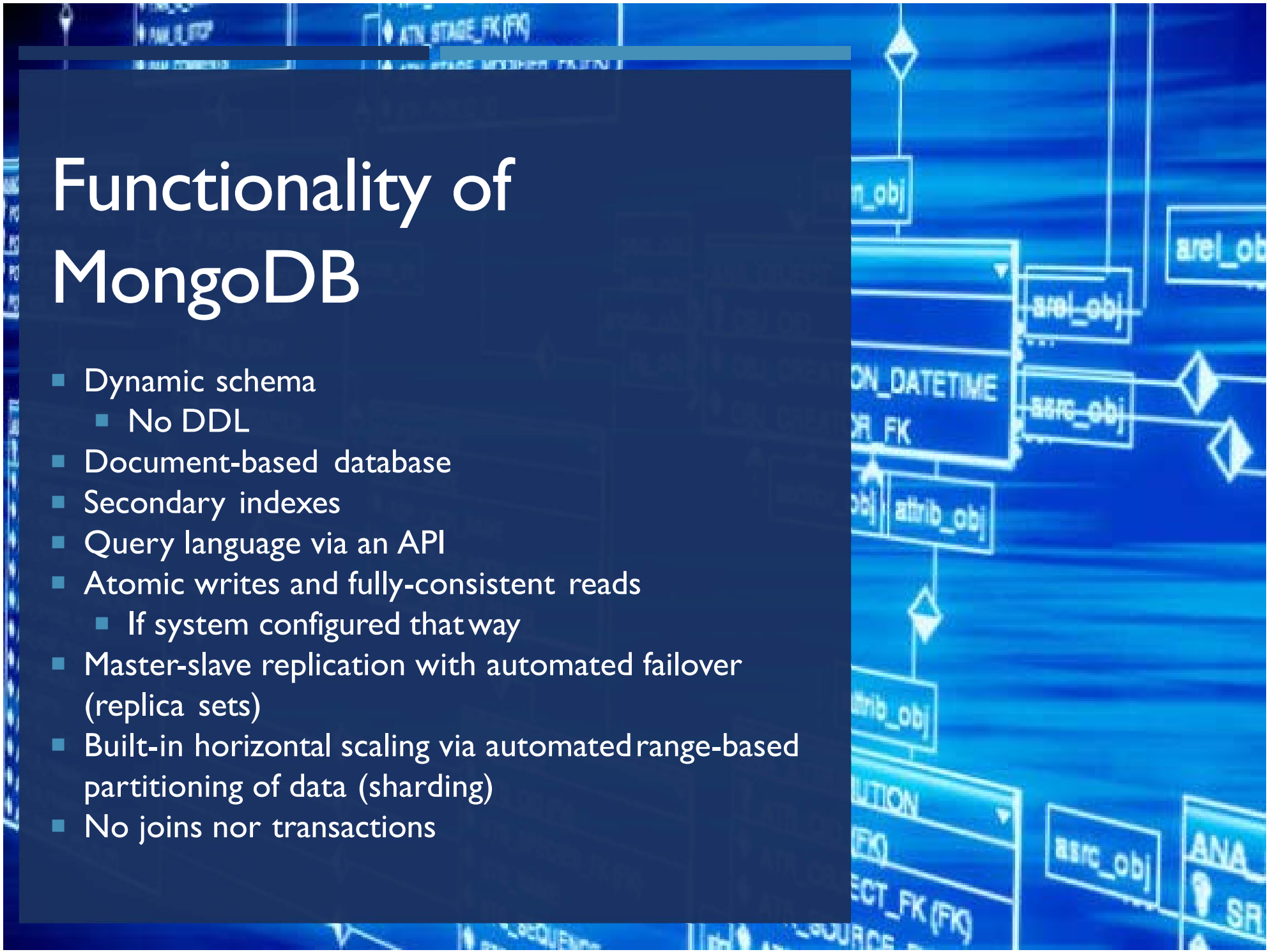
MongoDB

MSBA7024 / MACC7020 Database Design and Management

What is MongoDB?

- Developed by 10gen
  - Founded in 2007
- Document-oriented, NoSQL database
  - Hash-based, *schema-less database*
    - No Data Definition Language
    - In practice, this means you can store hashes with any keys and values that you choose
      - Keys are a basic data type but in reality stored as strings
      - Document Identifiers (_id) will be created for each document, field name reserved by system
    - Application tracks the schema and mapping
    - Uses BSON format
- Supports APIs (drivers) in many computer languages
  - JavaScript, Python, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang



Functionality of MongoDB

- Dynamic schema
  - No DDL
- Document-based database
- Secondary indexes
- Query language via an API
- Atomic writes and fully-consistent reads
  - If system configured that way
- Master-slave replication with automated failover (replica sets)
- Built-in horizontal scaling via automated range-based partitioning of data (sharding)
- No joins nor transactions

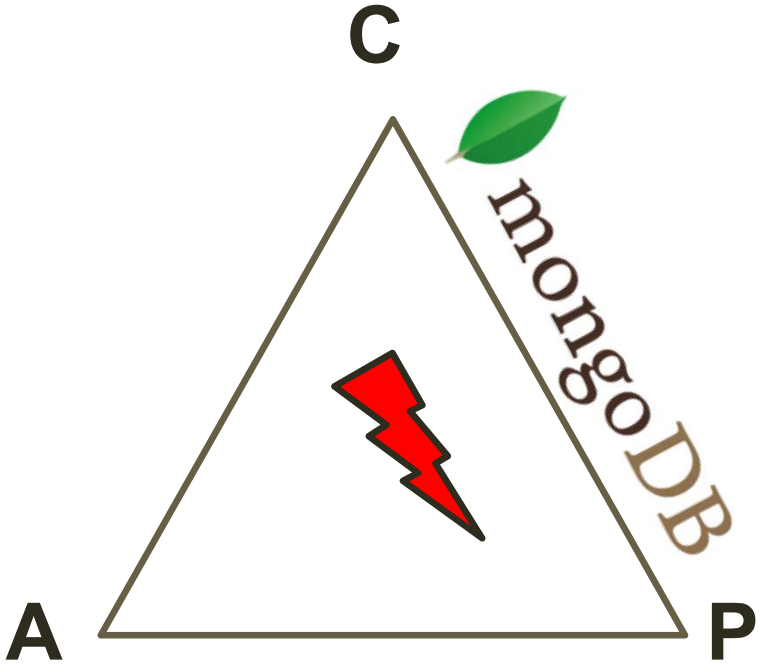
Choices Made for Design of MongoDB

- Scale horizontally over commodity hardware
  - Lots of relatively inexpensive servers
- Keep the functionality that works well in RDBMSs
  - Ad hoc queries
  - Fully featured indexes
  - Secondary indexes
- What **doesn't** distribute well in RDBMS?
  - Long running multi-row transactions
  - Joins

MongoDB: CAP approach

Focus on **Consistency** and **Partition tolerance**

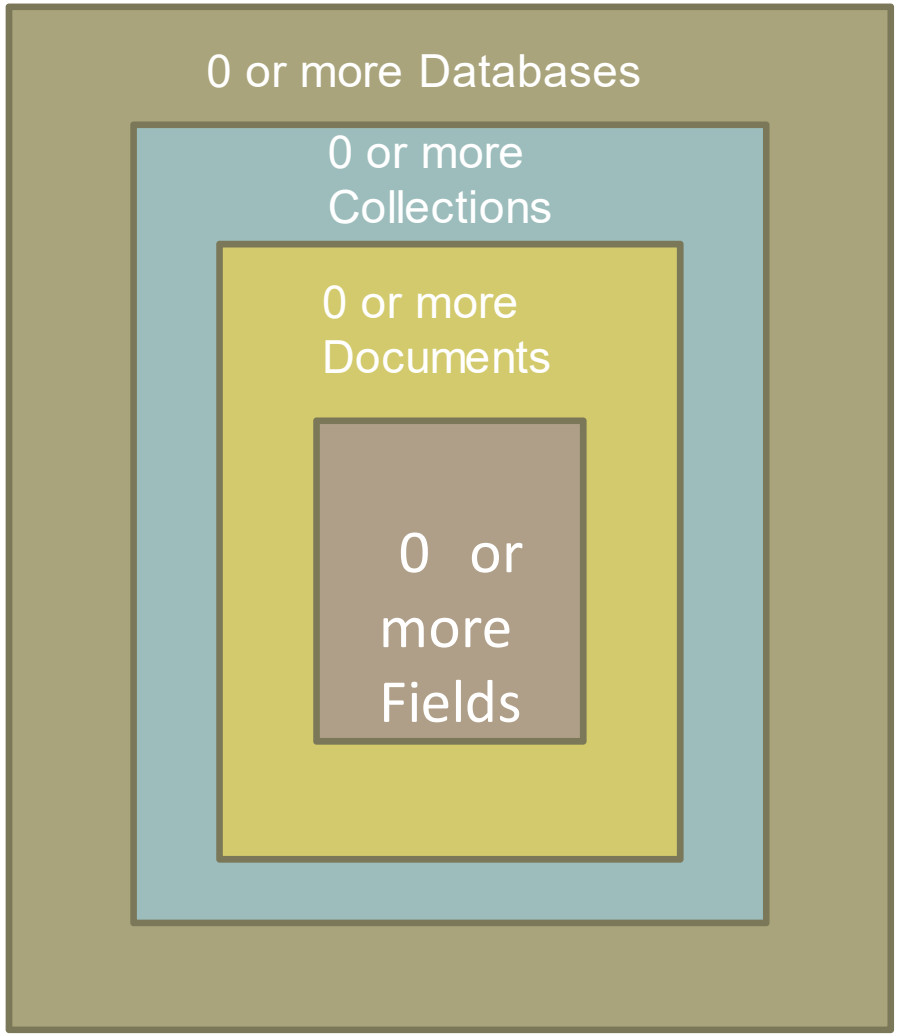
- **C**onsistency
  - all replicas contain the same version of the data
- **A**vailability
  - system remains operational on failing nodes
- **P**artition tolerance
  - multiple entry points
  - system remains operational on system split



CAP Theorem:
it is **impossible** to satisfy all three at the same time

MongoDB: Hierarchical Objects

- A MongoDB instance may have zero or more 'databases'
- A database may have zero or more 'collections'
- A collection may have zero or more 'documents'
- A document may have one or more 'fields'
- MongoDB 'Indexes' function much like their RDBMS counterparts



RDBMS Concepts to NoSQL

RDBMS		MongoDB
Database		Database
Table, View		Collection
Row		Document (BSON)
Column	➡	Field
Index		Index
Join		Embedded Document
Foreign Key		Reference
Partition		Shard

Collection is not strict about what it stores

Schema-less

Hierarchy is evident in the design

Embedded Document

MongoDB Processes and Configuration

- mongod – Database instance
- mongos – Sharding processes
  - Analogous to a database router
  - Processes all requests
  - Decides how many and which *mongod*s should receive the query
  - *mongos* collates the results, and sends it back to the client

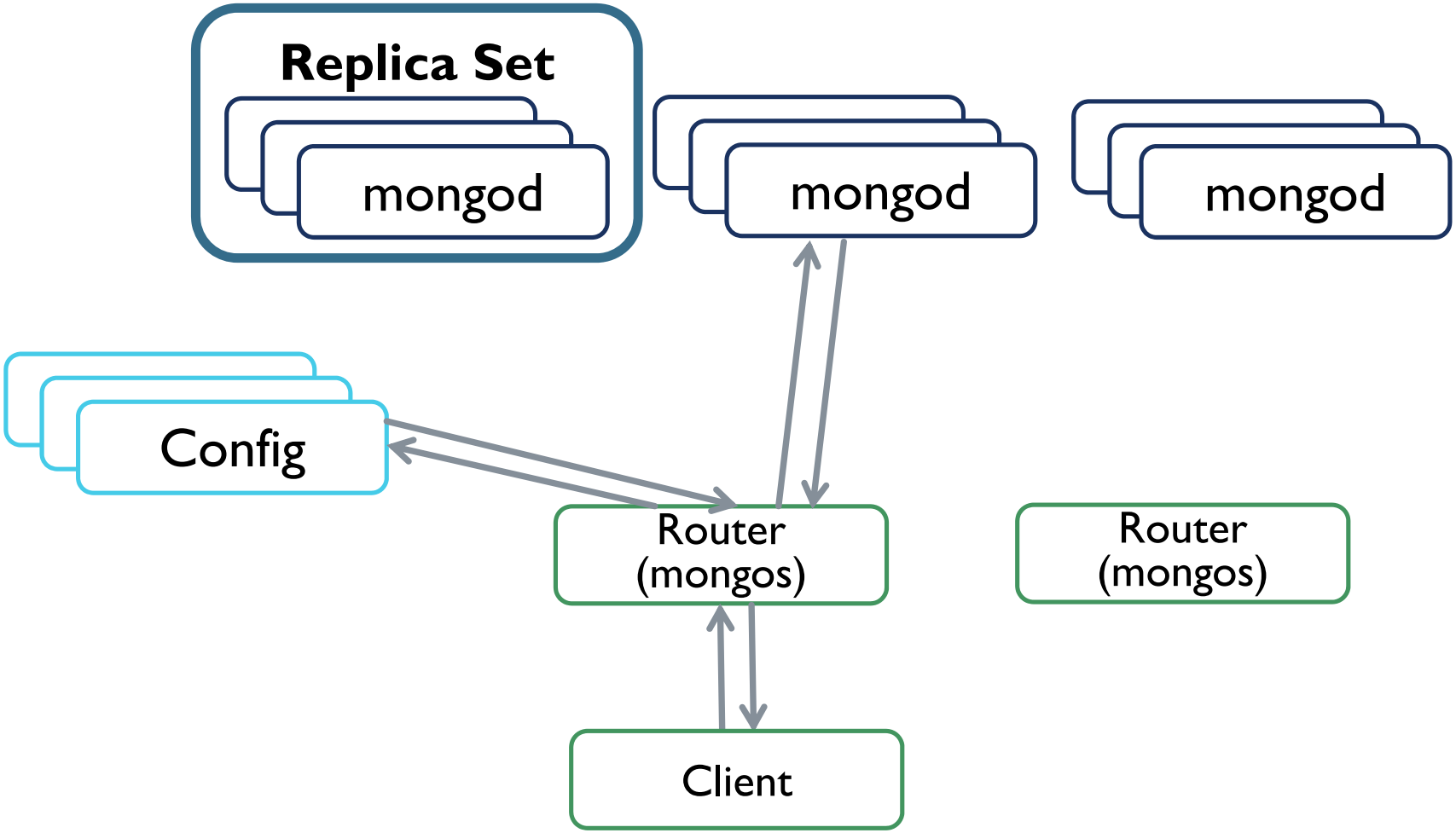
MongoDB Processes and Configuration

- Data split into chunks, based on shard key (~ primary key)
- Either use hash or range-partitioning
- Shard: collection of chunks
- Shard assigned to a replica set

MongoDB Processes and Configuration

- Replica set consists of multiple mongod servers (typically 3 mongods)
- Replica set members are mirrors of each other
  - One is primary
  - Others are secondary
- You can have one *mongos* for the whole system no matter how many mongods you have
- Or you can have one local *mongos* for every client if you want to minimize network latency.

MongoDB Processes and Configuration

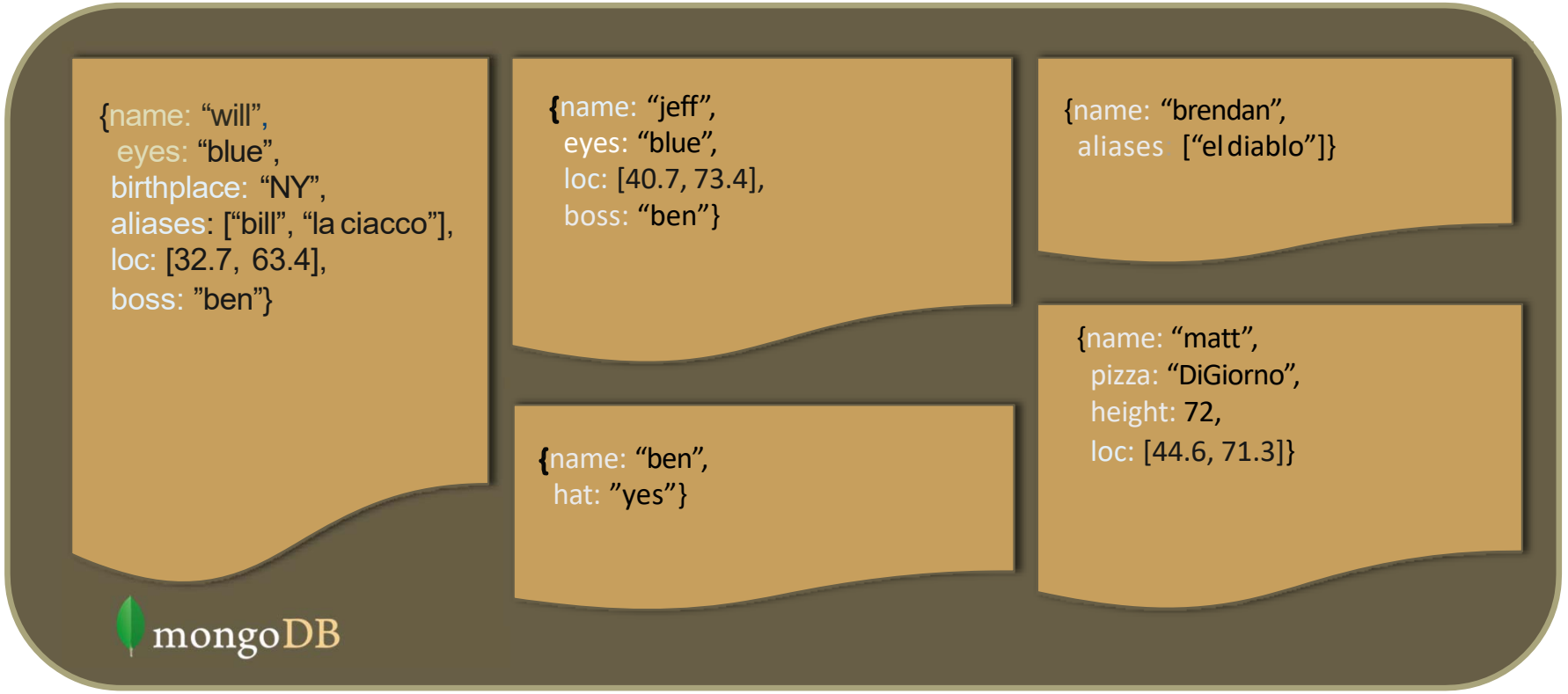


MongoDB Processes and Configuration

- MongoCompass
  - A graphical client to interact with the MongoDB server
- Mongo – an interactive shell (also aclient)
  - Fully functional JavaScript environment for use with a MongoDB

Schema Free

- MongoDB does not need any pre-defined data schema
- Every document in a collection could have different data
  - Addresses NULL data fields



BSON Format

- Binary-encoded serialization of JSON-like documents
- Zero or more key/value pairs are stored as a single entity
- Each entry consists of a field name, a data type, and a value
- Large elements in a BSON document are prefixed with a length field to facilitate scanning

JSON Format

- Data is in name / value pairs
- A name/value pair consists of a field name followed by a colon, followed by a value:
  - Example: “name”: “R2-D2”
- Data is separated by commas
  - Example: “name”: “R2-D2”, race : “Droid”
- Curly braces hold objects
  - Example: {“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”}
- An array is stored in brackets []
  - Example: [{“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”}, {“name”: “Yoda”, affiliation: “rebels”}]

JSON Format

- Example of a document

```
{
        name: "travis",
        salary: 30000,
        designation: "Computer Scientist",
        teams: [ "front-end",  "database" ]
}
```

Index Functionality

- B+ tree indexes
- An index is automatically created on the _id field (the primary key)
- Users can create other indexes to improve query performance or to enforce Unique values for a particular field
- Supports single field index as well as Compound index
  - Like SQL order of the fields in a compound index matters
  - If you index a field that holds an array value, MongoDB creates separate index entries for *every* element of the array
- Sparse property of an index ensures that the index only contain entries for documents that have the indexed field (so ignore records that do not have the field defined)
- If an index is both unique and sparse – then the system will reject records that have a duplicate key value but allow records that do not have the indexed field defined

CRUD Operations

- Create
  - db.collection_name.insert(<document>)
  - db.collection_name.save(<document>)
- Read
  - db.collection_name.find(<query>, <projection>)
  - db.collection_name.findOne(<query>, <projection>)
- Update
  - db.collection_name.update(<query>, <update>, <options>)
- Delete
  - db.collection_name.deleteOne(<query>)
  - db.collection_name.deleteMany(<query>)

db.collection **specifies** the collection or the 'table' for the operation.

Create Operations

db.collection specifies the collection or the 'table' to store the document

- db.collection_name.insert(<document>)
  - Omit the _id field to have MongoDB generate a unique key
  - Example
    - db.**parts**.insert({type: "screwdriver", quantity: 15 })
    - db.**parts**.insert({_id: 10, type: "hammer", quantity: 1 })
- db.collection_name.update(<query>, <update>, { upsert: true })
  - Will update 1 or more records in a collection satisfying query
- db.collection_name.save(<document>)
  - Updates an existing record or creates a new record

Create Operations

Insert a row entry for new employee Sally

```
db.employee.insert({
                        name :  "sally",
                        salary :  15000,
                        designation :  "MTS",
                        teams :  [ "cluster-management" ]
                        })
```

Read Operations

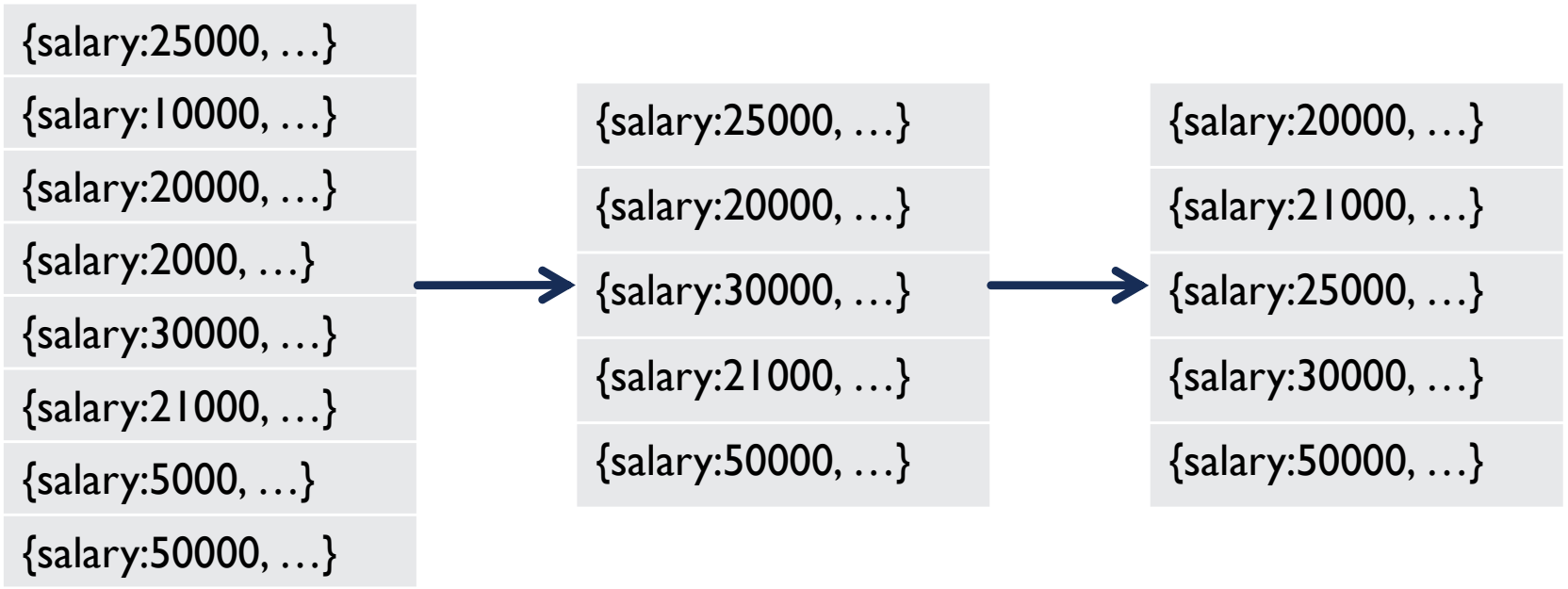
- db.collection_name.find(<query>, <projection>)
  - Provides functionality similar to the SELECT command
    - <query> where condition , <projection> fields in result set
  - Example
    - db.parts.find({type:“hammer”}).limit(5)
  - Can use a cursor to handle a result set
    - var PartsCursor = db.parts.find({type:“hammer”}).limit(5)
  - Can modify the query to impose limits, skips, and sort orders.
  - Can specify to return the ‘top’ number of records from the result set
- db.collection_name.findOne(<query>, <projection>)

Read Operations

Query all employee names with salary greater than 18000 sorted in ascending order

```
db.employee.find({salary:{$gt:18000}, {name:1}}).sort({salary:1})
```

Collection — Condition — Projection — Modifier



Query Operators

Name	Description
$eq	Matches value that are equal to a specified value
$gt, $gte	Matches values that are greater than (or equal to a specified value
$lt, $lte	Matches values less than or (equal to) a specified value
$ne	Matches values that are not equal to a specified value
$in	Matches any of the values specified in an array
$nin	Matches none of the values specified in an array
$or	Joins query clauses with a logical OR returns all
$and	Join query clauses with a loginalAND
$not	Inverts the effect of a query expression
$nor	Join query clauses with a logical NOR
$exists	Matches documents that have a specified field

Query Operators

- db.employee.find()
- db.employee.find({name: "Sally"})
- var cursor = db.employee.find({salary: {$in: [5000, 2000] } })
  - Use next() to access the rest of the records

Query Operators

- db.employee.find({name: "Steve", salary: {$lt: 3000} })
- db.employee.find({ $or: [{ name: "Bill" }, { salary: { $gt: 9000 } }] })

- Find records of all managers who earn more than 5000

  db.employee.find({designation:"Manager", salary: {$gt: 5000}})

Update Operations

- db.collection_name.update(<query>, <update>, <options>)
  - Will update 1 or more records in a collection satisfying query
- db.collection_name.findAndModify(<query>, <sort>, <update>, <new>, <fields>,<upsert>)
  - Modify existing record(s) – retrieve old or new version of the record

Update Operations

All employees with salary greater than 18000 get a designation of Manager

Update Criteria
Update Action
Update Option

```
db.employee.update(
        {salary:{$gt:18000}},
        {$set: {designation: "Manager"}},
        {multi: true}
)
```

Multi-option allows multiple document update.

Update Operations

- Increment salary of all managers by 1000
  - db.employee.update({ designation : "Manager" }, { $inc : { salary : 1000 } }) -> update one only
  - db.employee.update({ designation : "Manager" }, { $inc : { salary : 1000 } } , { multi: **true** })
- Increment salary of all managers working in cluster-management group by 5000
  - db.employee.update({ designation : "Manager", teams: "cluster-management"}, { $inc : { salary : 5000 } } , { multi: **true** })

Delete Operations

- db.collection_name.deleteOne(<query>)
  - Delete the first record from a collection or matching a criterion
  - db.parts.deleteOne({ type: /^h/ }) – remove all parts starting with h
- db.collection_name.deleteMany(<query>)
  - Delete all records from a collection or matching a criterion

Delete Operations

Remove all employees who earn less than 10000

Remove Criteria

```
db.employee.deleteMany(
    {salary:{$lt:10000}},
)
```

Aggregated Functionality

Aggregation framework provides SQL-like aggregation functionality

- Pipeline documents from a collection pass through an aggregation pipeline, which transforms these objects as they pass through
- Expressions produce output documents based on calculations performed on input documents
- Example: db.**parts**.aggregate ({$group : {_id: "$type", totalquantity : { $sum: "$quantity"} } })

Document Structures

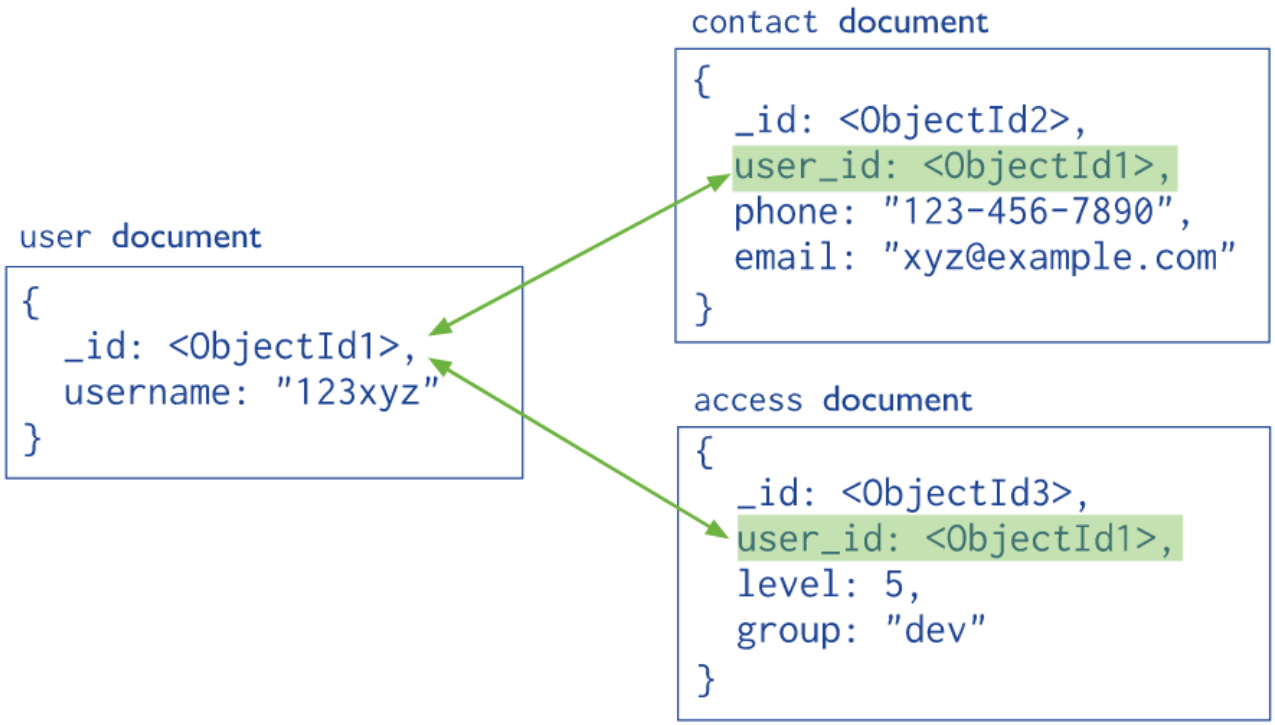
- Two main document structures in MongoDB
  - Embedded Data
  - References

Document Structures

- Embedded Data: capture relationships between data by storing related data in a single document structure.

Document Structures

- References: store the relationships between data by including links or references from one document to another.
- Normalized data models



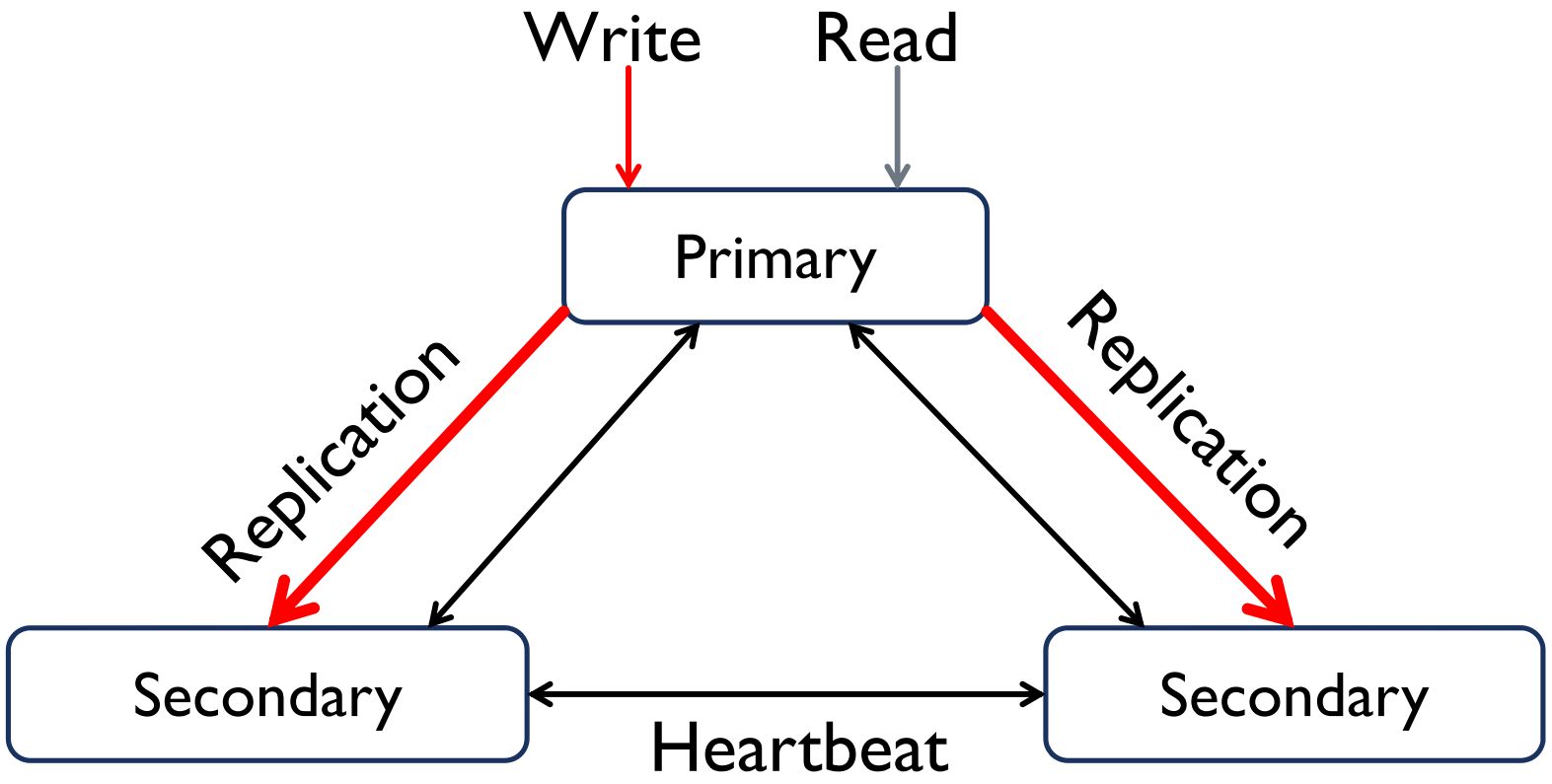
Document Structures

- Embedded data
  - Fast retrieval – retrieve all related data in one read operation
  - Normally used for one-to-one and one-to-many relationships between entities
  - May have duplicated data
- References
  - Slow retrieval – need multiple read operations
  - Can model more complex many-to-many relationship or large hierarchical data

Document Structures

- Embedded document pattern can lead to large documents
  - e.g., a product with several thousand reviews
- Subset Pattern
  - Instead of storing all of the reviews with the product, we split the collection into two collections:
    - The product collection stores information on each product, including the product's ten most recent reviews
    - The review collection stores all reviews. Each review contains a reference to the product for which it was written.
  - Fast to see the product page with top 10 reviews
  - If a user wants to see additional reviews, the application makes a call to the review collection.
- Need to consider:
  - Data duplication
  - Make sure the subset is the desired/correct subset

Replication of Data



(the nodes send pings to each other every two seconds to identify the current stats)

Replication of Data

- Ensures redundancy, backup, and automatic failover
- Replication occurs through groups of servers known as replica sets
  - Primary set – set of servers that client tasks directly updates to
  - Secondary set – set of servers used for duplication of data
  - At most can have 12 replica sets
    - Many different properties can be associated with a secondary set i.e. secondary-only, hidden delayed, arbiters, non-voting
  - If the primary set fails, the secondary sets ‘vote’ to elect the new primary set

Read Preference

- Determine where to route read operation
- Default is primary. Some other options are
  - primary-preferred
  - secondary
  - nearest
- Helps reduce latency, improve throughput
- Reads from secondary may fetch stale data

Write Concern

- Determines the guarantee that MongoDB provides on the success of a write operation
- Default is *acknowledged* (primary returns answer immediately).
  - Other options are
    - journaled (typically at primary)
    - replica-acknowledged (quorum with a value of W), etc.
- Weaker write concern implies faster write time

Consistency of Data

- All read operations issued to the primary of a replica set are consistent with the last write operation
    - Reads to a primary have **strict consistency**
        - Reads reflect the latest changes to the data
    - Reads to a secondary have **eventual consistency**
        - Updates propagate gradually
    - If clients permit reads from secondary sets – then client may read a previous state of the database
    - Failure occurs before the secondary nodes are updated
        - System identifies when a rollback needs to occur
        - Users are responsible for manually applying rollback changes