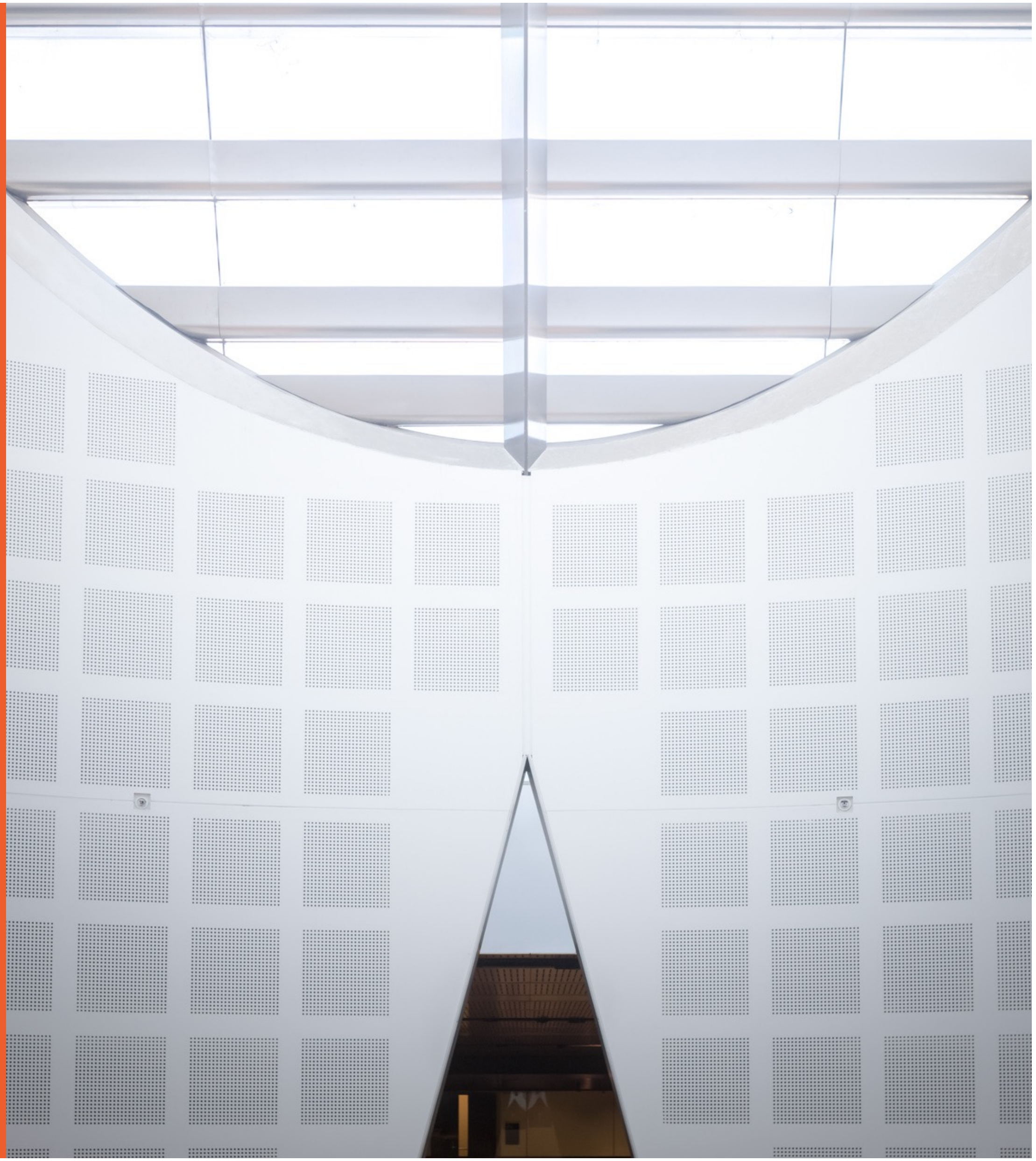
DATA2001 – Data Science, Big Data, and Data Diversity
Week 12: Big Data

Presented by A/Prof Uwe Roehm
School of Computer Science

THE UNIVERSITY OF SYDNEY



Learning Objectives

- **Big Data**
  - The three V's: Volume, Velocity and Variety
  - Ethical challenges for Big Data Processing

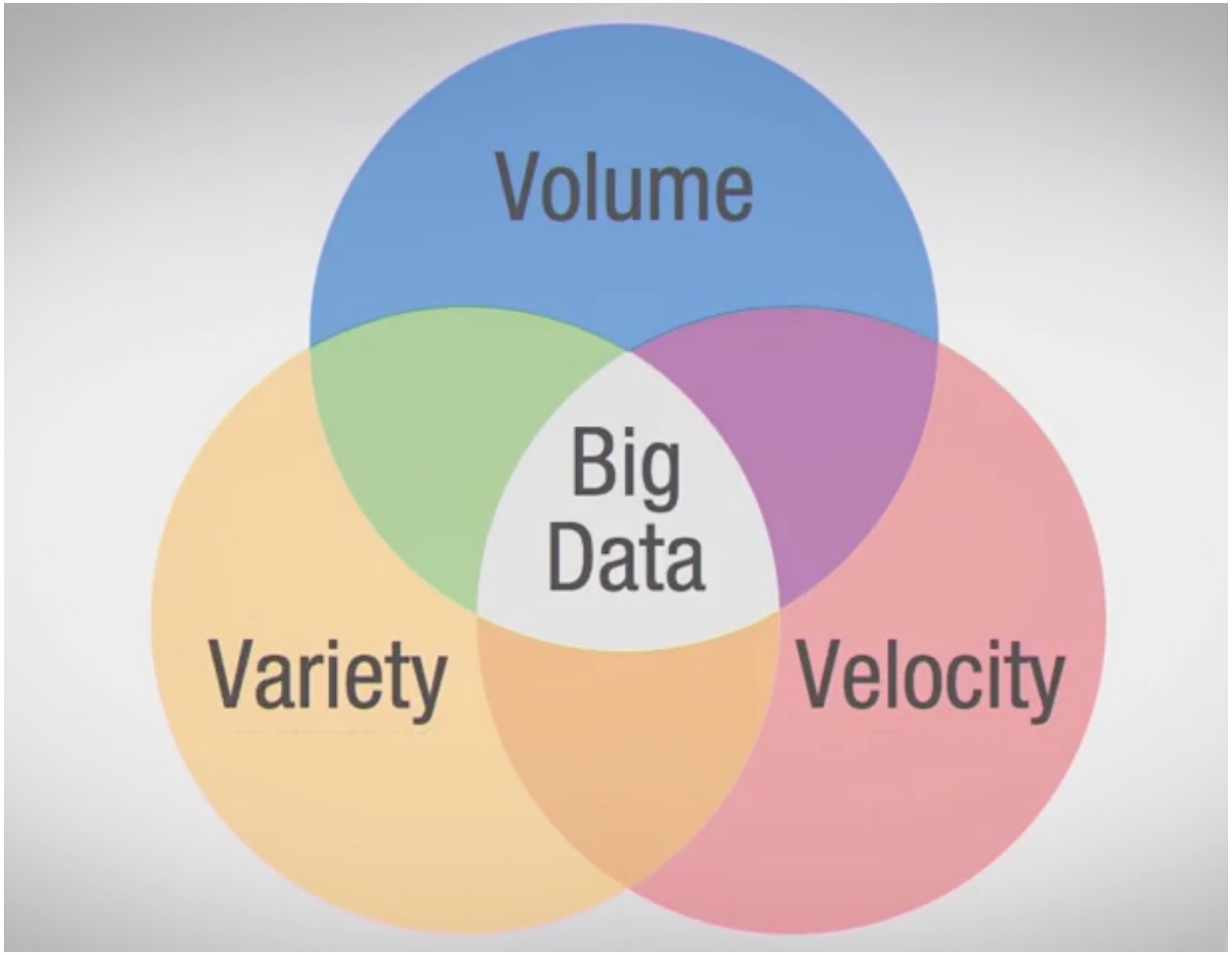
- **Scale-Agnostic Data Analytics Platforms**
  - Scale-Up vs. Scale-Out
  - MapReduce principle; similarities to SQL
  - Role of modern Big Data platforms
    MapReduce, Apache Spark, Flink or Hive

Big Data

Big Data

the three Vs:

[cf. article by
Doug Laney, 2001]



[Barton Poulson "Techniques and Concepts of Big Data", 2014]

Big Data: Volume

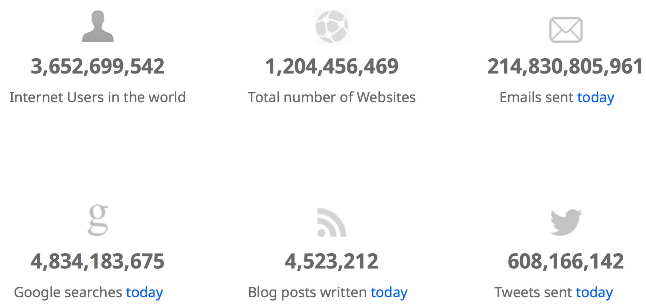
- very relative due to Moore's Law
  - What once was considered big data, is considered a main-memory problem nowadays
  - eg. Excel: In 2003 max 65000 rows, now max 1 million rows, still ...

- Nowadays: Terabyte to Exabyte

Big Data: Velocity

- conventional scientific research:
  - months to gather data from 100s cases, weeks to analyze the data and years to publish.
  - Example: Iris flower data set by Edgar Anderson and Ronal Fisher from 1936

- on the other end of the scale: Twitter
  - average 6000 tweets/sec, 500 million per day or 200 billion per year
  - Cf. life Twitter Usage Statistics http://www.internetlivestats.com/twitter-statistics/



Big Data: Variety

- Structured Data, such as CSV or RDBMS
- Semi-structured Data, such as JSON or XML
- Unstructured Data, ie. text, e-mails, images, video
  - an estimated 80% of enterprise data is unstructured

- study by Forester Research: **variety biggest challenge in Big Data**

Big Data Examples

Big Data for Consumers (examples)

- Siri, Yelp!, Spotify, Amazon, Netflix, Google Now
- Some Big Data Variety examples:
  - "Neighborland" App [https://neighborland.com]
  - "WalkScore.com" [https://www.walkscore.com]

Big Data for Businesses (examples)

- Google Ads Searches
- Predictive Marketing
  - Example "EDITED.com": predicting fashion trends
- Fraud Detection

Big Data Examples: Big Data for Research

- Astronomy: Sloan Digital Sky Survey (SDDS) SkyServer
- Cern's Large Hadron Collider (LHC)
- The Human Brain Project
- Personalities in the United States
  (cf Journal of American Psychological Association)
- Google Flu trends (only historic data; stopped publishing new trends)
- Apple COVID19 Mobility Trends (https://www.apple.com/covid19/mobility)
- Google Books project
  - (eg. changes of word usage over time (eg. maths vs arithmetic vs algebra) https://books.google.com/ngrams/graph?content=math,arithmetic,algebra&case_insensitive=off&year_start=1800)

Big Data Challenges beyond Technical Aspects

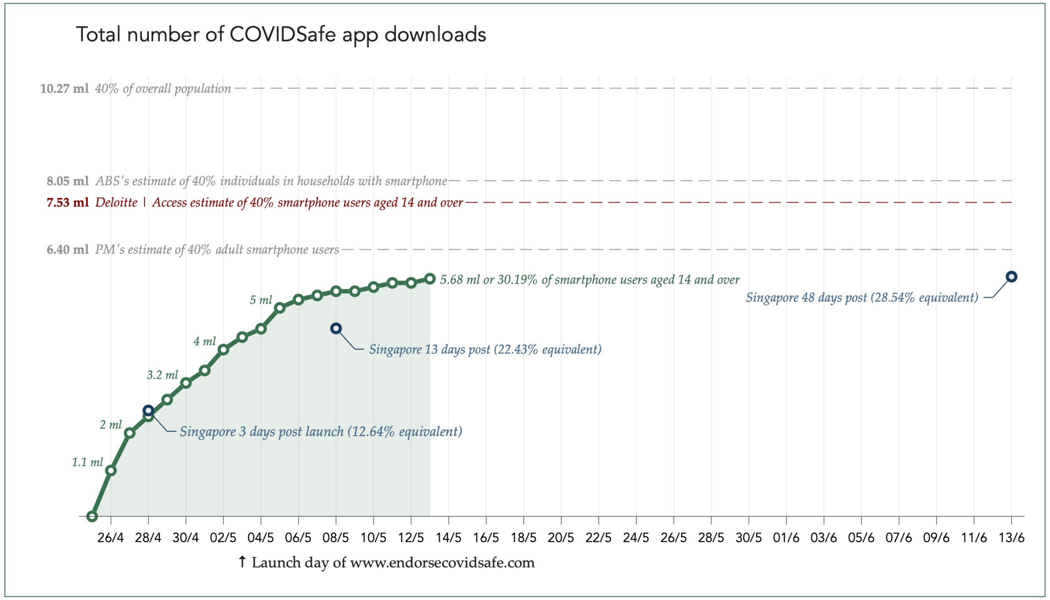
"[…] consider that great responsibility follows inseparably from great power" [French National Convention,1793]

- **Data Privacy**
  - Some data sources, such as "Internet-of-Things", allow tracking anyone
    - Do you really need to know *who* was travelling a route in order to predict, e.g., traffic densities?
    - Personal data can be inferred sometimes => New York Taxi data set example
  - Privacy laws
    - Always check: Are you allowed to use some data or process is anywhere?
    - Some personal data, especially regarding health or tax, is specially protected; e.g., not allowed to leave a jurisdictive area
    - e.g. EU's General Data Protection Regulation (GDPR) applies to any company holding data about any European Union citizen
- **Data Security**
  - Can your users trust you to keep their data safe?
  - Big data can expose your organization to serious privacy and security attacks!

Use Case: COVIDSafe App

https://www.health.gov.au/resources/apps-and-tools/covidsafe-app

- Tool to help contact tracing – who was in close contact to a known COVID-19 case?
- The app does not collect location data, but just events of being in close proximity of another COVIDSafe app user (via BT)
- Data is **stored encrypted** locally on the phone **for 21 days**, then overwritten.
  - Data only uploaded to *cloud* (AWS…) on request after **personal permission**
- Benefit to society vs. Privacy concerns
  - Which data collected and how stored?
    - Locally: **anonymised close contacts** (date, time, distance, duration, and other user's refcode); cloud: meta-data (refcode; phone#, nickname, age range, postcode)
  - Where is data processed? => cloud, resp. by contact tracers
  - Who has access to this data? => only contract tracers; protected by **Biosecurity Privacy Laws**
  - Does it work? False positives/negatives are possible => risk of false sense of security…



Big Data Challenges beyond Technical Aspects (cont'd)

- **Data Discrimination**
  - Is it acceptable to discriminate against people based on data on their lives?
  - Credit card scoring? Health insurance?
  - Cf. FTC: "Big Data – A Tool for Inclusion or Exclusion?" [https://www.ftc.gov/system/files/documents/reports/big-data-tool-inclusion-or-exclusion-understanding-issues/160106big-data-rpt.pdf]
- Check:
  - Are you working on a representative sample of users/consumers?
  - Do your algorithms prioritize fairness? Aware of the biases in the data?
  - Check your Big Data outcomes against traditionally applied statistics practices
- Keep in mind other Vs of Big Data:
  - **Validity** (data quality), **Veracity** (data accuracy / trustworthiness), …

Big Data Examples

Customer

- Twitter Life Statistics: http://www.internetlivestats.com/twitter-statistics/
- Walkscore: https://www.walkscore.com
- Neighborland: https://neighborland.com

Business

- Predictive Marketing: EDITED.com

Journalism

- TimesMachine: http://timesmachine.nytimes.com/browser
- Panama Papers: http://panamapapers.sueddeutsche.de/en/

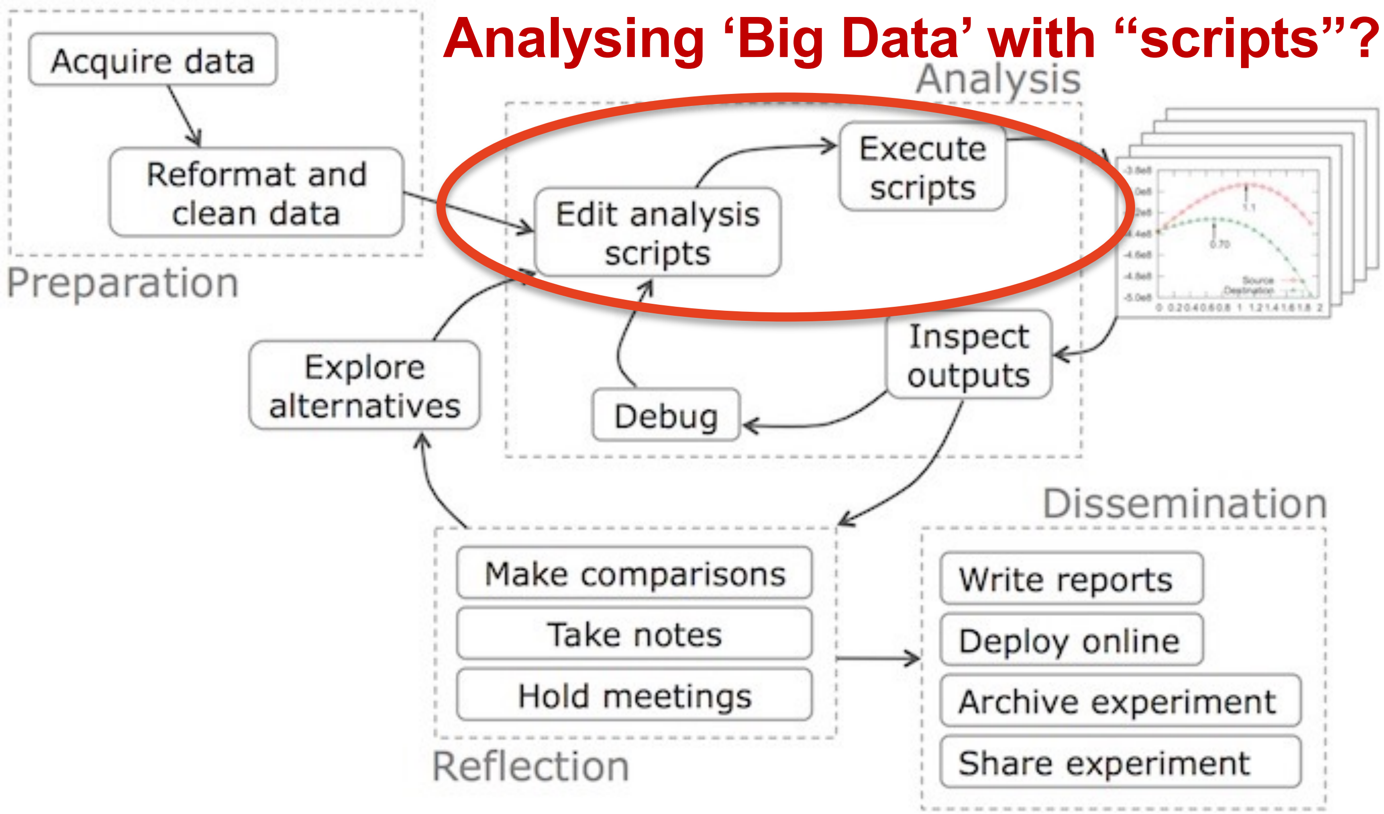
Research

- Cern LHC open data access: http://opendata.cern.ch/?ln=en
- SDDS SkyServer: http://skyserver.sdss.org/dr12/
- Human Brain Project: https://www.humanbrainproject.eu
- Google Flu Trends: https://www.google.org/flutrends/about/
- Google Books nGrams: https://books.google.com/ngrams/

Analysing Big Data

THE UNIVERSITY OF SYDNEY

Data Science Workflow



[Source: http://cacm.acm.org/blogs/blog-cacm/169199-data-science-workflow-overview-and-challenges/]

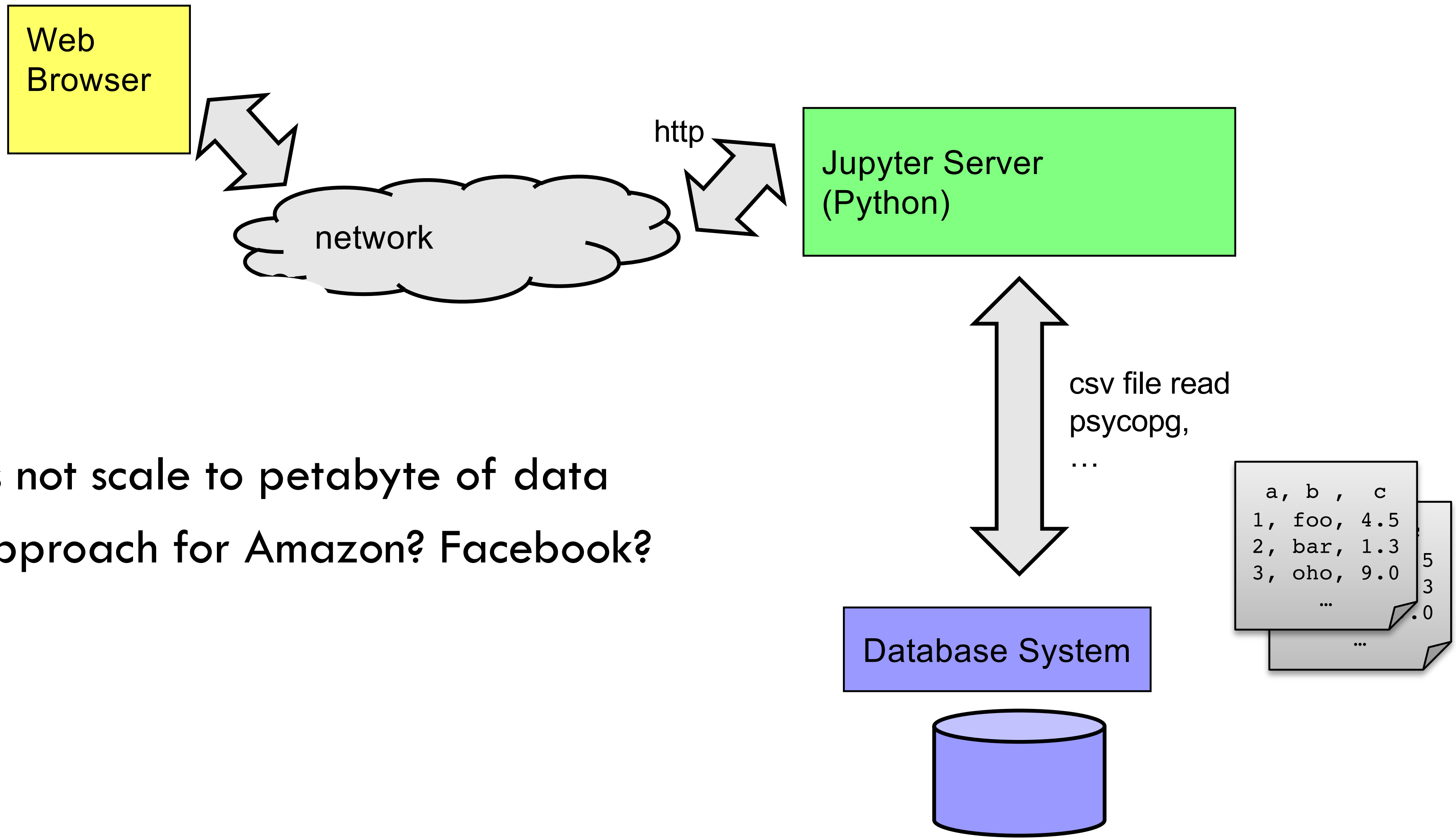
Case for Data Science Platforms

- Data is either
  - too large (volume),
  - too fast (velocity), or
  - needs to be combined from diverse sources (variety)

  for processing with scripts or on single server.

- Need for
  - scalable platform
  - processing abstractions

Jupyter Notebooks as Platform for Big Data?

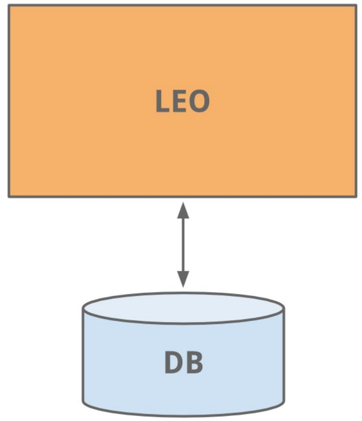


- This does not scale to petabyte of data
- Which approach for Amazon? Facebook?

Case Study: LinkedIn

Source: https://engineering.linkedin.com/architecture/brief-history-scaling-linkedin

- Started in 2003
  - 2700 members in first week
  - Single database and web server

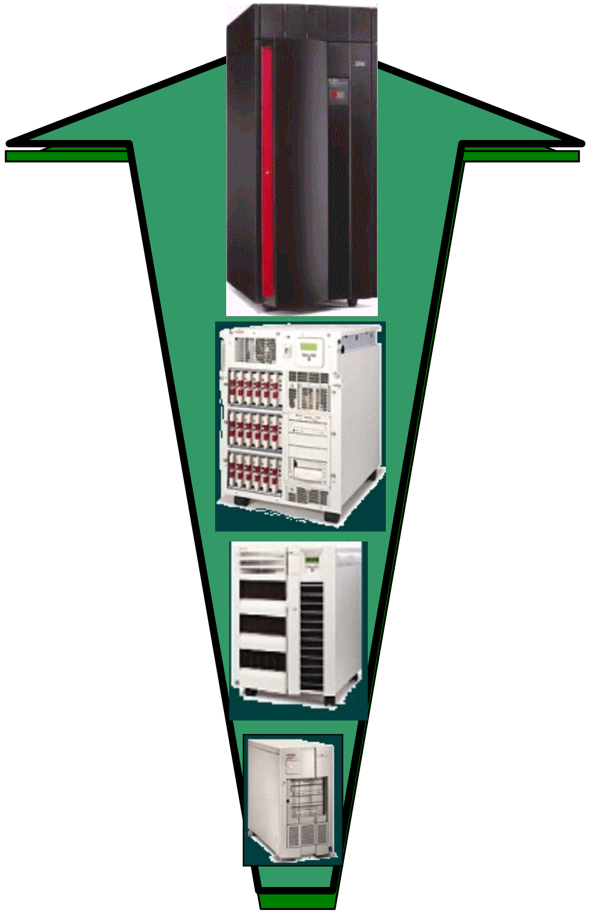


- for years experienced exponential growth…
- As of Jan 2018:
  (https://www.omnicoreagency.com/linkedin-statistics/)
  - 500 million members
  - 250 million active users / month
  - Many users with hundreds of connections => huge graph
  - Fun Fact: Statistical Analysis and Data Mining are Top skills on Linkedin

  - world's 34th-most-popular website in terms of overall visitor traffic (Alexa, Dec-16)
    (https://www.alexa.com/topsites)
    - For comparison: Microsoft is #37

Scale-Up

- The traditional approach:
  - To scale with increasing load, buy more powerful, larger hardware
    - from single workstation
    - to dedicated db server
    - to large massive-parallel database appliance

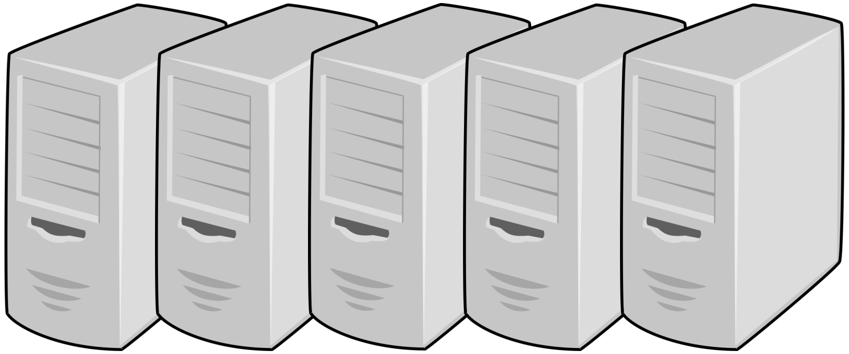


[source: Jim Gray, HPTS99]

The Alternative: Scale-Out

[recall Wk5]

A single server has limits…
For real Big Data processing, need to
scale-out to a cluster of multiple servers (nodes):



[Source: Server.png from PinClipart.com]

State-of-the-Art:

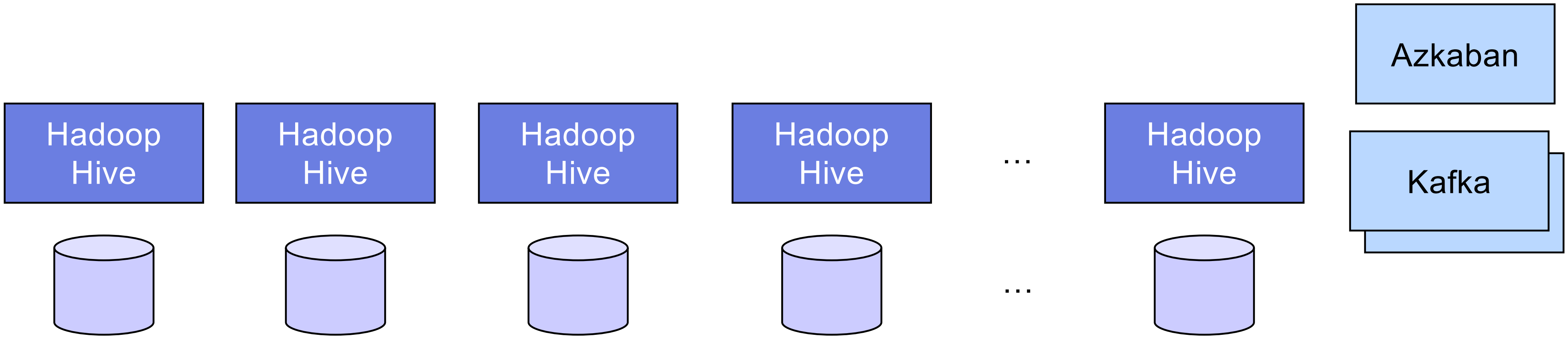
shared-nothing architecture

Case Study: LinkedIn Analytical Architecture

"We have multiple grids divided up based upon purpose.
Hardware:
- ***~800*** *Westmere-based HP SL 170x, with 2x4 cores, 24GB RAM, 6x2TB SATA*
- ***~1900*** *Westmere-based SuperMicro X8DTT-H, with 2x6 cores, 24GB RAM, 6x2TB SATA*
- ***~1400*** *Sandy Bridge-based SuperMicro with 2x6 cores, 32GB RAM, 6x2TB SATA*
- *…*

We use these things for discovering People You May Know and other fun facts."

LinkdIn via https://wiki.apache.org/hadoop/PoweredBy/



Challenges

- **Scale-Agnostic Data Management**
  - **sharding** for performance
  - **replication** for availability
  - ideally such that applications are unaware of underlying complexities
  - cf. Week 5
- **Scale-Agnostic Data Processing**
  - Nowadays we collect massive amounts of data; how can we analyze it?
    - Answer: use lots of machines… (hundreds/thousands of CPUs, can grow)
  - Performance: parallel processing
  - Availability: Ideally, the system never down; can handle failures transparent

  => Map/Reduce processing paradigm

Scale-Agnostic Data Analysis

The MapReduce Principle

THE UNIVERSITY OF SYDNEY

Big Data Analytics Stack

- Layered stack of frameworks for distributed data management and processing
- Many choices of distributed data processing platforms

Application

Data Processing

Storage

Infrastructure

[slide by Ion Stoica, UCB, 2013]

MapReduce Overview

- Scan large volumes of data
- **Map:** Extract some interesting information
- Shuffle and sort intermediate results
- **Reduce:** aggregate intermediate results
- Generate final output

- Key idea: provide an abstraction at the point of these two operations (map and reduce)
  - Higher-order functions
  - Cf. map functions in functional programming languages such as Lisp or Haskell

MapReduce Paradigm

- Functional Programming approach to data processing
  - *map*() : applies a given function *f* to all elements of a collection; returns a new collection
  - map (f, originalList)

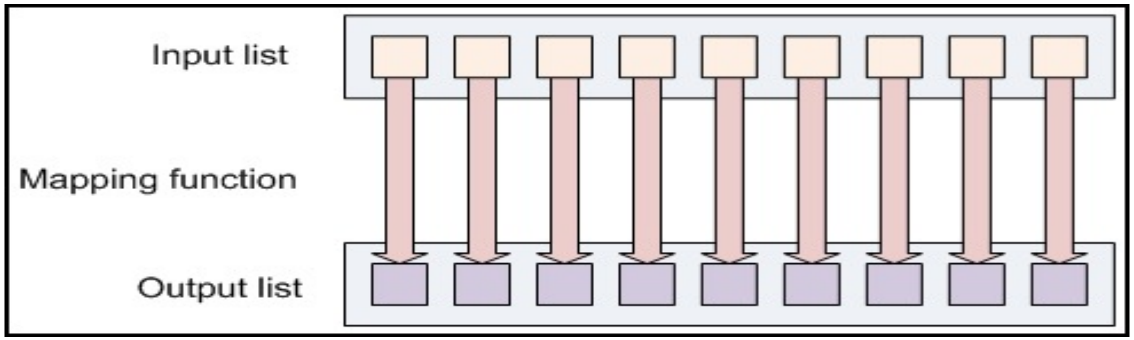


Diagram from Yahoo! Hadoop Tutorial

MapReduce Paradigm: Reduce()

***reduce*()**: applies a given function *g* to all elements of an input list;
produces, starting from a given initial value, a single (aggregate) output value

Keys divide the reduce space
all of the output values are not usually reduced together. All of the values *with the same key* are presented to a single reducer together

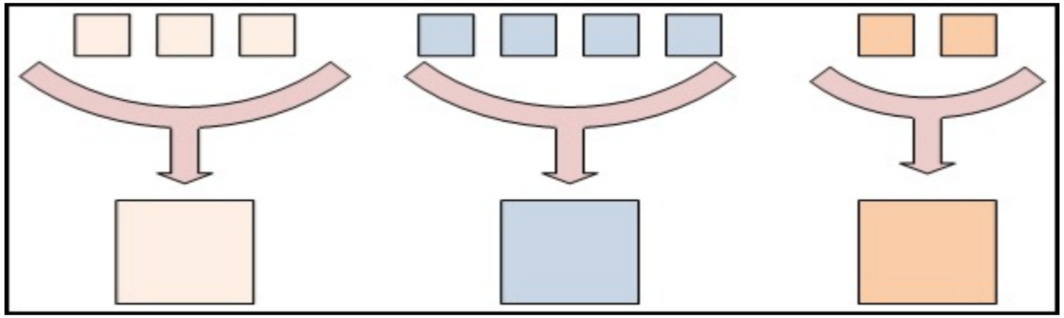


Diagram from Yahoo! Hadoop Tutorial

Similarities between SQL-Queries and MapReduce

- A standard map-reduce task is similar in its functionality to declarative aggregation queries in SQL:

```
SELECT out_key, reduce(out_value)
  FROM map(InputData)
GROUP BY out_key
```

New in MR-Paradigm: map and reduce() as **higher-order functions** which take a user-defined function with the actual functionality.

Example: Word Count program

- Word Count programmed as standard linear program
  - Two nested for loops
  - Difficult to generalise or parallelise

```
from collections import Counter

def word_count_old(documents):
    """word count not using MapReduce"""
    return Counter(word
        for document in documents
            for word in tokenize(document))
```

Example: Word Count

- Input:
  - List of documents that contain text
  - Provided to MapReduce in the form of
    (k: documentID, v: textcontent) pairs

- Goal:
  - Determine which words occur in the documents and how often
  - E.g. for text indexing…

MapReduce Approach

To solve the same problem using MapReduce, we need

1. *map()* function (aka *mapper*)

2. *reduce()* function (aka *reducer*)

3. Some control code that connects mapper and reducers

Example: Word Count in MapReduce

- Word Count programmed using Map/Reduce paradigm

```python
def wc_mapper(document):
    """for each word in the document, emit (word,1)"""
    for word in tokenize(document):
        yield (word, 1)
```

```python
def wc_reducer(word, counts):
    """sum up the counts for a word"""
    yield (word, sum(counts))
```

```python
def word_count(documents):
    """count the words in the input documents using MapReduce"""

    # place to store grouped values
    collector = defaultdict(list)

    for document in documents:
        for word, count in wc_mapper(document):
            collector[word].append(count)

    return [output
            for word, counts in collector.iteritems()
            for output in wc_reducer(word, counts)]
```

MapReduce Generalised

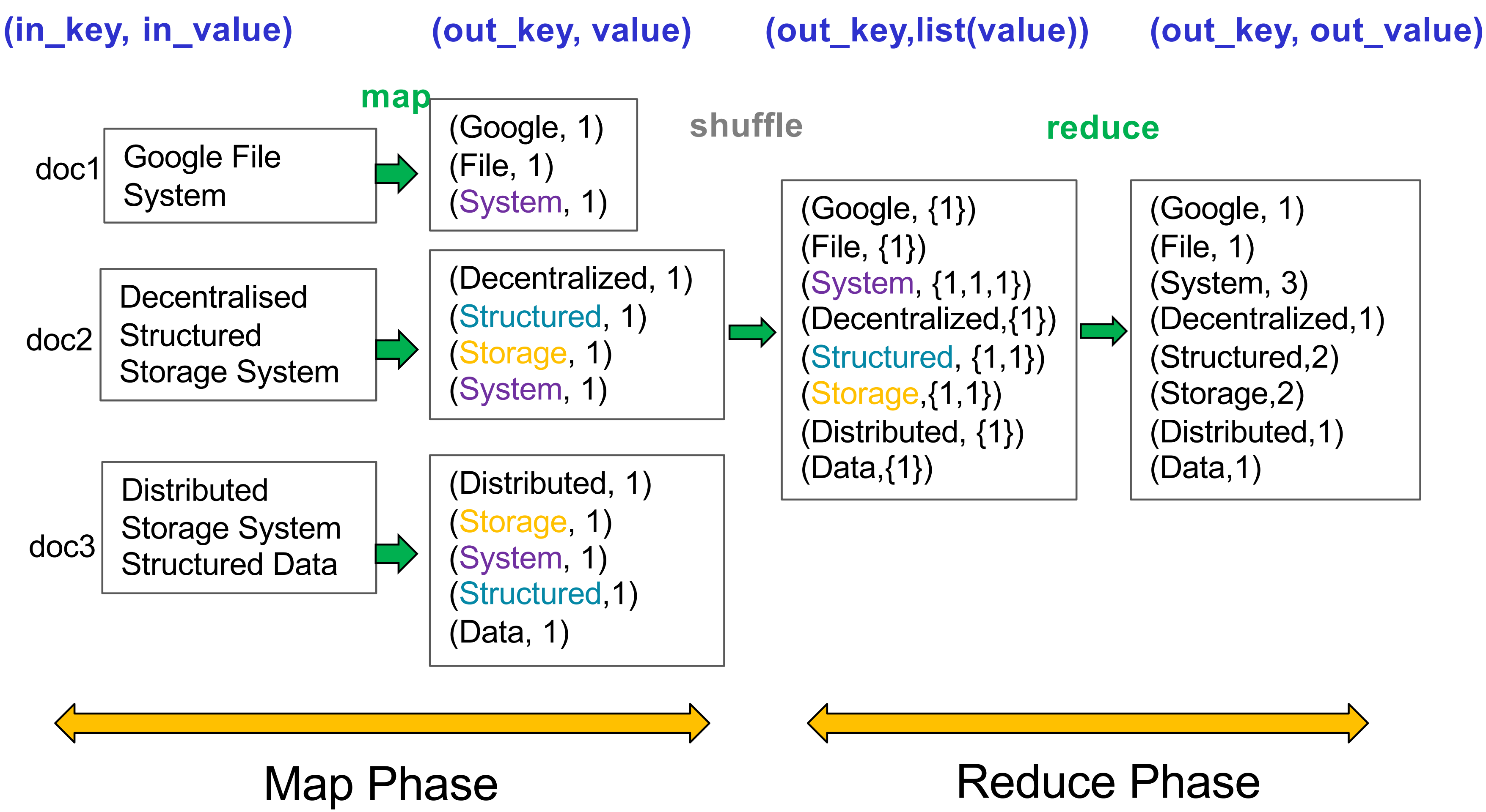
- Previous example was hard-coded for word count
- We can generalise the pattern for the *driver* code even further

mapper and *reducer* are now also inputs

```
def map_reduce(inputs, mapper, reducer):
    """runs MapReduce on the inputs using mapper and reducer"""
    collector = defaultdict(list)

    for input in inputs:
        for key, value in mapper(input):
            collector[key].append(value)

    return [output
            for key, values in collector.items()
            for output in reducer(key,values)]

word_counts = map_reduce(documents, wc_mapper, wc_reducer)
```

Call to function needs 3 arguments: data, *mapper* and *reducer*

Example Word Count with MapReduce



Why Scale-Agnostic?

- Note that the functions given to map() and reduce() only rely on local input
  - functions without side-effects and independent of each other
  - function invocation is *agnostic* to the scale (size) of the overall dataset

- Can hence be parallelized easily
  - Partition the dataset over multiple nodes
  - apply different instances of the same map/reduce functions to each partition independently / in parallel

- Fits perfectly to a scale-out approach
  - bigger data => more nodes and data partitions => more parallelism
=> same or faster speed

MapReduce Discussion

Pros:

- very **flexible** due to the user-defined functions
- great **scalability** because FP approach
- easy **parallelism** due to stateless functions
- **fault-tolerance**

Cons:

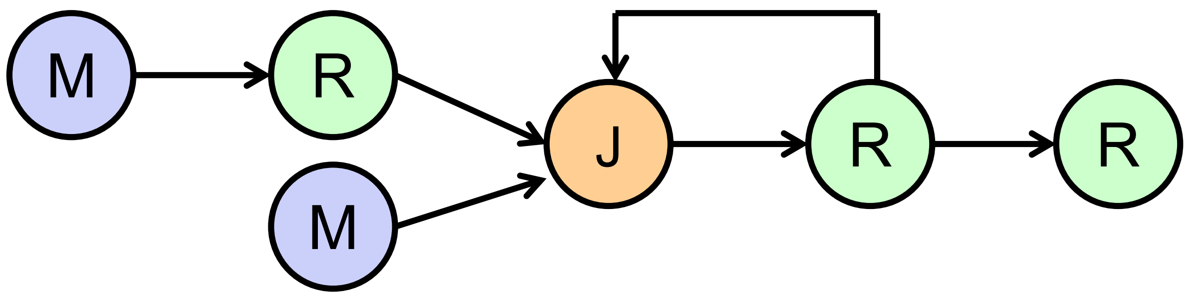
- requires **programming** skills and functional thinking
- relatively **low-level**, even filtering to be coded manually
  - **complex** frameworks
- **batch-processing** oriented

Distributed, Dataflow-Oriented Analytics Platforms

THE UNIVERSITY OF SYDNEY

Challenge: Iterative Algorithms

- Many **data mining** and **machine learning** algorithms rely on global state and iterations
- Examples:
  - data clustering (eg. *k*-Means)
  - frequent itemset mining (eg. Apriori algorithm)
  - linear regression
  - collaborative filtering
  - PageRank
  - …



Distributed Data Analytics Frameworks

- **Apache Hadoop**
  - Open-source implementation of original MapReduce from Google; Apache top-level project
  - Java framework, but also provides a Python interface nowadays
  - Parts: own distributed file system (HDFS), job scheduler (YARN), MR framework (Hadoop)

- **Apache Spark**
  - Distributed cluster computing framework on top of HDFS/YARN
  - Concentrates on **main-memory** processing and more **high-level data flow control**
  - Originates from research project from UC Berkeley

- **Apache Flink**
  - Efficient data flow runtime on top of HDFS/YARN
  - Similar to Spark, but more emphasize on **build-in dataflow optimiser** and **pipelined processing**
  - Strong for data stream processing
  - Origin: Stratosphere research project by TU Berlin, Humboldt University Berlin and HPI Potsdam

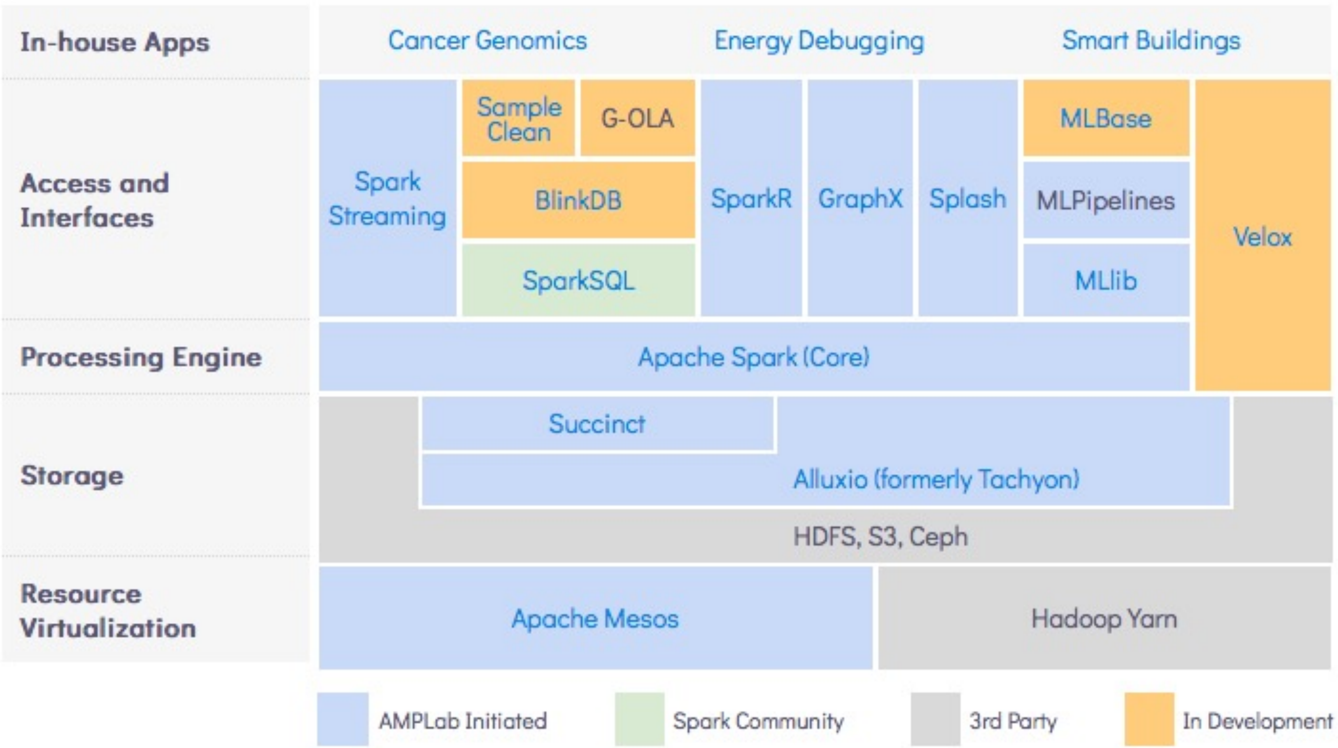
Distributed Data Analytics Frameworks (continued)

- **Apache Hive**
  - Provides an SQL-like interface on top of Hadoop / HDFS
  - Allows to define a relational schema on top of HDFS files, and to query and analyse data with HiveQL (SQL dialect)
  - Queries automatically translated to MR jobs and executed in parallel in cluster
  - Example: WordCount in HIVE

```
DROP TABLE IF EXISTS docs;
CREATE TABLE docs (line STRING);
LOAD DATA INPATH 'input_file' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
       SELECT word, count(1) AS count
         FROM (SELECT explode(split(line, '\s')) AS word FROM docs) temp
        GROUP BY word
        ORDER BY word;
```

- **Many more high-level frameworks for advanced data analytics.**

Example Dataflow Execution Stack: Apache Spark



[Apache Spark Architecture, Apache website]

Example: WordCount in Apache Flink (Python)

```
from flink.plan.Environment import get_environment
from flink.functions.GroupReduceFunction import GroupReduceFunction

env = get_environment()
data= env.read_text("hdfs://…");

data.flat_map(lambda x,c: [(word,1) for word in x.lower().split()]) \
    .group_by(0)                                                     \
    .sum(1)                                                          \
    .write_csv("hdfs://…")

env.execute()
```

[Cf.: https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/batch/python.html]

Summary

- **Big Data**
  - The three V's: Volume, Velocity and Variety
  - Ethical challenges for Big Data Processing
  - Scale-Up versus Scale-Out
- **Scale-Agnostic Computation**
  - Parallelisable higher-order functions map & reduce
  - MapReduce principle; similarities to existing material and SQL
- **Scale-Agnostic Data Analytics Platforms**
  - Data Scientists need more high-level tools and interfaces than MapReduce
  - Examples: **Apache Spark** or **Apache Flink** or **Apache Hive**
  - Componentized infrastructure: SQL querying, ML-Libraries, Streaming, etc.

Learn More

- DATA3404 Data Science Platforms