

qaware.de



QA|WARE
SOFTWARE ENGINEERING

Big Data

Franz Wimmer
franz.wimmer@qaware.de

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”

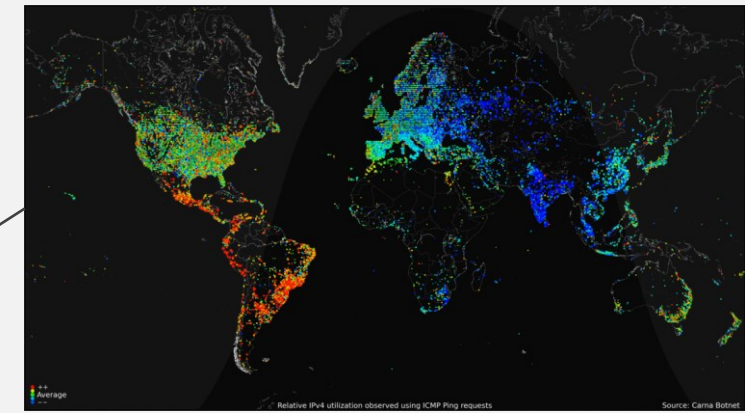
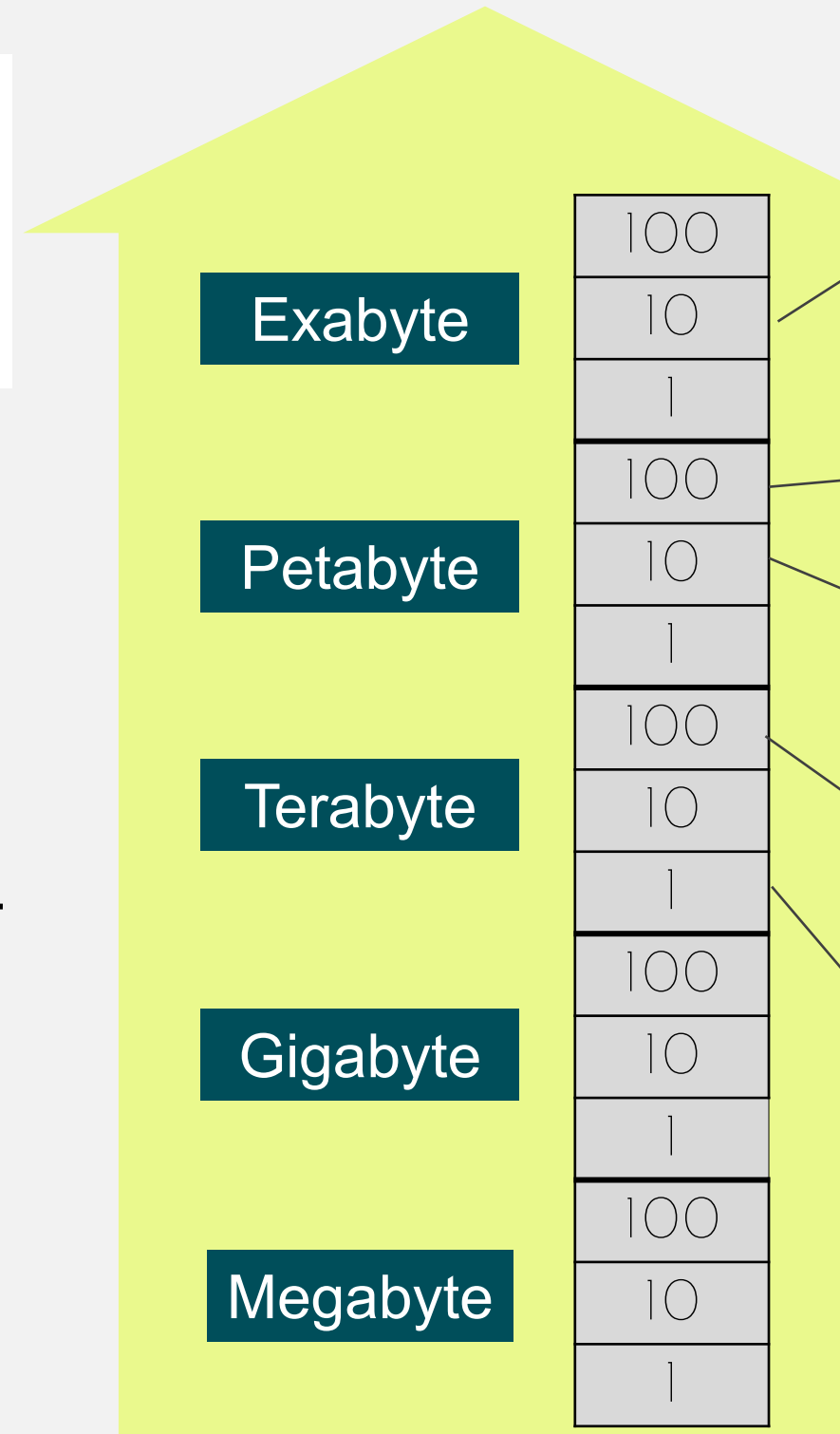
Quelle: . S. Ward und A. Barker. Undefined by data: a survey of big data definitions. arXiv preprint arXiv:1309.5821, 2013.

Big Data

Big Data
Verarbeitung
großer
Datenmengen
durch:

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

Big Data



Das Internet



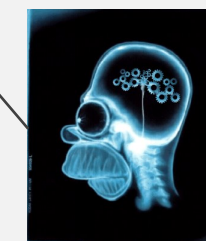
Facebook



Alle Schriften
der Menschheit



Der Google Index



Speichervermögen des
menschlichen Gehirns

DATA & AI LANDSCAPE 2019

INFRASTRUCTURE

The diagram illustrates the landscape of data integration and processing, categorized into three main areas:

- HADOOP ON-PREMISE:** Includes vendors like Cloudera, Hortonworks, MapR, Pivotal, IBM InfoSphere, and Jethro.
- HADOOP IN THE CLOUD:** Includes vendors like AWS, Microsoft Azure, Google Cloud, SAP Cloud Platform, IBM InfoSphere BigInsights, ARM, Google, and CAZENA.
- STREAMING / IN-MEMORY:** Includes vendors like Amazon Kinesis, Databricks, SAP Cloud Platform, Oracle, Confluent, Striim, Hazelcast, GridGain, GIGASPACE, Wallaroo, and FASTDATA.

The collage displays logos for various cloud and database services, organized into six categories:

- NoSQL DATABASES:** Includes logos for Google Cloud, AWS, Oracle, Microsoft Azure, MongoDB, MarkLogic, Couchbase, DataStax, Redis Labs, Aerospike, Amazon ElastiCache, and Scylla.
- NewSQL DATABASES:** Includes logos for SAP, Clustrix, Pivotal, Microsoft, Cloud Spanner, MemSQL, InfluxData, Cockroach Labs, VoltDB, Splice, Oracle, Percona, and TiDB.
- GRAPH DBs:** Includes logos for Neo4j, Amazon Neptune, IBM, Oracle, and GraphDB.
- MPP DBs:** Includes logos for Teradata, Vertica, IBM Data Warehouse Systems, Cognitio, Kognitio, Exasol, and Dremio.
- CLOUD EDW:** Includes logos for AWS, Google Cloud, Microsoft Azure, Pivotal, Snowflake, and Infoworks.
- SERVERLESS:** Includes logos for Amazon Lambda, AWS, Google Cloud, Microsoft Azure, Pivotal, Pulsar, Nuduo, and Pivotal.

DATA TRANSFORMATION

- talend
- pentaho
- alteryx
- TRIFACTA
- tmr
- Paxata
- StreamSets
- UNIFI

DATA INTEGRATION

- SAP Data Services
- Informatica
- Magent
- TEALUM
- inmagick
- enigma
- Data Data Catalog
- Segment
- ATTUNIS
- zaloni
- Import.io
- InfoWorks
- Pluxton
- SNOWFLOW
- MATILLION

DATA GOVERNANCE

- Informatica
- SailPoint
- IBM
- Make My Privacy
- collibra
- dremio
- Alation
- Watermark
- SHRUTA
- OKERA
- data.world
- MANITA

MGMT / MONITORING

- AWS
- New Relic
- octrino
- rubrik
- APPDYNAMICS
- dynatrace
- WINGERT FRONT
- Signifx
- druid
- splunk
- Mosssoft
- perceptivity
- unwired
- Numery
- datakind
- 2025
- Ops Rater
- INACINITY

The collage is organized into six main categories, each with a header and a collection of logos:

- STORAGE**: Includes logos for AWS, Google Cloud, Microsoft Azure, Pure Storage, Alluxio, Wasabi, Minio, and Cohesity.
- CLUSTER SVCS**: Includes logos for Amazon ECS, Amazon EMR, Databricks, Mesosphere, and Cylus Cloud.
- DATA GENERATION & LABELLING**: Includes logos for Amazon SageMaker, Upwork, Appen, Scale, Hive, Labelbox, and Lionbridge.
- AI OPS**: Includes logos for Algorithmia, Vertex AI, Databricks, and others.
- GPU DBs & CLOUD**: Includes logos for Kinetic, Databricks, and others.
- HARDWARE**: Includes logos for Google TPU, ARM, NVIDIA, and others.

CROSS-INFRASTRUCTURE/ANALYTICS

ANALYTICS & MACHINE INTELLIGENCE

The diagram is divided into two main sections by a horizontal line. The top section is titled 'DATA ANALYST PLATFORMS' in red. It contains logos for Microsoft, pentaho, alteryx, Digital Reasoning, guavus, AYASDI, ATTIVO, Datameer, incorta, inter|ana, MODE, ENDOR, and SESO. The bottom section is titled 'DATA SCIENCE PLATFORMS' in red. It contains logos for IBM, databricks, dataiku, COMINO, rapidminer, TIBCO, ANACONDA, sas, JURE, and KNIME. The MathWorks logo is also present at the bottom right of the diagram.

The collage is organized into three vertical sections, each with a header in red text:

- BI PLATFORMS:** Includes logos for Looker, Alteryx, Amazon Analytics, AWS, Domo, Algorika Data, Thoughtspot, ATSCALE, Qlik, GoodData, Information Builders, Birst, Microsoft, Kogni IQ, and others.
- VISUALIZATION:** Includes logos for Tableau, Power BI, SAP, Google Cloud, Celonis, Periscope, Zepi, XorMatrix, Plotly, CHARTIO, and others.
- MACHINE LEARNING:** Includes logos for SAS, Amazon SageMaker, Google Cloud AI/ML, H2O, DataRobot, gammlor, VIZEN, ELEMENT, and others.

COMPUTER VISION

- Microsoft Azure
- Amazon Rekognition
- Clarifai
- EverAI
- Deepomatic
- Neura
- Twentybn
- Ubiquity
- Adee
- Synthesia
- Trax

HORIZONTAL AI

- IBM Watson
- Cortana
- Facore
- Sentient
- Voyager.ai
- Affective
- Processe
- Humanista
- Petuum
- Norlogics
- Curious AI
- OSARO
- Gigamonks
- Pathosix

SPEECH & NLP

- Google Cloud
- Twilio
- Baidu
- Amazon Translate
- semantic scholar
- semantic routing
- Melvoxy
- Openlogic
- Soundhound Inc.
- PRIMER
- Firstful
- Acoustic
- Speechmatics
- Cogito
- Snips
- Shutterstock
- Unbabel
- Polylab

The collage displays logos for various SaaS companies, organized into four distinct categories, each highlighted with a colored background and a red border:

- SEARCH** (Blue background): Includes logos for Elasticsearch, Oracle, Algolia, Coveo, Luckworks, Attivio, Swifttype, and Xignite.
- LOG ANALYTICS** (Green background): Includes logos for Splunk, Sumologic, SolarWinds, Timber, Kibana, and Logz.io.
- SOCIAL ANALYTICS** (Orange background): Includes logos for Hootsuite, Sprinklr, Netbase, Synthesio, Trex, Smackreach, Bitly, and SimilarWeb.
- WEB / MOBILE / COMMERCE ANALYTICS** (Red background): Includes logos for Google Analytics, Mixpanel, Amplitude, Airtable, Resciscience, Sigopt, and Granify.

APPLICATIONS – ENTERPRISE

SALES (15)

CHORUS
INSIDESALES.COM
peopleplea
conversica
clari
fusemachines

MARKETING - B2B (15)

RADIUS
EVERSTRING
Lattice
MINTIGO
sense
tubular
ENOGGIO
KNOTCH
mrpe

MARKETING - B2C (15)

zeta
bloomreach
SendGrid
braze
ACTIONIQ
BLUECORE
CONTENTRELM
TEALUM
imparticle
Amplero
Sampert
GUANTIFIND
Simon
Lifepa
[PERSADO]
remesh

CUSTOMER EXPERIENCE / SERVICE (15)

qualtrics
MEDALLIA
SurveyMonkey
LoopTesting
CLARABRIDGE
zendesk
Customer
freshdesk
Intercom
Drift
LIFEPIPER
Gainsight
pendo
HEAP
Amplitude
Workday Assistant
OutSystems
DigitalGlobe
DigitaiKinesis
ASAPP
ada
AUTOMATAD
ahmifi
Covintek
INFORMA
Covintek
fractal

ENTERPRISE PRODUCTIVITY (15)

slack
ORACLE
GURU
lumiata
DIFFBOT
Clarity
talla
Koristo

Sector	Count
HUMAN CAPITAL	10
LEGAL	10
REGTECH & COMPLIANCE	10
FINANCE	10
BACK OFFICE AUTOMATION & RPA	10
SECURITY	10
CYBERSECURITY	10
AI/ML	10
BLOCKCHAIN	10
IOT	10
AR/VR	10
SPACE	10

– APPLICATIONS – INDUSTRY

Sector	Count
Advertising	10
Education	10
Real Estate	10
Gov't	10
Intelligence	10
Finance - Investing	10
Finance - Lending	10
Insurance	10

HEALTHCARE

flatiron, Clover, KYRUS, HealthTap, METABIOTA, Ginger.io, Glow, babykin, 30Med, zebra, PathAI, ovig, TEMPUS, patientsLikeMe, AICure, insitro, notable, Khoroscient, citizen, FOCUSUR, enitic, imago, bioception, prognos, Qvenbus, ARTERYS, iMAGEN, BAYLABS, innovaccor, KIRITA, BIOGEN IDEC, LeonBio

LIFE SCIENCES

Exelixis, color, Corcept, BenevolentAI, verily, WUXNextCODE, Z, Genentech, BION, Clear Labs, Neenome, NANOPORE, CNAnexus, PAPERPORE, CITRINE, twoSTAR, CEMIC GENOMICS, ADELPHI, OXION, SYRTECH GENOMICS

TRANSPORTATION

UBER, TESLA, WAYMO, ZO XO, CLEARPATH, CRUISE, NURO, AIGO, Instamoney, drive.ai, CAMBRIDGE AI, Aurora, nauto, AMOTIVE, G7, PILOT.AI, NIO, OPTIMUS, moovit, Iliac, nexor, Kodiak, comma.ai, ntradyne, Carbon Robotics, Civil Maps, xango, thinkr, INRIX

AGRICULTURE

FARMERS' BRIDGES, Granular, JOHN DEERE, BLUEIVER, FarmersEdge, AgroStar, FarmLogs, TARANIS, GAMAYA, terrostream, prospera

COMMERCE

Instacart, FAIR, STITCH FIX, Da & Co, HoxGood, heartkick, eharmony, stem, Amper, ByteDance, Huggins, collect, SOJERN, BOXEVER, VERIDIGIS, duetto, Julekade, Electric, ZINIER, Movano

INDUSTRIAL

AVEVA, SIEMENS, PREDIX, UPTAKE, SCORTEX, KODAK, TACHYUS

OPEN SOURCE

The diagram illustrates a wide range of open-source tools organized into 11 functional categories:

- FRAMEWORKS:** Includes logos for Hadoop, Spark, Flink, YARN, Tez, Mesos, Kubernetes, CDAP, and HEDX.
- QUERY / DATA FLOW:** Includes Spark, SQL, Presto, Apache Drill, SLAMDATA, GraphQL, and Flink.
- DATA ACCESS & DATABASES:** Includes Cassandra, MongoDB, Redis, Cockroach Labs, Druid, ClickHouse, Apache Kudu, and others.
- ORCHESTRATION & MGMT:** Includes Talend, Apache Airflow, and others.
- STREAMING & MESSAGING:** Includes Spark, NiFi, Flink, Beam, Kafka, and Storm.
- STAT TOOLS & LANGUAGES:** Includes Jupyter, Scala, R, and others.
- AI OPS & INFRA:** Includes MLflow, Kubeflow, and others.
- AI / MACHINE LEARNING / DEEP LEARNING:** Includes TensorFlow, Keras, PyTorch, and others.
- SEARCH:** Includes Elasticsearch and Solr.
- LOGGING & MONITORING:** Includes ELK Stack (Elasticsearch, Logstash, Kibana), Prometheus, and others.
- VISUALIZATION:** Includes Matplotlib, TensorBoard, Seaborn, and others.
- COLLABORATION:** Includes BeakerX, Jupyter, and others.
- SECURITY:** Includes Apache Ranger, Knox, and others.

DATA SOURCES & APIs

HEALTH

- Apple
- VALIDIC
- practicefusion
- fitbit
- GARMIN
- HUMANAPI
- kinso
- MIMIC

IOT

- GE Digital
- UPTAKE
- thingworx
- helium
- samsara
- airbyte

FINANCIAL & ECONOMIC DATA

- Bloomberg
- THOMSON REUTERS
- DOW JONES
- SEPCAPITAL IQ
- CBRIGHTS
- PLAID
- SECOND MILEAGE
- FAIRVECT
- YODLEE
- StockTwits
- xignite
- Thickstack
- earnest
- predata

AIR / SPACE / SEA

- Orbital Insight
- planet
- satellite
- AIRBOTICS
- spire
- cospry
- USERSTORY
- trifusion
- WINDWARD
- MarineTraffic
- BornDeploy
- ES

PEOPLE / ENTITIES

- axiom
- experian
- SPSILON
- insideView
- Crimson Hexagon
- BASIS
- Quantcast
- SAFEGRAPH

LOCATION INTELLIGENCE

- FOURSQUARE
- mapbox
- sense360
- penumbra
- HEXAGON
- PlaceIQ
- esri
- factual
- CARTO
- Mopilly
- StreetView
- cuebiq
- Radar
- OpenStreetMap

OTHER

- DATA.GOV
- IMAGENET
- Lab41
- YPOUNDS
- CRUX
- algofit.io

DATA RESOURCES

DATA SERVICES

- OPERA
- LUCA
- DATA SCIENCE
- fractal
- kaggle
- EXL
- DataKind
- innodexus

INCUBATORS & SCHOOLS

- PLURALSIGHT
- GA
- galvanize
- DataCamp
- DataElite
- INSIGHT
- The Data Incubator
- METIS

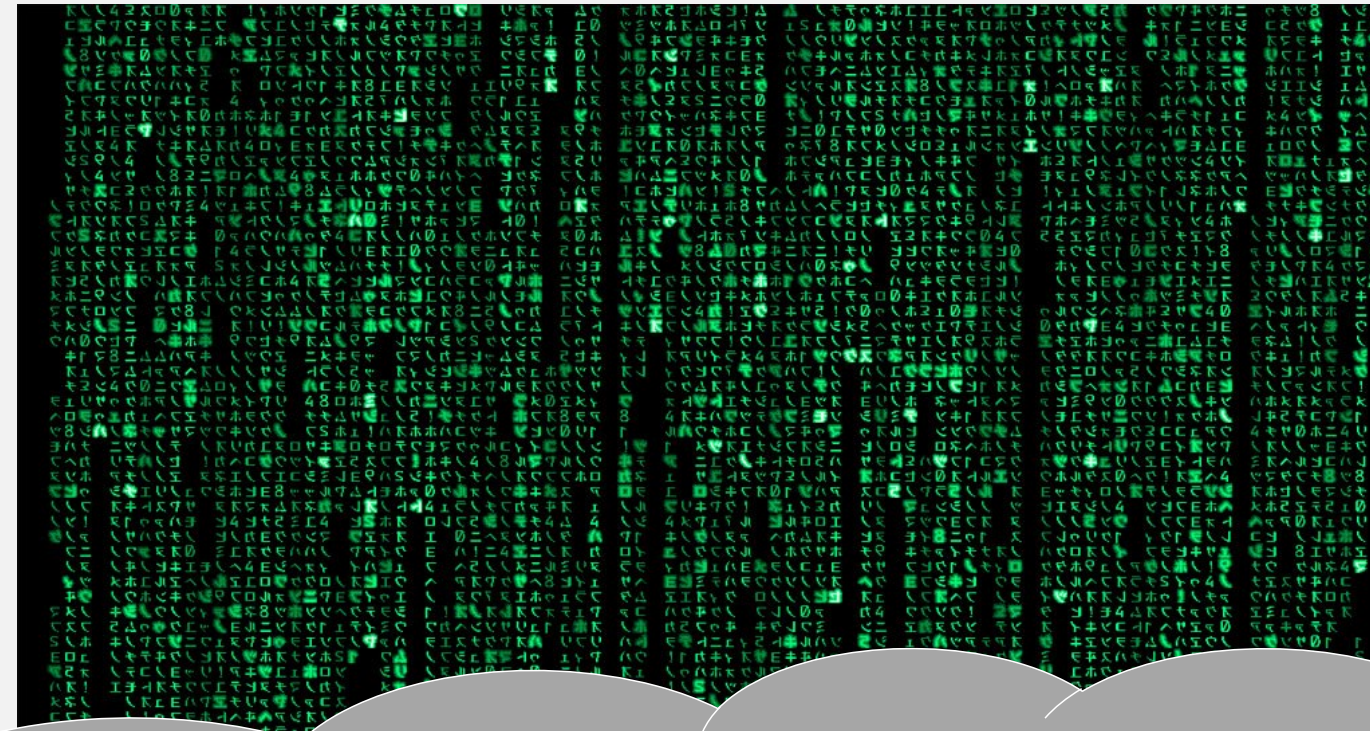
RESEARCH

- OpenAI
- facebook research
- MIRI
- VECTOR INSTITUTE
- AI2
- Allen Institute for Artificial Intelligence

Wie verwalte und erschließe ich große Datenmengen?



QA|WARE



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));
    }
}
```

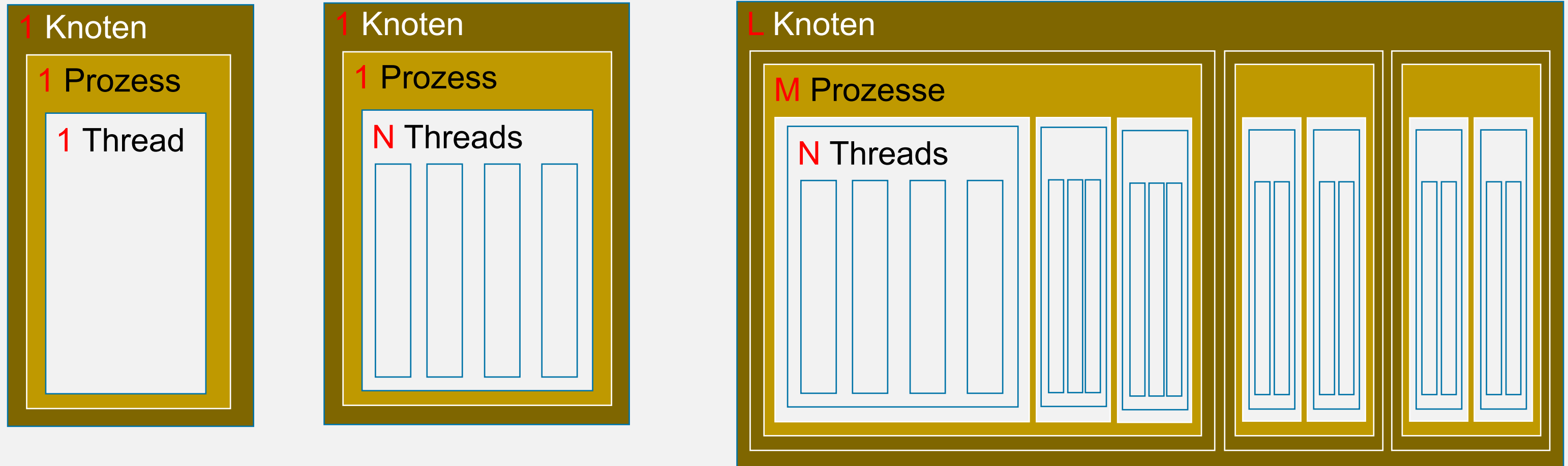
- Funktioniert nicht: Berechnung der Fibonacci-Folge ($F_{k+2} = F_k + F_{k+1}$). Berechnung ist nicht parallelisierbar.

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.

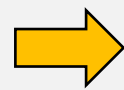
Parallele Programmierung basiert oft auf funktionaler Programmierung.

- 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.

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



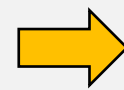
Keine
Parallelität



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.
Die vier Paradigmen der Parallelität im Großen:



Folgt aus Datenmenge
im Vergleich zur
Programmgröße

Das Grundprinzip von
paralleler Verarbeitung.

Folgt aus
Praxisanforderung:
Viele Knoten bedeutet
viele Ausfallmöglichkeiten

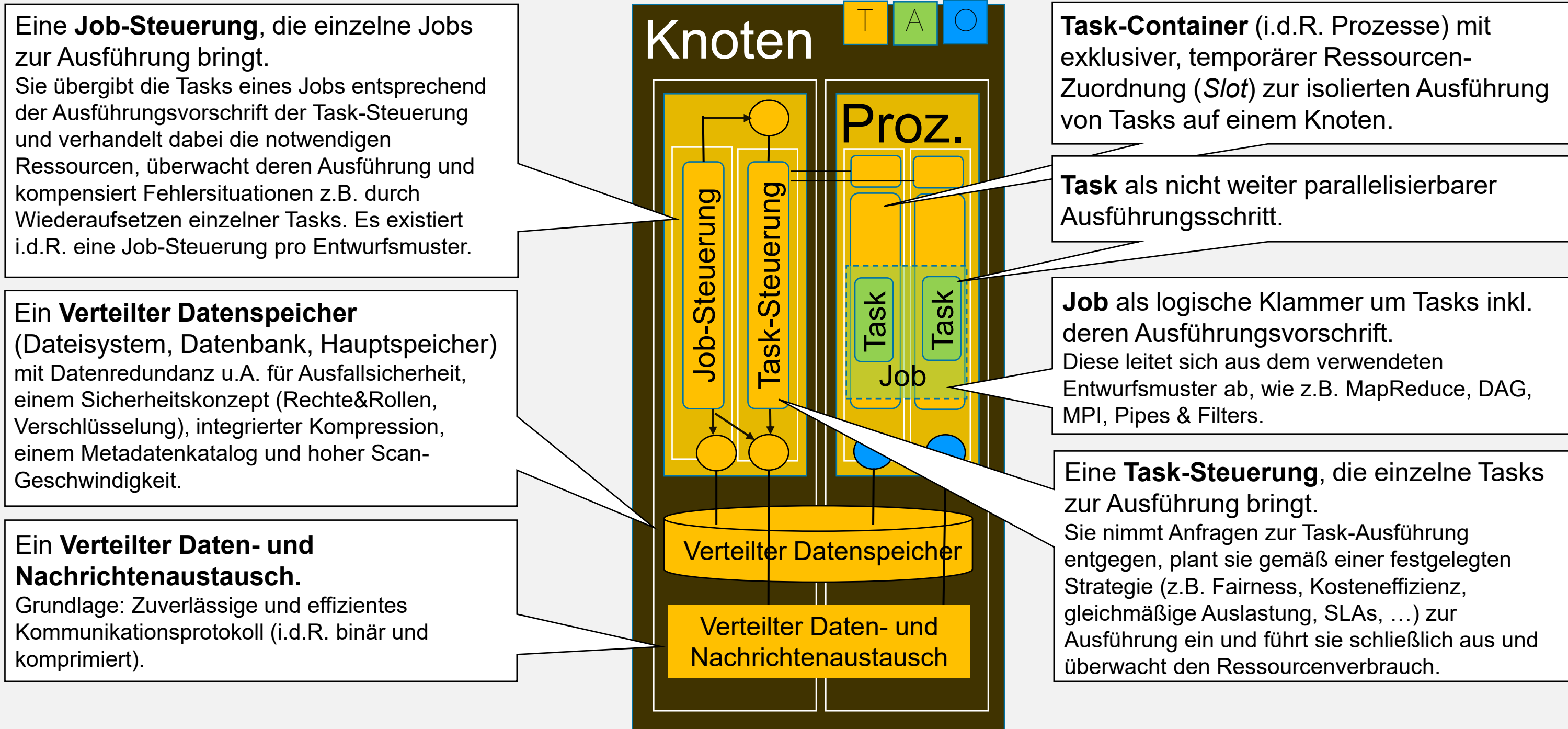
Folgt aus potenziell
großer Datenmenge
und Verarbeitungs-
geschwindigkeit

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.

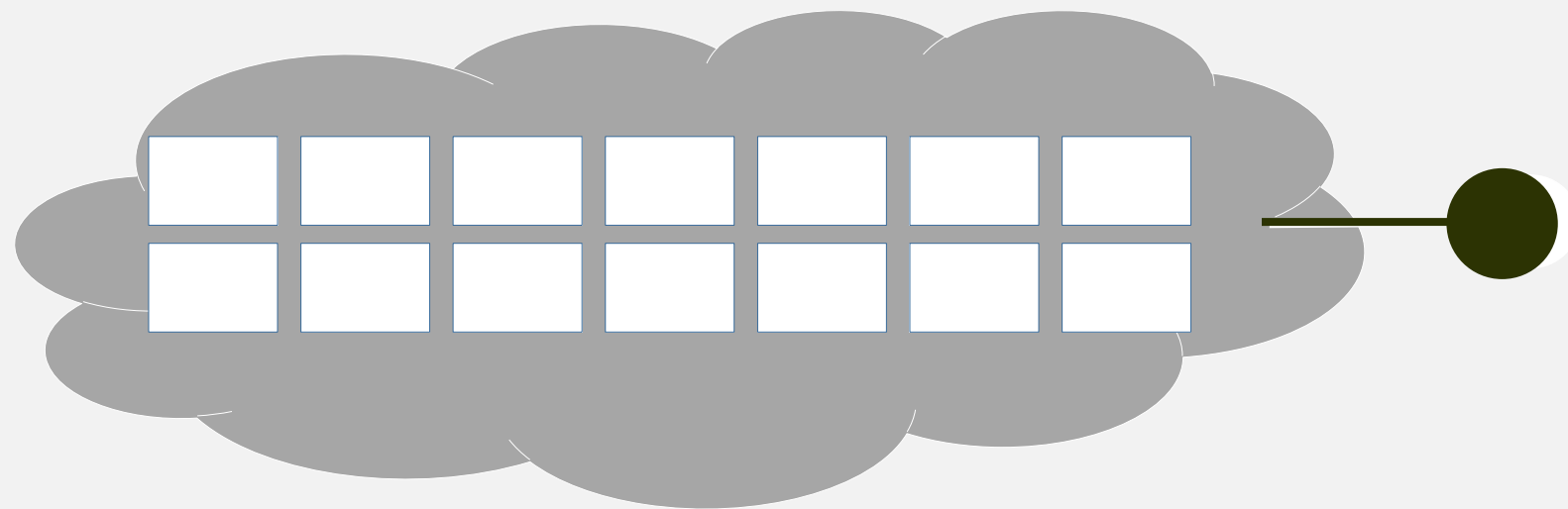
Notwendige Architekturkonzepte

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?



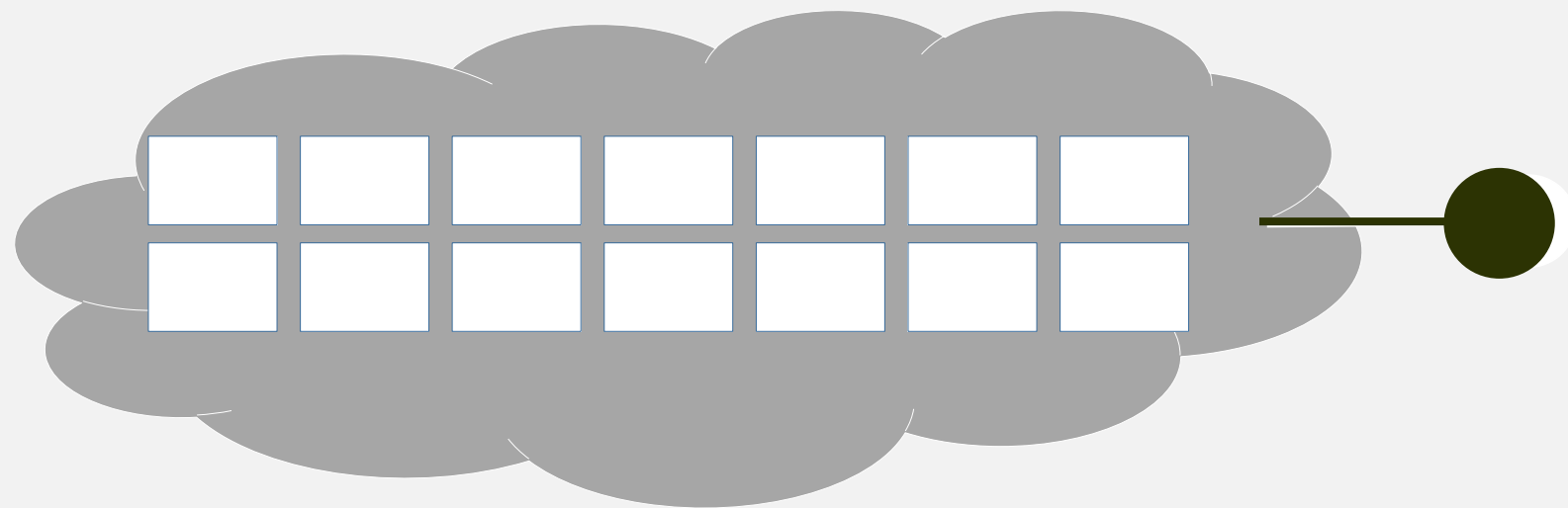
- Big Data Engines (low level)
 - MapReduce
 - RDD (Resilient Distributed Dataset)
- Big Data Datenbanken (high level)
 - NoSQL Datenbanken
 - NewSQL Datenbanken (NoSQL + SQL)
- Verteilte Dateisysteme
- In-Memory Data Grids / Elastic Memory



QA|WARE

Verteilte Algorithmen

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 *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.

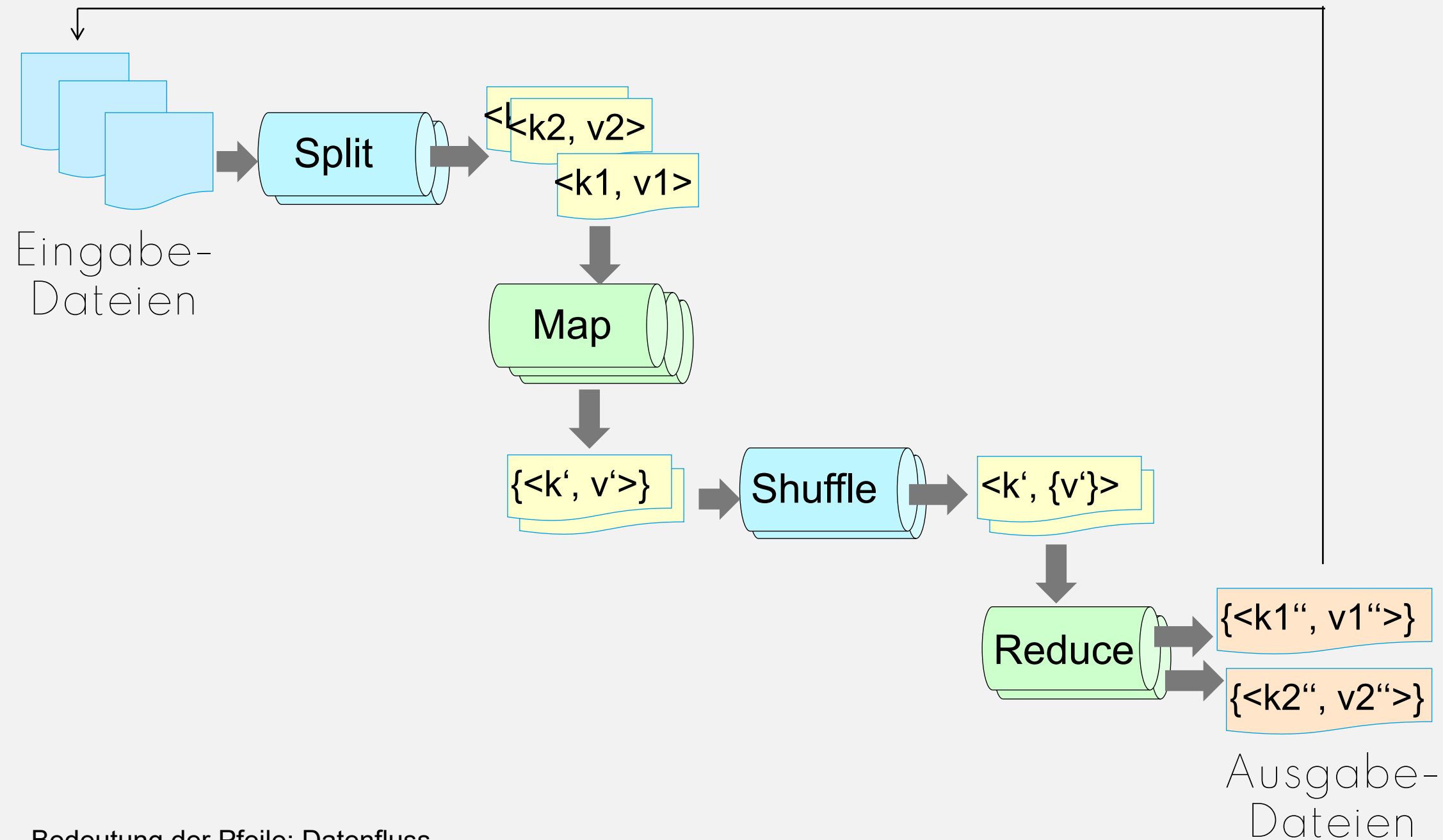
Signatur: **map**(k, v) \rightarrow list(<k', v'>)

- 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: |list(<k'', v''>)| \ll |list(<k', v'>)|

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



Bedeutung der Pfeile: Datenfluss

Die Map-Phase

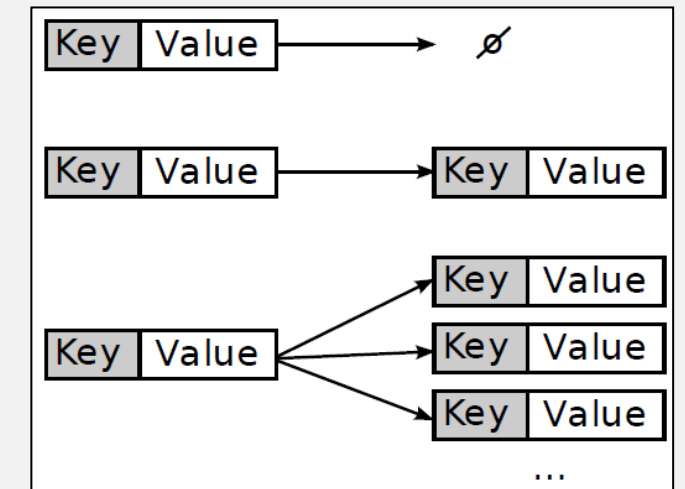
Split

Map

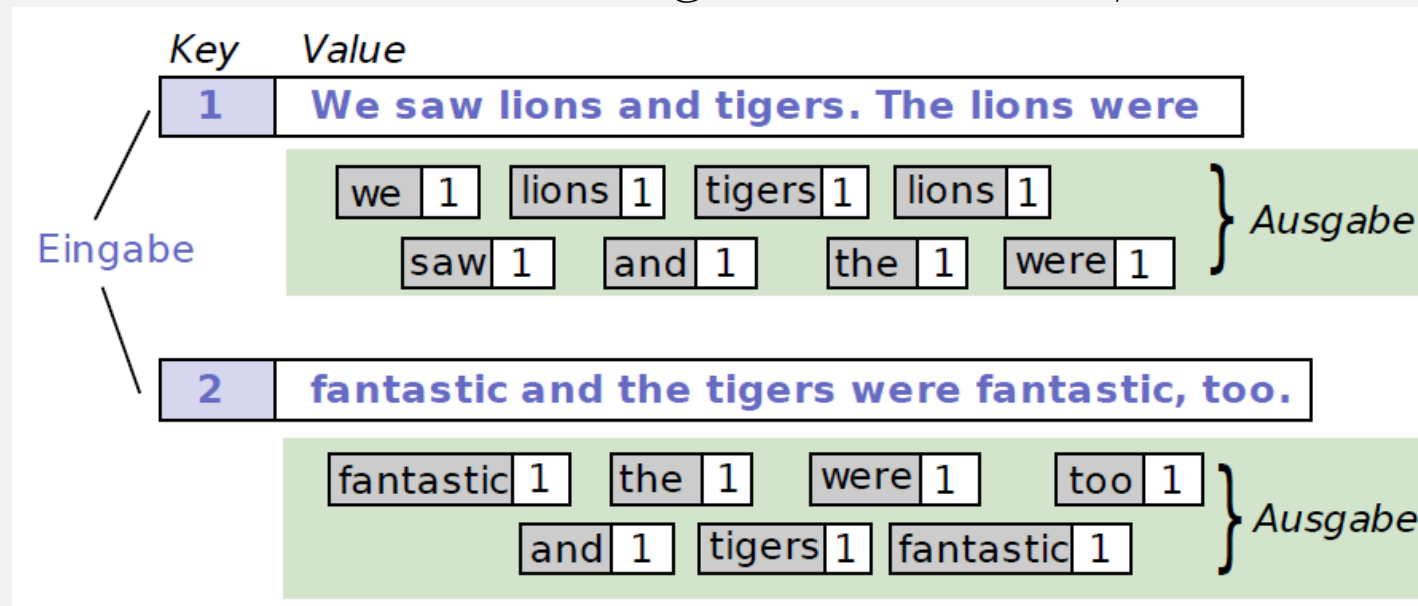
Shuffle

Reduce

- 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



Ein- und Ausgabe der Map-Phase:



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

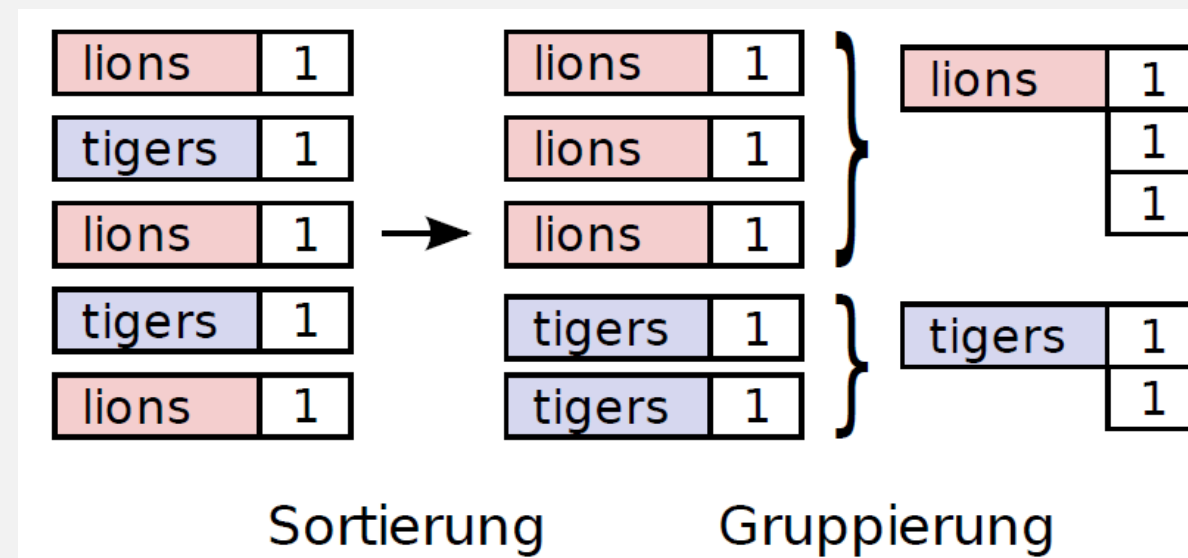
Split

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

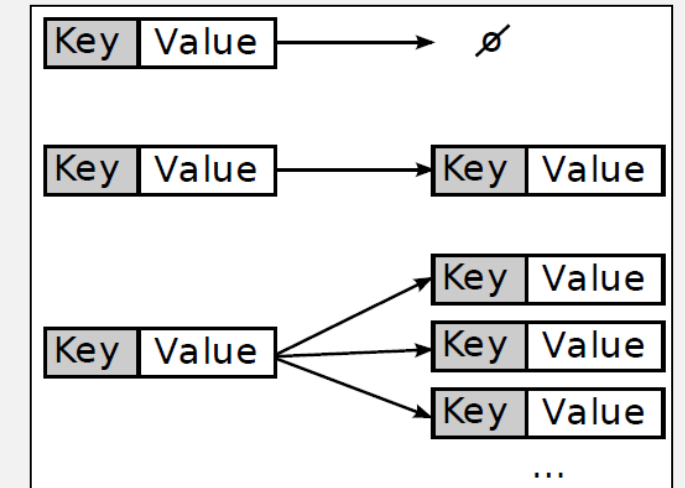
Split

Map

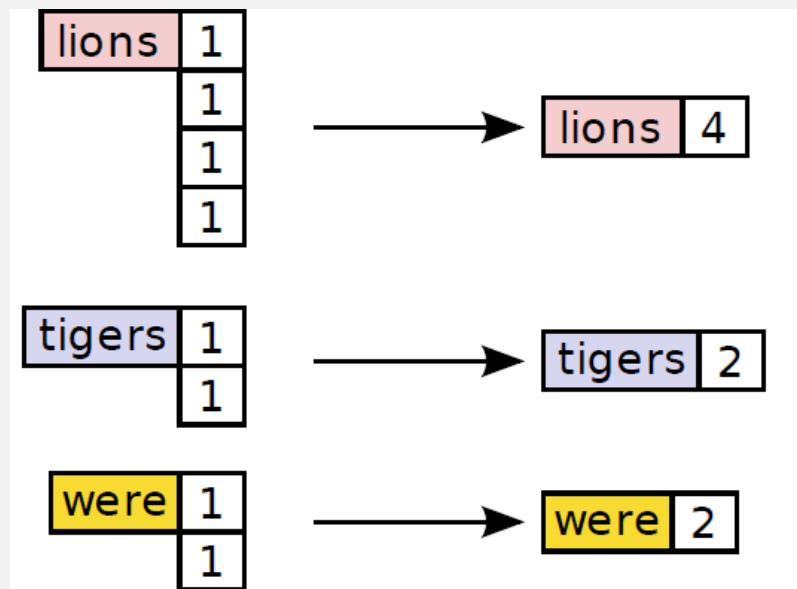
Shuffle

Reduce

- 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:



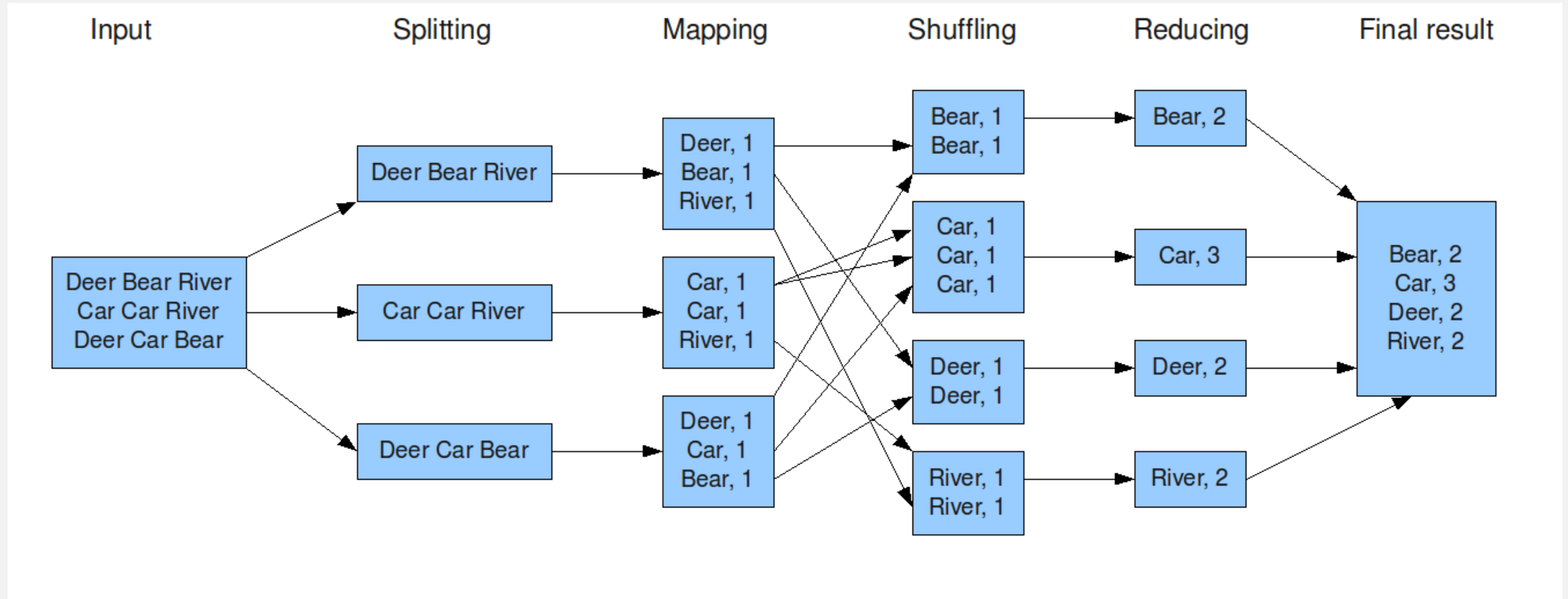
Ein- und Ausgabe der Reduce-Phase:



Pseudocode Reduce-Phase:

```
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.jteam.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) → <Wort, 1>: Erkennt einzelne Wörter im Textfragment.

■ **reduce**(<Wort, list(1)>) → <Wort, Anzahl>: Zählt die Anzahl zusammen.

Verteiler regulärer Ausdruck

In welchen Zeilen eines Textes kommt ein Suchmuster vor?

■ **map**(Textfragment) → <Zeile, 1>: Findet das Suchmuster im Textfragment.

■ **reduce**(<Zeile, list(1)>) → <Zeile, Anzahl>: 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) → <Ziel, Quelle>: Findet für die Quelle einzelne Verweise auf Ziel-Seiten.

■ **reduce**(<Ziel, list(Quelle)>) → <Ziel, set(Quelle)>: 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>



QA|WARE



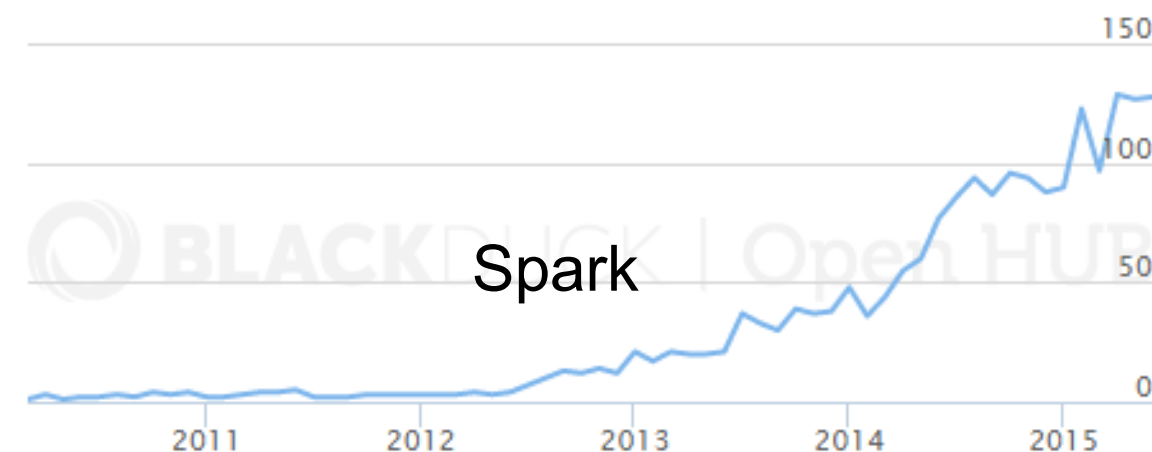
Apache Spark

Spark läuft Hadoop aktuell deutlich den Rang ab.

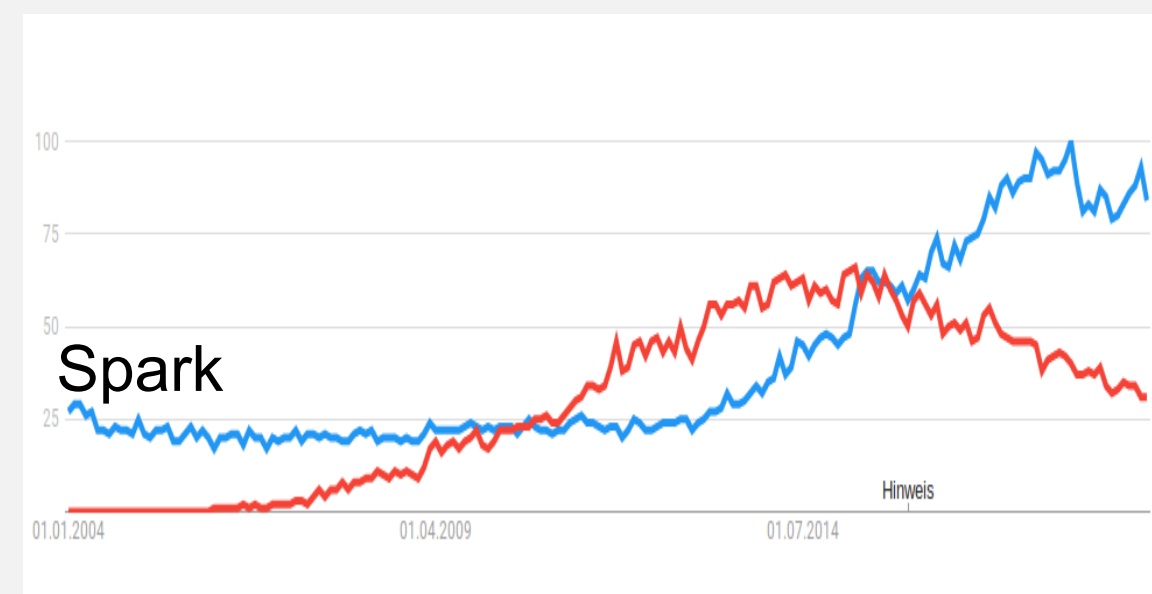
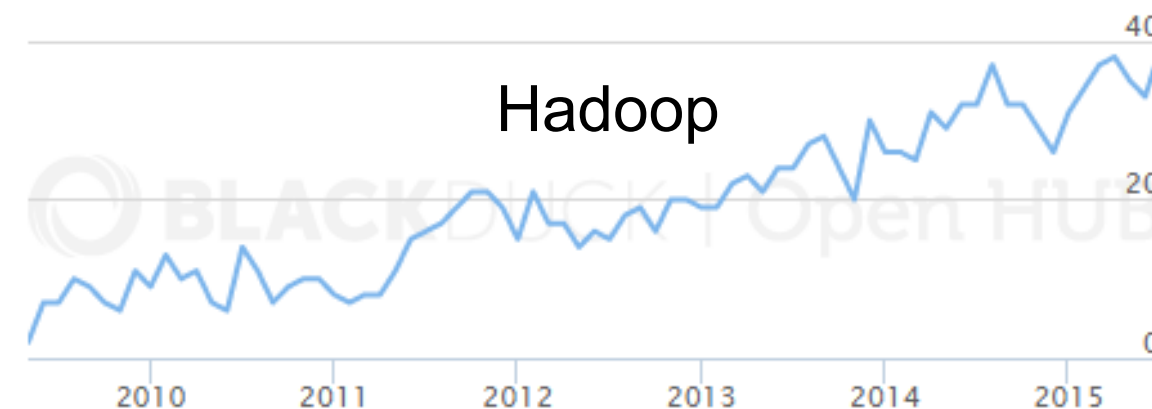
	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Sort rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min

<http://sortbenchmark.org>

Contributors Per Month



Contributors Per Month



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

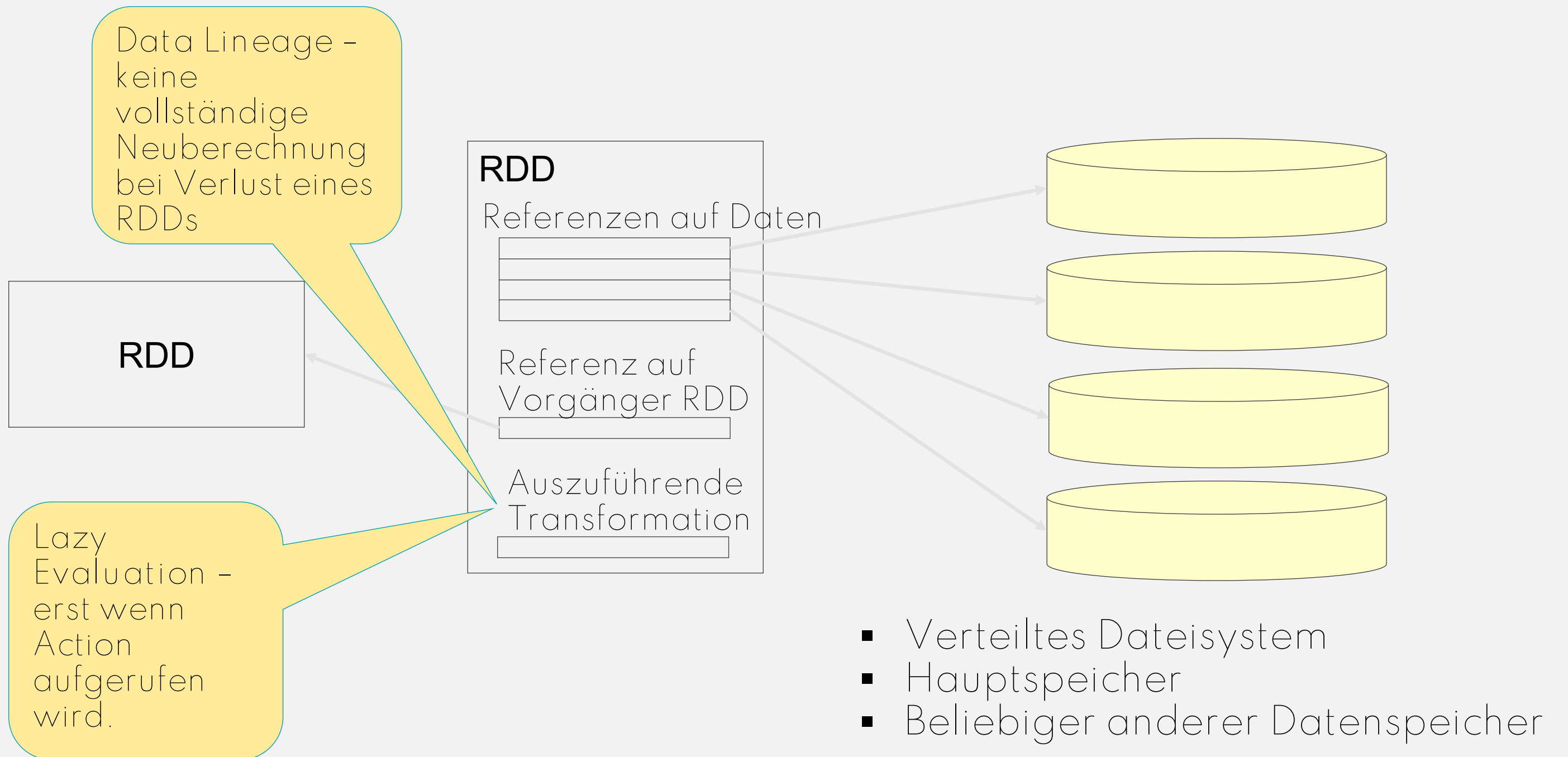
Eine RDD ist in der Außensicht ein klassischer Collection-Typ mit Transformations- und Aktionsmethoden.

RDD → RDD

RDD → skalarer Typ, Collection, Storage

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>
...	...

Die Anatomie eines RDDs.



Daten 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

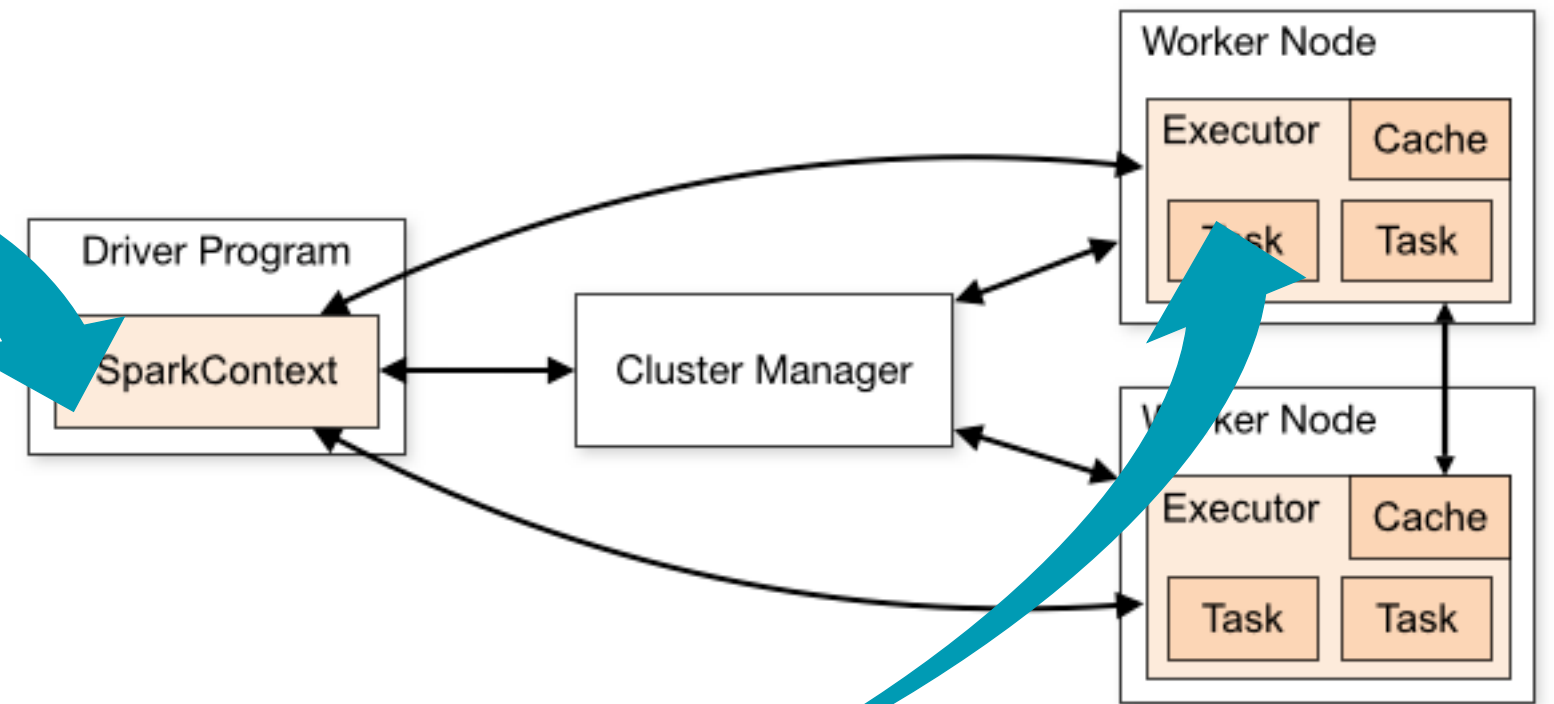
```
val sorted = logData.sortBy(l => l.length)
```

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.m
    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
  }
}
```





QAWARE

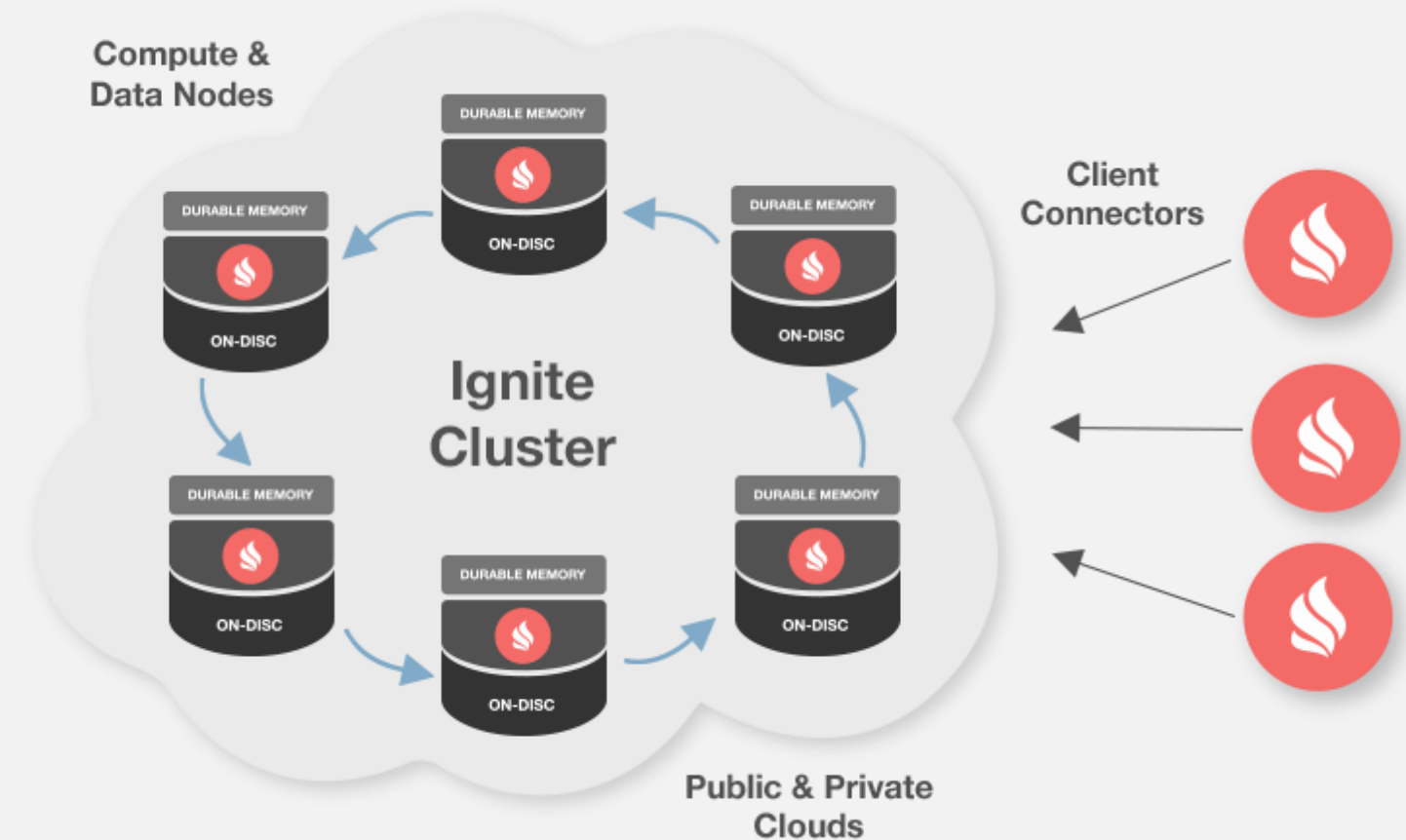
Apache Ignite



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

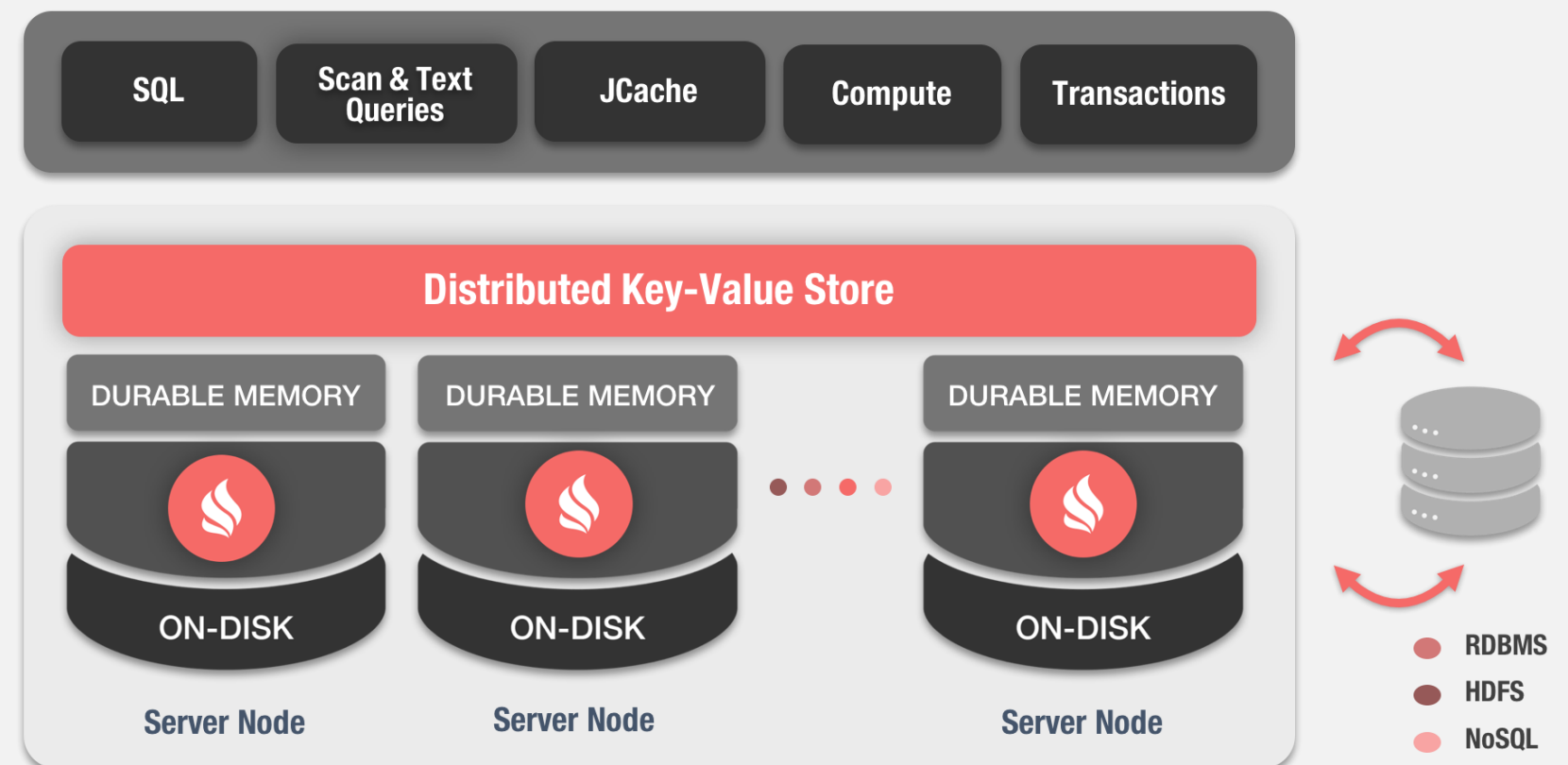
Apache Ignite

- 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();

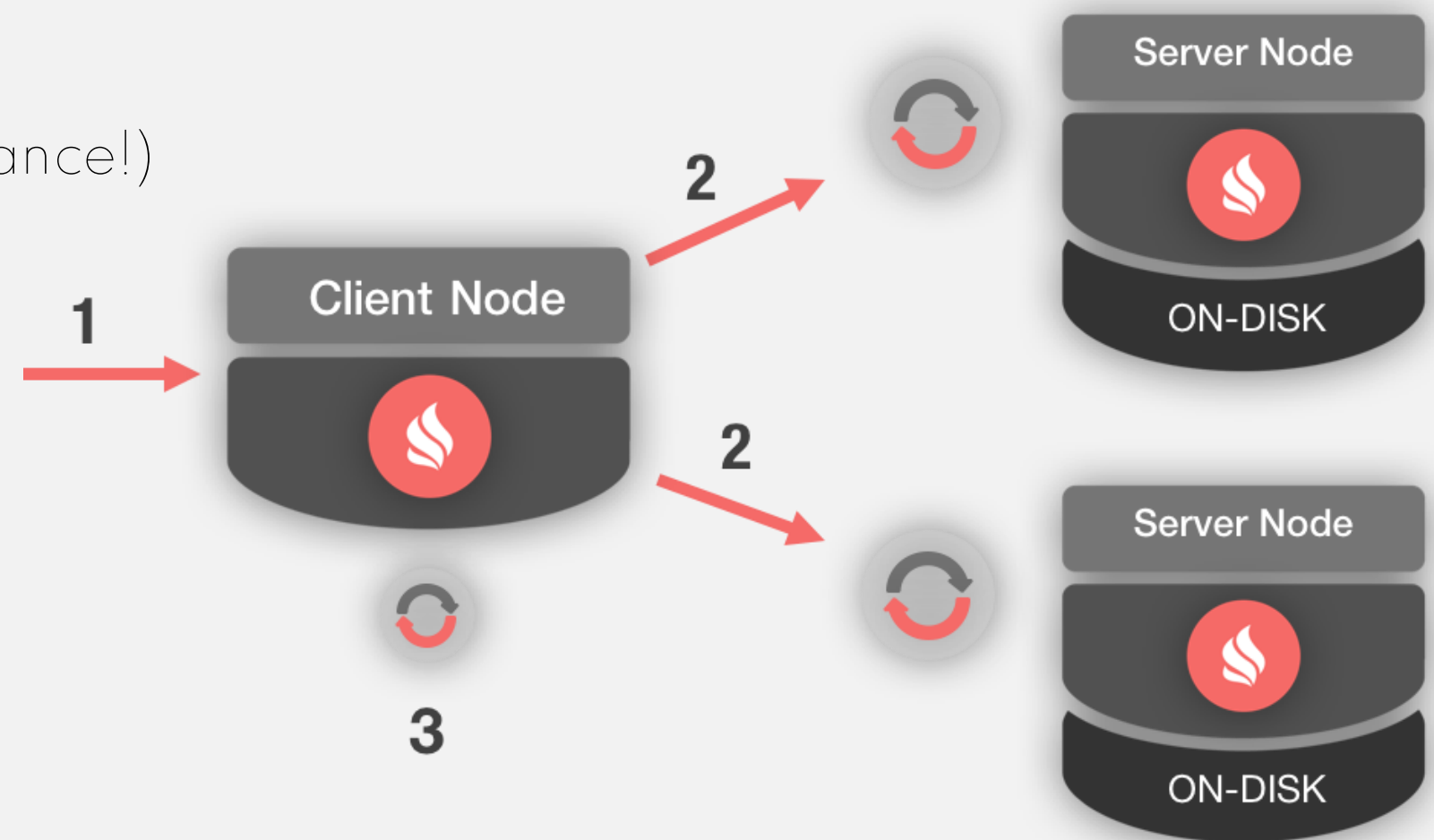
final IgniteCache<Integer, String> cache = ignite.cache("cacheName");

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())) {

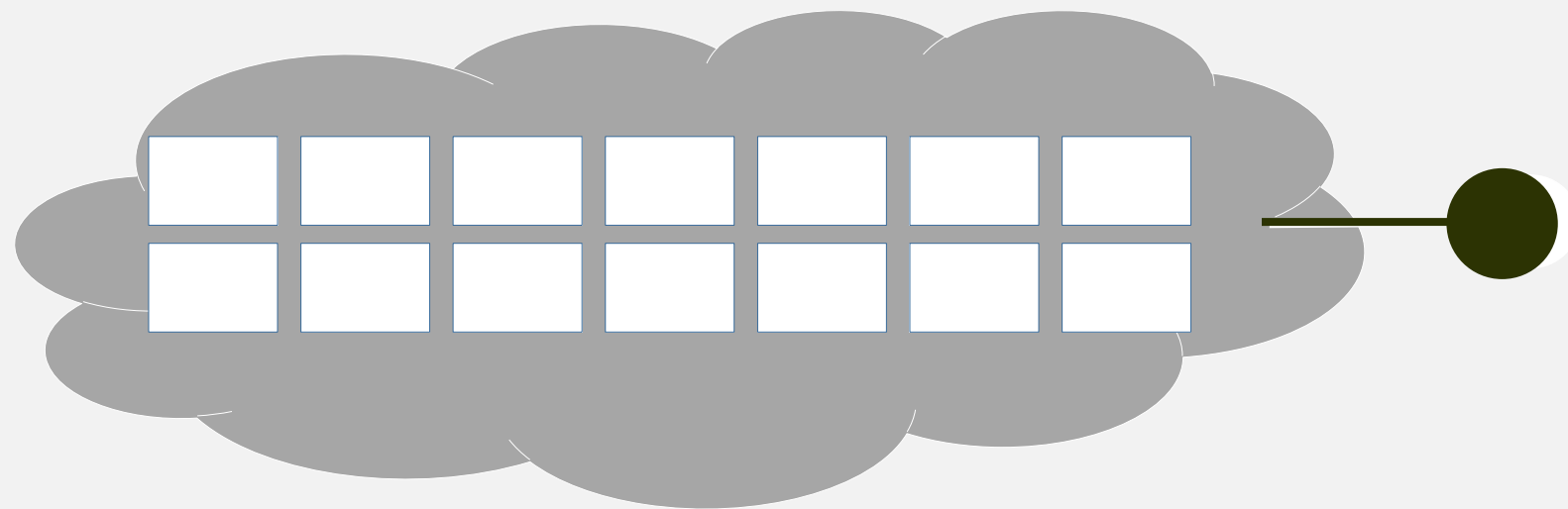
    while(true) {
        String randomWord = RandomStringUtils.randomAlphanumeric(12);
        // Stream words into Ignite.
        streamer.addData(randomWord, randomWord);
    }
}
```



QAWARE

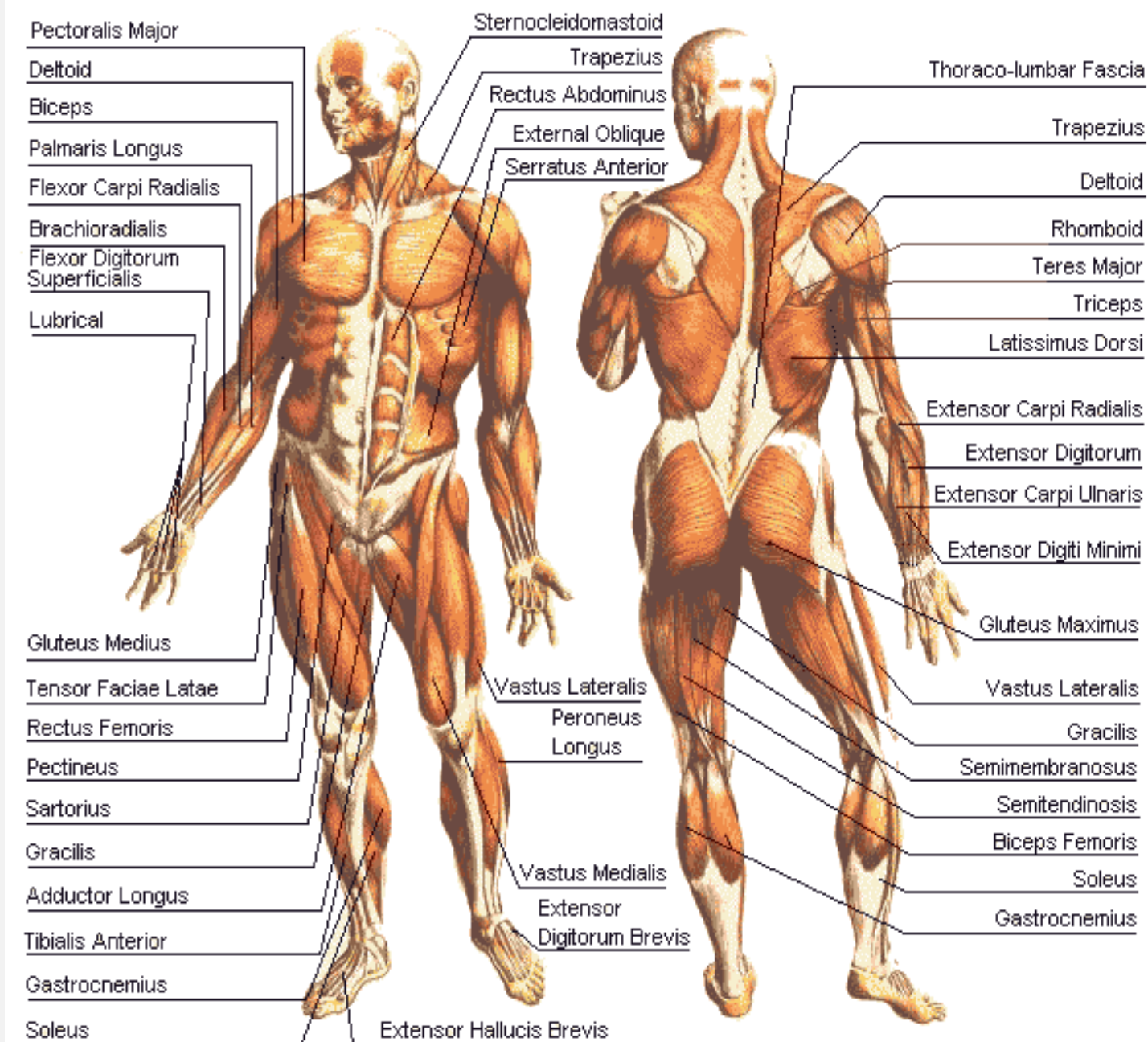
Big Data Datenbanken

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

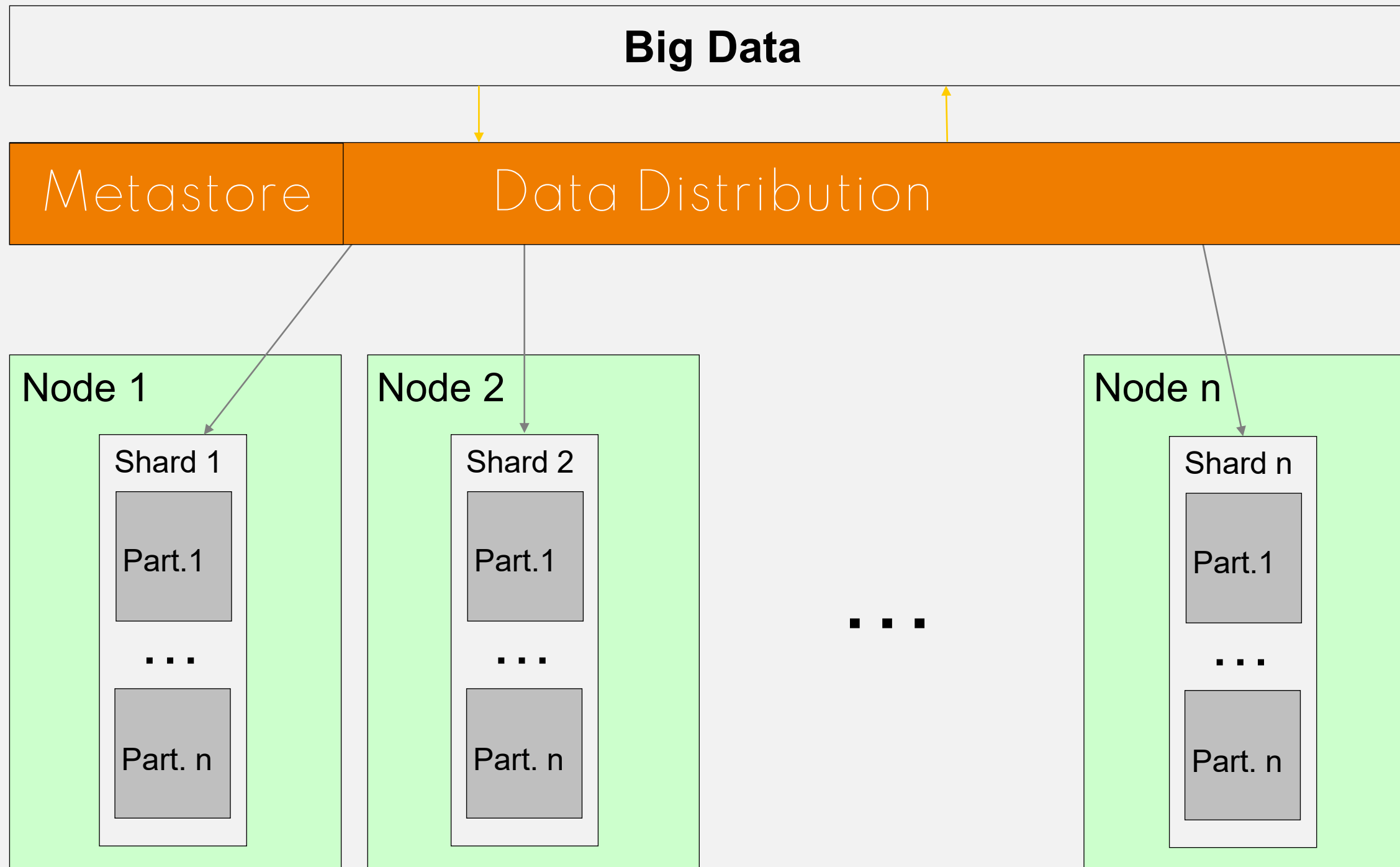


Query Distribution

Data Distribution

Data Persistence

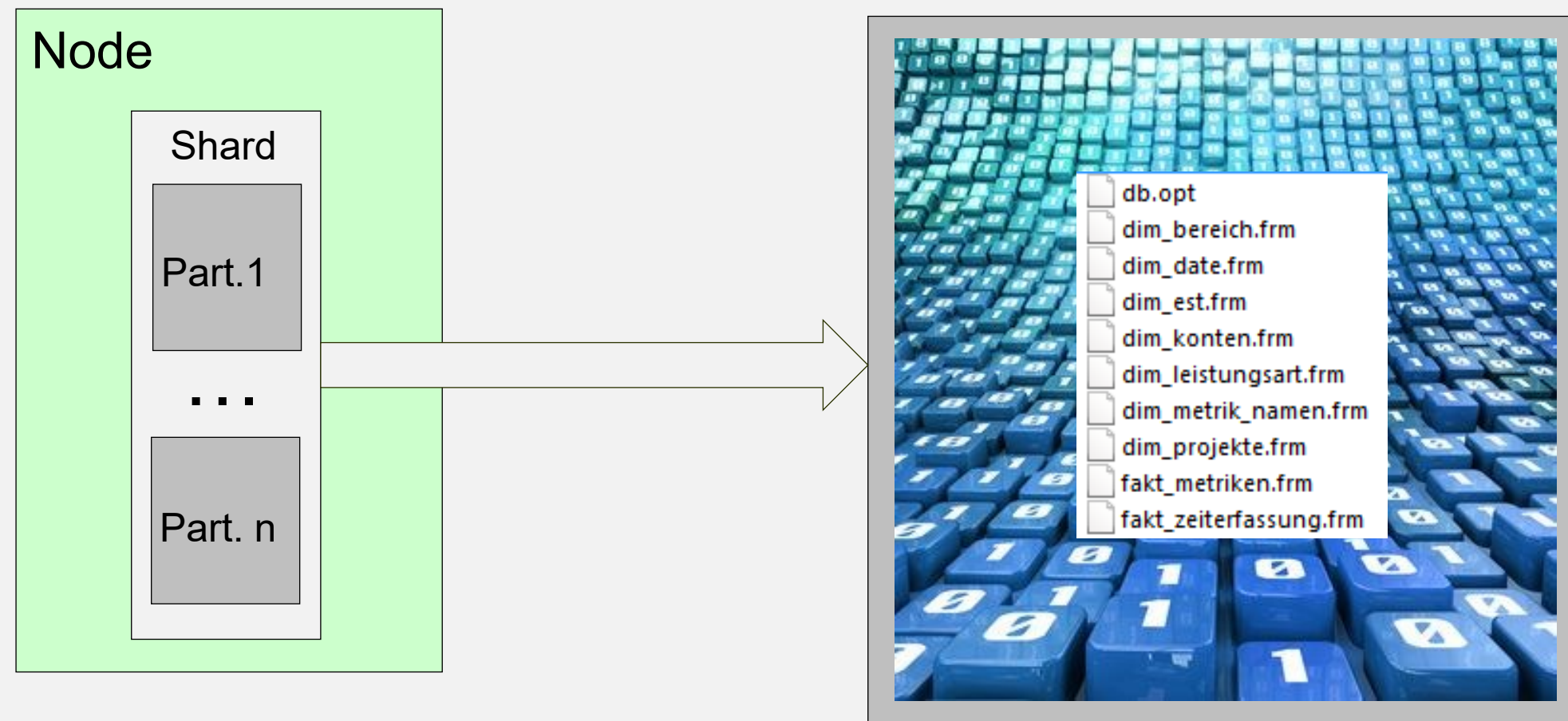
Sharding and Partitioning: Verteilung und Stückelung von großen Datenmengen.



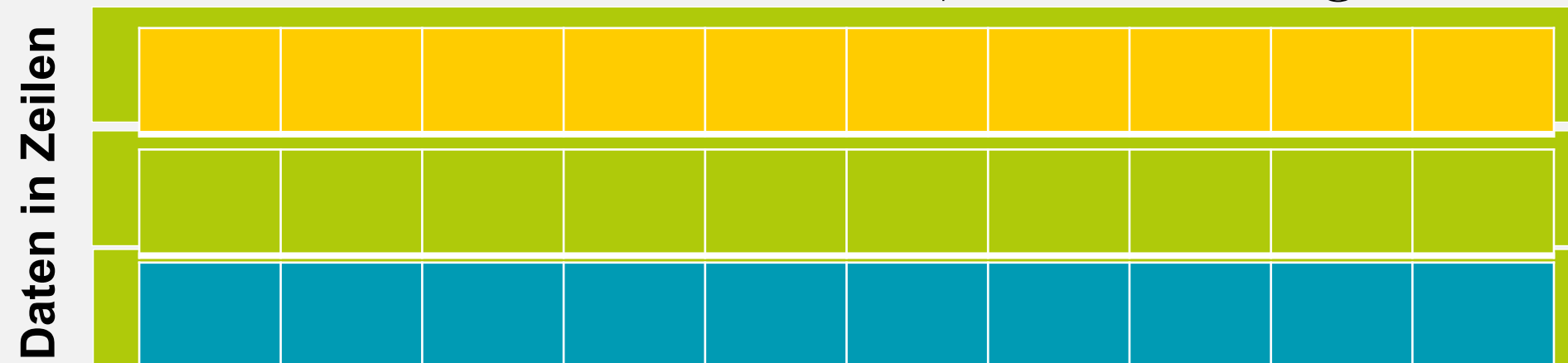
(Re-) Sharding- und Partitioning-Funktion:
 $f(\text{Daten}) \rightarrow \text{Shard}$
 $f(\text{Daten}) \rightarrow \text{Partition.}$
+ Replikationsstrategie.
+ Konsistenzstrategie.

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

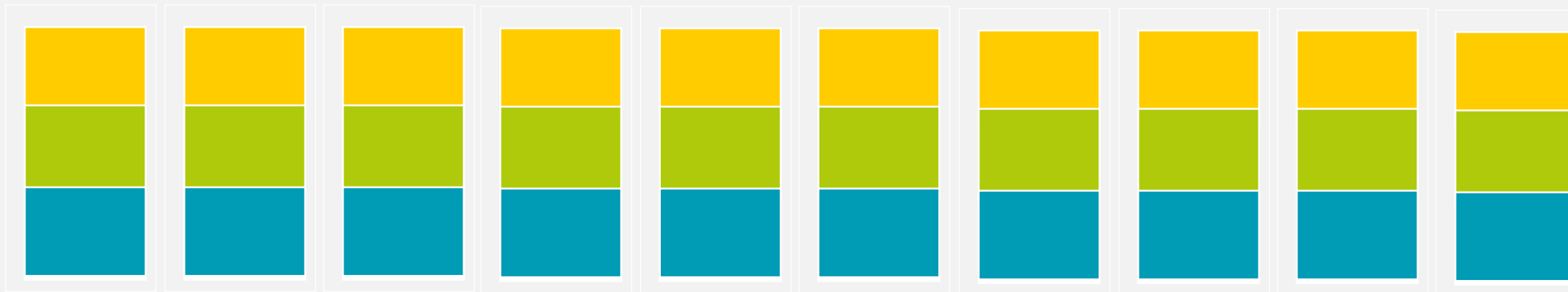
Dateien im (verteilten) Dateisystem



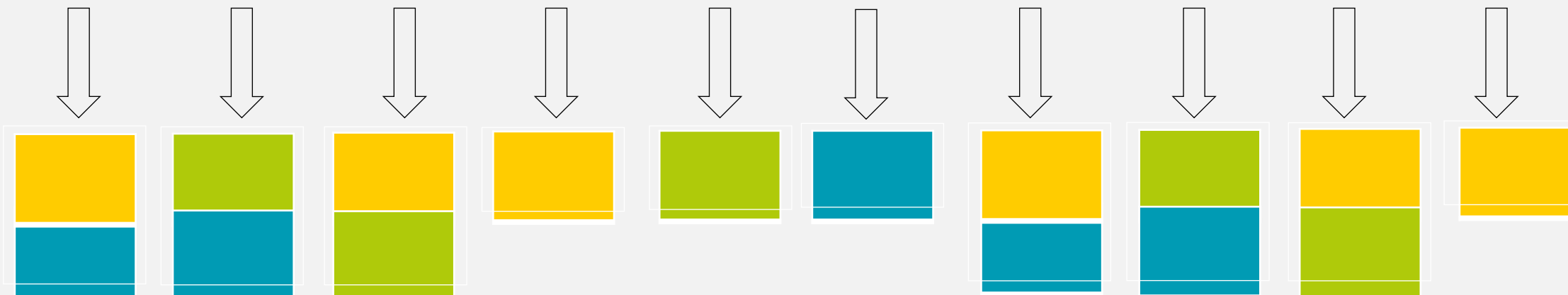
Spalten-orientierte Datenspeicherung.



Daten in Spalten



**Komprimierte
Daten in Spalten**

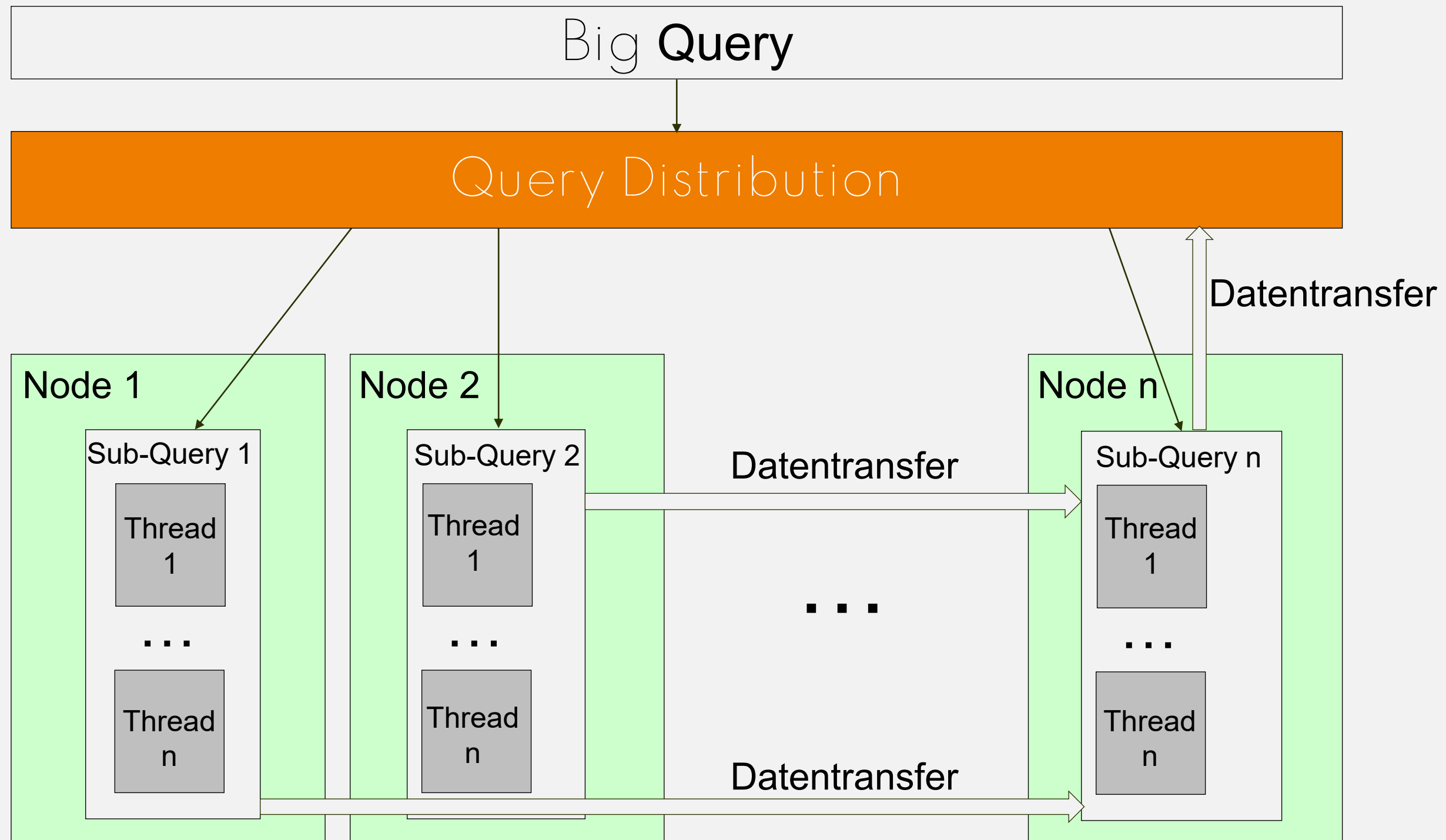


The fastest I/O is the one that never takes place: Es werden nur diejenigen Spalten gelesen, die benötigt werden (gerade bei breiten Tabellen wichtig)

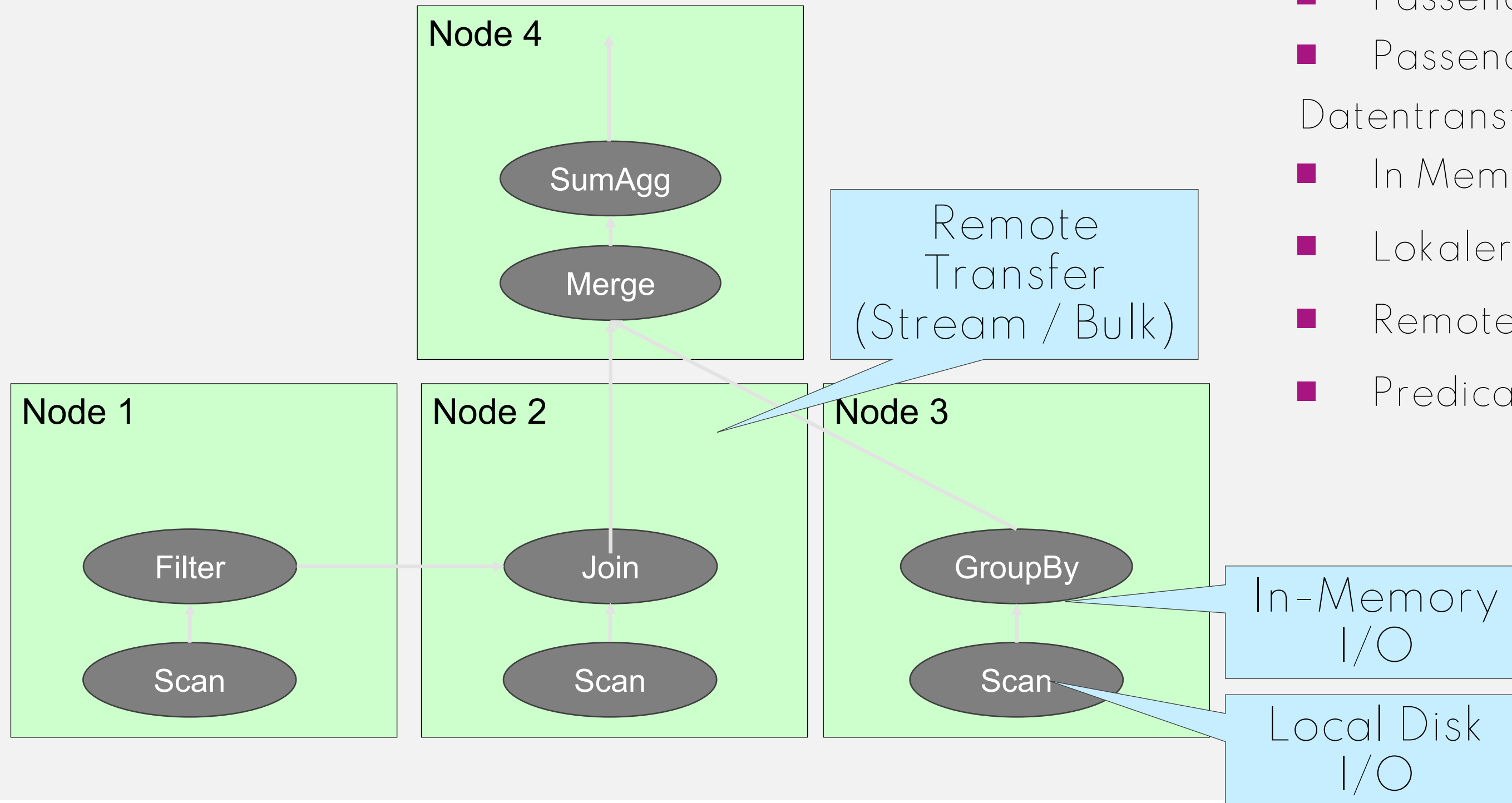
Kompression
(funktioniert bei Spalten besser als bei Zeilen):

- Datentyp-spezifisch (z.B. Dictionaries) + ggf. Spalten-Index

Verteilte und parallelisierte Ausführung von Abfragen.



Ein verteilter Ausführungsplan: Ein azyklischer Funktionsgraph.



Logik folgt den Daten

- Passende Sharding-Funk
 - Passende Partitioning-Fu
 - Passende Replikation
- Datentransfer-Optimierung:
- In Memory vor ...
 - Lokaler Disk I/O vor ...
 - Remote-Transfer.
 - Predicate Pushdown.

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)

Further reading / viewing

Vortrag "Consistency, Availability and Partition tolerance in practice – A deep dive into CockroachDB"

- <https://www.slideshare.net/QAware/consistency-availability-and-partition-tolerance-in-practice>

Vortrag "Neues aus dem Tindergarten: Auswertung "privater" APIs mit Apache Ignite"

@ MRMCD 2018 – Darmstadt

- Video: <https://media.ccc.de/v/2018-151-neues-aus-dem-tindergarten-auswertung-privater-apis-mit-apache-ignite>
- Folien: <https://de.slideshare.net/QAware/neues-aus-dem-tindergarten-auswertung-privater-apis-mit-apache-ignite>