Document stores

Taxonomy of NoSQL

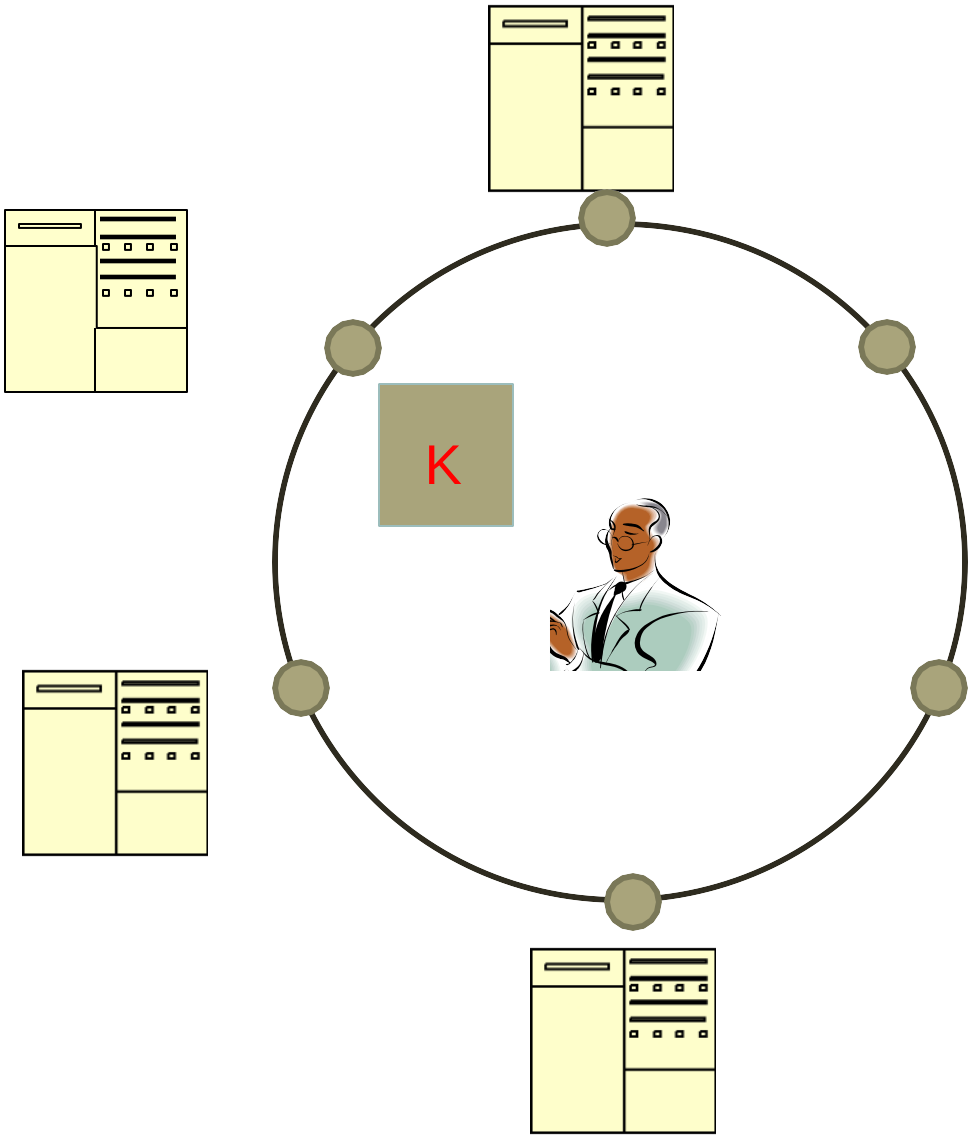
- Key-value

- Graph database

- Document-oriented

- Column family

Typical NoSQLarchitecture



Hashing function maps each key to a server (node)

Document stores

- Flexible schema
- JSON/BSON documents
  - Embedded documents
  - Referenced documents

- CouchDB: http://couchdb.apache.org/
- MongoDB: http://www.mongodb.org/

We'll use the mongo shell for class, but if you want to use a GUI to interact with MongoDB, you may want to look into Robo 3T (previously robomongo) https://robomongo.org/

What is MongoDB?

- Developed by 10gen
  - Founded in 2007
- A 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
      - Based on JSON – B stands for Binary
  - Written in C++
  - Supports APIs (drivers) in many computer languages

MongoDB Features

- Dynamic schema
- Document-Oriented storage
- Full Index Support
- Replication & High Availability
- Auto-Sharding
  - Built-in horizontal scaling via automated range-based partitioning of data
- Querying
- Fast In-Place Updates
- Map/Reduce functionality

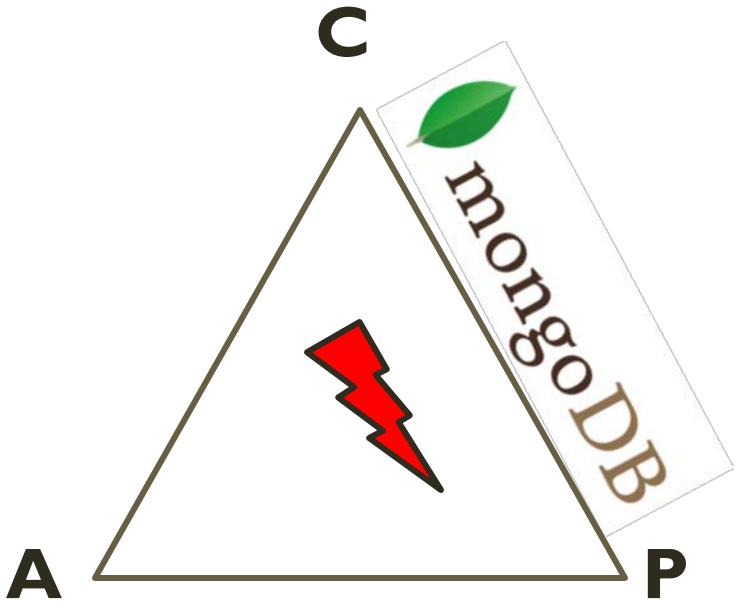
Agile

Scalable

MongoDB: CAP approach

Focus on Consistency and Partition tolerance

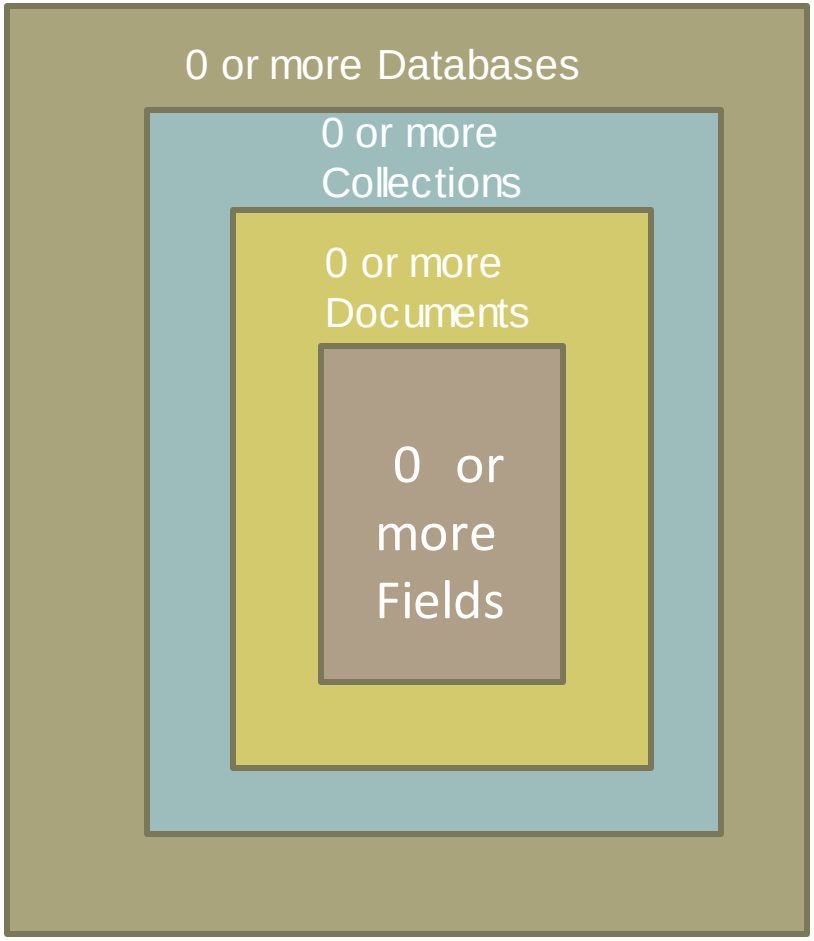
- **C**onsistency
  - all replicas contain the same version of the data
- **A**vailability
  - system remains operational on failing nodes
- **P**artition tolarence
  - multiple entry points
  - system remains operational on system split



CAP Theorem:
satisfying all three at the same time is impossible

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.



MongoDB

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

Mongo hits a sweet spot between the powerful queryability of a relational database and the distributed nature of other databases

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”}]

Document store

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (JSON, B
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Docume
Foreign Key	⇨	Reference
Partition	⇨	Shard

```
> db.user.findOne({age:39})
{
        "_id" : ObjectId("5114e0bd42…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39,
        "interests" : [
                "Reading",
                "Mountain Biking ]
        "favorites": {
                "color": "Blue",
                "sport": "Soccer"}
}
```

CRUD

Create

db.collection.insert(<document>)

db.collection.save(<document>)

db.collection.update(<query>, <update>, { upsert: true })

Read

db.collection.find(<query>, <projection>)

db.collection.findOne(<query>, <projection>)

Update

db.collection.update(<query>, <update>, <options>)

Delete

db.collection.remove(<query>, <justOne>)

CRUD example

```
> db.user.insert({
        first: "John",
        last : "Doe",
        age: 39
})
```

```
> db.user.find ()
{
        "_id" : ObjectId("51…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39
}
```

```
> db.user.update(
        {"_id" : ObjectId("51…")},
        {
            $set: {
                age: 40,
                salary: 7000}
        }
)
```

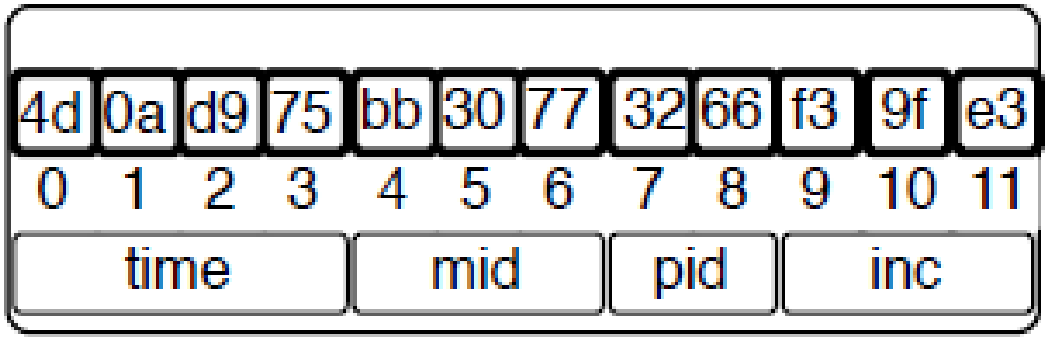
```
> db.user.remove({
        "first": /^J/
})
```

Let's get started – command-line fun

```
> mongo moderndb

> db.towns.insert({
name: "New York", population:
22200000,
lastCensus: ISODate("2016-07-
01"),
famousFor: [ "the MOMA",
"food", "Derek Jeter" ], mayor
: {
    name : "Bill de Blasio",
    party : "D"
}
})

> show collections

> db.towns.find()
```

ObjectId

- _id field of type ObjectId.
- Akin to SERIAL incrementing a numeric primary key in PostgreSQL.
- The ObjectId is always 12 bytes, composed of a timestamp, client machine ID, client process ID, and a 3-byte incremented counter.



Javascript

- Native tong of MongoDB

- Ask for help
  - db.help()
  - db.towns.help()

- Object and functions
  - typeof db
  - typeof db.towns
  - typeof db.towns.insert

- Get source code of a function (call it without parameters)
  - db.towns.insert

Let's insert data using our own function

```
function insertCity(name, population, lastCensus, famousFor,
mayorInfo) {
   db.towns.insert({ name: name,
   population: population, lastCensus: ISODate(lastCensus),
   famousFor: famousFor,
   mayor : mayorInfo });
}
```

Now we can call it

```
> insertCity("Punxsutawney", 6200, '2016-01-31', ["Punxsutawney
Phil"], { name : "Richard Alexander" }
)

> insertCity("Portland", 582000, '2016-09-20',
["beer", "food", "Portlandia"], { name : "Ted Wheeler", party :
"D" })
```

Querying data

```
db.towns.find({ "_id" : ObjectId("59094288afbc9350ada6b807") })

db.towns.find({ _id : ObjectId("59094288afbc9350ada6b807") }, {
name : 1 })

db.towns.find({ _id : ObjectId("59094288afbc9350ada6b807") }, {
name : 0 })
```

Perl-compatible regular expression (PCRE)

```
db.towns.find(
{ name : /^P/, population : { $lt : 10000 } },
{ _id: 0, name : 1, population : 1 }
)
```

Can construct operations as you would objects

```
> var population_range = {$lt: 1000000, $gt: 10000}
> db.towns.find( { name : /^P/, population : population_range },{
name: 1 })

> db.towns.find(
{ lastCensus : { $gte : ISODate('2016-06-01') } },
{ _id : 0, name: 1 })
```

Querying nested array data

Matching exact values

```
> db.towns.find(
{ famousFor : 'food' },
{ _id : 0, name : 1, famousFor : 1 }
)
```

Matching partial values:

```
> db.towns.find(
    { famousFor : /moma/ },
    { _id : 0, name : 1, famousFor : 1 })
```

Query by all matching values:

```
> db.towns.find(
    { famousFor : { $all : ['food', 'beer'] } },
    { _id : 0, name:1, famousFor:1 }
)
```

Or the lack of matching values:

```
> db.towns.find(
    { famousFor : { $nin : ['food', 'beer'] } },
    { _id : 0, name : 1, famousFor : 1 }
)
```

Querying nested documents

Find towns with mayors from the Democratic party

```
> db.towns.find(
{ 'mayor.party' : 'D' },
{ _id : 0, name : 1, mayor : 1 }
```

Find towns with mayors who don't have a party

```
> db.towns.find(
{ 'mayor.party' : { $exists : false } },
{ _id : 0, name : 1, mayor : 1 }
)
```

New collection for countries

```
> db.countries.insert({
        _id : "us",
        name : "United States",
        exports : {
        foods : [
                { name : "bacon", tasty : true },
                { name : "burgers" }
        ]}
    })
> db.countries.insert({
        _id : "ca",
        name : "Canada",
        exports : {
        foods : [
                { name : "bacon", tasty : false },
                { name : "syrup", tasty : true }
        ]}
    })
> db.countries.insert({
        _id : "mx",
        name : "Mexico",
        exports : {
        foods : [{
                name : "salsa", tasty : true, condiment : true
                  }]}
    })
```

elemMatch

Find a country that not only exports bacon but exports tasty bacon

```
> db.countries.find(
      { 'exports.foods.name' : 'bacon', 'exports.foods.tasty' : true },
      { _id : 0, name : 1 }
)
```

Canada?

ElemMatch to the rescue:

```
> db.countries.find(
  {
      'exports.foods' : {
      $elemMatch : {
            name : 'bacon',
            tasty : true
      }}
  },
  { _id : 0, name : 1 }
  )
```

```
> db.countries.find(
  {
      'exports.foods' : {
            $elemMatch : {
                  tasty : true,
                  condiment : { $exists : true }
            }
      }
  },
  { _id : 0, name : 1 }
  )
```

Boolean operators

```
> db.countries.find(
      { _id : "mx", name : "United States" },
      { _id : 1 }
)


db.countries.find(
      {
            $or : [
                  { _id : "mx" },
                  { name : "United States" }
            ]
      },
      { _id:1 }
)
```

Some mongodb commands

Command	Description
$regex	Match by any PCRE-compliant regular expression string (or just use the // delimiters as shown earlier)
$ne	Not equal to
$lt	Less than
$lte	Less than or equal to
$gt	Greater than
$gte	Greater than or equal to
$exists	Check for the existence of a field
$all	Match all elements in an array
$in	Match any elements in an array
$nin	Does not match any elements in an array
$elemMatch	Match all fields in an array of nested documents
$or	or
$nor	Not or
$size	Match array of given size
$mod	Modulus
$type	Match if field is a given datatype
$not	Negate the given operator check

- Check the mongo documentation for a complete list
- Cheat sheet on ICON

For today..

- Select a town via a case-insensitive regular expression containing the word *new*.
- Find all towns whose names contain an *e* and are famous for food or beer.
- Summit the two queries to ICON.