Repo

github.com/chaitanyavaranasi-mdb/AtlasWorkshop/tree/main

or

shorturl.at/ghEH9

Exercise 0

Create MongoDB cluster -> M0
(if not already done)

Add sample data

Enable Network Access

Create User w/ read & write access

shorturl.at/ghEH9

Mongo Exercise 1

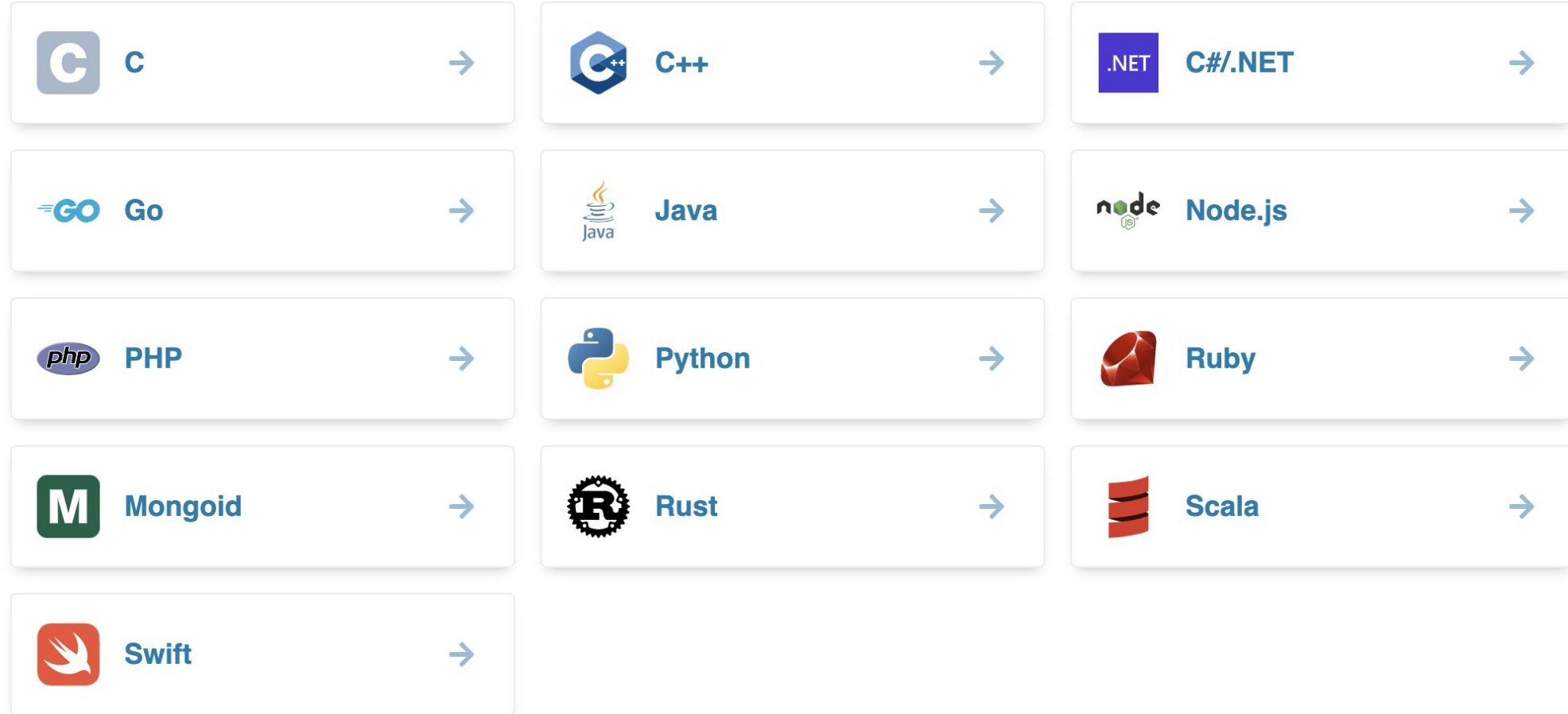Rich Queries - MQL

MongoDB Query API: rich and expressive

Expressive queries	• Find anyone with phone # "1-212…" • Check if the person with number "555…" is on the "do not call" list
Geospatial	• Find the best offer for the customer at geo coordinates of 42nd St. and 6th Ave
Text search	• Find all tweets that mention the firm within the last 2 days
Aggregation	• Count and sort number of customers by city, compute min, max, and average spend
Native binary JSON support	• Add an additional phone number to Mark Smith's record without rewriting the document • Update just 2 phone numbers out of 10 • Sort on the modified date
JOIN ($lookup)	• Query for all San Francisco residences, lookup their transactions, and sum the amount by person
Graph queries ($graphLookup)	• Query for all people within 3 degrees of separation from Mark

MongoDB

```
{
    customer_id : 1,
    first_name : "Mark",
    last_name : "Smith",
    city : "San Francisco",
    phones: [
        {
            number : "1-212-777-1212",
            type : "work"
        },
        {
            number : "1-212-777-1213",
            type : "cell"
        }
    ]
}
```

Intuitive: client drivers

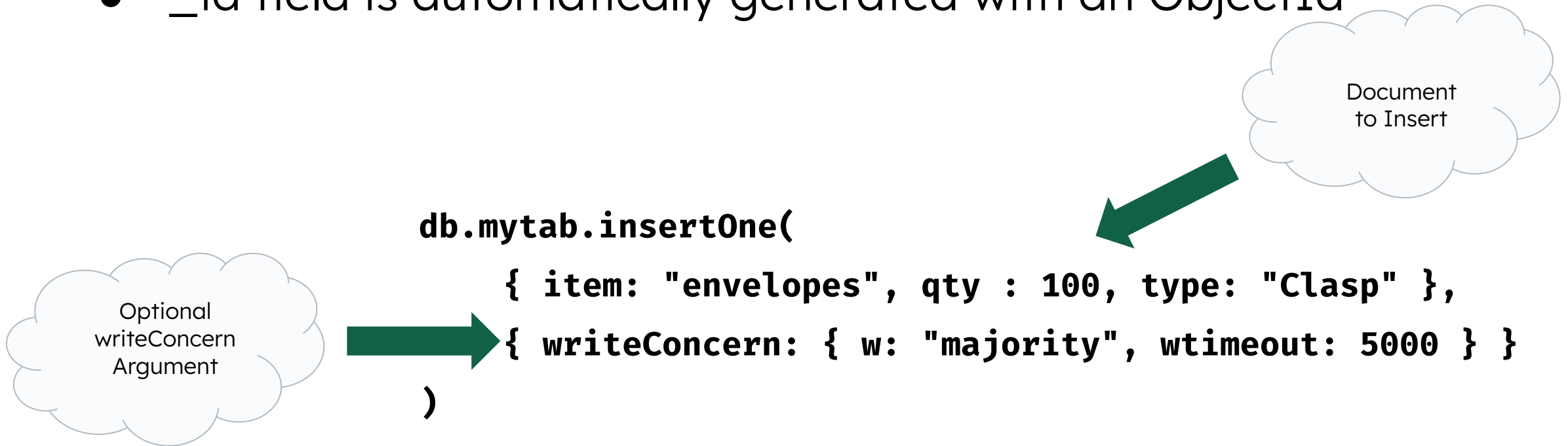|		
C →	C++ →	C#/.NET →
Go →	Java →	Node.js →
PHP →	Python →	Ruby →
Mongoid →	Rust →	Scala →
Swift →		

- Common CRUD capabilities but idiomatic to each language
- Uniform HA & Failover capabilities across all

insertOne()

- Inserts document(s) into a collection
- Can optionally define the writeConcern of the operation
- _id field is automatically generated with an ObjectId

Document to Insert

```
db.mytab.insertOne(
    { item: "envelopes", qty : 100, type: "Clasp" },
    { writeConcern: { w: "majority", wtimeout: 5000 } }
)
```

Optional writeConcern Argument

find() and findOne()

- Get documents from the collection (table)
- Can filter by content (query) and by what is returned (project)
- `findOne()` returns just the first document

`db.mytab.find({x:1, y:2}, {a:1, b:1, c:1})`

Document to Query

Fields to Project

updateMany() and updateOne()

- Modifies existing document(s) in a collection
- Use with a field update operator (MQL)
- Query for the documents you want to update
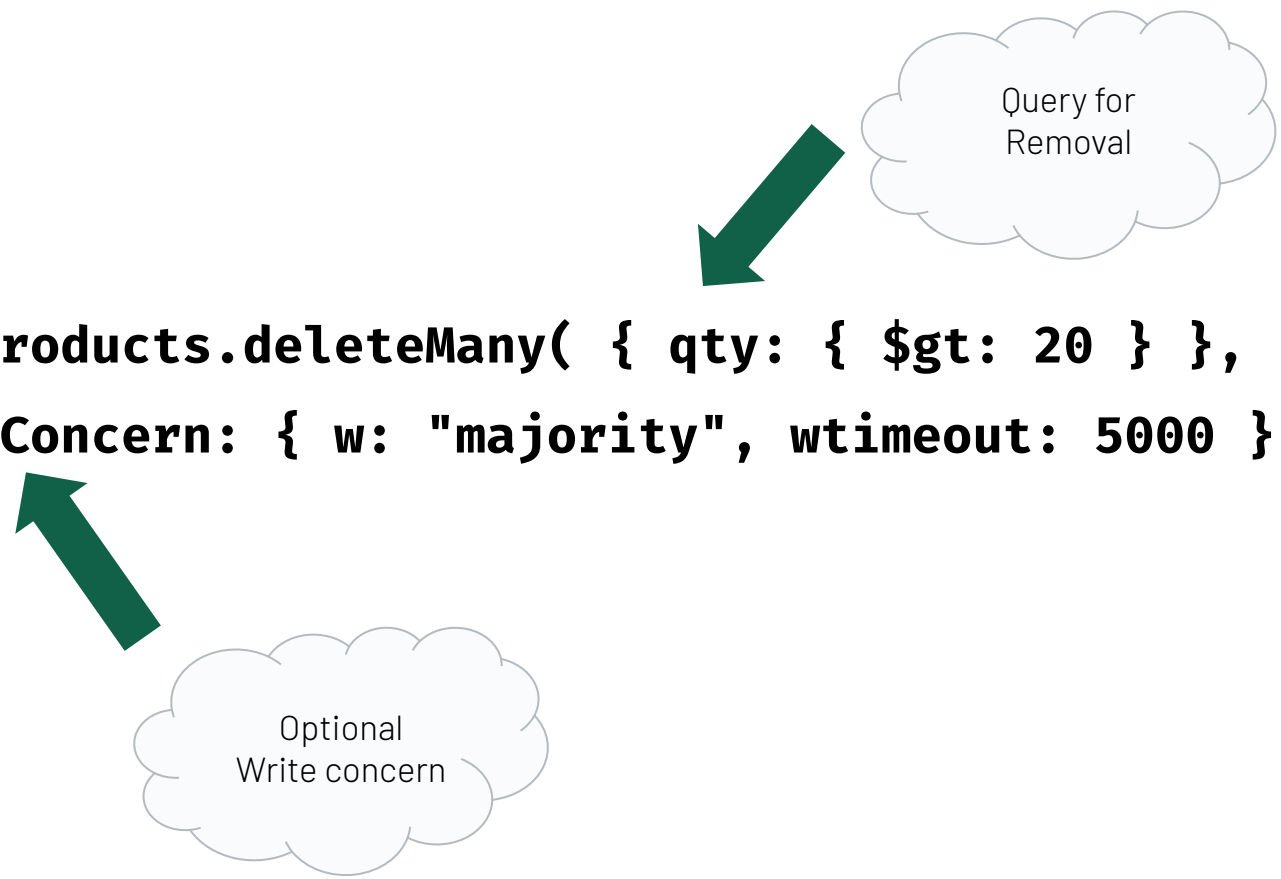- Can optionally upsert, multi-update, define writeConcern, etc.

```
db.books.updateOne(
    { author: "Tom Gleitsmann" },
    { $set: { author: "Thomas Gleitsmann" }},
  { multi: true }
  )
```

Document
to Query

Fields to
set

Optional
3rd Argument
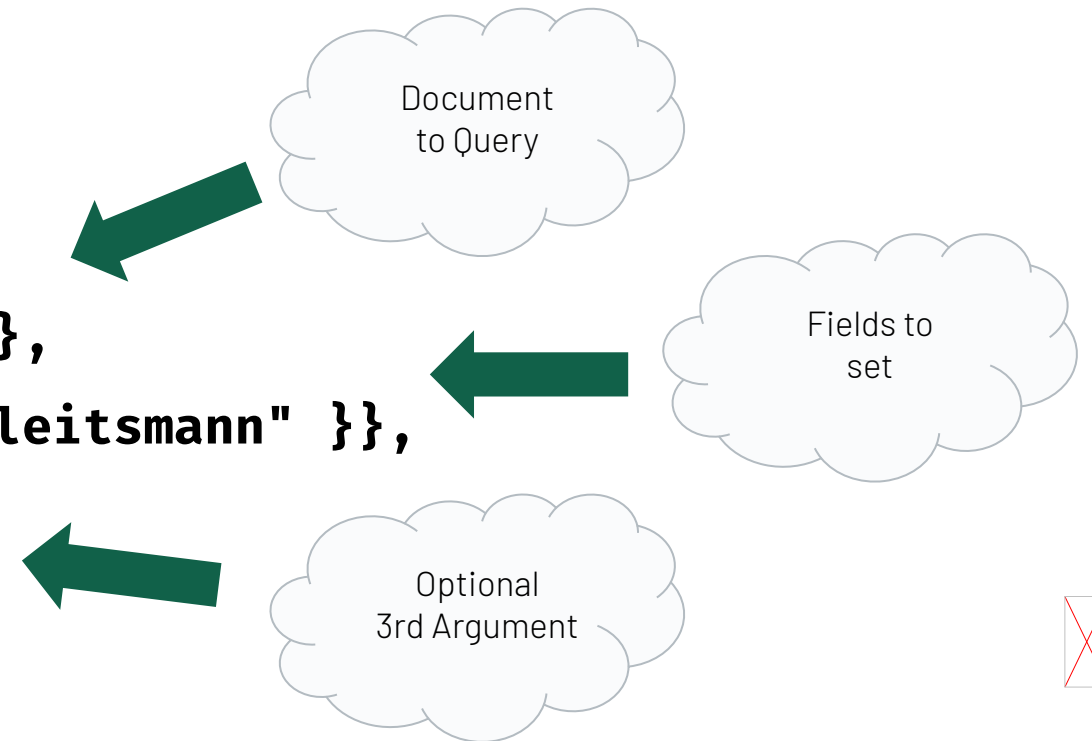
deleteMany()

- Deletes a document from the collection
- Takes an optional writeConcern argument

Query for
Removal

```
db.products.deleteMany( { qty: { $gt: 20 } },
{ writeConcern: { w: "majority", wtimeout: 5000 } )
```

Optional
Write concern

Comparison Query Operators

$lt   :   Exists and less than

$lte   :   Exists and less than or equal to

$gt   :   Exists and greater than

$gte   :   Exists and greater than or equal to

$ne   :   Does not exist or does but not equal to

$in   :   Exists and in a set

$nin   :   Does not exist or not in a set

```
{ age : { $gte : 21 }}
{ temp : { $gt: 10, $lt: 30 }}
```

Field Update Operators (well some…)

$currentDate :   Sets the value of a field to current date

$inc   :   Increments value

$min   :   Update if field is less than $min value

$max   :   Update if field is greater than $max value

$mul   :   Multiply field value

$set   :   Set value of field

$unset   :   Removes field from document

```
{ $set : { bestDB : "MongoDB"}}
{ $mul: { amount : 5, ... } }
```

Hands-On: Compass

sample_airbnb.listingsAndReviews

5.6k DOCUMENTS **4** INDEXES

Documents Aggregations Schema Explain Plan Indexes Validation

Filter 🔗 🕐 ▾ `{"address.market":"Hong Kong", "property_type" : "Apartment"}` Reset **Find** `</>` More Options ▶

⬇ ADD DATA ▾ 📤 EXPORT COLLECTION 1 – 20 of 444 ↻ ‹ › ☰ {} ⊞

```
_id: "10893326"
listing_url: "https://www.airbnb.com/rooms/10893326"
name: "Designer Apartment in Sheung Wan"
summary: "We live in a beautiful and spacious one bedroom apartment in the heart…"
space: ""
description: "We live in a beautiful and spacious one bedroom apartment in the heart…"
neighborhood_overview: "Often described as the new SOHO, there are chic coffee shops and inter…"
notes: ""
transit: "The area is well served by minibuses, City Bus, taxis, and the old sty…"
access: ""
interaction: ""
house_rules: ""
property_type: "Apartment"
room_type: "Entire home/apt"
bed_type: "Real Bed"
minimum_nights: "1"
maximum_nights: "1125"
cancellation_policy: "strict_14_with_grace_period"
last_scraped: 2019-03-11T04:00:00.000+00:00
calendar_last_scraped: 2019-03-11T04:00:00.000+00:00
first_review: 2016-03-06T05:00:00.000+00:00
last_review: 2019-02-11T05:00:00.000+00:00
accommodates: 2
```

```
use sample_airbnb
db.listingsandreviews.find({"address.market":"Hong Kong","property_type" : "Apartment"})
```

sample_airbnb.listingsAndReviews

Documents   Aggregations   Schema   Explain Plan   Indexes   Validation

Filter ⧉ 🕐 ▾   { "amenities": { "$all": ["Wifi", "Kitchen"]}}   Reset **Find** </> More Options ▸

⬇ ADD DATA ▾   ⬈ EXPORT COLLECTION   1 – 20 of 4767 ⟳ ‹ ›   ☰ {} ⊞

```
_id: "10047964"
listing_url: "https://www.airbnb.com/rooms/10047964"
name: "Charming Flat in Downtown Moda"
summary: "Fully furnished 3+1 flat decorated with vintage style.  Located at the…"
space: "The apartment is composed of 1 big bedroom with double sized bed, a gu…"
description: "Fully furnished 3+1 flat decorated with vintage style.  Located at the…"
neighborhood_overview: "With its diversity Moda- Kadikoy is one of the most colorfull neighbou…"
notes: ""
transit: ""
access: ""
interaction: ""
house_rules: "Be and feel like your own home, with total respect and love..this woul…"
property_type: "House"
room_type: "Entire home/apt"
bed_type: "Real Bed"
minimum_nights: "2"
maximum_nights: "1125"
cancellation_policy: "flexible"
last_scraped: 2019-02-18T05:00:00.000+00:00
calendar_last_scraped: 2019-02-18T05:00:00.000+00:00
first_review: 2016-04-02T04:00:00.000+00:00
last_review: 2016-04-02T04:00:00.000+00:00
accommodates: 6
```

```
use sample_airbnb
db.listingsandreviews.find({
"amenities": { "$all": ["Wifi",
"Kitchen"]}})
```

sample_airbnb.listingsAndReviews

5.6k DOCUMENTS **4** INDEXES

Documents   Aggregations   Schema   **Explain Plan**   Indexes   Validation

Filter ⎘ 🕐 ▾   {"address.market":"Hong Kong", "property_type" : "Apartment", "bedrooms": {"$gte": 3}, '   [Reset]   [**Explain**]   [</>]   More Options ▶

VIEW   **VISUAL TREE**   RAW JSON

Query Performance Summary   Learn more ⎘

Documents Returned: **4**

Index Keys Examined: **3626**

Documents Examined: **3626**

Actual Query Execution Time (ms): **14**

Sorted in Memory: **no**

Query used the following index: [PROPERTY_TYPE ↑] [ROOM_TYPE ↑] [BEDS ↑]
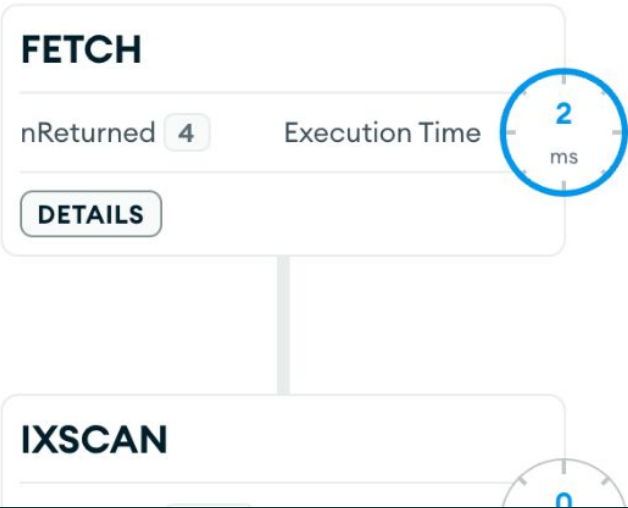
FETCH

nReturned **4**   Execution Time   **2** ms

[DETAILS]

IXSCAN

0

```
use sample_airbnb
db.listingsandreviews.find({"address.mar
ket":"Hong Kong", "property_type" :
"Apartment", "bedrooms": {"$gte": 3},
"price": {"$gte": 1400, "$lte": 1500},
"amenities": { "$all": ["Wifi",
"Kitchen"]}}).explain()
```

shorturl.at/ghEH9

5 min break

Mongo Exercise 2

Aggregation Framework

Aggregations 101

- Until now, we have only filtered data
  - What Documents
  - What Fields
  - What Order
- Aggregation allows us to compute new data
  - Calculated fields
  - Summarised and grouped values
  - Reshape documents

Aggregation -> Pipeline

- Each transformation is a single step known as a stage.
- Compared to one huge SQL style statement this is
  - Easier to understand
  - Easier to debug
  - Easier for MongoDB to rewrite and optimize

Aggregation pipelines

$match → $group → $project → $merge

Allows developers to extend MongoDB functionality by creating their own expressions and have them execute as part of an aggregation pipeline

Aggregation pipelines

```
                          Collection
{
  cust_id: "A123",                |
  amount:500,
  status: "A",          db.orders.aggregate([
}
                           $match stage  ────────▶  {$match: {status: "A"}},
                           $group stage  ────────▶  {$group:{_id:"cust_id",total:{$sum: "$amount"}}}
{                        ])
  cust_id: "A123",
  amount:250,
  status: "A",
}                                          {
                                             cust_id: "A123",
                                             amount:500,
{                                          } status: "A",
  cust_id: "B212",
  amount:200,                              {                              {
  status: "A",          $match               cust_id: "A123",   $group        id: "A123",
}                                            amount:250,                       total:750
                                             status: "A",
                                           }                              }
{
  cust_id: "A123",                         {                              {
  amount:300,                                cust_id: "B212",
  status: "D",                               amount:200,                     id:"B212",
}                                            status: "A",                    Total: 200
                                           }                              }

    Orders
```

Stages we know

- $match   equivalent to find(query)
- $project   equivalent to find({},projection)
- $sort   equivalent to find().sort(order)
- $limit   equivalent to find().limit(num)
- $skip   equivalent to find().skip(num)
- $count   equivalent to find().count()

When these are used at the start of a pipeline, they are transformed to a find() by the query optimizer.

Comparing Aggregation Syntax

Find the name of the host in Canada with the most "total listings":

Using find()

```
db.listingsAndReviews.find(
    {"address.country":"Canada"},
    {"_id":0, "host.host_total_listings_count":1, "host.host_name":1}
).sort({"host.host_total_listings_count":-1}).limit(1)
```

Using aggregate()

```
db.listingsAndReviews.aggregate([
    {$match:{"address.country":"Canada"}},
    {$sort: {"host.host_total_listings_count":-1 }},
    {$limit:1},
    {$project:{"_id":0, "host.host_total_listings_count":1, "host.host_name":1}}
])
```

Dollar Overloading

```
{$match: {a: 5}}
```

Dollar on left means a stage name - in this case a **$match** stage

```
{$set: {b: "$a"}}
```

 Dollar on right of colon "**$a**" refers to the value of field a

```
{$set: {area: {$multiply: [5,10]}}
```

$multiply is an expression name left of colon

```
$map:{input: "$quizzes", as: "grade",in: { $add: [ "$$grade", 2 ] }}}
```

$$grade refers to a variable in $map statements

Think as a programmer - not as a database shell.

Define variables. Doing this helps you keep track of brackets.

It also helps you really understand the Object concept.

```
//Do it THIS way for ease of testing and debugging


> no_celebs = {$match:{"user.followers_count":{$lt:200000}}}


> name_only = {$project:{"user.name":1, "user.folowers_count":1,_id:0}}


> most_popular = {$sort: {"user.followers_count":-1}}


> first_in_list = {$limit:1}


> pipeline = [no_celebs,name_only,most_popular,first_in_list]


> db.twitter.aggregate(pipeline)
```

shorturl.at/ghEH9

Hands-On: MongoDB Shell

Exercise 1

Using the Shell or Compass/Atlas Aggregation Builder and the Airbnb listings data (`sample_airbnb.listingsAndReviews`) read the following question and output the answer. You will need to use a `$set` stage and a `$project` stage with expressions.

To keep in mind...

1. The price for the basic rental is in the **$price** field; for that price, you can have the number of guests provided in **$guests_included** field.
2. The property may take more guests in total (maximum guests is in **$accommodates** property).
3. You have to pay **$extra_people** dollars per person for every person more than **$guests_included**

Requirement:

- Calculate how many extra guests you can have for each property and add that as a field using `$set`
- Calculate how much it would cost with these extra guests. `$project` the basic price and the price if fully occupied with `$accommodates` people.

ANSWER: Exercise 1

Using the Airbnb listings data:

- Calculate how much it would cost for these extra guests - they cost "$extra_people" each and $project the basic price and the price if fully populated

```
addextra = { $set: { numguestsextra : { $subtract:
["$accommodates","$guests_included"]}}}

db.listingsAndReviews.aggregate([addextra]).pretty()

extraguestcost = { $multiply : ["$extra_people","$numguestsextra"]}

finaloutput = { $project: { price: 1 , maxprice: { $add: [
"$price",extraguestcost]}}}

db.listingsAndReviews.aggregate([addextra,finaloutput]).pretty()
```

Exercise 2

Calculate how many Airbnb properties there are in each country, ordered by count, list the one with the largest count first. You should continue working with namespace `sample_airbnb.listingsAndReviews`.

- **Hint:** to count things, you add the explicit value 1 to an accumulator using $sum

ANSWER: Exercise 2

Calculate how many Airbnb properties there are in each country, ordered by count, list the one with the largest count first.

```
groupfield = "$address.country"
groupstage = { $group: { _id: groupfield, count:{$sum:1}}}
sortstage = {$sort:{count:-1}}
pipe = [groupstage,sortstage]
db.listingsAndReviews.aggregate(pipe).pretty()
```

or

```
db.listingsAndReviews.aggregate([{$sortByCount:"$address.country"}]).pretty()
```

Exercise 3

What are the top 10 most popular amenities across all listings?

ANSWER: Exercise 3

What are the top 10 most popular amenities across all listings?

```
unwindstage = { $unwind:"$amenities"}
group = { $group: { _id: "$amenities", count: { $sum: 1 } } }
sortstage = {$sort:{count:-1}}
limitstage = { $limit : 10 }
pipe=[unwindstage,group,sortstage, limitstage]
db.listingsAndReviews.aggregate( pipe ).pretty()
```

shorturl.at/ghEH9

5 min break

shorturl.at/ghEH9

Search

What is Search?

Search, or "Full-Text Search" is the ability to search across all of your data and efficiently return a list of results, ranked based on how well they matched to the search term

Usually defined as a Search bar -------->

🔍 …

Not an exact match, but a **LIST OF RESULTS BASED ON RELEVANCE**

Results based on a score

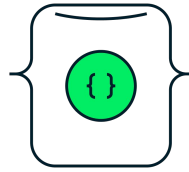**Result 1**

Result 2

Result 3

Result 4

How developers build app search today

Stand up a database to provide the application's persistence layer

Database

Operational database serving
pre-defined queries

CRUD

Change Streams

Aggregations

Secondary Indexes

ACID Transactions

...

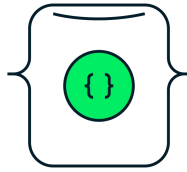Check my account balance

View my purchase history

View my wish list

How developers build app search today

Bolt-on a search engine to power search across application data

Database

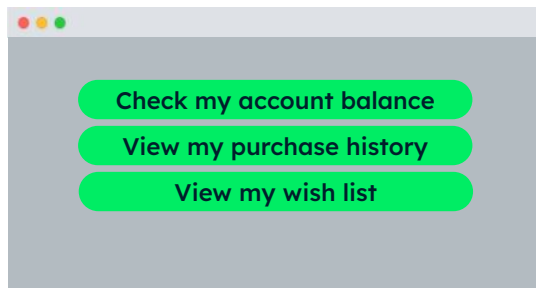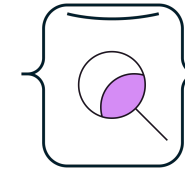Operational database serving pre-defined queries

- CRUD
- Change Streams
- Aggregations
- Secondary Indexes
- ACID Transactions
- ...

- Check my account balance
- View my purchase history
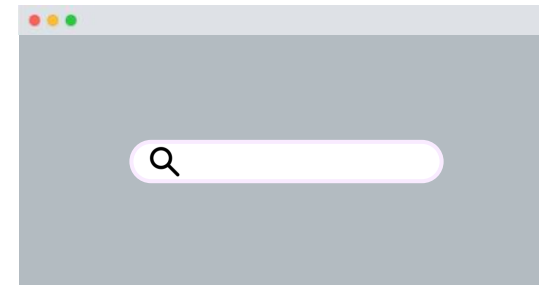- View my wish list

Search Engine

elasticsearch
Lucene
Solr

Search engine inferring intent from free form, natural language queries

- Fuzzy matching
- Autocomplete
- Facets and counts
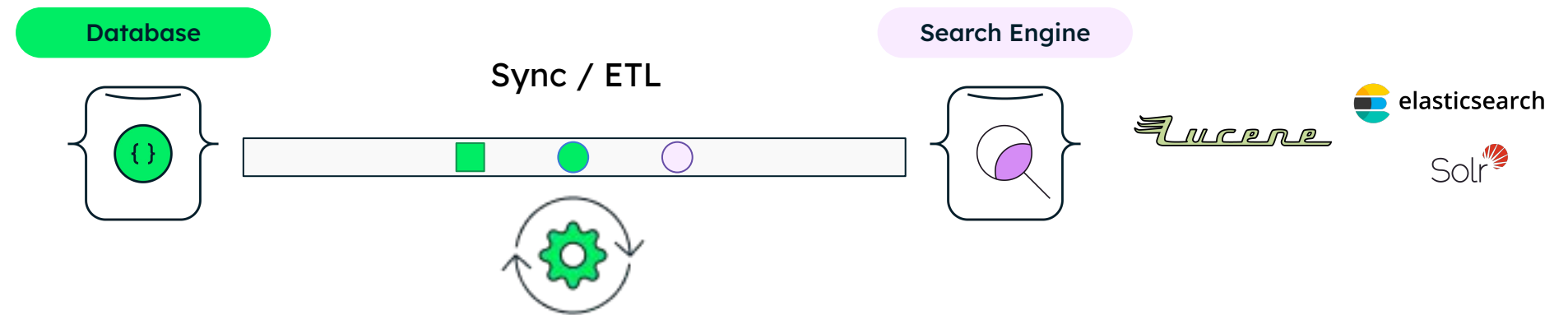- Synonyms
- Scoring / Weighting
- moreLikeThis

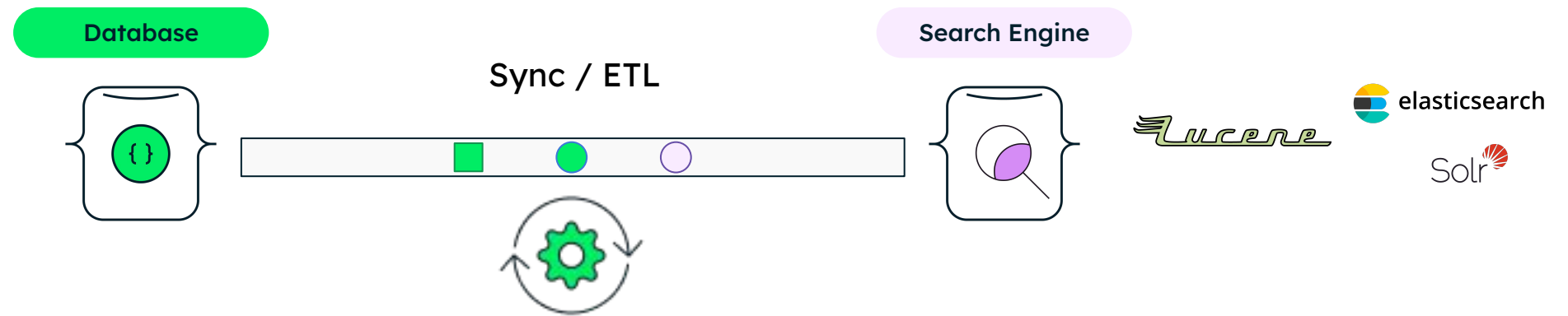# How developers build app search today

Stand up a replication mechanism to keep the systems in sync

How developers build app search today

Architectural complexity in our application stack

Database

Sync / ETL

Search Engine

elasticsearch

lucene

Solr

❌ **Developer friction**
Different query APIs and drivers for database and search, coordinate schema changes
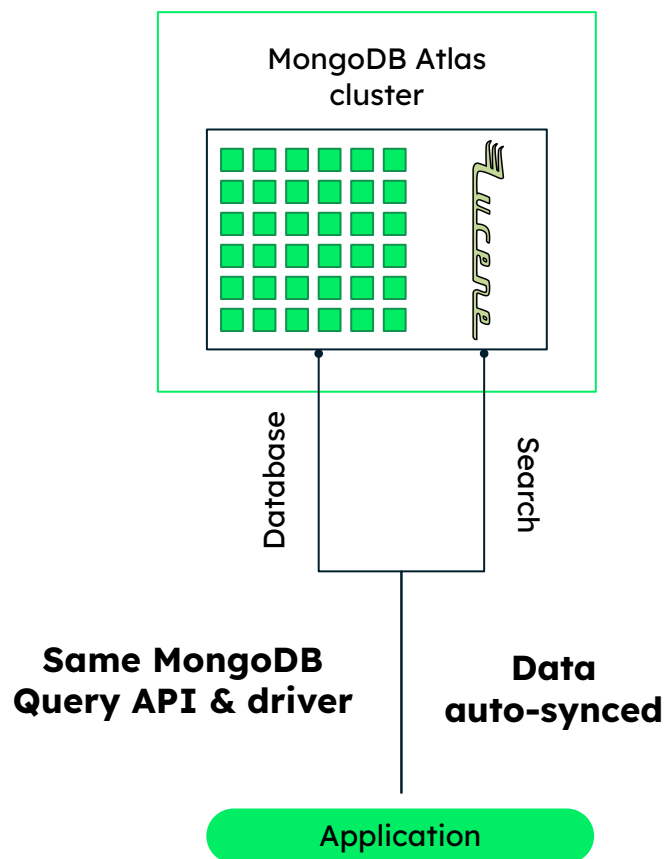
❌ **Pay the sync tax**
Requires its own systems and skills. Recovering sync errors can consume 10% of a developer's time

❌ **Operational overhead**
More to provision, secure, upgrade, patch, back up, monitor, scale, etc.

How does Atlas Search address that pain?

MongoDB
Atlas

MongoDB Atlas
cluster

Database

Search

**Same MongoDB
Query API & driver**

**Data
auto-synced**

Application

Embeds a fully managed Apache Lucene index right alongside the database

✅ **Improved developer productivity**

Build database and search features using the same query API and driver

✅ **Simplified data architecture**

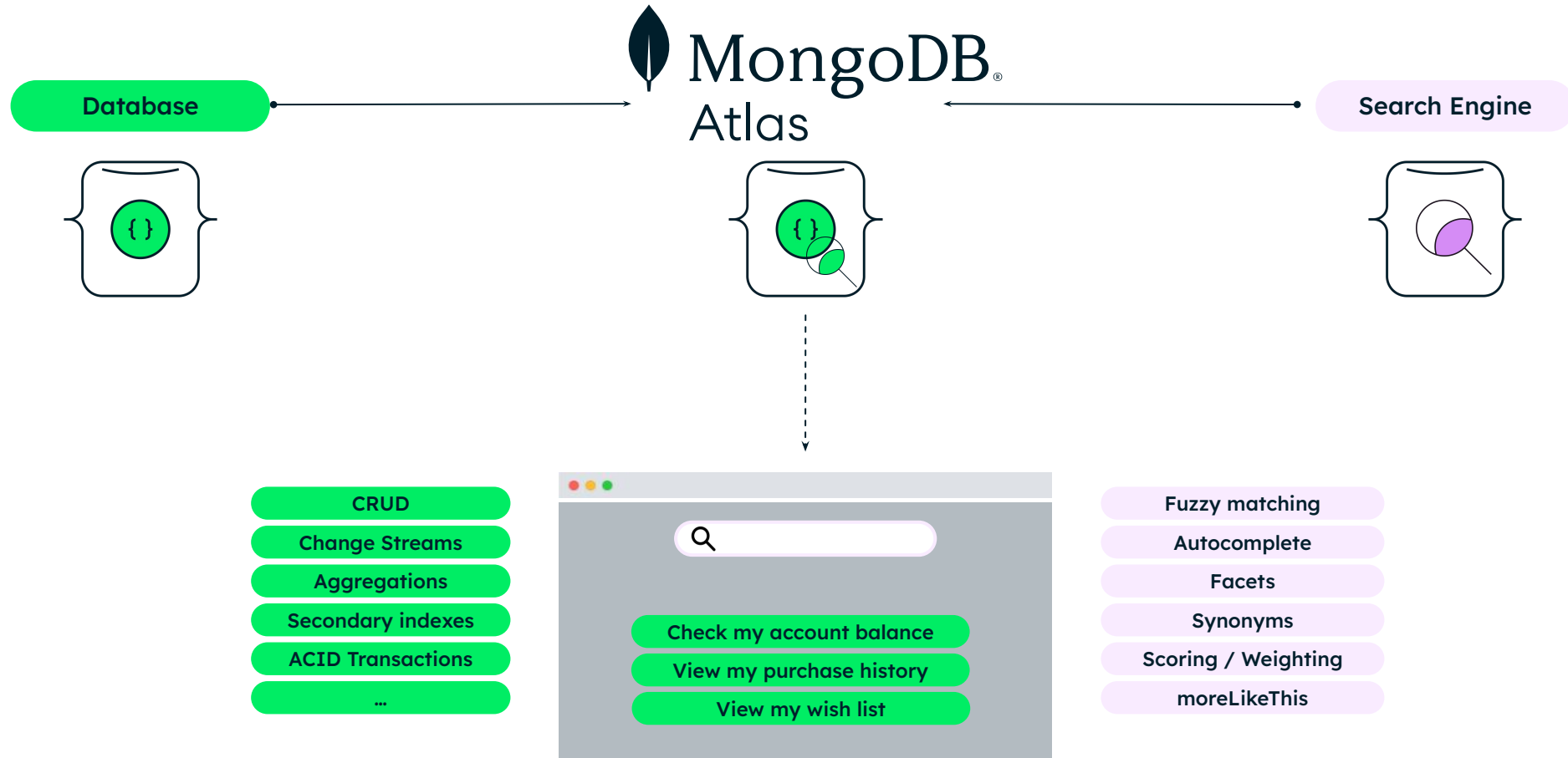Automatic data synchronization, even as your data and schema changes

✅ **Fully managed platform**

Get the security, performance, and reliability of Atlas

Unifying database and search

Compress 3 systems into 1, ship search features **30%-50% faster**

MongoDB. Atlas

Database → ← Search Engine

CRUD
Change Streams
Aggregations
Secondary indexes
ACID Transactions
...

Check my account balance
View my purchase history
View my wish list

Fuzzy matching
Autocomplete
Facets
Synonyms
Scoring / Weighting
moreLikeThis

Atlas Search architecture

Atlas Node

mongod

MongoDB

mongot

API
API

Lucene

Initial sync & change streams

objectID + metadata

Configure Indexes

Start & stop process

MongoDB Atlas Automation

Indexes

Data
Fields mapped onto ObjectIDs

App

$search agg query

Search query results

User

Search index configuration
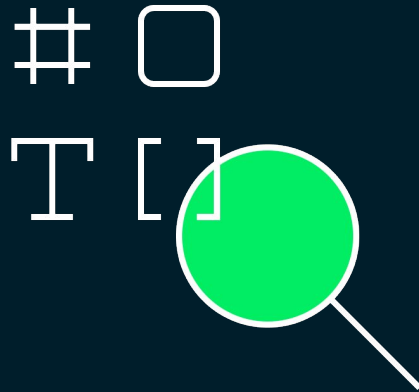
Atlas UI
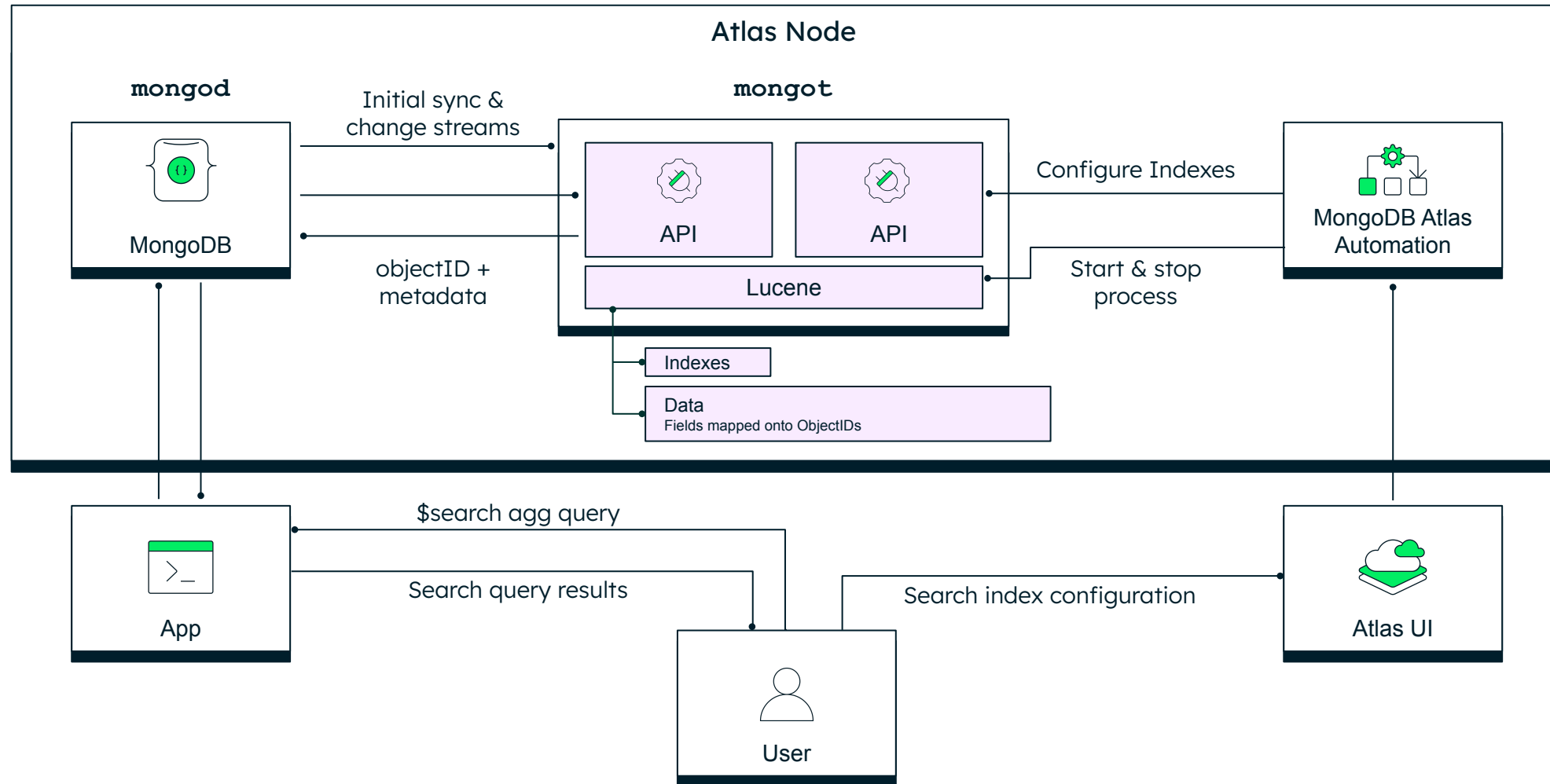
Rich search features

Atlas Search offers the features needed to fine-tune your search results to help users find what they need, including:

- Fuzzy search
- Autocomplete
- Filters
- Synonyms
- Facets
- Multi-data type support
- Multi-language support
- Highlighting
- Custom scoring
- Rich query language
- More Like This

Analyzers

We start with an original query

Lions and tigers and bears, oh my!

Analyzers apply parsing and language rules to the query

Tokens are the individual terms that are queried; think of these as the words that matter in a search query

Analyzers

We start with an original query

Analyzers apply parsing and language rules to the query

Tokens are the individual terms that are queried; think of these as the words that matter in a search query

Lions and tigers and bears, oh my!

~~L~~ lions ~~and~~ tigers ~~and~~ bears~~,~~ oh my~~!~~

lions   tigers   bears   oh   my

Analyzers

Analyzers also support language rules

A search for...

...can return results for related words

"bank"

["bank", "banks", "banker", "banking", ...]

Other supported search features

Search features are all about helping customers make their data more **discoverable** and **relevant** for end users

- **Support for 30+ languages**

- Autocomplete

- Highlighting

- Fuzzy matching (typo tolerance)

- Faceting

- Multiple data types

- Synonyms

- Custom scoring

- Rich query language

- More like this

lucene.arabic	lucene.armenian	lucene.basque	lucene.bengali	lucene.brazilian
lucene.bulgarian	lucene.catalan	lucene.cjk	lucene.czech	lucene.danish
lucene.dutch	lucene.english	lucene.finnish	lucene.french	lucene.galician
lucene.german	lucene.greek	lucene.hindi	lucene.hungarian	lucene.indonesian
lucene.irish	lucene.italian	lucene.latvian	lucene.lithuanian	lucene.norwegian
lucene.persian	lucene.portuguese	lucene.romanian	lucene.russian	lucene.sorani
lucene.spanish	lucene.swedish	lucene.turkish	lucene.thai	

Each language analyzer has built-in rules based on the language's usage pattern

How do I search data in atlas?

Create Atlas Cluster

Database Collections

Create Search Index

Query via $search
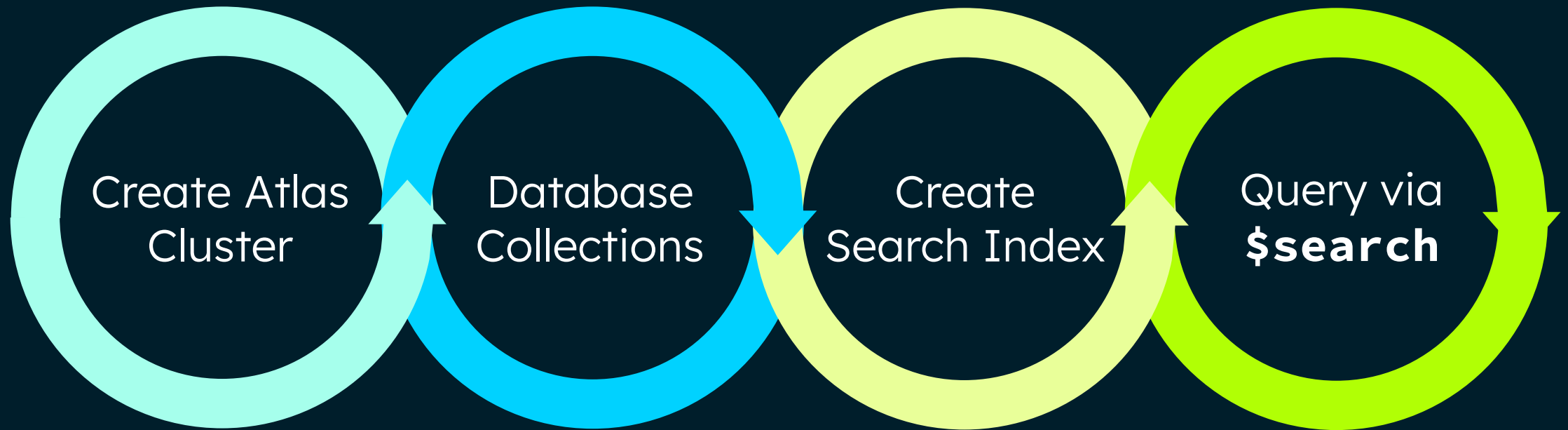
shorturl.at/xHX78

Search Exercise 1

Configuring Search Indexes
1. Default Search Index
2. Autocomplete on Title Index

Query lifecycle

MongoDB Atlas Host

app

```
db.col.aggregate([
{$search: {
    text: {
      path: "name",
      query: "star wars"
    }
 }}])



[ { _id: "123", title:
"Star Wars"}, {…}]
```

Wire Protocol
(over internet to
MongoDB Atlas)

mongod

```
aggregate([ {$search: {
    text: {
      path: "name",
      query: "star wars"
    }
 }}]
```

```
Lookup([{_id: "123"],
{…}])
```

```
[ { _id: "123",score:
1.23,highlights: […] },
{…}]
```

Wire protocol
(localhost)

mongot

```
text: {
    path: "name",
    query: "star wars"
}
```

```
Lucene booleanQuery:
 (should
   (term("name", "star"),
   term("name", wars"))
```

```
{ "name" :
   { "star" : [123,124],
    "wars" :
[123,125,…]}}
```

shorturl.at/ghEH9

Search Exercise 2

Our first Search aggregations in Atlas UI
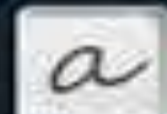
Self Serve Exercise 3 (Optional)

Integrate your endpoints into a web app

Q bollywood

Hollywood Shuffle

The Guru

Corman's World: Exploits of a Hollywood Rebel

Bollywood/Hollywood

Score: 7.507

Score: 7.221

Score: 7.068

Score: 6.904

Year: 1987

Year: 2002

Year: 2011
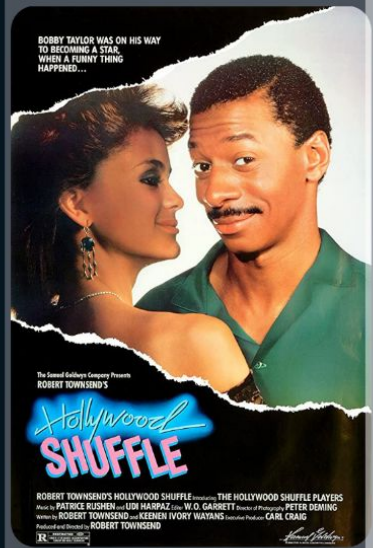
Year: 2002

Rating: 7

Rating: 5.4

Rating: 7.6

Rating: 6

He just has to convince **Hollywood** that
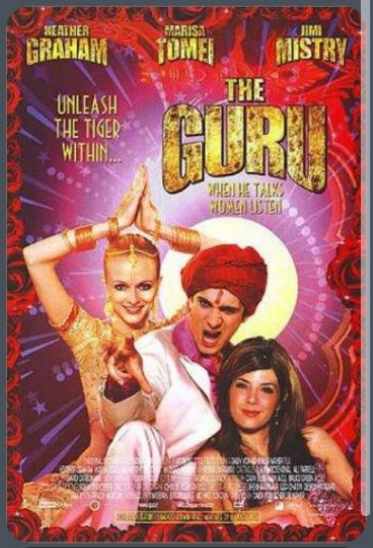
Bored with **Bollywood** movies but
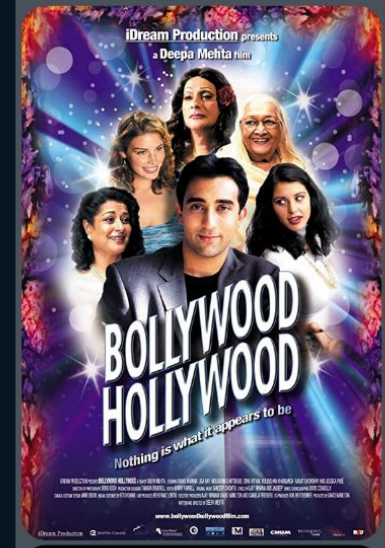
Self Serve Demo 4

Search feature tour

atlassearchrestaurants.com

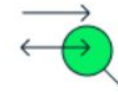github.com/mongodb-developer/WhatsCooking
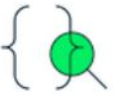
MongoDB

Aggregation

Function Score 💯

Synonyms

Data & Indexes

Atlas Search Demo: Restaurant Finder

What's Cooking

restaurants...

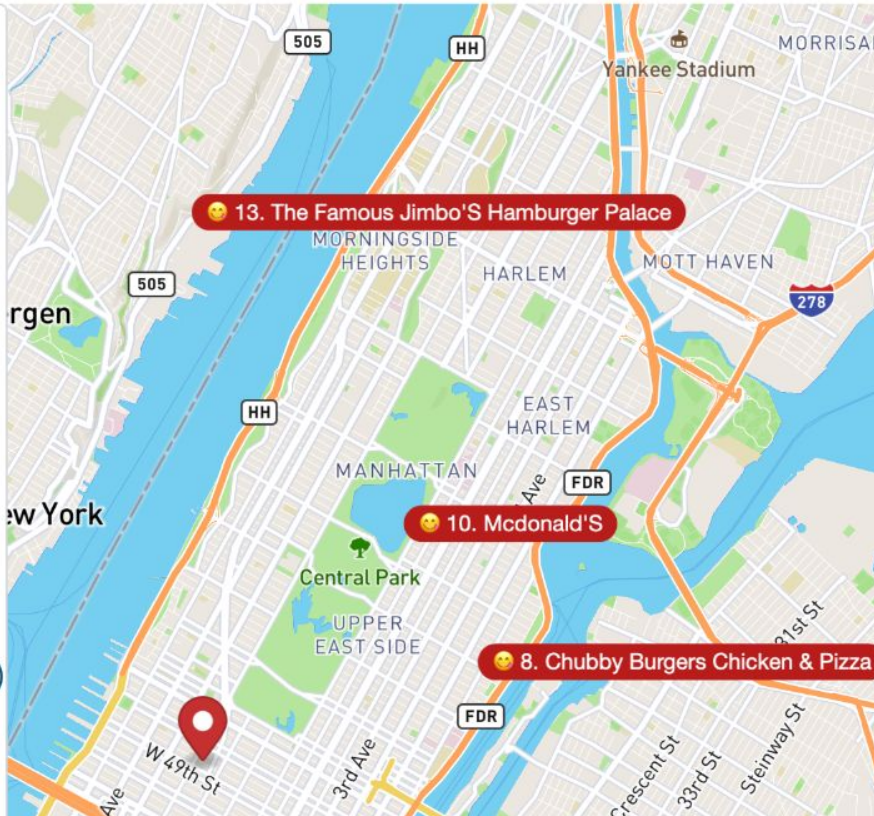burger

Find

$search

Geospatial Search Options

near | geoWithin

Average Star Rating:

⭐★★★★

Overall Count: 5813

☐ American (5380)
☐ Chinese (0)
☐ French (0)
☐ Hamburgers (433)
☐ Italian (0)
☐ Japanese (0)

13. The Famous Jimbo'S Hamburger Palace

10. Mcdonald'S

8. Chubby Burgers Chicken & Pizza

score: 1.8

White Castle

Hamburgers
Myrtle
Avenue
Brooklyn

⭐⭐⭐⯨ 💲 131 reviews
★

Show Menu 👓

score: 1.8

Mcdonald'S

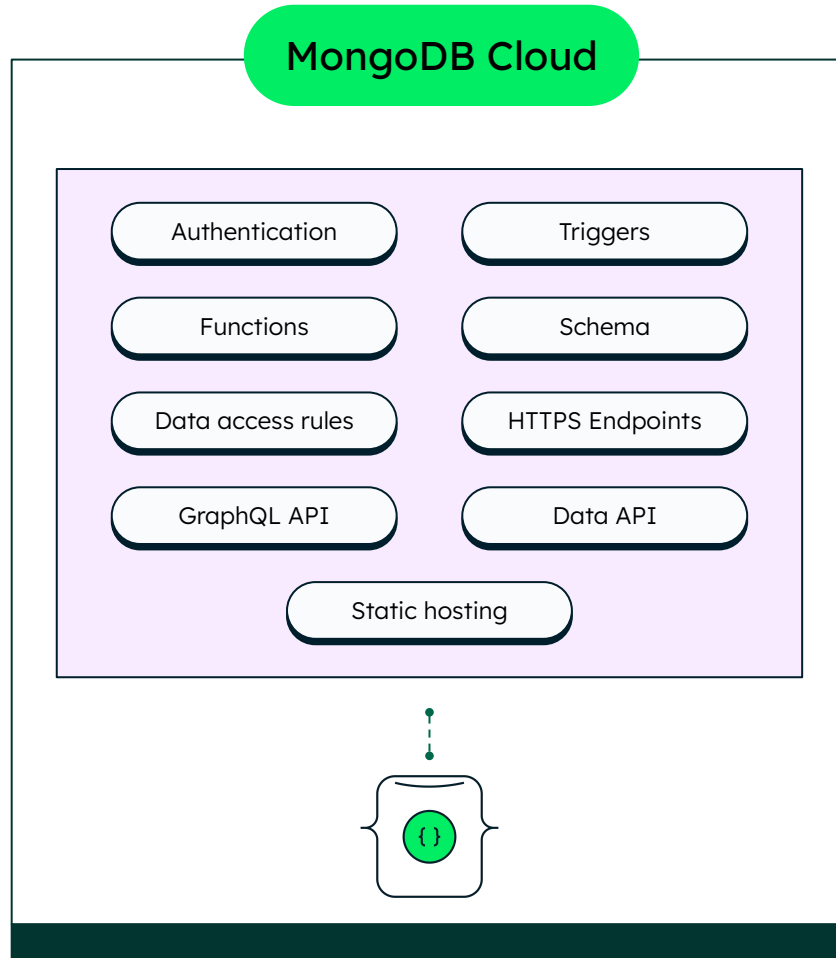Hamburgers
Atlantic
Avenue
Brooklyn

```
{ $search :

// optional, defaults to "default"

index: < indexName >

{
  "text": {
    "query": "burger",
    "path": "menu",
    "synonyms": "MenuSynonyms"
  }
}

highlight :
```

What is App Services?

MongoDB Cloud

Authentication

Triggers

Functions

Schema

Data access rules

HTTPS Endpoints

GraphQL API

Data API

Static hosting

{}

Part of MongoDB Cloud

Developer services on top the core MongoDB Atlas Database
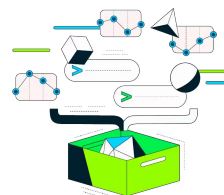
Fully managed serverless architecture

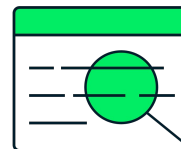Consumption based pricing

Vector Search
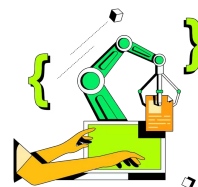
PUBLIC PREVIEW

Atlas Vector Search

Store vector embeddings right next to your source data and metadata. Vectors inserted or updated in the database are automatically synchronized to the vector index

Query the database and Vector Search via the unified MongoDB Query API, providing developers with a consistent experience

Remove operational heavy lifting with the battle tested, fully managed Atlas platform

Vector Search powers a number of key use cases

Semantic Search

Similarity Search

Recommendation Engines

Long-term memory for LLMs
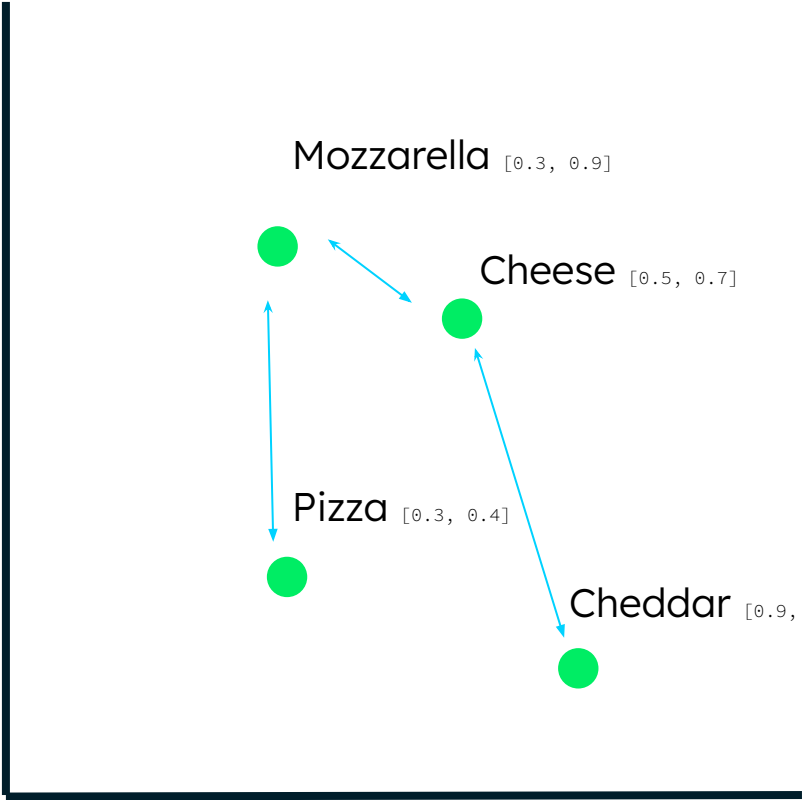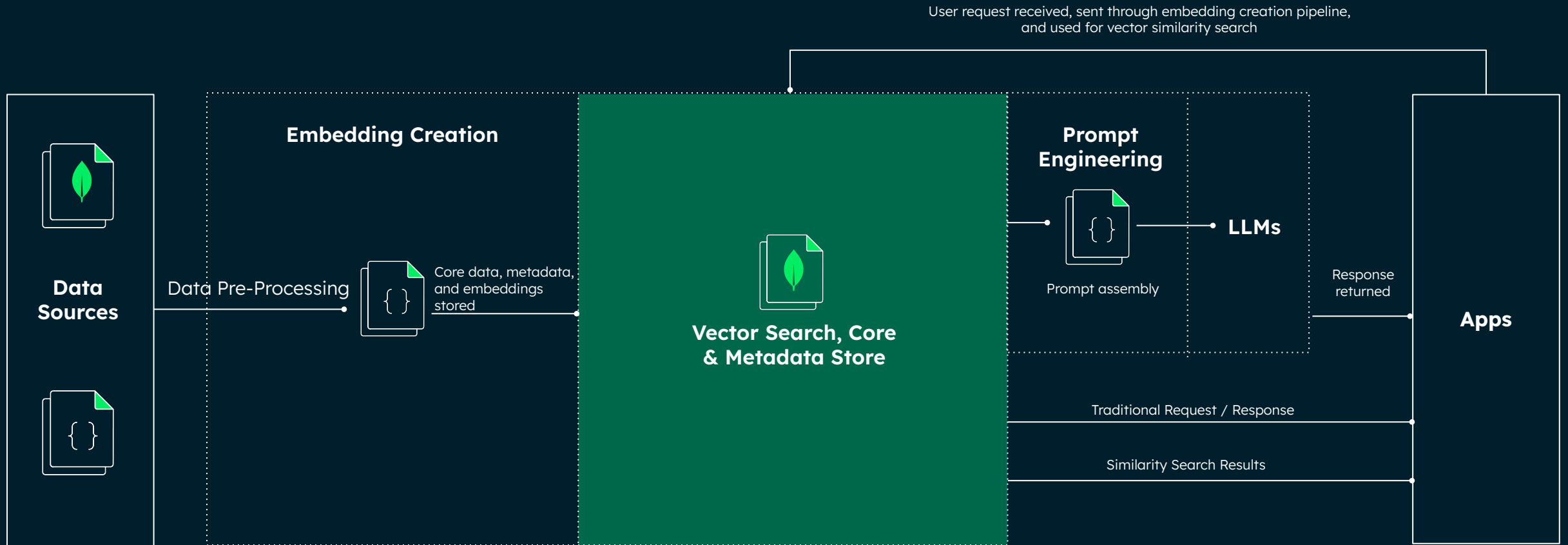
Q & A Systems

Image, Audio, Multimedia Search

SEMANTIC SIMILARITY

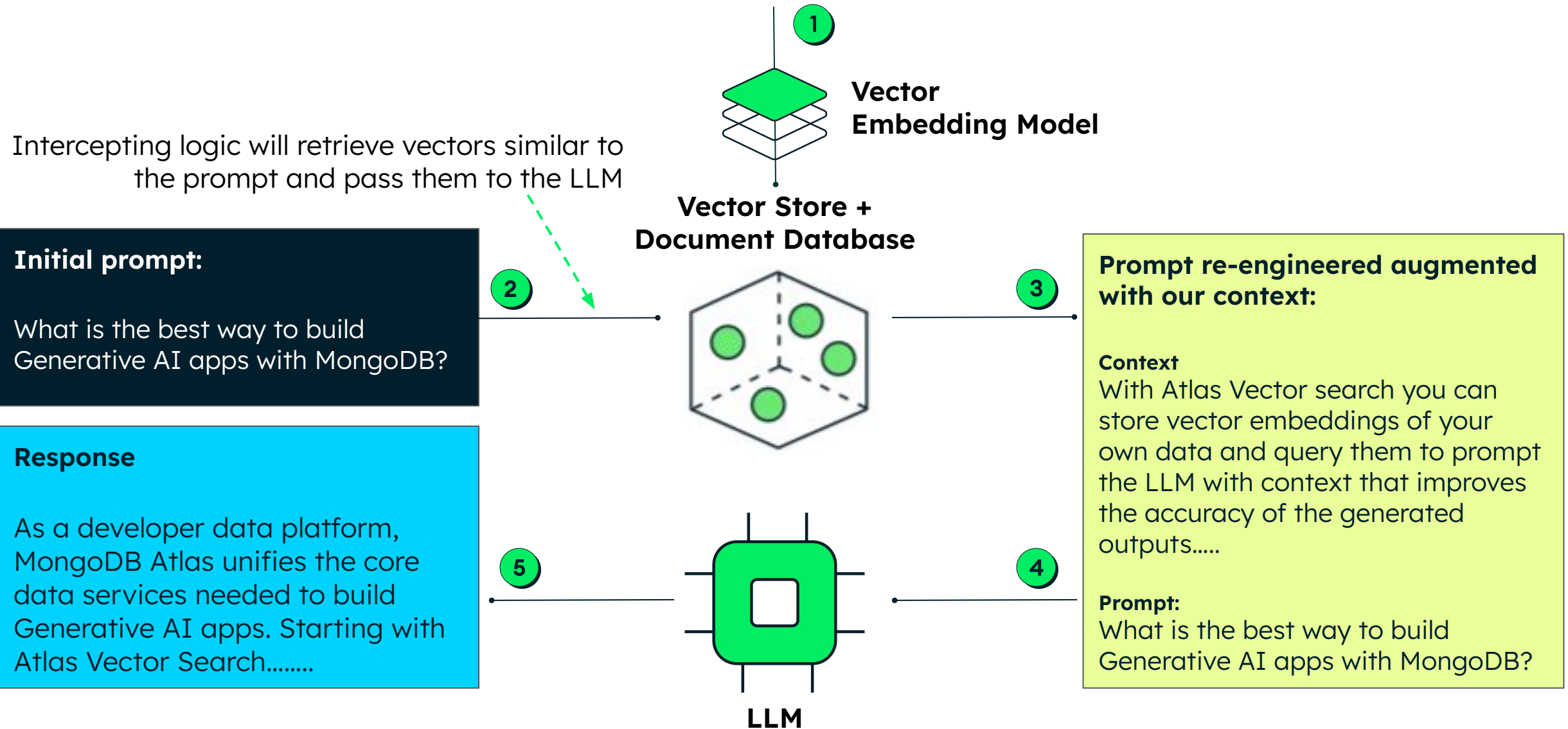## Demonstrated via simplified vector (in two dimensions)

Mozzarella [0.3, 0.9]

Cheese [0.5, 0.7]

Pizza [0.3, 0.4]

Cheddar [0.9, 0.2]

Vector Search With MongoDB

Internal Data Sources

Databases ERP/CRM Knowledge Bases Docs & repos White papers Etc....

① Vector Embedding Model

Vector Store + Document Database

Intercepting logic will retrieve vectors similar to the prompt and pass them to the LLM

Initial prompt:

What is the best way to build Generative AI apps with MongoDB?

②

③ Prompt re-engineered augmented with our context:

Context
With Atlas Vector search you can store vector embeddings of your own data and query them to prompt the LLM with context that improves the accuracy of the generated outputs.....

Prompt:
What is the best way to build Generative AI apps with MongoDB?

④

Response

As a developer data platform, MongoDB Atlas unifies the core data services needed to build Generative AI apps. Starting with Atlas Vector Search.......

⑤

LLM

Vector Search Demo

Thank you!

Appendix