Chapter 2:

NoSQL Databases

Outline

- History
- Concepts
  - ACID
  - BASE
  - CAP
- Data Models
  - Key-Value
  - Document
  - Column-based
  - Graph

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:
  - LNR: L1
  - Lname: Meier
  - Status: 20
  - Sitz: Wetter
- Project:
  - PNR: P2
  - Pname: Pleite
  - Ort: Bonn
- Pieces:
  - TNR: T6
  - Tname: Schraube
  - Farbe: rot
  - Gewicht: 03
  - Menge: 700

LNR	Lname	Status	Sitz
…	…	…	…
…	…	…	…
…	…	…	…

PNR	Pname	Ort
…	…	…
…	…	…
…	…	…

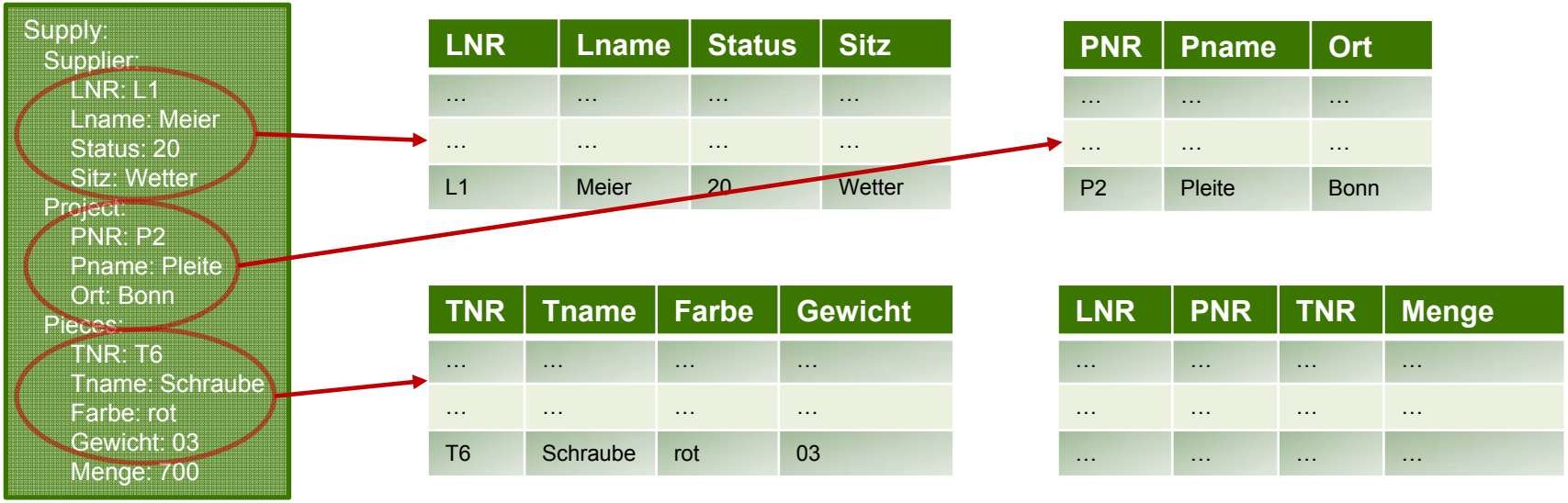
TNR	Tname	Farbe	Gewicht
…	…	…	…
…	…	…	…
…	…	…	…

LNR	PNR	TNR	Menge
…	…	…	…
…	…	…	…
…	…	…	…

Given the LTP scheme from Datenbanksysteme I and an object of type Supply:

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

History – Impedance Mismatch

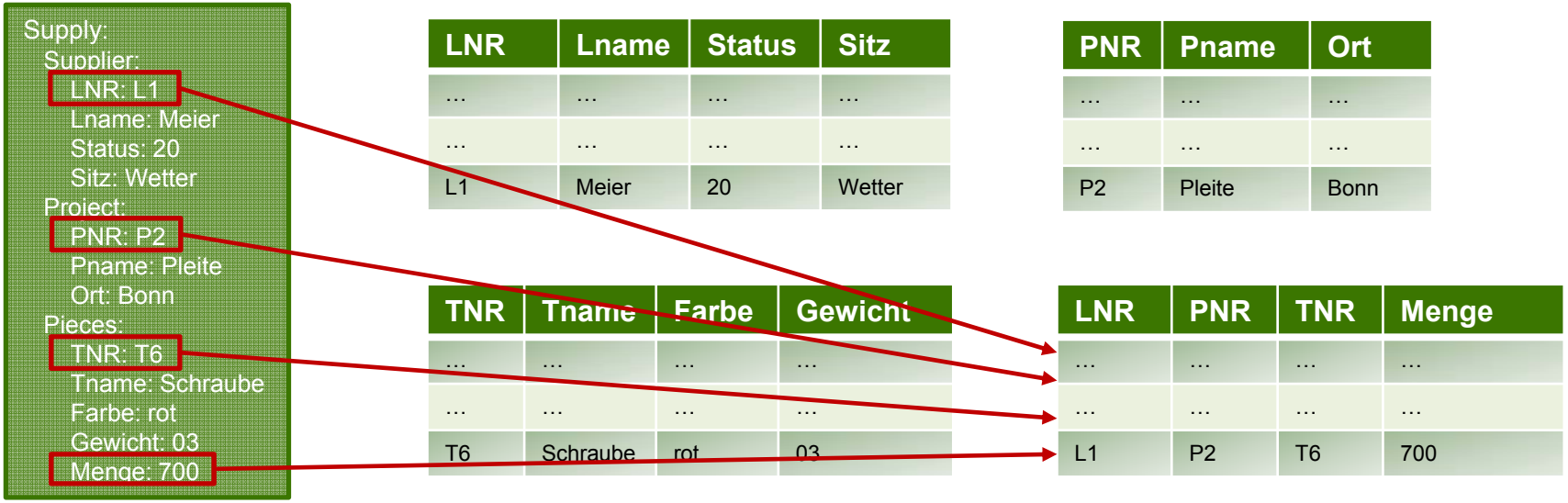



```
INSERT INTO L VALUES (Supply.getSupplier().getLNR(), ...);

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)
- Avoidance of object-relational mapping
  → Programmer-friendly
- But still Relational Databases prevailed in the 90's

Mid 2000's: Rise of Web 2.0

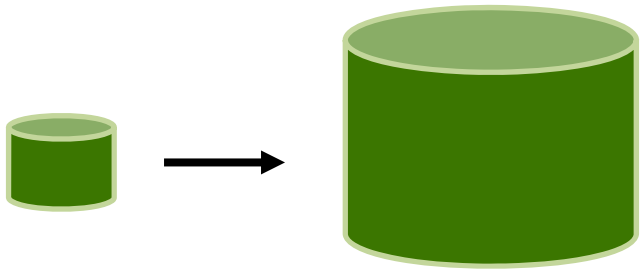
- Lots of user generated data through web applications

  → Storage systems had to become scaled up

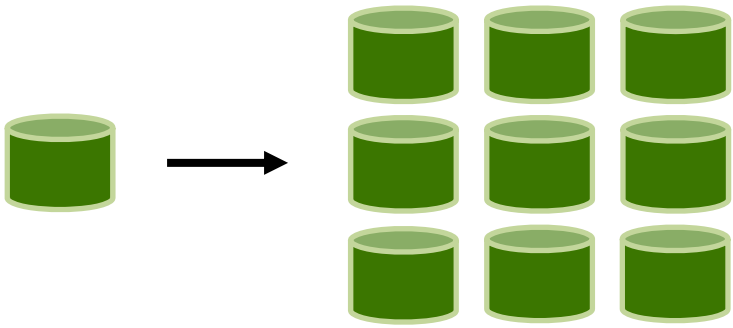
History

Approaches to scale up storage systems

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



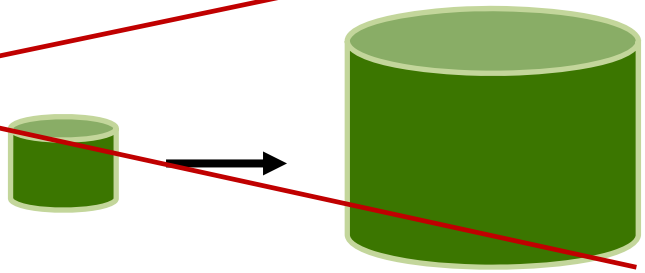
  - Horizontal scaling
    Use many commodity machines and form *computer clusters* or *grids*
    - Cluster maintenance



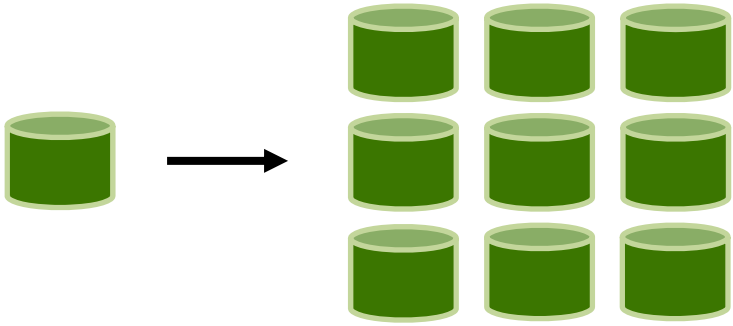
History

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 machines 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)

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

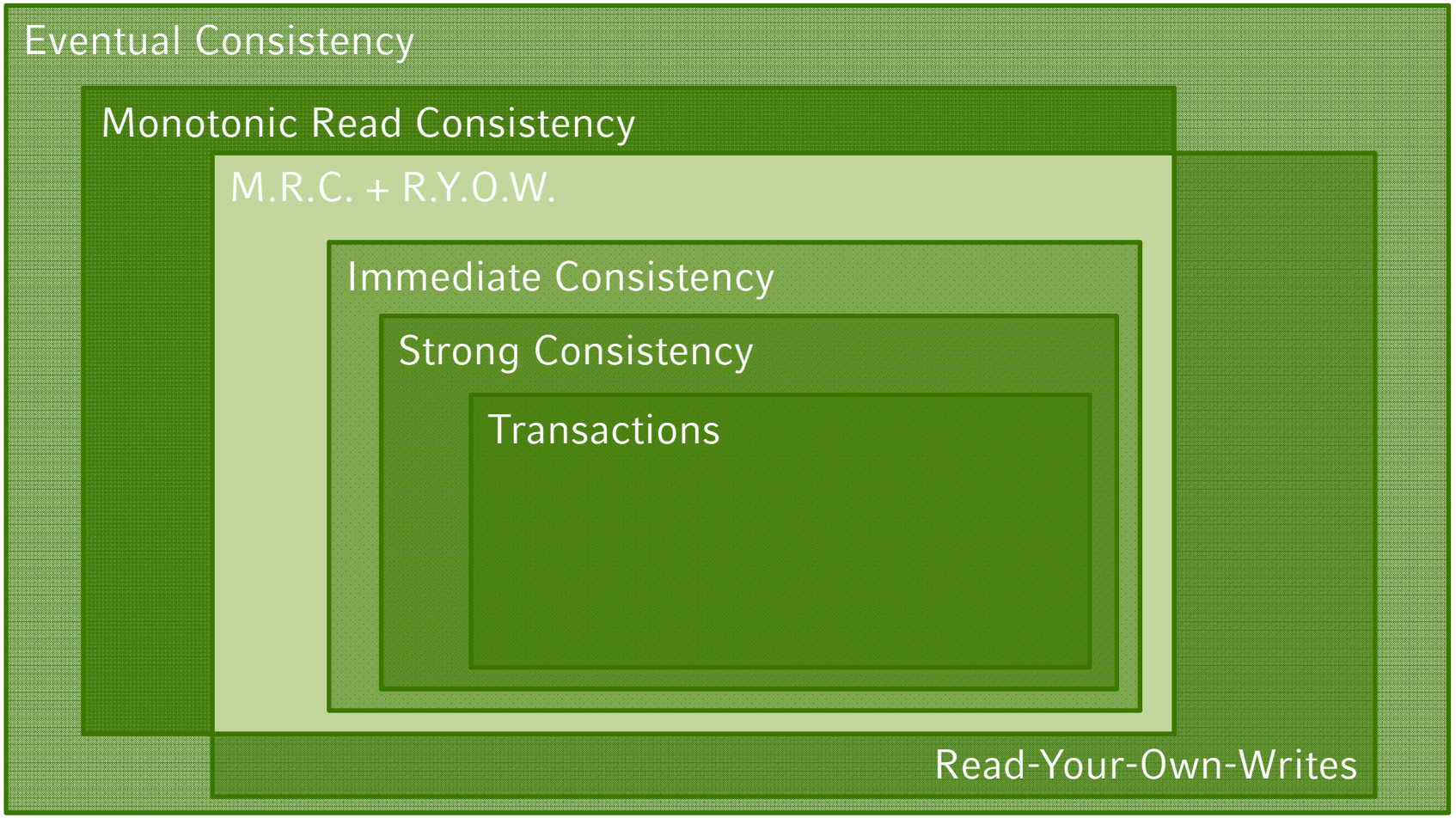
BASE – An artificial concept for NoSQL databases:

- **B**asically **A**vailable: The system is generally available, but some data might not at any time (e.g. due to node failures)
- **S**oft State: The system's state changes over time. Stale data may expire if not refreshed.
- **E**ventual consistency: 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

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

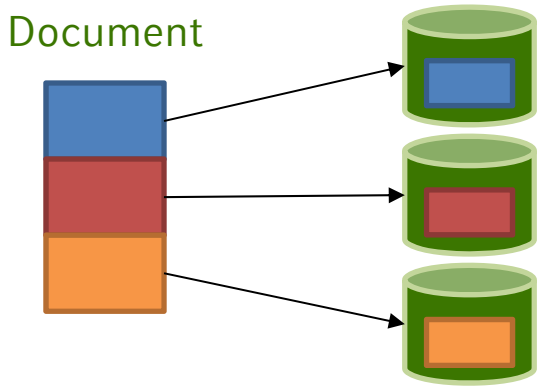
Levels of Consistency:

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

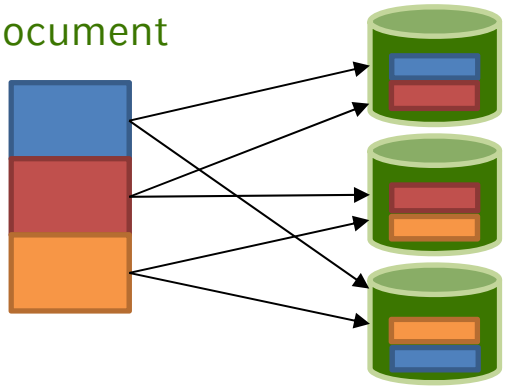
- Transactions: Full support of ACID transaction model

About the concepts behind NoSQL Databases

Data sharding



Data replication

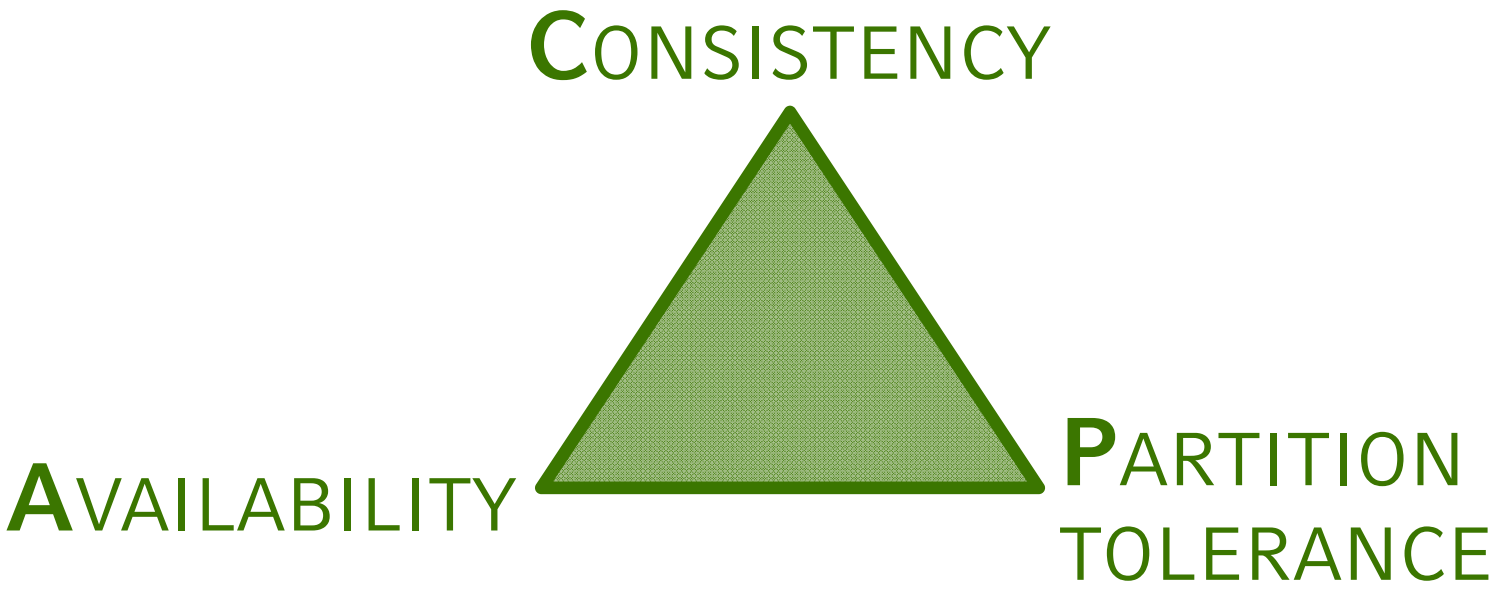


The two types of consistency:

- 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

Brewer's CAP Theorem:



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

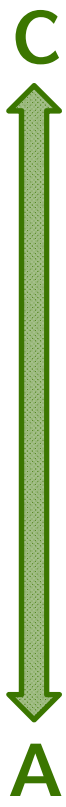
About the concepts behind NoSQL Databases

DB-Systems allowed by CAP Theorem:

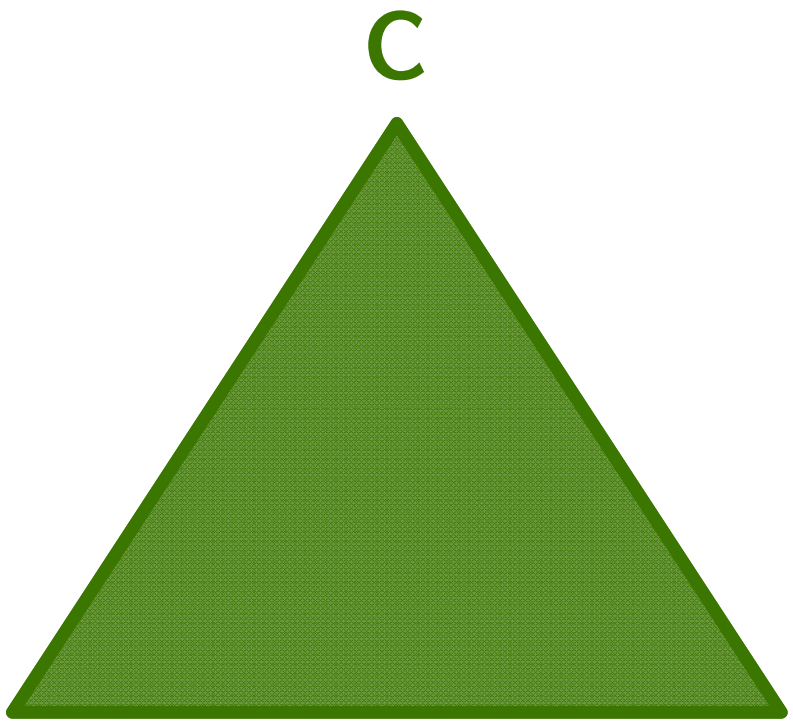
- **CP**-Systems: Fully consistent and partitioned systems renounce availability. Only consistent nodes are available.
- **AP**-Systems: Fully available and partitioned systems renounce consistency. All nodes answer to queries all the time, even if answers are inconsistent.
- **AC**-Systems: Fully available and consistent systems renounce partitioning. Only possible if the system is not distributed.

Big Picture

CAP Theorem:



All clients always have the same view of the data

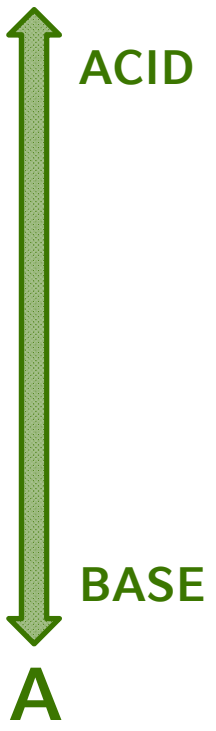


Each client can always read and write

The system works well despite physical network partitions

Big Picture

CAP Theorem:



All clients always have the same view of the data

C

AC-Systems

- RDBMSs (MySQL, Postgres, …)

CP-Systems

A

Each client can always read and write

AP-Systems

P

The system works well despite physical network partitions