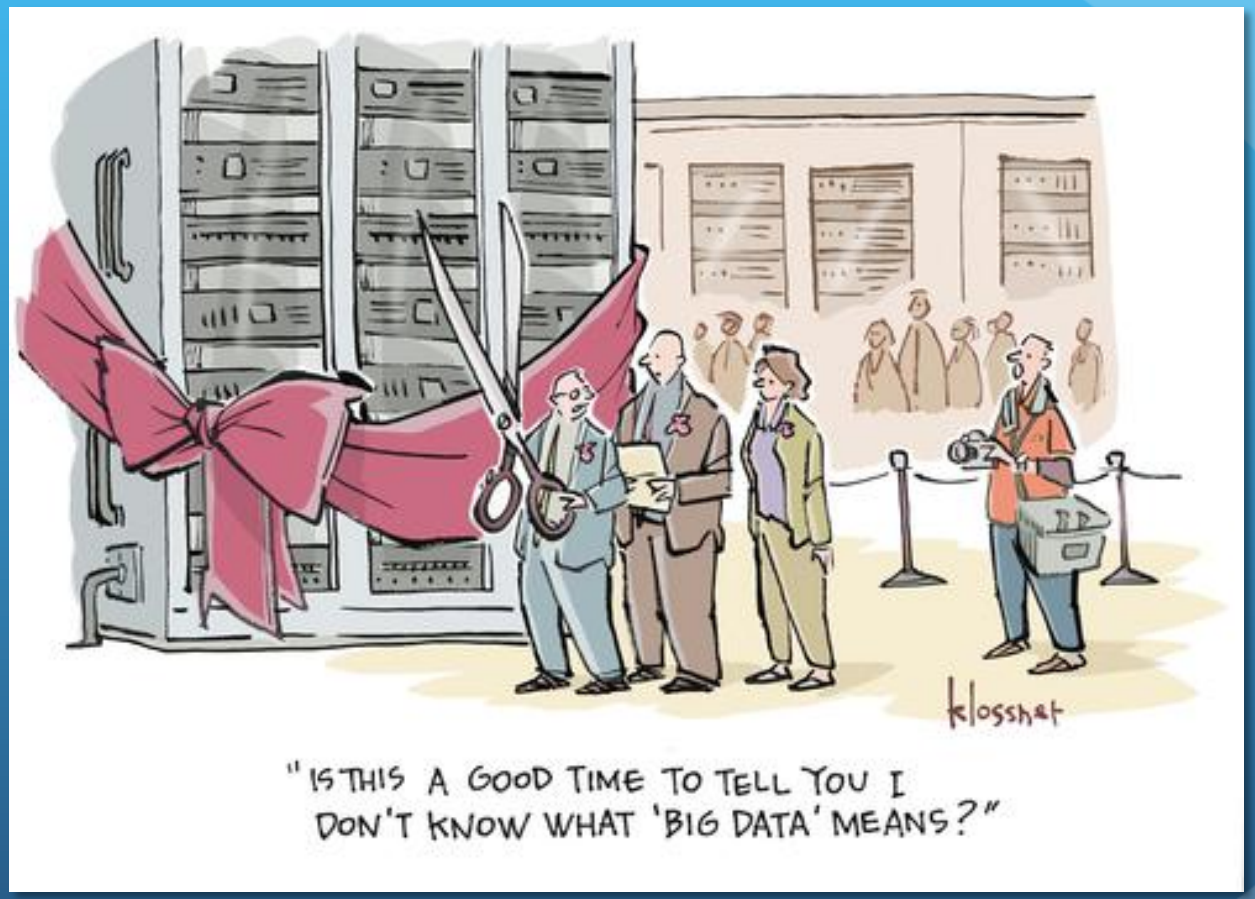
Präsentation der Masterarbeit

Big Data Processing mit Apache Spark

Sascha P. Lorenz
Hochschule Emden-Leer
Medieninformatik

Was ist „Big Data“?

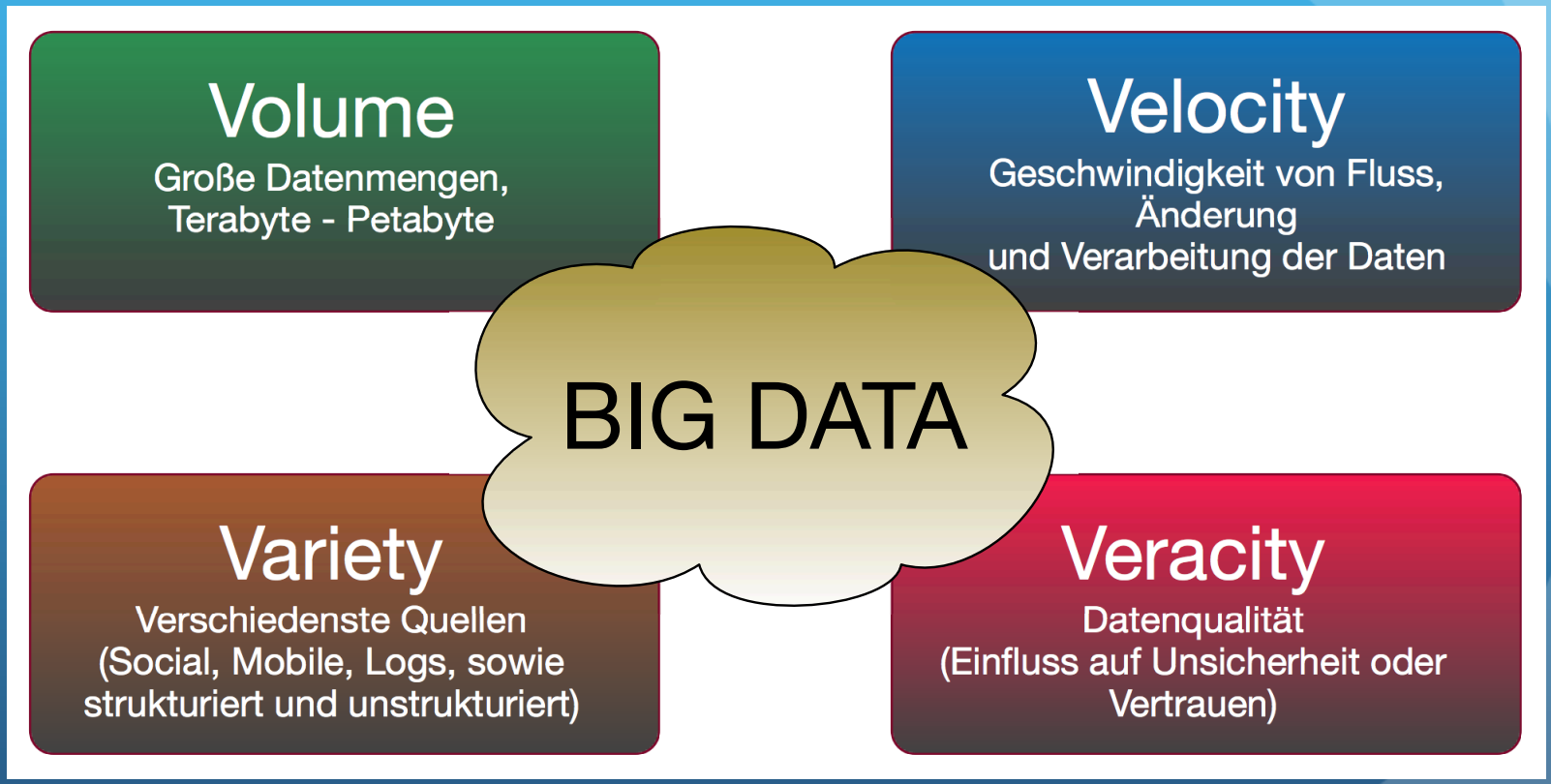


Quelle : http://www.networkcomputing.com/big-data-defined/d/d-id/1204588

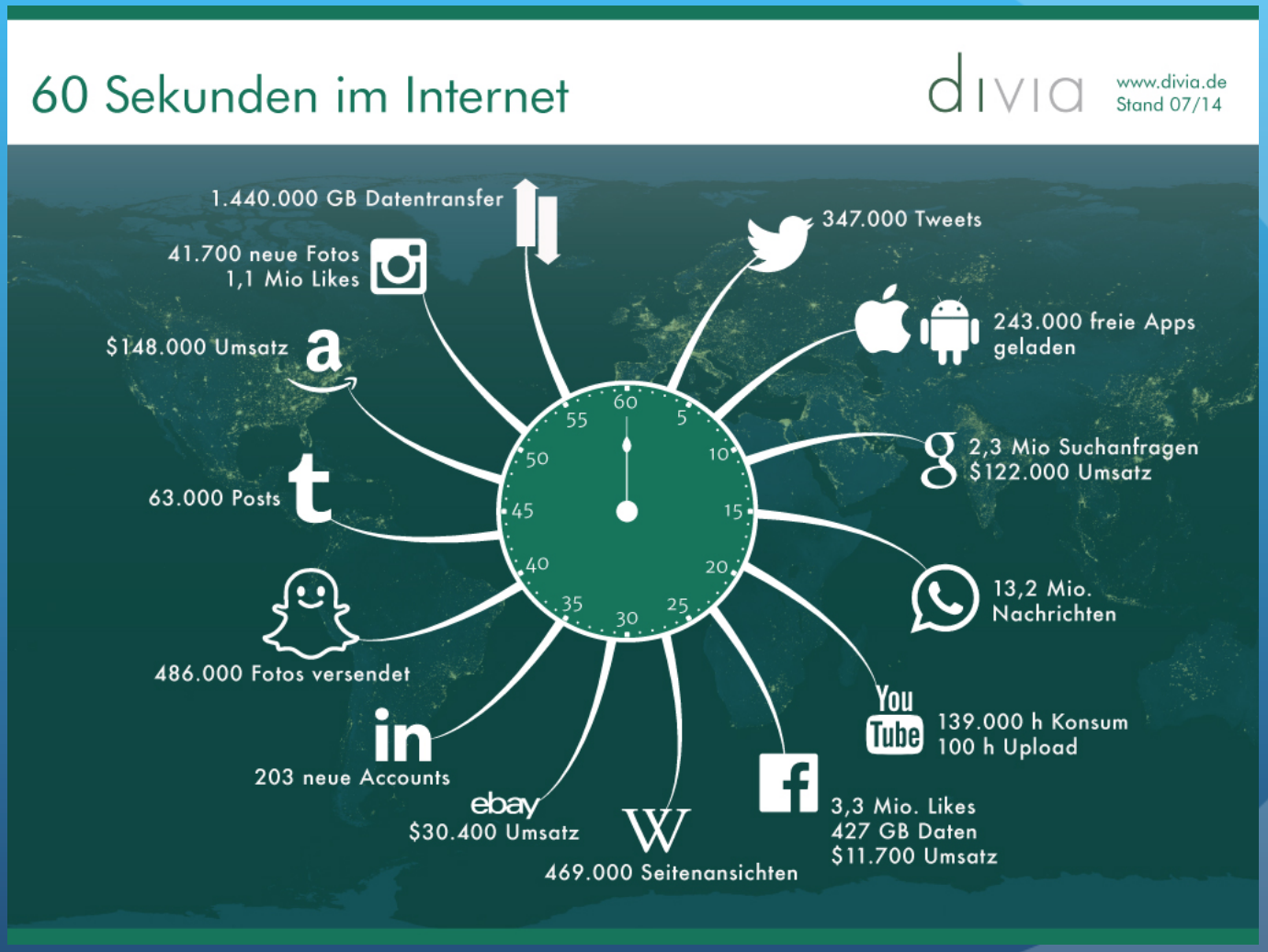
Definition von „Big Data“

- Wikipedia sagt dazu: „... bezeichnet Daten-Mengen, die zu groß, oder zu komplex sind, oder sich zu schnell ändern, um sie mit händischen und klassischen Methoden der Datenverarbeitung auszuwerten.“
- Andere Definition: „Big Data“ bezeichnet nicht nur Datenvolumen, sondern auch Werkzeuge und Prozesse für den Umgang mit großen Volumen
- Problematisch vor allem Erfassung, Speicherung, Suche, Verteilung, Analyse, Visualisierung von großen Datenmengen

Die vier „V" → „Big Data"

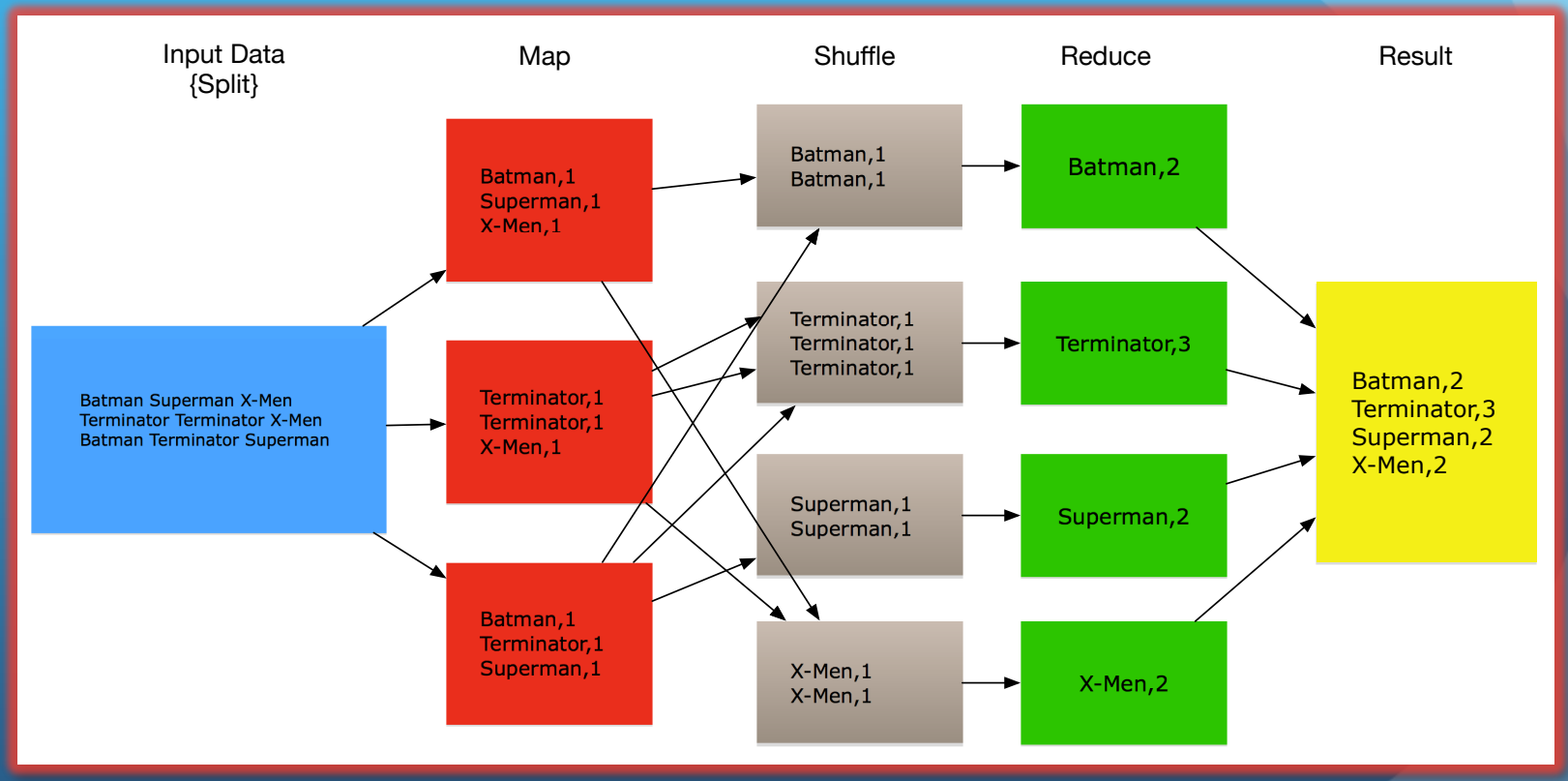


Nutzen von „Big Data“



Wie geht man damit um?

MapReduce


Input Data
{Split}
Map
Shuffle
Reduce
Result
Batman Superman X-Men
Terminator Terminator X-Men
Batman Terminator Superman
Batman,1
Superman,1
X-Men,1
Terminator,1
Terminator,1
X-Men,1
Batman,1
Terminator,1
Superman,1
Batman,1
Batman,1
Terminator,1
Terminator,1
Terminator,1
Superman,1
Superman,1
X-Men,1
X-Men,1
Batman,2
Terminator,3
Superman,2
X-Men,2
Batman,2
Terminator,3
Superman,2
X-Men,2

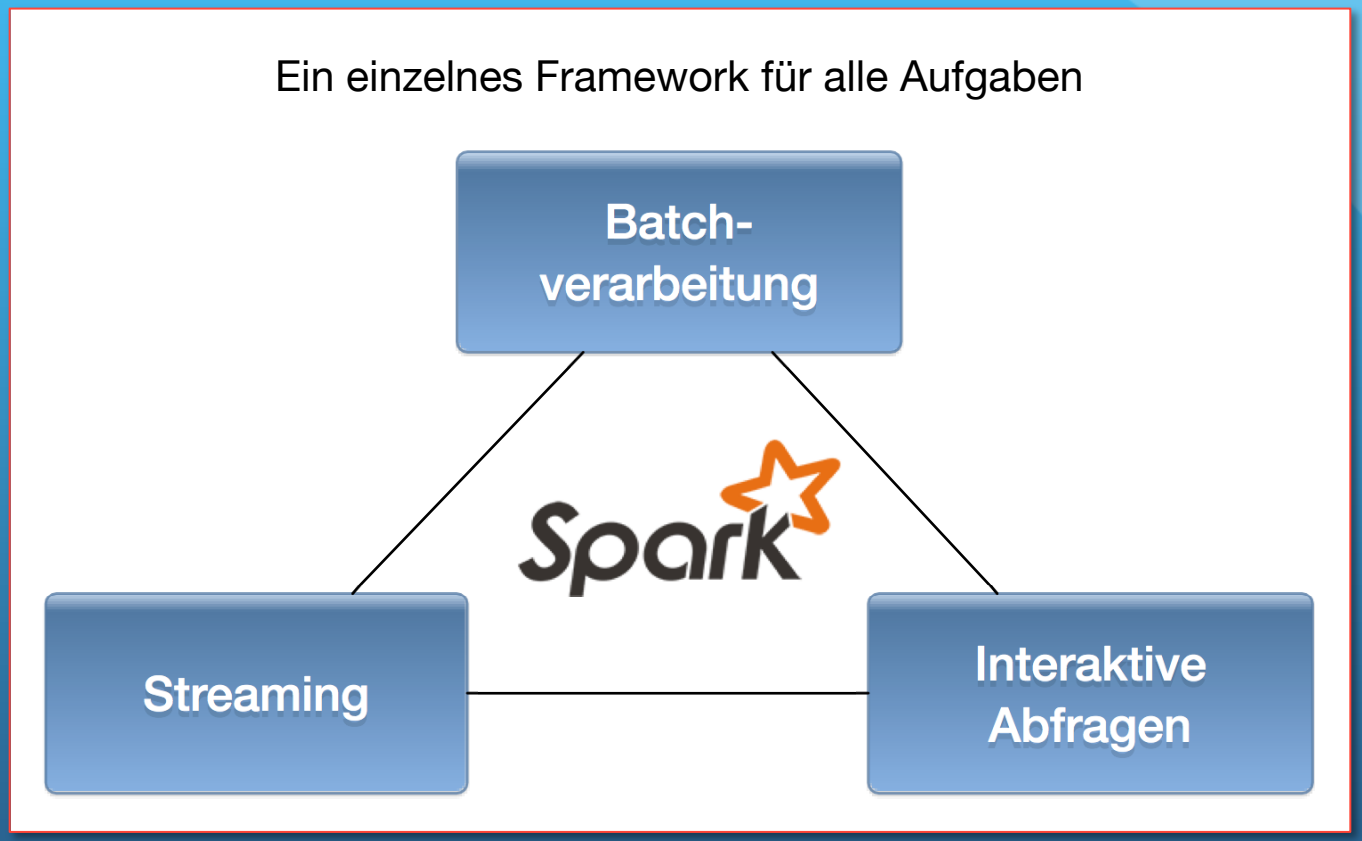
MapReduce

- *Map:* bildet Paar aus Key und Value auf Liste von Zwischenergebnissen ab. Für jedes Wertpaar aus Eingabeliste wird Map unabhängig aufgerufen, Ausführung nebenläufig und verteilt.
- *Shuffle:* Ergebnisse werden vor Reduce nach neuem Schlüssel in Liste gruppiert. Bei verteilten Systemen Datenaustausch über Netzwerk. (deshalb evtl. vorher Combine auf gleichem Knoten wie Map-Funktion)
- *Reduce:* Für jede Liste Aufruf der benutzerdefinierten Reduce-Funktion. Diese berechnet Ausgabeliste mit Ergebnissen. Ausführung nebenläufig und verteilt.

Schwächen von Hadoop

- Zwischenergebnisse werden immer persistiert, sind also I/O-Abhängig.
- Kein Problem für große Batch-Jobs wie ETL, Daten-konsolidierung, Datenbereinigung, hingegen problematisch für Streaming-Daten oder interaktive Abfragen
- Mittlerweile umfangreiches Ökosystem – dadurch verschiedene Technologie-Stacks für verschiedene Zwecke, häufig Versionskonflikte, nicht ideal für schnellen Datenaustausch zwischen Paralleljobs

Apache Spark



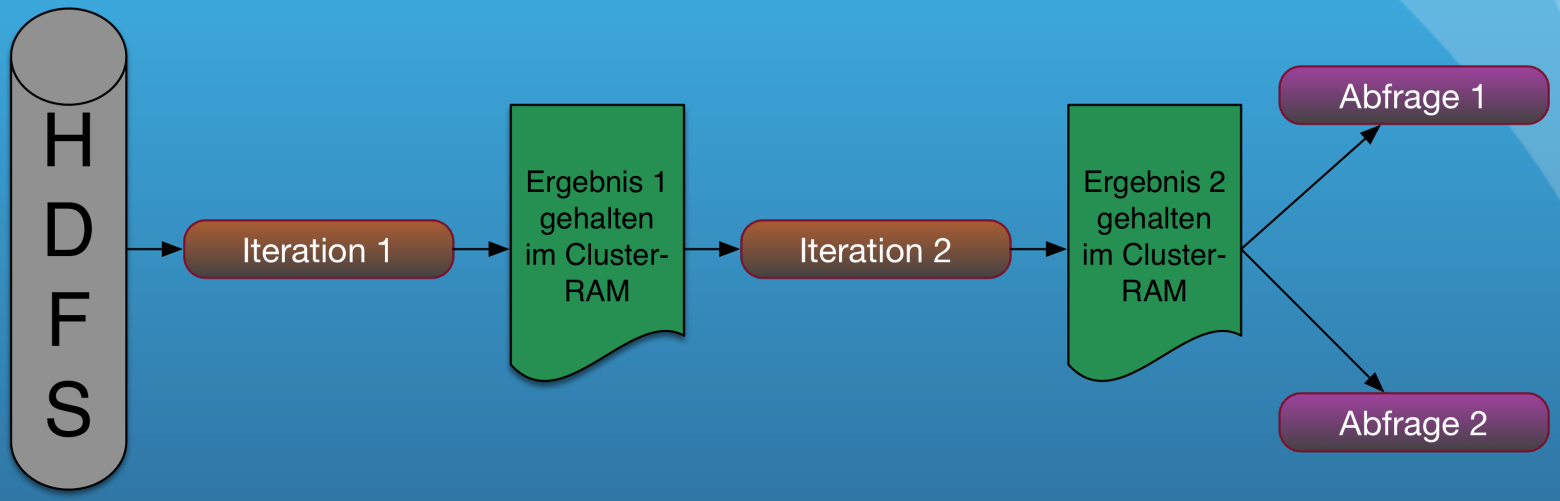
„Lightning-fast cluster computing“

Was ist Apache Spark?

- „...a fast and general engine for large-scale data“
- „Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk“
- „Write applications quickly in Java, Scala or Python“
- „...combines SQL, streaming, and complex analytics“
- „...runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, S3“
- „...is one of the most actively developed open source projects. It has over 465 contributors in 2014, making it the most active project in the Apache Software Foundation and among Big Data open source projects.“

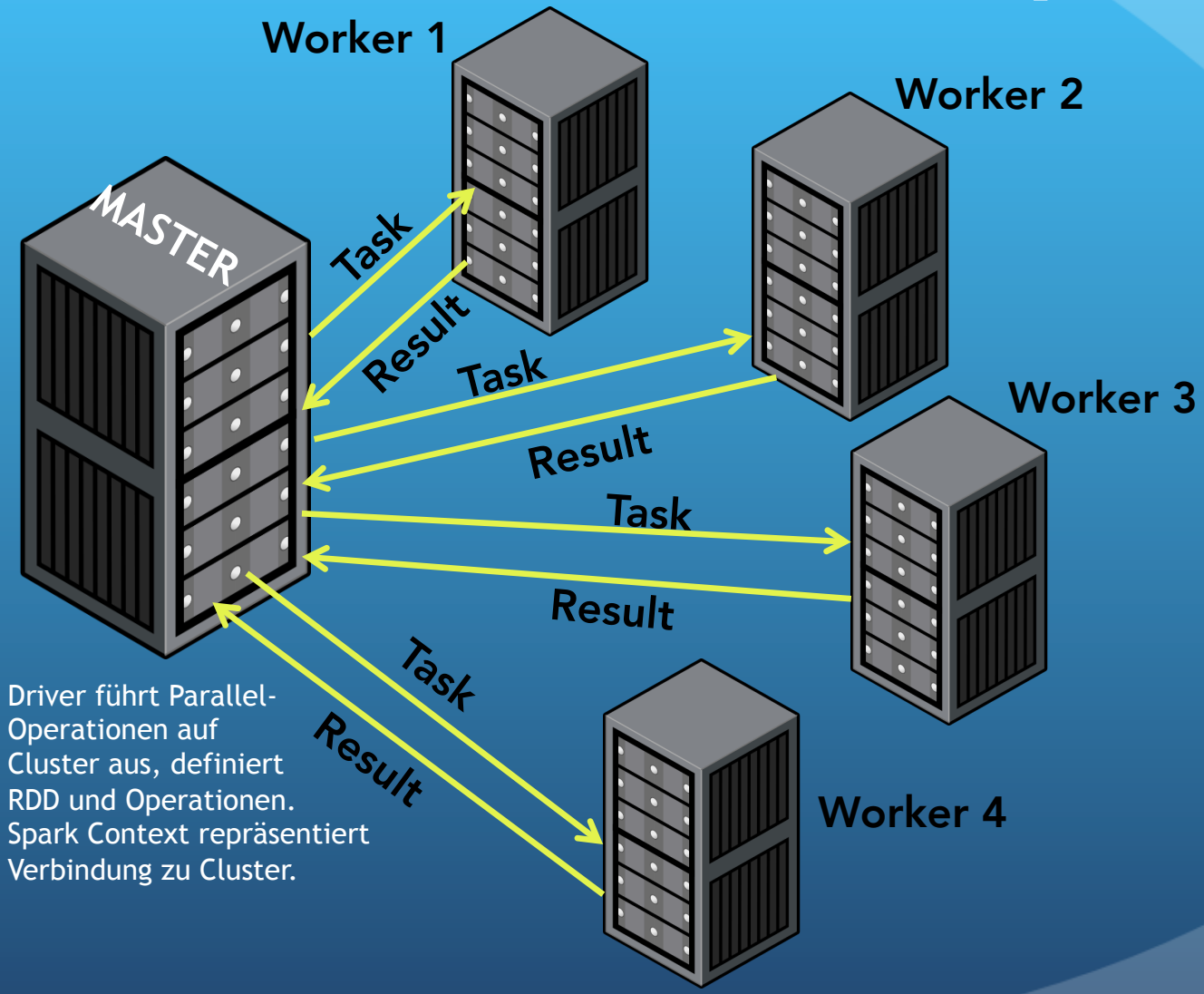
Quellen: Apache Foundation & Wikipedia

Wie funktioniert Spark?



Spark's In-Memory Computation Model

Wie funktioniert Spark?



Driver führt Parallel-Operationen auf Cluster aus, definiert RDD und Operationen. Spark Context repräsentiert Verbindung zu Cluster.

Die Resilient Distributed Datasets

- ...sind Kernkonzept von Spark.
- ...sind verteilte „Immutable Collections“ von Objekten.
- ...sind in verschiedene Partitionen aufgeteilt, die auf verschiedenen Knoten des Clusters berechnet werden können.
- ...können alle Arten von Objekten (Java, Scala, Python) enthalten.
- ...können auf zwei Arten erstellt werden:

  - durch Laden von externen Datensätzen

```
val lines = sc.textFile("/path/to/README.md")
```

  - durch das Verteilen einer Object-Collection (List, Set) im Treiber.

```
val lines = sc.parallelize(List("pandas", "i like pandas"))
```

Die Resilient Distributed Datasets

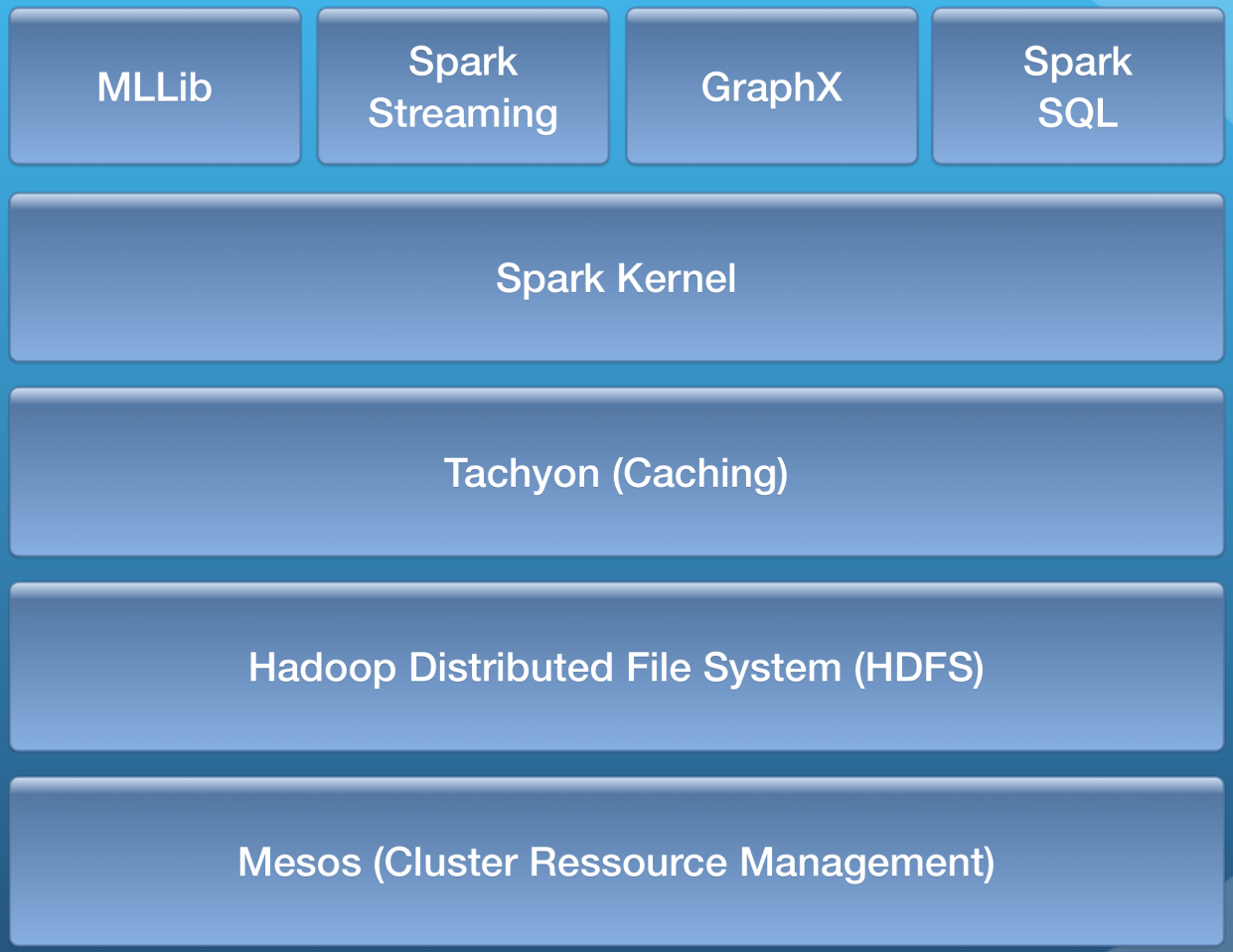
- Zwei Arten von Operationen: transformation und action
- Transformation: filter(), map(), union(), intersection(), subtract(), cartesian(),... – erzeugen jeweils neue RDDs
- Action: reduce(by x), count(), first(), take(n), collect(),... – lösen Berechnung aus, geben Wert an Master zurück, oder persistieren Daten
- Beispiel: `val sum = rdd.reduce((x, y) => x + y)`
- Beispiel:

```
val input = sc.parallelize(List(1, 2, 3, 4))
val result = input.map(x => x * x)
println(result.collect().mkString(","))
```

Die Resilient Distributed Datasets

- Resilient: Spark Master/Driver merkt sich die Transformationen, die zur Erstellung eines RDD geführt haben. Wenn Knoten ausfällt, können betreffende RDDs rekonstruiert werden.
- Persistenz: Spark berechnet bei jeder Action die RDDs inklusive aller Abhängigkeiten neu. Besonders für iterative Algorithmen teuer – RDD können für Wiederverwendung persistiert werden.
- Persistenzlevel: MEMORY_ONLY, MEMORY_ONLY_SER, MEMORY_AND_DISK, MEMORY_AND_DISK_SER, DISK_ONLY.

Der Berkeley Data Analytics Stack



Streaming, GraphX, Spark SQL

- Spark Streaming: Ermöglicht Verarbeitung von Live-Datenstömen (Produktionslogs, Messages, Tweets...). API ist nah an den RDD-Basisfunktionen um Prozesse auf persistente und Streamingdaten anwenden zu können.
- GraphX: Bibliothek zur Manipulation von Graphen (z.B. soziale Netzwerke) um hier Parallelverarbeitung zu ermöglichen. API für RDDs ist erweitert um Knoten und Kanten zu erstellen. Standardalgorithmen wie PageRank.
- Spark SQL: Paket für strukturierte Daten (Hive Tables, Parquet, JSON). Ermöglicht den Mix von SQL mit Analysefunktionen auf RDDs innerhalb einer Applikation.

MLLib

- Machine Learning Library
- Enthält verschiedene Arten von Machine-Learning-Algorithmen (Classifikation, Regression, Colaborative Filtering, Clustering)
- Bietet Funktionen zur Modellevaluierung und zum Datenimport.
- Enthält einige ML-Primitiven (z.B. Gradientenverfahren)
- Alle Elemente sind so entworfen, dass sie auf Cluster gut skalieren.

Ein Beispiel: Clustering

```
import org.apache.spark.mllib.clustering.KMeans

// Load and parse the data
val data = sc.textFile("kmeans_data.txt")
val parsedData = data.map( _.split(' ').map(_.toDouble))

// Cluster the data into two classes using KMeans
val numIterations = 20
val numClusters = 2
val clusters = KMeans.train(parsedData, numClusters, numIterations)

// Evaluate clustering by computing Within Set Sum of Squared Errors
val WSSSE = clusters.computeCost(parsedData)
println("Within Set Sum of Squared Errors = " + WSSSE)
```

Zweck der Masterarbeit

- Einarbeitung in die Konzepte von Spark
- Aufbau verschiedener Clusterumgebungen
- Definition von Metriken
- Test und Messungen aller Bibliotheken unter verschiedenen Konfigurationen
- Implementierung, Test und Messungen von eigenen Komponenten gegen die APIs der Bibliotheken
- Vergleich mit Alternativimplementierungen (Flink, H2O)

Vielen Dank für die Aufmerksamkeit!

Gibt es Fragen?

Gerne auch im Anschluss per Email an:

sascha.lorenz@contexagon.com