Big Data
01
Einführung

Eric MSP Veith <veith@offis.de>

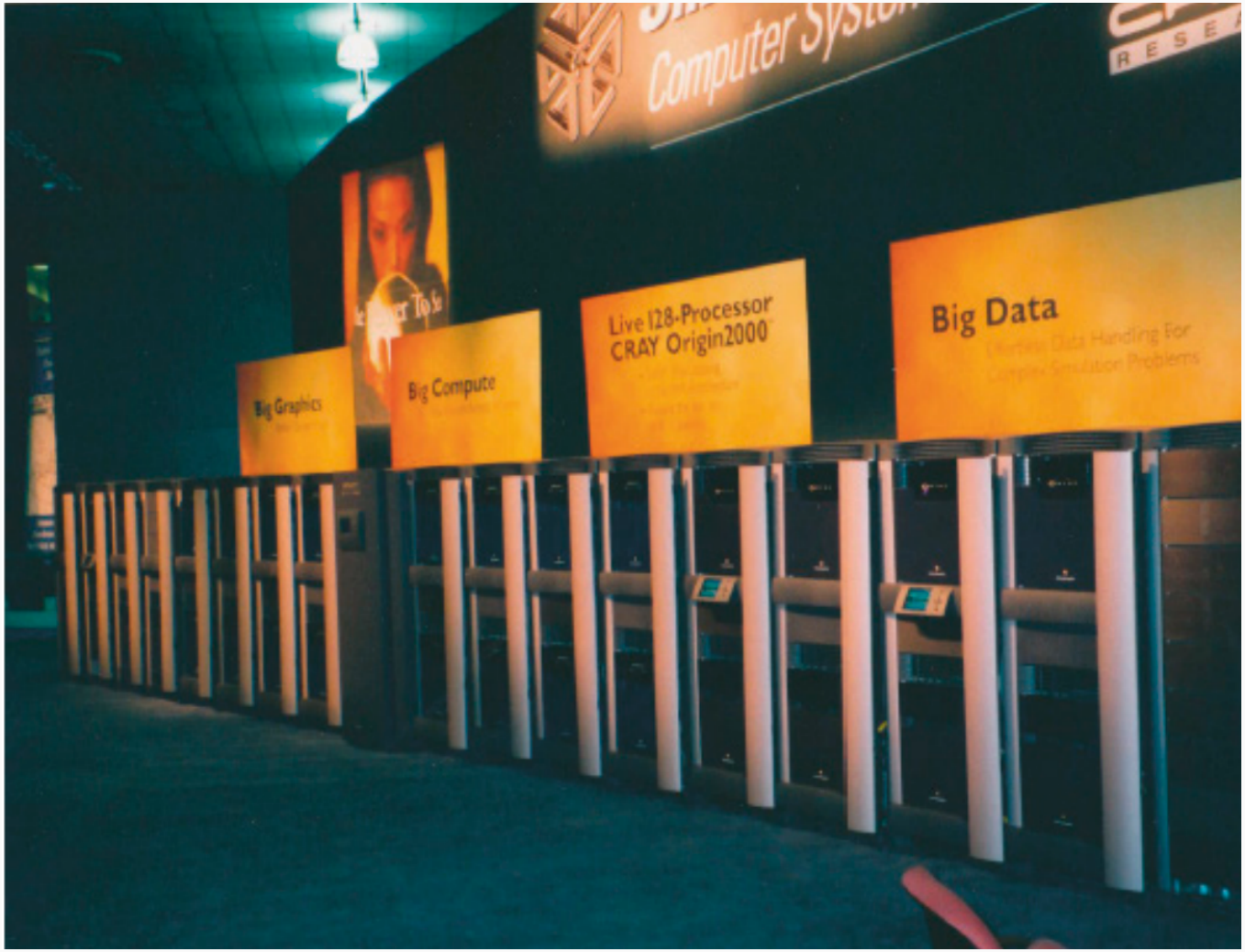
March 2, 2020

Organisatorisches

- Prüfungsleistung: Seminararbeit
  - 10 Seiten Ausarbeitung zum Thema: Arial o.ä., Schriftgröße 12pt, einfacher Zeilenabstand, 2cm Rand, keine separate Titelseite, Seitenzahl inkl. Literaturverzeichnis
  - 15 Minuten Vortrag zum Thema
- Seminarthemen & Folien online verfügbar:
  `https://vhome.offis.de/~eveith`

Die Einführung des Begriffs

Historie



IEEE Supercomputing Conference, Pittsburgh, USA

Photo: Michael Woodacre

Historie

- Erster Wikipedia-Artikel dazu November 2009 – sofort wieder gelöscht:

  > *„Delete as per nom – it is simply a combination of big and data, dictionary words which have no place here. I'm not even sure it's a neologism, and even if it was it doesn't need an article." (John Blackburne)*

- Gartner Hype Cycle: 2011
- *Hadoop* seit 2006 entwickelt (!)

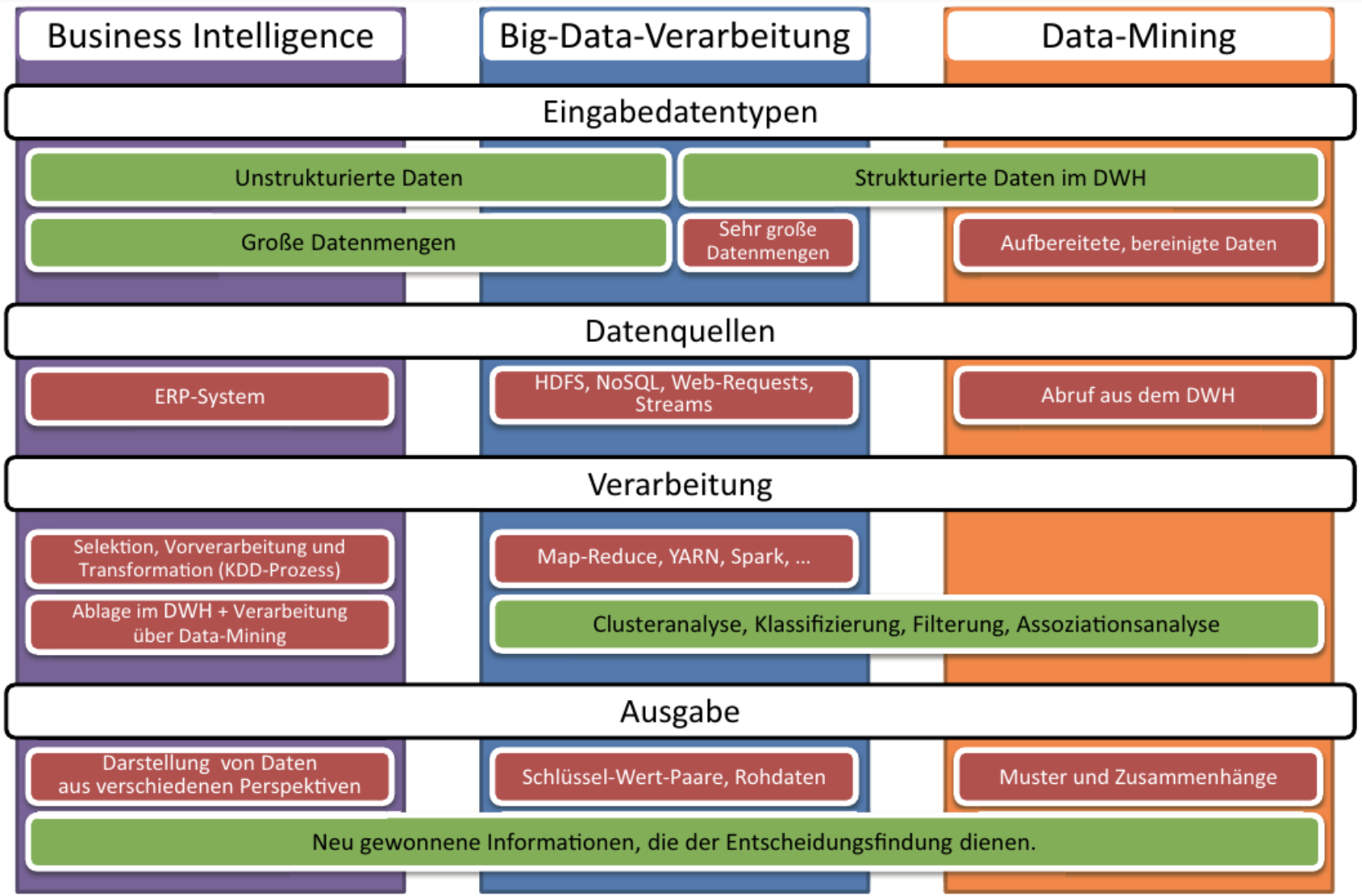
Der Begriff *Big Data*

„Der aus dem englischen Sprachraum stammende Begriff Big Data (von englisch big ‚groß‘ und data ‚Daten‘, deutsch auch Massendaten) bezeichnet Datenmengen, welche beispielsweise zu groß, zu komplex, zu schnelllebig oder zu schwach strukturiert sind, um sie mit manuellen und herkömmlichen Methoden der Datenverarbeitung auszuwerten.“ (Wikipedia)

Der Begriff *Big Data*

„Ohne in die Tiefe zu gehen, lässt sich Big Data so definieren: Big Data sind Datenmengen, die zu groß für traditionelle Datenbanksysteme sind, eine hohe Halbwertszeit besitzen und in ihrer Form nicht den Richtlinien herkömmlicher Datenbanksysteme entsprechen. Ziel ist es nun, diese Daten dennoch zu speichern und zu verarbeiten, sodass aus ihnen zeitnah wertvolle, neue Informationen gewonnen werden können.“ (Freiknecht, & Papp: Big Data in der Praxis)

Business Intelligence – Big Data – Data Mining



Freiknecht, & Papp. „Big Data in der Praxis“

Was ist nun Big Data...?

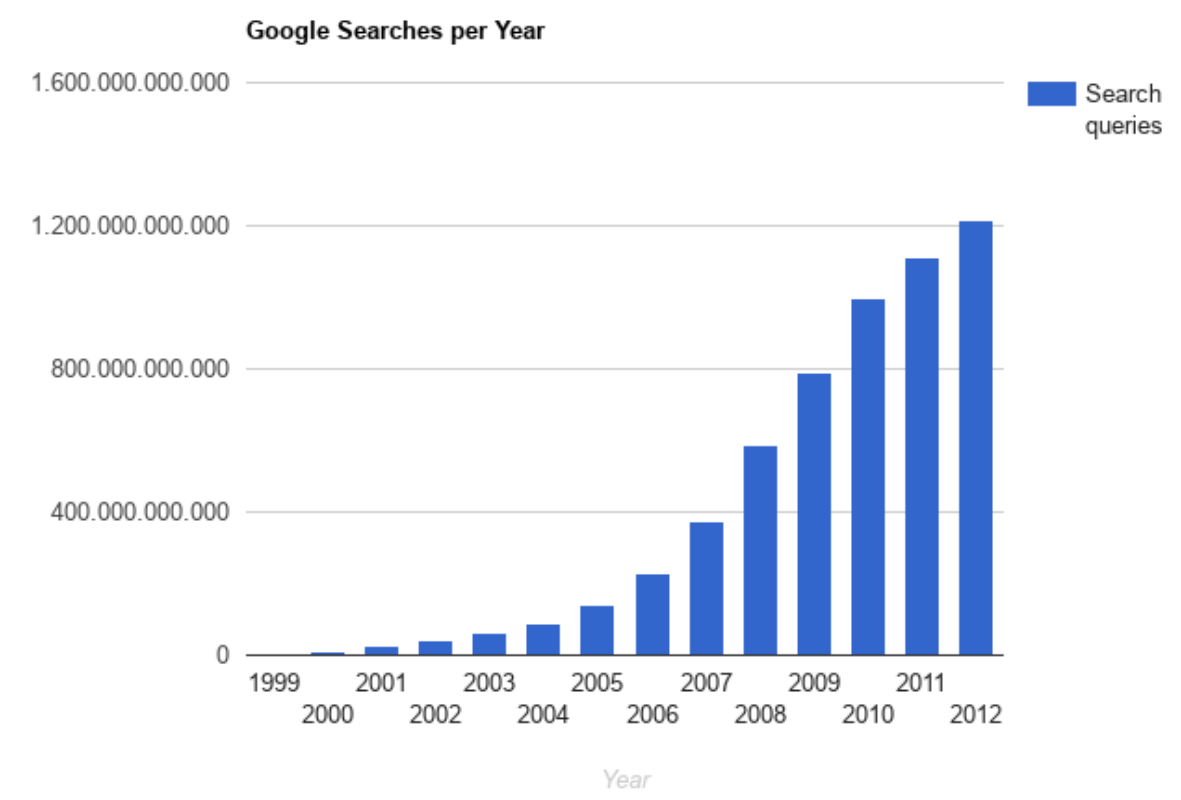
Volume	Große, stetig wachsende Datenmengen im hohen Terabyte-/Petabyte-Bereich
Velocity	Daten werden mit hoher Frequenz generiert, Behandlung nahe Echtzeit nötig
Variety	Keine uniformen Datenformate, sondern sogar teils unstrukturierte Daten
Veracity	Daten sind auch fehlerbehaftet/nicht vorverarbeitet und aufbereitet
Value	Daten allein sind noch kein Mehrgewinn; Big Data ist eine „Zubringertechnologie“

Volume: Twitter

- $\sim$ 6000 Tweets pro Sekunde – $\sim$ 500 Mio Tweets pro Tag
- 12 TB pro Tag (inkl. Indices) (= 4.3 PB pro Jahr)
- Ursprünglich *Ruby on Rails* mit *MySQL*
- Jetzt hauptsächlich JVM-basierte Sprachen (Scala, etc.) + MySQL Load Balancer
- Nicht nur Twitter – natürlich auch Google: "Sorting 1 PB of data with MapReduce" (`https://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html`)

Velocity: Google

- 40 000 Suchanfragen pro Sekunde = 3.5 Mrd pro Tag



Suchanfragen pro Jahr; Quelle: Google Zeitgeist 2012

Variety

Format Datenbanken, Text-/CSV-Dateien, HDF5, ...

Struktur Unterschiedliche Schemata der Datenbanken (z.B. je nach Anwendung), unterschiedliche Primärschlüssel, ...

Unstrukturierte Daten PDFs, E-Mails, Word-Dateien, Wiki, ...

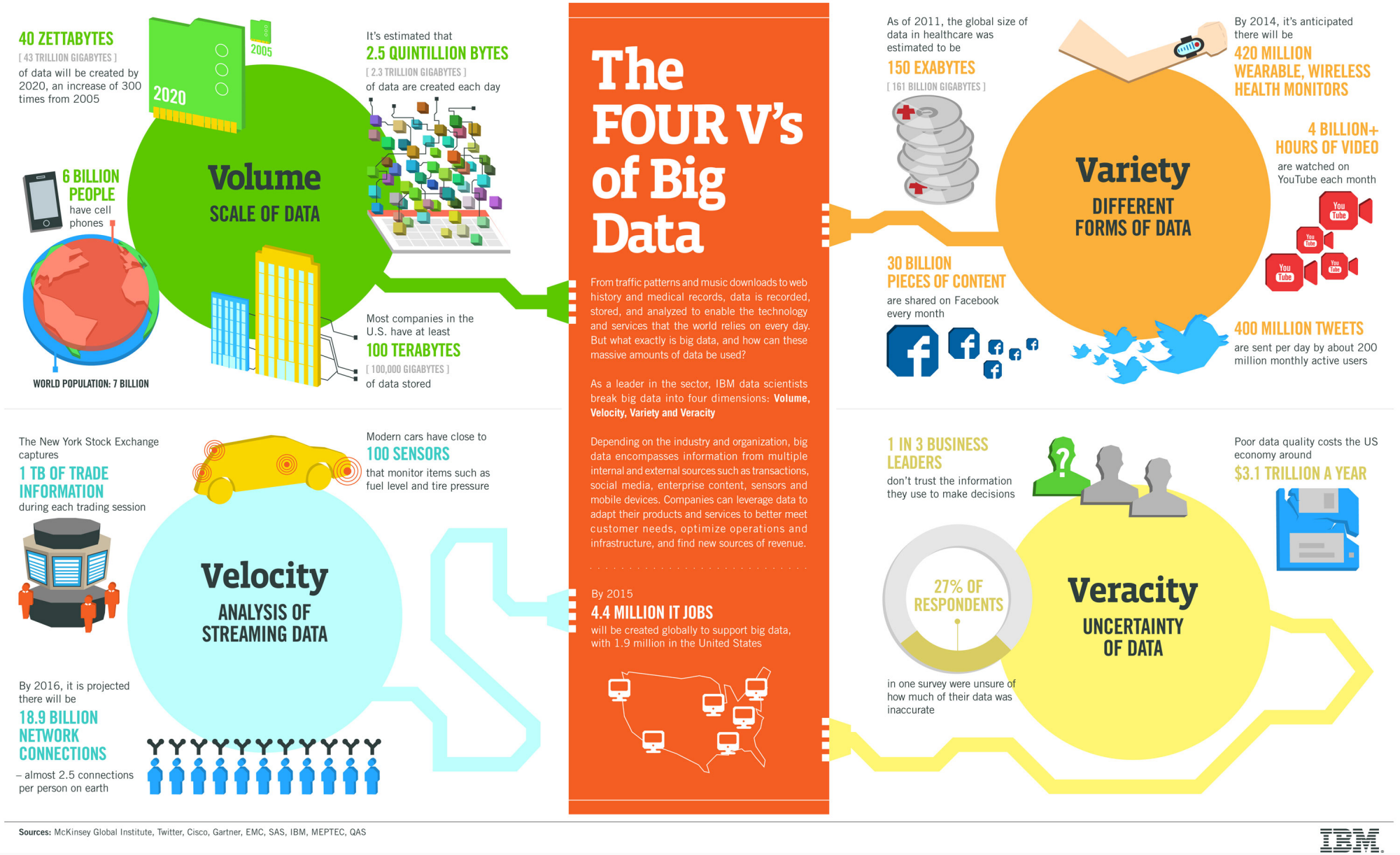
Natürliche Sprache Deutsch, Englisch, „Denglisch“ (Mehrdeutigkeiten im Allgemeinen), ...

Medientypen Text, Bilder, Videos, ...

Veracity

Herkunft	„Abstammung“ der Daten (auch weiterverarbeiteter) – Quelle von Unstimmigkeiten/Ungenauigkeiten identifizieren
Bugs	Logische Softwarefehler
Rauschen	von Sensoren, Zufallszahlen, . . .
Ausreißer	Fehlmessungen, Verhalten von Bots, . . .
Sicherheit	Manipulation durch Angreifer
Präzision	Zeitliche Auflösung der Daten, Aggregationen, . . .
Alter	Verfallsdatum von Informationen

Die vier (!) „V“ von Big Data



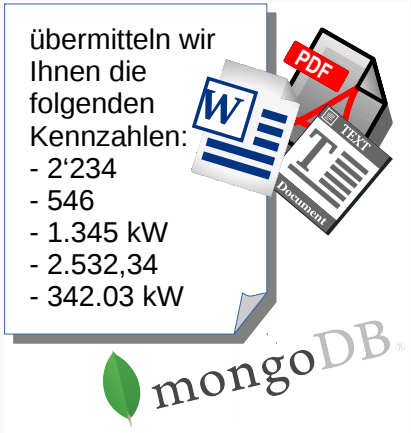
Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

IBM

Datenhaltung „historisch“
Zubringer für Big Data

Unstrukturierte Daten

- Kein Datenmodell
- Keine Metainformationen
- Hoher Aufwand beim Einlesen (Parsen); zeit- und fehleranfällig




```
import re

pat = re.compile("^- ([-0-9.,']+)")
with open('foo') as fp:
    for line in fp:
        m = pat.search(line)
        kw = int(m.group(0)) if m

Traceback (most recent call last):
  File "<stdin>", line 1, in
  ↪  <module>
ValueError: invalid literal for
↪  int()
    with base 10: "2'234"
```

Excel

- Annähernd strukturiert
- Metainformationen
- Achtung: i18n:
  - Zahlenformate
  - Namen von Tabellen („Tabelle1“)
- Proprietäres Datenformat :-(

```
import openpyxl
from openpyxl import Workbook
wb =
 ↪  openpyxl.load_workbook(data_file,
 ↪  data_only=True)
ws = wb['Tabelle1']
ws['A1'] = datetime.datetime(2018,
 ↪  3, 18)
ws['A1'].number_format
# => 'yyyy-mm-dd h:mm:ss'
wb.guess_types = True
```

CSV

- Bei weitem nicht auf Komma und Zeilenumbruch beschränkt
- Offenes Format:
  - Text
  - Trenner für Datensätze, Datenfelder und Feldbegrenzer
  - Keine Standardisierung
- Mit das am häufigsten vorkommende Datenformat
- Problem: Interpretation (kein Datenmodell, keine Metainformationen)

```
4; 54,2 kW; 8:09; ok
       ...
15  5  00:15
       ....
"3","4,56","14:35","\\"Abschluss\
```

HDF5

- Vor allem in Forschung präsent
- Effizient auch bei großen Datenmengen
- Hierarchisches Dateiformat
  - *Dataset*: Mehrdimensionale Matrix eines homogenen Datentyps
  - *Group*: Container für *Datasets* und andere Gruppen
- Offener Standard (interoperabel)
- Vergleichsweise geringe

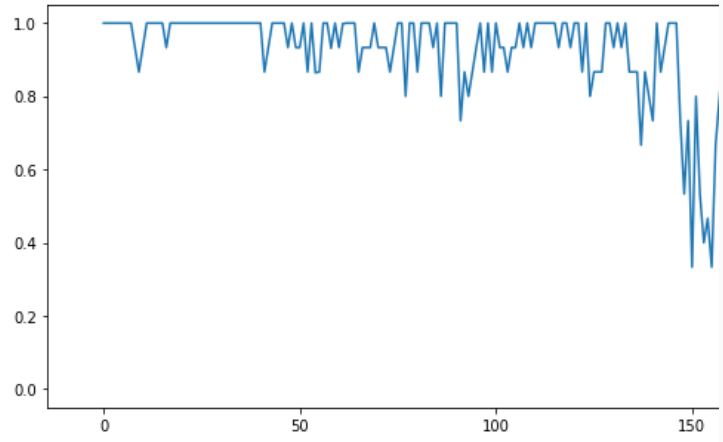
```
In [21]: %matplotlib inline

         import h5py
         import numpy as np
         import matplotlib.pyplot as plt

In [20]: h5 = h5py.File('wind_braunschweig.hdf5', 'r')
         for k, v in h5['feedin'].attrs.items():
             print("%s = %s" % (k, v))

         start = 1293836400
         resolution = 900

In [24]: fig, ax = plt.subplots(figsize=(15, 5))
         _ = ax.plot(h5['feedin'][0:4*72])
```



PostgreSQL

- Wohlbekannt (geringe Schulungskosten!)
- alle Vorteile einer SQL-Datenbank
- Auch Dokumente via JSON

```
'{"a":1,"b":2}'::json->'b'
```

- `LISTEN`/`NOTIFY` für asynchrone Arbeit
- Schemata: Daten in gutem Format für Deep Learning

Asynchrone Verarbeitung mit PostgreSQL

- `LISTEN channel;`

  lauscht auf Kanal (erzeugt, falls nötig)
- `NOTIFY channel payload;`

  benachrichtigt alle Zuhörer (`payload` optional)
- Kombination mit asynchroner Behandlung sinnvoll, z. B. `select(·)`

Asynchrone Verarbeitung mit PostgreSQL

```
import select; import psycopg2; import psycopg2.extensions

conn = psycopg2.connect(DSN)
conn.set_isolation_level(
  psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
curs = conn.cursor()
curs.execute("LISTEN somechannel;")

while True:
    if select.select([conn], [], [], TIMEOUT) != ([], [], []):
        conn.poll()
        while conn.notifies:
            notify = conn.notifies.pop(0)
            print("Got NOTIFY: ", notify.pid,
                                  notify.channel,
                                  notify.payload)
```

Nachteile von PostgreSQL

- Schemata: Vorverarbeitung nötig
- Abfragegeschwindigkeit abhängig von Tabellengröße

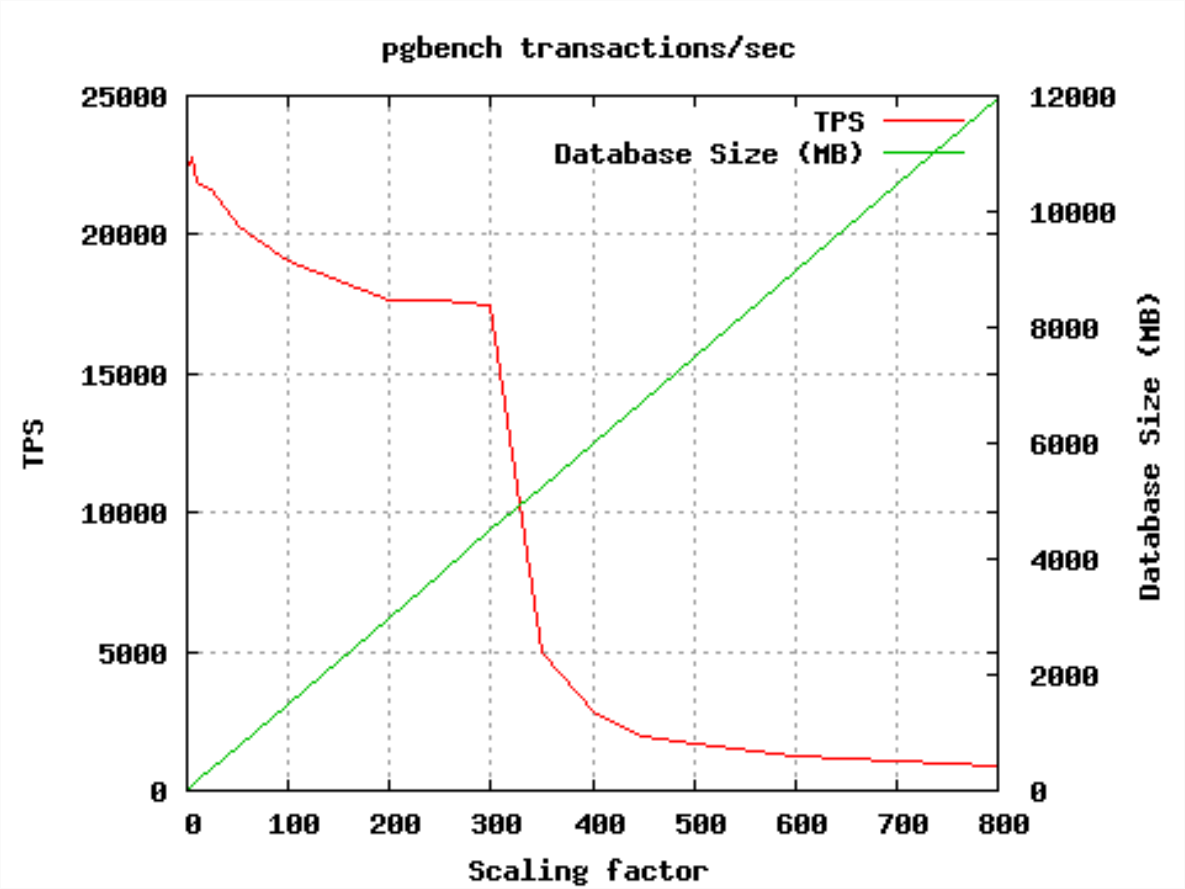


Table Partitioning

- Max. Größe einer Tabelle: 32 TB
- Je größer die Tabelle, desto kleiner die Abfragegeschwindigkeit
- Lösung: **Table Partitioning**

```
CREATE TABLE consumption (
    id SERIAL,
    sensor_id INT NOT NULL,
    log_ts TIMESTAMP WITH TIME ZONE NOT NULL,
    value REAL NOT NULL)
PRIMARY KEY (id)
PARTITION BY RANGE (log_ts);

CREATE TABLE consumption_y2006m02 PARTITION OF consumption
FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');
```

Table Partitioning

```
CREATE OR REPLACE FUNCTION create_partition_and_insert()
RETURNS trigger AS
$BODY$
DECLARE partition_date TEXT; partition TEXT;
BEGIN
  partition_date := to_char(NEW.log_ts, 'YYYY_MM_DD');
  partition := TG_RELNAME || '_' || partition_date;
  IF NOT EXISTS(SELECT relname FROM pg_class WHERE relname=parti
    EXECUTE 'CREATE TABLE ' || partition
      || ' INHERITS (' || TG_RELNAME || ');';
  END IF;
  EXECUTE 'INSERT INTO ' || partition
    || ' SELECT(' || TG_RELNAME || ' ' || quote_literal(NEW)
    || ').* RETURNING id;';
  RETURN NULL;
END;
$BODY$
```

Table Partitioning

```
CREATE TRIGGER measurements_partition_insert_trigger
BEFORE INSERT ON measurements
FOR EACH ROW EXECUTE PROCEDURE
    create_partition_and_insert();
```

- Partitionen normalerweise manuell angelegt
- Alternative: Trigger (wie oben), verlangsamt aber Einfügevorgang
- **Default Partition** möglich

```
CREATE TABLE measurement_default
PARTITION OF measurement DEFAULT;
```

Table Partitioning

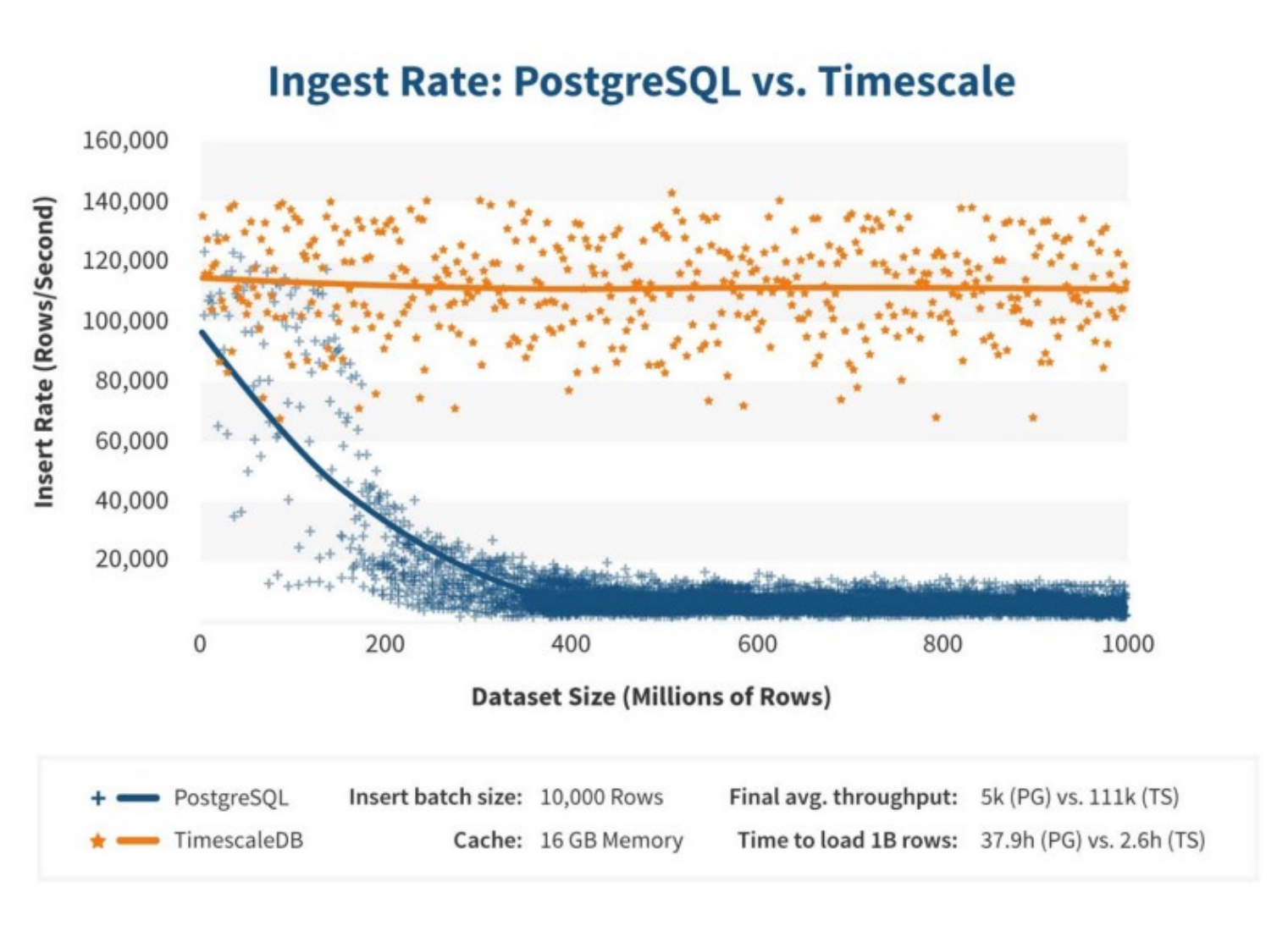
- Speichermedium je nach Aktualität der Partition (z. B. HDD vs. SDD)
- Weitehin Indices, Trigger, etc. möglich

```
CREATE INDEX ON consumption (log_ts);
CREATE INDEX ON consumption (sensor_id);
```

- UPDATE verschiebt Datensätze zwischen Partitionen
- Auch Subpartitionen möglich

```
CREATE TABLE consumption_y2006m02
PARTITION OF consumption
FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');
PARTITION BY RANGE (sensor_id);
```

Table Partitioning



TimescaleDB

- Viele Daten im Deep-Learning-Kontext sind **Zeitreihendaten**
- **Zeitzentriert:** Immer mit Zeitstempel
- **Append-only:** Daten werden angehängt, selten verändert, nie gelöscht
- **Recency:** Neue Daten beziehen sich fast ausschließlich auf aktuelle Zeitintervalle; kein Auffüllen

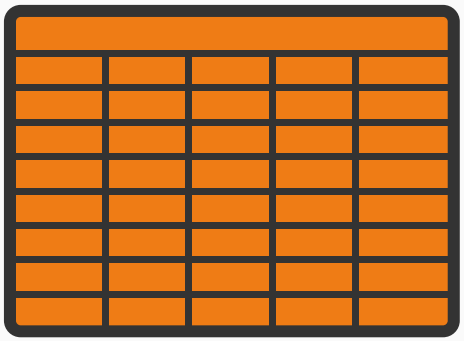
Timestamp	Power	Wind Speed	Wind Dir.
2014-10-23T21:10:00+01:00	223.12	348.65	4.98
2014-10-23T21:20:00+01:00	149.22	349.00	4.42
2014-10-23T21:30:00+01:00	181.65	349.10	4.74

Datenorganisation von TimescaleDB

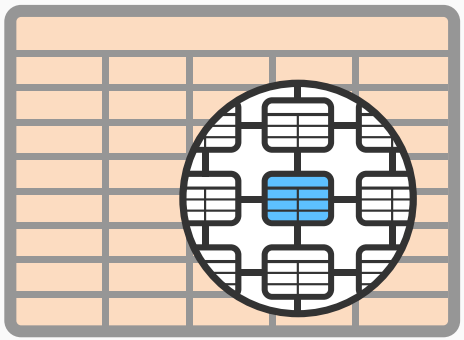
- Benutzung sehr einfach:

```
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;
SELECT create_hypertable('measurements', 'log_ts');
```

- Zeitreihentabelle (**Hypertable**) besteht aus mehreren disjunkten **Chunks**
- Chunks: rechtslastige B-Bäume
- Chunks sollten in den Arbeitsspeicher passen (ca. 25% RAM für die neusten Chunks); Zeitintervall entsprechend wählen

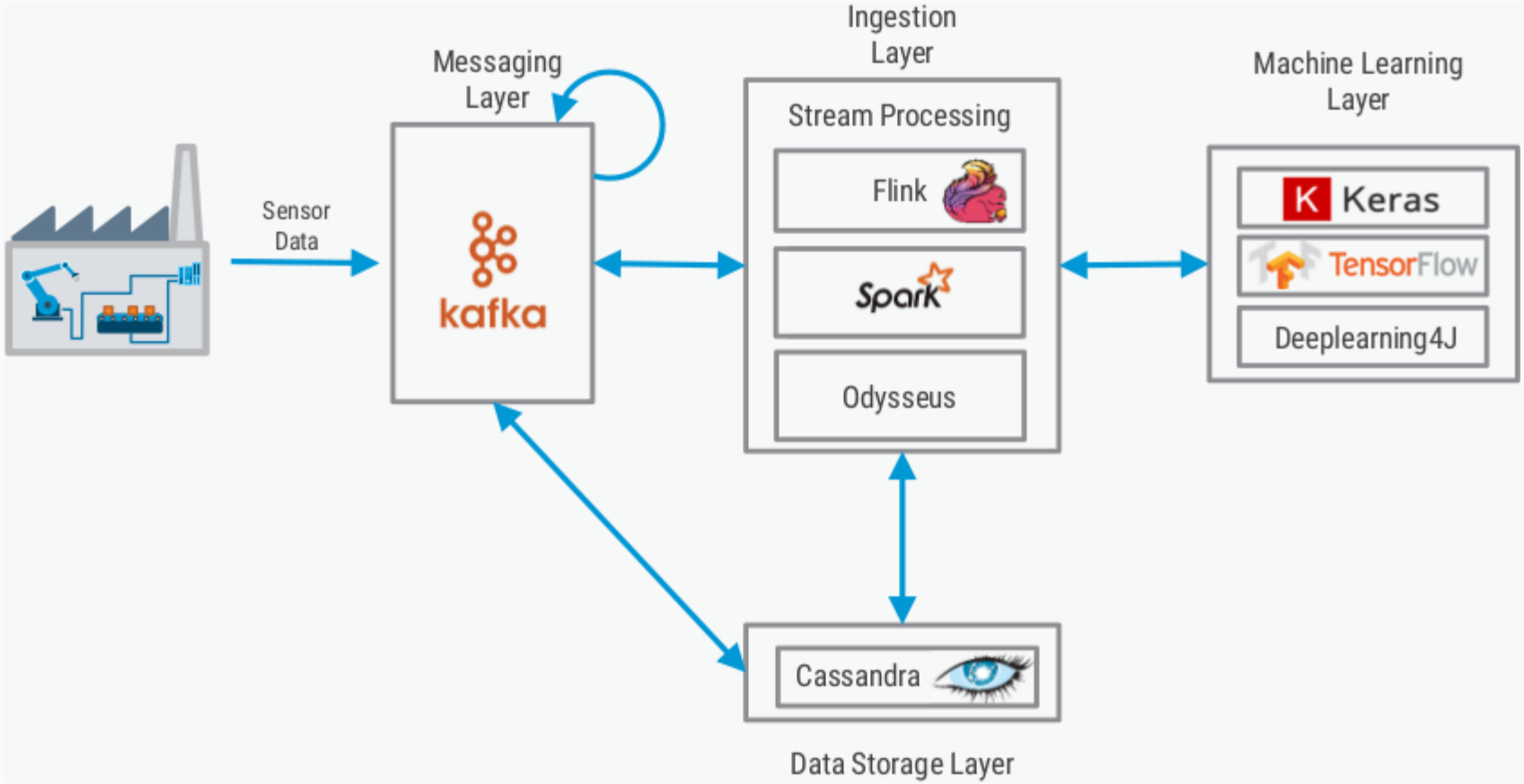


Hypertable



Chunk

Ein *Big Data Stack*



Zusammenfassung

- **Pragmatismus:** Nutzen Sie, was funktioniert und was die Datenquelle liefern kann!
- **Kommunikation:** Reden Sie mit dem Datenlieferanten!
- **Dokumentation:** Schnittstellen sind nur so gut wie ihre Beschreibung.

Wenn Auswahl besteht, dann...

- Explorativ:
  1. **Kleine Datenmengen:** HDF5, CSV
  2. **Große, homogene/vorverarbeitete Daten:** HDF5
- Produktiv:
  1. HDF5
  2. **TB–PB Daten, aktive Datenquelle:** Kafka & Cassandra
  3. **PB Daten, gerichtete Verarbeitung:** Hadoop
  4. CSV