Data Science 2

Vorlesung - Big Data

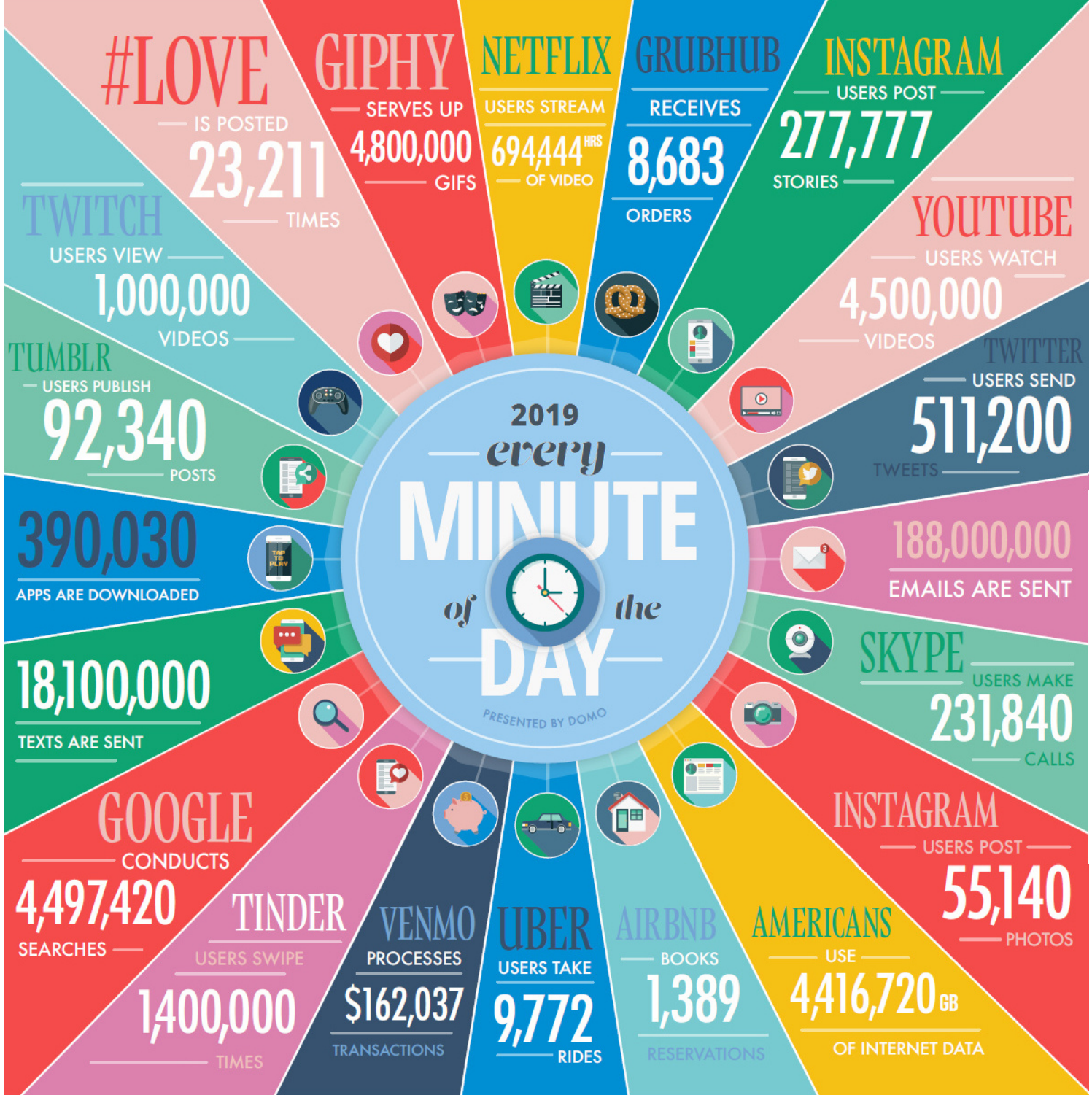
Prof. Dr. Christian Bockermann

Hochschule Bochum

Wintersemester 2023 / 2024

Inhalt

Was ist Big Data?


#LOVE
IS POSTED
23,211
TIMES
GIPHY
SERVES UP
4,800,000
GIFS
NETFLIX
USERS STREAM
694,444 HRS
OF VIDEO
GRUBHUB
RECEIVES
8,683
ORDERS
INSTAGRAM
USERS POST
277,777
STORIES
TWITCH
USERS VIEW
1,000,000
VIDEOS
YOUTUBE
USERS WATCH
4,500,000
VIDEOS
TUMBLR
USERS PUBLISH
92,340
POSTS
TWITTER
USERS SEND
511,200
TWEETS
2019
every
MINUTE
of the
DAY
PRESENTED BY DOMO
390,030
APPS ARE DOWNLOADED
188,000,000
EMAILS ARE SENT
18,100,000
TEXTS ARE SENT
SKYPE
USERS MAKE
231,840
CALLS
GOOGLE
CONDUCTS
4,497,420
SEARCHES
INSTAGRAM
USERS POST
55,140
PHOTOS
TINDER
USERS SWIPE
1,400,000
TIMES
VENMO
PROCESSES
$162,037
TRANSACTIONS
UBER
USERS TAKE
9,772
RIDES
AIRBNB
BOOKS
1,389
RESERVATIONS
AMERICANS
USE
4,416,720 GB
OF INTERNET DATA

SO WHAT IS A PETABYTE ANYWAY?

Source – www.mozy.com

WHAT IS A PETABYTE?

TO UNDERSTAND A PETABYTE WE MUST FIRST UNDERSTAND A GIGABYTE.

1 GIGABYTE	7 MINUTES OF HD-TV VIDEO
2 GIGABYTES	20 YARDS OF BOOKS ON A SHELF
4.7 GIGABYTES	SIZE OF A STANDARD DVD-R

THERE ARE A MILLION GIGABYTES IN A PETABYTE

"Let me repeat that: we create as much information in two days now as we did from the dawn of man through 2003." (That's something like 5 Exabytes of Data). - Eric Schmidt – Google 8/10

A PETABYTE IS A LOT OF DATA

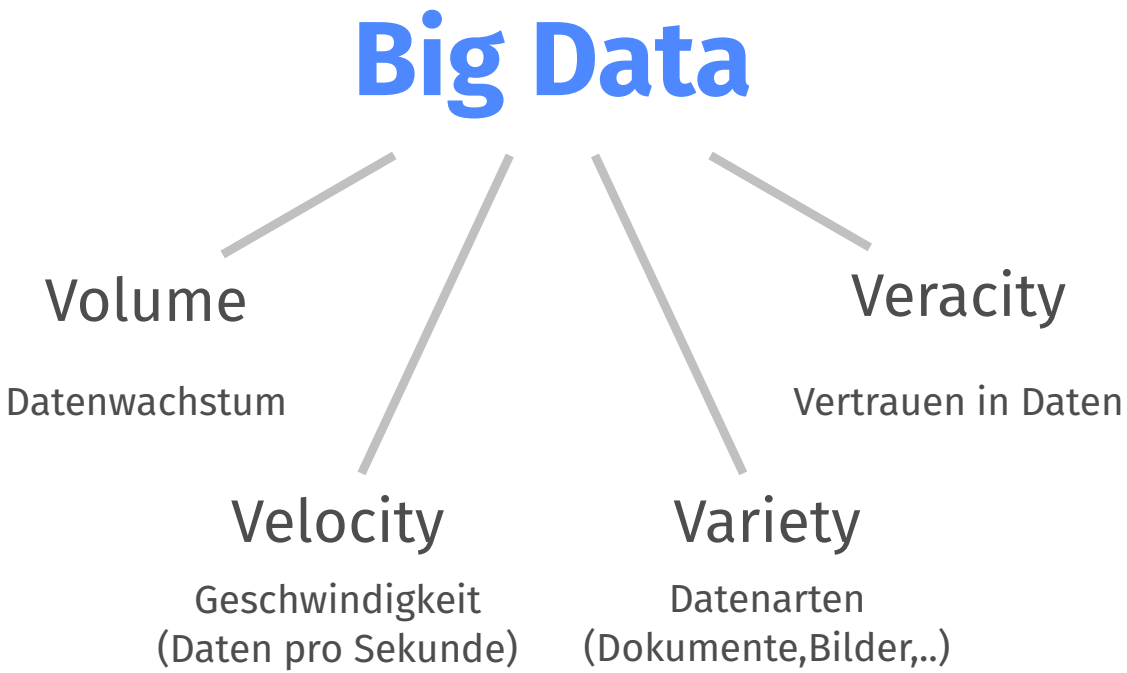
1 PETABYTE	20 MILLION FOUR-DRAWER FILING CABINETS FILLED WITH TEXT
1 PETABYTE	13.3 YEARS OF HD-TV VIDEO
1.5 PETABYTES	SIZE OF THE 10 BILLION PHOTOS ON → FACEBOOK
15+ PETABYTES	INTERNET USER'S DATA BACKED UP ON MOZY.COM
20 PETABYTES	THE AMOUNT OF DATA PROCESSED BY GOOGLE PER DAY
20 PETABYTES	TOTAL HARD DRIVE SPACE MANUFACTURED IN 1995
50 PETABYTES	THE ENTIRE WRITTEN WORKS OF MANKIND, FROM THE BEGINNING OF RECORDED HISTORY, IN ALL LANGUAGES

Twitter:
Over 7TB a Day in Tweets.

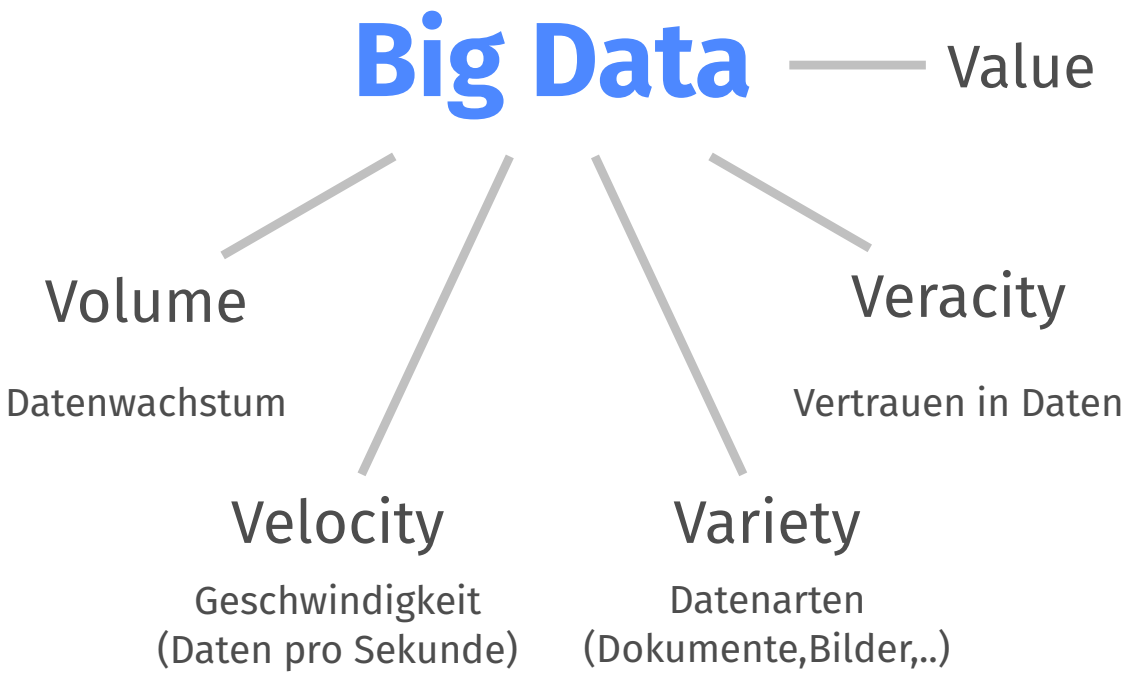
A ZETABYTE IS ONE MILLION PETABYTES!

Facebook:
More that 750 Million Users.
Average user creates 90 Pieces of content each month.
More than 30B pieces of content shared each month.

Die berühmten 4 V's



Die berühmten 5 V's



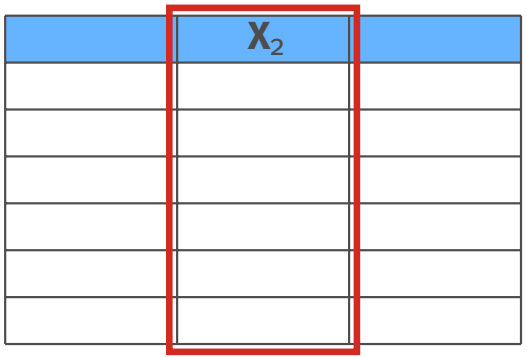
Big Data Techniken (Volume)

Beispiel: Durchschnitt berechnen

	$\mathbf{X}_2$	

$$\overline{\mathbf{X}_2} = \frac{\sum_{i=1}^{n} x_{i,2}}{n}$$

Beispiel: Durchschnitt berechnen



$$\overline{x_2} = \frac{\sum_{i=1}^{n} x_{i,2}}{n}$$

```
# mit Pandas DataFrame:
df['X2'].sum() / len(df)

# oder:
df['X2'].mean()
```

Beispiel: Durchschnitt berechnen

- Pandas liest DataFrame in Hauptspeicher
- Schnelle Berechnung, schneller Zugriff

Datensätze im DataScience Kurs

- Iris Daten - 3.8 KB
- Auto MPG Daten - 18 KB
- Hausarbeit - 3.7 MB und 5.8 MB
- RKI Covid19 Daten - 44 MB

Beispiel: Durchschnitt berechnen

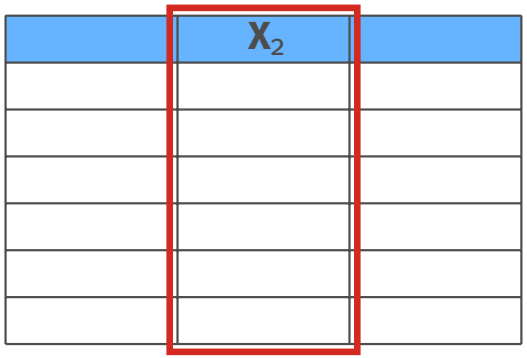
- Pandas liest DataFrame in Hauptspeicher
- Schnelle Berechnung, schneller Zugriff

Datensätze im DataScience Kurs

- Iris Daten - 3.8 KB
- Auto MPG Daten - 18 KB
- Hausarbeit - 3.7 MB und 5.8 MB
- RKI Covid19 Daten - 44 MB

Was ist, wenn unsere Daten 4 TB groß sind?

Beispiel: Durchschnitt berechnen

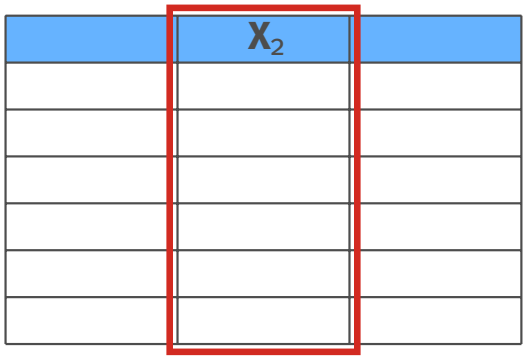


4 TB = ca. 4000 GB

handelsübliche Festplatte

- Lesegeschwindigkeit einer Festplatte 100 - 200 MB/s
- Lesen der Daten benötigt 5,5 - 11 Minuten

Beispiel: Durchschnitt berechnen

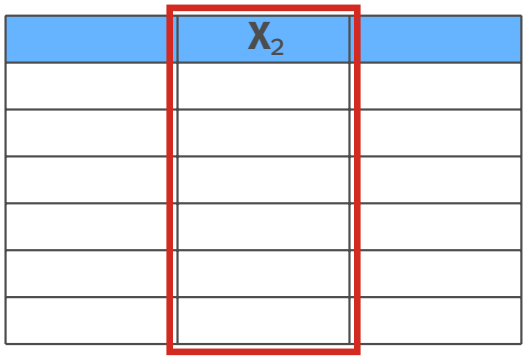


4 TB = ca. 4000 GB

handelsübliche Festplatte

- Lesegeschwindigkeit einer Festplatte 100 - 200 MB/s
- Lesen der Daten benötigt 5,5 - 11 Minuten
- SSD Platte + SATA mit 600 MB/s braucht knapp 2 Minuten

Beispiel: Durchschnitt berechnen



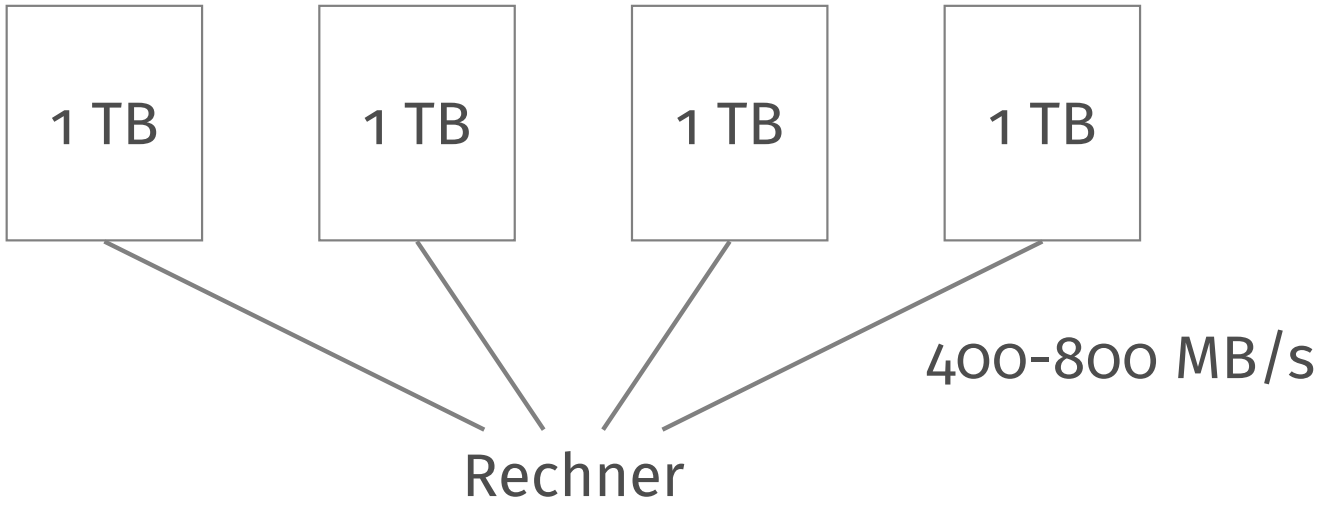
4 TB = ca. 4000 GB

handelsübliche Festplatte

- Lesegeschwindigkeit einer Festplatte 100 - 200 MB/s
- Lesen der Daten benötigt 5,5 - 11 Minuten
- SSD Platte + SATA mit 600 MB/s braucht knapp 2 Minuten
- Festplatte: 0,024 EUR/GB, SSD-Platte: 0,09 EUR/GB
- Größte SSD 8 TB, sehr teuer

Wie können wir die Berechnung beschleunigen?

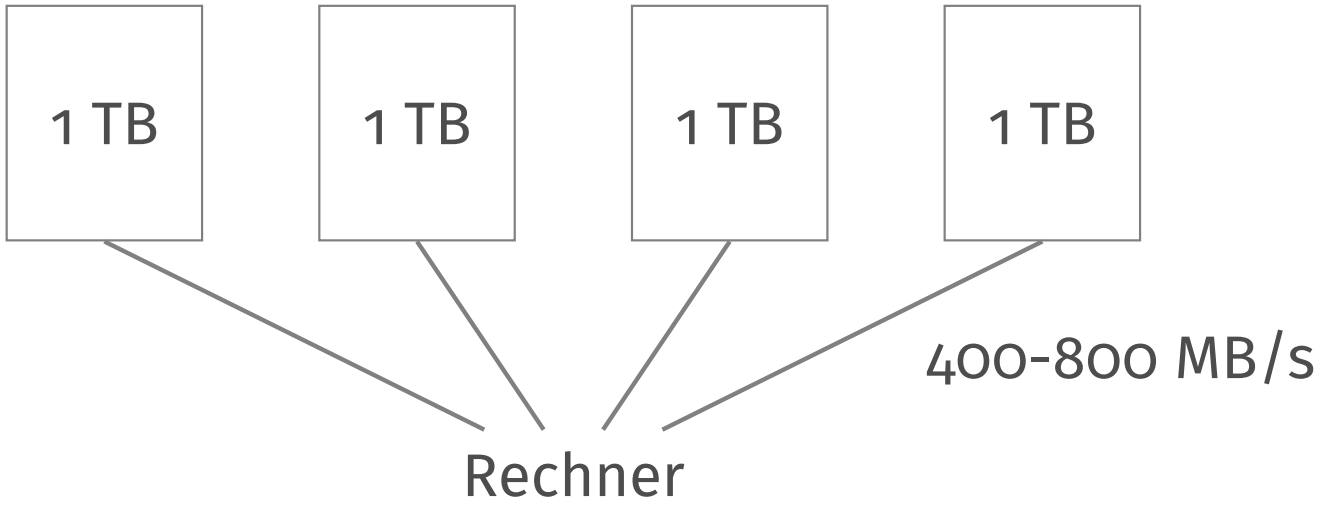
1. Durchsatz erhöhen mit mehreren Festplatten:



- Von allen Platten gleichzeitig lesen: ca. 1,5 Minuten

Wie können wir die Berechnung beschleunigen?

1. Durchsatz erhöhen mit mehreren Festplatten:



- Von allen Platten gleichzeitig lesen: ca. 1,5 Minuten
- Daten müssen dann auch **parallel** verarbeitet werden!!

2. Parallele Berechnung des Durchschnitts

Wir teilen die Daten in (z.B. 4) Blöcke auf

$$\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{k_1}\} \cup \{\mathbf{x}_{k_1+1}, \ldots, \mathbf{x}_{k_2}\}, \ldots$$

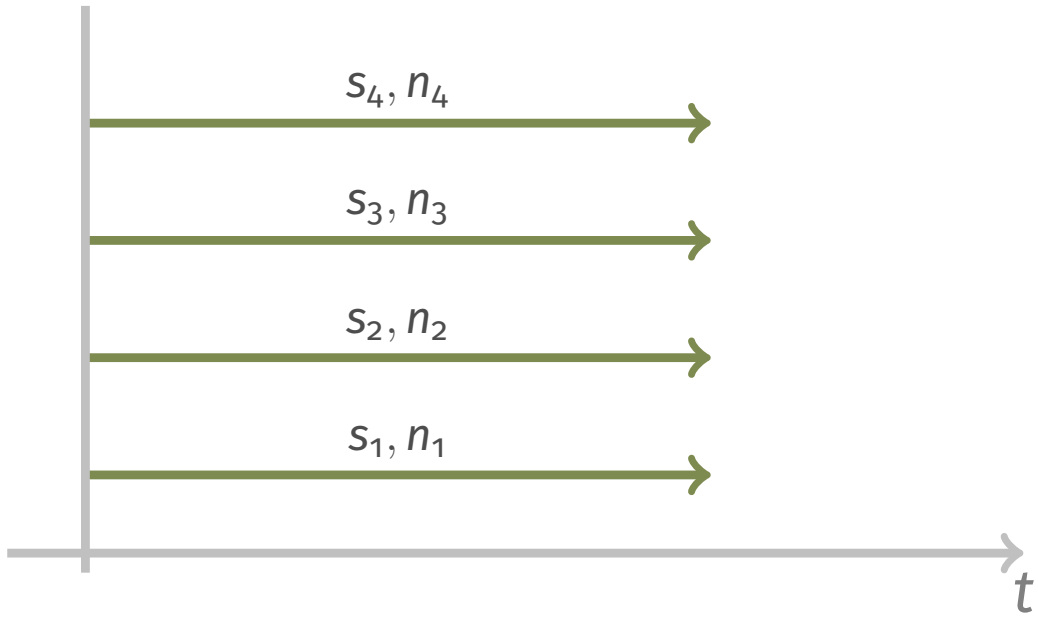
und zerlegen auch die Berechnung in Teilprobleme:

$$\overline{\mathbf{X}_2} = \frac{\sum_{i=1}^{n} x_{i,2}}{n} = \frac{s_1 + s_2 + s_3 + s_4}{n_1 + n_2 + n_3 + n_4}$$

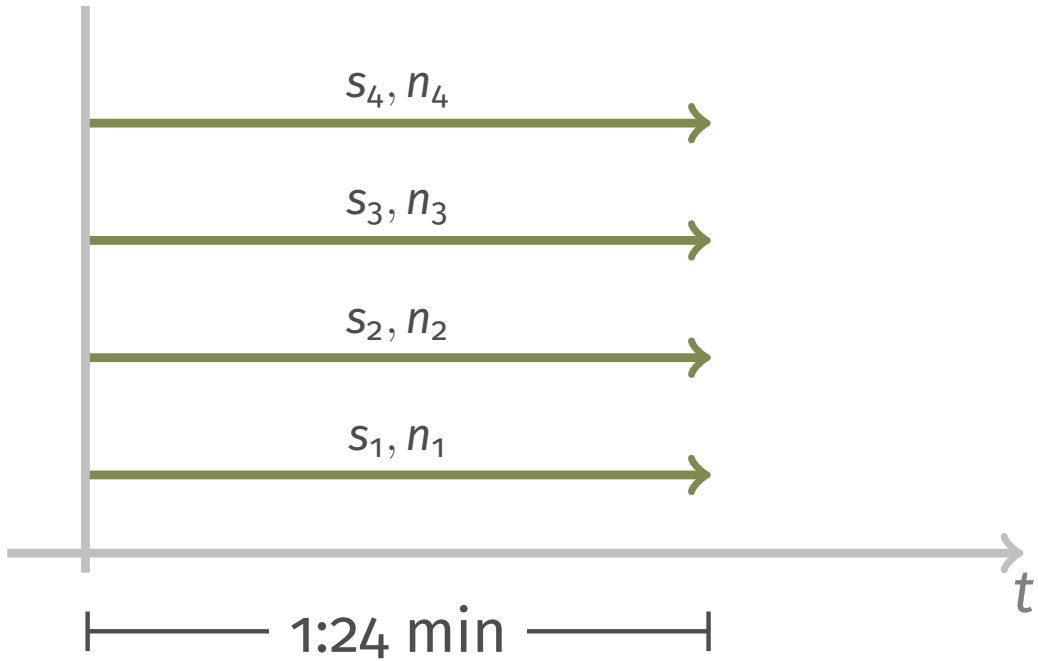
mit

$$s_1 = \sum_{i=1}^{k_1} x_{i,2},\ s_2 = \sum_{i=k_1+1}^{k_2} x_{i,2}, \ldots,\ s_4 = \sum_{i=k_3+1}^{k_4} x_{i,2}$$

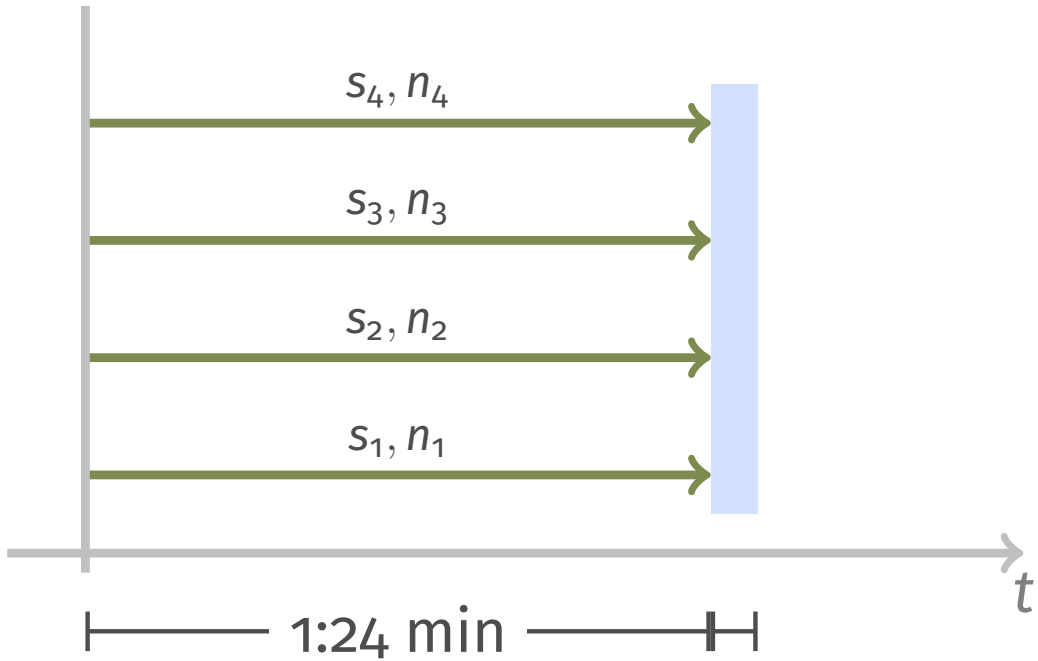
Parallele Berechnung des Durchschnitts



Parallele Berechnung des Durchschnitts



Parallele Berechnung des Durchschnitts



Die Idee von Map-Reduce

Seien die Blöcke Python-Listen:

```
bloecke = [ [1,2,3,4], [5,6,7,8], ...]

block0 = bloecke[0]   # [1,2,3,4]
```

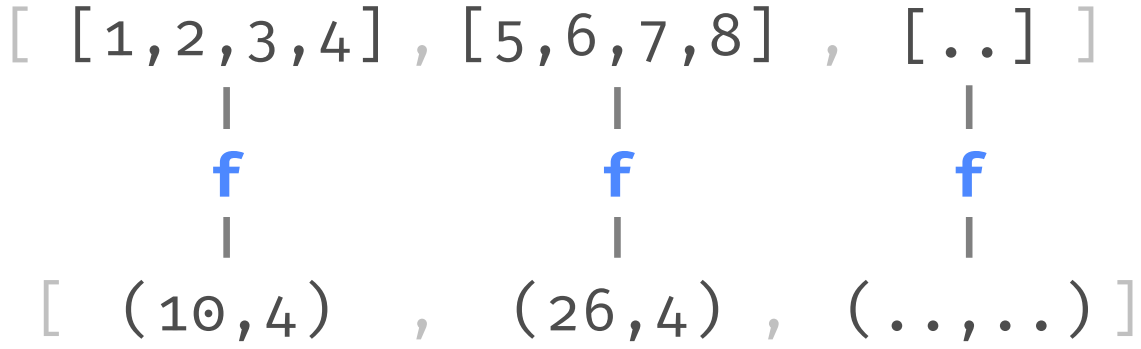
Wir definieren eine Funktion f, die für jeden Block s_i und n_i berechnet:

```
def f(block):
  s = 0
  for x in block:
    s = s + x
  return (s, len(block))
```

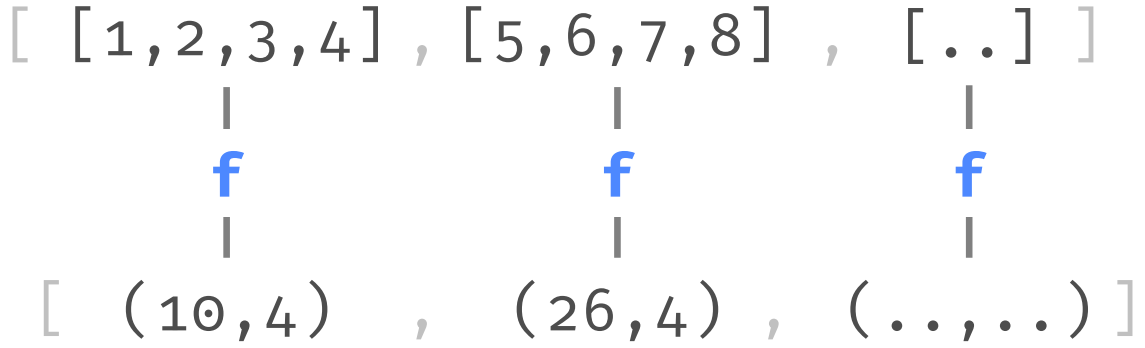
map: Funktion auf alle Elemente einer Liste anwenden

```
[ [1,2,3,4] , [5,6,7,8] ,  [..] ]
```

map: Funktion auf alle Elemente einer Liste anwenden



map: Funktion auf alle Elemente einer Liste anwenden




```
bloecke = [ [1,2,..], [5,6,..], [..], ..]

# Zwischenergebnisse:
xs = map(f, bloecke)
```

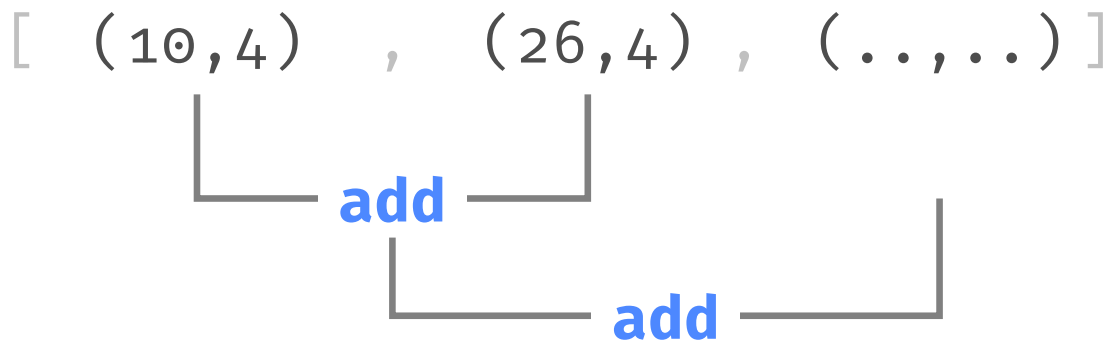
`reduce`: Liste zu Ergebnis aggregieren

Definiere eine Funktion `add`, die zwei Elemente verknüpft:

```
def add(x, y):
  return (x[0] + y[0], x[1] + y[1])
```

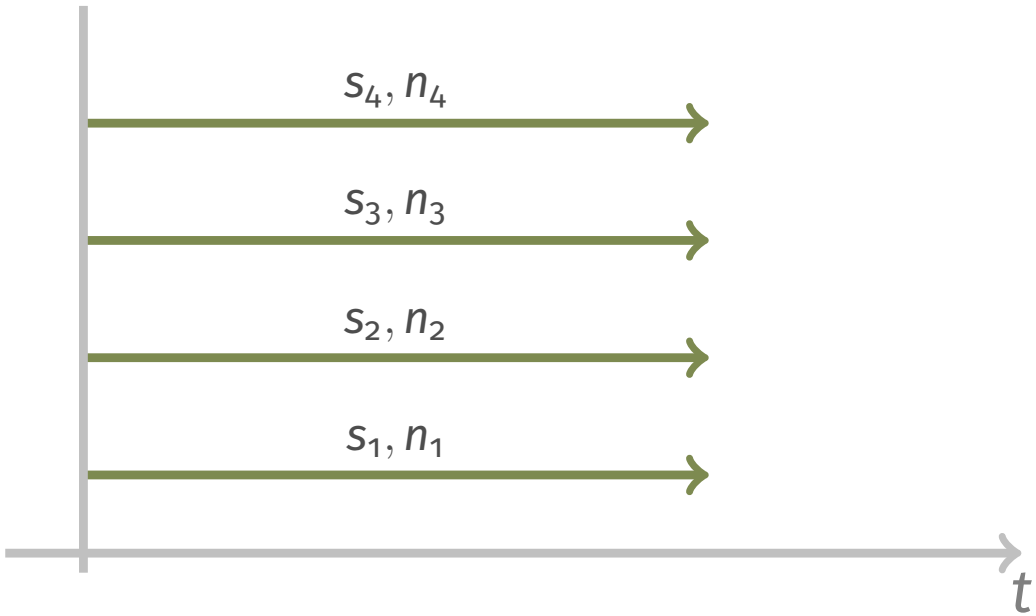
Diese Funktion benutzen wir mit `reduce` ähnlich wie `f` mit `map` zum aggregieren.

reduce: Liste zu Ergebnis aggregieren

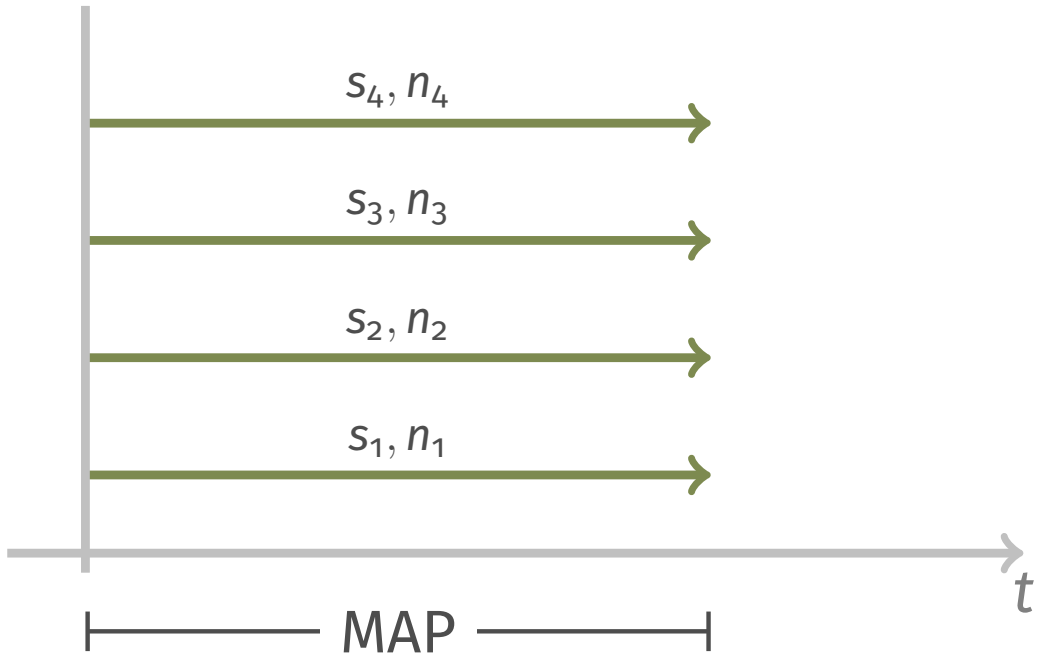



```
from functools import reduce

xs = map(f, bloecke)
(s,n) = reduce(add, xs)

avg = s / n
```

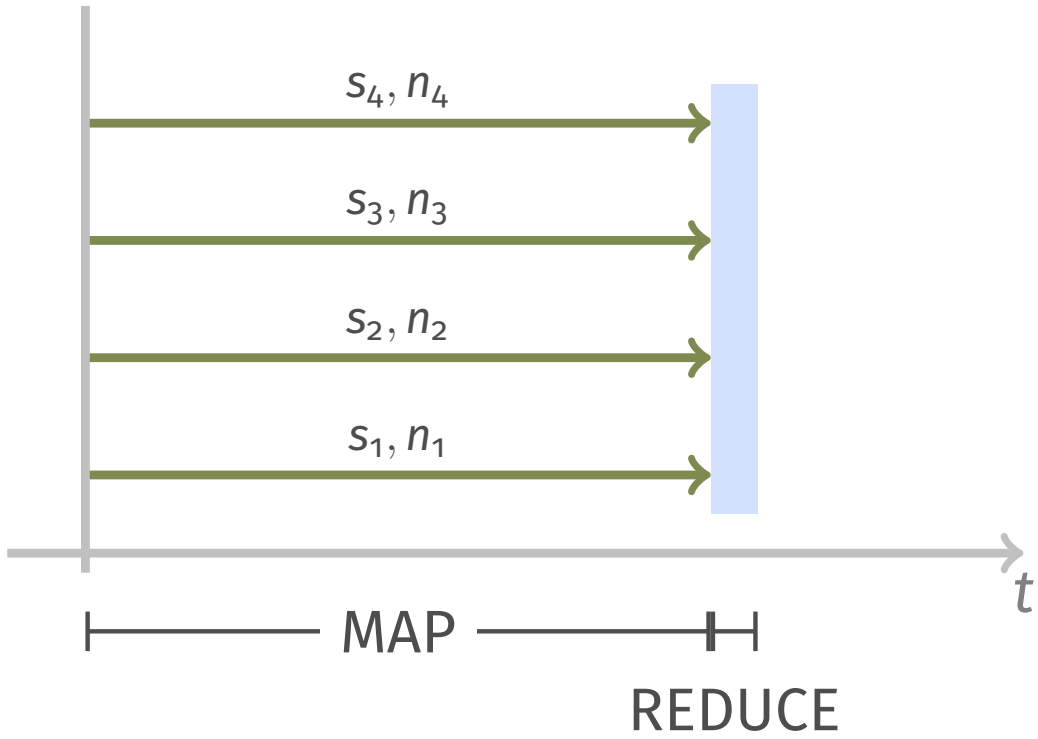
Parallelisierung mit Map-Reduce



Parallelisierung mit Map-Reduce



Parallelisierung mit Map-Reduce



Scale-Up: Mehr Power + Speicher

Server aufrüsten mit

- mehr Speicher
- zustäzlichen CPUs
- weiteren Festplatten

Scale-Up: Mehr Power + Speicher

Server aufrüsten mit

- mehr Speicher
- zustäzlichen CPUs
- weiteren Festplatten

Aufrüstung begrenzt: max. Speicher, Anzahl Festplatten,...

Scale-Up: Mehr Power + Speicher

Server aufrüsten mit

- mehr Speicher
- zustäzlichen CPUs
- weiteren Festplatten

Aufrüstung begrenzt: max. Speicher, Anzahl Festplatten,...

Zur Erinnerung: Wir suchen nach **Big** Data!

4 PB = 4000 TB = 1000 Festplatten mit 4 TB Speicher.

Scale-Out: Mehr Server

- Verteilung auf mehrere Server
- Jeder Server hat zig Platten
- zur Erweiterung werden zusätzliche Server hinzugenommen

Scale-Out: Mehr Server

- Verteilung auf mehrere Server
- Jeder Server hat zig Platten
- zur Erweiterung werden zusätzliche Server hinzugenommen

4 PB = 4000 TB
1 Server mit 10 x 4 TB Platten = 100 Server für 4 PB

Apache Hadoop: Open-Source Map-Reduce System

- Hadoop ist ein populäres Map-Reduce System
- Enthält ein Dateisystem (HDFS) zum Speichern von Daten
- Einen Scheduler für Map- und Reduce-Jobs

Apache Hadoop: Open-Source Map-Reduce System

- Hadoop ist ein populäres Map-Reduce System
- Enthält ein Dateisystem (HDFS) zum Speichern von Daten
- Einen Scheduler für Map- und Reduce-Jobs

Wem ist bereits eine Festplatte kaputtgegangen?

Apache Hadoop: Open-Source Map-Reduce System

- Hadoop ist ein populäres Map-Reduce System
- Enthält ein Dateisystem (HDFS) zum Speichern von Daten
- Einen Scheduler für Map- und Reduce-Jobs

Wem ist bereits eine Festplatte kaputtgegangen?

Fault-Tolerance:

- Hadoop kümmert sich um Fehlerbehandlung
- Redundantes Speichern für Ausfallsicherheit

Apache Hive: Datenbank mit Map-Reduce

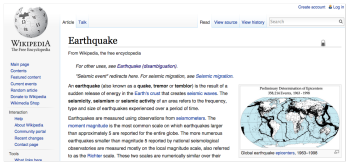
```
SELECT count(*) from txnrecords;
```

```
hive> select count(*) from txnrecords;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201402270420_0005, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201402270420_0005
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201402270420_0005
2014-02-28 20:02:41,231 Stage-1 map = 0%,  reduce = 0%
2014-02-28 20:02:48,293 Stage-1 map = 50%,  reduce = 0%
2014-02-28 20:02:49,309 Stage-1 map = 100%,  reduce = 0%
2014-02-28 20:02:55,350 Stage-1 map = 100%,  reduce = 33%
2014-02-28 20:02:56,367 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201402270420_0005
OK
50000
Time taken: 19.027 seconds
hive>
```

Mehr schneller! (Velocity)

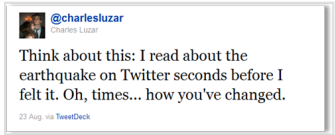
Lebensdauer von Informationen

Lebensdauer von Informationen



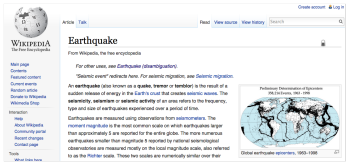
static web

blogs



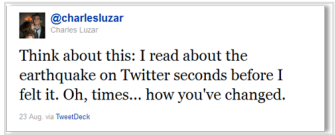
status

Lebensdauer von Informationen



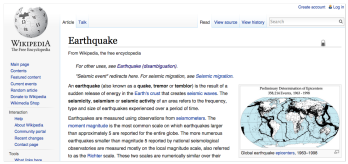
static web

blogs



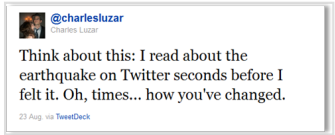
status

Lebensdauer von Informationen



static web

blogs

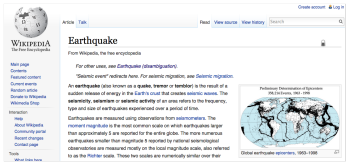


status

Google™

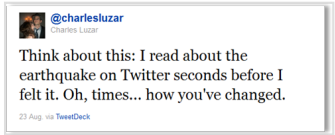
Map-Reduce

Lebensdauer von Informationen



static web

blogs



status

2008

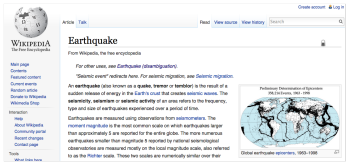
Google

Map-Reduce

Google

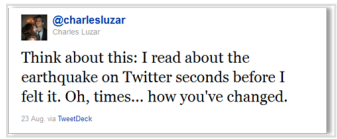
Caffeine

Lebensdauer von Informationen



static web

blogs



status

2008

2013

Google

Map-Reduce

Google

Caffeine

Google

MillWheel

Verarbeitung in Near-Time

- Map-Reduce funktioniert gut, ist aber träge
- Abfragen/Ausführung dauert lange

Wie funktionieren dann z.B. Trends?

- Big Data Verarbeitung in Echtzeit/Neartime
- Ähnliche Prinzipien: Verteilte Berechnungen

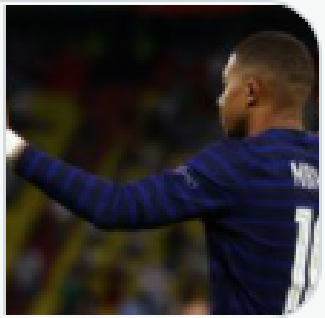
Twitter: >500k Tweets pro Minute

Sports · Trending

#FRASUI

Sports

France vs Switzerland:
Switzerland advances to ...



659K Tweets

Trending in Germany

#Unwetter

2,825 Tweets

Twitter Trends: Top-k Anfragen

Finde die *k* häufigsten Hashtags!

Twitter Trends: Top-k Anfragen

Finde die *k* häufigsten Hashtags!

Einfacher Ansatz:

- 4-Byte Zähler pro Hashtag
- Führt zu 1.8 GB Speicherverbrauch für 1-6 stellige Hashtags

Twitter Trends: Top-k Anfragen

Finde die *k* häufigsten Hashtags!

Einfacher Ansatz:

- 4-Byte Zähler pro Hashtag
- Führt zu 1.8 GB Speicherverbrauch für 1-6 stellige Hashtags

Auch hier: Das funktioniert nur verteilt! (Scale-Out)

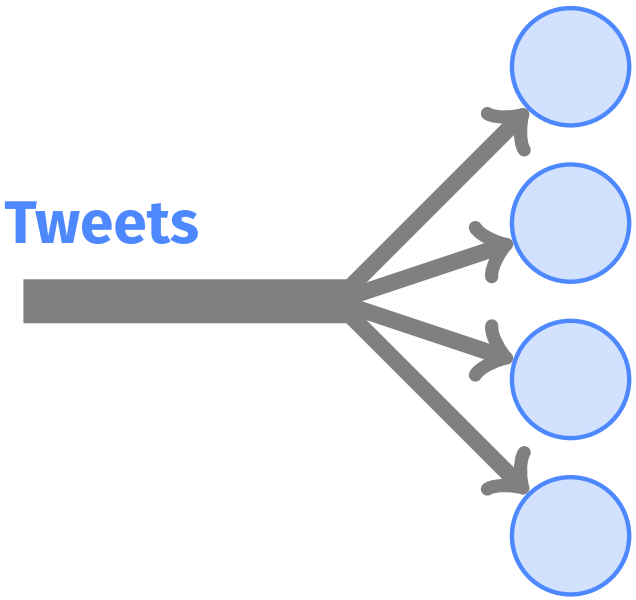
Frage:

Wie können wir die Daten so aufteilen, dass sie zählbar bleiben?

Tweets

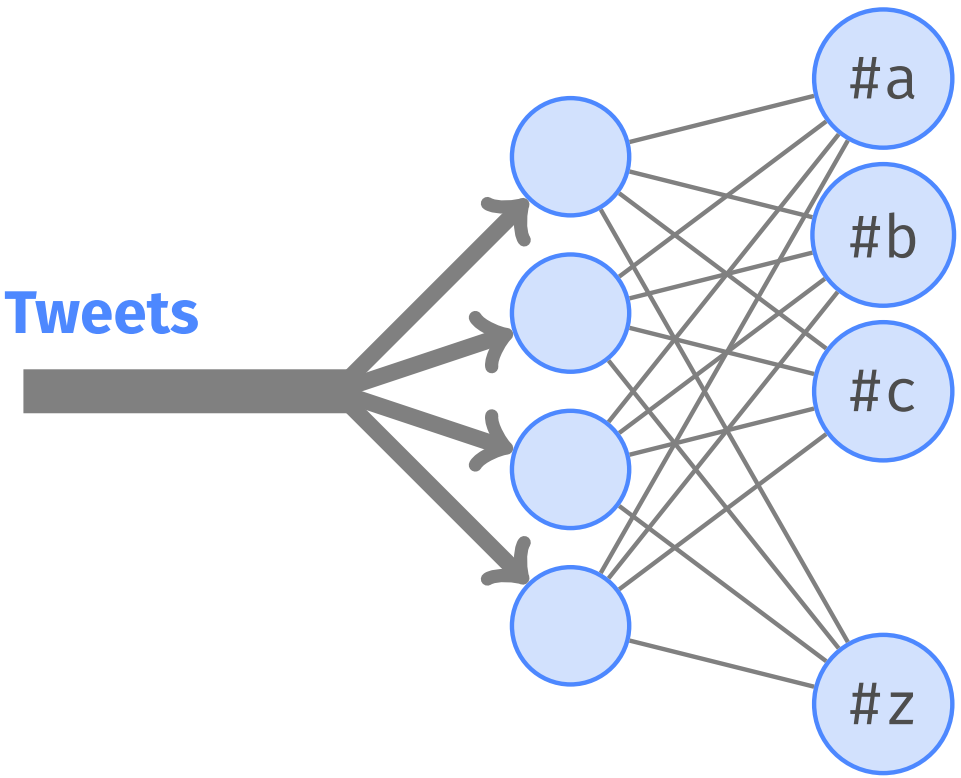
Frage:

Wie können wir die Daten so aufteilen, dass sie zählbar bleiben?



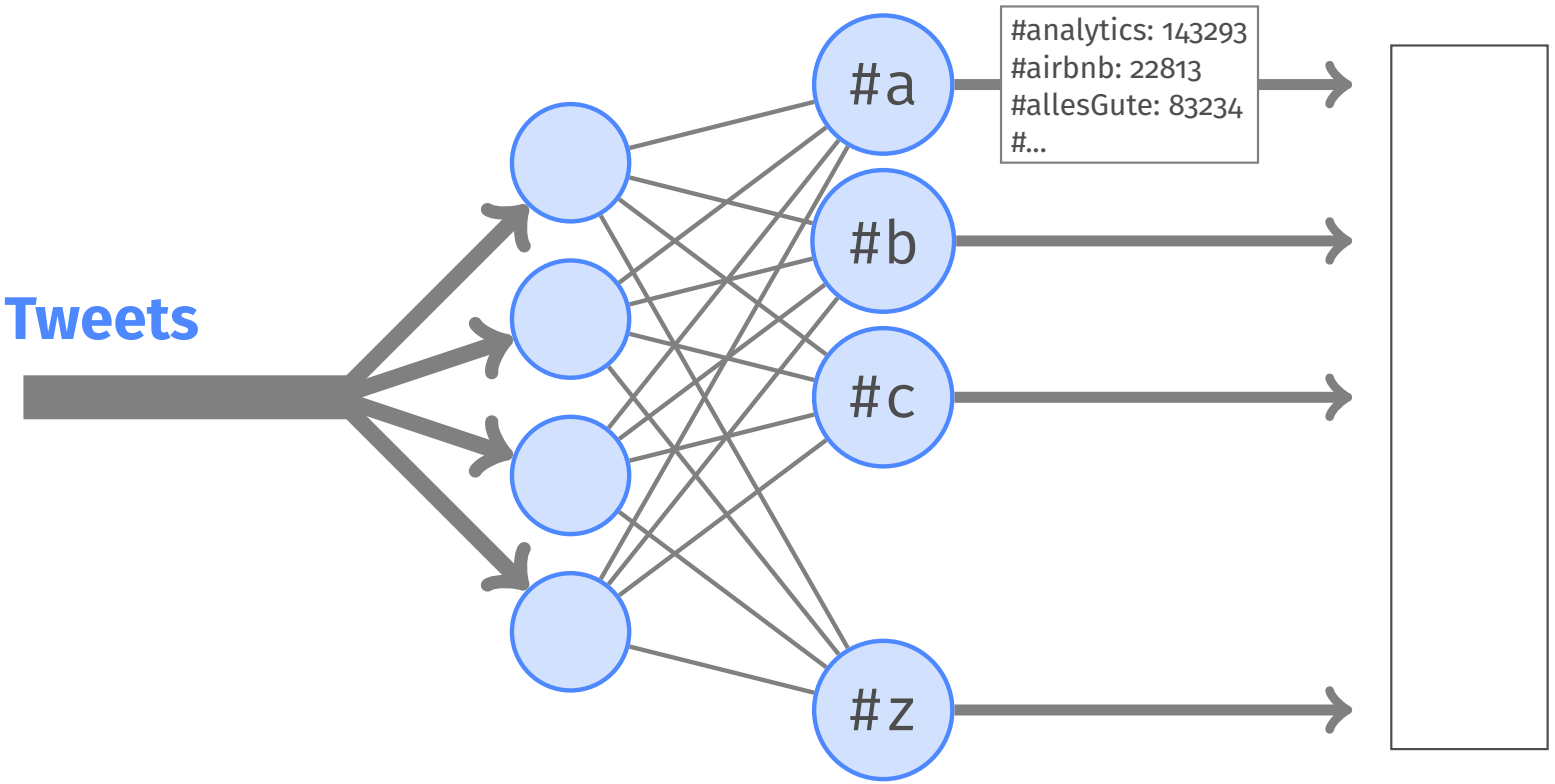
Frage:

Wie können wir die Daten so aufteilen, dass sie zählbar bleiben?



Frage:

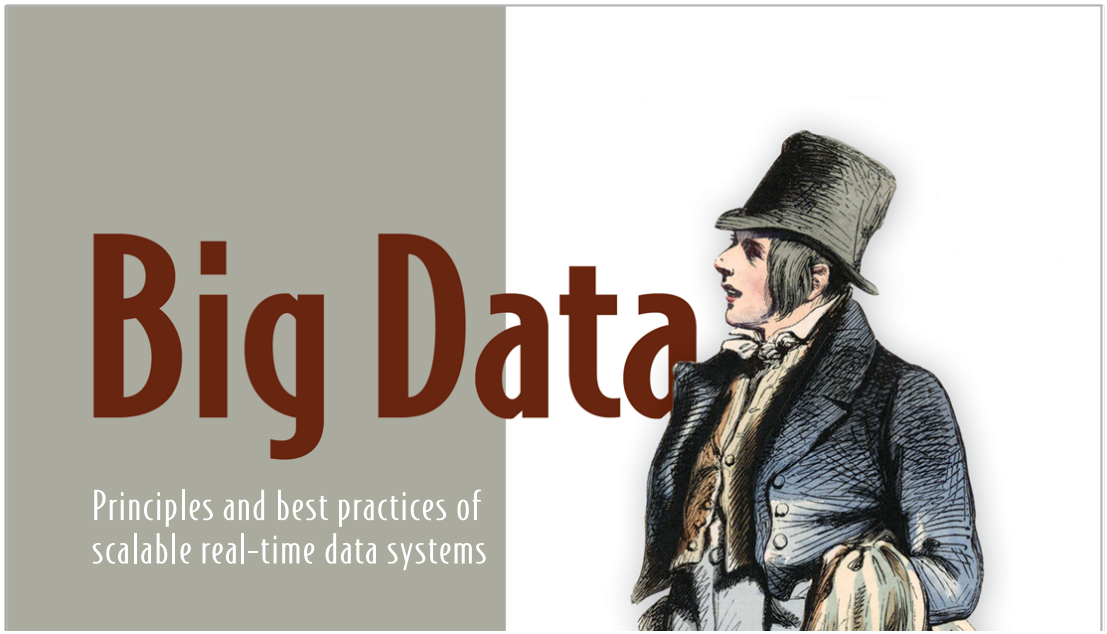
Wie können wir die Daten so aufteilen, dass sie zählbar bleiben?



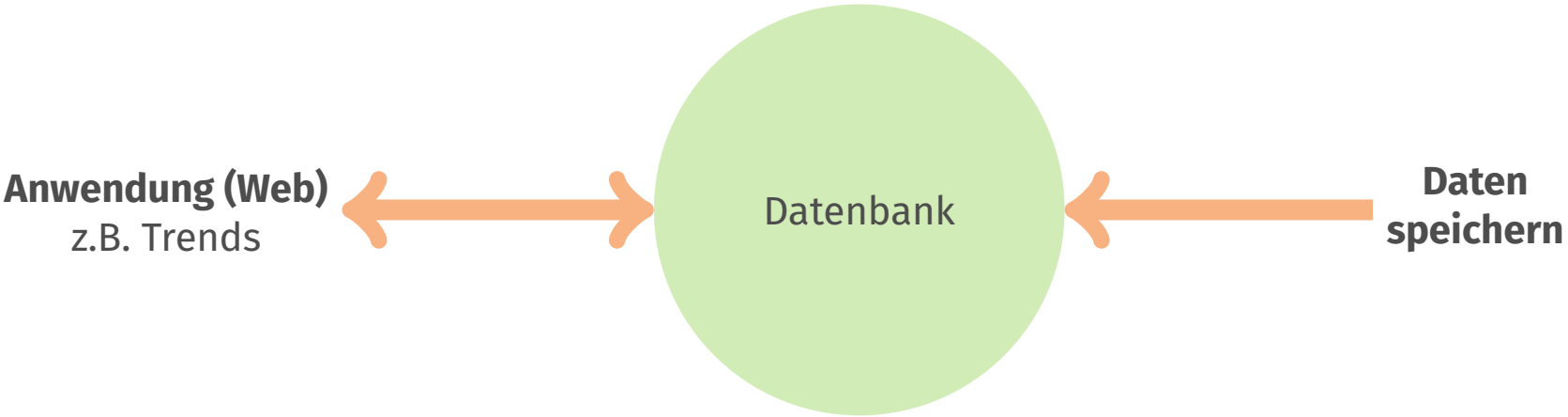
Big Data Architektur

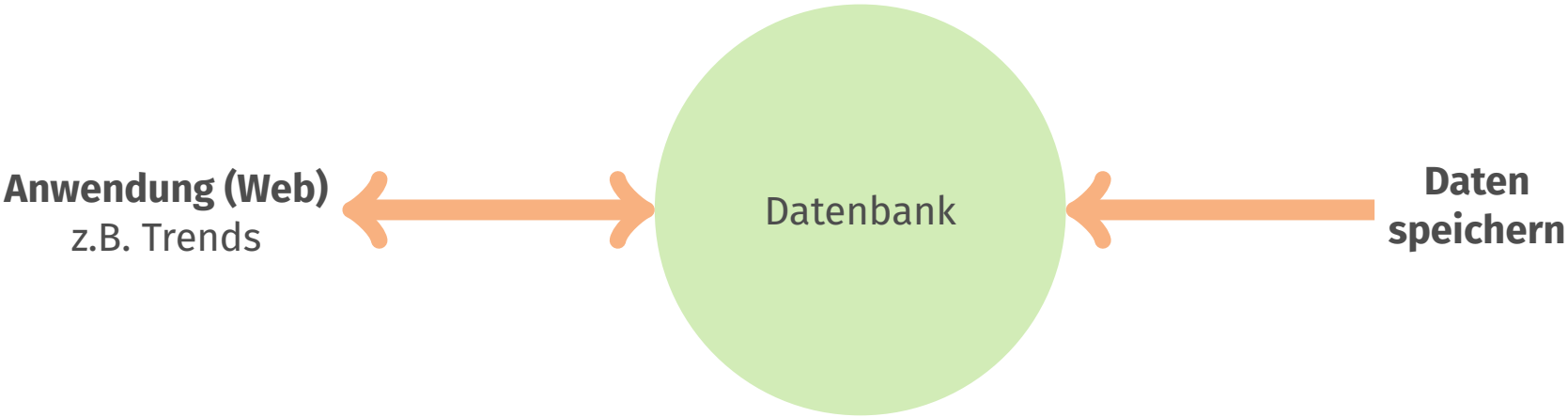
Analytics: Beantwortung von Fragen

```
result = query ( all data )
```



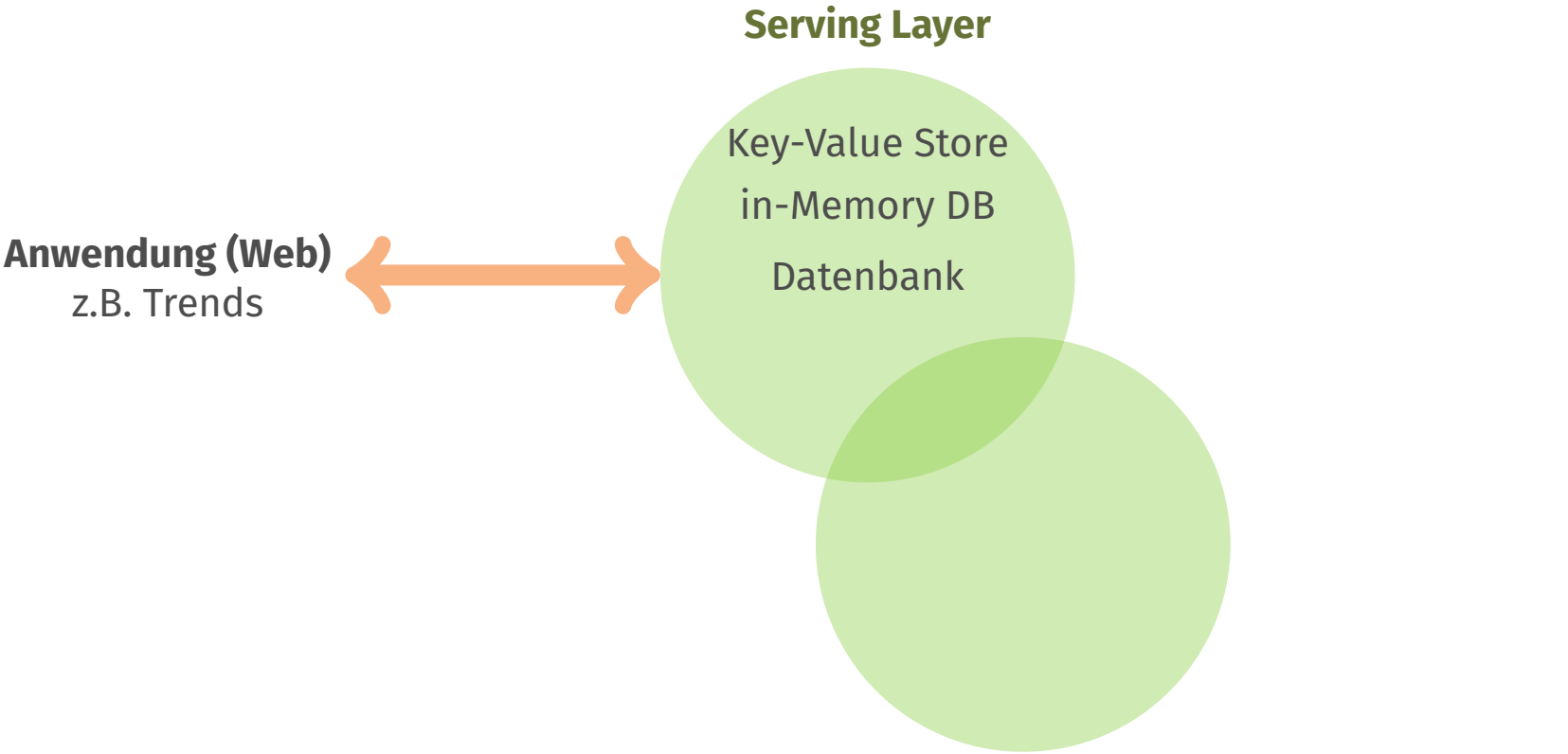
Lambda Architektur



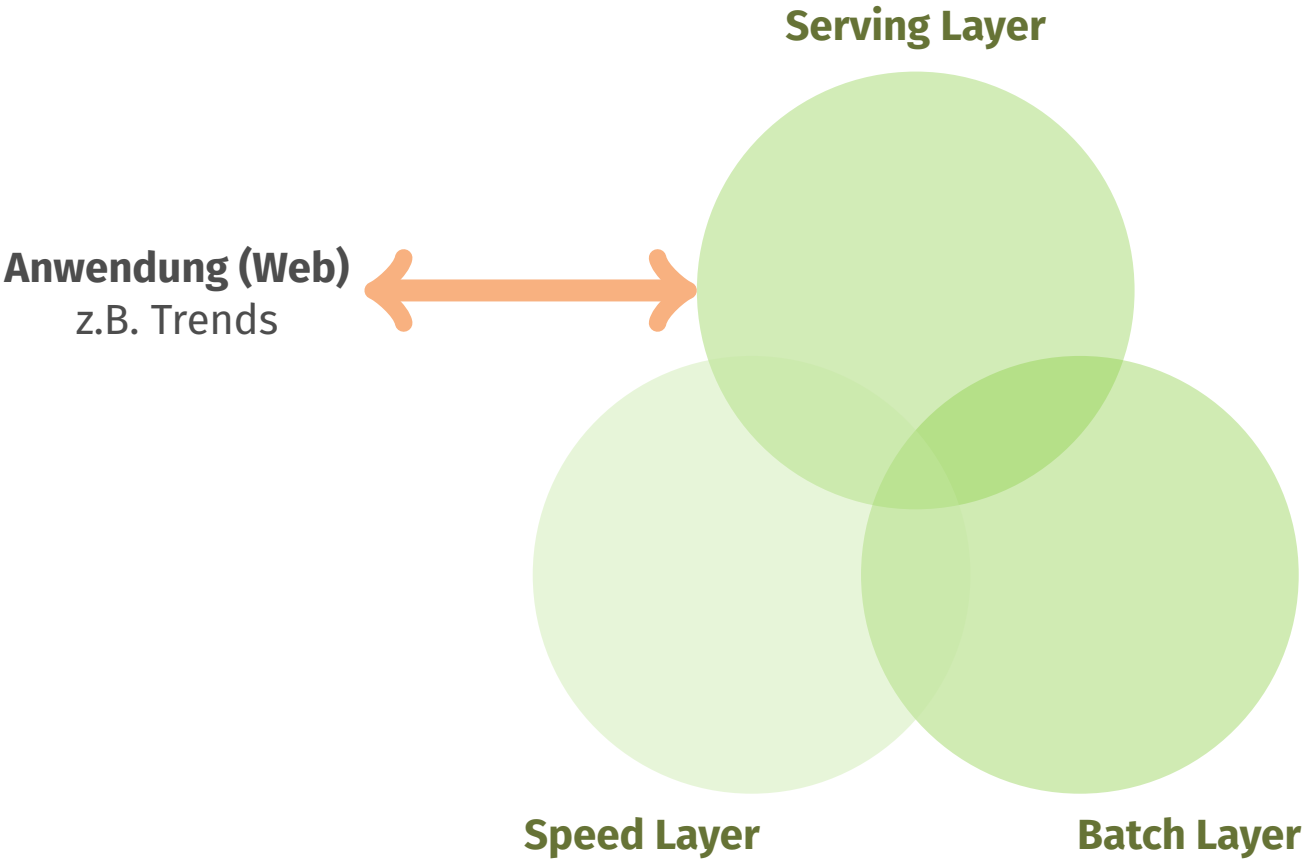


Typische Architektur einer Web-Anwendung

Irgendwann erreicht die Datenbank eine Größe/Komplexität,
dass Anfragen nicht mehr ad-hoc berechnet werden können.

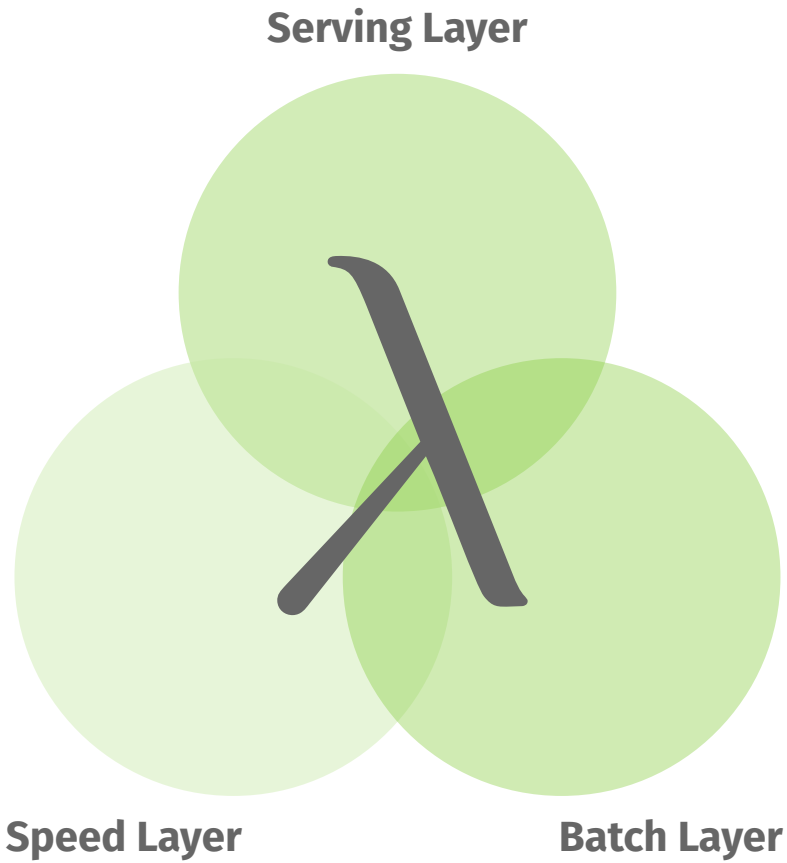

Serving Layer
Key-Value Store
in-Memory DB
Datenbank
Anwendung (Web)
z.B. Trends
Batch Layer
Vorberechnen von Ergebnissen/Aggre-
gationen - die Datenbank wird entlastet;
Daten werden in verteiltem System gespeichert

Lambda Architektur



Aktuelle Informationen werden online vorberechnet
und die Ergebnisse in die Datenbank geschrieben.

Lambda Architektur



Kappa-Architektur

- Kein Batch-Layer mehr
- Berechnungen nur noch online/im Stream
- Großer Storage aus dem die Vergangenheit “gestreamt” werden kann

Beispiel: Besucherzahlen

Beispiel: Analyse von Studierenden-Zahlen


```
{id:252,date:'2022-12-06 09:47',room:'AW-01-39',value:+1}
```

Beispiel: Analyse von Studierenden-Zahlen


```
{id:252,date:'2022-12-06 09:47',room:'AW-01-39',value:+1}
```

Ziele:

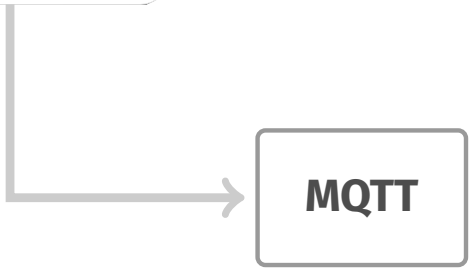
- Echtzeit-Darstellung der Raumbelegung
- Möglichkeit zur Analyse historischer Daten

Beispiel: Analyse von Studierenden-Zahlen


```
{id:252,date:'2022-12-06 09:47',room:'AW-01-39',value:+1}
{id:253,date:'2022-12-06 09:48',room:'AW-01-39',value:-1}
{id:254,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:255,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:256,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:258,date:'2022-12-06 09:52',room:'AW-01-39',value:+1}
```

Beispiel: Analyse von Studierenden-Zahlen


```
{id:252,date:'2022-12-06 09:47',room:'AW-01-39',value:+1}
{id:253,date:'2022-12-06 09:48',room:'AW-01-39',value:-1}
{id:254,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:255,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:256,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:258,date:'2022-12-06 09:52',room:'AW-01-39',value:+1}
```



Broker

Beispiel: Analyse von Studierenden-Zahlen

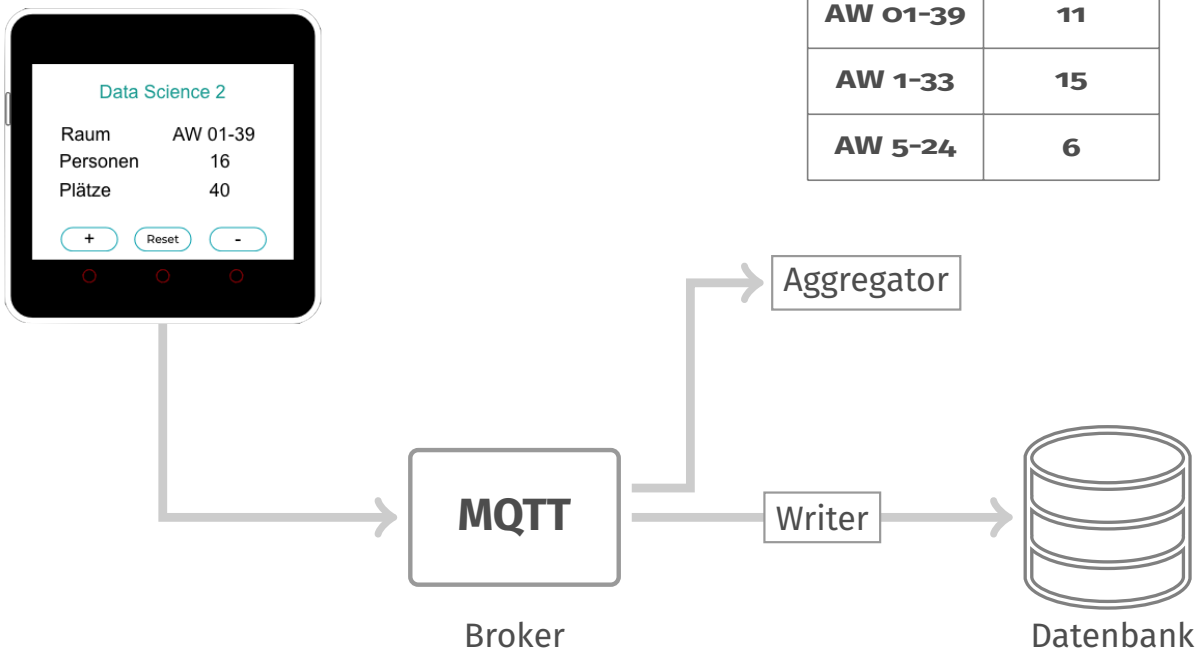

```
{id:252,date:'2022-12-06 09:47',room:'AW-01-39',value:+1}
{id:253,date:'2022-12-06 09:48',room:'AW-01-39',value:-1}
{id:254,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:255,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:256,date:'2022-12-06 09:51',room:'AW-01-39',value:+1}
{id:258,date:'2022-12-06 09:52',room:'AW-01-39',value:+1}
```

Broker

Datenbank
(= historischer Speicher)

Beispiel: Analyse von Studierenden-Zahlen

Raum	Anzahl
AW 01-39	11
AW 1-33	15
AW 5-24	6



Nachrichtenverarbeitung im Aggregator

```
def on_message(client, userdata, msg):
    print(f"New message from ‘{msg.topic}‘ topic")
    global counts
    data = loads(msg.payload.decode())
    room = data['room']

    if room not in counts:
        counts[room] = 1
    else:
        counts['room'] = counts['room'] + 1
```

```
# Topic, auf dem Messwerte publiziert werden
topic = 'counts'

def subscribe(mqtt):
    mqtt.subscribe(topic)
    mqtt.on_message = on_message
```

Going Big Data: Was passiert, wenn der Aggregator abstürzt?

- Status geht verloren (aktuelle Anzahlen)
- Neustart beginnt mit 0 Werten für alle Räume

Apache Kafka als Broker

- Hoch-Skalierbarer Message Broker
- Clusterfähig (hochverfügbar)
- Entwickelt bei LinkedIn, Open Source

Apache Kafka als Broker

- Hoch-Skalierbarer Message Broker
- Clusterfähig (hochverfügbar)
- Entwickelt bei LinkedIn, Open Source

Nachrichten im Topic werden
temporär gespeichert (z.B. 1 Monat)