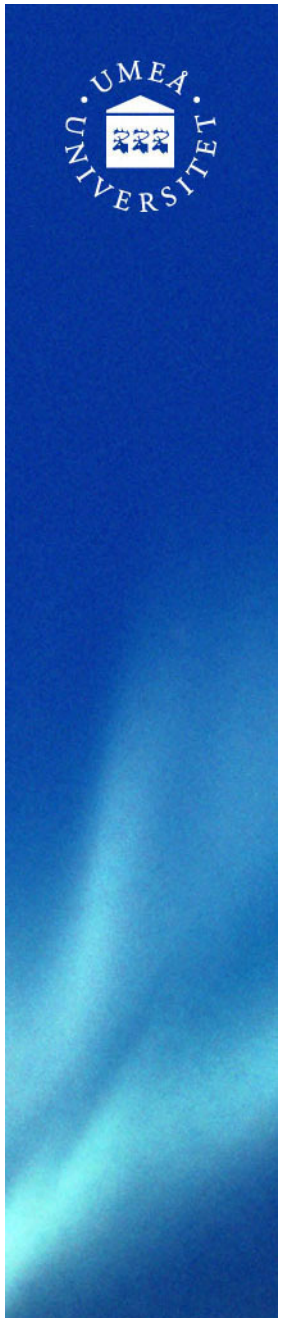


Distributed Systems
Cassandra

2014-09-26

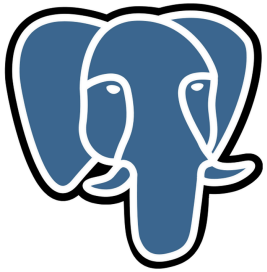
Cristian Klein
Department of Computing Science
Umeå University

Outline

- Review of SQL databases
- CAP theorem
- NoSQL movement
- Apache Cassandra

SQL Databases

- Centered around **transactions**
  - "Unit of work treated in a coherent and reliable way" (Wikipedia)
- Examples



SQL Philosophy

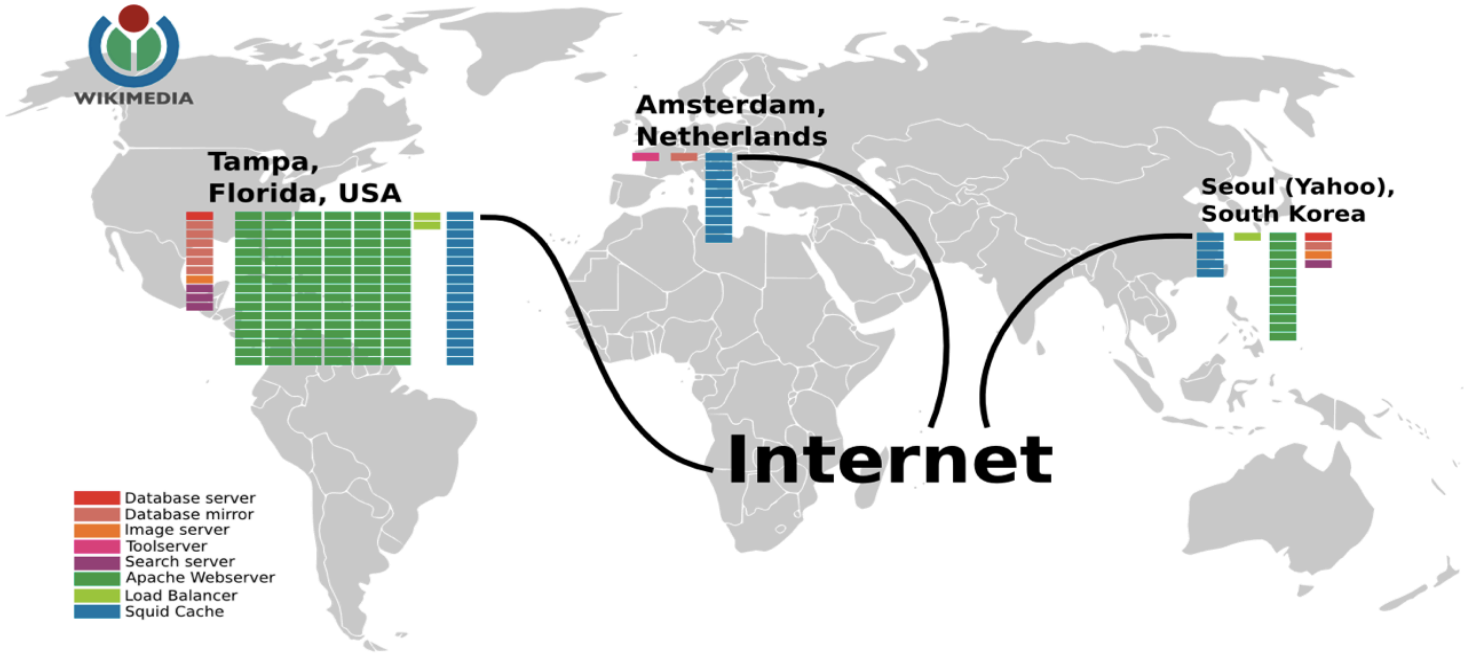
- ACID
  - Atomicity
    - `UPDATE users SET shell="/usr/bin/bash"`
    - Transaction looks atomic from the outside world, including triggers executed, etc.
  - Consistency
    - User-defined rules are always enforced
    - E.g., foreign keys

```
CREATE TABLE Orders (
        Order_Id int NOT NULL,
        OrderNo int NOT NULL,
        User_Id int,
        PRIMARY KEY (Order_Id),
        FOREIGN KEY (User_Id) REFERENCES Users(Users_Id)
)
```

SQL Philosophy (cont'd)

- Integrity
  - **Concurrent** transactions bring the system into a state, as if they had been executed **serially**
  - E.g:
    - `UPDATE users SET a = 1;`
    - `UPDATE users SET a = 2;`
- Durability
  - Once a transaction is committed, it stays so
  - E.g., despite software crash, power failure

Internet-scale Applications



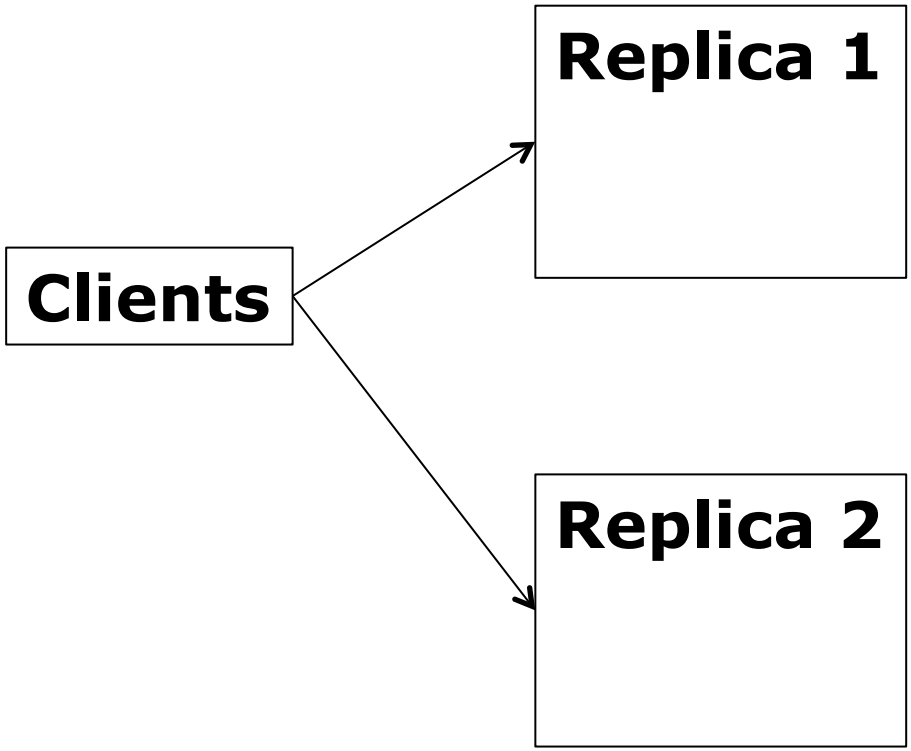
- Reduce latency (datacenter close to user)
- Scalable
- Resilient against
  - Node failures
  - Network partitions

CAP Theorem

- Conjectured Brewer 2000, proven Gilbert 2002
- Desirable properties of a distributed database
  - Consistent
    - Same data is seen from everywhere
  - Available
    - Requests are served successfully
    - Or failure is signaled immediately
  - Partition tolerant
    - Database continues working despite network partitions
- CAP theorem: **choose any two**

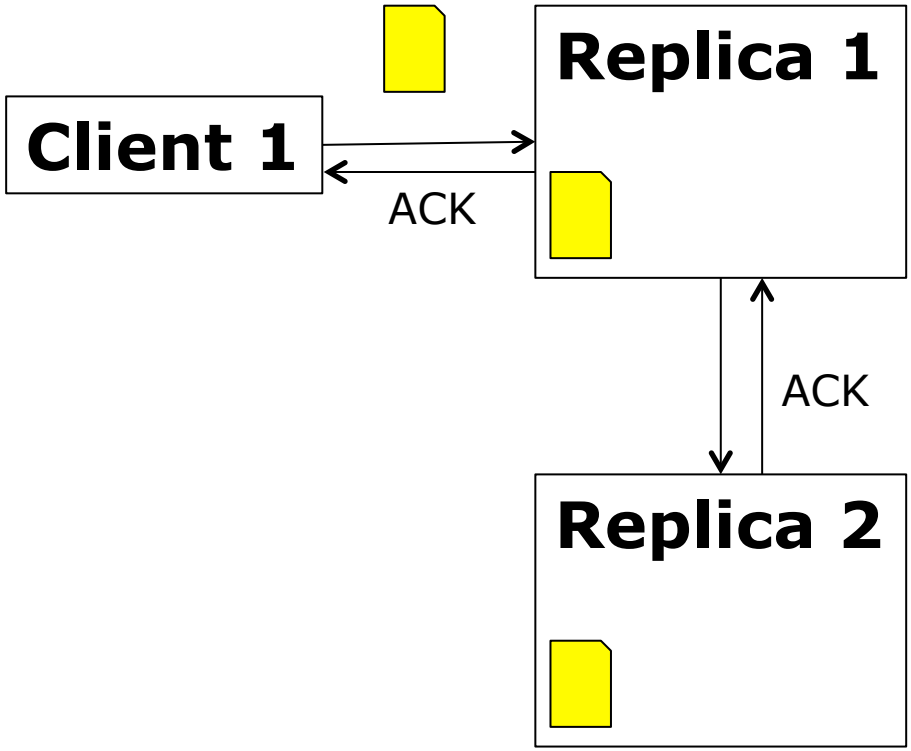
Intuition Behind CAP

Scalable read-write repository



Intuition Behind CAP

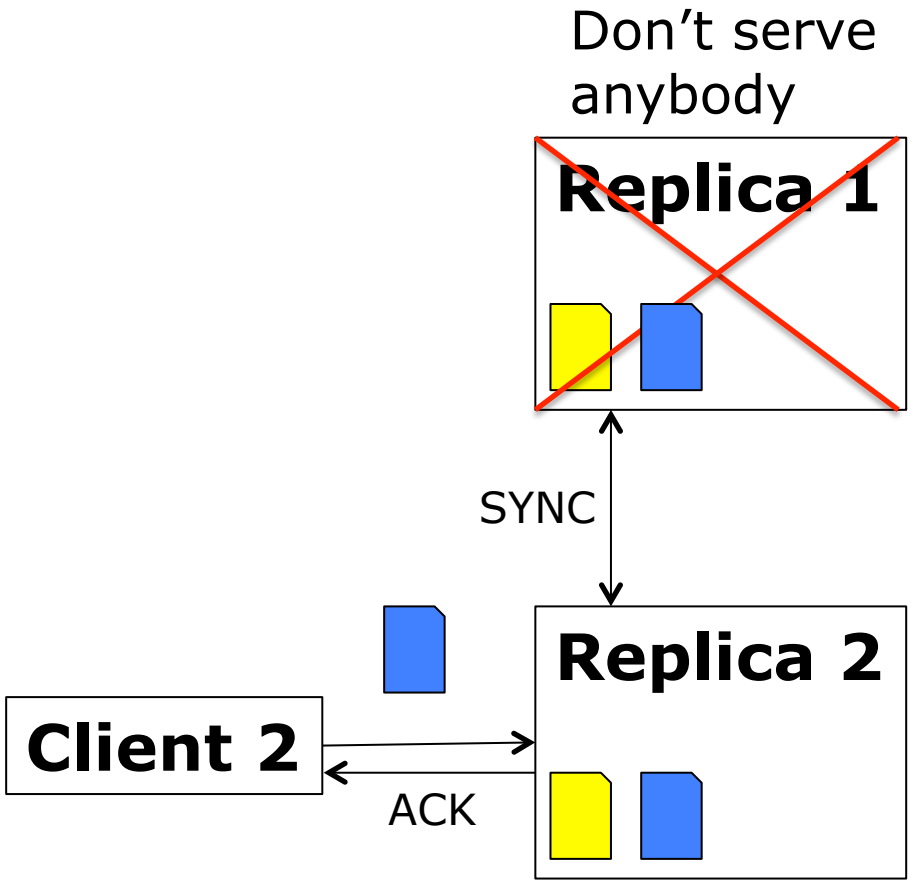
Scalable read-write repository



Consistency

Intuition Behind CAP

Scalable read-write repository

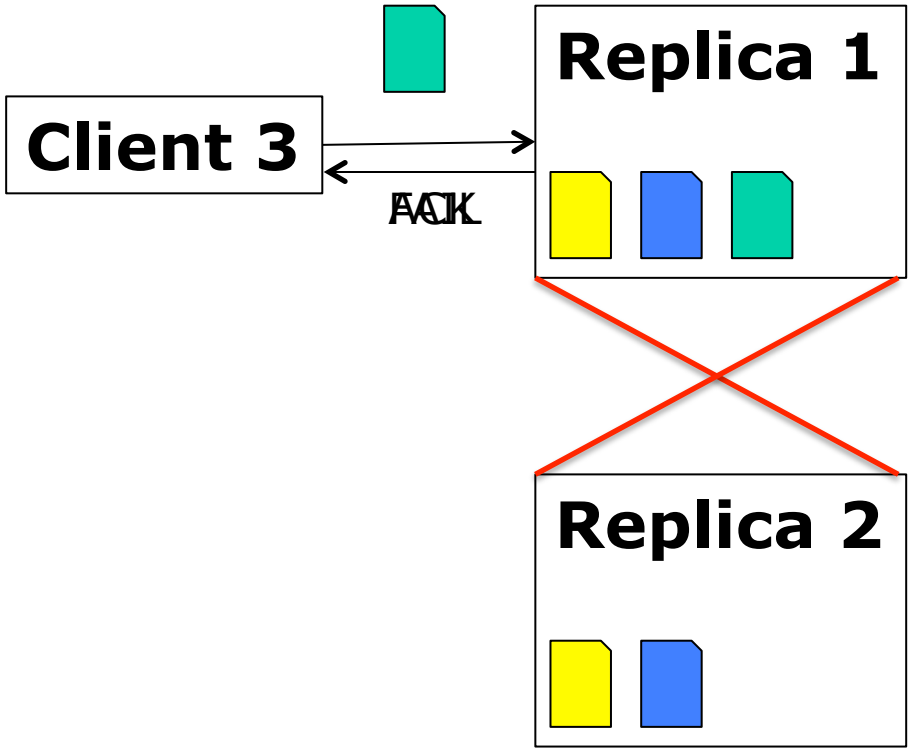


Consistency

Availability

Intuition Behind CAP

Scalable read-write repository



NO Consistency

NO Availability

Partition tolerance

Who needs consistency?

- Strong consistency
  - Buying a product
  - Transferring money
  - Reserving a seat
- Eventual consistency
  - Posting a status
  - Changing profile picture
  - Okey to read **stale** data
- Q: What about banks?

NoSQL Philosophy

- Aim for availability and partition tolerance, sacrifice consistency
- BASE
  - **B**asically **A**vailable
  - **S**oft state
  - **E**ventually-consistent
- Key-value store

Amazon SimpleDB

Apache Cassandra

- Was used by facebook
- Used by NETFLIX

- Overview
- Software architecture
- Data model

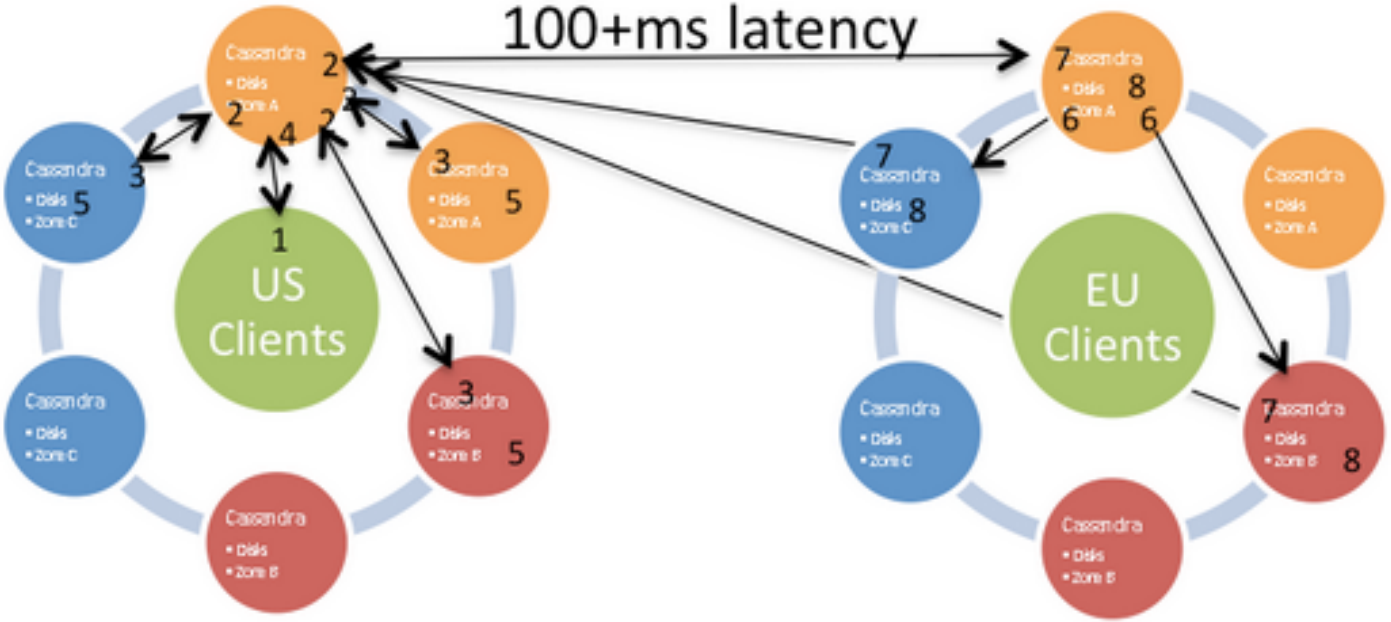
Overview

- Key-value store (NoSQL database)
- No atomicity
- No transactions (not SQL-like)
- Not durable
  - Data is **not** immediately written to disk
  - RAIN philosophy
    - Redundant Array of Independent Nodes

Overview (cont'd)

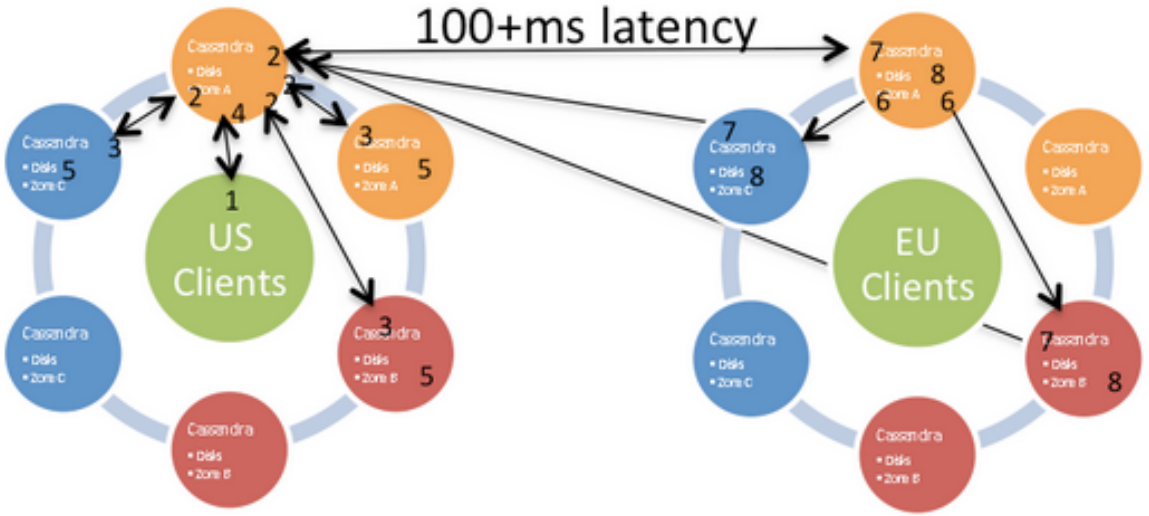
- Distributed, no master
  - Highly resilient
- Focuses on performance
  - Writes **sequentially** to disk
- Tunable replication
  - How many copies a key has
  - Where to place them
    - Data-center aware, rack aware
- Tunable consistency
  - Strong consistency: R+W > N
    - E.g., R = W = N/2 + 1 (quorum)
  - Low latency: R = 1, W = 1

Architecture



- Multiple **nodes** ordered in a **ring**
  - Each manages some **keys**
  - Uses **consistent hashing**
- Client connects to any node
  - Reads and writes go to **all** replicas
  - Consistency level: how many ACKs to wait for

Consistency Level



- ONE: lowest latency
- QUORUM: quorum of replica nodes
- LOCAL_QUORUM: quorum in **current** data-center
- EACH_QUORUM: quorum in **each** data-center
- ALL

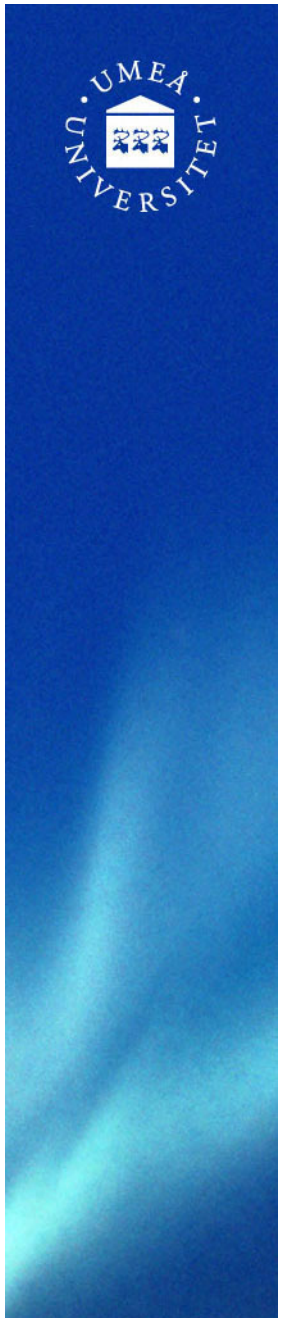
- quorum = replication / 2 + 1
- E.g., for replication = 3, quorum is 2
- Strong consistency: R+W>N

Maintaining Consistency

- All writes have a time-stamp
  - Pray NTP works on your nodes
- Read-repair
  - Update stale data on a node during read
- Hinted sign-off
  - Node B is down, node B back up
  - Replicated keys will be out-of-date
  - Node A stores “log” of operations
  - Quickly brings node B up-to-date

Data model

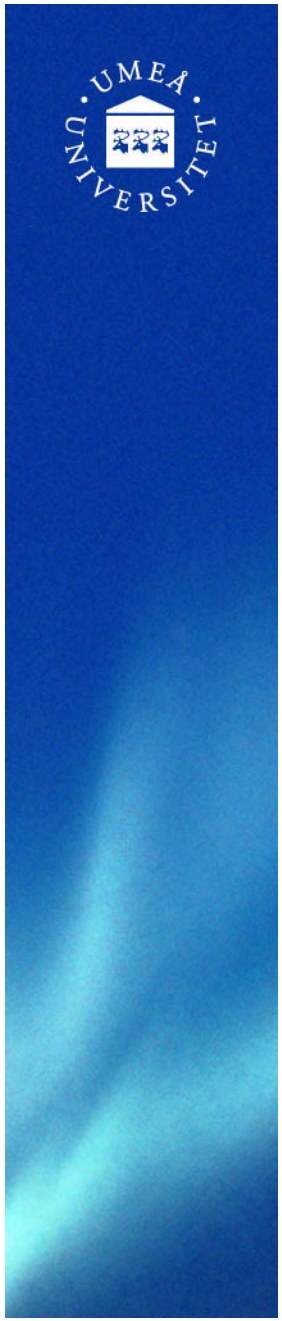
- Keyspace (i.e., database)
- Table
- Columns
  - One **must** be the key
- Indexes
  - To search secondary columns



Spoiler Alert: Tutorial

- Cassandra Query Language (CQL)
  - Simplified version of SQL

- Setting up a cluster
- Creating a keyspace
- Creating a table
- Creating an index
- Writing / reading data
- Consistency level and resilience



Summary

- Transactional (SQL) databases
  - ACID properties
- Internet-scale applications need
  - Availability, tolerance to partitions
- CAP theorem
- NoSQL movement
  - BASE properties
- Apache Cassadra