MongoDB Internals

Agenda

Overview
Architecture
Storage Engines
Data Model
Query Engine
Client APIs

Norberto Leite

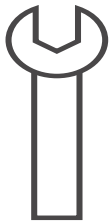
Developer Advocate

Twitter: @nleite
norberto@mongodb.com


mongoDB Overview

MongoDB

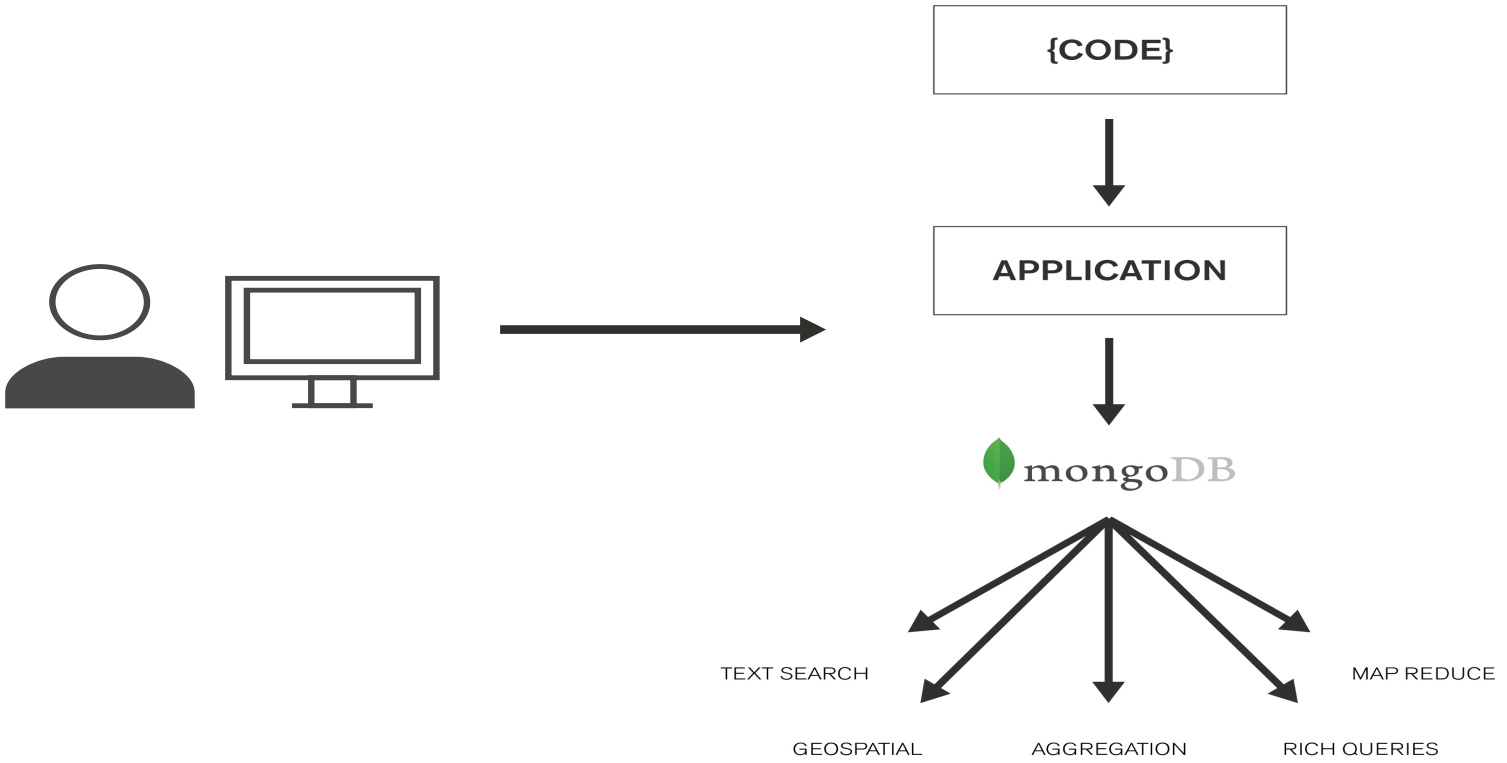
GENERAL PURPOSE



DOCUMENT DATABASE

OPEN-SOURCE

MongoDB is Fully Featured



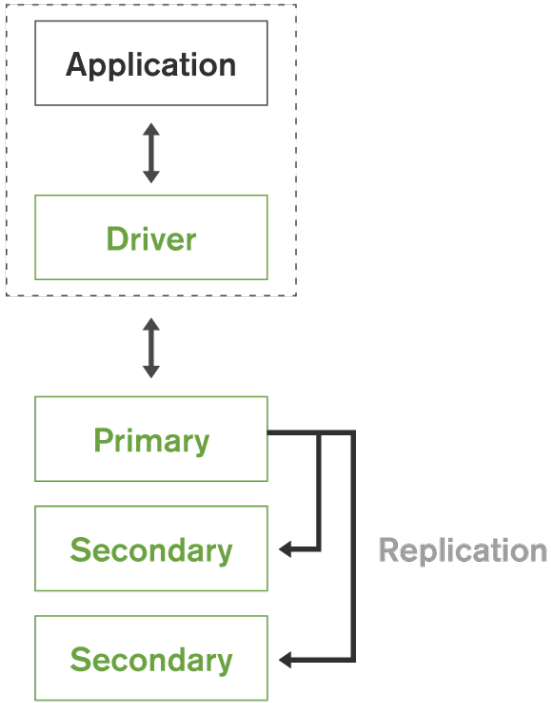
Horizontal Scalable

Three types: hash-based, range-based, location-aware

Increase or decrease capacity as you go

Automatic balancing

Replica Sets



Replica Set – 2 to 50 copies

Self-healing shard

Data Center Aware

Addresses availability considerations:

High Availability

Disaster Recovery

Maintenance

Workload Isolation: operational & analytics

Over **10,000,000** downloads

300,000 Students for MongoDB University

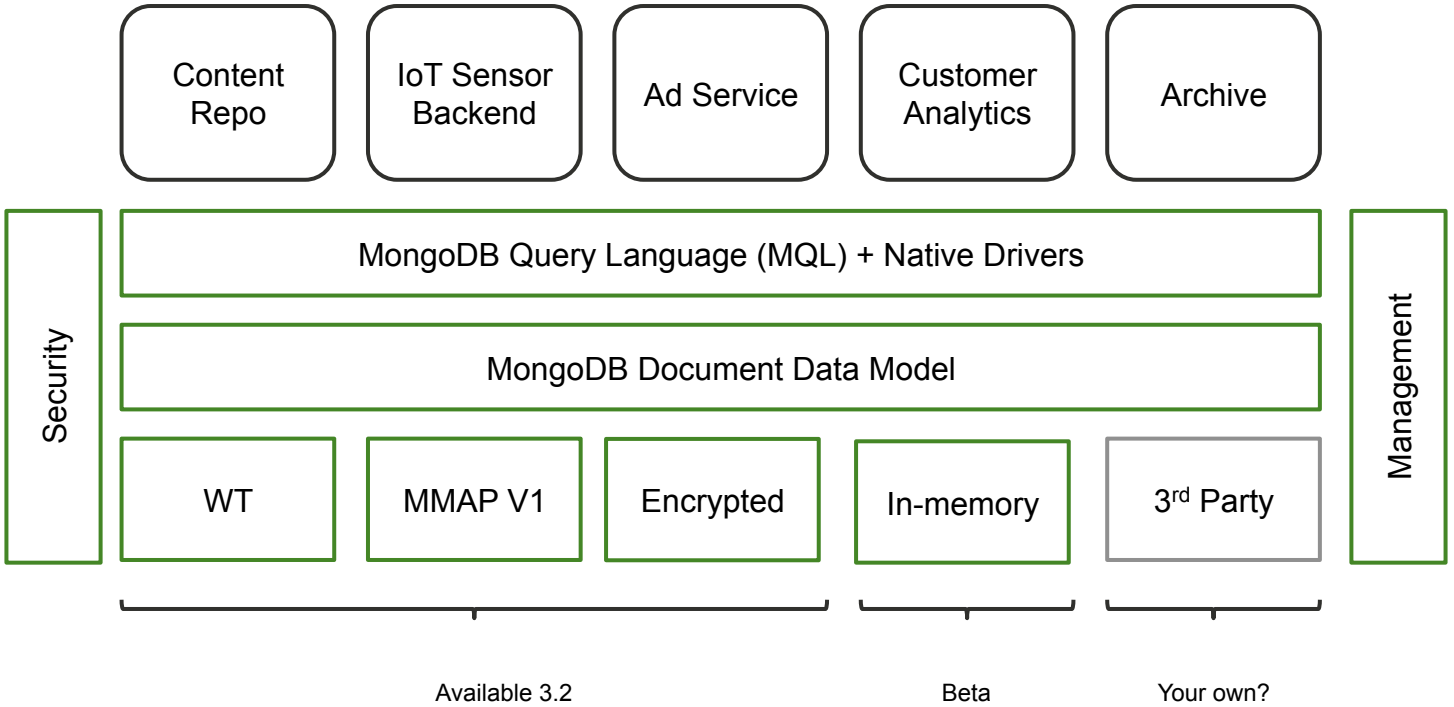
35,000 attendees to MongoDB events annually

Over **1,000** Partners

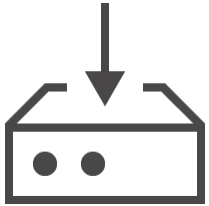
Over **2,000** Paying Customers

MongoDB Architecture

MongoDB Architecture

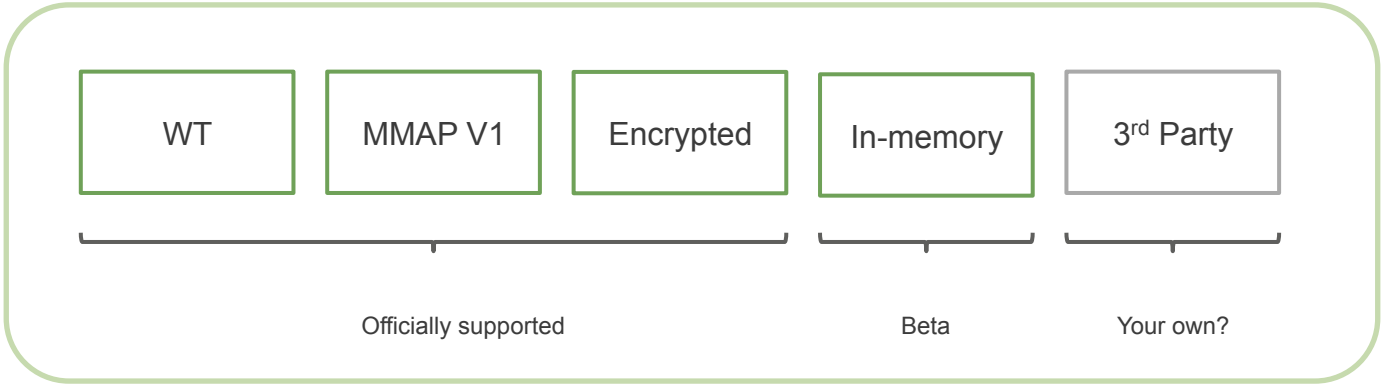


MongoDB Architecture



Storage Layer

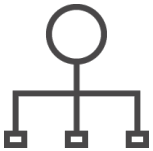
- Different workloads require different storage strategies
- Exposed by a Storage Engine API
- Provides more flexibility to your deployments



MongoDB Architecture

BSON

Collections



Indexes

Databases

MongoDB Document Data Model

Data Model

MongoDB Architecture

Java

Ruby

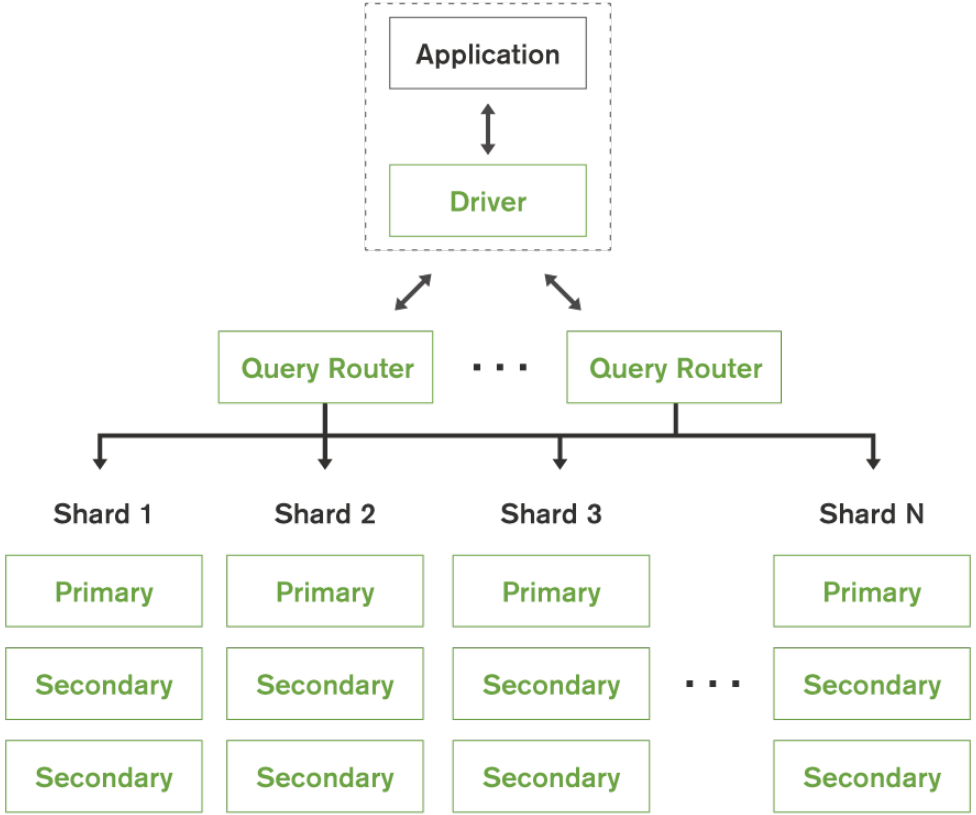
Python

Perl

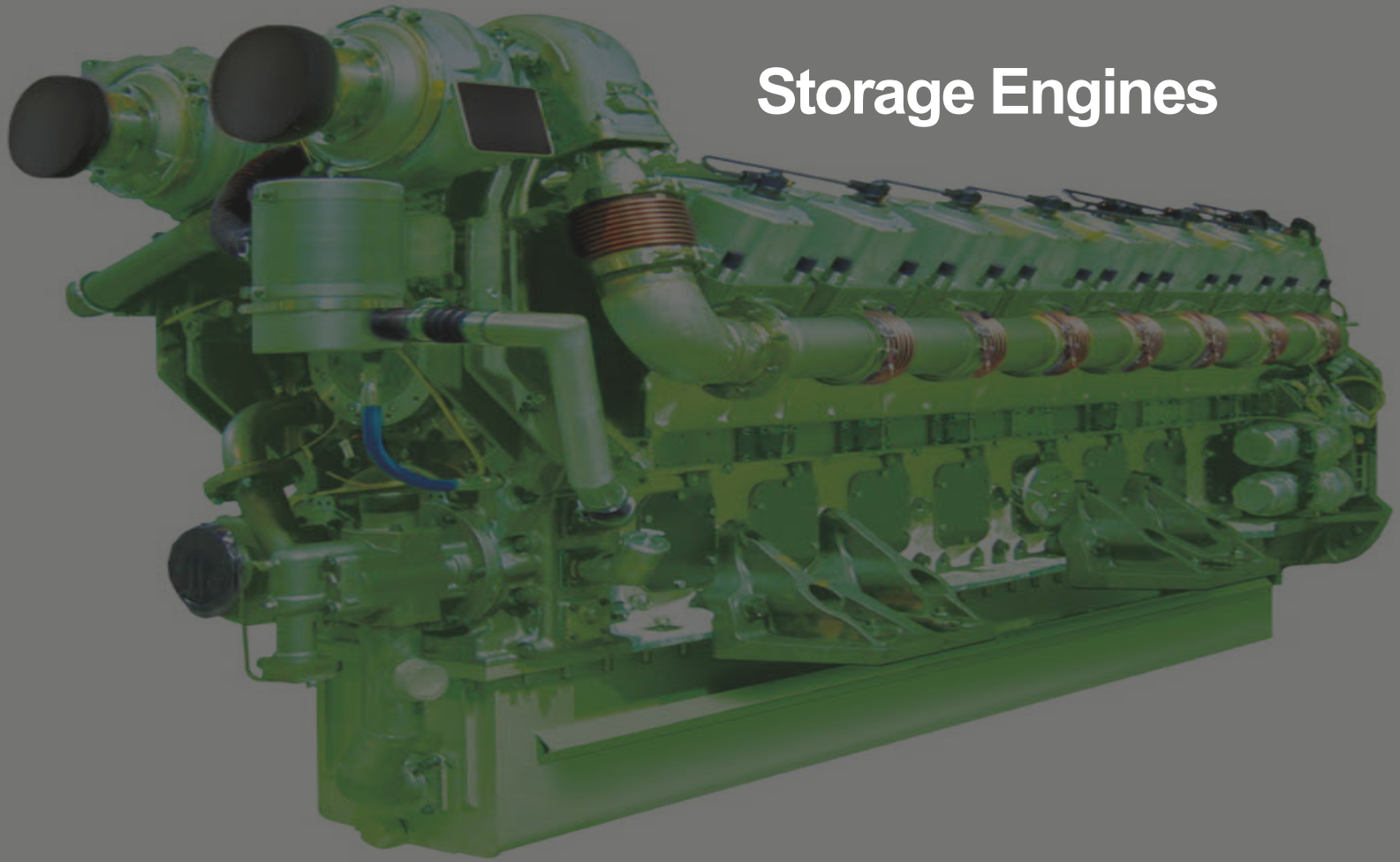
MongoDB Query Language (MQL) + Native Drivers


```
db.coll.insert({'name': 'Norberto'})
db.coll.update({'name': 'Norberto'}, {'$set':{'role': 'Developer Advocate'})

db.coll.find({'role': /^Developer/})
db.coll.find({}).skip(10).limit(20)

db.coll.aggregate({'$group': { '_id': '$role', "howmany": {"$sum":1} }})

db.serverStatus()
```

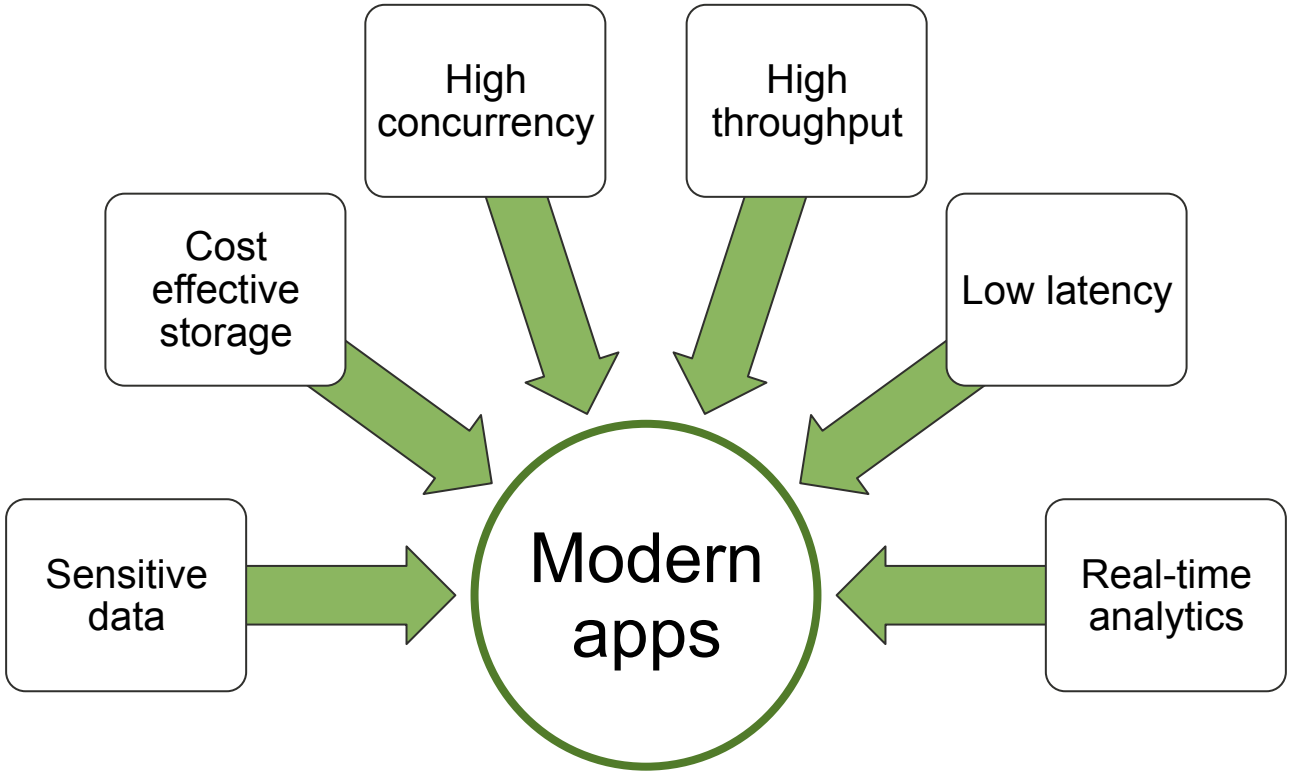
Distributed Database

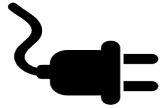


Storage Engines



Varying Access & Storage Requirements





Storage Engine API

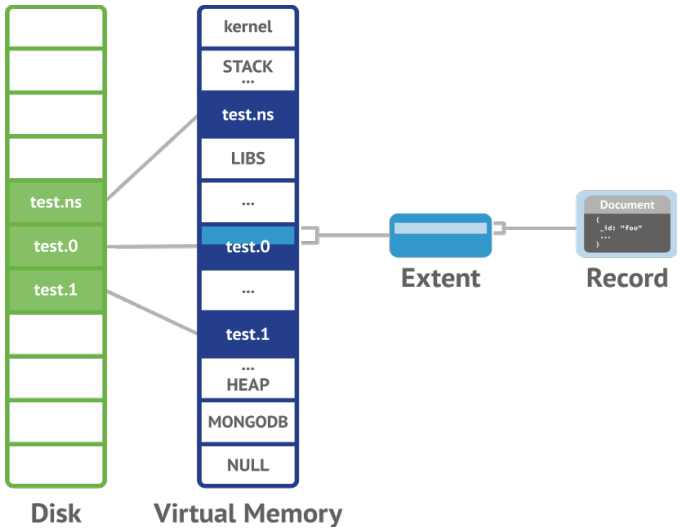
- Allows to "plug-in" different storage engines
  - Different use cases require different performance characteristics
  - mmapv1 is not ideal for all workloads
  - More flexibility
    - Can mix storage engines on same replica set/sharded cluster
- Opportunity to integrate further (HDFS, native encrypted, hardware optimized …)

WiredTiger is the New Default

WiredTiger – widely deployed with 3.0 – is now the default storage engine for MongoDB.

- Best general purpose storage engine
- 7-10x better write throughput
- Up to 80% compression

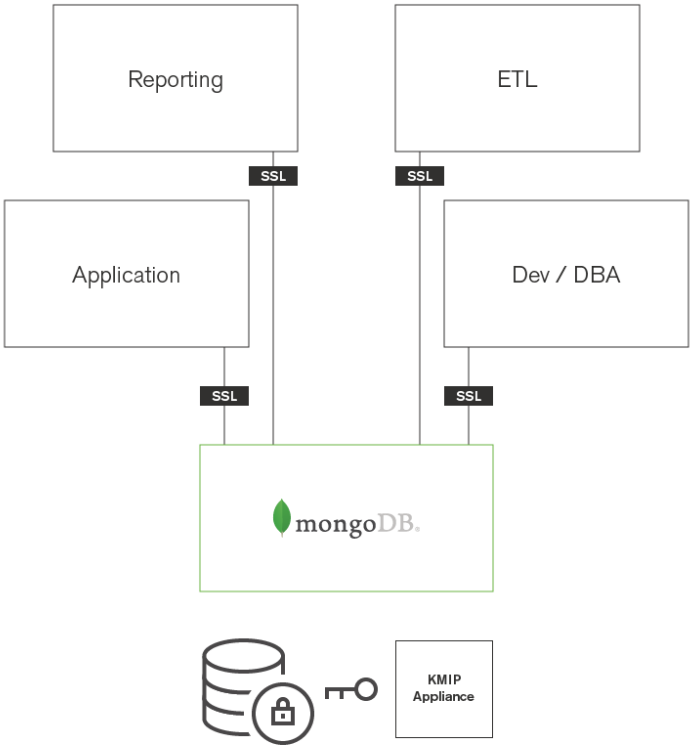
Good Old MMAPv1



MMAPv1 is our traditional storage engine that allows a great deal of performance for read heavy applications

- Improved concurrency control
- Great performance on read-heavy workloads
- Data & Indexes memory mapped into virtual address space
- Data access is paged into RAM
- OS evicts using LRU
- More frequently used pages stay in RAM

Encrypted Storage Engine



Encrypted storage engine for end-to-end encryption of sensitive data in regulated industries

- Reduces the management and performance overhead of external encryption mechanisms
- AES-256 Encryption, FIPS 140-2 option available
- Key management: Local key management via keyfile or integration with 3rd party key management appliance via KMIP
- Based on WiredTiger storage engine
- Requires MongoDB Enterprise Advanced

mongoDB

In-Memory Storage Engine (Beta)

Handle ultra-high throughput with low latency and high availability

- Delivers the extreme throughput and predictable latency required by the most demanding apps in Adtech, finance, and more.
- Achieve data durability with replica set members running disk-backed storage engine
- Available for beta testing and is expected for GA in early 2016

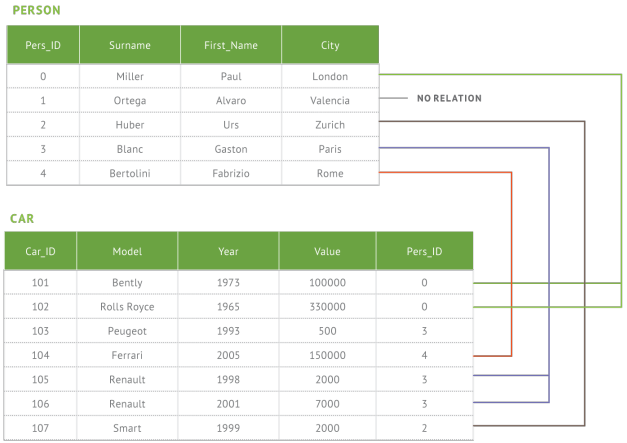
Data Model

Terminology

RDBMS	MongoDB
Database	Database
Table	Collection
Index	Index
Row	Document
Join	Embedding & Linking

Document Data Model

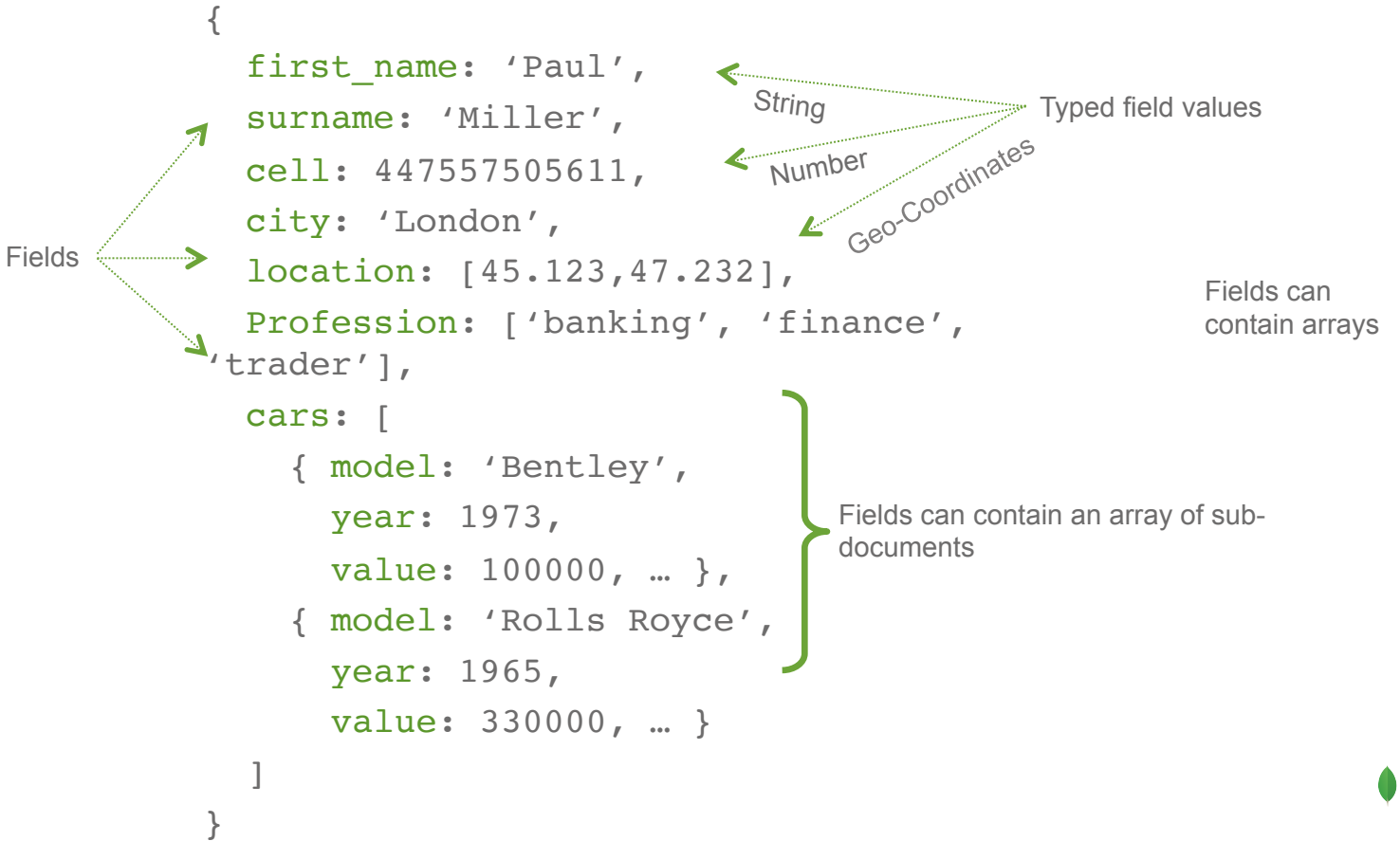
Relational



MongoDB

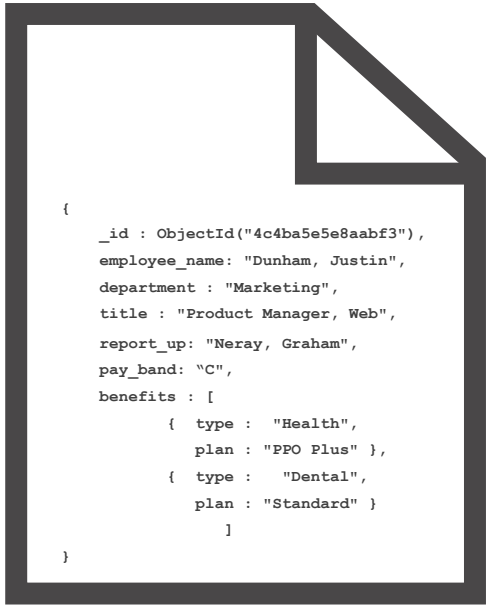
```
{
  first_name: ‘Paul’,
  surname: ‘Miller’,
  city: ‘London’,
  location:
[45.123,47.232],
  cars: [
    { model: ‘Bentley’,
      year: 1973,
      value: 100000, … },
    { model: ‘Rolls Royce’,
      year: 1965,
      value: 330000, … }
  ]
}
```

Documents are Rich Data Structures



mongoDB

Document Model Benefits



Agility and flexibility

Data model supports business change

Rapidly iterate to meet new requirements

Intuitive, natural data representation

Eliminates ORM layer

Developers are more productive

Reduces the need for joins, disk seeks

Programming is more simple

Performance delivered at scale

Dynamic Schemas

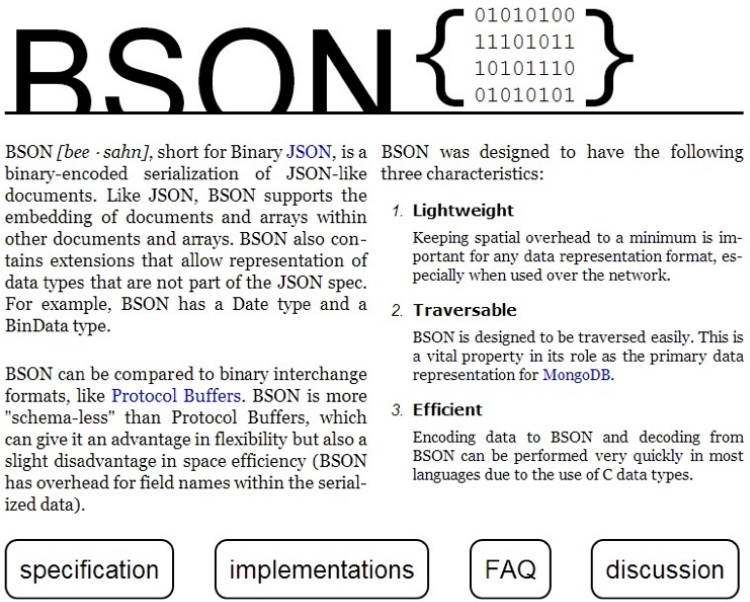
```
{
    policyNum: 123,
    type: auto,
    customerId: abc,
    payment: 899,

    deductible: 500,
    make: Taurus,
    model: Ford,
    VIN: 123ABC456,
}
```

```
{
    policyNum: 456,
    type: life,
    customerId: efg,
    payment: 240,

    policyValue: 125000,
    start: jan, 1995
    end: jan, 2015
    }
```

```
{
  policyNum: 789,
  type: home,
  customerId: hij,
   payment: 650,

  deductible: 1000,
  floodCoverage: No,
  street: “10 Maple Lane”,
   city: “Springfield”,
   state: “Maryland”
}
```

BSON

- Binary JSON serialization format
- JSON with types
- The language MongoDB speaks



BSON { 01010100 11101011 10101110 01010101 }

BSON *[bee · sahn]*, short for Binary JSON, is a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays. BSON also contains extensions that allow representation of data types that are not part of the JSON spec. For example, BSON has a Date type and a BinData type.

BSON can be compared to binary interchange formats, like Protocol Buffers. BSON is more "schema-less" than Protocol Buffers, which can give it an advantage in flexibility but also a slight disadvantage in space efficiency (BSON has overhead for field names within the serialized data).

BSON was designed to have the following three characteristics:

1. **Lightweight**

   Keeping spatial overhead to a minimum is important for any data representation format, especially when used over the network.

2. **Traversable**

   BSON is designed to be traversed easily. This is a vital property in its role as the primary data representation for MongoDB.

3. **Efficient**

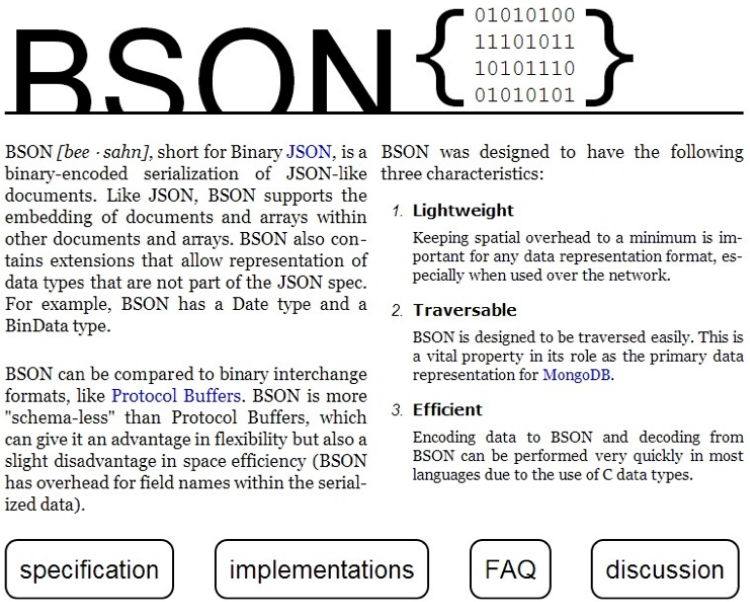
   Encoding data to BSON and decoding from BSON can be performed very quickly in most languages due to the use of C data types.

specification | implementations | FAQ | discussion

http://bsonspec.org

BSON Data Types

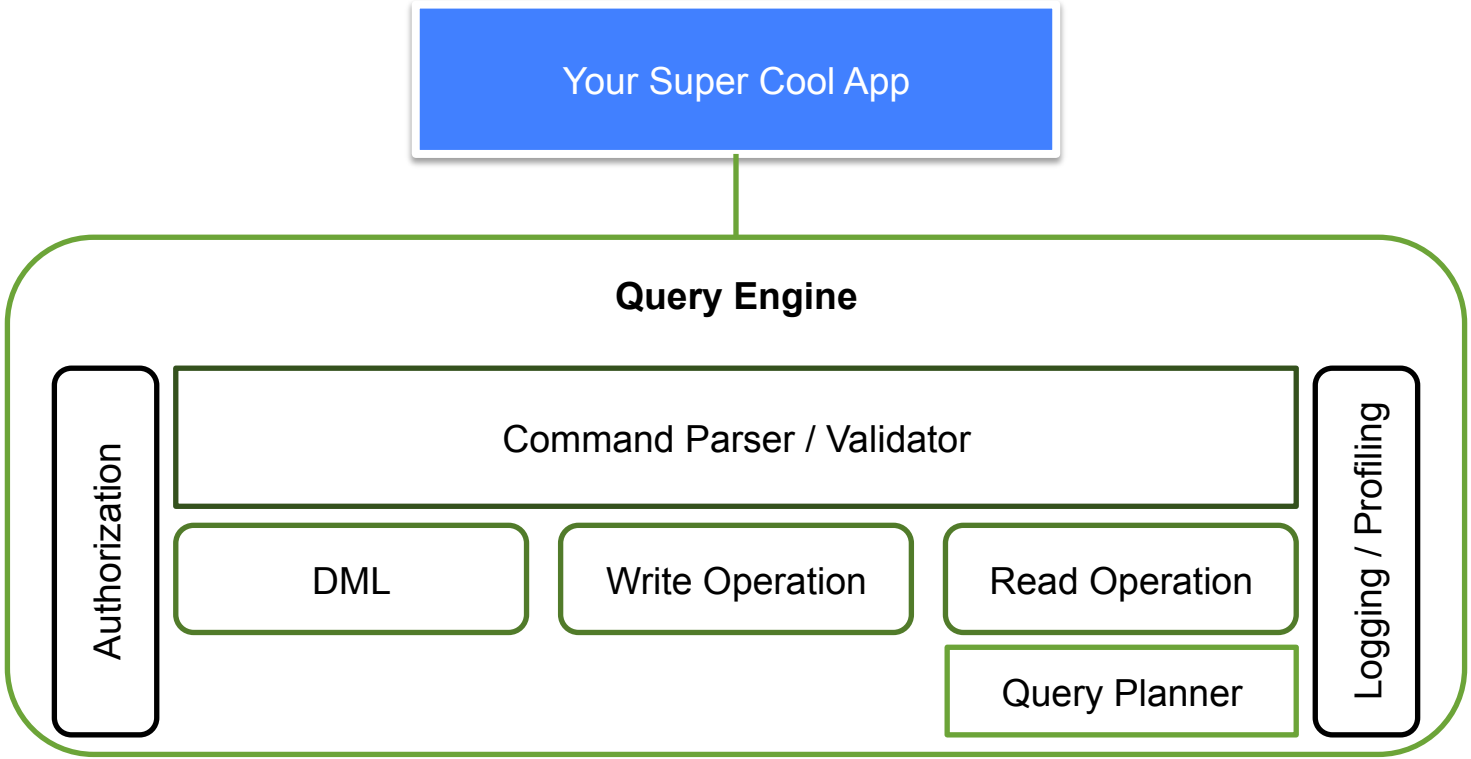
- String
- Document
- Array
- Binary
- ObjectId
- Boolean
- Date
- Timestamp
- Double (64 bit)
- Long (64 bit)
- Integer (32 bit)
- Min Key
- Max Key
- Javascript
- Javascript with Scope
- Null value



http://bsonspec.org

Query Engine

Query Engine



Document Validation

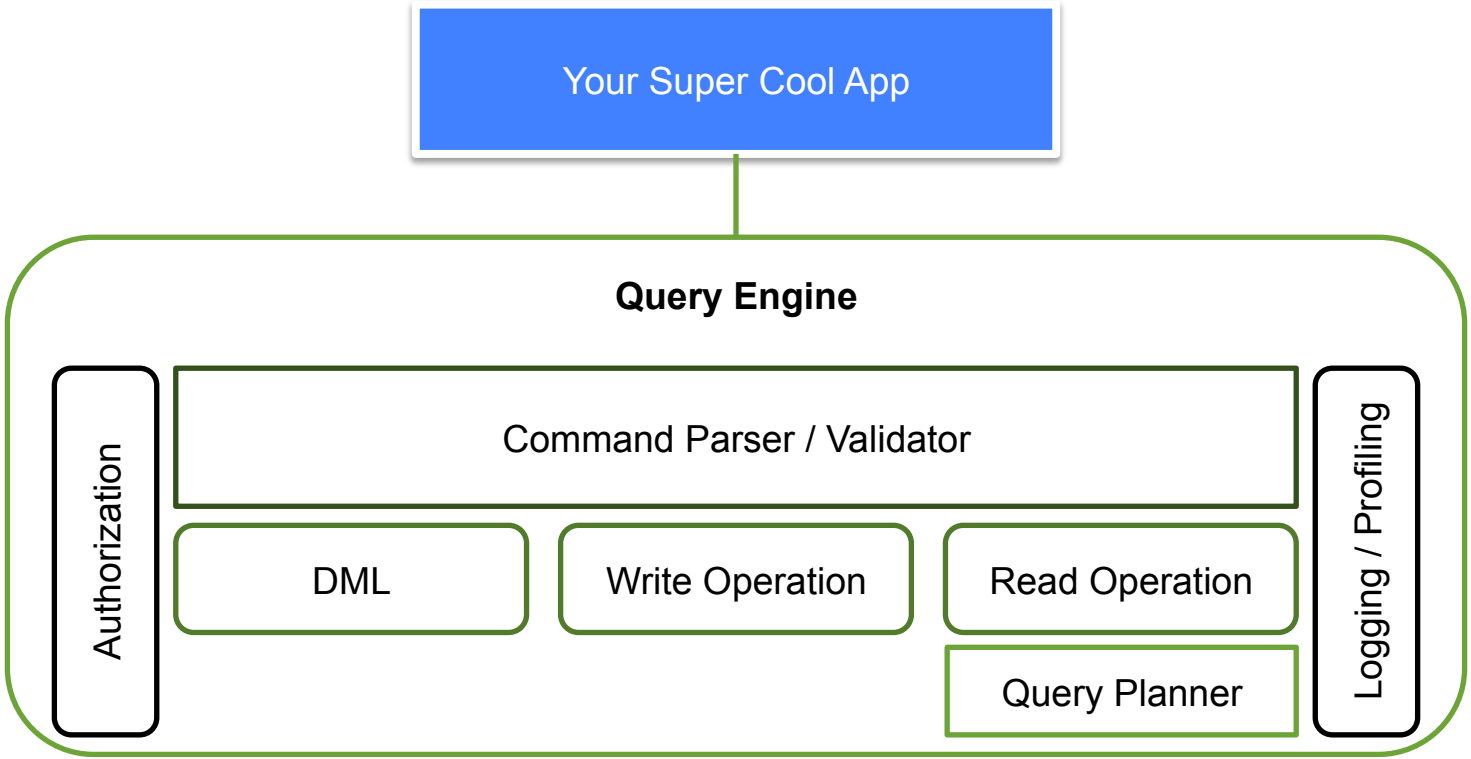
Go and try it out:

https://jira.mongodb.org/browse/SERVER-18227

http://www.eliothorowitz.com/blog/2015/09/11/document-validation-and-what-dynamic-schema-means/

https://docs.mongodb.org/manual/release-notes/3.2/#document-validation

Query Engine



CRUD Commands

- **find** command
- **getMore** command
- **killCursors** command
- **insert** command
- **update** command
- **delete** command
- other commands

Find command parameters

- find: <string>
- filter: <document>
- sort: <document>
- projection: <document>
- hint: <document|string>
- skip: <int64>
- limit: <int64>
- batchSize: <int64>
- singleBatch: <boolean>
- comment: <string>
- maxScan: <int32>
- maxTimeMS: <int32>
- min: <document>
- returnKey: <bool>
- showRecordId: <bool>
- snapshot: <bool>
- tailable: <bool>
- oplogReply: <bool>
- noCursorTimeout: <bool>
- awaitData: <bool>
- allowPartialResults: <bool>
- readConcern: <document>

Query Operators

Conditional Operators

$all, $exists, $mod, $ne, $in, $nin, $nor, $or, $size, $type

$lt, $lte, $gt, $gte

```
// find all documents that contain name field
> db.teams.find( {name: {$exists: true }} )

// find names using regular expressions
> db.names.find( {last: /^Nor*/i } )

// count people by city
> db.teams.find( {city: "Madrid"} ).count()
```

Write Operators

Field $set, $unset, $mul, $min, $max ,$inc, $currentDate, $setOnInsert

Array $push, $pop, $addToSet, $pull, $

Modifiers $each, $slice, $sort, $position

```
// Increment value by one
>db.events.update({_id:1}, {attendees: {$inc: 1}} )

// set array on insert
> db.names.update( {name: 'Bryan'}, { $setOnInsert: {talks:[0,0,0,0]}},
   {upsert:true} )

// unset a field
> db.names.update( {name: "Bryan"}, {$unset: {mug_member: "" }})
```

Indexes

```
// Index nested documents
> db.customers.ensureIndex( “policies.agent”:1 )
> db.customers.find({‘policies.agent’:’Fred’})


// geospatial index
> db.customers.ensureIndex( “property.location”: “2d” )
> db.customers.find( “property.location” : { $near : [22,42] } )

// text index
> db.customers.ensureIndex( “policies.notes”: “text” )
```

Partial Indexes

Go and try it out:

https://docs.mongodb.org/manual/release-notes/3.2/#partial-indexes

Client APIs

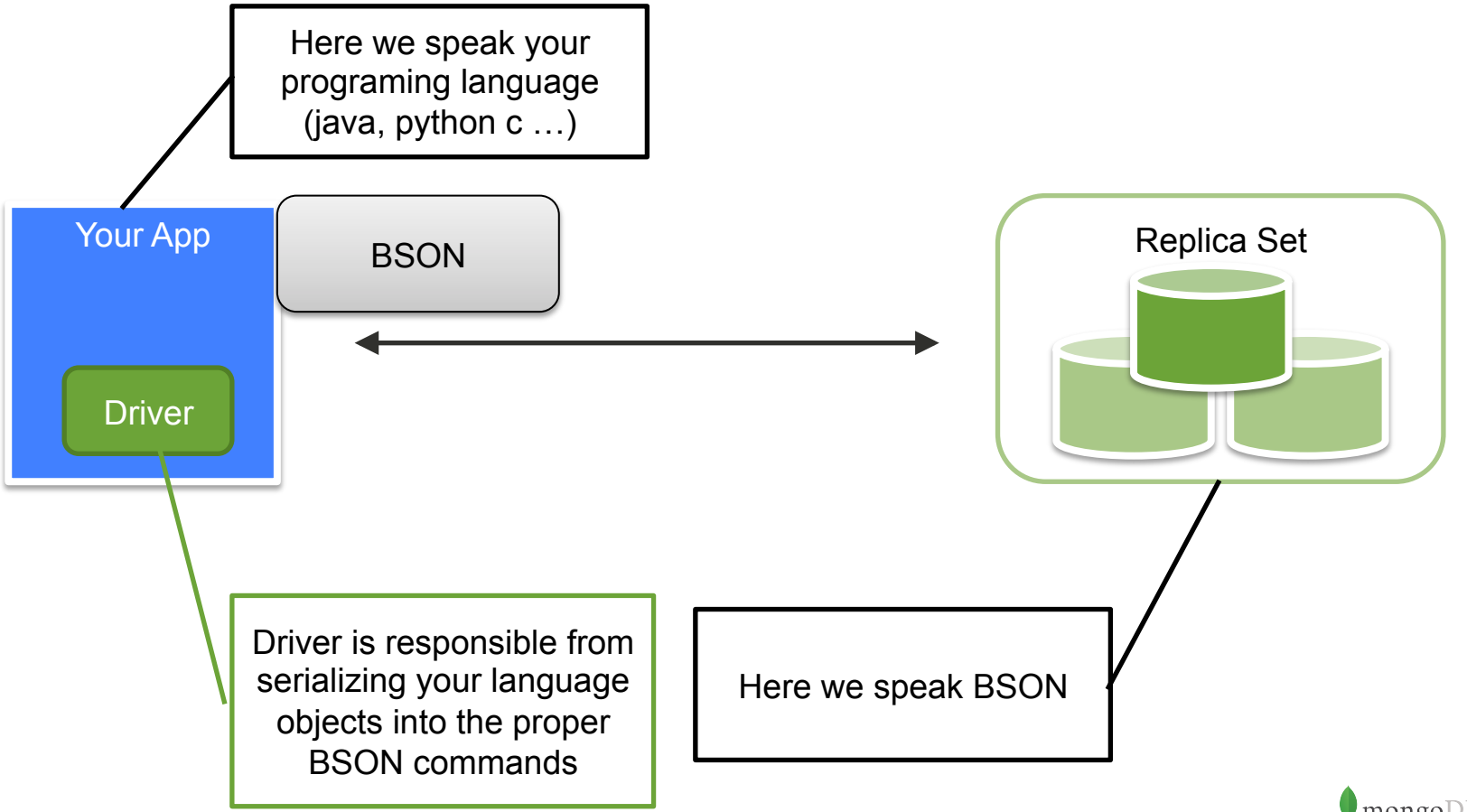
Client APIs

```java
//java maps
Document query = new Document("_id", "PSG");
Map<Object> m = collection.find(query).first();
Date established = (Date)m.get("established");
```

```python
#python dictionaries
query = {'_id': 'PSG'}
document = collection.find_one(query)
established = document['established'].year
```

```ruby
#ruby hashmaps
query = {:_id=> 'PSG'}
document = collection.find_one(query)
date = document[:established]
```

Client APIs


Here we speak your programing language (java, python c …)
Your App
Driver
BSON
Replica Set
Driver is responsible from serializing your language objects into the proper BSON commands
Here we speak BSON

Drivers & Ecosystem

Support for the most popular languages and frameworks


Java


Ruby


Python

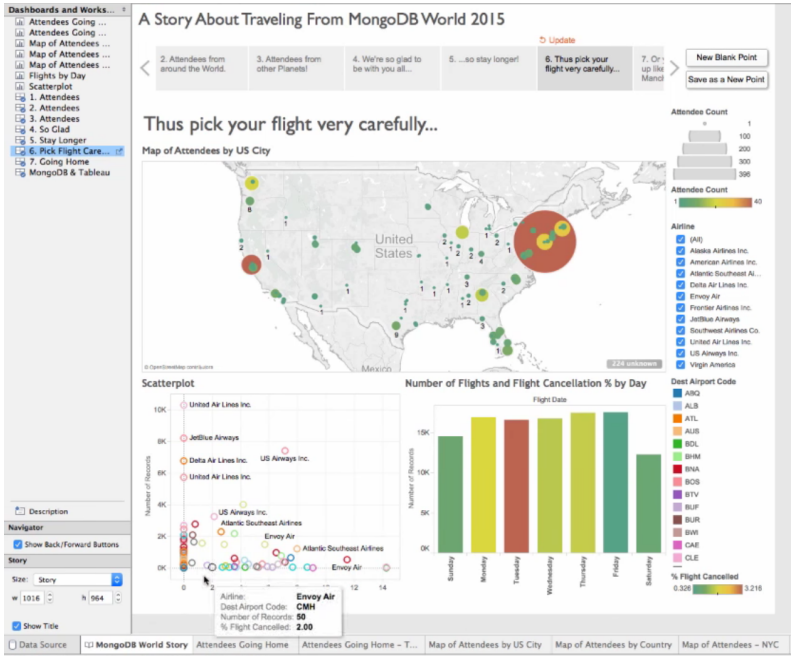

Perl

MEAN Stack

Morphia

Analytics & BI Integration

MongoDB Connector for BI

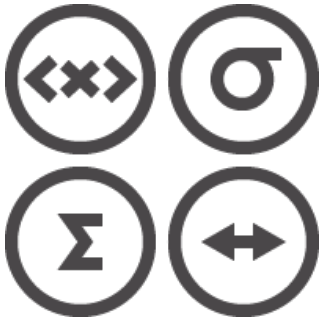


Visualize and explore multi-dimensional documents using SQL-based BI tools. The connector does the following:

- Provides the BI tool with the schema of the MongoDB collection to be visualized
- Translates SQL statements issued by the BI tool into equivalent MongoDB queries that are sent to MongoDB for processing
- Converts the results into the tabular format expected by the BI tool, which can then visualize the data based on user requirements

mongoDB.

Improved In-Database Analytics & Search



New Aggregation operators extend options for performing analytics and ensure that answers are delivered quickly and simply with lower developer complexity

- Array operators: $slice, $arrayElemAt, $concatArrays, $filter, $min, $max, $avg, $sum, and more
- New mathematical operators: $stdDevSamp, $stdDevPop, $sqrt, $abs, $trunc, $ceil, $floor, $log, $pow, $exp, and more
- Random sample of documents: $sample
- Case sensitive text search and support for additional languages such as Arabic, Farsi, Chinese, and more

Let's see some code!

What's Next?

- Download the Whitepaper
  - https://www.mongodb.com/collateral/mongodb-3-2-whats-new
- Read the Release Notes
  - https://docs.mongodb.org/manual/release-notes/3.2/
- Not yet ready for production but download and try!
  - https://www.mongodb.org/downloads#development
- Detailed blogs
  - https://www.mongodb.com/blog/
- Feedback
  - MongoDB 3.2 Bug Hunt
    - https://www.mongodb.com/blog/post/announcing-the-mongodb-3-2-bug-hunt
  - https://jira.mongodb.org/

DISCLAIMER: MongoDB's product plans are for informational purposes only. MongoDB's plans may change and you should not rely on them for delivery of a specific feature at a specific time.

Never Stop Learning

Join the Team

Sales & Account Management

Finance & People Operations

Pre-Sales Engineering

Engineering

Marketing

View all jobs and apply: http://grnh.se/pj10su

Obrigado!

Norberto Leite
Technical Evangelist
norberto@mongodb.com
@nleite