Modul
- Internet of Things (IoT) -

05/06 – Vorlesung *Big Data

Prof. Dr. Marcel Tilly

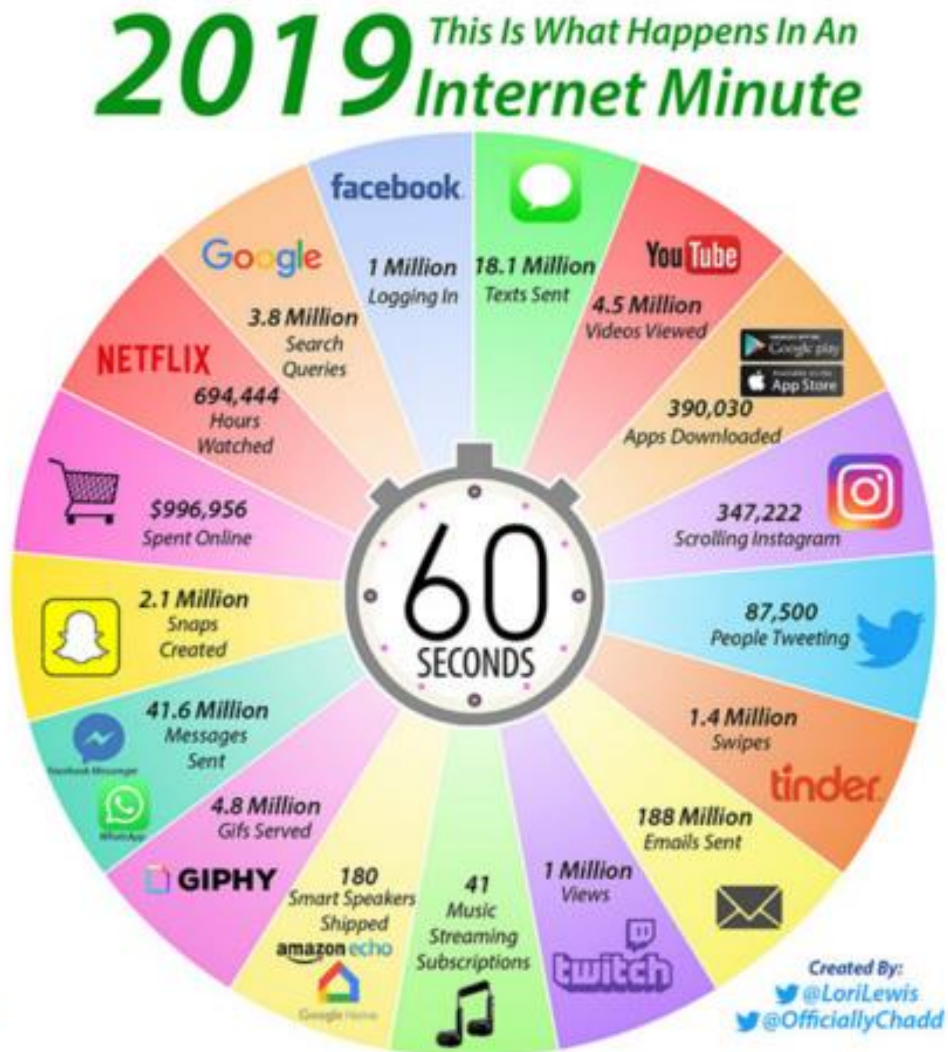
Fakultät Informatik, Cloud Computing

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

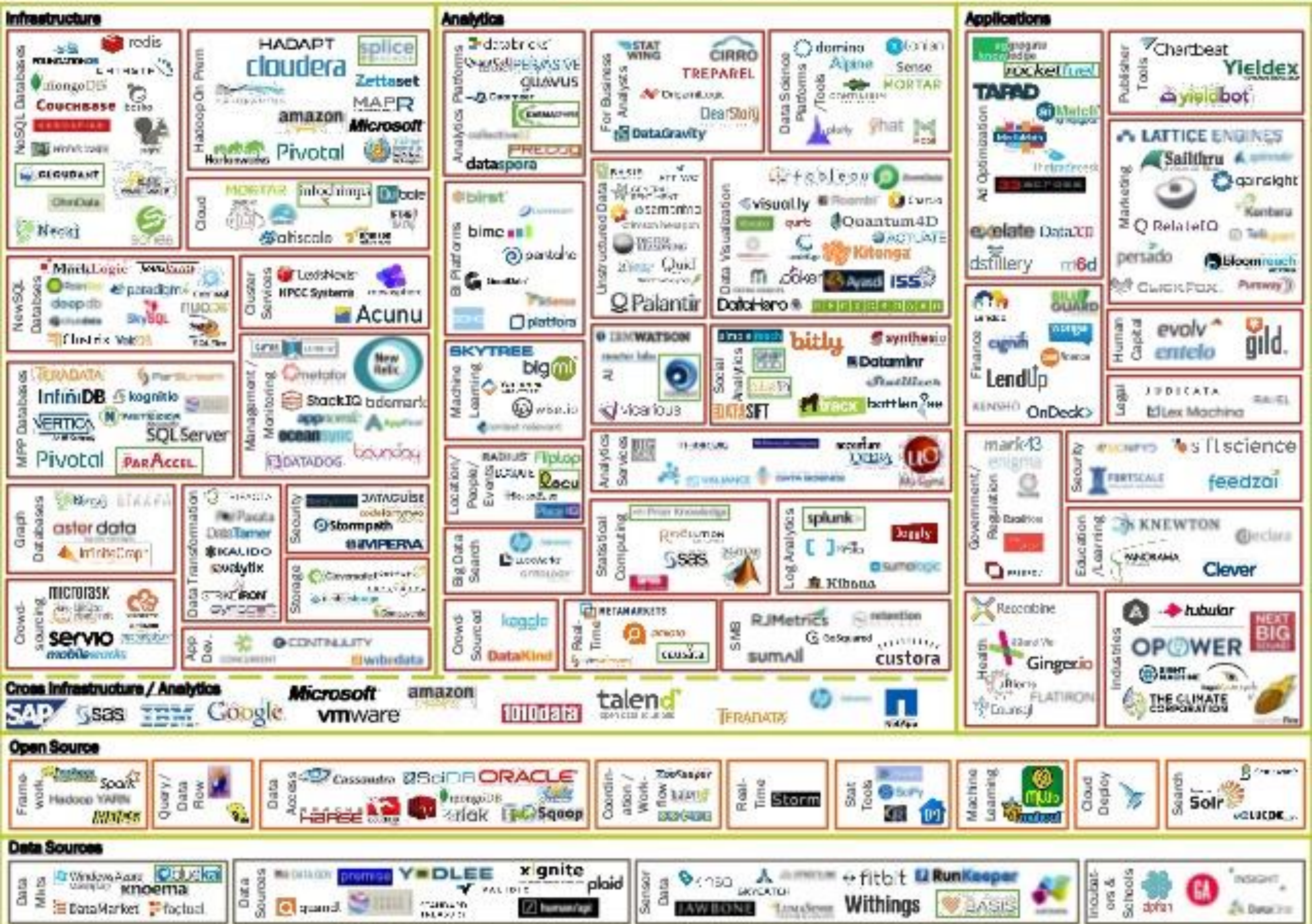
PStA

Technische Hochschule Rosenheim
Technical University of Applied Sciences



BIG DATA LANDSCAPE, VERSION 3.0

Exited: Acquisition or IPO



© Matt Turck (@mattturck), Sutian Dong (@sutiandong) & FirstMark Capital (@firstmarkcap)

Data Analysis & Platforms
Hadoop, ParAccel, Storm, HPCC Systems, MapReduce, GridGain, Dremel, Hortonworks, Apache Drill, calpont, Oracle Timesten, HD, Zettaset

Databases / Data warehousing
Infobright, Cassandra, Apache HBase, Hibari, riak, Infinispan, Bigdata®, OrientDB, Neo4j, Hypertable, Terrastore, Hive, redis, Globals

Operational
Versant JPA, MarkLogic, mobject

Multivalue database
Rocket U2, Revelation Software, northgate, OpenQM, jBASE International

Big Data search
Lucene, Apache Solr, elasticsearch.

Data aggregation
Sqoop, Flume, Chukwa

Business Intelligence
talend, Jaspersoft, SpagoBI, Jedox, pentaho, Palo, BIRT Exchange by Actuate, KNIME

Data Mining
Rapid Miner, Rapid Analytics, mahout, orange, WEKA, KEEL, togaware, jHepWork, SPMF

Social
Apache Kafka, ThinkUp, Corona

Graphs
Gephi, InfiniteGraph, FlockDB, AllegroGraph® 4.9, GraphBuilder, Gremlin, InfoGrid, HyperGraphDB, *dex, meronymy, GraphBase, BrightstarDB

Multidimensional
GT.M, SciDB, rasdaman

Grid Solutions
GigaSpaces, Hazelcast, Galaxy

KeyValue
Aerospike, leveldb, GenieDB, Chordless Beta, Tokyo Cabinet, Scalaris, Scalien, Project Voldemort – A distributed database., hamsterdb, RaptorDB, FairCom, STSDB, HyperDex, IQLECT, OpenLDAP, ioremap.net – Storage and beyond

Document Store
mongoDB, Couchbase, CouchDB, Raven DB, Clusterpoint, RaptorDB, EJDB, djondb, JasDB, SchemafreeDB, sisodb, densodb

Object databases
db4objects, Starcounter, Zope, NEOPPOD, Picolisp, Magma, Sterling, siaqodb, Morantex Information Systems, HSS Database™, EyeDB, Ramer D, NDatabase – C# Lightweight Object Database

Multimodel
ArangoDB, alchemydatabase

XML Databses
eXistdb, BaseX, Qizx, sedna, Xindice

Motivation

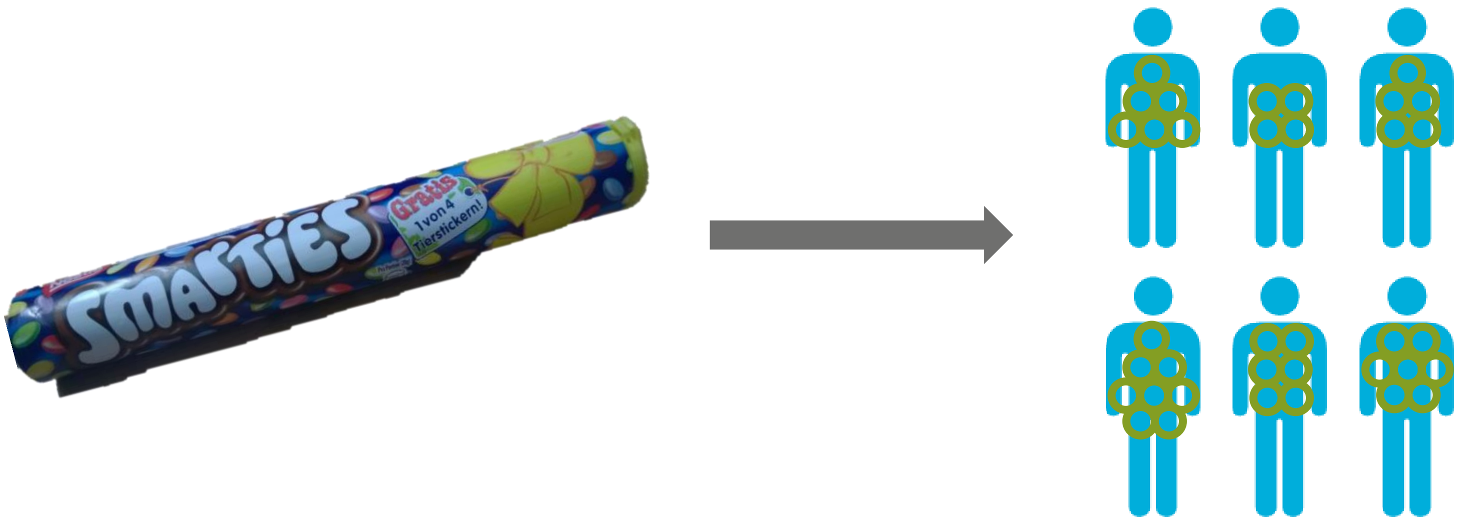
- Process lots of data
  - Google processed > 24 petabytes of data per day
- A single machine **cannot** serve all the data
  - You need a distributed system to store and process in parallel
- Parallel programming?
  - Threading is hard!
  - How do you facilitate communication between nodes?
  - How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

MapReduce

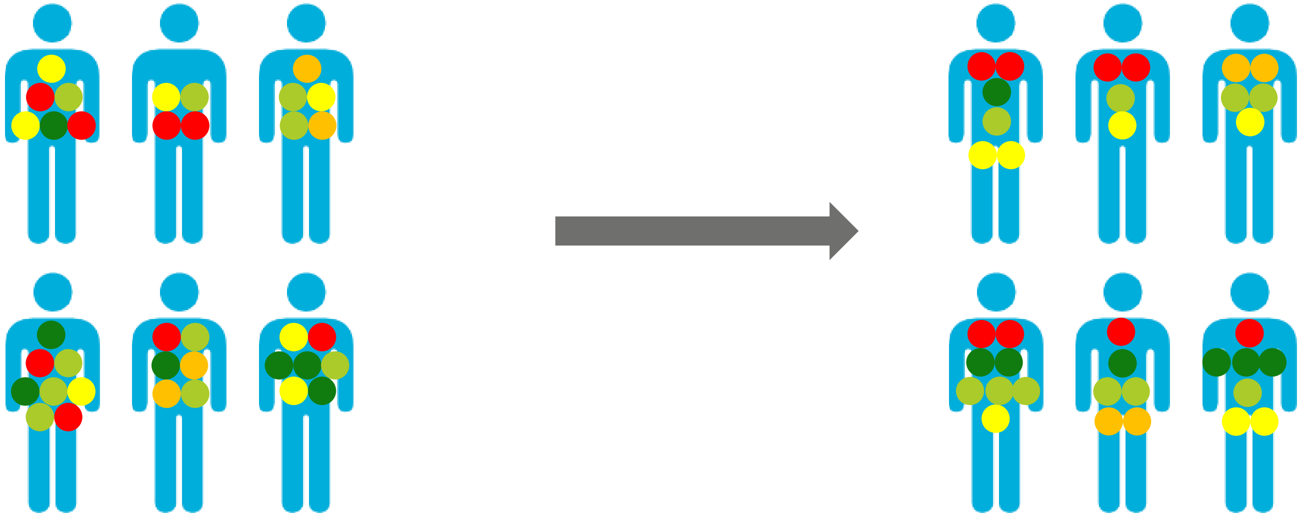
- MapReduce provides
  - Automatic parallelization, distribution
  - I/O scheduling
    - Load balancing
    - Network and data transfer optimization
  - Fault tolerance
    - Handling of machine failures

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

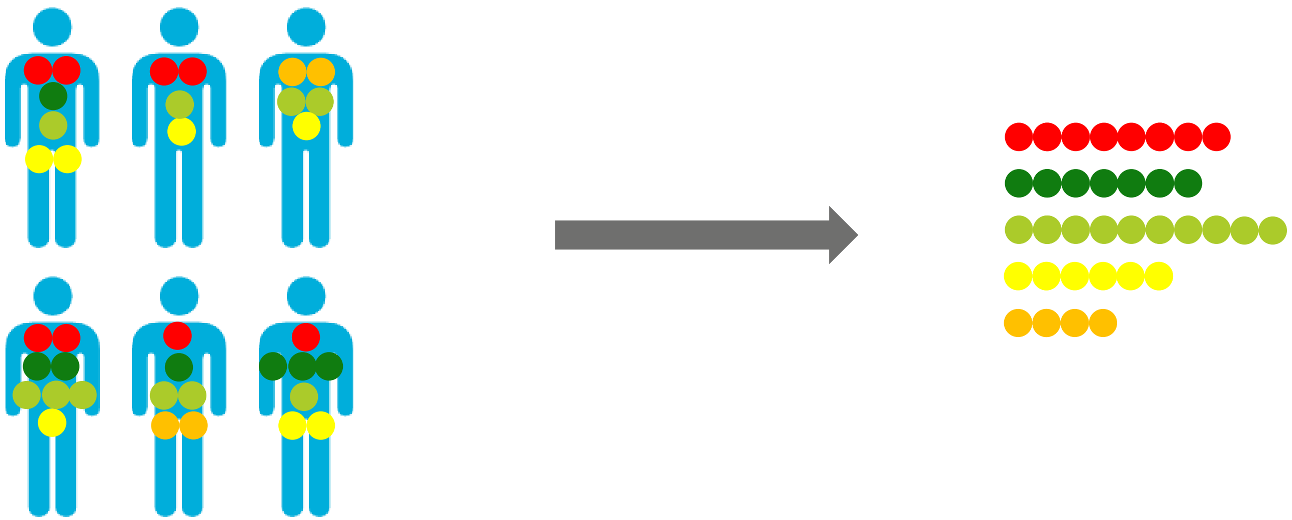
Introduction into MapReduce



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 did it start?

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

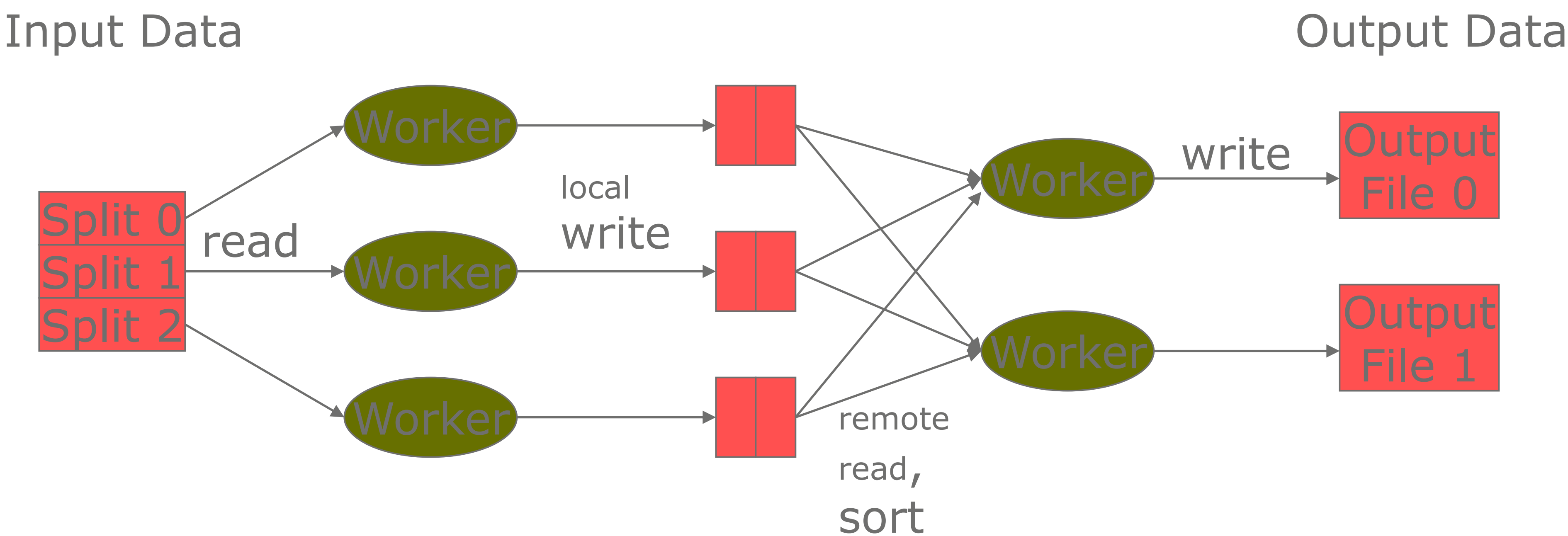
MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with 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-

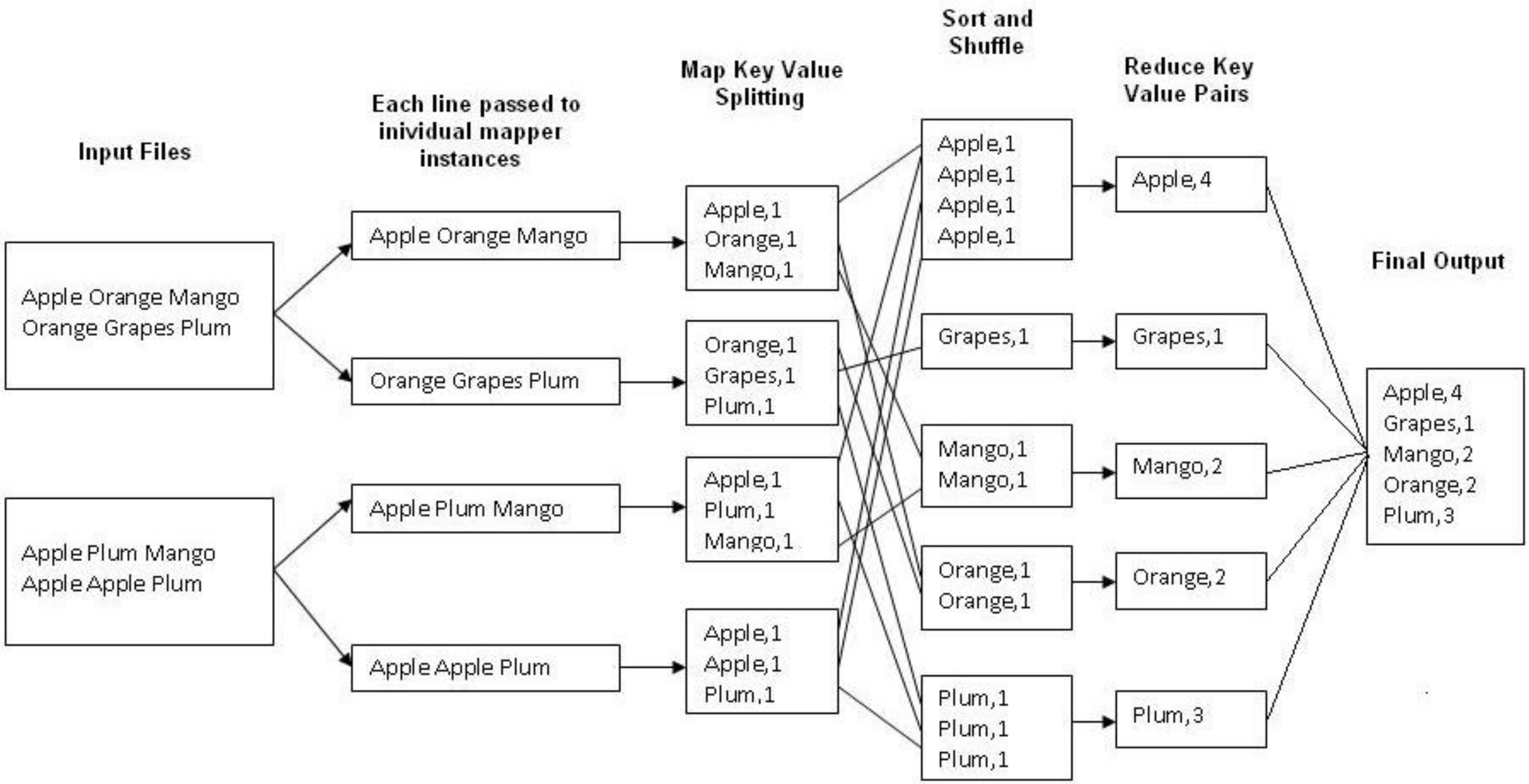
MapReduce workflow



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

Mapper

- Reads in input pair <Key,Value>

- Outputs a pair <K', V'>
  - Let's count number of each word in user queries (or Tweets/Blogs)
  - The input to the mapper will be <queryID, QueryText>:
    <Q1,"The teacher went to the store. The store was closed; the store opens in the morning. The store opens at 9am." >
  - The output would be:
    <The, 1> <teacher, 1> <went, 1> <to, 1> <the, 1> <store,1> <the, 1> <store, 1> <was, 1> <closed, 1> <the, 1> <store,1> <opens, 1> <in, 1> <the, 1> <morning, 1> <the 1> <store, 1> <opens, 1> <at, 1> <9am, 1>

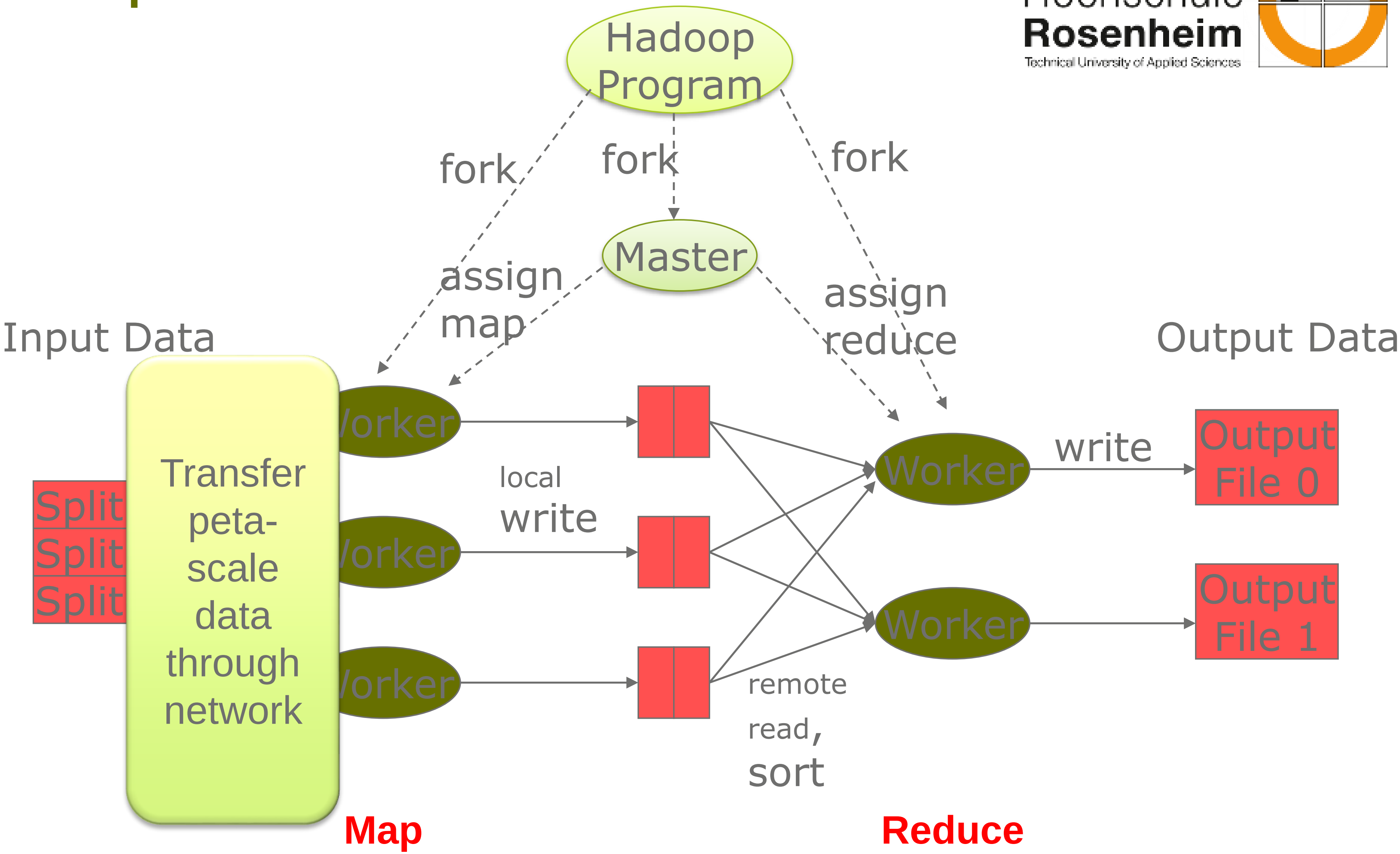
Reducer

- Accepts the Mapper output, and aggregates values on the key
  - For our example, the reducer input would be:

    <The, 1> <teacher, 1> <went, 1> <to, 1> <the, 1> <**store**, 1> <the, 1> <store, 1> <was, 1> <closed, 1> <the, 1> <**store**, 1> <opens,1> <in, 1> <the, 1> <morning, 1> <the 1> <**store**, 1> <opens, 1> <at, 1> <9am, 1>
  - The output would be:

    <The, 6> <teacher, 1> <went, 1> <to, 1> **<store, 3>** <was, 1> <closed, 1> <opens, 1> <morning, 1> <at, 1> <9am, 1>

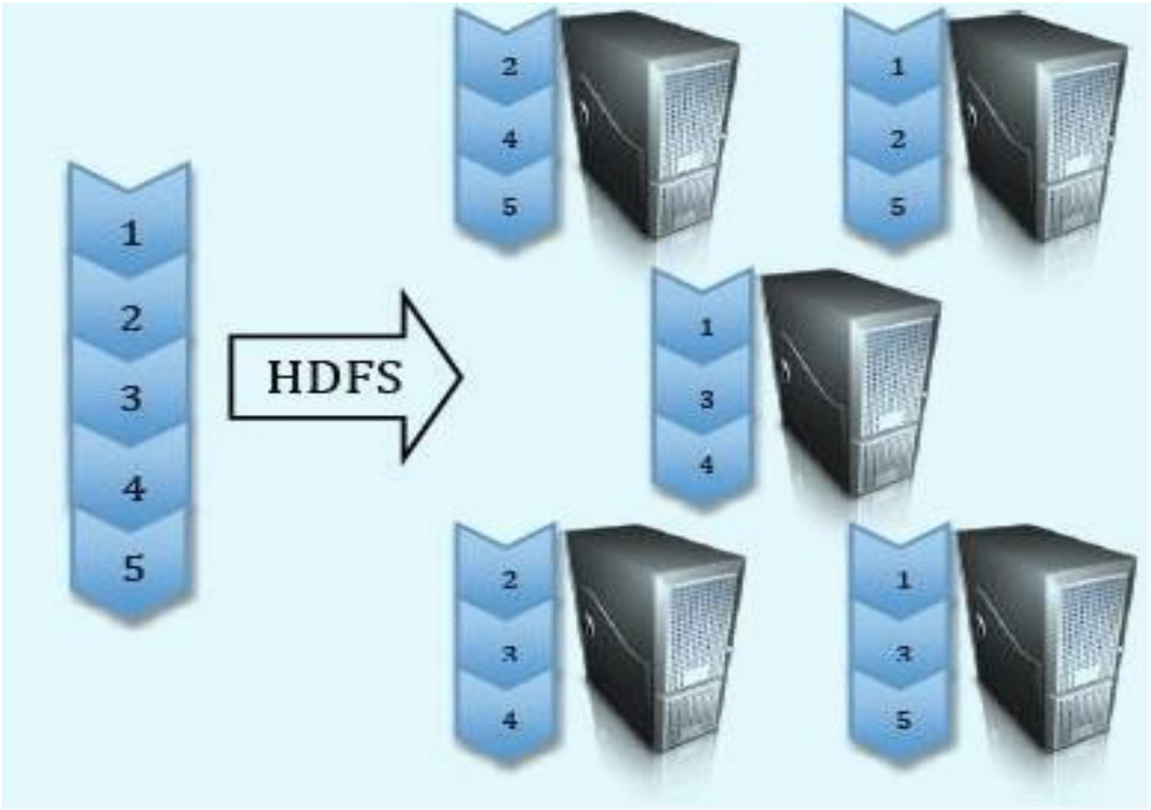
MapReduce



Hadoop Program
fork
fork
fork
Master
assign map
assign reduce
Input Data
Output Data
Split
Split
Split
Transfer peta-scale data through network
Worker
Worker
Worker
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

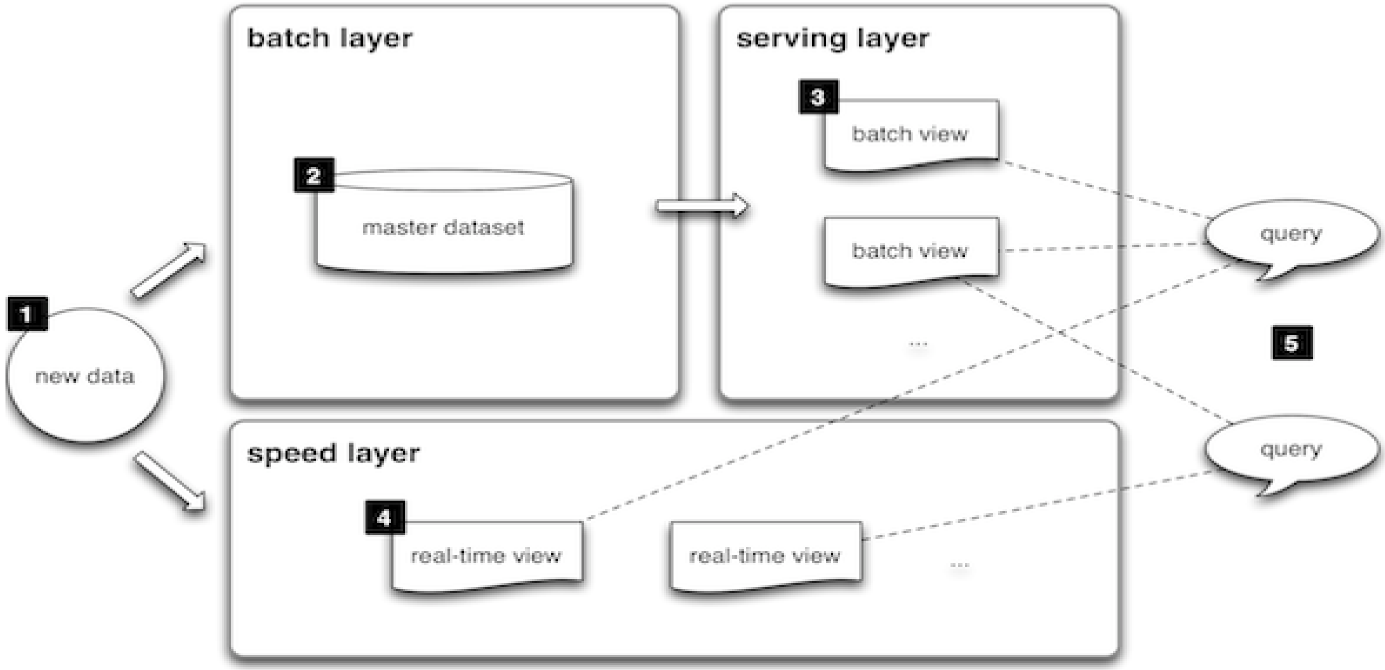


λ Lambda Architecture

- 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

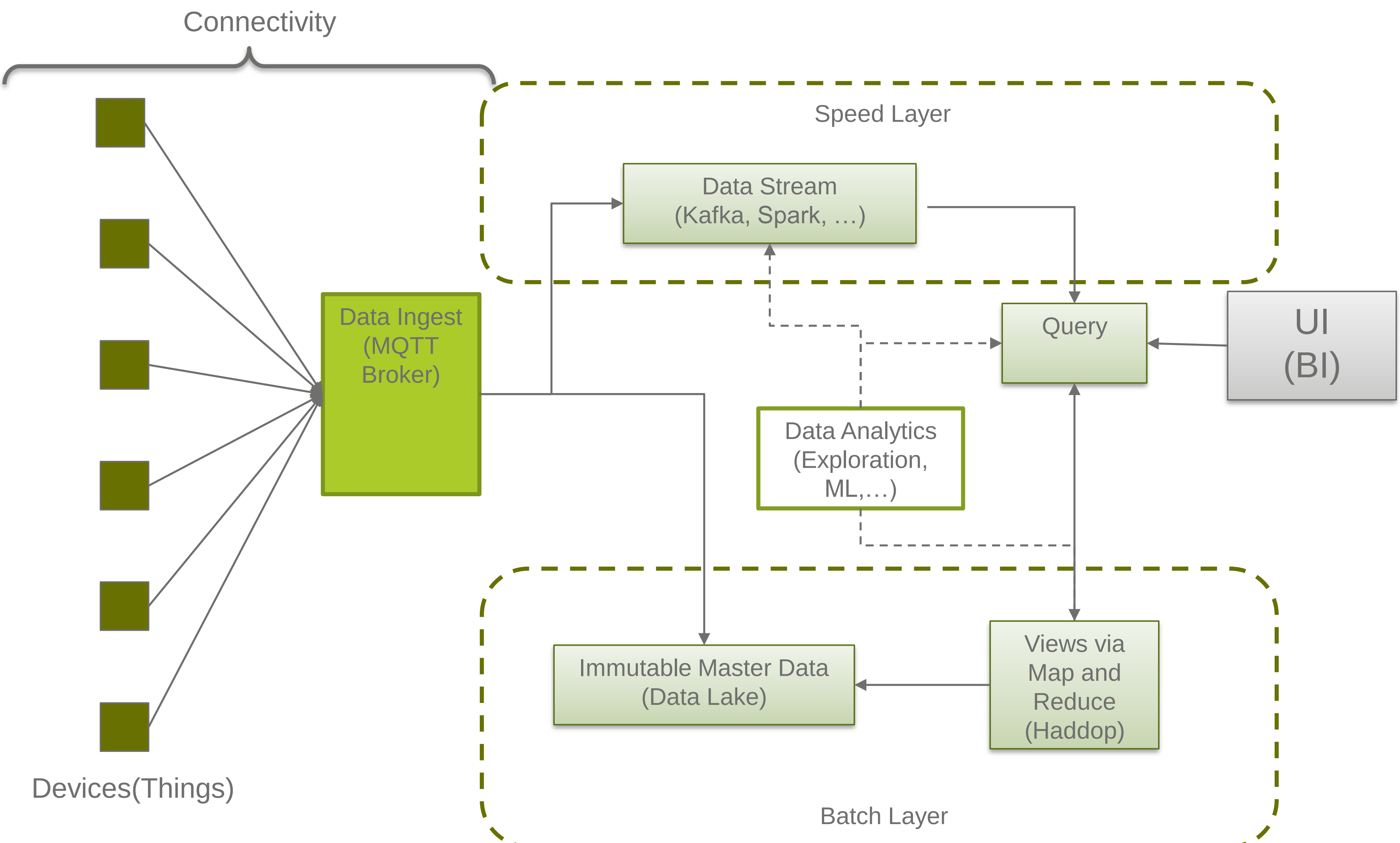
Lambda Architecture

Technische Hochschule Rosenheim
Technical University of Applied Sciences



- 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.

IoT Architecture



Data Analysis with Python

Python Libraries for Data Science

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

Visualization libraries

- matplotlib
- Seaborn

and many more …

NumPy

NumPy **Link:** http://www.numpy.org/

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects

- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance

- many other python libraries are built on NumPy

SciPy

SciPy: **Link:** https://www.scipy.org/scipylib/

- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack
- built on NumPy

Pandas

Pandas: **Link:** http://pandas.pydata.org/

- adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- allows handling missing data

SciKit-Learn

Technische Hochschule Rosenheim
Technical University of Applied Sciences

SciKit-Learn: **Link:** http://scikit-learn.org/

- provides machine learning algorithms: classification, regression, clustering, model validation etc.

- built on NumPy, SciPy and matplotlib

matplotlib

matplotlib: **Link:** https://matplotlib.org/

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

Python Libraries for Data Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Seaborn: **Link:** https://seaborn.pydata.org/

- based on matplotlib
- provides high level interface for drawing attractive statistical graphics
- Similar (in style) to the popular ggplot2 library in R

Data Frames attributes

Python objects have *attributes* and *methods*.

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Data Frames methods

Unlike attributes, python methods have *parenthesis.*
All attributes and methods can be listed with a *dir()* function: **`dir(df)`**

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Missing Values

There are a number of methods to deal with missing values in the data frame:

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

Missing Values

- When summing the data, missing values will be treated as zero
- If all values are missing, the sum will be equal to NaN
- cumsum() and cumprod() methods ignore missing values but preserve them in the resulting arrays
- Missing values in GroupBy method are excluded (just like in R)
- Many descriptive statistics methods have *skipna* option to control if missing data should be excluded . This value is set to *True* by default (unlike R)

Aggregation Functions in Pandas

Aggregation - computing a summary statistic about each group, i.e.

- compute group sums or means
- compute group sizes/counts

Common aggregation functions:

min, max
count, sum, prod
mean, median, mode, mad
std, var

Basic Descriptive Statistics

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation
sem	Standard error of mean
skew	Sample skewness
kurt	kurtosis

Graphics

	description
distplot	histogram
barplot	estimate of central tendency for a numeric variable
violinplot	similar to boxplot, also shows the probability density of the data
jointplot	Scatterplot
regplot	Regression plot
pairplot	Pairplot
boxplot	boxplot
swarmplot	categorical scatterplot
factorplot	General categorical plot

Basic statistical Analysis

statsmodel and scikit-learn - both have a number of function for statistical analysis
The first one is mostly used for regular analysis using R style formulas, while   scikit-learn is more tailored for Machine Learning.

statsmodels:

- linear regressions
- ANOVA tests
- hypothesis testings
- many more ...

scikit-learn:

- kmeans
- support vector machines
- random forests
- many more ...