Modul

- Internet of Things (IoT) -

05 – Vorlesung *Big Data

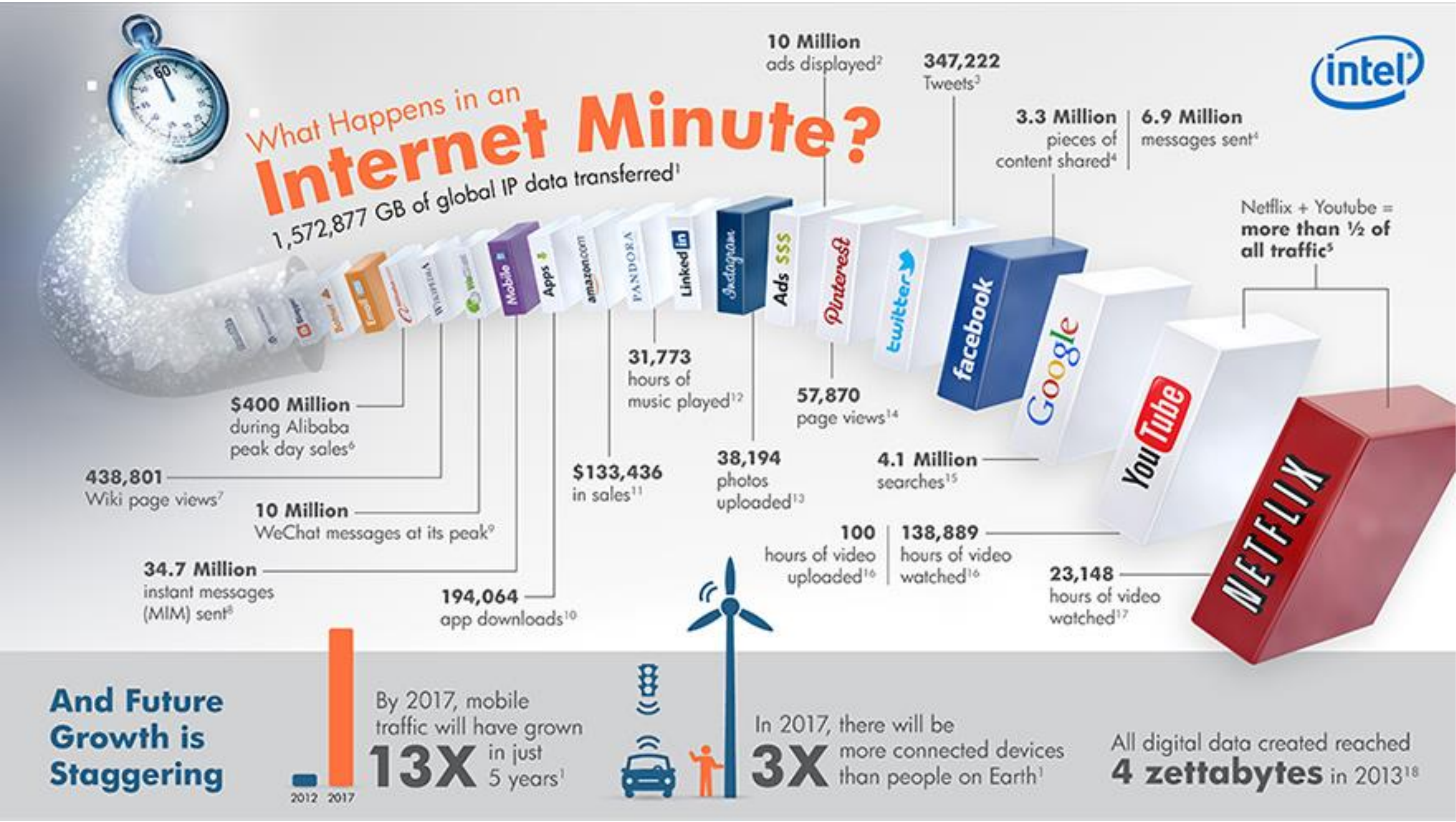
Prof. Dr. Marcel Tilly

Fakultät Informatik, Cloud Computing

Überblick

21. März	Einführung in das Internet der Dinge
28. März	IoT Architekturen
4. April	Things und Sensoren
11. April	From Device to Cloud
18. April	Vorlesungsfrei – Ostern
25. April	IoT Analytics
02. Mai	Big Data in IoT
9. Mai	Data Exploration
16. Mai	IoT Platformen
23. Mai	Entwicklung einer IoT Lösung
30. Mai	Vorlesungsfrei; Christi Himmelfahrt
05. Juni	opt. Gastvortrag – Digitalisierung
13. Juni	Data Science in IoT
20. Juni	Vorlesungsfrei – Fronleichnam
27. Juni	Intelligente Cloud und intelligente Edge
04. Juli	PStA Abschlusspraesentationen

PStA



What Happens in an
Internet Minute?
1,572,877 GB of global IP data transferred[1]
intel
10 Million ads displayed[2]
347,222 Tweets[3]
3.3 Million pieces of content shared[4]
6.9 Million messages sent[4]
Netflix + Youtube = more than ½ of all traffic[5]
$400 Million during Alibaba peak day sales[6]
438,801 Wiki page views[7]
10 Million WeChat messages at its peak[9]
34.7 Million instant messages (MIM) sent[8]
194,064 app downloads[10]
31,773 hours of music played[12]
$133,436 in sales[11]
38,194 photos uploaded[13]
57,870 page views[14]
4.1 Million searches[15]
100 hours of video uploaded[16]
138,889 hours of video watched[16]
23,148 hours of video watched[17]
And Future Growth is Staggering
2012
2017
By 2017, mobile traffic will have grown 13X in just 5 years[1]
In 2017, there will be 3X more connected devices than people on Earth[1]
All digital data created reached 4 zettabytes in 2013[18]

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Motivation

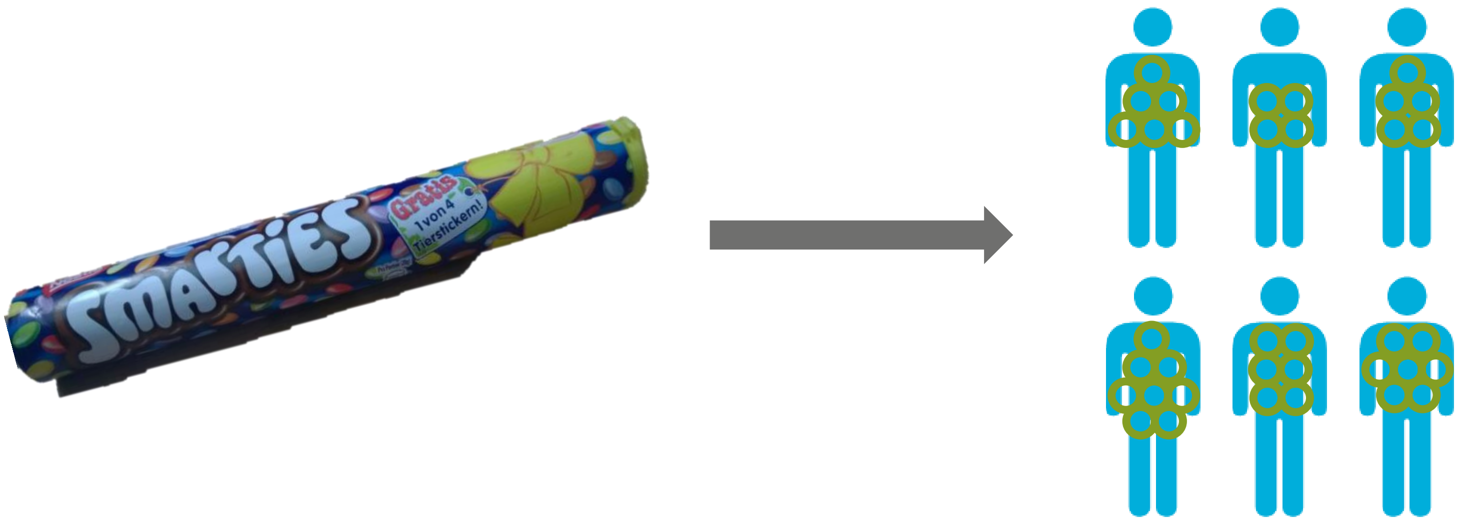
- Process lots of data
    - Google processed > 24 petabytes of data per day
- A single machine **cannot** serve all the data
    - You need a distributed system to store and process in parallel
- Parallel programming?
    - Threading is hard!
    - How do you facilitate communication between nodes?
    - How do you scale to more machines?
    - How do you handle machine failures?

MapReduce

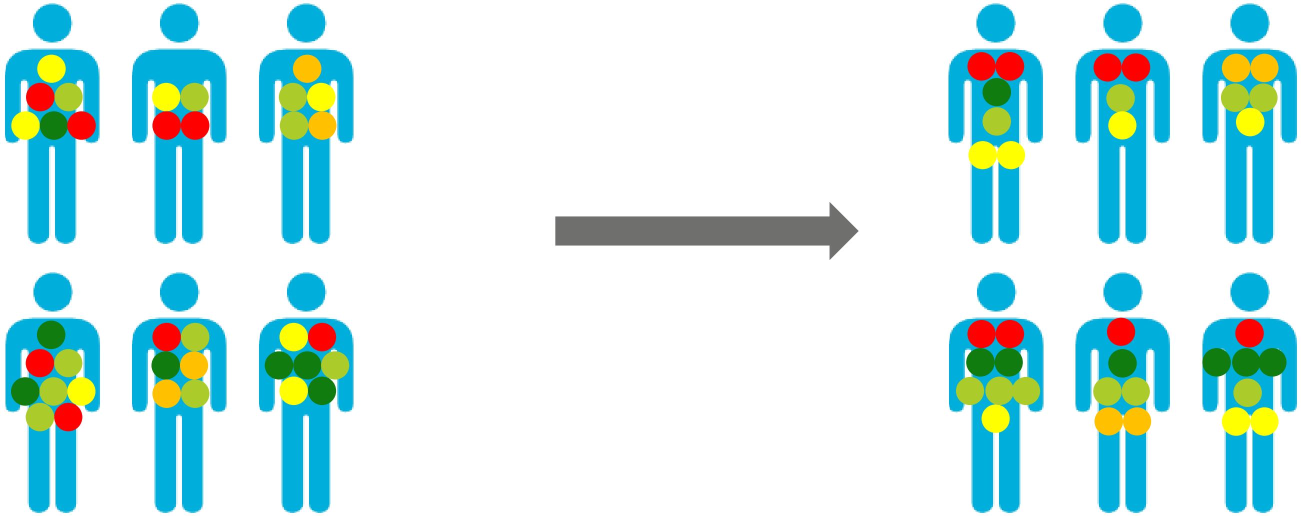
- MapReduce provides
  - Automatic parallelization, distribution
  - I/O scheduling
    - Load balancing
    - Network and data transfer optimization
  - Fault tolerance
    - Handling of machine failures

- Need more power: Scale out, not up!
    - Large number of commodity servers as opposed to some high end specialized servers

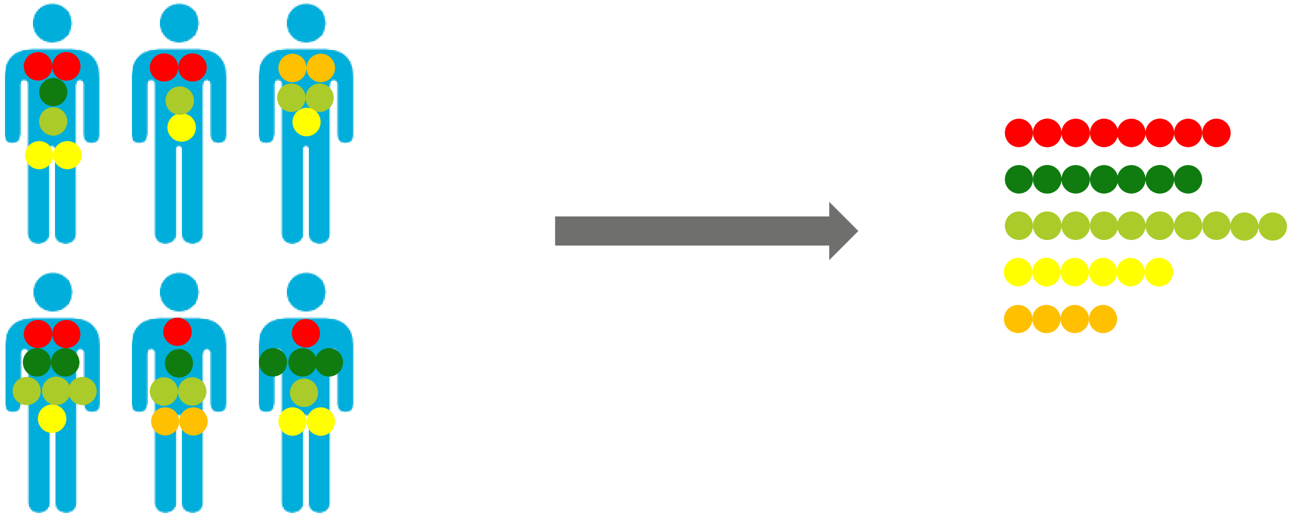
Introduction into MapReduce



MapReduce: Map



MapReduce: Reduce



Typical problem solved by MapReduce

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, or transform
- Write the results

How it started?

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Extension to Google File System (GFS, 2003)
- MapReduce paper published 2004 at OSDI
    - Used to recalculate search indices
- Became synonymous for BigData

➔ Hadoop

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

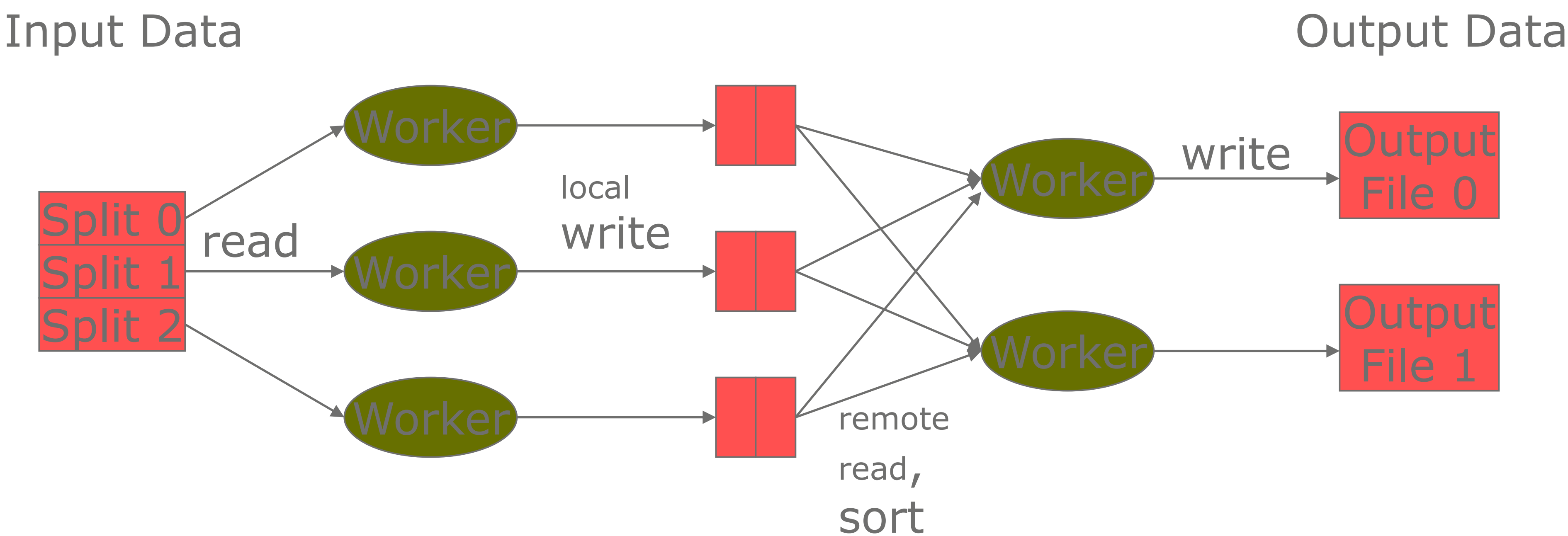
MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the pro-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is in-

MapReduce workflow



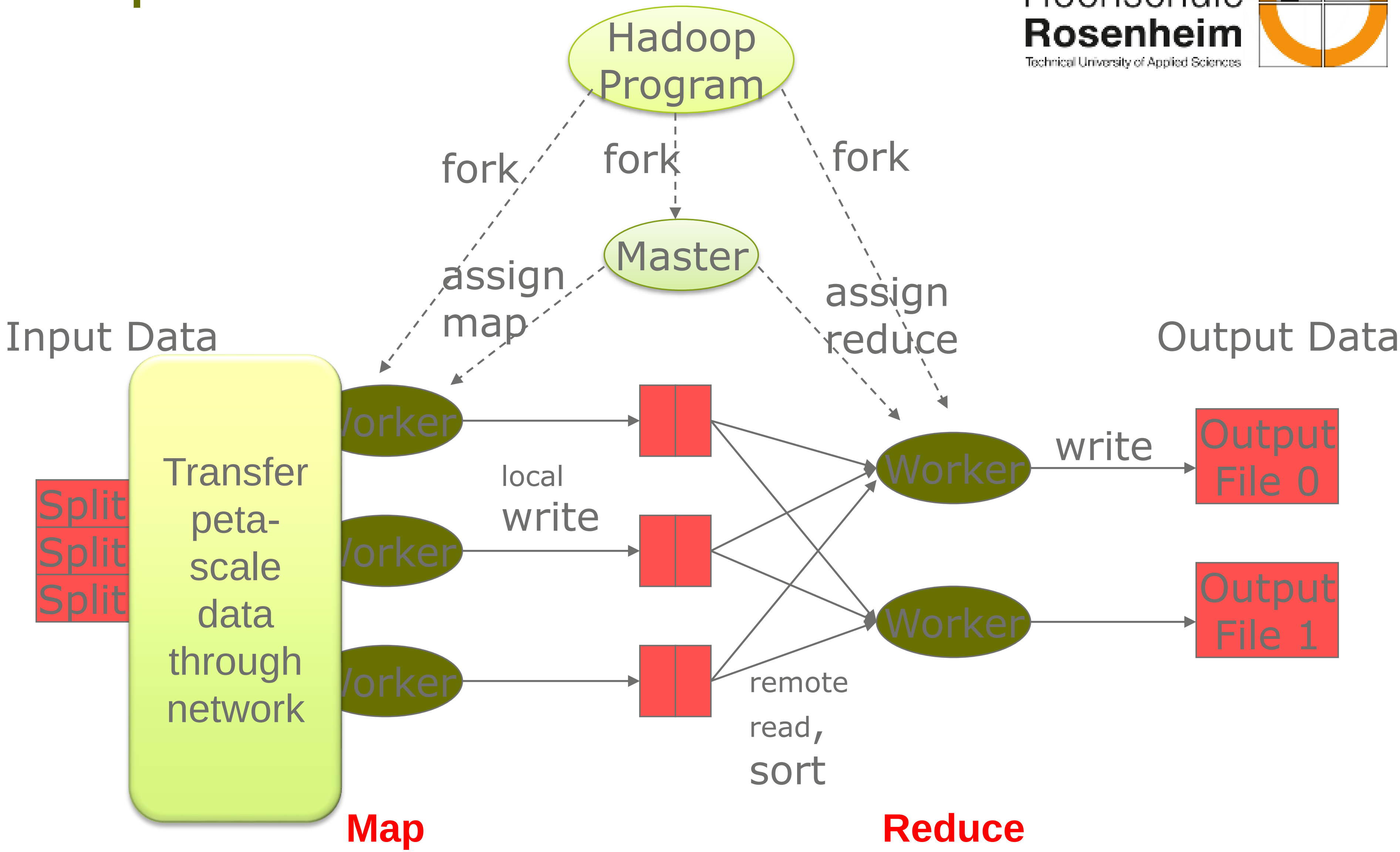
Map
extract something
you care about from
each record

Reduce
aggregate,
summarize,
filter, or
transform

Example: Word Count

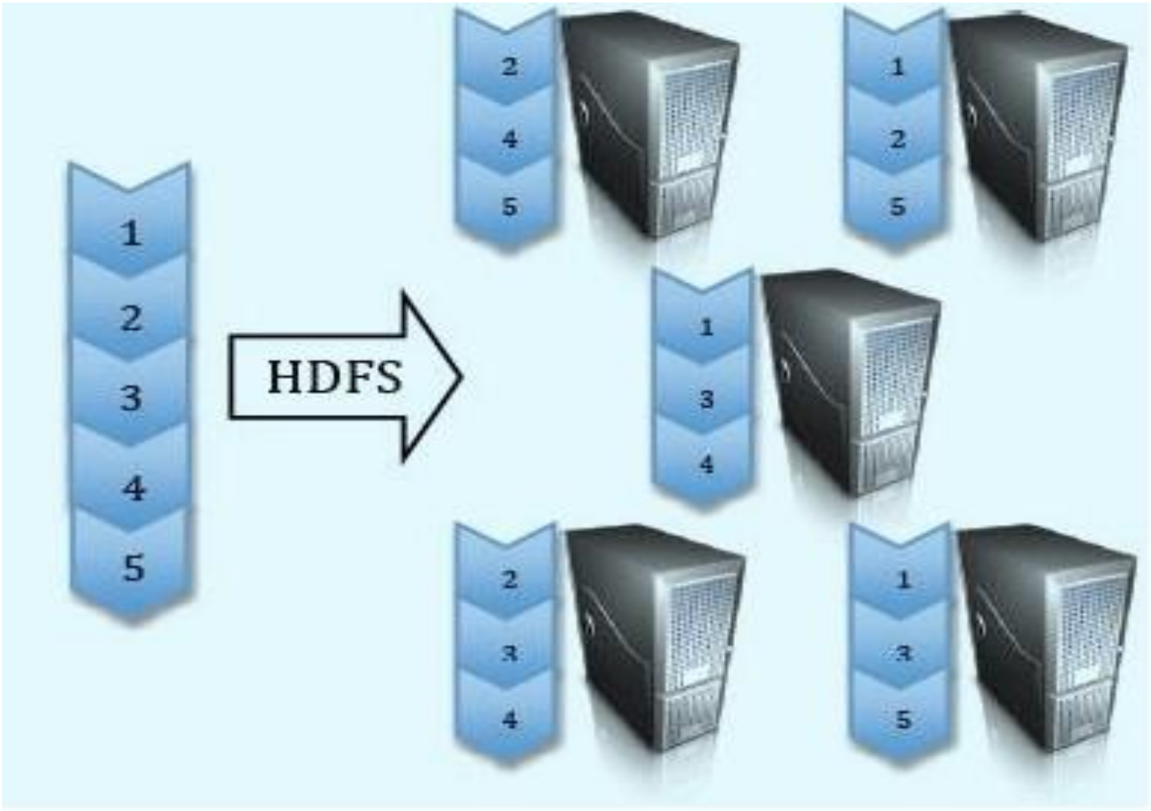
http://kickstarthadoop.blogspot.ca/2011/04/word-count-hadoop-map-reduce-example.html

MapReduce


Hadoop Program
fork
fork
fork
Master
assign map
assign reduce
Input Data
Output Data
Split
Split
Split
Transfer peta-scale data through network
local write
Worker
Worker
write
Output File 0
Output File 1
remote read, sort
Map
Reduce

Hadoop Distributed File System (HDFS)

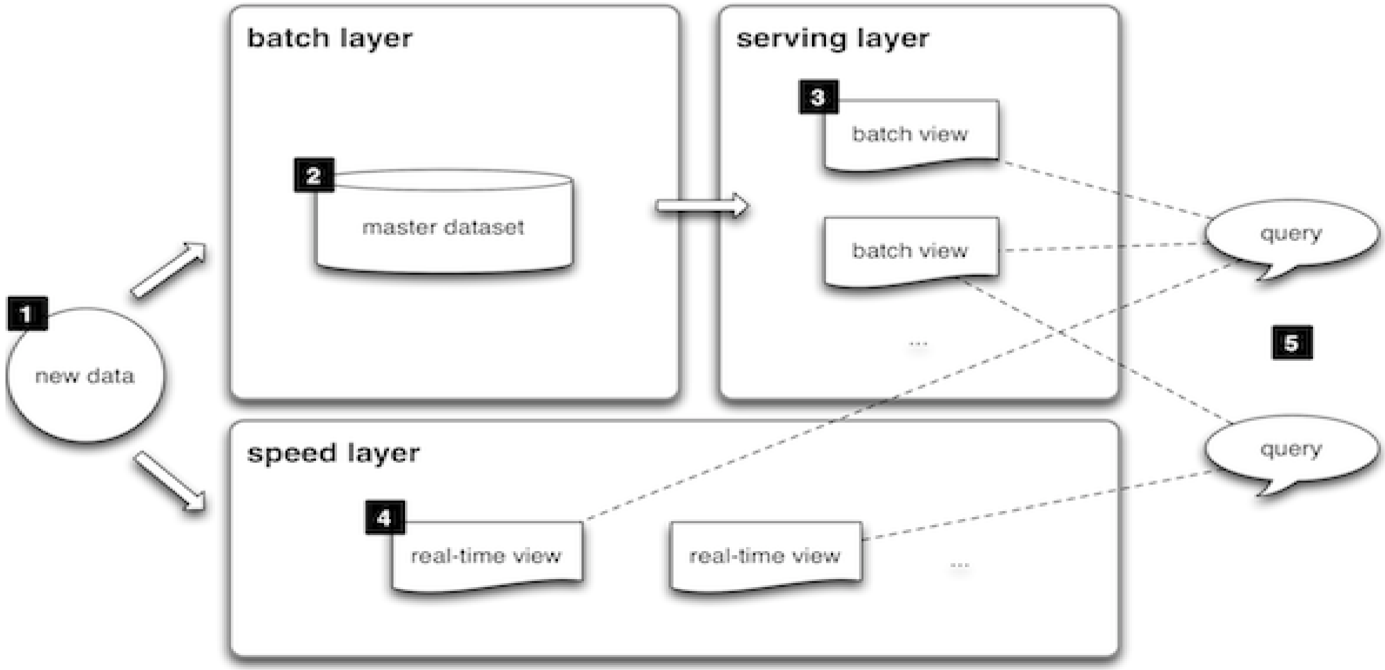
- Split data and store 3 replica on commodity servers



λ Lambda Architecture

- Created by Nathan Marz (worked at BackType, Twitter, Developed Storm)
- Architecture for generic, scalable and fault-tolerant data processing
- Robust system that is fault-tolerant against hardware failures and human mistakes
- Addresses the problem of timely insights

Lambda Architecture



- All **data** entering the system is dispatched to both the batch layer and the speed layer for processing.
- The **batch layer** has two functions: (i) managing the master dataset (an immutable, append-only set of raw data), and (ii) to pre-compute the batch views.
- The **serving layer** indexes the batch views so that they can be queried in low-latency, ad-hoc way.
- The **speed layer** compensates for the high latency of updates to the serving layer and deals with recent data only.
- Any incoming **query** can be answered by merging results from batch views and real-time views.