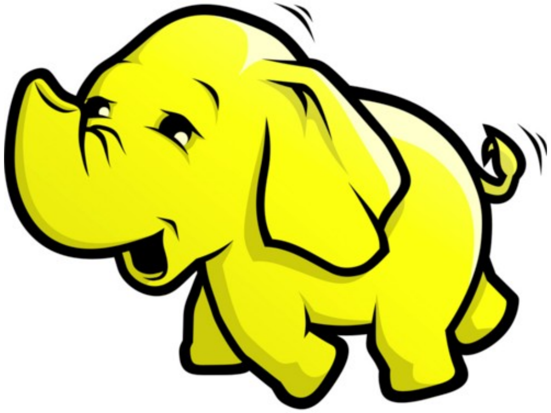
Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Übung Datenbanksysteme II

Web-Scale Data Management

Leon Bornemann

Folien basierend auf
Maximilian Jenders,
Thorsten Papenbrock



- Feedback praktische Übung
  - Abgabetermin?
  - Zeitaufwand?
- Stand Vorlesung

MapReduce: Introduction

- MapReduce …
  - is a **paradigm** derived from functional programming.
  - is implemented as **framework**.
  - operates primarily **data-parallel** (not task-parallel).
  - **scales-out** on multiple nodes of a cluster.
  - uses the Hadoop distributed filesystem.
  - is designed for **Big Data Analytics**:
    - Log-files
    - Weather-statistics
    - Sensor-data
    - …
- “Competitors“:

Spark

Stratosphere

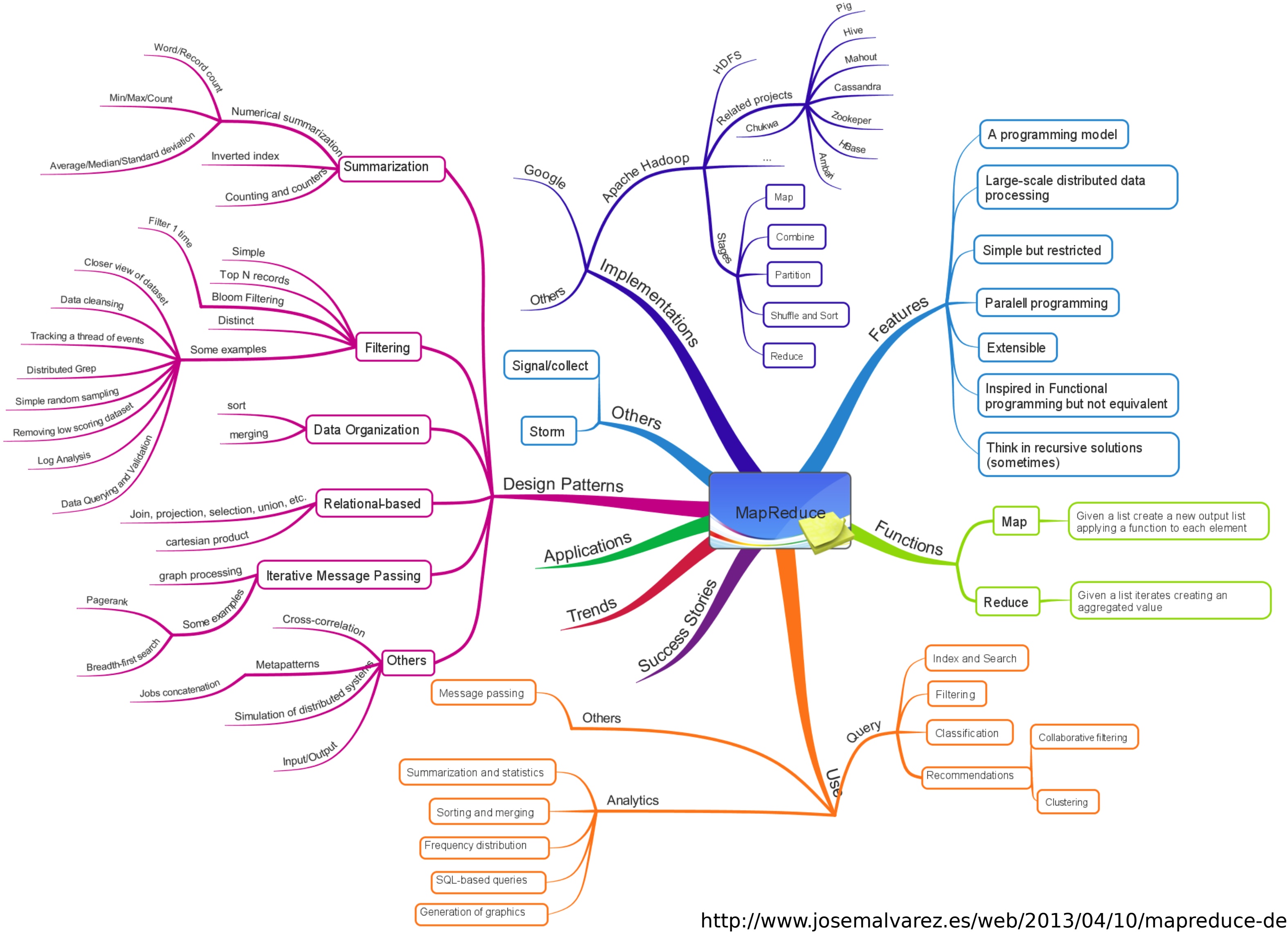
Introduction

- Who is using Hadoop?
  - Yahoo!
    - Biggest cluster: *2000 nodes*, used to support research for Ad Systems and Web Search.
  - Amazon
    - Process millions of sessions daily for analytics, using both the Java and streaming APIs. Clusters vary from *1 to 100 nodes*.
  - Facebook
    - Use Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics. *600 machine* cluster.
  - ...

http://wiki.apache.org/hadoop/PoweredBy

MapReduce: Introduction

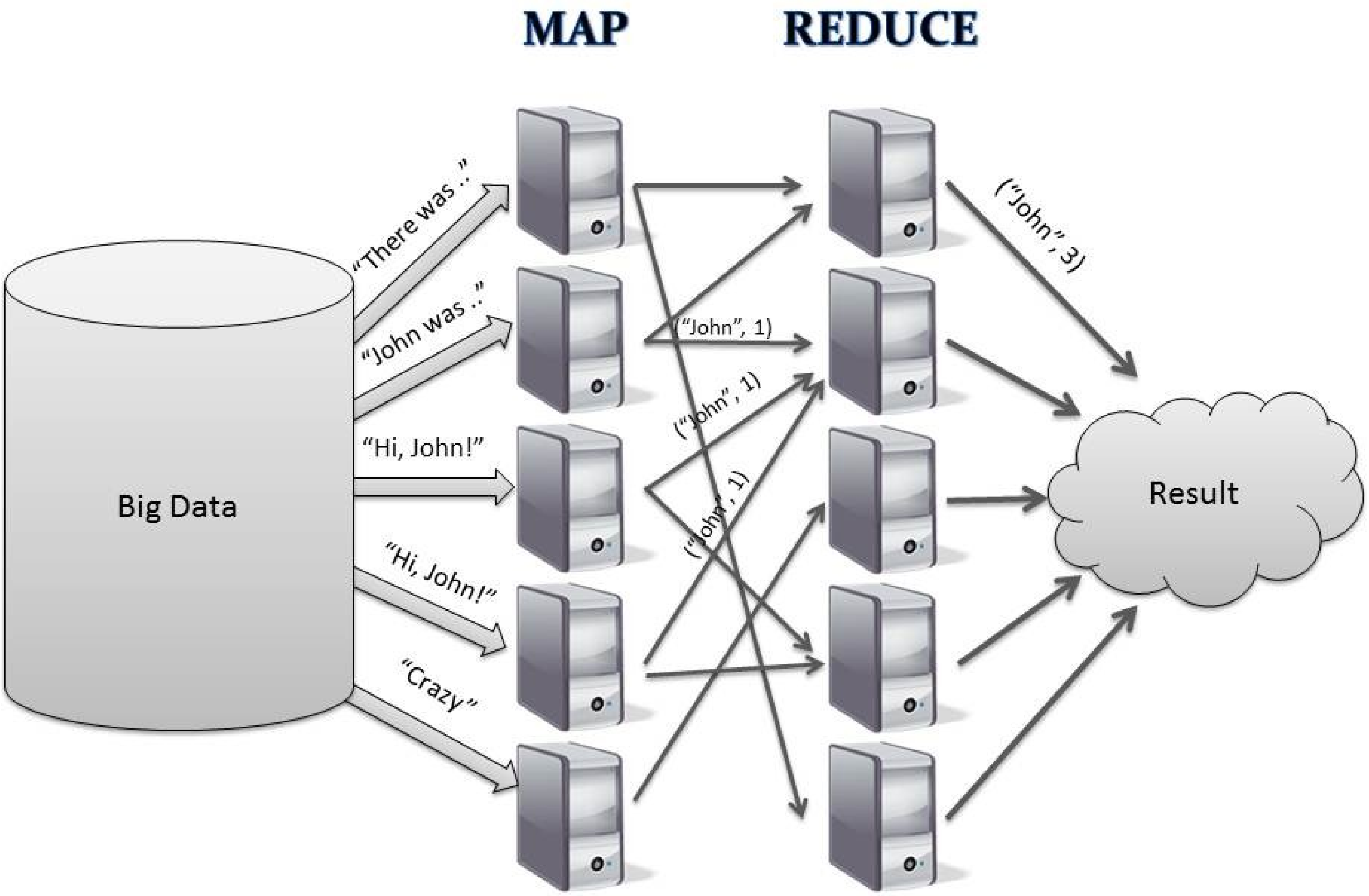
HPI Hasso Plattner Institut



- MapReduce
  - Design Patterns
    - Summarization
      - Numerical summarization
        - Word/Record count
        - Min/Max/Count
        - Average/Median/Standard deviation
      - Inverted index
      - Counting and counters
    - Filtering
      - Filter 1 time
      - Simple
      - Top N records
      - Bloom Filtering
      - Distinct
      - Some examples
        - Closer view of dataset
        - Data cleansing
        - Tracking a thread of events
        - Distributed Grep
        - Simple random sampling
        - Removing low scoring dataset
        - Log Analysis
        - Data Querying and Validation
    - Data Organization
      - sort
      - merging
    - Relational-based
      - Join, projection, selection, union, etc.
      - cartesian product
    - Iterative Message Passing
      - graph processing
      - Some examples
        - Pagerank
        - Breadth-first search
    - Others
      - Cross-correlation
      - Metapatterns
        - Jobs concatenation
      - Simulation of distributed systems
      - Input/Output
  - Implementations
    - Google
    - Apache Hadoop
      - HDFS
      - Related projects
        - Pig
        - Hive
        - Mahout
        - Cassandra
        - Zookeper
        - HBase
        - Ambari
        - Chukwa
      - ...
      - Stages
        - Map
        - Combine
        - Partition
        - Shuffle and Sort
        - Reduce
    - Others
  - Others
    - Signal/collect
    - Storm
  - Features
    - A programming model
    - Large-scale distributed data processing
    - Simple but restricted
    - Paralell programming
    - Extensible
    - Inspired in Functional programming but not equivalent
    - Think in recursive solutions (sometimes)
  - Functions
    - Map
      - Given a list create a new output list applying a function to each element
    - Reduce
      - Given a list iterates creating an aggregated value
  - Use
    - Query
      - Index and Search
      - Filtering
      - Classification
      - Recommendations
        - Collaborative filtering
        - Clustering
    - Others
      - Message passing
    - Analytics
      - Summarization and statistics
      - Sorting and merging
      - Frequency distribution
      - SQL-based queries
      - Generation of graphics
  - Applications
  - Trends
  - Success Stories

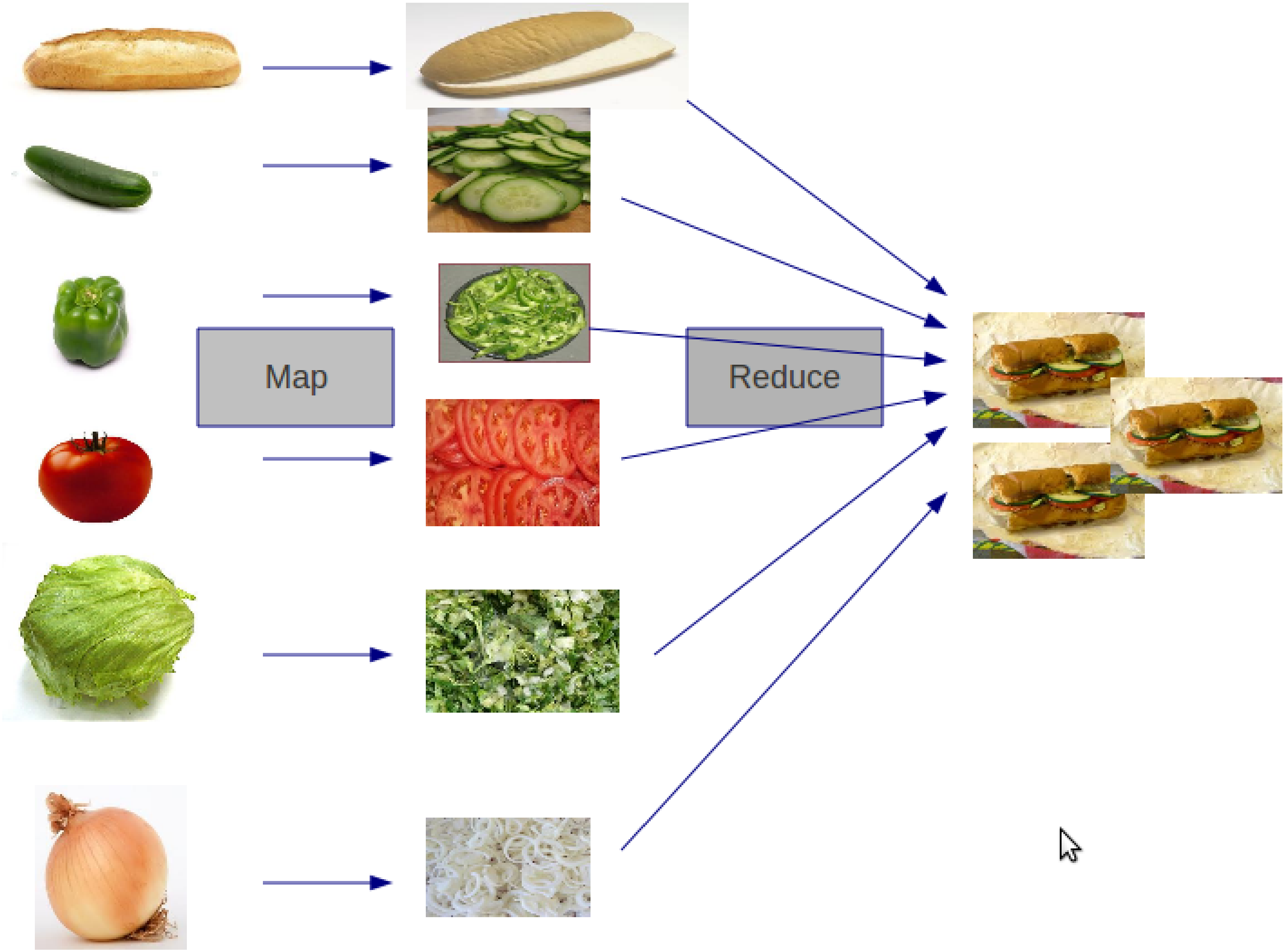
http://www.josemalvarez.es/web/2013/04/10/mapreduce-design-patterns/

MapReduce: Introduction



http://dme.rwth-aachen.de/de/research/projects/mapreduce

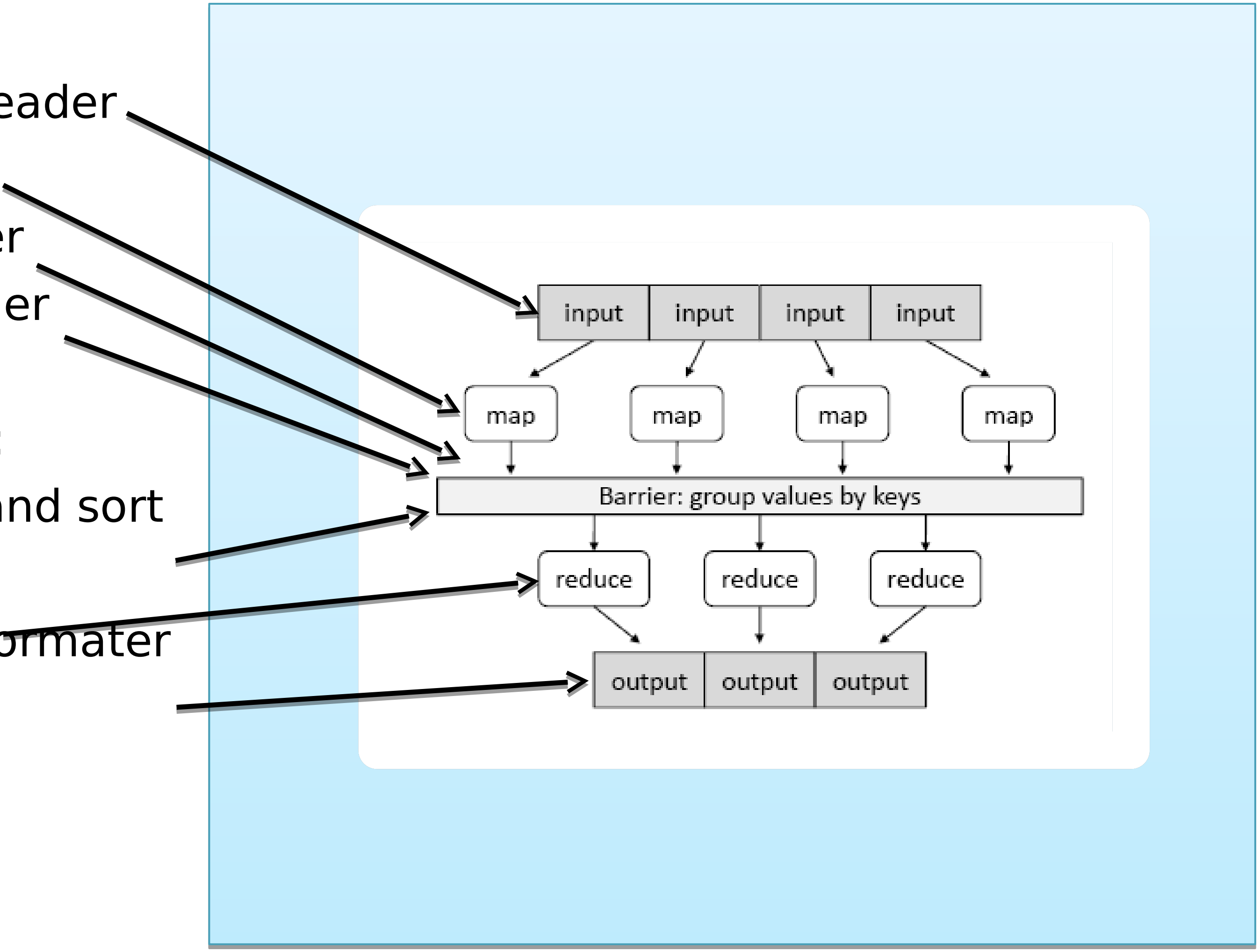
MapReduce:
Introduction



http://mohamednabeel.blogspot.de/2011/03/starting-sub-sandwitch-business.html

MapReduce:
Phases

- map-task:
  - record reader
  - mapper
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - reducer
  - output formater



MapReduce:

Phases

Nicht zwangsweise

- map-task:
  - record reader
  - mapper
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - reducer
  - output formater

- Input: `<data entry>` (row/split/ m)
- Output: <`key, record`>

- “`key`“ is usually positional information
- “`record`“ represents a raw data record

- Translates a given input into records
- Parses data into records but not the records itself

MapReduce:

Phases

- map-task:
  - record reader
  - mapper
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - reducer
  - output formater

- Input: <key, record>
- Output: <key*, value>

- “key*“ is a problem-specific key
  - e.g. the word for the word-count-task
- “value“ is a problem-specific value
  - e.g. “1“ for the occurence of a word

- Executes user defined code that starts solving the given task
- Defines the grouping of the data

- A single mapper can emit multiple <key*, value> output pairs for a single <key, record> input pair

In der Praxis oft „flatmap“ genannt

MapReduce:

Phases

- map-task:
  - record reader
  - mapper
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - reducer
  - output formater

- Input: `<key*, values>`
- Output: `<key*, value>`

- “key*“ is a problem-specific key
  - e.g. the word for the word-count-task
- “value“ is a problem-specific value
  - e.g. “1“ for the occurence of a word

- Executes user defined code that merges a set of values
- Pre-aggregates values to reduce network traffic
- Is an optional, localized reducer

Beispiel folgt gleich

MapReduce: Phases

- map-task:
  - record reader
  - mapper
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - reducer
  - output formater

- Input: `<key*, value>`
- Output: `<key*, value> + reducer`
- “`reducer`“ is the reducer number that should handle this key/value pair; reducer might be located on other compute nodes
- Distributes the keyspace randomly to the reducers
- Calculates the reducer by e.g. `key*.hashCode() % (number of reducers)`

MapReduce:

Phases

- map-task:
  - record reader
  - mapper
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - reducer
  - output formater

- Input: `<key*, value> + reducer`
- Output: `<key*, value> + reducer`
- Downloads the `<key*, value>` data to the local machines that run the corresponding reducers

MapReduce: Phases

- map-task:
  - record reader
  - mapper
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - reducer
  - output formater

- Input: `<key*, values>`
- Output: `<key*, result>`

- “`result`“ is the solution/answer for the given “key*“

- Executes user defined code that merges a set of values
- Calculates the final solution/answer to the problem statement for the given key

MapReduce:
Phases

- map-task:
  - record reader
  - mapper
  - combiner
  - partitioner

- reduce-task:
  - shuffle and sort
  - reducer
  - output formater

- Input: `<key*, result>`
- Output: `<key*, result>`

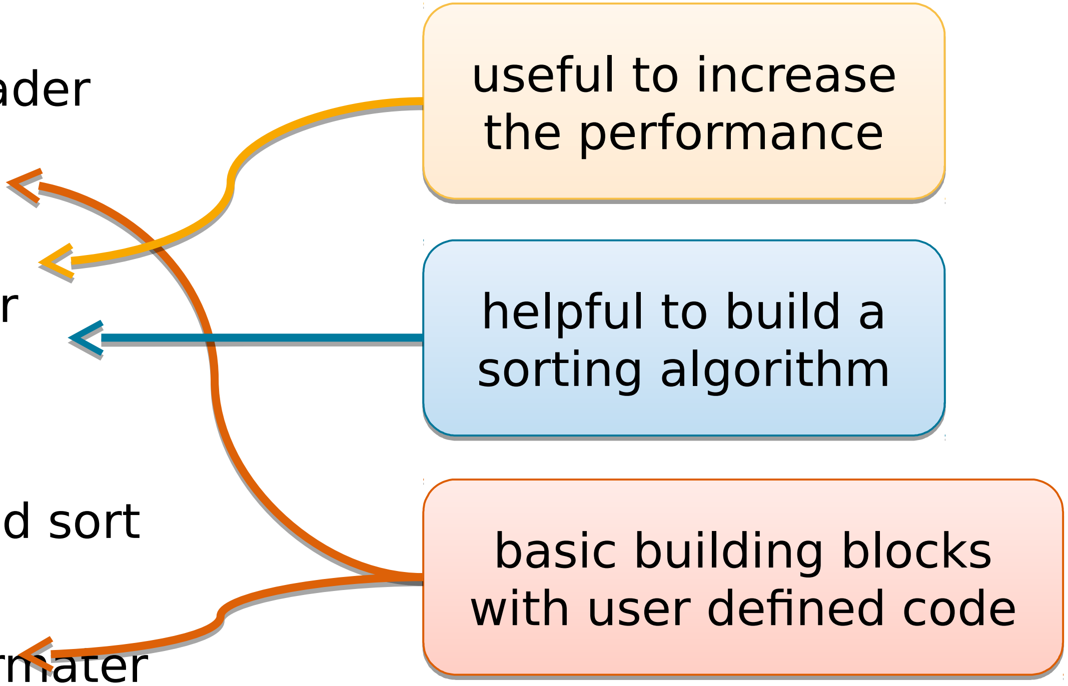
- Writes the key/result pairs to disk
- Formates the final result and writes it record-wise to disk

MapReduce:

Phases

- map-task:
  - record reader
  - **mapper**
  - combiner
  - partitioner

- reduce-task:
  - shuffle and sort
  - **reducer**
  - output formater



useful to increase the performance

helpful to build a sorting algorithm

basic building blocks with user defined code

MapReduce:

Example 1: Distinct

- map-task:
  - record reader
  - **mapper**
  - combiner
  - partitioner

- reduce-task:
  - shuffle and sort
  - **reducer**
  - output formater

- Input:
  - A relational table instance *Car(name, vendor, color, speed, price)*
- Output:
  - A distinct list of all *vendors*

```
map (key, record) {
   emit (record.vendor, null);
}
```

```
reduce (key, values) {
   write (key);
}
```

MapReduce: Example 2: Index-Generation

- map-task:
  - record reader
  - **mapper**
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - **reducer**
  - output formater

- Input:
  - A relational table instance *Car(name, vendor, color, speed, price)*
- Output:
  - An index on *Car.vendor*

```
map (key, record) {
    emit (record.vendor, key);
}

reduce (key, values) {
    String refs = concat(values);
    write (key, refs);
}
```

MapReduce:

Example 3: Join

- map-task:
  - record reader
  - **mapper**
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - **reducer**
  - output formater

- Input:
  - Two relational table instances *Car(name, vendor, color, speed, price)* *Plane(id, weight, length, speed, seats)*
- Output:
  - All pairs of *cars* and *planes* with the same *speed*

MapReduce:
Example 3: Join

- map-task:
  - record reader
  - **mapper**
  - combiner
  - partitioner

- reduce-task:
  - shuffle and sort
  - **reducer**
  - output formater

Car(name, vendor, color, speed, price)
Plane(id, weight, length, speed, seats)

```
map (key, record) {
   emit (speed, {
     ‚table‘ -> table(record),
     ‚record‘ -> record});
}

reduce (speed, values) {
   cars = valuesWhere(‘table‘, ‘car‘);
   planes = valuesWhere(‘table‘, ‘plane‘);
   for (car : cars)
      for (plane : planes)
         write (car.record, plane.record);
}
```

MapReduce:
Example 4: Wordcount

- map-task:
  - record reader
  - **mapper**
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - **reducer**
  - output formater

- Input:
  - A text file, line by line
- Output:
  - The number of occurences of each word

MapReduce:

Example 4: Wordcount

- map-task:
  - record reader
  - **mapper**
  - combiner
  - partitioner

Combine summiert lokal → Reduziert Datentransfer vor Reduce-Phase

  - **reducer**
  - output formater

```
map (key, line) {
   for(word : line)
      emit (word,1);

combine(word,counts){
   emit(word,sum(counts));
}

reduce (word, counts) {
    write(word, sum(counts))
}
```

Kann man noch optimieren

MapReduce:

Example 5: Set Difference

- map-task:
  - record reader
  - **mapper**
  - combiner
  - partitioner
- reduce-task:
  - shuffle and sort
  - **reducer**
  - output formater

- Input:
  - Two Tables
  - R(A,B,C)
  - S(A,B,C)
- Output:
  - All tuples in R that are not in S

MapReduce:

Example 5: Set Difference

- map-task:
  - record reader
  - **mapper**
  - combiner
  - partitioner

- reduce-task:
  - shuffle and sort
  - **reducer**
  - output formater

```
map (key, record) {
   emit (record, table(record));
}

reduce (record, values) {
   isInS = values.contains(‘S‘);
   isInR = values.contains(‘R‘);
   if(isInR && !isInS)
      emit(record)
}
```