** gesteuert.)



MapReduce arbeitet strikt phasenweise: erst Map, dann Shuffle/Sort, dann Reduce.





## Key-Value-Pairs – Map Phase

- Ein Key-Value-Paar besteht aus zwei miteinander verknüpften Datenelementen:
- einem Key, der als eindeutiger Identifikator dient,
- und einem Value, der die dem Key zugeordneten Daten enthält.
- Einfaches Beispiel: Zählen, wie oft jedes Wort in einem Textdokument vorkommt.
- Key: Ein bestimmtes Wort im Text.
- Value: Die Anzahl, wie oft dieses Wort vorkommt.
- Example  
"I like bananas and apples. But I like apples more than bananas. Apples are usually green."
- „Ich mag Bananen und Äpfel. Aber ich mag Äpfel mehr als Bananen. Äpfel sind normalerweise grün.“
- In der Map-Phase würde für jedes Wort ein Key-Value-Paar erzeugt, z. B. (Äpfel, 1), (Bananen, 1), ...



## Key-Value-Pairs - Map Phase

- Example

"I like bananas and apples. But I like apples more than bananas. Apples are usually green."

- |                   |                    |
|-------------------|--------------------|
| 1. ("i", 1)       | 9. ("apples", 1)   |
| 2. ("like", 1)    | 10. ("more", 1)    |
| 3. ("bananas", 1) | 11. ("than", 1)    |
| 4. ("and", 1)     | 12. ("bananas", 1) |
| 5. ("apples", 1)  | 13. ("apples", 1)  |
| 6. ("but", 1)     | 14. ("are", 1)     |
| 7. ("i", 1)       | 15. ("usually", 1) |
| 8. ("like", 1)    | 16. ("green", 1)   |

## Key-Value-Pairs - Reduce Phase

1. ("i", 2)
2. ("like", 2)
3. ("bananas", 2)
4. ("and", 1)
5. ("apples", 3)
6. ("but", 1)
7. ("more", 1)
8. ("than", 1)
9. ("are", 1)
10. ("usually", 1)
11. ("green", 1)

In einer realen MapReduce-Ausführung

- a) sind die Datenmengen deutlich größer, und
- b) der Prozess umfasst die Verteilung der Aufgaben auf mehrere Knoten eines Clusters, um eine parallele Verarbeitung zu ermöglichen.

## Map-Reduce

Verarbeitung eines Datensatzes X mithilfe eines MapReduce-Programms P, um ein Ergebnis Y zu berechnen.

Das Programm P besteht aus zwei Teilprogrammen:

- Map-Task M
- Reduce-Task R

Bring die Berechnung zu den Daten, nicht die Daten zur Berechnung.

## Algorithmus (vereinfacht)

1. Identifiziere alle Clusterknoten C, auf denen Teile des Datensatzes X gespeichert sind (Datenlokalität).
2. Führe den Map-Task M auf allen Knoten C aus.
3. Der JobTracker koordiniert die Ergebnisse der Map-Phase (inkl. Shuffle- und Sort-Phase).
4. Führe die Reduce-Tasks R auf ausgewählten Knoten aus.
5. Der JobTracker koordiniert die Ergebnisse der Reduce-Phase, die das Endergebnis Y bilden.

### Zählen von Wortvorkommen in Texten

- Die folgende Implementierung zeigt ein Python-Programm, das Texte zeilenweise aus einer Datei einliest.
- Daraus wird eine Liste einzelner Wörter erzeugt, jeweils zusammen mit der Anzahl ihrer Vorkommen (Termfrequenz).
- Das Programm nutzt den MapReduce-Algorithmus.
- Zur Simulation mehrerer Knoten werden Threads verwendet.

## Zählen von Wortvorkommen in Texten

```
# Path to your text file
file_path = 'short Richtlinien_CanSat-2023-2024.txt'

# Number of map and reduce threads
num_map_threads = 3
num_reduce_threads = 2

# Running MapReduce
word_counts = map_reduce(file_path, num_map_threads, num_reduce_threads)
print(word_counts)
```

## Zählen von Wortvorkommen in Texten

```
36 # Main Function
37 def map_reduce(file_path, num_map_threads, num_reduce_threads):
38     # Queue for Map Phase
39     mapped_values = Queue()
40
41     # Reading texts from file
42     with open(file_path, 'r') as file:
43         texts = file.readlines()
44
45     # Splitting texts for multiple map threads
46     splitted_texts = split_data(texts, num_map_threads)
47
48     # Start Map Phase in multiple Threads
49     map_threads = []
50     for i in range(num_map_threads):
51         thread = threading.Thread(target=map_worker, args=(splitted_texts[i], mapped_values))
52         map_threads.append(thread)
53         thread.start()
54
55     # Wait for all map threads to finish
56     for thread in map_threads:
57         thread.join()
58
59     # Shuffle and Sort Phase
60     grouped_values = shuffle_and_sort(mapped_values)
61
```

This is the main function that orchestrates the MapReduce process. It sets up threads for the map phase, collects and groups their outputs, and then sets up a thread for the reduce phase.

## Zählen von Wortvorkommen in Texten

```
62      # Splitting grouped_values for multiple reduce threads
63      items = list(grouped_values.items())
64      splitted_grouped_values = split_data(items, num_reduce_threads)
65
66      # Queue for Reduce Phase
67      result_queue = Queue()
68
69      # Start Reduce Phase in multiple Threads
70      reduce_threads = []
71      for i in range(num_reduce_threads):
72          thread = threading.Thread(target=reduce_worker, args=(dict(splitted_grouped_values[i]), result_queue))
73          reduce_threads.append(thread)
74          thread.start()
75
76      # Wait for all reduce threads to finish
77      for thread in reduce_threads:
78          thread.join()
79
80      # Collect and Return Results
81      results = {}
82      while not result_queue.empty():
83          key, value = result_queue.get()
84          results[key] = value
85
86      # Sort results by frequency (value)
87      sorted_results = sorted(results.items(), key=lambda x: x[1], reverse=True)
88
89      return sorted_results
90
```

## Zählen von Wortvorkommen in Texten

```

1 import threading
2 from collections import defaultdict
3 from queue import Queue
4
5 # Map Function
6 def map_function(text, queue):
7     for word in text.split():
8         queue.put((word, 1))
9
10 # Thread Worker for Map Phase
11 def map_worker(texts, queue):
12     for text in texts:
13         map_function(text, queue)
14
15 # Shuffle and Sort Function
16 def shuffle_and_sort(mapped_values):
17     grouped_values = defaultdict(list)
18     while not mapped_values.empty():
19         key, value = mapped_values.get()
20         grouped_values[key].append(value)
21     return grouped_values
22
23 # Reduce Function
24 def reduce_function(grouped_values, result_queue):
25     for key, values in grouped_values.items():
26         result_queue.put((key, sum(values)))
27
28 # Thread Worker for Reduce Phase
29 def reduce_worker(grouped_values, result_queue):
30     reduce_function(grouped_values, result_queue)
31
32 # Helper function to split data into chunks
33 def split_data(data, chunks):
34     return [data[i::chunks] for i in range(chunks)]

```

This function takes a string of text and a queue as input. It splits the text into words and puts each word into the queue as a tuple (word, 1). This tuple represents the word (as a key) and a count of 1 (as a value).

This function serves as a worker for the map phase in a threaded environment. It processes a list of texts, using the map\_function on each text.

This function simulates the "shuffle and sort" phase of MapReduce. It takes a queue of mapped values (key-value pairs) and groups them by key.

This function performs the "reduce" operation. It processes each group of values (counts) for each key (word) and combines them to produce a single output per key.

The reduce\_worker function is a worker function designed to be run in a separate thread during the reduce phase of the MapReduce process.

The split\_data function is used to divide the data into smaller chunks for parallel processing in the map phase.

## Zählen von Wortvorkommen in Texten

```
[('der', 23), ('und', 18), ('die', 15), ('eine', 10), ('zu', 10), ('von', 9), ('einer', 8), ('ist', 7), ('Teams', 7), ('in', 6), ('wird', 5), ('CanSat', 5), ('Die', 5), ('den', 5), ('Sekundärmision', 4), ('ein', 4), ('einem', 4), ('mi', 4), ('an', 3), ('oder', 3), ('für', 3), ('Mission', 3), ('des', 3), ('Der', 3), ('Schüler:innen', 2), ('Missio', 2), ('Rakete', 2), ('entwickeln', 2), ('soll', 2), ('(z.B.', 2), ('werden.', 2), ('einzelnen', 2), ('Dabei', 2), ('nach', 2), ('auch', 2), ('ihre', 2), ('Primärmision', 2), ('Daten', 2), ('Das', 2), ('Getränkendose', 2), ('umfass', 2), ('Höhe', 2), ('dann', 2), ('Komponenten', 2), ('Wettbewerb', 2), ('einen', 2), ('das', 2), ('Bergungssystem', 2), ('Projekt', 2), ('sowie', 2), ('sollte', 2), ('echten', 2), ('ESA', 2), ('Größe', 2), ('dem', 2), ('CanSats', 2), ('durchführen.', 2), ('gebaut', 1), ('programmiert', 1), ('Dieser', 1), ('vorgeschrriebene', 1), ('dieselben', 1), ('d', 1), ('urchlaufen', 1), ('Planung', 1), ('Ergebnisse.', 1), ('diese', 1), ('bietet', 1), ('Raumfahrtbranche.', 1), ('Highlight', 1), ('Startkampagne', 1), ('Ende', 1), ('Zusätzlich', 1), ('Teilnehmer:innen', 1), ('Woche', 1), ('1:', 1), ('Idee', 1), ('Dies', 1), ('unter', 1), ('Auswahl', 1), ('Feststoffrakete', 1), ('700m', 1), ('Finanzplanung', 1), ('Durch', 1), ('Öffentlichkeitsarbeit', 1), ('Teilnehmenden', 1), ('Aus', 1), ('gewonnenen', 1), ('wissenschaftlichen', 1), ('Einfallsreichtum', 1), ('Lauf', 1), ('Motivation', 1), ('sekundäre', 1), ('dargelegt', 1), ('bisherigen', 1), ('au', 1), ('hundert', 1), ('gebracht', 1), ('Konstruktion', 1), ('Satelliten,', 1), ('naturwissenschaftlich-interessier', 1), ('Deutsche', 1), ('seit', 1), ('jährlich', 1), ('Hinter', 1), ('fünf', 1), ('„City', 1), ('exklusiven', 1), ...]
```

## Zählen von Wortvorkommen in Texten

```
1 public static class TokenizerMapper
2     extends Mapper<Object, Text, Text, IntWritable>{
3
4     private final static IntWritable one = new IntWritable(1);
5     private Text word = new Text();
6
7     public void map(Object key, Text value, Context context
8                     ) throws IOException, InterruptedException {
9         StringTokenizer itr = new StringTokenizer(value.toString());
10        while (itr.hasMoreTokens()) {
11            word.set(itr.nextToken());
12            context.write(word, one);
13        }
14    }
15 }
```

The map function reads input key-value pairs and processes them to generate intermediate key-value pairs. For word count, the input value is a line of text, and the map function emits each word as a key, with the number 1 as the value.

## Zählen von Wortvorkommen in Texten

```
1 public static class IntSumReducer
2     extends Reducer<Text, IntWritable, Text, IntWritable> {
3     private IntWritable result = new IntWritable();
4
5     public void reduce(Text key, Iterable<IntWritable> values,
6                         Context context
7                     ) throws IOException, InterruptedException {
8         int sum = 0;
9         for (IntWritable val : values) {
10             sum += val.get();
11         }
12         result.set(sum);
13         context.write(key, result);
14     }
15 }
16 }
```

The reduce function accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. In the case of the word count, the reduce function adds up the counts for each word.

## Zählen von Wortvorkommen in Texten

```
1 public static void main(String[] args) throws Exception {  
2     Configuration conf = new Configuration();  
3     Job job = Job.getInstance(conf, "word count");  
4     job.setJarByClass(WordCount.class);  
5     job.setMapperClass(TokenizerMapper.class);  
6     job.setCombinerClass(IntSumReducer.class);  
7     job.setReducerClass(IntSumReducer.class);  
8     job.setOutputKeyClass(Text.class);  
9     job.setOutputValueClass(IntWritable.class);  
10    FileInputFormat.addInputPath(job, new Path(args[0]));  
11    FileOutputFormat.setOutputPath(job, new Path(args[1]));  
12    System.exit(job.waitForCompletion(true) ? 0 : 1);  
13 }
```

```
hadoop jar wordcount.jar org.myorg.WordCount /input /output
```

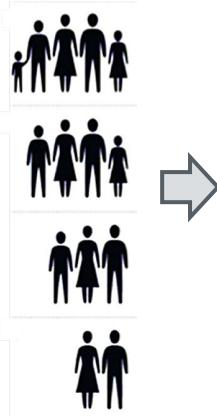
## Map Reduce Übung

- Implementiere eine Sortierung von Wörtern unter Verwendung des MapReduce-Algorithmus.

## Weitere Beispiele Map Reduce

## Hadoop Ecosystem – MapReduce Example

- Big-Data-Fragestellung: Wie viele Filme hat jeder Benutzer im Datensatz bewertet?



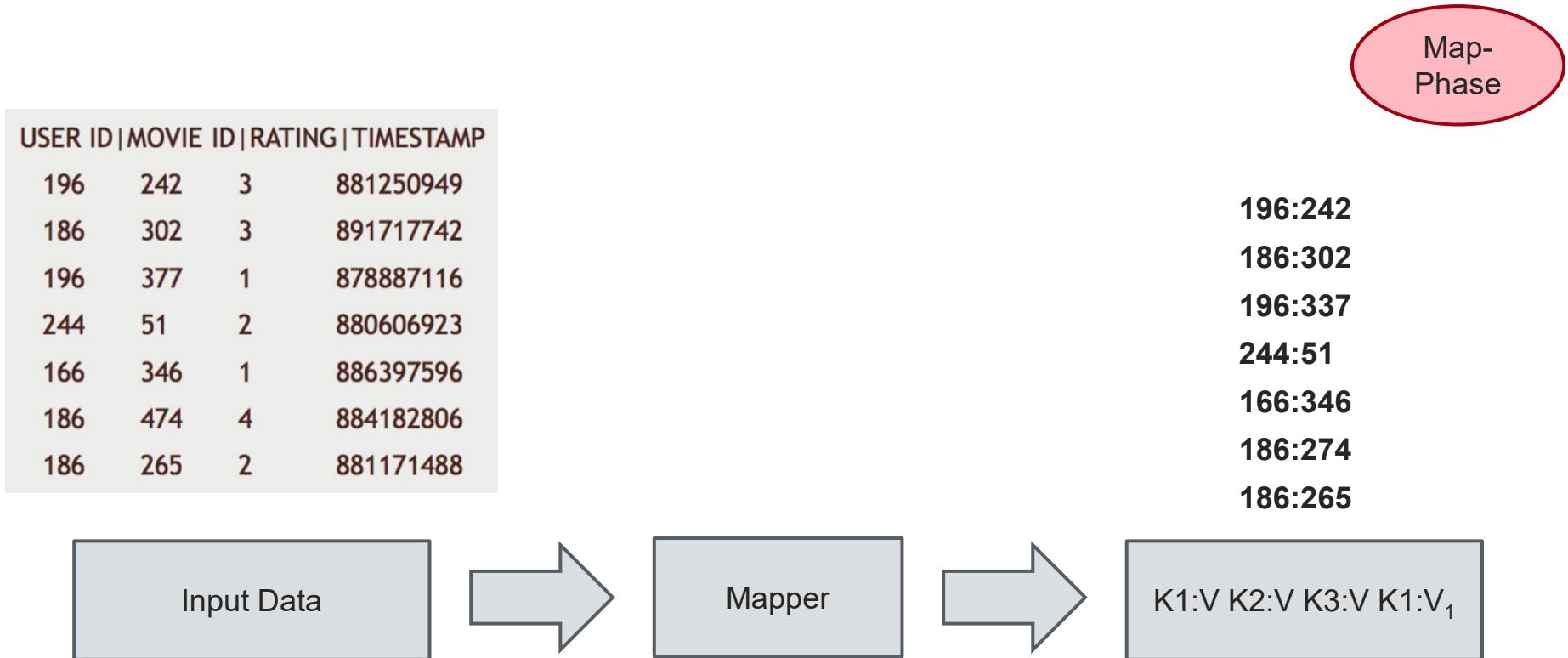
USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

- Kopieren und Ausführen des Sourcecodes für den Map-Task M auf allen Knoten C
- Map-Task M
  - Input: Eine Zeile aus dem Datensatz
  - Verarbeitung: Extraktion eines KEY:VALUE-Paars aus den Inputdaten
  - Output: Ein KEY:VALUE Paar
  - Jeder Knoten verarbeitet nur die Daten von X, die auf ihm gespeichert sind

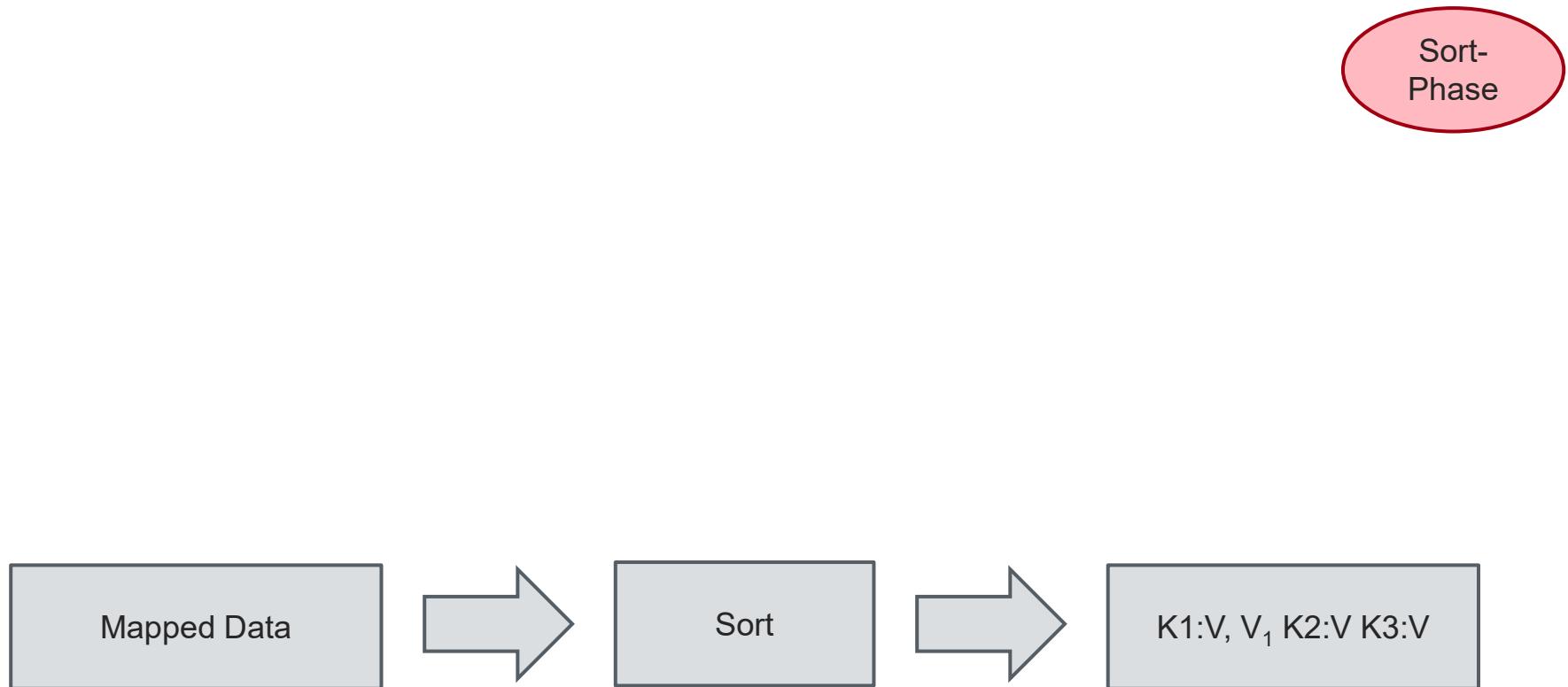
Map-Phase



## Extraktion des KEY:VALUE Paars “USER ID”：“MOVIE ID”



- Job Tracker aggregiert die Ergebnisse der Map-Phase (KEY:VALUE Paare) anhand des KEY



## Gruppieren der KEY:VALUE Paare nach KEY

Sort-Phase

196:242

186:302

196:337

244:51

166:346

186:274

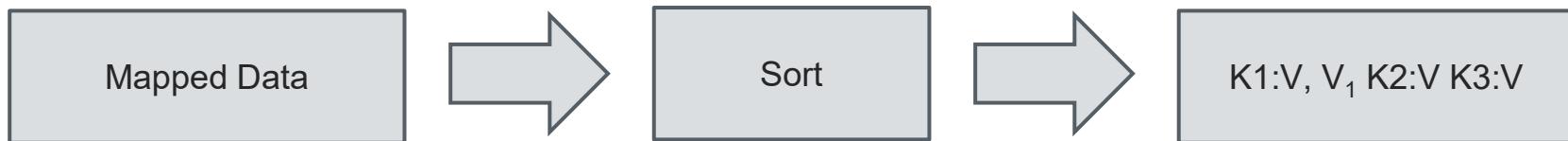
186:265

244:51

196:[242, 337]

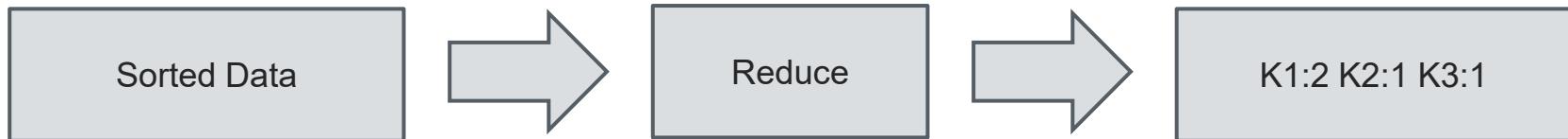
186:[302, 274, 265]

166:346



- Kopieren und Ausführen des Sourcecodes für den Reduce-Task R auf einem Teil der Knoten
- Reduce-Task R
  - Input: Ein KEY und eine Liste der dazugehörigen VALUES (KEY:VALUES)
  - Verarbeitung: Aggregierung der VALUES zu einem Ergebnis (VALUE)
  - Output: Ein KEY:VALUE Paar

Reduce-Phase

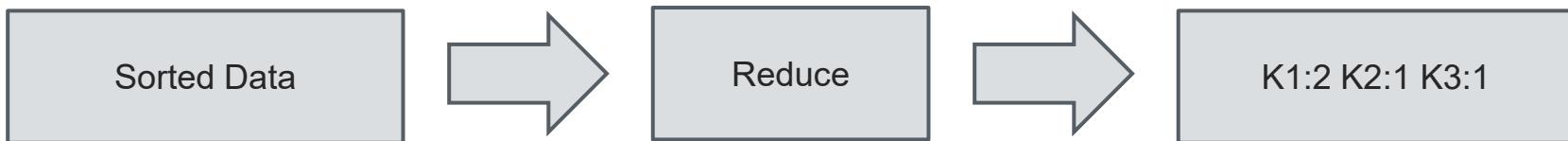


- Für jedes KEY:VALUES Paar wird die Anzahl der VALUES bestimmt und in Form eines neuen Paares KEY:ANZAHL(VALUES) gespeichert

Reduce-  
Phase

196:[242, 337]  
244:51  
166:346  
186:[302, 274, 265]

244:1  
196:2  
186:3  
166:1

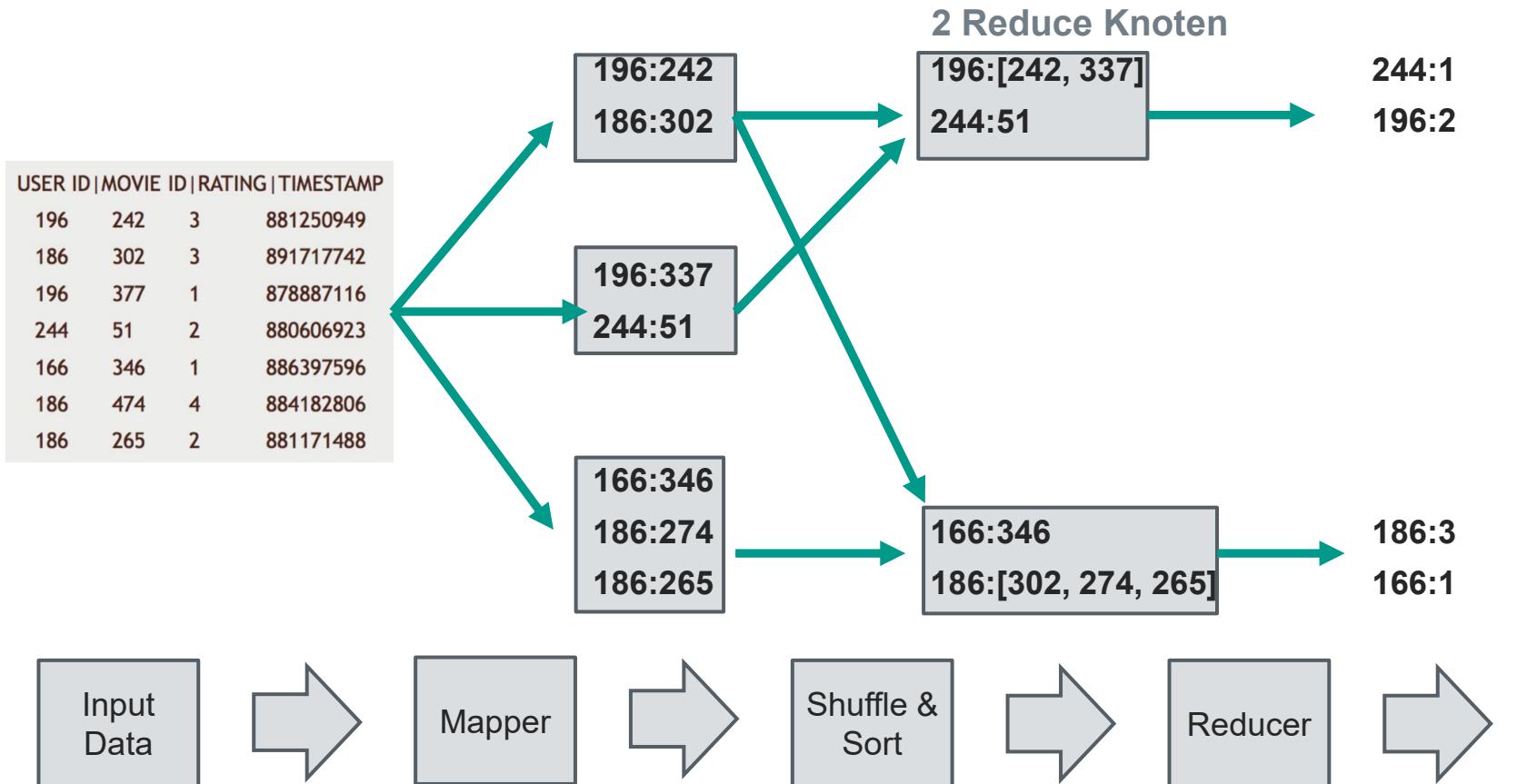


USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

<b>196:242</b>		
<b>186:302</b>		
<b>196:337</b>		
<b>244:51</b>	<b>196:[242, 337]</b>	<b>244:1</b>
<b>166:346</b>	<b>244:51</b>	<b>196:2</b>
<b>186:274</b>	<b>166:346</b>	<b>186:3</b>
<b>186:265</b>	<b>186:[302, 274, 265]</b>	<b>166:1</b>



### 3 MAP Knoten



- Weitere Big-Data-Fragestellung: Wie viele Bewertungen gibt es je Bewertungsstufe (Rating)?

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488



## Implementierung des Mappers

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

3:1  
3:1  
1:1  
2:1  
1:1  
4:1  
2:1

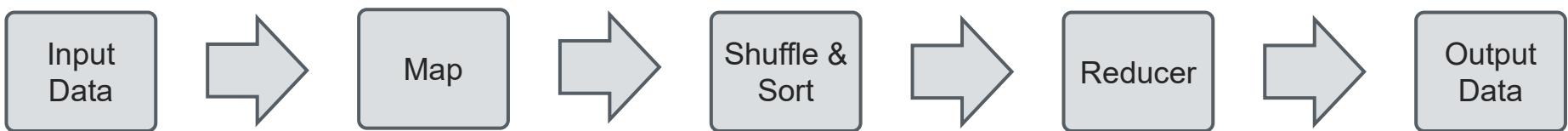


```
def mapper_get_ratings(self, _, line):
    (userID, movieID, rating, timestamp) = line.split('\t')
    yield rating, 1
```

## Sortieren

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

3:1  
3:1  
1:1  
2:1                  1:[1, 1]  
1:1                  2:[1, 1]  
4:1                  3:[1, 1]  
2:1                  4:1



## Implementierung des Reducers

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

3:1			
3:1			
1:1			
2:1	1:[1, 1]		1:2
1:1	2:[1, 1]		2:2
4:1	3:[1, 1]		3:2
2:1	4:1		4:1



```
def reducer_count_ratings(self, key, values):
    yield key, sum(values)
```

## Codesnippet

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield rating, 1

    def reducer_count_ratings(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    RatingsBreakdown.run()
```



# Big Data Analytics Apache Spark



## ApacheSpark

- „Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.“
- Open-Source In-Memory-Cluster-Computing-Framework, ursprünglich an der University of California, Berkeley entwickelt.
- Spark wurde mit dem Ziel entwickelt, MapReduce als Verarbeitungsmodell zu ersetzen (insbesondere wegen Performance und Flexibilität), kann jedoch komplementär zu Hadoop eingesetzt werden, z. B. mit HDFS als Speicher und YARN oder Kubernetes als Ressourcenmanager.
- Wichtig: Spark ersetzt nicht Hadoop insgesamt, sondern primär MapReduce.
- <https://spark.apache.org/>
- „The most widely-used engine for scalable computing  
Thousands of companies, including 80% of the Fortune 500, use Apache Spark™.  
Over 2,000 contributors to the open-source project from industry and academia.!“

## Batch- und Streaming-Daten

- Vereinheitlichte Verarbeitung von Batch-Daten und Echtzeit-Streams mit der bevorzugten Programmiersprache: Python, SQL, Scala, Java oder R.

## SQL-Analytics

- Ausführung schneller, verteilter ANSI-SQL-Abfragen für Dashboards und Ad-hoc-Reports.
- In vielen Anwendungsfällen schneller als klassische Data-Warehouse-Lösungen.

## Data Science im großen Maßstab

- Durchführung von Exploratory Data Analysis (EDA) auf Petabyte-großen Datensätzen, ohne auf Downsampling zurückgreifen zu müssen.

## Maschinelles Lernen

- Training von Machine-Learning-Algorithmen auf einem Laptop und nahtlose Skalierung desselben Codes auf fehlertolerante Cluster mit Tausenden von Knoten.

## “A fast and general engine for large-scale data processing“



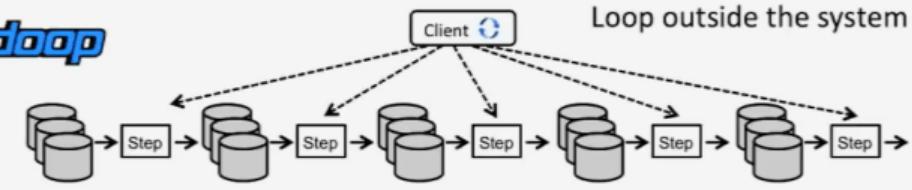
vs.



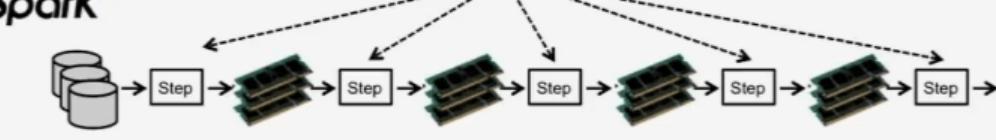
Fast
Batch processing
Stores data on disk
Java Codebase
Low on Cost
More line of codes (MR)
Batch processing
Data replication in HDFS

100x faster than MapReduce
Real-time processing
Stores data in memory
Scala Codebase
Higher costs
High-level APIs Java, Python, ...
Real-time processing
Resilient Distributed Datasets

- Spark wird häufig in Kombination mit Hadoop eingesetzt, wobei HDFS als Speicherschicht und YARN als Ressourcenmanager verwendet werden.

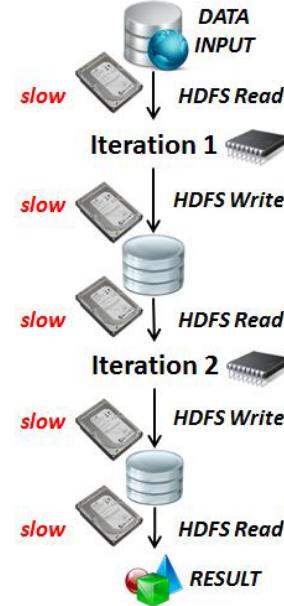


→ Move data through disk and network (HDFS)

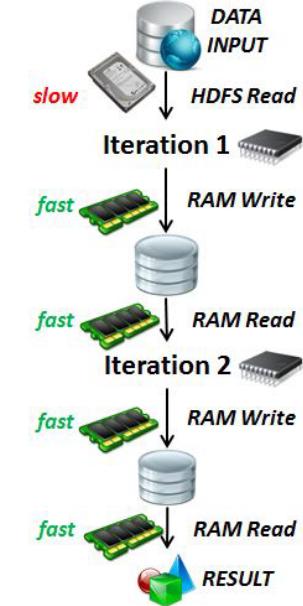


→ User can cache data in memory

### Apache Hadoop

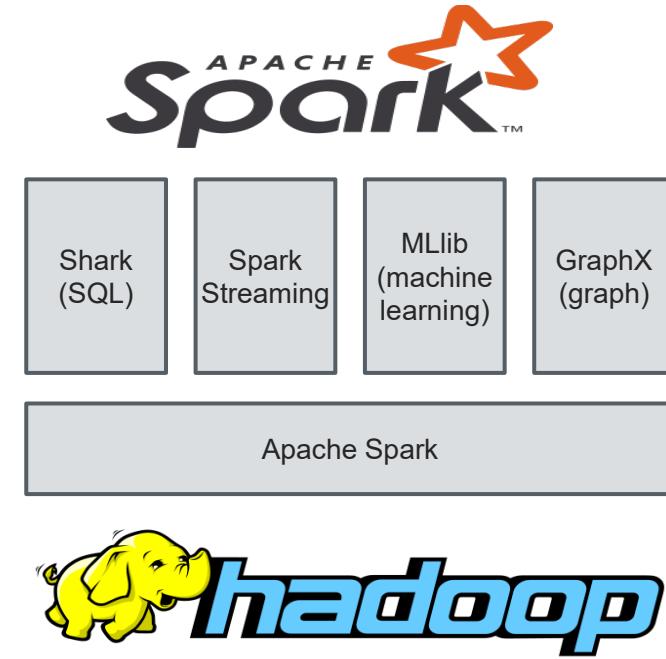


### Apache Spark



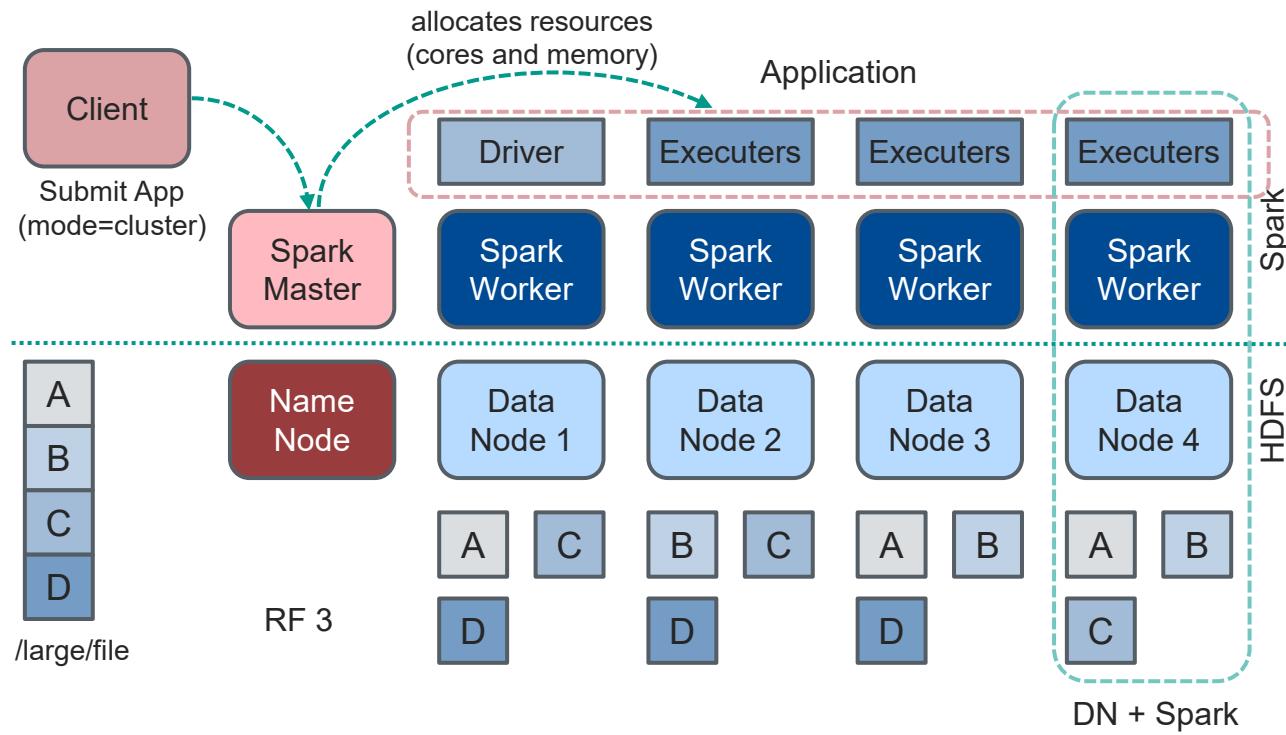
Bildquellen: <https://acadgild.com/blog/hadoop-vs-spark-best-big-data-frameworks/>

<http://www.big-data.tips/apache-spark-vs-hadoop>



Bildquelle: By Apache Software Foundation [Apache License 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>)], via Wikimedia Commons  
By Apache software foundation (<https://spark.apache.org/images/spark-logo.eps>) [Apache License 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>)], via Wikimedia Commons

Basiert auf einem zentralen Konzept: dem Resilient Distributed Dataset (RDD).



Bildquelle: <https://trongkhoanguyen.com/assets/post-images/2014/spark-hadoop.png>

- Diese Demo erstellt eine einfache Spark-Session, lädt einen Datensatz in ein DataFrame, führt eine Transformation durch und führt anschließend eine Aktion aus, um die Ergebnisse zu sammeln.

```
from pyspark.sql import SparkSession

# Initialize a SparkSession
spark = SparkSession.builder \
    .appName("PySpark Demo") \
    .getOrCreate()

# Sample data
data = [
    ('James', 'Smith', 'M', 30),
    ('Anna', 'Rose', 'F', 41),
    ('Robert', 'Williams', 'M', 62),
]

# Define schema of the data
columns = ["firstname", "lastname", "gender", "age"]

# Create DataFrame
df = spark.createDataFrame(data).toDF(*columns)

# Show the DataFrame
df.show()

# Transformation: Filter by age
filtered_df = df.filter(df.age > 40)

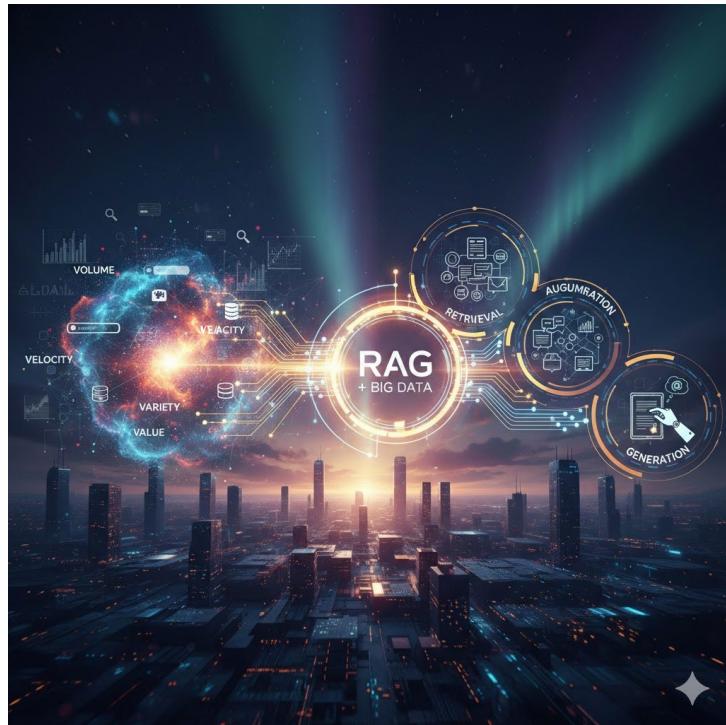
# Action: Collect the data
result = filtered_df.collect()

# Stop the SparkSession
spark.stop()

# Output the result
for row in result:
    print(row)
```

## Fallstudie Apache Spark

- Erkläre den Begriff “Resilient Distributed Datasets“ in Spark.
- Finde einen Anwendungsfall, in dem Apache Spark in Ihrem Unternehmen eingesetzt werden kann.



# Big Data Analytics RAG, LLMs und Big Data

## Retrieval-Augmented Generation (RAG)

kombiniert Large Language Models (LLMs) mit externen Wissensquellen.

Big Data liefert die skalierbare Grundlage:

- Datenmengen im TB/PB-Bereich
- Verteilte Speicherung (Data Lake, HDFS, Cloud Object Storage)
- Schnelle Suche & Filterung (z. B. Spark, SQL, Vektordatenbanken)

## Vektordatenbanken – Grundlagen

- Speichern numerische Vektoren (Embeddings)
- Vektoren repräsentieren Bedeutung von Texten, Bildern, etc.
- Ermöglichen Ähnlichkeitssuche statt exakter Schlüsselabfragen
- Zentrale Technologie für KI- und LLM-Anwendungen

- Daten → Embeddings (z. B. durch neuronale Netze)
- Speicherung im hochdimensionalen Vektorraum
- Suche über Distanzmaße (Kosinus, euklidisch)
- Nutzung spezieller Indexstrukturen (ANN, HNSW)

## Vektordatenbanken in RAG & Big Data

- Zentrale Komponente in Retrieval-Augmented Generation (RAG)
- Finden relevanter Dokumente für LLMs
- Kombination mit Data Lakes, Spark & Hadoop möglich
- Typische Systeme: FAISS, Milvus, Qdrant, Pinecone

## Architektur: RAG auf Big-Data-Basis

### 1. Datenhaltung:

- Data Lake (z. B. S3, ADLS, HDFS)
- Strukturierte & unstrukturierte Daten

### 2. Verarbeitung:

- Spark für ETL, Chunking, Embeddings

### 3. Retrieval:

- Vektordatenbank (z. B. FAISS, Milvus)

### 4. Generation:

- LLM erzeugt Antwort auf Basis der gefundenen Kontexte



## Typische Anwendungsfälle

- Unternehmens-Chatbots über große Dokumentbestände
- Wissensmanagement & semantische Suche
- Support- & Service-Automatisierung

## Mehrwert

- Aktuelle, überprüfbare Antworten
- Skalierbarkeit durch Big-Data-Technologien
- Trennung von Modell & Wissen (flexibel, kosteneffizient)