Big-Data-Technologien

Kapitel 13: NoSQL – Dokumentendatenbanken

Hochschule Trier
Prof. Dr. Christoph Schmitz

Dokumentenorientierte Datenbanken

- … speichern **semistrukturierte Daten**
- … benötigen **kein fixes Schema**
- … bieten **flexible Anfragemöglichkeiten**
- … **verallgemeinern** Key-Value-Stores

Semistrukturiert?!

- **Strukturierte** Daten
  - Relationale Tabellen
  - Schema: Felder, Datentypen, …
- **Unstrukturierte** Daten
  - Fließtext
  - Grafik
  - Video

Semistrukturierte Daten

- Daten sind **maschinenlesbar** und **selbstbeschreibend**
- Schema **möglich**, aber optional
- Oft **hierarchisch** gegliedert
- Vertreter
  - **XML**
  - **JSON**
  - **YAML**

- "Hu**mongo**us Database"

- Schema-freie, **dokumentenorientierte** Datenbank
- Dokumente in **JSON**
- Skalierbar über **Sharding**

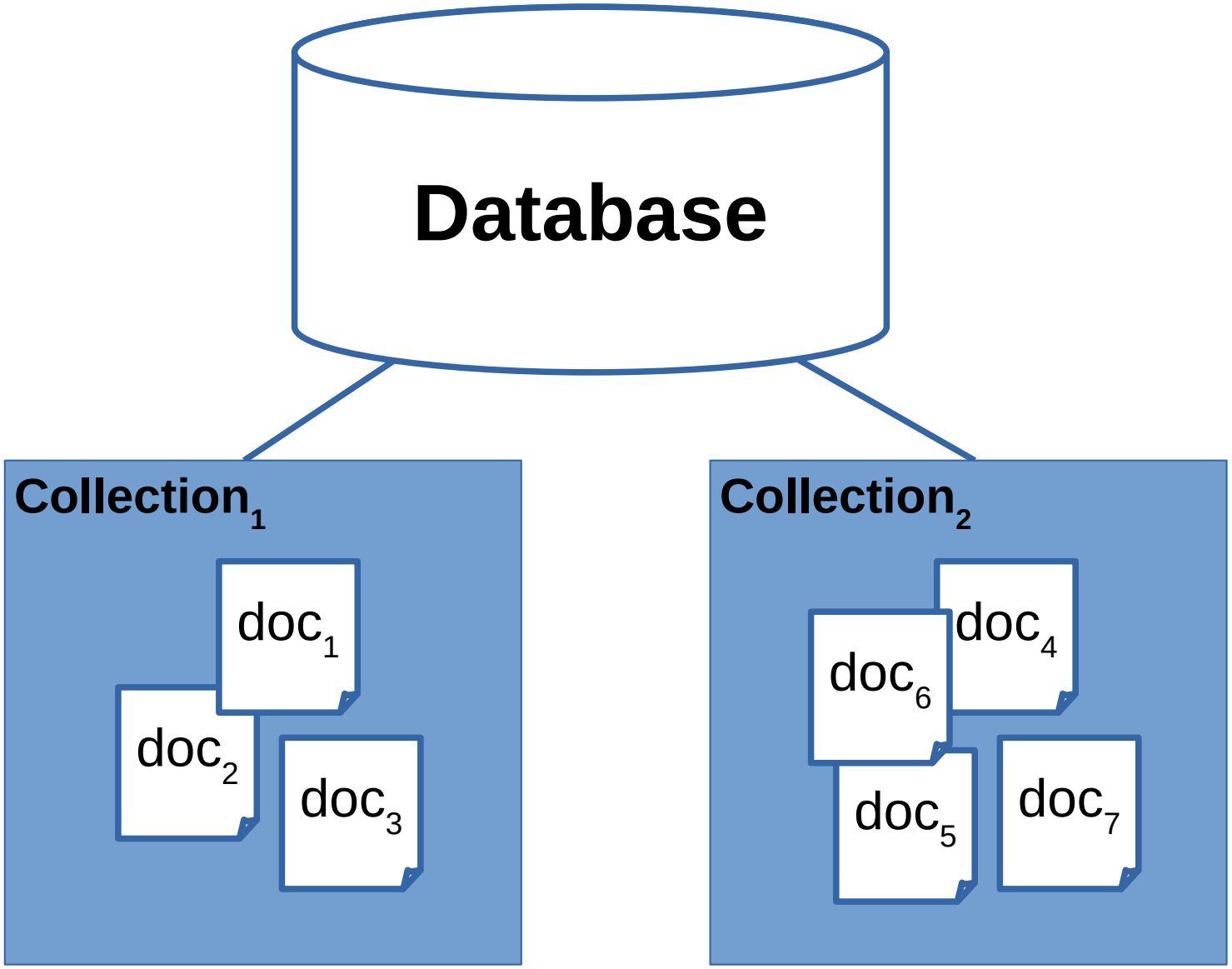
Dokument

```
{ "_id" : ObjectId("5925c3d0"),
  "name" : "Peter Müller",
  "country" : "Germany" }
```

Dokument

```
{ "_id" : ObjectId("5925c3d0"),
  "name" : "Peter Müller",
  "country" : "Germany",
  "addresses": [
      { "street": "Hauptstr. 3",
        "zip": "54329",
        "city": "Konz" },
      { "street": "Kaiserstr. 5",
        "zip": "76135",
        "city": "Karlsruhe" }
]}
```

Struktur einer Datenbank



Beispielsitzung

```
$ mongo --host …
> use bigdata200
switched to db bigdata200

> db.createCollection("users")
{ "ok": 1 }

> db.users.insert({name: "Eva Meier", age: 23})
WriteResult({ "nInserted" : 1 })

> db.users.find()
{ "_id" : ObjectId("59258a7328d7bc62e9dd0c82"),
"name" : "Eva Meier", "age" : 23 }
```

Beispielsitzung

```
> db.users.insert({name: "Eva Meier", age: 23})
WriteResult({ "nInserted" : 1 })
> db.users.insert({name: "Paul Müller", age:
30})
…

> db.users.find()
{ "_id" : ObjectId("59258a7328d7bc62e9dd0c82"),
"name" : "Eva Meier", "age" : 23 }
{ "_id" : ObjectId("59258add09d1f714f0dac423"),
"name" : "Eva Meier", "age" : 23 }
{ "_id" : ObjectId("59258ba209d1f714f0dac424"),
"name" : "Paul Müller", "age" : 30 }
```

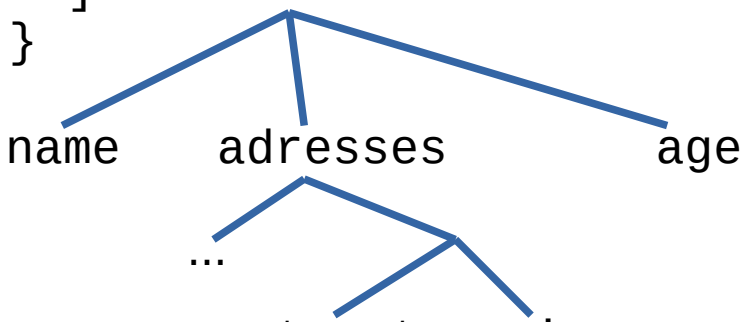
Beispielsitzung

```
> db.users.find({ age: 30 })
{ "_id" : ObjectId("59258ba209d1f714f0dac424"),
"name" : "Paul Müller", "age" : 30 }
```

Datenmodell und Anfragen

- **JSON** (JavaScript Object Notation)/**BSON**
  - **Skalare** (Strings, Zahlen, …)
  - **Listen**
  - **Dictionaries (Objekte)**

```
{ name: "Eva Meier",
  age: 23,
  addresses: [
    { street: "Paulinstraße 17", zip: 54290, city: "Trier" },
    { street: "Wilhelmstraße 3", zip: 34117, city: "Kassel" }
  ]
}
```




```
doc.addresses[0].zip → 54290
```

Anfragen

- Formuliert in JSON
- `db.<collection>.find()`

```
db.users.find({ age: 30 })
db.users.find({ _id: ObjectId("ab82...fd")})
db.users.find({ age: { $gt: 30 }})
db.users.find({ "address.zip": 54290 })
```

- Mehrere Bedingungen → UND
- ODER: `$or`

```
db.users.find({ $or: [ { age: { $lt: 18 }},
                       { age: { $gt: 80 }} ]})
```

Anfragen: Operatoren

- `$in: {name: {$in: ["Eva", "Paul"]}}`
- `$all: {features: {$all: ["USB-C", "NFC"]}}`
- `$elemMatch: {a: {$elemmatch: {b: 1, c: 2}}}`
- `$mod: {a: {$mod: [3, 0]}}`
- `$type: {a: {$type: 2}}` → bsonspec.org
- Text: `{$text: {$search: "hello"}}`

Anfragen: Operationen

- Nicht durch Indizes unterstützt:
  - Nicht-Enhaltensein:
    ```
    {a: {$nin: ["foo", "bar"]}}
    ```
  - Array-Länge: `{a: {$size: 3}}`
  - Existenz eines Felds: `{a: {$exists: true}}`
  - Regex: `{a: /foo.*bar/}`
  - Negation: `{a: {$not: {$type: 2}}`

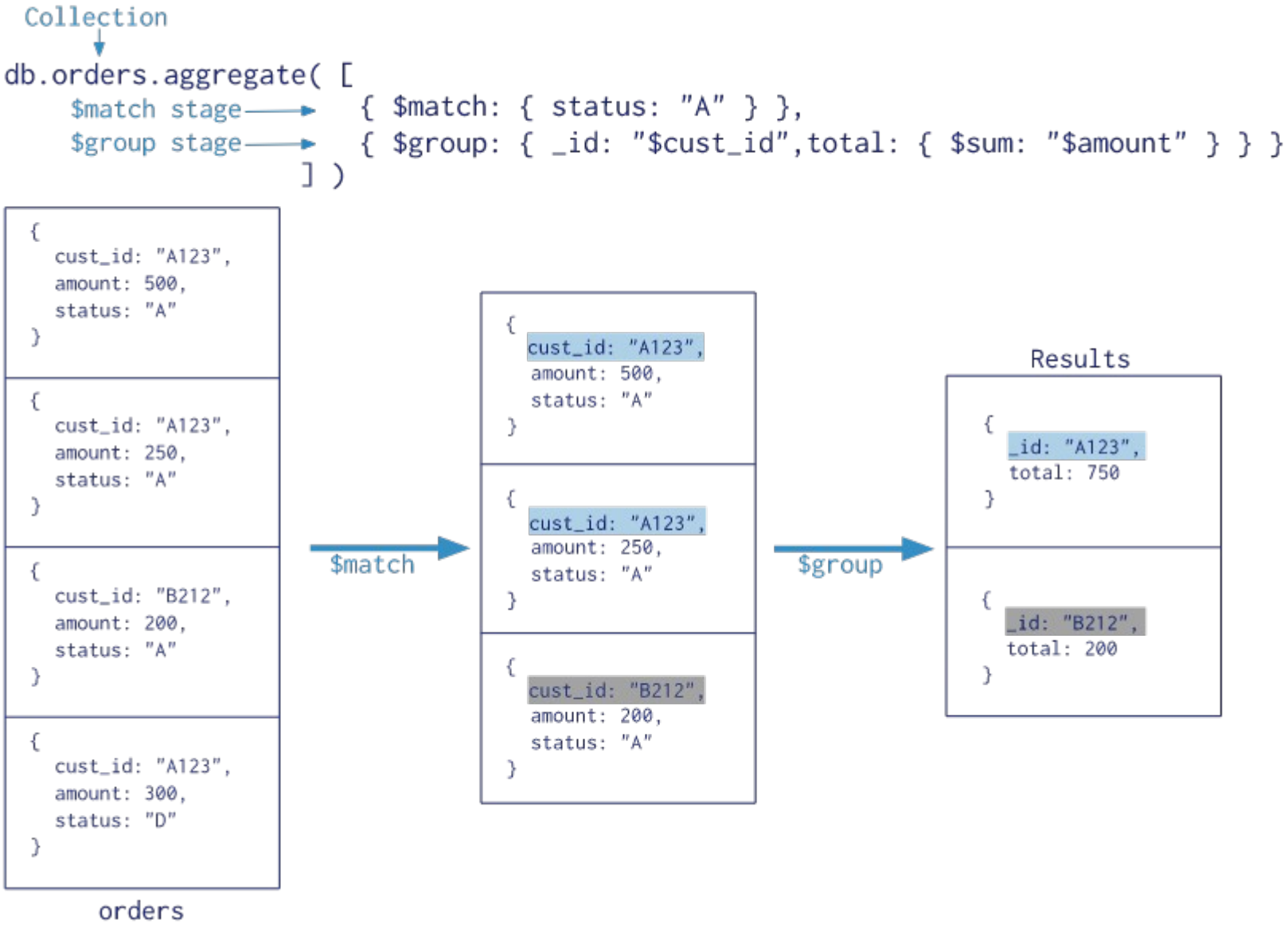
Anfragen: Projektion

- Projektion auf bestimmte Felder

```
db.users.find({ age: { $gt: 30 }},
      { "address.zip": 1, _id: 0 })


{ "address": { "zip": 54329 }}
{ "address": { "zip": 54290 }}
...
```

Aggregation: Aggregation Pipeline



Quelle: docs.mongodb.com

Aggregation Pipeline: Operationen

- `$match`, `$skip`, `$limit`, `$redact`, `$sample`
- `$project`, `$addfields`
- `$groupBy`, `$sort`, `$sortByCount`
- `$bucket`, `$bucketAuto`
- `$unwind`
- `$out`, `$count`

Aggregation: Beispiele

```
> db.users.aggregate([ {$bucket: { groupBy: "$age",
boundaries: [0, 14, 20, 30, 40, 50, 60, 100] }}])

{ "_id" : 20, "count" : 2 }
{ "_id" : 30, "count" : 2 }
```

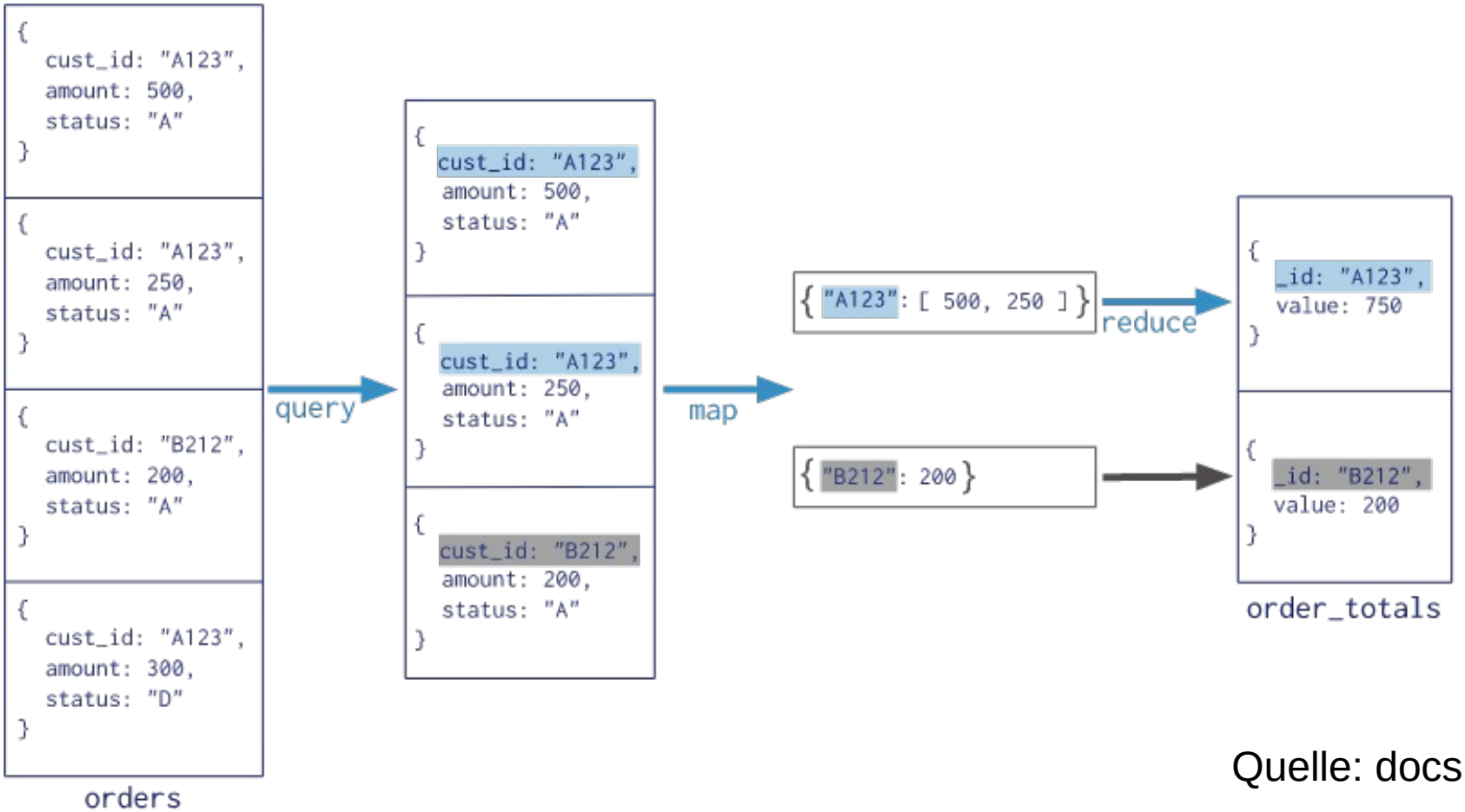
```
> db.users.aggregate([ {$group: { _id: "$gender",
average_age: { $avg: "$age"} }},
{ $sort: {"average_age": -1}}])

{ "_id" : "m", "average_age" : 32 }
{ "_id" : "w", "average_age" : 23 }
```

Aggregation: MapReduce

Collection

```
db.orders.mapReduce(
          map      ──►  function() { emit( this.cust_id, this.amount ); },
          reduce   ──►  function(key, values) { return Array.sum( values ) },
                        {
          query    ──►    query: { status: "A" },
          output   ──►    out: "order_totals"
                        }
                      )
```



Quelle: docs.mongodb.com

Aggregation (vereinfacht)

- `db.collection.count()`
- `db.collection.distinct("cust_id")`

Updates: Vorsicht!

```
> db.c.find()
{ "_id" : ObjectId("592c1777754e8ed1400cd110"), "name" : "Alice" }

> db.c.update({"name": "Alice"}, {"age": 23})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.c.find()
{ "_id" : ObjectId("592c1777754e8ed1400cd110"), "age" : 23 }
```

Updates

```
> db.c.find()
{ "_id" : ObjectId("592c1777754e8ed1400cd110"),
"name" : "Alice" }

> db.c.update({"name": "Alice"}, {$set: {"age": 23}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0,
"nModified" : 1 })

> db.c.find()
{ "_id" : ObjectId("592c1777754e8ed1400cd110"),
"name": "Alice", "age" : 23 }
```

Weitere Update-Operationen

- Inkrementieren: `{$inc: {age: 1}}`
- Max/Min: `{$max: {age: 50}}`
- Multiplikation: `{$mul: {salary: 1.05}}`
- Umbenennen: `{$rename: {salary: "income"}}`

- Bit-Operationen, Datum, Upsert, …

Updates auf Arrays

- Anfügen: `{$push: {a: 4}}}`
  `{$push: {a: {$each: [1, 2]}}}`
- Mengen-Vereinigung: `{$addToSet: {a: 4}}`
- Entfernen: `$pop`, `$pull`, `$pullAll`

Fremdschlüssel
(oder etwas Ähnliches…)

```
db.orders.insert({
    "item": "abc",
    "quantity": 2,
    "price": 89.99
})

db.inventory.insert({
    "sku": "abc",
    "description": "product 1",
    "instock": 120
})
```

Joins: $lookup

- Left Outer Join
- Nicht in Kombination mit Sharding

```
db.orders.aggregate([
    {
      $lookup:
        {
          from: "inventory",
          localField: "item",
          foreignField: "sku",
          as: "inventory_docs"
        }
   }
])
```

SQL:

```
SELECT …
FROM orders o LEFT OUTER JOIN inventory i
ON o.item = i.sku
…
```

Joins: $lookup

orders

```
{ "_id" : 1, "item" : "abc", "price" : 12, "quantity" : 2 }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1 }
{ "_id" : 3  }
```

items

```
{ "_id" : 1, "sku" : "abc", description: "product 1", "instock" : 120 }
{ "_id" : 2, "sku" : "def", description: "product 2", "instock" : 80 }
{ "_id" : 3, "sku" : "ijk", description: "product 3", "instock" : 60 }
{ "_id" : 4, "sku" : "jkl", description: "product 4", "instock" : 70 }
{ "_id" : 5, "sku": null, description: "Incomplete" }
{ "_id" : 6 }
```

Query

```
db.orders.aggregate([{
      $lookup:
        {
          from: "inventory",
          localField: "item",
          foreignField: "sku",
          as: "inventory_docs"
        }
}])
```

Resultat

```
{ "_id" : 1,
  "item" : "abc",
  "price" : 12,
  "quantity" : 2,
  "inventory_docs" : [
    { "_id" : 1, "sku" : "abc",
      description: "product 1", "instock" : 120 }
  ]
}
...
```

Document Validation

- Vergleichbar mit Trigger in RDBMS
- Beispiel:

```
db.createCollection( "contacts",
   { validator: { $or:
      [
         { phone: { $type: "string" } },
         { email: { $regex: /@mongodb\.com$/ } },
         { status: { $in: [ "Unknown", "Incomplete" ] } }
      ]
   }
} )
```

Codebeispiele: docs.mongodb.com

Indizes

- Effiziente **Zugriffspfade** auf Dokumente
  - … wie in RDBMS
- Ansonsten: Collection Scan

- Standardmäßig auf `_id`

Arten von Indizes

- **Einzelnes Feld**
- **Zusammengesetzt** (z. B. `(name, vorname)`)
- **Multikey** (auf Arrays)
- **Geo-Koordinaten** (räumlich)
- **Text** (Stemming, Stopwords, …)

Eigenschaften von Indizes

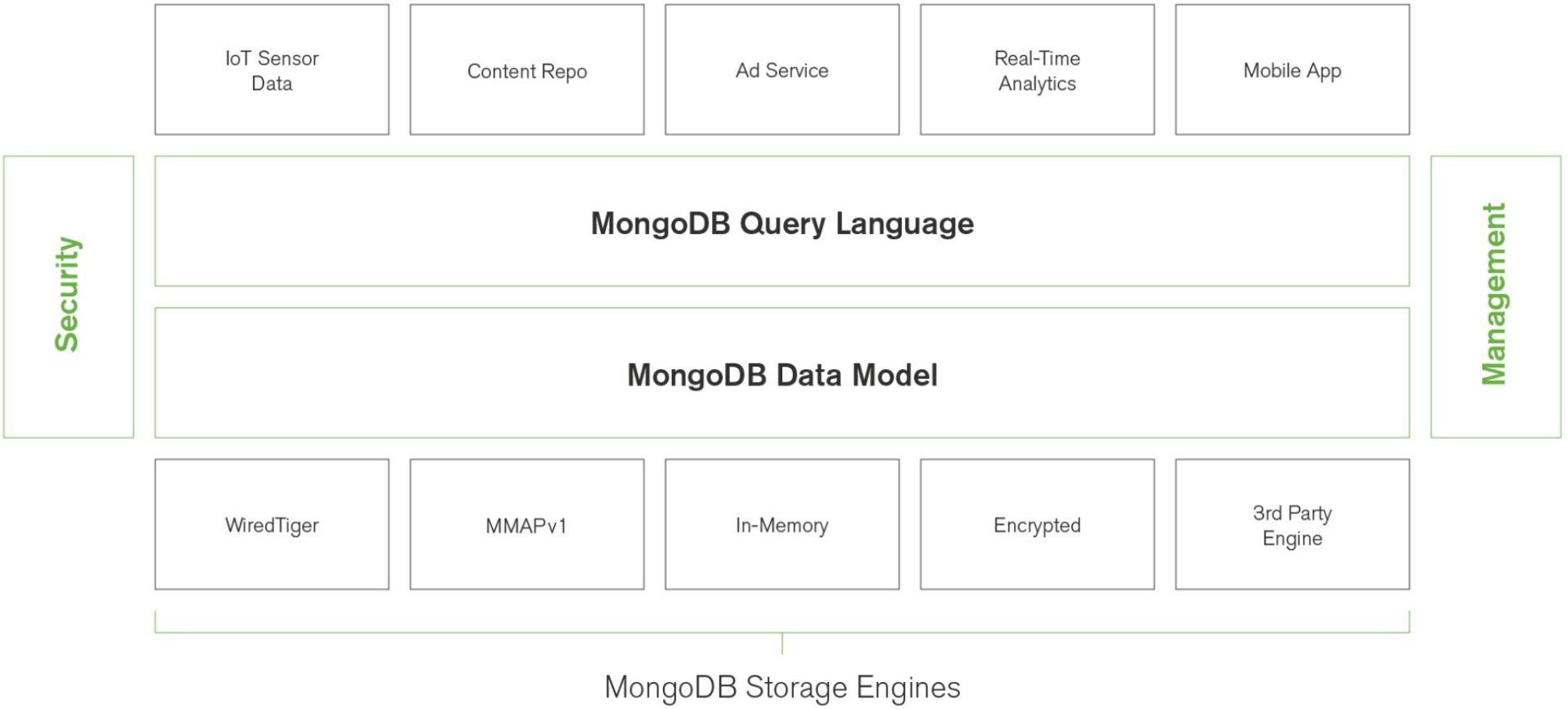
- **Eindeutig**
  - Duplikate ablehnen
- **Eingeschränkt**
  - nur bestimmte Dokumente
- **Sparse**
  - nur Dokumente, die Feld beinhalten
- **TTL**
  - automatisches Herausaltern von Dokumenten

Indizes: Beispiel

```
> db.users.explain().find({age: 23})
{ "queryPlanner" : { ...
  "winningPlan" : { "stage" : "COLLSCAN",
    "filter" : { "age" : { "$eq" : 23 }},
    "direction" : "forward"
  ...
  }
}

> db.users.createIndex({"age": 1})
> db.users.explain().find({age: 23})
{ "queryPlanner" : { ...
  "winningPlan" : { "stage" : "FETCH",
    "inputStage" : { "stage" : "IXSCAN",
      "keyPattern" : { "age" : 1 },
      "indexName" : "age_1",
      ...
      "indexBounds" : { "age" : [ "[23.0, 23.0]" ]}
  }
}
```

MongoDB: Architektur



Quelle: MongoDB Architecture Guide (MongoDB 3.4, 2016)

Storage Engines

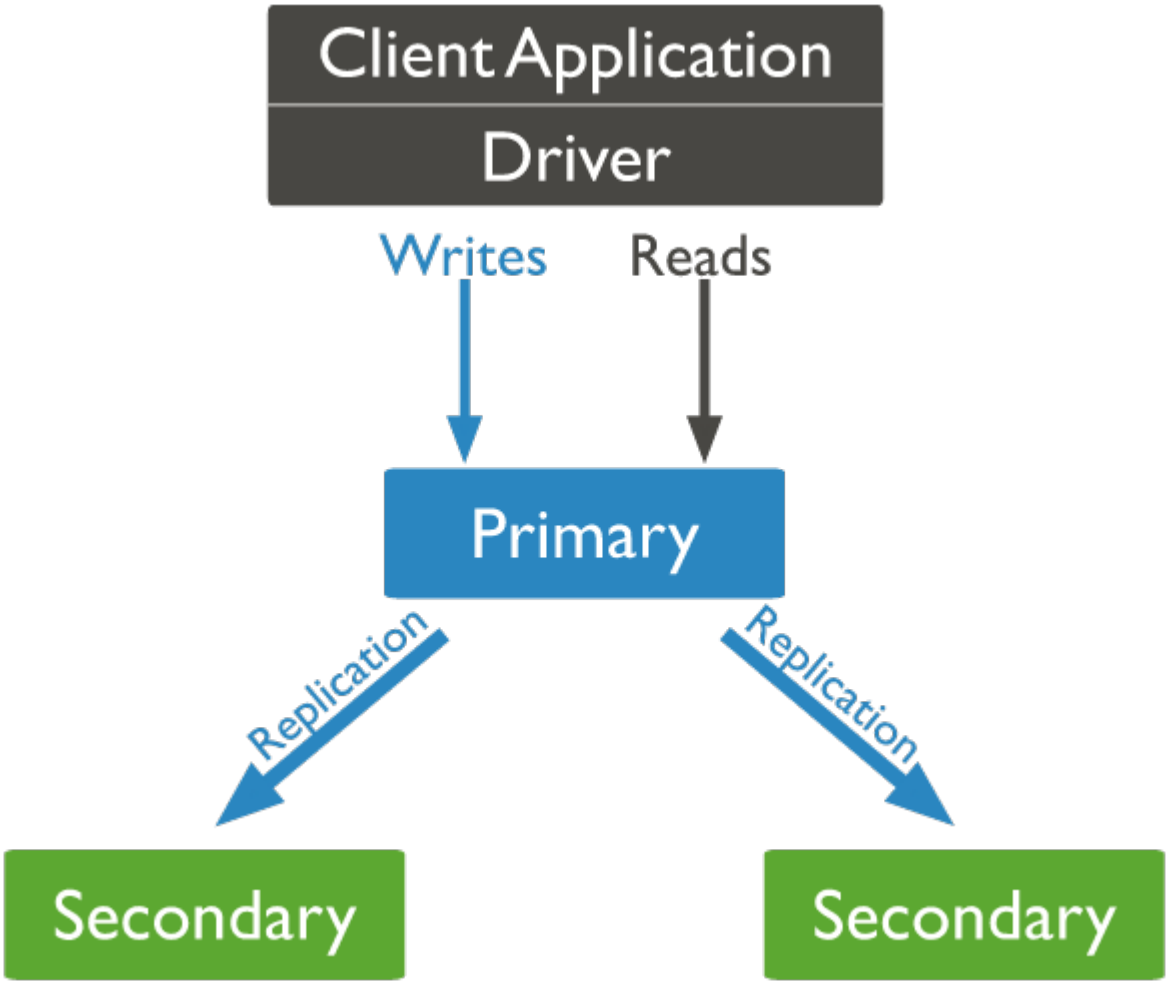
- **Storage Engine** bestimmt
  - physisches Datenmodell
  - nonfunktionale Eigenschaften
- Mischbetrieb möglich

- Standard
  - **MMAPv1**
  - **WiredTiger**
  - **InMemory**

Storage Engine: WiredTiger

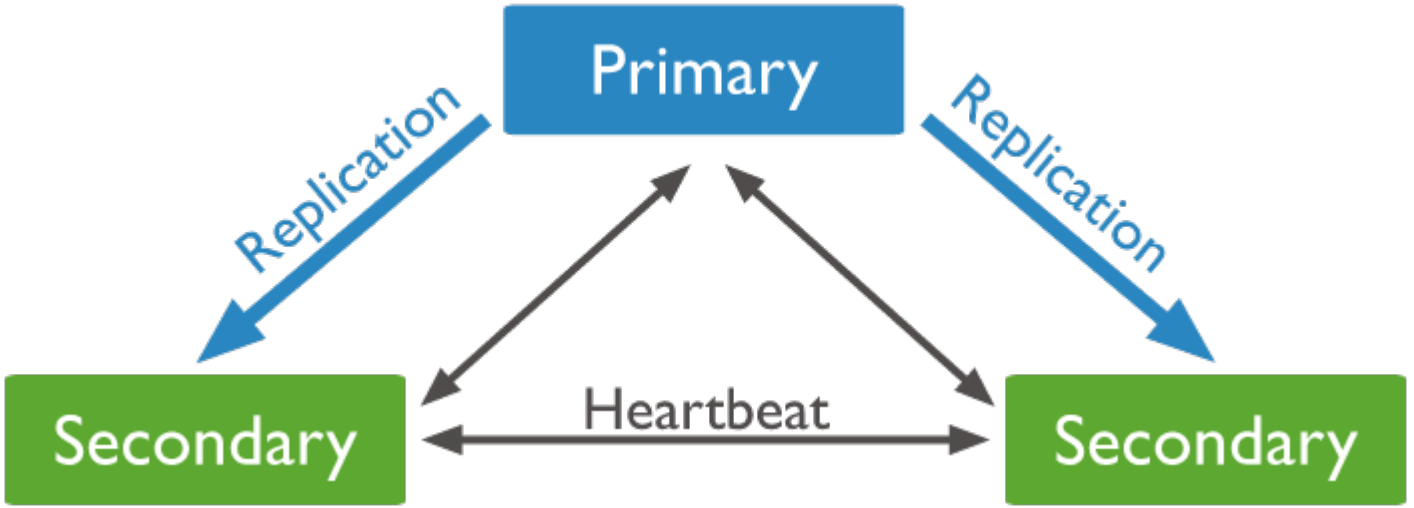
- **Transaktionen**
  - **Isolation** auf Dokumenten-Ebene
  - MVCC: **Multi-Version Concurrency Control**
    - Checkpoint bei Beginn der Transaktion
    - Durchführen der Operation
    - Atomares Umschalten auf neuen Stand
    - Fehler/Wiederholung bei Konflikten
  - Ergänzend: **Write-Ahead Log** (Journal)
    - Operationen zwischen Checkpoints

Replikation

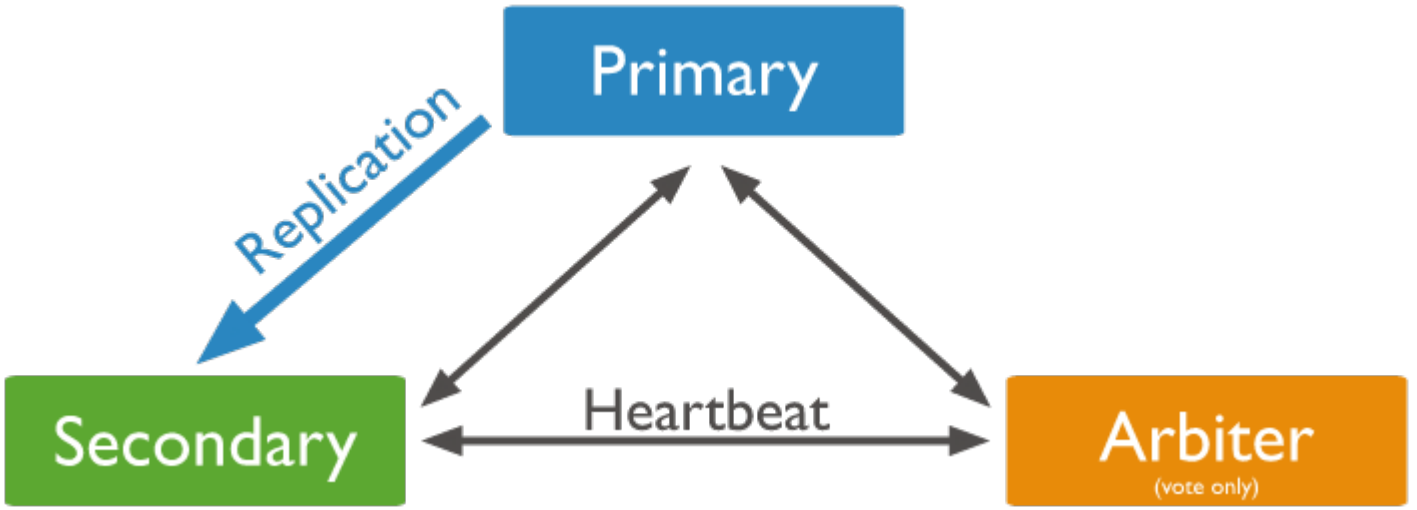


Quelle: docs.mongodb.com

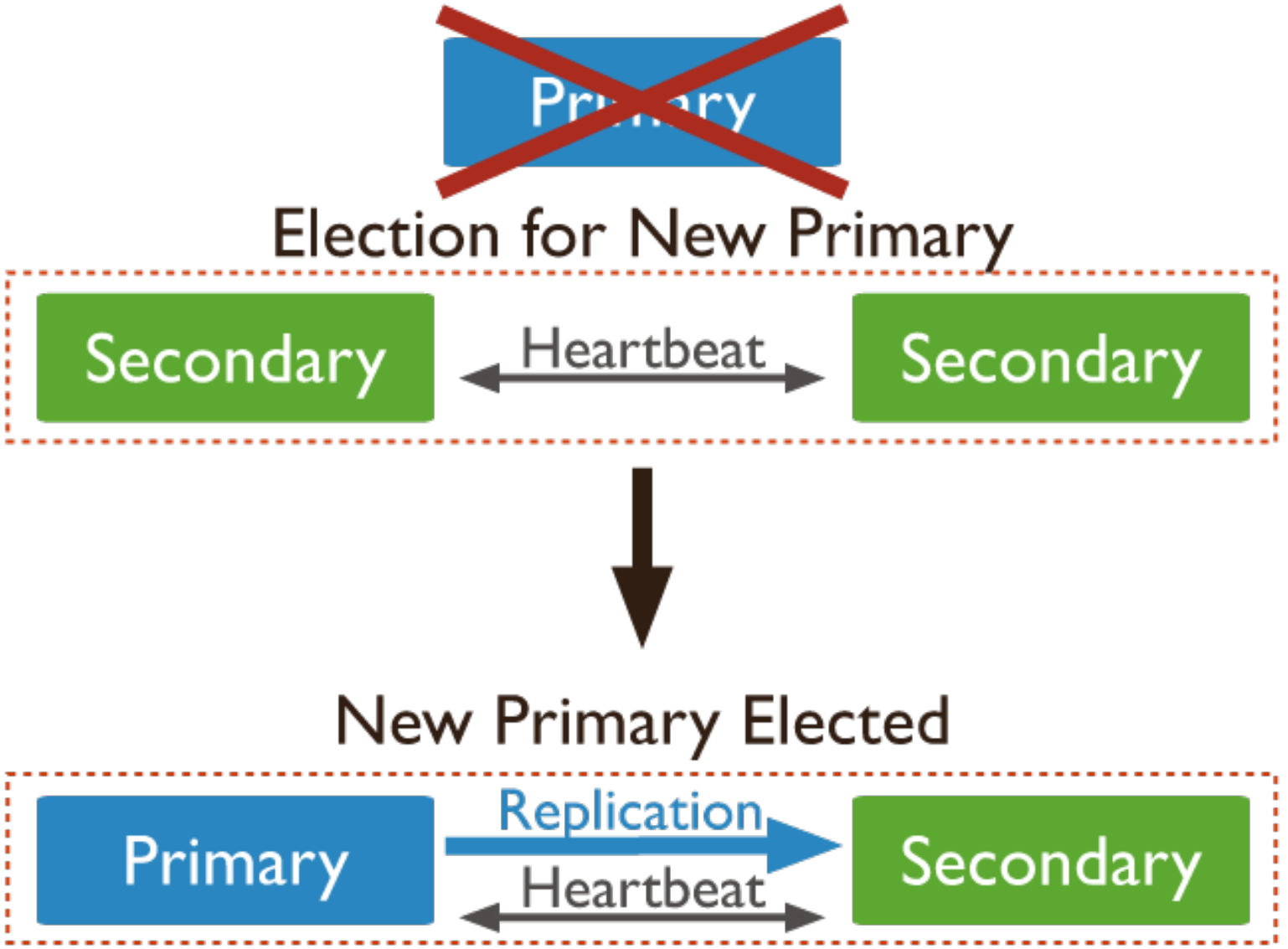
Replikation: Failover



Replikation: Failover



Replikation: Failover



Leseoperationen: "Read Preference"

- Woher soll gelesen werden?
  → Client-Einstellung

- primary/primaryPreferred
- secondary/secondaryPreferred
- nearest

Schreiboperationen: Tuneable Consistency

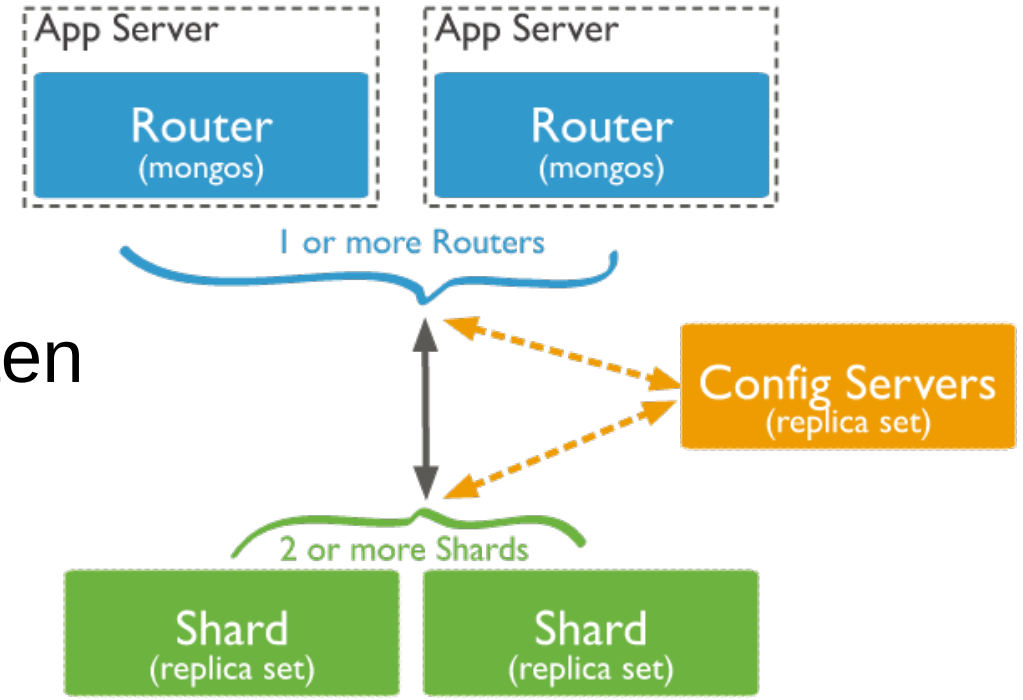
```
db.products.insert(
   { item: "envelopes", qty : 100, type: "Clasp" },
   { writeConcern: { w: 2, wtimeout: 5000 } }
)
```

Mindestens 2 Replikas

Timeout nach 5 Sekunden

Sharding

- **Shard**
  - Teilmenge der Daten
  - Möglicherweise ein Replica Set
- **mongos (Query Router)**
  - repräsentiert Cluster für Clients
- **Config Servers**
  - Replica Set mit Metadaten

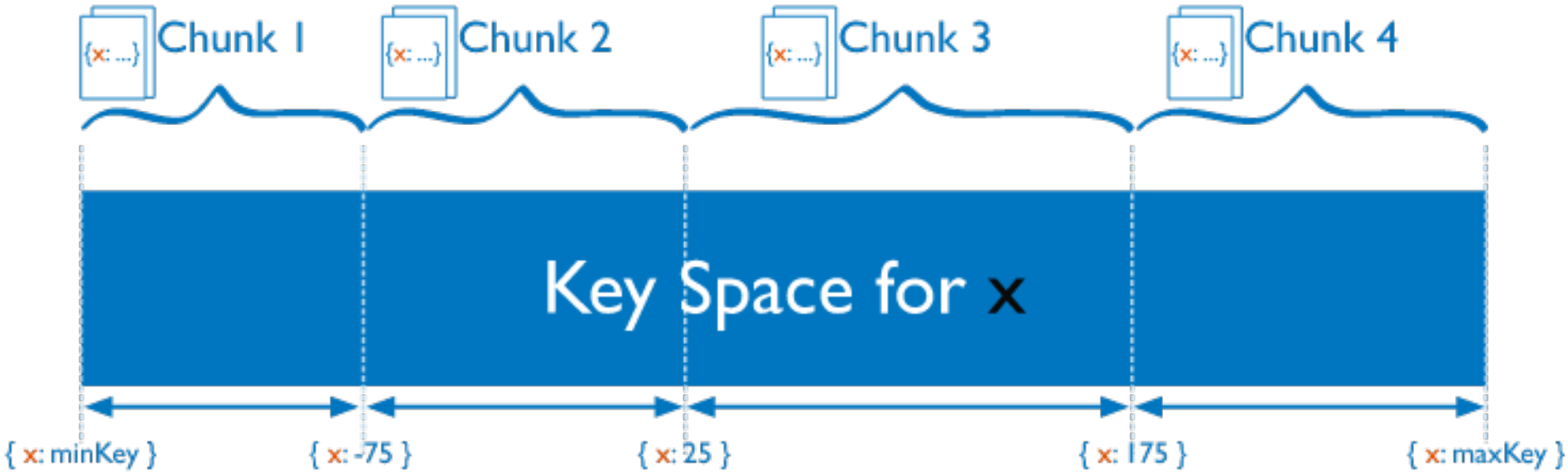


Sharding

- **Shard Key**
  - **Partitionierungskriterium** in den Dokumenten
  - **Unveränderlich**
  - Vorbedingung: **Index** auf Shard Key

```
sh.shardCollection("records.people", { zipcode: 1 } )
```

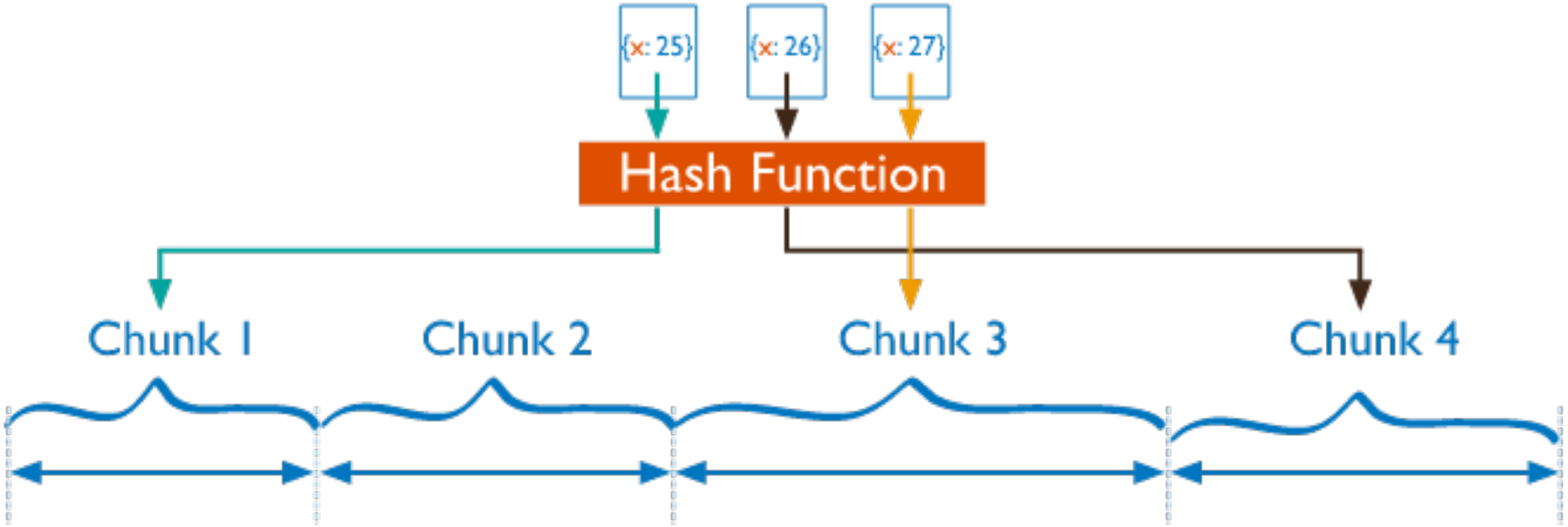
Shards und Chunks



Ranged Sharding

- Typische Chunk-Größe: 64 MB

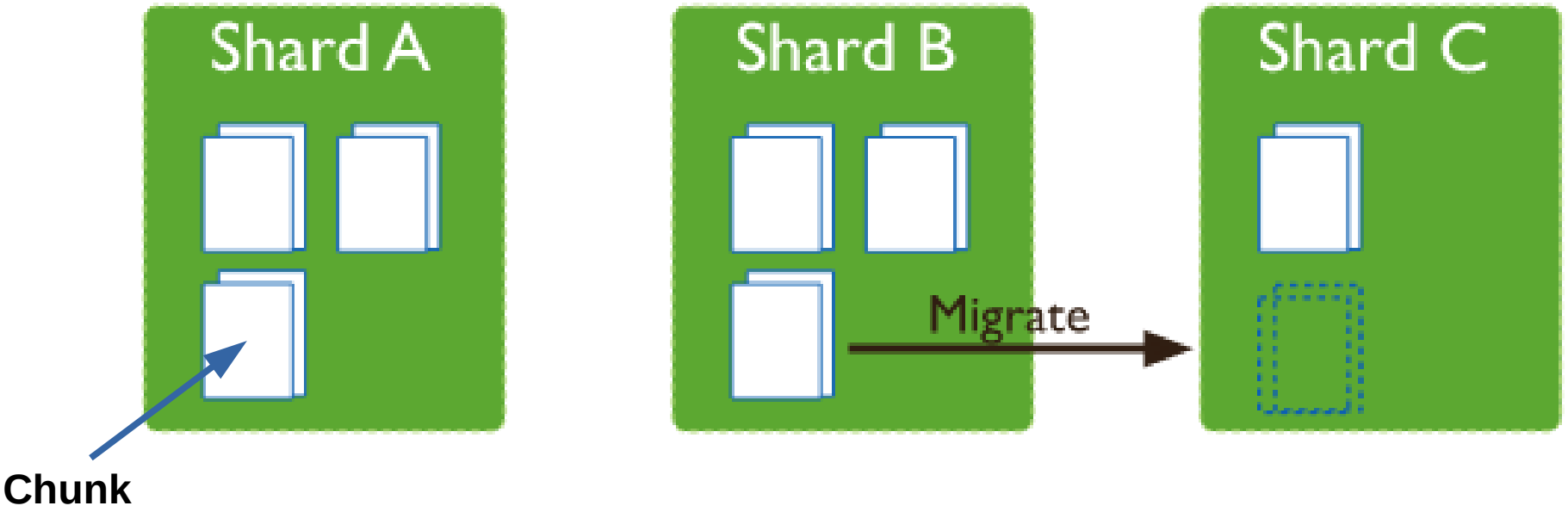
Shards und Chunks



Hashed Sharding

- **Pro:** Gleichmäßigere Verteilung
- **Contra:** Bereichsanfragen?!

Shards und Chunks



Chunk-Metadaten

```
{
   "_id" : "mydb.foo-a_\"cat\"",
   "lastmod" : Timestamp(1000, 3),
   "lastmodEpoch" : ObjectId("5078407bd58b175c5c225fdc"),
   "ns" : "mydb.foo",
   "min" : {
         "animal" : "cat"
   },
   "max" : {
         "animal" : "dog"
   },
   "shard" : "shard0004"
}
```

Untergrenze

Obergrenze

Shard-Zuordnung

→ Verwaltung in DB "config" auf den Config-Servern

Client-Programmierung (Python)

```
from pymongo import MongoClient

client = MongoClient("infbdt02.fh-trier.de")
db = client.bigdata200

# Kerberos!
db.authenticate("bigdata200@BDT.FH-TRIER.DE", mechanism = "GSSAPI")

for user in db.users.find({"name": "Eva Meier"}):
    print user


{   u'gender': u'w', u'age': 23.0, u'_id': ObjectId('59…d0c82'),
    u'name': u'Eva Meier',
    u'addresses': [{  u'city': u'Trier',
                      u'street': u'Simeonstra\xdfe 20'}]}
...
```

Client-Programmierung (Java)

- Eigentlich…

```
MongoClient client = new MongoClient(
            new ServerAddress(server, 27017));
MongoCollection<Document> collection =
            client.getDatabase("bigdata200").getCollection("users");
BasicDBObject query = new BasicDBObject("name", "Eva Meier");

MongoCursor<Document> cursor = collection.find(query).iterator();
try {
    while (cursor.hasNext()) {
        Document doc = cursor.next();
        System.out.println(doc.get("name") + ": " + doc.getInteger("age", -1));
    }
} finally {
    cursor.close();
}

client.close();
```

Client-Programmierung (Java)

- … doch dann…

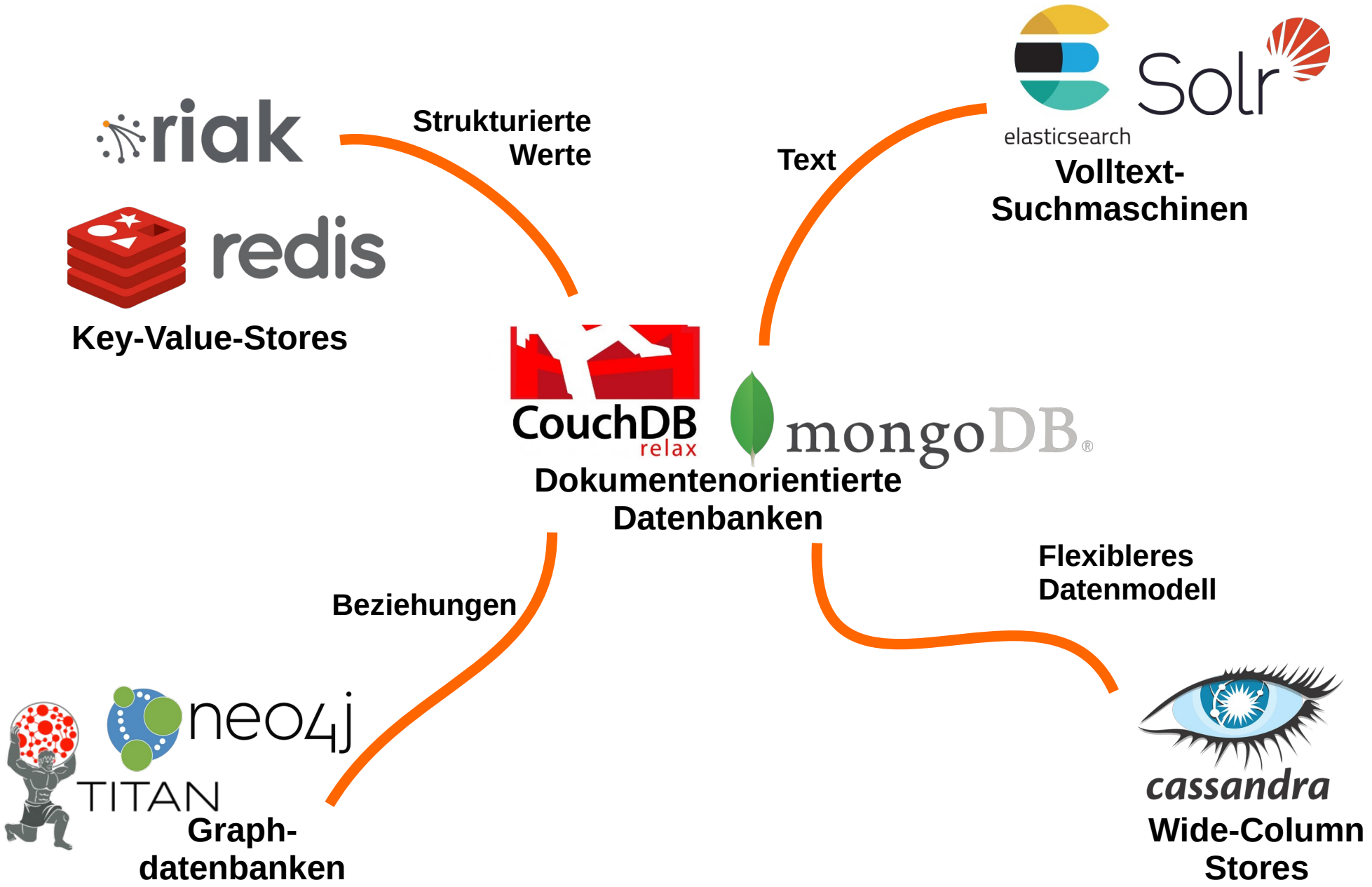
```
System.setProperty("javax.security.auth.useSubjectCredsOnly", "false");
System.setProperty("java.security.krb5.realm", "BDT.FH-TRIER.DE");
System.setProperty("java.security.krb5.kdc", "infbdt03.fh-trier.de");

String server = "infbdt02.fh-trier.de";
String user = "bigdata200@BDT.FH-TRIER.DE";

MongoCredential credential = MongoCredential.createGSSAPICredential(user);
MongoClient client = new MongoClient(
        new ServerAddress(server, 27017), Arrays.asList(credential));

...
```

Mitbewerber



Fazit: Dokumentenorientierte Datenbanken

- **"Dokumente"** = JSON, XML, …
- Beliebige **Struktur**
- **Schema** optional
- Reichhaltige **Anfrageoperationen**

- Effizient über **Schlüssel** zugreifbar…
- … oder über **Indizes**

- **Verteilungsmechanismen** analog zu anderen NoSQL-Technologien