MI-PDB, MIE-PDB: **Advanced Database Systems**

http://www.ksi.mff.cuni.cz/~svoboda/courses/2015-2-MIE-PDB/

Lecture 8:

Big Data and NoSQL Databases

12. 4. 2016

Lecturer: **Martin Svoboda**

svoboda@ksi.mff.cuni.cz

Author: **Irena Holubová**

Faculty of Mathematics and Physics, Charles University in Prague

Course NDBI040: **Big Data Management and NoSQL Databases**

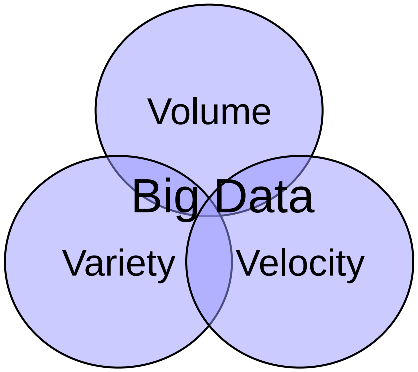
What is Big Data?

- buzzword?
- bubble?
- gold rush?
- revolution?

"Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it."

Dan Ariely

What is Big Data?



- No standard definition
- First occurrence of the term: High Performance Computing (HPC)

Gartner: *“**Big Data**” is high **v**olume, high **v**elocity, and/or high **v**ariety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization.*

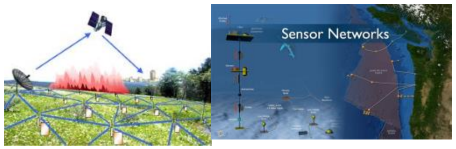
3 (4, 5) Vs

What is Big Data?

Mobile devices
(tracking all objects all the time)

Social media and networks
(all of us are generating data)

Scientific instruments
(collecting all sorts of data)

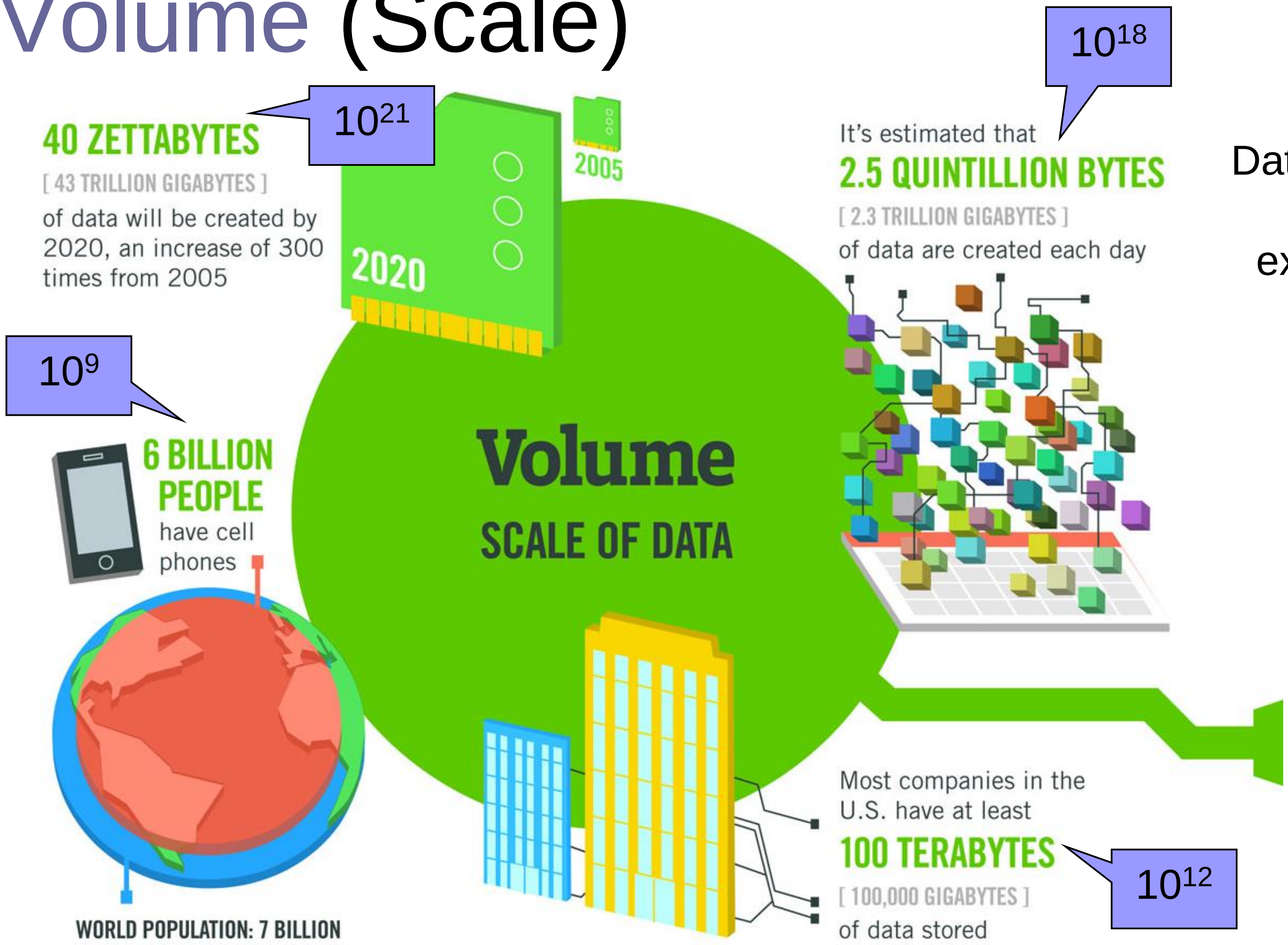


Sensor technology and networks
(measuring all kinds of data)

IBM: *Depending on the industry and organization, **Big Data** encompasses information from internal and external sources such as transactions, social media, enterprise content, sensors, and mobile devices.*
Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

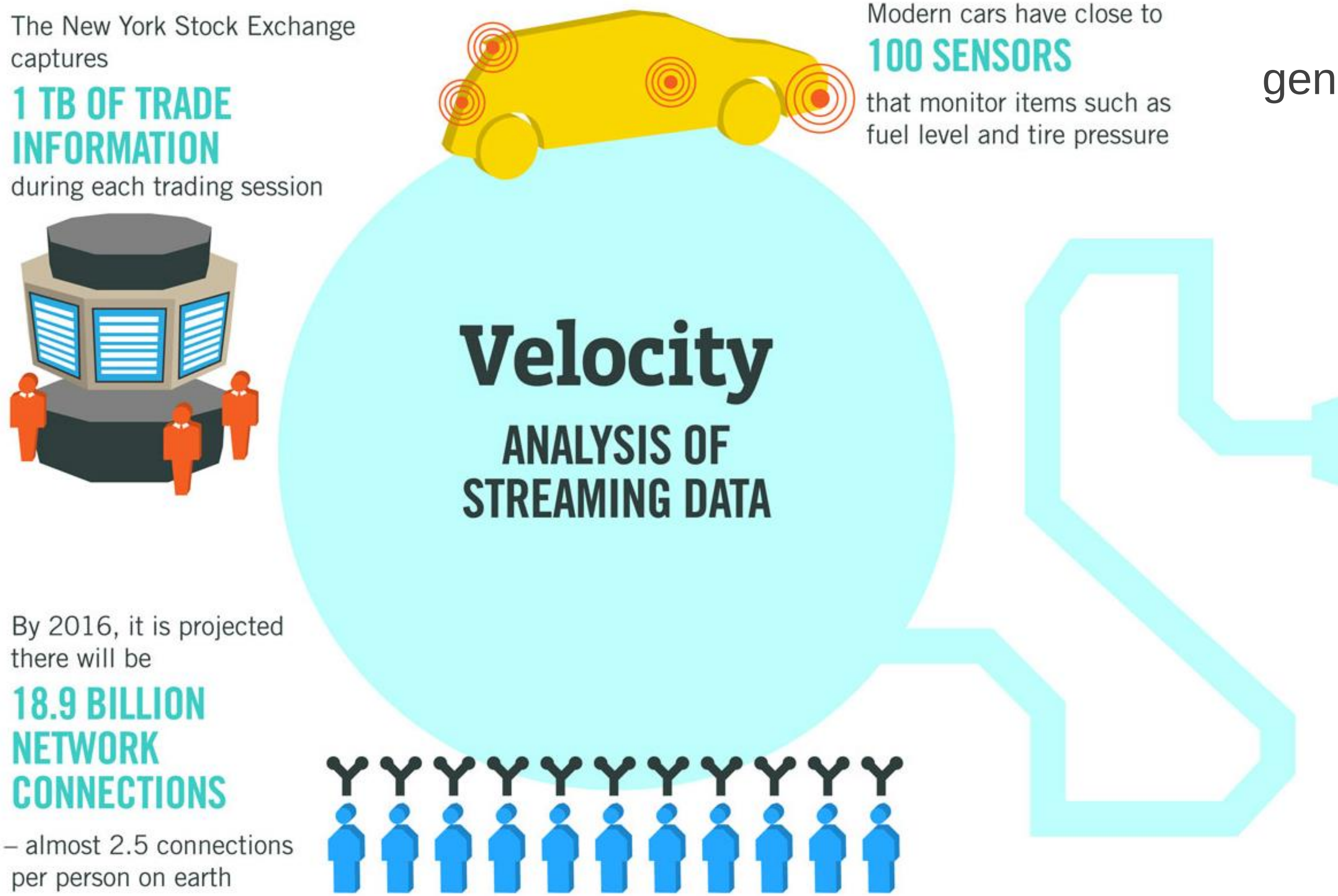
http://www.ibmbigdatahub.com/

Big Data Characteristics: Volume (Scale)



Data volume is increasing exponentially, not linearly

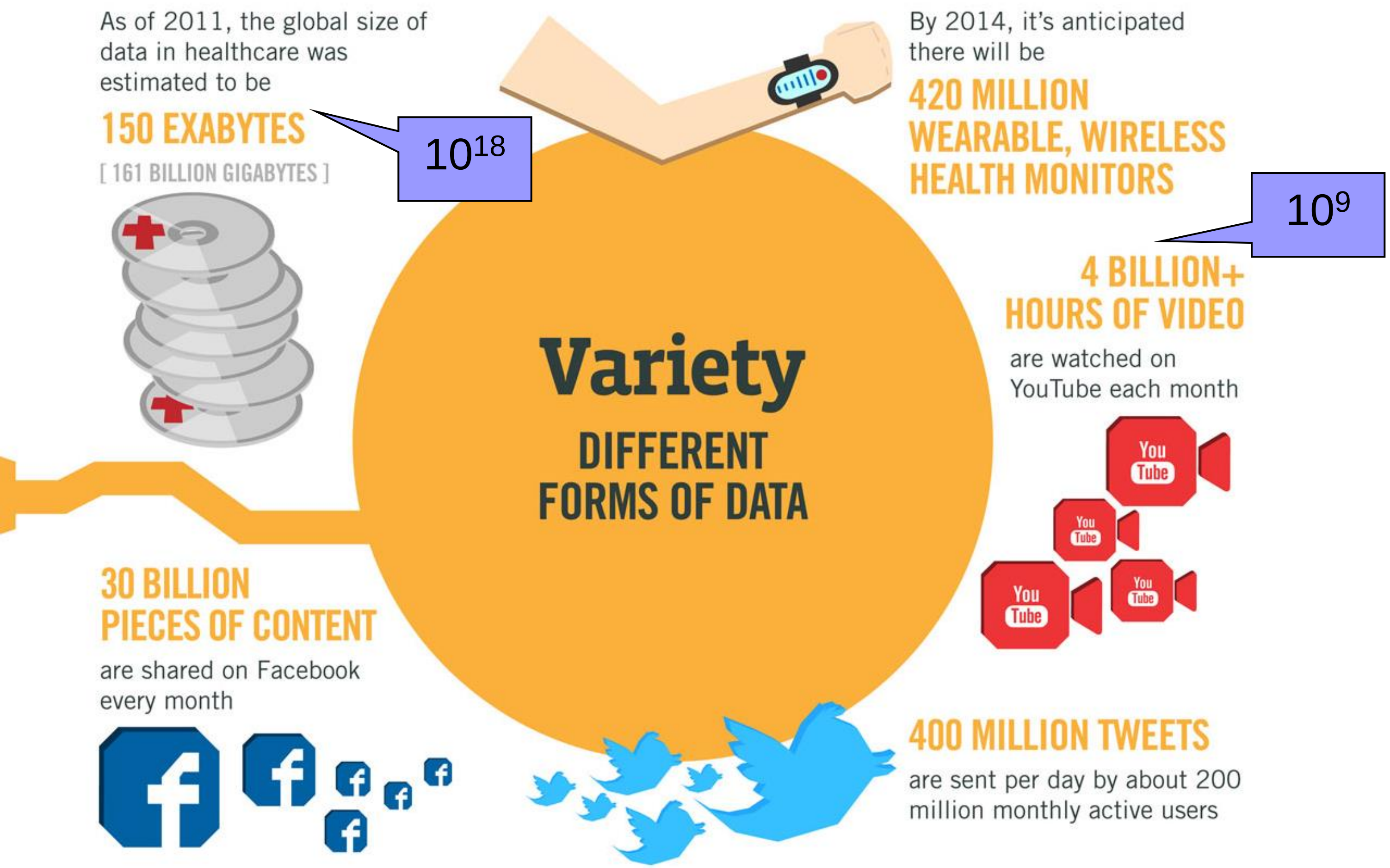
Big Data Characteristics: Velocity (Speed)



Data is being generated fast and need to be processed fast

Online Data Analytics

Big Data Characteristics: Variety (Complexity)

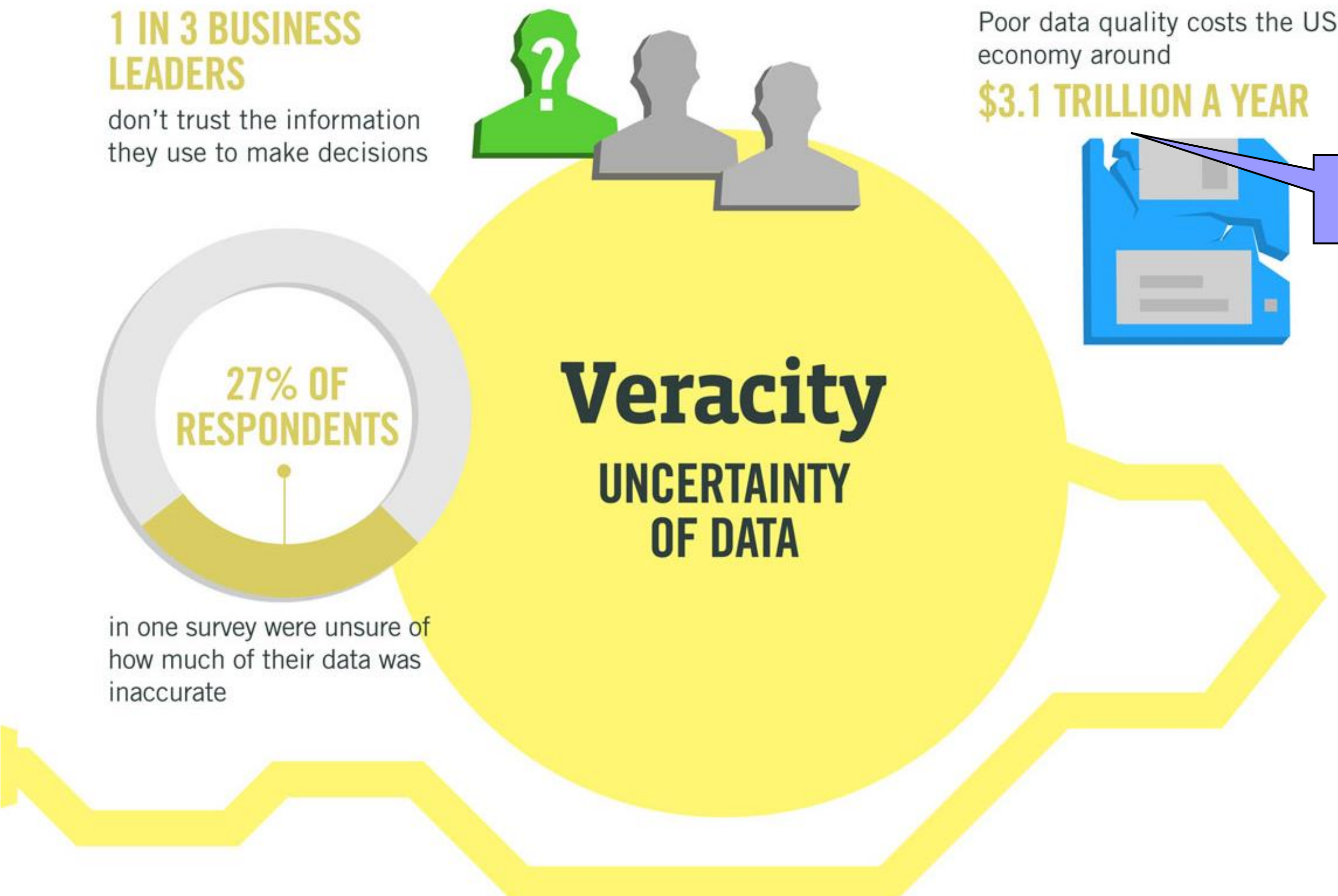


Various formats, types, and structures (from semi-structured XML to unstructured multimedia)

Static data vs. streaming data

http://www.ibmbigdatahub.com/

Big Data Characteristics: Veracity (Uncertainty)



10^{12}

Uncertainty due to inconsistency, incompleteness, latency, ambiguities, or approximations.

Processing Big Data

- OLTP: Online Transaction Processing (DBMSs)
  - Database applications
  - Storing, querying, multiuser access
- OLAP: Online Analytical Processing (Data Warehousing)
  - Answer multi-dimensional analytical queries
  - Financial/marketing reporting, budgeting, forecasting, …
- RTAP: Real-Time Analytic Processing (Big Data Architecture & Technology)
  - Data gathered & processed in a real-time
    - Streaming fashion
  - Real-time data queried and presented in an online fashion
  - Real-time and history data combined and mined interactively

Key Big Data-Related Technologies

- Distributed file systems
- NoSQL databases
- Grid computing, cloud computing
- MapReduce and other new paradigms
- Large scale machine learning

Relational Database Management Systems (RDMBSs)

- Predominant technology for storing structured data
  - Web and business applications
- Relational calculus, SQL
- Often thought of as the only alternative for data storage
  - Persistence, concurrency control, integration mechanism, …
- Alternatives: Object databases or XML stores
  - Never gained the same adoption and market share

„NoSQL“

- 1998 first used for a relational database that omitted the use of SQL
  - Carlo Strozzi
- 2009 used for conferences of advocates of non-relational databases
  - Eric Evans
    - Blogger, developer at Rackspace

NoSQL movement = “the whole point of seeking alternatives is that you need to solve a problem that relational databases are a bad fit for”

„NoSQL“

- Not „no to SQL“
  - Another option, not the only one
- Not „not only SQL“
  - Oracle DB or PostgreSQL would fit the definition
- „Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable. The original intention has been modern web-scale databases. Often more characteristics apply as: schema-free, easy replication support, simple API, eventually consistent (BASE, not ACID), a huge data amount, and more“

The End of Relational Databases?

- Relational databases are <u>not</u> going away
- Compelling arguments for most projects
  - Familiarity, stability, feature set, and available support
- We should see relational databases as one option for data storage
  - Polyglot persistence – using different data stores in different circumstances
  - Search for optimal storage for a particular application

NoSQL Databases
Five Advantages

1. Elastic scaling

- "Classical" database administrators scale up – buy bigger servers as database load increases
- Scaling out – distributing the database across multiple hosts as load increases

2. Big Data

3. Goodbye DBAs (see you later?)

- Automatic repair, distribution, tuning, …

4. Economics

- Based on cheap commodity servers

5. Flexible Data Models

- Non-existing/relaxed data schema $\rightarrow$ cheap structural changes

http://www.techrepublic.com/blog/10things/10-things-you-should-know-about-nosql-databases/1772

NoSQL Databases

Five Challenges

1. **Maturity**
   - Still in pre-production phase
   - Key features yet to be implemented
2. **Support**
   - Mostly open source, result from start-ups
   - Limited resources or credibility
3. **Administration**
   - Require lot of skill to install and effort to maintain
4. **Analytics and Business Intelligence**
5. **Expertise**
   - Few number of NoSQL experts available in the market

Data Assumptions

RDBMS	**NoSQL**
integrity is mission-critical	OK as long as most data is correct
data format consistent, well-defined	data format unknown or inconsistent
data is of long-term value	data are expected to be replaced
data updates are frequent	write-once, read multiple (no updates, or at least not often)
predictable, linear growth	unpredictable growth (exponential)
non-programmers writing queries	only programmers writing queries
regular backup	replication
access through master server	sharding across multiple nodes

NoSQL Data Model

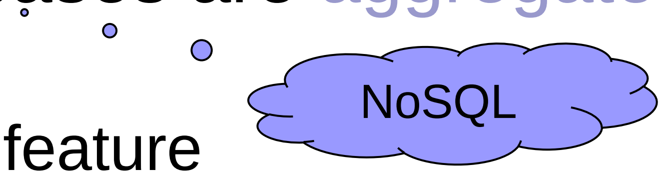
Aggregates

- Data model = the model by which the database organizes data
- Each NoSQL solution has a different model
  - Key-value, document, column-family, graph
  - First three orient on aggregates
- Aggregate
  - A data unit with a complex structure
    - Not just a set of tuples like in RDBMS
  - Domain-Driven Design: “an aggregate is a collection of related objects that we wish to treat as a unit”
    - A unit for data manipulation and management of consistency

NoSQL Data Model

Aggregates – aggregate-ignorant

- There is no universal strategy how to draw aggregate boundaries
  - Depends on how we manipulate the data
- RDBMS and graph databases are aggregate-ignorant
  - It is not a bad thing, it is a feature
  - Allows to easily look at the data in different ways
  - Better choice when we do not have a primary structure for manipulating data



NoSQL Data Model

Aggregates – aggregate-oriented

- Aggregate orientation
  - Aggregates give the database information about which bits of data will be manipulated together
    - Which should live on the same node
  - Helps greatly with running on a cluster
    - We need to minimize the number of nodes we need to query when we are gathering data
- Consequence for transactions
  - NoSQL databases support atomic manipulation of a single aggregate at a time

Types of NoSQL Databases

Core:

- Key-value databases
- Document databases
- Column-family (column-oriented/columnar) stores
- Graph databases

Non-core:

- Object databases
- XML databases
- …

http://nosql-database.org/

Key-value store
Basic characteristics

- The simplest NoSQL data stores
- A simple hash table (map), primarily used when all access to the database is via primary key
- A table in RDBMS with two columns, such as ID and NAME
  - ID column being the key
  - NAME column storing the value
    - A BLOB that the data store just stores
- Basic operations:
  - Get the value for the key
  - Put a value for a key
  - Delete a key from the data store
- Simple $\rightarrow$ great performance, easily scaled
- Simple $\rightarrow$ not for complex queries, aggregation needs

Key-value store

Representatives

MemcachedDB

MEMCACHED

riak

redis

ORACLE

BERKELEY DB

HamsterDB
embedded database

amazon
web services
Amazon DynamoDB

not
open-source

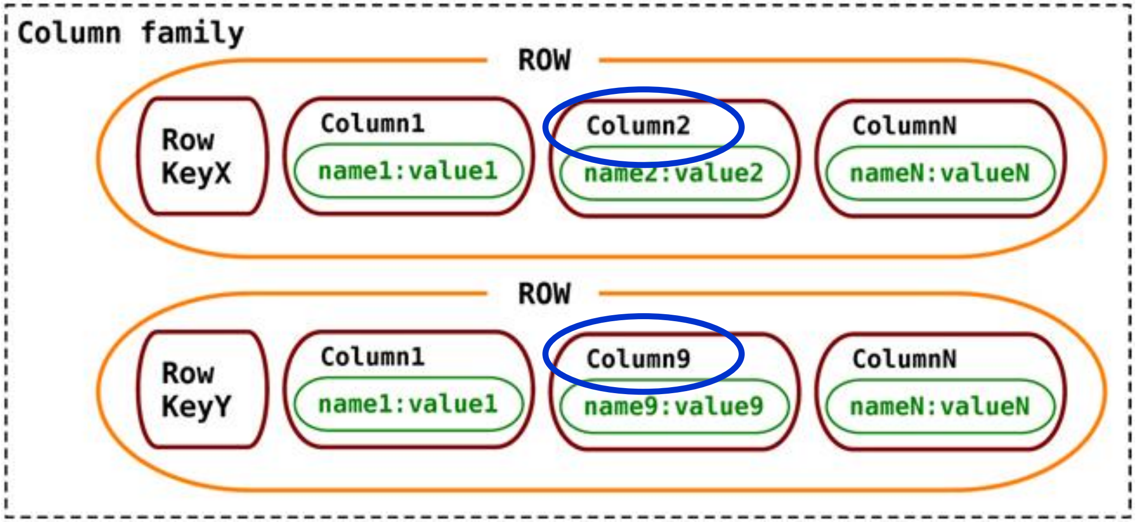
open-source
version

Project Voldemort

Column-Family Stores

Basic Characteristics

- Also “columnar” or “column-oriented”
- Column families = rows that have many columns associated with a row key
- Column families are groups of related data that is often accessed together
  - e.g., for a customer we access all profile information at the same time, but not orders



Column-Family Stores

Representatives

Google's BigTable

H·BASE

Cassandra

HYPERTABLE

SimpleDB

Document Databases

Basic Characteristics

- Documents are the main concept
  - Stored and retrieved
  - XML, JSON, …
- Documents are
  - Self-describing
  - Hierarchical tree data structures
  - Can consist of maps, collections (lists, sets, …), scalar values, nested documents, …
- Documents in a collection are expected to be similar
  - Their schema can differ
- Document databases store documents in the value part of the key-value store
  - Key-value stores where the value is examinable

Document Databases

Representatives

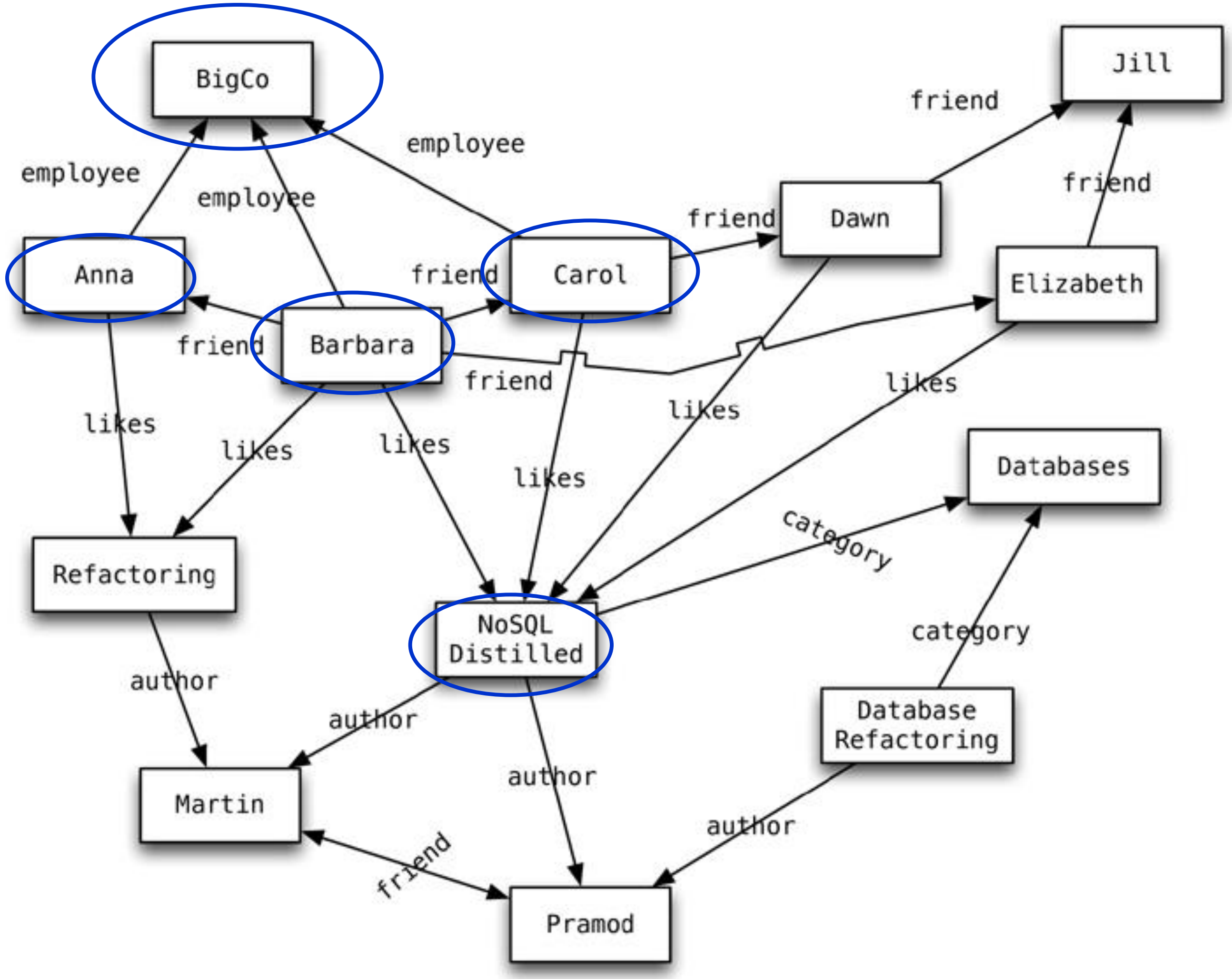
Lotus Notes
Storage Facility

Graph Databases

Basic Characteristics

- To store entities and relationships between these entities
  - Node is an instance of an object
  - Nodes have properties
    - e.g., name
  - Edges have directional significance
  - Edges have types
    - e.g., likes, friend, …
- Nodes are organized by relationships
  - Allow to find interesting patterns
  - e.g., “Get all nodes employed by Big Co that like NoSQL Distilled”

Example:


BigCo
Jill
friend
employee
employee
employee
friend
Dawn
friend
Anna
friend
Carol
Elizabeth
friend
Barbara
friend
likes
likes
likes
likes
likes
likes
likes
Databases
Refactoring
category
NoSQL Distilled
category
author
author
Database Refactoring
Martin
author
author
friend
Pramod

Graph Databases
Representatives

Neo4j
the graph database

InfiniteGraph
The Distributed Graph Database
for the Cloud and Beyond

OrientDB®

FlockDB

Basic Principles

- Scalability
  - How to handle growing amounts of data without losing performance
- CAP theorem
- Distribution models
  - Sharding, replication, consistency, …
  - How to handle data in a distributed manner

Scalability

Vertical Scaling (scaling up)

- Traditional choice has been in favour of <u>strong consistency</u>
  - System architects have in the past gone in favour of scaling up (vertical scaling)
    - Involves larger and more powerful machines
- Works in many cases but…
- Vendor lock-in
  - Not everyone makes large and powerful machines
    - Who do, often use proprietary formats
  - Makes a customer dependent on a vendor for products and services
    - Unable to use another vendor

Scalability

Vertical Scaling (scaling up)

- Higher costs
  - Powerful machines usually cost a lot more than commodity hardware
- Data growth perimeter
  - Powerful and large machines work well until the data grows to fill it
  - Even the largest of machines has a limit
- Proactive provisioning
  - Applications have no idea of the final large scale when they start out
  - Scaling vertically = you need to budget for large scale upfront

Scalability

Horizontal Scaling (scaling out)

- Systems are distributed across multiple machines or nodes (horizontal scaling)
  - Commodity machines, cost effective
  - Often surpasses scalability of vertical approach
- Fallacies of distributed computing:
  - The network is reliable
  - Latency is zero
  - Bandwidth is infinite
  - The network is secure
  - Topology does not change
  - There is one administrator
  - Transport cost is zero
  - The network is homogeneous

https://blogs.oracle.com/jag/resource/Fallacies.html

CAP Theorem

Consistency

- After an update, all readers in a distributed system see the same data
- All nodes are supposed to contain the same data at all times
- Example:
  - A single database instance is always consistent
  - If multiple instances exist, all writes must be duplicated before write operation is completed

CAP Theorem

Availability

- All requests (reads, writes) are always answered, regardless crashes
- Example:
  - A single instance has an availability of 100% or 0%
  - Two servers may be available 100%, 50%, or 0%

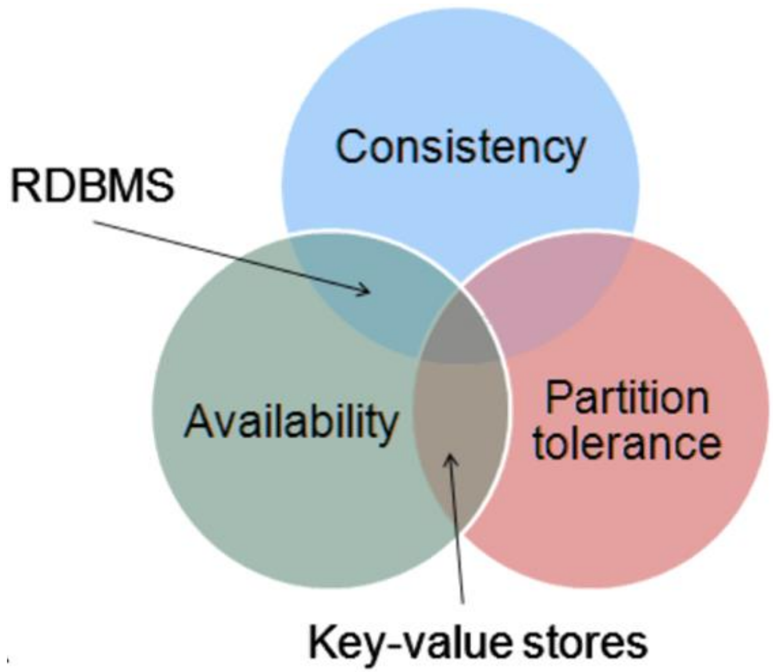
Partition Tolerance

- System continues to operate, even if two sets of servers get isolated
- Example:
  - Failed connection will not cause troubles if the system is tolerant

CAP Theorem

ACID vs. BASE

- **Theorem**: *Only 2 of the 3 guarantees can be given in a "shared-data" system.*
  - Proven in 2000, the idea is older
- (Positive) consequence: we can concentrate on two challenges
- ACID properties guarantee consistency and availability
  - **pessimistic**
  - e.g., database on a single machine
- BASE properties guarantee availability and partition tolerance
  - **optimistic**
  - e.g., distributed databases



CAP Theorem

Consistency

- A single-server system is a CA system
- Clusters have to be tolerant of network partitions
  - CAP theorem: you can only get two out of three
  - Reality: you can trade off a little Consistency to get some Availability
    - It is not a binary decision

BASE

- In contrast to ACID
- Leads to levels of scalability that cannot be obtained with ACID
  - At the cost of (strong) consistency

Basically Available

- The system works basically all the time
- Partial failures can occur, but without total system failure

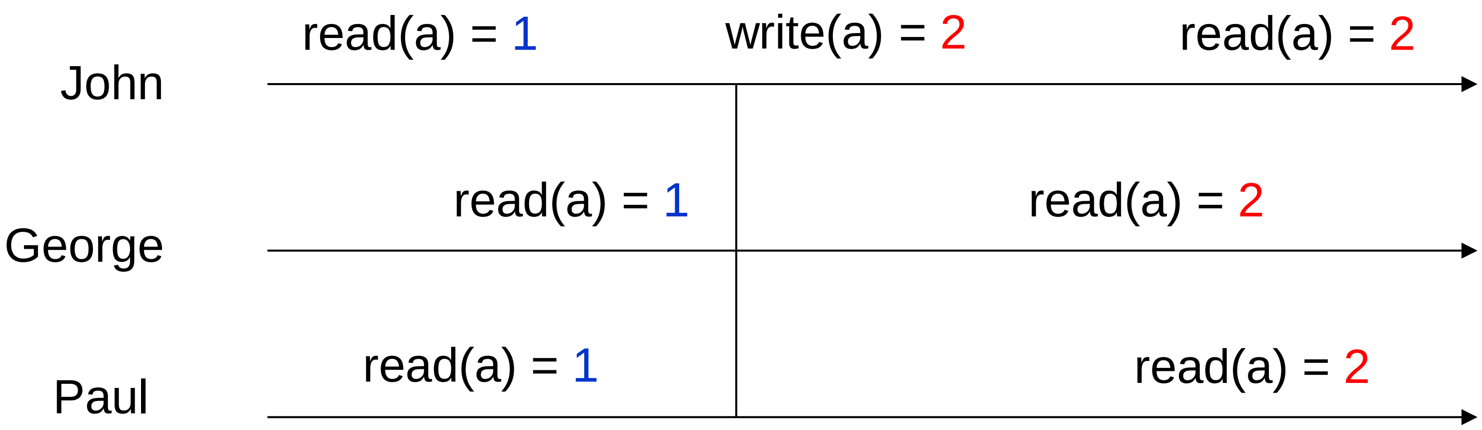
Soft State

- The system is in flux and non-deterministic
- Changes occur all the time

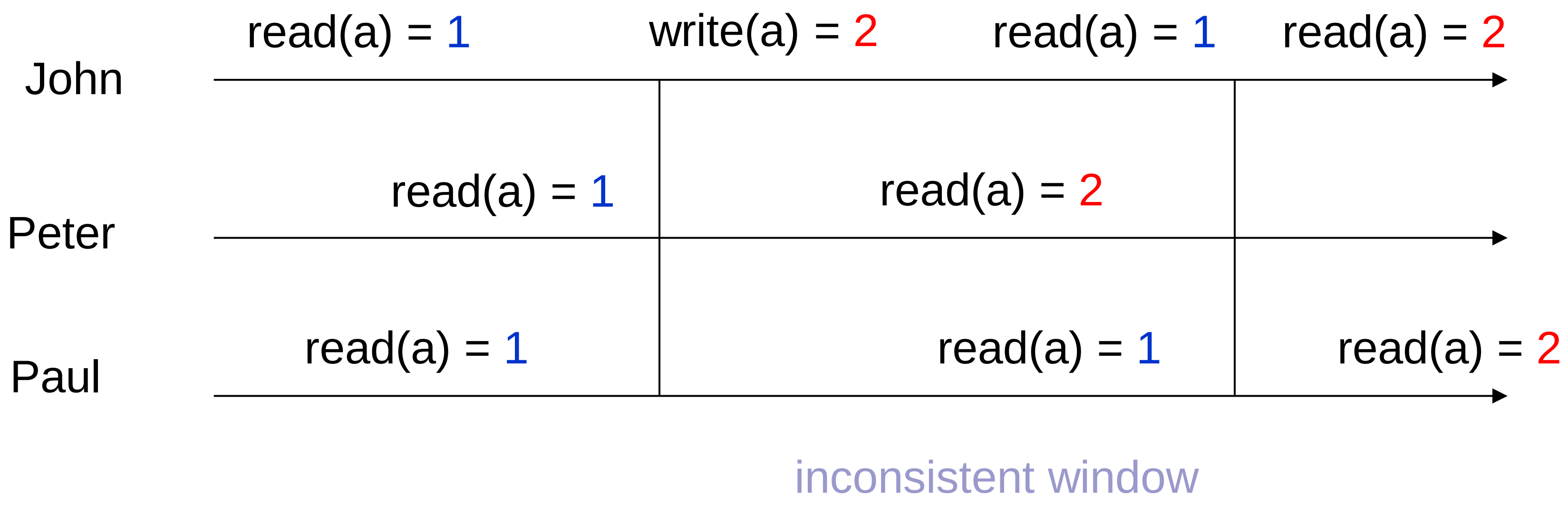
Eventual Consistency

- The system will be in some consistent state
- At some time in future

Strong Consistency



Eventual Consistency



Distribution Models

- Scaling out = running the database on a cluster of servers
- Two orthogonal techniques to data distribution:
  - Replication – takes the same data and copies it over multiple nodes
    - Master-slave or peer-to-peer
  - Sharding – puts different data on different nodes
- We can use either or combine them

Distribution Models

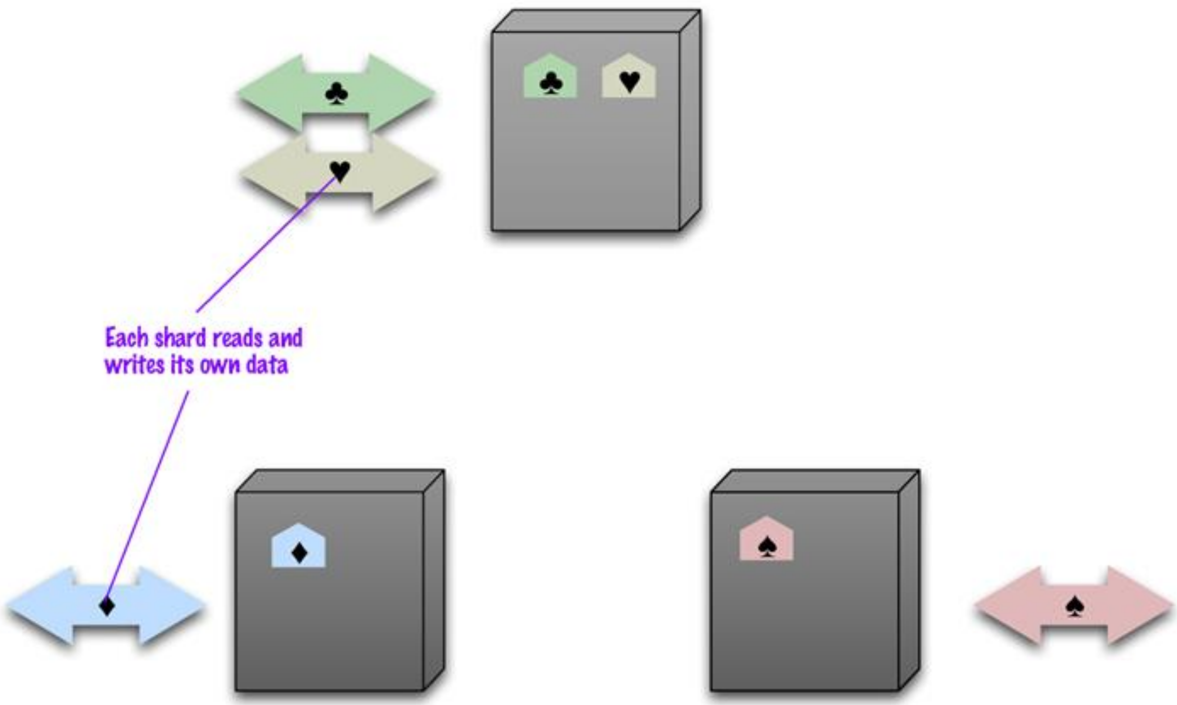
Single Server

- No distribution at all
  - Run the database on a single machine
- It can make sense to use NoSQL with a single-server distribution model
  - Graph databases
    - The graph is "almost" complete → it is difficult to distribute it

Distribution Models

Sharding

- Horizontal scalability → putting different parts of the data onto different servers
- Different people are accessing different parts of the dataset



Distribution Models

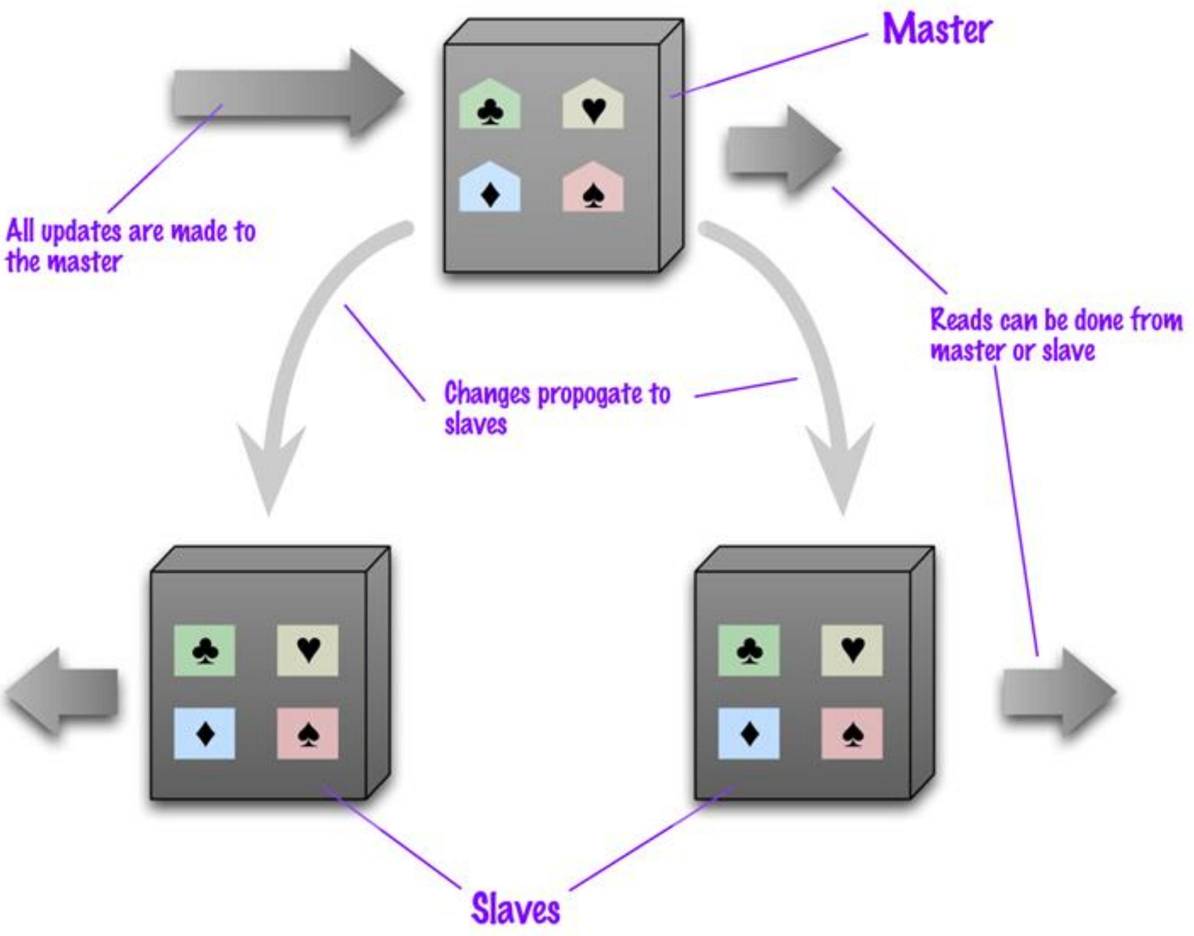
Sharding

- The ideal case is rare
- To get close to it we have to ensure that data that is accessed together is clumped together
- How to arrange the nodes:
  a. One user mostly gets data from a single server
  b. Based on a physical location
  c. Distributed across the nodes with equal amounts of the load
- Many NoSQL databases offer auto-sharding
- A node failure makes shard's data unavailable
  - Sharding is often combined with replication

Distribution Models

Master-slave Replication

- We replicate data across multiple nodes
- One node is designed as primary (master), others as secondary (slaves)
- Master is responsible for processing any updates to that data



Distribution Models

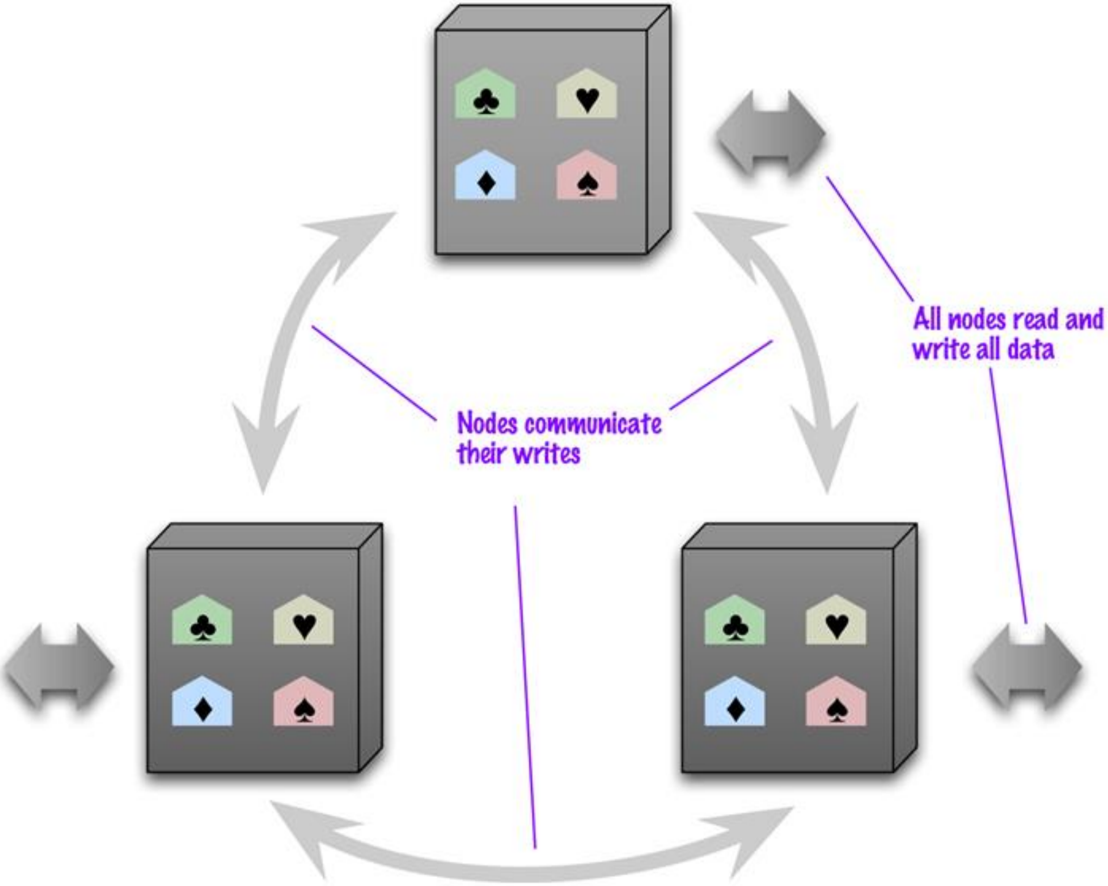
Master-slave Replication

- For scaling a read-intensive dataset
  - More read requests → more slave nodes
  - The master fails → the slaves can still handle read requests
    - A slave can be appointed a new master quickly (it is a replica)
- Limited by the ability of the master to process updates
- Masters are appointed manually or automatically
  - User-defined vs. cluster-elected

Distribution Models

Peer-to-peer Replication

- Problems of master-slave replication:
  - Does not help with scalability of writes
  - Provides resilience against failure of a slave, but not of a master
  - The master is still a bottleneck
- Peer-to-peer replication: no master
  - All the replicas have equal weight



Distribution Models

Peer-to-peer Replication

- Problem: consistency
  - We can write at two different places: a write-write conflict
- Solutions:
  - Whenever we write data, the replicas coordinate to ensure we avoid a conflict
    - At the cost of network traffic
  - But we do not need all the replicas to agree on the write, just a majority

Distribution Models

Combining Sharding and Replication

- Master-slave replication and sharding:
  - We have multiple masters, but each data item only has a single master
  - A node can be a master for some data and a slave for others
- Peer-to-peer replication and sharding:
  - A common strategy for column-family databases
  - A good starting point for peer-to-peer replication is to have a replication factor of 3, so each shard is present on three nodes

Consistency

Write (update) Consistency

- Problem: two users want to update the same record (write-write conflict)
  - Issue: lost update
- Pessimistic (preventing conflicts from occurring) vs. optimistic solutions (let conflicts occur, but detect them and take actions to sort them out)
  - Write locks, conditional update, save both updates and record that they are in conflict, …

Consistency

Read Consistency

- Problem: one user reads, other writes (read-write conflict)
  - Issue: inconsistent read
- Relational databases support the notion of transactions
- NoSQL databases support atomic updates within a single aggregate
  - But not all data can be put in the same aggregate
- Update that affects multiple aggregates leaves open a time when clients could perform an inconsistent read
  - Inconsistency window
- Another issue: replication consistency
  - A special type of inconsistency in case of replication
  - Ensuring that the same data item has the same value when read from different replicas

Consistency

Quorums

- How many nodes need to be involved to get strong consistency?
- Write quorum: $W > N/2$
  - N = the number of nodes involved in replication (replication factor)
  - W = the number of nodes participating in the write
    - The number of nodes confirming successful write
  - “If you have conflicting writes, only one can get a majority.”
- How many nodes you need to contact to be sure you have the most up-to-date change?
- Read quorum: $R + W > N$
  - R = the number of nodes we need to contact for a read
  - „Concurrent read and write cannot happen.“