Schneller als Hadoop?

Einführung in Spark Cluster Computing

Agenda

1. Einführung
2. Motivation
3. Infrastruktur
4. Performance
5. Ausblick

EINFÜHRUNG

Spark

- In-Memory Cluster Computing System
- Schnelle Analyse und Verarbeitung großer Datenmengen
- Unterstützung von Hadoop‘s Speichermodulen
  - HBase, HDFS, SequenceFiles
- Scala, Python und Java API‘s

Entstehung

2009

- Projektstart an der University of California, Berkeley

2010

- Freigabe als Open Source Projekt

2013

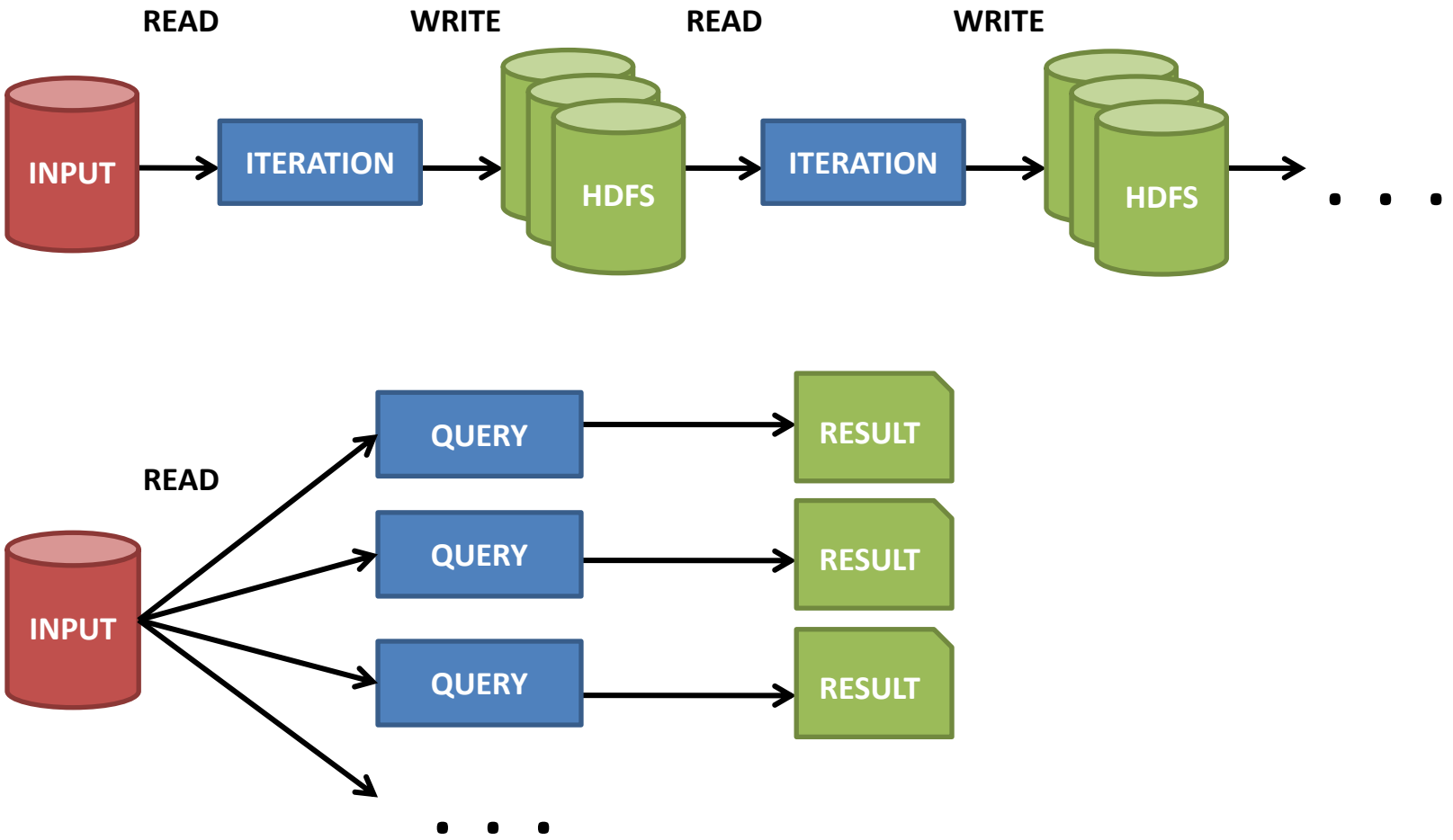
- Apache Incubator Projekt
- 25 Firmen mit 90 Entwicklern unterstützen die Entwicklung

1. MOTIVATION

Warum ein neues Programmiermodell?

- MapReduce hat die Verarbeitung großer Datenmengen stark vereinfacht
- Ineffizient für Anwendungen die Daten bei parallelen Berechnungen wiederverwenden
  - Iteratives maschinelles Lernen
  - Graphalgorithmen (PageRank, Logistic Regression, K-Means)
  - Interaktives Data Mining

Datenzugriff bei Hadoop

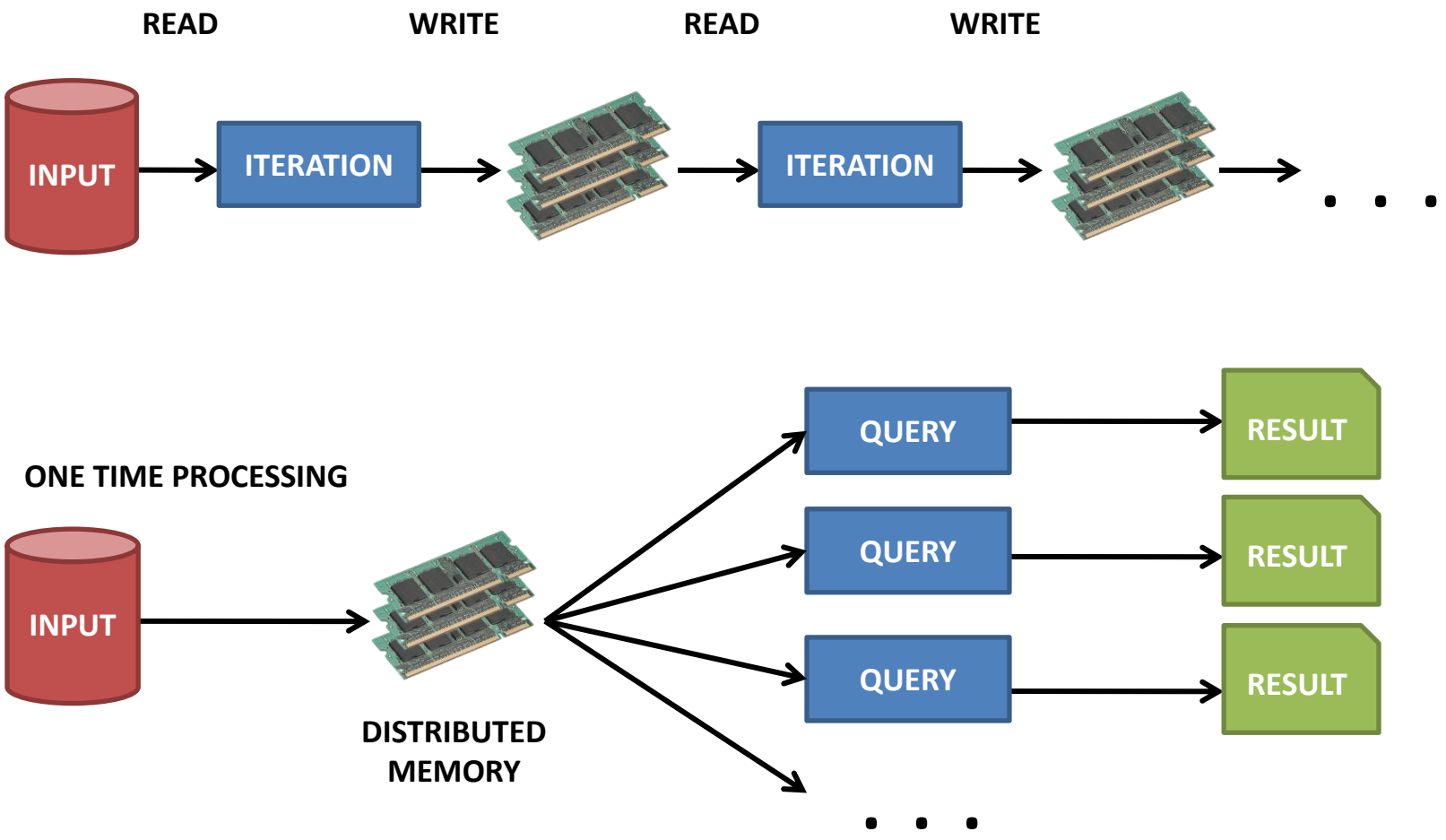


Probleme bei Hadoop

- Serialisierung
- Replikation
- Festplattenzugriffe

→ **Langsame Ausführung**

Datenzugriff bei Spark



Lösung bei Spark

- Speicherzugriffe
- Wiederverwendung
- Gemeinsamer Zugriff

→ 10x – 100x schnellere Ausführung

2. INFRASTRUKTUR

Resilient Distributed Datasets

- Schreibgeschützte und Partitionierte Datensätze
  - Keine Veränderung einzelner Datensätze
  - Veränderung durch Transformationen
- Verteilt über Clusterknoten im Arbeitsspeicher
- Fehlertolerant
  - Pflege einer Art Erblinie (Transformationen)
  - Wiederherstellung nach Ausfall eines Knotens

Operationen

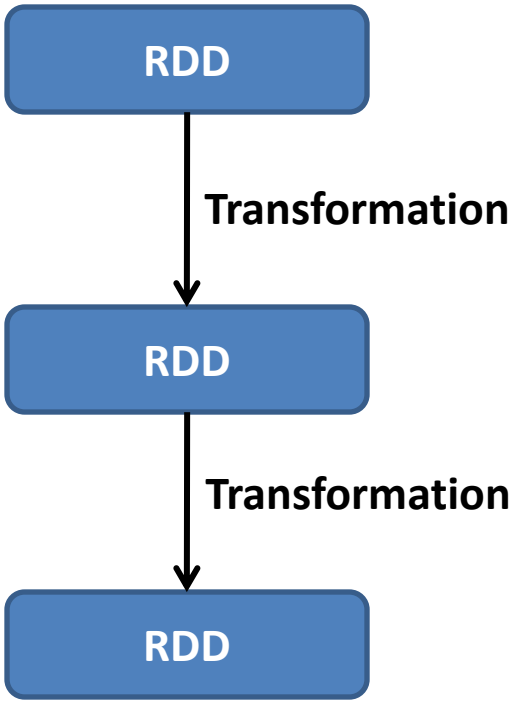
- **Transformation**
  - Definition von RDD (lazy operations)
  - Ergebnis = RDD
  - filter(*f* : T => Bool) : RDD[T] => RDD[T]
- **Aktion**
  - Ausführung von Berechnungen
  - Ergebnis = Wert
  - count() => RDD[T] => Long

Fehlertoleranz

Lineage Graph

- Transformationen werden automatisch protokolliert

- **Wiederherstellung**
  - Abarbeitung der Abhängigkeiten einer Partition
  - Parallel auf verschiedenen Clusterknoten



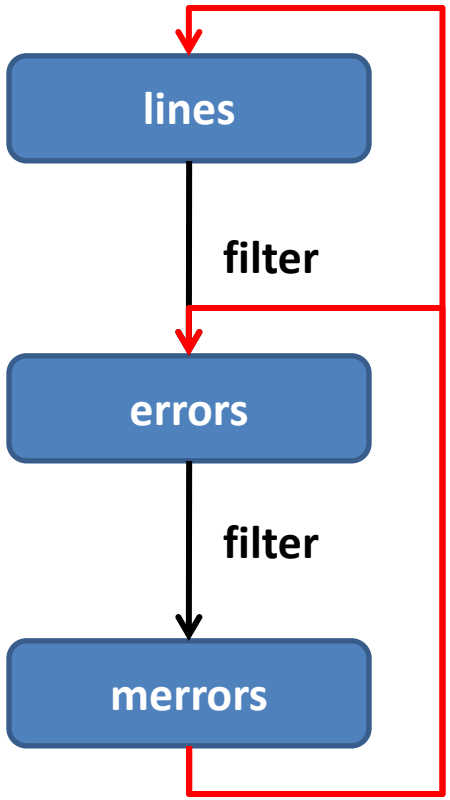
Fehlertoleranz

Log Mining

Quelltext

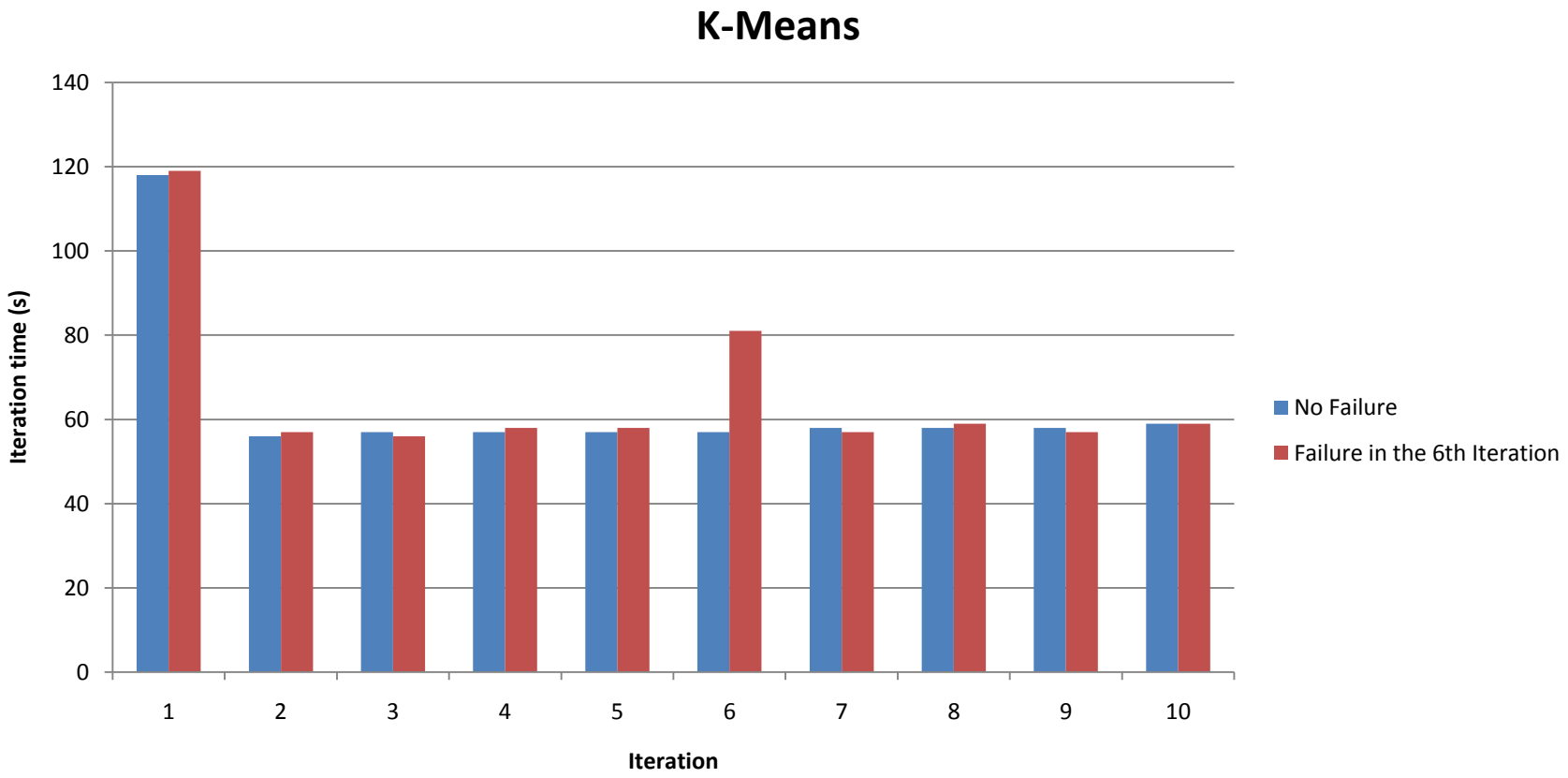
```
// Einlesen der Textdatei (Stabiler Speicherbereich)
lines  = spark.textFile()
// Filter>ung (Transformation)
error = lines.filter(.startsWith(„ERROR“))
// Speicherung (Checkpointing)
error.persist()
// Filterung (Transformation)
merrors = errors.filter(_.contains(„MySQL“))
// Zählen (Aktion)
merrors.count()
```

Lineage Graph



Fehlertoleranz

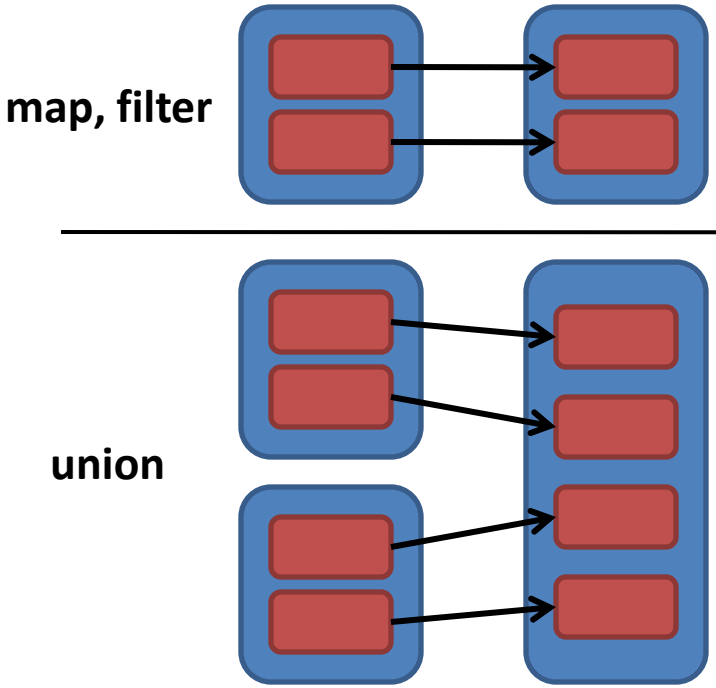
75 Clusterknoten, 100 GB Daten, 10 Iterationen



Anhängigkeiten

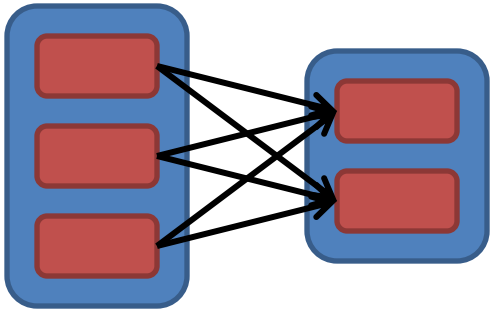
Narrow Dependencies

- Elternpartition von einer Kindpartition genutzt



Wide Dependencies

- Mehrere Kindpartition von mehreren Elternpartitionen abhängig



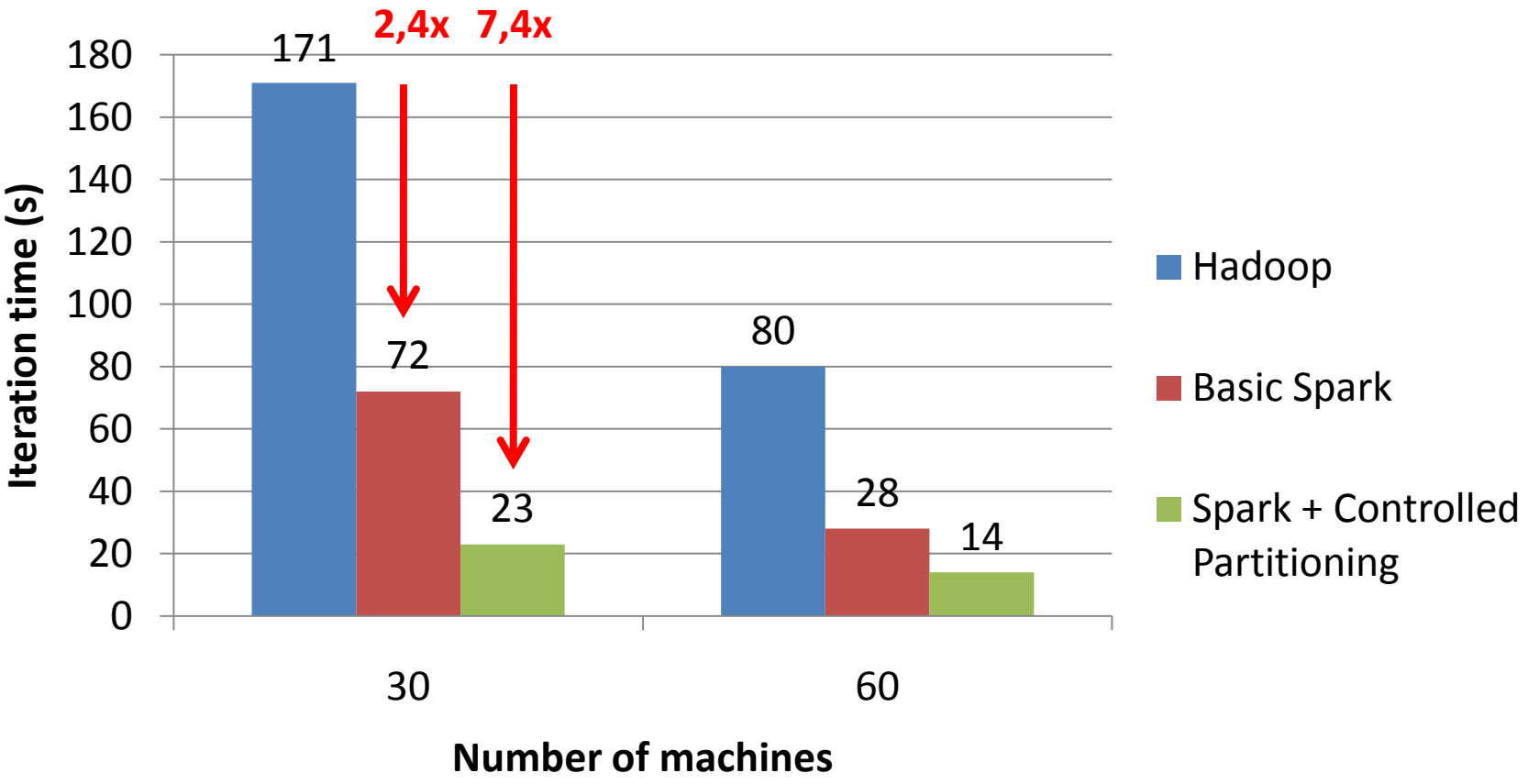
Feintuning

- **Individuell kontrollierte Partitionierung**
  - Schlüssel (Wie)
  - Platzierung (Wo)
- **Individuell kontrollierte Persistierung**
  - RDD (Was)
  - Zeitpunkt (Wann)
  - Ort (Wo)
    - In-Memory
    - Disk

3. PERFORMANCE

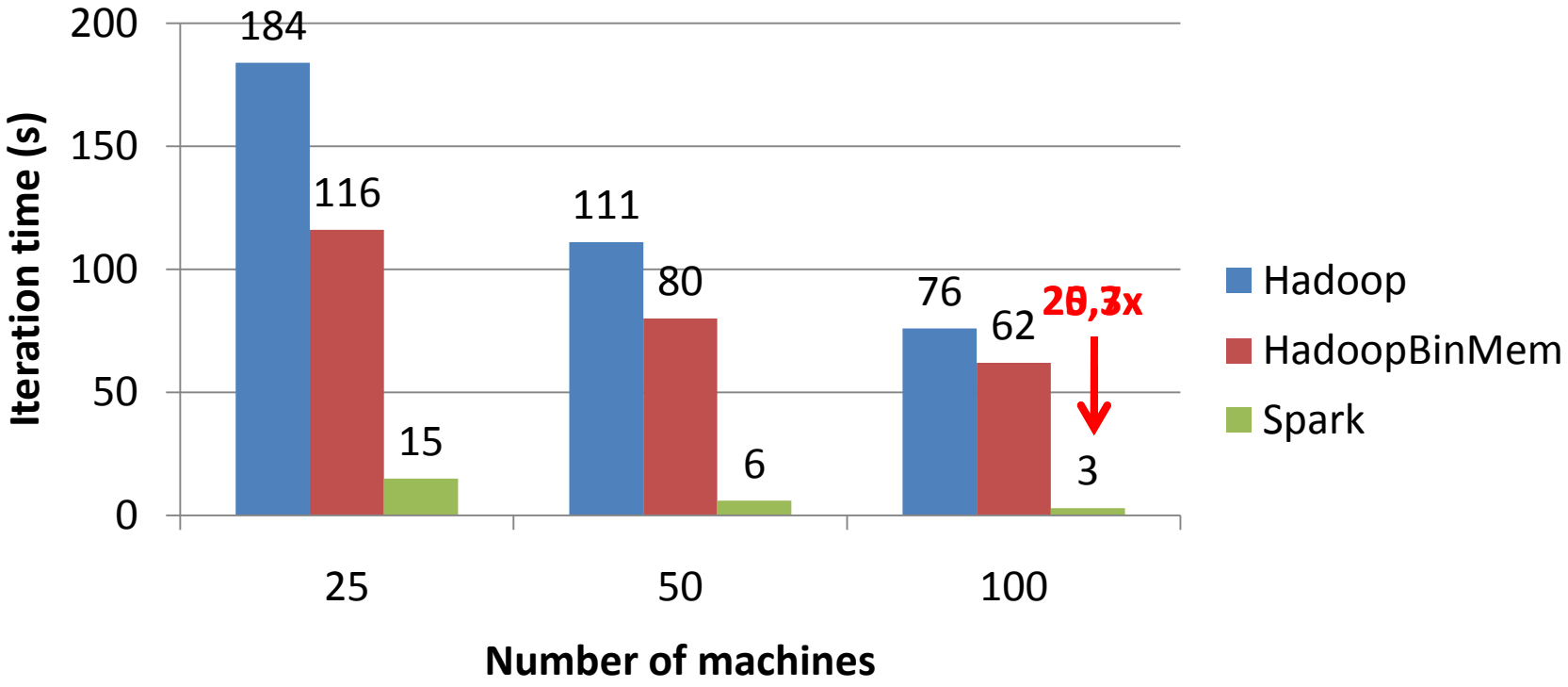
PageRank

54 GB Daten, 10 Iterationen



Logistic Regression

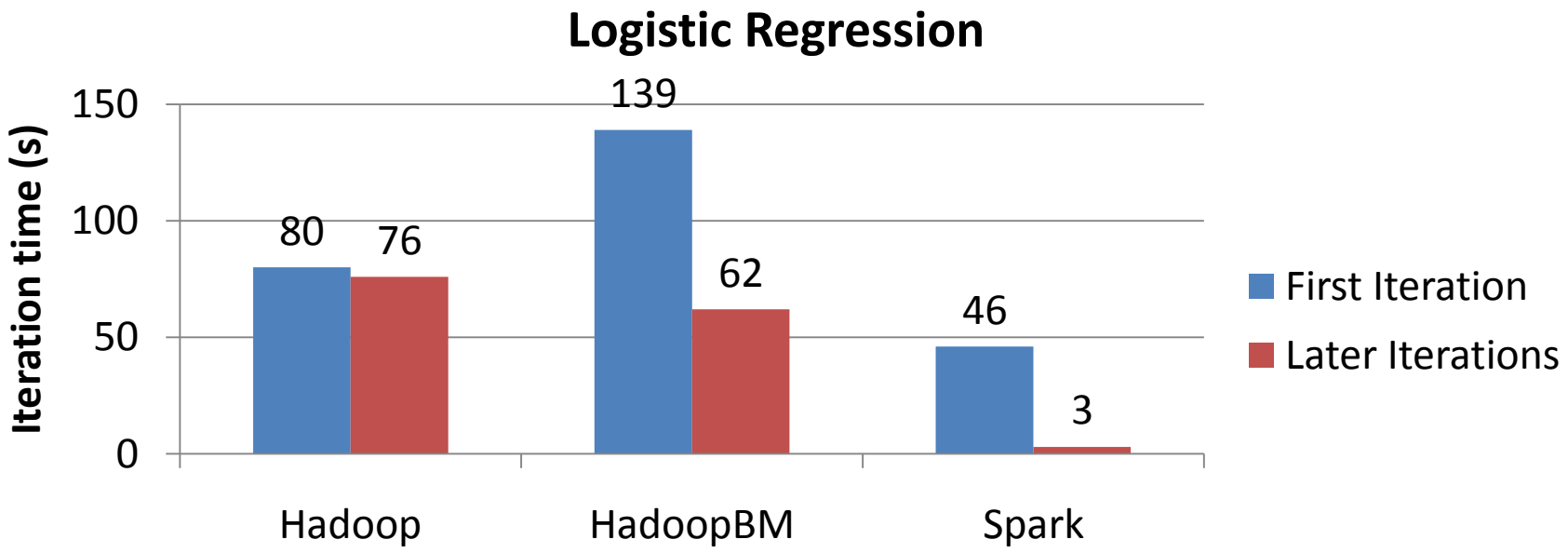
100 GB Daten, 10 Iterationen



Warum ist Spark 20,7x schneller als HadoopBinMem?

Logistic Regression

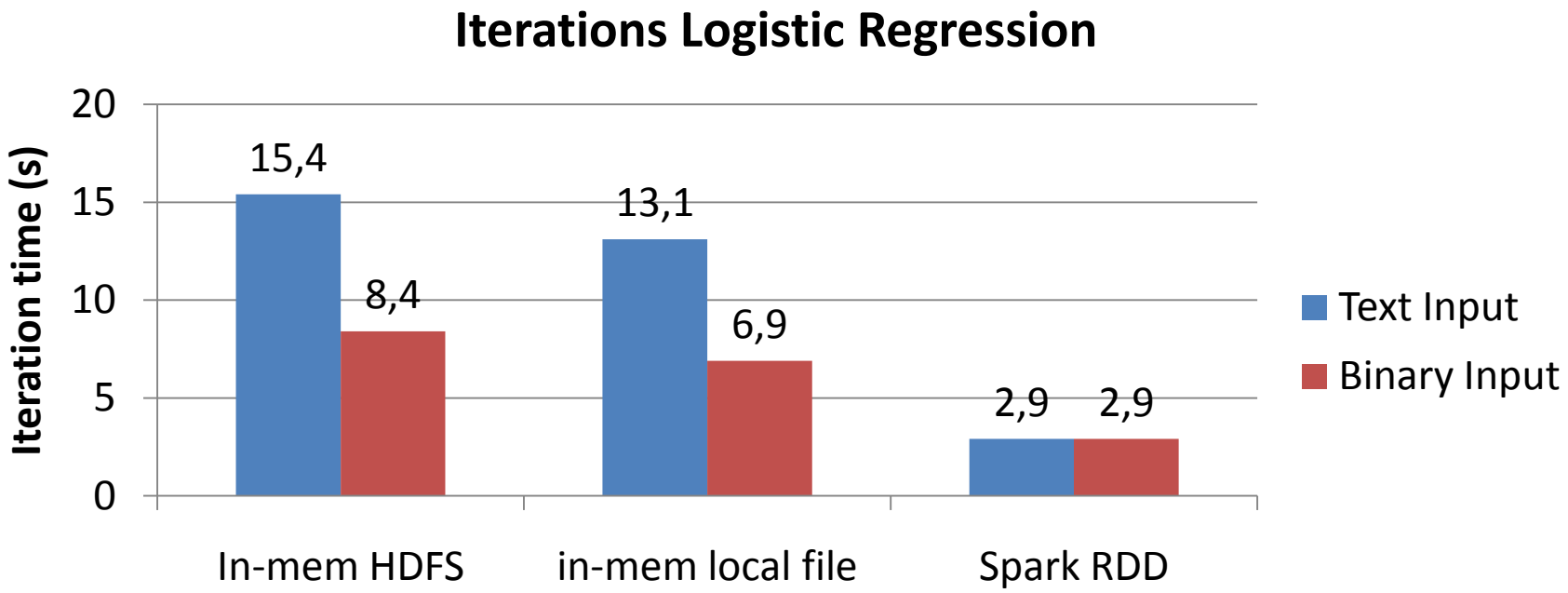
100 GB Daten, 10 Iterationen



HadoopBinMem wandelt Eingabedaten bei der ersten Iteration in Binärformat um

Logistic Regression

1 Machine, 256 MB Daten, 1 Iteration

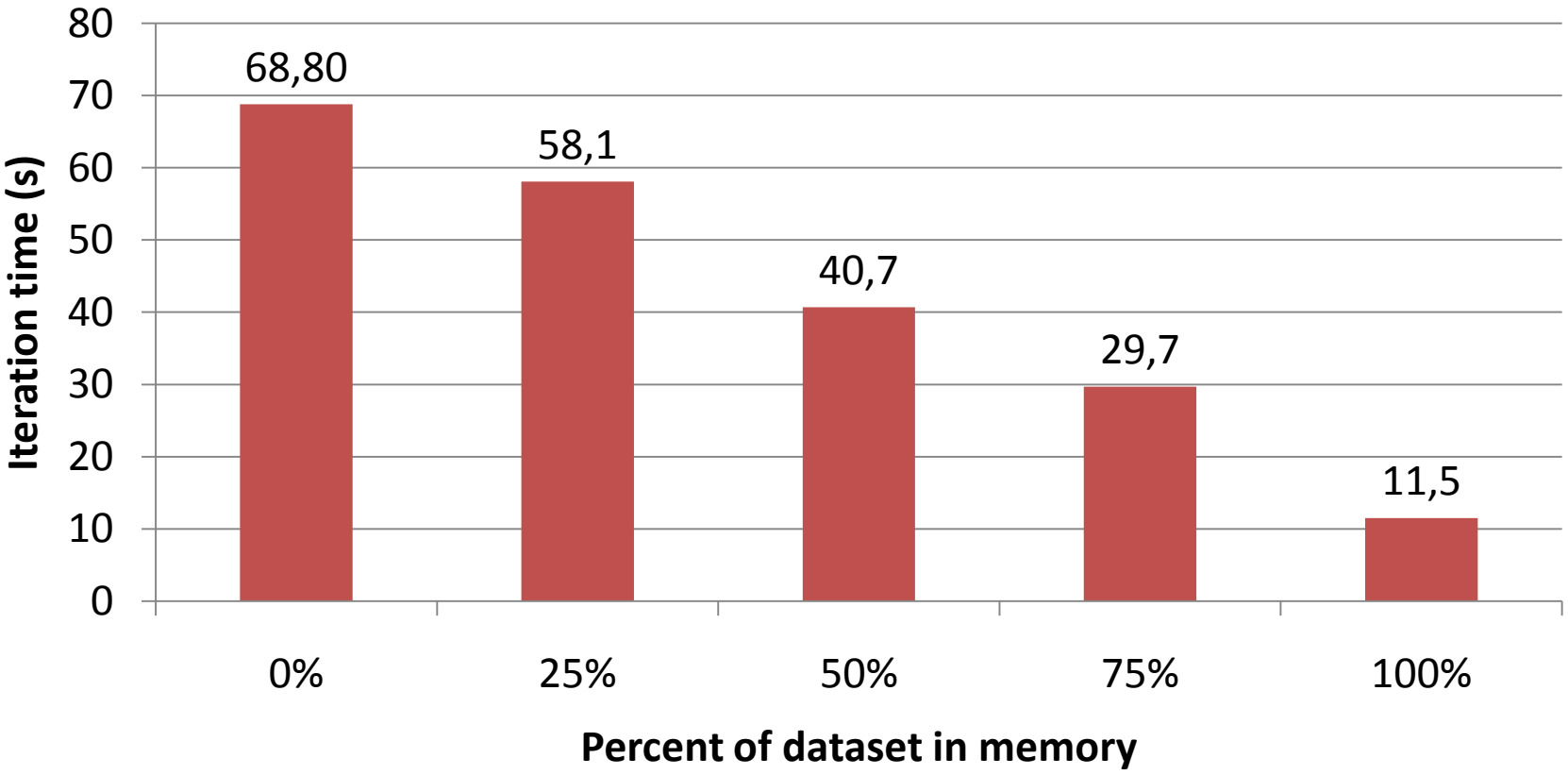


7 Sekunden Umwandlung in Binärformat

3 Sekunden Deserialisierung = Laufzeit der Logistic Regression

Logistic Regression

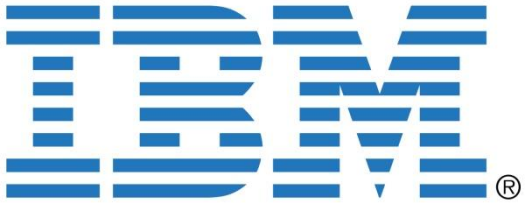
75 Machinen, 100 GB Daten, 10 Iterationen



4. AUSBLICK

Spark Anwender

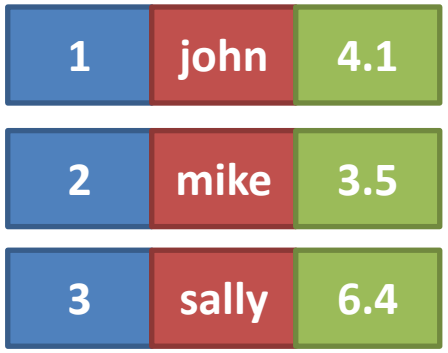
36 Firmen!



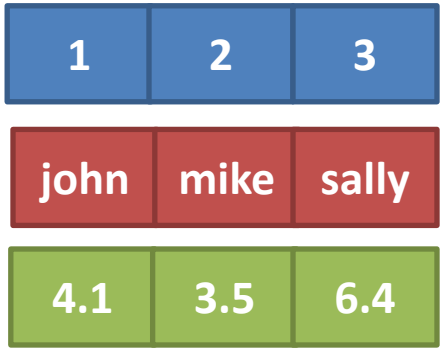
Shark

- Data Warehouse System Hive mit Spark
- Effiziente Speicherung
  - Spaltenorientiert mit Arrays aus primitiven Datentypen
  - Datengröße fast wie bei Serialisierung und 5x schnellerer Datenzugriff

Row Storage



Column Storage



Quellen

- **Websites**
  - http://spark-project.org/research/
  - http://spark.incubator.apache.org/
  - http://cwiki.apache.org/confluence/display/SPARK/Wiki+Homepage
  - http://github.com/amplab/shark/wiki
- **Forschungsbericht**
  - http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

Vielen Dank für Ihre Aufmerksamkeit!

Haben Sie noch Fragen?