Data Modeling with MongoDB

Yulia Genkina
Curriculum Engineer @ MongoDB

Agenda

Key Considerations

Agenda

Key Considerations

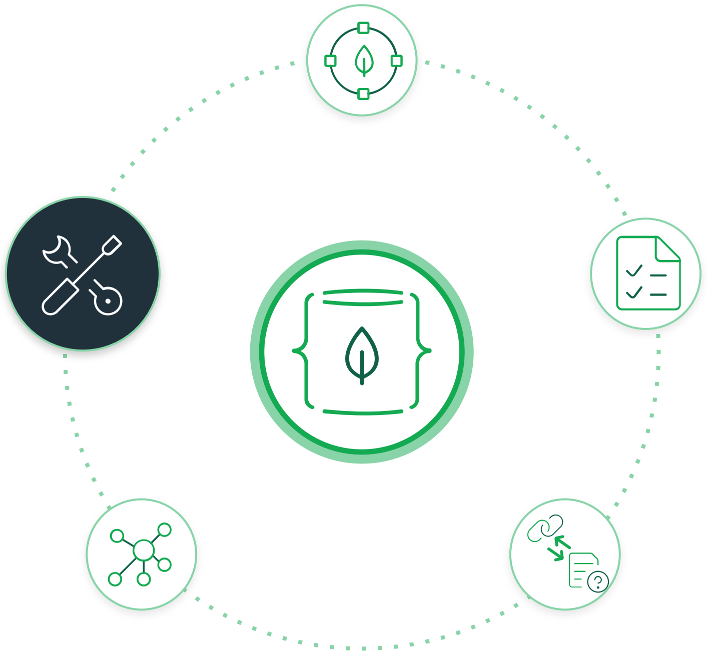
Linking vs. Embedding

Agenda

Key Considerations

Linking vs. Embedding

Design Patterns



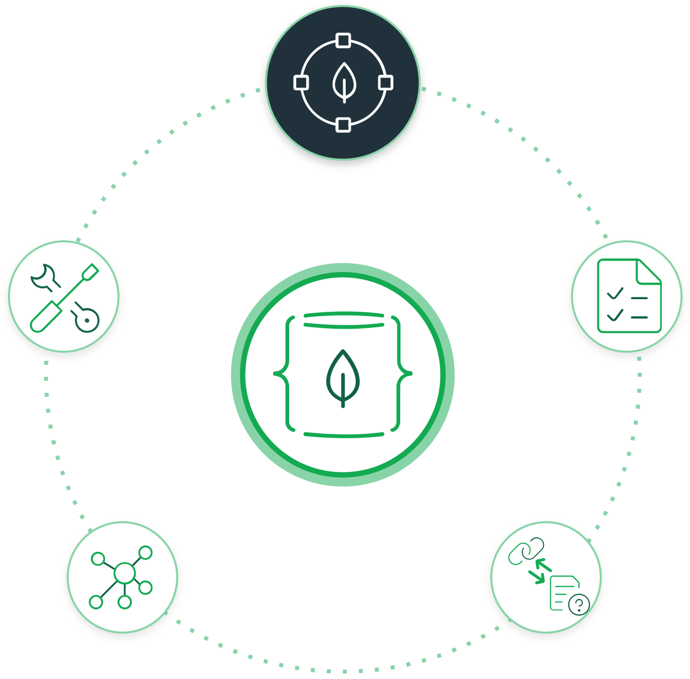
Sub - Bullet points

Key Considerations

Linking vs. Embedding

Design Patterns

Use Case Example



Agenda

Key Considerations

Linking vs. Embedding

Design Patterns

Use Case Example

Conclusion

Let’s Compare

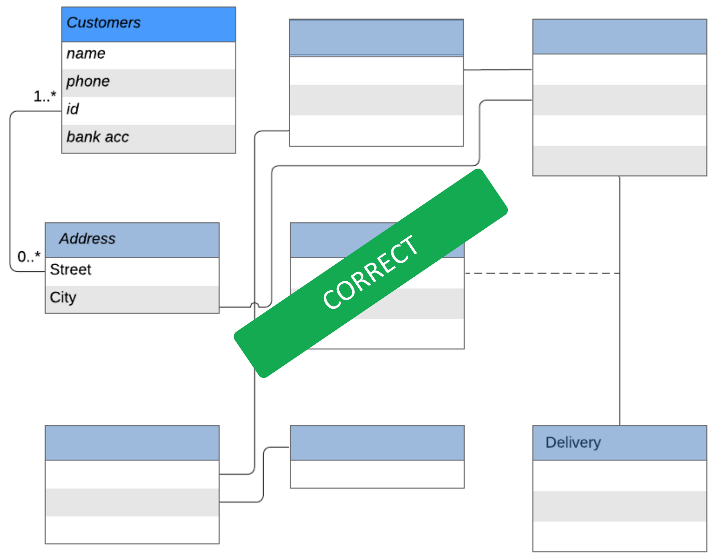
RDBMS approach to data modeling vs. MongoDB

Modeling for RDBMS

Step 1: Define the Schema

Step 2: Develop the application and queries

Concerns

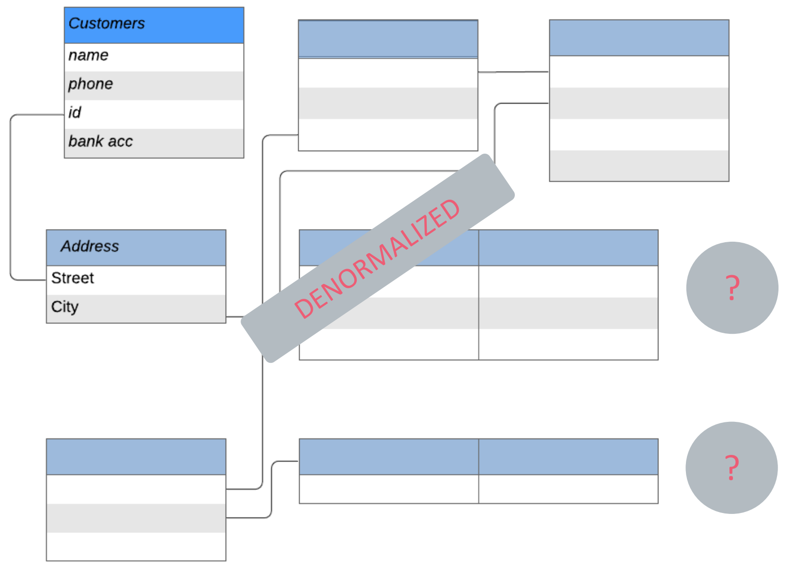


Modeling for RDBMS

Step 1: Define the Schema

Step 2: Develop the application and queries

Concerns

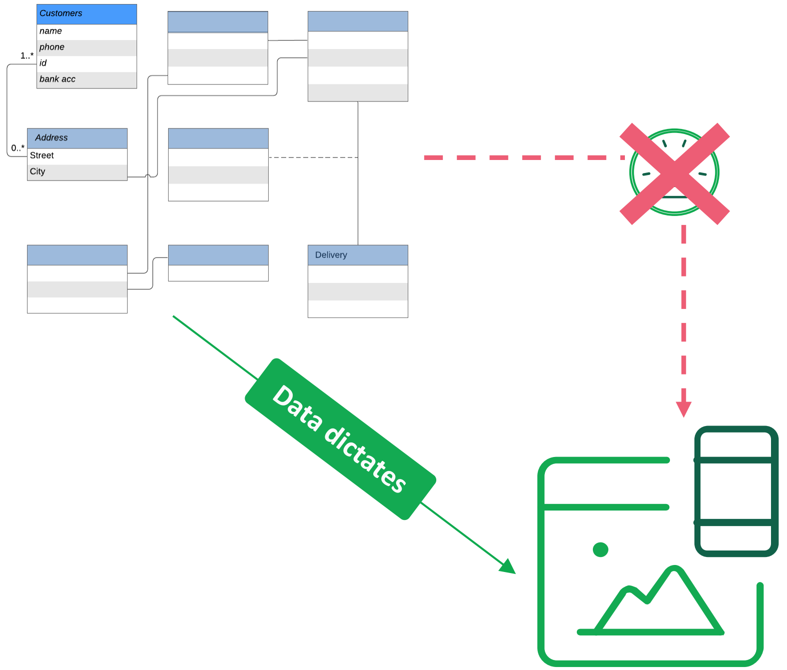


Modeling for RDBMS

Step 1: Define the Schema

Step 2: Develop the application and queries

Concerns

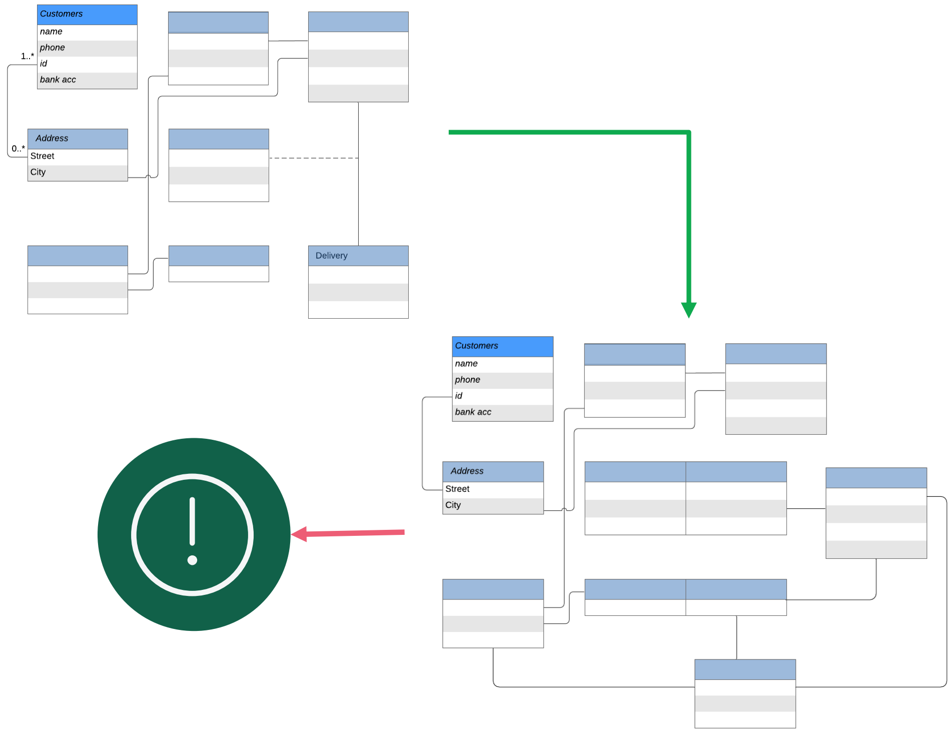


Modeling for RDBMS

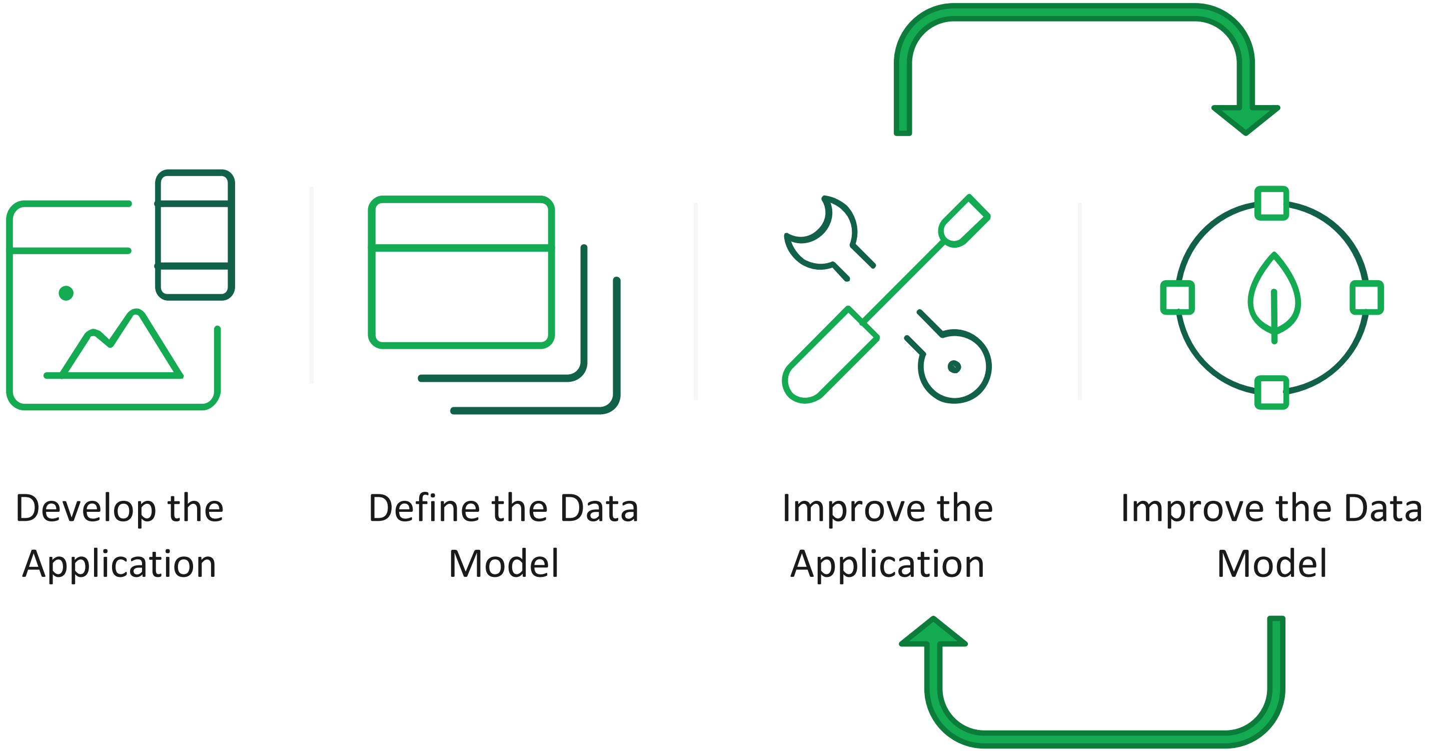
Step 1: Define the Schema

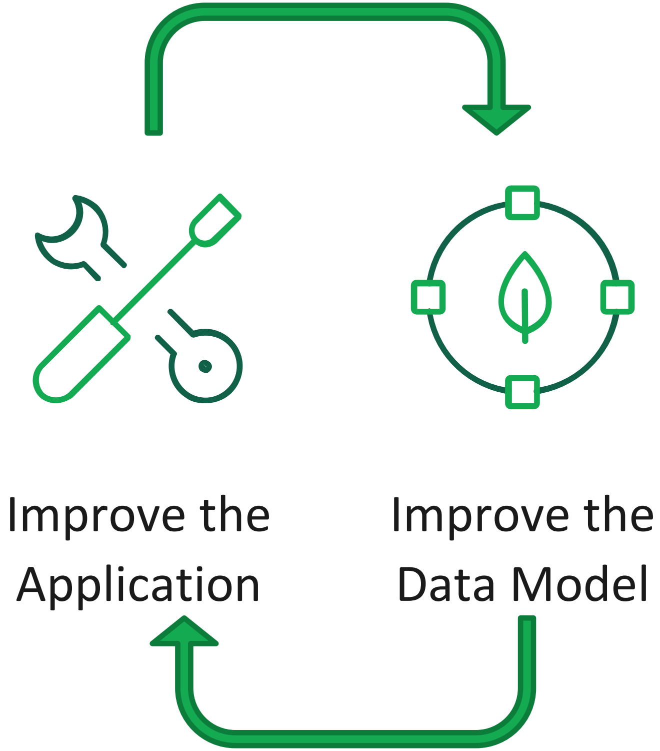
Step 2: Develop the application and queries

Concerns



Data Modeling with MongoDB





Many design options

Designed for the usage pattern

Data model evolution is easy

Can evolve without any downtime

Key Considerations

For Data Modeling with MongoDB

There Is No Magic Formula, but There Is A Method

Data model is defined at the application level

Design is part of each phase of the application lifetime

What affects the data model:

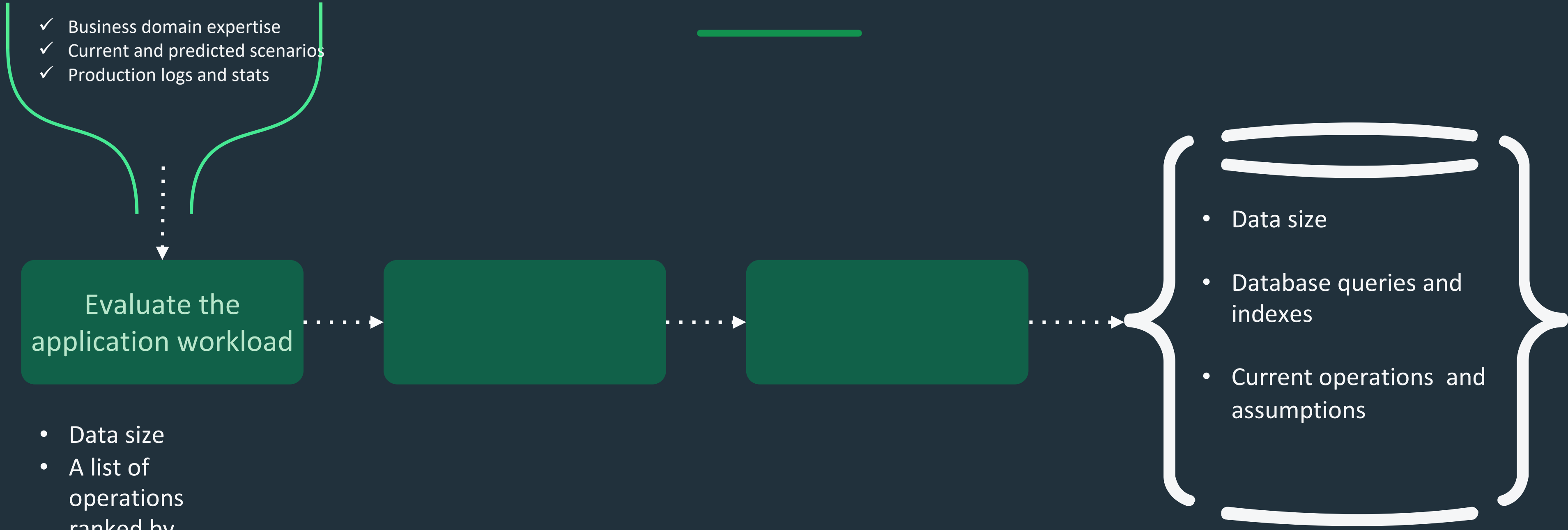
- The data that your application needs
- Application’s read and write usage of the data

Data Modeling

Methodology to Achieve a Near Magic Almost Formula

Step-by-step Iteration

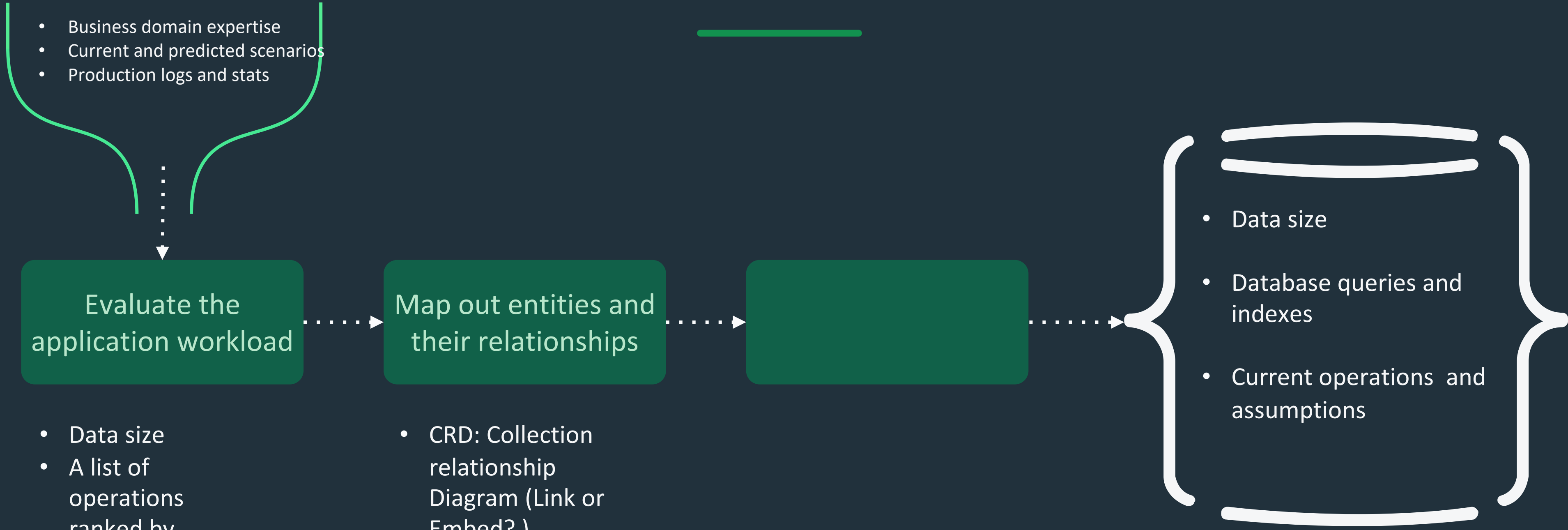
- ✓ Business domain expertise
- ✓ Current and predicted scenarios
- ✓ Production logs and stats



- Data size
- A list of operations ranked by importance

- Data size
- Database queries and indexes
- Current operations and assumptions

Step-by-step Iteration



Evaluate the application workload
Map out entities and their relationships

- Business domain expertise
- Current and predicted scenarios
- Production logs and stats

- Data size
- Database queries and indexes
- Current operations and assumptions

- Data size
- A list of operations ranked by importance

- CRD: Collection relationship Diagram (Link or Embed?)

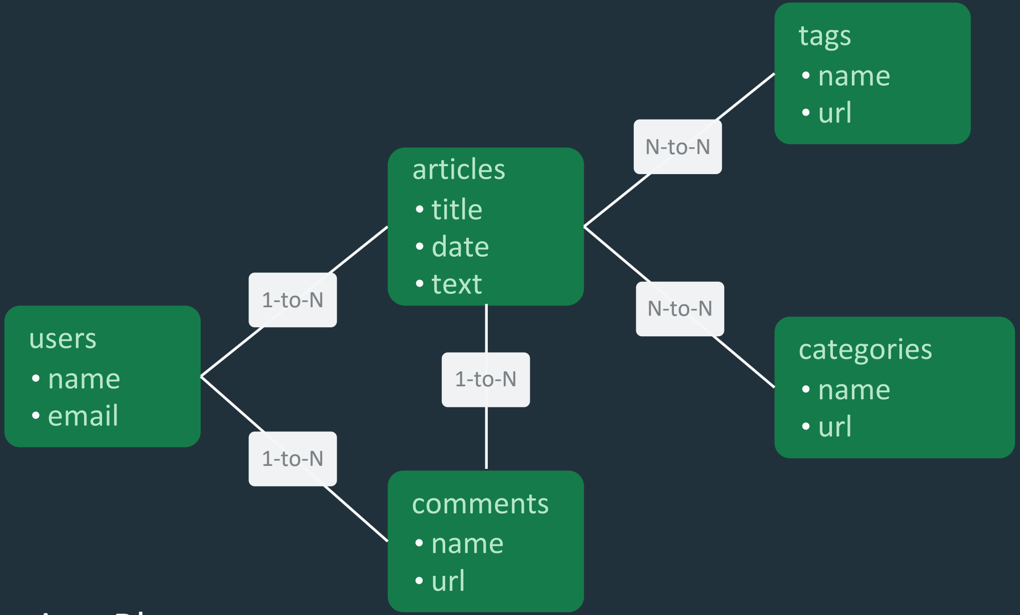
Link vs. Embed

Which is the Right Decision and What Does it Mean?

What Can Be Linked?

Relationships:

- One-to-one
- One-to-many
- Many-to-many



Example: Entities and relationships in a Blog

One-to-One Linked

```
Book = {                                                              // either side can track
    "_id": 1,
    "title": "Harry Potter and the Methods of Rationality",
    "slug": "9781857150193-hpmor",
    "author": 1,                                         // more fields follow…
}

Author = {
    "_id": 1,
    "firstName": "Eliezer",
    "lastName": "Yudkowsky"
    "book": 1,                                           // more fields follow…
}
```

One-to-One Embedded

```
Book = {
  "_id": 1,
  "title": "Harry Potter and the Methods of Rationality",
  "slug": "9781857150193-hpmor",
  "author": {
    "firstName": "Eliezer",
    "lastName": "Yudkowsky"
  },
  // more fields follow…
}
```

One-to-Many: Array in Parent

```
Author= {
   "_id": 1,
   "firstName": "Eliezer",
   "lastName": "Yudkowsky",
   "books": [1, 5, 17],
   // more fields follow…
}
```

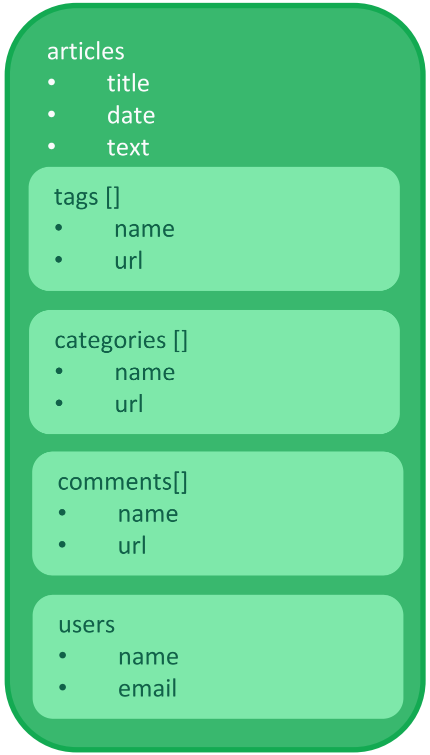
One-to-Many: Scalar in Child

```
Book1= {
  "_id": 1,
  "title": "Harry Potter and the Methods of Rationality",
  "slug": "9781857150193-hpmor",
  "author": 1,                    // more fields follow…
}

Book2= {
  "_id": 5,
  "title": "How to Actually Change Your Mind",
  "slug": "1939311179490-how-to-change",
  "author": 1,                    // more fields follow…
}
```

Many-to-Many: Arrays on either side

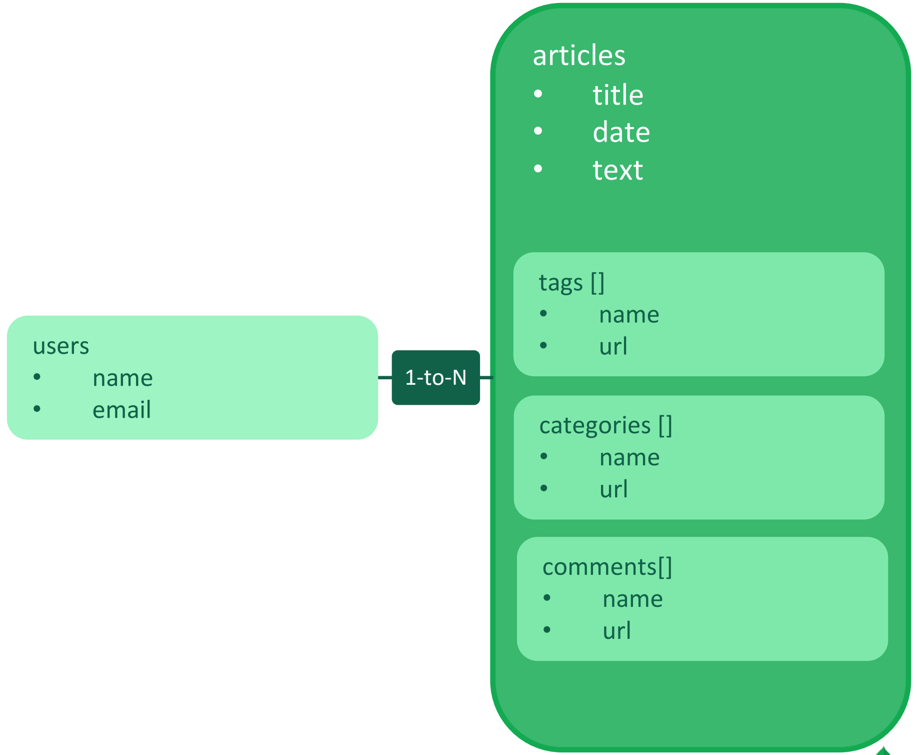
```
Book = {                                                        //either side can track
   "_id": 5,
   "title": "Harry Potter and the Methods of Rationality",
   "slug": "9781857150193-hpmor",
   "authors": [1, 3],                          // more fields follow…
}

Author = {
   "_id": 1,
   "firstName": "Eliezer",
   "lastName": "Yudkowsky",
   "books": [5, 7],                            // more fields follow…

}
```

Embed All



Queries by **articles**

Embed **&Link**



Queries by **articles** **or** **users**

To Link or Embed?

- ? How often does the embedded information get accessed?
- ? Is the data queried using the embedded information?
- ? Does the embedded information change often?

Step-by-step Iteration

- Business domain expertise
- Current and predicted scenarios
- Production logs and stats

Evaluate the application workload

- Data size
- A list of operations ranked by importance

Map out entities and their relationships

- CRD: Collection relationship Diagram (Link or Embed?)

Finalize the data model for each collection

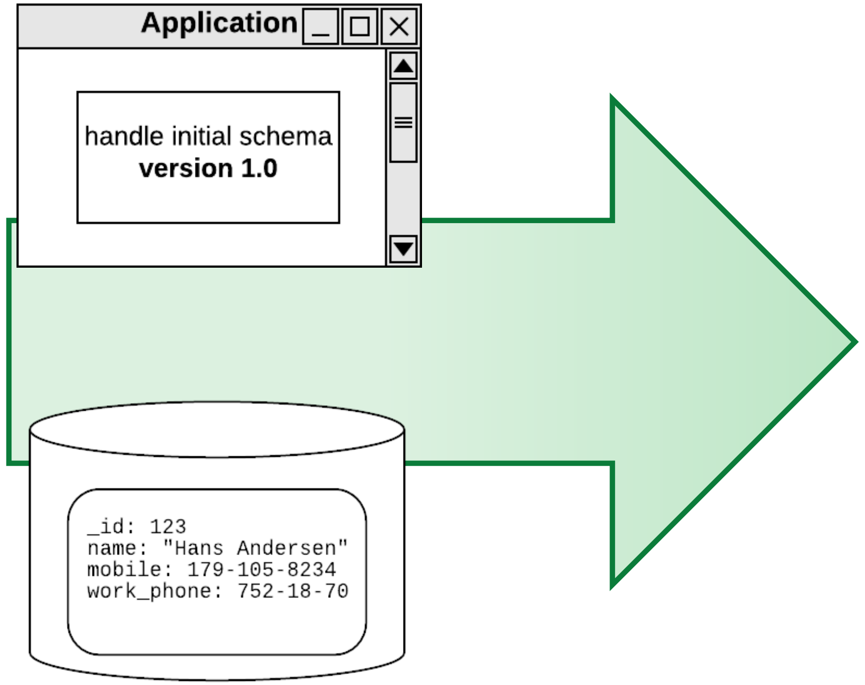
- Identify and apply relevant design patterns

- Collections with documents fields and shapes for each
- Data size
- Database queries and indexes
- Current operations assumptions, and growth projections

Design Patterns

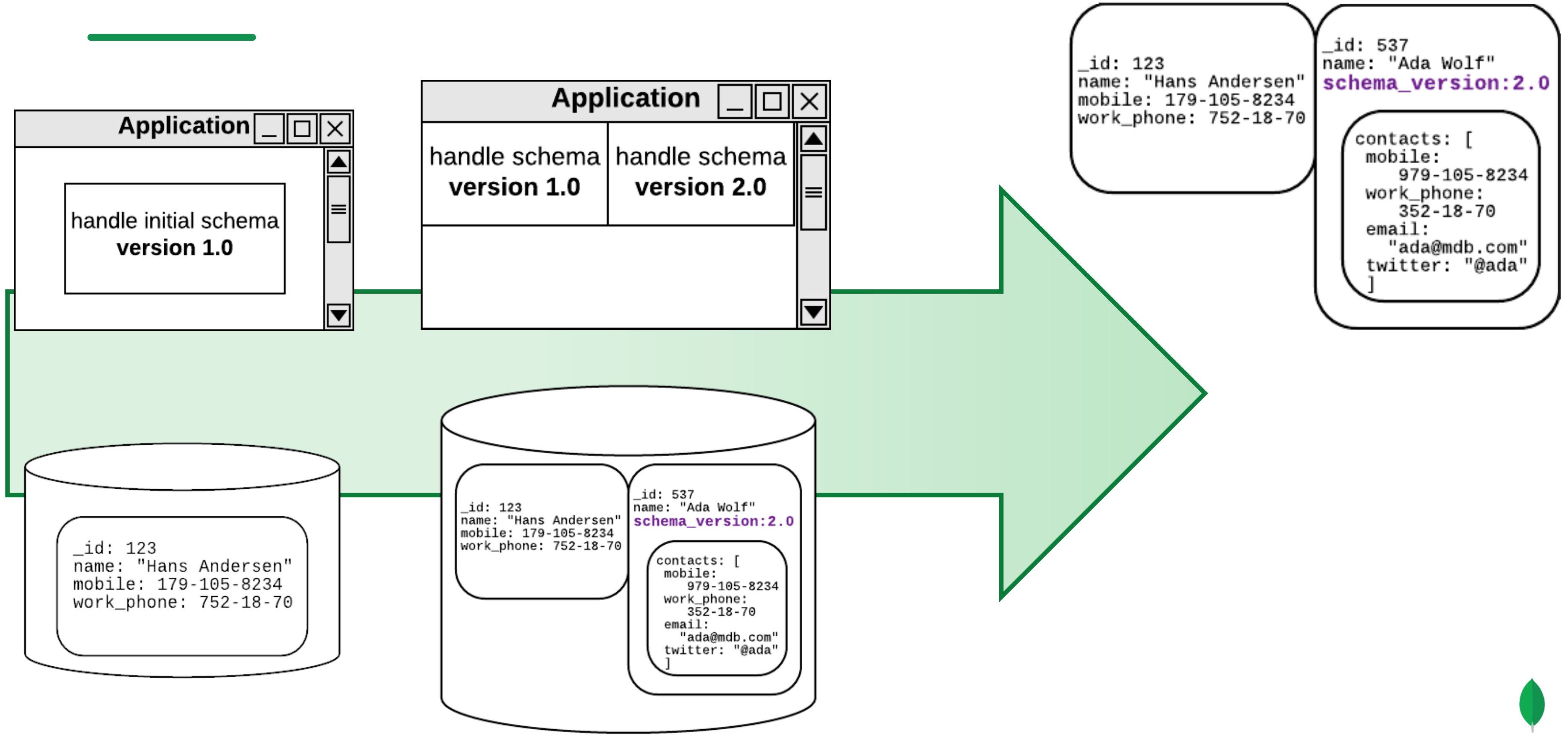
Brief introduction

The Schema Versioning Pattern

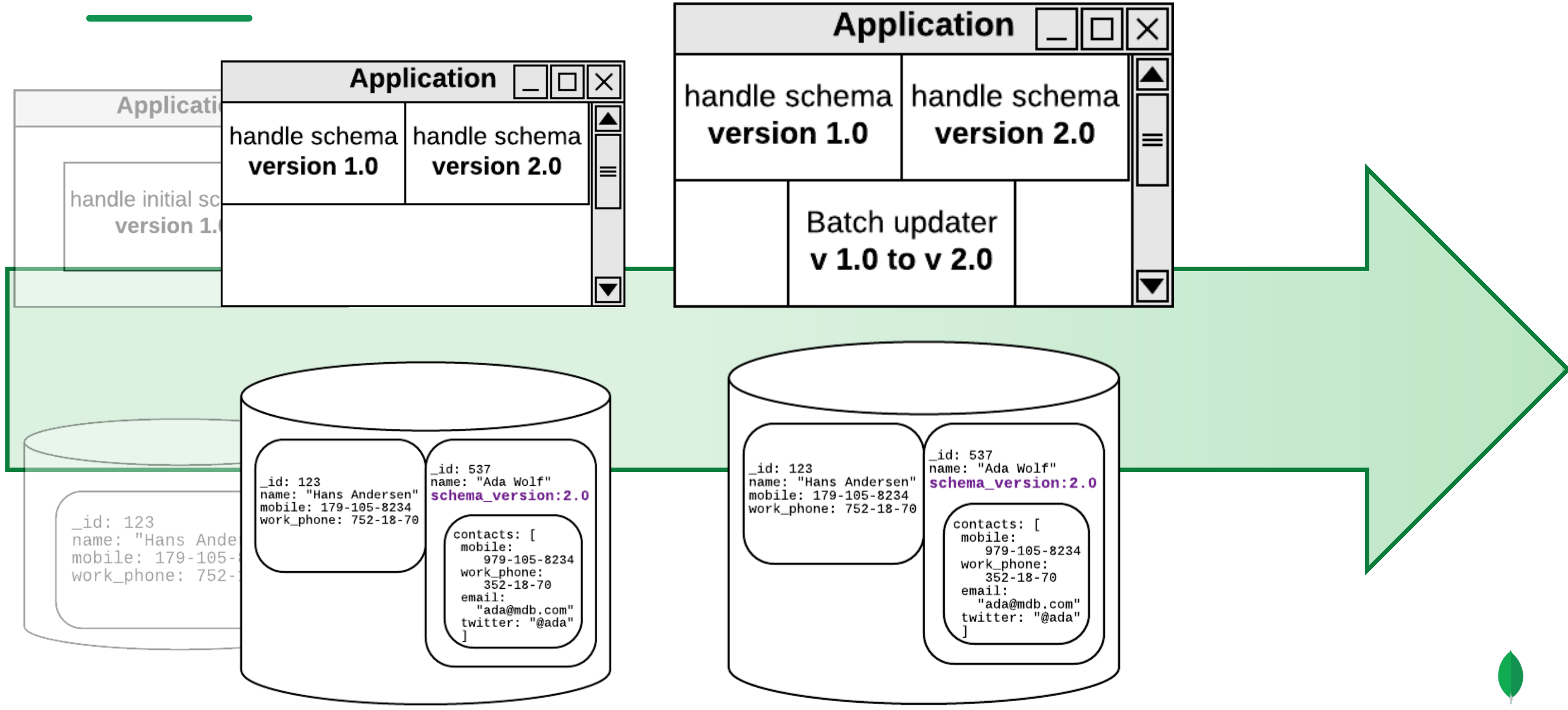


```
_id: 123
name: "Hans Andersen"
mobile: 179-105-8234
work_phone: 752-18-70
```

The Schema Versioning Pattern



The Schema Versioning Pattern

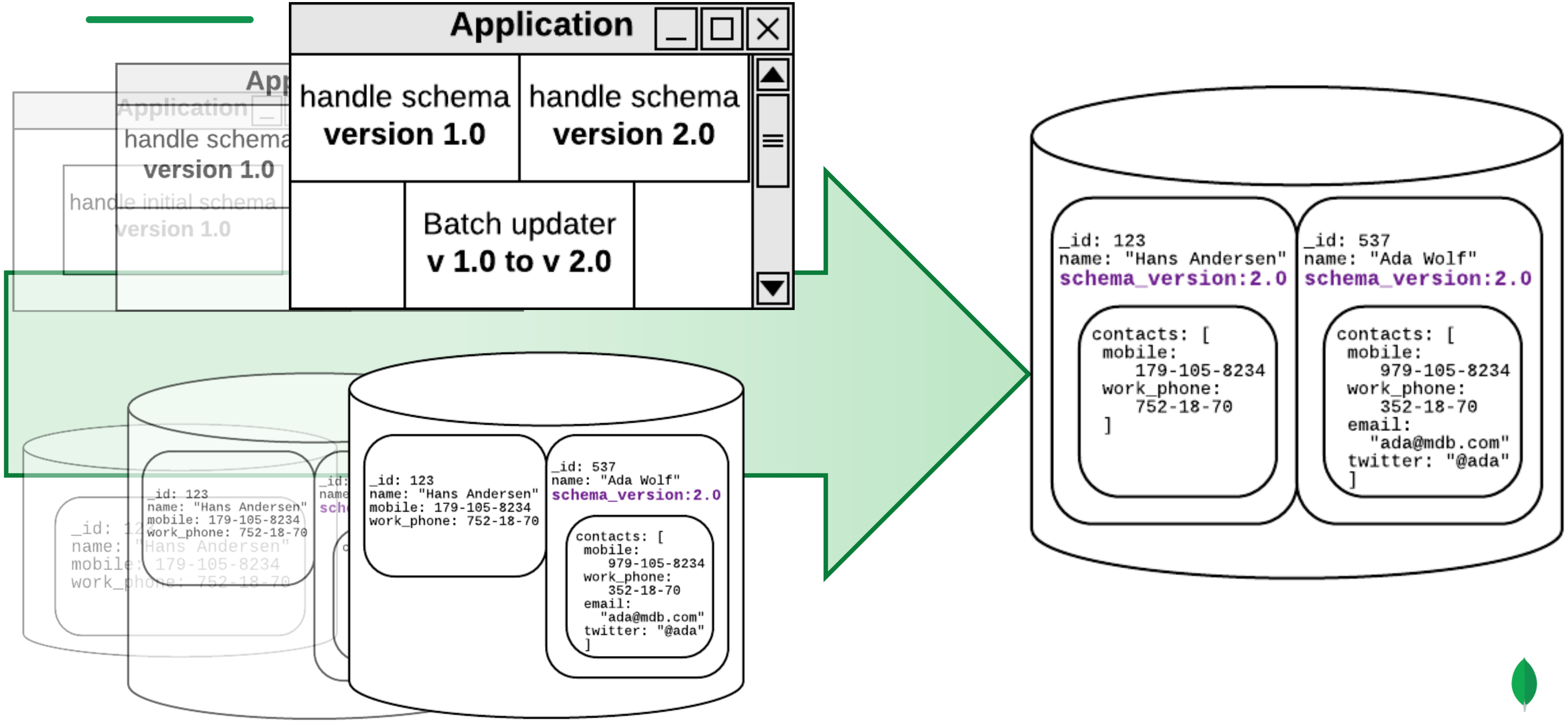


Application
handle initial schema version 1.0
Application
handle schema version 1.0
handle schema version 2.0
Application
handle schema version 1.0
handle schema version 2.0
Batch updater v 1.0 to v 2.0

```
_id: 123
name: "Hans Andersen"
mobile: 179-105-8234
work_phone: 752-18-70
```

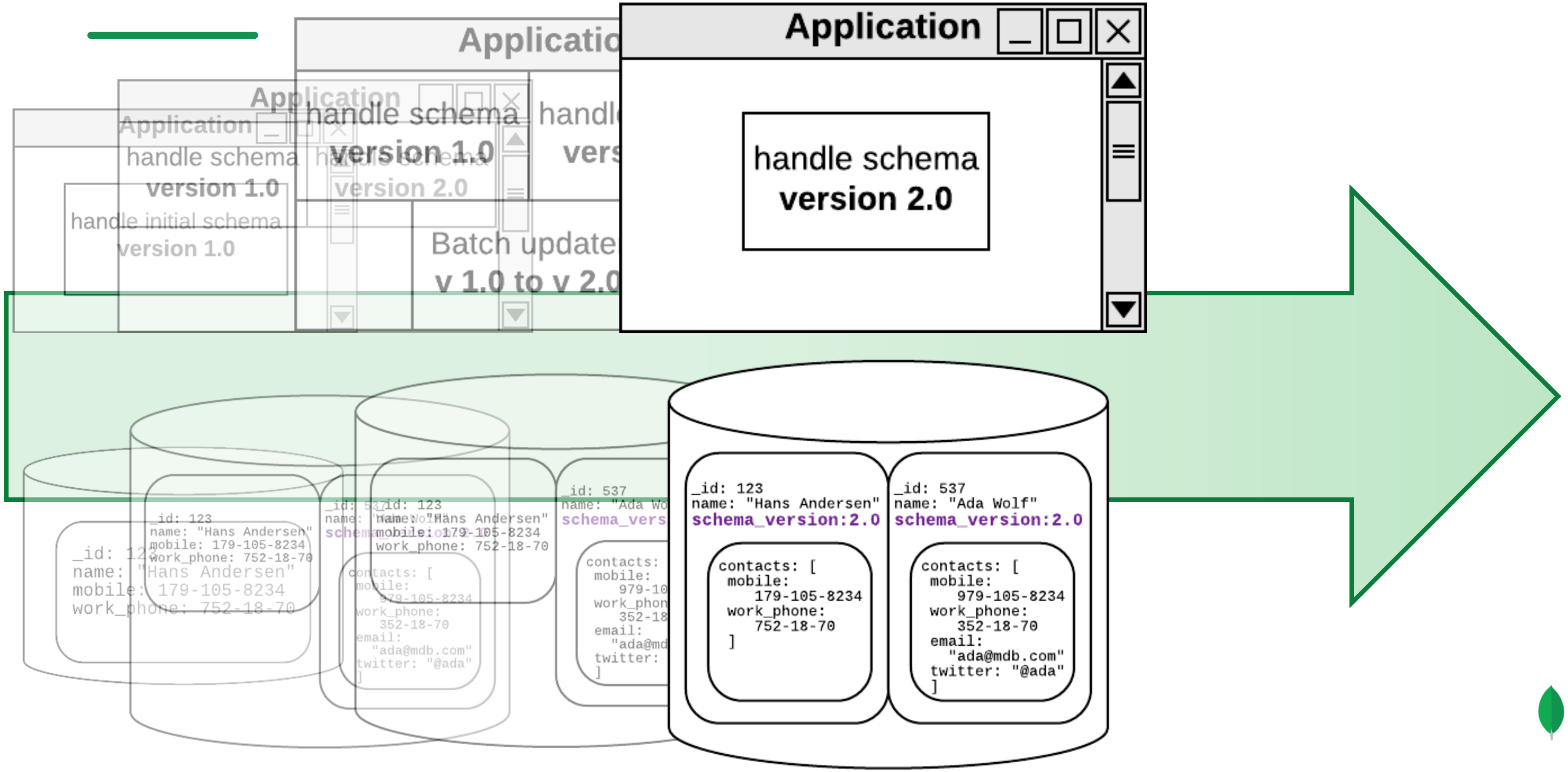
```
_id: 537
name: "Ada Wolf"
schema_version:2.0
contacts: [
  mobile:
     979-105-8234
  work_phone:
     352-18-70
  email:
    "ada@mdb.com"
  twitter: "@ada"
]
```

The Schema Versioning Pattern



Application
handle schema version 1.0
handle schema version 2.0
Batch updater v 1.0 to v 2.0
handle schema version 1.0
handle initial schema version 1.0
_id: 123
name: "Hans Andersen"
mobile: 179-105-8234
work_phone: 752-18-70
_id: 537
name: "Ada Wolf"
schema_version:2.0
contacts: [mobile: 979-105-8234 work_phone: 352-18-70 email: "ada@mdb.com" twitter: "@ada"]
_id: 123
name: "Hans Andersen"
schema_version:2.0
contacts: [mobile: 179-105-8234 work_phone: 752-18-70]
_id: 537
name: "Ada Wolf"
schema_version:2.0
contacts: [mobile: 979-105-8234 work_phone: 352-18-70 email: "ada@mdb.com" twitter: "@ada"]

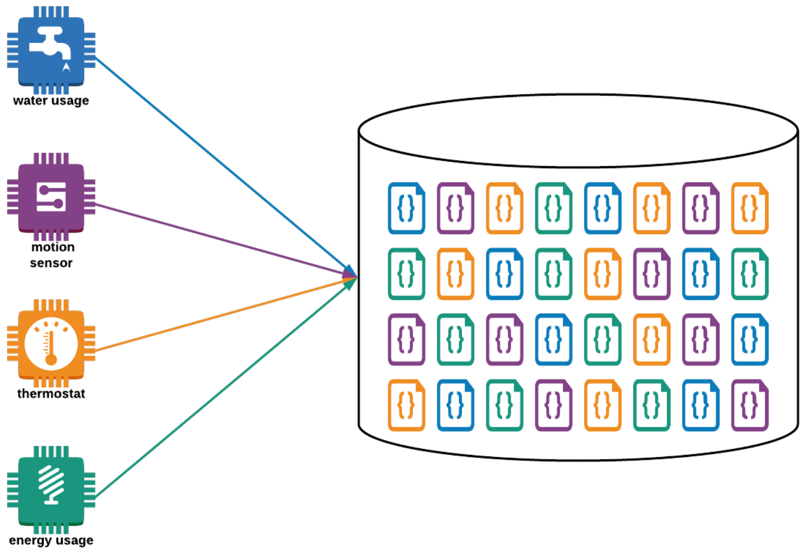
The Schema Versioning Pattern



The Bucket Pattern

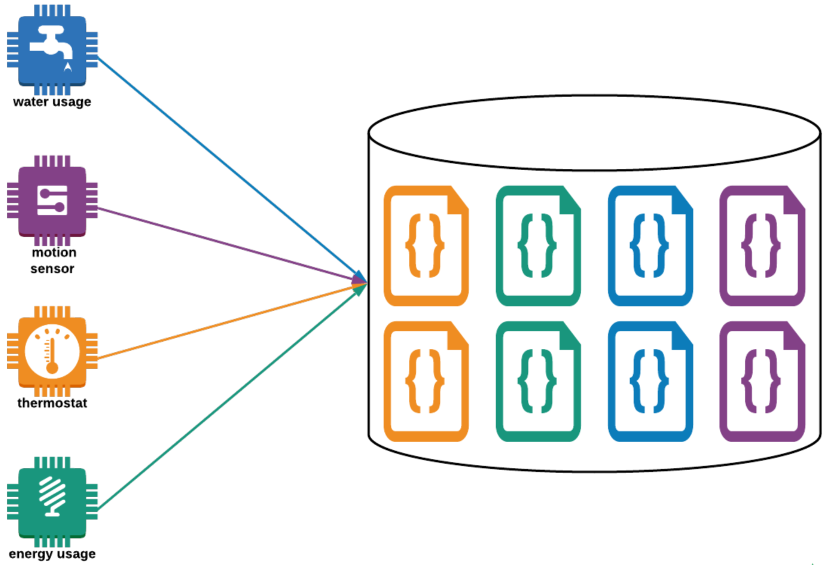
Tabular Approach

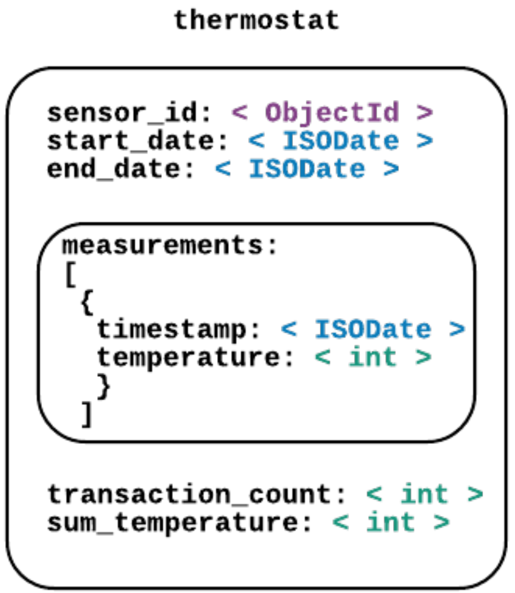
New document for each sensor reading



Document Approach

New document per time unit per sensor





```
sensor_id: < ObjectId >
start_date: < ISODate >
end_date: < ISODate >

measurements:
[
 {
  timestamp: < ISODate >
  temperature: < int >
 }
]

transaction_count: < int >
sum_temperature: < int >
```

The Bucket Pattern

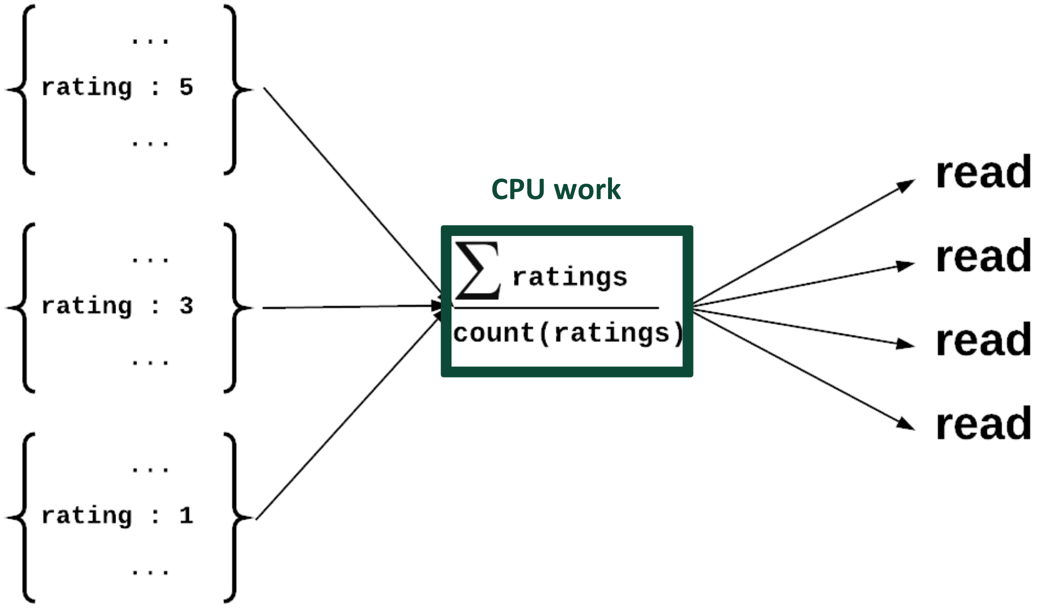
Enables the Computed Pattern

- Really benefits from the document model
- Used to store small, related data items
  - Bank Transactions – related by account and date
  - IoT Readings – related by sensor and date
- Reduces index sizes by a large magnitude
- Increases speed of retrieval of related data

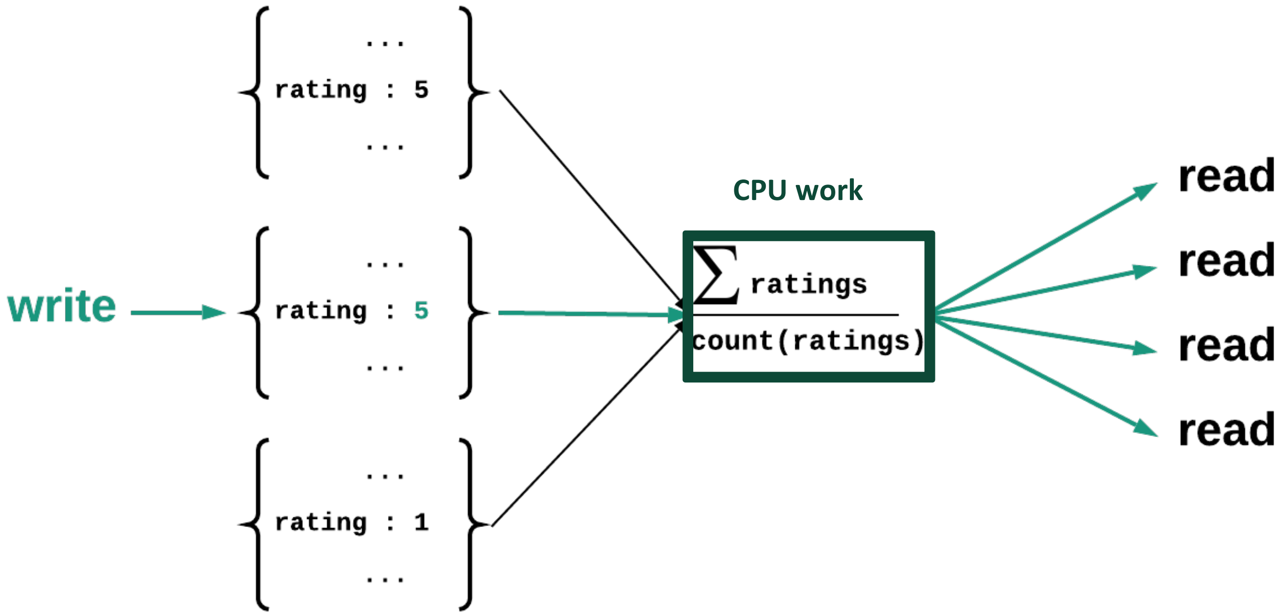
The Bucket Pattern Implementation

```
sensor = 5, value = 22, time = Date('2020-05-11')

db.iot.updateOne({ "sensor": reading.sensor,
                   "valcount": { "$lt": 200 } },
                 { "$push": { "readings": { "v": value, "t": time } },
                        "$inc": { "valcount": 1 } },
                 { upsert: true })

{ "_id": ObjectId("abcd12340101"), "sensor": 5, "valcount": 3,
  "readings": [                    {"v": 11, "t": Date("2020-05-09")},
                                   {"v": 81, "t": Date("2020-05-10")},
                                   {"v": 22, "t": Date("2020-05-11")} ] }


}
```

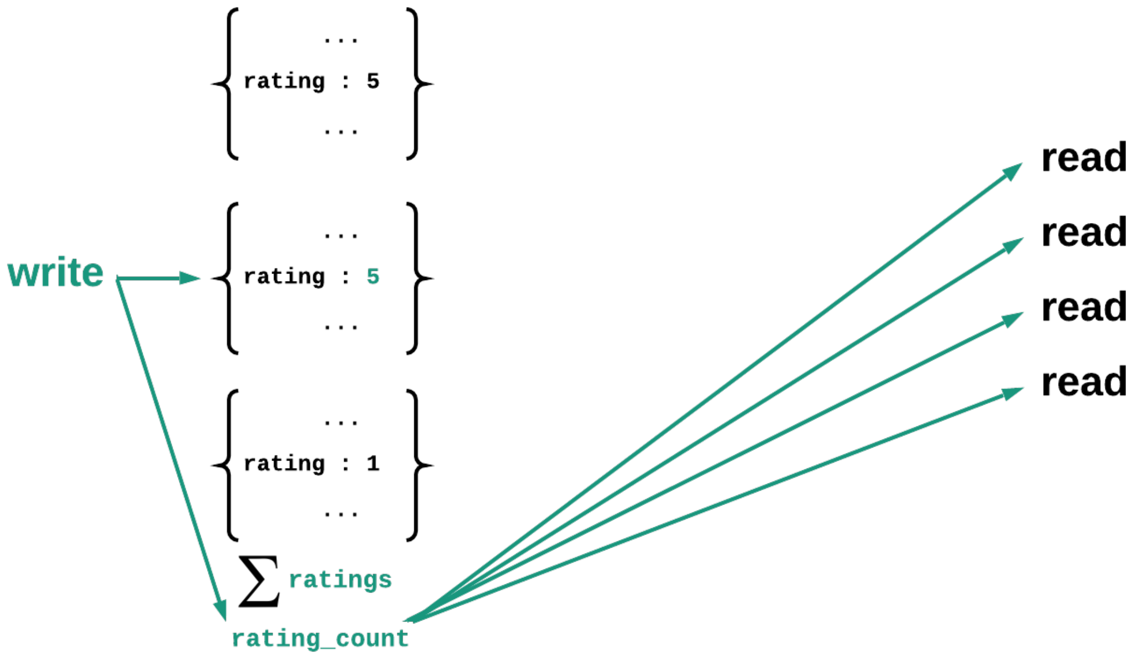
The Computed Pattern



The Computed Pattern



The Computed Pattern



movies

```
_id : ObjectId,
title : str,
director : str,
description : str,
description : str,
cast : [ str ],
total_ratings : int,
rating_count : int,

  reviews : [
    { name : str,
      rating : int,
      review : str
    }
  ]
```

The Computed Pattern

- ✓ "Never recompute what you can **precompute**"
- ✓ Reads are often more common than writes
- ✓ Compute on write is less work than compute on read
- ✓ When updating the database, update some summary records too
- ✓ Can be thought of as a caching pattern

Computed Pattern with the Bucket Pattern

```
sensor = 5, value = 22, time = Date('2020-05-11')

db.iot.updateOne({ "sensor": reading.sensor,
                   "valcount": { $lt:200 } },
                         { "$push": { "readings": { "v": value, "t": time } },
                           "$inc": { "valcount": 1, "tot": value } },
                  { upsert: true })

{ "_id": ObjectId("abcd12340101"), "sensor": 5, "valcount": 3, "tot": 114,
  "readings": [                    { "v": 11, "t": Date("2020-05-09" )},
                                   { "v": 81, "t": Date("2020-05-10" )},
                                   { "v": 22, "t": Date("2020-05-11" )} ] }
```

Learning

Other Patterns and Where To Find Them

MongoDB Blog, MongoDB Developer Portal and MongoDB University are all great resources to continue learning about data modeling and patterns.

Design Patterns: Elements of Reusable Object-Oriented Software – a book!

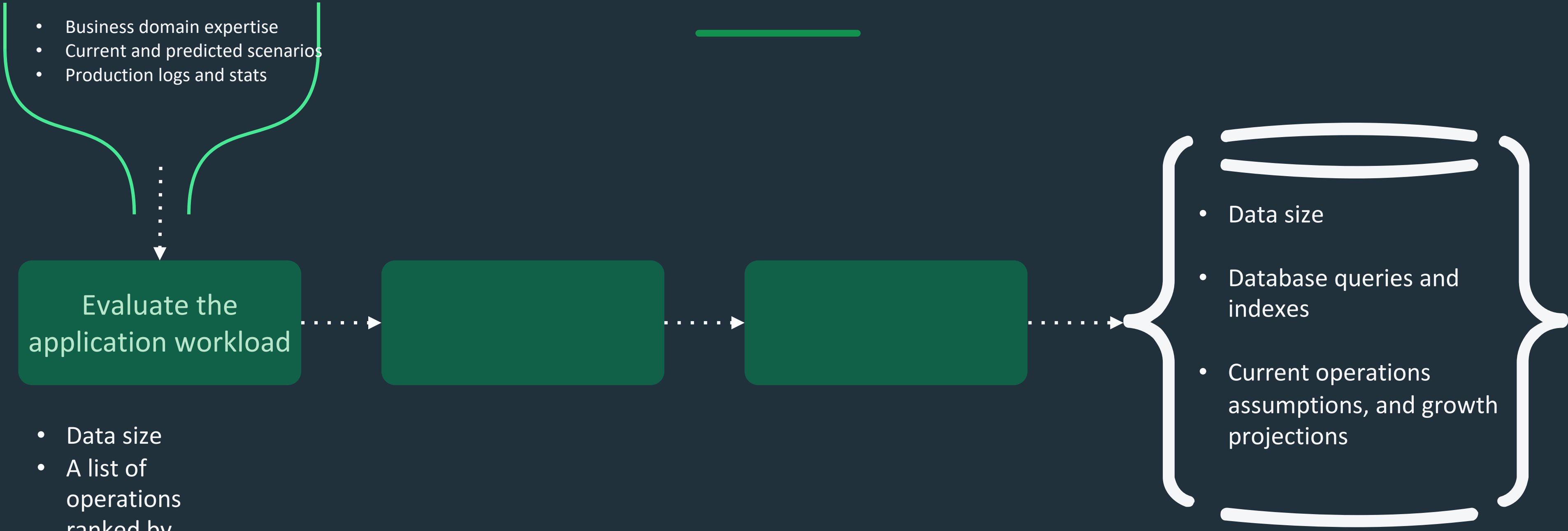
Other talks at this conference:

- Advanced Schema Design Patterns
- A Complete Methodology to Data Modeling
- Using JSON Schema to Save Lives
- Attribute Pattern and the Wildcard Index: Is the Attribute Pattern Obsolete?

Design an Online Shopping App: MongoMart

A Use Case Example

Step 1



- Business domain expertise
- Current and predicted scenarios
- Production logs and stats

Evaluate the application workload

- Data size
- A list of operations ranked by importance

- Data size
- Database queries and indexes
- Current operations assumptions, and growth projections

Evaluate the Application Workload

1000 stores

50 employees per stores
1 store lookup per customer per year

10 Million items

100 reviews per item
500 thousand updates per day

100 Million user accounts

- 500 thousand new accounts per week
- Logging in 20 times a year
- Looking up 100 items per year
- Creating 5 carts per year
- Reviewing 2 items per year

Placing 4 items in the cart
Buying an average of 2 items per cart

Analytics

10 data scientists each running 10 queries a day

Workload Evaluation Summary

Most important queries

- r2: user views a specific item – has to be under 1 ms
- w3: user adds item to cart – write concern: majority

Required indexes

- {"category": 1, "item_name": 1}
- {"category": 1, "item_name": 1, "price": 1}
- {"username": 1} and more..

Assumptions and Projections

- Data will be stored for a maximum of **5 years**
- Number of items sold and number of users will double each year

List of Entities:

- **carts**
- **categories**
- **items**
- **reviews**
- **staff**
- **stores**
- **users**
- **views**

Step-by-step Iteration

- Business domain expertise
- Current and predicted scenarios
- Production logs and stats

Evaluate the application workload

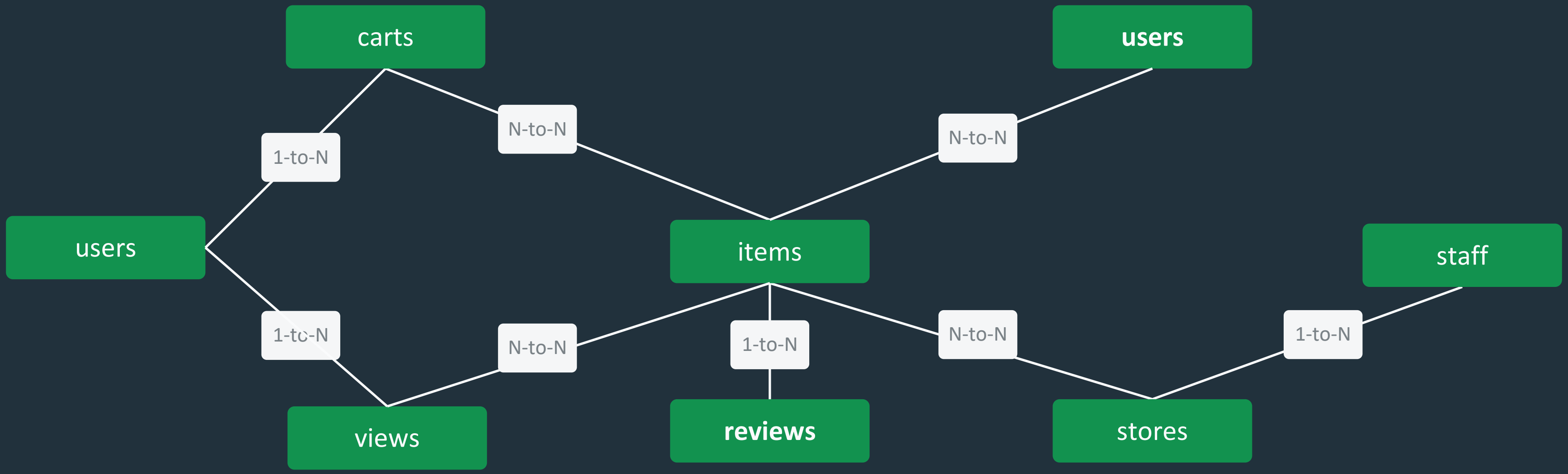
- Data size
- A list of operations ranked by importance

Map out entities and their relationships

- CRD: Collection relationship Diagram (Link or Embed?)

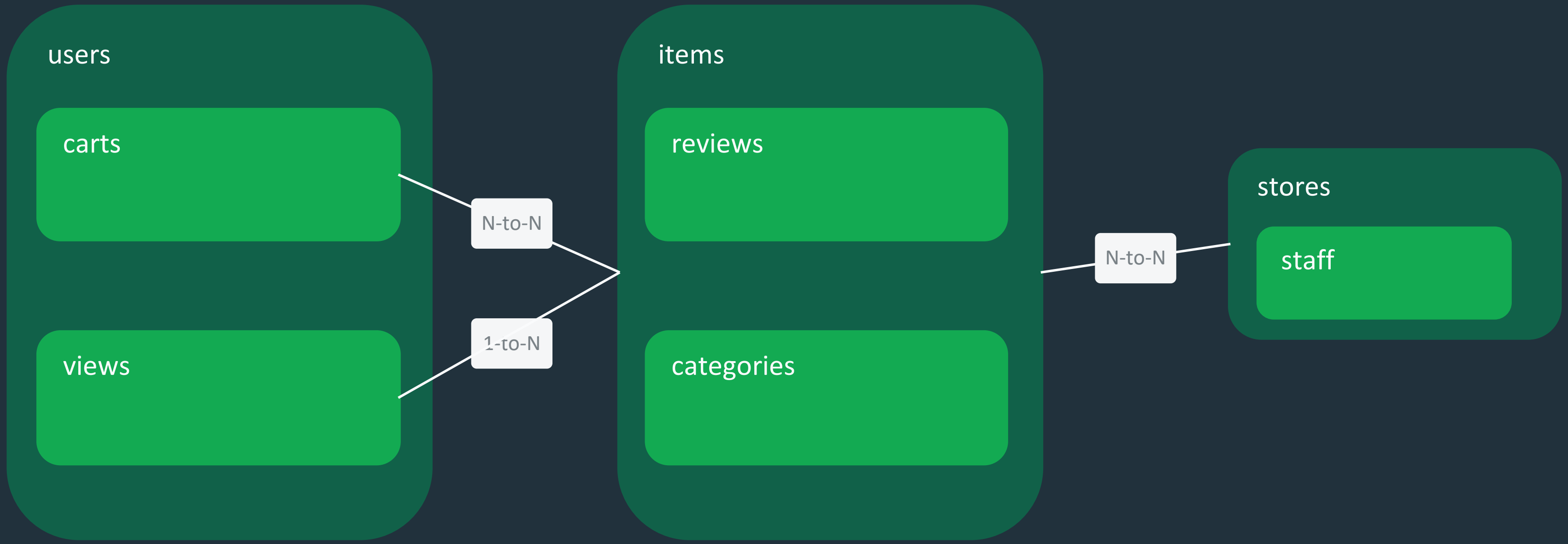
- Collections with documents fields and shapes for each
- Data size
- Database queries and indexes
- Current operations assumptions, and growth projections

Entity Relationship Diagram



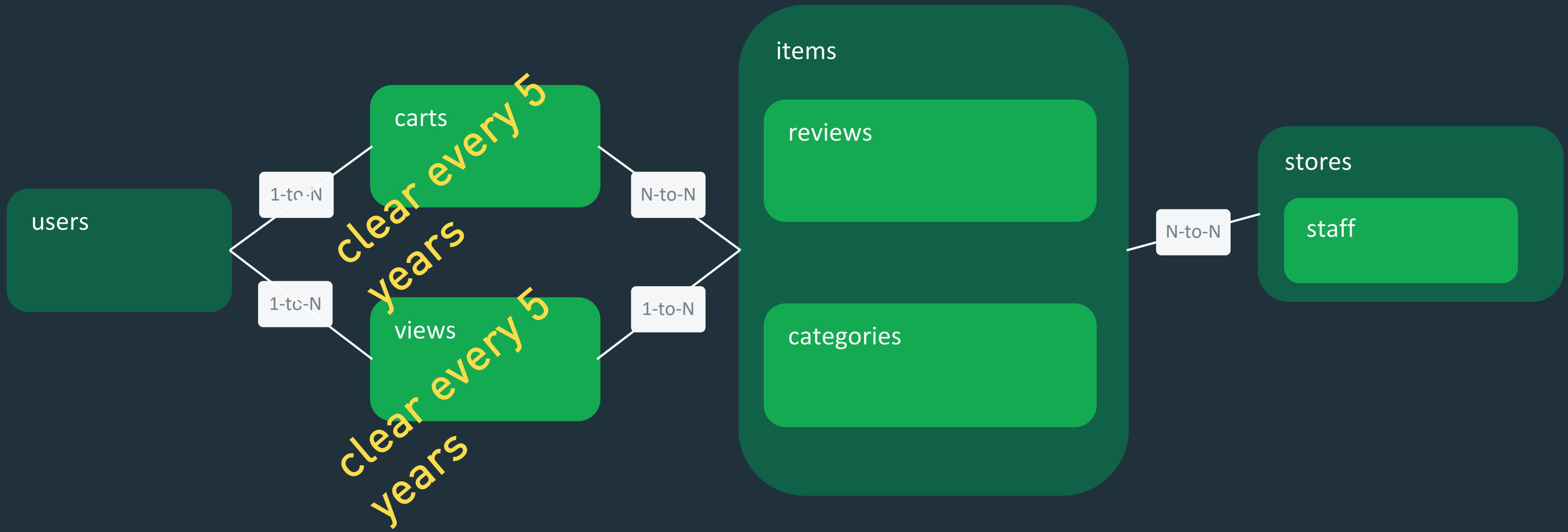
Collections Relationship Diagram (Simple)

Embed Everything!

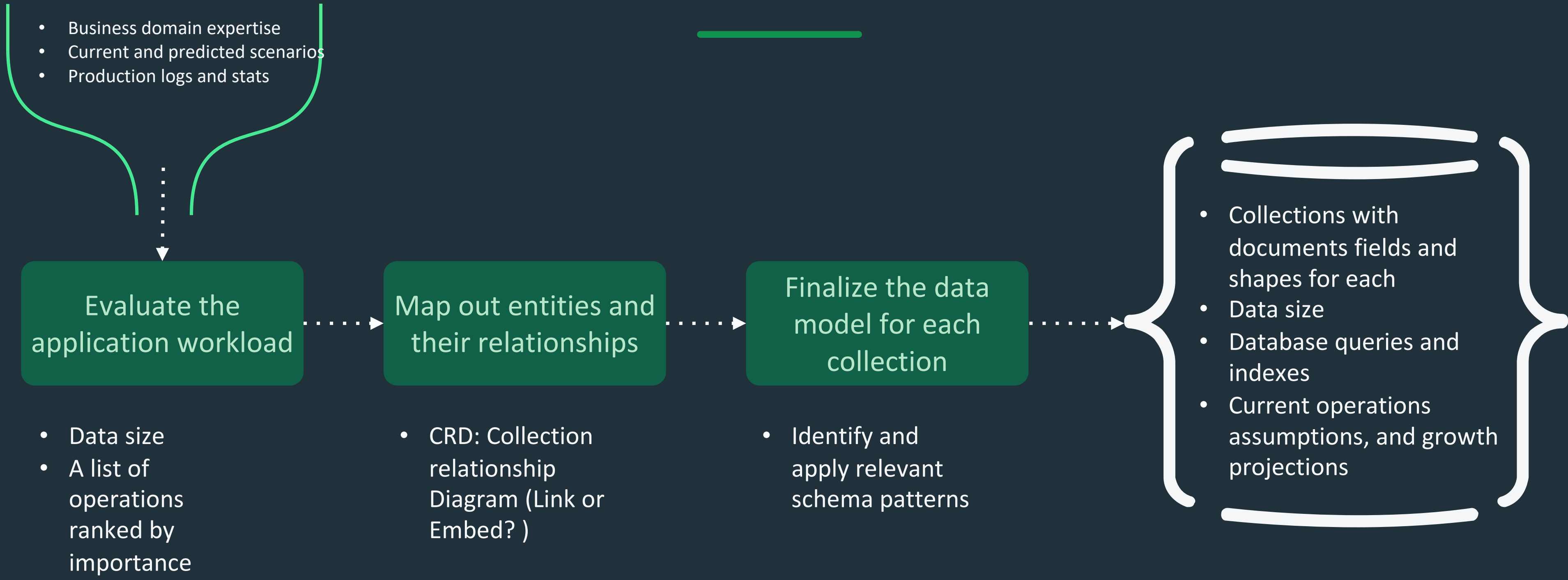


Collections Relationship Diagram (Better)

Accommodate for assumptions.
Embed & Link!



Step-by-step Iteration



- Business domain expertise
- Current and predicted scenarios
- Production logs and stats

Evaluate the application workload

- Data size
- A list of operations ranked by importance

Map out entities and their relationships

- CRD: Collection relationship Diagram (Link or Embed?)

Finalize the data model for each collection

- Identify and apply relevant schema patterns

- Collections with documents fields and shapes for each
- Data size
- Database queries and indexes
- Current operations assumptions, and growth projections

Apply all the Patterns!

items

```
_id : ObjectId,
schema : int,
sku : str,
name : str,
price : decimal,
description : str,
sold_at : [ str ],
tot_rating : int,
num_ratings: int,

  top_reviews : [
    { name : str,
      rating : int,
      review : str
    }
  ]
categories : [ str ]
```

stores

```
_id : ObjectId,
schema : int,
name : str,
address: {
  number : str,
  street : str,
  city : str,
  postal_code : str
 },
items_in_stock: [ str ]

  staff: [
    {
      role : str,
      name : int,
      id : ObjectId
      contact_info:
       {
        mobile : str,
        email : str
       }
    }
  ]
```

reviews

```
_id : ObjectId,
schema : int,
start_date : date,
end_date : date,
sku : str,
reviews : [
  {
    timestamp : date,
    username : str,
    rating : int,
    review : str
   }
]
sum_reviews : int,
num_reviews : int
```

Patterns Used:

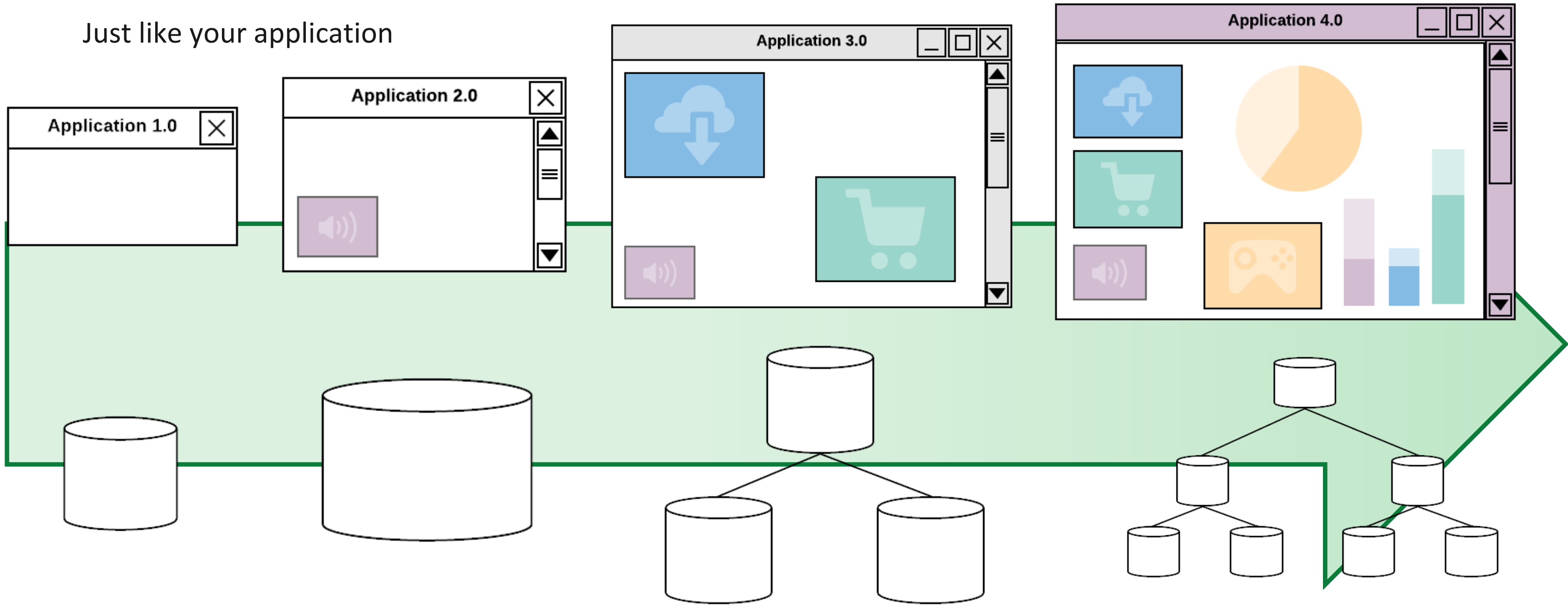
- Schema Versioning
- Subset
- Computed
- Bucket
- Extended Reference

Conclusion

And additional considerations

Your Data Model Will Evolve

Just like your application



Small team

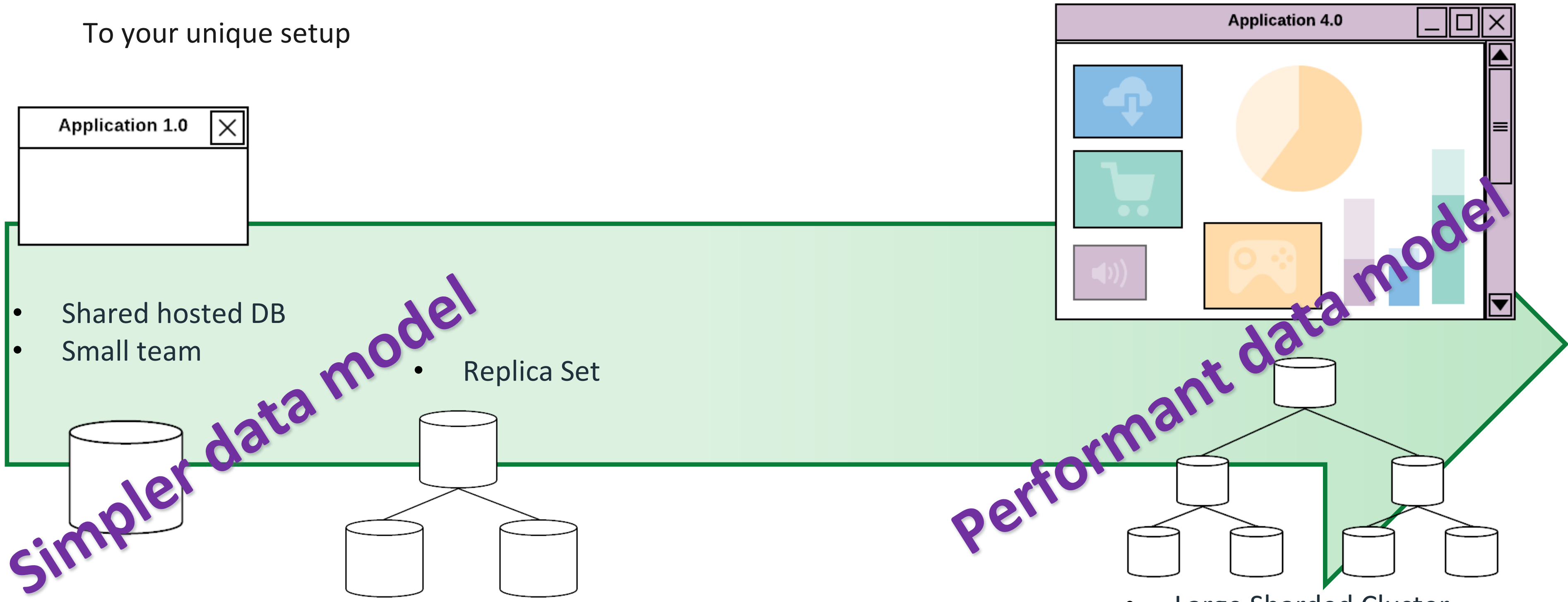
Medium team

Large team

Very big team team

Tailor the Data Model

To your unique setup



- Shared hosted DB
- Small team

- Replica Set

- Large Sharded Cluster

Small team

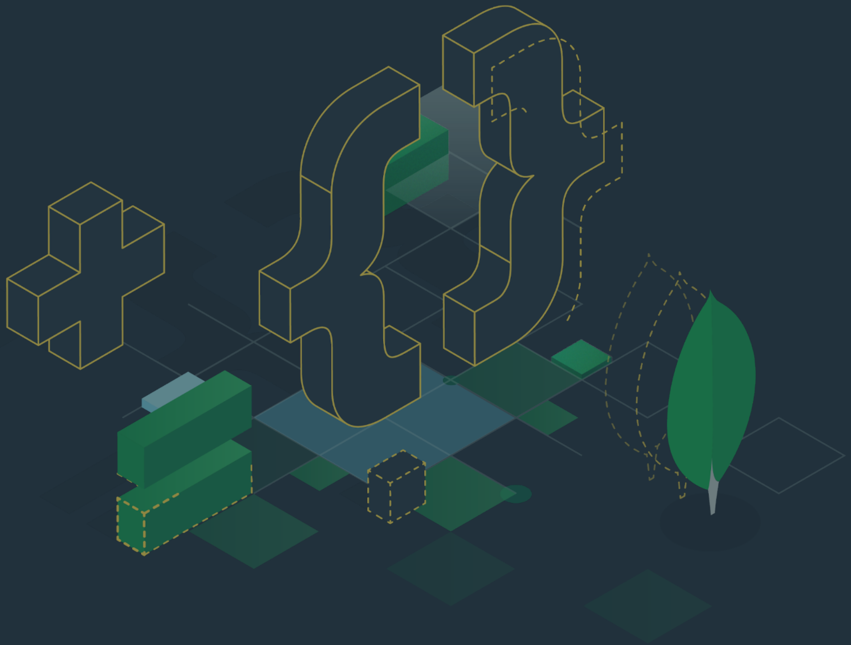
Medium team

Large team

Very big team team

Flexible Data Modeling Approach

	For a **Simpler** data model focus on:	For a bit of **both**:	For the most **Performant** data model focus on:
Evaluate the application workload	The most **frequent** operation	• **Data size** • The most **frequent** operations	• **Data size** • The most **frequent** operations • The most **important** operations
Map out the entities and their relationships	Embedding data	Embedding and linking data	Embedding and linking data
Finalize schema for each collection	Use few patterns	Use as many patterns as necessary	Use as many patterns as necessary



#MDBlive

Visit our product "booths" for new features, like the new Schema Advisor in Atlas!

mongodb.com/live/product

MONGODB .LIVE

#MDBlive

Special Thanks to:
John Page, Daniel Coupal, Eoin Brazil for excellent content support