FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
INSTITUT FÜR INFORMATIK

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LEHRSTUHL FÜR DATENBANKSYSTEME UND DATA MINING

Lecture Notes to

Big Data Management and Analytics

Winter Term 2018/2019

NoSQL Databases

DBS

NoSQL Database Systems

Outline

- History
- Concepts
  - ACID
  - BASE
  - CAP
- Data Models
  - Key-Value Stores
  - Document Databases
  - Wide Column Stores
  - Graph Databases

History

60s: IBM developed the Hierarchical Database Model

- Tree-like structure
- Data stored as *records* connected by *links*
- Support only one-to-one and one-to-many relationships

Mid 80‘s: Rise of Relational Database Model

- Data stored in a collection of tables (rows and columns)
  → Strict relational scheme
- SQL became standard language (based on relational algebra)

→ Impedance Mismatch!

History – Impedance Mismatch

Supply:
- Supplier:
  - SNR: L1
  - Sname: Meier
  - status: 20
  - location: Wetter
- Project:
  - PNR: P2
  - Pname: Pleite
  - location: Bonn
- Parts:
  - ONR: T6
  - Oname: screw
  - color: rot
  - weight: 03
  - amount: 700

SNR	Sname	Status	location
…	…	…	…
…	…	…	…
…	…	…	…

PNR	Pname	Ort
…	…	…
…	…	…
…	…	…

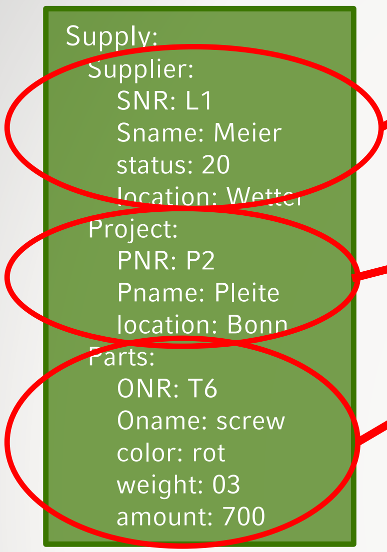
ONR	Oname	color	weight
…	…	…	…
…	…	…	…
…	…	…	…

SNR	PNR	ONR	amount
…	…	…	…
…	…	…	…
…	…	…	…

Given the following database scheme and an object of type Supply:

How to incorporate the data bundled in the object Supply into the relational DB?

History – Impedance Mismatch



SNR	Sname	Status	location
…	…	…	…
…	…	…	…
…	…	…	…

PNR	Pname	location
…	…	…
…	…	…
…	…	…

ONR	Oname	color	weight
…	…	…	…
…	…	…	…
…	…	…	…

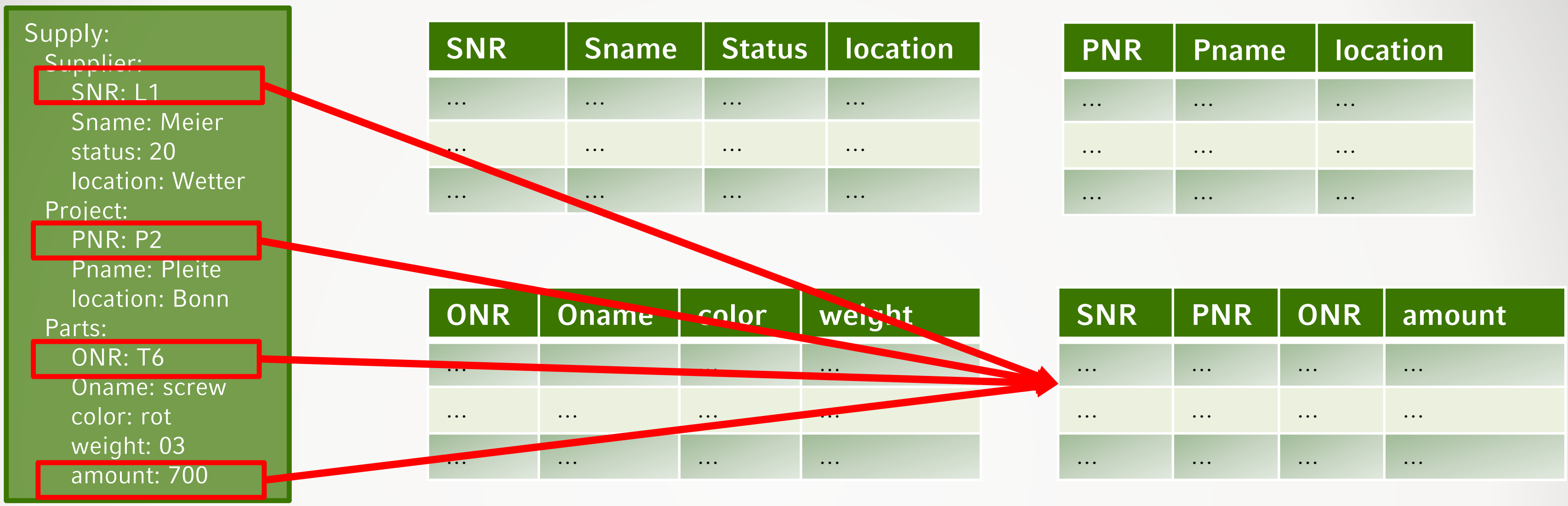
SNR	PNR	ONR	amount
…	…	…	…
…	…	…	…
…	…	…	…

```
INSERT INTO L VALUES (Supply.getSupplier().getLNR(), ...);

INSERT INTO P VALUES (Supply.getProject().getPNR(), ...);

...
```

History – Impedance Mismatch




```
INSERT INTO LTP VALUES (...);
```

- Object-oriented encapsulation vs. storing data distributed among several tables

  → Lots of data type maintenance by the programmer

History

Mid 90‘s: Trend of the Object-Relational Database Model

- data stored as objects (including data and methods)
- avoids of object-relational mapping
  → Programmer-friendly
- but still Relational Databases prevailed in the 90‘s

Mid 2000‘s: Rise of Web 2.0

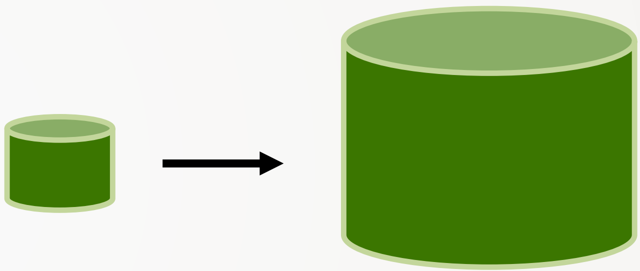
- Lots of user generated data through web applications

  → storage systems had to become scaled up

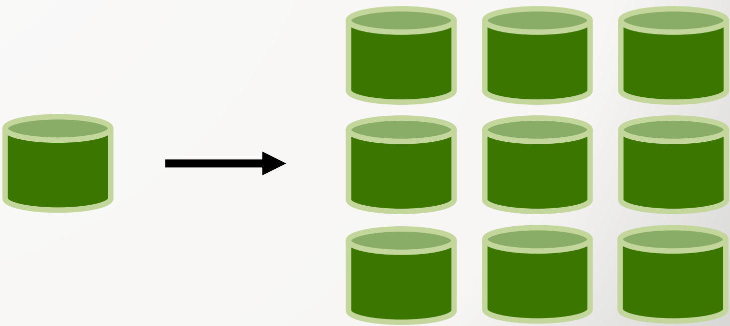
History

Approaches to scale up storage systems

- Two opportunities to solve the rising storage system:
  - Vertical scaling
    Enlarge a single machine
    - Limited in space
    - Expensive



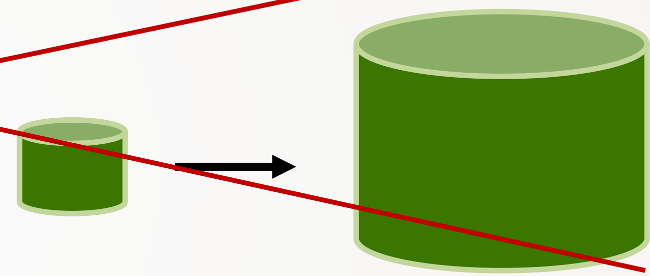
  - Horizontal scaling
    Use many commodity machines and form *computer clusters* or *grids*
    - Cluster maintenance



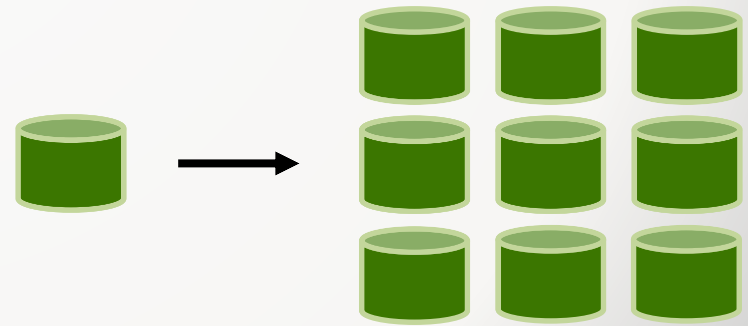
History

Approaches to scale up storage systems

- Two opportunities to solve the rising storage system:
  - ~~Vertical scaling~~
    ~~Enlarge a single machine~~
    - ~~Limited in space~~
    - ~~Expensive~~



  - Horizontal scaling
    Use many commodity ma-chines and form *computer clusters* or *grids*
    - Cluster maintenance



History

Mid 2000's: Birth of the NoSQL Movement

- Problem of computer clusters:
  Relational databases do not scale well horizontally

→ Big Players like Google or Amazon developed their own storage systems: NoSQL („Not-Only SQL") databases were born

Today: Age of NoSQL

- Several different NoSQL systems available (>225)

Google Bigtable | redis | neo4j | HYPERTABLE INC
Cassandra | amazon DynamoDB | mongoDB | APACHE HBASE

Characterstics of NoSQL Databases

There is no unique definition but some characteristics for NoSQL Databases:

- Horizontal scalability (cluster-friendliness)
- Non-relational
- Distributed
- Schema-less
- Open-source (at least most of the systems)

About the concepts behind NoSQL Databases

ACID – The holy grail of RDBMSs:

- **A**tomicity: Transactions happen entirely or not at all. If a transaction fails (partly), the state of the database is unchanged.
- **C**onsistency: Any transaction brings the database from one valid state to another and does not break one of the pre-defined rules (like constraints).
- **I**solation: Concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially.
- **D**urability: Once a transaction has been commited, it will remain so.

About the concepts behind NoSQL Databases

BASE – An artificial concept for NoSQL databases:

- <u>**B**asically **A**vailable:</u> The system is generally available, but some data might not at any time (e.g. due to node failures)

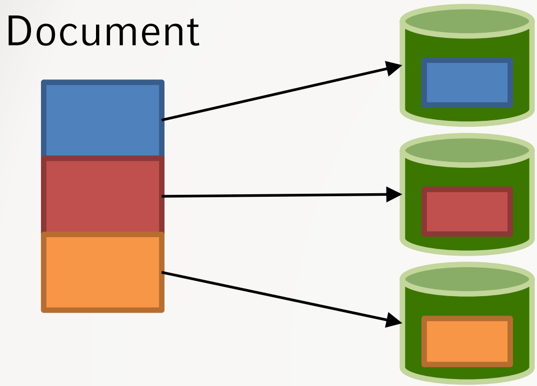
- <u>**S**oft State:</u> The system's state changes over time. Stale data may expire if not refreshed.

- <u>**E**ventual consistency:</u> The system is consistent from time to time, but not always. Updates are propagated through the system if there is enough time.

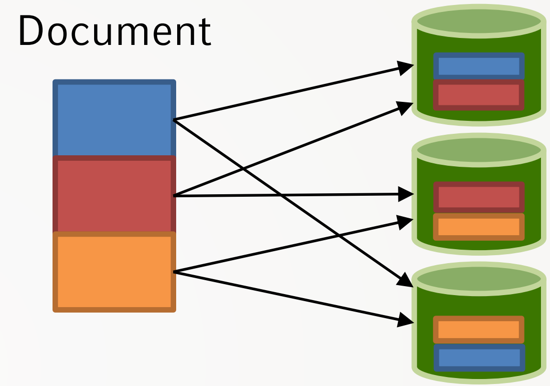
→ **BASE is settled on the opposite site to ACID when considering a „consistency-availability spectrum"**

About the concepts behind NoSQL Databases

Data sharding



Data replication



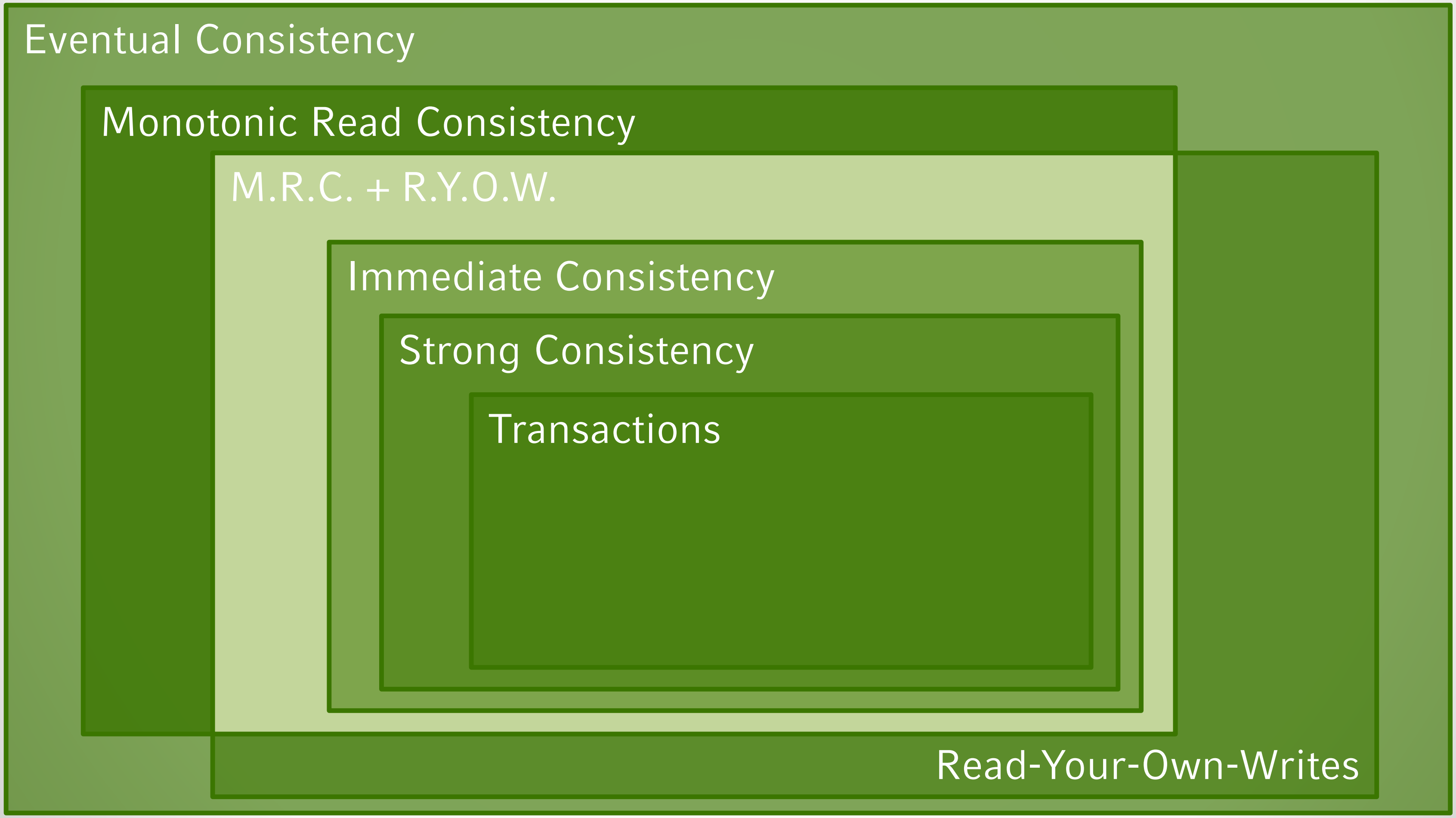
The two types of consistency:

- Logical consistency:
  Data is consistent within itself (Data Integrity)

- Replication consistency:
  Data is consistent across multiple replicas (on multiple machines)

About the concepts behind NoSQL Databases

Levels of Consistency:



About the concepts behind NoSQL Databases

Levels of Consistency:

- Eventual Consistency: Write operations are not spread across all servers/partitions immediately
- Monotononic Read Consistency: A client who read an object once will never read an older version of this object
- Read Your Own Writes: A client who wrote an object will never read an older version of this object
- Immediate Consistency: Updates are propagated immediately, but not atomic

About the concepts behind NoSQL Databases

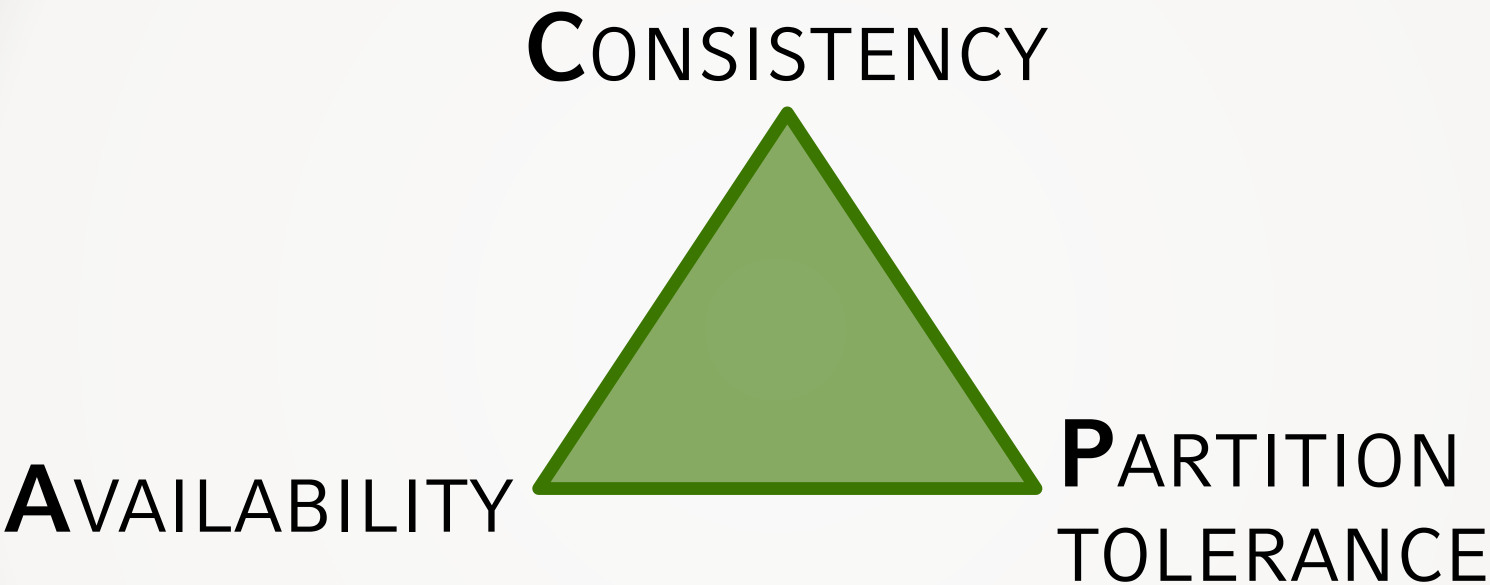
Levels of Consistency:

- Strong consistency: Updates are propagated immediately + support of atomic operations on single data entities (usually on master nodes)

- Transactions: Full support of ACID transaction model

About the concepts behind NoSQL Databases

Brewer‘s CAP Theorem:



Any networked shared-data system can have at most two of the three desired properties!

About the concepts behind NoSQL Databases

Possible DB-Systems by CAP Theorem:

- **<u>CP</u>**<u>-Systems:</u> Fully consistent and partitioned systems renounce availability. Only consistent nodes are available.

- **<u>AP</u>**<u>-Systems:</u> Fully available and partitioned systems renounce consistency. All nodes answer to queries all the time, even if answers are inconsistent.

- **<u>AC</u>**<u>-Systems:</u> Fully available and consistent systems renounce partitioning. Only possible if the system is not distributed.

Big Picture

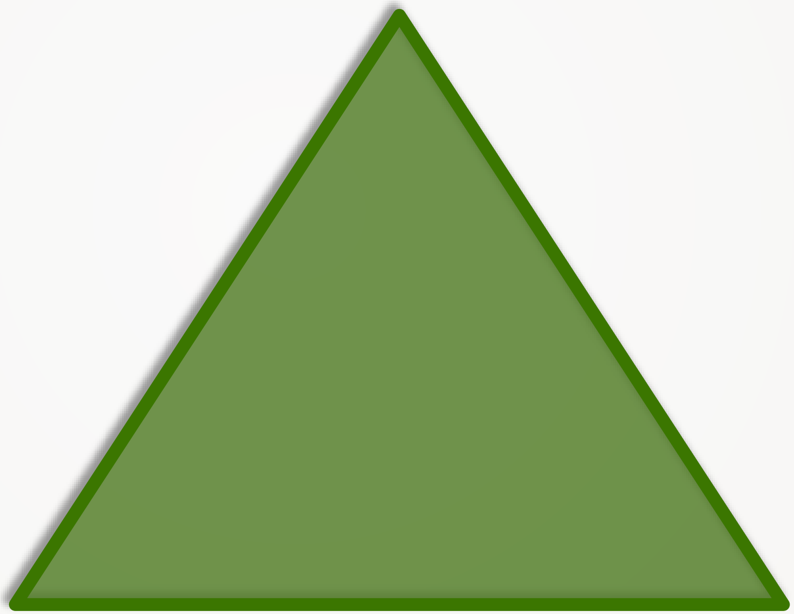
CAP Theorem:

C

A

All clients always
have the same view
of the data

C



A

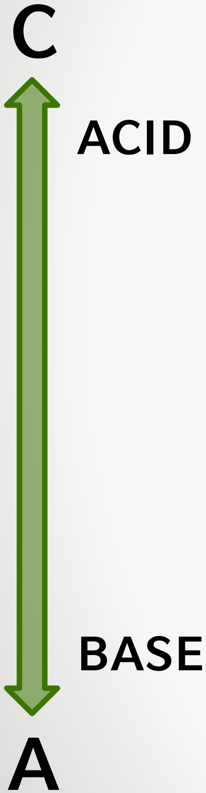
Each client can al-
ways read and write

P

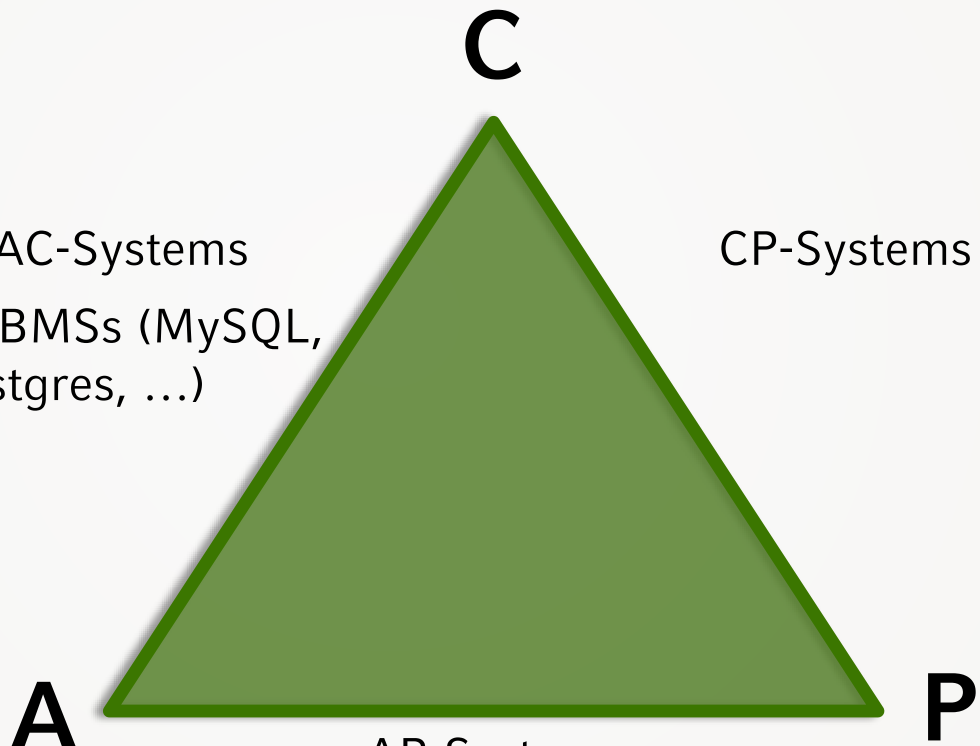
The system works well
despite physical
network partitions

Big Picture

CAP Theorem:



All clients always have the same view of the data



AC-Systems

- RDBMSs (MySQL, Postgres, …)

CP-Systems

AP-Systems

Each client can always read and write

The system works well despite physical network partitions

NoSQL Data Models

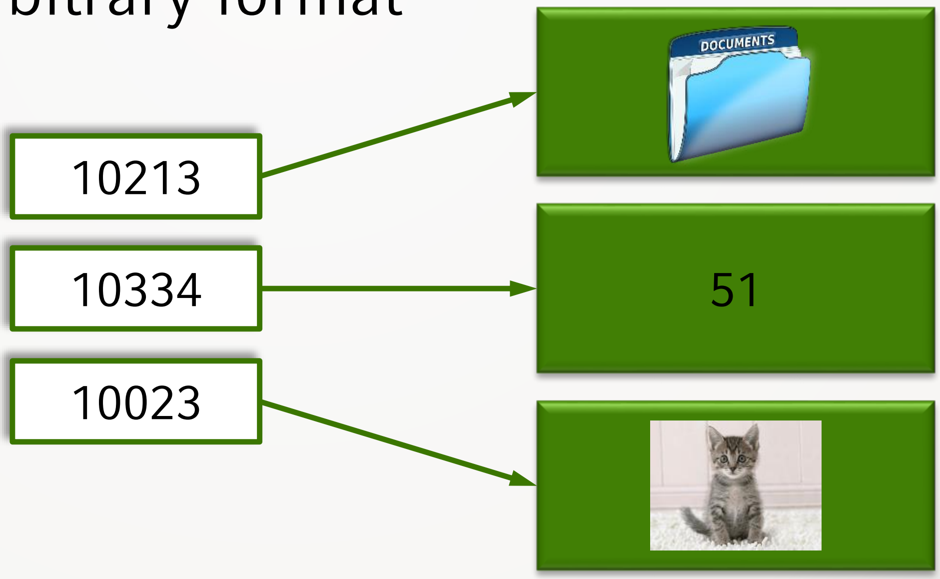
The 4 Main NoSQL Data Models:

- **Key/Value Stores**
- **Document Stores**
- **Wide Column Stores**
- **Graph Databases**

NoSQL Data Models

Key/Value Stores:

- Most simple form of database systems
- Store key/value pairs and retrieve values by keys
- Values can be of arbitrary format



NoSQL Data Models

Key/Value Stores:

- consistency models range from *eventual consistent* to *serializable*
- some systems support ordering keys which enables efficient query processing, e.g., for range queries
- some systems support in-memory data maintenance and others manage data based on hard drives and SSDs

→ Available Systems are very heterogeneous

NoSQL Data Models

Key/Value Stores - Redis:

- in-memory data structure store with built-in replication, transactions and different levels of on-disk persistence

- supports complex types like lists, sets, hashes, …

- supports various *atomic* operations

```
>> SET val 1
>> GET val => 1
>> INCR val => 2
>> LPUSH my_list a (=> 'a')
>> LPUSH my_list b (=> 'b','a')
>> RPUSH my_list c (=> 'b','a','c')
>> LRANGE my_list 0 1 => b,a
```

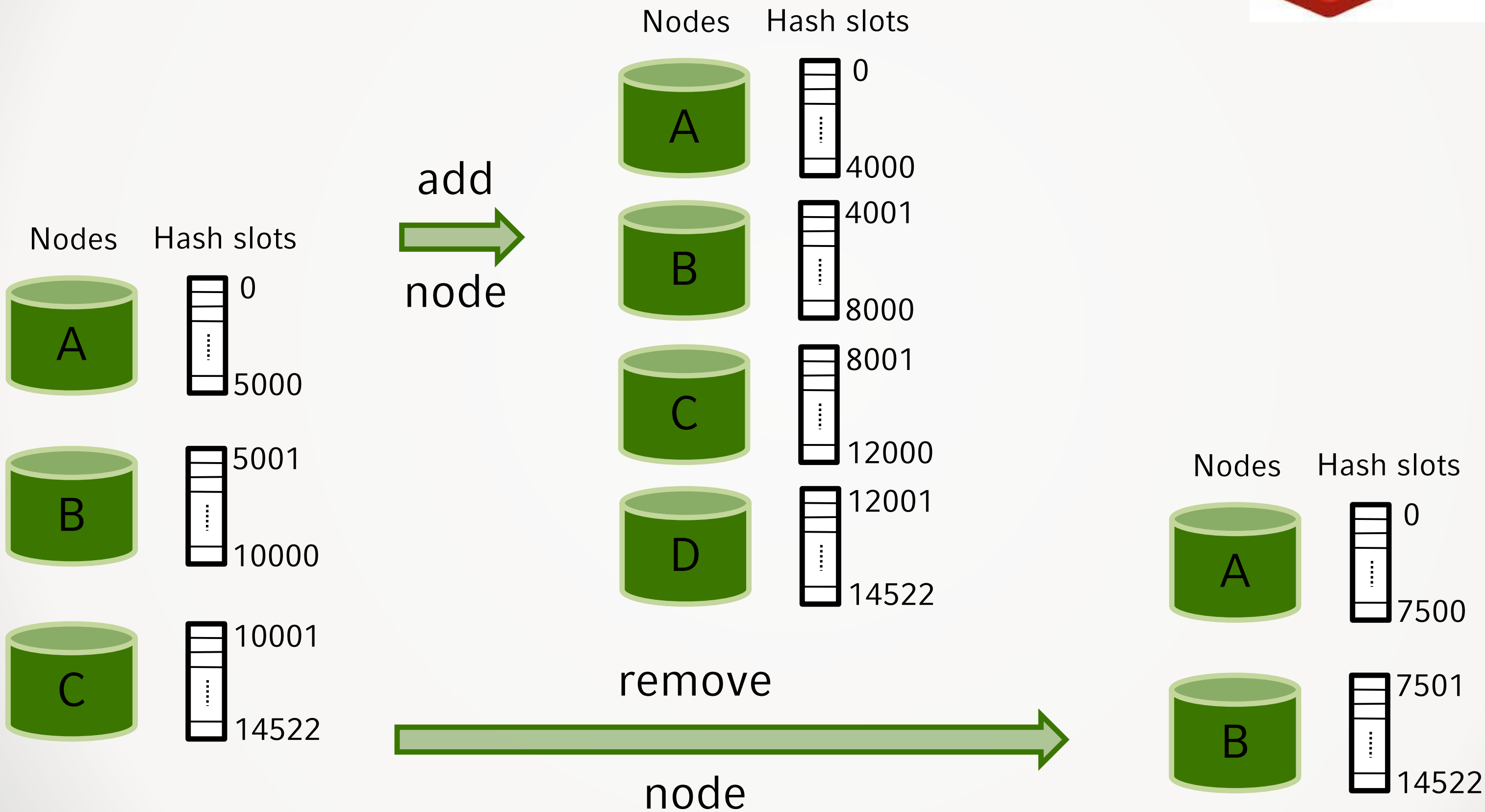
NoSQL Data Models

Key/Value Stores – The Redis cluster model:

- data is automatically sharded across nodes
- some degree of availability, achieved by master-slave architecture (but cluster stops in the event of larger failures)
- easily extendable

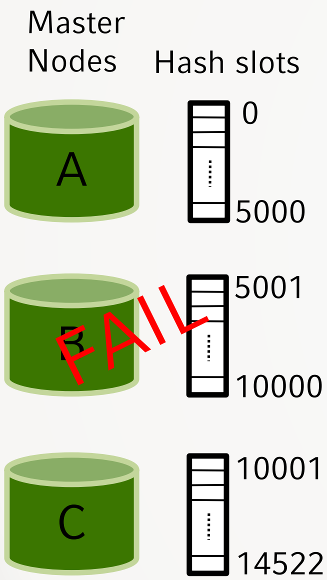
NoSQL Data Models

Key/Value Stores – the redis cluster model:

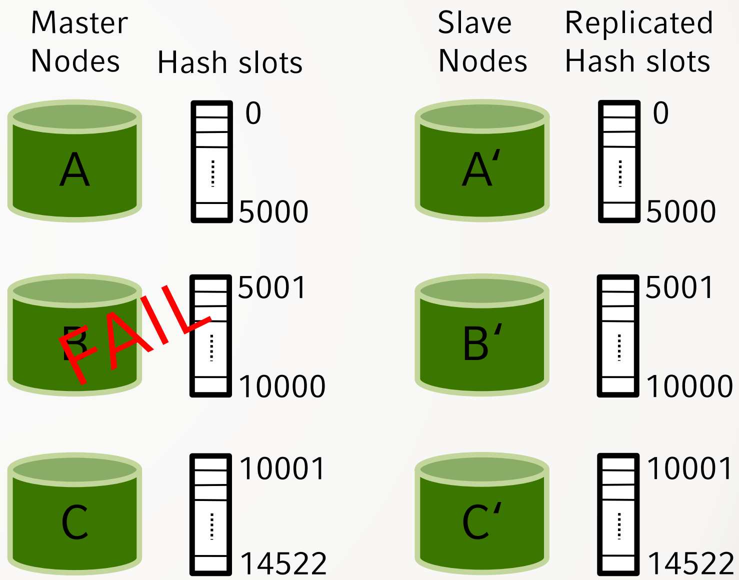


NoSQL Data Models

Key/Value Stores – The Redis cluster model:



Hash slots 5001 – 10000 cannot be used anymore



slave node B' is promoted as the new master and hash slots 5001 – 10000 are still available

Big Picture

CAP Theorem:

C

ACID

BASE

A

Key/Value Stores

all clients always
have the same view
of the data

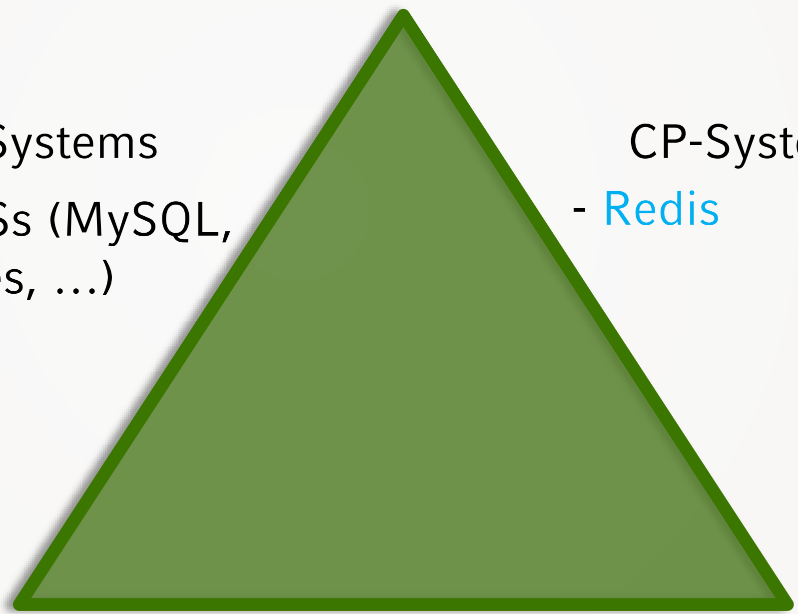
C

AC-Systems

- RDBMSs (MySQL, Postgres, ...)

CP-Systems

- Redis



A

each client can al-
ways read and write

AP-Systems

- Dynamo

P

the system works well
despite physical
network partitions

NoSQL Data Models

Document Stores:

- store documents in form of XML or JSON
- semi-structured data records that do not have a homogeneous structure
- columns can have more than one value (arrays)
- documents include internal structure, or metadata
- data structure enables efficient use of indexes

NoSQL Data Models

Document Stores:

given following text:

Max Mustermann
Musterstraße 12
D-12345 Musterstadt

```
<contact>
  <first_name>Max</first_name>
  <last_name>Mustermann</last_name>
  <street>Musterstraße 12</street>
  <city>Musterstadt</city>
  <zip>12345</zip>
  <country>D</country>
</contact>
```

→ find all <contact>s where <zip> is “12345“

NoSQL Data Models

Document Stores: mongoDB

- data stored as documents in binary representation (BSON)
- similarly structured documents are bundled in collections
- provides own ad-hoc query language
- supports ACID transactions on document level

NoSQL Data Models

Document Stores:

MongoDB Data Management:

- automatic data sharding
- automatic re-balancing

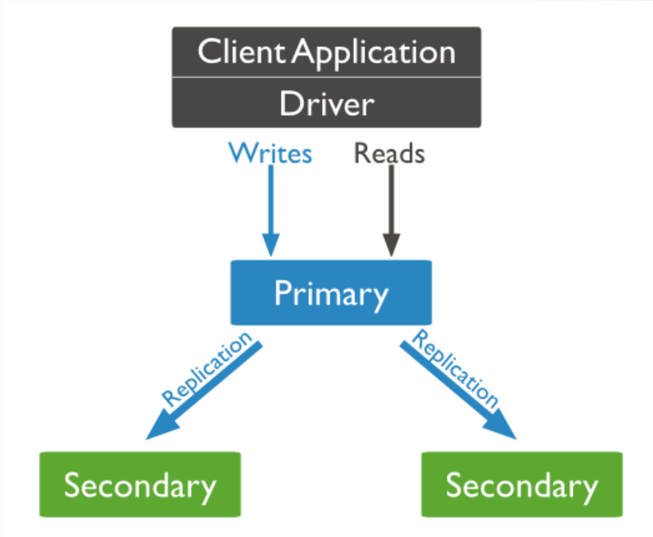
- multiple sharding policies:
  - Hash-based sharding: Documents are distributed according to an MD5 hash → uniform distribution
  - Range-based sharding: Documents with shard key values close to one another are likely to be co-located on the same shard → works well for range queries
  - Location-based sharding: Documents are partitioned w.r.t. a user-specified configuration that associates shard key ranges with specific shards and hardware

NoSQL Data Models

Document Stores: mongoDB

MongoDB Consistency & Availabilty:

- default: strong consistency (but configurable)
- increased availability through replication
  - *replica sets* consist of one *primary* and multiple *secondary members*
  - MongoDB applies writes on the primary and then records the operations on the primary's *oplog*



Big Picture

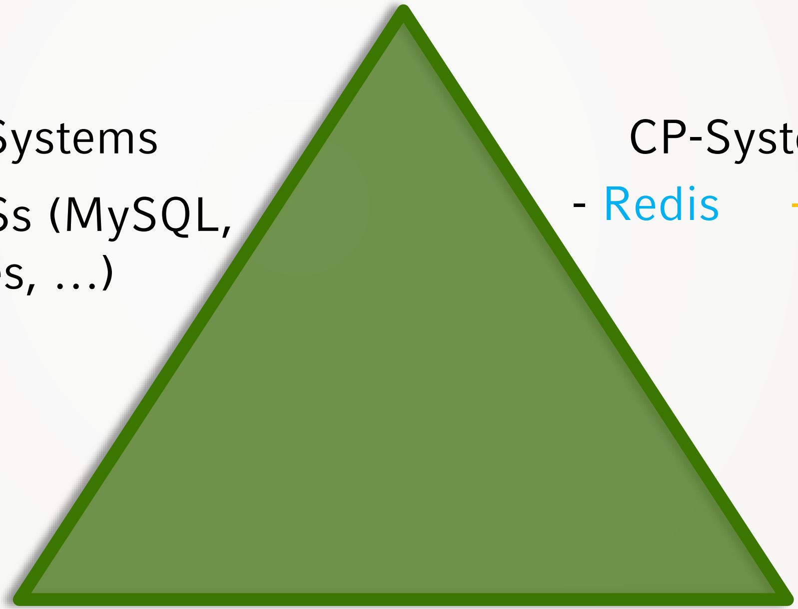
CAP Theorem:

C

A

all clients always have the same view of the data

C



AC-Systems

- RDBMSs (MySQL, Postgres, …)

CP-Systems

- Redis - MongoDB

Key/Value Stores
Document Stores

A

each client can always read and write

AP-Systems

- Dynamo - CouchDB

P

the system works well despite physical network partitions

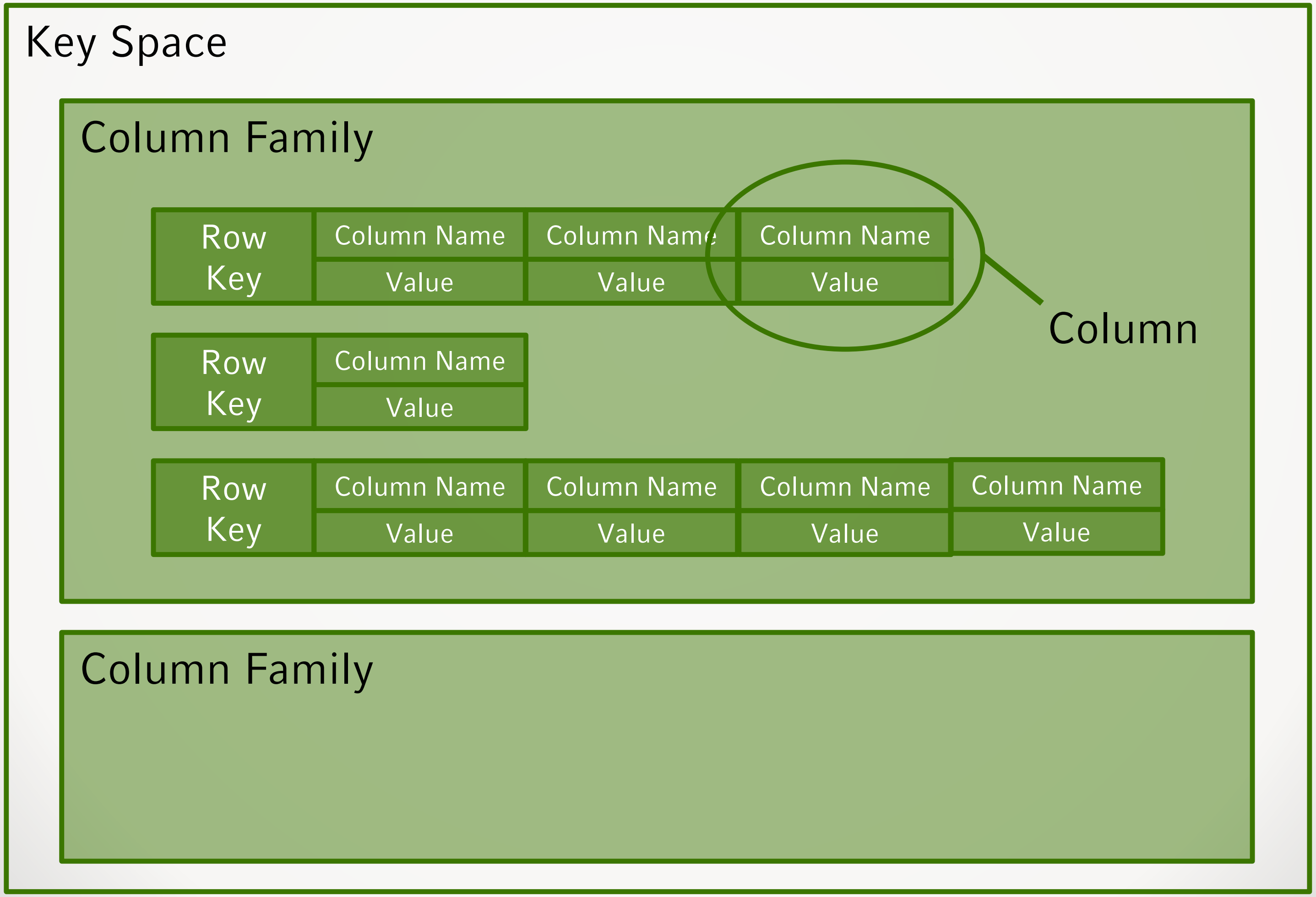
NoSQL Data Models

Wide Column Stores:

- rows are identified by keys
- rows can have different numbers of columns (up to millions)
- order of rows depend on key values (locality is important!)
- multiple rows can be summarized to *families* (or *tablets*)
- multiple families can be summarized to a *key space*

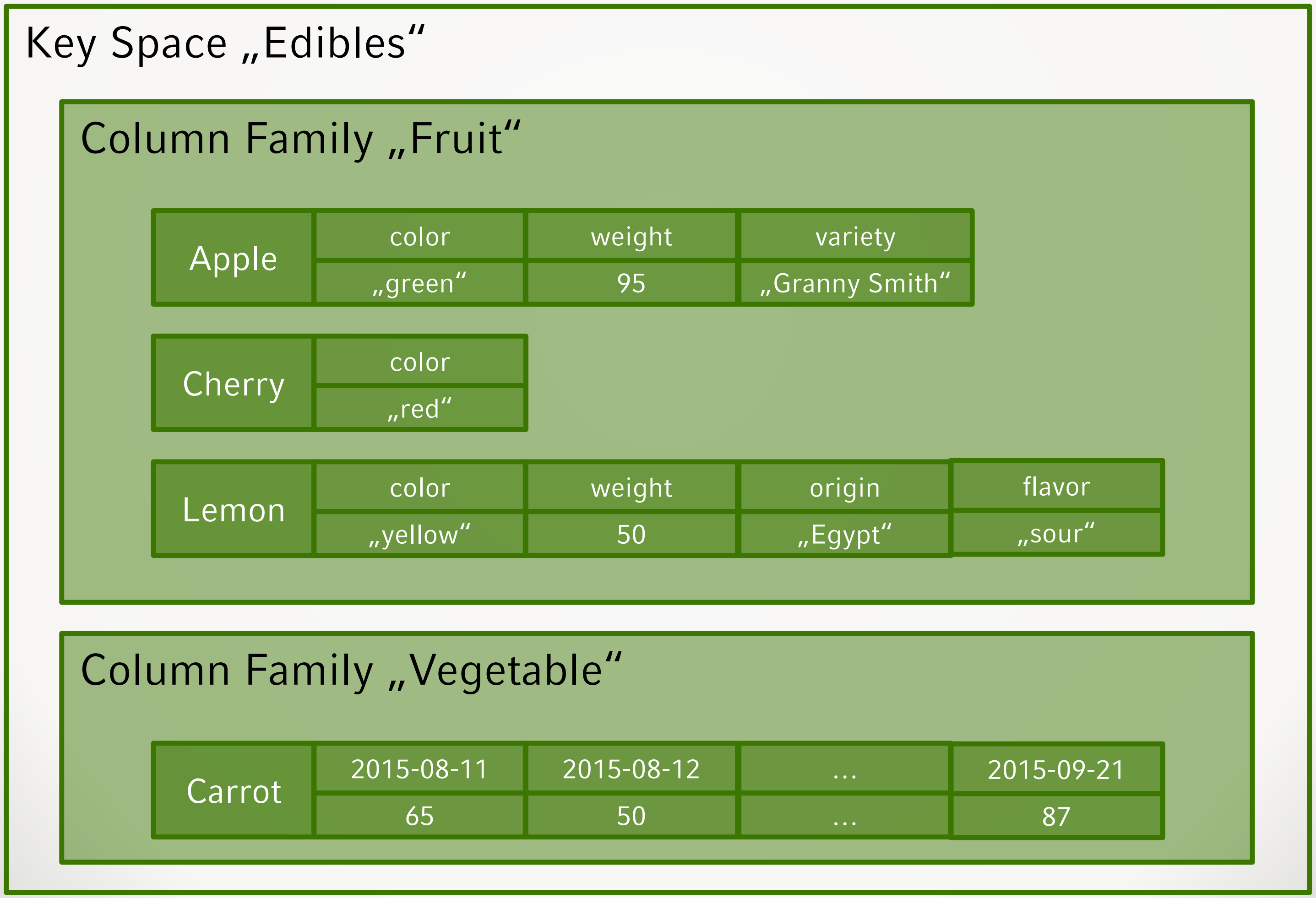
NoSQL Data Models

Wide Column Stores:



NoSQL Data Models

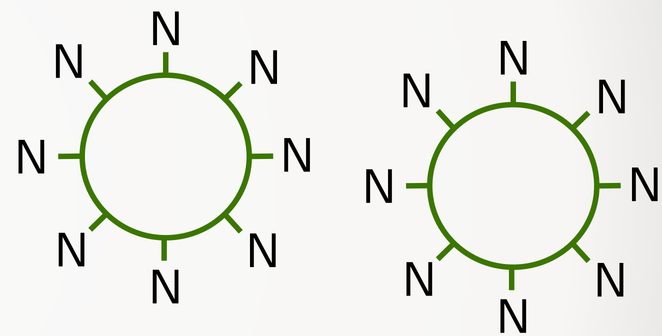
Wide Column Stores:



NoSQL Data Models

Wide Column Stores: Cassandra

- developed by Facebook, Apache project since 2009
- cluster Architecture:
  - P2P system (ordered as rings)
  - Each node plays the same role (decentralized)
  - Each node accepts read/write operations



- user access through nodes via *Cassandra Query Language (CQL)*

NoSQL Data Models

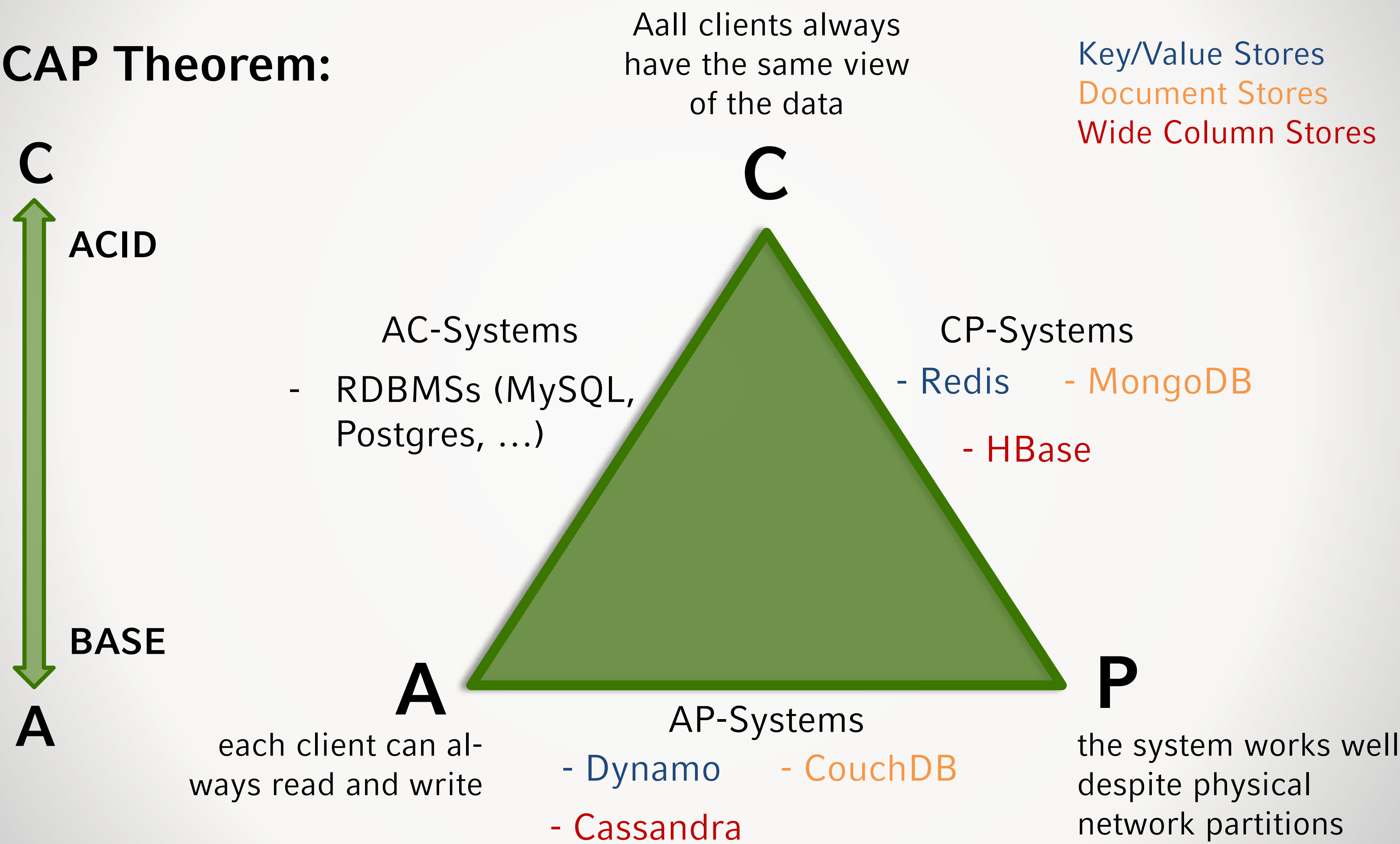
Wide Column Stores: Cassandra

Consistency

- tunable Data Consistency (chosable per operation)
- read repair: if stale data is read, Cassandra issues a read repair → find most up-to-date data and update stale data
- generally: Eventually consistent
- main focus on availability!

Big Picture

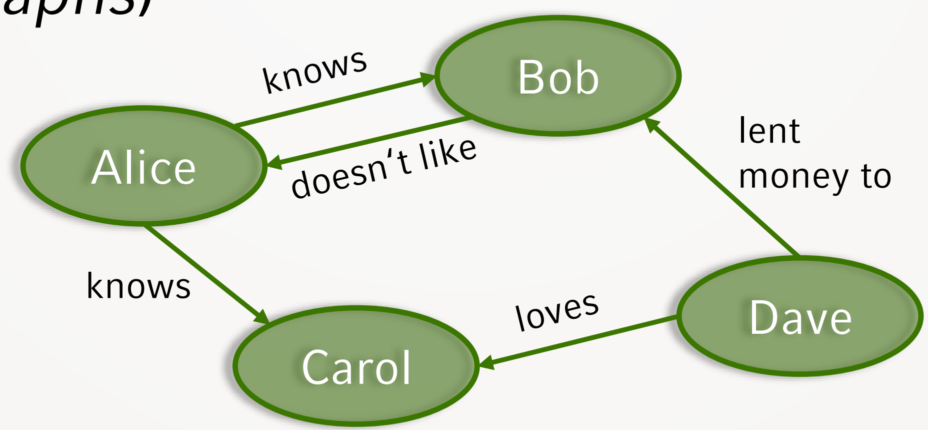
CAP Theorem:



NoSQL Data Models

Graph Databases:

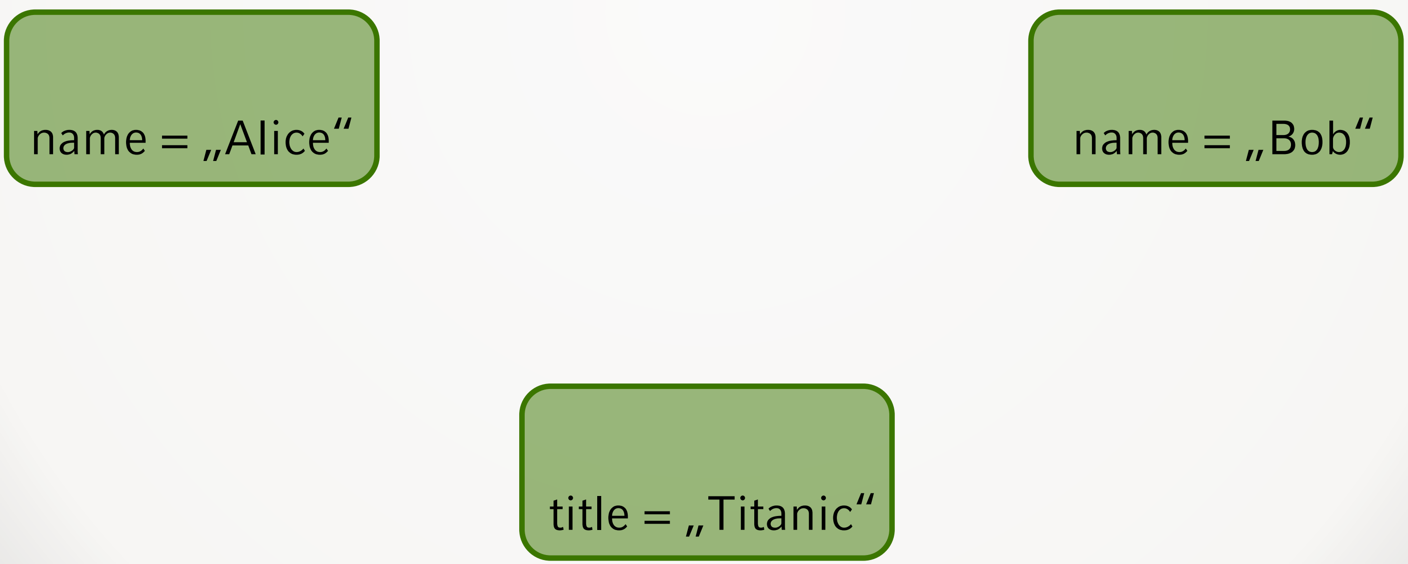
- use graphs to store and represent relationships between entities
- composed of *nodes* and *edges*
- each node and each edge can contain *properties* (*Property-Graphs*)



NoSQL Data Models

Graph Databases:

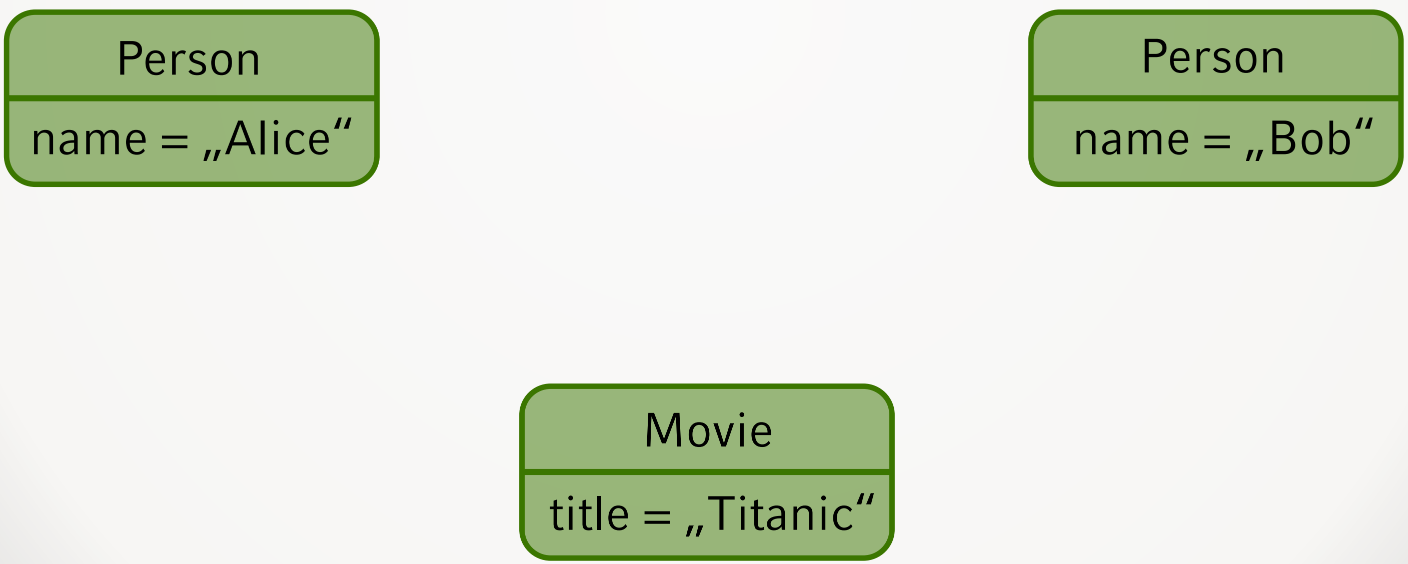
Alice is a friend of Bob and vice versa. They both love the movie „Titanic“.



NoSQL Data Models

Graph Databases:

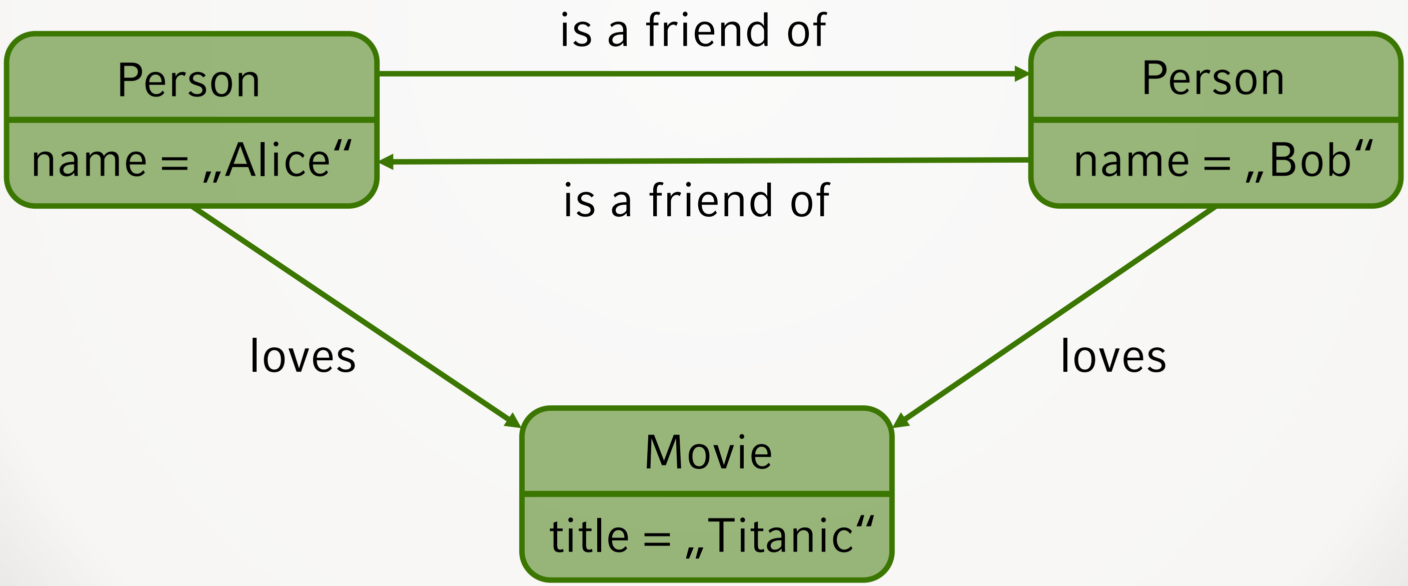
Alice is a friend of Bob and vice versa. They both love the movie „Titanic“.



NoSQL Data Models

Graph Databases:

Alice is a friend of Bob and vice versa. They both love the movie „Titanic“.



NoSQL Data Models

Graph Databases:

neo4j

- master-slave replication (no partitioning!)
- consistency: eventual consistency (tunable to Immediate consistency)
- support of ACID Transactions
- cypher Query Language
- schema-optional

Big Picture

CAP Theorem:

Key/Value Stores
Document Stores
Wide Column Stores
Graph Databases

C

ACID

BASE

A

all clients always have the same view of the data

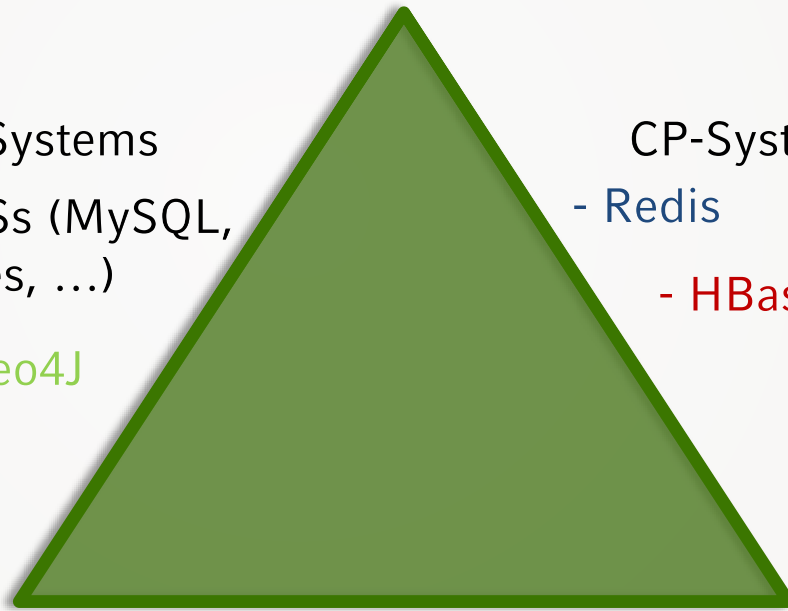
C

AC-Systems

- RDBMSs (MySQL, Postgres, ...)
- Neo4J

CP-Systems

- Redis
- MongoDB
- HBase



A

each client can always read and write

AP-Systems

- Dynamo
- CouchDB
- Cassandra

P

the system works well despite physical network partitions