Distributed Data Management
Foundations

Thorsten Papenbrock

F-2.04, Campus II

Hasso Plattner Institut

Overview

Foundations

- Big Data
- Big Data Analytics
- Distributed Computing
- Data-Intensive Applications
- Consistency Models

Overview

Foundations

- **Big Data**
- Big Data Analytics
- Distributed Computing
- Data-Intensive Applications
- Consistency Models

Definition

Big data is a term that applies to the growing availability of large datasets in information technology. Big data analytics is …

used to refer to the *study and applications of* big

- Big data is a term ~~for~~ data sets that are so ~~large~~ or complex that traditional ~~database management tools or data processing software~~ data-processing application software are inadequate to deal with them.

(Wikipedia ~~2017~~ ~~2018~~ 2021)

- The challenges include data …
  - capturing
  - storage
  - extraction
  - curation
  - analysis
  - search
  - sharing
  - transfer
  - visualization
  - querying
  - updating
  - privacy

If data is so **large**, **fast** or **hard** that processing it in a specific way is a challenge for existing software or hardware, then it is Big Data.

Context matters!

Properties of Big Data – Gartner’s 3 V’s

Volume

- 12 terabytes of Tweets (calculate sentiment analysis)
- 350 billion annual meter readings (predict power consumption)

Velocity

- 5 million daily trade events (identify potential fraud)
- 500 million daily call detail records (predict customer churn faster)

Variety

- 100’s of live video feeds from surveillance cameras (find persons)
- 80% data growth in images, videos and documents
  (improve customer satisfaction)

Gartner’s 3 V’s: M. Beyer: Gartner Says Solving „Big Data“ Challenge Involves More Than Just Managing Volumes of Data, www.gartner.com/it/page.jsp

Examples for V’s: www.ibm.com/software/data/bigdata

Properties of Big Data – More V's

Veracity (Wahrhaftigkeit)

- Trust in correctness and completeness of the data

Viscosity

- Integration and dataflow friction

Venue

- Different locations that require different access & extraction methods

Vocabulary

- Different language and vocabulary

Value

- Added-value of data to organization and use-case

Virality

- Speed of dispersal among community

Variability

- Data, formats, schema, semantics change

Big vs. Large

Big Data can be very small:

- Example: streaming data from aircraft sensors
  - A sensor produces an eight byte reading every second (8 byte/sec)
  - Hundred thousand sensors on an aircraft
  - About 2.7 GB of data in an hour of flying
    (100,000 sensors x 60 min/hour x 60 sec/min x 8 bytes/sec)
  - Difficult to process due to strong real-time requirements and on plane!

Not all large datasets are "big":

- Example: video streams plus metadata
  - A live TV stream sends about twenty megabyte per second (20 MB/sec)
  - About 70 GB of data in an hour of streaming
    (60 min/hour x 60 sec/min x 20 MB/sec)
  - Easy to parse and process, because content is well structured

➢ The task at hand makes data "big"

http://mike2.openmethodology.org/wiki/Big_Data_Definition

Big Data in Use – Business Data

Amazon.com

- Millions of back-end operations every day
- Catalog, searches, clicks, wish lists, shopping carts, third-party sellers, …

Walmart

- > 1 million customer transactions per hour
- 2.5 petabytes (2560 terabytes)

Facebook

- 250 PB, 600TB added daily (2013)
- 1 billion photos on one day (Halloween)

FICO Credit Card Fraud Detection

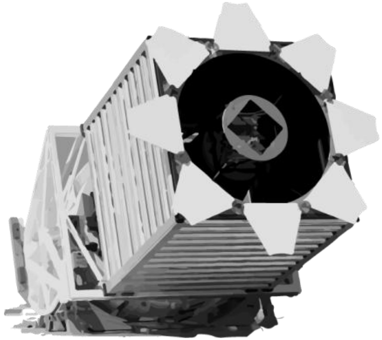
- Protects 2.1 billion active accounts

Big Data in Use – Science

Large Hadron Collider

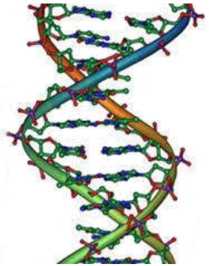
- 150 million sensors: 40 million deliveries and 600 million collisions per sec
- Theoretically: 500 exabytes per day (500 quintillion bytes)
- Filtering: 100 collisions of interest per second (→ 99.999% reduction rate)
- 200 petabytes annual rate

Sloan Digital Sky Survey (SDSS)



- Began collecting astronomical data in 2000
- 200 gigabyte per night; 140 terabytes overall
  (more data in first few weeks than all data in the history of astronomy)
- Large Synoptic Survey Telescope, successor to SDSS since 2016
  - Acquires that amount of data every five days!

Human Genome Project



- Human genome: 3,234.83 Mb
- Processing one genome originally took 10 years; now less than a day

Overview

Foundations

- Big Data
- **Big Data Analytics**
- Distributed Computing
- Data-Intensive Applications
- Consistency Models

Correlation vs. Causation

Correlation

- Correlation describes a linear statistical relationship of two random variables (or bivariate data), i.e., the values of both variables change synchronously.

Causation

- Causation describes a directed, semantic dependence of one variable (= cause) to another variable (= effect) such that a change in the first variable always causes a corresponding change in the second variable.
- Correlating variables might share the same causal variable.

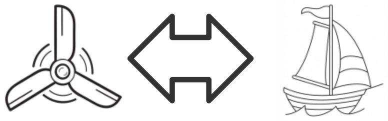
➢ Correlation ≠ Causation

Big Data Analytics

Correlation vs. Causation

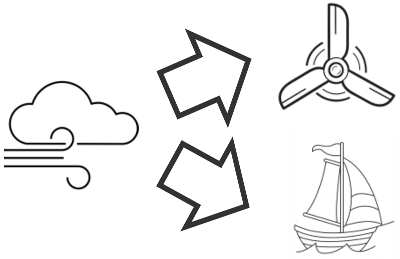
Correlation

- “energy production of wind turbines” and “top-speed of sailing boats”



Causation

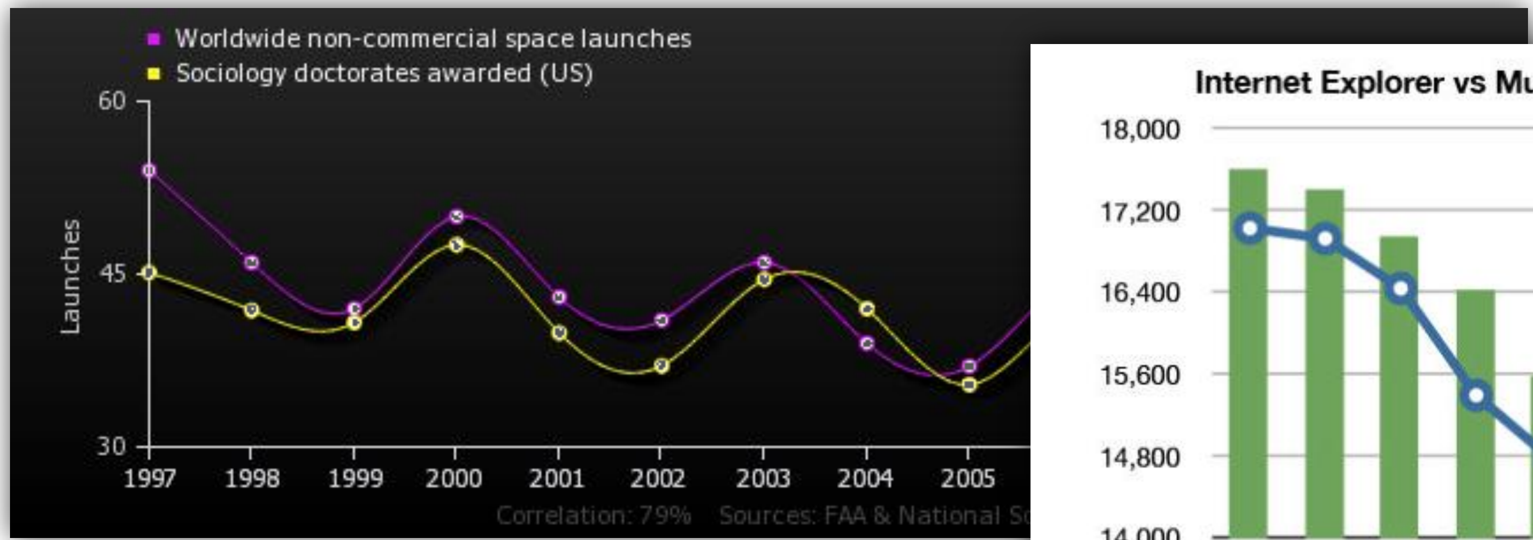
- “wind speed” causes “energy production of wind turbines”
- “wind speed” causes “top-speed of sailing boats”

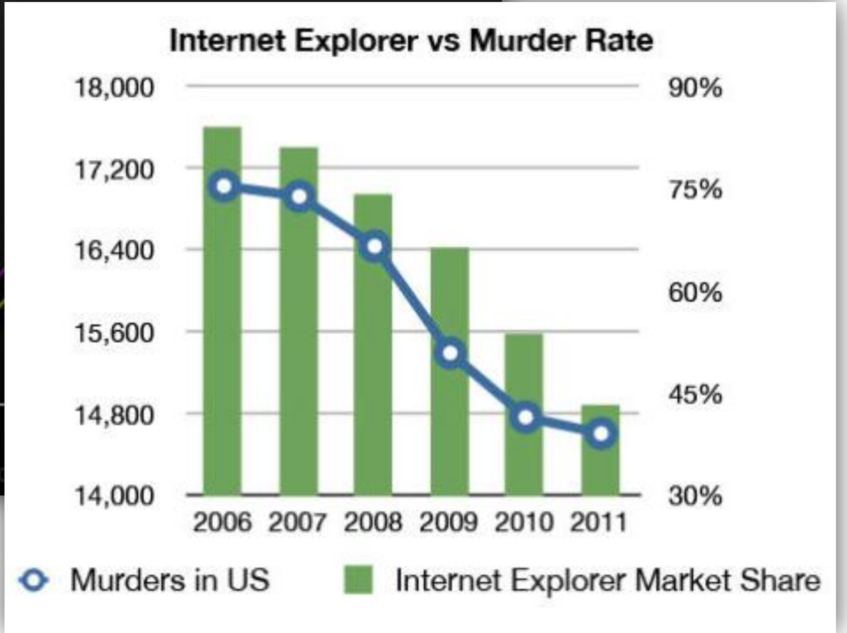


➢ Correlation ≠ Causation

Correlation vs. Causation

- Correlation ≠ Causation
  - Examples:

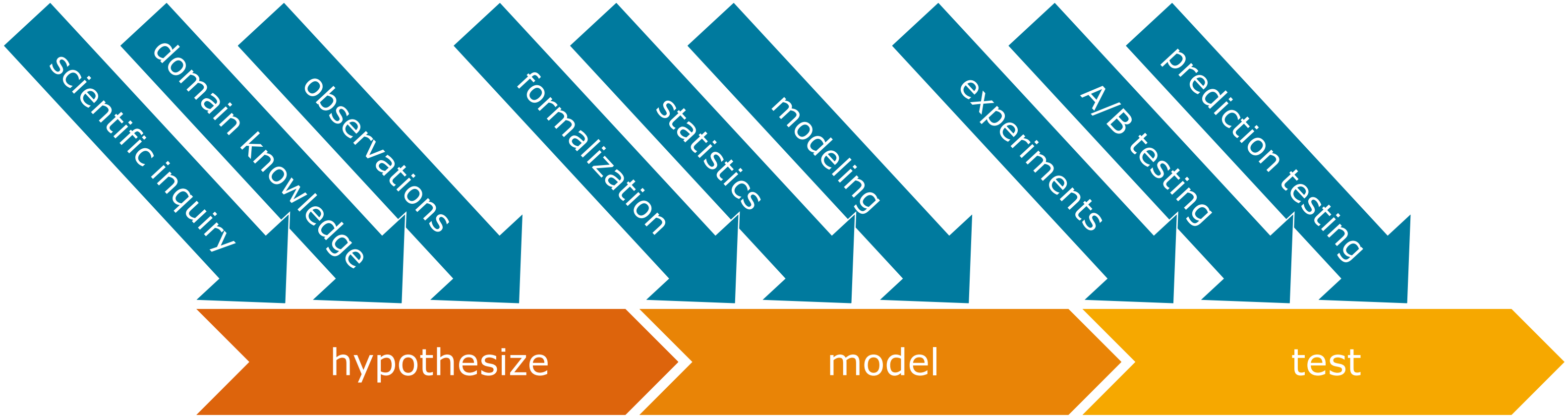




Correlation vs. Causation

Hasso Plattner Institut

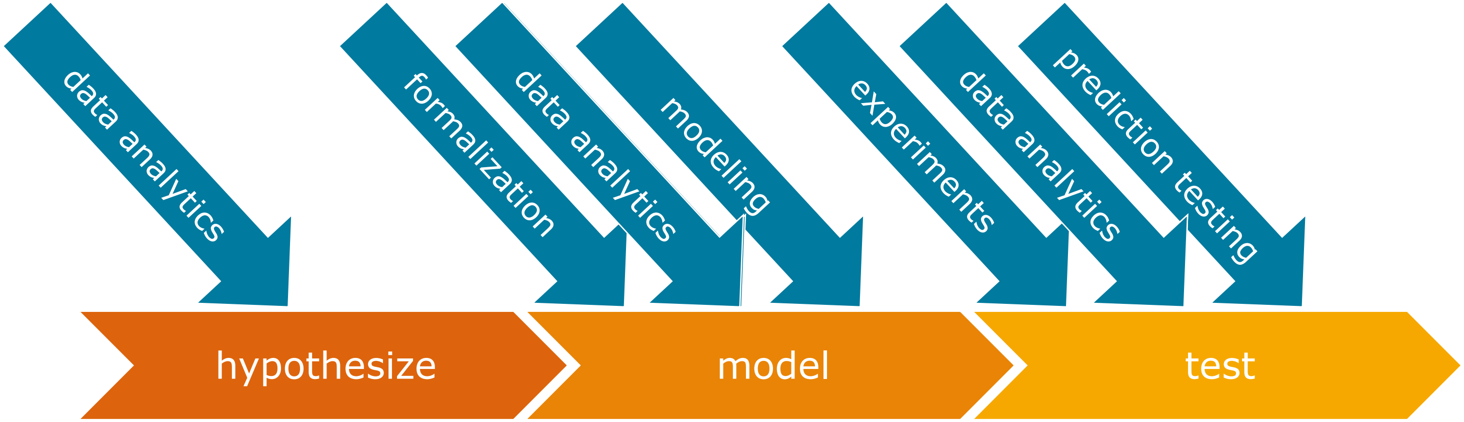
- Correlation ≠ Causation
  - Good science before Big Data:



Correlation vs. Causation

Hasso Plattner Institut

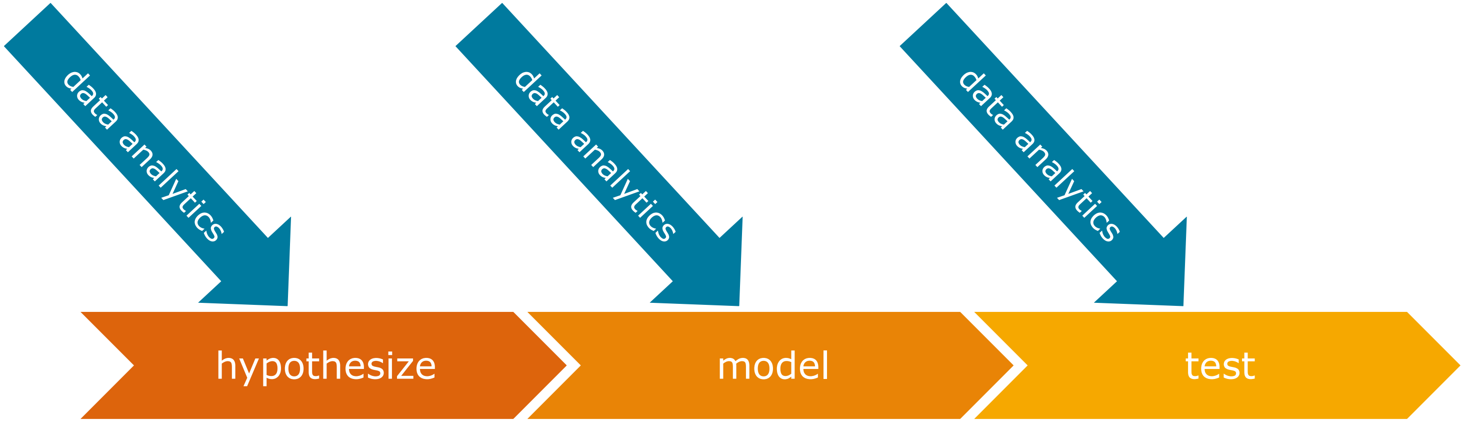
- Correlation ≠ Causation
  - Good science with Big Data:



- Hypothesizing is hard: Use discovered correlations to formulate them!
- Modeling is hard: Use automatically trained models!
- Testing is hard: Use Big Data to verify your model!

Correlation vs. Causation

- Correlation ≠ Causation
  - Good science with Big Data:



  - If correlation holds for very large data sets, it's likely a causation.
    - Big Data Analytics: find correlations → derive causations

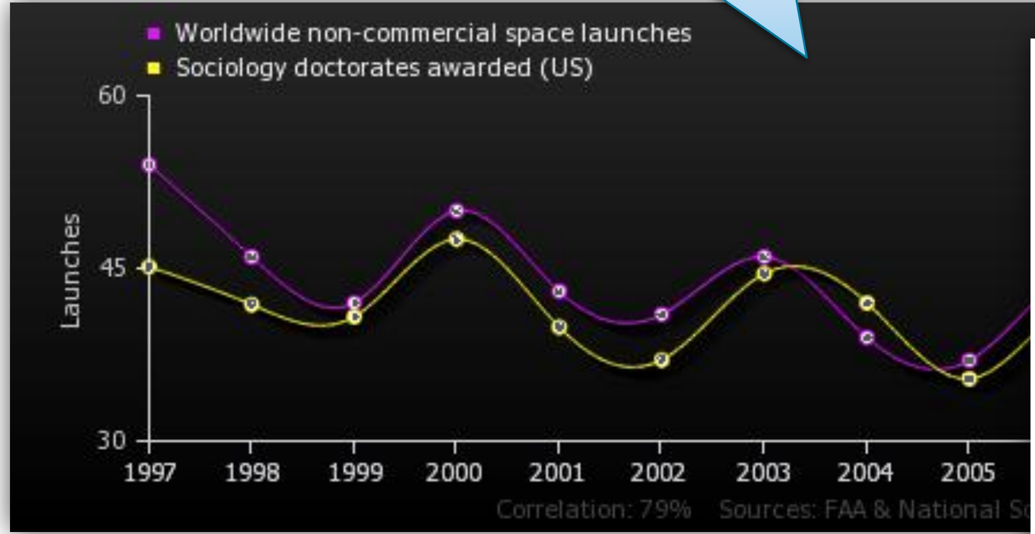
https://www.wired.com/2008/06/pb-theory

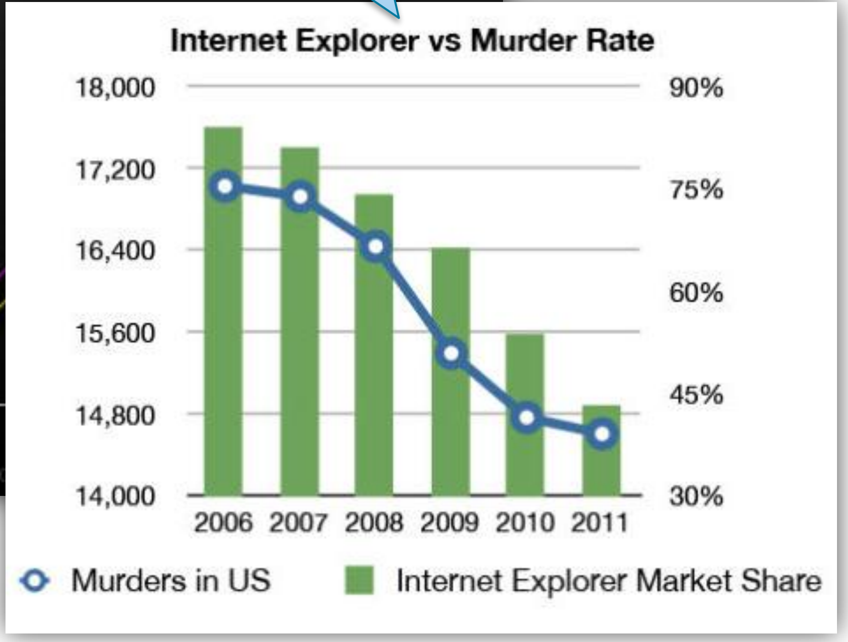
Correlation vs. Causation

- Correlation = Causation ?
  - Why did it fail here?

13 measurements

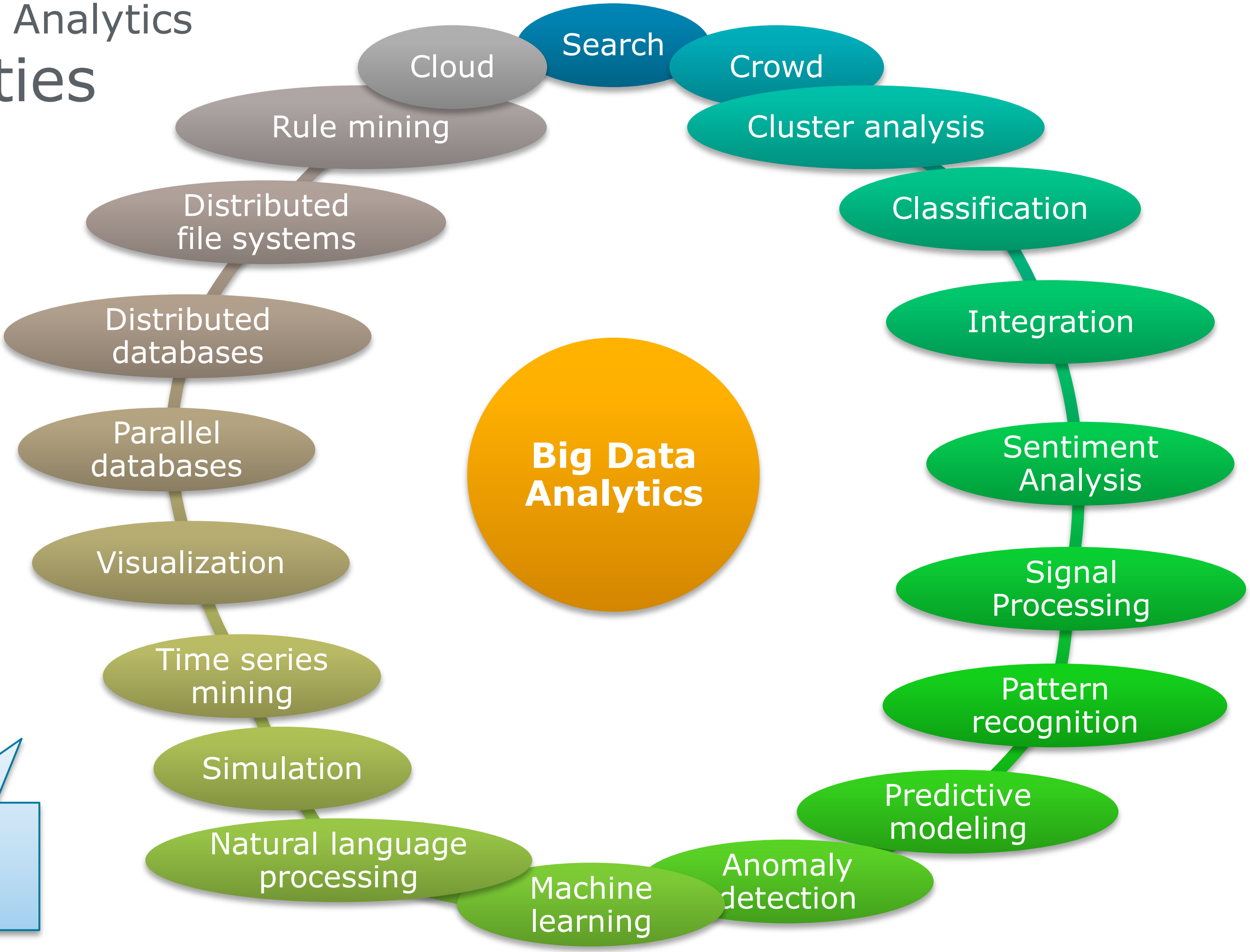
6 measurements





Big Data Analytics

Activities



And many more …

Distributed Data Management

Foundations

ThorstenPapenbrock

Overview

Foundations

- Big Data
- Big Data Analytics
- **Distributed Computing**
- Data-Intensive Applications
- Consistency Models

Distributed System

Definition 1:

"A *distributed system* is a collection of **autonomous computing elements** that appears to its users as a **single coherent system**."

(Maarten van Steen, Andrew S. Tanenbaum: "Distributed Systems")

Definition 2:

"A *distributed computing system* is a number of **autonomous processing elements** (not necessarily homogeneous) that are interconnected by a computer network and that **cooperate** in performing their assigned task."

(M. Tamer Özsu, Patrick Valduriez: "Principles of Distributed Database Systems" V3)

Definition 3:

"A *distributed system* is a system whose components are located on **different networked computers**, which **communicate and coordinate** their actions by passing messages to one another."

(Wikipedia, 2021)

Distributed System

Autonomous Computing Elements (Nodes)

- Can refer to hardware devices and/or processes (usually both, i.e., different processes on different hardware devices).
- Can be heterogeneous, i.e., have access to different quantities of resources (memory, CPU/GPU cores, CPU/GPU speeds, external devices, interconnects, …).
- Do not share state and cannot access each others state.
  - Communicate via explicit exchange of messages.
- Act independently from each other.
  - Can (in principle) run simultaneously and do not require other elements.
  - A failing element does not necessarily cause other elements to fail.

E.g. a server, a workstation and an embedded device all in the same system.

Interconnected System

- Represents a common collaboration goal and the system as an entity to the outside.
- Based technically on interconnections, such as networks, busses, …
  - Can be heterogeneous in topology and hardware.
- Based logically on communication protocols.

Distributed Computing
Distributed System

What is a distributed system?

Independent systems connected via network

Multiple, connected machines

Data in multiple caches, in memory, on disk …

Control-flow over multiple cores, CPUs, GPUs, …

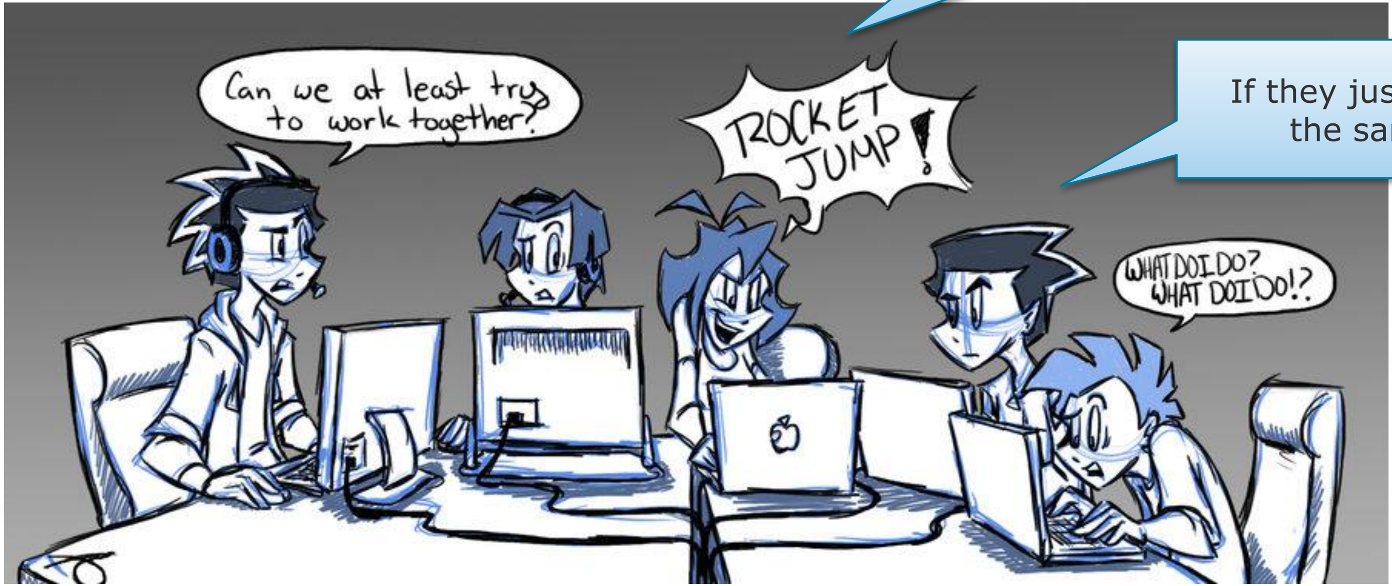
One machine

Specialized racks with shared infrastructure …

One big machine

Distributed System

What is a distributed system?



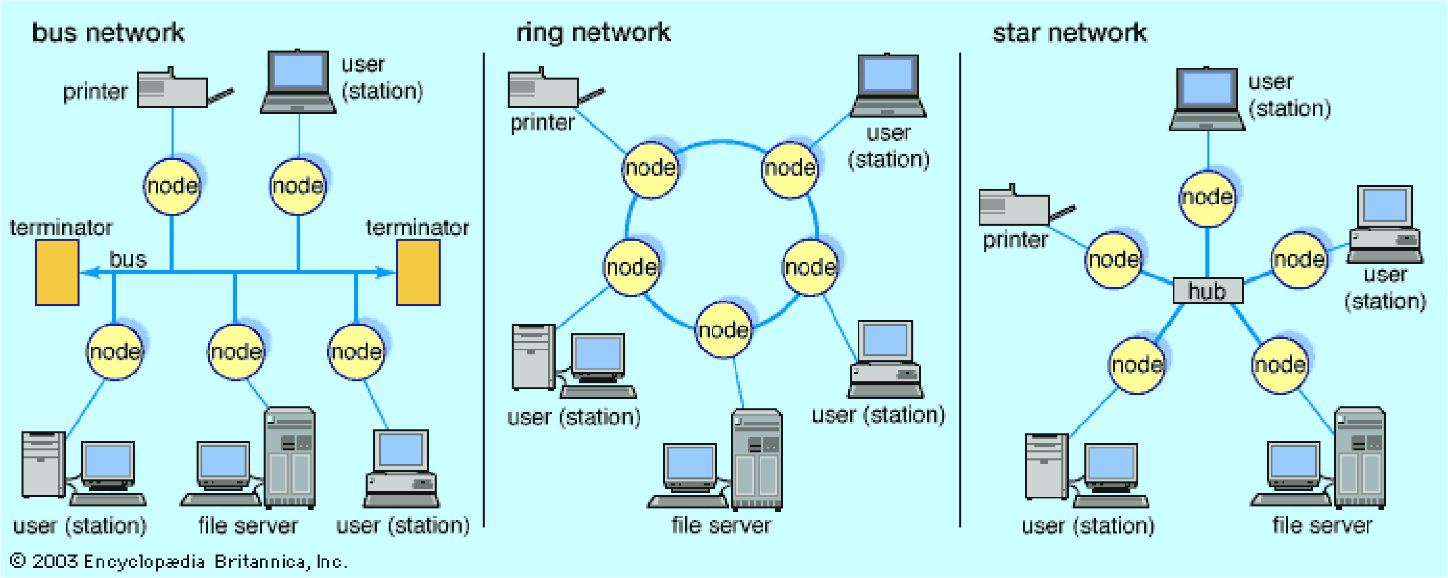
If they play the same game

If they just happen to share the same connection

A LAN party with multiple interconnected computers

Interconnects

- Various forms of messaging-based interconnection architectures
- Popular architectures are:

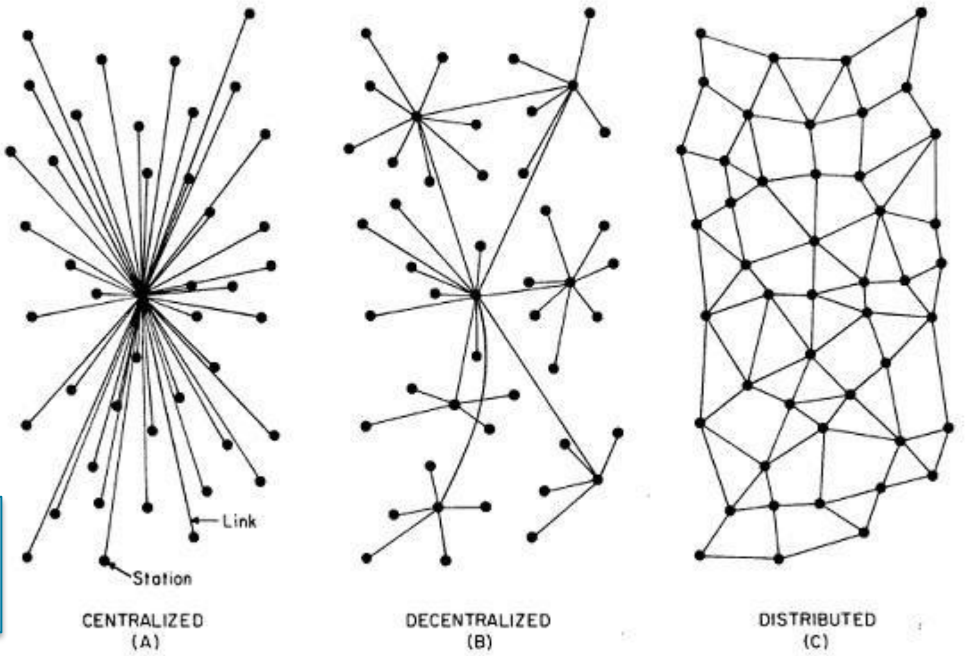


- One distributed system can use combinations of such architectures.

Interconnects

An outdated topological definition:

"A *distributed computing system* is a (fully) decentralized network of computing elements/stations, i.e., one that has multiple roots."



peer-to-peer systems

single-client or single-master systems

Distributed Computing

Interconnects

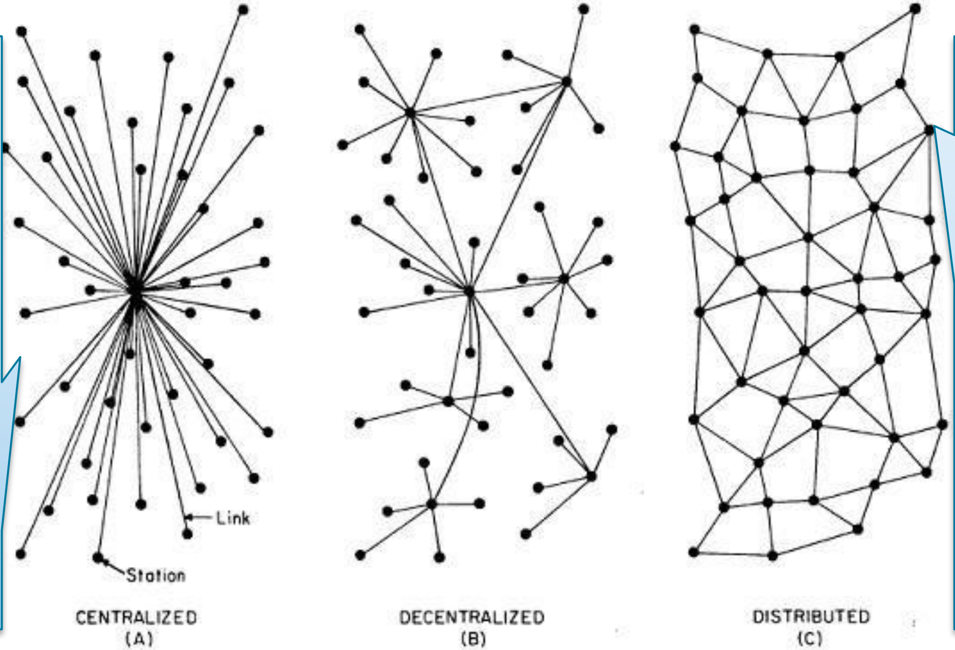
Hasso Plattner Institut

An outdated topological definition:

"A *distributed computing system* is a (fully) decentralized network of computing elements/stations, i.e., one that has multiple roots."

System examples:

- Weather stations and their central control station
- Human workers and the central MTurk web service in Amazon Mechanical Turk



System examples:

- BitTorrent file sharing clients
- Bitcoin miner networks
- InterPlanetary File System (IPFS) that connects arbitrary computers to a DFS storing hypermedia

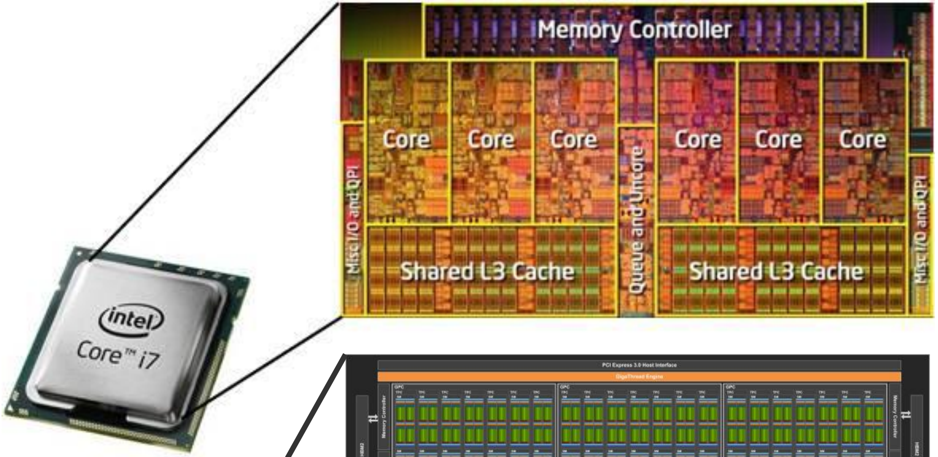
Parallel Computing

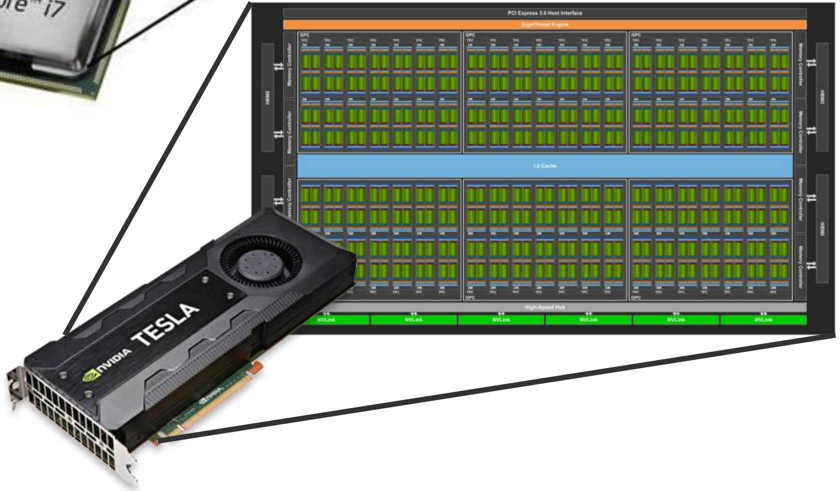
Parallelization

- Multiple processing units perform work simultaneously, i.e., in parallel
- Long tradition in databases
- One approach to address Big Data issues

Trends

- Multicore CPUs
  - E.g. java.util.concurrent or pthread
- General-purpose computing on GPUs (GPGPU)
  - E.g. OpenCL or CUDA
- Cluster frameworks
  - E.g. Hadoop MapReduce, Spark, or Flink

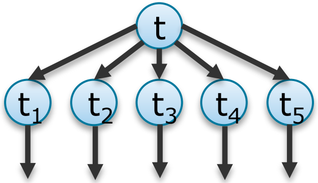




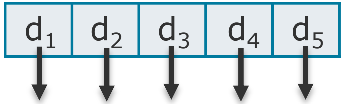
Parallel Computing

Approaches

- Task parallelism:
  - Breaks the task into sub-tasks that are processed in parallel
  - Each processing unit performs a different subtask
    - ➢ Usually OLTP: Akka, RabbitMQ, Kafka, …



- Data parallelism:
  - Breaks the data of a task into packages that are processed in parallel
  - Each processing unit performs the same task on different data
    - ➢ Usually OLAP: MapReduce, Spark, Flink, …



- Instruction-level parallelism:
  - Breaks the task into instructions that are processed in parallel
  - One processing unit performs multiple instructions simultaneously
    - In hardware: instruction pipelining, superscalar, branch prediction, …

Distributed Computing

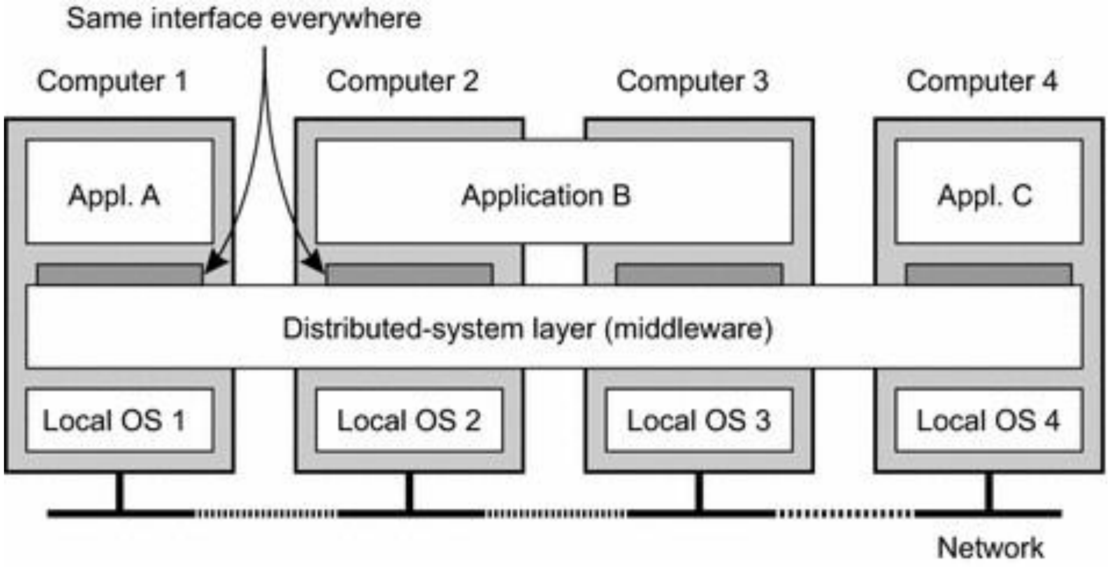
Distributed computing vs. multi-threading:

- Shared nothing:
  - Communication and data sharing only via messaging
  - No shared memory, shared process resources, shared error handling, shared garbage collection, …
- Autonomous processing elements:
  - Synchronization only via messaging
  - No mutexes, semaphores, atomic counters, …
  - If processing elements happen to be processes on the same machine, then the operating system must ensure the autonomy of processes.

➢ More constricted parallelism:
  - ➢ A distributed algorithm can run parallel on one machine but a multi-threaded algorithm (usually) cannot run on many machines.

Distributed Computing

Distributed-system layer (middleware):



- Offers same interfaces and distributed services
  (e.g. Remote Procedure Calls, Load Balancing, Reliable Messaging, …)
- Hides operating system details, resource heterogeneity, failures, …

Amdahl’s Law

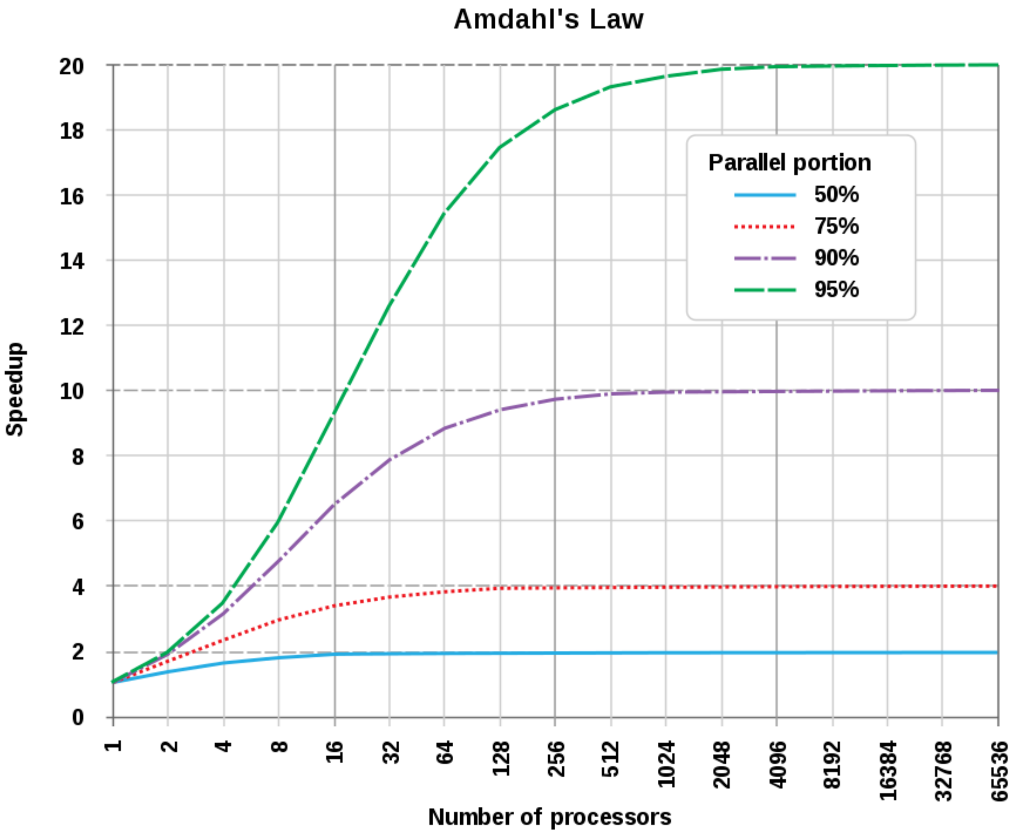
Amdahl’s Law

“The speedup of a program using multiple processors for parallel computing is limited by the sequential fraction of the program”

$$Spee^{d}\quad(s) = \frac{1}{(1-p)+\frac{p}{s}}$$

s: degree of parallelization (e.g. #cores)

p: percentage of the algorithm that profits from parallelization



Amdahl's Law

Amdahl's Law

"The speedup of a program using multiple processors for parallel computing is limited by the sequential fraction of the program"

$$Speed(s) = \frac{1}{(1-p)+\frac{p}{s}}$$

s: degree of parallelization (e.g. #cores)

p: percentage of the algorithm that profits from parallelization

Example 1:

- You developed a new algorithm and measured the runtime for different steps. From that, you learned that 80% of the runtime belongs to steps that can be parallelized. How much faster would the algorithm get on 16 nodes?

$$Speed(16) = \frac{1}{(1-0.8)+\frac{0.8}{16}}$$

$$= \frac{1}{\frac{1}{5}+\frac{4}{5}*\frac{1}{16}} = \frac{1}{\frac{4}{20}+\frac{1}{20}} = \frac{20}{5} = 4$$

Ideal: Does not consider the additional parallelization overhead and communication costs!

Amdahl's Law

Amdahl's Law

"The speedup of a program using multiple processors for parallel computing is limited by the sequential fraction of the program"

$$Spee^{d}\quad(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

s: degree of parallelization (e.g. #cores)

p: percentage of the algorithm that profits from parallelization

Example 2:

- You developed a new algorithm and measured the runtime for different steps. From that, you learned that 80% of the runtime belongs to steps that can be parallelized. What is the maximum speedup via parallelization?

$$Spee^{d}\quad(\infty) = \frac{1}{(1-0.8) + \frac{0.8}{\infty}}$$

$$= \frac{1}{\frac{1}{5} + 0} = 5$$

Amdahl’s Law

Amdahl’s Law

“The speedup of a program using multiple processors for parallel computing is limited by the sequential fraction of the program”

$$Spee^{d}(s) = \frac{1}{(1-p)+\frac{p}{s}}$$

s: degree of parallelization (e.g. #cores)

p: percentage of the algorithm that profits from parallelization

Example 3:

- You developed a new algorithm and measured the runtime for different steps. From that, you learned that 80% of the runtime belongs to steps that can be parallelized. How many nodes do you need for a speedup of 4.5?

$$Spee^{d}(s) = \frac{1}{(1-p)+\frac{p}{s}}$$

$$\Leftrightarrow s = \frac{p * Spee^{d}(s)}{1-(1-p) * Spee^{d}(s)}$$

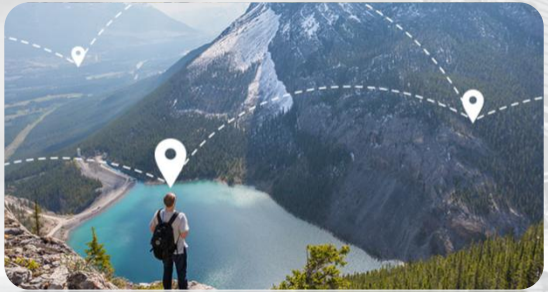
$$s = \frac{0.8 * 4.5}{1-(1-0.8) * 4.5}$$

$$= \frac{4/5 * 9/2}{1-1/5 * 9/2} = 36$$

Overview

Foundations

- Big Data
- Big Data Analytics
- Distributed Computing
- **Data-Intensive Applications**
- Consistency Models



Building Blocks

Databases
- Data storage and persistence

Search indexes
- Keyword search and filtering

Caches
- Optimization of expensive and re-occurring queries

Visualization
- Presentation of data and control options to human users

Batch processing
- Processing of large amounts of accumulated data (transform, analyze)

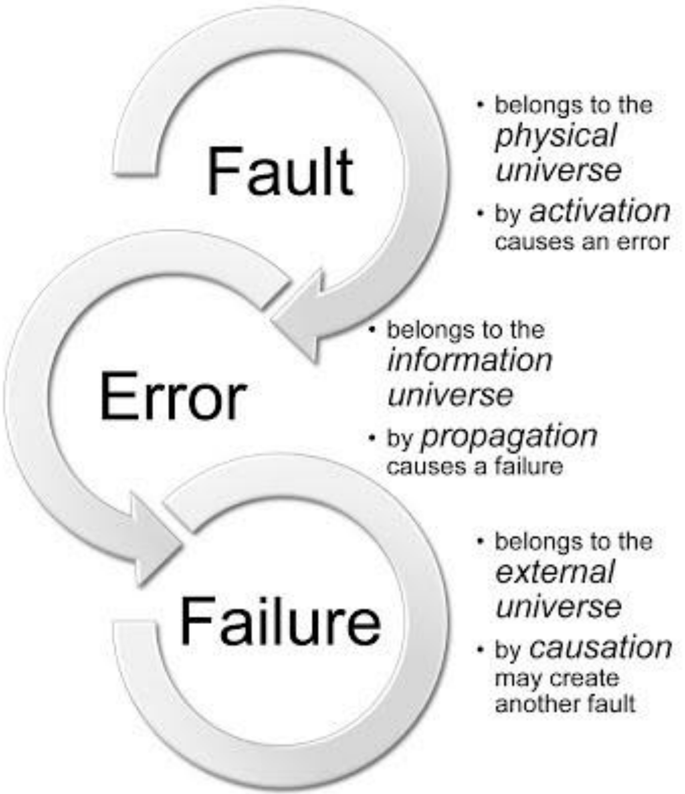
Stream processing
- Processing of continuous data flows (operate, analyze, store)

Design Concerns

1. Reliability
2. Scalability
3. Maintainability

Design Concerns

Reliability



Design Concerns

Reliability

- "The system continues to work **correctly** (= correct functionality at the desired level of performance) even in the face of **adversity** (= hardware or software faults; human faults)."
- = *fault-tolerance*:

fault/defect → may cause → error → may **not** cause → failure

- Techniques to ensure Reliability:
  - Careful design (clear interfaces, decoupling of code, …)
  - Testing (fault-injection, unit/integration/system/random tests, …)
  - ! Redundancy (RAID systems, failover systems, backups, …)
  - ! Process isolation (allowing processes to crash and restart)
  - Measuring, monitoring, and analyzing system behavior in production

Design Concerns

Scalability

- "The system supports **growths** (in data volume, traffic volume, or complexity) with reasonable ways of dealing with it (e.g. more resources)."
- Load:
  - *= measure to quantify scalability*
  - E.g.: requests per second (= throughput), cache hit rate, read/write ratio to disk, …
- Performance:
  - *= load a system can handle*
  - Usually calculated as the mean, median, or x-percentile of load measurements
- Reasoning:
  a) How does an increasing load with fixed resources affect performance?
  b) How much must the resources be increase when the load increases and the performance should be fix?

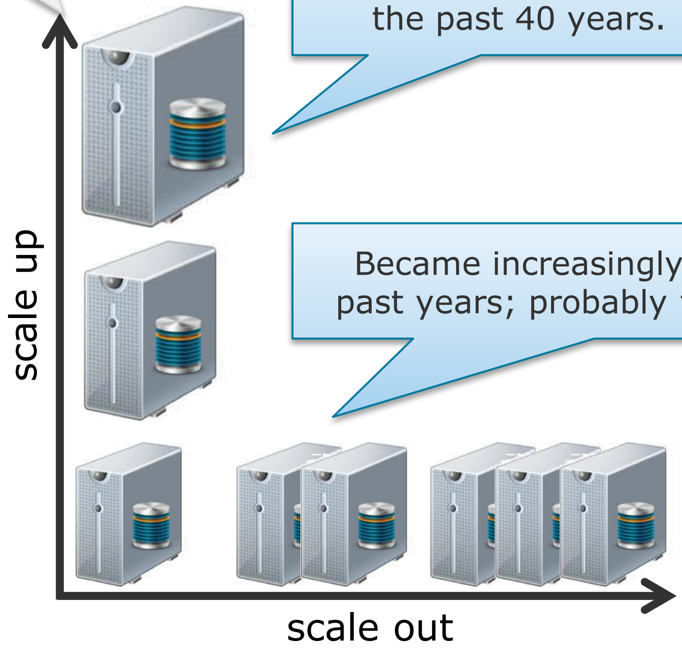
Design Concerns

Scalability (cont.)

- Approaches to cope with load:
  - Vertical scaling (scale up)
    - ➢ Add CPUs, RAM, Disk
    - ➢ Replace entire machine
  - Horizontal scaling (scale out)
    - ➢ Add additional machines
- Scalable software design:
  - a) Manual scaling (a human scales the system resources manually)
  - b) Elastic scaling (the system automatically adds resources if the load increases)

➢ Easier for programmers
➢ More expensive

The default strategy for the past 40 years.



Became increasingly important in the past years; probably the future default.

Design Concerns

Maintainability

- "The system allows its productive, further **development** by different engineers at different times in its operation."
- Design principles to achieve maintainability:
  - Operability: Make it easy for operators to **keep the system running**.
    - ➢ Monitoring, documentation, testing, design patterns, …
  - Simplicity: Make it easy for engineers to **understand the system**.
    - ➢ Clear interfaces, abstraction layers, no over-engineering, …
  - Evolvability: Make it easy for engineers to **change the system**.
    - ➢ Agile techniques, test-driven development, pair programming, …

➢ *See lectures "Software-Architecture" and "Software-Technique" for details!*

➢ *See also: "Spotify Engineering Culture"*
https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/

Overview

Foundations

- Big Data
- Big Data Analytics
- Distributed Computing
- Data-Intensive Applications
- **Consistency Models**

ACID

ACID

- The ACID consistency model stands for the following four guarantees:
  - Atomicity: All operations in a transaction succeed or every operation is rolled back.
  - Consistency: Before the start and after the completion of a transaction, the database is structurally sound.
  - Isolation: Transactions do not contend with one another. Contentious access to data is moderated by the database so that transactions appear to run sequentially.
  - Durability: The results of applying a transaction are permanent, even in the presence of failures.
- ➢ Requires moderated data access, locks, and failover protection
- ➢ Ensures a safe and reliable data storage environment for applications

CAP Theorem

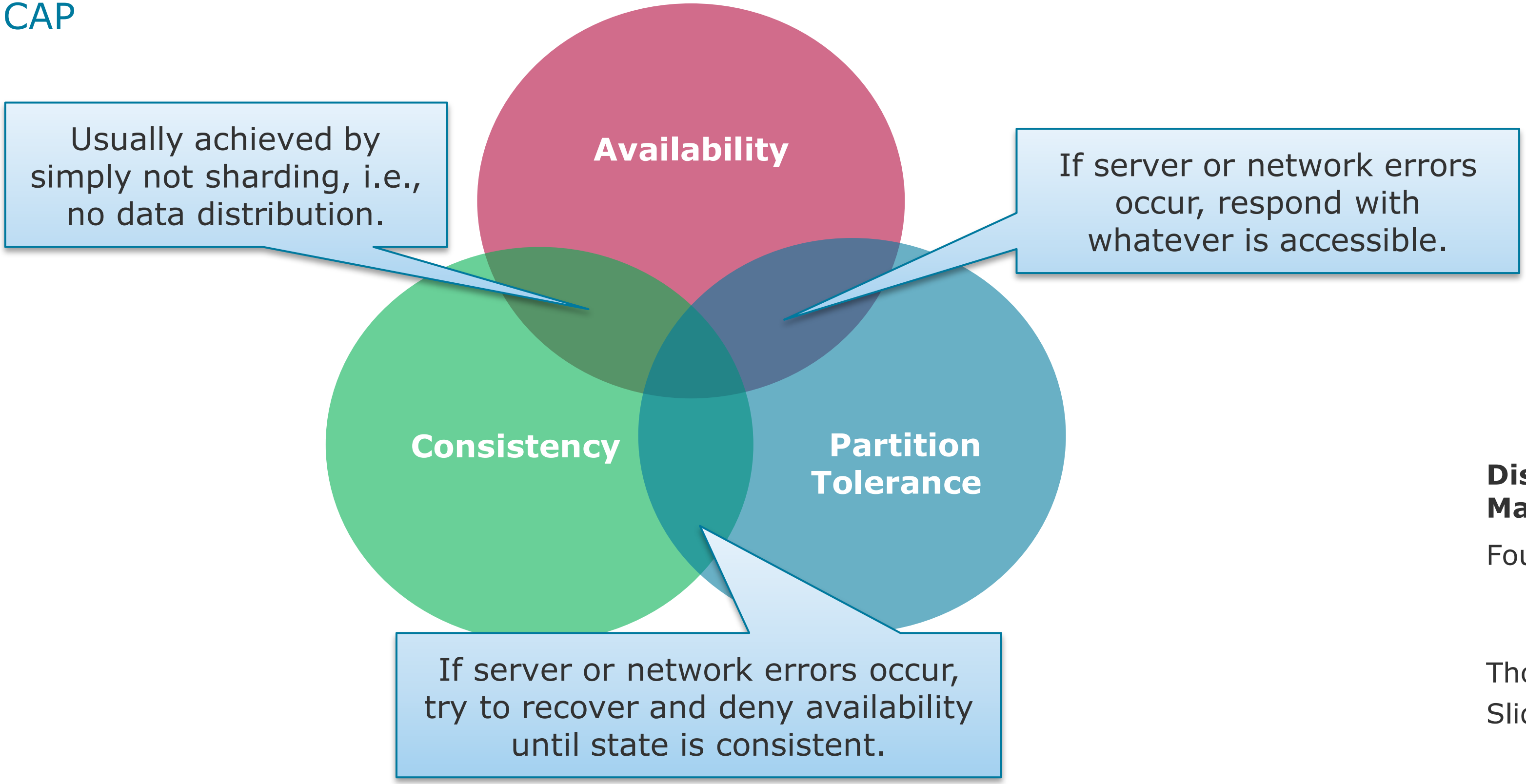
- It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:
  - Consistency: Every read receives the most recent write or an error. This condition includes consistency from ACID, i.e., consistent transaction processing, but also widens the scope from an individual node's data consistency to cluster-wide data consistency.
  - Availability: Every request receives a (non-error) response – without guarantee that it contains the most recent write. Server crashes, query congestion, or resource overload may deny service availability.
  - Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes. Only total network failure might cause the system to respond incorrectly.

Usually stores achieve all three, but they must drop one dimension **if they are distributed and errors occur**.

Seth Gilbert and Nancy Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51–59

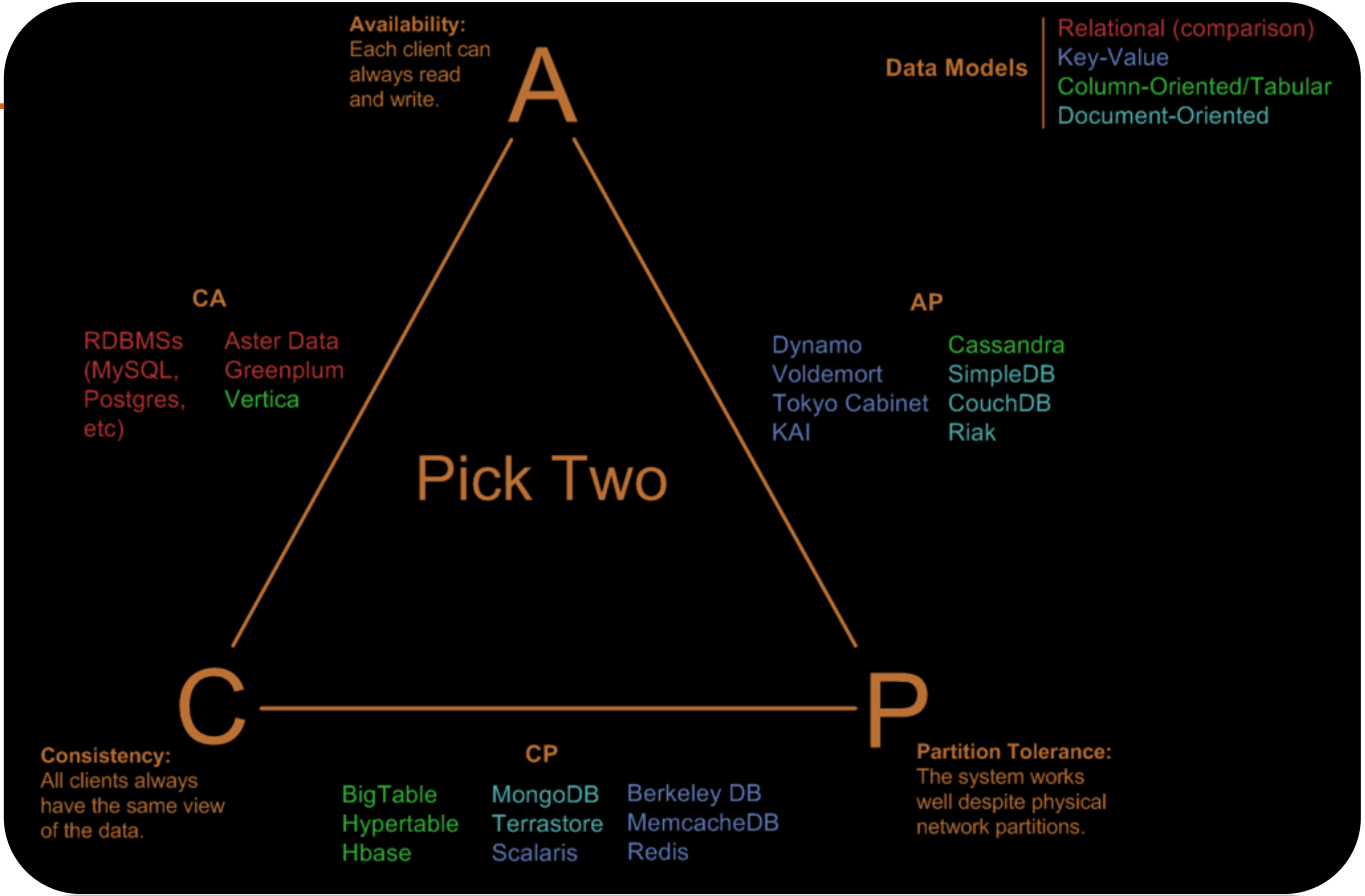
CAP

CAP


Usually achieved by simply not sharding, i.e., no data distribution.
Availability
If server or network errors occur, respond with whatever is accessible.
Consistency
Partition Tolerance
If server or network errors occur, try to recover and deny availability until state is consistent.

CAP

CAP



http://wikibon.org/wiki/v/21_NoSQL_Innovators_to_Look_for_in_2020

BASE

BASE

BASE = "not (fully) ACID"

- The BASE consistency model relaxes CAP dimensions:
  - Basic Availability: The database appears to work most of the time.
    - Availability might be less than 100%
    - "Most of the time" is often quantified as lower bound, e.g., 90%
  - Soft-state: Stores don't have to be write-consistent, nor do different replicas have to be mutually consistent all the time.
    - Stored data might be inconsistent, but the store can derive consistent states
  - Eventual consistency: Stores exhibit consistency at some later point (e.g., lazily at read time).
    - Usually consistent within milliseconds
    - Does not mean "no-consistency", which would be fatal for a store

BASE

BASE

- In comparison to ACID **often** means:

ACID	BASE
Transactions	Programmer managed
Strong consistency	Weak consistency
Isolation	Last write wins
Robust database	Simple database
Simpler application code	Harder application code
Conservative (pessimistic)	Aggressive (optimistic)