What next?
5 Critical Considerations

Alvin Richards

Technical Director, EMEA
alvin@10gen.com
@jonnyeight

http://gotocon.stefankohler.nl/

10gen mongoDB

Everything is Done

So, you’ve just created the worlds best wombat counting and sorting application ever. You’ve rolled it out to all your friends it works perfect and everyone agrees it will take the world by storm.

You’ve set up unit testing and a continuous integration framework, you’ve documented it to a tee, you’ve purchased beefy hardware and you’ve staffed up devops and sysadmin support personel.

Nothing is left to do but flip the switch and pop the champagne corks, right?

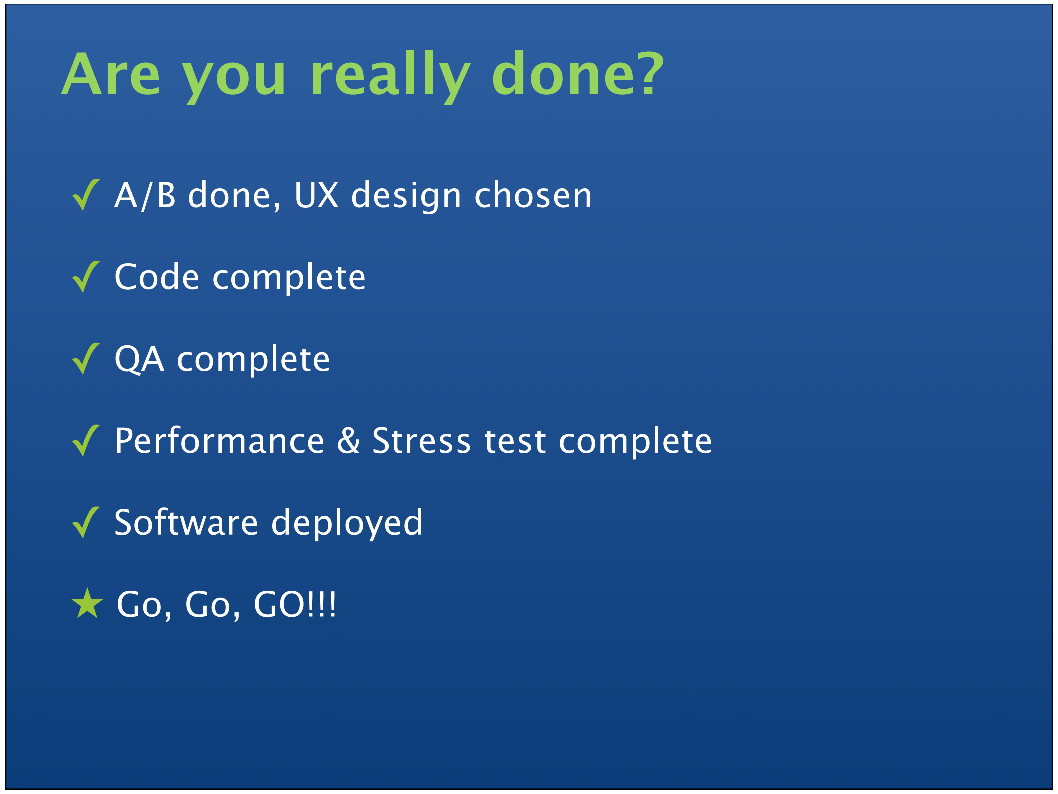
There's a few possible scenarios that can occur:

1. Everything just runs according to plan. Your growth curve is smooth and consistent, your schema design is efficient and hardware functions without a hitch. HAH

2. You've over-provisioned and lots of hardware is sitting idle. This is generally speaking less bad (and also generally less whined about). Of course disk manufacturers and cloud providers love this scenario.

3. Much worse scenario -- and much more commonly reported -- is underprovisioning. This is the catastrophic scenario where you've just opened up your application to the surging Australian wombat counting market and you see a 1000x increase in adoption. You keep celebrating for a few days until suddenly you fall off a cliff. Performance plummets and everything ends in tears.

4. Worser still. Say you're deployed in a spanking new data center. You've got an array of SSDs and the best hardware. Everything is going swimmingly until a pizza deliveryman drives into a transformer and knocks out power to half your center. Or less dramatically, a disk goes down and brings a member of your replica set down with it. Are you prepared for that? Do you have backups? Have you ever brought up a new replica set member? Or more typically, are you the only member of your team that knows the steps required to bring up that new member? We'll talk a little about the worst case scenarios and what you need to prepare for.



Are you really done?

- ✓ A/B done, UX design chosen
- ✓ Code complete
- ✓ QA complete
- ✓ Performance & Stress test complete
- ✓ Software deployed
- ★ Go, Go, GO!!!

not schema less - dynamic schema
schema is just as important, or more important than relational
understand write vs read tradeoffs

You are starting here...

http://community.qlikview.com/cfs-filesystemfile.ashx/__key/CommunityServer.Blogs.Components.WeblogFiles/theqlikviewblog/Cutting-Grass-with-Scissors-_2D00_-2.jpg

And want to get here...

http://www.bitquill.net/blog/wp-content/uploads/2008/07/pack_of_harvesters.jpg

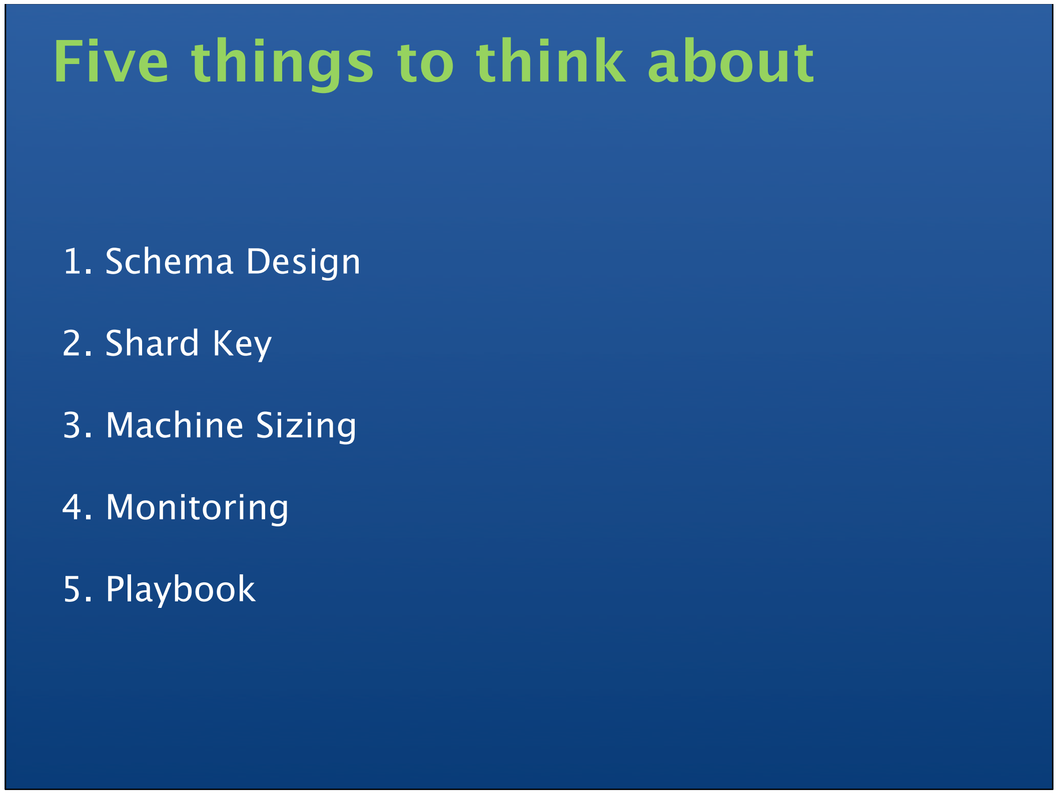
Avoiding this...

http://media.egotvonline.com/wp-content/uploads/2011/07/train-wreck-1.jpg

Avoiding the train wreck

You do all this to avoid the train wreck. They always occur first with a phone call at 2am when the system is down and then super human effort to get the system back and on a even track. This takes away your core resources and key talent from focusing on what the business needs, innovating and providing real value back to the business.

Some recipes of scaling from...



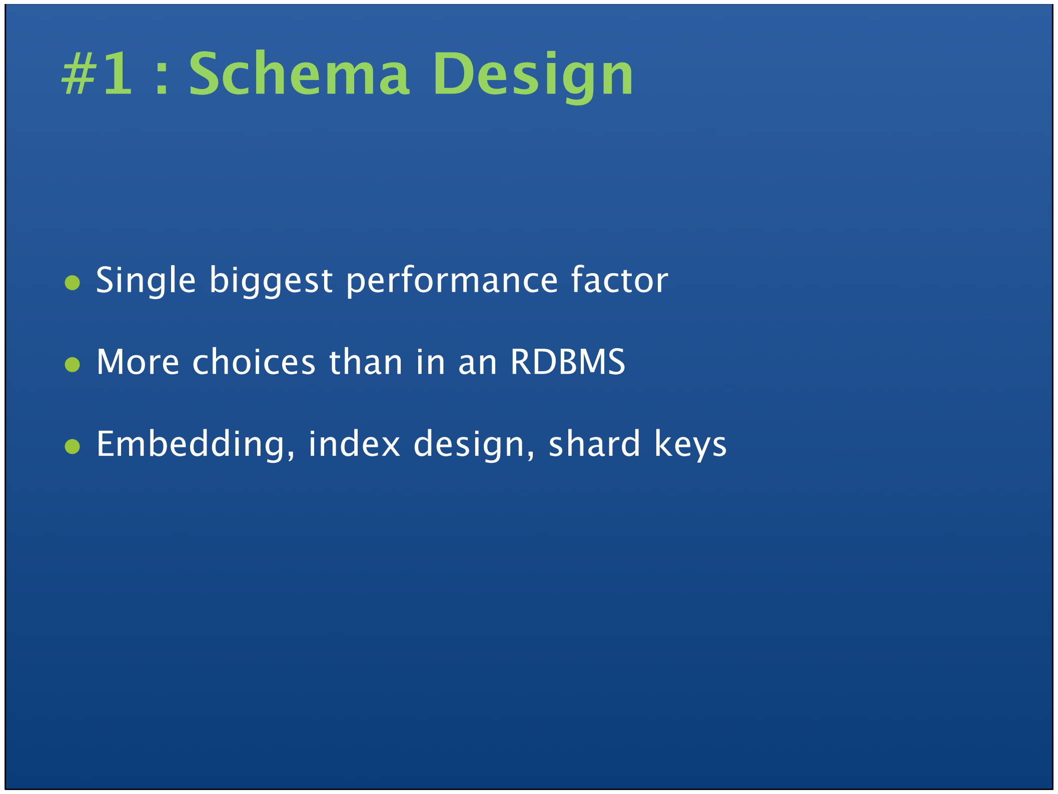
Five things to think about

1. Schema Design
2. Shard Key
3. Machine Sizing
4. Monitoring
5. Playbook

The things I'm going to talk about are completely inter-related and intertwined.

There will be talks that go into much greater details on these topics.

Armed with the information you gather and confident in the skills your team has practiced, you should be able to spot long term problems well before it's too late and handle the emergencies that are sure to arise.



#1 : Schema Design

- Single biggest performance factor
- More choices than in an RDBMS
- Embedding, index design, shard keys

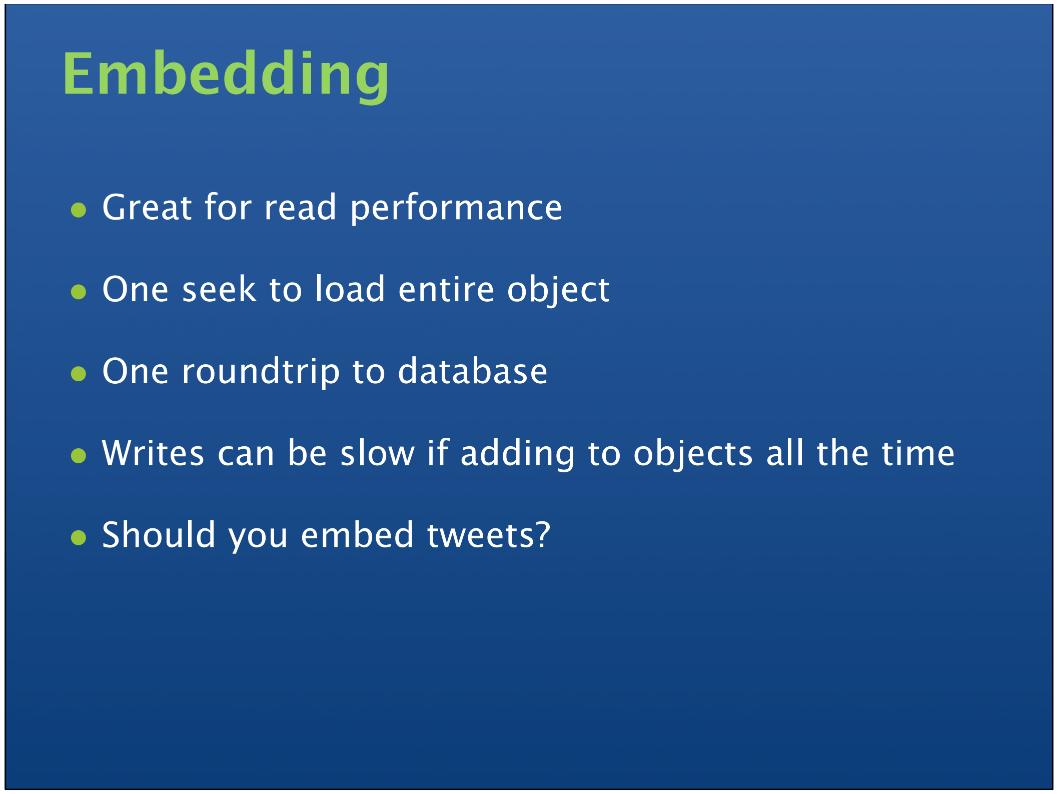
not schema less - dynamic schema
schema is just as important, or more important than relational
understand write vs read tradeoffs

Activity Stream – Embedded

```
// users - one doc per user with all tweets
{  _id:    "alvin",
   email: "alvin@10gen.com",
   tweets: [
     {
        user:   "bob",
        tweet: "20111209-1231",
        date: ISODate("2011-09-18T09:56:06.298Z"),
        text:   "Best Tweet Ever!"
     }
  ]
}
```

Activity Stream – Linking

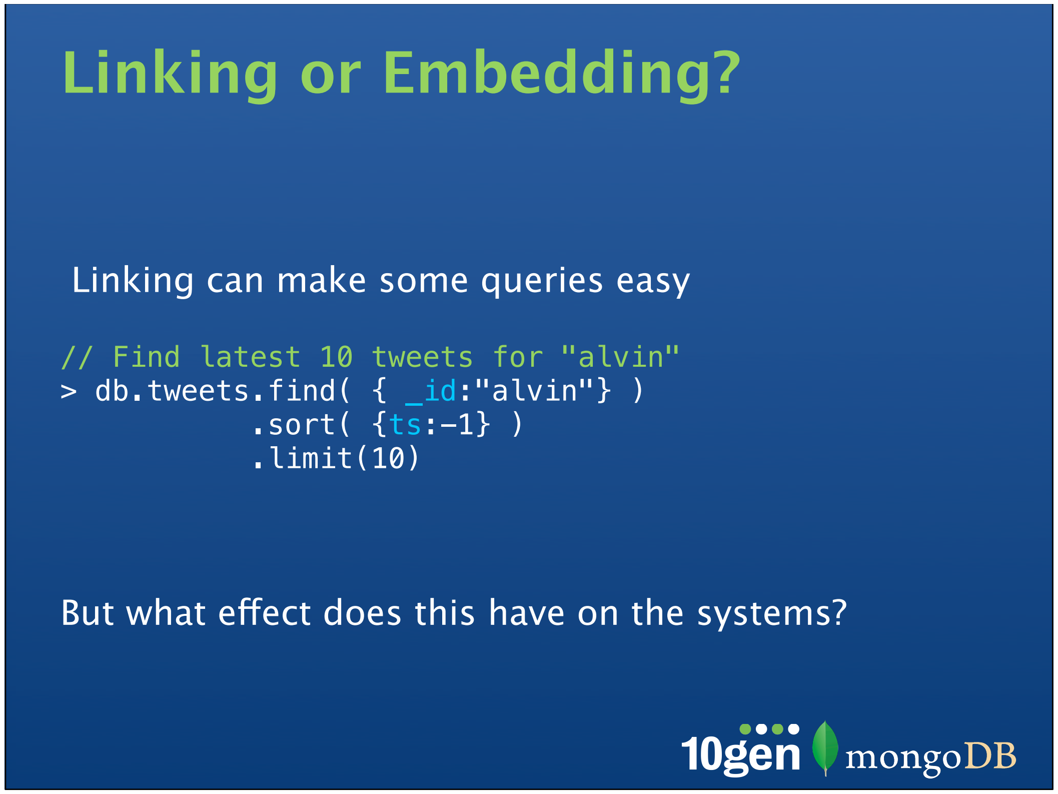
```
// users - one doc per user
  {  _id:    "alvin",
     email: "alvin@10gen.com"
  }
// tweets - one doc per user per tweet
  {
      user:   "bob",
      tweet: "20111209-1231",
      date: ISODate("2011-09-18T09:56:06.298Z"),
      text:   "Best Tweet Ever!"
  }
```



Embedding

- Great for read performance
- One seek to load entire object
- One roundtrip to database
- Writes can be slow if adding to objects all the time
- Should you embed tweets?

compare to mysql here



Linking or Embedding?

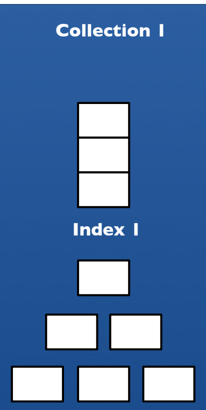
Linking can make some queries easy

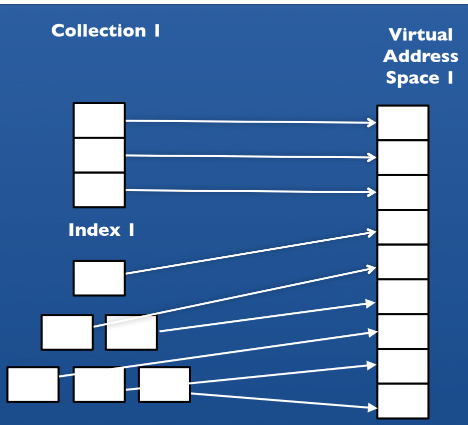
```
// Find latest 10 tweets for "alvin"
> db.tweets.find( { _id:"alvin"} )
           .sort( {ts:-1} )
           .limit(10)
```

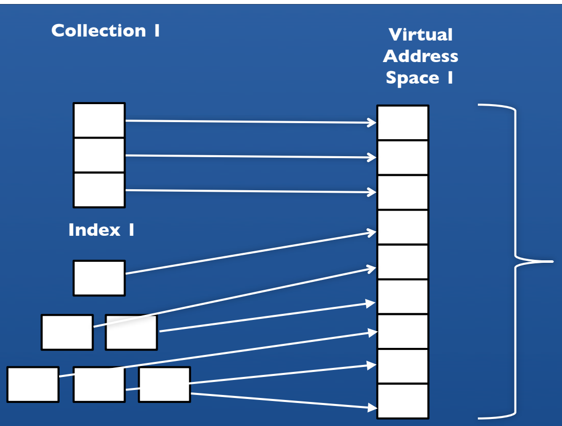
But what effect does this have on the systems?

Linking makes filtering on the many end very simple, but a good designer must understand the implications of their choice.

The following sections tries to show that some designs may cause longer response times because of how data is mapped to memory and to disk and the cost of memory versus disk access.

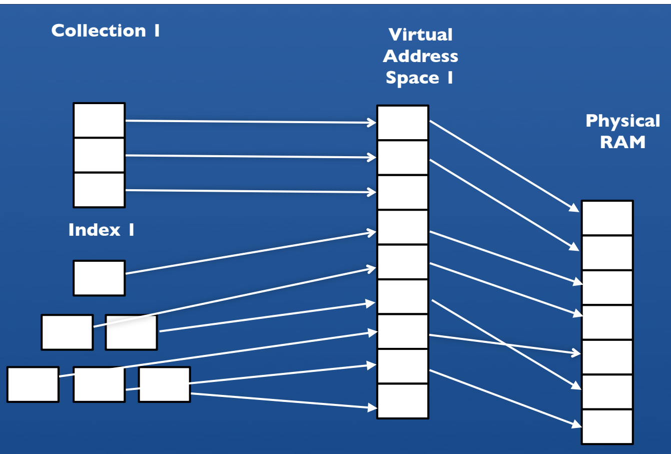

Collection 1
Index 1

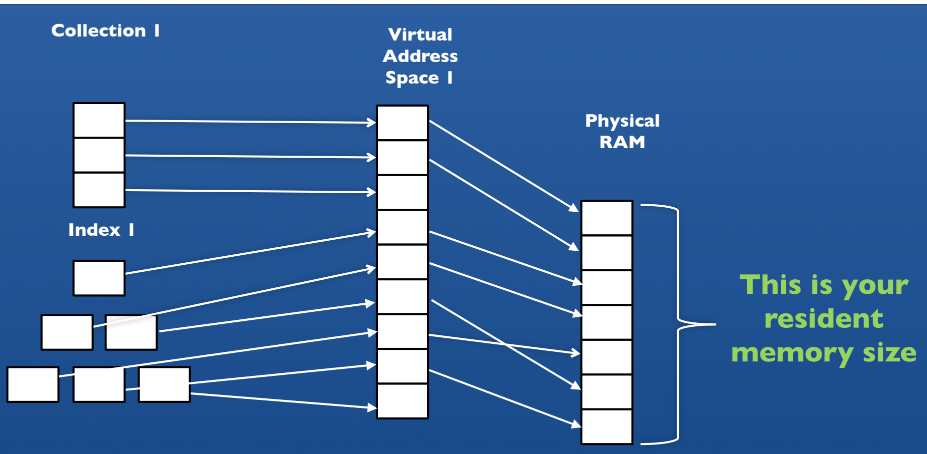

Collection 1
Virtual Address Space 1
Index 1



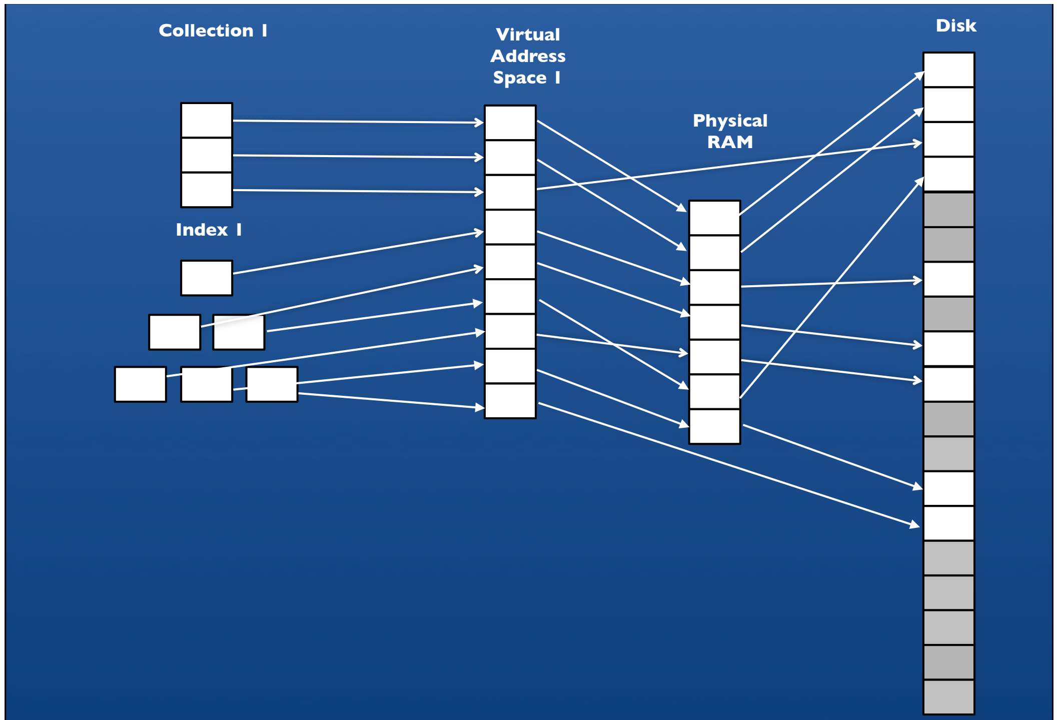
This is your virtual memory size (mapped)

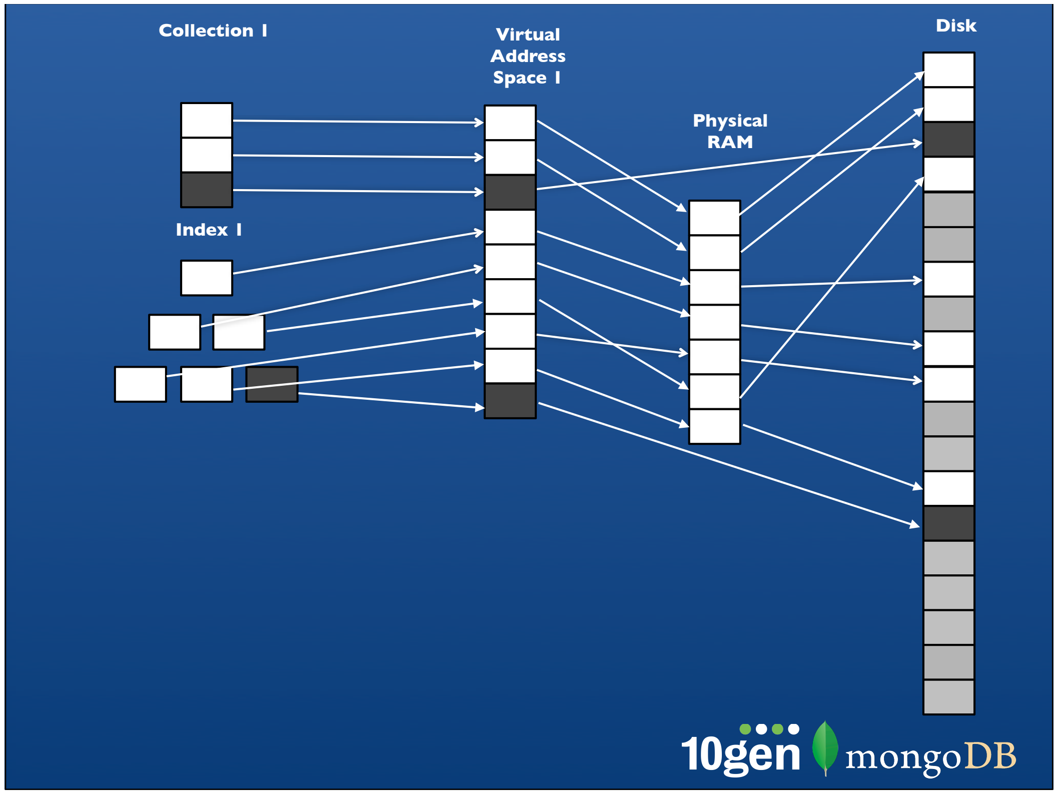
10gen mongoDB


Collection 1
Virtual Address Space 1
Physical RAM
Index 1

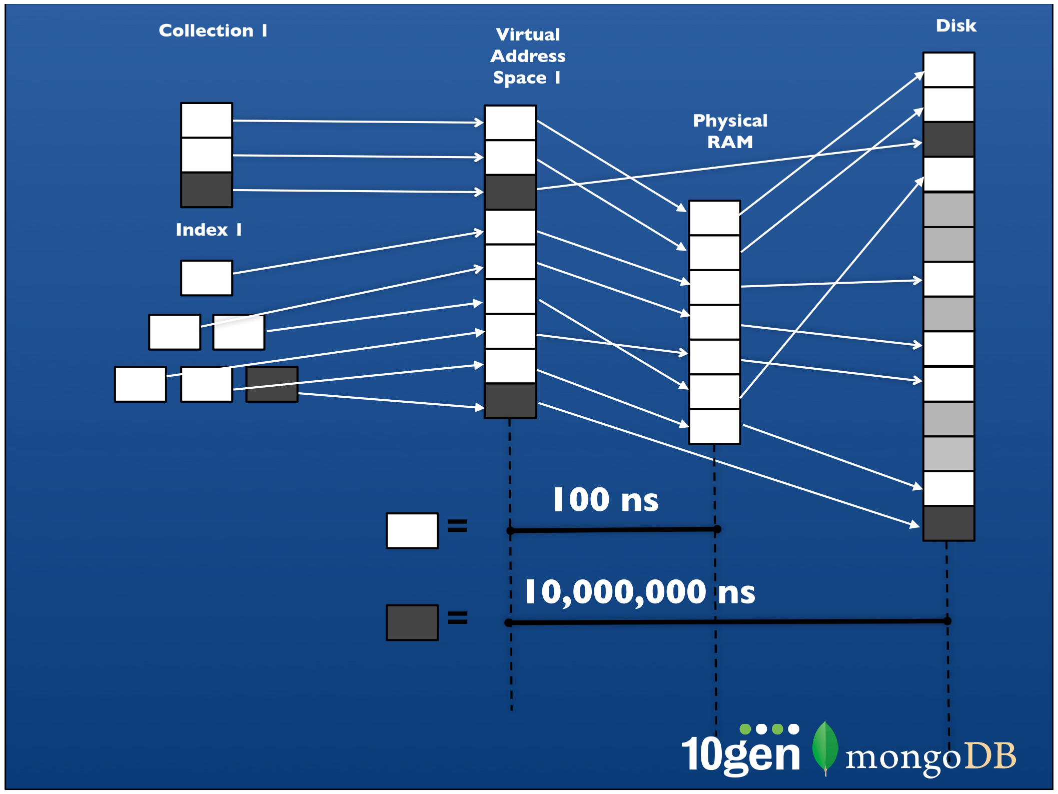


10gen mongoDB


Collection 1
Virtual Address Space 1
Disk
Physical RAM
Index 1



Key point here is that not all data is in memory.



Can map this back to the linking versus embedding question.

For an embedded object, the worst case is page fault(s) to read the object, but on disk this will be a seek followed by a sequential read

For a linked objects, the worst case is a page fault for each, which is a disk seek for each linked object

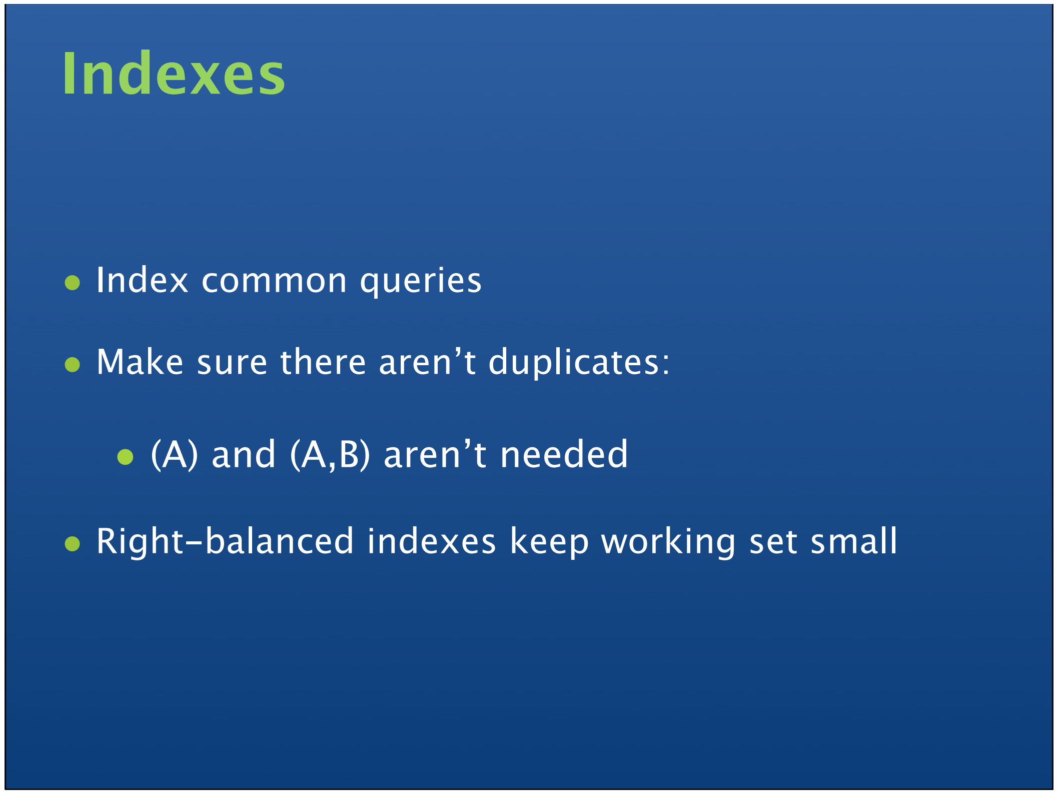
Activity Stream – Buckets

```
// tweets : one doc per user per day

   {
     _id: "alvin-20111209",
     email: "alvin@10gen.com",
     tweets: [
       { user:  "Bob",
         tweet: "20111209-1231",
         text:  "Best Tweet Ever!" } ,
       { author: "Joe",
         date:   "May 27 2011",
         text:   "Stuck in traffic (again)" }
     ]
    }
```

Last 10 Tweets

```
db.tweets.find( { _id: "alvin-2011/12/09" },
                { tweets: { $slice : 10 } }
              )
         .sort( { _id: -1 } )
         .limit(1)
```



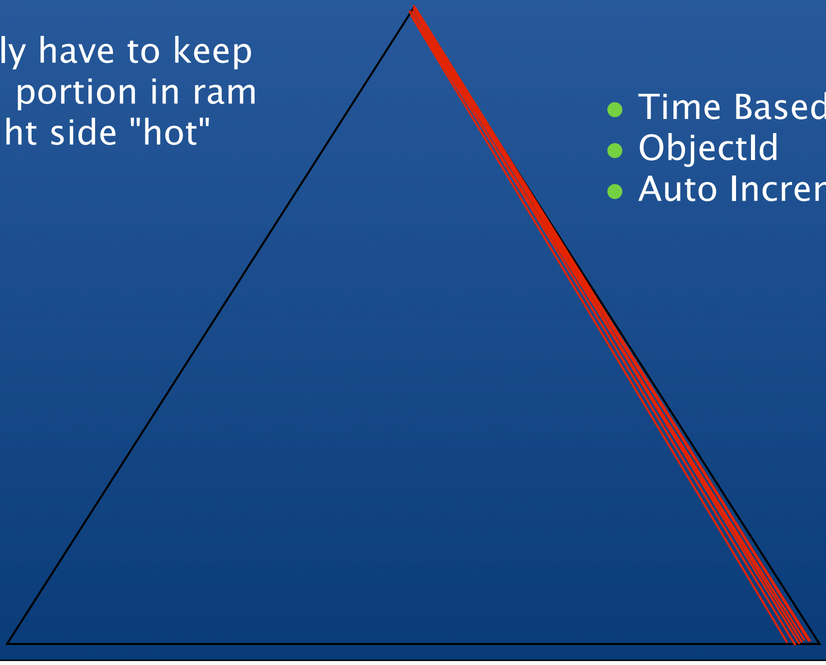
Indexes

- Index common queries
- Make sure there aren’t duplicates:
    - (A) and (A,B) aren’t needed
- Right-balanced indexes keep working set small

most common performance problem
why _id index can be ignored

Right balanced access

- Only have to keep small portion in ram
- Right side "hot"

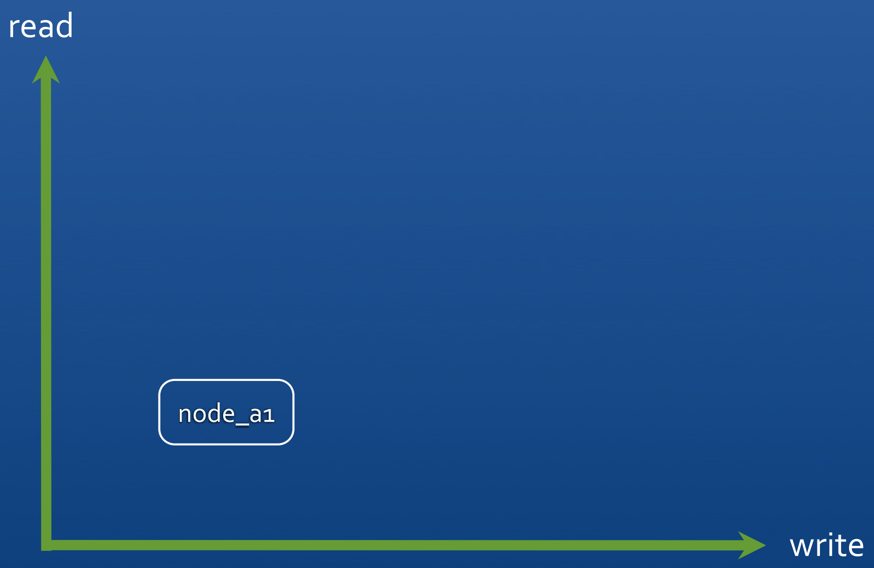


- Time Based
- ObjectId
- Auto Increment

# 2 : Choosing a Shard Key

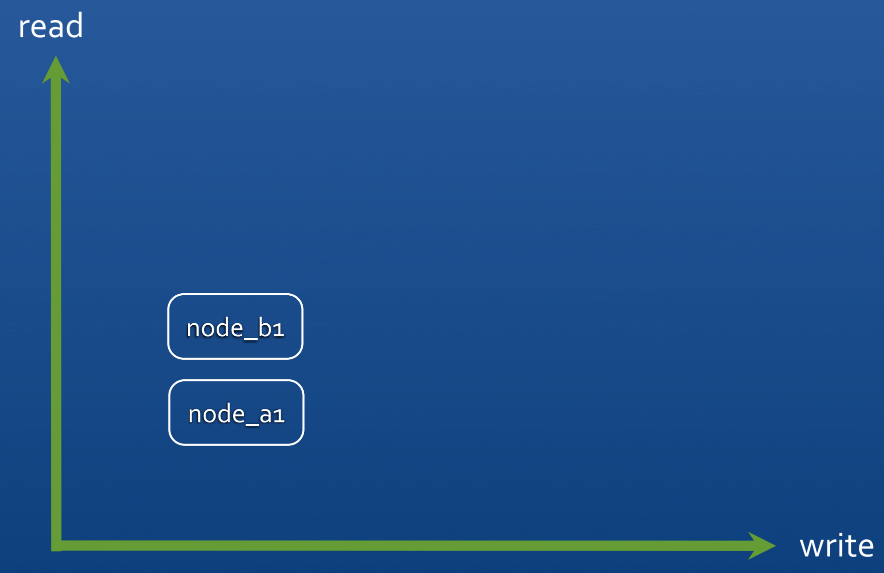
- Sharding is used to scale writes <u>and</u> reads
- Shard key determines how data is partitioned
- Critical scalability decision

MongoDB Scaling - Single Node



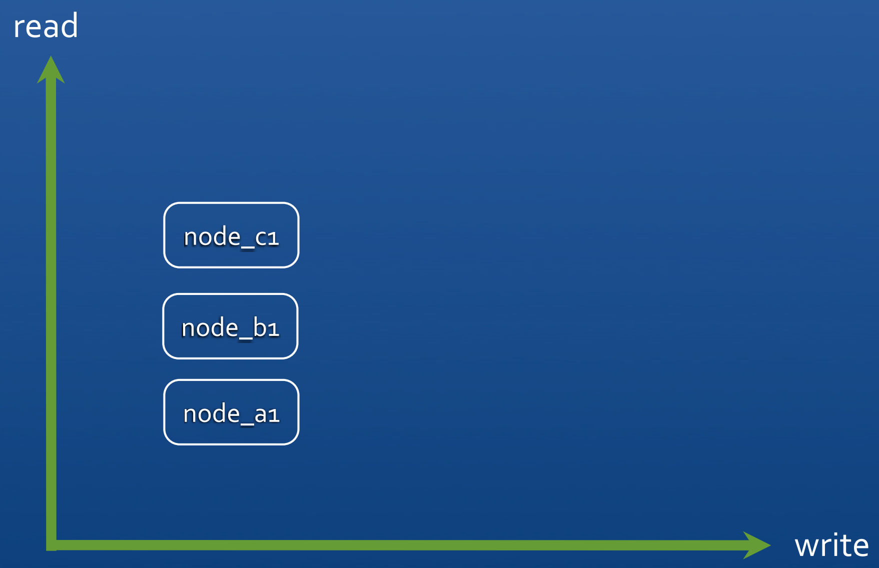
10gen mongoDB

Read scaling - add Replicas



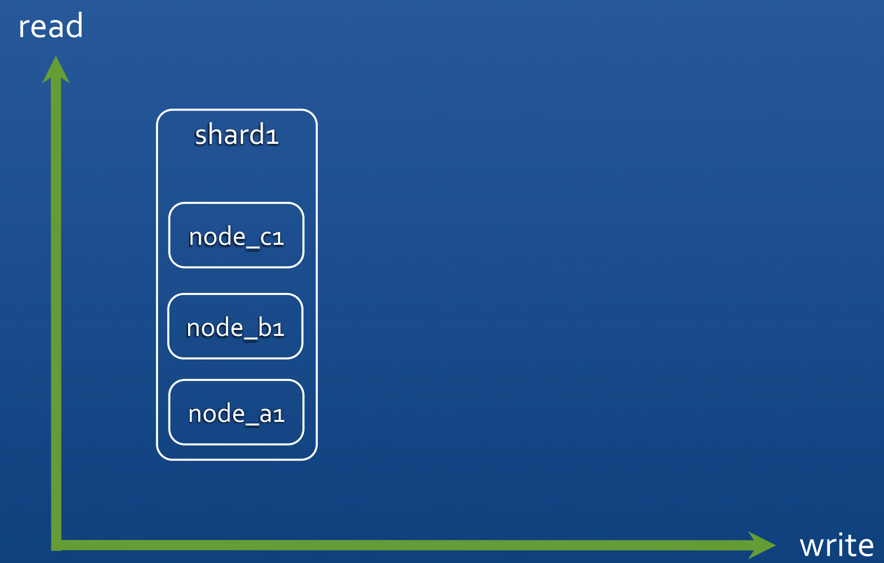
10gen mongoDB

Read scaling - add Replicas



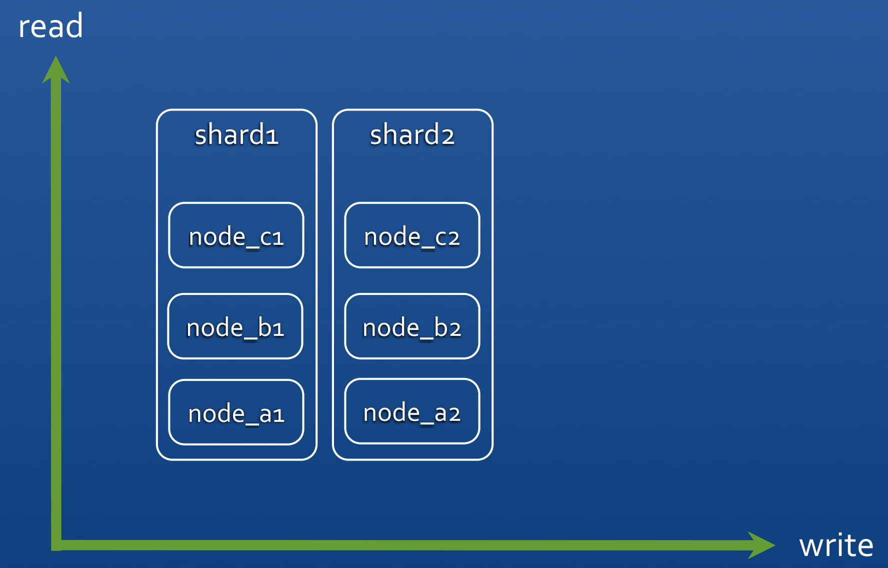
10gen mongoDB

Write scaling - Sharding



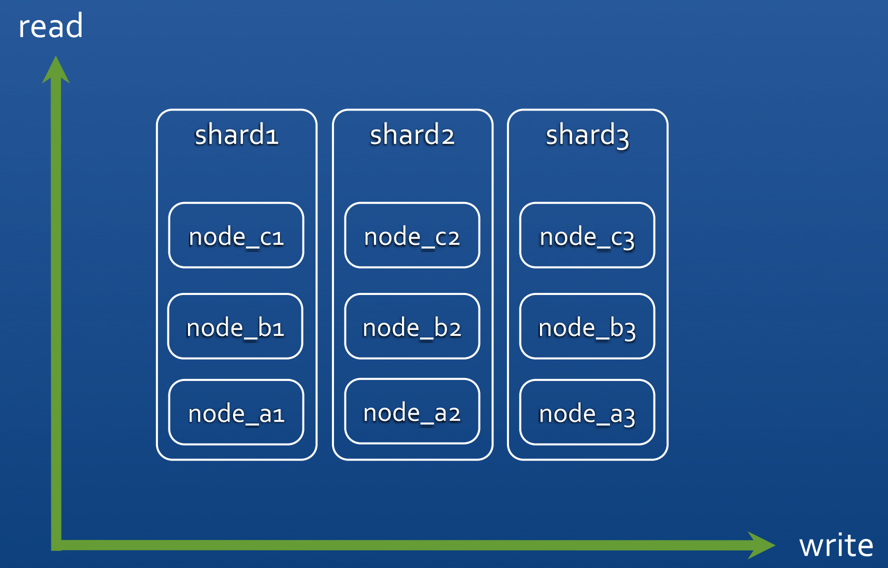
10gen mongoDB

Write scaling - add Shards



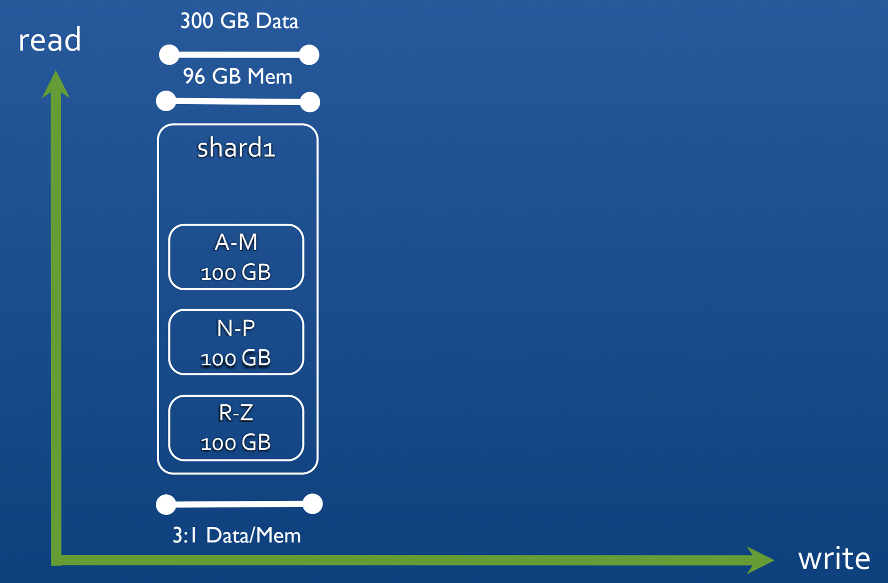
10gen mongoDB

Write scaling - add Shards

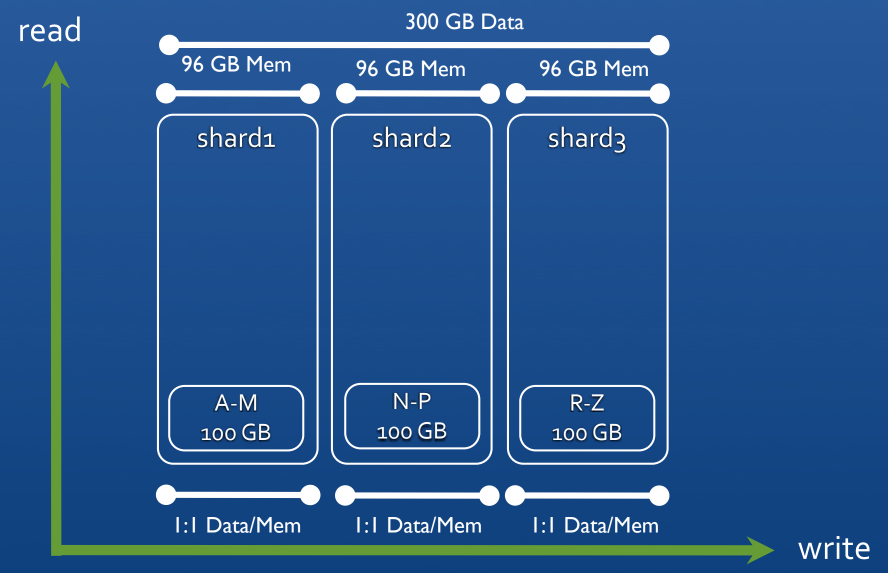


10gen mongoDB

Sharding for caching



Sharding for caching



Auto Sharding

- Fully consistent
- Application code unaware of data location
- Zero code changes
- Add capacity
    - On-line
    - When needed
    - Zero downtime

Use Case: Photos

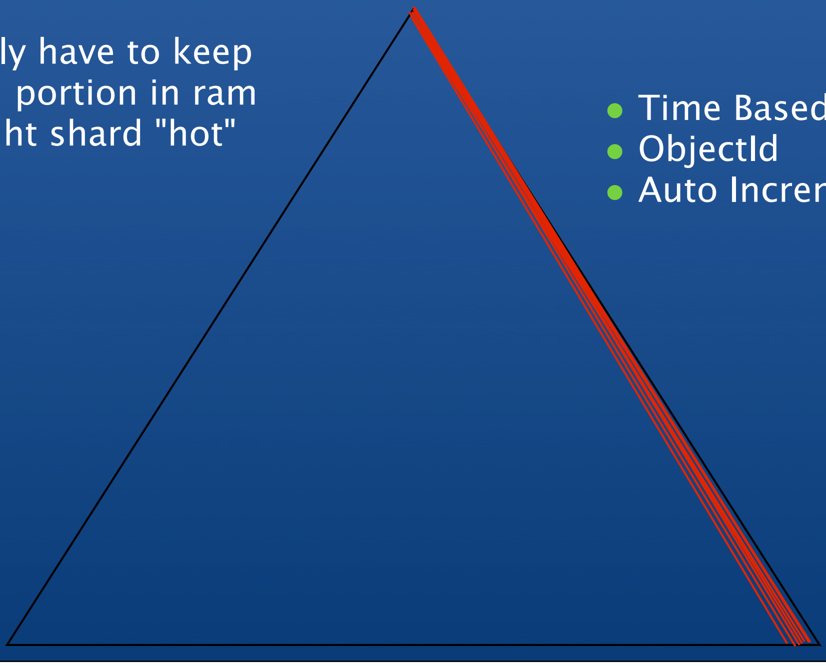
```
{ photo_id :  ???? , data : <binary> }
```

What’s the right key?

- auto increment
- MD5(data)
- month() + MD5(data)

Right balanced access

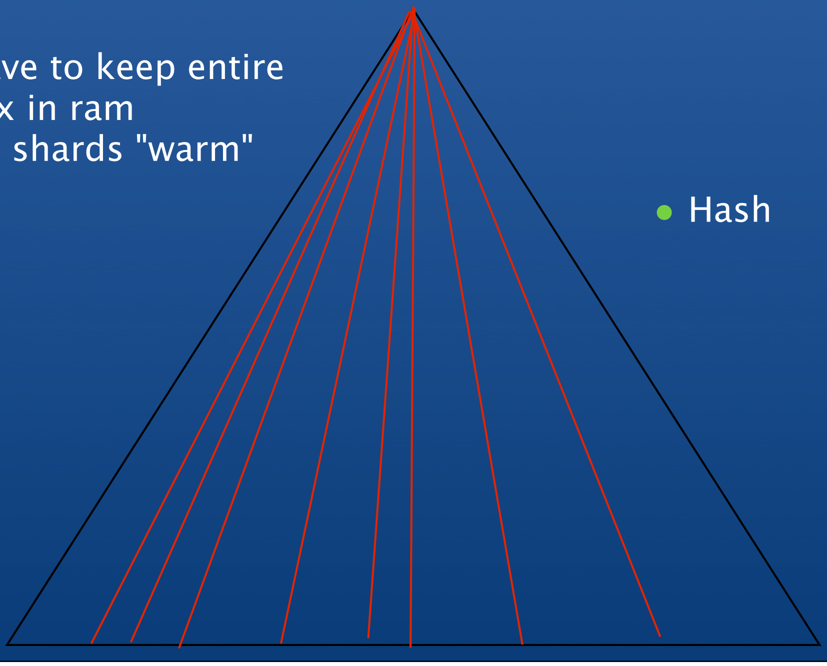
- Only have to keep small portion in ram
- Right shard "hot"



- Time Based
- ObjectId
- Auto Increment

Random access

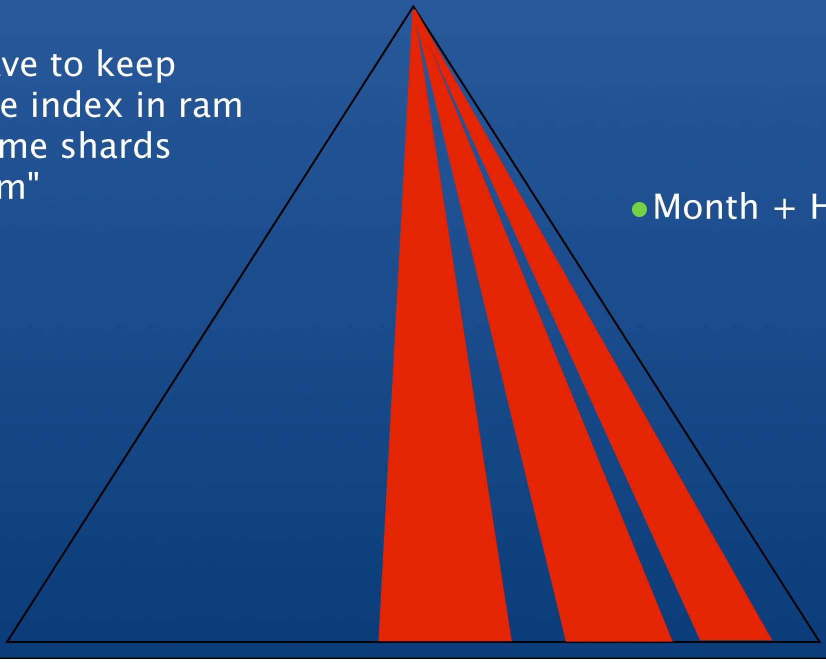
- Have to keep entire index in ram
- All shards "warm"



- Hash

Segmented access

- Have to keep entire index in ram
- Some shards "warm"



- Month + Hash

Use Case: User Profiles

```
{ email : "alvin@10gen.com" ,
  addresses : [ { state : "CA", country: "USA" },
                { country: "UK" } ]
}
```

- Shard on { email : 1 }
- Lookup by email hits 1 node
- Index on { “addresses.country” : 1 }

Use Case: User Profiles
Multiple Identities

```
{ email: "alvin@10gen.com",
  facebook: "alvin.richards",
  twitter:  "jonnyeight",
  addresses : [ { state : "CA", country: "USA" },
                { country: "UK" }
 ]
}
```

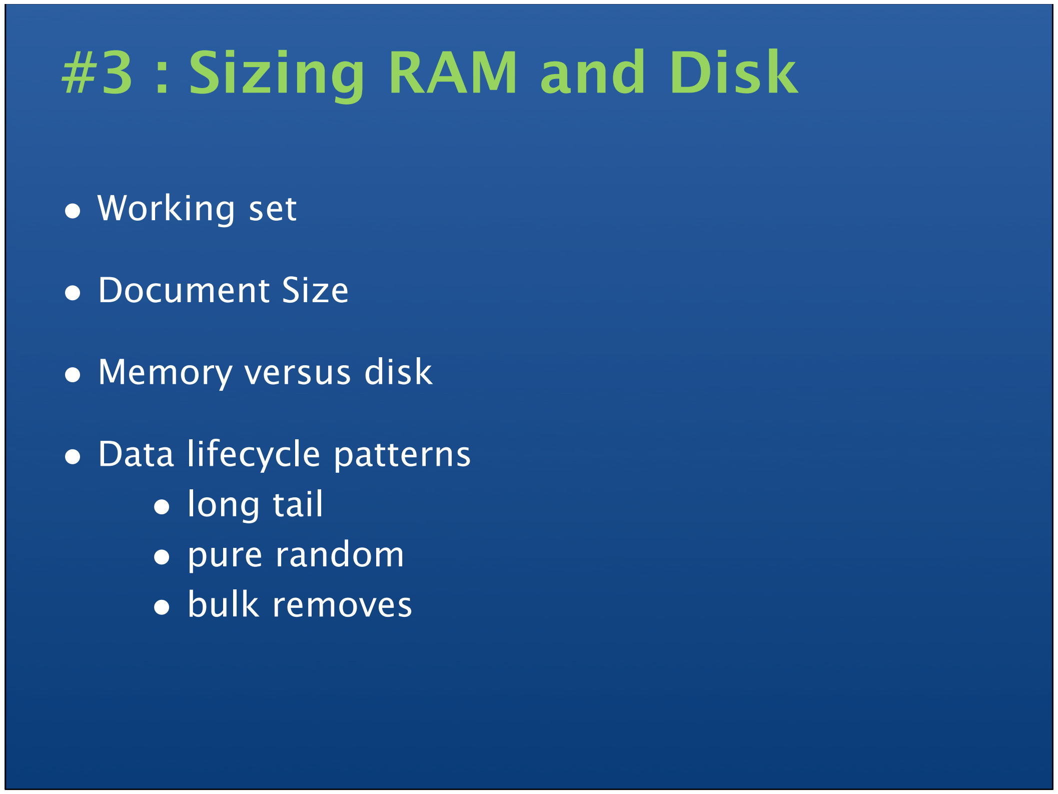
- Shard on { email:1 }
- Lookup by email hits 1 node
- Lookup by twitter or facebook is scatter gather

Use Case: User Profiles Multiple Identities – linking

```
identities
{ type: "email", val: "alvin@10gen.com", info: "1200-42"}
{ type: "fb",    val: "alvin.richards",  info: "1200-42"}
{ type: "tw",    val: "jonnyeight",      info: "1200-42"}

info
{ _id: "1200-42",
  addresses : [ { state : "CA", country: "USA" },
                { country: "UK" }]
}
```

- Shard identities on { type : 1, val : 1 }
- Lookup by type & val hits 1 node
- Shard info on { _id: 1 }
- Lookup on _id hits one node



#3 : Sizing RAM and Disk

- Working set
- Document Size
- Memory versus disk
- Data lifecycle patterns
    - long tail
    - pure random
    - bulk removes

Since we're talking about data stores, specifically, MongoDB, before you do anything else at all, you need to understand your data.

How big is your data set in total?
How big is your working set? that is, the size of the data and indexes that need to fit in RAM
Reads vs. writes? (example and use case)
Long tail or random access? (example)

Armed with this knowledge, you can accommodate both massive growth spurts without excessive over-provisioning.

Random access:
Take a user database
Long tail:
Twitter feed
You need to be ready for 1MM users, how do I size my
Use collection.stats to extrapolate

Determine the working set

```
> db.profiles.stats()
{
	"ns" : "test.profiles",
	"count" : 1338330,
	"size" : 46915928,
	"avgObjSize" : 35.05557523181876,
	"storageSize" : 86092032,
	"numExtents" : 12,
	"nindexes" : 2,
	"lastExtentSize" : 20872960,
	"paddingFactor" : 1,
	"flags" : 0,
	"totalIndexSize" : 99860480,
	"indexSizes" : {
		"_id_" : 55877632,
		"name_1" : 43982848
	}
}
```

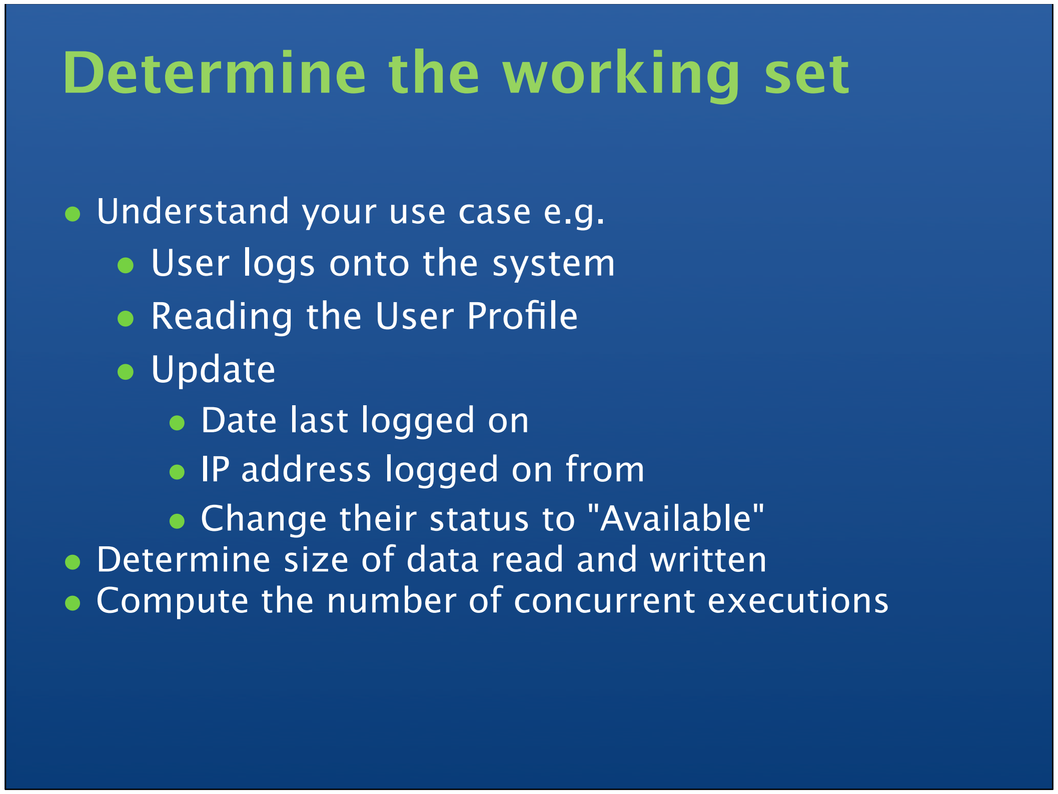
Size of data

Average doc size

Size on disk (and in memory!)

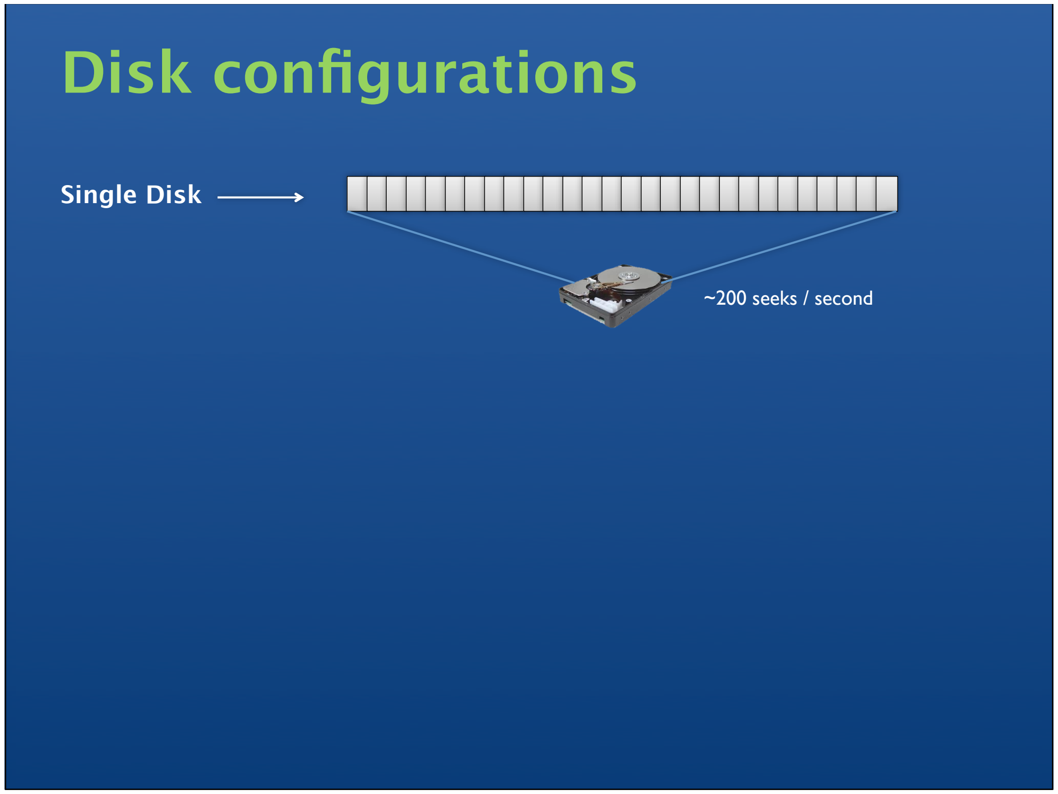
Size of indexes

Size of each index



Determine the working set

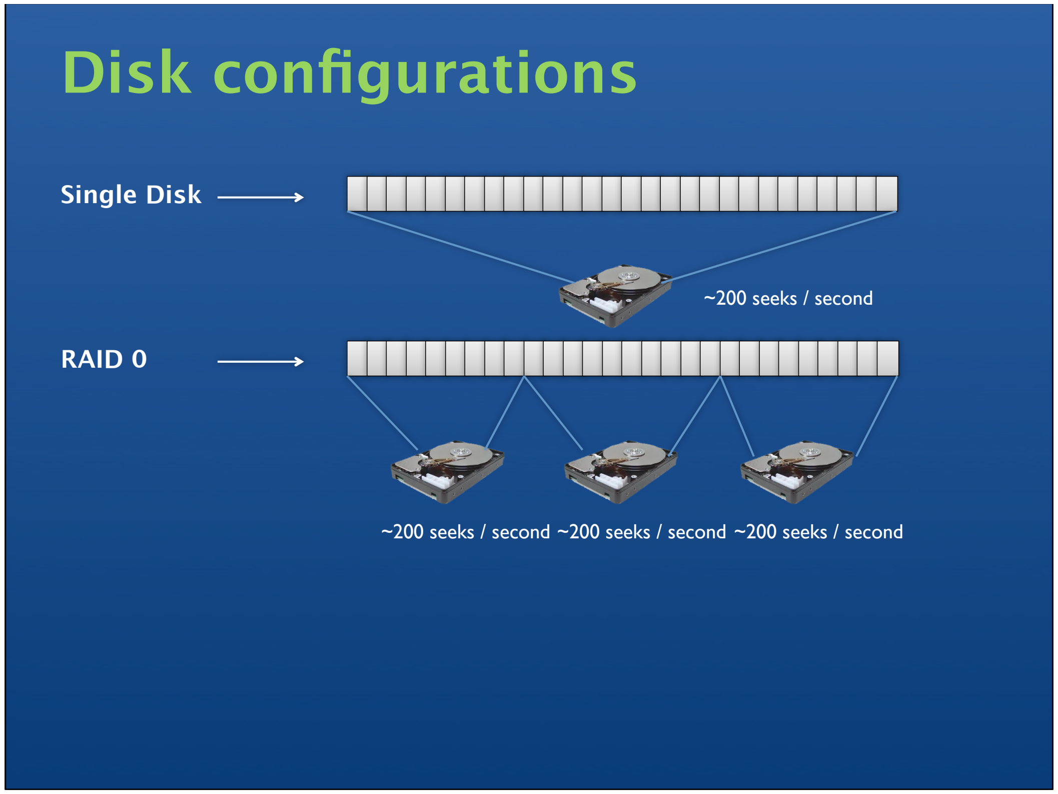
- Understand your use case e.g.
  - User logs onto the system
  - Reading the User Profile
  - Update
    - Date last logged on
    - IP address logged on from
    - Change their status to "Available"
- Determine size of data read and written
- Compute the number of concurrent executions



Disk configurations

Using standard enterprise spinning disks you can get about 200 seeks / second

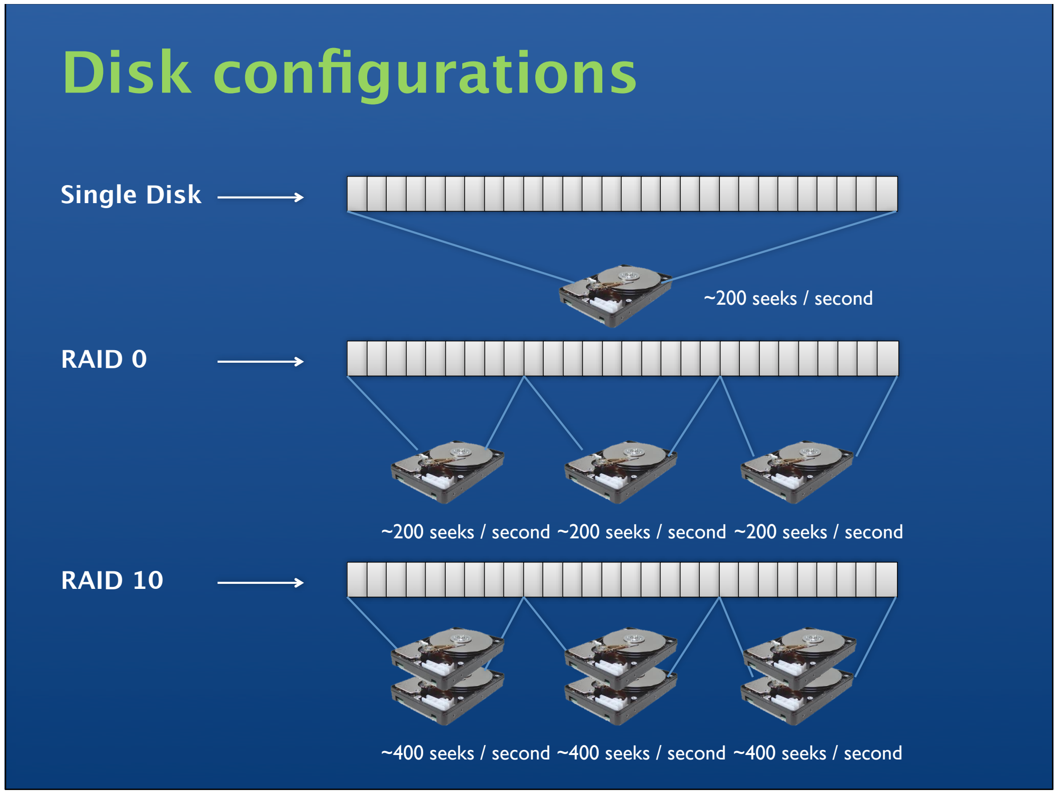
So, you want to be thinking about how you can increase my seeks / second



Disk configurations

Here, if you can imagine that you're not pulling all your data from a single partition, you can actually increase you throughput by spreading the load across multiple stripes.

So in this case gaining potentially three times the speed.



Disk configurations

Single Disk
~200 seeks / second
RAID 0
~200 seeks / second ~200 seeks / second ~200 seeks / second
RAID 10
~400 seeks / second ~400 seeks / second ~400 seeks / second

What we typically recommend to run RAID10 in production which adds a mirror volume for each stripe.

We've found that this configuration really works out well for most use cases.

You get the benefit of increased redundancy and parallelization, despite the cost of writing each update to two volumes.

SSD vs Disk?

- Seek time of 0.1ms vs 5ms
- 10000 seeks / sec vs 200 seeks / sec

But expensive (but getting cheaper)

Consider SDD for
- Journal files
- Specific Databases

Takeaway

Know how important page faults are

- If you want low latency, avoid page faults

Size memory appropriately

- Avoid page faults, fit working set in RAM
- Collection Data + Index Data

Provision disk appropriately

- RAID10 is recommended
- SSD's are fast, if you can afford them

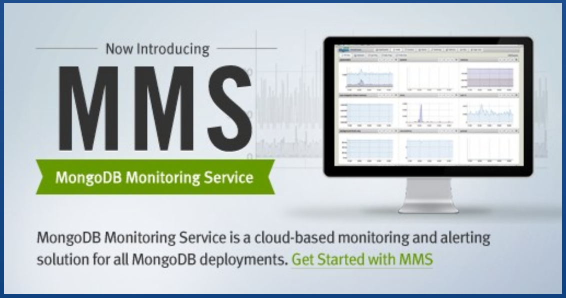
Choose the right file-system

- EXT4 or XFS

#4 : Monitoring

MongoDB Monitoring Service

- SaaS solution providing instrumentation and visibility into MongoDB systems
- Included in the 10gen commercial subscriptions
- Free community version available
- 3,500+ customers signed up and using service



MongoDB Monitoring Service is a cloud-based monitoring and alerting solution for all MongoDB deployments. Get Started with MMS

#5 : Play book

Backups are a bit trickier than just taking a copy of your dbpath files..

As your dataset grows, you may be surprised that a simple mongodump command that worked with your 500MB dataset is inappropriate once you grow to 50GB.

Load Testing!

Understand what you think the system should do

- Load and test your hypothesis
  - Use the profiler e.g. db.setProfilingLevel(2)
- Use a trending monitoring tool to analyze
  - MMS, munin, etc.

Backups

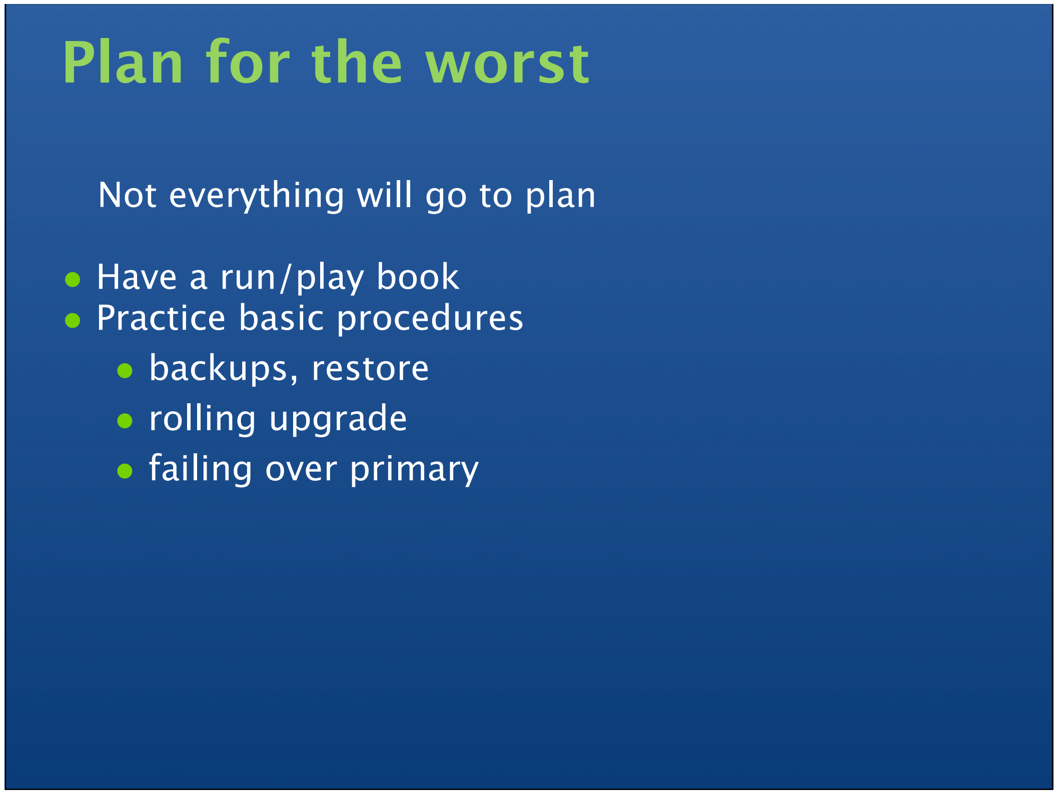
Backup

- mongodump reads all data (page faults), write to file
- snapshots avoid page faults
- journaling avoids need to fsync+lock

Restore

- member, whole rep set, whole cluster

Don't forget your config dbs in a sharded cluster



Plan for the worst

Not everything will go to plan

- Have a run/play book
- Practice basic procedures
  - backups, restore
  - rolling upgrade
  - failing over primary

If you’re not there, can *SOMONE* go and provision a new instance? Is there a script? At least a list of manual steps.

Ops Playbook and process references:
Can you back up? Can you restore? Can the new guy who’s on call on Saturday night do it?

Monitoring:
oz of prevention
Understand how to read the charts

Resources:
IRC, office hours, jira

Who is 10gen ?

- Began the MongoDB project
  - Core maintainers
  - Own the copyright, distribute under aGPL 3.0
- Provide MongoDB services
  - Support, Subscribers Build, Monitoring
  - Consulting, Training
- 120+ employees
- Offices in New York, Palo Alto, London, Dublin & Sydney
- Investors: Sequoia, Flybridge, Union Square

download at mongodb.org

We're Hiring !

http://www.10gen.com/jobs

conferences, appearances, and meetups

http://www.10gen.com/events

Facebook | Twitter | LinkedIn

http://bit.ly/mongofb @mongodb http://linkd.in/joinmongo

10gen mongoDB