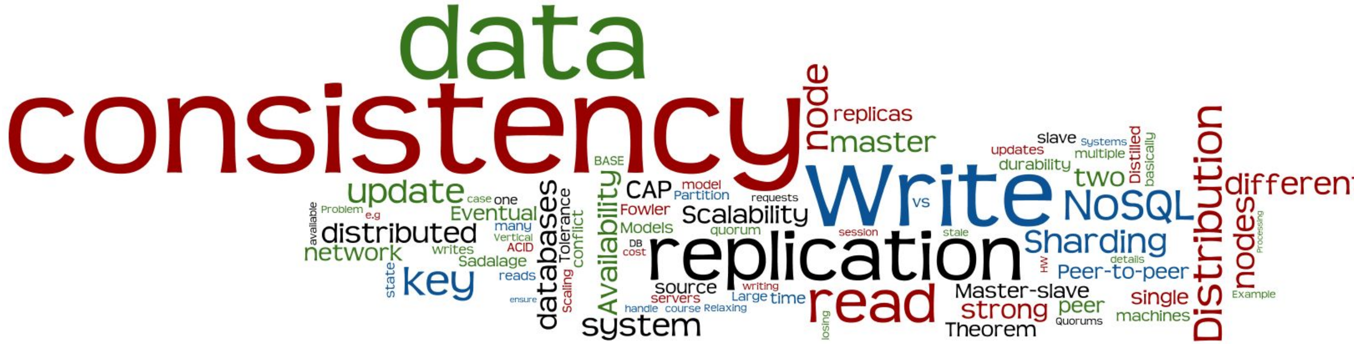


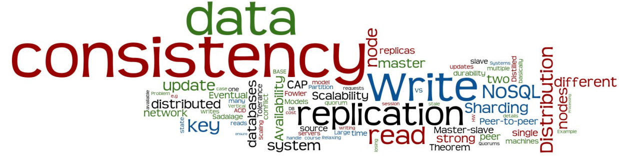
Principles of NoSQL Databases

Data Model, Distribution & Consistency

Lecture 3 of *NoSQL Databases* (PA195)

David Novak & Vlastislav Dohnal
Faculty of Informatics, Masaryk University, Brno

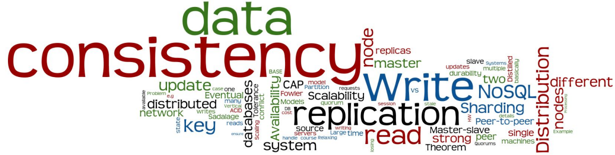
http://disa.fi.muni.cz/vlastislav-dohnal/teaching/nosql-databases-fall-2019/



Agenda

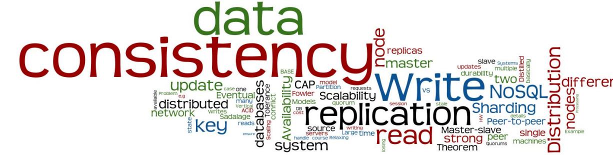
- Fundamentals of RDBMs and NoSQL Databases
- Data Model of Aggregates
- Models of Data Distribution
  - scalability, sharding
  - replication: master-slave, peer-to-peer
  - combination
- Consistency
  - write-write vs. read-write conflict
  - strategies and techniques
  - relaxing consistency

Agenda



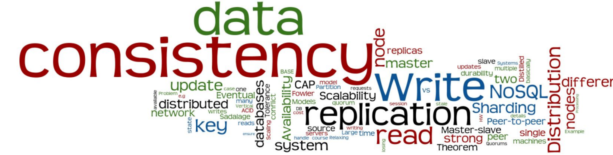
- Fundamentals of RDBMs and NoSQL Databases
- Data Model of Aggregates
- Models of Data Distribution
  - scalability, sharding
  - replication: master-slave, peer-to-peer
  - combination
- Consistency
  - write-write vs. read-write conflict
  - strategies and techniques
  - relaxing consistency

Fundamentals of RDBMS



Relational Database Management Systems (RDMBS)

1. Data structures are broken into the smallest units
    - normalization of database schema (3NF, BCNF)
        - because the data structure is known in advance
        - and users/applications query the data in different ways
    - database schema is rigid
2. Queries merge the data from different tables
3. Write operations are simple, search can be slower
4. Strong guarantees for transactional processing



From RDBMS to NoSQL

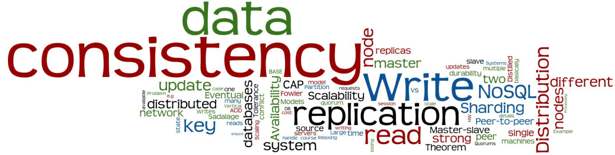
Efficient implementations of table joins and of transactional processing require centralized system.

NoSQL Databases:

- Database schema tailored for specific application
    - keep together data pieces that are often accessed together
- Write operations might be slower but read is fast
- Weaker consistency guarantees

=> efficiency and horizontal scalability

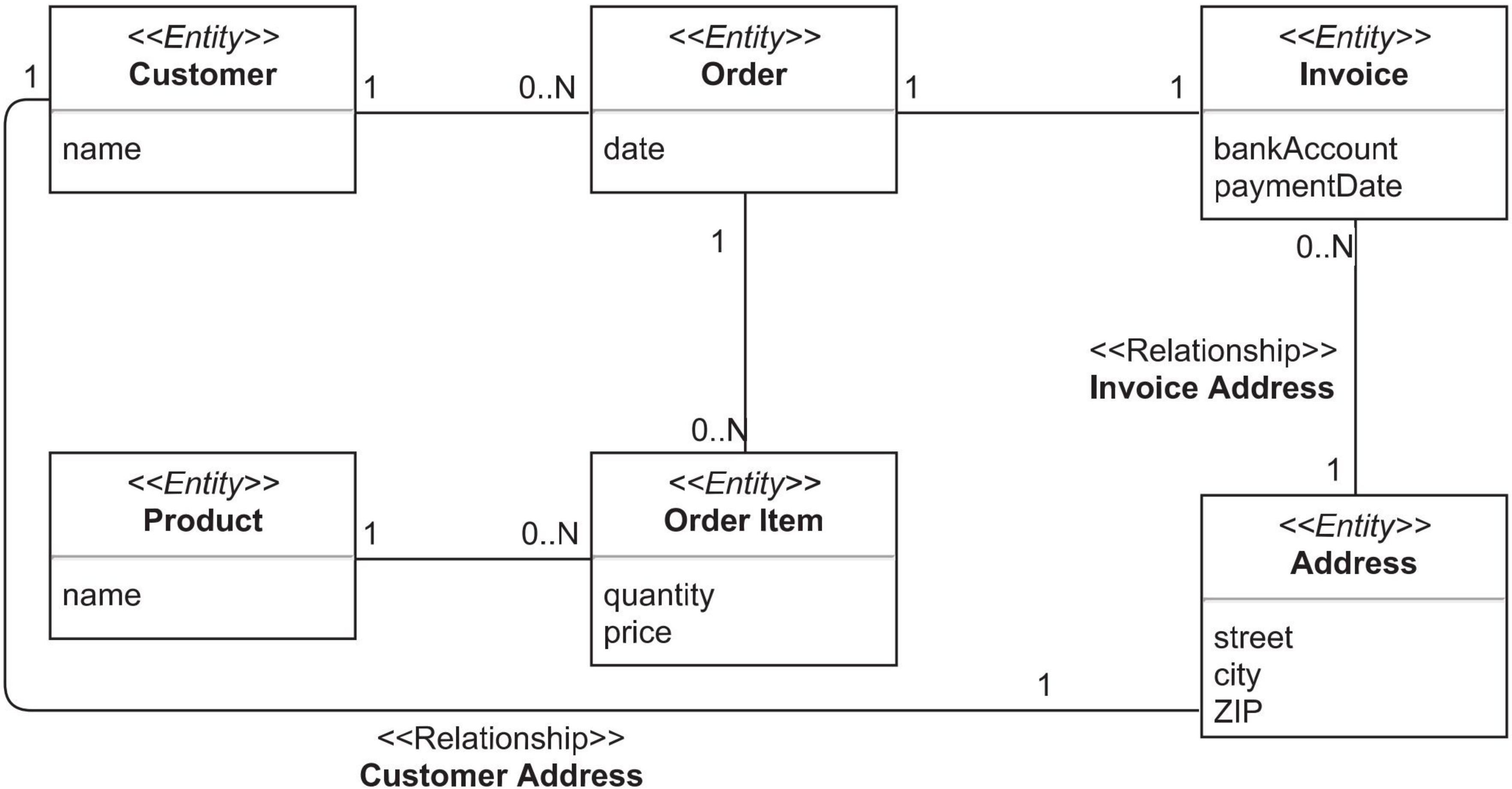
Data Model



- The model by which the database organizes data
- Each NoSQL DB type has a different data model
  - Key-value, document, column-family, graph
  - The first three are oriented on aggregates

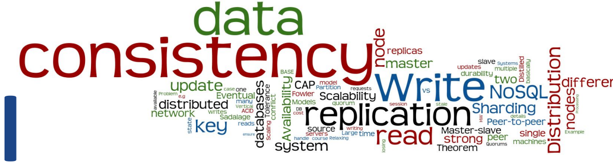
- Let us have a look at the classic relational model

Example (1): UML Model



source: Holubová, Kosek, Minařík, Novák. Big Data a NoSQL databáze. 2015.

Example (2): Relational Model



Customer
<u>customerID</u>
name
addressID (FK)

Order
<u>orderNumber</u>
date
customerID (FK)

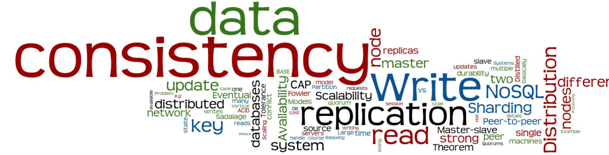
Invoice
<u>invoiceID</u>
bankAccount
paymentDate
addressID (FK)
orderNumber (FK)

Product
<u>productID</u>
name

OrderItem
<u>orderNumber (FK)</u>
<u>productID (FK)</u>
quantity
price

Address
<u>addressID</u>
street
city
ZIP

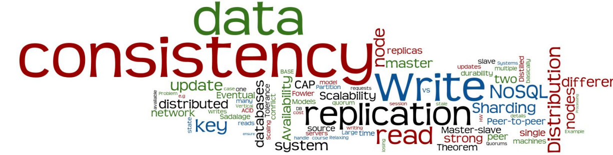
source: Holubová, Kosek, Minařík, Novák. Big Data a NoSQL databáze. 2015.



Agenda

- Fundamentals of RDBMs and NoSQL Databases
- Data Model of Aggregates
- Models of Data Distribution
  - scalability, sharding
  - replication: master-slave, peer-to-peer
  - combination
- Consistency
  - write-write vs. read-write conflict
  - strategies and techniques
  - relaxing consistency

Aggregates

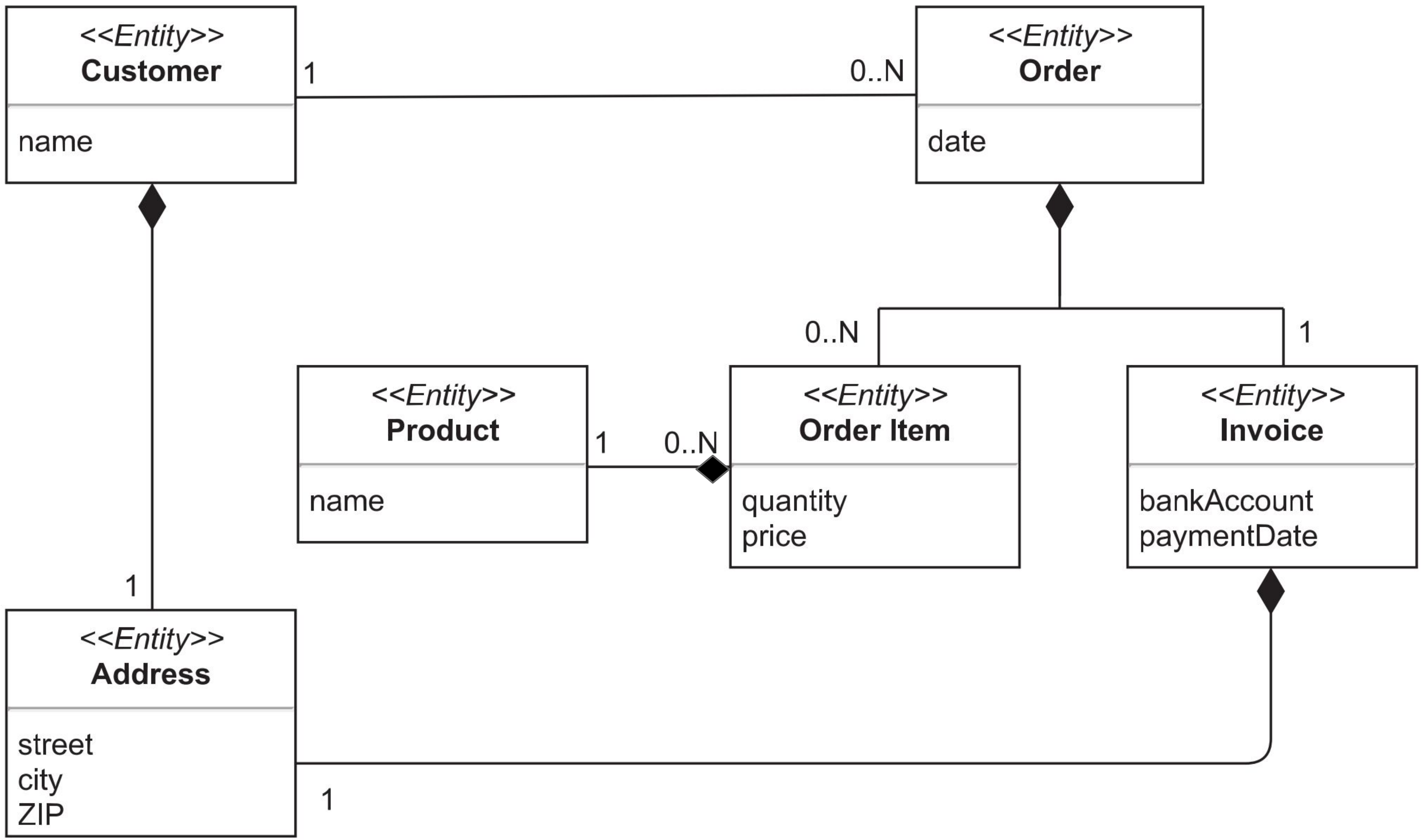


An aggregate

- A data unit with a complex structure
  - Not simply a tuple (a table row) like in RDBMS
- A collection of related objects treated as a unit
  - unit for data manipulation and management of consistency

- Relational model is aggregate-ignorant
  - It is not a bad thing, it is a feature
  - Allows to easily look at the data in different ways
  - Best choice when there is no primary structure for data manipulation

Example (3): Aggregates



source: Holubová, Kosek, Minařík, Novák. Big Data a NoSQL databáze. 2015.

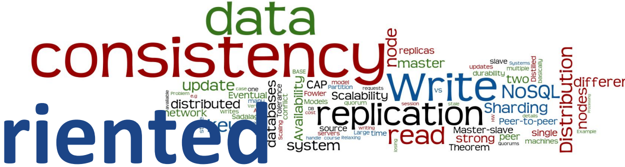
Example (4): Aggregates

```
// collection "Customer"
{
  "customerID": 1,
  "name": "Jan Novák",
  "address": {
    "city": "Praha",
    "street": "Krásná 5",
    "ZIP": "111 00"
  }
}

// collection "Invoice"
{
  "invoiceID": 2015003,
  "orderNumber": 11,
  "bankAccount": "64640439/0100",
  "paymentDate": "2015-04-16",
  "address": {
    "city": "Brno",
    "street": "Slunečná 7",
    "ZIP": "602 00"
  }
}
```

```
// collection "Order"
{
  "orderNumber": 11,
  "date": "2015-04-01",
  "customerID": 1,
  "orderItems": [
    {
      "productID": 111,
      "name": "Vysavač ETA E1490",
      "quantity": 1,
      "price": 1300
    },
    {
      "productID": 112,
      "name": "Sáček k ETA E1490",
      "quantity": 10,
      "price": 300
    }
  ],
  "invoice": { "bankAccount": …, …}
}
```

NoSQL Databases: Aggregate-oriented

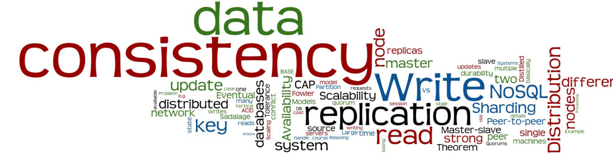


Many NoSQL stores are aggregate-oriented:

- There is no general strategy to set aggregate boundaries
- Aggregates give the database information about which bits of data will be manipulated together
  - What should be stored on the same node

- Minimize the number of nodes accessed during a search
- Impact on concurrency control:
  - NoSQL databases typically support atomic manipulation of a single aggregate at a time

Agenda



- Fundamentals of RDBMs and NoSQL Databases
- Data Model of Aggregates
- Models of Data Distribution
    - scalability, sharding
    - replication: master-slave, peer-to-peer
    - combination
- Consistency
    - write-write vs. read-write conflict
    - strategies and techniques
    - relaxing consistency

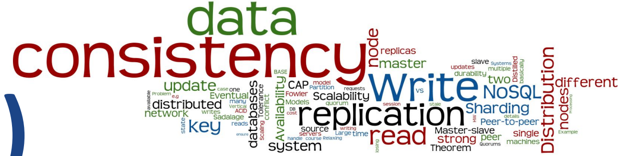
Scalability of Database Systems

- Scalability = handling growing amounts of data and queries without losing performance

Two general approaches:

- vertical scalability
- horizontal scalability

Vertical Scalability (Scaling up)

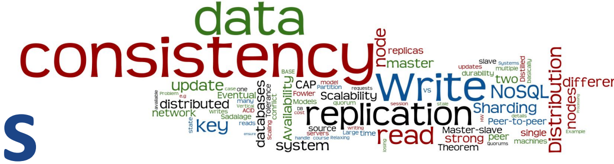


- Involve larger and more powerful machines
  - large disk storage using disk arrays
  - massively parallel architectures
  - large main memories

- Traditional choice
  - in favour of strong consistency
  - very simple to realize (no handling of data distribution)

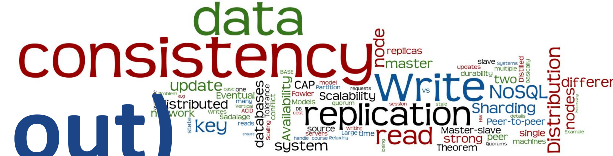
- Works in many cases but…

Vertical Scalability: Drawbacks



- Higher costs
  - Large machines cost more than equivalent commodity HW
- Data growth limit
  - Large machine works well until the data grows to fill it
  - Even the largest of machines has a limit
- Proactive provisioning
  - In the beginning, no idea of the final scale of the application
  - An upfront budget is needed when scaling vertically
- Vendor lock-in
  - Large machines are produced by a few vendors
  - Customer is dependent on a single vendor (proprietary HW)

Horizontal Scalability (Scaling out)

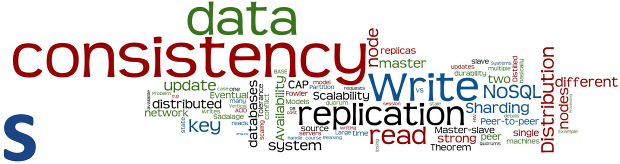


System is distributed across multiple machines/nodes

- Commodity machines, cost effective
- Provides higher scalability than vertical approach
  - Data is partitioned over many disks
  - Application can use main memory of all machines
  - Distribution computational model

- Introduces new problems:
  - synchronization, consistency, partial failures handling, etc.

Horizontal Scalability: Fallacies



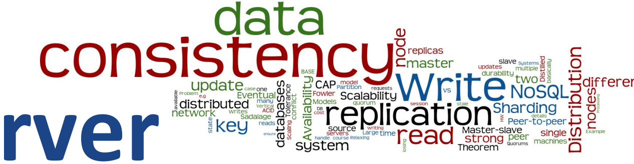
- Typical false assumptions of distributed computing:
    - The network is reliable
    - Latency is zero
    - Bandwidth is infinite
    - The network is secure
    - The network is homogeneous
    - Topology of the network does not change
    - There is one network administrator

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

Distribution Models: Overview

- Horizontal scalability = scaling out
- Two generic ways of data distribution:
  - Replication – the same data is copied over multiple nodes
    - Master-slave vs. peer-to-peer
  - Sharding – different data chunks are put on different nodes (data partitioning)
    - Master-master
- We can use either or combine them
  - Distribution models = specific ways to do sharding, replication or combination of both

Distribution Model: Single Server



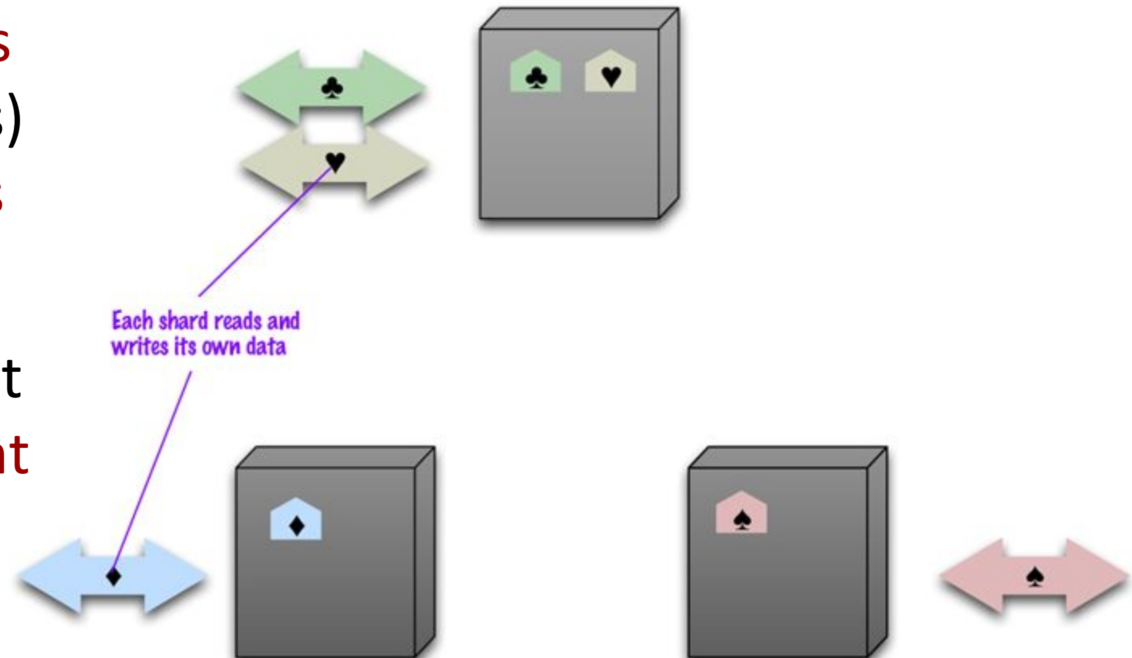
- Running the database on a single machine is always the preferred scenario
  - it spares us a lot of problems

- It can make sense to use a NoSQL database on a single server
  - Other advantages remain: Flexible data model, simplicity
  - Graph databases: If the graph is “almost” complete, it is difficult to distribute it

Sharding (Data Partitioning)

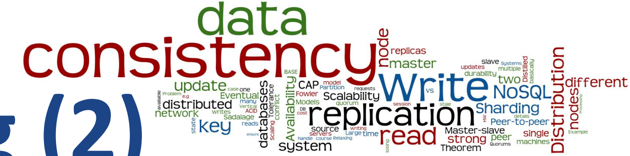
- Placing different parts of the data (card suits) onto different servers

- Applicability: Different clients access different parts of the dataset



source: Sadalage & Fowler: NoSQL Distilled, 2012

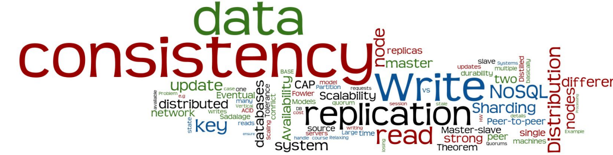
Distribution Models: Sharding (2)



We should try to ensure that

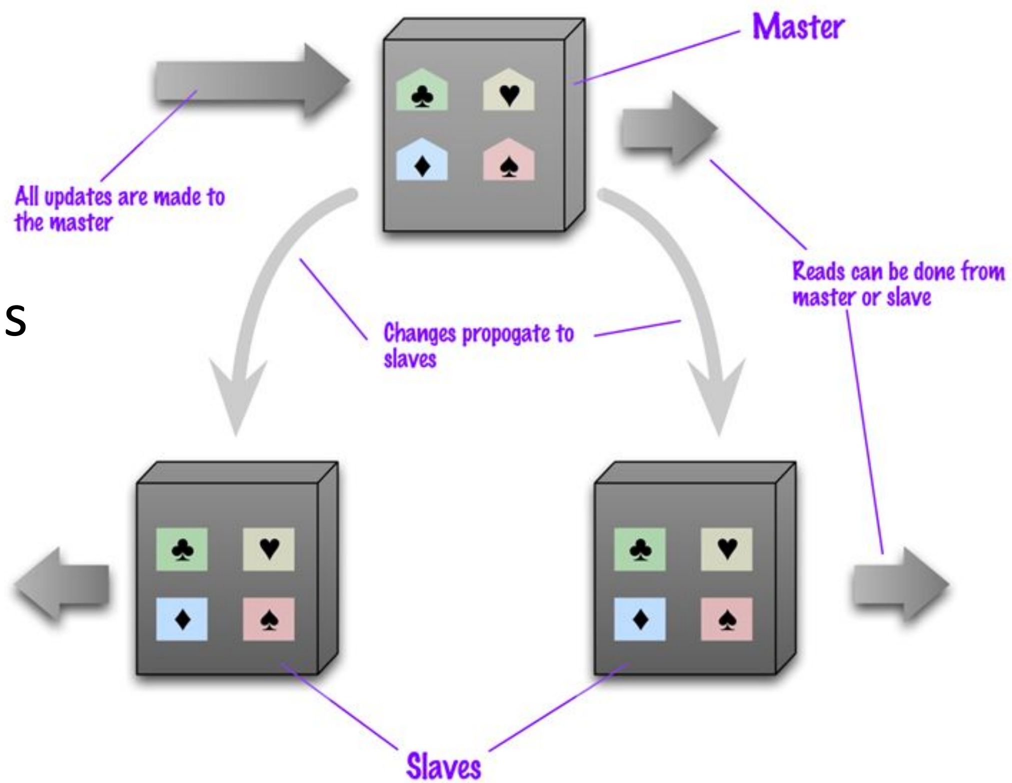
1. Data accessed together is kept together
   - So that user gets all data from a single server
   - Aggregates data model helps achieve this
2. Arrange the data on the nodes:
   - Keep the load balanced (can change in time)
   - Consider the physical location (of the data centers)

- Many NoSQL databases offer auto-sharding
- A node failure makes shard’s data unavailable
   - Sharding is often combined with replication



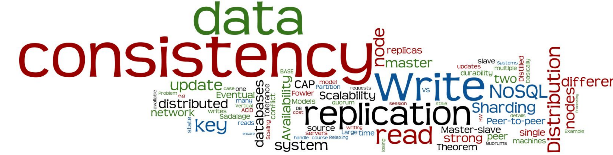
Master-slave Replication

- We replicate data across multiple nodes
- One node is designated as primary (master), others as secondary (slaves)
- Master is responsible for processing all updates to the data
- Reads from any node



source: Sadalage & Fowler: NoSQL Distilled, 2012

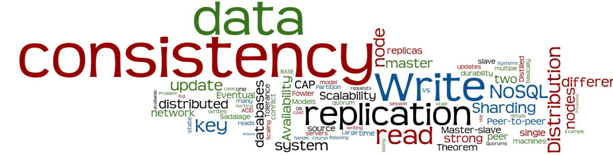
Master-slave Replication (2)



- For scaling a read-intensive application
  - More read requests → more slave nodes
  - The master fails → the slaves can still handle read requests
  - A slave can become a new master quickly (it is a replica)

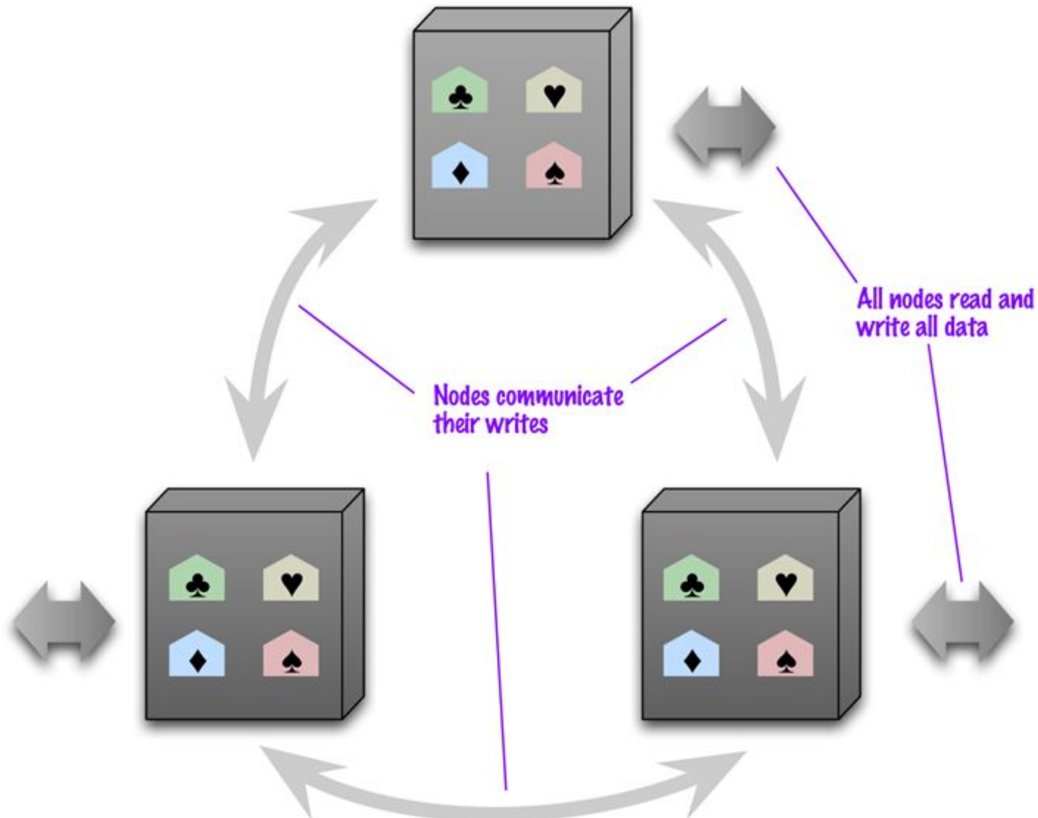
- Limited by ability of the master to process updates

- Masters are selected manually or automatically
  - User-defined vs. cluster-elected



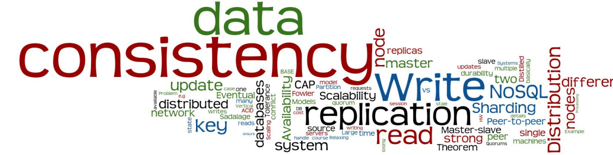
Peer-to-peer Replication

- No master, all the replicas are equal
- Every node can handle a write and then spreads the update to the others



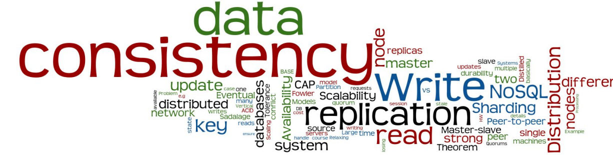
source: Sadalage & Fowler: NoSQL Distilled, 2012

Peer-to-peer Replication (2)



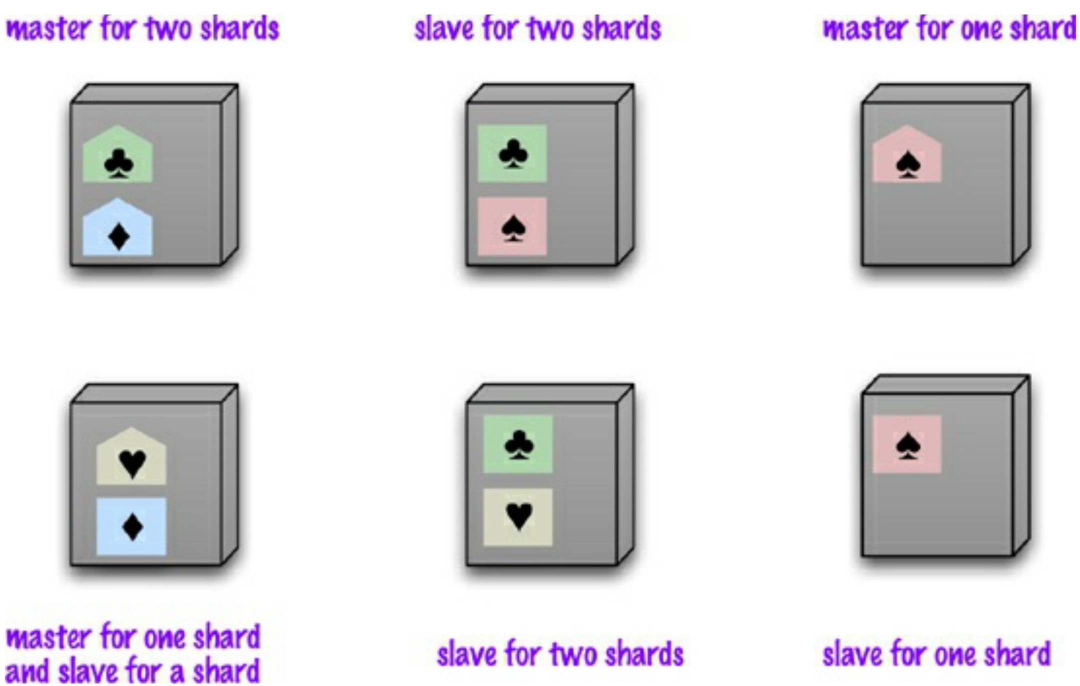
- Problem: consistency
  - Users can write simultaneously at two different nodes

- Solution:
  - When writing, the replicas coordinate to avoid conflict
    - At the cost of network traffic
    - The write operation waits till the coordination process is finished
  - Not all replicas need to agree on the write, just a majority (details below)

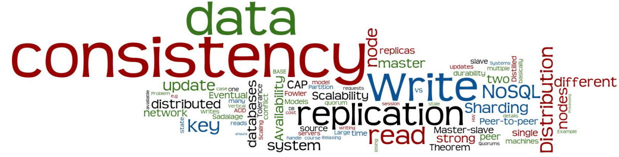


Sharding & Replication (1)

- Sharding and master-slave replication:
  - Each data shard is replicated (via a single master)
  - A node can be a master for some data and a slave for other

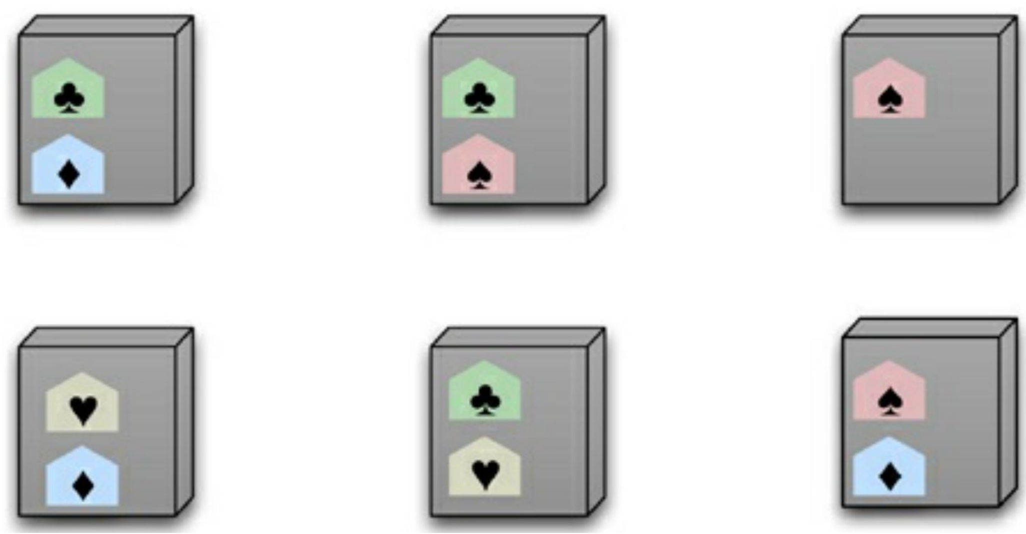


source: Sadalage & Fowler: NoSQL Distilled, 2012



Sharding & Replication (2)

- Sharding and peer-to-peer replication:
  - A common strategy for column-family databases
  - A typical default is replication factor of 3
    - each shard is present on three nodes

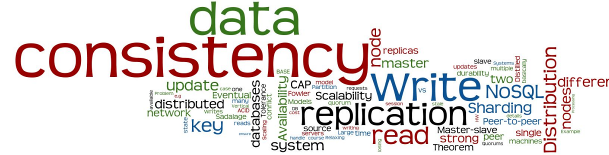


=> we have to solve consistency issues

(let's first talk more about what consistency means)

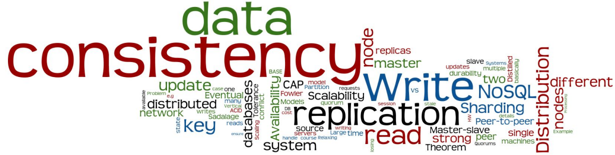
source: Sadalage & Fowler: NoSQL Distilled, 2012

Agenda



- Fundamentals of RDBMs and NoSQL Databases
- Data Model of Aggregates
- Models of Data Distribution
  - scalability, sharding
  - replication: master-slave, peer-to-peer
  - combination
- Consistency
  - write-write vs. read-write conflict
  - strategies and techniques
  - relaxing consistency

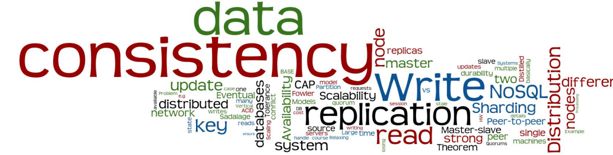
Consistency in Databases



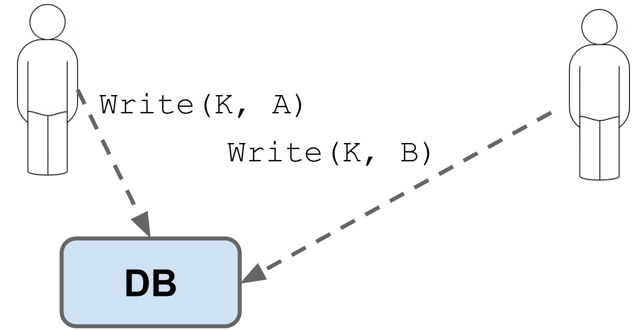
- "Consistency is the lack of contradiction in the DB"
- Centralized RDBMS ensure strong consistency

- Distributed NoSQL databases typically relax consistency (and/or durability)
  - Strong consistency → eventual consistency
  - BASE (basically available, soft state, eventual consistency)
  - CAP theorem
  - tradeoff between consistency and availability

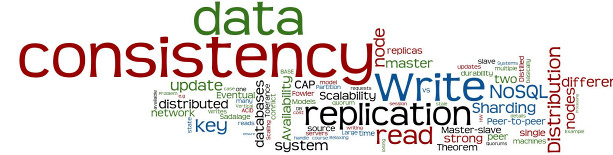
Write (Update) Consistency



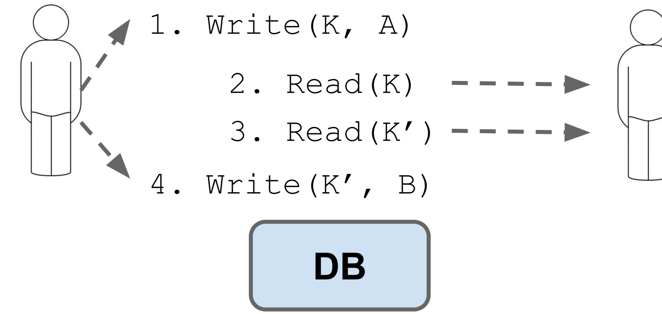
- Problem: two users want to update the same record (write-write conflict)
  - Issues: lost update, second update is based on stale data



- Two general solutions
  - Pessimistic approach: preventing conflicts from occurring
    - acquiring write locks before update
  - Optimistic approach: lets conflicts occur, but detects them and takes actions to resolve them
    - conditional update, save both updates and record the conflict
    - implementation by, e.g., version stamps (details later in the course)

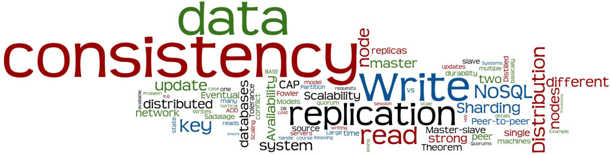


Read Consistency



- Problem: one user reads in the middle of other user’s writes (read-write conflict, inconsistent read)
  - this leads to *logical inconsistency*

- Ideal solution: transactions (ACID)
  - strong consistency

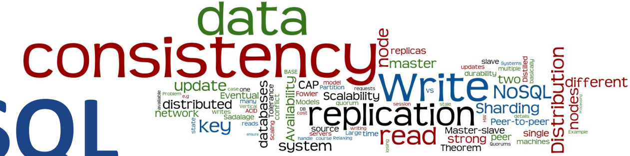


Read Consistency in NoSQL

- NoSQL databases inherently support atomic updates only within a single aggregate
  - Update that affects multiple aggregates leaves a time slot when clients could perform an inconsistent read
  - Inconsistency window

- Graph Databases
  - Typically strong consistency (if centralized)

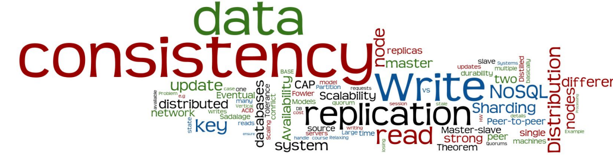
Transaction Processing in NoSQL

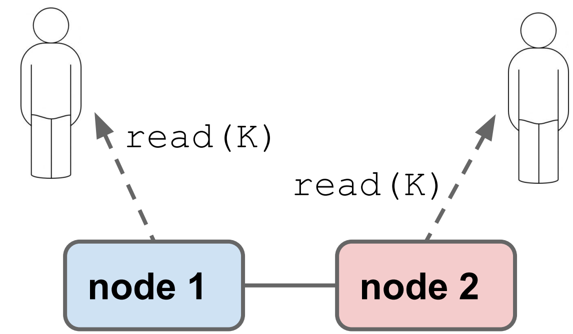


- Basically, no problem if the DB is centralized
  - ACID can be implemented
  - Various levels of isolation (details later in the course)
    - read uncommitted
    - read committed
    - repeatable reads
    - serializable

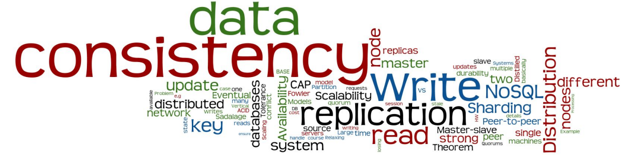
- Distributed transactions (details later in the course)
  - X/Open Distributed Trans. Processing Model (X/Open XA)
  - Two-phase Commit Protocol (2PC)
  - Strong Strict Two-phase Locking (SS2PL)

Replication Consistency





- Consistency among replicas
  - Ensuring that the same data item has the same value when reading from different replicas
- After some time, the write propagates everywhere
  - Eventual consistency, in the meanwhile: stale data
  - Various levels of consistency (e.g. quorums - see below)
- Read-your-writes (session consistency)
  - Is violated if one user writes and reads on different replicas
  - Solution: sticky session (session affinity)

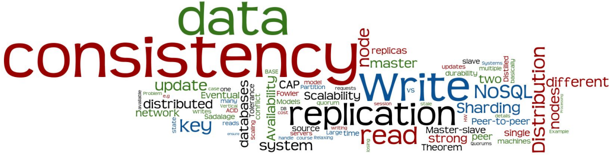


CAP Theorem

CAP = Consistency, Availability, Partition Tolerance

Consistency

- After an update, all readers in a distributed system (assuming replication) see the same data
- Example:
  - A single server database is always consistent
  - If the replication factor > 1, the system must handle the writes and/or reads in a special way



CAP Theorem (2)

Availability

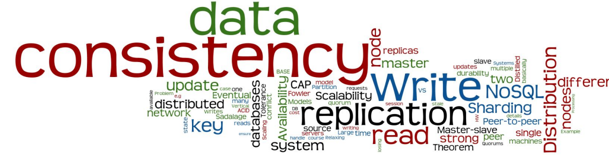
- Every request must result in a response
  - If a node (server) is working, it can read and write data

Partition Tolerance

- System continues to operate, even if two sets of servers get isolated
  - A connection failure should not shut the system down

It would be great to have all these three CAP properties!

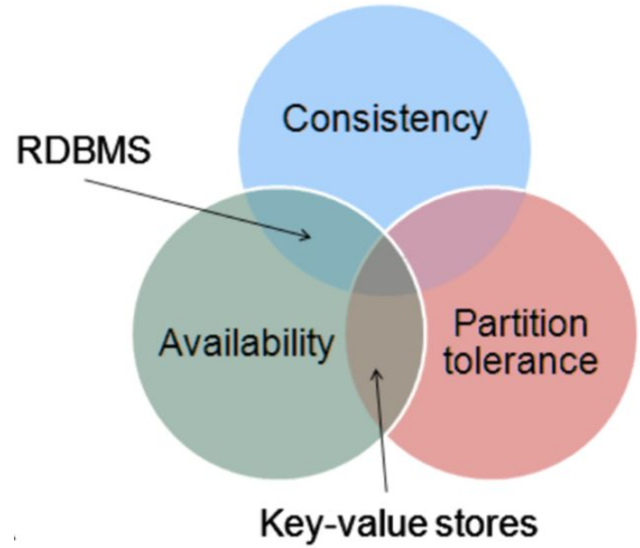
CAP Theorem: Formulation



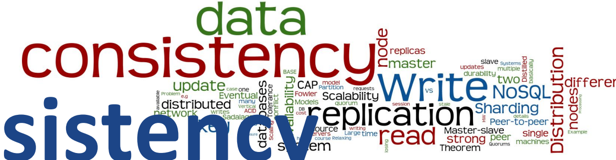
- CAP Theorem: A “shared-data” system cannot have all three CAP properties
  - Or: only two of the three CAP properties are possible
    - This is the common version of the theorem
- First formulated in 2000: prof. Eric Brewer
  - PODC Conference Keynote speech
    - www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
- Proven in 2002: Seth Gilbert & Nancy Lynch
  - SIGACT News 33(2) http://dl.acm.org/citation.cfm?id=564601

CAP Theorem: Real Application

- A single-server system is always CA
  - As well as all ACID systems

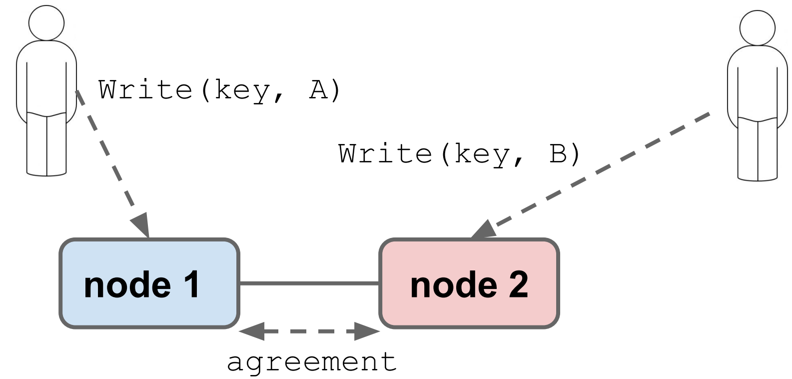


- A distributed system practically has to be tolerant of network Partitions (P)
  - because it is difficult to detect all network failures
- So, tradeoff between Consistency and Availability
  - in fact, it is not a binary decision



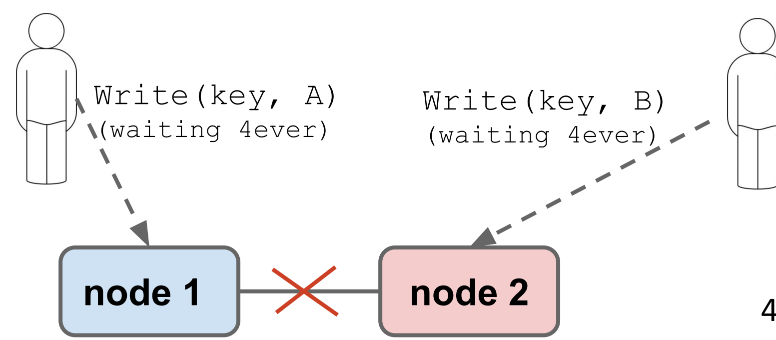
PC: Partition Tolerance & Consistency

Example: two users, two nodes, two write attempts



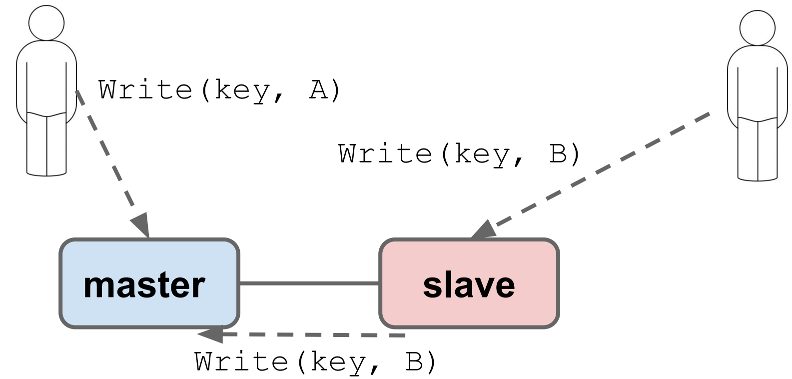
- Strong consistency:
  - Before the write is committed, both nodes have to agree on the order of the writes

- If the nodes are partitioned, we are losing Availability
  - (but reads are still available)

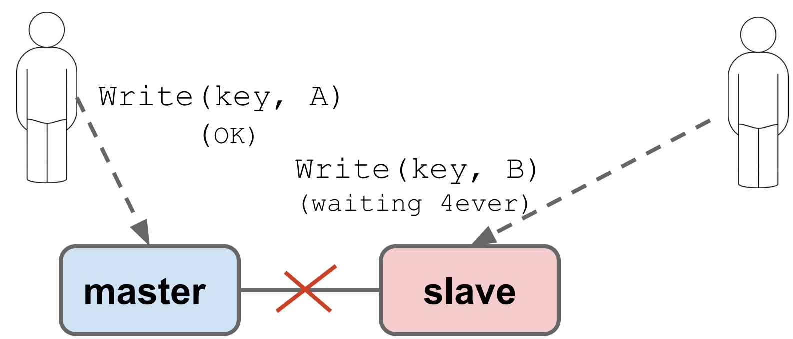


PC: Partition Tolerance & Consistency (2)

- Adding some availability:
  - Master-slave replication

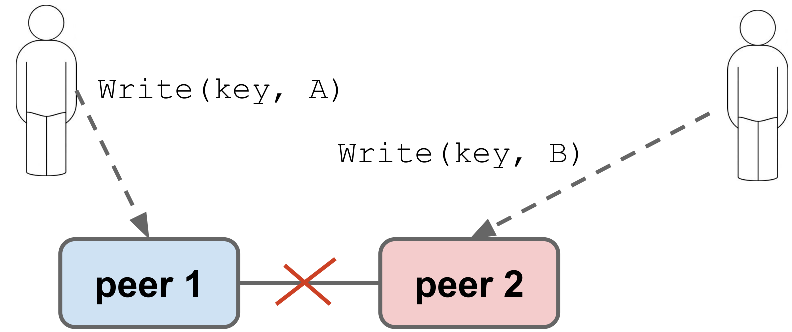


- In case of partitioning, master can commit write
  - Losing some Consistency: Data on slave will be stale for read



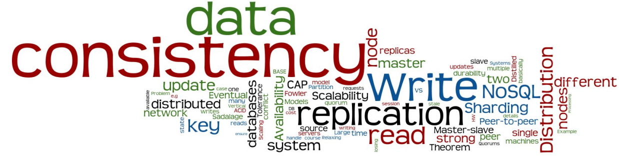
PA: Partition Tolerance & Availability

- Choosing Availability:
  - Peer-to-peer replication
  - Eventual consistency



- In case of Partitioning
  - All requests are answered (full Availability)
  - We risk losing consistency guarantees completely

- But we can do something in the middle: Quorums

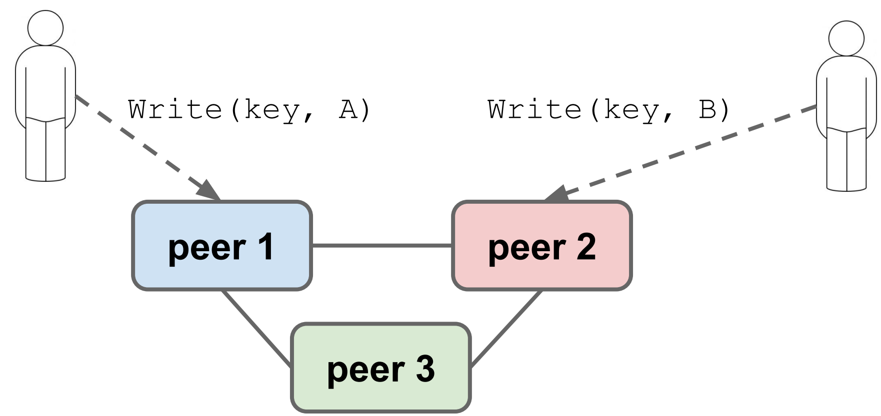


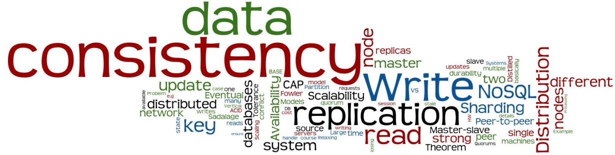
Quorums

- Peer-to-peer replication with replication factor *N*
  - Number of replicas of each data object
- Write quorum: *W*
  - When writing, at least *W* replicas have to agree
  - Having $W > N/2$ results in write consistency
    - in case of two simultaneous writes, only one can get the majority

Example:

- Replication factor $N = 3$
- Write quorum: $W = 2$
  ($W > N/2$)



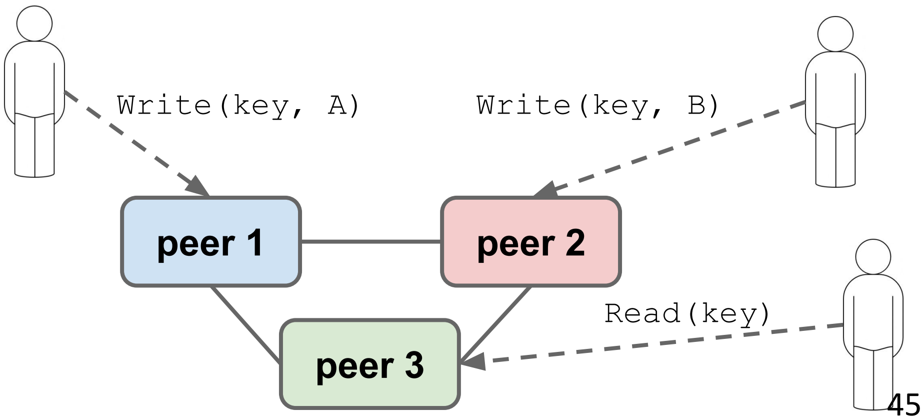


Quorums (2)

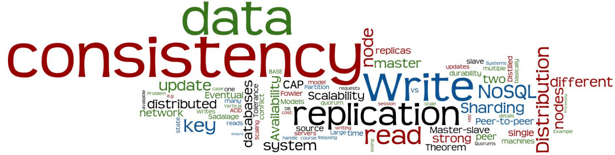
- Read quorum: *R*
  - Number of peers contacted for a single read
    - Assuming that each value has a time stamp (time of write) to tell the older value from the newer
  - For a strong read consistency: $R + W > N$
    - reader surely does not read stale data

Example:

- Read quorum: $R = 2$
  ($R + W > N$)
- 2 nodes contacted for read
  => the newest data returned



Relaxing Durability

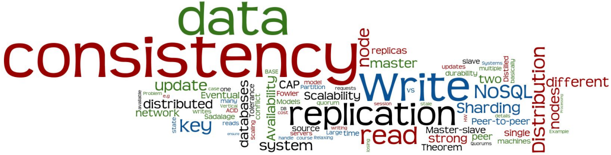


Durability:

- When Write is committed, the change is permanent
- In some cases, strict durability is not essential and it can be traded for scalability (write performance)
  - e.g., storing session data, collection sensor data

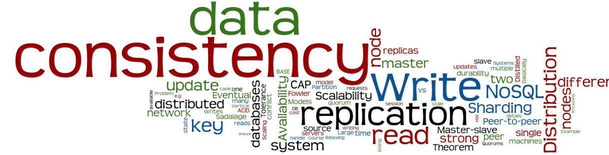
A simple way to relax durability:

- Store data in memory and flush to disk regularly
  - if the system shuts down, we loose updates in memory



Relaxing Durability II

- **Replication durability** (of a write operation)
  - The writing node can either
    1. acknowledge (answer) the write operation immediately
       - not wait until spread to other replicas
       - if the writing node crashes before spreading, durability fails
       - write-behind (write-back)
    2. or it can first spread the update to other replicas
       - operation is answered only after acknowledgement from the others
       - write-through
  - both variants are possible for P2P repl., master-slave replication, quora...



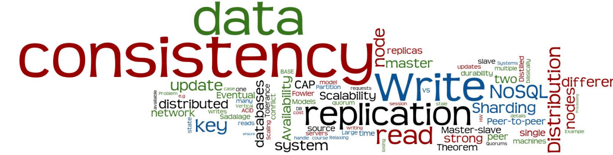
BASE Concept

BASE is a vague term often used as contrast to ACID

- 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 (unstable), non-deterministic state
  - Changes occur all the time
- Eventual consistency
  - The system will be in some consistent state
  - At some time in future

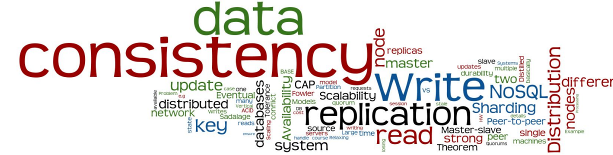
source: Eric Brewer: Towards Robust Distributed Systems. www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

Summary of the Lesson



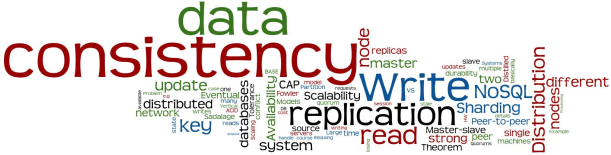
- Aggregate-oriented data modelling
- Sharding vs. replication
  - Master-slave vs. peer-to-peer replication
    - Combination of sharding & replication
- Database consistency:
  - Write/Read consistency (write-write & write-read conflict)
    - Replication consistency (also, read-your-own-writes)
- Relaxing consistency:
  - CAP (Consistency, Availability, Tolerance to Partitions),
    - Eventual consistency
  - Quoras (write/read quorum)
    - can ensure strong replication consistency; wide range of settings

Conclusions



- There is a wide range of options influencing
  - Scalability
    - of data storage, of read operations, of update (write) requests
  - Availability
    - How the system behaves in case of HW (e.g. network) failure
  - Consistency
    - Consistency has many facets and it depends how important they are
  - Durability
    - Can I rely on confirmed updates (and is it so important)?
  - Fault-tolerance
    - Do I have copies of data to recover after a complete HW fail?
- It’s good to know the options and choose wisely

References



- I. Holubová, J. Kosek, K. Minařík, D. Novák. Big Data a NoSQL databáze. Praha: Grada Publishing, 2015. 288 p.
- Sadalage, P. J., & Fowler, M. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional, 192 p.
- RNDr. Irena Holubova, Ph.D. MMF UK course NDBI040: Big Data Management and NoSQL Databases
- Eric Brewer: Towards Robust Distributed Systems. www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf