Handling Advanced Operations in Aggregation

Axel Sirota
MACHINE LEARNING ENGINEER
@AxelSirota

Categorizing Documents on the Fly with $bucket!

Aggregate Artists by Year Born

```
db.artists.insertMany([
...   { "_id" : 1, "last_name" : "Bernard", "first_name" : "Emil", "year_born" : 1868, "year_died" :
1941, "nationality" : "France" },
...   { "_id" : 2, "last_name" : "Rippl-Ronai", "first_name" : "Joszef", "year_born" : 1861, "year_died"
: 1927, "nationality" : "Hungary" },
...   { "_id" : 3, "last_name" : "Ostroumova", "first_name" : "Anna", "year_born" : 1871,
"year_died" : 1955, "nationality" : "Russia" },
...   { "_id" : 4, "last_name" : "Van Gogh", "first_name" : "Vincent", "year_born" : 1853, "year_died"
: 1890, "nationality" : "Holland" },
...   { "_id" : 5, "last_name" : "Maurer", "first_name" : "Alfred", "year_born" : 1868, "year_died" :
1932, "nationality" : "USA" },
...   { "_id" : 6, "last_name" : "Munch", "first_name" : "Edvard", "year_born" : 1863, "year_died" :
1944, "nationality" : "Norway" },
...   { "_id" : 7, "last_name" : "Redon", "first_name" : "Odilon", "year_born" : 1840, "year_died" :
1916, "nationality" : "France" },
...   { "_id" : 8, "last_name" : "Diriks", "first_name" : "Edvard", "year_born" : 1855, "year_died" :
1930, "nationality" : "Norway" }
... ])
```

Projecting Years into Decades

```
{
    $project: {
        "_id" : 1,
        "last_name" : 1,
        "first_name" : 1,
        "decade_born" : { $multiply: [ { $trunc:
[ { $divide: [ "$year_born" , 10 ] } , 0] }, 10 ] } ,
        "year_died" : 1,
        "nationality" : 1
    }
}
```

- **# If we divide by 10 and truncate we map 1861 -> 186.1 -> 186**
- **# If we multiply by 10 later on, we would map 1861 -> 1860!**

Aggregate Artists by Year Born

{ "_id" : 1, "last_name" : "Bernard", "first_name" : "Emil", "year_died" : 1941, "nationality" : "France", **"decade_born" : 1860** }
{ "_id" : 2, "last_name" : "Rippl-Ronai", "first_name" : "Joszef", "year_died" : 1927, "nationality" : "Hungary", **"decade_born" : 1860** }
{ "_id" : 3, "last_name" : "Ostroumova", "first_name" : "Anna", "year_died" : 1955, "nationality" : "Russia", **"decade_born" : 1870** }
{ "_id" : 4, "last_name" : "Van Gogh", "first_name" : "Vincent", "year_died" : 1890, "nationality" : "Holland", **"decade_born" : 1850** }
{ "_id" : 5, "last_name" : "Maurer", "first_name" : "Alfred", "year_died" : 1932, "nationality" : "USA", **"decade_born" : 1860** }
{ "_id" : 6, "last_name" : "Munch", "first_name" : "Edvard", "year_died" : 1944, "nationality" : "Norway", **"decade_born" : 1860** }
{ "_id" : 7, "last_name" : "Redon", "first_name" : "Odilon", "year_died" : 1916, "nationality" : "France", **"decade_born" : 1840** }
{ "_id" : 8, "last_name" : "Diriks", "first_name" : "Edvard", "year_died" : 1930, "nationality" : "Norway", **"decade_born" : 1850** }

Group by Decade

```
{
    $group: {
        _id: {
            decade_born: "$decade_born"
        },
        count: { $sum : 1},
        artist: {
            $addToSet: {
                name: { $concat: [ "$first_name" , "$last_name" ] },
                year_born: "$year_born",
                year_died: "$year_died",
                nationality: "$nationality"
            }
        }
    }
}
```

- # Now we group by decade!
- # Get the count
- # And lets not forget to have an array of the artists in that group!

Aggregate Artists by Year Born

{ "_id" : { "decade_born" : 1850 }, "count" : 2, "artist" : [{ "name" : "EdvardDiriks", "year_died" : 1930, "nationality" : "Norway" }, { "name" : "VincentVan Gogh", "year_died" : 1890, "nationality" : "Holland" }] }
{ "_id" : { "decade_born" : 1860 }, "count" : 4, "artist" : [{ "name" : "JoszefRippl-Ronai", "year_died" : 1927, "nationality" : "Hungary" }, { "name" : "EdvardMunch", "year_died" : 1944, "nationality" : "Norway" }, { "name" : "AlfredMaurer", "year_died" : 1932, "nationality" : "USA" }, { "name" : "EmilBernard", "year_died" : 1941, "nationality" : "France" }] }
{ "_id" : { "decade_born" : 1840 }, "count" : 1, "artist" : [{ "name" : "OdilonRedon", "year_died" : 1916, "nationality" : "France" }] }
{ "_id" : { "decade_born" : 1870 }, "count" : 1, "artist" : [{ "name" : "AnnaOstroumova", "year_died" : 1955, "nationality" : "Russia" }] }

Some Things to Consider

What if we couldn’t **“easily”** project the field (decade in this case)

If there are decades without artists?

If the boundaries are in another scale?

Same Query with $bucket

```
{
      $bucket: {
        groupBy: "$year_born",
        boundaries: [ 1840, 1850, 1860, 1870,
1880 ],
        default: "Other",
        output: {
          count: { $sum: 1 },
          artists :
            {
              $addToSet: {
                name: { $concat: [ "$first_name", " ",
"$last_name"] },
                year_born: "$year_born",
                year_died: "$year_died",
                nationality: "$nationality"
}}}}}
```

- # Now we group by year_born
- # And assign to these buckets
- # If it isn't in a bucket assign to bucket "Other"
- # Do the same group calculations as before

Best Buckets with $bucketAuto

```
db.artists.aggregate([
  {
    $bucketAuto: {
      groupBy: "$year_born",
      buckets: 5,
      output: {
        count: { $sum: 1 },
      }
    }
  }
])
```

◀ **# Group by year_born**
◀ **# I want 5 buckets**

◀ **# And only report the count**

Thats it! Mongo will find out the best buckets!

Automatic Buckets

```
{ "_id" : { "min" : 1840, "max" : 1855 }, "count" : 2 }
{ "_id" : { "min" : 1855, "max" : 1863 }, "count" : 2 }
{ "_id" : { "min" : 1863, "max" : 1871 }, "count" : 3 }
{ "_id" : { "min" : 1871, "max" : 1871 }, "count" : 1 }
```

Granularities for $bucketAuto

1,2,5

0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, etc...

POWERSOF2

1, 2, 4, 8, 16, 32, 64, etc...

Comparing Granularities

Using **$bucketAuto** with default granularity

```
$bucketAuto: {
    groupBy: "$num_price",
    buckets: 10,
    output: {
      count: { $sum: 1 },
    }}
```

Comparing Granularities

Using **$bucketAuto** with default granularity

```
$bucketAuto: {
    groupBy: "$num_price",
    buckets: 10,
    output: {
      count: { $sum: 1 },
    }}
```

```
{ "_id" : { "min" : 0, "max" : 50 }, "count" : 5330 }
{ "_id" : { "min" : 50, "max" : 61 }, "count" : 5376 }
{ "_id" : { "min" : 61, "max" : 76 }, "count" : 5313 }
{ "_id" : { "min" : 76, "max" : 96 }, "count" : 5682 }
{ "_id" : { "min" : 96, "max" : 116 }, "count" : 5490 }
{ "_id" : { "min" : 116, "max" : 141 }, "count" : 5183 }
{ "_id" : { "min" : 141, "max" : 176 }, "count" : 5832 }
{ "_id" : { "min" : 176, "max" : 226 }, "count" : 5242 }
{ "_id" : { "min" : 226, "max" : 396 }, "count" : 5100 }
{ "_id" : { "min" : 396, "max" : 10000 }, "count" : 2248 }
```

Comparing Granularities

Using **$bucketAuto** with 1, 2, 5 granularity

```
$bucketAuto: {
    groupBy: "$num_price",
    buckets: 10,
    output: {
      count: { $sum: 1 },
    },
    granularity: "1-2-5"
}
```

Comparing Granularities

Using **$bucketAuto** with 1, 2, 5 granularity

```
$bucketAuto: {
    groupBy: "$num_price",
    buckets: 10,
    output: {
      count: { $sum: 1 },
    },
    granularity: "1-2-5"
}
```

```
{ "_id" : { "min" : 0, "max" : 50 }, "count" : 5330 }
{ "_id" : { "min" : 50, "max" : 100 }, "count" : 17538 }
{ "_id" : { "min" : 100, "max" : 200 }, "count" : 17804 }
{ "_id" : { "min" : 200, "max" : 500 }, "count" : 8749 }
{ "_id" : { "min" : 500, "max" : 10000 }, "count" : 1375 }
```

Comparing Granularities

Using **$bucketAuto** with 1, 2, 5 granularity

```
$bucketAuto: {
    groupBy: "$num_price",
    buckets: 10,
    output: {
      count: { $sum: 1 },
    },
    granularity: "1-2-5"
}
```

```
{ "_id" : { "min" : 0, "max" : 50 }, "count" : 5330 }
{ "_id" : { "min" : 50, "max" : 100 }, "count" : 17538 }
{ "_id" : { "min" : 100, "max" : 200 }, "count" : 17804 }
{ "_id" : { "min" : 200, "max" : 500 }, "count" : 8749 }
{ "_id" : { "min" : 500, "max" : 10000 }, "count" : 1375 }
```

Comparing Granularities

Using **$bucketAuto** with POWERSOF2 granularity

```
$bucketAuto: {
    groupBy: "$num_price",
    buckets: 10,
    output: {
      count: { $sum: 1 },
    },
    granularity: "POWERSOF2"
}
```

Comparing Granularities

Using **$bucketAuto** with POWERSOF2 granularity

```
$bucketAuto: {
    groupBy: "$num_price",
    buckets: 10,
    output: {
      count: { $sum: 1 },
    },
    granularity: "POWERSOF2"
}
```

```
{ "_id" : { "min" : 0, "max" : 64 }, "count" : 11005 }
{ "_id" : { "min" : 64, "max" : 128 }, "count" : 18997 }
{ "_id" : { "min" : 128, "max" : 256 }, "count" : 15365 }
{ "_id" : { "min" : 256, "max" : 2048 }, "count" : 5275 }
{ "_id" : { "min" : 2048, "max" : 16384 }, "count" : 154 }
```


IT'S NOT A BUG
D
FEATURE
('The' + ' ' + 'Programmers') ...

If we prefer to manage our boundaries, use $bucket

If we let mongo manage arbitrary boundaries, use $bucketAuto

If we prefer a logarithmic scale, use $bucketAuto with 1-2-5 or POWERSOF2 granularity

If we prefer to manage our boundaries, use $bucket

If we let mongo manage arbitrary boundaries, use $bucketAuto

If we prefer a logarithmic scale, use $bucketAuto with 1-2-5 or POWERSOF2 granularity

For more resources on granularities, check here

Handling Multiple Streams: Faceting

An Art Collection to Sell...

{ "_id" : 1, "title" : "The Pillars of Society", "artist" : "Grosz", "year" : 1926, "price" : 199.99,
  "tags" : ["painting", "satire", "Expressionism", "caricature"] }
{ "_id" : 2, "title" : "Melancholy III", "artist" : "Munch", "year" : 1902, "price" : 280.00,
  "tags" : ["woodcut", "Expressionism"] }
{ "_id" : 3, "title" : "Dancer", "artist" : "Miro", "year" : 1925, "price" : 76.04,
  "tags" : ["oil", "Surrealism", "painting"] }
{ "_id" : 4, "title" : "The Great Wave off Kanagawa", "artist" : “Hokusai", "price" : 167.30,
  "tags" : ["woodblock", "ukiyo-e"] }
{ "_id" : 5, "title" : "The Persistence of Memory", "artist" : "Dali", "year" : 1931, "price" : 483.00,
  "tags" : ["Surrealism", "painting", "oil"] }
{ "_id" : 6, "title" : "Composition VII", "artist" : "Kandinsky", "year" : 1913, "price" : 385.00,
  "tags" : ["oil", "painting", "abstract"] }
{ "_id" : 7, "title" : "The Scream", "artist" : "Munch", "year" : 1893,
  "tags" : ["Expressionism", "painting", "oil"] }
{ "_id" : 8, "title" : "Blue Flower", "artist" : "O'Keefe", "year" : 1918, "price" : 118.42,
  "tags" : ["abstract", "painting"] }

Categorizing by Price

```
db.artwork.aggregate( [
 {
   $match: { price: { $exists: 1 } }
 },
 {
   $bucket: {
     groupBy: "$price",
     boundaries: [  0, 150, 200, 300, 400 ],
     default: "Other",
     output: {
       "count": { $sum: 1 },
       "titles": { $push: "$title" }
     }
   }
 }
])
```

Categorizing by Price

{ "_id" : 0, "count" : 2, "titles" : ["Dancer", "Blue Flower"] }
{ "_id" : 150, "count" : 2, "titles" : ["The Pillars of Society", "The Great Wave off Kanagawa"] }
{ "_id" : 200, "count" : 1, "titles" : ["Melancholy III"] }
{ "_id" : 300, "count" : 1, "titles" : ["Composition VII"] }
{ "_id" : "Other", "count" : 1, "titles" : ["The Persistence of Memory"] }

Categorizing by Tags

```
db.artwork.aggregate( [
 { $unwind: "$tags" },
 { $sortByCount: "$tags" }
])
```

```
{ "_id" : "painting", "count" : 6 }
{ "_id" : "oil", "count" : 4 }
{ "_id" : "Expressionism", "count" : 3 }
{ "_id" : "Surrealism", "count" : 2 }
{ "_id" : "abstract", "count" : 2 }
{ "_id" : "satire", "count" : 1 }
{ "_id" : "caricature", "count" : 1 }
{ "_id" : "woodcut", "count" : 1 }
{ "_id" : "woodblock", "count" : 1 }
{ "_id" : "ukiyo-e", "count" : 1 }
```

Doing Both Aggregations in Parallel

Aggregations

```
db.inventory.aggregate({$bucket
... })
db.inventory.aggregate({$unwind
... })
```

Insert

```
db.inventory.insertOne({tags:
['marble', 'renaissance'], name:
"David", "price": 900  })
```

Mongo

Doing Both Aggregations in Parallel

Aggregations

```
db.inventory.aggregate({$bucket
... })
db.inventory.aggregate({$unwind
... })
```

Insert

```
db.inventory.insertOne({tags:
['marble', 'renaissance'], name:
"David", "price": 900  })
```

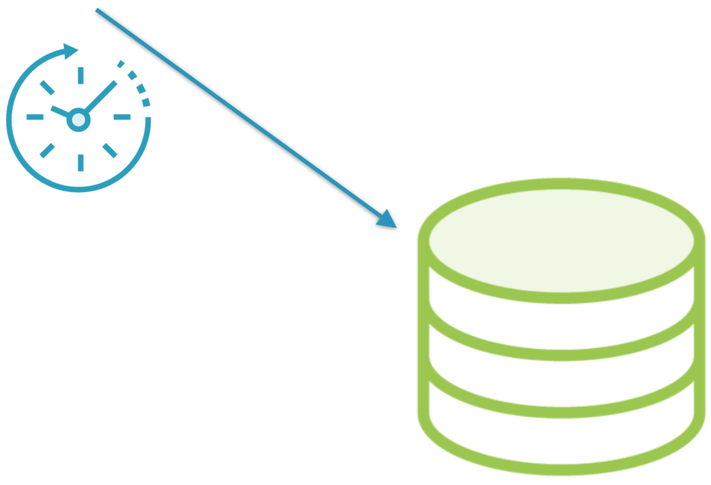
Doing Both Aggregations in Parallel

Aggregations

```
db.inventory.aggregate({$bucket
... })
db.inventory.aggregate({$unwind
... })
```

Insert

```
db.inventory.insertOne({tags:
['marble', 'renaissance'], name:
"David", "price": 900  })
```



Mongo

Doing Both Aggregations in Parallel

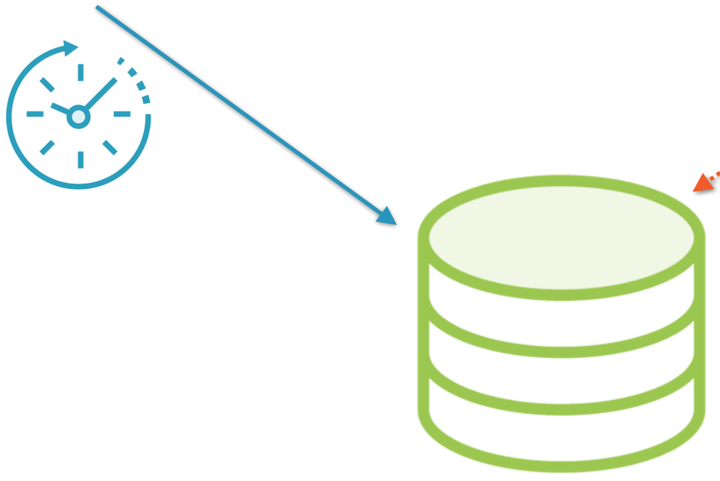
Aggregations

db.inventory.aggregate({$bucket ... })

db.inventory.aggregate({$unwind ... })

Insert

db.inventory.insertOne({tags: [‘marble’, ‘renaissance’], name: “David”, “price”: 900 })



Mongo

Doing Both Aggregations in Parallel

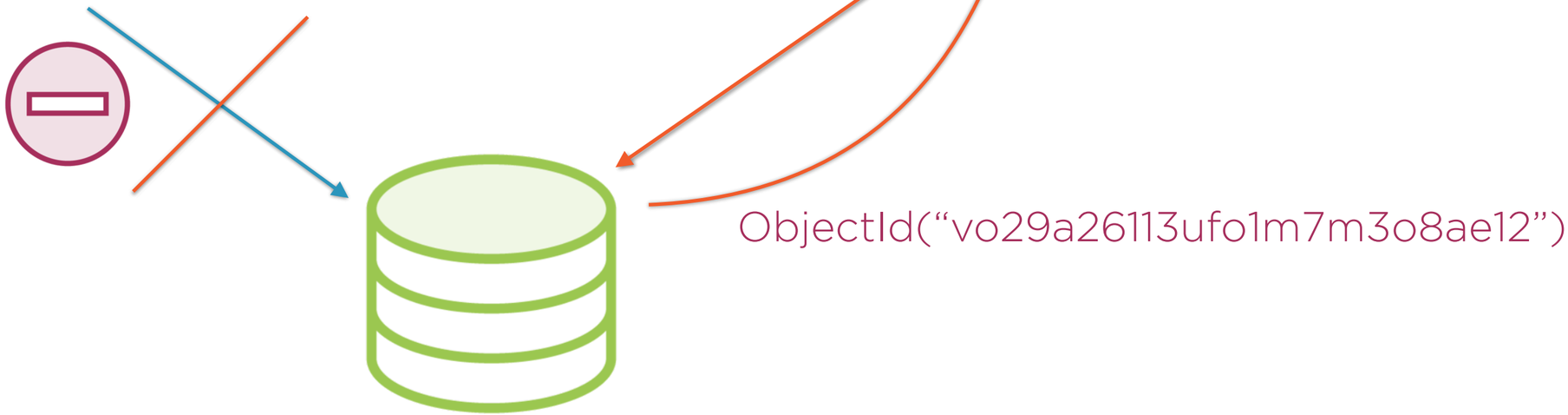
Aggregations

db.inventory.aggregate({$bucket ... })

db.inventory.aggregate({$unwind ... })

Insert

db.inventory.insertOne({tags: [‘marble’, ‘renaissance’], name: “David”, “price”: 900 })



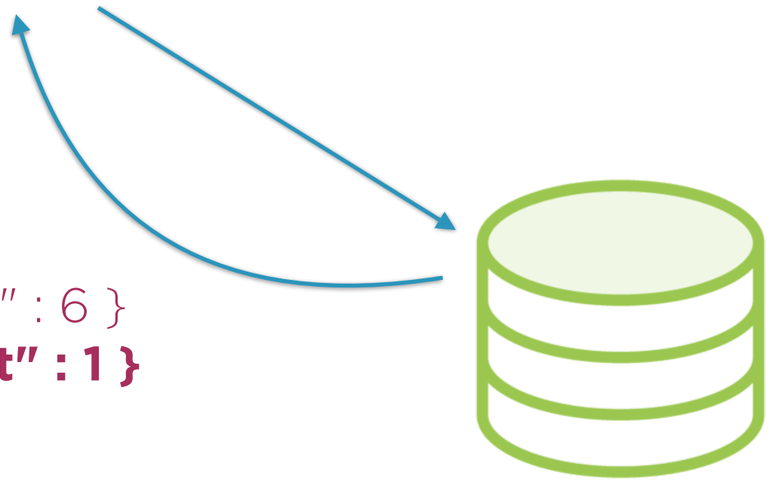
ObjectId(“vo29a26113ufo1m7m3o8ae12”)

Mongo

Doing Both Aggregations in Parallel

Aggregations

```
db.inventory.aggregate({$bucket
... })
db.inventory.aggregate({$unwind
... })
```



{ "_id" : "painting", "count" : 6 }
{ "_id" : "marble", "count" : 1 }

Mongo

Insert

```
db.inventory.insertOne({tags:
['marble', 'renaissance'], name:
"David", "price": 900  })
```

The Magic of $facet

{ "_id" : 1, "title" : "The Pillars of Society",
"artist" : "Grosz",
"year" : 1926, "price" : 199.99,
"tags" : ["painting",
"satire",
"Expressionism",
"caricature"] }

The Magic of $facet

```
{ "_id" : 1, "title" : "The
Pillars of Society",
"artist" : "Grosz",
"year" : 1926, "price" :
199.99,
  "tags" : [ "painting",
"satire",
"Expressionism",
"caricature" ] }
```

```
{$facet:
“categorizedByPrice”:
[...]
, “categorizedByTags”:
[...],
}
```

The Magic of $facet

```
{ "_id" : 1, "title" : "The
Pillars of Society",
"artist" : "Grosz",
"year" : 1926, "price" :
199.99,
  "tags" : [ "painting",
"satire",
"Expressionism",
"caricature" ] }
```

```
{$facet:
“categorizedByPrice”:
[...]
, “categorizedByTags”:
[...],
}
```

```
{ $unwind: "$tags" },
{ $sortByCount: "$tags" }
```

```
{ $match: {
    price: { $exists: 1 } } },
{ $bucket: {
    groupBy: “$price",
...
}
```

Same Query with $bucket

```
db.artwork.aggregate( [
 {$match: {year: { $lt: 1930 }}},
 {$facet: {
    "categorizedByTags": [
      { $unwind: "$tags" },
      { $sortByCount: "$tags" }
    ],
    "categorizedByPrice": [
      { $match: { price: { $exists: 1 } } },
      {
        $bucket: {
          groupBy: "$price",
          boundaries: [  0, 150, 200, 300, 400 ],
          default: "Other",
          output: {
            "count": { $sum: 1 },
            "titles": { $push: "$title" }
}}}])}])
```

◀ **# We can have any stages before**

◀ **# First pipeline: categorize by tags**

◀ **# Second pipeline: categorize by price**

Both will get the documents in parallel!
The results will be coherent!

Faceting Pipelines


```
{
"categorizedByTags" : [ { "_id" : "painting", "count" : 5 }, { "_id" : "Expressionism", "count" : 3 }, { "_id" : "oil", "count" : 3 }, { "_id" : "abstract", "count" : 2 }, { "_id" : "Surrealism", "count" : 1 }, { "_id" : "satire", "count" : 1 }, { "_id" : "caricature", "count" : 1 }, { "_id" : "woodcut", "count" : 1 } ],

"categorizedByPrice" : [ { "_id" : 0, "count" : 2, "titles" : [ "Dancer", "Blue Flower" ] }, { "_id" : 150, "count" : 1, "titles" : [ "The Pillars of Society" ] }, { "_id" : 200, "count" : 1, "titles" : [ "Melancholy III" ] }, { "_id" : 300, "count" : 1, "titles" : [ "Composition VII" ] } ]
}
```

When We Need to Save the Result: Write in a Pipeline

The $merge Stage

```
{ $merge: {
    into: <collection> -or- { db: <db>, coll: <collection> },
    on: <identifier field> -or- [ <identifier field1>, ...],  // Optional
    let: <variables>,                                         // Optional
    whenMatched: <replace|keepExisting|merge|fail|pipeline>,  // Optional
    whenNotMatched: <insert|discard|fail>                     // Optional
} }
```

A lot can happen, so for further reference check the docs here!!

Same Query with $bucket

```
{
  $merge: {
    into: {
      db: "pluralsight",
      coll: "report"
    }
  }
}
```

◀ **# If we add this final stage to write to the report collection**

```
> db.report.find({}, {_id:0, categorizedByTags:1,
categorizedByPrice: 0})
{"categorizedByTags" : [ { "_id" : "painting",
"count" : 5 }, { "_id" : "Expressionism", "count" :
3 }, { "_id" : "oil", "count" : 3 }, { "_id" :
"abstract", "count" : 2 }, { "_id" : "satire", "count"
: 1 }, { "_id" : "caricature", "count" : 1 }, { "_id" :
"woodcut", "count" : 1 }, { "_id" : "Surrealism",
"count" : 1 } ] }
```

◀ **# Success!**

A note is that $merge needs to be the final stage!

Demo

Demo

Demo

Write to a new collection the budgets for different fiscal years

Demo

Write to a new collection the budgets for different fiscal years

Update them with new headcount

Demo

Write to a new collection the budgets for different fiscal years

Update them with new headcount

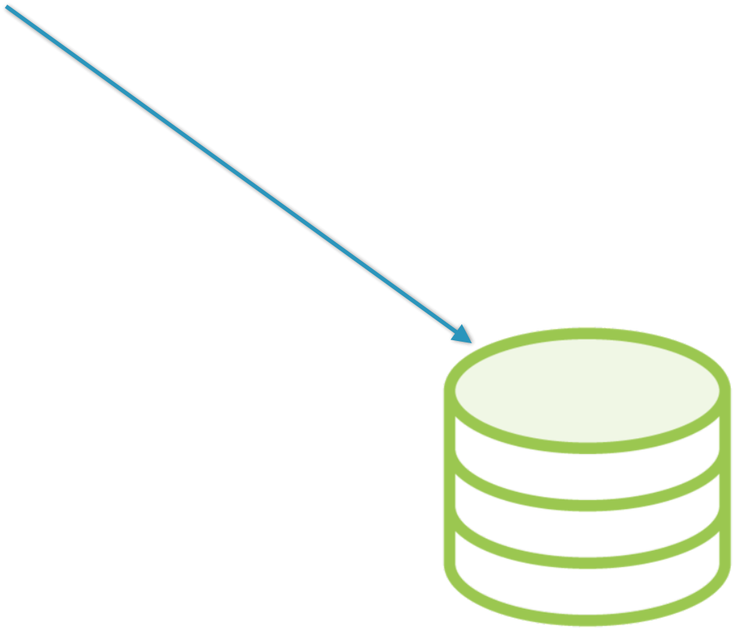
Merge on a different set of keys

When We Need to Aggregate over Databases

“Killing” the DB

“Killing” the DB

db.movies.createIndex({"$**": "text"})



“Killing” the DB

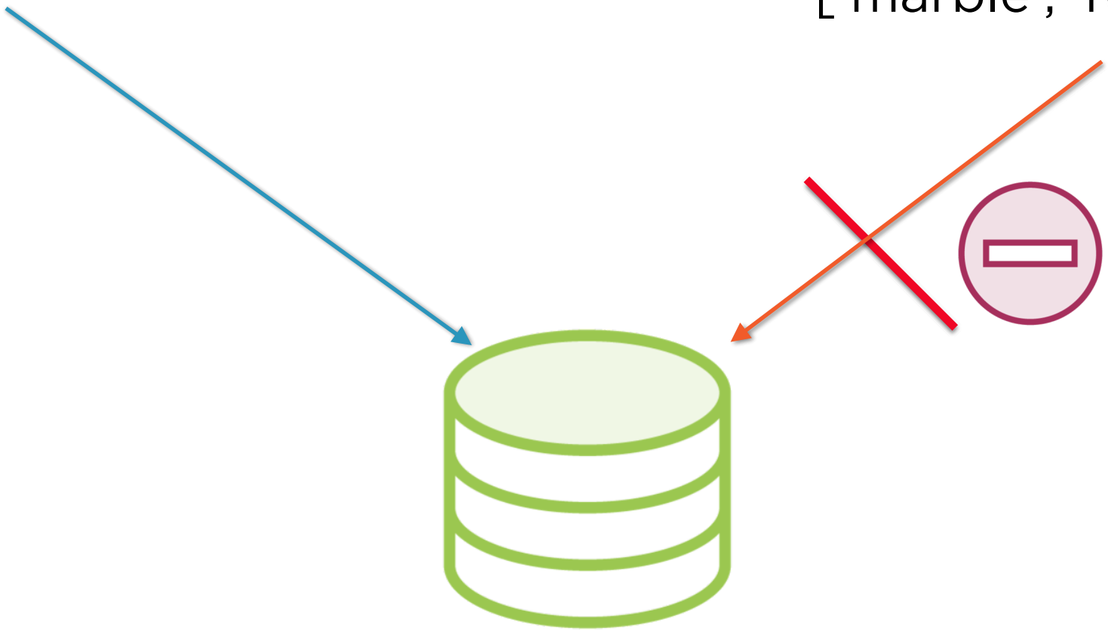
db.movies.createIndex({"$**": "text"})

db.inventory.insertOne({tags:
[‘marble’, ‘renaissance’], name:

“Killing” the DB

db.movies.createIndex({"$**": "text"})

db.inventory.insertOne({tags:
[‘marble’, ‘renaissance’], name:



“Killing” the DB

db.movies.createIndex({"$**": "text"})

db.inventory.insertOne({tags:
[‘marble’, ‘renaissance’], name:

For more information on why this can happen, check course **“Searching for Text in MongoDB”**

Sessions

For doing an operation, you need a driver

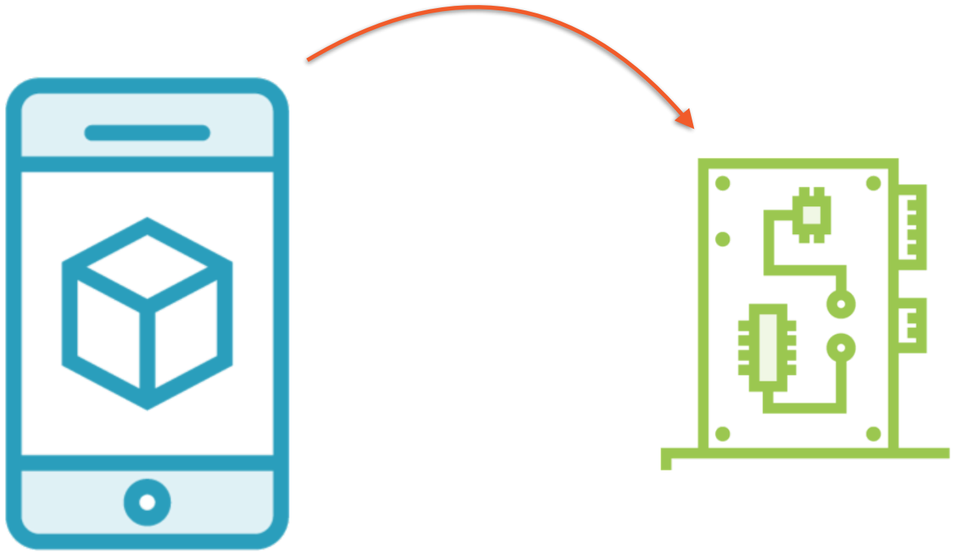
That session comes with a socket to send the commands

Starting in Mongo 3.6, all operations are bundled in a **session**

Instantiating a Session

APP

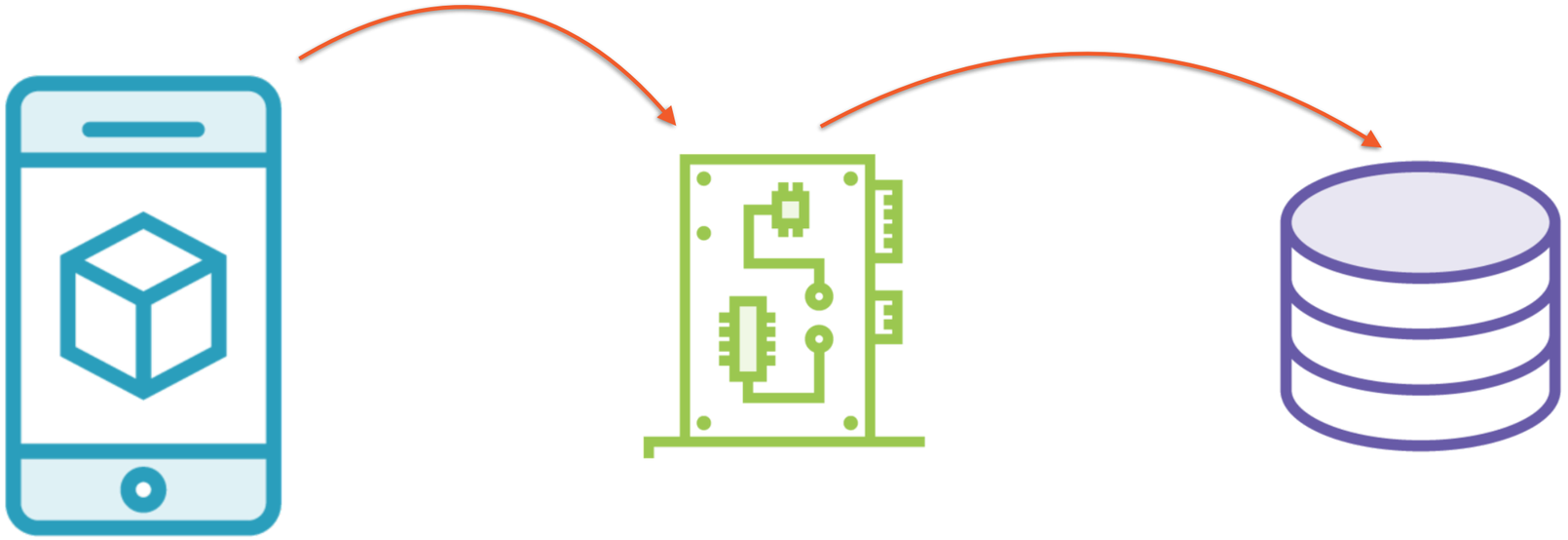
Instantiating a Session



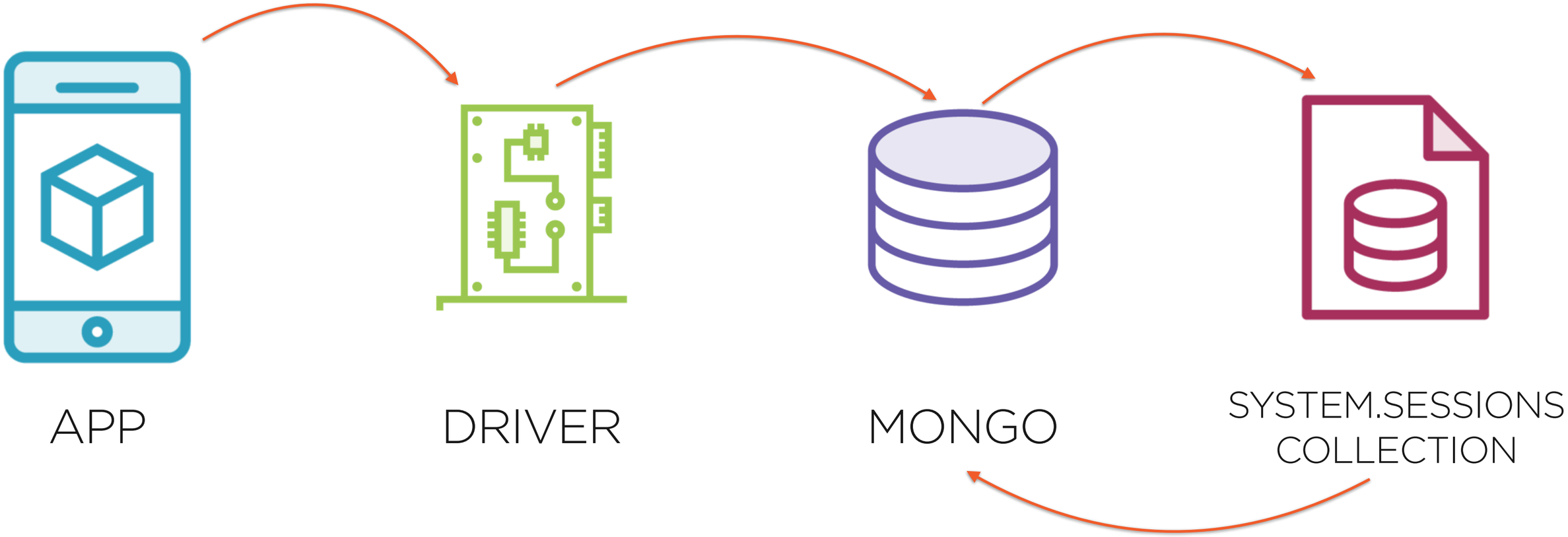
APP

DRIVER

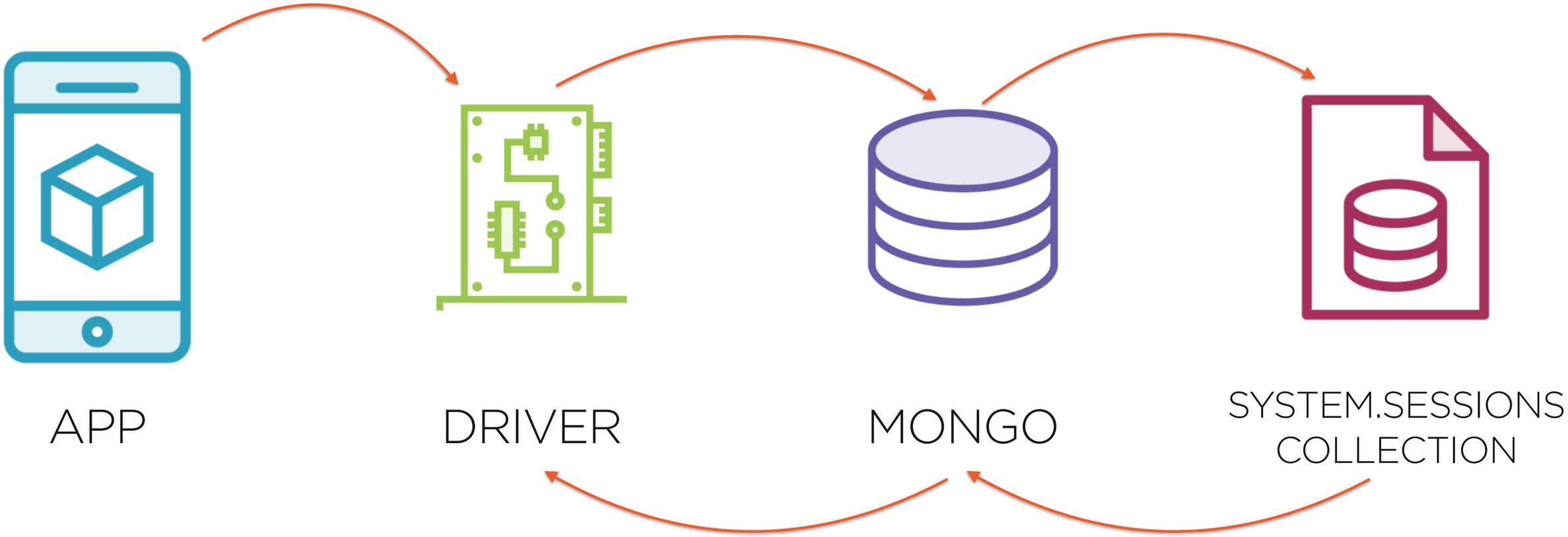
Instantiating a Session



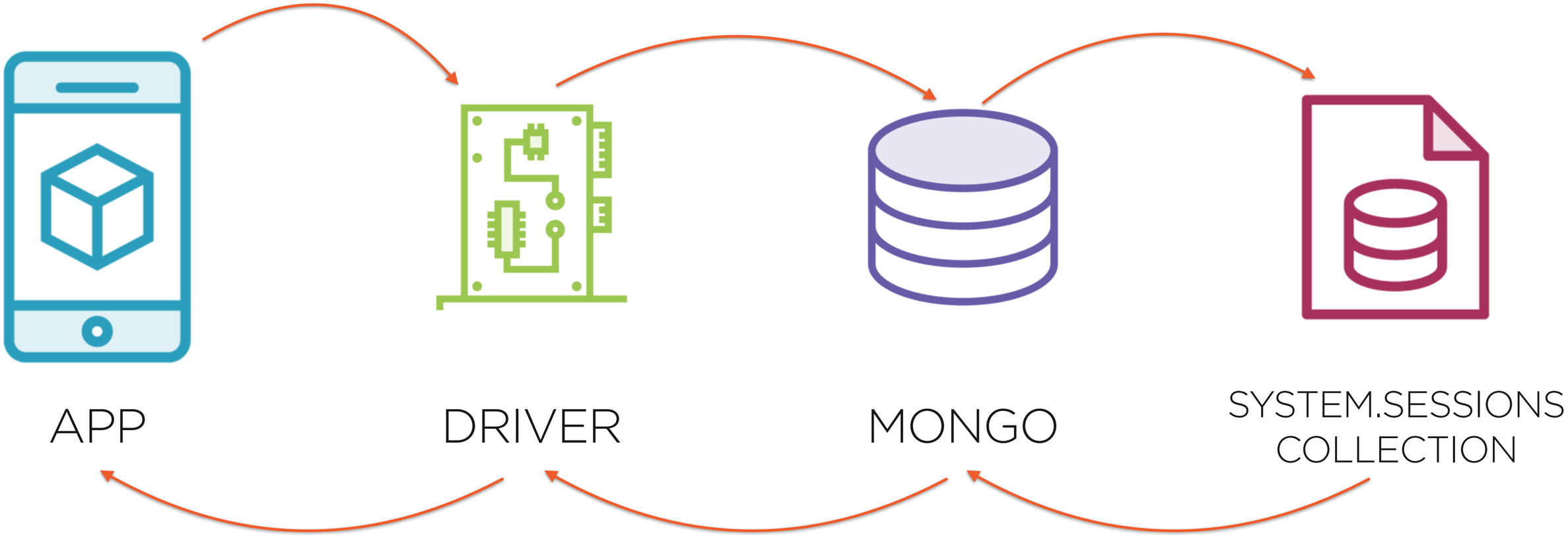
Instantiating a Session



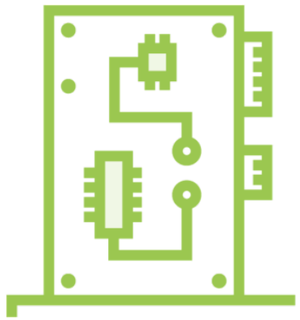
Instantiating a Session



Instantiating a Session



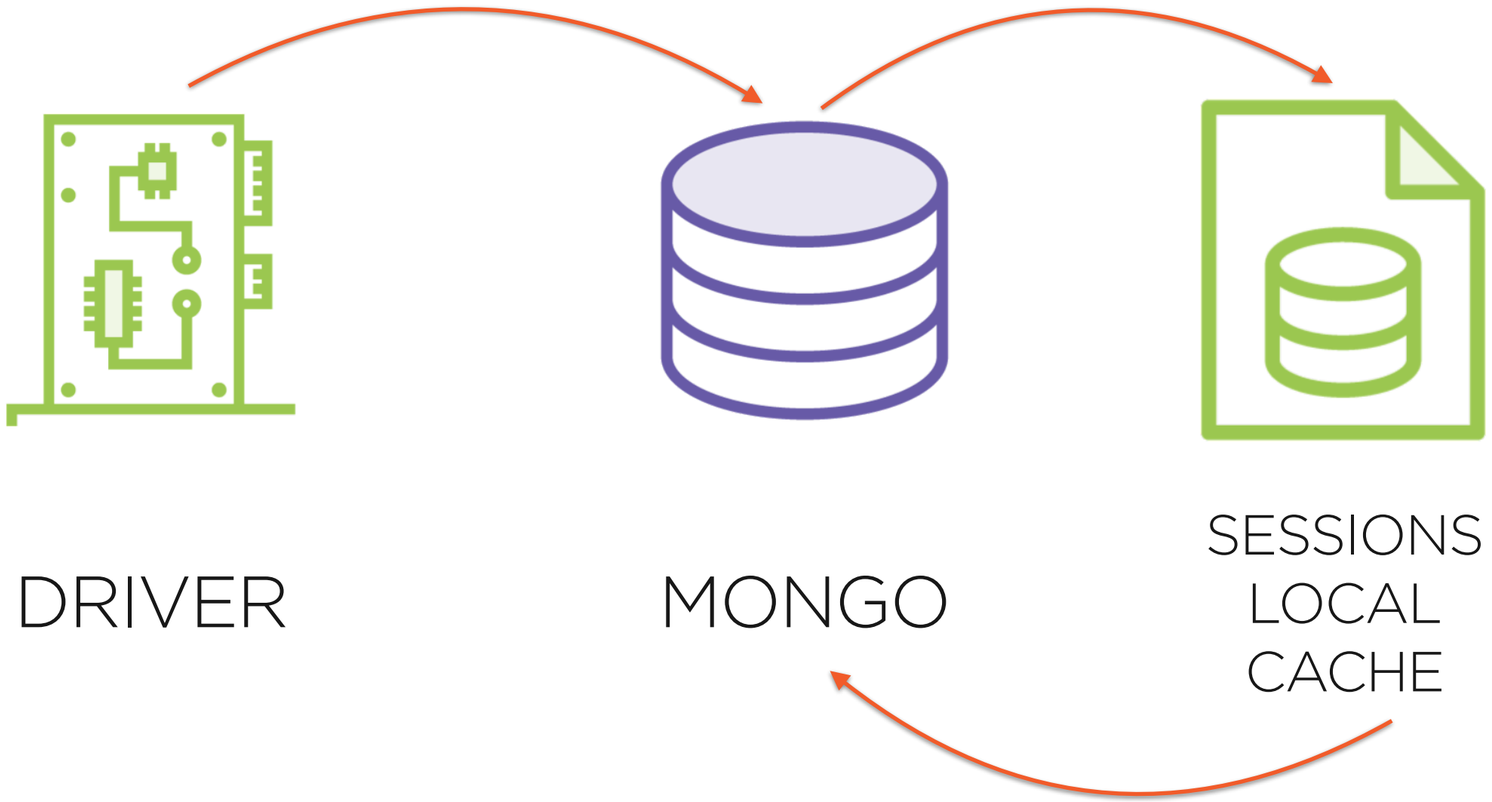
Local Sessions vs. System Sessions



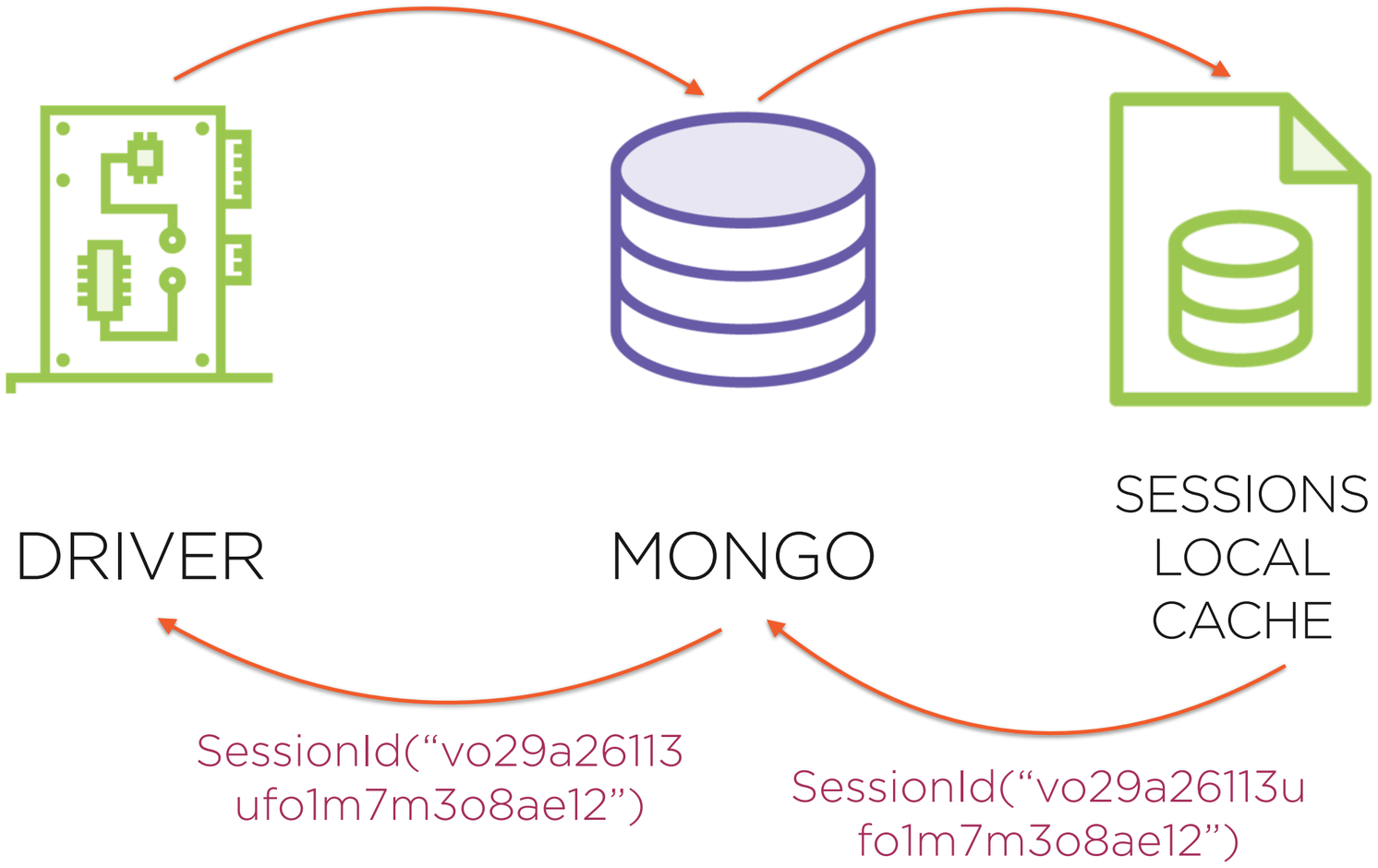
DRIVER

Local Sessions vs. System Sessions

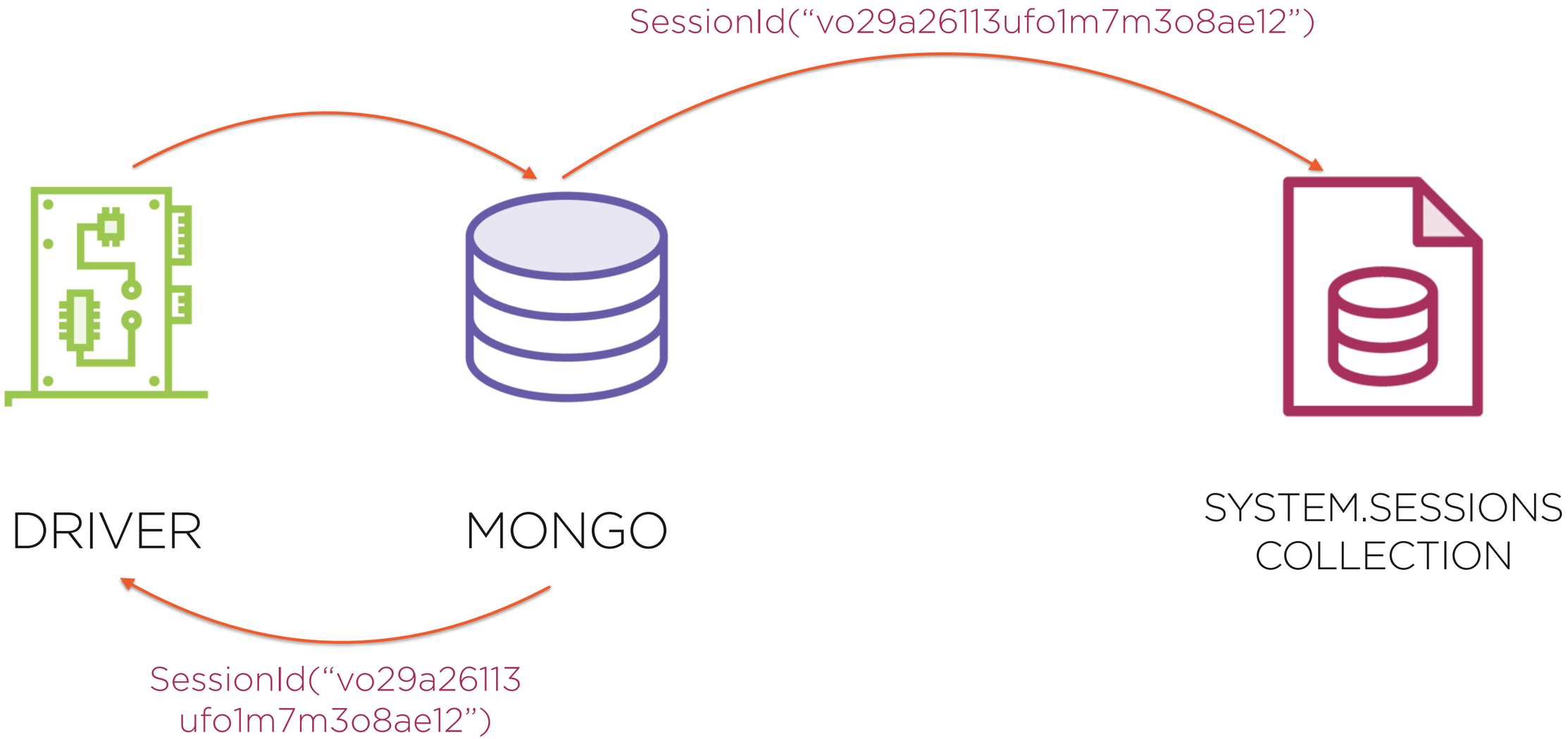
Local Sessions vs. System Sessions



Local Sessions vs. System Sessions



Local Sessions vs. System Sessions



$listSessions

```
use config

db.system.sessions.aggregate( [
    { $listSessions: { allUsers: true } }
])
```

```
db.system.sessions.aggregate( [
 { $listSessions: {
     users: [ {user: "myAppReader", db: "test" } ]
  }}
])
```

$listSessions

```
use config

db.system.sessions.aggregate( [
    { $listSessions: { allUsers: true } }
])
```

```
db.system.sessions.aggregate( [
 { $listSessions: {
     users: [ {user: "myAppReader", db: "test" } ]
  }}
])
```

$listLocalSessions for Local Sessions

LISTS LOCAL SESSIONS
ON A DB

OPERATES ON
DATABASES
AGGREGATIONS

NOT COLLECTIONS

Listing Local Sessions

```
> db.aggregate( [ { $listLocalSessions: { } } ] )

{ "_id" : { "id" : UUID("8ab43c9c-daed-4d76-8552-6e83e68f559f"), "uid" : BinData(0,"47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=") }, “lastUse" : ISODate("2020-04-18T18:40:15.154Z") }
{ "_id" : { "id" : UUID("f197c787-9013-4a8e-ae84-4231d17356b7"), "uid" : BinData(0,"47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=") }, "lastUse" : ISODate("2020-04-18T18:40:12.364Z") }
```

◀ **# Note that this is on the DB!**

◀ **# Information on the local session!**

A note is that $listLocalSessions needs to be the first stage!

$currentOp for All Operations

GETS **ALL** OPERATIONS
AND SESSIONS

OPERATES ON THE **ADMIN**
DATABASES
AGGREGATIONS

NOT COLLECTIONS

$currentOp Fields

```
{ $currentOp: {
    allUsers: <boolean>,
    idleConnections: <boolean>,
    idleCursors: <boolean>,
    idleSessions: <boolean>,
    localOps: <boolean>
} }
```

allUsers: To add all users or just us
idleConnections: Include iddle connections that may have a lock on a transaction
idleCursors: To add all cursors
idleSessions: To add also idle sessions **that persisted to the** system.sessions collection
localOps: Adding also local operations from Mongo itself

An Example with $currentOp

```
> db.getSiblingDB("admin").aggregate( [    { $currentOp : { allUsers: true } },    { $match :
{ locks: { $ne: {} } } } , {$project: {type: 1, connectionId: 1, command: 1, locks: 1,
microsecs_running:1}}] ).pretty()
{
    "type" : “op", “connectionId" : 26, “microsecs_running" : NumberLong(2981355),
    "command" : {
        "createIndexes" : "rent",
        "indexes" : [{“key" : {“$**" : “text”},”name" : “$**_text”}], "$db" : “pluralsight"
    }, ”locks" : {
        "ParallelBatchWriterMode" : "r",
        "ReplicationStateTransition" : "w",
        "Global" : "w",
        "Database" : "w",
        "Collection" : "r"}}
```

Summary

Summary

Summary

For categorizing documents use
$bucket and $bucketAuto

Summary

For categorizing documents use $bucket and $bucketAuto

For multiple aggregations in parallel use $facet

Summary

For categorizing documents use $bucket and $bucketAuto

For multiple aggregations in parallel use $facet

For writing to a new or existent collection use $merge

Summary

For categorizing documents use $bucket and $bucketAuto

For multiple aggregations in parallel use $facet

For writing to a new or existent collection use $merge

We can even have pipelines for admin tasks in Mongo!