# Performance Tuning and Optimization

6ème Octobre, 2015

Rubén Terceño
Solutions Architect

## Agenda

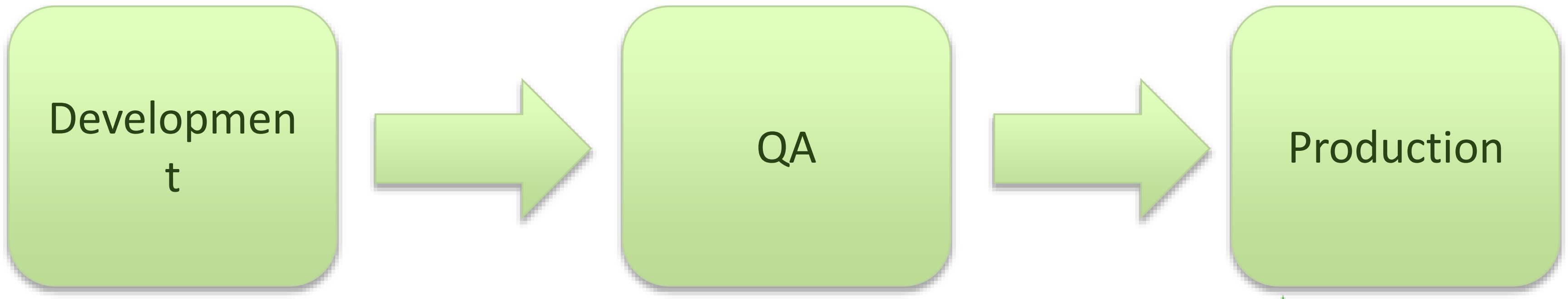
- Definition of terms
- When to do it
- Measurement tools
- Effecting Change
- Examples

## Performance Tuning vs Optimizing

- Optimizing – Modifying a system to work more efficiently or use fewer resources
- Performance Tuning – Modifying a system to handle increased load

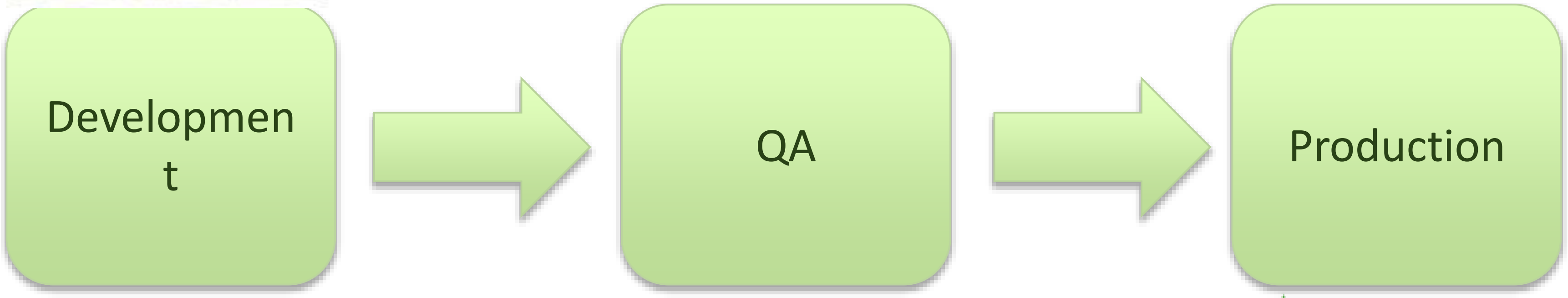
## Performance Tuning vs Optimizing

- Optimizing – Modifying a system to work more efficiently or use fewer resources
- Performance Tuning – Modifying a system to handle increased load



# Performance Tuning vs Optimizing

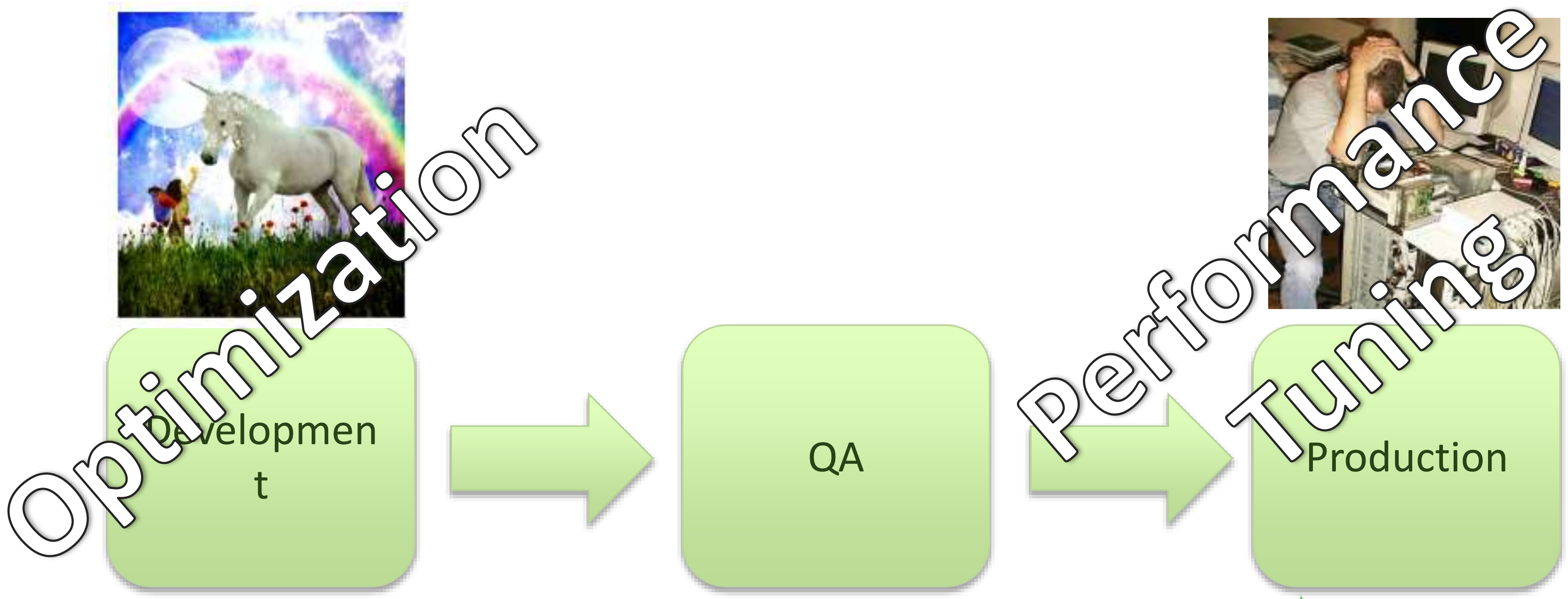
- Optimizing – Modifying a system to work more efficiently or use fewer resources
- Performance Tuning – Modifying a system to handle increased load



mongoDB
FOR GIANT IDEAS

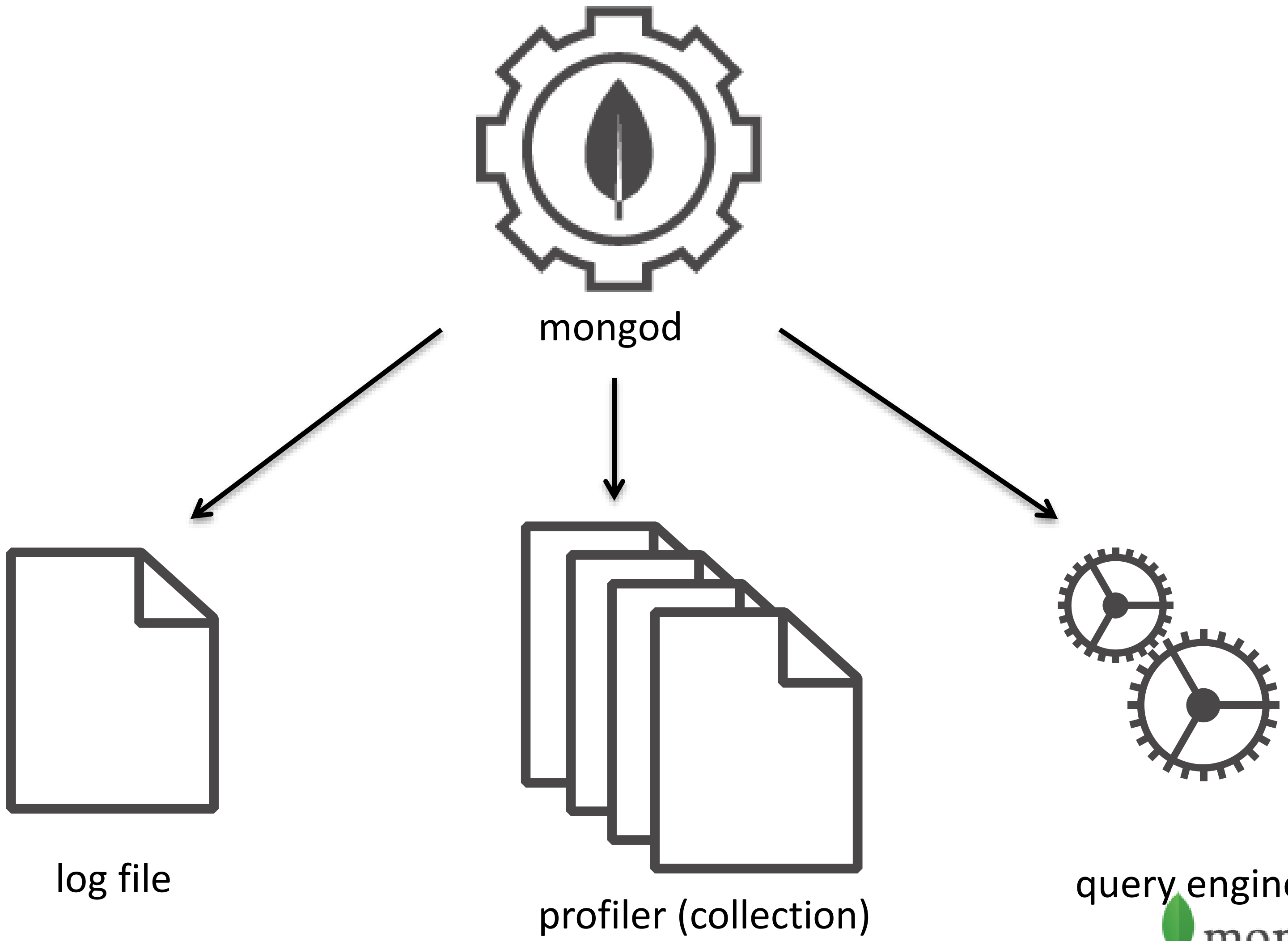
# Performance Tuning vs Optimizing

- Optimizing – Modifying a system to work more efficiently or use fewer resources
- Performance Tuning – Modifying a system to handle increased load



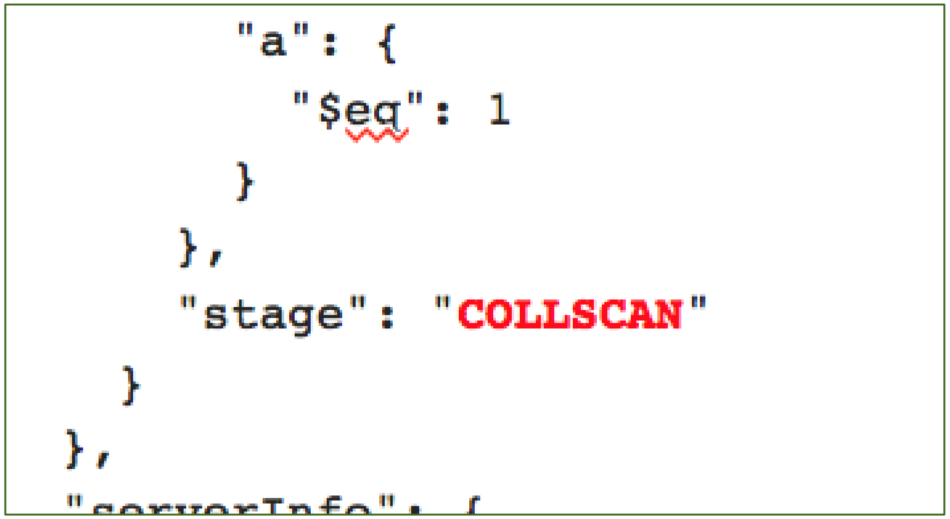
## Measurement Tools

# Log files, Profiler, Query Optimizer



# Explain plan – Query Planner

```
Jakes-MacBook-Pro(mongod-3.0.1)[PRIMARY] test> db.example.find({a:1}).explain()      // using the old <3.0 syntax
{
  "ok": 1,
  "queryPlanner": {
    "indexFilterSet": false,
    "namespace": "test.example",
    "parsedQuery": {
      "a": {
        "$eq": 1
      }
    },
    "plannerVersion": 1,
    "rejectedPlans": [ ],
    "winningPlan": {
      "direction": "forward",
      "filter": {
        "a": {
          "$eq": 1
        }
      },
      "stage": "COLLSCAN"
    }
  },
  "serverInfo": {
    "gitVersion": "534b5a3f9d10f00cd27737fbcd951032248b5952",
    "host": "Jakes-MacBook-Pro.local",
    "port": 27017,
    "version": "3.0.1"
  }
}
```

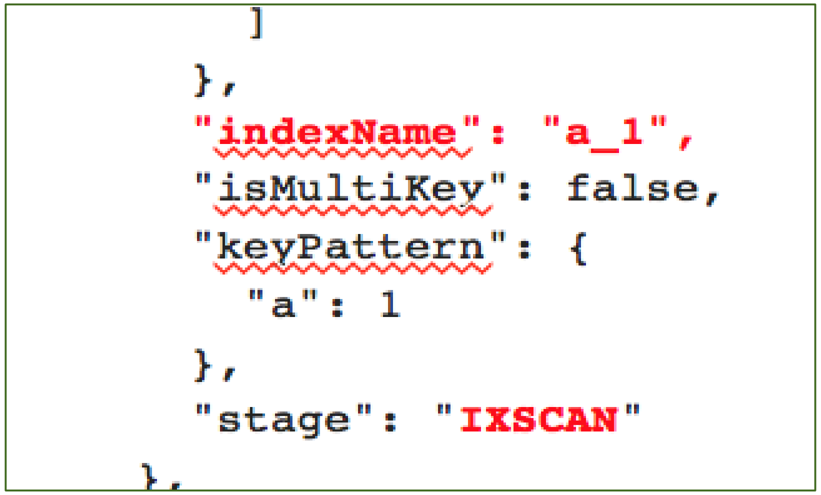


```
        "a": {
          "$eq": 1
        }
      },
      "stage": "COLLSCAN"
    }
  },
  "serverInfo": {
```

mongoDB. FOR GIANT IDEAS

# Explain plan – Adding an Index

```
Jakes-MacBook-Pro(mongod-3.0.1)[PRIMARY] test> db.example.ensureIndex({a:1})
Jakes-MacBook-Pro(mongod-3.0.1)[PRIMARY] test> db.example.find({a:1}).explain()     // using the old <3.0 syntax
{
  "ok": 1,
  "queryPlanner": {
    "indexFilterSet": false,
    "namespace": "test.example",
    "parsedQuery": {
      "a": {
        "$eq": 1
      }
    },
    "plannerVersion": 1,
    "rejectedPlans": [ ],
    "winningPlan": {
      "inputStage": {
        "direction": "forward",
        "indexBounds": {
          "a": [
            "[1.0, 1.0]"
          ]
        },
        "indexName": "a_1",
        "isMultiKey": false,
        "keyPattern": {
          "a": 1
        },
        "stage": "IXSCAN"
      },
      "stage": "FETCH"
    }
  }
[…]
```



```
        ]
      },
      "indexName": "a_1",
      "isMultiKey": false,
      "keyPattern": {
        "a": 1
      },
      "stage": "IXSCAN"
    },
```

# New Explain Syntax in MongoDB 3.0

- count, remove, aggregate, etc. now have an explain() method

```
> db.example.find({a:1}).count().explain() // <3.0
E QUERY TypeError: Object 3 has no method
'explain'
    at (shell):1:32

> db.example.explain().find({a:1}).count() // 3.0
```

- Explain a remove operation without actually removing anything

```
> db.example.explain().remove({a:1}) // doesn't
  remove anything
```

# Explain Levels in MongoDB 3.0

- **queryPlanner** (default level): runs the query planner and chooses the winning plan without actually executing the query
  - Use case: "Which plan will MongoDB choose to run my query?"
- **executionStats** – runs the query optimizer, then runs the winning plan to completion
  - Use case: "How is my query performing?"
- **allPlansExecution** – same as executionStats, but returns all the query plans, not just the winning plan.
  - Use case: "I want as much information as possible to diagnose a slow query."

mongoDB.
FOR GIANT IDEAS

# Explain plan – Query Planner

```
Jakes-MacBook-Pro(mongod-3.0.1)[PRIMARY] test> db.example.explain().find({a:1}) // new 3.0 syntax, default level
{
  "ok": 1,
  "queryPlanner": {
    "indexFilterSet": false,
    "namespace": "test.example",
    "parsedQuery": {
      "a": {
        "$eq": 1
      }
    },
    "plannerVersion": 1,
    "rejectedPlans": [ ],
    "winningPlan": {
      "inputStage": {
        "direction": "forward",
        "indexBounds": {
          "a": [
            "[1.0, 1.0]"
          ]
        },
        "indexName": "a_1",
        "isMultiKey": false,
        "keyPattern": {
          "a": 1
        },
        "stage": "IXSCAN"
      },
      "stage": "FETCH"
    }
  }
[…]
```

**queryPlanner** (default level): runs the query planner and chooses the winning plan without actually executing the query

# Explain plan – Query Optimizer

```
> db.example.explain("executionStats").find({a:1}) // new 3.0 syntax
{
  "executionStats": {
    "executionStages": {
      "advanced": 3,
      "alreadyHasObj": 0,
      "docsExamined": 3,
      "executionTimeMillisEstimate": 0,
      "inputStage": {
        "advanced": 3,
        "direction": "forward",
        "dupsDropped": 0,
        "dupsTested": 0,
        "executionTimeMillisEstimate": 0,
        "indexBounds": {
          "a": [
            "[1.0, 1.0]"
          ]
        },
        "indexName": "a_1",
        "invalidates": 0,
        "isEOF": 1,
        "isMultiKey": false,
        "keyPattern": {
          "a": 1
        },
        "keysExamined": 3,
        "matchTested": 0,
        "nReturned": 3,
        "needFetch": 0,
        "needTime": 0,
        "restoreState": 0,
        "saveState": 0,
        "seenInvalidated": 0,
        "stage": "IXSCAN",
        "works": 3
      },
      "invalidates": 0,
      "isEOF": 1,
      "nReturned": 3,
      "needFetch": 0,
      "needTime": 0,
      "restoreState": 0,
      "saveState": 0,
      "stage": "FETCH",
      "works": 4
    },
    "executionSuccess": true,
    "executionTimeMillis": 0,
    "nReturned": 3,
    "totalDocsExamined": 3,
    "totalKeysExamined": 3
  },
  "ok": 1,
  "queryPlanner": {
    […]
  }
}
```

**executionStats** – runs the query optimizer, then runs the winning plan to completion

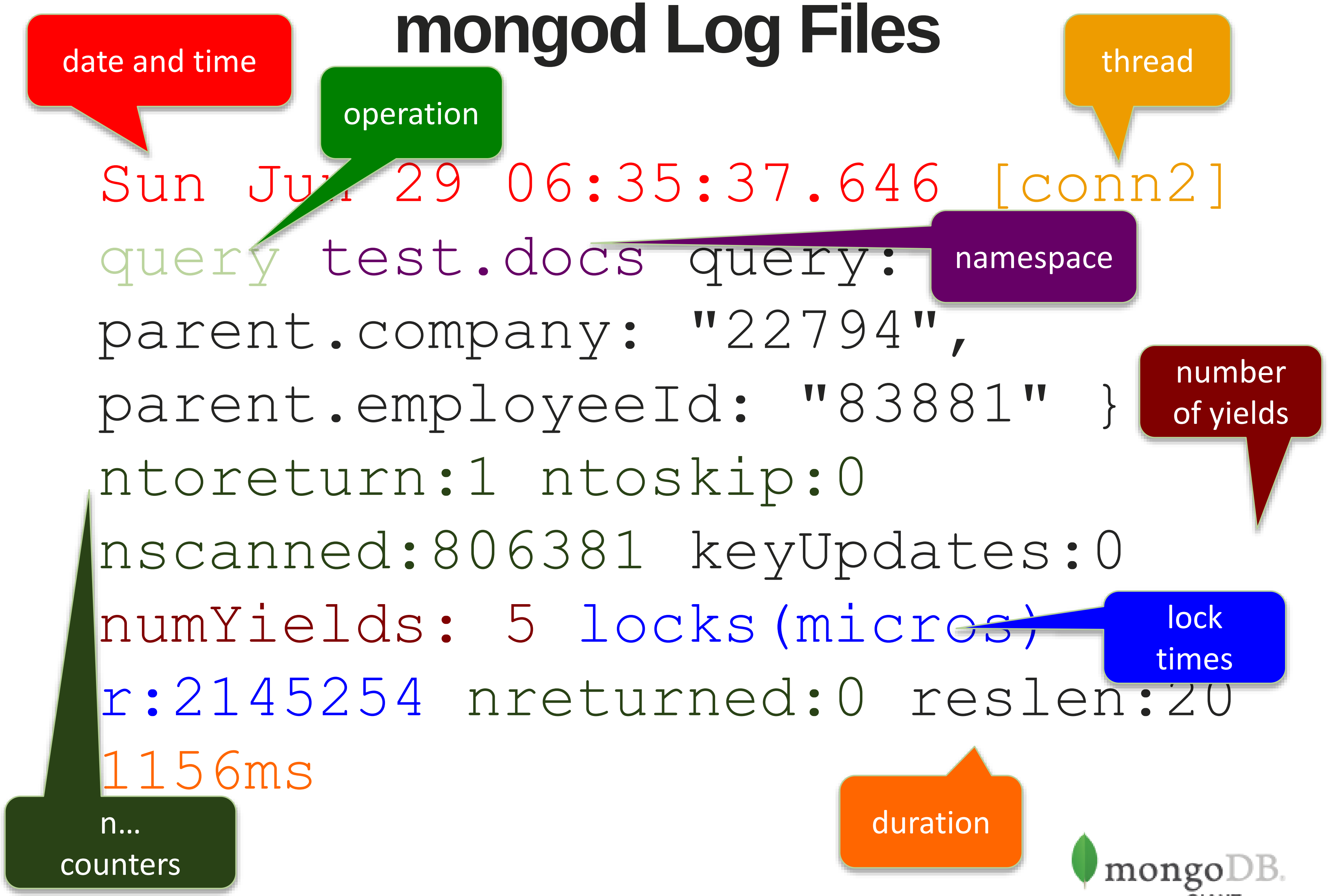
## Profiler

- 1MB capped collection named system.profile per database, per replica set
- One document per operation
- Examples:

```
  > db.setProfilingLevel(1)         // log all operations greater than 100ms
  > db.setProfilingLevel(1, 20)     // log all operations greater than 20ms
  > db.setProfilingLevel(2)         // log all operations regardless of duration
  > db.setProfilingLevel(0)         // turn off profiling
  > db.getProfilingStatus()         // display current profiling level
      {
        "slowms": 100,
        "was": 2
      }
```

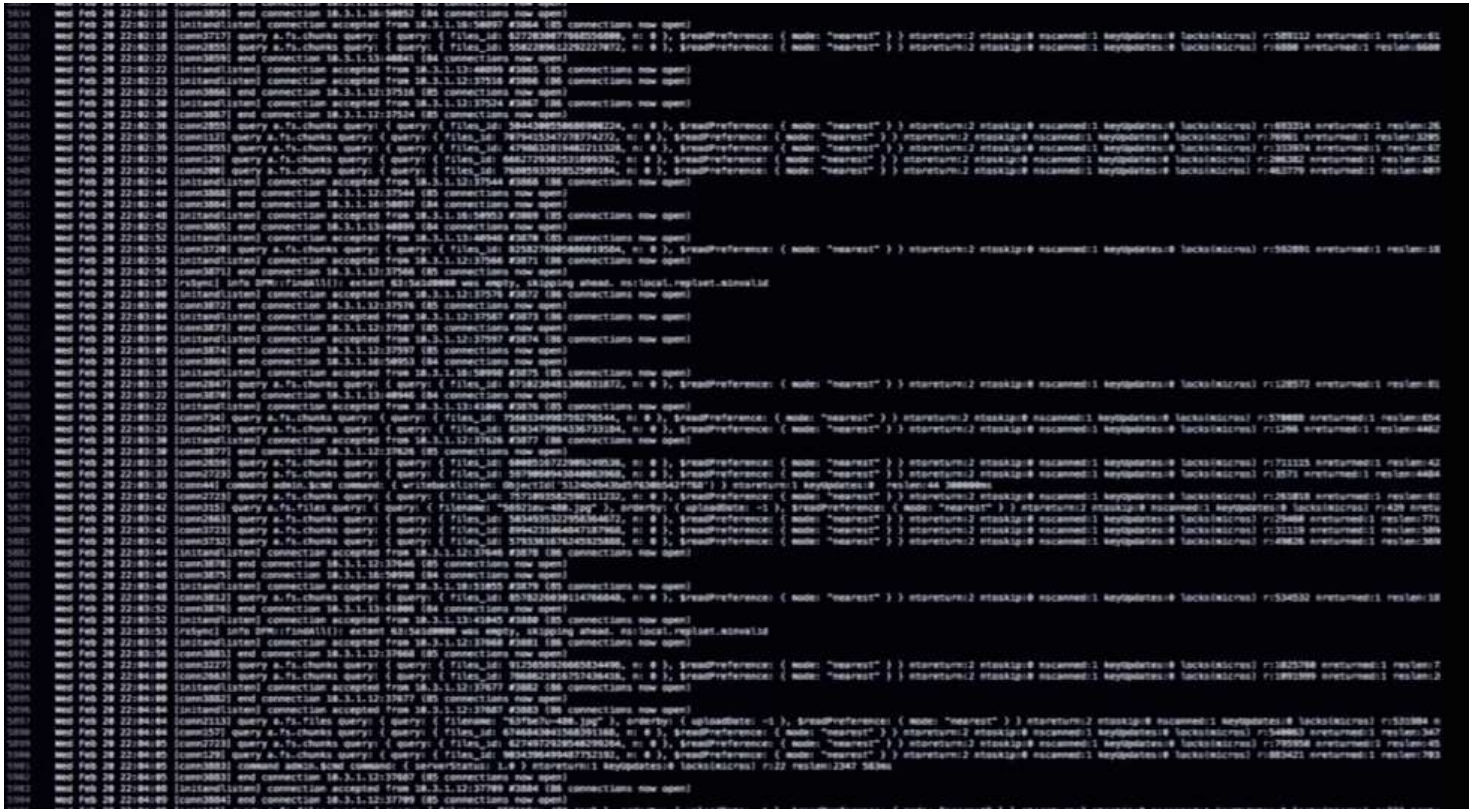
- In a sharded cluster, you will need to connect to each shard's primary mongod, not mongos

# mongod Log Files



```
Sun Jun 29 06:35:37.646 [conn2]
query test.docs query:
parent.company: "22794",
parent.employeeId: "83881" }
ntoreturn:1 ntoskip:0
nscanned:806381 keyUpdates:0
numYields: 5 locks(micros)
r:2145254 nreturned:0 reslen:20
1156ms
```

# Parsing Log Files



```
Wed Feb 20 22:02:18 [conn3858] end connection 10.3.1.16:50852 (84 connections now open)
Wed Feb 20 22:02:18 [initandlisten] connection accepted from 10.3.1.16:50897 #3864 (85 connections now open)
Wed Feb 20 22:02:18 [conn3717] query a.fs.chunks query: { query: { files_id: 6272830077068556000, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:509112 nreturned:1 reslen:61
Wed Feb 20 22:02:18 [conn2855] query a.fs.chunks query: { query: { files_id: 5582289612292227872, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:6880 nreturned:1 reslen:6680
Wed Feb 20 22:02:22 [conn3859] end connection 10.3.1.13:40841 (84 connections now open)
Wed Feb 20 22:02:22 [initandlisten] connection accepted from 10.3.1.13:40899 #3865 (85 connections now open)
Wed Feb 20 22:02:23 [initandlisten] connection accepted from 10.3.1.12:37516 #3866 (86 connections now open)
Wed Feb 20 22:02:23 [conn3866] end connection 10.3.1.12:37516 (85 connections now open)
Wed Feb 20 22:02:30 [initandlisten] connection accepted from 10.3.1.12:37524 #3867 (86 connections now open)
Wed Feb 20 22:02:30 [conn3867] end connection 10.3.1.12:37524 (85 connections now open)
Wed Feb 20 22:02:36 [conn2855] query a.fs.chunks query: { query: { files_id: 5044200550600000224, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:683314 nreturned:1 reslen:26
Wed Feb 20 22:02:36 [conn112] query a.fs.chunks query: { query: { files_id: 7079415347278774272, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:76961 nreturned:1 reslen:3205
Wed Feb 20 22:02:39 [conn2855] query a.fs.chunks query: { query: { files_id: 4790032819482211328, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:333974 nreturned:1 reslen:67
Wed Feb 20 22:02:39 [conn129] query a.fs.chunks query: { query: { files_id: 6662729382531899392, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:206382 nreturned:1 reslen:262
Wed Feb 20 22:02:42 [conn200] query a.fs.chunks query: { query: { files_id: 7608593395852509184, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:463779 nreturned:1 reslen:487
Wed Feb 20 22:02:44 [initandlisten] connection accepted from 10.3.1.12:37544 #3868 (86 connections now open)
Wed Feb 20 22:02:44 [conn3868] end connection 10.3.1.12:37544 (85 connections now open)
Wed Feb 20 22:02:48 [conn3864] end connection 10.3.1.16:50897 (84 connections now open)
Wed Feb 20 22:02:48 [initandlisten] connection accepted from 10.3.1.16:50953 #3869 (85 connections now open)
Wed Feb 20 22:02:52 [conn3865] end connection 10.3.1.13:40899 (84 connections now open)
Wed Feb 20 22:02:52 [initandlisten] connection accepted from 10.3.1.13:40946 #3870 (85 connections now open)
Wed Feb 20 22:02:52 [conn3720] query a.fs.chunks query: { query: { files_id: 8258276005000019504, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:562891 nreturned:1 reslen:18
Wed Feb 20 22:02:56 [initandlisten] connection accepted from 10.3.1.12:37566 #3871 (86 connections now open)
Wed Feb 20 22:02:56 [conn3871] end connection 10.3.1.12:37566 (85 connections now open)
Wed Feb 20 22:02:57 [rsSync] info DFM::findAll(): extent 63:5a1d0000 was empty, skipping ahead. ns:local.replset.minvalid
Wed Feb 20 22:03:00 [initandlisten] connection accepted from 10.3.1.12:37576 #3872 (86 connections now open)
Wed Feb 20 22:03:00 [conn3872] end connection 10.3.1.12:37576 (85 connections now open)
Wed Feb 20 22:03:04 [initandlisten] connection accepted from 10.3.1.12:37587 #3873 (86 connections now open)
Wed Feb 20 22:03:04 [conn3873] end connection 10.3.1.12:37587 (85 connections now open)
Wed Feb 20 22:03:09 [initandlisten] connection accepted from 10.3.1.12:37597 #3874 (86 connections now open)
Wed Feb 20 22:03:09 [conn3874] end connection 10.3.1.12:37597 (85 connections now open)
Wed Feb 20 22:03:18 [conn3869] end connection 10.3.1.16:50953 (84 connections now open)
Wed Feb 20 22:03:18 [initandlisten] connection accepted from 10.3.1.16:50998 #3875 (85 connections now open)
Wed Feb 20 22:03:19 [conn2847] query a.fs.chunks query: { query: { files_id: 8718230481300031872, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:128572 nreturned:1 reslen:81
Wed Feb 20 22:03:22 [conn3870] end connection 10.3.1.13:40946 (84 connections now open)
Wed Feb 20 22:03:22 [initandlisten] connection accepted from 10.3.1.13:41006 #3876 (85 connections now open)
Wed Feb 20 22:03:22 [conn734] query a.fs.chunks query: { query: { files_id: 7568334990759276544, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:370088 nreturned:1 reslen:654
Wed Feb 20 22:03:23 [conn2847] query a.fs.chunks query: { query: { files_id: 8283479894336733184, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:1296 nreturned:1 reslen:4462
Wed Feb 20 22:03:30 [initandlisten] connection accepted from 10.3.1.12:37626 #3877 (86 connections now open)
Wed Feb 20 22:03:30 [conn3877] end connection 10.3.1.12:37626 (85 connections now open)
Wed Feb 20 22:03:33 [conn2659] query a.fs.chunks query: { query: { files_id: 6000516722909249536, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:711125 nreturned:1 reslen:42
Wed Feb 20 22:03:33 [conn2723] query a.fs.chunks query: { query: { files_id: 5979050943040083968, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:3571 nreturned:1 reslen:4484
Wed Feb 20 22:03:38 [conn44] command admin.$cmd command: { writebacklisten: ObjectId('51240d0430d5f63005427f50') } ntoreturn:1 keyUpdates:0  reslen:44 300000ms
Wed Feb 20 22:03:42 [conn2723] query a.fs.chunks query: { query: { files_id: 7571093582590111232, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:263818 nreturned:1 reslen:82
Wed Feb 20 22:03:42 [conn315] query a.fs.files query: { query: { filename: "50921au-480.jpg" }, orderby: { uploadDate: -1 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:429 nretu
Wed Feb 20 22:03:42 [conn2863] query a.fs.chunks query: { query: { files_id: 5034935322958364672, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:29480 nreturned:1 reslen:771
Wed Feb 20 22:03:42 [conn3723] query a.fs.chunks query: { query: { files_id: 8994231064047187968, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:31119 nreturned:1 reslen:509
Wed Feb 20 22:03:42 [conn3732] query a.fs.chunks query: { query: { files_id: 5793381676245925888, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:49826 nreturned:1 reslen:309
Wed Feb 20 22:03:44 [initandlisten] connection accepted from 10.3.1.12:37646 #3878 (86 connections now open)
Wed Feb 20 22:03:44 [conn3878] end connection 10.3.1.12:37646 (85 connections now open)
Wed Feb 20 22:03:48 [conn3875] end connection 10.3.1.16:50998 (84 connections now open)
Wed Feb 20 22:03:48 [initandlisten] connection accepted from 10.3.1.16:51055 #3879 (85 connections now open)
Wed Feb 20 22:03:48 [conn3812] query a.fs.chunks query: { query: { files_id: 8578226030114766848, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:534532 nreturned:1 reslen:18
Wed Feb 20 22:03:52 [conn3876] end connection 10.3.1.13:41006 (84 connections now open)
Wed Feb 20 22:03:52 [initandlisten] connection accepted from 10.3.1.13:41045 #3880 (85 connections now open)
Wed Feb 20 22:03:53 [rsSync] info DFM::findAll(): extent 63:5a1d0000 was empty, skipping ahead. ns:local.replset.minvalid
Wed Feb 20 22:03:56 [initandlisten] connection accepted from 10.3.1.12:37668 #3881 (86 connections now open)
Wed Feb 20 22:03:56 [conn3881] end connection 10.3.1.12:37668 (85 connections now open)
Wed Feb 20 22:04:00 [conn3227] query a.fs.chunks query: { query: { files_id: 9125850920665834496, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:1825780 nreturned:1 reslen:7
Wed Feb 20 22:04:00 [conn2083] query a.fs.chunks query: { query: { files_id: 7068821816757430438, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:1093999 nreturned:1 reslen:2
Wed Feb 20 22:04:00 [initandlisten] connection accepted from 10.3.1.12:37677 #3882 (86 connections now open)
Wed Feb 20 22:04:00 [conn3882] end connection 10.3.1.12:37677 (85 connections now open)
Wed Feb 20 22:04:04 [initandlisten] connection accepted from 10.3.1.12:37687 #3883 (86 connections now open)
Wed Feb 20 22:04:04 [conn2113] query a.fs.files query: { query: { filename: "63fbe7u-480.jpg" }, orderby: { uploadDate: -1 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:531984 n
Wed Feb 20 22:04:04 [conn157] query a.fs.chunks query: { query: { files_id: 6746843041568391168, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:540063 nreturned:1 reslen:347
Wed Feb 20 22:04:05 [conn2723] query a.fs.chunks query: { query: { files_id: 6274972920548290264, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:795950 nreturned:1 reslen:45
Wed Feb 20 22:04:05 [conn129] query a.fs.chunks query: { query: { files_id: 9034390499487752192, n: 0 }, $readPreference: { mode: "nearest" } } ntoreturn:2 ntoskip:0 nscanned:1 keyUpdates:0 locks(micros) r:883421 nreturned:1 reslen:703
Wed Feb 20 22:04:05 [conn3883] command admin.$cmd command: { serverStatus: 1.0 } ntoreturn:1 keyUpdates:0 locks(micros) r:22 reslen:2347 583ms
Wed Feb 20 22:04:05 [conn3883] end connection 10.3.1.12:37687 (85 connections now open)
Wed Feb 20 22:04:08 [initandlisten] connection accepted from 10.3.1.12:37709 #3884 (86 connections now open)
Wed Feb 20 22:04:09 [conn3884] end connection 10.3.1.12:37709 (85 connections now open)
```

mongoDB
FOR GIANT IDEAS

# mtools

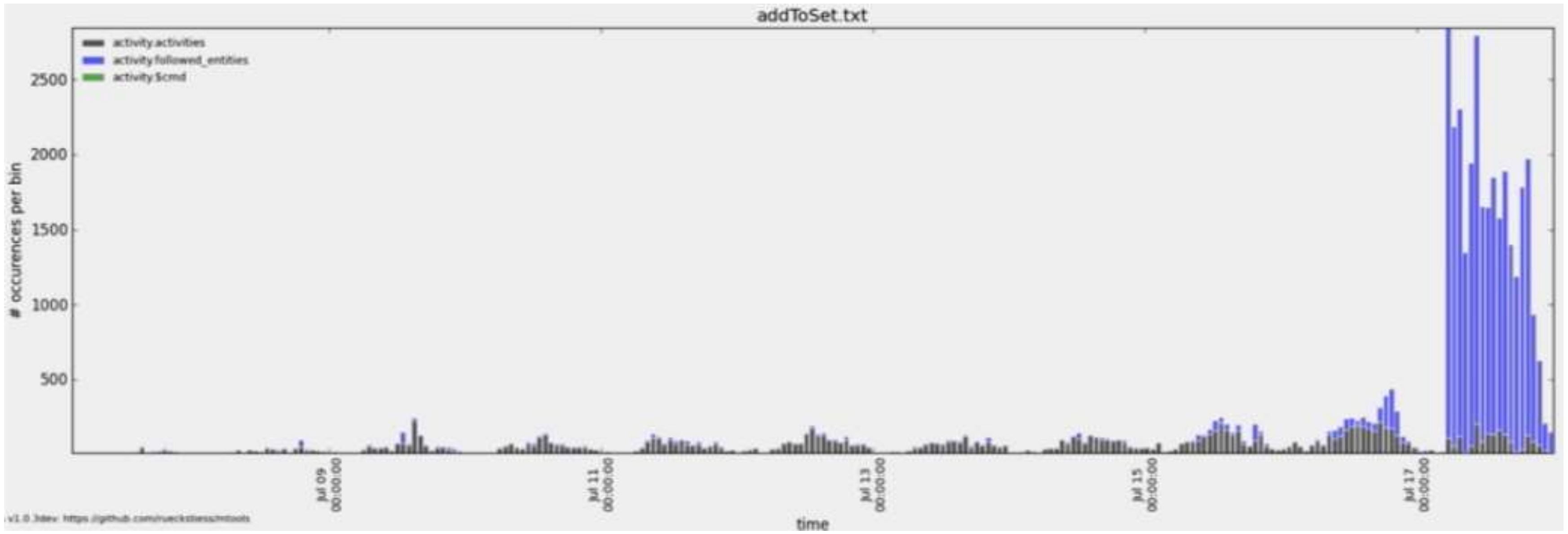
- http://github.com/rueckstiess/mtools
- log file analysis for poorly performing queries
  - Show me queries that took more than 1000 ms from 6 am to 6 pm:

```
$ mlogfilter mongodb.log --from 06:00 --to
  18:00 --slow 1000 > mongodb-filtered.log
```

mongoDB. FOR GIANT IDEAS

## mtools graphs

```
% mplotqueries --type histogram --group namespace --bucketSize 3600
```



## Command Line tools

- iostat
- dstat
- mongostat
- mongotop
- mongoperf

## Ops Manager / Cloud Manager

- Memory usage
- Opcounters
- Lock percentage
- Queues
- Background flush average
- Replication oplog window and lag

## Effecting Change

## Process

1. Measure current performance
2. Find the bottleneck (the hard part)
3. Remove the bottleneck
4. Measure again
5. Repeat as needed

## What can you change?

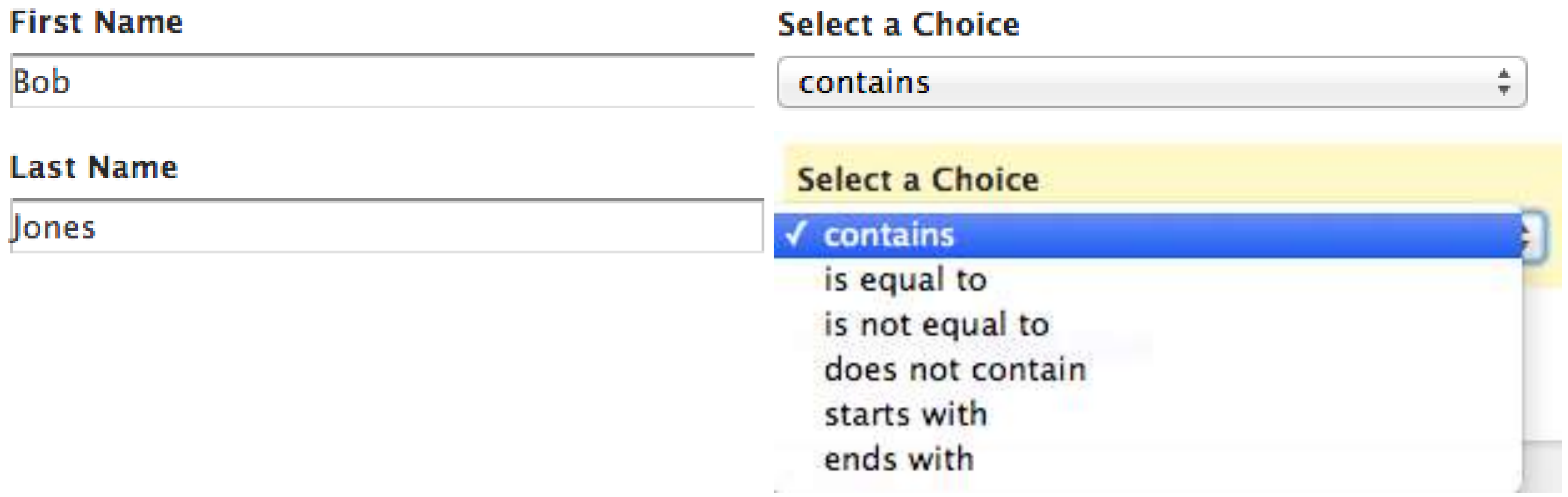
- Schema design
- Access patterns
- Indexes
- Instance
- Hardware

## Schema Design

- Replay Norberto’s conference in your mind.
- Now we’re done
- (La cuillère n’existe pas)

# Example: Access Patterns

- Application allowed searches for users by first and/or last name



| | |
|---|---|
| First Name | Select a Choice |
| Bob | contains |
| Last Name | Select a Choice |
| Jones | ✓ contains |

```
Tue Jul 1 13:08:29.858 [conn581923] query db.users query: {
$query: {$and: [ { $and: [ { firstName: /((?i)\Qbob\E)/ }, {
lastName: /((?i)\Qjones\E)/ } ] } ] }, $orderby: { lastName:
1 } } ntoreturn:25 ntoskip:0 nscanned:2626282 scanAndOrder:1
keyUpdates:0 numYields: 299 locks(micros) r:30536738
nreturned:14 reslen:8646 15504ms
```

## Example: Access Patterns

- Application was searching for unindexed, case-insensitive, unanchored regular expressions

```
{
    _id: 1,
    firstName: "Bob",
    lastName: "Jones"
}
```

- MongoDB is better at indexed, case-sensitive, left-anchored regular expressions

```
{
    _id: 1,
    firstName: "Bob",
    lastName: "Jones",
    fn: "bob",
    ln: "jones"
}
```

```
> db.users.ensureIndex({ln:1, fn:1})
> db.users.ensureIndex({fn:1, ln:1})
> db.users.find({fn:/^bob/}).sort
                              ({ln:1})
```

## Indexing Suggestions

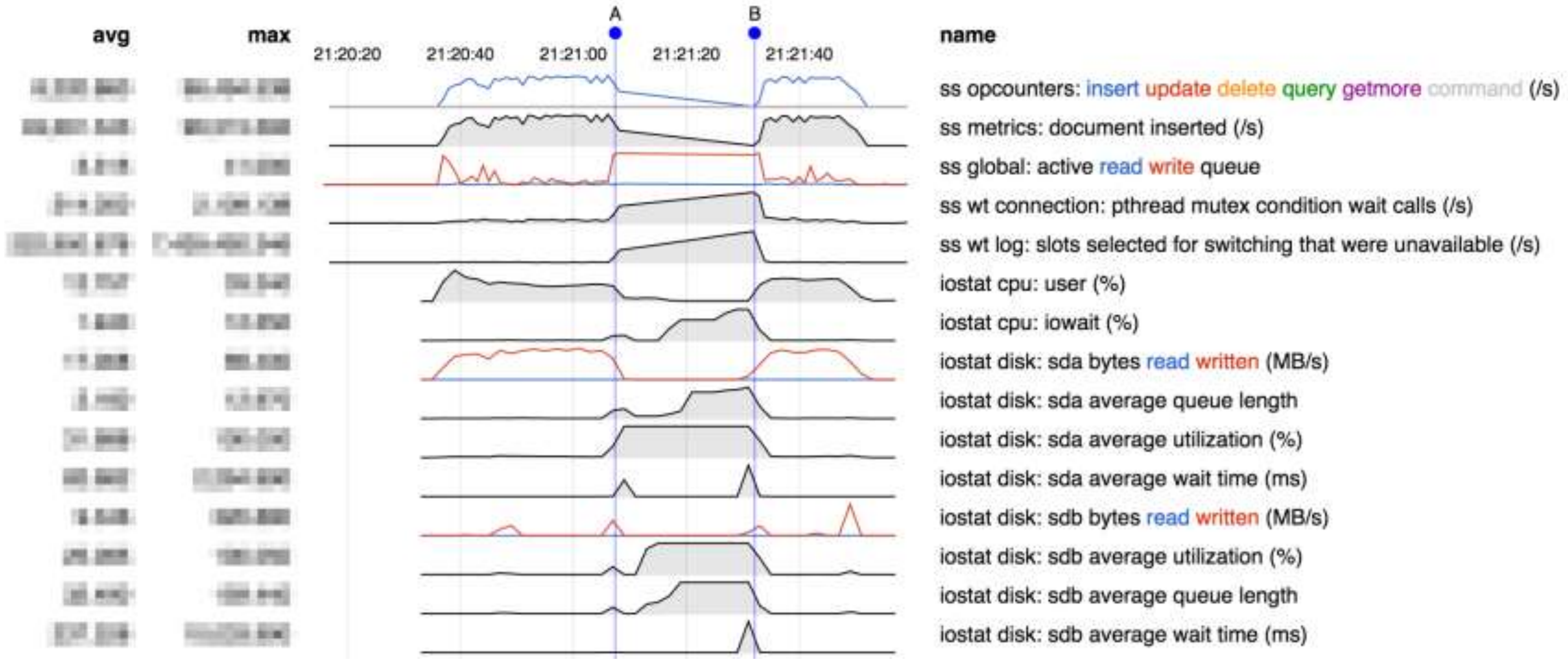
- Create indexes that support your queries!
- Create highly selective indexes
- Don't create unnecessary indexes and delete unused ones
- Eliminate duplicate indexes with a compound index, if possible

```
> db.collection.ensureIndex({A:1, B:1, C:1})
```

  – allows queries using leftmost prefix
- Order compound index fields thusly: equality, sort, then range
  – see http://emptysqua.re/blog/optimizing-mongodb-compound-indexes/
- Create indexes that support covered queries
- Prevent collection scans in *pre-production* environments

```
$ mongod --notablescan
> db.getSiblingDB("admin").runCommand( { setParameter: 1,
  notablescan: 1 } )
```

# Example: Hardware



mongoDB. FOR GIANT IDEAS

## Do's and Don’ts

- Do:
  - Read production notes in MongoDB documentation
  - Eliminate suspects in the right order (schema, indexes, operations, instance, hardware)
  - Know what is considered "normal" behavior by monitoring
- Don't:
  - confuse symptoms with root causes
  - shard a poorly performing system

mongoDB. FOR GIANT IDEAS

mongoDB

## Questions?