TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

**Macroarea di Ingegneria**
**Dipartimento di Ingegneria Civile e Ingegneria Informatica**

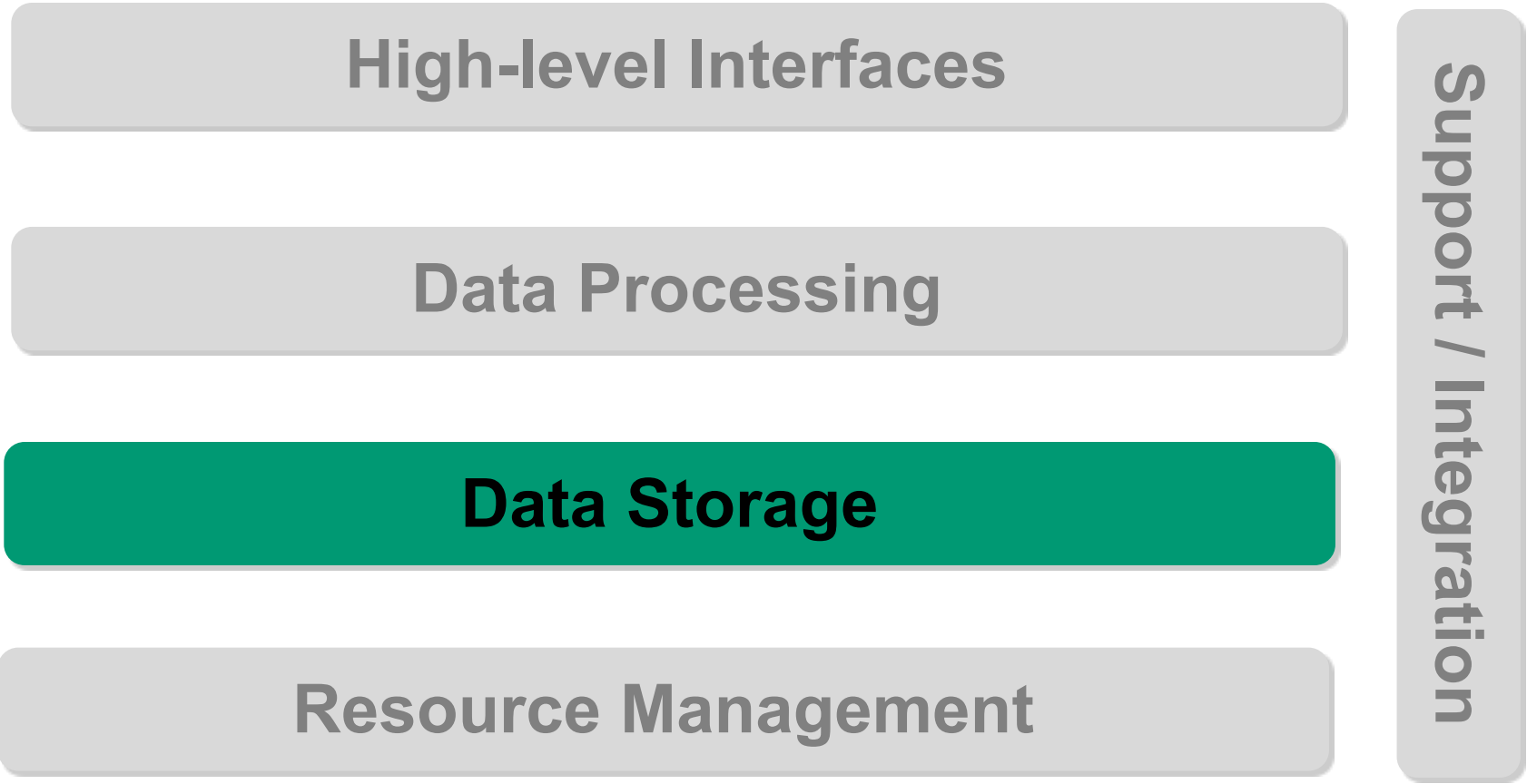
# NoSQL: MongoDB

A.A. 2023/24

Matteo Nardelli

Laurea Magistrale in
Ingegneria Informatica - II anno

# The reference Big Data stack



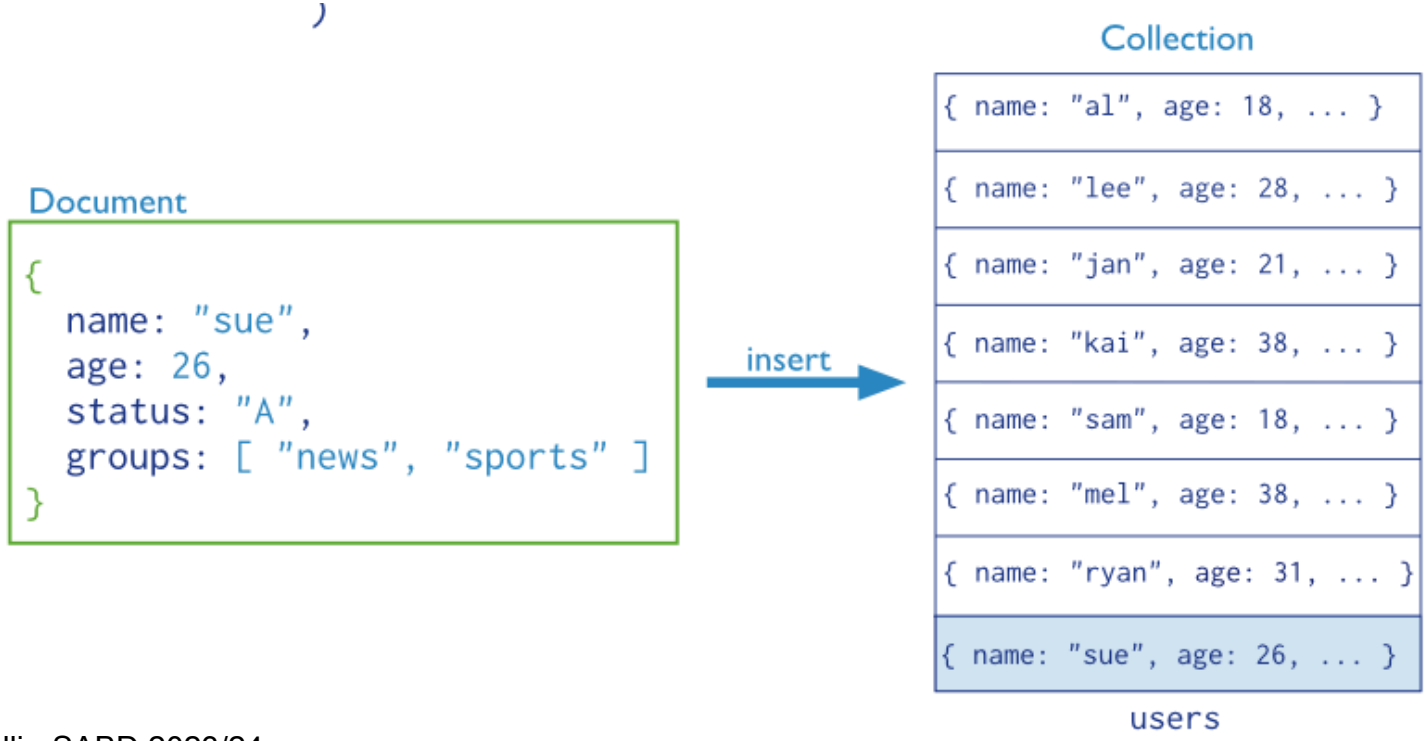
# Document data model

Document store: derived from the key-value data model

- Data model:
  - A set of <key,document> pairs
  - Document: an aggregate instance
- A document:
  - can contain complex data structures (nested objects)
  - does not require adherence to a fixed schema
- Access to the aggregate (document):
  - Structure of the aggregate visible
    - Often there are limitations on its content type
  - Queries based on the fields in the aggregate

In MongoDB:

- documents are grouped together into collections;
- inside each collection, a document should have a unique key;
- Documents can have different schema.



# MongoDB

| RDMS (e.g., mysql) | MongoDB |
|---|---|
| Tables | Collections |
| Records/Rows | Documents |
| Queries return record(s) | Queries return a cursor |

Document

```
{
  name: "sue",          <-- field: value
  age: 26,              <-- field: value
  status: "A",          <-- field: value
  groups: [ "news", "sports" ]   <-- field: value
}
```

# MongoDB

MongoDB represents JSON documents using BSON, a binary-encoded format that extends the JSON model to provide additional data types.

## Data Types

- String: combination of characters
- Boolean: True or False
- Integer: digits
- Double: a type of floating point number
- Null: not zero, not empty
- Array: a list of values
- Object: an entity which can be used in programming (value, variable, function, or data structure).
- Timestamp: a 64 bit value referring to a time
- Internationalized Strings: UTF-8 for strings
- Object IDs: every document must have an Object ID which is unique

# An example of document structure

```
{
 _id: ObjectId("5099803df3f4948bd2f98391"),
 name: { first: "Alan", last: "Turing" },
 birth: new Date('Jun 23, 1912'),
 death: new Date('Jun 07, 1954'),
 contribs: [ "Turing machine", "Turing test", "Turingery" ],
 views : NumberLong(1250000)
}
```

The above fields have the following data types:

- _id holds an ObjectId.
- name holds an embedded document that contains first and last.
- birth and death hold values of the Date type.
- contribs holds an array of strings.
- views holds a value of the NumberLong type.

# Dot notation

MongoDB uses the dot notation to access:

- the elements of an array: by concatenating the array name with the dot (.) and zero-based index position (in quotes)

```
{ ...
  contribs: [ "Turing machine", "Turing test", ... ],
... }
```

e.g., to specify the 3rd element: "contribs.2"

- the fields of an embedded document: by concatenating the embedded document name with the dot (.) and the field name

```
{ ...
  name: { first: "Alan", last: "Turing" },
... }
```

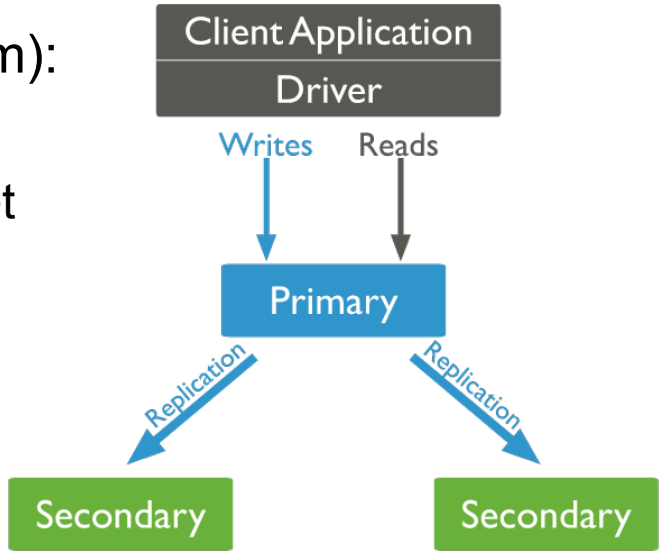
e.g., to specify the last name: "name.last"

https://docs.mongodb.com/manual/core/document/#dot-notation

# MongoDB: Consistency and Availability

Consistency (Mongo is considered a CP system):

- Monotonic Writes:
  - For standalone mongo instance and replica set
- Causal Consistency:
  - Across replica sets;
  - Configurable read/write quorum




Atomicity:

- Single document atomicity
- Multi-document Transactions with snapshot isolation

Replication and Scaling:

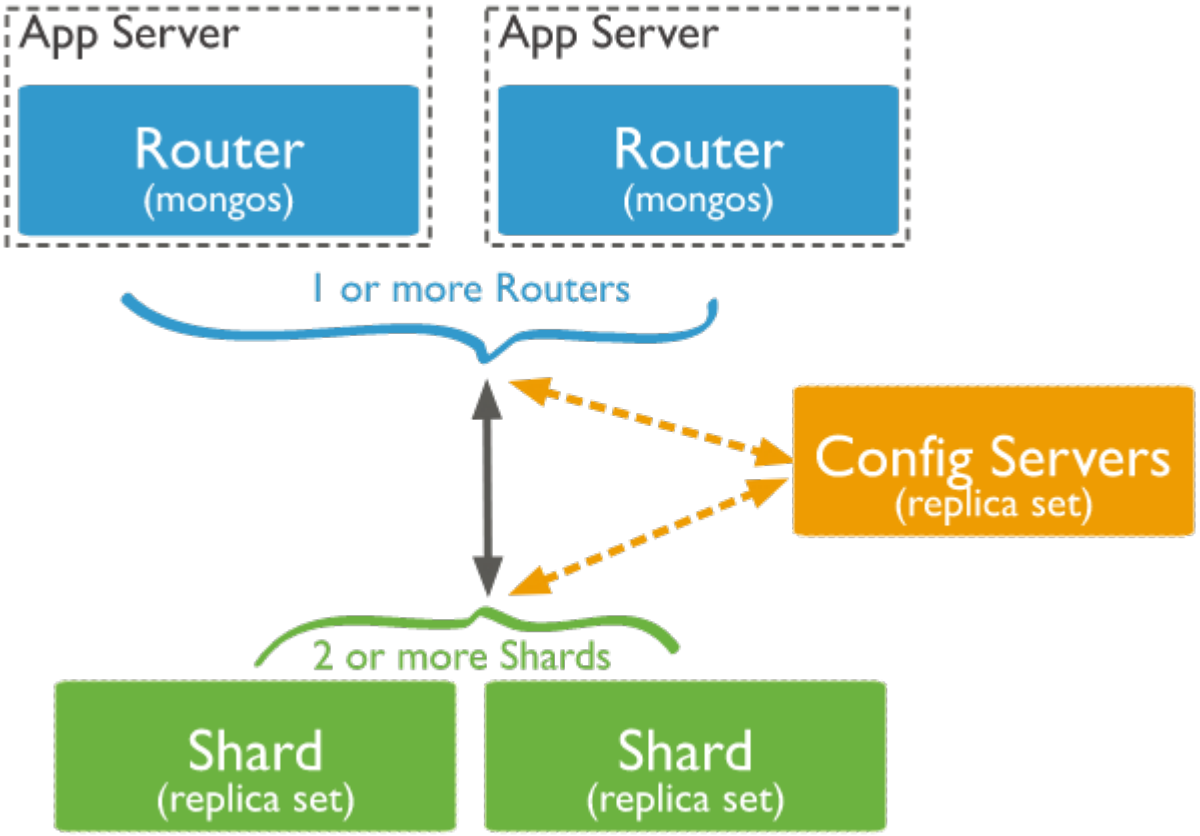
- Primary/Secondary Replication
- Sharding (horizontal scaling)

Indexing

- Can be used on one or several fields (implemented using B-trees)
- Also 2 dimensional spatial indexes for geometry-based data

Read more: https://docs.mongodb.com/manual/replication/ - https://docs.mongodb.com/manual/core/transactions/

# MongoDB: Shard Cluster




Read more: https://docs.mongodb.com/manual/core/sharded-cluster-components/

# Hands-on MongoDB
## (Docker image)

# MongoDB with Dockers

- We use the official container mongo preconfigured

```
$ docker pull mongo
```

- create a small network named mongonet with one server and one client

```
$ docker network create mongonet


$ docker run -it -p 27017:27017 --name mongo_server
        --network=mongonet mongo:latest
        /usr/bin/mongod --smallfiles --bind_ip_all


$ docker run -it --name mongo_cli
        --network=mongonet mongo:latest /bin/bash
```

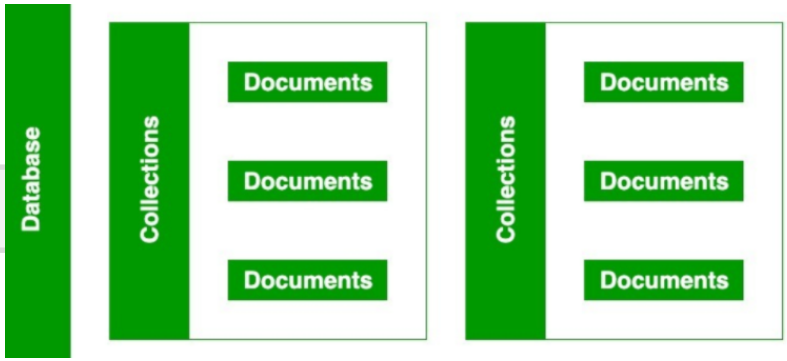
# Mongo CLI: basic operations

- Use the command line interface on the client to connect to the mongo server

```
$ mongosh mongo_server:27017
```

**Create and switch to a new database**

```
> use [databasename]
```




**Insert a document:** insert a document into a collection (e.g., named mycoll). The operation will create the collection if it does not exist yet.

```
> db.mycoll.insert(...)
```

# Mongo CLI: Basic operations

**Find documents:** the find() method issues a query to retrieve data from a collection. All queries have the scope of a single collection.

- Queries can return all documents or only those matching a specific filter or criteria
- The find() method returns results in a cursor (an iterable object that yields documents)

```
> db.mycoll.find()


# filter the documents using the query operators {...}
> db.mycoll.find({ <query> })
```

# Mongo CLI: Query operators

```
# Exact match
> db.mycoll.find({"price" : 300 })
# Comparison (eq, gt, gte, lt, lte, in, nin):
> db.mycoll.find({"price" : { $gt: 300 } })
> db.mycoll.find({"year" : { $in: [2012, 2016] } })
# Existence (if document contains a field):
> db.mycoll.find({"discount" : { $exists: true } })
# logical (and, or, not, nor):
# AND:
> db.mycoll.find({field1 : {...}, field2 : {...} })
# OR:
> db.mycoll.find({
    $or: [{...}, {...}]
  })
```

https://docs.mongodb.com/manual/reference/operator/query/

# Case Study (1)

We consider as use case a content management system, which needs to manage posts, images, videos, and so on, each with its own specific attributes.

```
> use cms
> db.cms.insert({ name : "hello world",
                  type : "post", size : 250,
                  comments : ["c1", "c2"] } )
> db.cms.insert({ name : "sunny day",
                  type : "image",
                  size : 300, url : "abc" })
> db.cms.insert({ name : "tutorial",
                  type : "video", length : 125,
          path :"/video.flv",
          metadata : {quality : "480p",
          color : "b/n", private : false } })
```

# Case Study (2)

```
> db.cms.find()

{ "_id" : ObjectId("5c76cafc8c3eb2953b2c8c2c"), "name" : "hello world",
"type" : "post", "size" : 250,
"comments" : [ "c1", "c2" ] }

{ "_id" : ObjectId("5c76cb0a8c3eb2953b2c8c2d"), "name" : "sunny day",
"type" : "image",
"size" : 300, "url" : "abc" }

{ "_id" : ObjectId("5c76cb158c3eb2953b2c8c2e"), "name" : "tutorial", "type"
: "video", "length" : 125, "path" : "/video.flv", "metadata" : { "quality" :
"480p", "color" : "b/n", "private" : false } }
```

# Case Study (3)

```
> db.cms.find( {size : { $gt : 100 } } )
```

```
{ "_id" : ObjectId("5c76cafc8c3eb2953b2c8c2c"), "name" : "hello world", "type" : "post",
"size" : 250,
"comments" : [ "c1", "c2" ] }
{ "_id" : ObjectId("5c76cb0a8c3eb2953b2c8c2d"),
"name" : "sunny day", "type" : "image", "size" : 300,
"url" : "abc" }
```

```
> db.cms.find( {size : { $lt : 100 } } )
> db.cms.find({ length : { $exists: true } } )
```

```
{ "_id" : ObjectId("5c76cb158c3eb2953b2c8c2e"),
"name" : "tutorial", "type" : "video", "length" : 125, "path" : "/video.flv", "metadata" :
{ "quality" : "480p", "color" : "b/n", "private" : false } }
```

# Case Study (4)

```
> db.cms.findOne({comments:{ $exists: true }})
{           "_id" : ObjectId("5c76cafc8c3eb2953b2c8c2c"),
            "name" : "hello world",
            "type" : "post",
            "size" : 250,
            "comments" : [
                        "c1",
                        "c2"
            ]
}
> db.cms.findOne({ comments:
              { $exists: true}}).comments[1]
  c2
```

# Case Study (5)

```
> db.cms.find({ comments : { $exists: true }})

{ "_id" : ObjectId("5c76cafc8c3eb2953b2c8c2c"), "name" : "hello world", "type" : "post",
"size" : 250,
"comments" : [ "c1", "c2" ] }

> db.cms.find({ "comments.2": { $exists: true }})

> db.cms.find({ "comments.1": { $exists: true }})

{ "_id" : ObjectId("5c76cafc8c3eb2953b2c8c2c"), "name" : "hello world", "type" : "post",
"size" : 250,
"comments" : [ "c1", "c2" ] }
```

# Case Study (6)

```
> db.cms.find( {size:{ $gt : 100 },
                                    type: "image"})
{ "_id" : ObjectId("5c76cb0a8c3eb2953b2c8c2d"), "name" : "sunny day",
"type" : "image", "size" : 300, "url" : "abc" }

> db.cms.find( { $or : [{size : { $gt : 100 }},
                                                    {type : "image"}]})

{ "_id" : ObjectId("5c76cafc8c3eb2953b2c8c2c"), "name" : "hello world",
"type" : "post", "size" : 250, "comments" : [ "c1", "c2" ] }
{ "_id" : ObjectId("5c76cb0a8c3eb2953b2c8c2d"), "name" : "sunny day",
"type" : "image", "size" : 300, "url" : "abc" }
```

# Mongo CLI: Query operators

**Sort query results:** to specify an order for the result set, append the sort() method to the query.

- Pass to sort() a document which contains the field(s) to sort by and the corresponding sort type (1 for ascending, -1 for descending)

```
> db.mycoll.find().sort( { "name" : 1 } )
```

https://docs.mongodb.com/manual/reference/operator/query/

# Case Study (7)

```
> db.cms.find().sort({ name : 1 })

{ "_id" : ObjectId("5c76cafc8c3eb2953b2c8c2c"),
"name" : "hello world", "type" : "post", "size" : 250, "comments" : [ "c1", "c2" ] }

{ "_id" : ObjectId("5c76cb0a8c3eb2953b2c8c2d"),
"name" : "sunny day", "type" : "image", "size" : 300, "url" : "abc" }

{ "_id" : ObjectId("5c76cb158c3eb2953b2c8c2e"),
"name" : "tutorial", "type" : "video",
"length" : 125, "path" : "/video.flv",
"metadata" : { "quality" : "480p", "color" : "b/n",
"private" : false } }
```

# Mongo CLI: Basic operations

Several update operators are available in mongo: update() - deprecated, updateOne(), updateMany(), replaceOne().

```
> db.mycoll.updateOne(<filter>, <update>)
```

<filter> and <update> are expressed as bson objects

$set sets the value of a field in a document. The update can be applied to one or multiple occurrencies that matches the update filter.

```
> db.mycoll.updateOne( { field : value },
            { $set:
                        { "address.street": "East 31st Street" }     } )
```

Update multiple occurrences

```
> db.mycoll.updateMany(<filter>, <update>)
```

https://docs.mongodb.com/manual/reference/operator/update/

# Case Study (8)

```
> db.cms.updateOne({ "name" : "Canon EOS 750D" }, {$set: {"address.street":"East 31st Street"}})
WriteResult({"nMatched":0, "nUpserted":0, "nModified":0})

> db.cms.updateOne({"name" : "mycms-logo"},
                                        {$set: {"metadata.quality": "hd"}})
WriteResult({ "nMatched":0, "nUpserted":0, "nModified":0})

> db.cms.find({name : "mycms-logo"})

> db.cms.find({type : "image"})

{ "_id" : ObjectId("5c76cb0a8c3eb2953b2c8c2d"), "name" : "sunny day", "type" : "image", "size" : 300, "url" : "abc" }
```

# Case Study (9)

```
> db.cms.updateMany({type : "image"},  {$set:
{"metadata.author": "myname"}})

WriteResult({ "nMatched":1, "nUpserted":0, "nModified":1 })

>  db.cms.find({type : "image"})

{ "_id" : ObjectId("5c76cb0a8c3eb2953b2c8c2d"), "name" : "sunny day",
"type" : "image", "size" : 300, "url" : "abc", "metadata" : { "author" :
"myname" } }

> db.cms.find({ type : "post" })

{ "_id" : ObjectId("5c76cafc8c3eb2953b2c8c2c"), "name" : "hello world",
"type" : "post", "size" : 250, "comments" : [ "c1", "c2" ] }
```

# Case Study (10)

```
> db.cms.update({ name : "hello world"},
            {$push: {"comments": "a new comment"}})

WriteResult({"nMatched":1, "nUpserted":0, "nModified":1 })

> db.cms.update({ name : "hello world" },
                        {$push: {"comments": "a second new comment"}})

WriteResult({"nMatched":1, "nUpserted":0, "nModified":1 })

> db.cms.find({ type : "post"})

{ "_id" : ObjectId("5c76cafc8c3eb2953b2c8c2c"), "name" : "hello world",
"type" : "post", "size" : 250, "comments" : [ "c1", "c2", "a new comment", "a
second new comment" ] }
```

# Mongo CLI: Basic operations

**Remove documents:** the remove() method removes documents from a collection. The method takes a conditions document that determines the documents to remove

```
> db.mycoll.remove(
        { "borough": "Manhattan" } )
> db.mycoll.remove(
        { "borough": "Queens" },
        { justOne: true } )


# remove all documents:
> db.mycoll.remove( { } )
```

https://docs.mongodb.com/manual/reference/operator/update/

# Mongo CLI: Basic operations

**Drop a collection:** to remove all documents from a collection (and the collection itself), the drop() operation should be used.

```
> db.mycoll.drop()
```

https://docs.mongodb.com/manual/reference/operator/update/

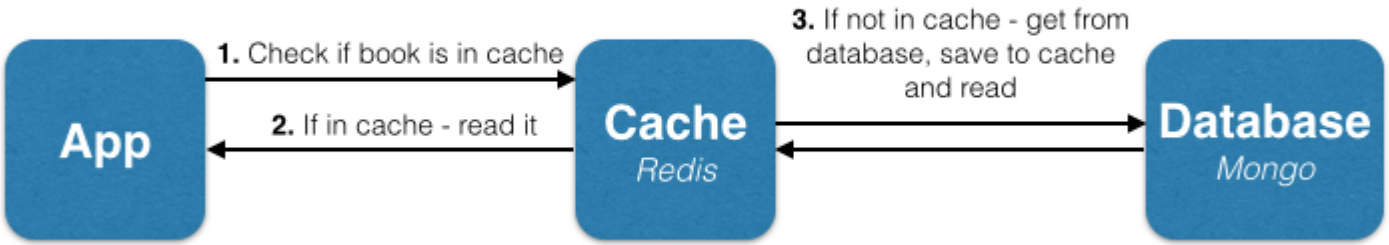
# Different needs, different solutions

- When storing data, it is best to use multiple data storage technologies
  - Chosen upon the way data is being used

A simple yet effective use case:

- A simple web library, which interacts with a (persistent) database
- the communication with the database can cause a big overhead

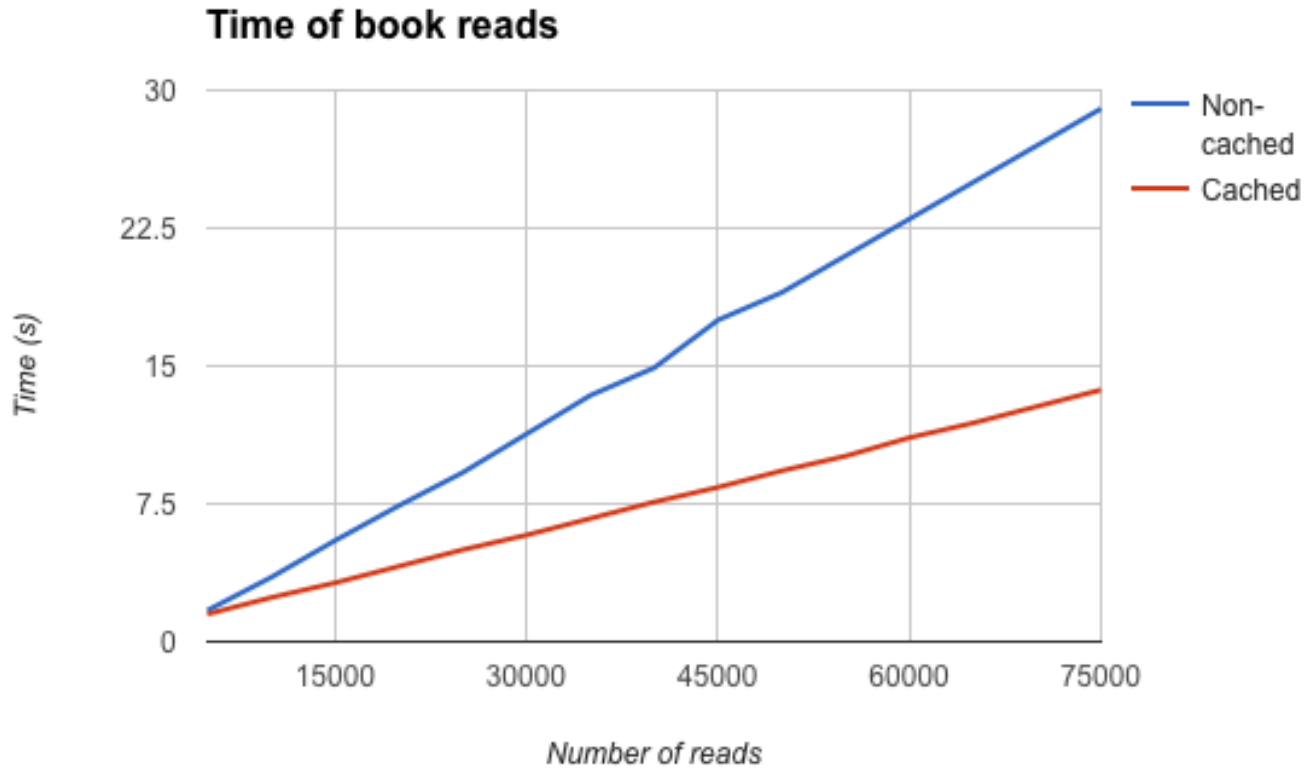
Solutions?
Use an in-memory key-value store as **caching** system!




Read more: https://www.sitepoint.com/caching-a-mongodb-database-with-redis/

# Different needs, different solutions

- Case study: the management of a library
- Books are stored in a Mongo database
- A web application can access and read books




Read more: https://www.sitepoint.com/caching-a-mongodb-database-with-redis/