

qaware.de



QA|WARE
SOFTWARE ENGINEERING

Big Data

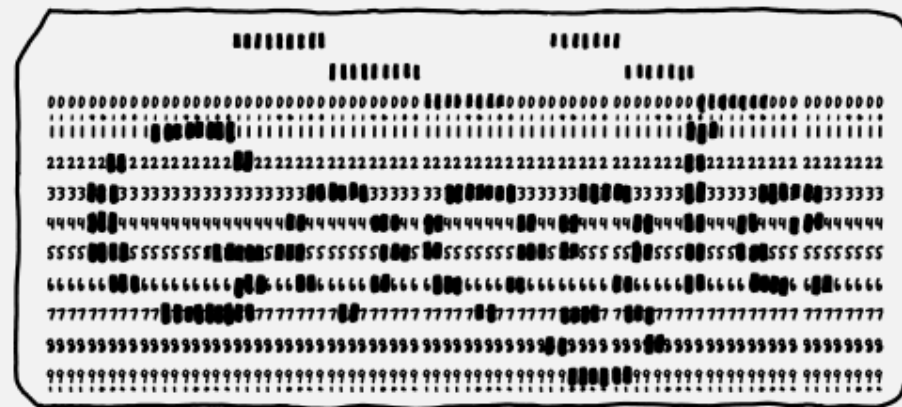
Franz Wimmer
franz.wimmer@qaware.de

Big Data

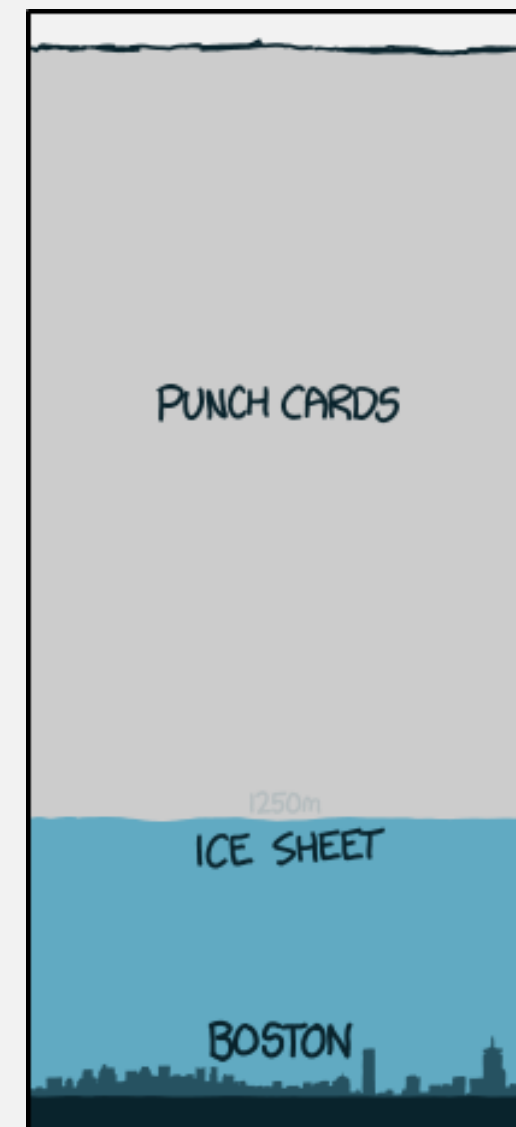


QA|WARE

“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=I64CQp6zOPk&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”

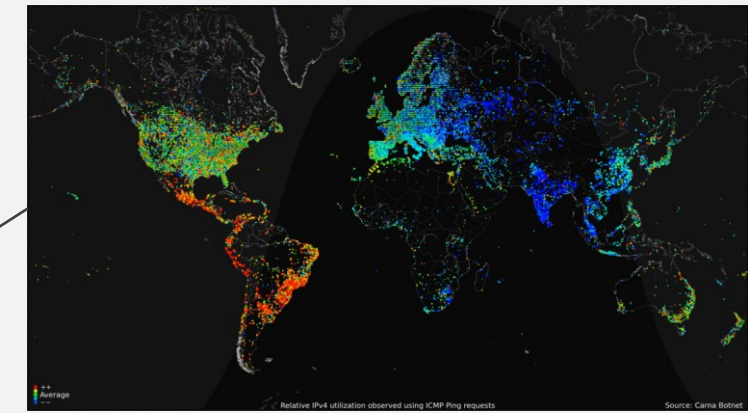
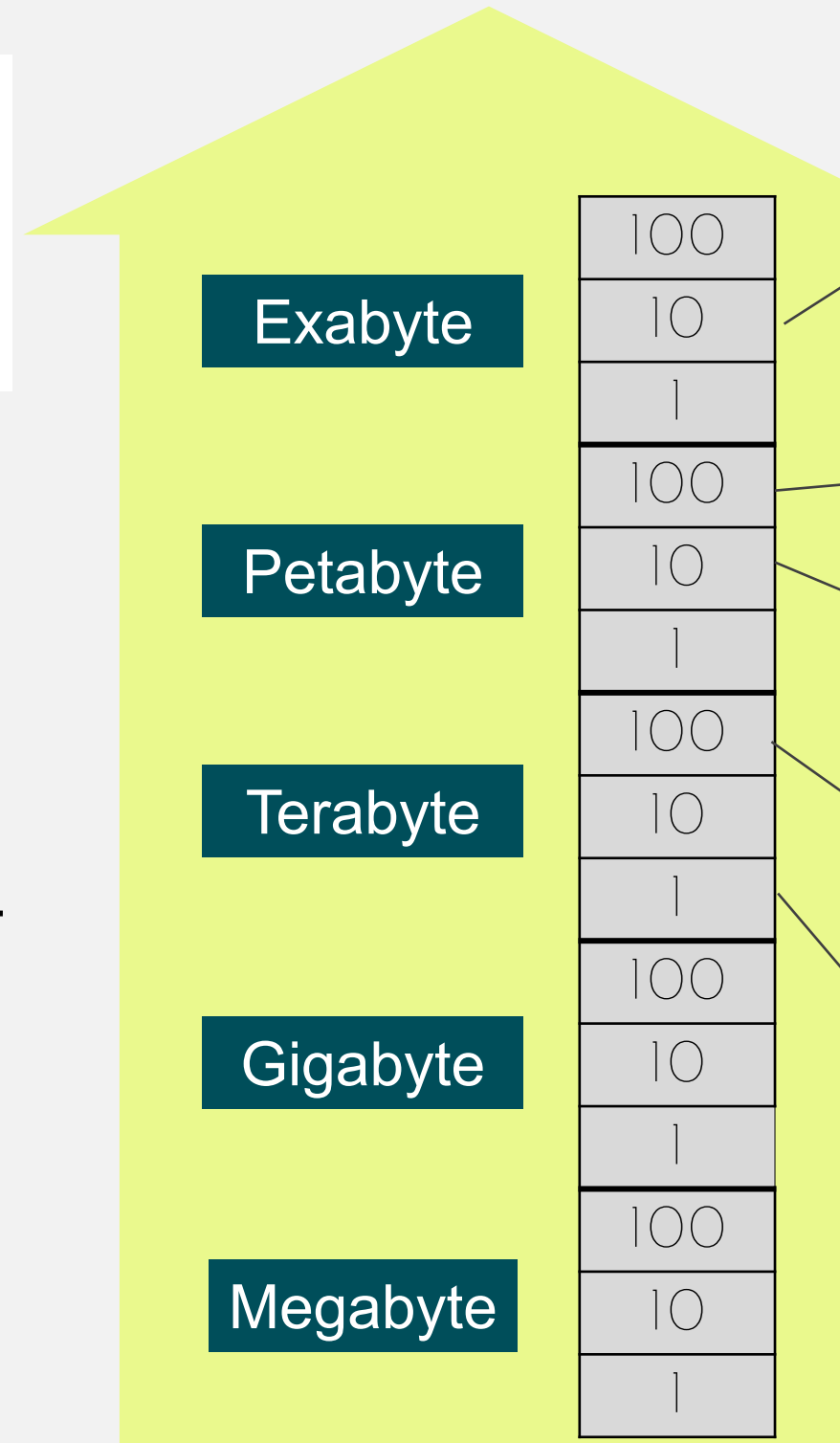
Quelle: . S. Ward und A. Barker. Undefined by data: a survey of big data definitions. arXiv preprint arXiv:1309.5821, 2013.

Big Data

Big Data
Verarbeitung
großer
Datenmengen
durch:

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

Big Data



Das Internet



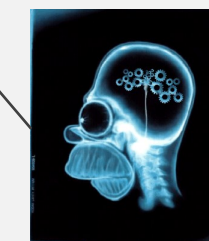
Facebook



Alle Schriften
der Menschheit



Der Google Index



Speichervermögen des
menschlichen Gehirns

DATA & AI LANDSCAPE 2019

INFRASTRUCTURE

HADOOP ON-PREMISE
cloudera Hortonworks
MAPR Pivotal
IBM InfoSphere
jethro

HADOOP IN THE CLOUD
aws Microsoft Azure
Google Cloud
SAP Cloud Platform
IBM InfoSphere
arm
Duple CAZENA

STREAMING / IN-MEMORY
Amazon Kinesis
SAP Cloud Platform ORACLE
confluent
strim hazelcast
GIGASPACE
WallaData
FASTDATA
loc

NoSQL DATABASES
Google Cloud AWS
ORACLE Microsoft Azure
mongoDB MarkLogic
Couchbase DTRITHA
redislabs
ArangoDB SCYLLA

NewSQL DATABASES
SAP Cloudera
Pivotal
Microsoft Azure
MEMSQL InfluxDB
Cockroach Labs
VOLTDB splice
perasoft

GRAPH DBs
Amazon Neptune
IBM
ORACLE
InfoGraph
Dgraph

MPP DBs
TERRA DATA
IBM InfoSphere
Cobion
Kognitio
Exasol
dremio
Infoworks

CLOUD EDW
aws
Google Cloud
Microsoft Azure
Pivotal
snowflake
nudio

SERVERLESS
Amazon AWS
Microsoft Azure
Pivotal
snowflake
nudio

DATA TRANSFORMATION
talend pentaho
alteryx TRIFACTA
tamr Paxata
StreamSets UNIFI

DATA INTEGRATION
SAP Data Services
Informatica
MuleSoft
segment
ZALONI
Import.io
Infoworks
Plexicon
snowflake
MATILLION

DATA GOVERNANCE
Informatica
IBM
SAP
collibra
dremio
Alation
OKERA
HANTA
dataworld

MGMT / MONITORING
aws New Relic
actrio
rubrik
dynatrace
SignalFx
splunk
Mogix
unimetric
Numerly
VEGAM
ZENPLOT
OptiTemp
MAGNITUDE

STORAGE
aws Google Cloud
Microsoft Azure
PURE STORAGE
ALLUXIO
woolabi
nimble storage
Qumulo
pantheon
COHERITY

CLUSTER SVCS
Amazon AWS
Microsoft Azure
Google Cloud
IBM
SAP
Hortonworks
Pivotal
InfoGraph
Dgraph

DATA GENERATION & LABELLING
amazon mechanical turk
upwork
appen
scale
HIVE
Labelbox
Mighty AI
ALABEVEE
LIONBRIDGE

AI OPS
ALGORITHMIA
control
Verta.ai
datmo
datastronaut
Playbook
Determined AI
fidler

GPU DBs & CLOUD
kinetica
IBM
Microsoft Azure
bryllyt
BLAZINGO
PG-Stream
FLOYDLB

HARDWARE
Google TPU
ARM
NVIDIA
GRAPHICORE
MYTHIC
Hailuo
Ahabana
VIAVIVE
CERULEAN
SPLIGHT
DEPINX

CROSS-INFRASTRUCTURE/ANALYTICS

aws Google Cloud Microsoft IBM SAP HCL Hewlett Packard Enterprise SAS 1010DATA VMware TIBCO TERRA DATA ORACLE NetApp syncsort MAPR cloudera

ANALYTICS & MACHINE INTELLIGENCE

DATA ANALYST PLATFORMS
Microsoft pentaho alteryx
Digital Reasoning
GUAVUS
AYASDI
ATTIVIO
Datameer incorta
interana
MODE
ENDOR
sisu
switchboard
Starburst

DATA SCIENCE PLATFORMS
IBM databricks dataiku
DOMINO rapidminer TIBCO
ANACONDA SAS
KNIME MathWorks

BI PLATFORMS
looker
Amazon AWS
ATSCALE
Qlik
MicroStrategy
Koen IO

VISUALIZATION
+tableau
Google Cloud
celonis
zapl
CHARTIO

MACHINE LEARNING
Google Cloud
Amazon AWS
H2O
gamalor
VISENZE
ELEMENT

COMPUTER VISION
Microsoft Azure
Amazon Rekognition
clarifai
mia
EVER AI
deepomatic
neura
twentybn
UBIQUITY
ADEE
YIYU
trax
synthesis
Qubarc

HORIZONTAL AI
IBM Watson Cortana
sentient
Voyager
Affective
ZEPHYRUS
Hanaata
PETUM
SI
NANOLOGICS
OSARO

SPEECH & NLP
Google Cloud
Amazon AWS
twilio
Amazon Translate
semantic machines
Moby
Open Technologies
SoundHound Inc.
PRIMER
cogito
scrips
DYNAMIC
Unlabeled
Polyglot

SEARCH
elasticsearch
ORACLE
algolia
COVEO
Lackworks
ATTIVIO
swiftype
EXALING
alphasense
MAANA
omnius
SINEQUA

LOG ANALYTICS
splunk
sumologic
solarwinds
tiemker
kibana
logz.io

SOCIAL ANALYTICS
Hootsuite
sprinkr
NETBASE
synthesio
trackx
smilereach
bitty
SimilarWeb

WEB / MOBILE / COMMERCE ANALYTICS
Google Analytics
mixpanel
Amplitude
Airtable
RESOI
SIGOPT
granify
custora

APPLICATIONS - ENTERPRISE

SALES
CHORUS
INSIDESALES.COM
people.ai
conversica
clari
avisio
tactai
fuse/machines

MARKETING - B2B
RADIUS
App Annie
EVERSTRING
Lattice
MINTIGO
sense
tubular
ZENAGIO
KNOTCH
mrpe

MARKETING - B2C
ZETA
bloomreach
SendGrid
brage
ACTIONIQ
BLUECORE
CONTINUOUSLIVE
TRAILBLAZE
Amplero
empathy
Simon
PERSADO
remesh

CUSTOMER EXPERIENCE / SERVICE
qualtrics
MEDALLIA
SurveyMonkey
zendesk
Kustomer
freshdesk
HEAP
Amplitude
DigitalGenius
ASAPP
ads
AUTOMAT
afiniti
CodaCheck
HUBSPOT
Frame

ENTERPRISE PRODUCTIVITY
slack
ORACLE
GURU
lumiata
DIFFBOT
clari
talla
Kasisto

HUMAN CAPITAL
HireVue
pymetrics
hiQ
textio
WoodsWendy
Stella
entelo
uncommon
beotery

LEGAL
RAVEL
Everlaw
disco
JUDICATA
BREVIA
IRONCLAD
PRISM
ROSS
CASETEXT

REGTECH & COMPLIANCE
Agio
TESSAN
text IQ
Comply Advantage
PARTNERSHIPS
CRUSHEAM
DATA SUM
DATA REBUG

FINANCE
Anaplan
ZUORO
SAHANA
TRADESHIFT
SCALE FACTOR
batkeeper
pilot

BACK OFFICE AUTOMATION & RPA
UiPath
Blueprints
VIDADO
AppZen
Workfusion
workato
Catalysis
ANTWORKS
KRYON
ALKYNI

SECURITY
TANUM
CYLANCE
Zscaler
StackPath
illumio
CODE42
CipherCloud
DARKTRACE
ANOMALI
Vectra
Sift Science
pindrop
exabeam
SICINYO
SentinelOne
SecurityScorecard
SOCURE
CodeSecure
Bitglass
BlueTalon
Securix
feedzai
Cyber
BTSIGHT
BlueHexagon
Semele
ORION
XDRIVE
SHIELD
Armorblox

APPLICATIONS - INDUSTRY

ADVERTISING
AppNexus
criteo
xAd
Integral
ORACLE
MOAT
OpenX
theTradeDesk
Adaptix
dstillery
LiveIntent
TAPAD
datax
gumgum
Optix

EDUCATION
edX
Knewton
Clever
edexra
kidapptv
PANDORA
knowre
gradscope

REAL ESTATE
REDFIN
Opendoor
VTS
CREDIFI
GEOPHY
reonomy
COMVSTAK
SPACEMARKER

GOVT
OPENGOV
mark43
Proton
LiveStories
Passport
SmartProcure
STREETLIGHTDATA
OpenCatalist

INTELLIGENCE
Palantir
Dataminr
Quid
PRIMER
FORGE

FINANCE - INVESTING
KENSHC
Quantopian
AODEPAR
ISENTIUM
ALORIX
FlowerPuck
PAGAYA

FINANCE - LENDING
ondeck
affirm
JIANPU.AI
Kreditech
AVANT
aure
CLEARBANC
upstart
100Credit
WeLab
WeCredit
TrueAccord
MoneyLion
ActiveAI
aire
aghi

INSURANCE
Insuramix
Anomix
CYENCE
Helo
Shift Technology
ROOT
zestyai
TRACABLE
CAPE

HEALTHCARE
Flatiron
Clover
Kyrus
Habitat
METABOTA
Gingerio
Glow
babyon
3D Med
zebra
PathA
ovig
TEMPUS
patientscience
AICure
insitro
notable
Heron
HECURE
prognos
@nitic
img
BioCrux
BAYLABS
Qventus
ARTERY
HAGEN
innovacor
PAGES
DATAWAVE
JoanLab

LIFE SCIENCES
BenevolentAI
verily
WUXNextCODE
Clear Labs
Havenoma
CytomX
CNAnexus
CITRINE
twoSTAR
Cytoscape
DANON

TRANSPORTATION
UBER
TESLA
WAYMO
ZOOX
CLEARPATH
CRUISE
NURO
AIGO
nuro
drive.ai
CAMBRIDGE
Aurora
NIO
OPTIMUS
moovit
Ilu
nexas
Kodiak
comma.ai
netradyn
Civil Maps
thinc
INRIX

AGRICULTURE
FARMERS
Granular
JOHN DEERE
BLUERIVER
FarmersEdge
AgroStar
FarmLogs
TARANIS
GAMAYA
terrotrans
prospero

COMMERCE
Instacart
FAIR
STITCH FIX
D&G
PetaNet
HowGood
heurtich
OTHER
ehamony
stem
Amper
ByteDance
happn
collect
SOJERN
BIXEVER
VERDIGIS
duetto
Jadebeck
Electric
ZINER
Spoke

INDUSTRIAL
AVEVA
SIEMENS
PREDIX
UPTAKE
SCORTEX
TACHYUS

OPEN SOURCE

FRAMEWORKS
Apache Spark
Flink
YARN
TEZ
HADOOP
KUBERNETES
K8S
Docker
Kubernetes
HELM

QUERY / DATA FLOW
Spark
SQL
Presto
SLAM DATA
Apache Drill
GraphQL
Flink

DATA ACCESS & DATABASES
cassandra
mongoDB
redis
Cockroach Labs
druid
Cockroach Labs
druid
SciDB
Flink
Flink
Flink
Flink

ORCHESTRATION & MGMT
talend
Apache Airflow
Wesos
etcd
Kong

STREAMING & MESSAGING
Spark
nifi
Flink
beam
kafka
storm
Apache RocketMQ

STAT TOOLS & LANGUAGES
Python
Scala
R
Julia
Stata
SAS
MATLAB

AI OPS & INFRA
mlflow
Kubeflow
DVC
Seldon
Pai

AI / MACHINE LEARNING / DEEP LEARNING
TensorFlow
Keras
theano
DM TK
OpenAI
VELES
neon
DSSTNE
mlib
MAHOUT
Aerosolve
miller

SEARCH
elasticsearch
Solr

LOGGING & MONITORING
elasticsearch
kibana
SENTRY
logstash
Prometheus
fluentbit
fluentd
Grafana

VISUALIZATION
matplotlib
TensorBoard
seaborn

COLLABORATION
Buckaroo
Jupyter

SECURITY
Apache Ranger
KNOX
Sentry
Securify

DATA SOURCES & APIs

HEALTH
Apple
VALIDIC
practicefusion
fitbit
GARMIN
HUMAN API
kinsa

IOT
GE Digital
UPTAKE
thingworx
helium
samsara

FINANCIAL & ECONOMIC DATA
Bloomberg
THOMSON REUTERS
DOW JONES
SEP CAPITALIQ
CBREIGHTS
PLAID
Gestimize
PREMISE
Quandl
Eagle Alpha
StockTwits
xignite
Thinkbox
earnest
predata

AIR / SPACE / SEA
Orbital Insight
planet
Aerobatics
spire
UNEP STORY
tellusdata
WINDWARD
MarineTraffic

PEOPLE / ENTITIES
axiom
experian
EPSILON
insideView
Crimson Hexagon
BASIS
SAFEGRAPH

LOCATION INTELLIGENCE
FOURSQUARE
mapbox
sense360
playbow
HEXAGON
PlaceIQ
esri
factual
Mapillary
StreetView
cuebiq
A Radar
Open StreetMap

OTHER
DATA.GOV
IMAGENET
Lab41
Kaggle
CRUX
uigrillio

DATA RESOURCES

DATA SERVICES
OPERA
fractal
kaggle
DataKind
innocuous

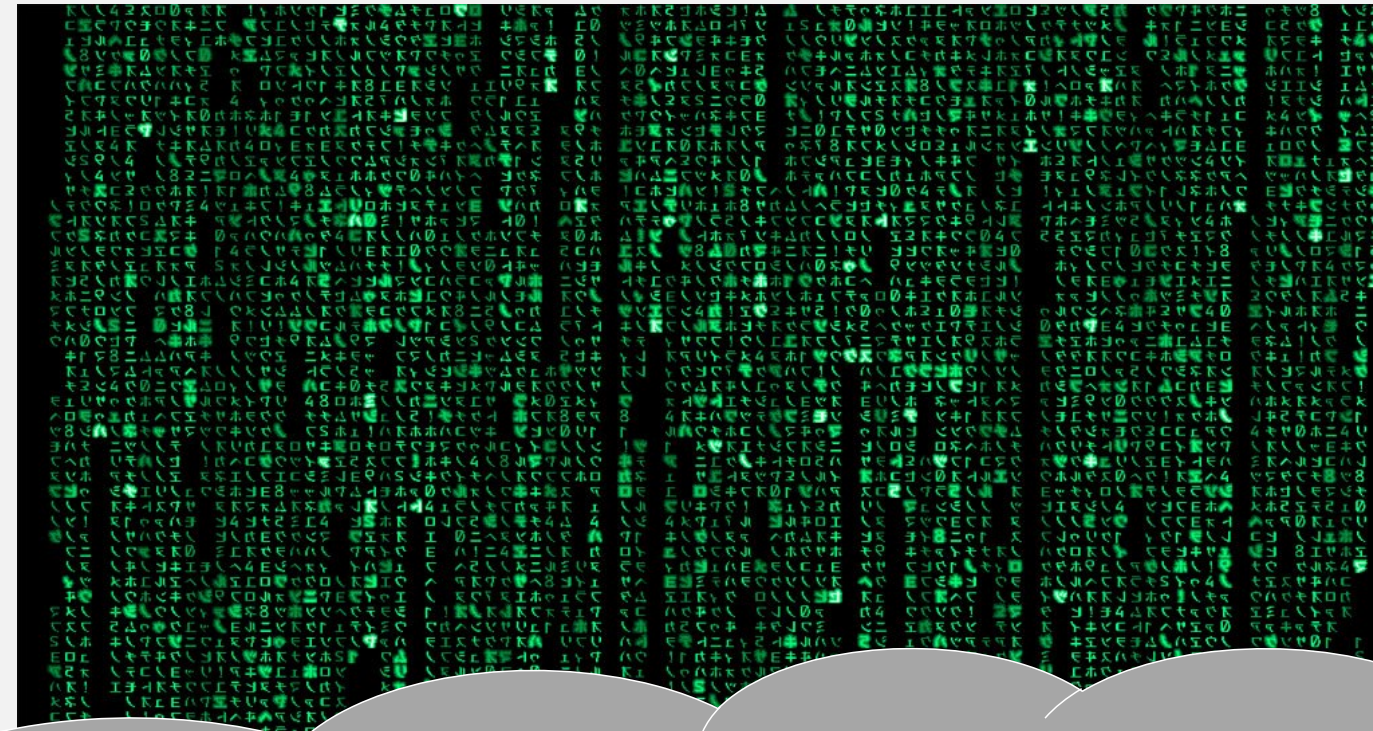
INCUBATORS & SCHOOLS
PLURALIGHT
galvanize
DataCamp
DataElite
INSIGHT
The Data Incubator
METIS

RESEARCH
facebook research
OpenAI
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));
    }
}
```

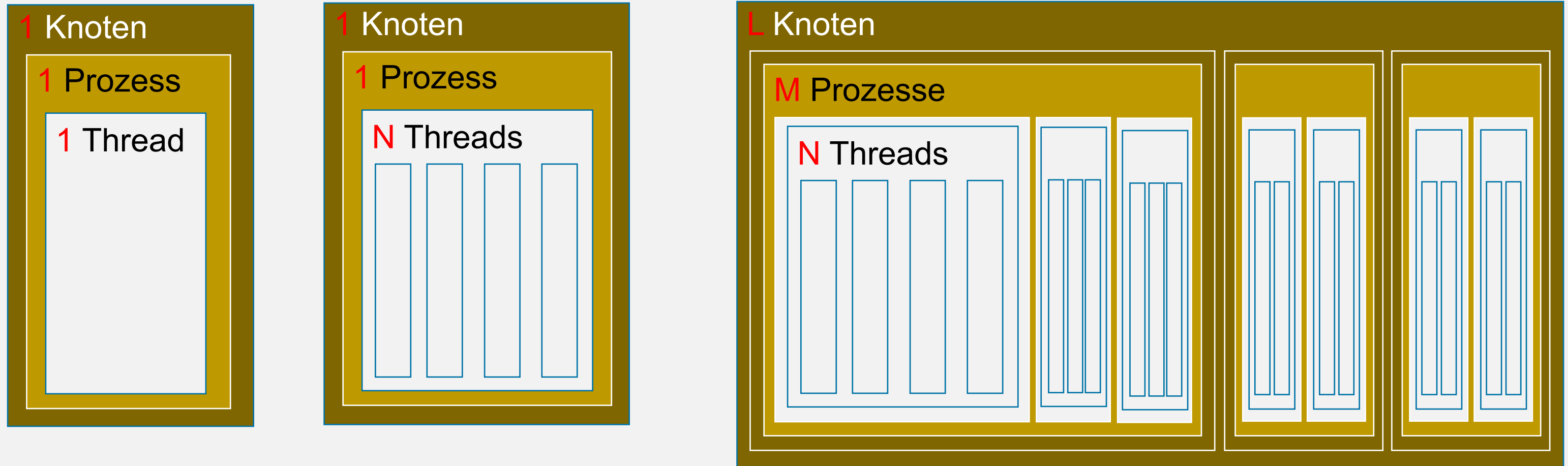
- Funktioniert nicht: Berechnung der Fibonacci-Folge ($F_{k+2} = F_k + F_{k+1}$). Berechnung ist nicht parallelisierbar.

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.

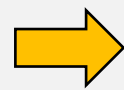
Parallele Programmierung basiert oft auf funktionaler Programmierung.

- 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.



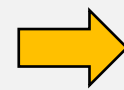
Keine
Parallelität



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

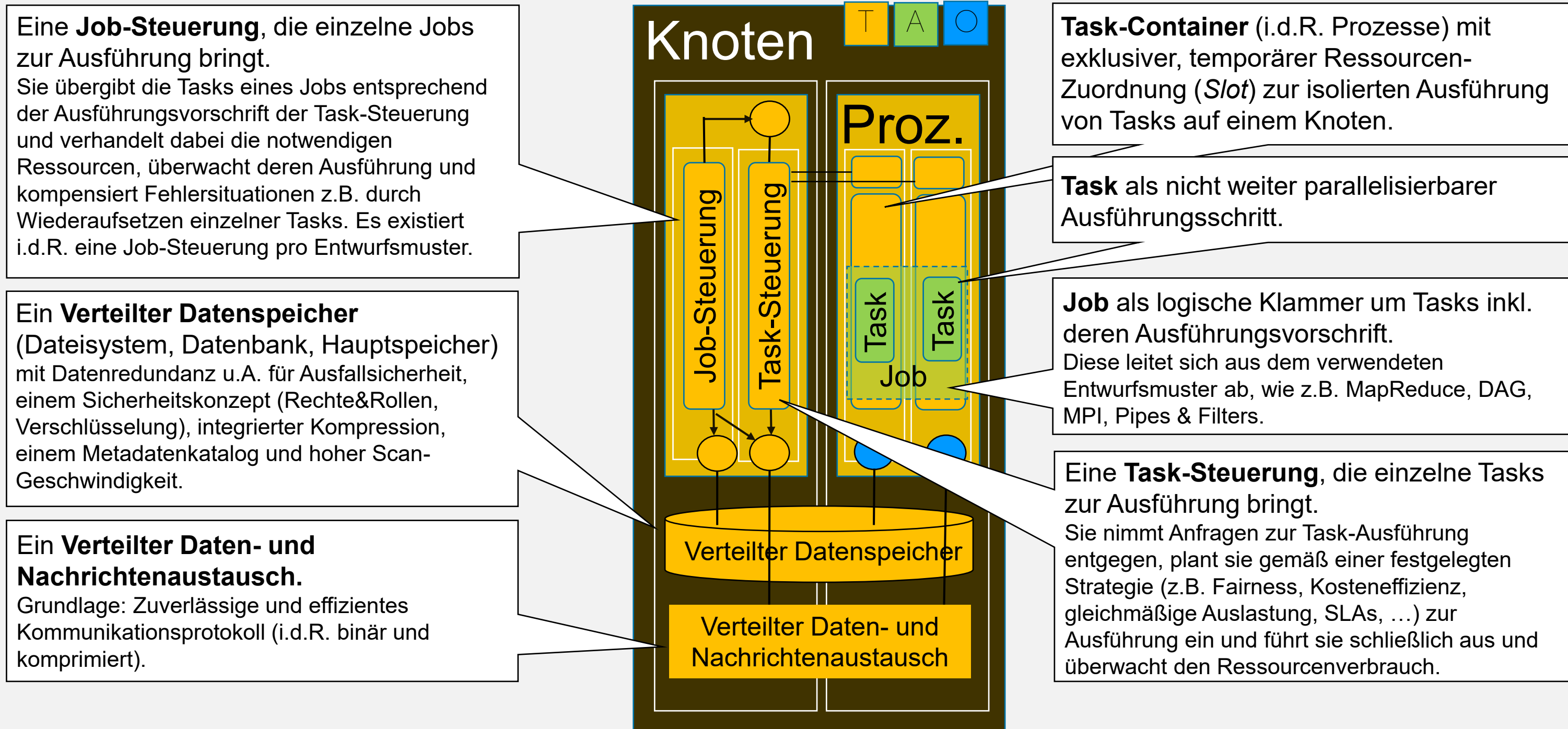
Folgt aus potenziell
großer Datenmenge
und Verarbeitungs-
geschwindigkeit

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.

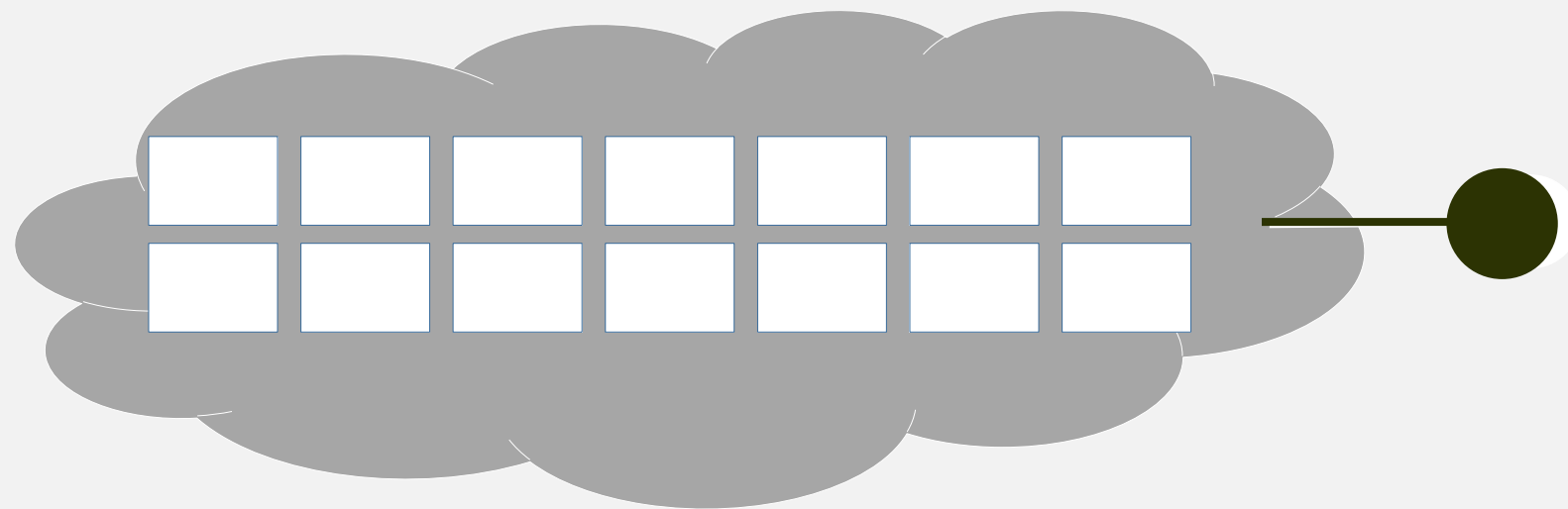
Notwendige Architekturkonzepte

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?



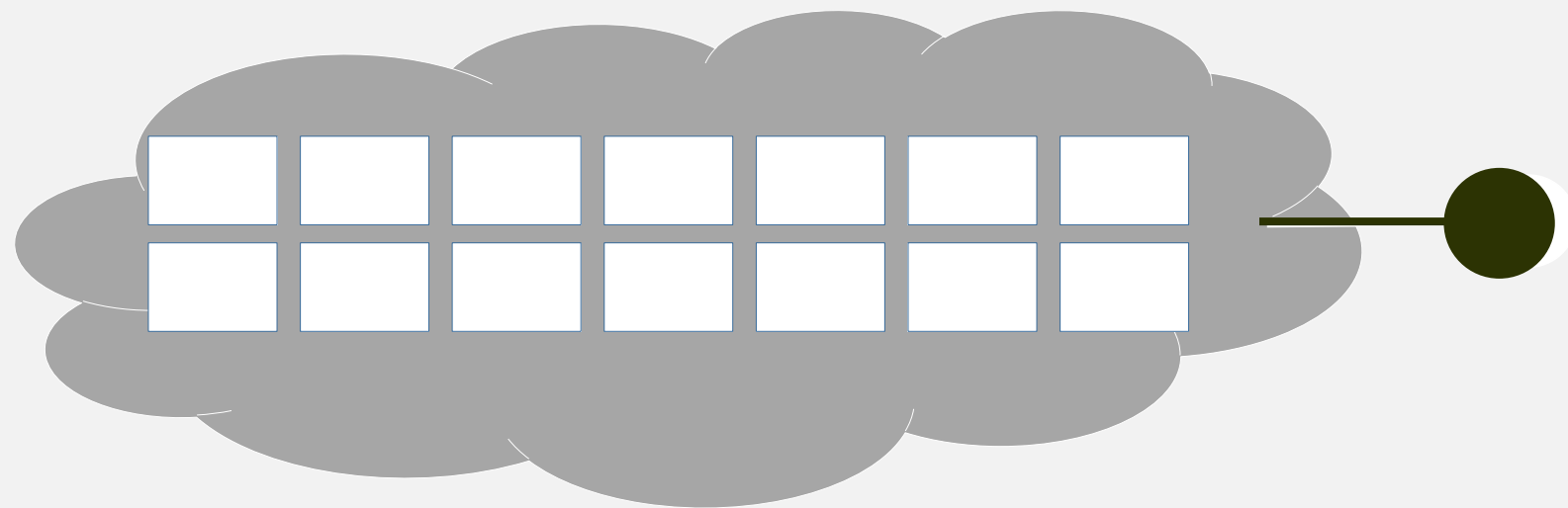
- Big Data Engines (low level)
 - MapReduce
 - RDD (Resilient Distributed Dataset)
- Big Data Datenbanken (high level)
 - NoSQL Datenbanken
 - NewSQL Datenbanken (NoSQL + SQL)
- Verteilte Dateisysteme
- In-Memory Data Grids / Elastic Memory



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.

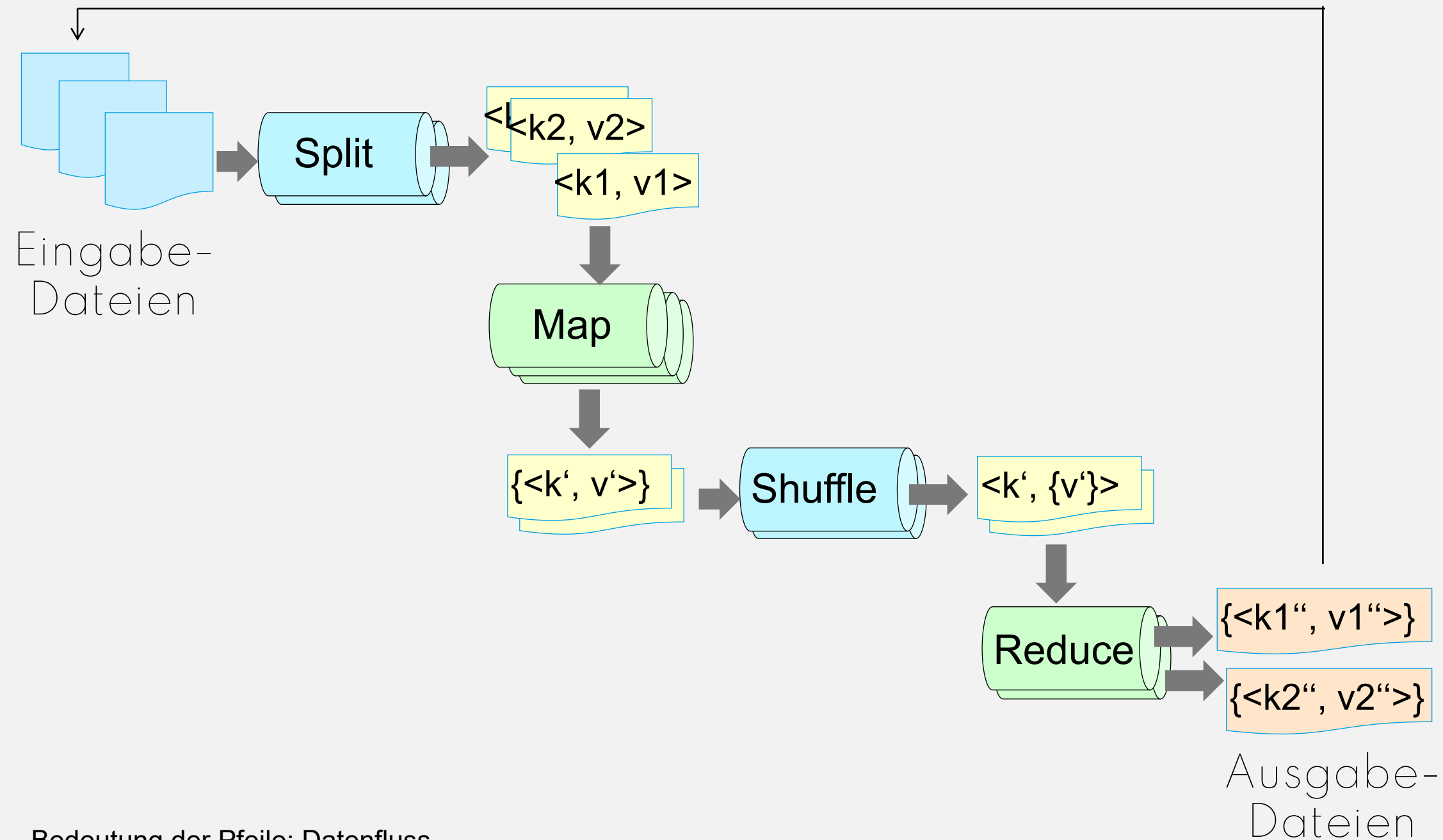
Signatur: **map**(k, v) \rightarrow list(<k', v'>)

- 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: $|\text{list}(\langle k'', v'' \rangle)| \ll |\text{list}(\langle k', v' \rangle)|$

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



Bedeutung der Pfeile: Datenfluss

Die Map-Phase

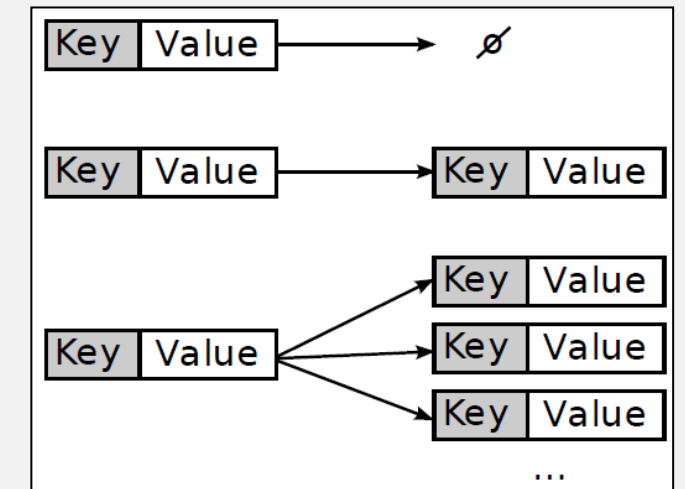
Split

Map

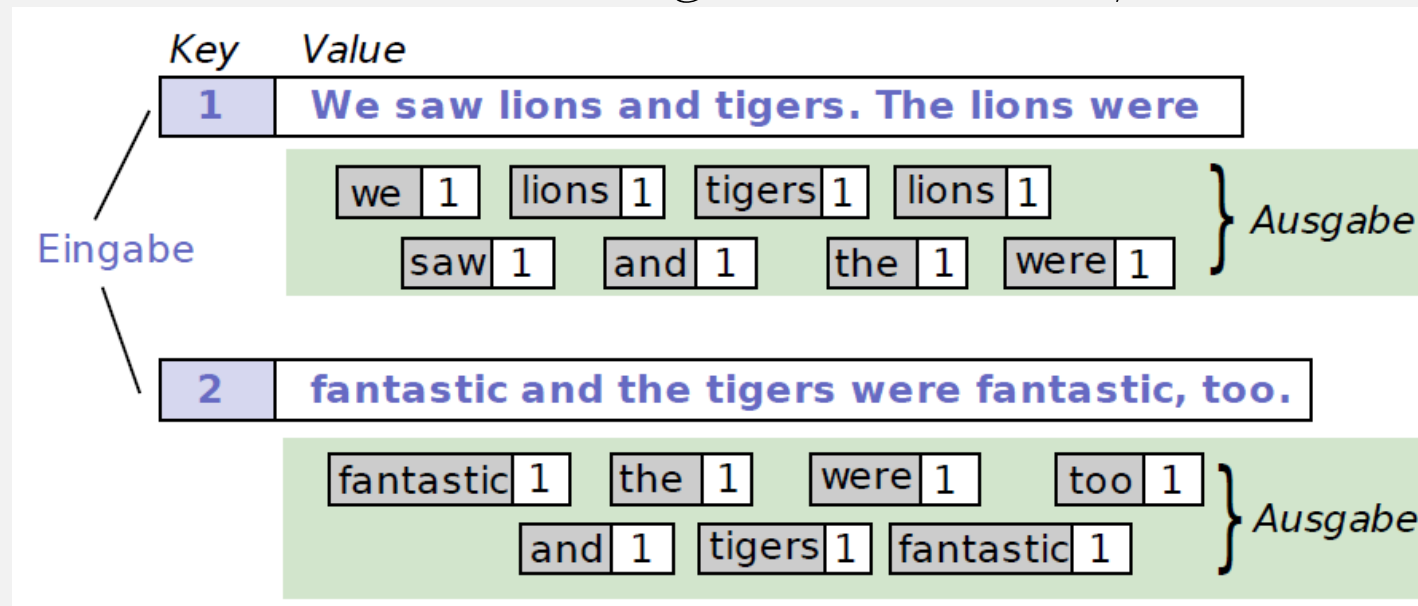
Shuffle

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

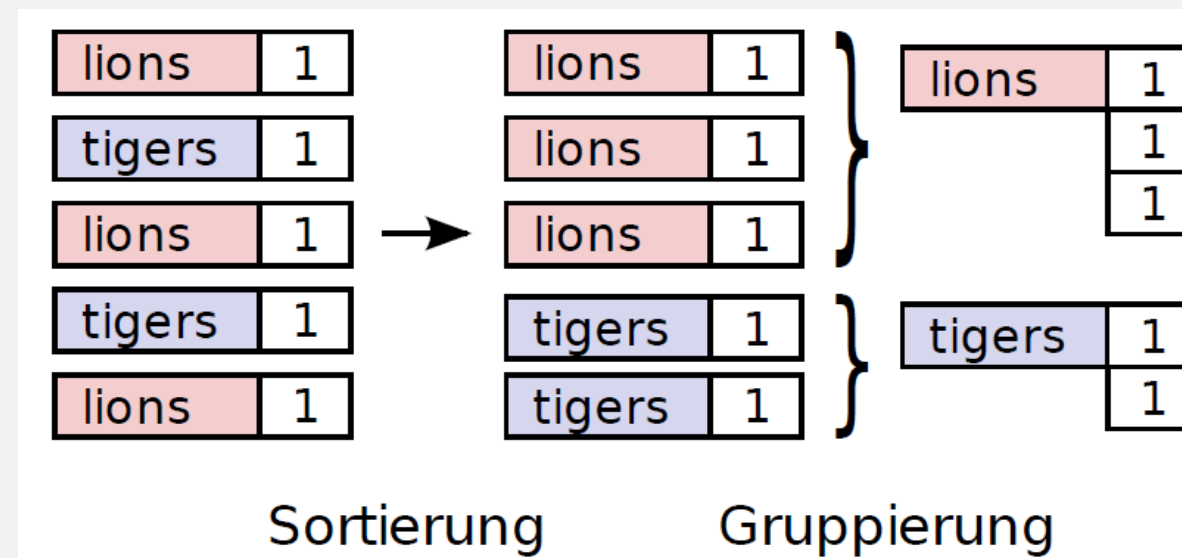
Split

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

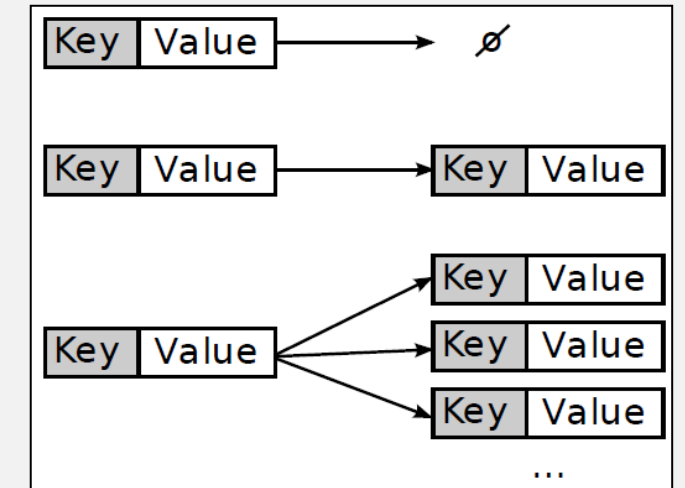
Split

Map

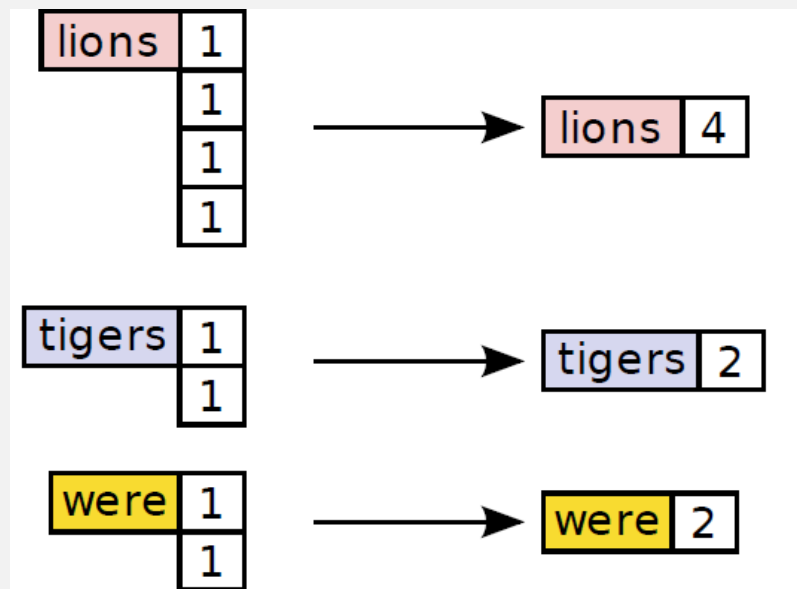
Shuffle

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:



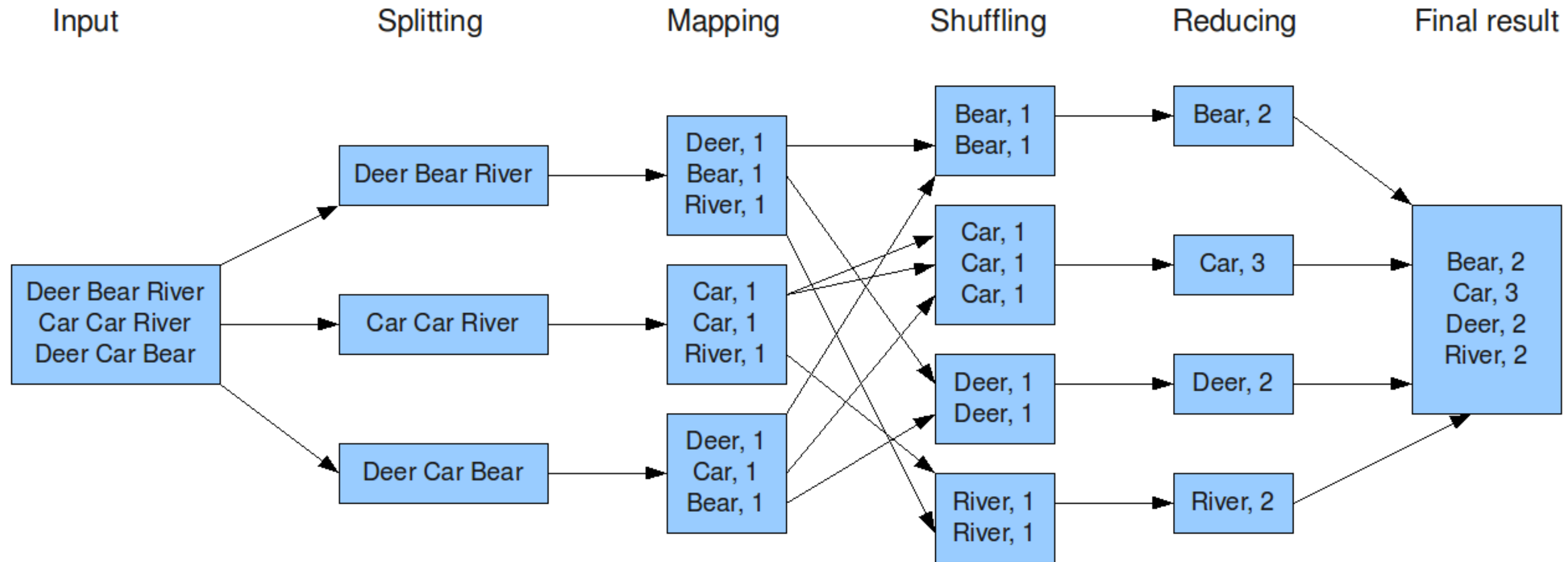
Ein- und Ausgabe der Reduce-Phase:



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) \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>



QA|WARE



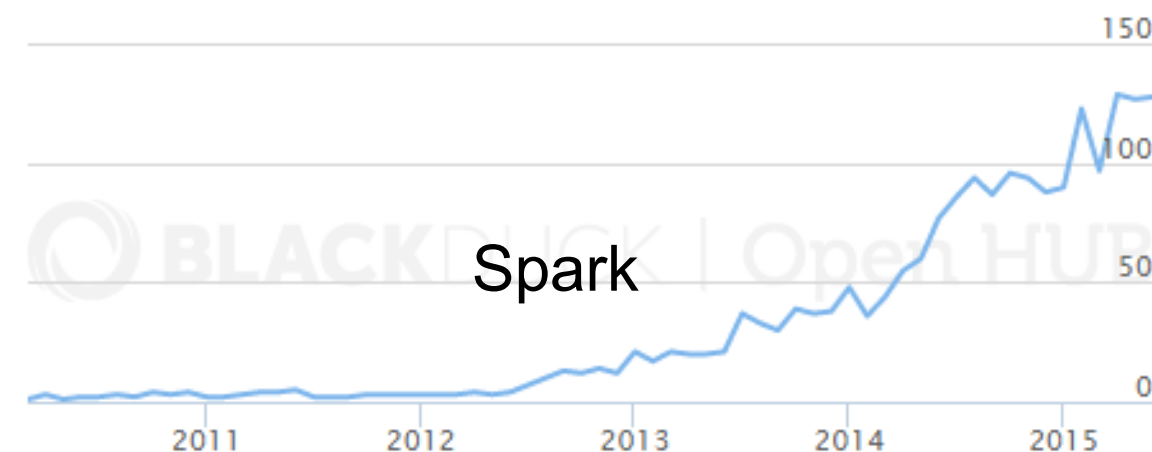
Apache Spark

Spark läuft Hadoop aktuell deutlich den Rang ab.

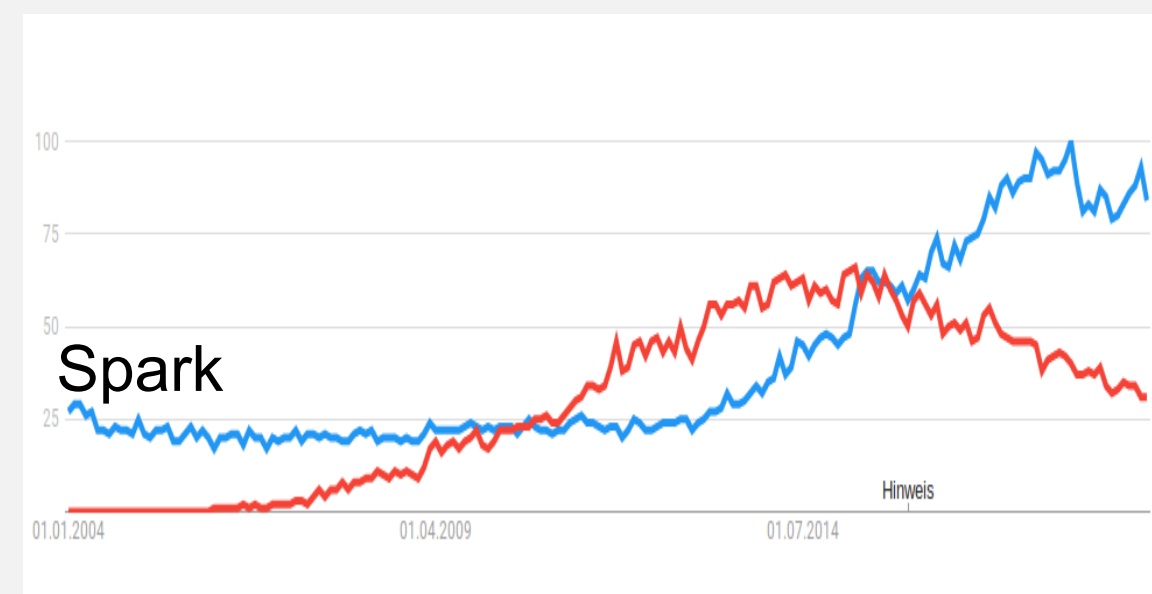
	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated 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.

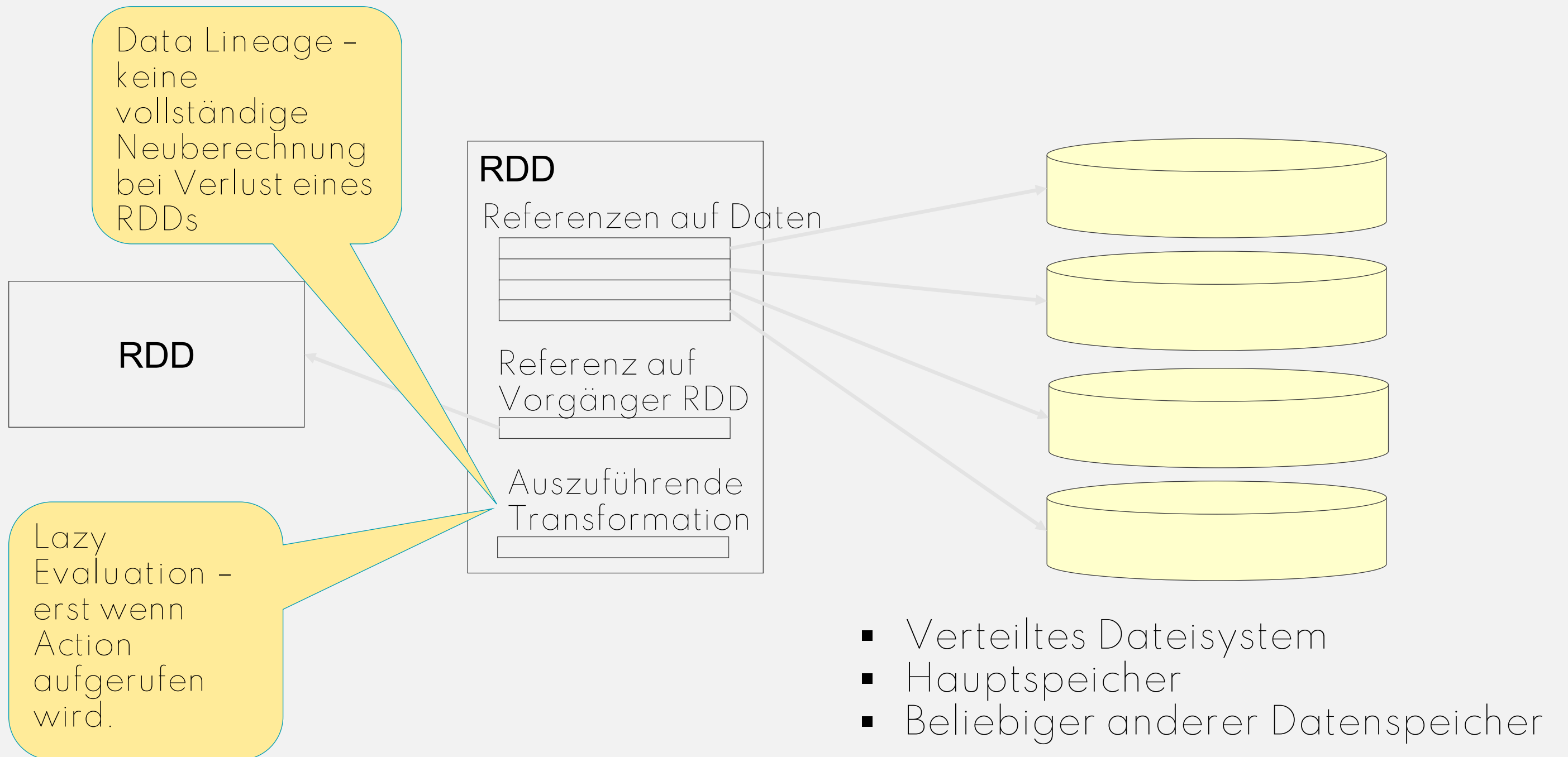
Eine RDD ist in der Außensicht ein klassischer Collection-Typ mit Transformations- und Aktionsmethoden.

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

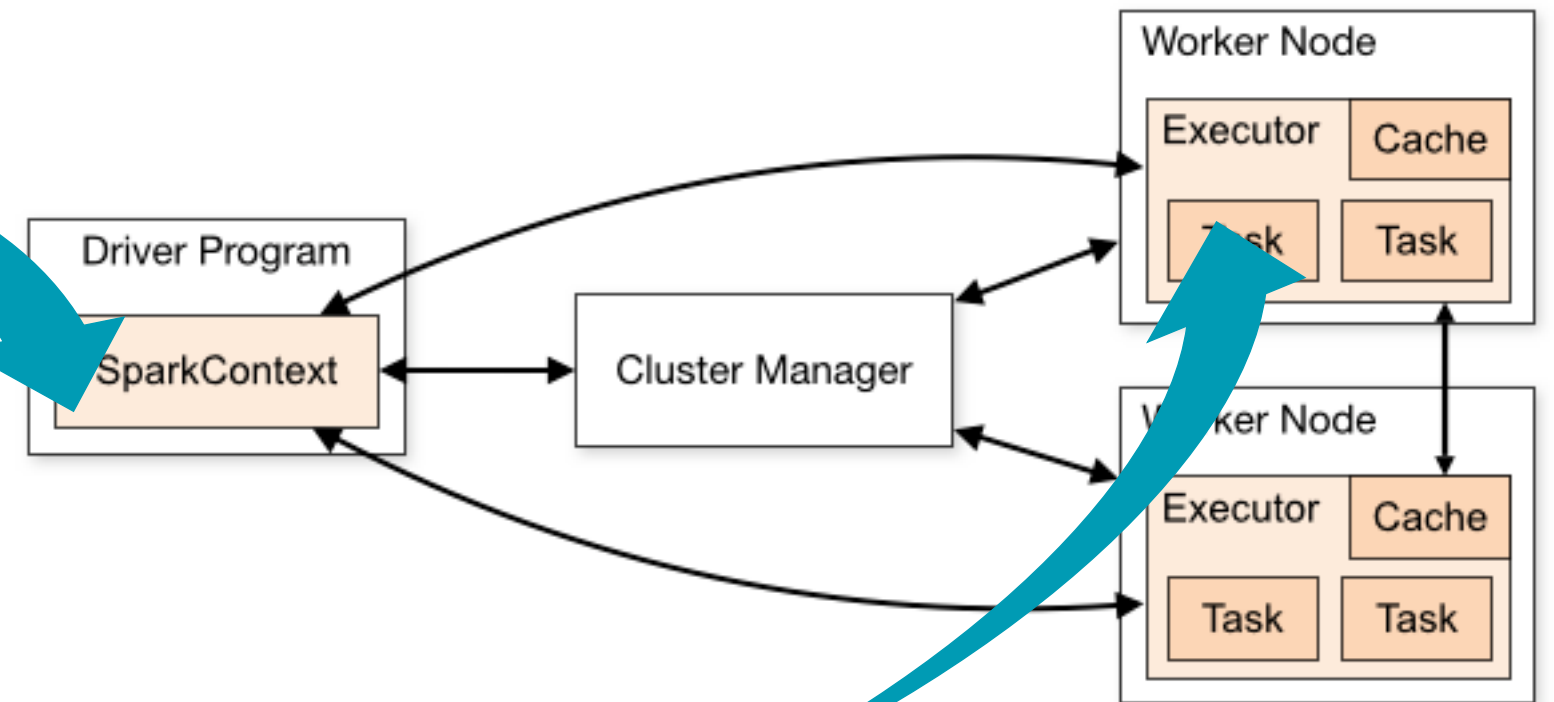
```
val sorted = logData.sortBy(l => l.length)
```

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
  }
}
```





Q|WARE

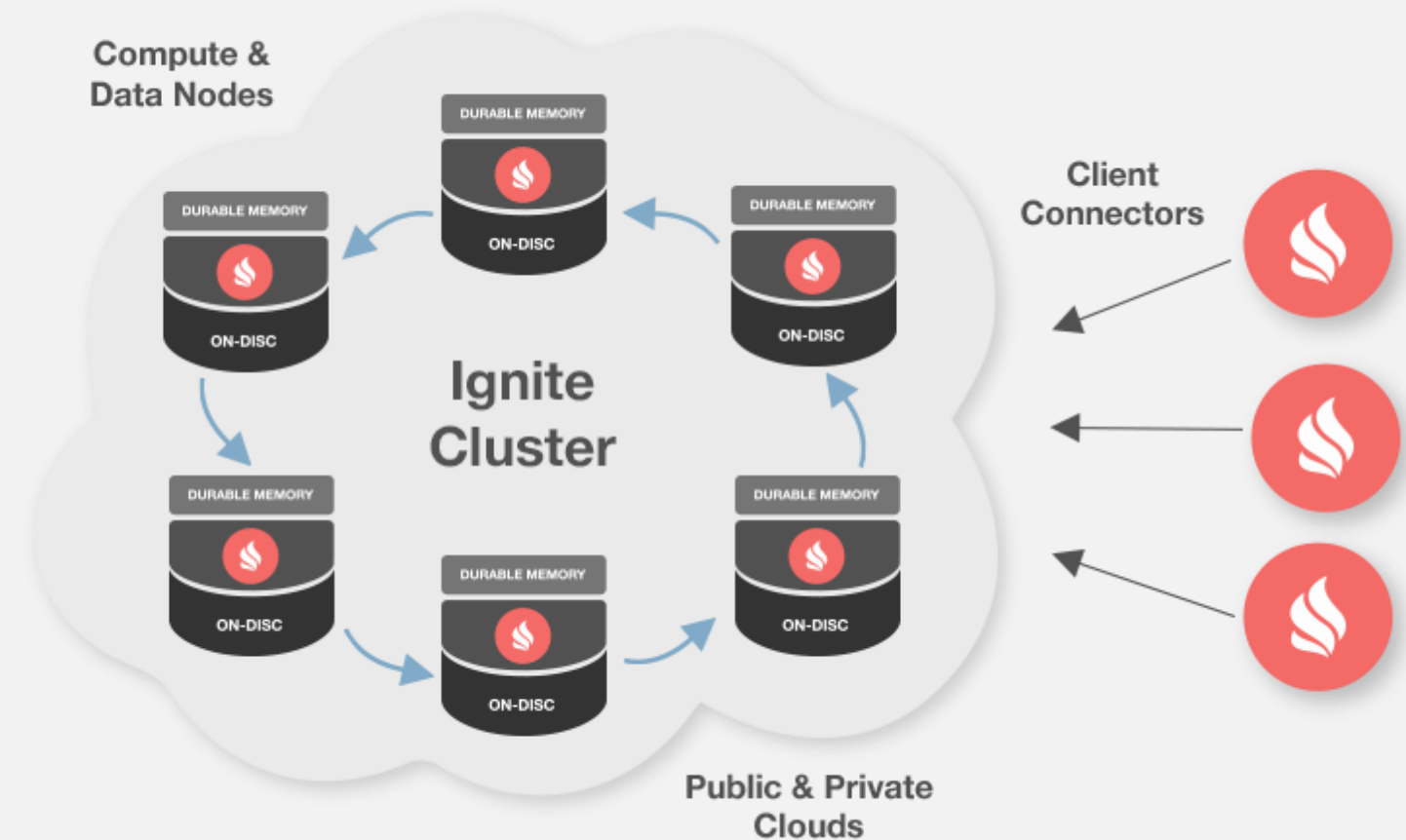
Apache Ignite



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

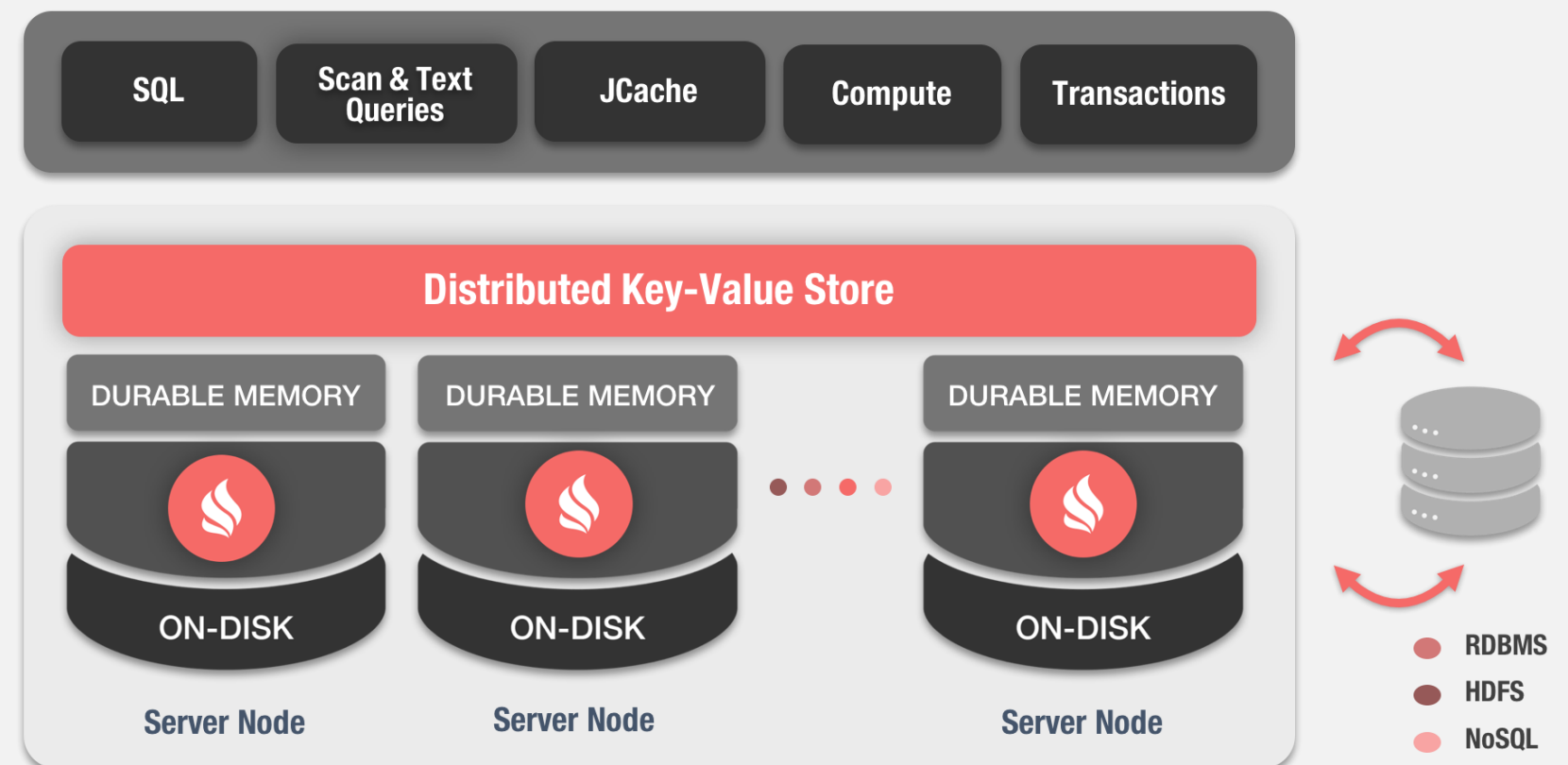
Apache Ignite

- 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();

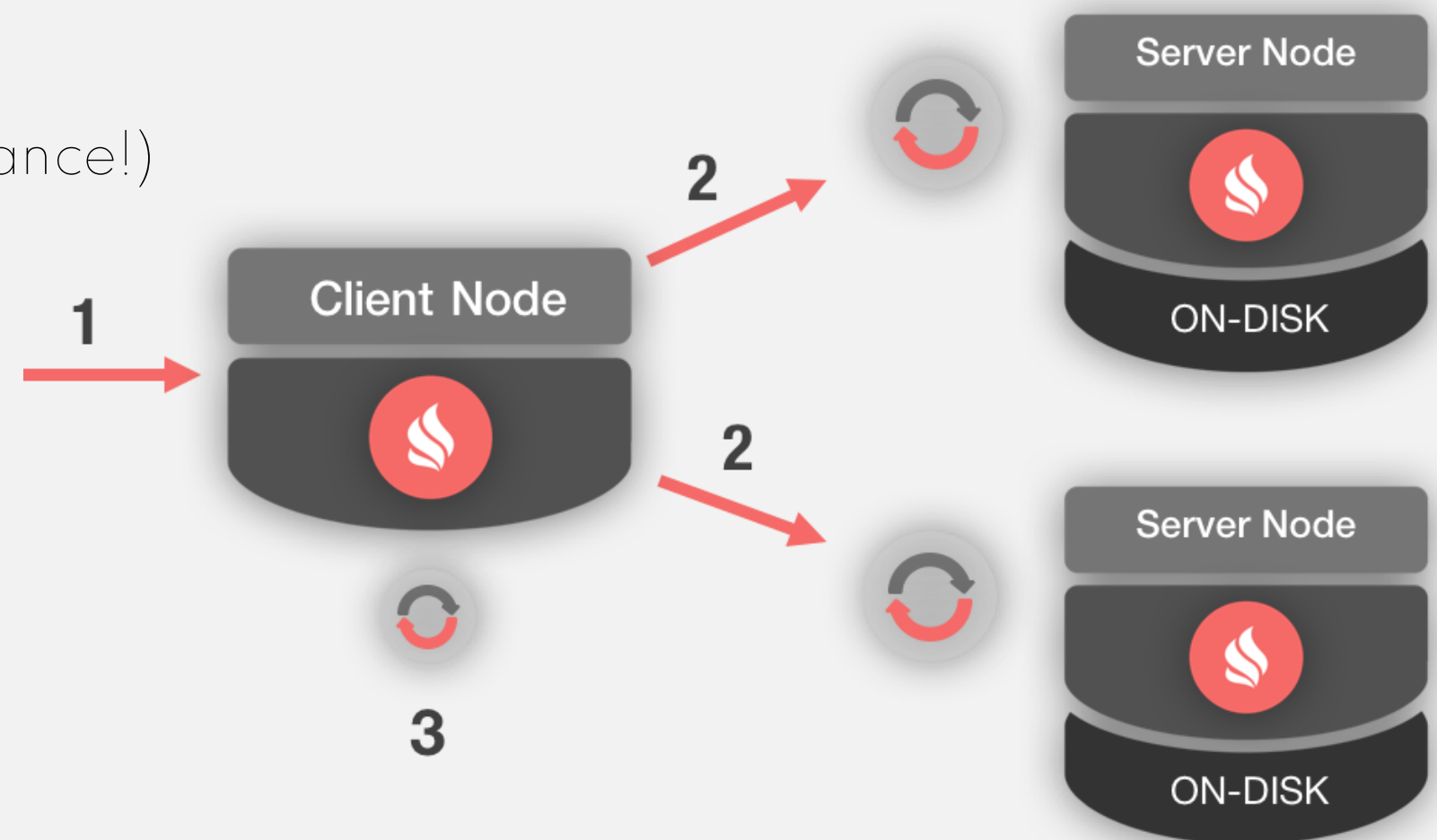
final IgniteCache<Integer, String> cache = ignite.cache("cacheName");

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())) {

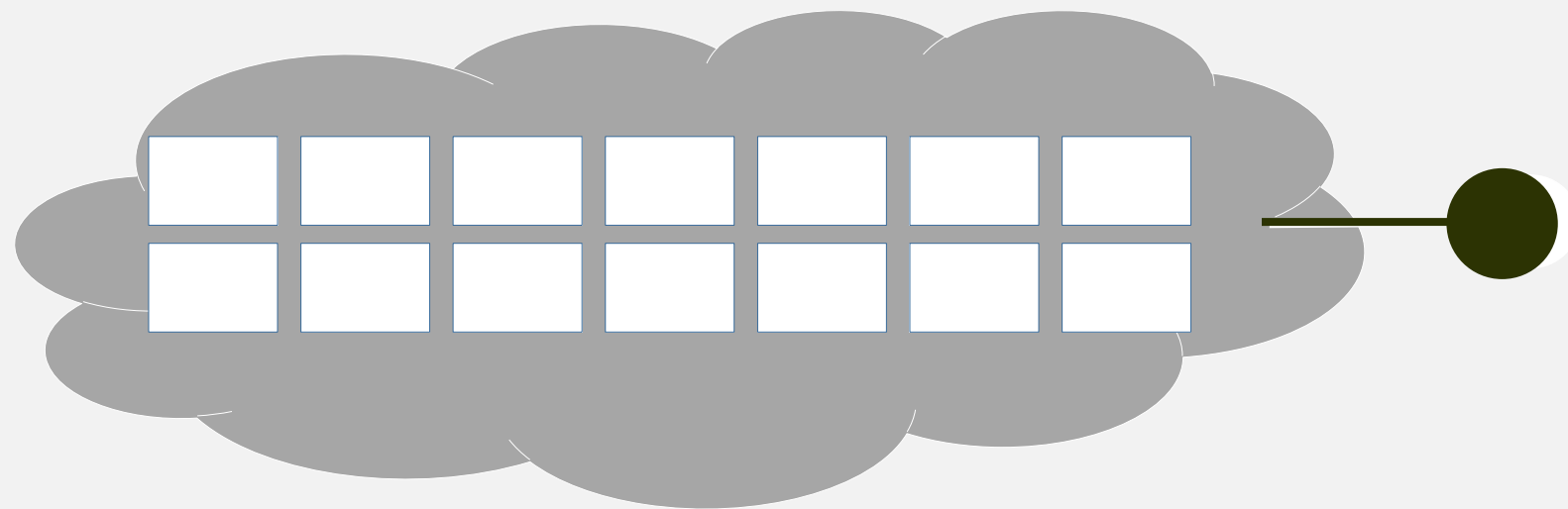
    while(true) {
        String randomWord = RandomStringUtils.randomAlphanumeric(12);
        // Stream words into Ignite.
        streamer.addData(randomWord, randomWord);
    }
}
```



QAWARE

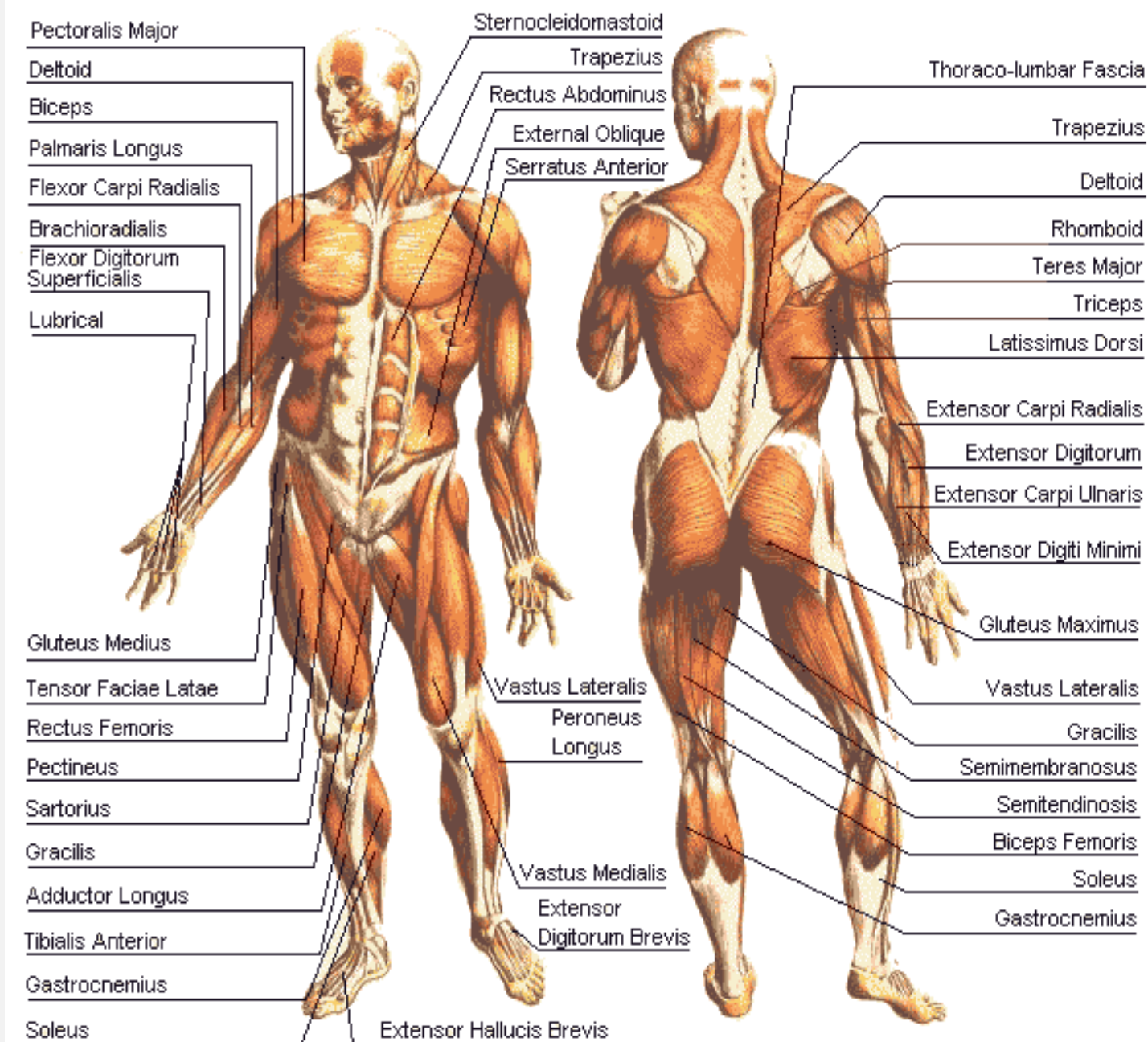
Big Data Datenbanken

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

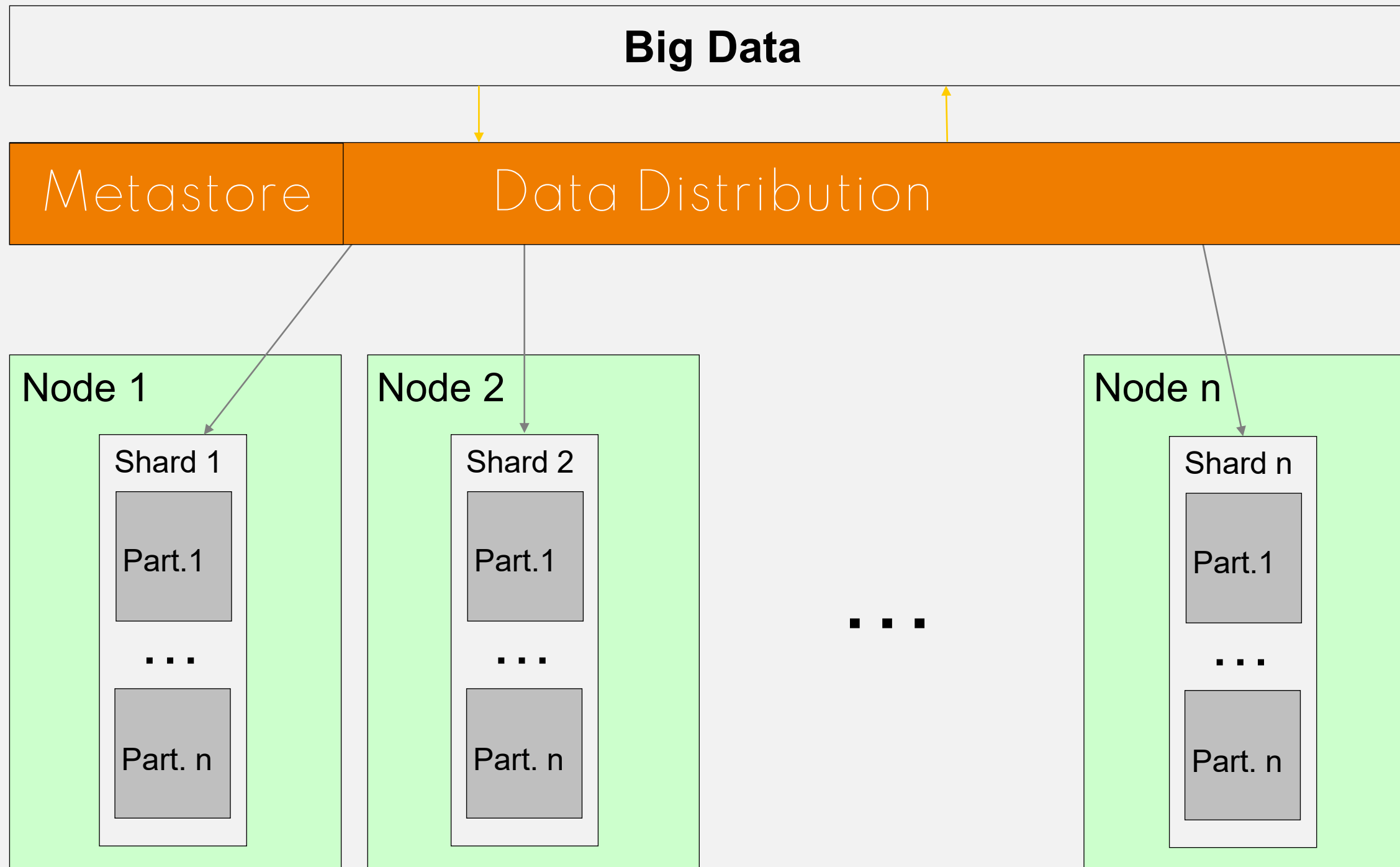


Query Distribution

Data Distribution

Data Persistence

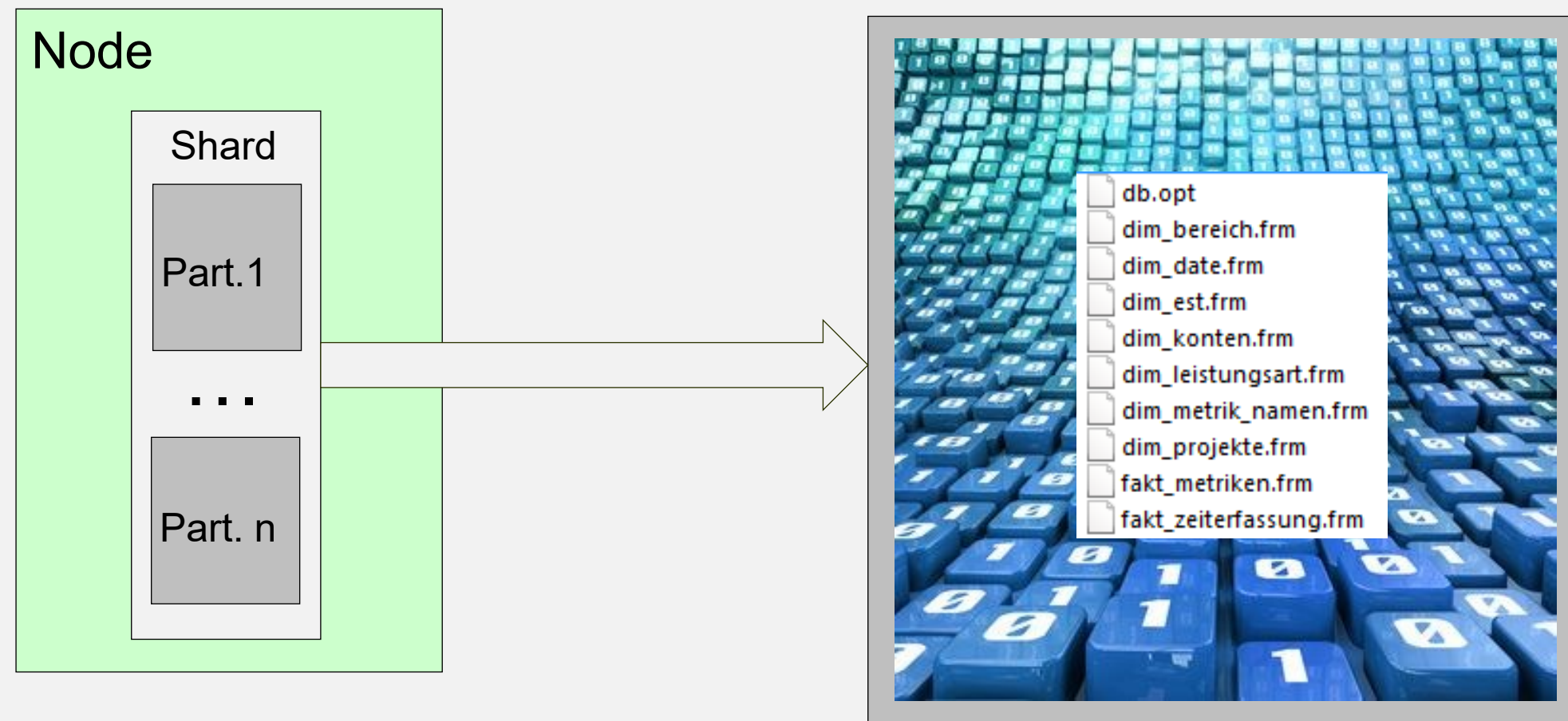
Sharding and Partitioning: Verteilung und Stückelung von großen Datenmengen.



(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?

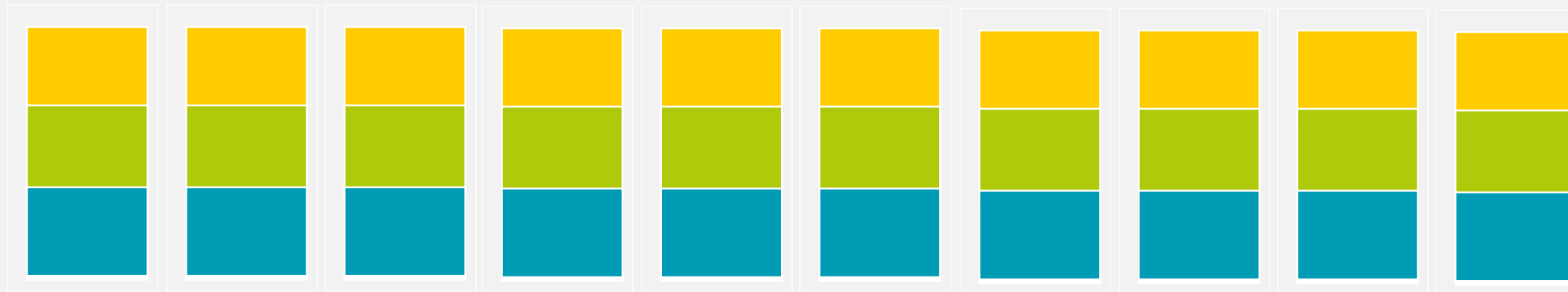
Dateien im (verteilten) Dateisystem



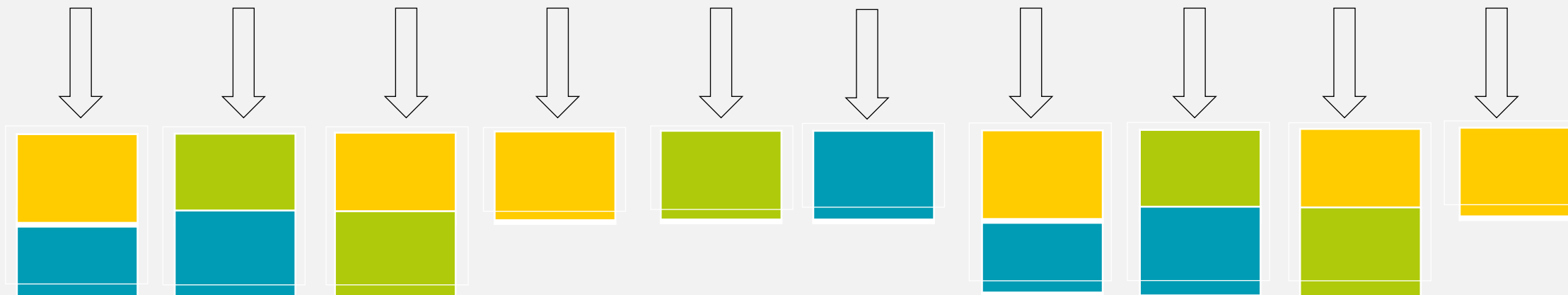
Spalten-orientierte Datenspeicherung.



Daten in Spalten



**Komprimierte
Daten in Spalten**

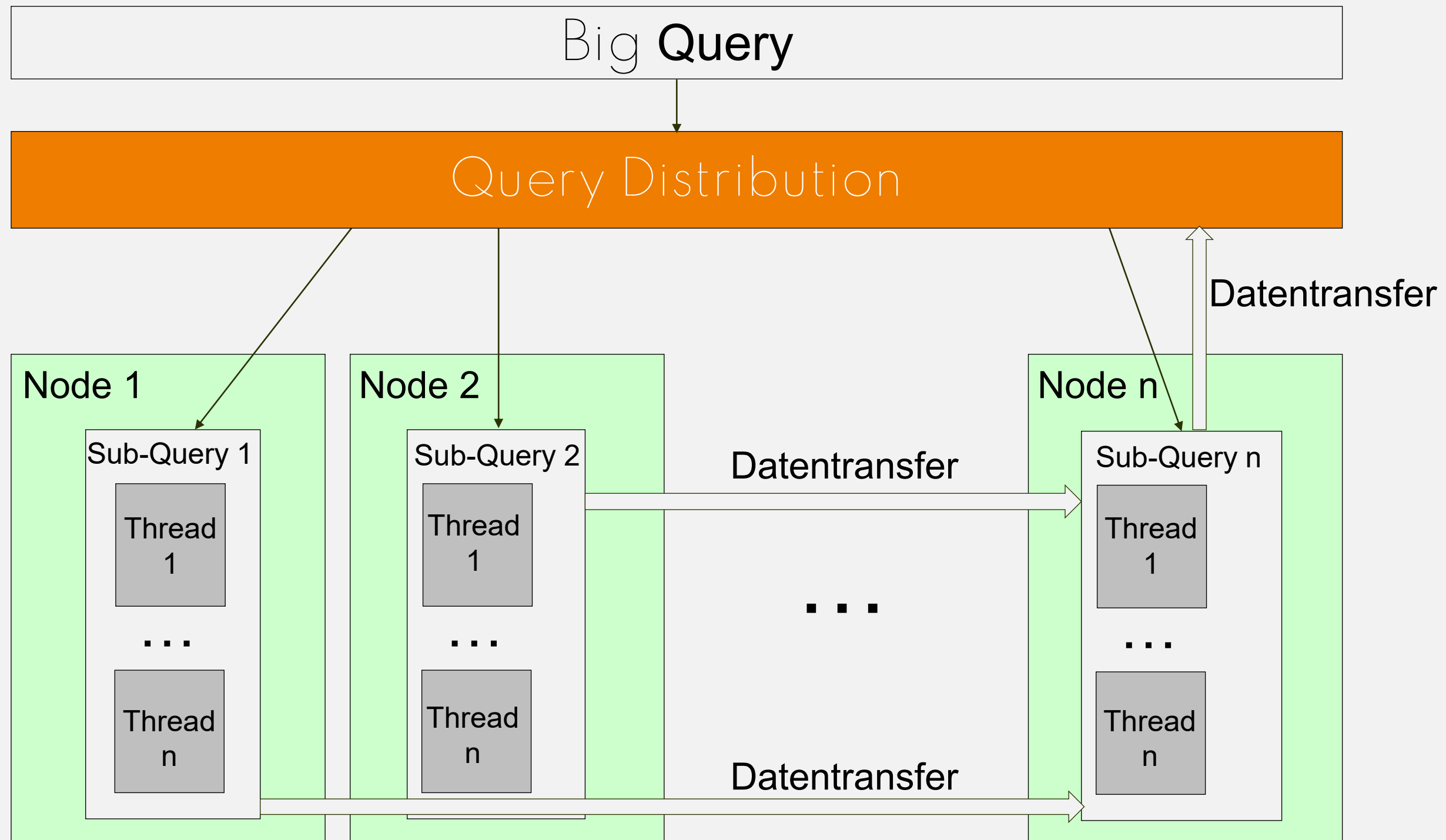


The fastest I/O is the one that never takes place: Es werden nur diejenigen Spalten gelesen, die benötigt werden (gerade bei breiten Tabellen wichtig)

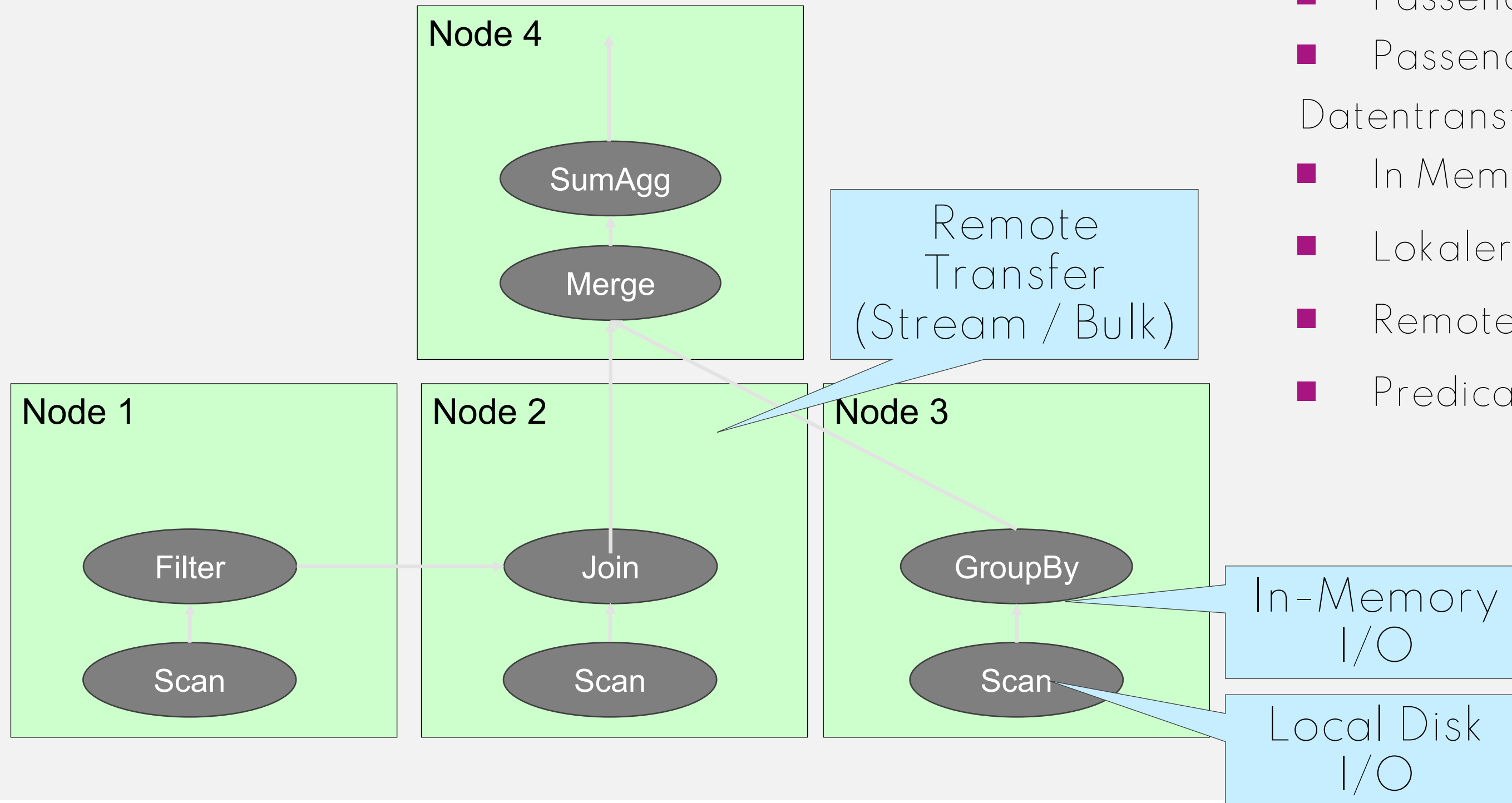
Kompression
(funktioniert bei Spalten besser als bei Zeilen):

- 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>