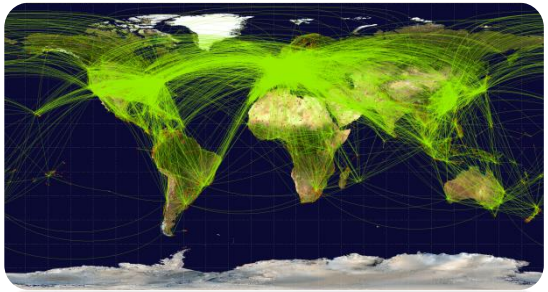
Distributed Data Analytics
Foundations

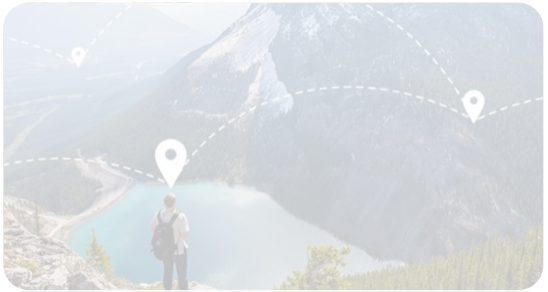
Thorsten Papenbrock
G-3.1.09, Campus III
Hasso Plattner Institut

Foundations

Big Data



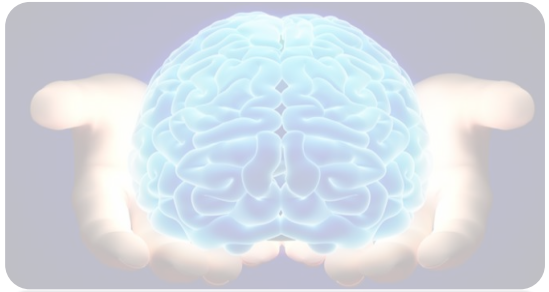
Data-Intensive Applications



Consistency Models

OLAP and OLTP

Distributed Computing



Distributed Data Analytics

Foundation

Definition

- Big data is a term for data sets that are so large or complex that traditional database management tools or data processing software is inadequate to deal with them.
- The challenges include data …
  - capturing
  - storage
  - extraction
  - curation
  - **analysis**
  - search
  - sharing
  - transfer
  - visualization
  - querying
  - updating
  - privacy

If data is **too big**, **too fast**, or **too hard** for existing tools to process, it is Big Data.

Big Data

Properties of Big Data – Gartner's 3 (+ 1) V's

Volume

- 12 terabytes of Tweets (product 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
- 80% data growth in images, videos and documents to improve customer satisfaction

Veracity (Wahrhaftigkeit)

- 1 in 3 business leaders does not trust the information he uses to make decisions

http://www.ibm.com/software/data/bigdata/

Properties of Big Data – More V's

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
  - Hundred thousand sensors on an aircraft are "big data"
  - Each producing an eight byte reading every second
  - Less than 3GB of data in an hour of flying
    (100,000 sensors x 60 minutes x 60 seconds x 8 bytes)
  - Difficult to process due to strong real-time requirements and on plane!

Not all large datasets are "big":

- Example: video streams plus metadata
- Example: telco calls and internet connections
  - 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

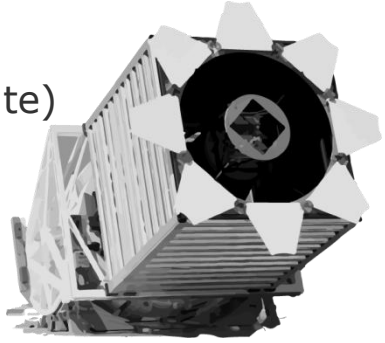
- Protects 2.1 billion active accounts

Big Data in Use – Science

HPI Hasso Plattner Institut

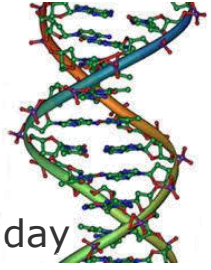
Large Hadron Collider

- 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

- Processing one genome originally took 10 years; now less than a day

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

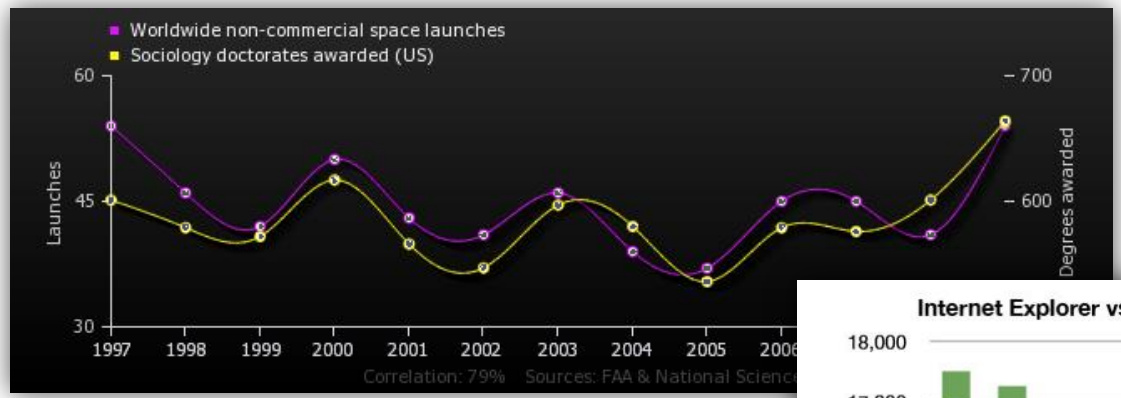
- Causation describes a directed, semantic dependence of one variable (= cause) to another variable (= effect) such that a change in the first variable always causes a corresponding change in the second variable.

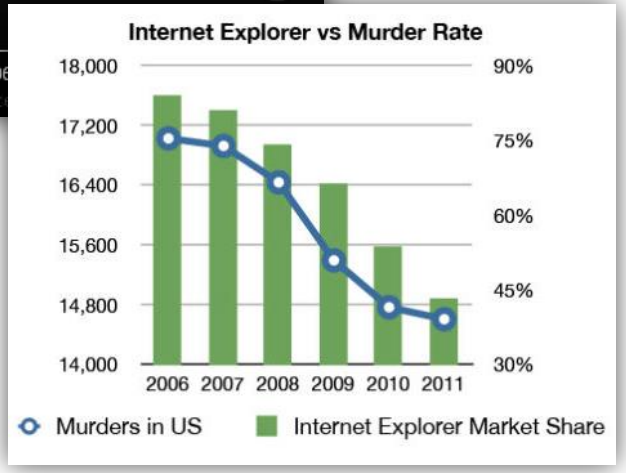
- Correlation ≠ Causation
  - Good science before Big Data: hypothesize → model → test
- But: If correlation holds for very large data sets, it's likely a causation
  - Big Data Analytics: find correlations → derive causations

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

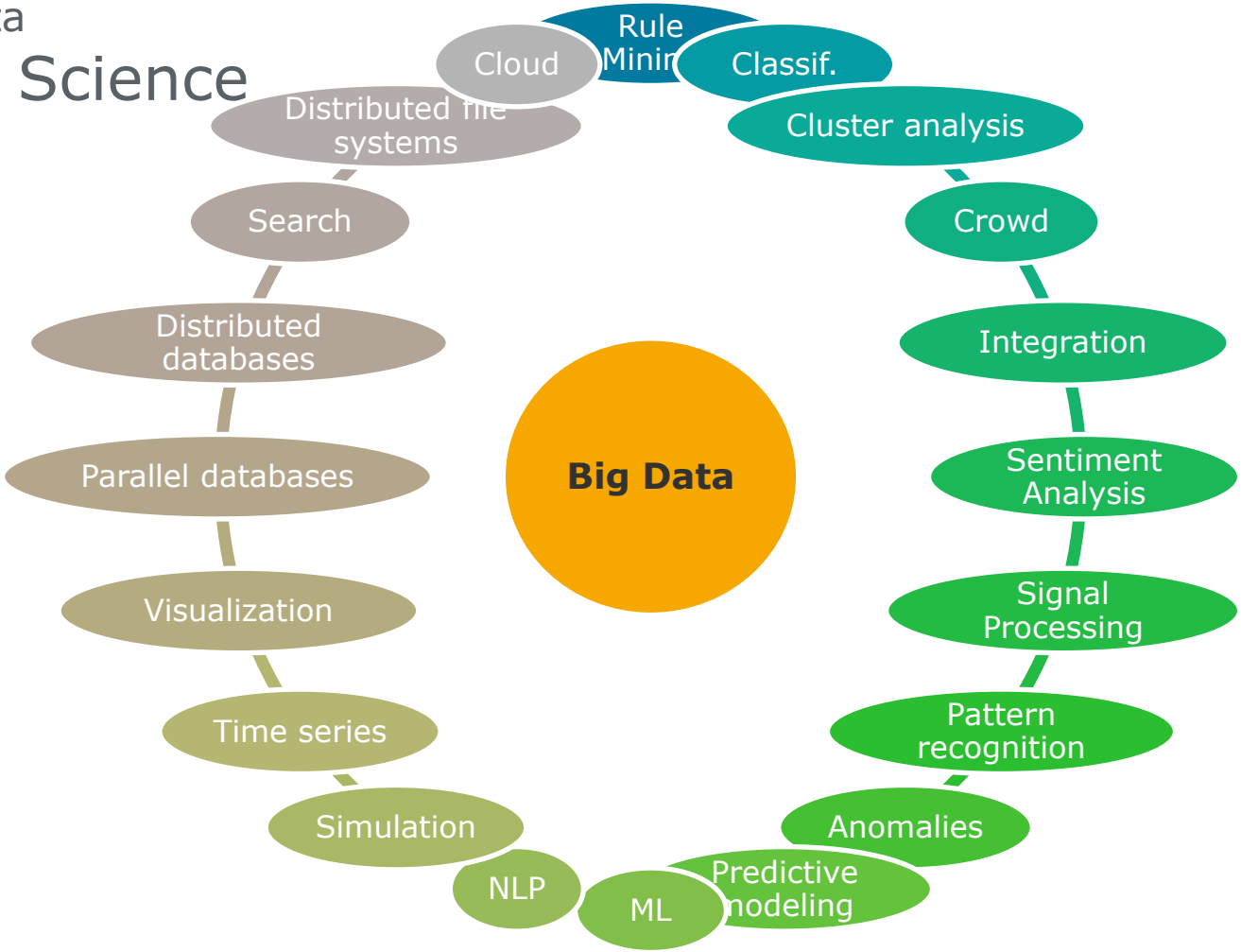
Correlation vs. Causation

Be careful though:



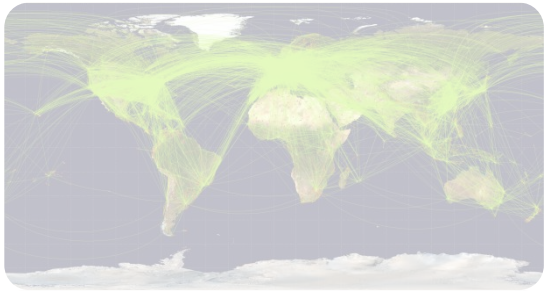


Data Science

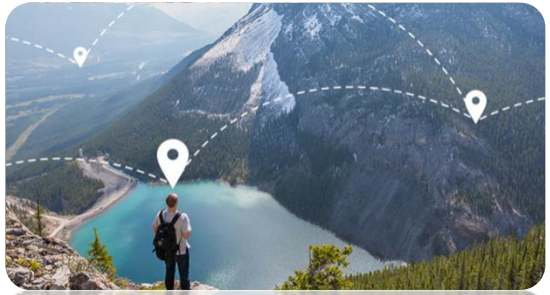


Foundations

Big Data



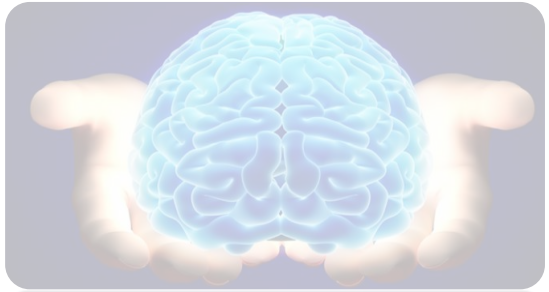
Data-Intensive Applications



Consistency Models

OLAP and OLTP

Distributed Computing



Distributed Data Analytics

Foundation

Building Blocks

Databases

- Data storage and persistence

Caches

- Optimization of expensive and re-occurring queries

Search indexes

- Keyword search and filtering

Stream processing

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

Batch processing

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

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 error)."
- = *fault-tolerance*:

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

Design Concerns

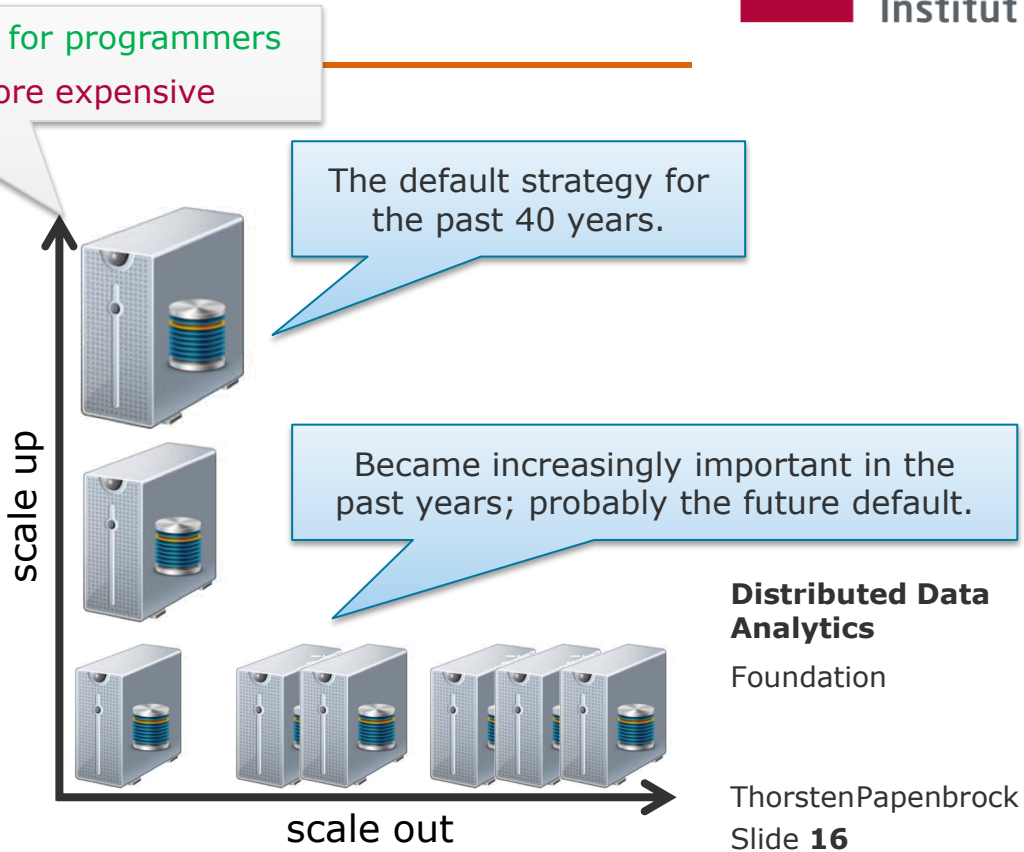
Scalability

- "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, cache hit rate, read/write ratio to disk, …
- Performance:
  - = *load a system can handle*
  - Usually calculated as the mean or median 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 auto-matically adds resources if the load increases)



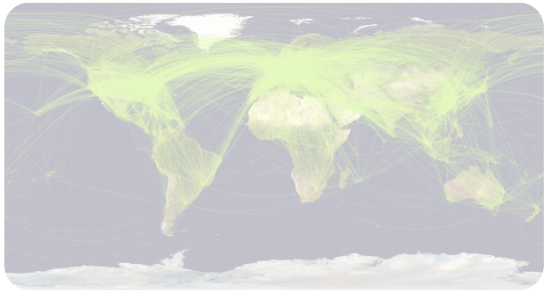
Design Concerns

Maintainability

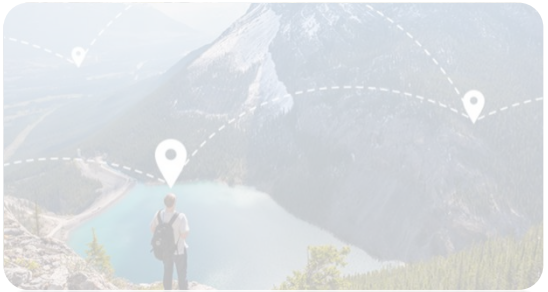
- "The system allows its productive, further **development** by different engineers at different times in its operation."
- Design principles to achieve maintainability:
  - Operability: Make it easy for operators to **keep the system running**.
    - Monitoring, documentation, testing, design patterns, …
  - 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/*

Foundations

Big Data



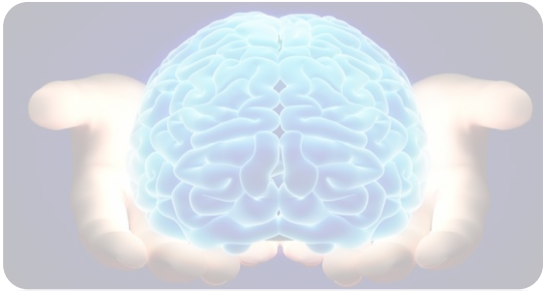
Data-Intensive Applications



Consistency Models

OLAP and OLTP

Distributed Computing



Distributed Data Analytics

Foundation

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

CAP

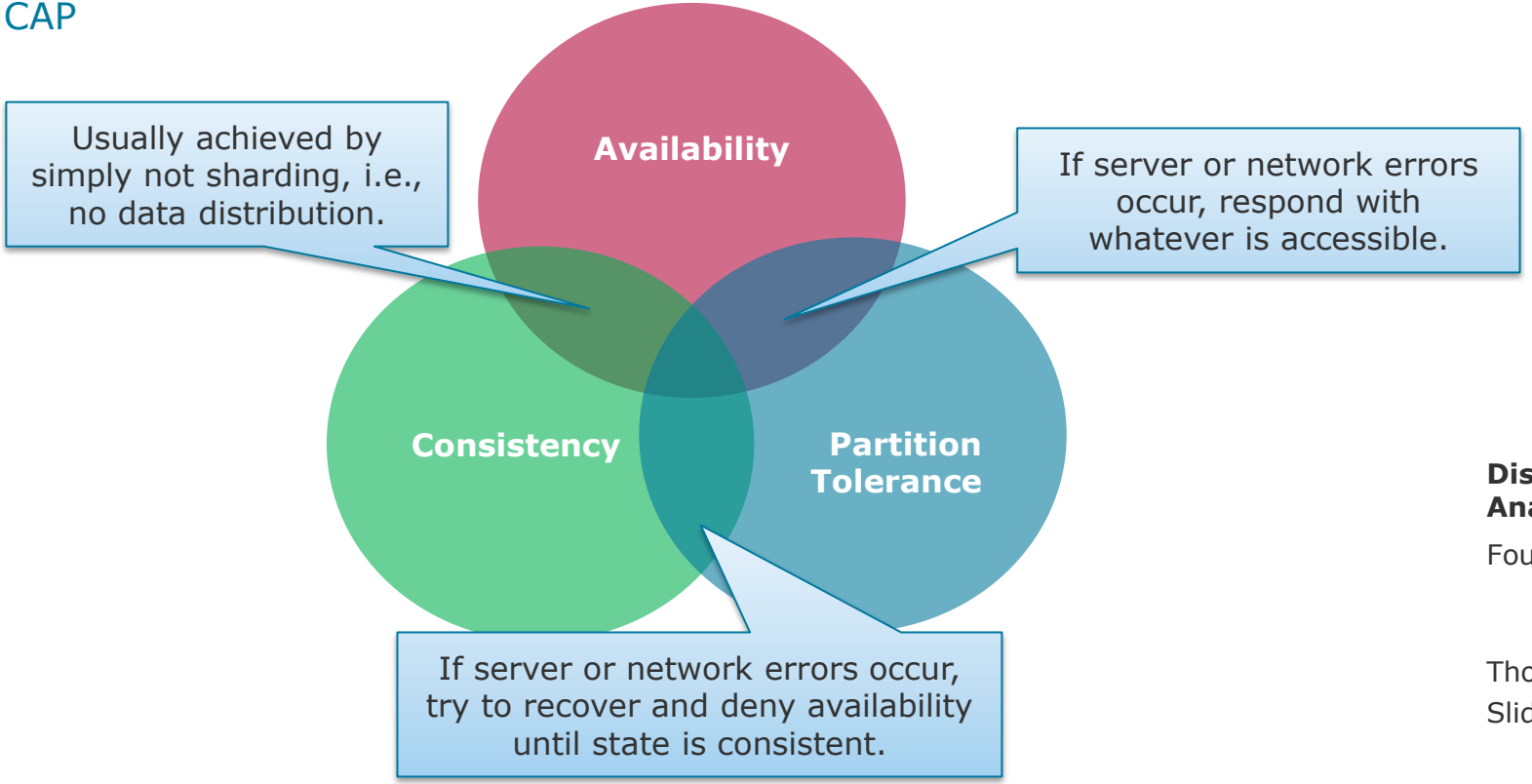
- It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:
  - Consistency: Every read receives the most recent write or an error. This condition includes consistency from ACID, i.e., consistent transaction processing, but also widens 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

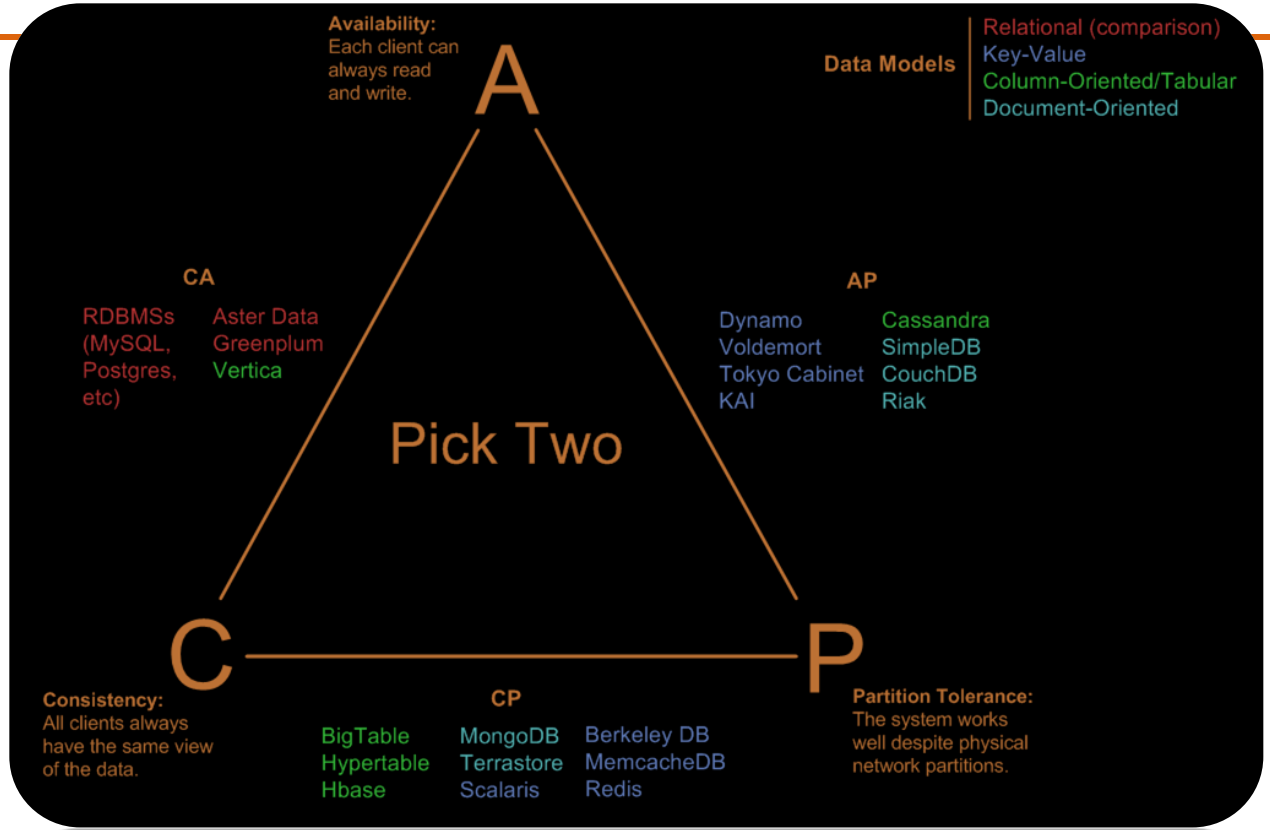
CAP

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

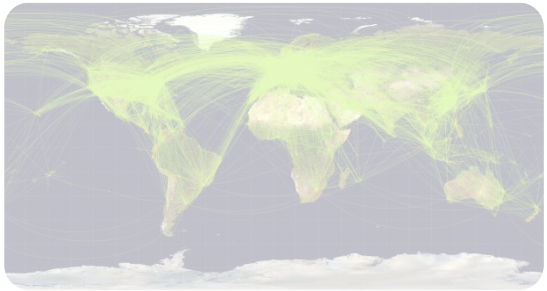
BASE

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

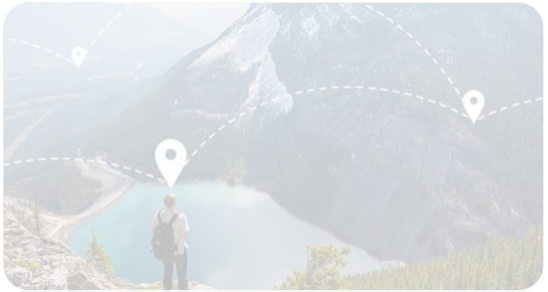
ACID	BASE
Transactions	Programmer managed
Strong consistency	Weak consistency
Isolation	Last write wins
Robust database	Simple database
Simpler application code	Harder application code
Conservative (pessimistic)	Aggressive (optimistic)

Foundations

Big Data



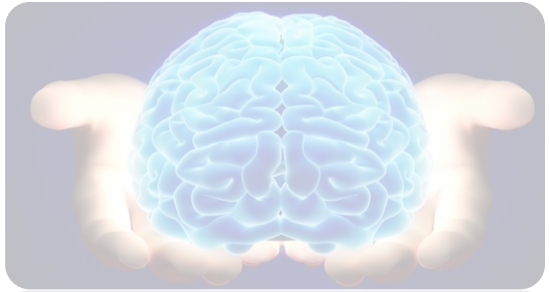
Data-Intensive Applications



Consistency Models

OLAP and OLTP

Distributed Computing



Distributed Data Analytics

Foundation

Definitions

Online Transaction Processing (OLTP)

- Fast processing of operational data, i.e., transactions while maintaining data integrity in multi-access environments
- Performance characteristic: transactions per second
- Often: time-critical, mixed-workload data with high velocity
- Dominating operations: INSERT, UPDATE, DELETE

Online Analytical Processing (OLAP)

- Effective answering of (arbitrary, complex) analytical queries on already collected data
- Performance characteristic: query response time
- Often: pre-aggregated, multi-dimensional, and historical data
- Dominating operations: SELECT

Comparison

Property	OLTP	OLAP
Main read pattern	Small number of records per query, fetched by key	Aggregate over large number of records
Main write pattern	Random-access, low-latency writes from user input	Bulk import (ETL) or event stream
Primarily used by	End user/customer, via (web) application	Internal analyst, for decision support
What data represents	Latest state of data (current point in time)	History of events that happened over time
Data size	Gigabytes to terabytes	Terabytes to petabytes

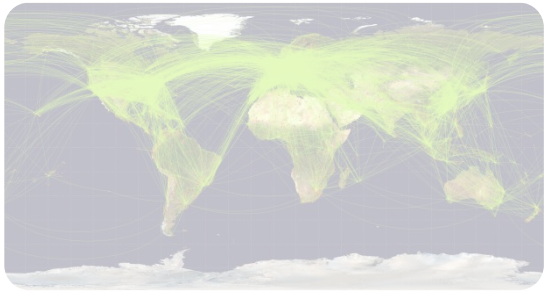
Comparison

	OLTP System Online Transaction Processing (Operational System)	OLAP System Online Analytical Processing (Data Warehouse)
Source of data	Operational data; OLTPs are the original source of the data.	Consolidation data; OLAP data comes from the various OLTP Databases
Purpose of data	To control and run fundamental business tasks	To help with planning, problem solving, and decision support
What the data	Reveals a snapshot of ongoing business processes	Multi-dimensional views of various kinds of business activities
Inserts and Updates	Short and fast inserts and updates initiated by end users	Periodic long-running batch jobs refresh the data
Queries	Relatively standardized and simple queries Returning relatively few records	Often complex queries involving aggregations
Processing Speed	Typically very fast	Depends on the amount of data involved; batch data refreshes and complex queries may take many hours; query speed can be improved by creating indexes
Space Requirements	Can be relatively small if historical data is archived	Larger due to the existence of aggregation structures and history data; requires more indexes than OLTP
Database Design	Highly normalized with many tables	Typically de-normalized with fewer tables; use of star and/or snowflake schemas
Backup and Recovery	Backup religiously; operational data is critical to run the business, data loss is likely to entail significant monetary loss and legal liability	Instead of regular backups, some environments may consider simply reloading the OLTP data as a recovery method

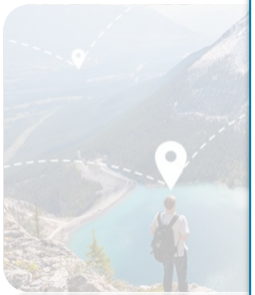
source: www.rainmakerworks.com

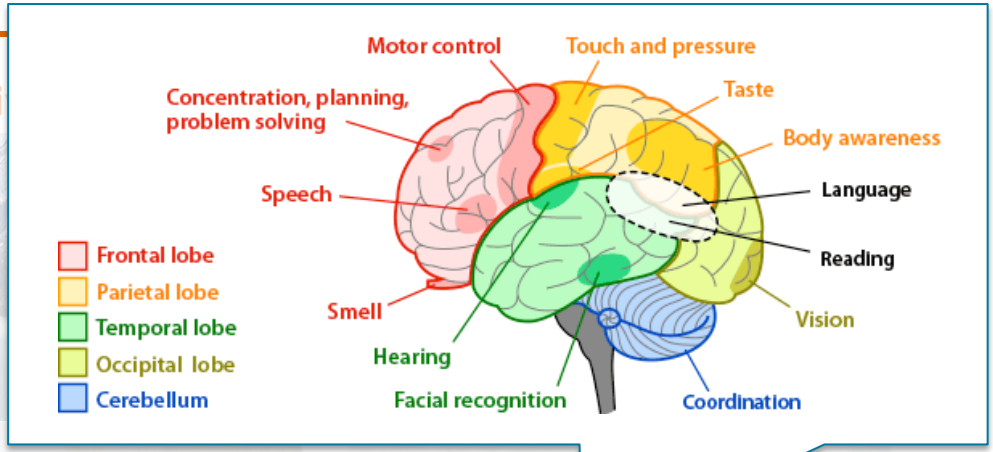
Foundations

Big Data



Data-Intensi





OLAP and OLTP

Distributed Computing

Distributed Data Analytics

Foundation

Definition

What is a distributed system?

Data in multiple caches, in memory, on disk …

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

One machine?

Specialized racks with shared infrastructure …

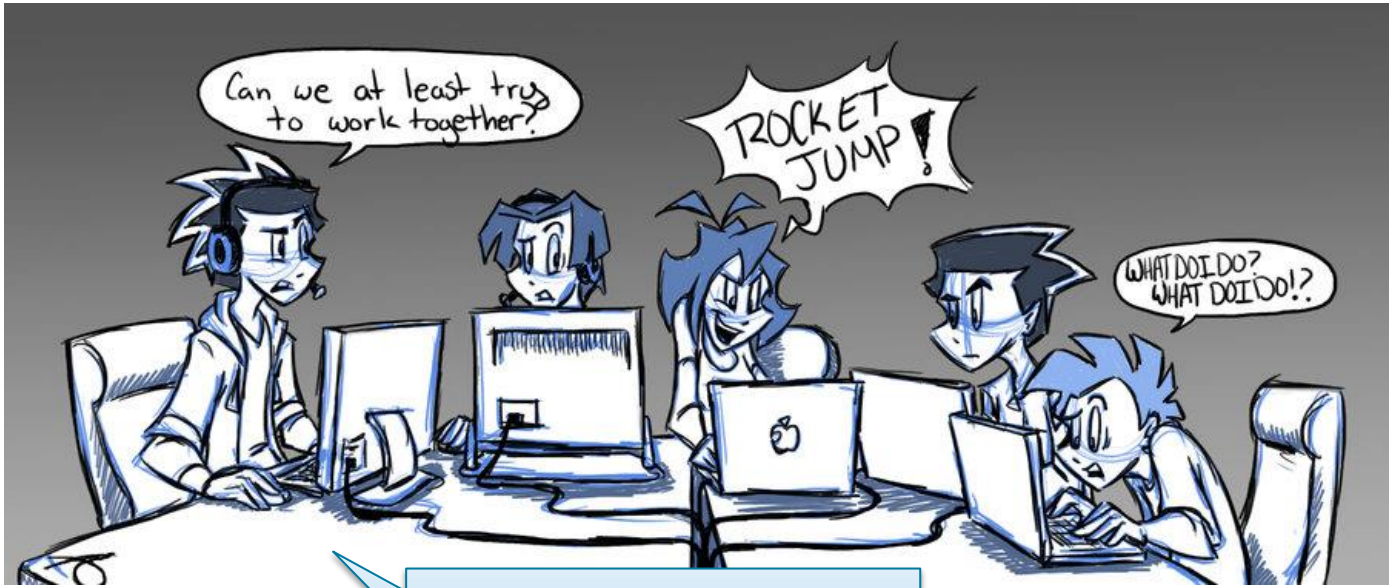
One big machine?

Independent systems connected via network

Multiple, connected machines?

Definition

What is a distributed system?



Do the system components need to work together?

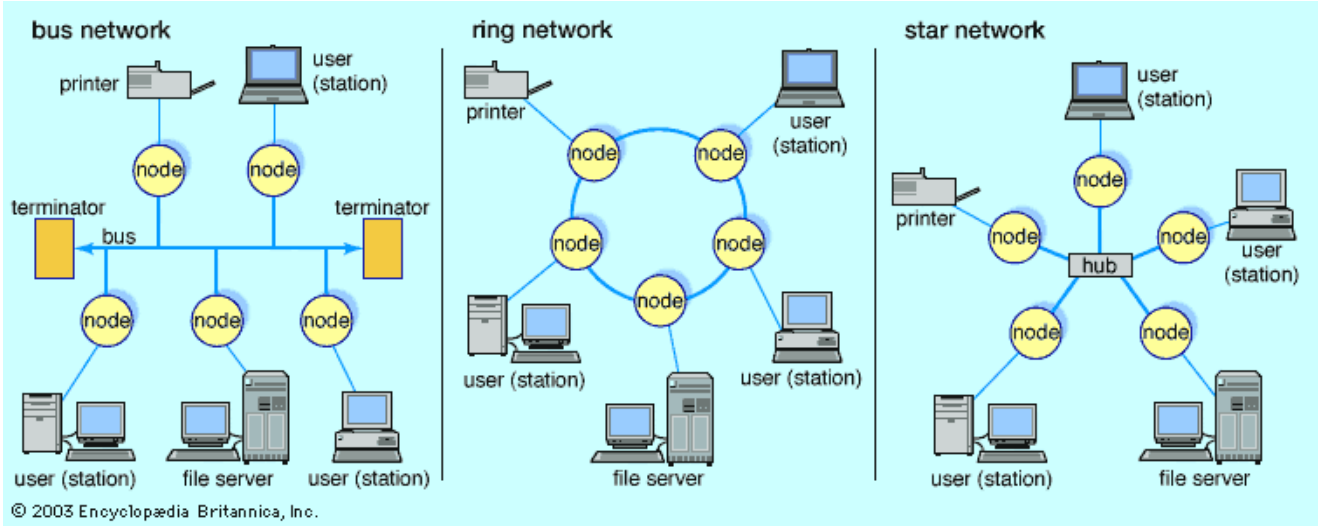
Definition

shared-nothing systems

Practical Definition:

"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."

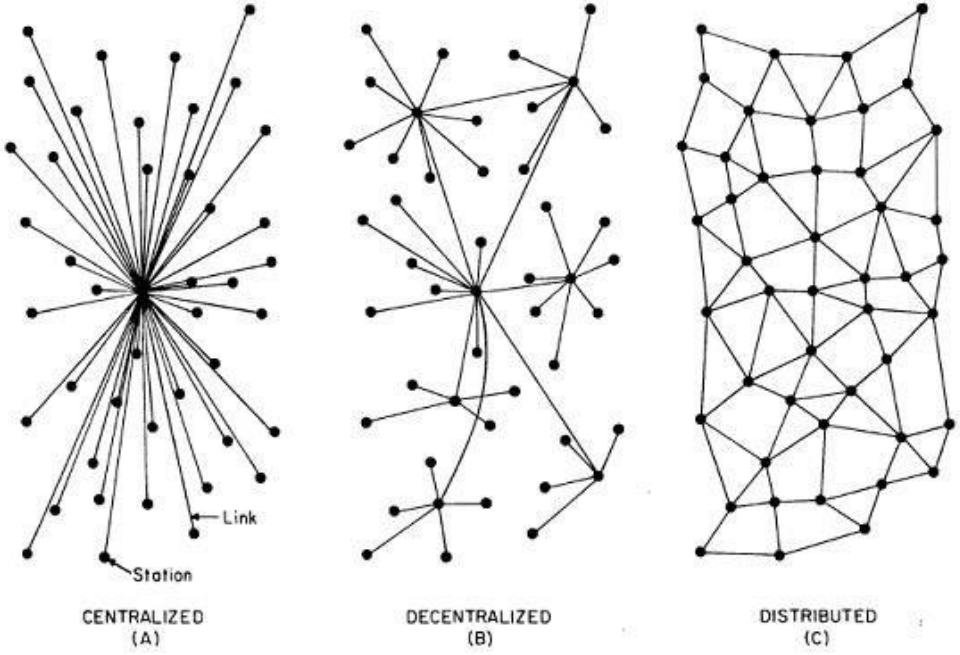
(M. Tamer Özsu, Patrick Valduriez: "Principles of Distributed Database Systems")



Definition

Topological Definition:

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



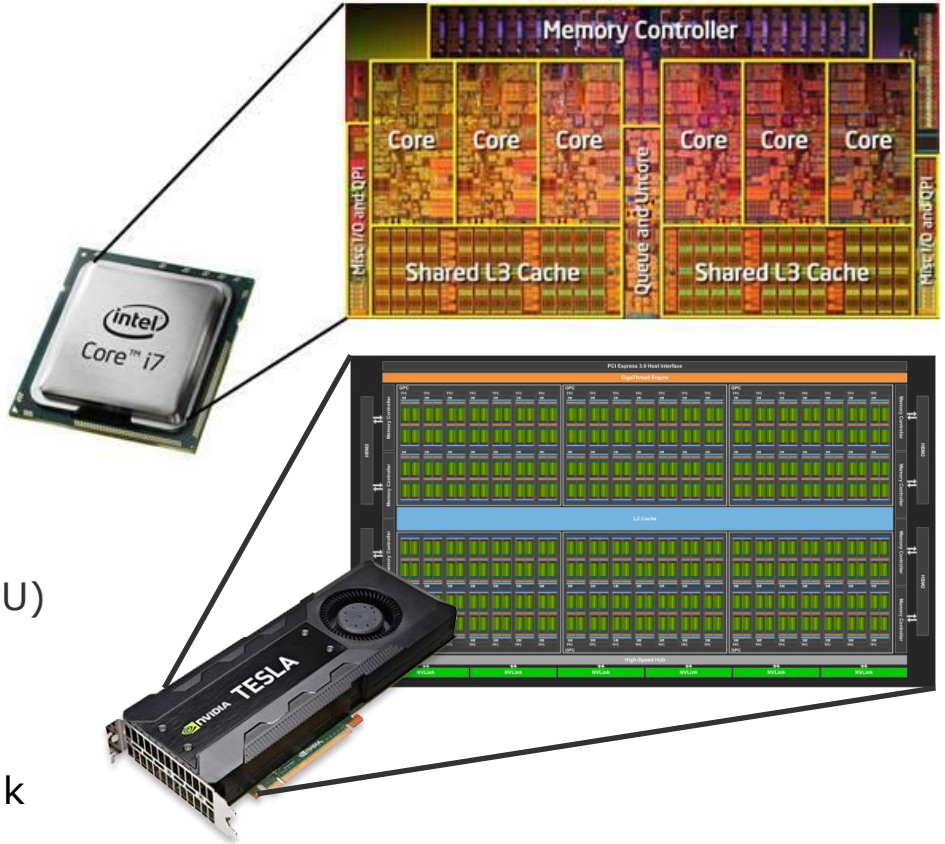
Parallel Computing

Parallelization

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

Trends

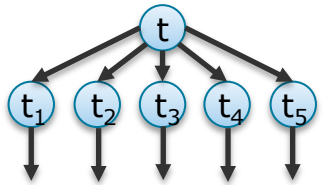
- Multicore CPUs
  - E.g. java.util.concurrent or pthread
- General-purpose computing on GPUs (GPGPU)
  - E.g. OpenCL or CUDA
- Cluster frameworks
  - E.g. Hadoop MapReduce, Spark, or Flink



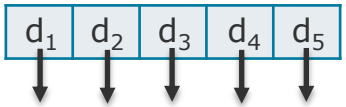
Parallel Computing

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 element performs multiple instructions simultaneously
    - In hardware: instruction pipelining, superscalar, branch prediction, …

Distinction

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 systems:
  - Synchronization only via messaging
  - No mutexes, semaphores, atomic counters, lock-free data structures, blocking queues, …
- ➢ More constricted parallelism:
  - ➢ A distributed parallel implementation can run on one machine but a multi-threaded parallel implementation (usually) not on many machines