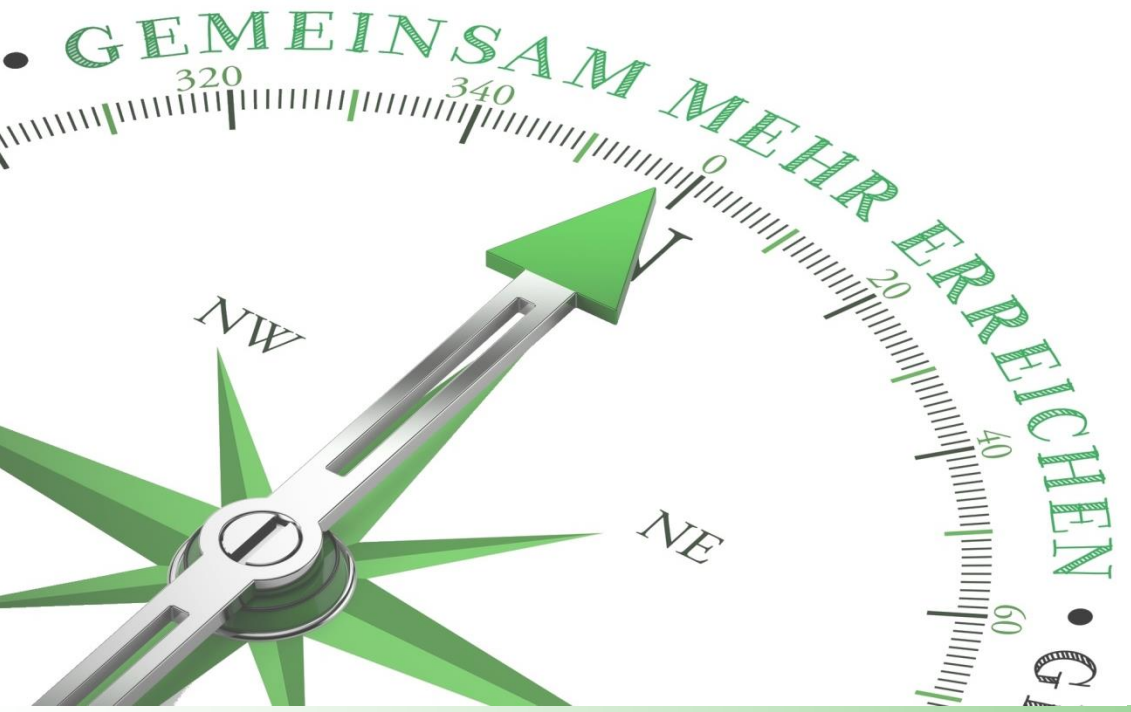
avision
REVIVING YOUR SOFTWARE

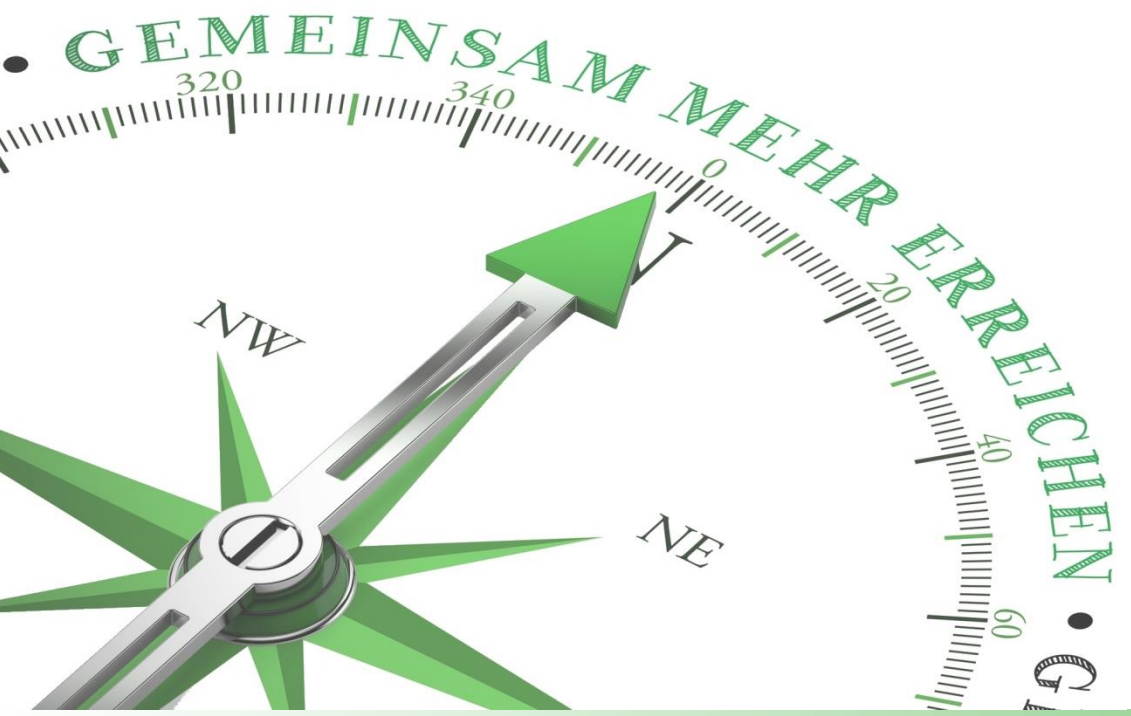
NoSQL & MongoDB



Avision GmbH

Dr. Gernot Pfanner

Inhalt



Grundlagen Datenbanken

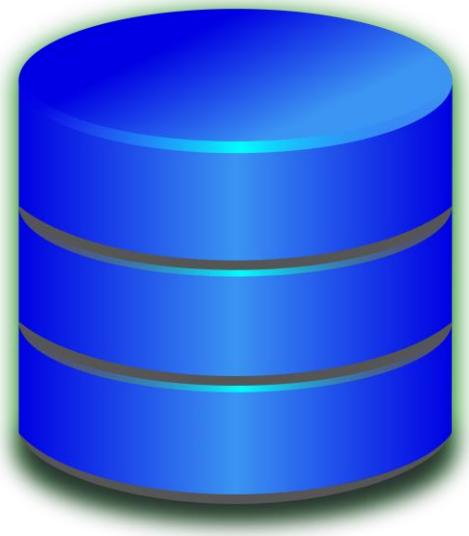
- ✓ Wozu Datenbanken?
- ✓ Relationales Datenbankmodell

NoSQL

- ✓ Grundlagen
- ✓ Grundlegende Einteilung
- ✓ CAP-Theorem
- ✓ SQL vs. NoSQL
- ✓ Der Weg zur richtigen DB
- ✓ MongoDB

Zusammenfassung

Wozu Datenbanken?



- Zentrale Speicherung von Daten
  - Konsistenz (Consistency)
  - Verringerung von Redundanz
  - Gleichzeitige Verfügbarkeit für mehrere Benutzer (Concurrency)
- Permanente Datensicherung (Fault-Tolerance)
- Suchanfragen + Query (Indexierung)
- Datenmanipulation
- Verarbeitung großer Datenmengen

Datenbank = Daten + Verknüpfungen (entscheidend für DB-Modell)

Relationales Datenmodell

ID	User	OS	Last Login
123	Batman	Linux	10.01.18
342	Spiderman	Linux	(null)
425	Superman	Windows	(null)
774	Iron Man	Linux	12.03.18

- **Prinzip:** Organisation von Daten in Tabellen
  - Tabelle: Relation (z.B. Konsument, Produkt)
  - Spalten: Attribute
  - Zeilen: Instanzen
  - Zeilen-ID: Primary Key
  - Übergreifende Verknüpfungen: Foreign Keys
- **Datenmodell** festgelegt durch (vordefiniertes) Schema
- **ACID**
  - Atomizität: "All or nothing" bei Transaktionen
  - Consistency: Transaktionen bewahren Datenkonsistenz in DB
  - Isolation: Keine Interferenz bei parallel gestarteten Transaktionen
  - Durable: Datenänderungen werden permanent geschrieben
- **Vertreter:** Oracle (1980), MySQL (1995), SQL Server (1989), PostgreSQL (1997)

NoSQL

avision
REVIVING YOUR SOFTWARE

- NoSQL: steht für „Not Only SQL“
- Kennzeichen
  - Keine vordefinierten Tabellenschema
  - BASE („Basically Available Soft state Eventual consistency“)
    - Designprinzip, welches absolute Konsistenz (zwischen verteilten Systemen) aufgibt
      - (+) Erhöhte Verfügbarkeit des Systems
      - (-) System-Zustand zeitweise undefiniert
    - Wordplay: „ACID“ (Säure) und „BASE“ (Base)

- NoSQL umfasst mehr als 200 verschiedene Datenbanken (http://nosql-database.org)
- Grundlegende Einteilung
  - Schlüssel/Wert-DB (Key/Value)
  - Spalten-DB (Wide-Column)
  - Dokumenten-DB (Document)
  - Graphen-DB (Graph)

Grundlegende Einteilung

Key/Value DB

Key	Value
“Batman‘‘	{“Bruce Wayne“,“Gotham City“}
“Spiderman“	{“Peter Parker“, “New York City“}
“Superman“	{“Clark Kent“, “Metropolis“}
“Iron Man“	{“Tony Stark“, “Malibu“}

- **Prinzip:**
  Hash-Tables mit Schlüssel/Wert-Paaren
- **Datenmodell:** Planar
- **Use Cases:**
  - URL-Kürzungsdienst
  - Kurzlebige Daten (z.B. Logfiles, Session-Daten, Cache, Queues)
- **Wichtigste Vertreter:**
  Memcached (2003), Redis (2009)

Grundlegende Einteilung

Wide-Column DB

Row Key	Column/ Key 1	Column/ Key 2	Column/ Key 3	Column/ Key 4
"Batman"	First Name → Bruce	Last Name → Wayne	Residence → Gotham	SavedEarth → Yes
"Superman"	First Name → Clark	Last Name → Kent	Residence → Metropolis	SavedEarth → Yes
"Iron Man"	First Name → Tony	Last Name → Stark	Residence → Malibu	SavedEarth → Yes
"The Avengers"			Residence → New York	SavedEarth → Yes

- **Prinzip:** Daten-Speicherung in Spalten
  - Column Family = Row key + Column keys + Values
- **Datenmodell:** Linear (für ein Objekt)
- **Use Cases:**
  - User-Profil Attribute
  - Metadaten (z.B. Songs, Künstler)
  - Real-Time Analytics
- **Wichtigste Vertreter:**
  Cassandra (2008), HBase (2008)

Grundlegende Einteilung

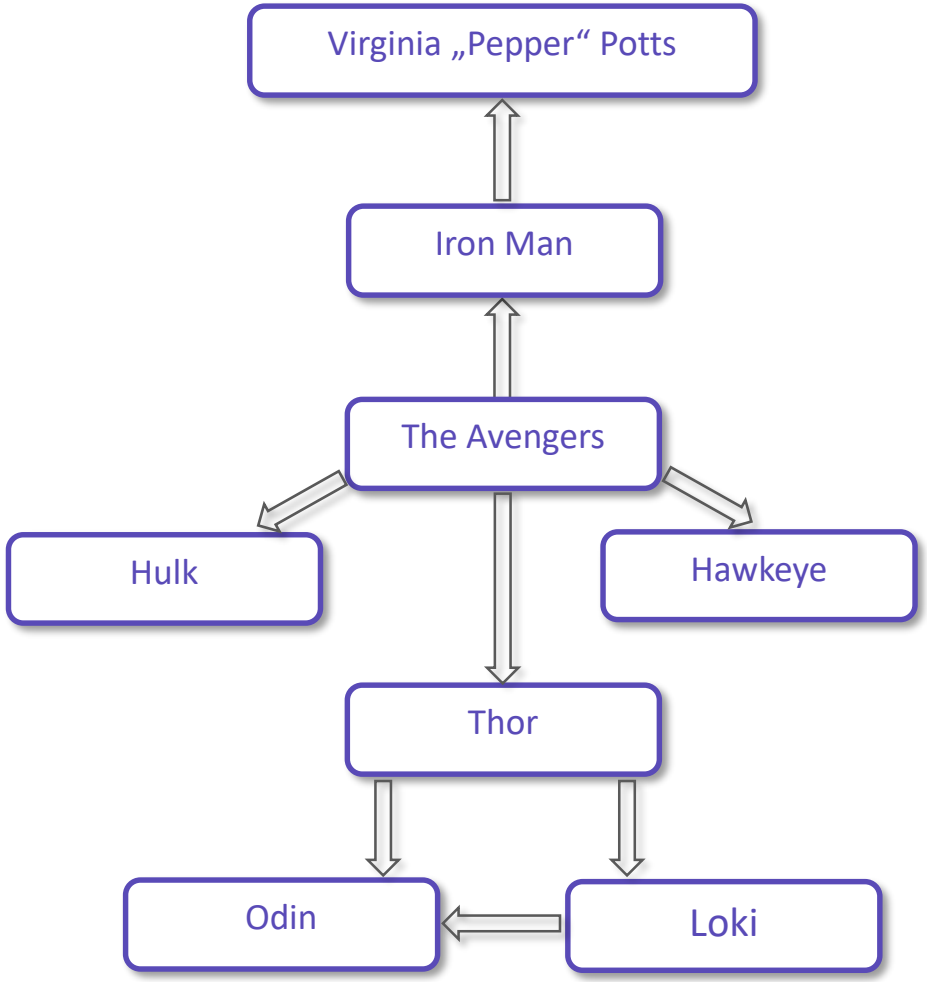
Document DB

```
{
Group_name: “The Avengers“,
Residence: “New York“,
Superheroes: [
        {Name: “Tony Stark“, Alias: “Iron
         Man“},
        {Name: “Bruce Banner“, Alias: “Hulk“},
        {Name: “Thor Odinson“, Alias: “Thor“},
        ... ]
}
```

- **Prinzip:**
  Collections von Key/Value-Paaren in Dokumenten-Struktur (XML, JSON, BSON)
- **Datenmodell:** Strukturiert
- **Use Cases:**
  - Historische Daten bzw. Archivierung
  - Single Views
  - Produkt Katalog bzw. Warenkorb
  - Benutzer-generierter Content (z.B. Blogs)
  - Real-Time Analytics
- **Wichtigste Vertreter:**
  MongoDB (2009), Couchbase (2011)

Grundlegende Einteilung

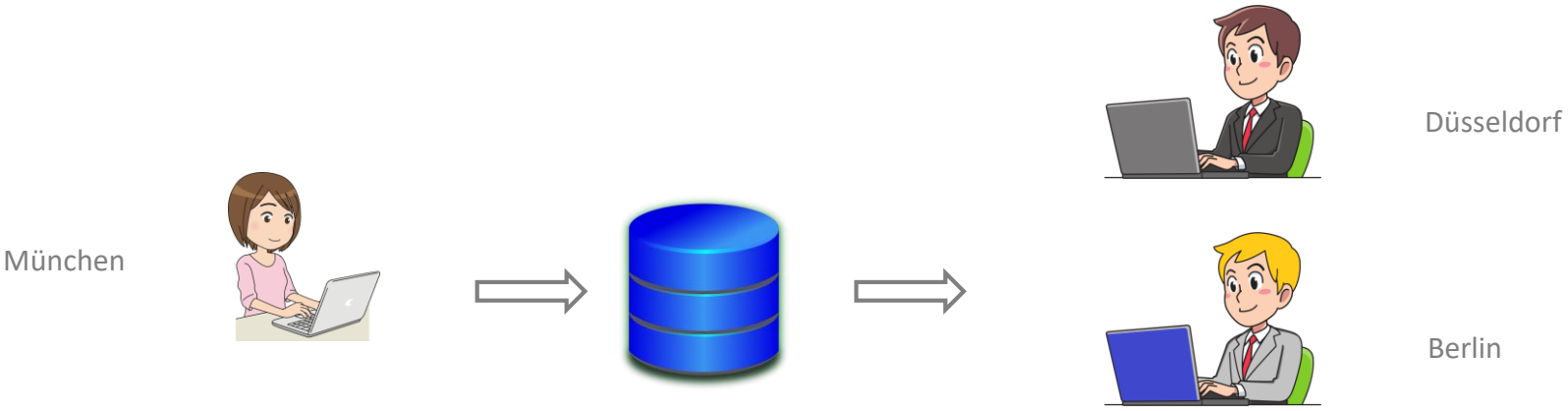
Graph DB



- **Prinzip:**
  Daten werden in Knoten + Linien (Verknüpfungen) gespeichert
- **Datenmodell:** Netzwerk
- **Use Cases:**
  - Produkt- oder Freundschafts-Empfehlungen
  - optimale Routen („Problem des Handlungsreisenden“)
- **Wichtigste Vertreter:**
  Neo4j (2007)

CAP-Theorem

Constistency

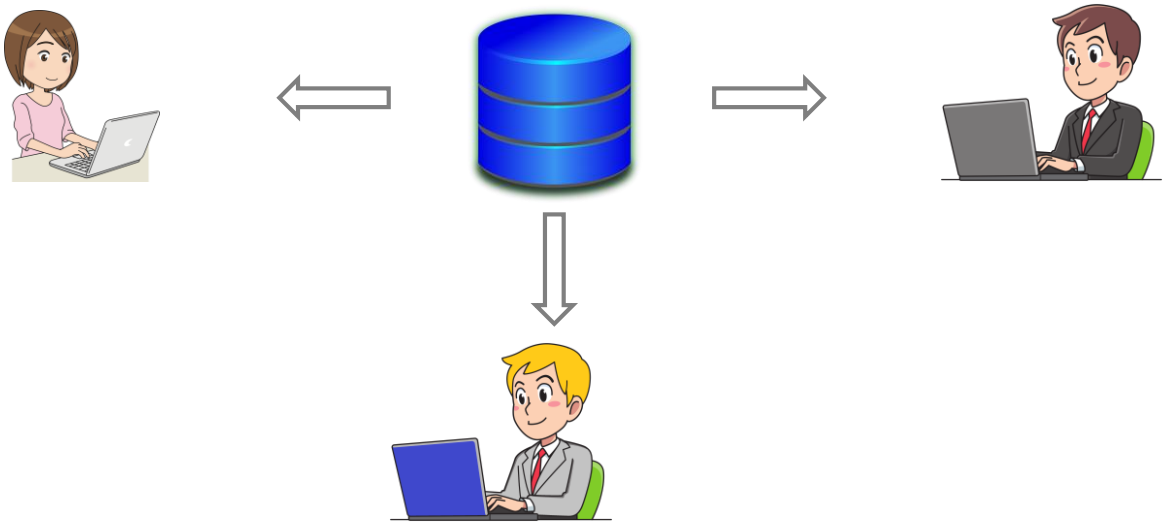


(A) Buche Zimmer ⟹ (B) Zimmer ist gebucht (für beide gleich)

- **Consistency (Konsistenz): Geschriebene Daten sind für alle Benutzer gleich**
- Availability (Verfügbarkeit): DB-System ist für jeden jederzeit verfügbar
- Partition tolerance: Kommunikationsfehler zwischen Komponenten stören nicht die Funktionsweise des Gesamtsystems

CAP-Theorem

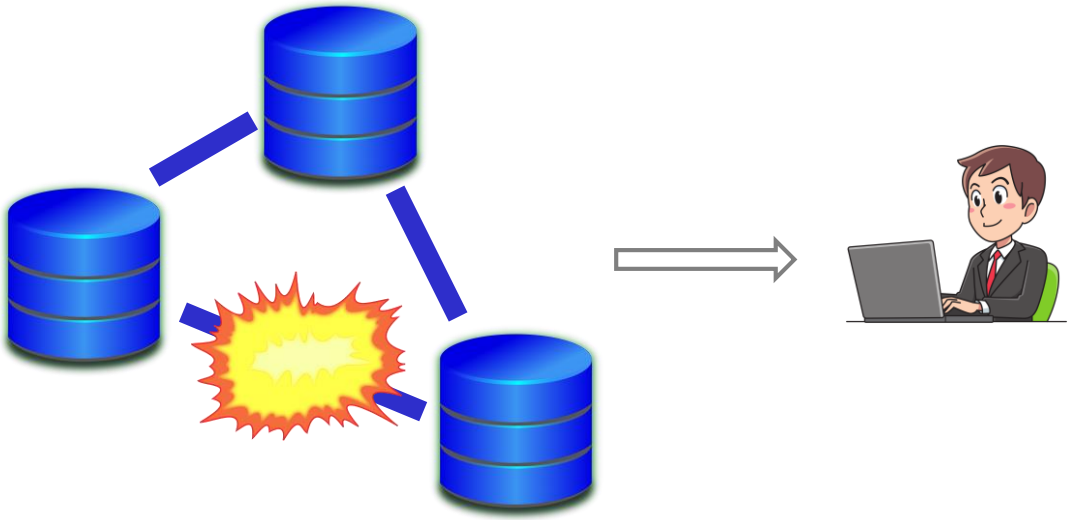
Availability



- Consistency (Konsistenz): Geschriebene Daten sind für alle Benutzer gleich
- **Availability (Verfügbarkeit): DB-System ist für jeden jederzeit verfügbar**
- Partition tolerance: Kommunikationsfehler zwischen Komponenten stören nicht die Funktionsweise des Gesamtsystems

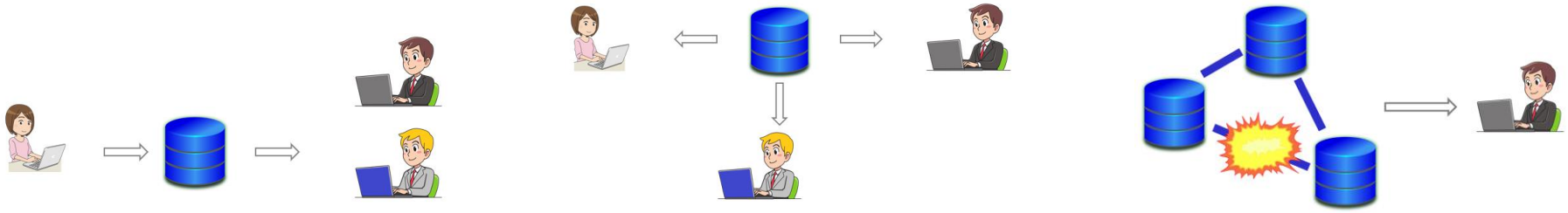
CAP-Theorem

Partition tolerance



- Consistency (Konsistenz): Geschriebene Daten sind für alle Benutzer gleich
- Availability (Verfügbarkeit): DB-System ist für jeden jederzeit verfügbar
- **Partition tolerance: Kommunikationsfehler zwischen Komponenten stören nicht die Funktionsweise des Gesamtsystems**

CAP-Theorem
Gesamtdarstellung



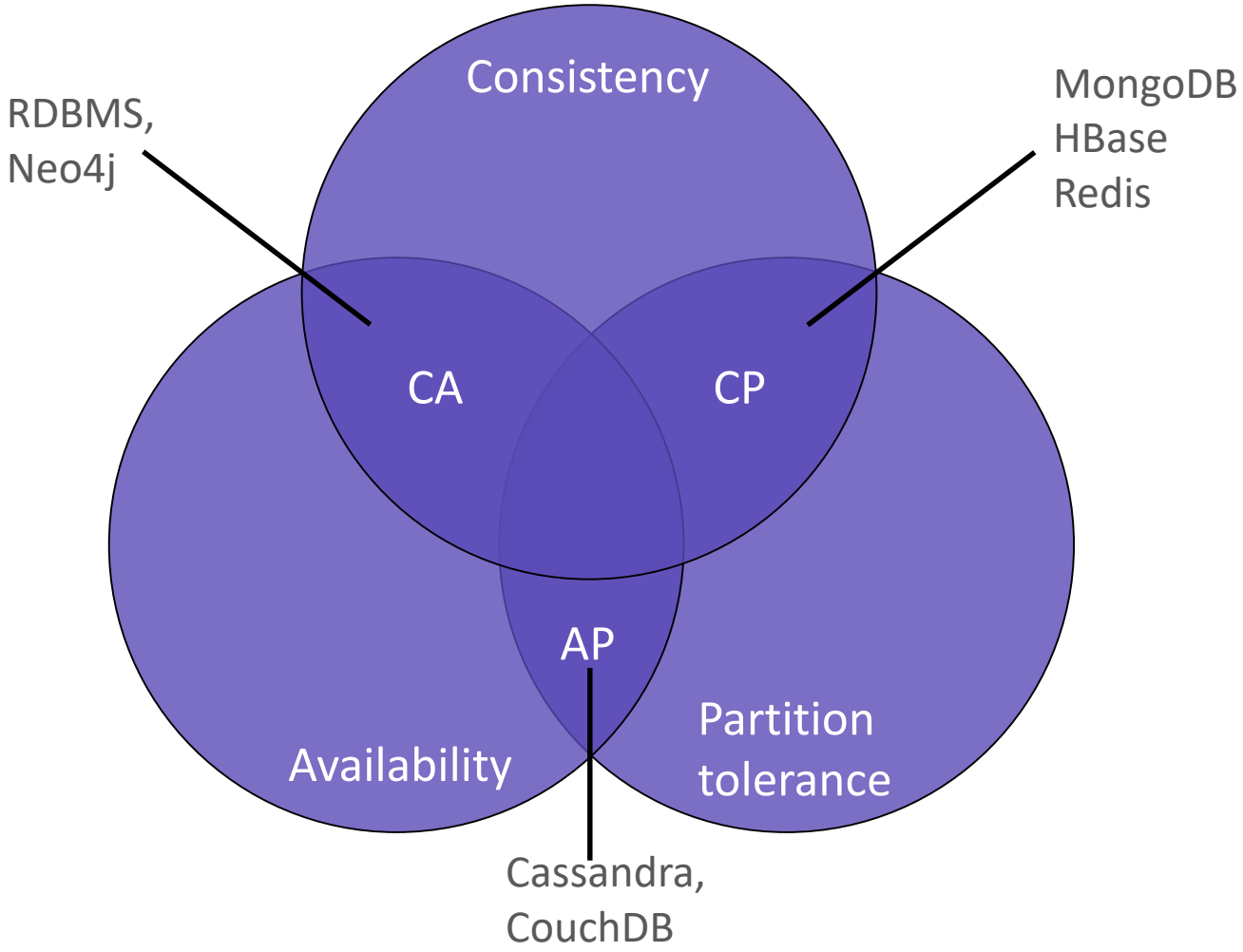
- Consistency (Konsistenz): Geschriebene Daten sind für alle Benutzer gleich
- Availability (Verfügbarkeit): DB-System ist für jeden jederzeit verfügbar
- Partition tolerance: Kommunikationsfehler zwischen Komponenten stören nicht die Funktionsweise des Gesamtsystems

CAP-Theorem: Wenn P auftritt, sind C und A nicht gleichzeitig erfüllbar

Anmerkung: Konsistenz und Verfügbarkeit haben eine andere Bedeutung als bei ACID

CAP-Theorem

DB-Ausprägungen



SQL vs. NoSQL

avision
REVIVING YOUR SOFTWARE

SQL

- Optimiert für zentralisierte Datenbanken (vertical scaling)
- Gewährt strenge Transaktions-Sicherheit (ACID)
- Festes, logisch-konsistentes, nicht-redundantes Datenmodell
- SQL ist high-level Query-Sprache

Probleme

- Skalierung (z.B. bei Performance-Problemen)
- Statistisches Datenmodell & Schema-Erweiterung
  - Workarounds zerstören oftmals die Konsistenz des DB-Modells

Use Cases

- Nur eine Datenquelle
- Systeme mit wenig Benutzern
- Strukturierte Daten
- Komplexe Transaktionen

NoSQL

- Optimiert für verteilte Datenbanken (horizontal scaling)
- Hohe Verfügbarkeit & einfache Skalierbarkeit (Sharding)
- Flexibles, agiles Datenmodell
- Bessere Lese- und Schreib-Performance für große Datenmengen

Probleme

- Transaktions-Sicherheit
- Umstrukturierungen des DB-Models (z.B. neue Relationen)
- Keine einheitliche Technologie

Use Cases

- Verschiedene Datenquellen
- Systeme mit vielen Benutzern & Cloud Computing
- Semi-Strukturierte, verschachtelte Daten
- Einfache Transaktionen

Der Weg zur richtigen Datenbank

avision
REVIVING YOUR SOFTWARE

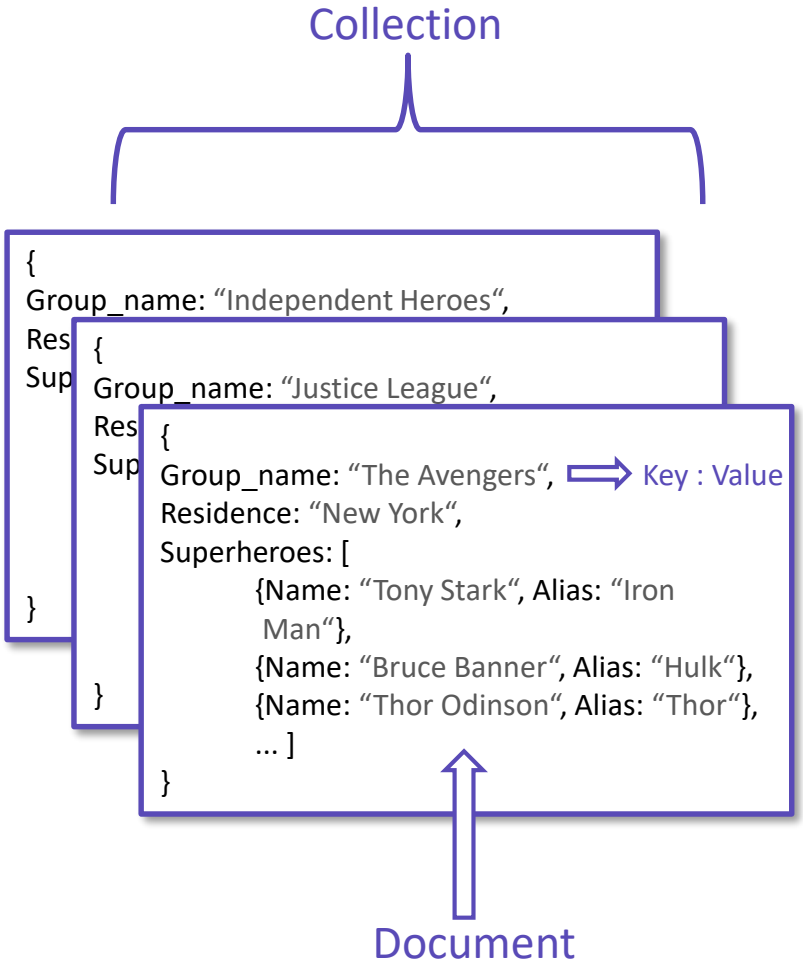
- Skalierung (horizontal/vertikal oder Cloud) & Performance
- Datenquellen (wenige/viele)
- Datenmodell klassifizieren (strukturierte/unstrukturierte Daten)
- Transaktionen (einfach/komplex)
- Popularität (z. B. https://db-engines.com)
- Legacy/Neu-Anwendung

  → Möglichkeiten für Legacy-Anwendungen
  - Migration des Datenmodell
  - Polyglot persistence: Kombination verschiedener Datenbanken-Technologien

MongoDB
Grundlegendes

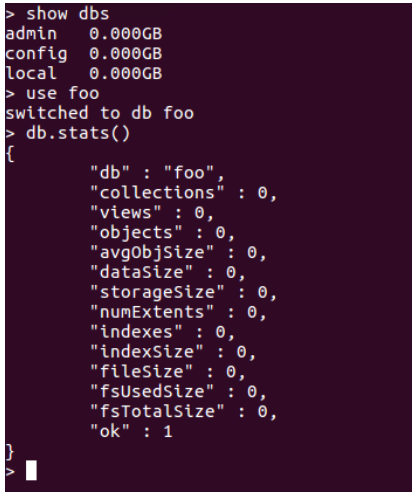
- Name abgeleitet von “humongous” (engl.): “gigantisch”
- Geschrieben in C++
- BSON-Basiert (Binary JSON)
- Varianten
  - Open Source: GNU AGPL v3.0
  - Kommerziell: "MongoDB Enterprise Advanced "
    - Tools (Ops Manager, MongoDB Compass)
    - Commercial development license
    - Zusätzliche Sicherheits-Features: Kerberos/LDAP-Authentifizerung, Auditing
    - In-Memory storage engine
- Plattformunabhängig (Windows, Linux, OS X, Solaris)
- GridFS (“Grid File System”) für Dateien größer als 16 MB (maximale Dokumentengröße)

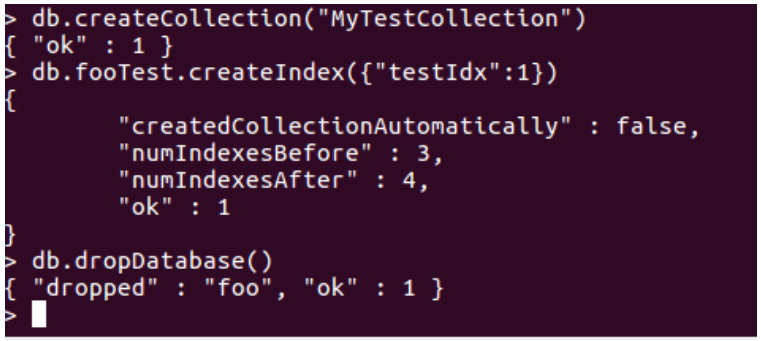
MongoDB
DB-Entwickler Sich



- Datenorganisation
  - SQL-Table: Collection von JSON-Dokumenten
  - SQL-Row: Document
    - Aufbau: `Key : Value`
    - Eindeutigkeit durch `_id` (ObjectId)
- Zugriff auf MongoDB
  - Code: JavaScript, Java, Python, Scala
  - GUI-Tools: MongoDB Compass, Robot 3T
  - DB-Shell: `mongo`
    - Interaktives JavaScript Interface
    - Kann JS-Skripts ausführen: `mongo <JS-File>`

MongoDB
Mongo Shell (Kommandos)





System

- Hilfe: `db.help()`
- Show database: `show dbs`
- Create/Switch to DB: `use <database>`
- Show statistics: `db.stats()`

Basics

- Create collection: `db.createCollection(<name>, <options>)`
- Create Index: `db.<collection>.createIndex( {y:1} )`
- Create view: `db.createView(<view>, <source>, <pipeline>, <collation> )`
- Drop database: `db.dropDatabase()`

MongoDB
Mongo Shell (CRUD Operations)

Allgemeines Kommando-Prinzip: db.<collection>.<command>(JSON)

JSON: strukturiert durch Operatoren (`$and`, `$or`, `$gt`, `$in`, …), <keys> und <values>

- Create
  - Prinzip: db.<collection>.insert(<document>)
  - Bsp.: `db.heroes.insertOne({ Name: "Bruce Wayne", Alias: "Batman", Age: 35 })`
- Read
  - Prinzip: db.<collection>.find(<query criteria><projections>)
    - Projections: Selektive Darstellung von gewünschten Feldern
  - Bsp.:
    - `db.heroes.find({ Age: { $gt: 18 } }, { Name: 0 } ).limit(5)`
    - `db.heroes.find({ Name: { $in: [ "Bruce Wayne", "Clark Kent" ] } })`
    - `db.heroes.find({ $or: [ { Alias: "Batman" }, { Age: 40 } ] })`
- Update
  - Prinzip: db.<collection>.update(<update filter>,<updated action>)
  - Bsp.: `db.heroes.updateOne({ Name: "Bruce Wayne" }, { $set: { Name: "B. W." } } )`
- Delete
  - Prinzip: db.<collection>.deleteOne(<delete filter>)
  - Bsp.: `db.heroes.deleteOne({ Age: 35 })`

Siehe "SQL to MongoDB Mapping Chart" (MongoDB Manual)

MongoDB
MongoDB & Java

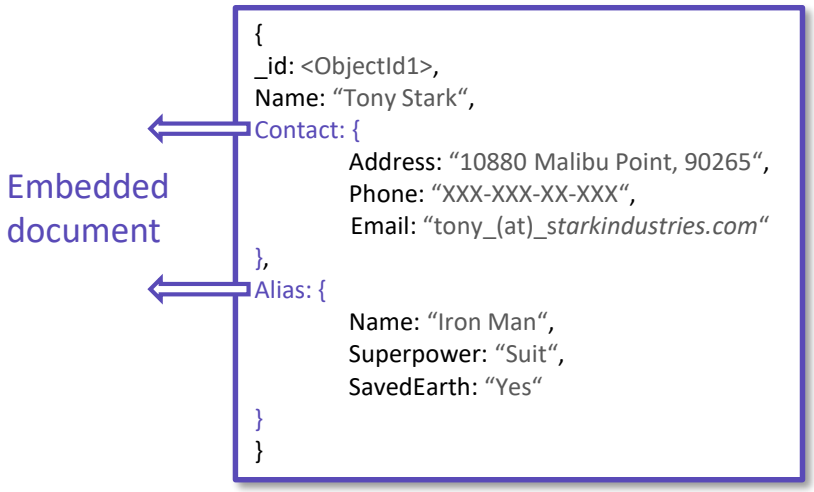
- Zwei Zugriffsmöglichkeiten
  - `MongoTemplate`
    - `MongoClient`: Interface between Java und MongoDB (connect to db, create collection, …)
    - Bsp.: `mongoTemplate.insert(new Address(), "Address");`
  - `MongoRepository`
    - Bsp.: `mongoTemplate.insert(new Address(), "Address");`
- `MongoConverter`
  - Mapping of Java Objekt → Dokument
  - Möglichkeiten
    - Direkt (Annotations-Basiert): `@Document`, `@ID`, …
    - Custom MongoConverter-Implementation
- `MongoTemplate` hat Methoden für fundamentale DB-Operationen
  - z.B. createCollection(), mapReduce(), …
  - vgl. mit Relationaler DB (JDBC): Execute SQL-Query
- Spring Boot 2: Reactive MongoDB (Zugriff gemäß Publisher-Subscriber Pattern)

```java
@Document
public class Person {

    @Id
    private String id;

    @Field("fName") //Mapping to MongoDB-Field "vName"
    private String firstName;

    @Indexed
    private String surName;

    private int age;

    @Transient //Feld wird bei dem Mapping ignoriert
    private int size;

    @PersistenceConstructor
    public Person(String firstName, String surName, int age) {
        this.firstName = firstName;
        this.surName = surName;
        this.age = age;
    }

    public String getId() {
        return id;
    }

    //...
}
```

Mongo DB

Schema-Design: Embedded documents

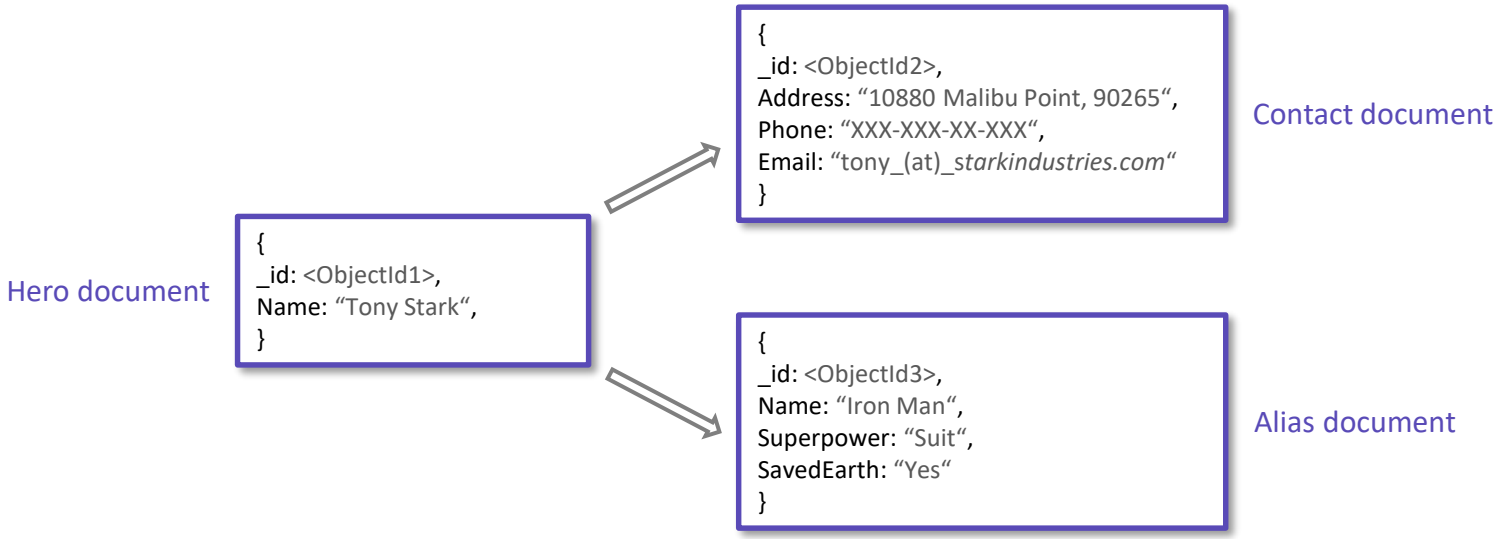
- **Prinzip:** flexibles Schema
  - Optionale Dokumenten-Validierung (bei update/insert) ist möglich
- **Schlüsselfaktoren**
  - Atomicity: Transaktions-Sicherheit (derzeit) nur auf Dokumenten-Ebene
  - Use Cases der Datenverarbeitung in der Anwendung (Queries, Updates,...)
  - Dokumentenstruktur
    - **Embedded documents**: für "contains- " & "one-to-many relationships" (1 Haupt-Dokument greift immer auf viele Unter-Dokumente zu)
    - Referenzen: für komplexe " many-to-many relationships"



Mongo DB

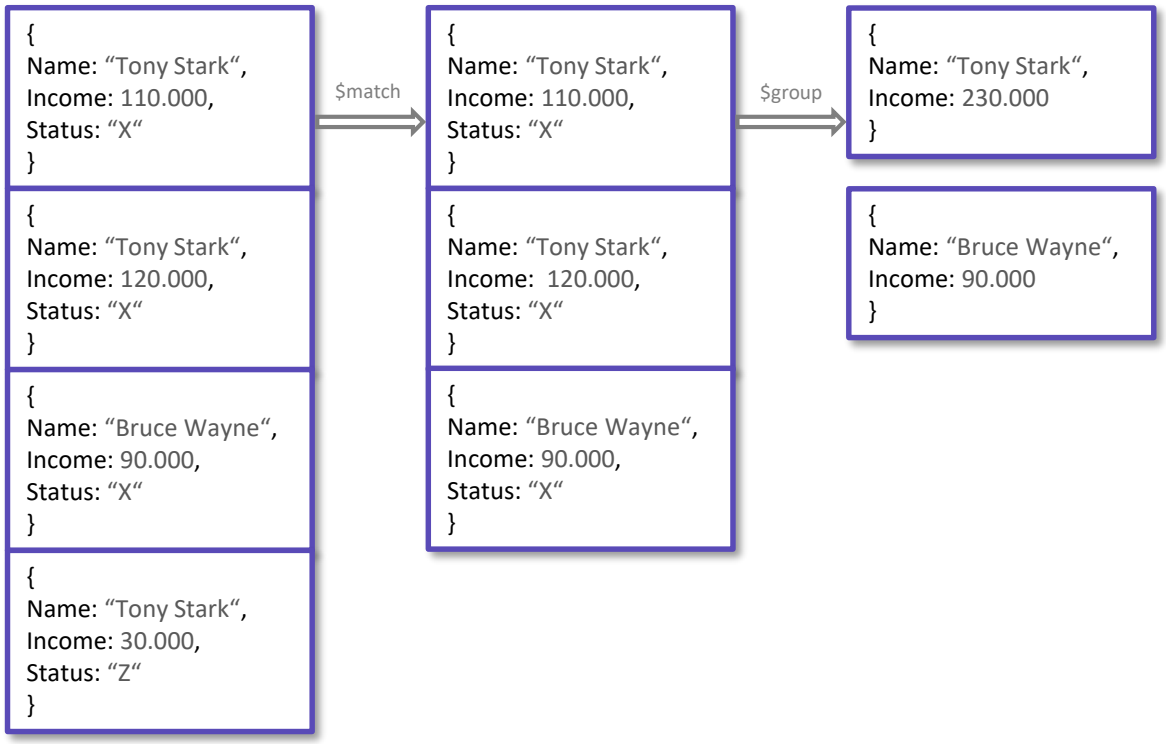
Schema-Design: Referenzen

- **Prinzip:** flexibles Schema
  - Optionale Dokumenten-Validierung (bei update/insert) ist möglich
- **Schlüsselfaktoren**
  - Atomicity: Transaktions-Sicherheit (derzeit) nur auf Dokumenten-Ebene
  - Use Cases der Datenverarbeitung in der Anwendung (Queries, Updates,...)
  - Dokumentenstruktur
    - Embedded documents: für "contains- " & "one-to-many relationships" (1 Haupt-Dokument greift immer auf viele Unter-Dokumente zu)
    - **Referenzen**: für komplexe " many-to-many relationships"



MongoDB
Aggregation

```
db.heroes.aggregate([
$match stage →        { $match: { Status: "X" } },
$group stage →        { $group: { _id: "$Name", total: { $sum: "$Income" } } }
])
```



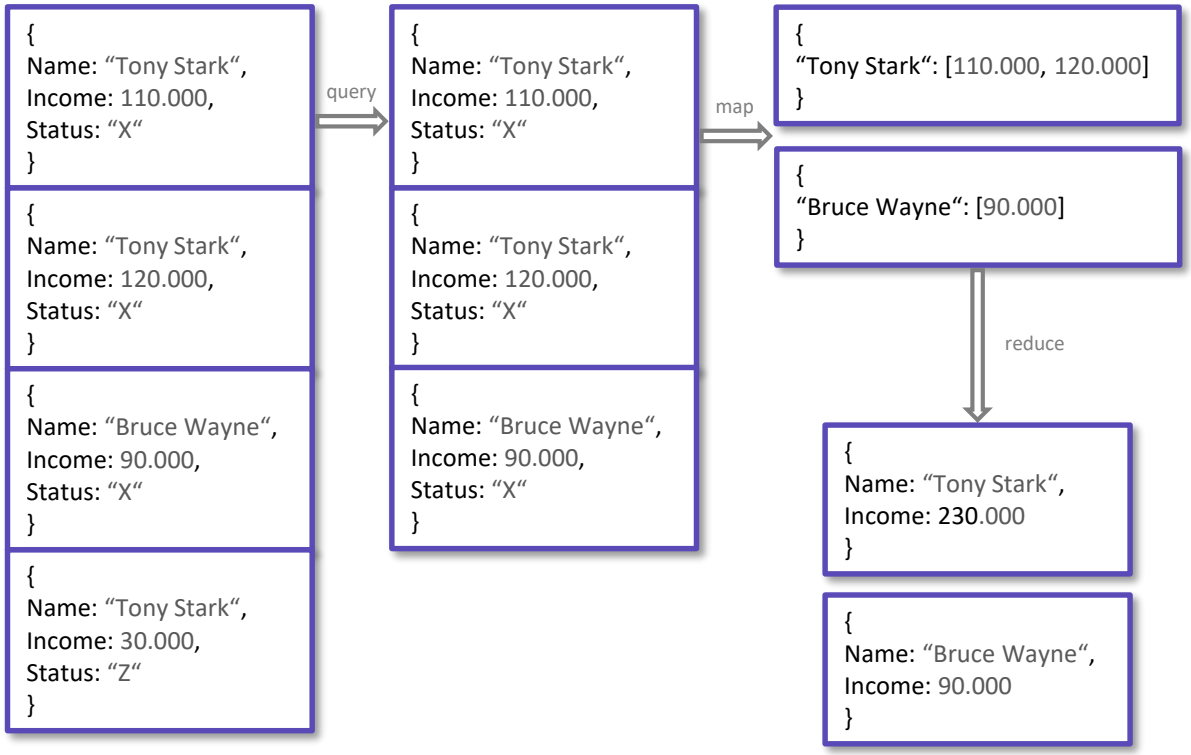
- **Prinzip**
  a. Gruppiere Werte aus verschiedenen Dokumenten
  b. Führe Operationen durch
  c. Reduziere auf einziges Ergebnis

- **Möglichkeiten**
  - **Aggregation pipeline**
  - Map-Reduce Funktion
  - Single-Purpose Aggregation-Methoden

siehe "SQL to Aggregation Mapping Chart" (MongoDB Manual)

MongoDB
Aggregation

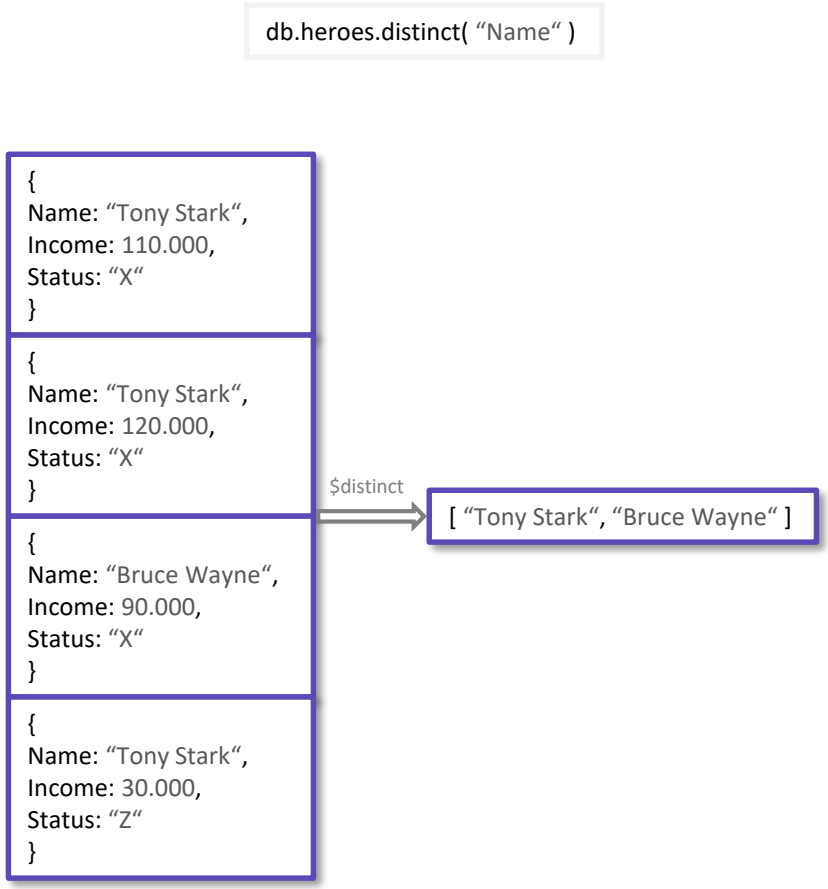
```
db.heroes.mapReduce(
map →        function() { emit( this.Name, this.Income ); },
reduce →     function(key, values) { return Array.sum(values) },
             {
query →       query: { Status: "X" },
output →      out: "result"
             } )
```



- **Prinzip**
  a. Gruppiere Werte aus verschiedenen Dokumenten
  b. Führe Operationen durch
  c. Reduziere auf einziges Ergebnis

- **Möglichkeiten**
  - Aggregation pipeline
  - **Map-Reduce Funktion**
  - Single-Purpose Aggregation-Methoden

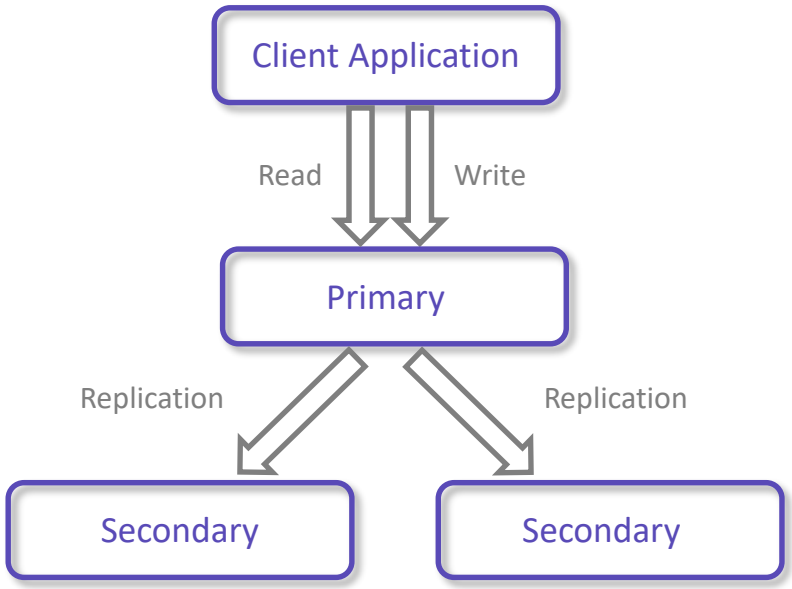
siehe "SQL to Aggregation Mapping Chart" (MongoDB Manual)



- **Prinzip**
  a. Gruppiere Werte aus verschiedenen Dokumenten
  b. Führe Operationen durch
  c. Reduziere auf einziges Ergebnis

- **Möglichkeiten**
  - Aggregation pipeline
  - Map-Reduce Funktion
  - **Single-Purpose Aggregation-Methoden**

siehe "SQL to Aggregation Mapping Chart" (MongoDB Manual)

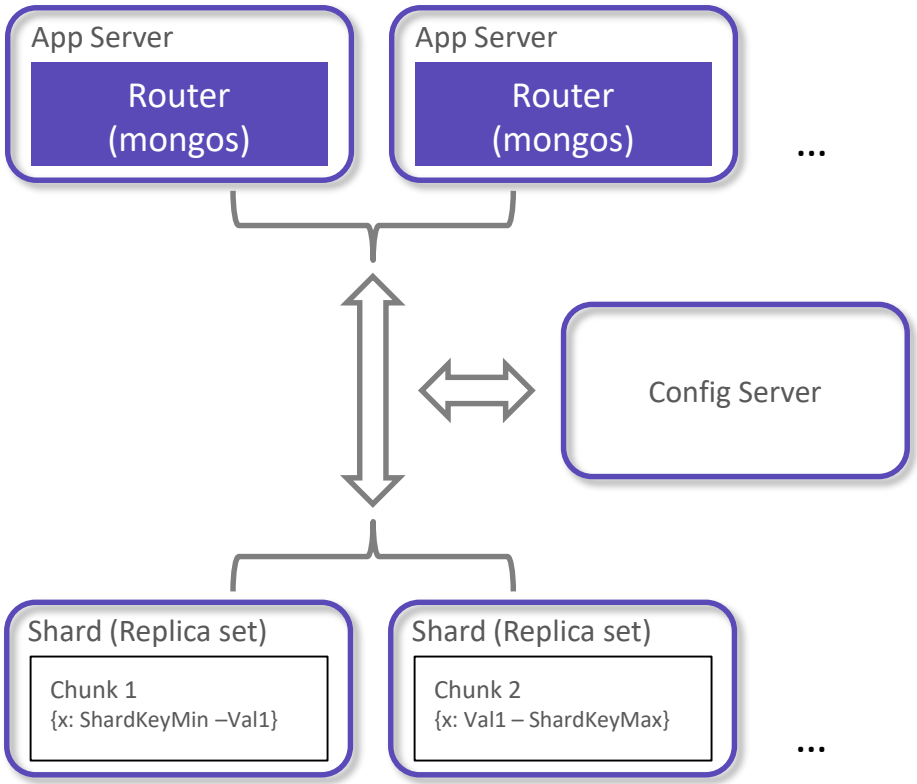


- **Prinzip:** Gruppe von `mongod`-Instanzen
  - 1 Primary: empfängt alle Write-Operationen
  - N Secondary: replizieren von Primary

- **Failover-Prozess**
  - Wenn Primary nicht erreichbar
    → Auswahlverfahren zwischen den Secondaries
    → Wahl eines neuen Primary
  - Schiedsrichter (engl. „Arbiter“): Instanz ohne Daten

MongoDB
Sharding

- **Prinzip**: Verteilung von Daten auf mehrere Maschinen (gemäß "horizontal scaling")
- **Komponenten**
  - Query router (Mongos)
  - Shards: Sub-Set der Daten
  - Config Server: Metadaten & Konfiguration
- **Shard Key**
  - Schlüssel, der die Partitionierung in Chuncks bestimmt
  - Index-Feld (einzeln, Compound), dass in jedem Dokument existiert
  - Wird beim "Sharden" der Collection festgelegt und kann dann nicht mehr geändert werden
  - Hat Auswirkung auf die Performance, Effizienz und Skalierbarkeit des Sharded Clusters



Zusammenfassung

- NoSQL erweitert die Datenbank-Landschaft
- Grundlegende Einteilung: Key/Value-, Wide-Column-, Document-, Graph-DB
- MongoDB
  - Ist eine dokumentenbasierte Datenbank mit Zugriffsmöglichkeiten über (Mongo-)Shell, GUI-Tools & Code
  - Programmiersprachen-Treiber (z.B. Java) erlauben das komplette DB-Design & Entwicklung
  - Horizontale Skalierung wird durch Sharding erreicht
- Schema-Design (vergleichbar mit Hausbau)
  - Relationale Datenbank: Fertighaus (Einrichtung gemäß fester Vorgaben?)
  - NoSQL-Datenbank: klassischer Hausbau (Wie bleibt Haus stabil?)