

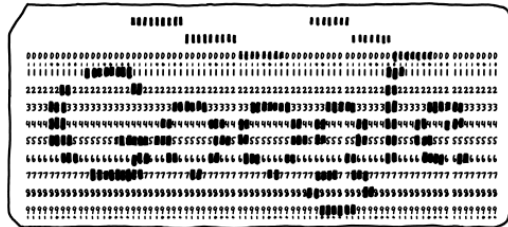
Kapitel 10: Big Data



Vorlesung
**CLOUD
COMPUTING**

Big Data

“If all digital data were stored on punch cards, how big would Google's data warehouse be?”



FOUR BOXES OF PUNCH
CARDS OUGHT TO BE
ENOUGH FOR ANYONE.



Quelle: <https://what-if.xkcd.com/63/>

<https://www.youtube.com/watch?v=I64CQp6z0Pk&t=275s> (Randall Munroe @ TED)

Big Data – was ist das überhaupt?

Charakteristische Eigenschaften:

- Die Größe des Datensatzes
- Die Komplexität des Datensatzes
- Die Technologien, die Verwendet werden, um den Datensatz zu verarbeiten

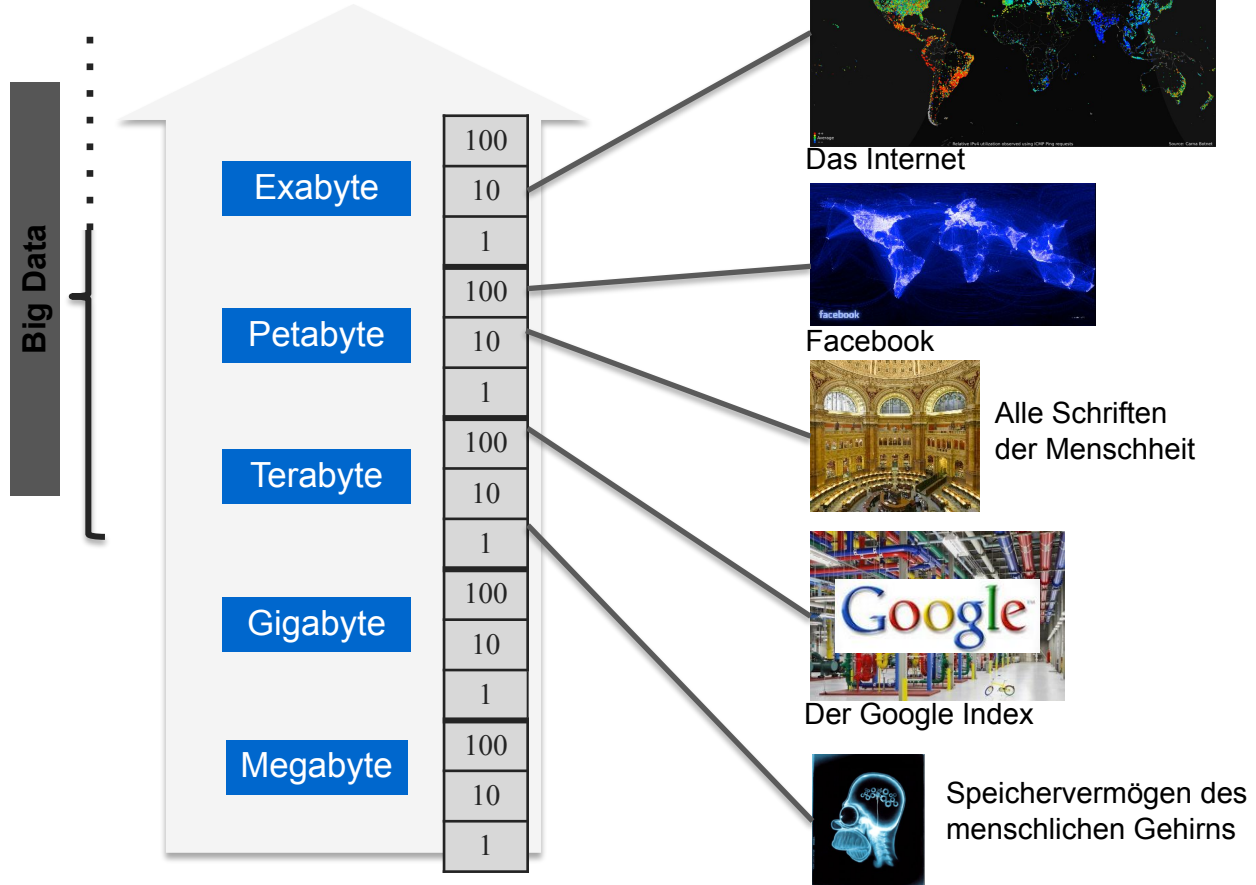
“Big data is a term describing the storage and analysis of large and or complex data sets using a series of techniques including, but not limited to: NoSQL, MapReduce and machine learning”

Quelle: . S. Ward und A. Barker. Undefined by data: a survey of big data definitions. arXiv preprint arXiv:1309.5821, 2013.




Big Data



Verarbeitung großer Datenmengen durch:

- verteilte und hochgradig parallelisierte Verarbeitung
- verteilte und effizient organisierte Datenablagen



- ANALYTICS & MACHINE INTELLIGENCE

APPLICATIONS - ENTERPRISE					
SALES          	MARKETING - B2B          	MARKETING - B2C          	CUSTOMER EXPERIENCE / SERVICE          	HUMAN CAPITAL          	

LEGAL	RETECH & COMPLIANCE	FINANCE	AUTOMATION & RPA	SECURITY
 RAVEL  CROSSER <i>Compliance</i>  BOSIS	 LegalSifter  Simpli Text	 Anaplan <i>2020</i>  YOUSOURCE  SAP Ariba	 UiPath  Marketplace  VEEVA  SAP Ariba  SAP Ariba  SAP Ariba	 Tenable  CyberArk  BlackBerry  Palo Alto Networks  Cisco Duo  Cisco Duo
 Pactera  Pactera  Pactera	 OneTrust  OneTrust  OneTrust	 Bluebeam  Bluebeam  Bluebeam	 UiPath  UiPath  UiPath	 Tenable  Tenable  Tenable

APPLICATIONS - INDUSTRY									
ADVERTISING Aspentech Modular Criteo IAS Santitas Altaba Gumgum Opster MobiData Tapar	EDUCATION Ludolph VVB Ucode Orchard Knewton Discovery Blackboard FutureLearn	REAL ESTATE Reform VVB Ucode Orchard Knewton Discovery Blackboard FutureLearn	GOV'T & INTELLIGENCE Palantir SPRINT Openbook Standard MARK43 Anduril AVAYA KENSIX AVAYA KENSIX AVAYA KENSIX	COMMERCE FAIRCODE STITCH FIX Affirm Modular ZESTIFY Capgemini JPMORGAN CHASE AVAYA KENSIX AVAYA KENSIX AVAYA KENSIX	FINANCE - LENDING Affirm Modular ZESTIFY Capgemini JPMORGAN CHASE AVAYA KENSIX AVAYA KENSIX AVAYA KENSIX	INSURANCE ROOT TRAVELERS BERKSHIRE HATHAWAY Capgemini JPMORGAN CHASE AVAYA KENSIX AVAYA KENSIX AVAYA KENSIX			

[illegible]

The banner displays a wide array of tools and frameworks used in data science and machine learning, organized into 13 distinct categories. Each category is represented by a header and a collection of logos for the relevant tools.

- FRAMEWORKS**: Includes logos for TensorFlow, PyTorch, Keras, and others.
- QUERY / DATA FLOW**: Includes logos for Apache Spark, Databricks, and others.
- DATA ACCESS & DATABASES**: Includes logos for PostgreSQL, MySQL, and others.
- ORCHESTRATION & PIPELINES**: Includes logos for Apache Airflow, Luigi, and others.
- STREAMING & MESSAGING**: Includes logos for Apache Kafka, Amazon Kinesis, and others.
- STAT TOOLS & LANGUAGES**: Includes logos for R, Python, and others.
- AI OPS & INFRA**: Includes logos for MLflow, DVC, and others.
- AI / MACHINE LEARNING / DEEP LEARNING**: Includes logos for TensorFlow, PyTorch, and others.
- SEARCH**: Includes logos for Elasticsearch, Solr, and others.
- LOGGING & MONITORING**: Includes logos for Prometheus, Grafana, and others.
- VISUALIZATION**: Includes logos for Tableau, Power BI, and others.
- COLLABORATION**: Includes logos for Jupyter, Databricks, and others.
- SECURITY**: Includes logos for AWS IAM, Azure AD, and others.

DATA SERVICES

- QUANTUMBLACK
- Booz | Allen | Hamilton
- Electrifi
- Fractal
- dataKind
- innocentius
- kaggle
- dataCamp
- DataFlix
- galvanize
- INSIGHT
- The Data Incubator

INCUBATORS & SCHOOLS

- FLORIANALBY
- GENERAL ASSEMBLY
- DATAFLEX
- DATAFLEX
- DATAFLEX

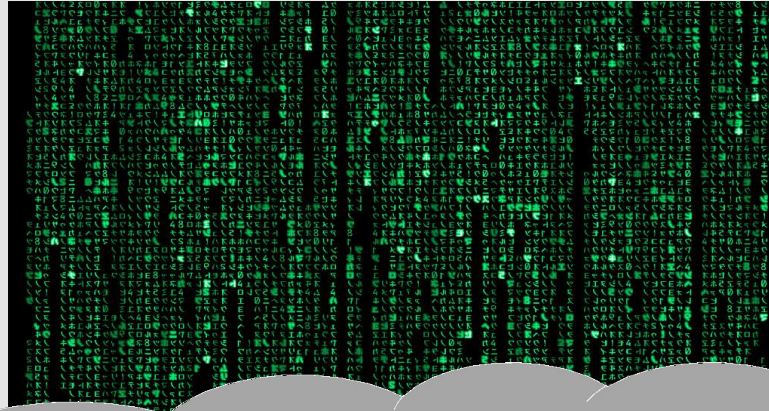
DATA RESOURCES

- OpenAI
- facebook research
- MIR
- VICTOR INSTITUTE
- AI2
- AI2
- AI2

RESEARCH

- OpenAI
- facebook research
- MIR
- VICTOR INSTITUTE
- AI2
- AI2
- AI2

Wie verwalte und erschließe ich große Datenmengen?



Die Cloud Computing Antwort:
Ich verteile sie auf viele Rechner
in der Cloud und schaffe eine
übergreifende
Zugriffsschnittstelle.



Große Datenmengen können effizient nur von parallelen Algorithmen verarbeitet werden.

Ein Algorithmus ist genau dann parallelisierbar, wenn er in einzelne Teile zerlegt werden kann, die keine Seiteneffekte zueinander haben.

- Funktioniert gut: Quicksort. Aufwand: $O(n \log n) \rightarrow n \times O(\log n)$

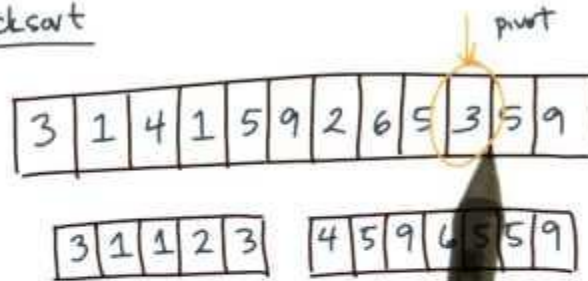
```
private void QuicksortParallel<T>(T[] arr, int left, int right)
where T : IComparable<T>
{
    if (right > left)
    {
        int pivot = Partition(arr, left, right);
        Parallel.Do(
            () => QuicksortParallel(arr, left, pivot - 1),
            () => QuicksortParallel(arr, pivot + 1, right));
    }
}
```

- Funktioniert nicht: Berechnung der Fibonacci-Folge ($F_{k+2} = F_k + F_{k+1}$). Berechnung ist nicht parallelisierbar.

Ein paralleler Algorithmus (Job) ist aufgeteilt in sequenzielle Berechnungsschritte (Tasks), die parallel zueinander abgearbeitet werden können. Der Entwurf von parallelen Algorithmen folgt oft dem Teile-und-Herrsche Prinzip.

Parallel Quicksort

qs-step



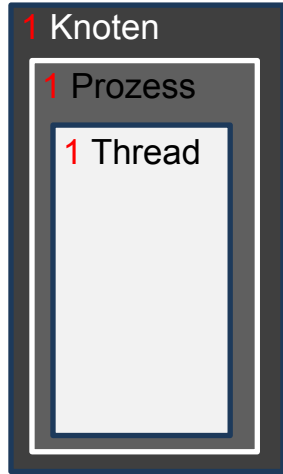
Parallele Programmierung basiert oft auf funktionaler Programmierung

- Ein funktionales Programm besteht (ausschließlich) aus Funktionen.
- Eine Funktion ist die Abbildung von Eingabedaten auf Ausgabedaten:
 $f(E) \rightarrow A$
Eine Funktion ändert die Eingabedaten dabei nicht.
- Funktionen sind idempotent:
 - Sie erzeugen neben den Ausgabedaten keine weiteren Seiteneffekte.
→ Funktionen sind somit ideal parallelisierbar und zur Beschreibung von Tasks geeignet.
- Sie erzeugen für die gleichen Eingabedaten auch stets die gleichen Ausgabedaten.
→ Funktionen können im Fehlerfall stets neu ausgeführt werden. Parallele Verarbeitung ist aus technischen Gründen oft fehleranfällig. Damit kann eine Fehlertoleranz sichergestellt werden.

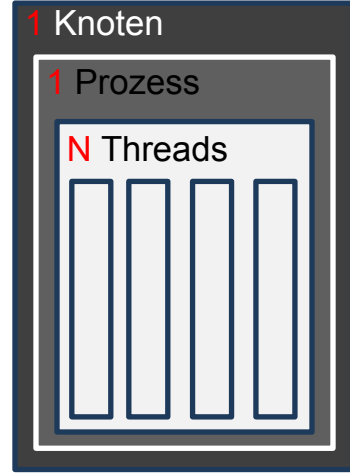
Beispiele:

- $f(x) = 2x$, also $1 \rightarrow 2$, $2 \rightarrow 4$, $3 \rightarrow 6$, ...
- Kombinationen:
 $g(x,y) = f(x) + f(y)$
- $h(x) = 1$ if x is even,
0 if x is odd
- ...

Parallele Programmierung kann sowohl im Kleinen als auch im Großen betrieben werden



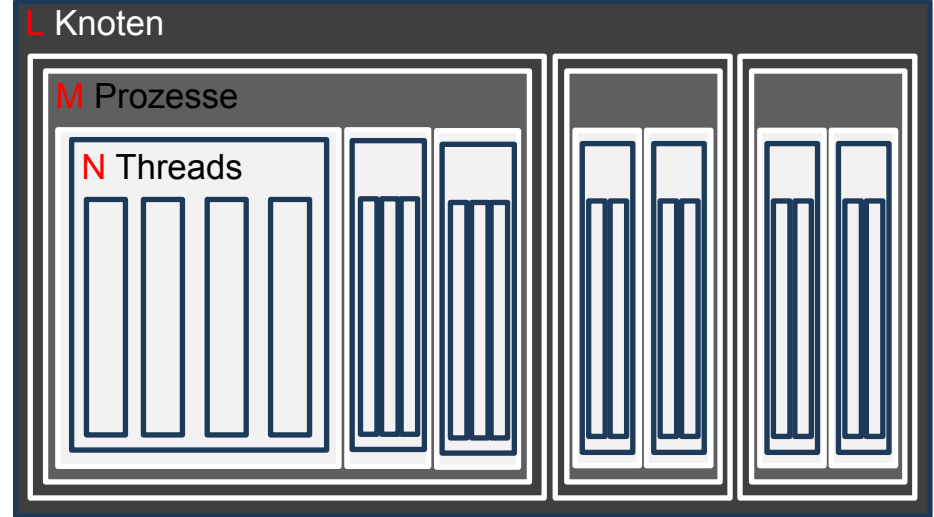
Keine
Parallelität



Parallelität im Kleinen

Vorteile im Vergleich:

- Höherer Durchsatz
- Bessere Auslastung der Hardware
- Vertikale Skalierung möglich



Parallelität im Großen

Vorteile im Vergleich:

- Höherer Durchsatz
- Horizontale Skalierung möglich (Scale Out).
- Keine hardwarebedingte Limitierung des Datenvolumens (→ Big Data ready).

Big Data erfordert Parallelität im Großen. Dabei muss man die vier Paradigmen der Parallelität im Großen beachten:



Folgt aus Datenmenge
im Vergleich zur Programmgröße

Das Grundprinzip von paralleler
Verarbeitung.

Folgt aus Praxisanforderung:
Viele Knoten
bedeutet
viele Ausfallmöglichkeiten

1. Die Logik folgt den Daten.

2. Falls Datentransfer notwendig, dann so schnell wie möglich:
In-Memory vor lokaler Festplatte vor Remote-Transfer.

3. Parallelisierung über *Tasks* (seiteneffektfreie Funktionen) und *Jobs*
(Ausführungsvorschrift für Tasks) sowie entsprechend partitionierter
Daten (*Shards*).

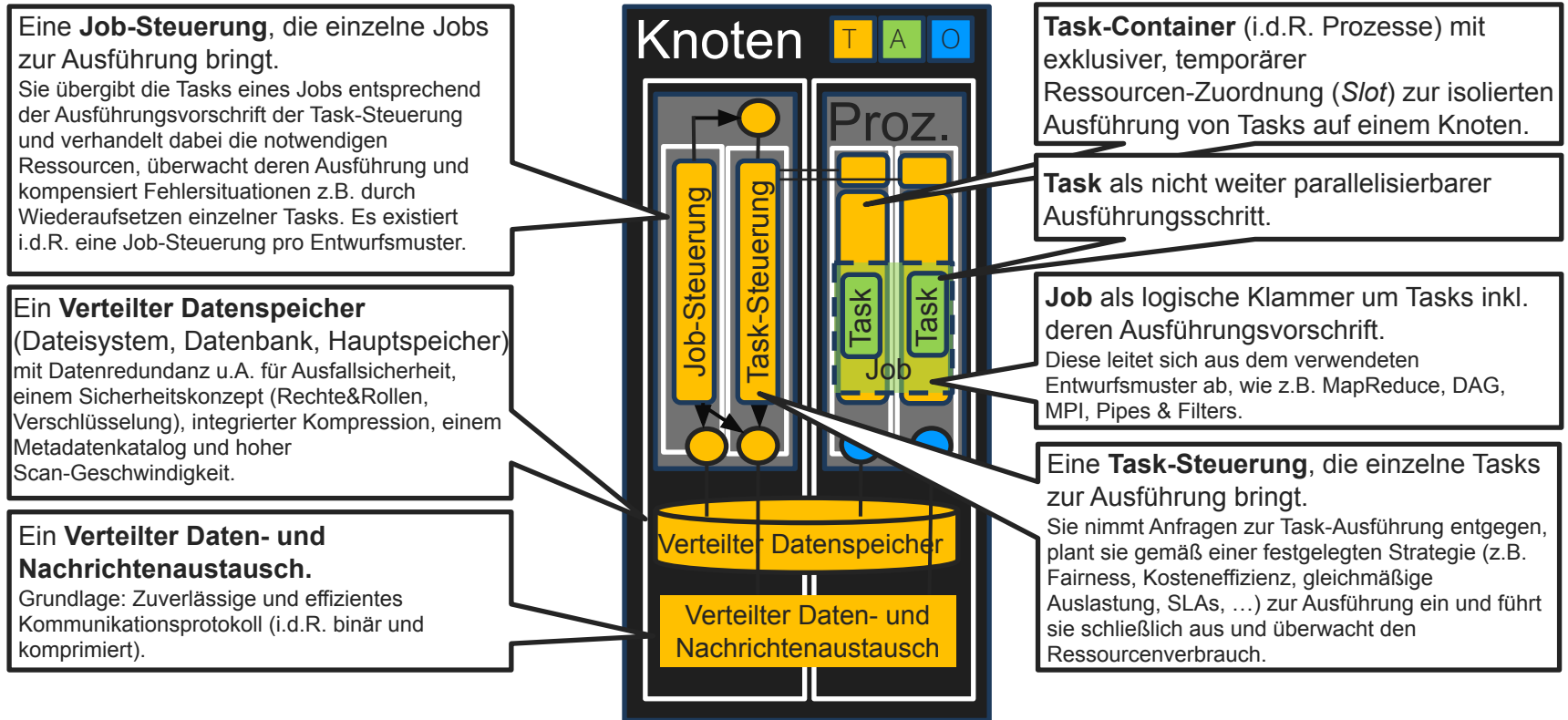
4. Design for Failure: Ausführungsfehler als Standardfall ansehen und
verzeihend und kompensierend sein.

Folgt aus potenziell großer
Datenmenge und
Verarbeitungs-geschwindigkeit

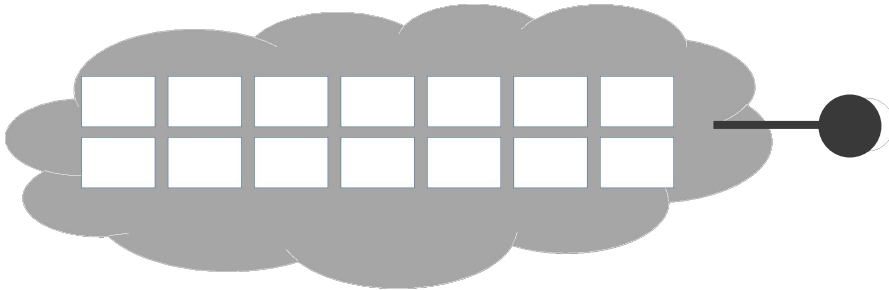
Notwendige Architekturkonzepte

1. Verteilung der Daten
2. Verteilung und Überwachung von Tasks
3. Aufteilung der Ressourcen
4. Entwurfsmuster zur Implementierung von Jobs

Eine Standardarchitektur für Parallelität im Großen



Welche Lösungen gibt es dafür im Cloud Computing?

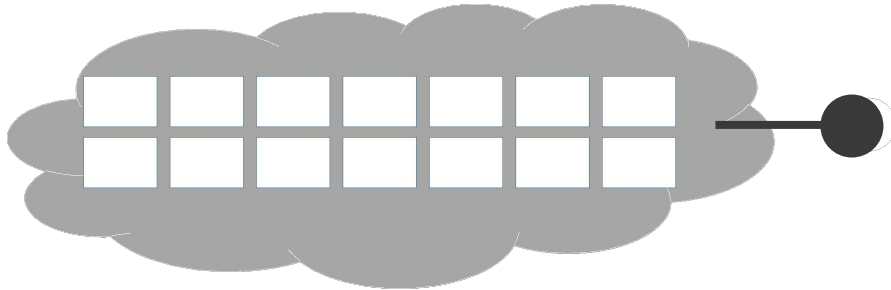


- **Big Data Engines (low level)**
 - MapReduce
 - RDD (Resilient Distributed Dataset)
- **Big Data Datenbanken (high level)**
 - NoSQL Datenbanken
 - NewSQL Datenbanken (NoSQL + SQL)
- Verteilte Dateisysteme
- In-Memory Data Grids / Elastic Memory

The background is a solid dark blue color. Overlaid on this background is a complex, abstract network of thin, light blue lines. These lines connect numerous small, light blue dots, creating a web-like pattern that resembles a data network or a molecular structure. The dots and lines are distributed across the entire frame, with some areas being more densely connected than others.

Big Data Engines

Welche Lösungen gibt es dafür im Cloud Computing?



- Big Data Engines (low level)
 - MapReduce
 - RDD (Resilient Distributed Dataset)
- Big Data Datenbanken (high level)
 - NoSQL Datenbanken
 - NewSQL Datenbanken (NoSQL + SQL)
- Verteilte Dateisysteme
- In-Memory Data Grids / Elastic Memory

MapReduce



<https://www.youtube.com/watch?v=cvhKoniK5Uo>

Die *map* und *reduce* Funktion.

- Die **map** Funktion: Transformation einer Menge von Datensätzen in eine Zwischendarstellung. Erzeugt aus einem Schlüssel und einem Wert eine Liste an Schlüssel-Wert-Paaren.

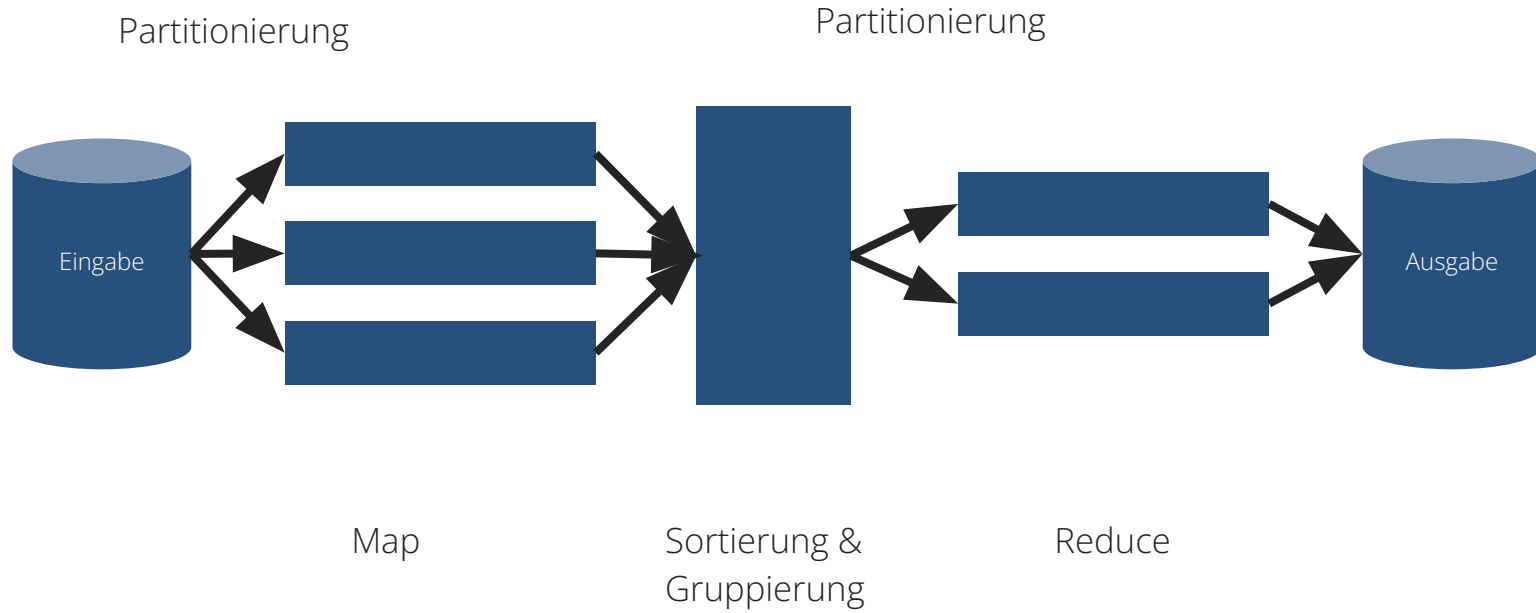
Signatur: **map**(k , v) \rightarrow `list(< k' , v' >)`

- Die **reduce** Funktion: Reduktion der Zwischendarstellung auf das Endergebnis. Verarbeitet alle Werte mit gleichem Schlüssel zu einer Liste an Schlüssel-Wert-Paaren.

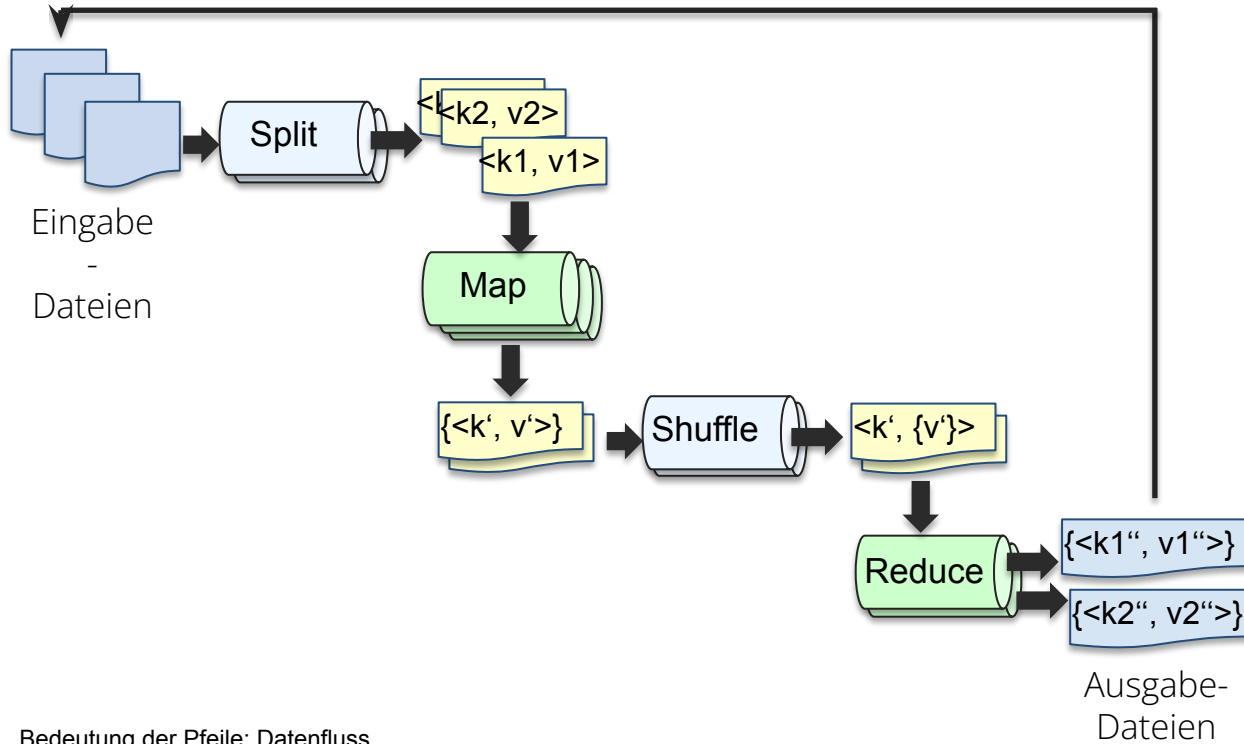
Signatur: **reduce**(k' , `list(v')`) \rightarrow `list(< k'' , v'' >)`

- Dabei soll gelten: $|\text{list}(<k'', v''>)| \ll |\text{list}(<k', v'>)|$

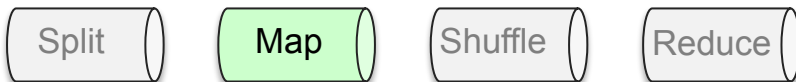
MapReduce Phasen



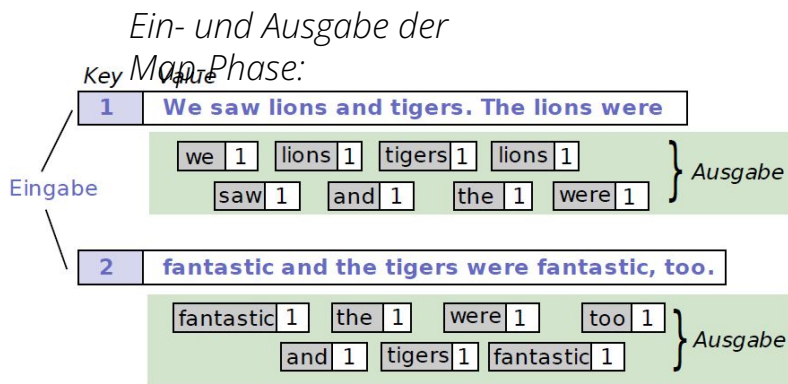
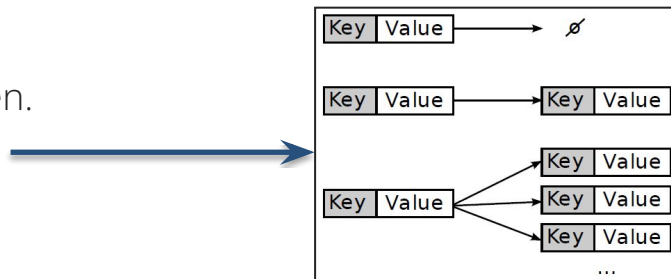
Programme werden in (mehrere) Map-Reduce-Zyklen aufgeteilt. Das Framework übernimmt die Parallelisierung.



Die Map-Phase



- Parallele Verarbeitung verschiedener Teilbereiche der Eingabedaten.
- Eingabedaten liegen in Form von Schlüssel/Wert-Paaren vor.
- Abbildung auf variable Anzahl von neuen Schlüssel/Wert-Paaren. Dabei sind alle Abbildungsvarianten zulässig:
- Beispiel: WordCount



Pseudocode
Map-Phase:

```
map(String key, String value):  
    //key: document name  
    //value: document contents  
    for each word in value:  
        EmitIntermediate(word, "1");
```

Die Shuffle-Phase

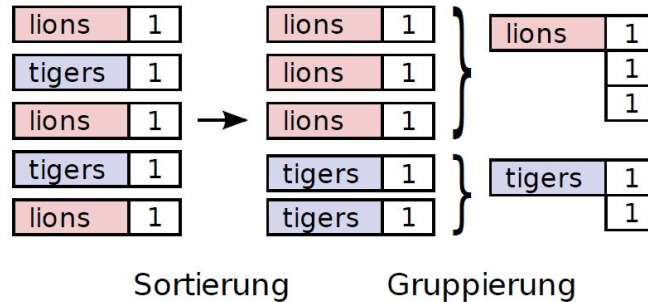
Split

Map

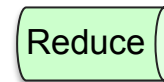
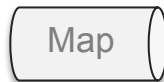
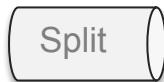
Shuffle

Reduce

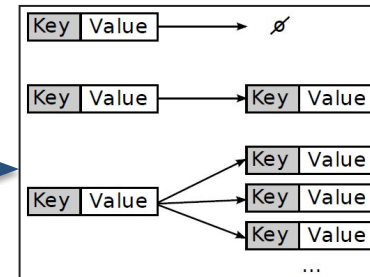
- Verarbeitung der Ergebnisse aus der Map-Phase.
- Ausgaben aus der Map-Phase werden entsprechend ihrem Schlüssel sortiert und gruppiert.
- Im Standard-Fall ist die Shuffle-Phase nicht parallelisiert.
- Sie kann jedoch mittels einer Vor-Sortierung in der Map-Phase über eine Partitionierungsfunktion (z.B. Hash) auf den Schlüssel parallelisiert werden.



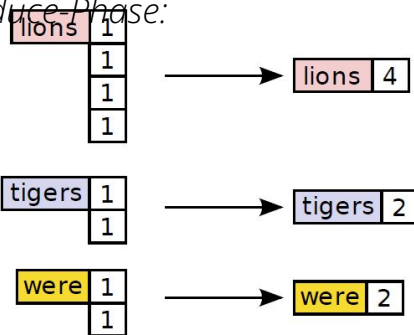
Die Reduce-Phase



- Parallele Verarbeitung von Ergebnis-Gruppen aus der Map-Phase.
Es wird pro Reduce-Vorgang genau eine dieser Gruppen verarbeitet.
- Eingabedaten liegen in Form von Schlüssel-Wertlisten vor.
- Abbildung auf variable Anzahl an Schlüssel/Wert-Paaren.
Dabei sind alle Abbildungsvarianten zulässig:



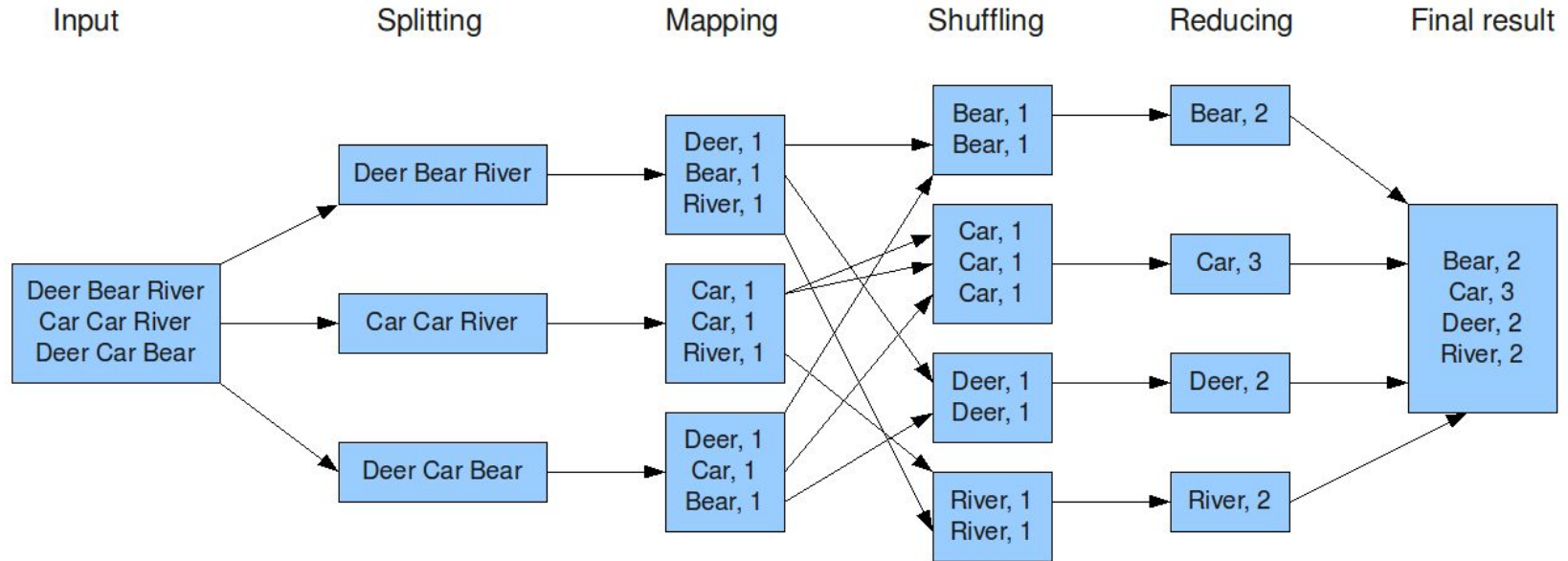
Ein- und Ausgabe der
Reduce-Phase:



Pseudocode

```
reduce(String key, Iterator values):  
    //key: a word  
    //values: a list of counts  
    for each value in values:  
        result += ParseInt(value);  
    Emit(AsString(Key +", "+result));
```


Übersicht über alle Phasen



<http://blog.iteam.nl/2009/08/04/introduction-to-hadoop>

Anwendungsbeispiele für MapReduce (1/2)

Verteilte Häufigkeitsanalyse

Wie häufig kommen welche Wörter in einem Text vor?

- **map**(Textfragment) \rightarrow $\langle \text{Wort}, 1 \rangle$: Erkennt einzelne Wörter im Textfragment.
- **reduce**($\langle \text{Wort}, \text{list}(1) \rangle$) \rightarrow $\langle \text{Wort}, \text{Anzahl} \rangle$: Zählt die Anzahl zusammen.

Verteiler regulärer Ausdruck

In welchen Zeilen eines Textes kommt ein Suchmuster vor?

- **map**(Textfragment) \rightarrow $\langle \text{Zeile}, 1 \rangle$: Findet das Suchmuster im Textfragment.
- **reduce**($\langle \text{Zeile}, \text{list}(1) \rangle$) \rightarrow $\langle \text{Zeile}, \text{Anzahl} \rangle$: Zählt pro Zeile die Anzahl zusammen.

Graph mit Seitenverweisen extrahieren

Welche Seiten verweisen aufeinander? Dies ist z.B. Grundlage für den PageRank-Algorithmus.

- **map**(Webseite) \rightarrow $\langle \text{Ziel}, \text{Quelle} \rangle$: Findet für die Quelle einzelne Verweise auf Ziel-Seiten.
- **reduce**($\langle \text{Ziel}, \text{list}(\text{Quelle}) \rangle$) \rightarrow $\langle \text{Ziel}, \text{set}(\text{Quelle}) \rangle$: Erzeugt eine Hyperkante und eliminiert doppelte Quellen pro Ziel.

Anwendungsbeispiele für MapReduce (2/2)

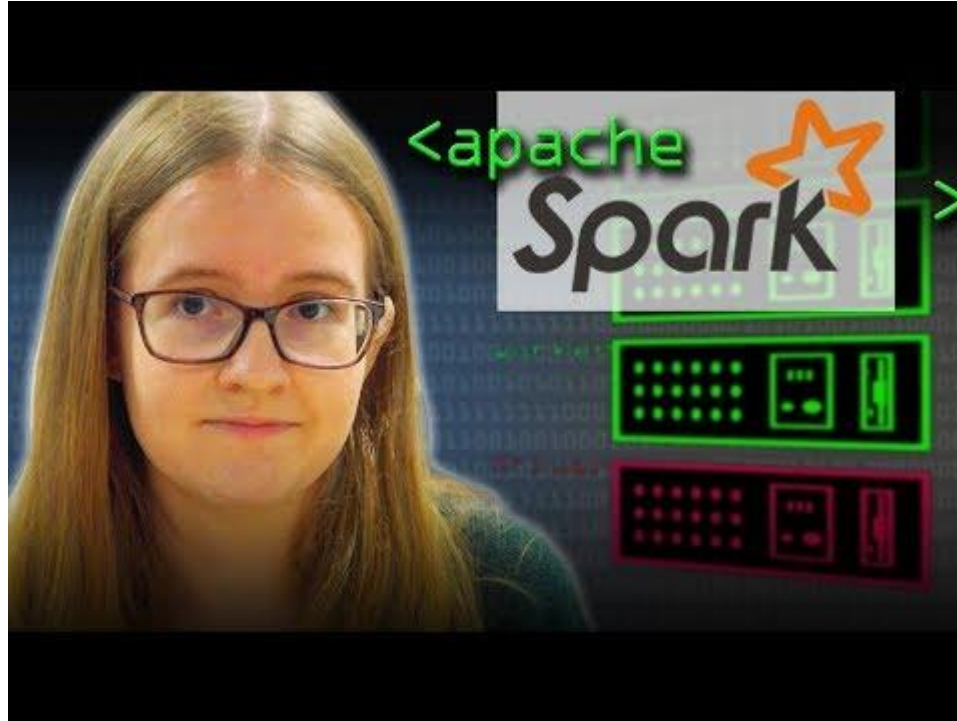
Weitere Beispiele:

- Dijkstra-Algorithmus (kürzester Pfad in einem Graphen):
<http://famousphil.com/blog/2011/06/a-hadoop-mapreduce-solution-to-dijkstra%E2%80%99s-algorithm/>
- Machine Learning Algorithmen: <http://mahout.apache.org>
- PageRank-Algorithmus: <http://www.cs.toronto.edu/~jasper/PageRankForMapReduceSmall.pdf>
- Allgemeine Graph-Algorithmen:
<http://www.adjoint-functors.net/su/web/354/references/graph-processing-w-mapreduce.pdf>
- Allgemeine Suche in Daten: <http://pig.apache.org>

Apache Spark



Resilient Distributed Dataset



<https://www.youtube.com/watch?v=tDVPcqGpEnM>

Die Resilient Distributed Dataset (RDD) Datenstruktur ist die Abstraktion des Spark Cores.

Eine RDD ist in der Außensicht ein klassischer Collection-Typ mit Transformations- und Aktionsmethoden.

RDD → RDD

Transformations

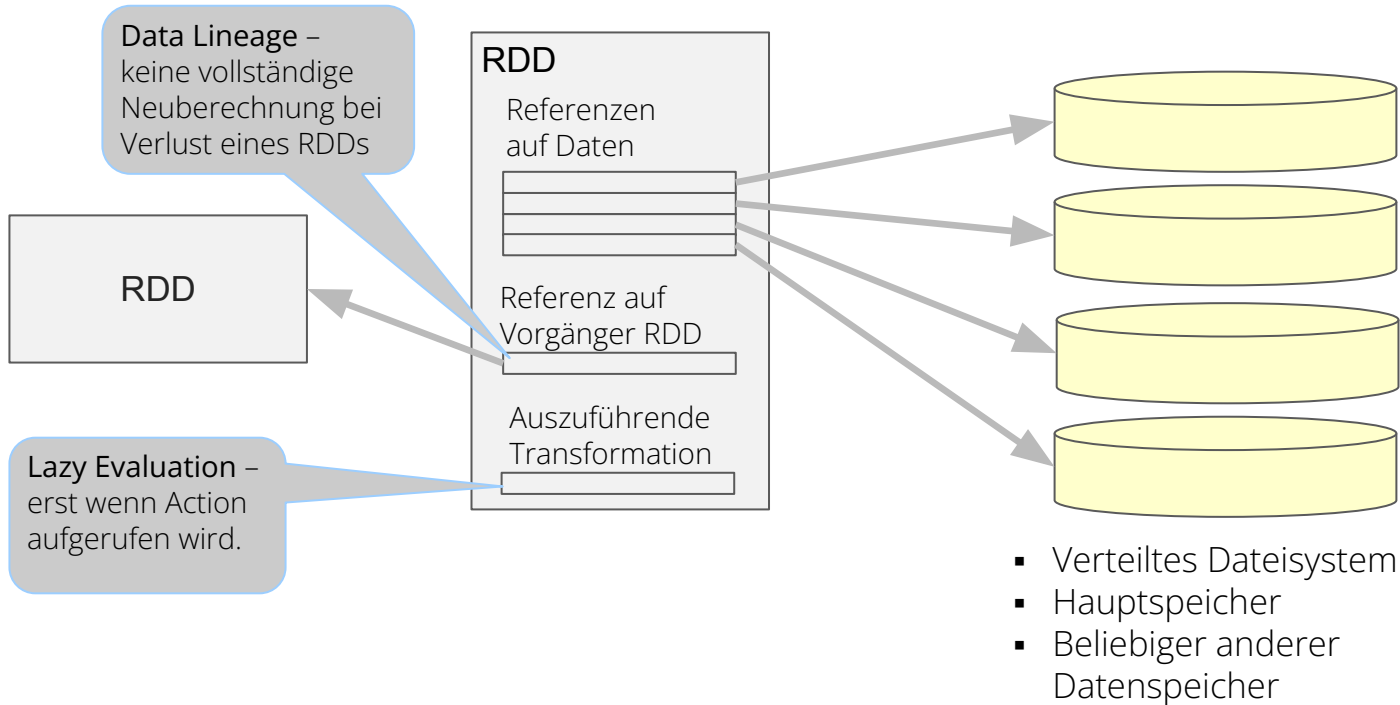
```
map(func)
flatMap(func)
filter(func)
groupByKey()
reduceByKey(func)
mapValues(func)
...
```

RDD → skalarer Typ, Collection, Storage

Actions

```
take(N)
count()
collect()
reduce(func)
takeOrdered(N)
top(N)
...
```

Die Anatomie eines RDDs



Daten mit Spark verarbeiten: Mehr als Map und Reduce

Filter

```
val numAs = logData.filter(line => line.contains("a")).count()
val numBs = logData.filter(line => line.contains("b")).count()
val numABs = logData.filter(line => line.contains("a"))
                    .filter(line => line.contains("b")).count()
```

Map

```
val lengths = logData.map(line => line.length)
```

Reduce

```
val maxLength = lengths.reduce(Math.max)
```

Sort

```
val sorted = logData.sortBy(l => l.length)
```

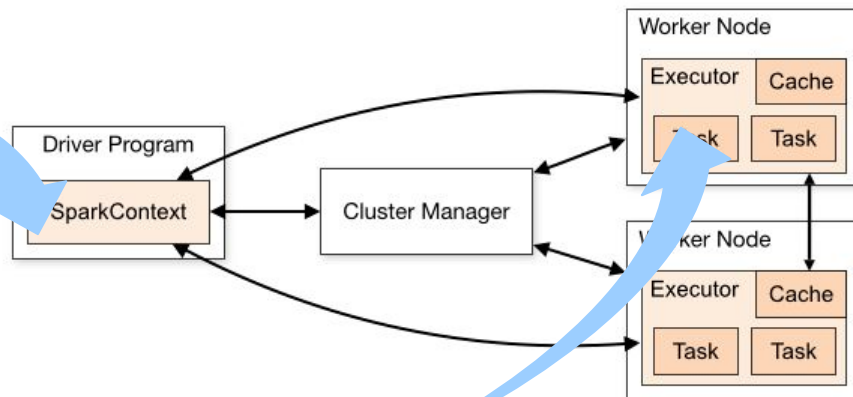
Transformations	Actions
<code>map(func)</code>	<code>take(N)</code>
<code>flatMap(func)</code>	<code>count()</code>
<code>filter(func)</code>	<code>collect()</code>
<code>groupByKey()</code>	<code>reduce(func)</code>
<code>reduceByKey(func)</code>	<code>takeOrdered(N)</code>
<code>mapValues(func)</code>	<code>top(N)</code>
...	...

Wie funktioniert das?

```
/* SimpleApp.scala */
```

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
```

```
object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.r
    val conf = new SparkConf().setAppName("
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).c
    val numAs = logData.filter(line => line.contains("a")).
    val numBs = logData.filter(line => line.contains("b")).
    println("Lines with a: %s, Lines with b: %s" format(num
  }
}
```



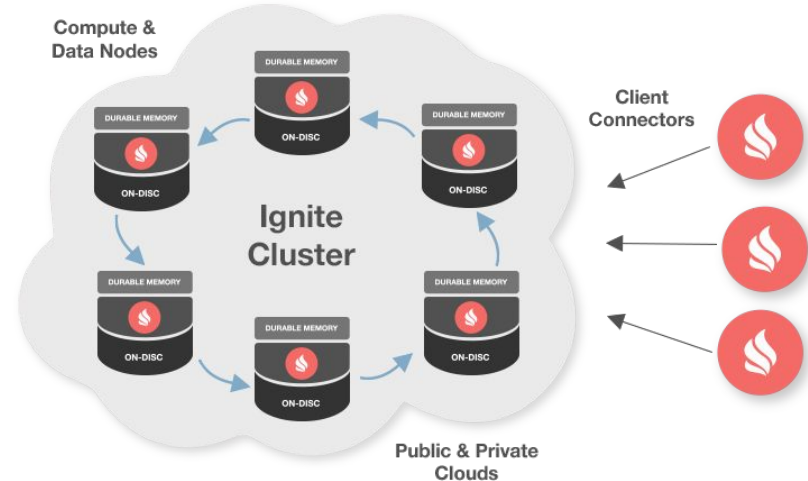
Apache Ignite



*“Distributed Database For
High-Performance Applications
With In-Memory Speed”*

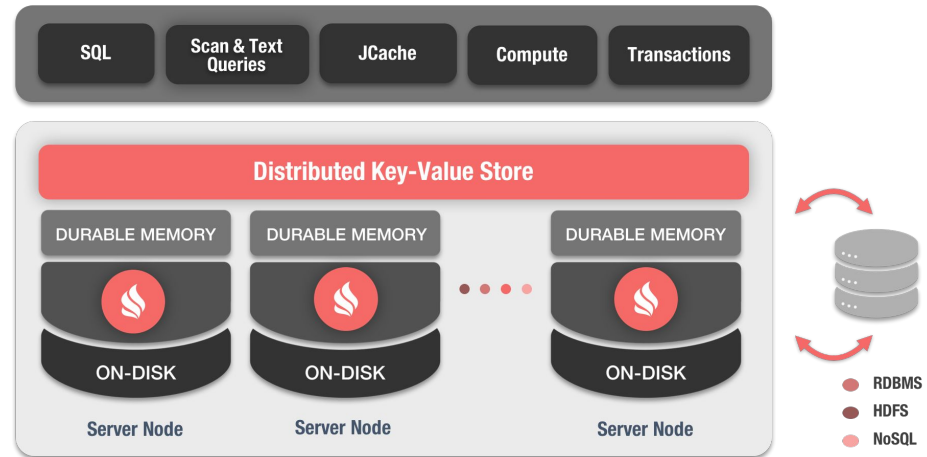
Apache Ignite

- Open-Source-Framework für In-Memory-Computing
- 2014 von GridGain vorgestellt, im selben Jahr ins Apache-Programm aufgenommen
- Hauptfeatures:
 - Distributed SQL
 - Distributed Key-Value Store
 - Collocated Processing
 - ACID Transactions
 - Machine Learning (Bingo!)



Ignite Data Grid

- In-Memory Key Value Store
- Implementiert die JCache-Spezifikation [**get()**, **put()**, **containsKey()**]
- Native Persistenz (=> Filesystem) vorhanden
- Eigene Storage-Provider möglich (z.B. SQL, MongoDB, ...)



Ignite Data Grid

Beispiel

```
Ignite ignite = Ignition.ignite();

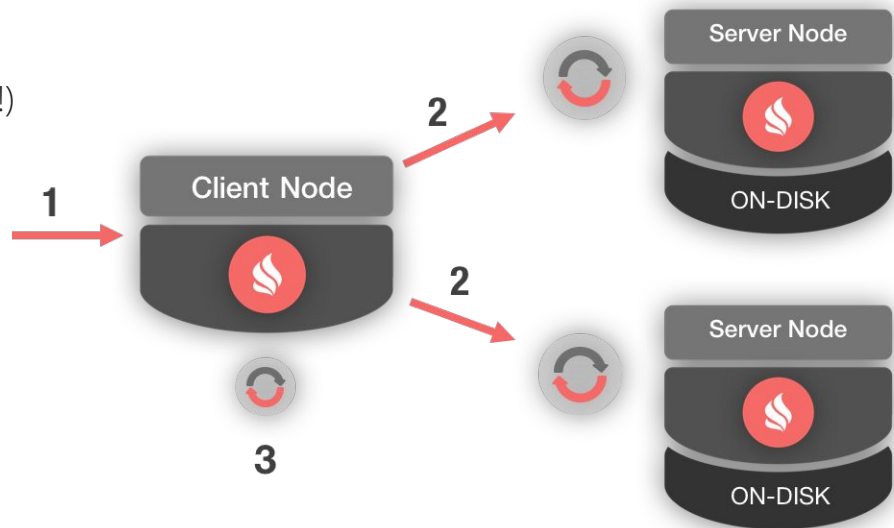
final IgniteCache<Integer, String> cache = ignite.cache("cacheName");

for (int i = 0; i < 10; i++) {
    cache.put(i, Integer.toString(i));
}

for (int i = 0; i < 10; i++) {
    Integer value = cache.get(i);
    System.out.println(value);
}
```


Ignite Compute

- Verteilte Verarbeitung von Daten
- Code wird zu den Daten gebracht (Performance!)
- Ähnliche Projekte:
 - Hadoop MapReduce
 - Apache Spark



- 1. Initial Request**
- 2. Co-located processing with data**
- 3. Reduce multiple results in one**

Ignite Compute Beispiel

```
final Ignite ignite = Ignition.ignite();

// Limit broadcast to remote nodes only.
IgniteCompute compute = ignite.compute(ignite.cluster().forServers());

// Print out hello message on remote nodes in the cluster group.
compute.broadcast(() ->
    System.out.println("Hello Node: " + ignite.cluster().localNode().id())
);
```

Apache Ignite

Compute - Map

```
List<String> words = Arrays.stream(arg.split(SEPARATOR_CHAR)).collect(Collectors.toList());
List<ComputeJob> jobs = new ArrayList<>(words.size());

for (String word : words) {

    ComputeJobAdapter adapter = new ComputeJobAdapter() {
        @Override
        public Object execute() throws IgniteException {
            Map<String, Integer> splitMap = new HashMap<>();
            splitMap.put(word, 1);
            return splitMap;
        }
    };
    jobs.add(adapter);
}

return jobs;
```

Apache Ignite

Compute - Reduce

```
Map<String, Integer> resultData = new TreeMap<>();

for (ComputeJobResult result : results) {
    Map<String, Integer> jobData = result.getData();
    for (Map.Entry<String, Integer> entry : jobData.entrySet()) {
        resultData.merge(entry.getKey(), entry.getValue(), (v1, v2) -> v1 + v2);
    }
}

return resultData;
```

Apache Ignite Streaming

- Manchmal ist der Datensatz so groß, dass er nicht im Ignite-Cluster Platz hat.
- Die Lösung: Streaming und Verarbeitung on the Fly!
 - “With Apache Ignite you can load and stream large finite — or never-ending — volumes of data in a scalable and fault-tolerant way into the cluster.”
- Beispiele:
 - Data Loading
 - Real-Time Data Streaming

Quelle: <https://ignite.apache.org/features/streaming.html>

Apache Ignite

Streaming - Beispiel

```
CacheConfiguration<String, String> configuration = new CacheConfiguration<>(CACHE_NAME);
configuration.setExpiryPolicyFactory(
    FactoryBuilder.factoryOf(new CreatedExpiryPolicy(new Duration(TimeUnit.SECONDS, 5)))
);

IgniteCache<String, String> streamCache = ignite.getOrCreateCache(config);

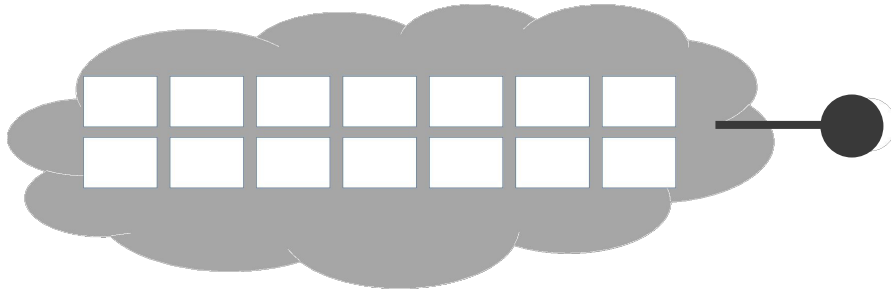
try (IgniteDataStreamer<String, String> streamer = ignite.dataStreamer(streamCache.getName())) {

    while(true) {
        String randomWord = RandomStringUtils.randomAlphanumeric(12);
        // Stream words into Ignite.
        streamer.addData(randomWord, randomWord);
    }
}
```



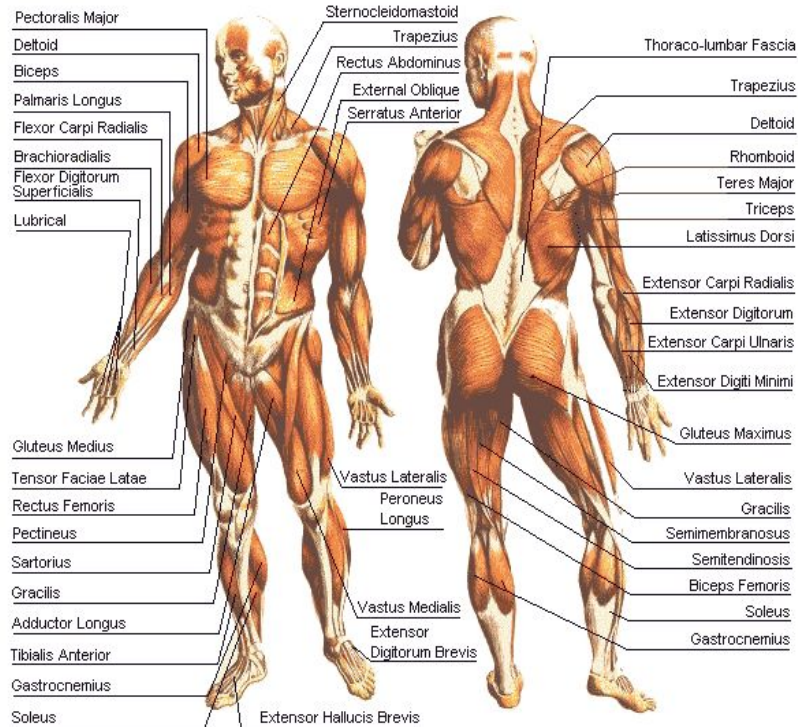
Big Data Datenbanken

Welche Lösungen gibt es dafür im Cloud Computing?



- Big Data Engines (low level)
 - MapReduce
 - RDD (Resilient Distributed Dataset)
- **Big Data Datenbanken (high level)**
 - NoSQL Datenbanken
 - NewSQL Datenbanken (NoSQL + SQL)
- Verteilte Dateisysteme
- In-Memory Data Grids / Elastic Memory

Die Anatomie von Big Data Datenbanken

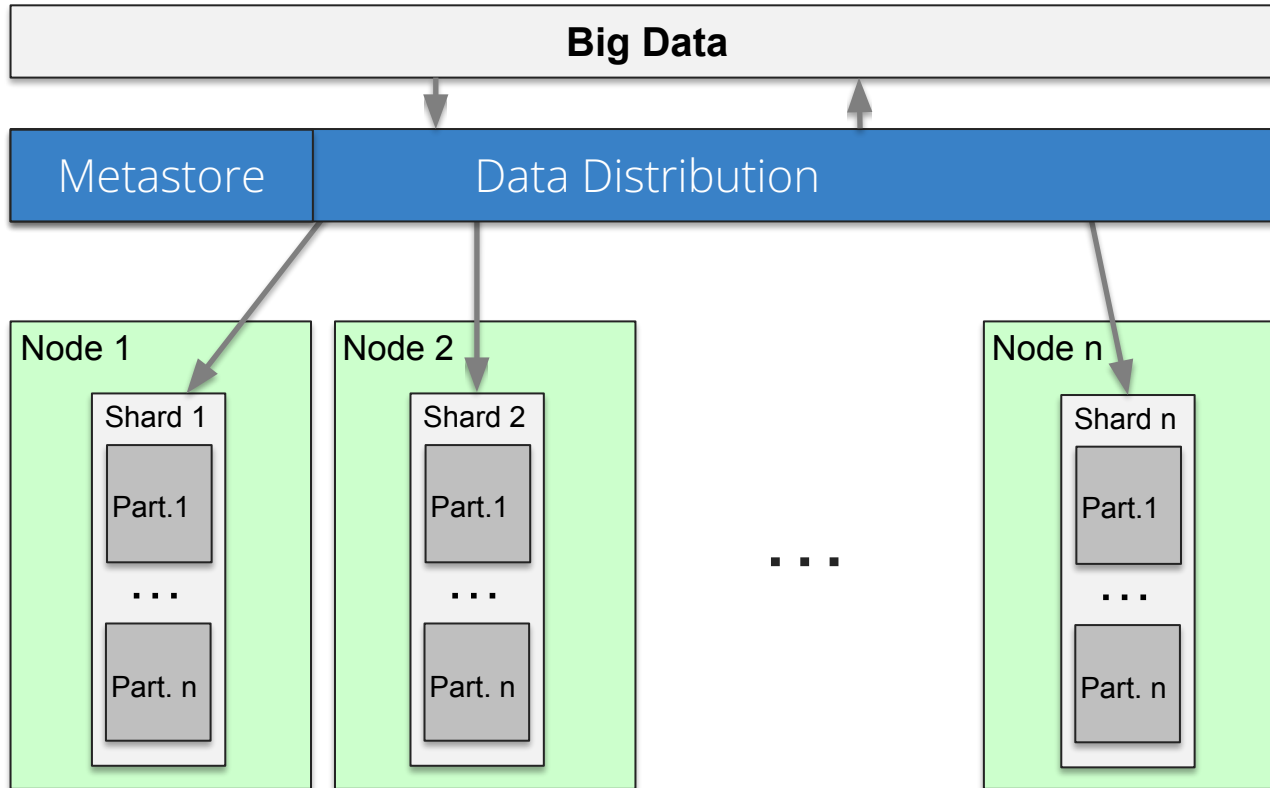


Query Distribution

Data Distribution

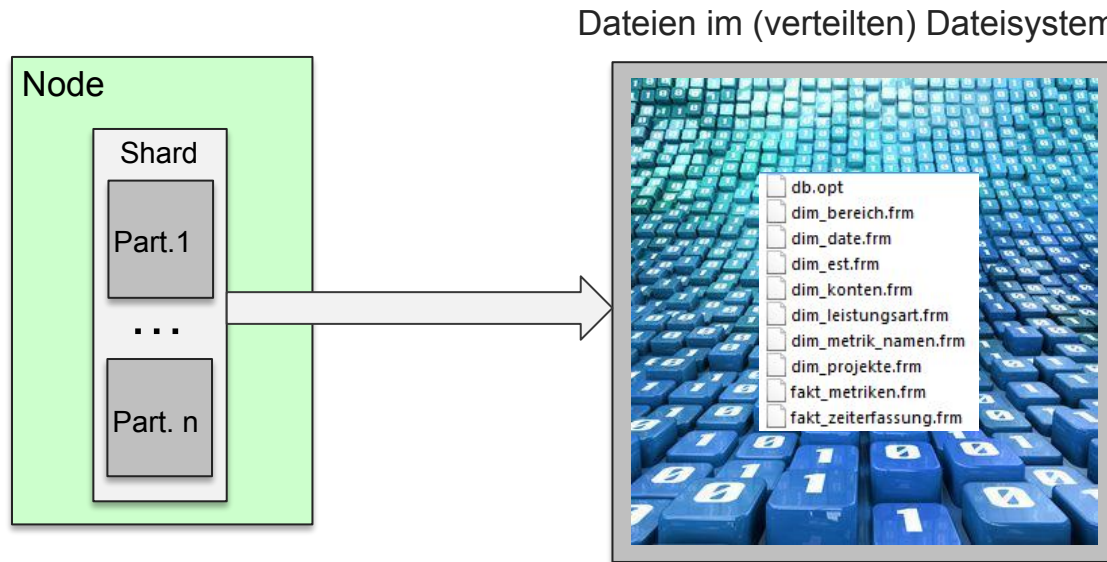
Data Persistence

Sharding and Partitioning: Verteilung und Stückelung von großen Datenmengen

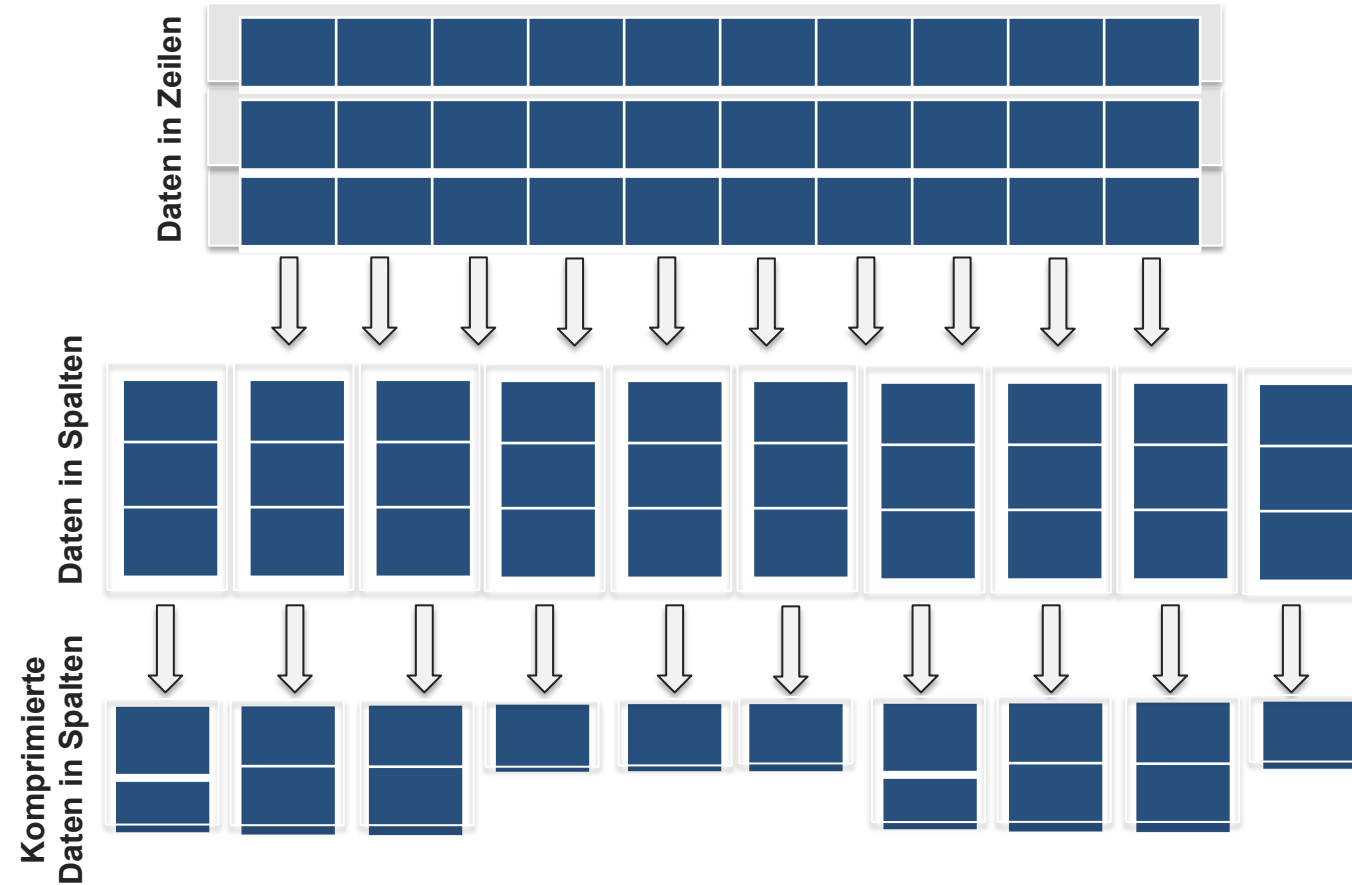


(Re-) Sharding- und Partitioning-Funktion:
 $f(\text{Daten}) \rightarrow \text{Shard}$
 $f(\text{Daten}) \rightarrow \text{Partition}$.
+ Replikationsstrategie.
+ Konsistenzstrategie.

Wie werden große Datenmengen technisch so gespeichert, dass eine schnelle Scan-Geschwindigkeit erreicht wird?



Spalten-orientierte Datenspeicherung

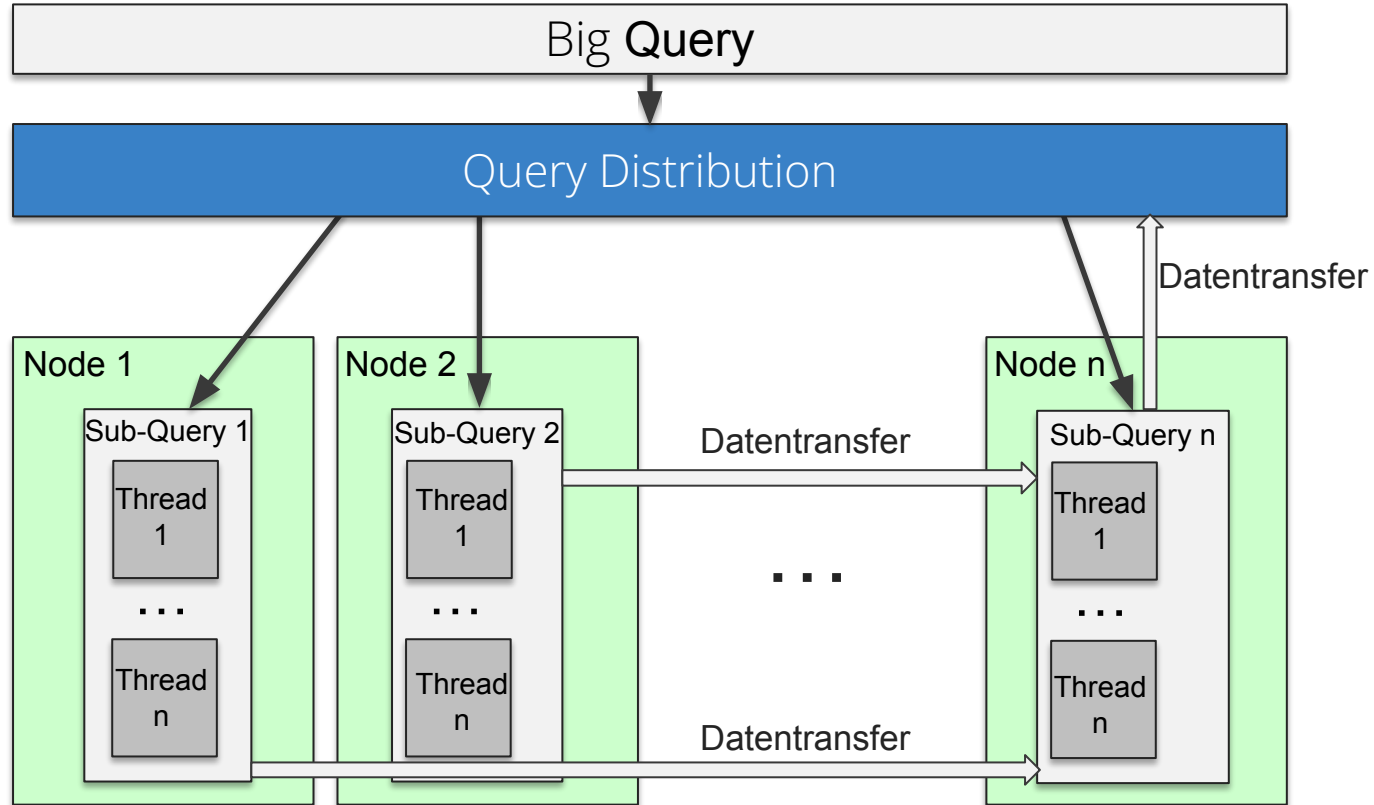


The fastest I/O is the one that **never takes place**: Es werden nur diejenigen Spalten gelesen, die benötigt werden (gerade bei breiten Tabellen wichtig)

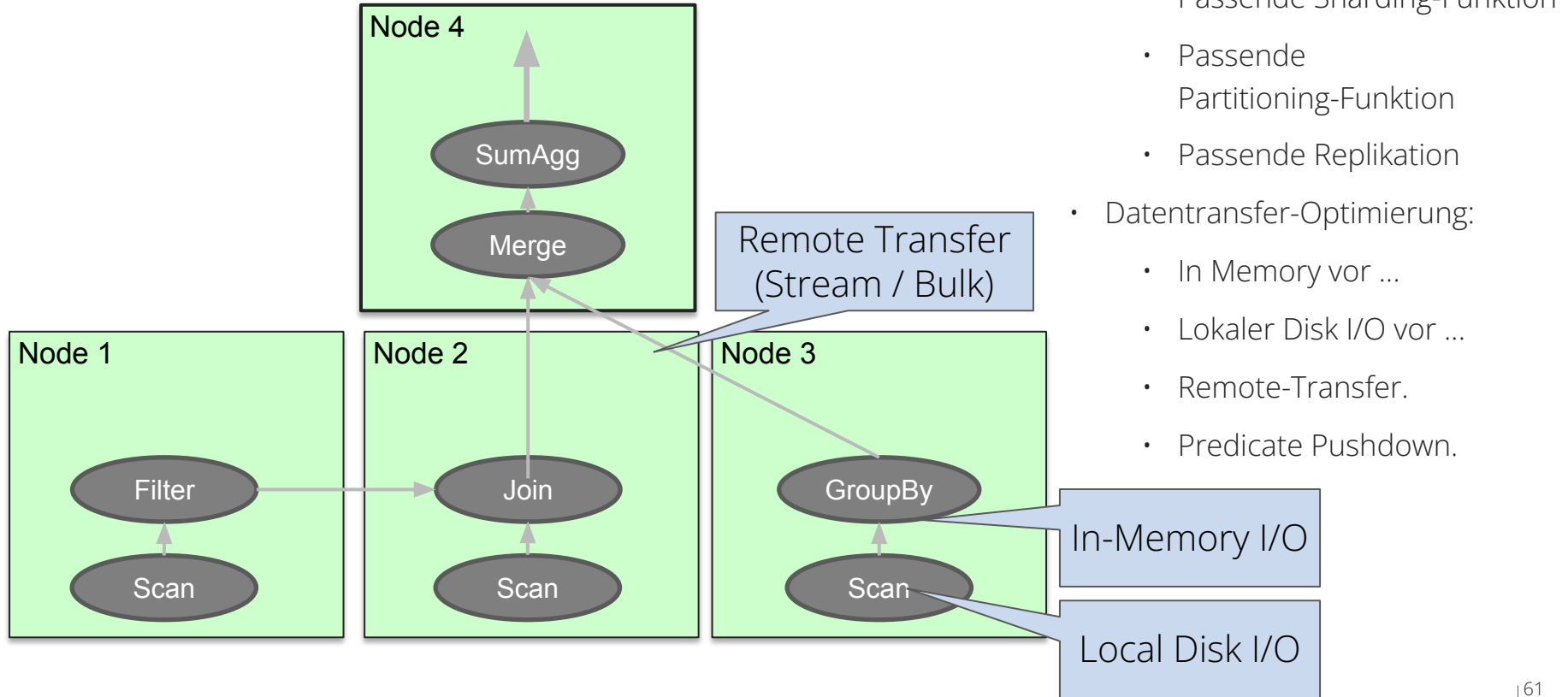
Kompression (funktioniert bei Spalten besser als bei Zeilen):

- Datentyp-spezifisch (z.B. Dictionaries)
- Allgemein (z.B. Snappy)
- + ggF. Spalten-Index

Verteilte und parallelisierte Ausführung von Abfragen



Ein verteilter Ausführungsplan: Ein azyklischer Funktionsgraph



Verteilte Datenbanken

- Apache Cassandra (Wide column store, Tables & Rows)
- Google Bigtable (Wide column store, no relational model)
- Couchbase (document oriented)
- CrateDB (document oriented)
- Amazon DynamoDB (Key-Value)
- Apache HBase (OSS-Implementierung von Bigtable)
- MongoDB (document oriented)
- LinkedIn Voldemort (Key-Value)
- Google Spanner (almost relational, Tables & Rows)
- CockroachDB (OSS-Implementierung von Spanner)