

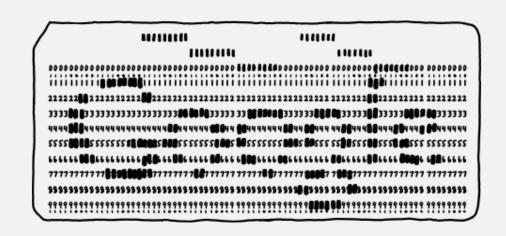
## Big Data

Franz Wimmer franz.wimmer@qaware.de

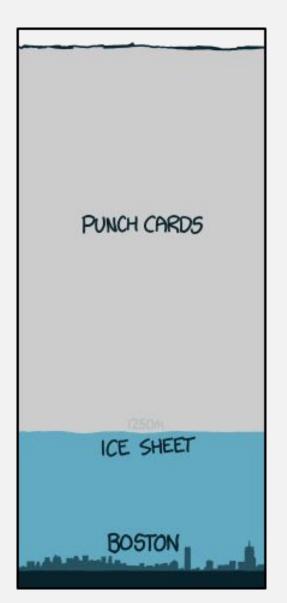
#### **Big Data**



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







Quelle: <a href="https://what-if.xkcd.com/63/">https://what-if.xkcd.com/63/</a>

https://www.youtube.com/watch?v=I64CQp6zOPk&t=275s (Randall Munroe @ TED)

## Big Data – was ist das überhaupt?



Ich muss mit Daten umgehen, die folgende Eigenschaften haben:

- Volume: Eine Datenmenge, meist > 1 TB
- Variety: Verschiedene Typen von Daten: Text, Binaries, Audio, Bilder...
- Velocity: Die Daten müssen schnell verarbeitet werden.

Quelle: . https://blogs.cfainstitute.org/investor/2021/01/25/when-to-use-big-data-and-when-not-to/

## Big Data – was ist das überhaupt?



#### Auch ein Faktor:

■ Die Technologien, die Verwendet werden, um den Datensatz zu verarbeiten

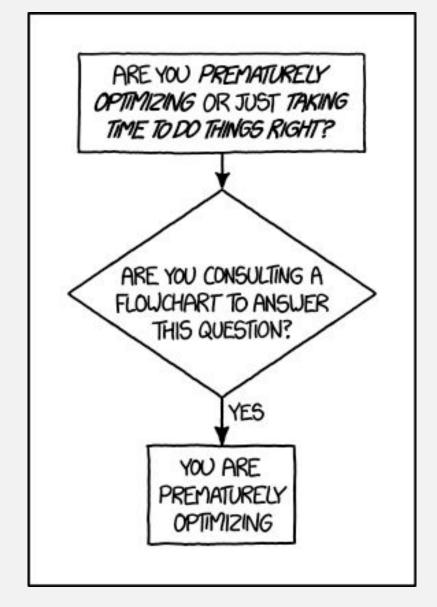
"Big data is a term describing the storage and analysis of large and or complex data sets using a series of techniques including, but not limited to: NoSQL, MapReduce and machine learning"

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

## Habe ich wirklich ein BigData-Problem?



- Wenn die Daten in den Arbeitsspeicher passen, brauche ich kein BigData.
- Wenn sich das Problem durch Optimierung meiner
   Datenbankschemas auflöst, brauche ich kein BigData.
- Wenn meine Queries ineffizient sind und ich das durch Refactoring lösen kann, brauche ich kein BigData.



Premature optimization is the root of all evil, so to start this project I'd better come up with a system that can determine whether a possible optimization is premature or not.

## **Big Data**

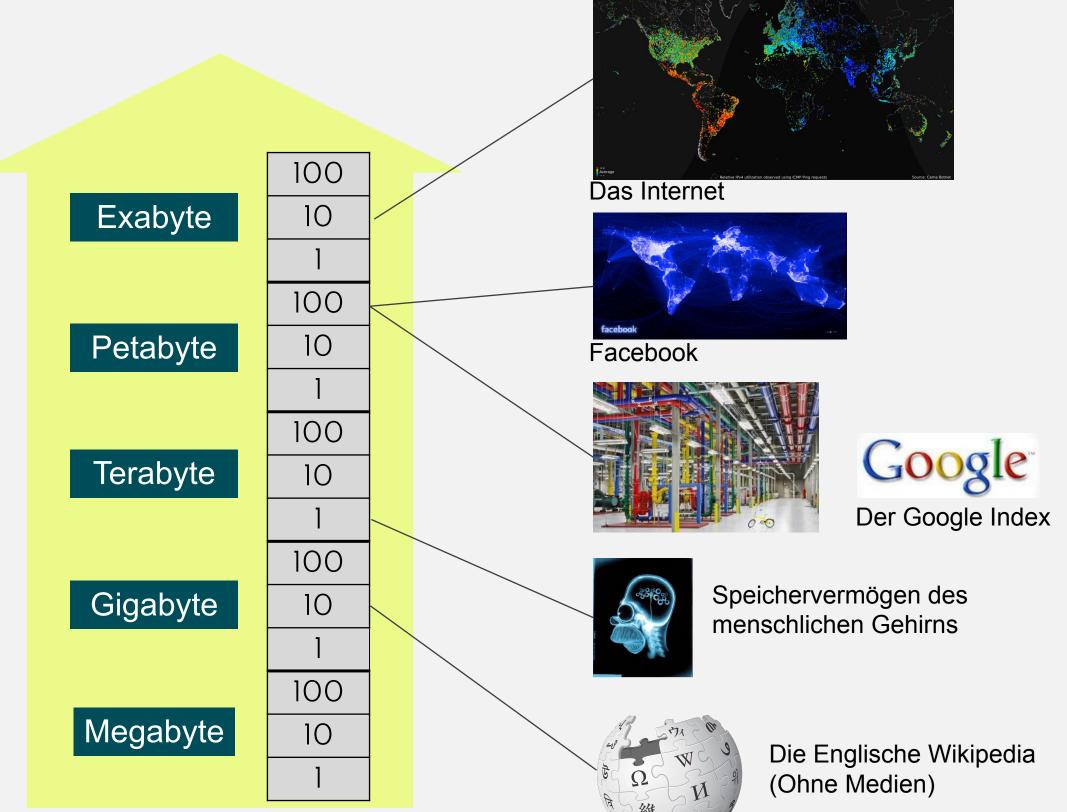
#### **Big Data**

Verarbeitung großer Datenmengen durch:

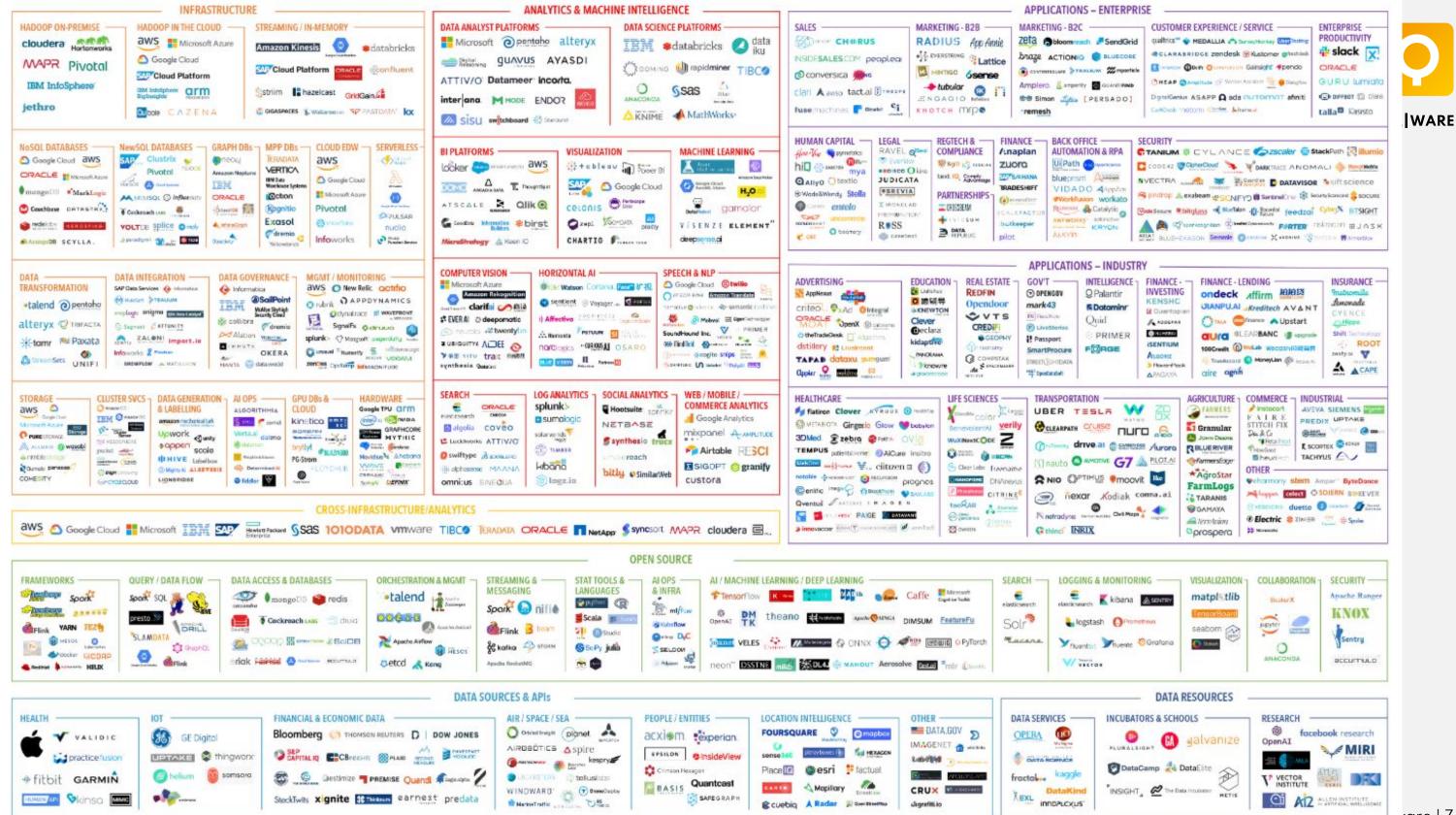
Data

Big

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



#### DATA & AI LANDSCAPE 2019



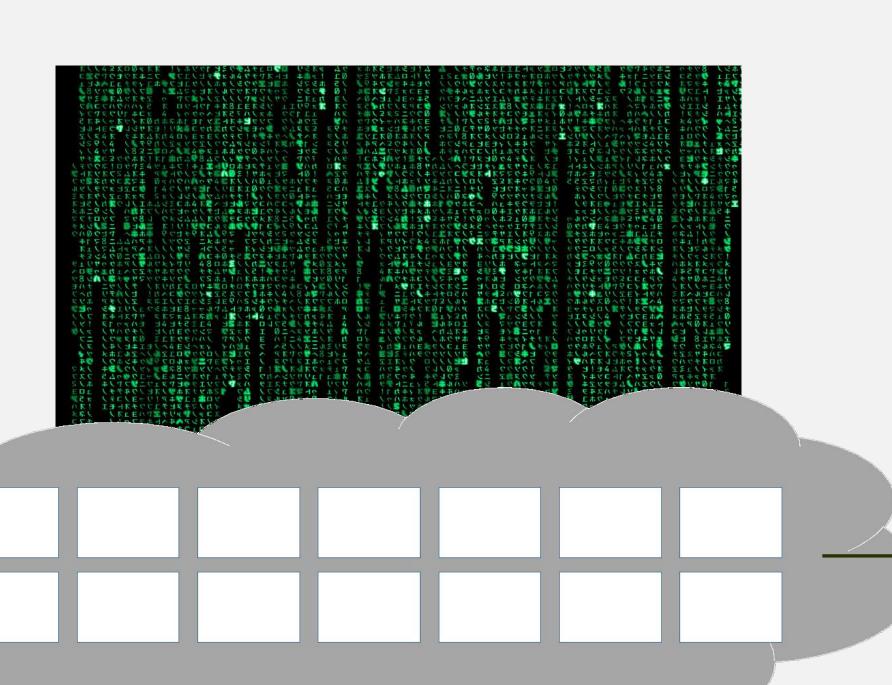
FIRSTMARK EARLY STAGE VENTURE CAPITAL

## 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:  $O(n \log n) \square n \times O(\log n)$ •

```
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));
```

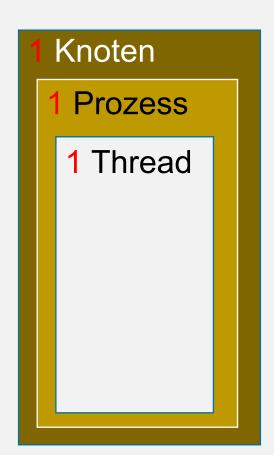
• Funktioniert nicht: Berechnung der Fibonacci-Folge ( $F_{k+2} = F_k + F_{k+1}$ ). Berechnung ist nicht parallelisierbar.

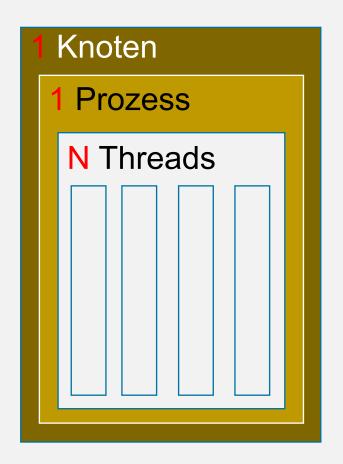
Ein paralleler Algorithmus (*Job*) ist aufgeteilt in sequenzielle Berechnungsschritte (*Tasks*), die parallel zueinander abgearbeitet werden können. Der Entwurf von parallelen Algorithmen folgt oft dem Teile-und-Herrsche Prinzip.

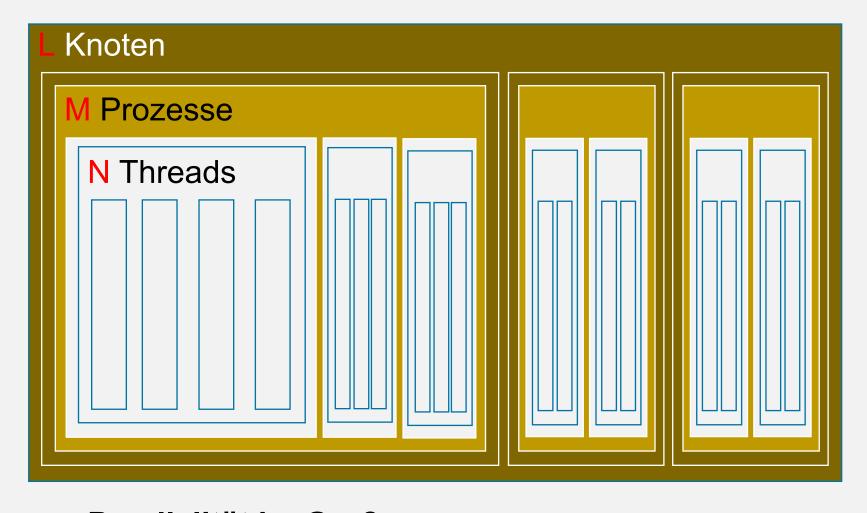
# Parallele Programmierung basiert oft auf funktionaler Programmierung.

- 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 potenziell großer Datenmenge und Verarbeitungs-geschwind igkeit



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.
- Falls Datentransfer notwendig, dann so schnell wie möglich:
   In-Memory vor lokaler Festplatte vor
  - In-Memory vor lokaler Festplatte vor Remote-Transfer.
- 3. Parallelisierung über *Tasks* (seiteneffektfreie Funktionen) und *Jobs* (Ausführungsvorschrift für Tasks) sowie entsprechend partitionierter Daten (*Shards*).
- Design for Failure: Ausführungsfehler als
   Standardfall ansehen und verzeihend und kompensierend sein.

## Notwendige Architekturkonzepte

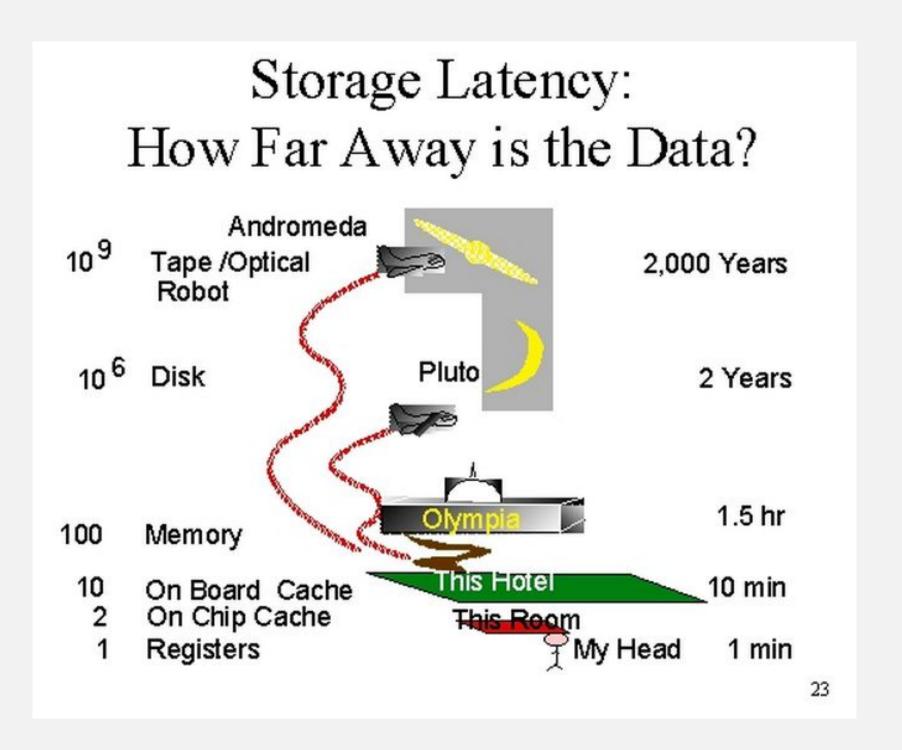
- 1. Verteilung der Daten
- 2. Verteilung und Überwachung von Tasks
- 3. Aufteilung der Ressourcen
- 4. Entwurfsmuster zur Implementierung von Jobs

#### Vorsicht: BigData hat seinen Preis.

```
0.5 ns - CPU L1 dCACHE reference
             ns - speed-of-light (a photon) travel a 1 ft (30.5cm) distance
           ns - CPU L1 iCACHE Branch mispredict
            ns - CPU L2 CACHE reference
            ns - CPU cross-QPI/NUMA best case on XEON E5-46*
             ns - MUTEX lock/unlock
        100
             ns - CPU own DDR MEMORY reference
        100
        135
             ns - CPU cross-QPI/NUMA best case on XEON E7-*
             ns - CPU cross-QPI/NUMA worst case on XEON E7-*
        202
             ns - CPU cross-QPI/NUMA worst case on XEON E5-46*
        325
             ns - Compress 1 KB with Zippy PROCESS (+GHz,+SIMD,+multicore tricks)
     10,000
    20,000
             ns - Send 2 KB over 1 Gbps NETWORK
   250,000
             ns - Read 1 MB sequentially from MEMORY
    500,000
             ns - Round trip within a same DataCenter
10,000,000
            ns - DISK seek
10,000,000
            ns - Read 1 MB sequentially from NETWORK
             ns - Read 1 MB sequentially from DISK
30,000,000
             ns - Send a NETWORK packet CA -> Netherlands
150,000,000
       ns
   us
ms
```

Quelle: <a href="https://stackoverflow.com/a/59750841/5764665">https://stackoverflow.com/a/59750841/5764665</a>

#### Latenzen im Vergleich



#### Eine Standardarchitektur für Parallelität im Großen

Eine **Job-Steuerung**, die einzelne Jobs zur Ausführung bringt.

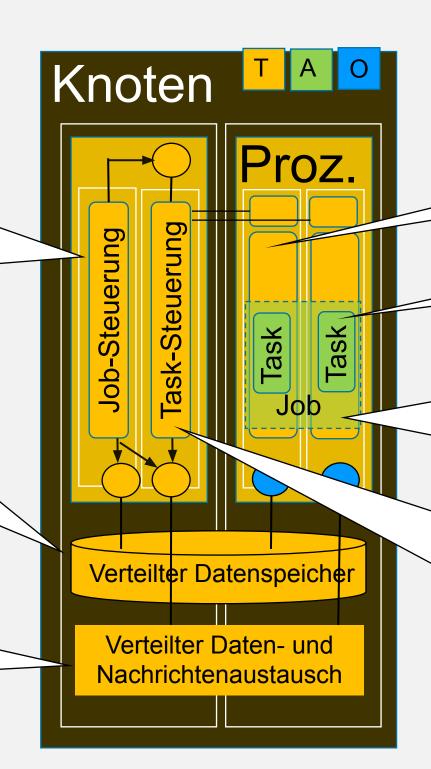
Sie übergibt die Tasks eines Jobs entsprechend der Ausführungsvorschrift der Task-Steuerung und verhandelt dabei die notwendigen Ressourcen, überwacht deren Ausführung und kompensiert Fehlersituationen z.B. durch Wiederaufsetzen einzelner Tasks. Es existiert i.d.R. eine Job-Steuerung pro Entwurfsmuster.

#### Ein Verteilter Datenspeicher

(Dateisystem, Datenbank, Hauptspeicher) mit Datenredundanz u.A. für Ausfallsicherheit, einem Sicherheitskonzept (Rechte&Rollen, Verschlüsselung), integrierter Kompression, einem Metadatenkatalog und hoher Scan-Geschwindigkeit.

## Ein Verteilter Daten- und Nachrichtenaustausch.

Grundlage: Zuverlässige und effizientes Kommunikationsprotokoll (i.d.R. binär und komprimiert).



**Task-Container** (i.d.R. Prozesse) mit exklusiver, temporärer Ressourcen-Zuordnung (*Slot*) zur isolierten Ausführung von Tasks auf einem Knoten.

**Task** als nicht weiter parallelisierbarer Ausführungsschritt.

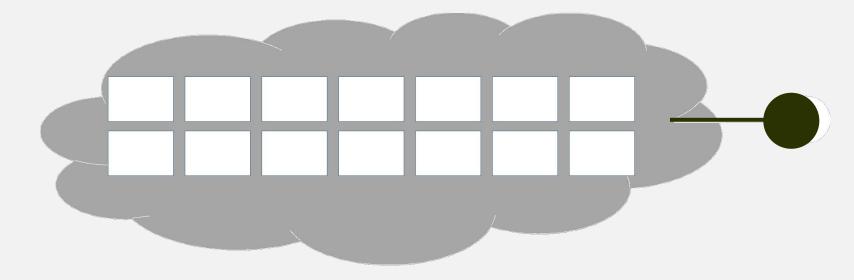
**Job** als logische Klammer um Tasks inkl. deren Ausführungsvorschrift.

Diese leitet sich aus dem verwendeten Entwurfsmuster ab, wie z.B. MapReduce, DAG, MPI, Pipes & Filters.

Eine **Task-Steuerung**, die einzelne Tasks zur Ausführung bringt.

Sie nimmt Anfragen zur Task-Ausführung entgegen, plant sie gemäß einer festgelegten Strategie (z.B. Fairness, Kosteneffizienz, gleichmäßige Auslastung, SLAs, ...) zur Ausführung ein und führt sie schließlich aus und überwacht den Ressourcenverbrauch.

## Welche Lösungen gibt es dafür im Cloud Computing?

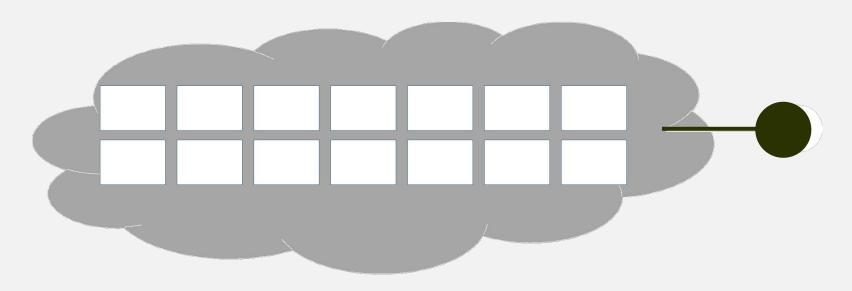


- Big Data Engines (low level)
  - MapReduce
  - RDD (Resilient Distributed Dataset)
- Big Data Datenbanken (high level)
  - NoSQL Datenbanken
  - NewSQL Datenbanken (NoSQL + SQL)
- Verteilte Dateisysteme
- In-Memory Data Grids / Elastic Memory



## 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.

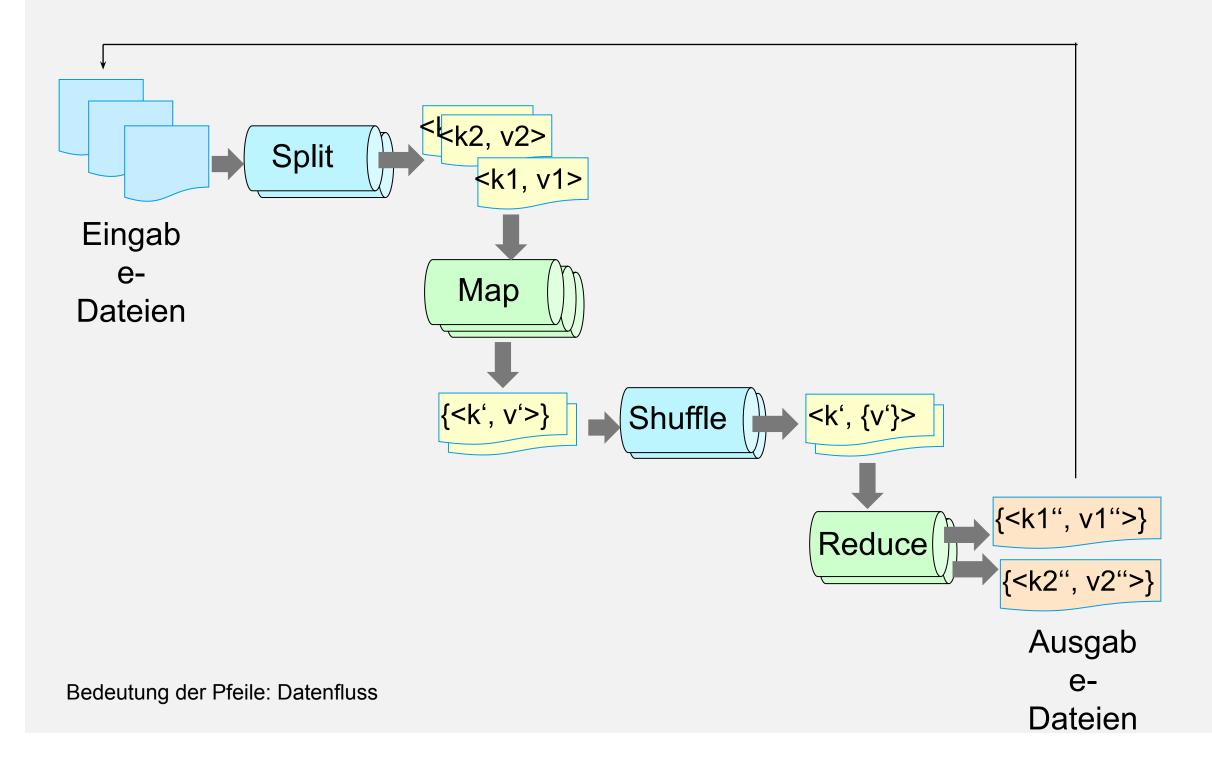
```
Signatur: map(k, v) \square list(\langle k', v' \rangle)
```

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')) □ list(<k'', v''>)
```

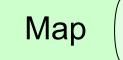
• Dabeisoll gelten: |list(<k'', v''>)| << |list(<k', v'>)|

# 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

Ein- und Ausgabe der Man-Phase:

```
We saw lions and tigers. The lions were

we 1 lions 1 tigers 1 lions 1

saw 1 and 1 the 1 were 1

fantastic 1 the 1 were 1 too 1

and 1 tigers 1 fantastic 1

Ausgabe

and 1 tigers 1 fantastic 1
```

```
Key Value

Key Value
```

```
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

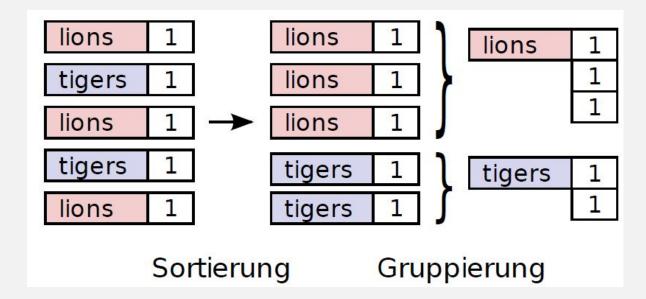


Мар

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

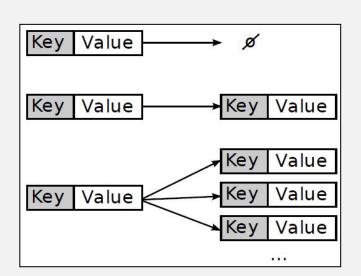


Мар

Shuffle

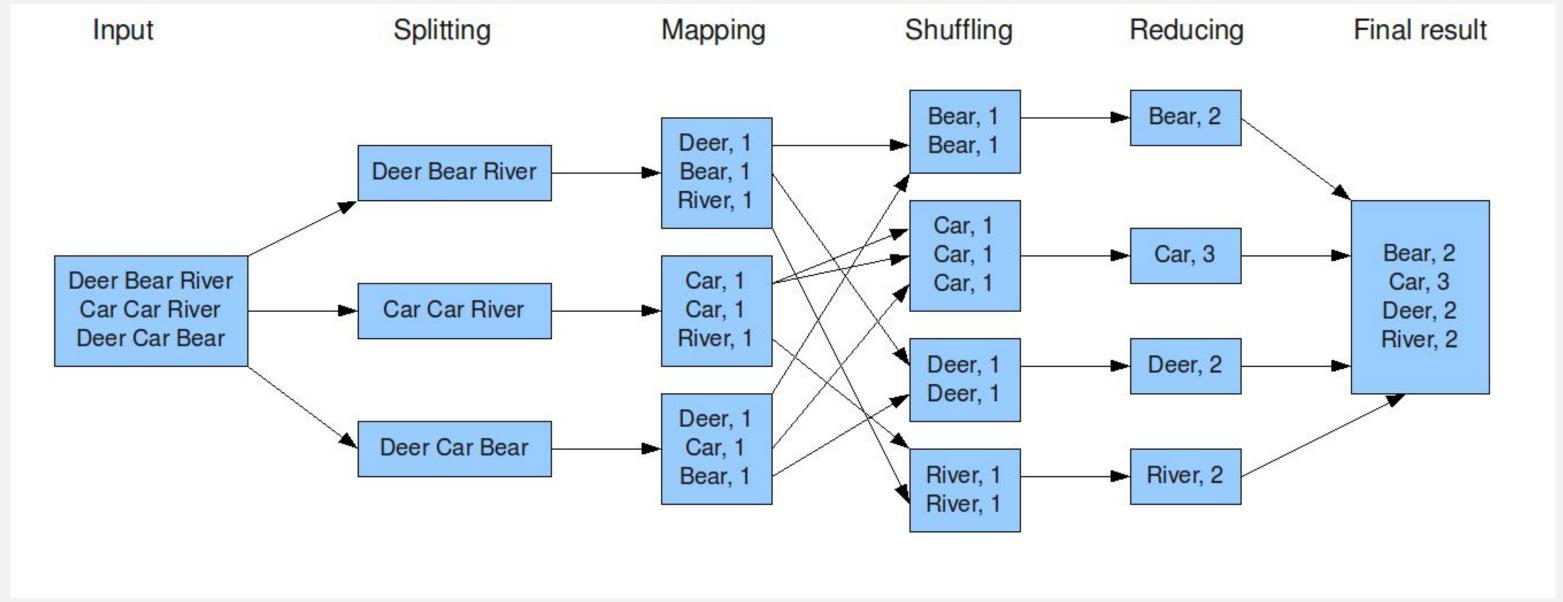
Reduce

- 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:



```
Pseudocode
reduce(String Reduce Phase return values):
   //key: a word
   //values: a list of counts
   for each value in values:
     result += ParseInt(value);
     Emit(AsString(Key +", "+result));
24
```

#### 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.

## Anwendungsbeispiele für MapReduce (2/2)

#### Weitere Beispiele:

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







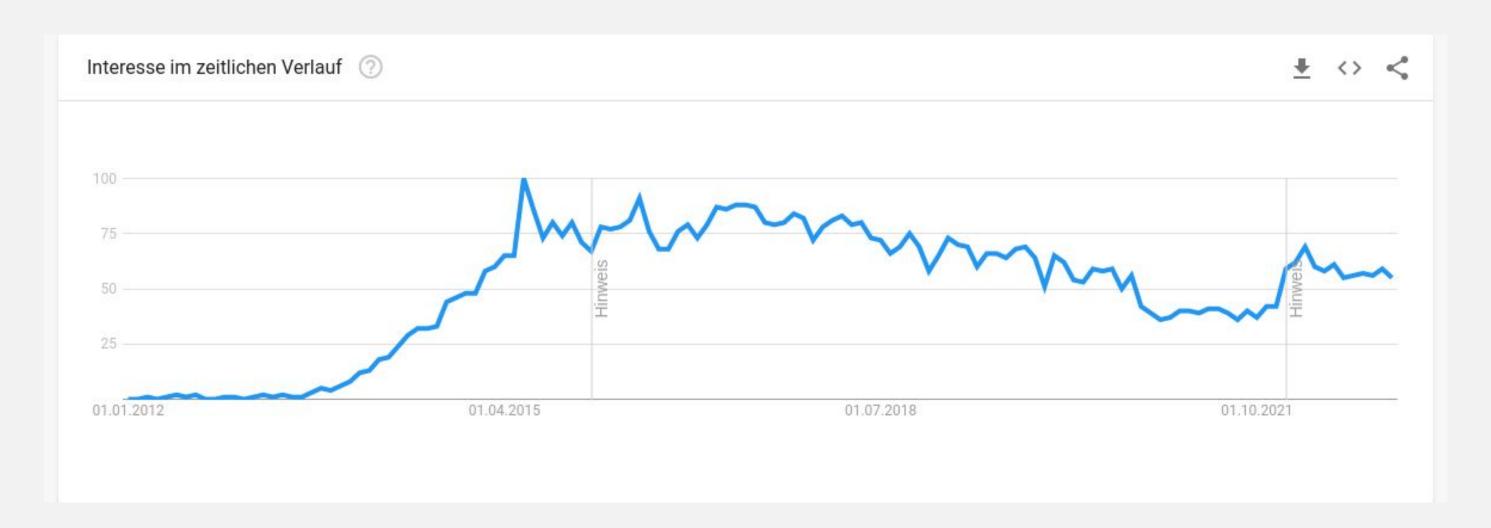


## Apache Spark

## Apache Spark



BigData-Framework auf Basis von Scala



# 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.



#### Die Anatomie eines RDDs.

Data Lineage – keine vollständige Neuberechnung bei Verlust eines RDDs

**RDD** 

Lazy Evaluation

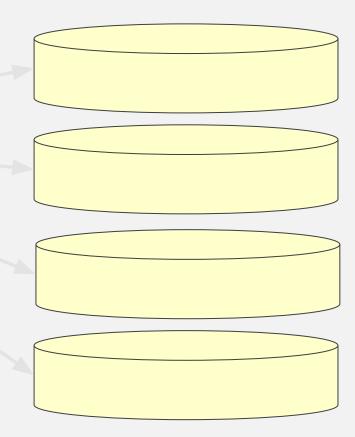
– erst wenn
Action
aufgerufen wird.

RDD

Referenzen auf Daten

Referenz auf Vorgänger RDD

Auszuführende Transformation



- Verteiltes Dateisystem
- Hauptspeicher
- Beliebiger anderer Datenspeicher

#### Daten verarbeiten: Mehr als Map und Reduce.

#### Filter

#### Map

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

#### Reduce

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

#### Sort

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

```
Transformations

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

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

#### Wie funktioniert das?

```
/* SimpleApp.scala */
                                                  Driver Program
import org.apache.spark.SparkContext
                                                  SparkContext
                                                                  Cluster Manager
import org.apache.spark.SparkConf
object SimpleApp {
 def main(args: Array tring]) {
   val logFile = "YOU SPARK HOME/README.md"
   val conf = new Spanconf().setAppName("Simple Application")
   val sc = new SparkContext(conf)
   val logData = sc.textFile(logFile, 2).cache()
   val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
   println("Lines with a: %s, Lines h b: %s".format(numAs, numPs)
```

Worker Node

Executor

Task

Worker Node

Executor

Task

Cache

Task

Cache

Task



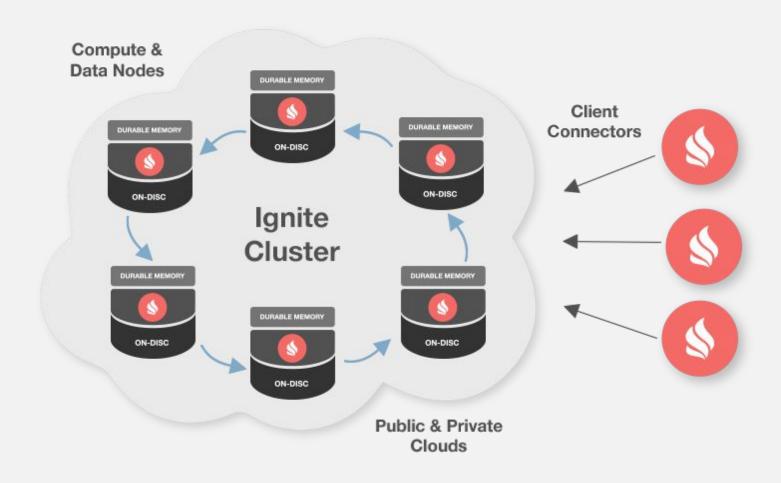
## Apache Ignite



"Distributed Database For High-Performance Applications With In-Memory Speed"

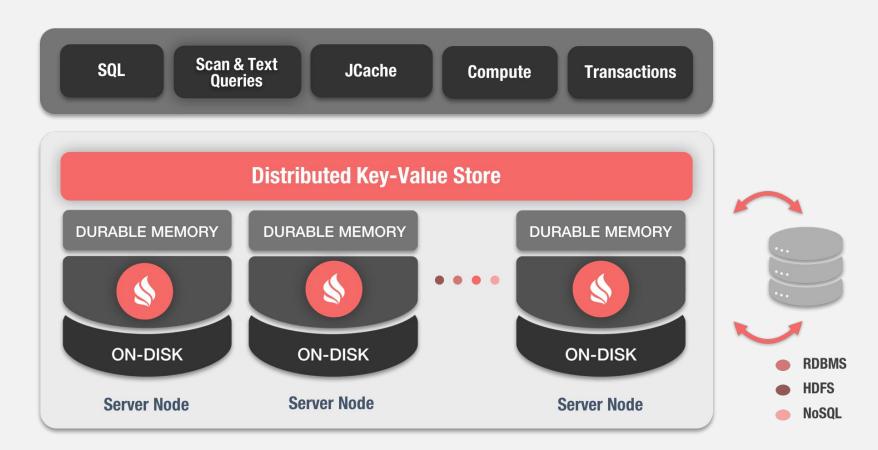
## Apache Ignite

- Open-Source-Framework f
  ür 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, ...)



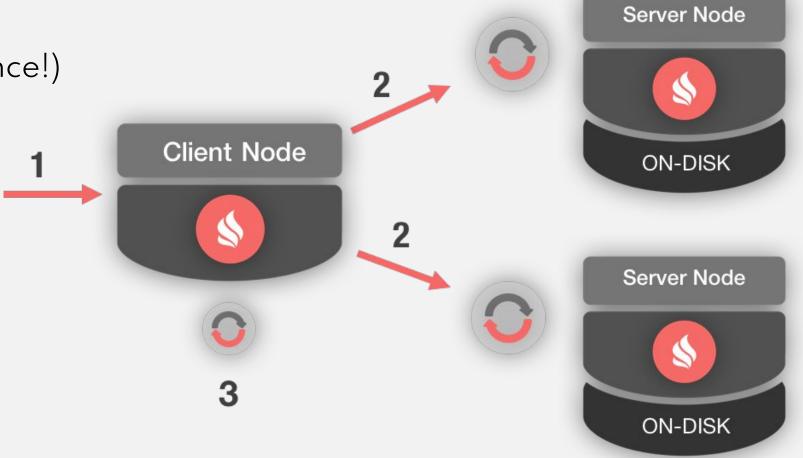
### Ignite Data Grid Beispiel

```
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

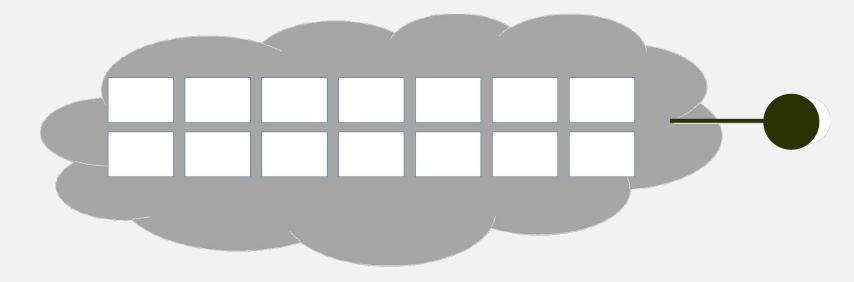
### Apache Ignite Streaming - Beispiel

```
CacheConfiguration<String, String> configuration = new CacheConfiguration<>(CACHENAME);
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);
```



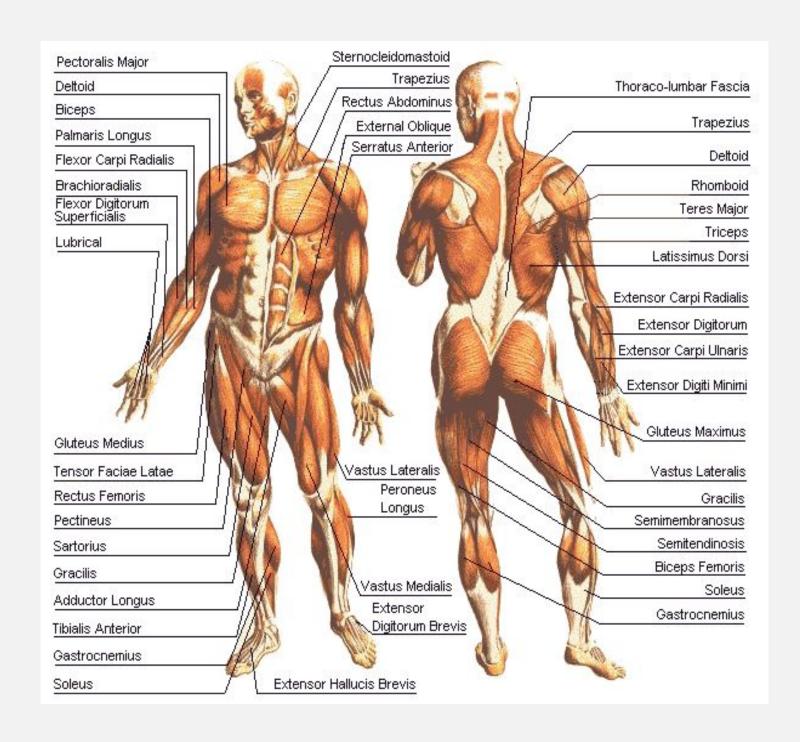
## Big Data Datenbanken

#### 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

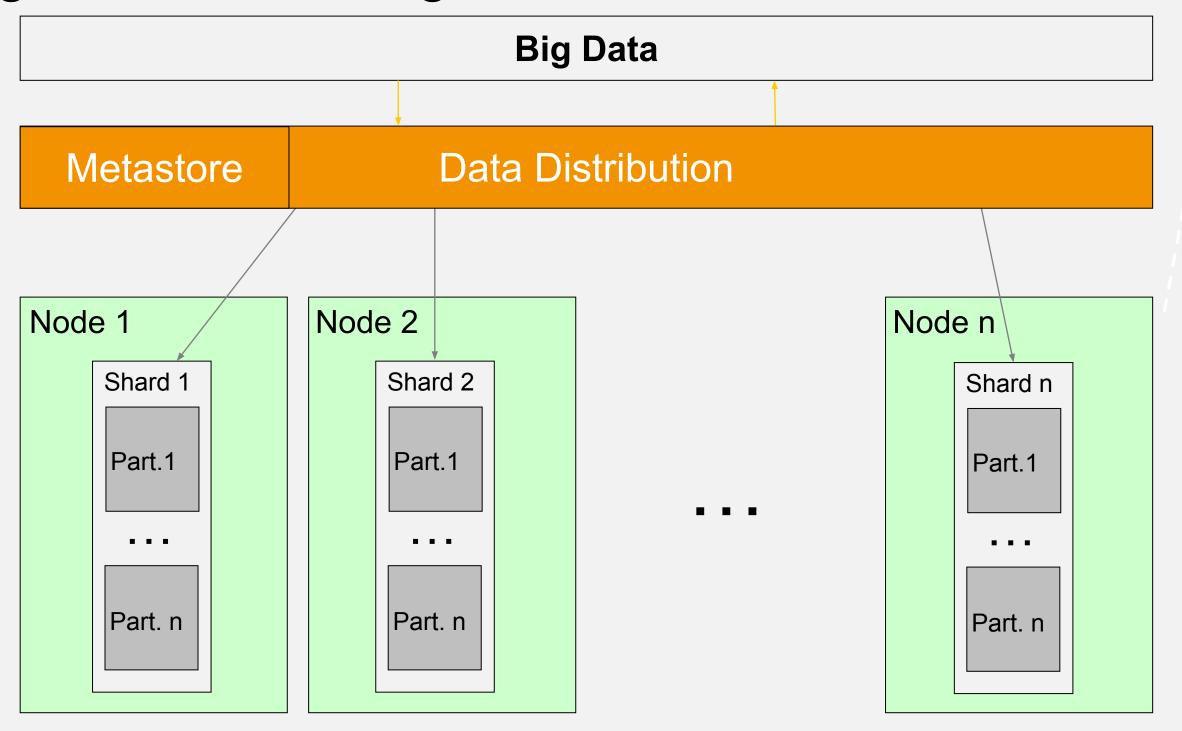


**Query Distribution** 

**Data Distribution** 

Data Persistence

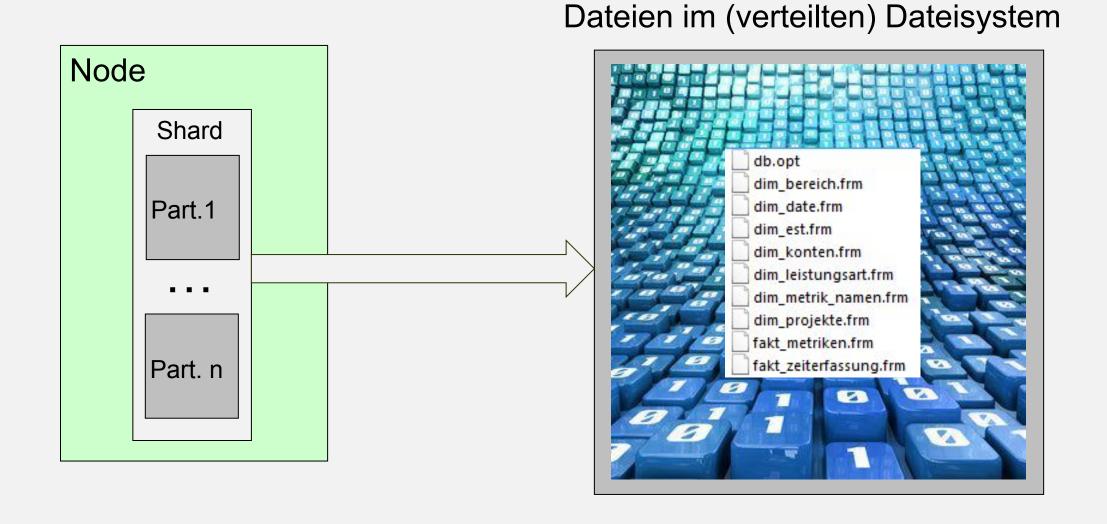
Sharding and Partitioning: Verteilung und Stückelung von großen Datenmengen.



(Re-) Sharding- undPartitioning-Funktion:f(Daten) □ Shardf(Daten) □ Partition.+ Replikationsstrategie.

+ Konsistenzstrategie.

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

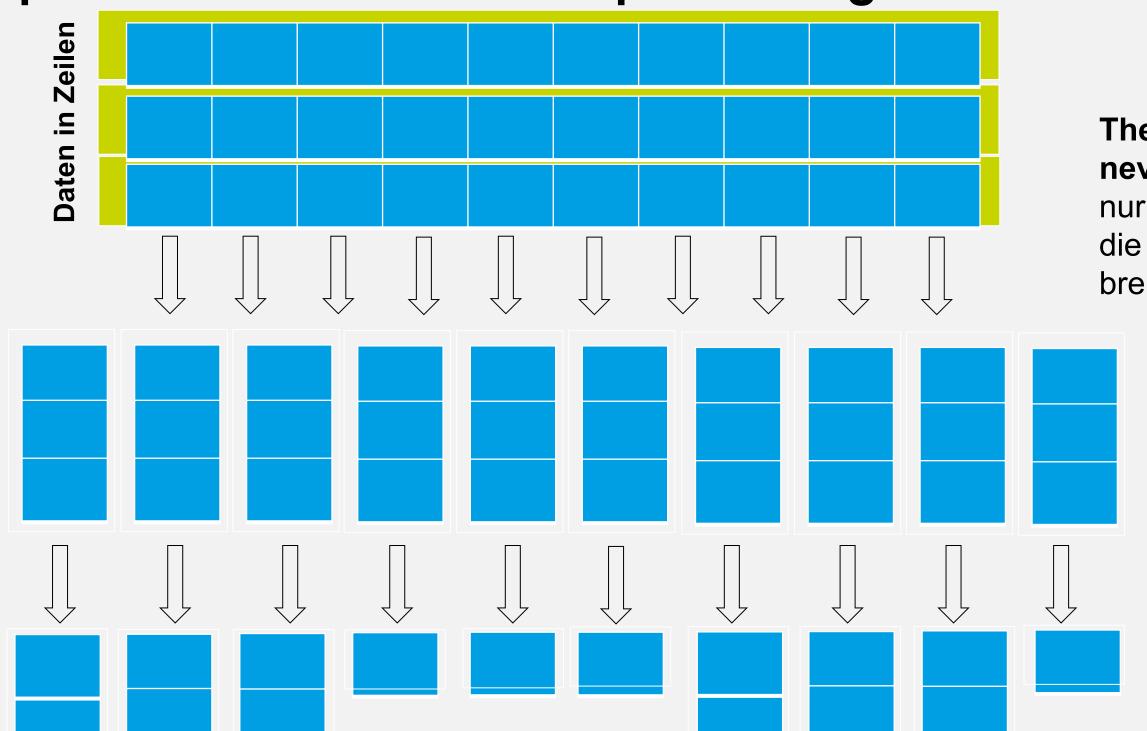


#### Spalten-orientierte Datenspeicherung.

Daten in Spalten

Daten in Spalten

Komprimierte

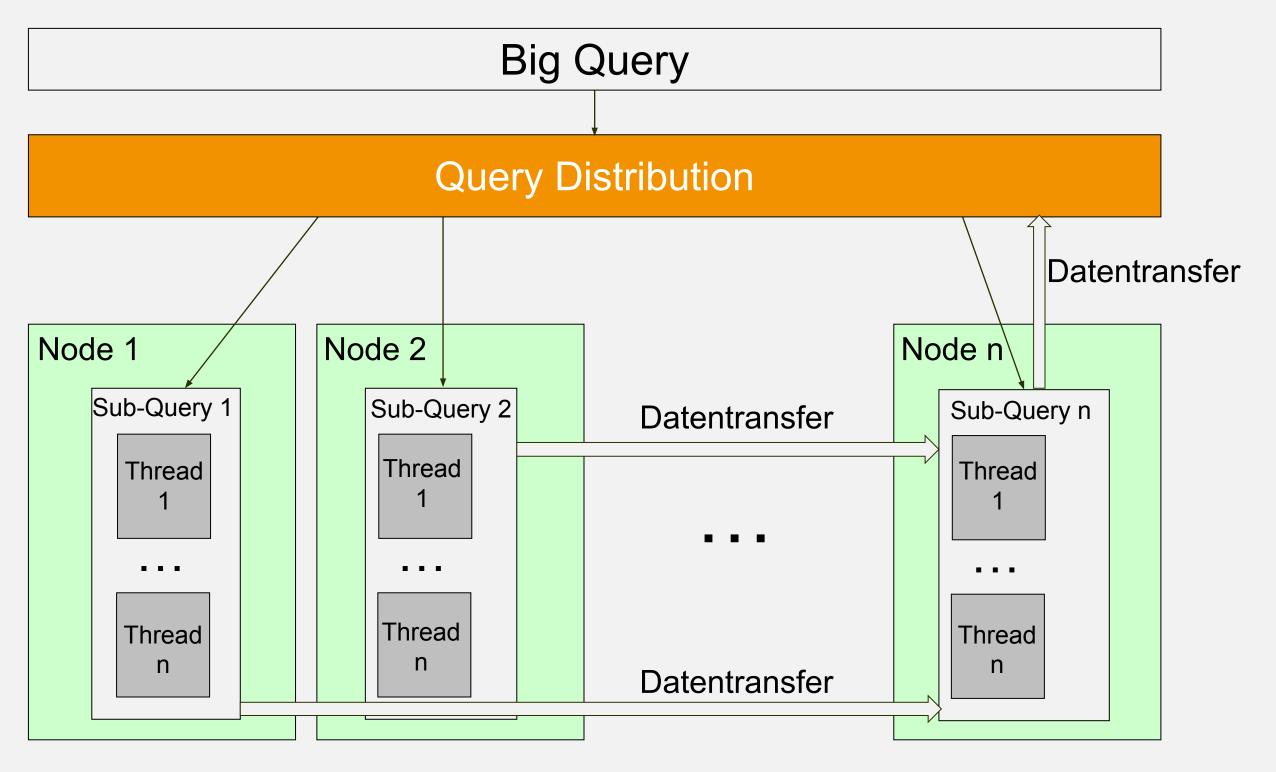


The fastest I/O is the one that never takes place: Es werden nur diejenigen Spalten gelesen, die benötigt werden (gerade bei breiten Tabellen wichtig)

**Kompression** (funktioniert bei Spalten besser als bei Zeilen):

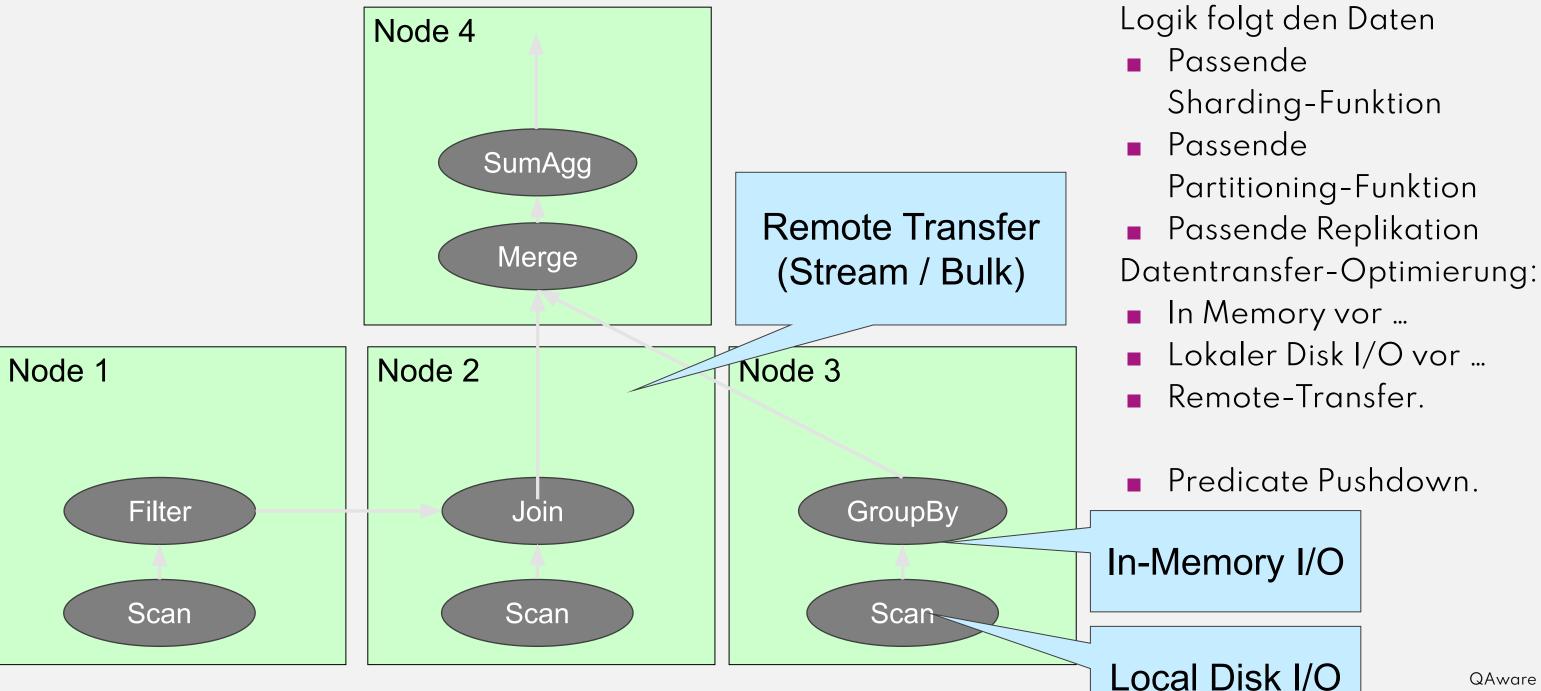
- Datentyp-spezifisch (z.B. Dictionaries)
- + ggF. Spalten-Index

#### Verteilte und parallelisierte Ausführung von Abfragen.



#### Ein verteilter Ausführungsplan: Ein azyklischer Funktionsgraph.





#### Verteilte Datenbanken

- Apache Cassandra (Wide column store, Tables & Rows)
- Google Bigtable (Wide column store, no relational model)
- Couchbase (document oriented)
- CockroachDB (OSS-Implementierung von Spanner)
- 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)

#### Further reading / viewing

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

• <a href="https://www.slideshare.net/QAware/consistency-availability-and-partition-tolerance-in-practice">https://www.slideshare.net/QAware/consistency-availability-and-partition-tolerance-in-practice</a>

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-apa
   che-ignite
- Folien:
   <u>https://de.slideshare.net/QAware/neues-aus-dem-tindergarten-auswertung-privater-apis-mit-apache-ignite</u>