Aggregating Data across Documents in MongoDB

MEETING THE MONGODB AGGREGATION FRAMEWORK

Axel Sirota
MACHINE LEARNING ENGINEER
@AxelSirota

What Is All That Fuzz with Aggregation?

With Aggregation You Go from This

```
{ "_id" : ObjectId("5e8936cd6347c0057a053363"), "name" : "Modern NYC",
"neighbourhood_cleansed" : "Washington Heights" }
{ "_id" : ObjectId("5e8936cd6347c0057a053364"), "name" : "Skylit Midtown Castle",
"neighbourhood_cleansed" : "Midtown" }
{ "_id" : ObjectId("5e8936cd6347c0057a053365"), "name" : "Cozy Entire Floor of
Brownstone", "neighbourhood_cleansed" : "Clinton Hill" }
{ "_id" : ObjectId("5e8936cd6347c0057a053366"), "name" : "Large Cozy 1 BR Apartment
In Midtown East", "neighbourhood_cleansed" : "Murray Hill" }
{ "_id" : ObjectId("5e8936cd6347c0057a053367"), "name" : "Super Room in Great area.",
"neighbourhood_cleansed" : "Lower East Side" }
```

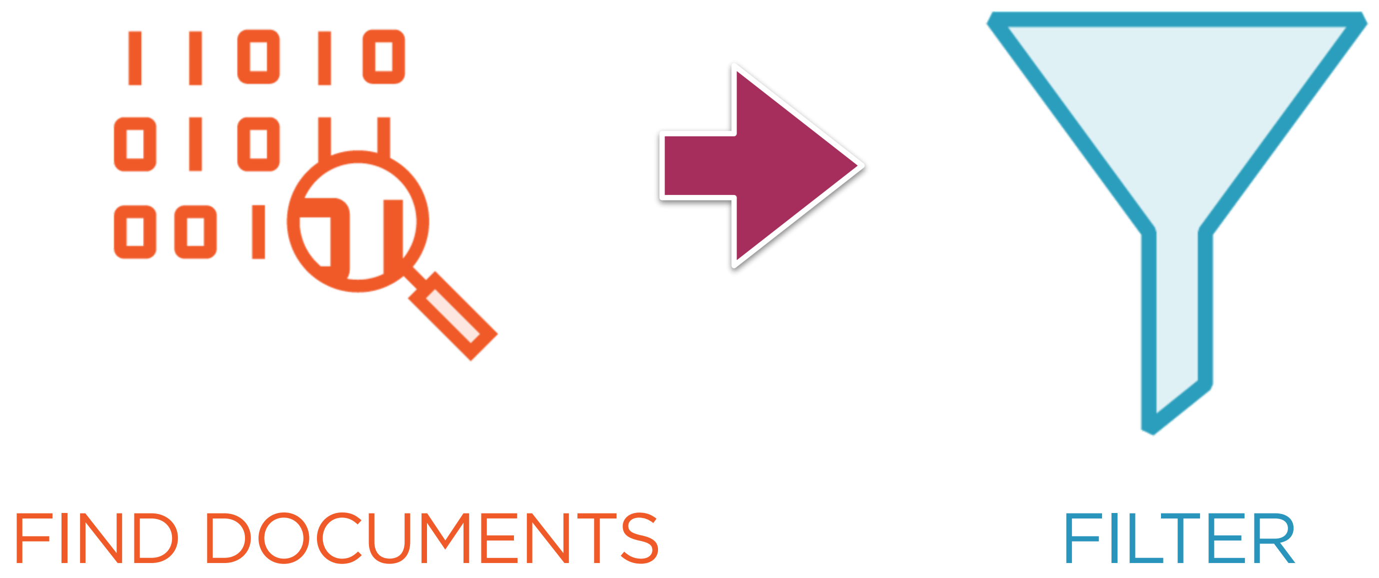
To This

```
> db.rent.aggregate([{$group: { _id: { zone: "$neighbourhood_cleansed"},  count: {$sum :
1}}}, {$sort: {count: -1}}, {$limit: 5}])
{ "_id" : { "zone" : "Williamsburg" }, "count" : 3844 }
{ "_id" : { "zone" : "Bedford-Stuyvesant" }, "count" : 3831 }
{ "_id" : { "zone" : "Harlem" }, "count" : 2753 }
{ "_id" : { "zone" : "Bushwick" }, "count" : 2498 }
{ "_id" : { "zone" : "Hell's Kitchen" }, "count" : 2143 }
```

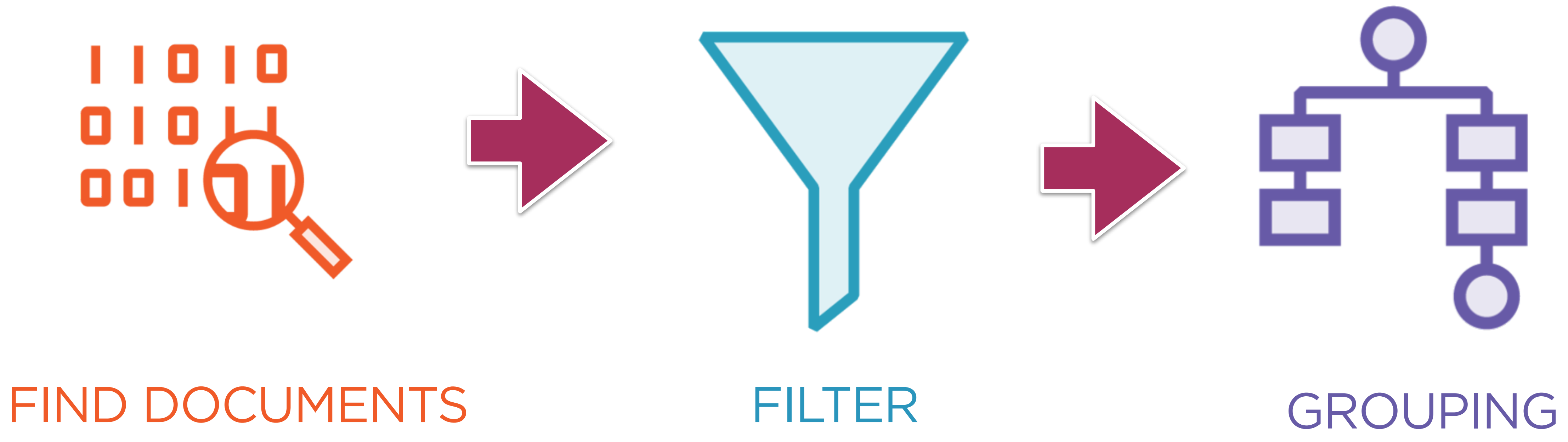
Aggregation at a Glance

FIND DOCUMENTS

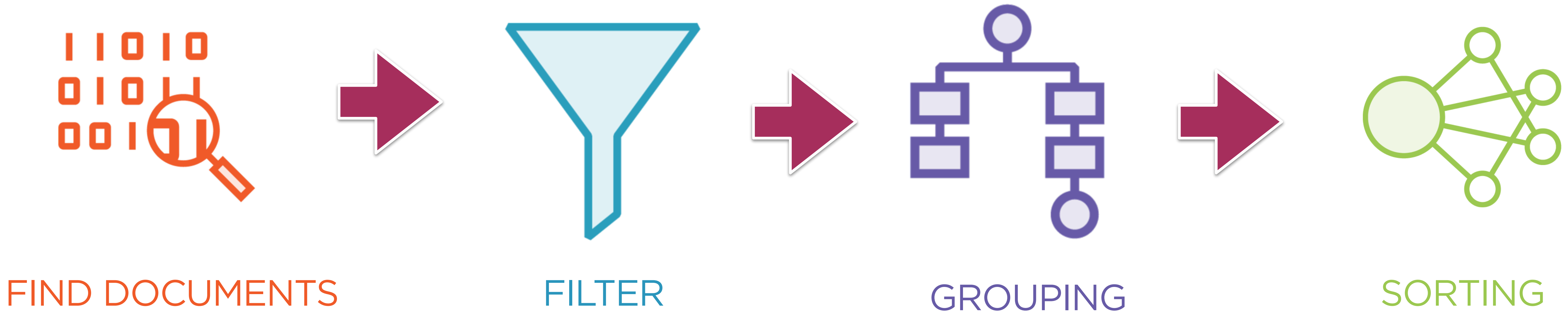
Aggregation at a Glance



Aggregation at a Glance



Aggregation at a Glance



Two Flavors of Aggregation

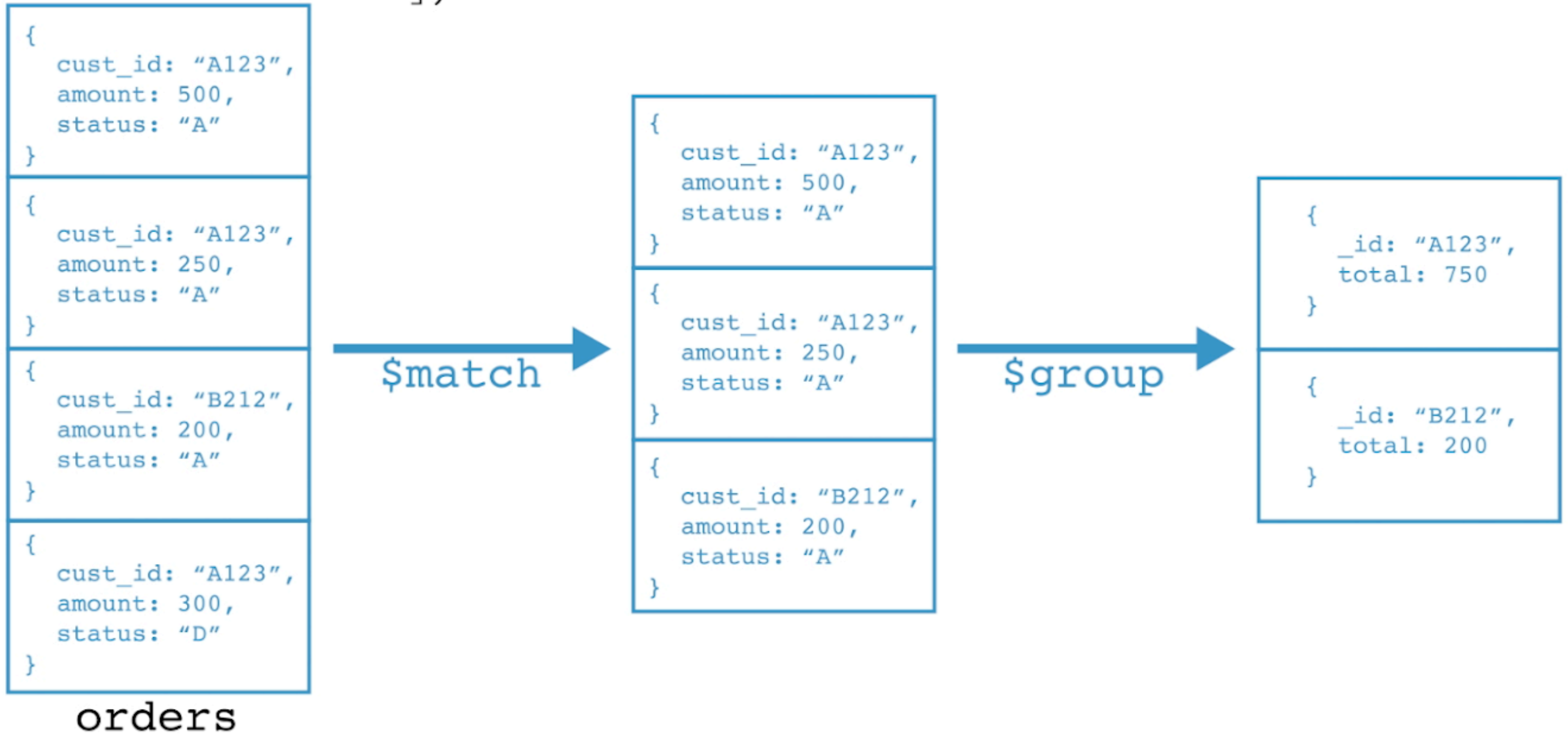
AGGREGATION FRAMEWORK

MAP REDUCE

An Example: Aggregation Framework

Collection

```
db.orders.aggregate( [
  $match stage  →{ $match: { status: "A" } },
  $group stage  →{ $group: { _id: "$cust_id",total:  {$sum: "$amount" } } }
                ])
```

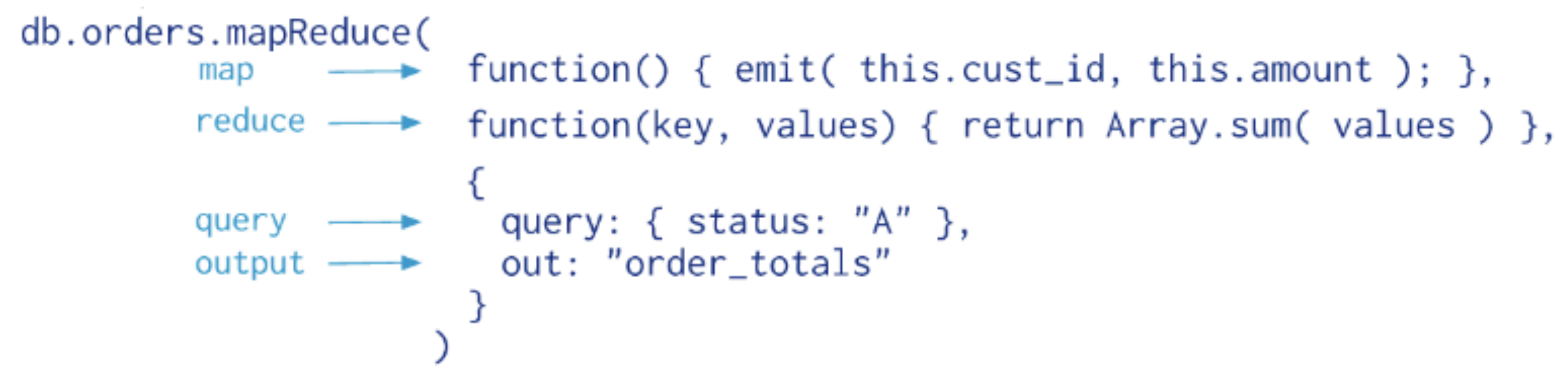


```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
```

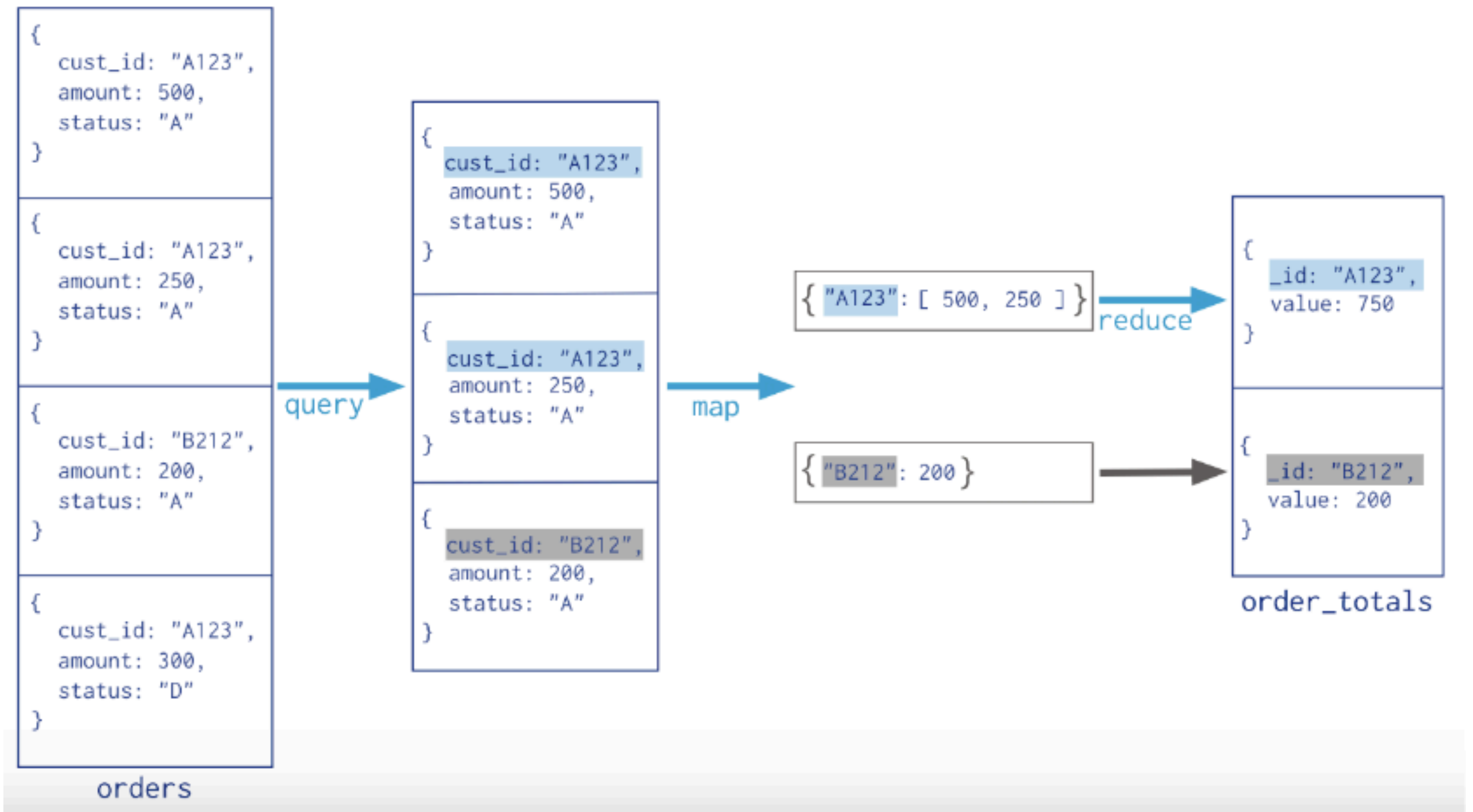
```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
```

```
{
  _id: "A123",
  total: 750
}
{
  _id: "B212",
  total: 200
}
```

An Example: Map Reduce



```
db.orders.mapReduce(
           map    ——→  function() { emit( this.cust_id, this.amount ); },
           reduce ——→  function(key, values) { return Array.sum( values ) },
                       {
           query  ——→    query: { status: "A" },
           output ——→    out: "order_totals"
                       }
                      )
```



```
orders:
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}

query ——→
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}

map ——→
{ "A123": [ 500, 250 ] }
{ "B212": 200 }

reduce ——→
order_totals:
{
  _id: "A123",
  value: 750
}
{
  _id: "B212",
  value: 200
}
```

A Difference in Concept

A Difference in Concept

Aggregation Framework

Map Reduce

A Difference in Concept

Aggregation Framework

- ✴ The operators are fixed

Map Reduce

- ✴ The collection is fixed

A Difference in Concept

Aggregation Framework

- ✳ The operators are fixed
- ✳The entire collection passes through pipeline

Map Reduce

- ✳ The collection is fixed
- ✳A function executed a subset of the collection

A More Detailed Comparison

A More Detailed Comparison

AGGREGATION FRAMEWORK

- Uses a “pipeline” approach
- Set of predefined pipeline operators
- Operators can be repeated
- Operators need not produce output
- Easy to setup
- Designed for small to mid-sized collections

A More Detailed Comparison

AGGREGATION FRAMEWORK

- Uses a “pipeline” approach
- Set of predefined pipeline operators
- Operators can be repeated
- Operators need not produce output
- Easy to setup
- Designed for small to mid-sized collections

MAP REDUCE

- Can perform complex or incremental aggregation
- Custom map, reduce and finalize functions
- Flexibility in output
- Complex to setup
- Designed for gigantic collections

95% of cases Aggregation Framework is THE choice

We will analyze the Aggregation Framework *a mondo*

To know more of Map Reduce check out: "Querying Data using Map-Reduce in MongoDB" course at Pluralsight!

Simple Reminder: CRUD in MongoDB

Inserting Data

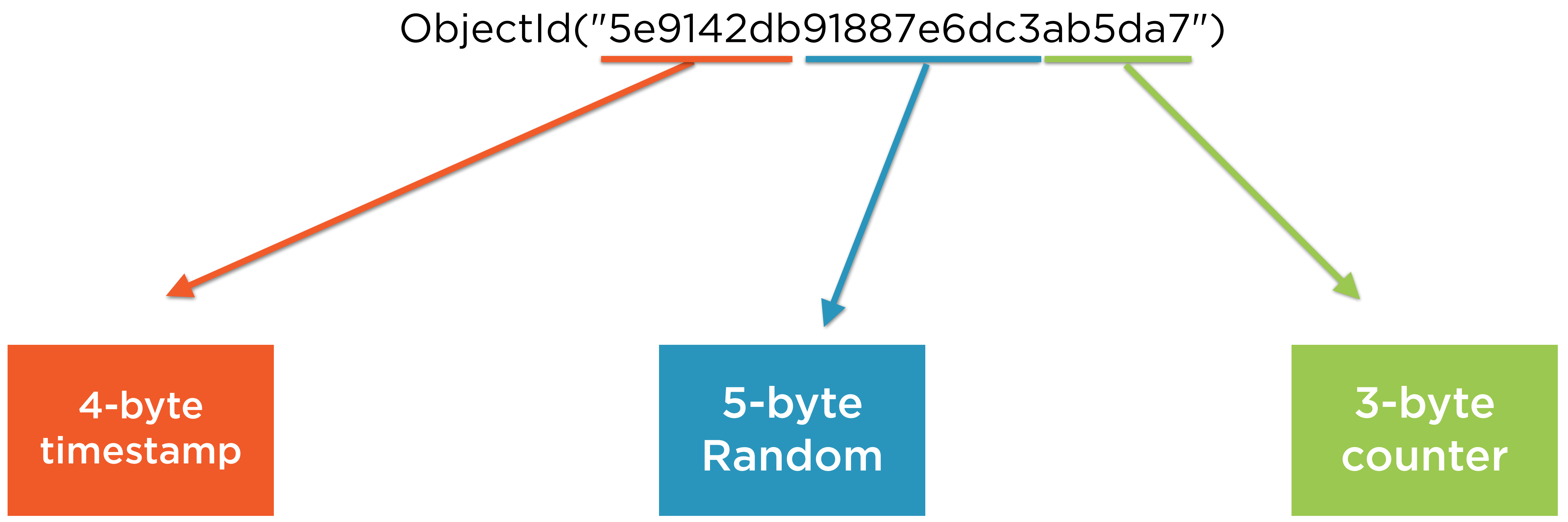
insertOne

insertMany

An Example

```
> db.inventory.insertOne(
...    { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }
... )
{
    "acknowledged" : true,
    "insertedId" : ObjectId("5e9142db91887e6dc3ab5da7")
}
```

The ObjectId



Another Example with Many

```
> db.inventory.insertMany([
...    { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
...    { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
...    { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }
... ])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5e91466391887e6dc3ab5da8"),
        ObjectId("5e91466391887e6dc3ab5da9"),
        ObjectId("5e91466391887e6dc3ab5daa")
    ]
}
```

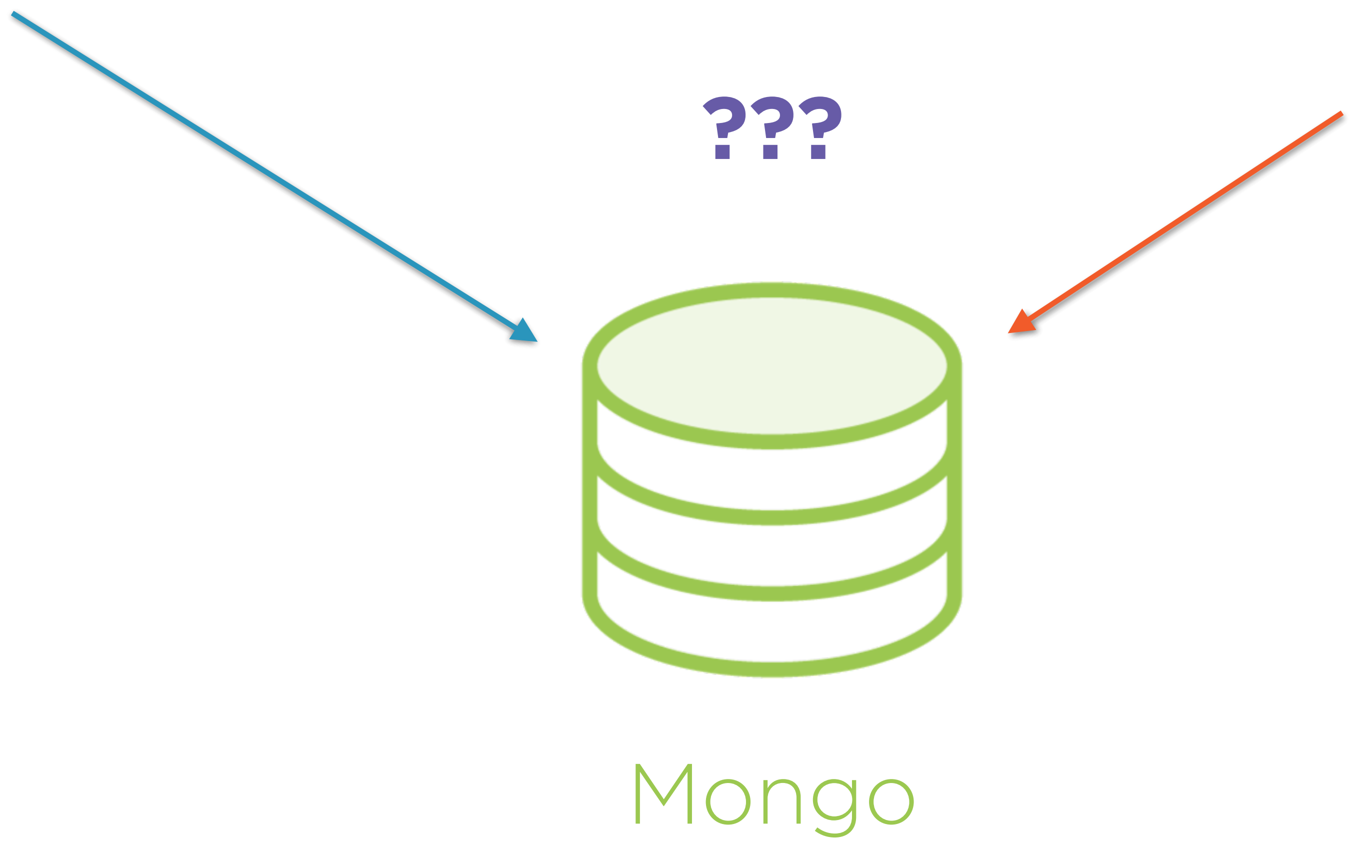
An Ordering App

Python

{ item: "tea", qty: 50,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }

Shell

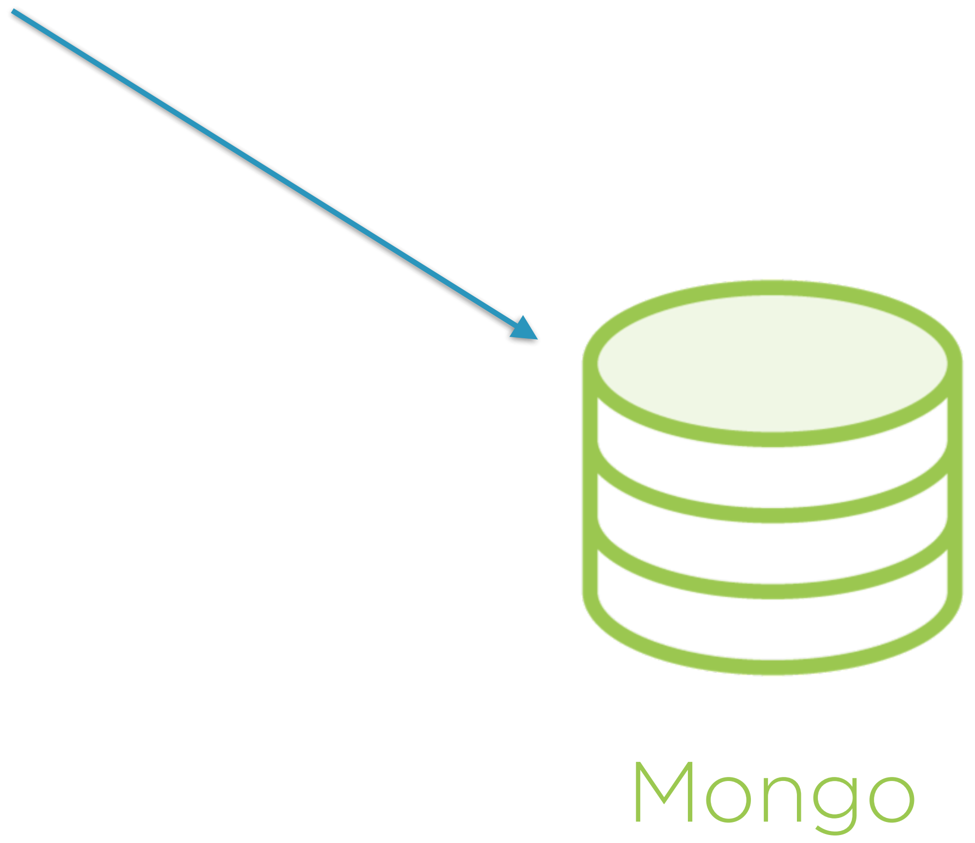
{ item: "coffee", qty: 12,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
{ item: "mate", qty: 60,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }



An Ordering App: Scenario 1

Python

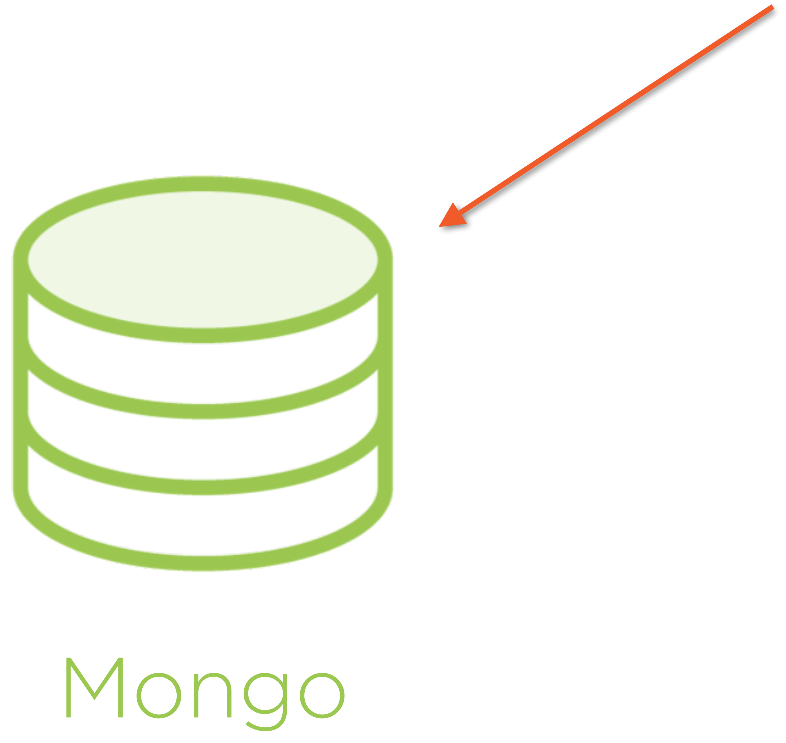
{ item: "tea", qty: 50,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }



An Ordering App: Scenario 1

Shell

```
{ item: "coffee", qty: 12,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
{ item: "mate", qty: 60,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
```



An Ordering App: Scenario 1

```
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5e91466391887e6dc3ab5da8"),
        ObjectId("5e91466391887e6dc3ab5da9")
    ]
}
```

An Ordering App: Scenario 2

Shell

```
{ item: "coffee", qty: 12,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
{ item: "mate", qty: 60,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
```

An Ordering App: Scenario 2

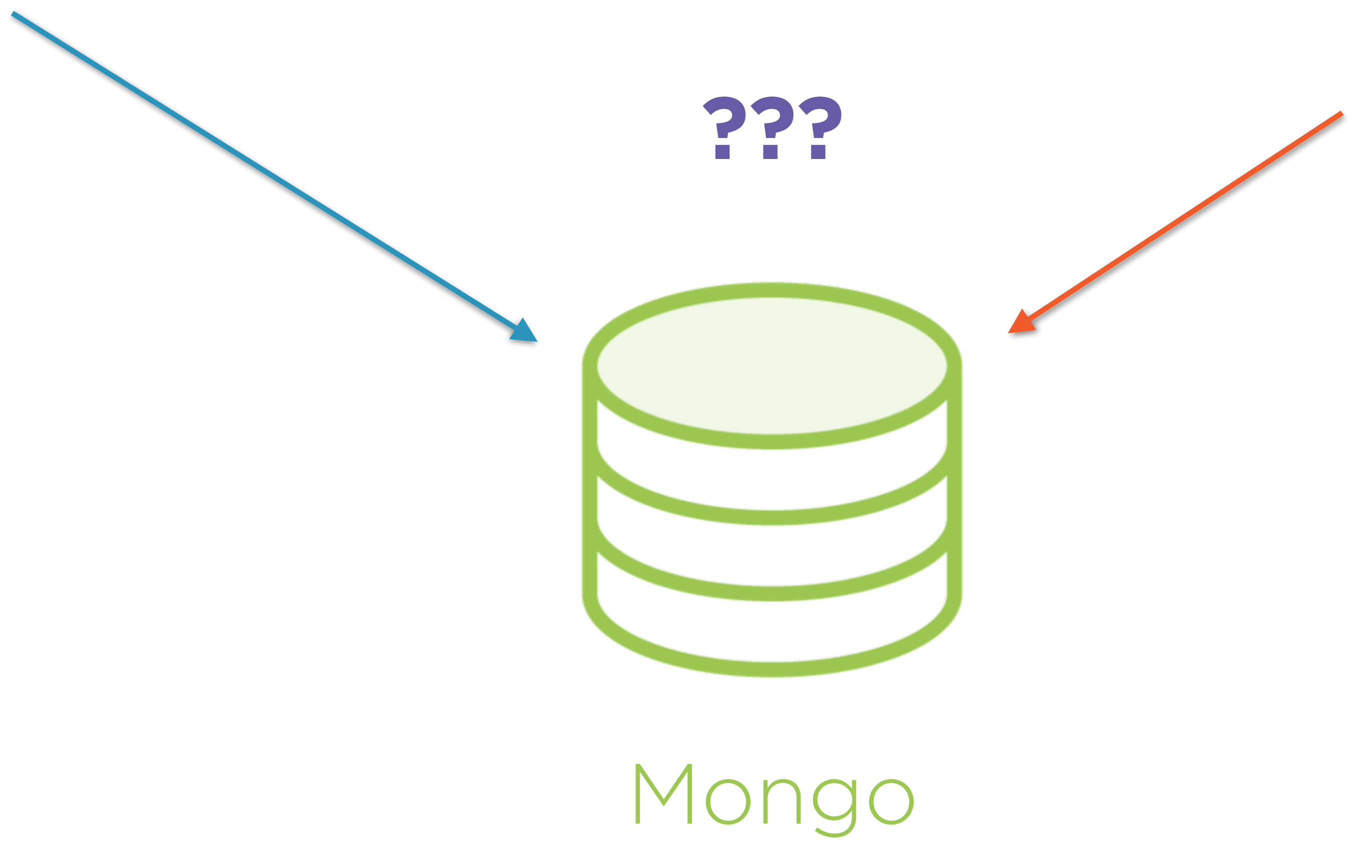
Python

{ item: "tea", qty: 50,size: { h: 12.9, w: 13.2, uom: "cm" } }

Shell

{ item: "coffee", qty: 12,size: { h: 12.9, w: 13.2, uom: "cm" } }

{ item: "mate", qty: 60,size: { h: 12.9, w: 13.2, uom: "cm" } }



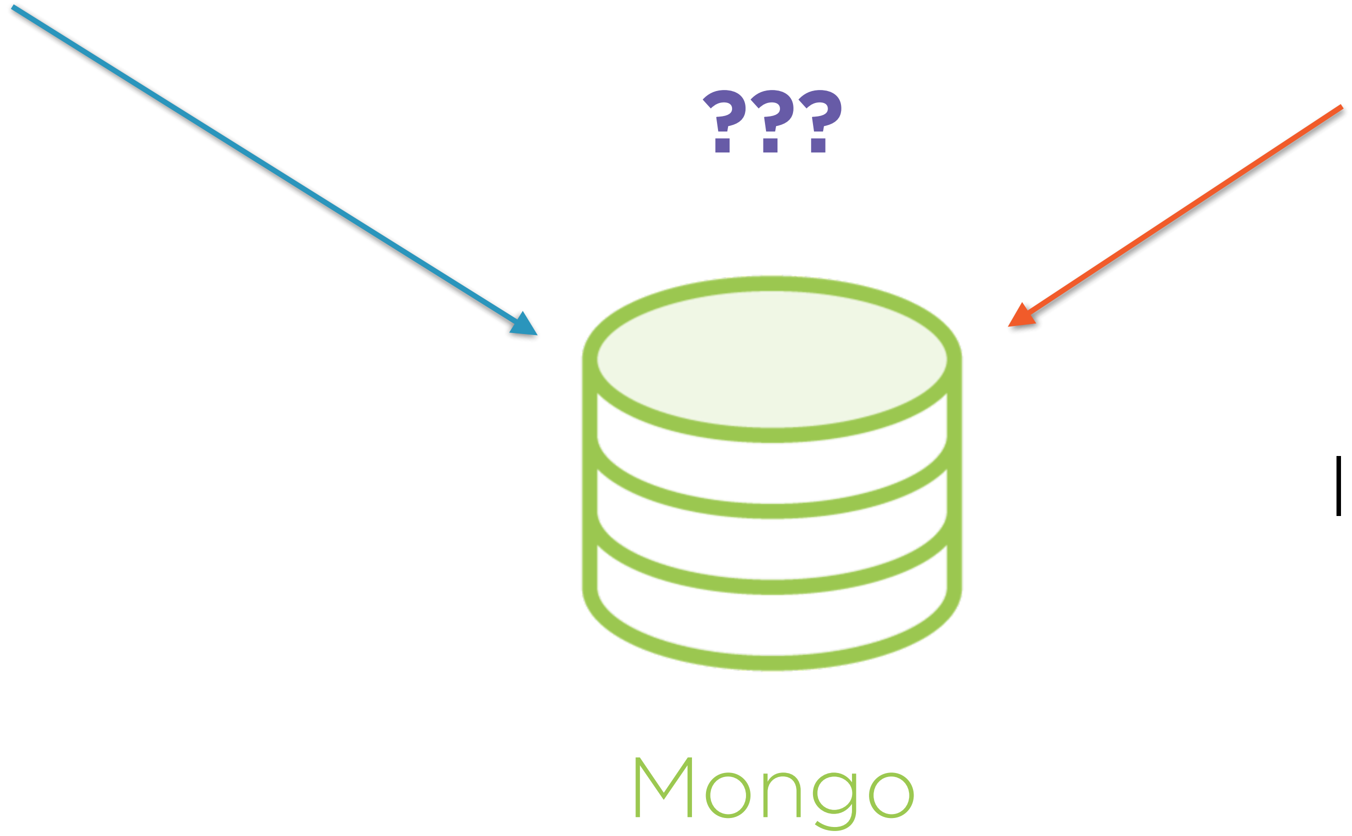
An Ordering App: Scenario 2

Python

```
{ item: "tea", qty: 50,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
```

Shell

```
{ item: "coffee", qty: 12,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
{ item: "mate", qty: 60,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
```



In SQL is clearly this one, but in Mongo it may happen...

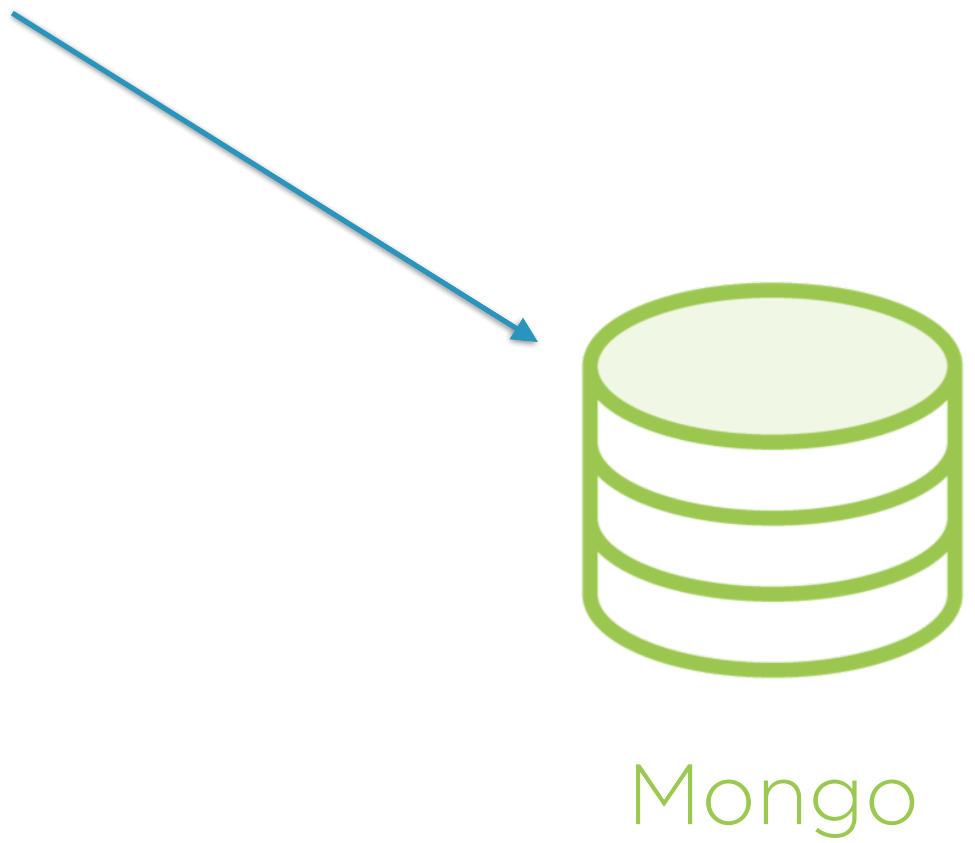
An Ordering App: Scenario 2

Python

```
{ item: "tea", qty: 50,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
```

Shell

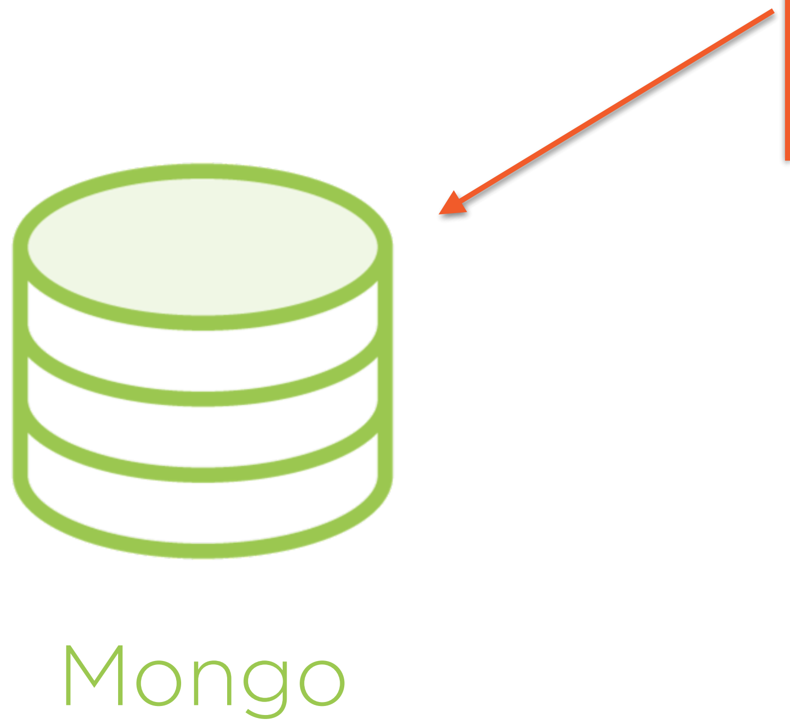
```
{ item: "coffee", qty: 12,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
{ item: "mate", qty: 60,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
```



An Ordering App: Scenario 2

Shell

```
{ item: "coffee", qty: 12,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
{ item: "mate", qty: 60,size:
{ h: 12.9, w: 13.2, uom:
"cm" } }
```



An Ordering App: Scenario 2

```
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5e91466391887e6dc3ab5da8"),
# Note there is no 5e91466391887e6dc3ab5da89!!!
        ObjectId("5e91466391887e6dc3ab5daa")
    ]

}
```

Querying Data

findOne

Find

More on this topic on the course **Querying Data from MongoDB**

Finding a Document

Finding a Document

Making the query in Mongo

```
> db.inventory.find({item: "journal"})
{
 "_id" : ObjectId("5e91466391887e6dc3ab5da8"),
"item" : "journal", "qty" : 25, "tags" : [ "blank",
"red" ], "size" : { "h" : 14, "w" : 21, "uom" : "cm" }
}
```

Finding a Document

Making the query in Mongo

```
> db.inventory.find({item: "journal"})
{
 "_id" : ObjectId("5e91466391887e6dc3ab5da8"),
"item" : "journal", "qty" : 25, "tags" : [ "blank",
"red" ], "size" : { "h" : 14, "w" : 21, "uom" : "cm" }
}
```

SQL equivalent

```
SELECT * FROM INVENTORY WHERE ITEM = “journal”
```

Another Example with Find

```
> db.inventory.find({tags: {$exists: true } }, {_id:0, item:1, tags:1})
{ "item" : "canvas", "tags" : [ "cotton" ] }
{ "item" : "journal", "tags" : [ "blank", "red" ] }
{ "item" : "mat", "tags" : [ "gray" ] }
{ "item" : "mousepad", "tags" : [ "gel", "blue" ] }
```

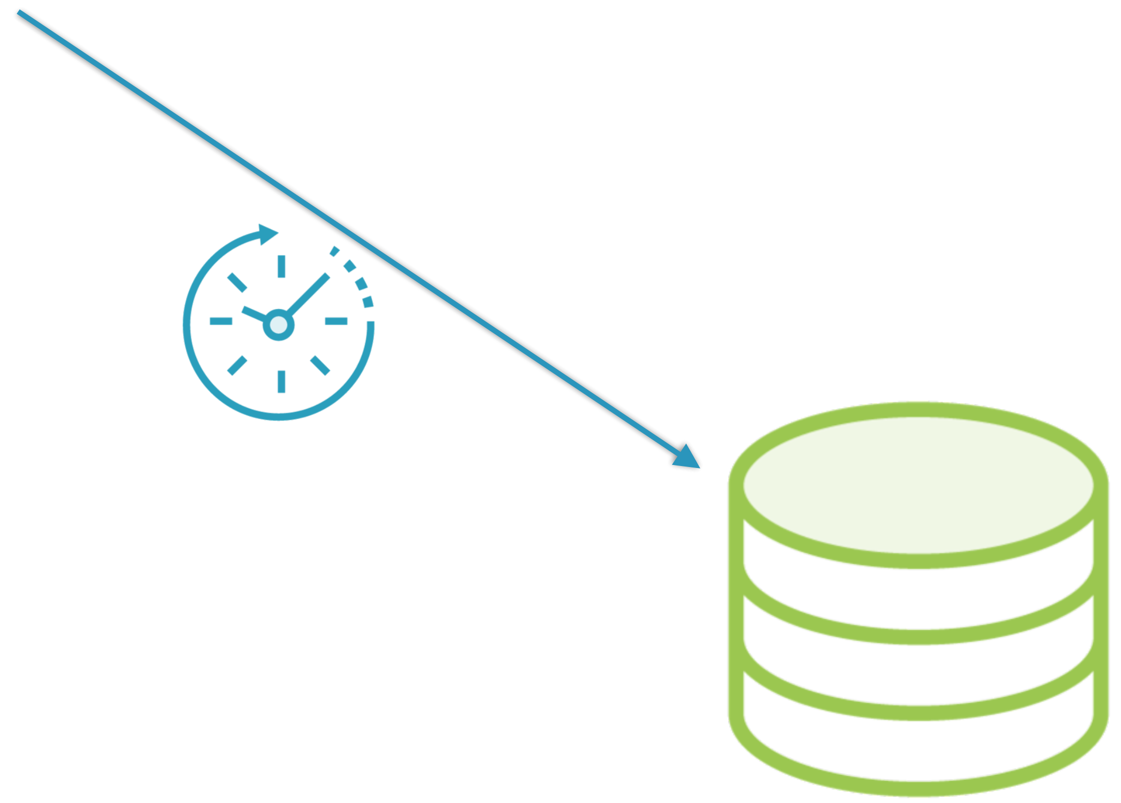
WiredTiger in Action

Query

```
db.inventory.find({tags: {$exists:
true } }, {_id:0, item:1, tags:1})
```

Insert

```
db.inventory.insertOne({tags:
['orange', 'cheap'], name:
"Oranges from the farm" })
```



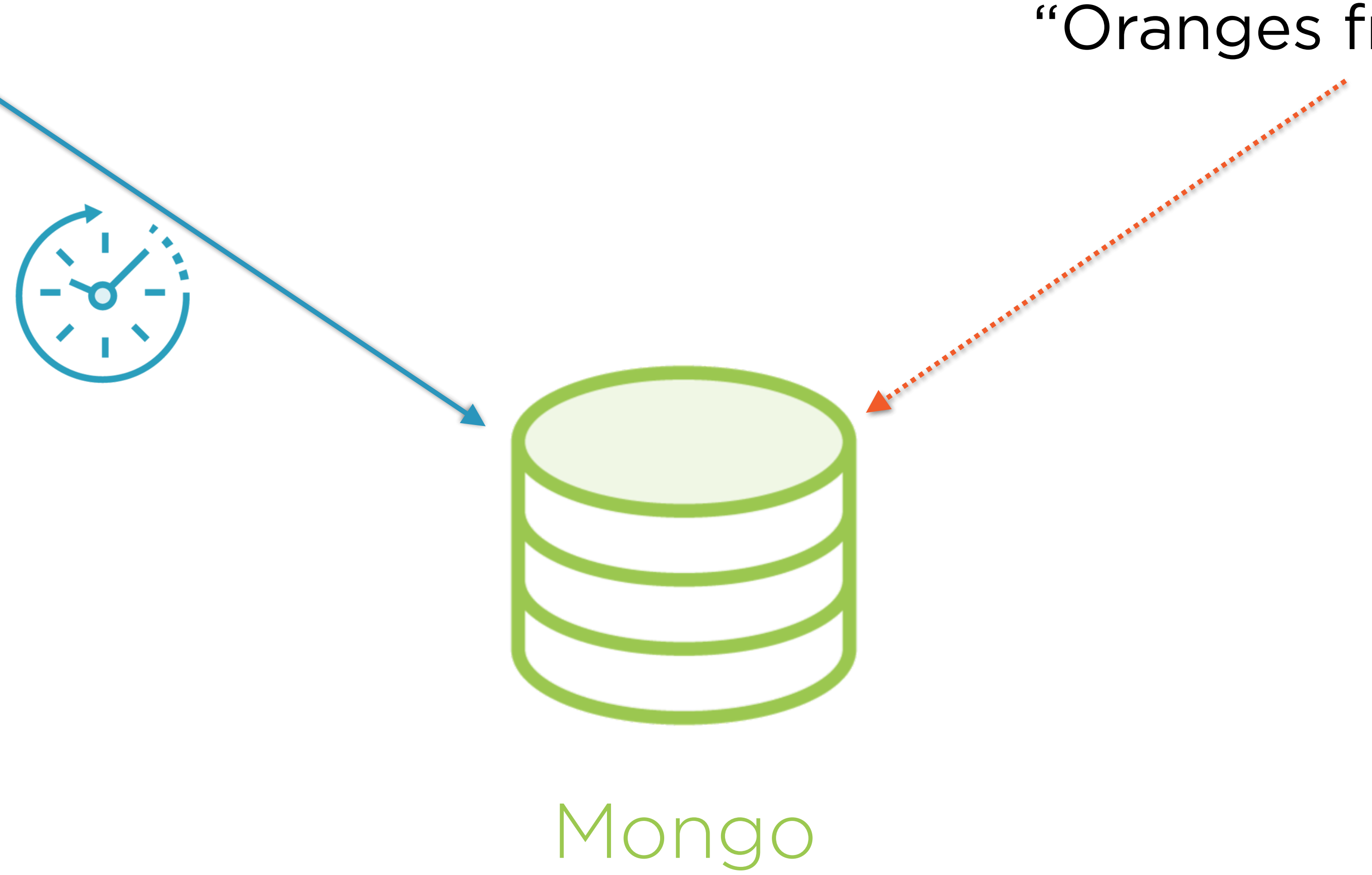
WiredTiger in Action

Query

```
db.inventory.find({tags: {$exists:
true } }, {_id:0, item:1, tags:1})
```

Insert

```
db.inventory.insertOne({tags:
['orange', 'cheap'], name:
"Oranges from the farm" })
```



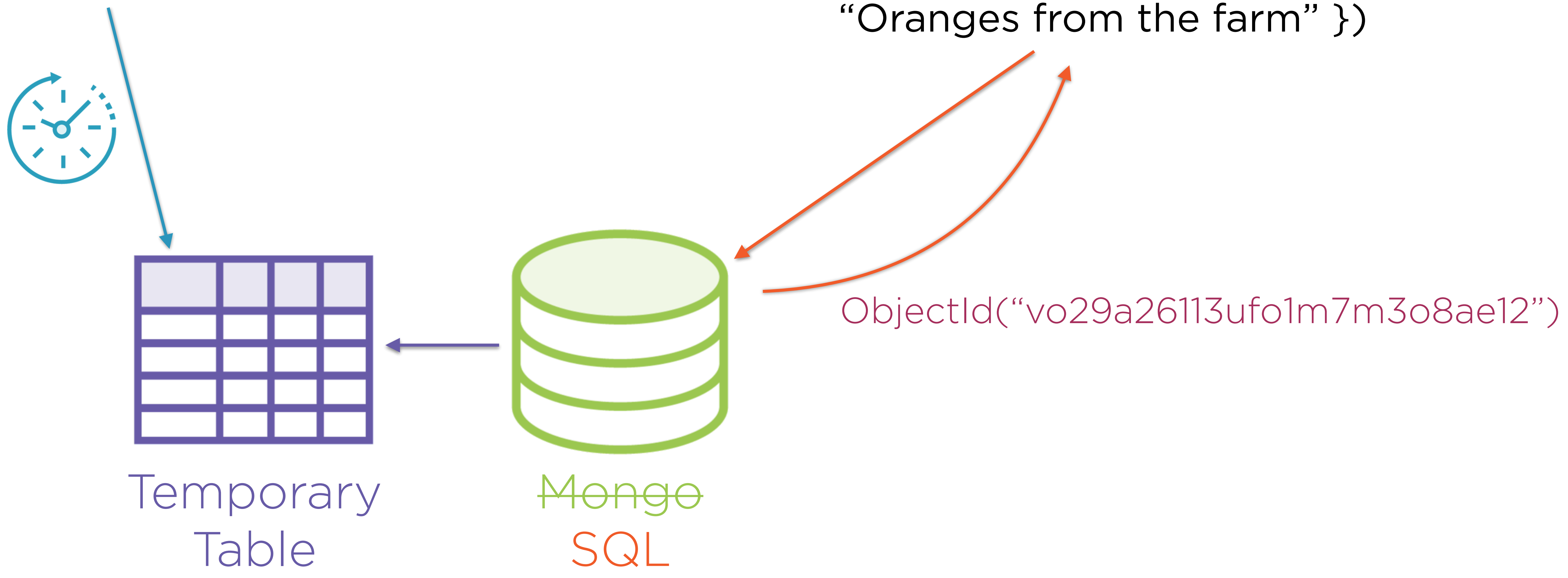
WiredTiger in Action

Query

```
db.inventory.find({tags: {$exists:
true } }, {_id:0, item:1, tags:1})
```

Insert

```
db.inventory.insertOne({tags:
['orange', 'cheap'], name:
"Oranges from the farm" })
```



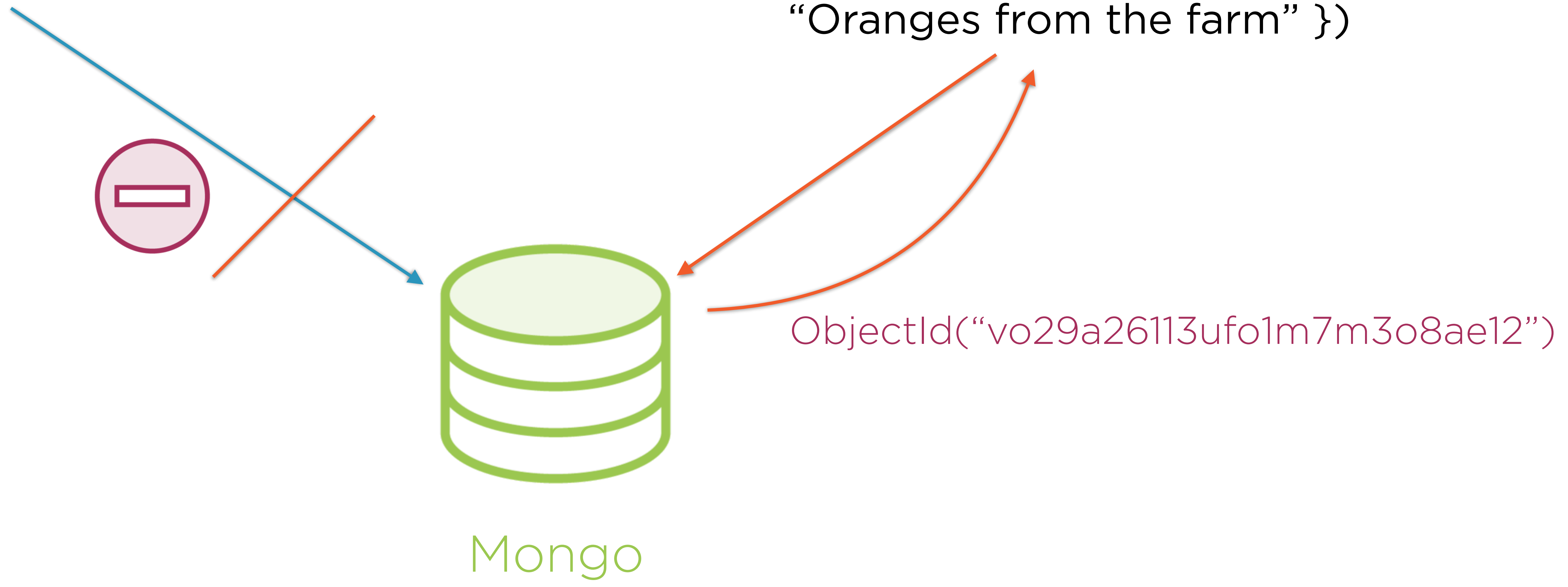
WiredTiger in Action

Query

```
db.inventory.find({tags: {$exists:
true } }, {_id:0, item:1, tags:1})
```

Insert

```
db.inventory.insertOne({tags:
['orange', 'cheap'], name:
"Oranges from the farm" })
```

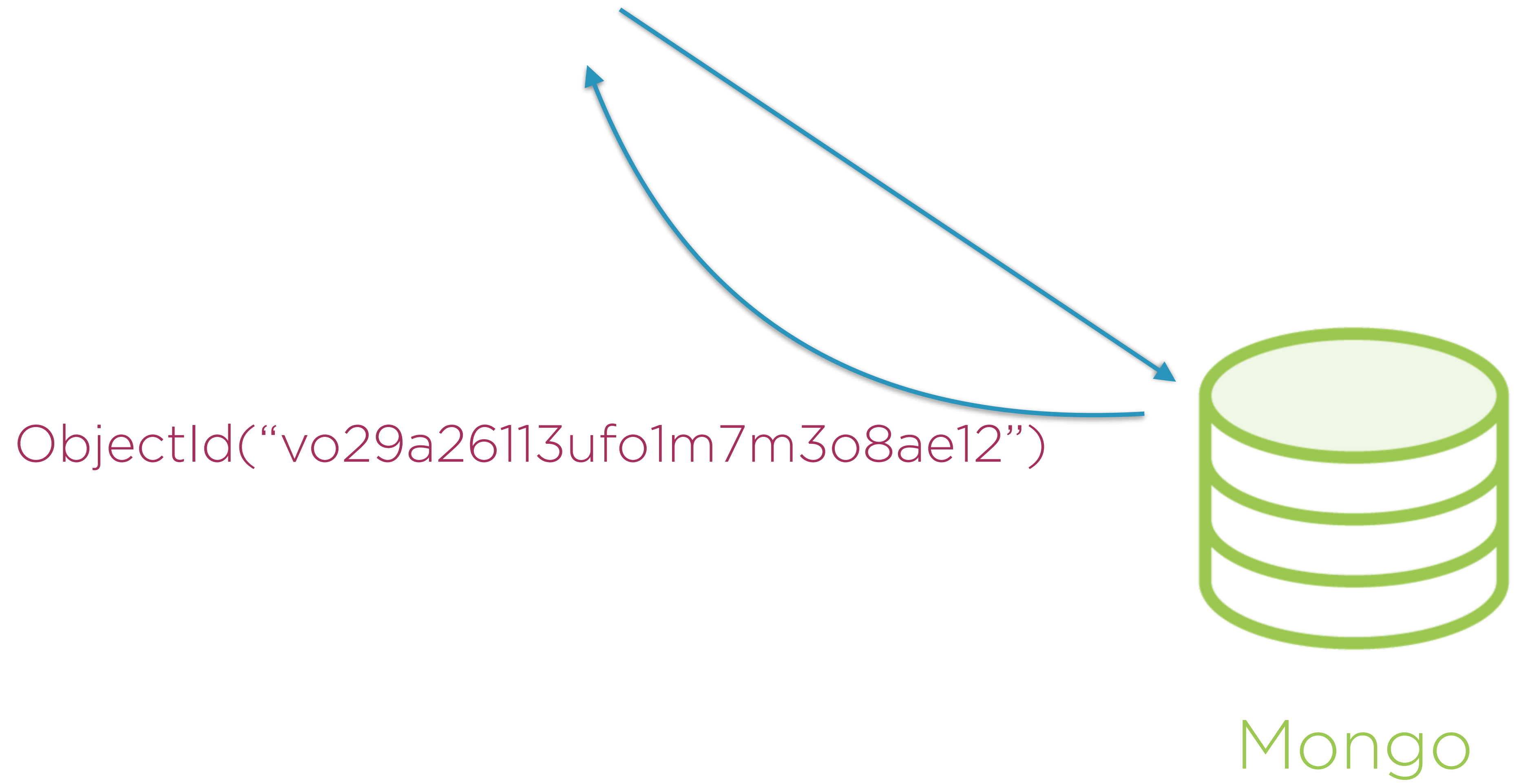


```
ObjectId("vo29a26113ufo1m7m3o8ae12")
```

WiredTiger in Action

Query

db.inventory.find({tags: {$exists: true } }, {_id:1, item:0, tags:0})



Updating Data

updateOne

updateMany

replaceOne

Example with Update

```
>  db.inventory.updateMany(
...    { "qty": { $lt: 50 } },
...    {
...      $set: { "size.uom": "in", status: "P" },
...      $currentDate: { lastModified: true }
...    },
...   {upsert: true}
... )
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
> db.inventory.find({ "qty": { $lt: 50 } })
{ "_id" : ObjectId("5e91466391887e6dc3ab5da8"), "item" : "journal", "qty" : 25, "tags" :
[ "blank", "red" ], "size" : { "h" : 14, "w" : 21, "uom" : "in" }, "lastModified" :
ISODate("2020-04-11T15:43:13.287Z"), "status" : "P" }
{ "_id" : ObjectId("5e91466391887e6dc3ab5daa"), "item" : "mousepad", "qty" : 25, "tags" :
[ "gel", "blue" ], "size" : { "h" : 19, "w" : 22.85, "uom" : "in" }, "lastModified" :
ISODate("2020-04-11T15:43:13.287Z"), "status" : "P" }
```

Deleting

```
> db.inventory.deleteMany( { qty: {$lt: 120} } )
{ "acknowledged" : true, "deletedCount" : 1 }
```

FIND DOCUMENTS

Deleting

```
> db.inventory.deleteMany( { qty: {$lt: 120} } )
{ "acknowledged" : true, "deletedCount" : 1 }
```

FIND DOCUMENTS

DELETES THE IDS

Demo

Demo

Demo

Do basic CRUD operations on a collection in MongoDB

Demo

Do basic CRUD operations on a collection in MongoDB

Remembering filters, projections and sets

Basic Aggregations: $match and $project

Some Stages to Recall

$match

$project

$group

$limit/$skip

$unwind

$sort

The $match Stage

✓

✓

✓

✓

The $match Stage

```
{ $match: { <query> } }
```

The $match Stage

- ✓ `{ $match: { <query> } }`
- ✓ The filter can have any expression or search
- ✓
- ✓

The $match Stage

- { $match: { <query> } }
- The filter can have any expression or search
- It leverages indexes!

The $match Stage

- ✓ `{ $match: { <query> } }`
- ✓ The filter can have any expression or search
- ✓ It leverages indexes!
- ✓ Better to use early in the pipeline

Some Examples

```
> db.rent.aggregate([{$match:
{$text: {$search: "Tribeca"}}}])
```

```
> db.rent.aggregate([{$match:
{neighbourhood_cleansed:
"Tribeca"}}])
```

For a reminder on full-text search check **“Searching for Text in MongoDB”**

A More Complex $match

```
> db.rent.aggregate([
   {$match: {
      $text: {$search: "Tribeca"},
      $and: [
         {reviews_per_month: { $gt : 1}},
         { minimum_nights: {$gt : 3 } }
      ]
   }}
])
```

◀ **# Full text search**
◀ **# More than 1 review per month**
◀ **# At least 3 nights**

What Do We Gain?

```
> db.rent.findOne({_id: new ObjectId("5e8936cf6347c0057a0563c7")}, {name:1,
host_response_rate:1, price:1, cleaning_fee: 1})
{
    "_id" : ObjectId("5e8936cf6347c0057a0563c7"),
    "name" : "Tribeca/Soho Garden Apartment",
    "host_response_rate" : "100%",
    "price" : "$400.00",
    "cleaning_fee" : "$100.00"
}
```

What Do We Gain?

```
> db.rent.findOne({_id: new ObjectId("5e8936cf6347c0057a0563c7")}, {name:1,
host_response_rate:1, price:1, cleaning_fee: 1})
{
    "_id" : ObjectId("5e8936cf6347c0057a0563c7"),
    "name" : "Tribeca/Soho Garden Apartment",
    "host_response_rate" : "100%",
    "price" : "$400.00",
    "cleaning_fee" : "$100.00"
}
```

Here is where **$project** comes to the rescue!

The $project Stage

```
$project: {
    ...,
    neighbourhood: "$neighbourhood_cleansed",
    ...,
}
```

◀ # We reference neighbourhood_cleansed and rename it as neighbourhood

# If we had not used $, it would hardcode the string "neighbourhood_cleansed"

What Does $ Mean?

$$\text{\$name} \xrightarrow{\text{Mongo}} \text{\$\$CURRENT.name}$$

What Does $ Mean?

$$\begin{tikzcd}
 & \text{\{name: “Pepito”, surname: “Laguna”\}} \arrow[dl] \arrow[dr] & \\
\text{\$ = \{name: “Pepito”, surname: “Laguna”\}} & & \text{\$name = “Pepito”}
\end{tikzcd}$$

The Power of $project

{name: “Pepito”,
surname: “Laguna”}

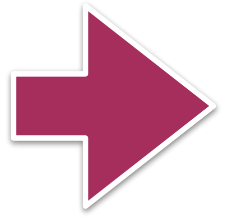
{name:”Sacarias”,
surname: “Flores del
campo”}

```
{
    $project: {
        complete_name: {
            $concat: [
                “$name”,
                “ ”,
                “$surname”
            ]
        }
    }
}
```

The Power of $project

```
{name: "Pepito",
surname: "Laguna"}

{name:"Sacarias",
surname: "Flores del
campo"}
```



```
{
    $project: {
        complete_name: {
            $concat: [
                "$name",
                " ",
                "$surname"
            ]
        }
    }
}
```

$ = {name: "Pepito", surname: "Laguna"}

$.name: "Pepito"
$.surname: "Laguna"

The Power of $project

{name: “Pepito”,
surname: “Laguna”}

{name:”Sacarias”,
surname: “Flores del
campo”}

```
{
    $project: {
        complete_name: {
            $concat: [
                “$name”,
                “ ”,
                “$surname”
            ]
        }
    }
}
```

{complete_name: “Pepito
Laguna”}

$ = {name: “Pepito”, surname: “Laguna”}

$.name: “Pepito”
$.surname: “Laguna”

The Power of $project

{name: “Pepito”,
surname: “Laguna”}

{name:”Sacarias”,
surname: “Flores del
campo”}

```
{
    $project: {
        complete_name: {
            $concat: [
                “$name”,
                “ ”,
                “$surname”
            ]
        }
    }
}
```

The Power of $project

{name: “Pepito”,
surname: “Laguna”}

{name:”Sacarias”,
surname: “Flores del
campo”}

```
{
    $project: {
        complete_name: {
            $concat: [
                “$name”,
                “ ”,
                “$surname”
            ]
        }
    }
}
```

$ = {name: “Sacarias”, surname: “Flores del Campo”}

$.name: “Sacarias”
$.surname: “Flores del Campo”

The Power of $project

{name: “Pepito”,
surname: “Laguna”}

{name:”Sacarias”,
surname: “Flores del
campo”}

```
{
    $project: {
        complete_name: {
            $concat: [
                "$name",
                " ",
                "$surname"
            ]
        }
    }
}
```

{complete_name: “Pepito
Laguna”}

{complete_name:
“Sacarias Flores del
Campo”}

$ = {name: “Sacarias”, surname: “Flores del Campo”}

$.name: “Sacarias”
$.surname: “Flores del Campo”

A Complex Example

```
{
    "_id" :
ObjectId("5e8936cf6347c0057a0563c7"),
    "name" : "Tribeca/Soho Garden Apartment",
    "host_response_rate" : "100%",
    "price" : "$400.00",
    "cleaning_fee" : "$100.00"
}
```

◀ **# How do we remove the ‘%’?**

A Complex Example

```
$project: {
    ...,
    num_accept_rate: {
        $toDouble: {
            $substr: [
                "$host_acceptance_rate",
                0,
                {
                     $subtract: [{$strLenCP:
"$host_acceptance_rate" },1]
                 }
            ]
        }
    }
}
```

- # We transform to a number
- # The substring without the %
- # Which is from the start to length -1
- # That we get by subtracting 1 from the length!

This way we transform '38%' to 38

The $match stage takes a filter and filters out documents that do not match

The $project stage lets us transform the data

Which is key for performance

For more resources on operators, check here

Demo

Demo

Demo

Use $match for filtering bad cases

Demo

Use $match for filtering bad cases

Delve into $project for wild formats

Demo

Use $match for filtering bad cases

Delve into $project for wild formats

Learn about $addFields, $sort and $limit stages

Summary

Summary

Summary

Learned about different paradigms for aggregation in MongoDB

Summary

Learned about different paradigms for aggregation in MongoDB

Revisited CRUD operations

Summary

Learned about different paradigms for aggregation in MongoDB

Revisited CRUD operations

Learned about $match and $project stages