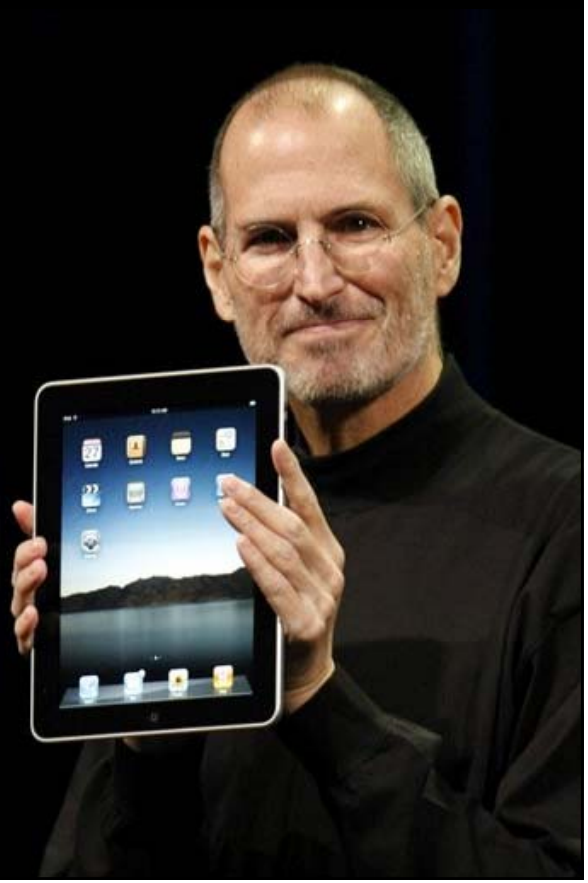
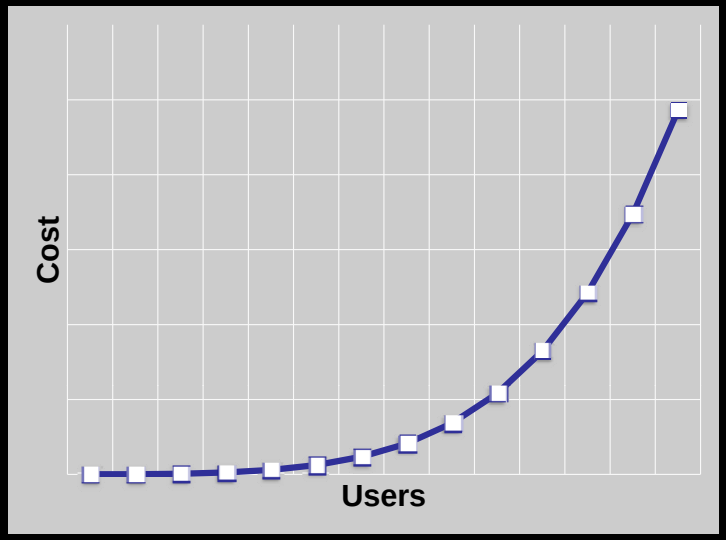
Cost effective scaling for the enterprise

You just released your new product

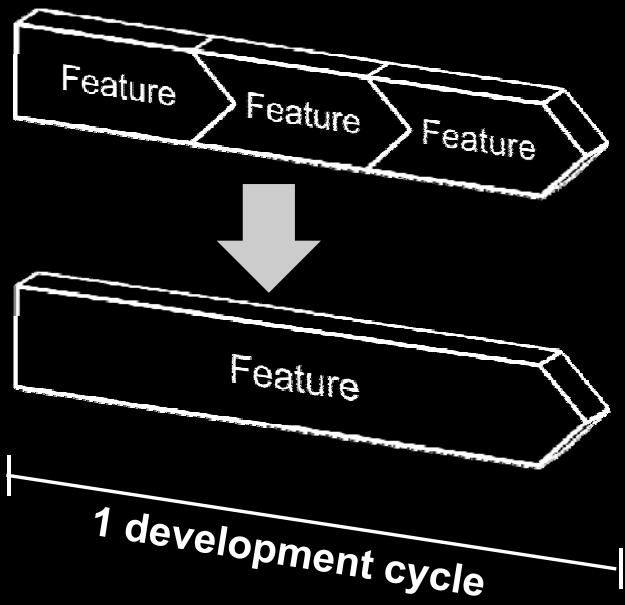


But there are troubling signs

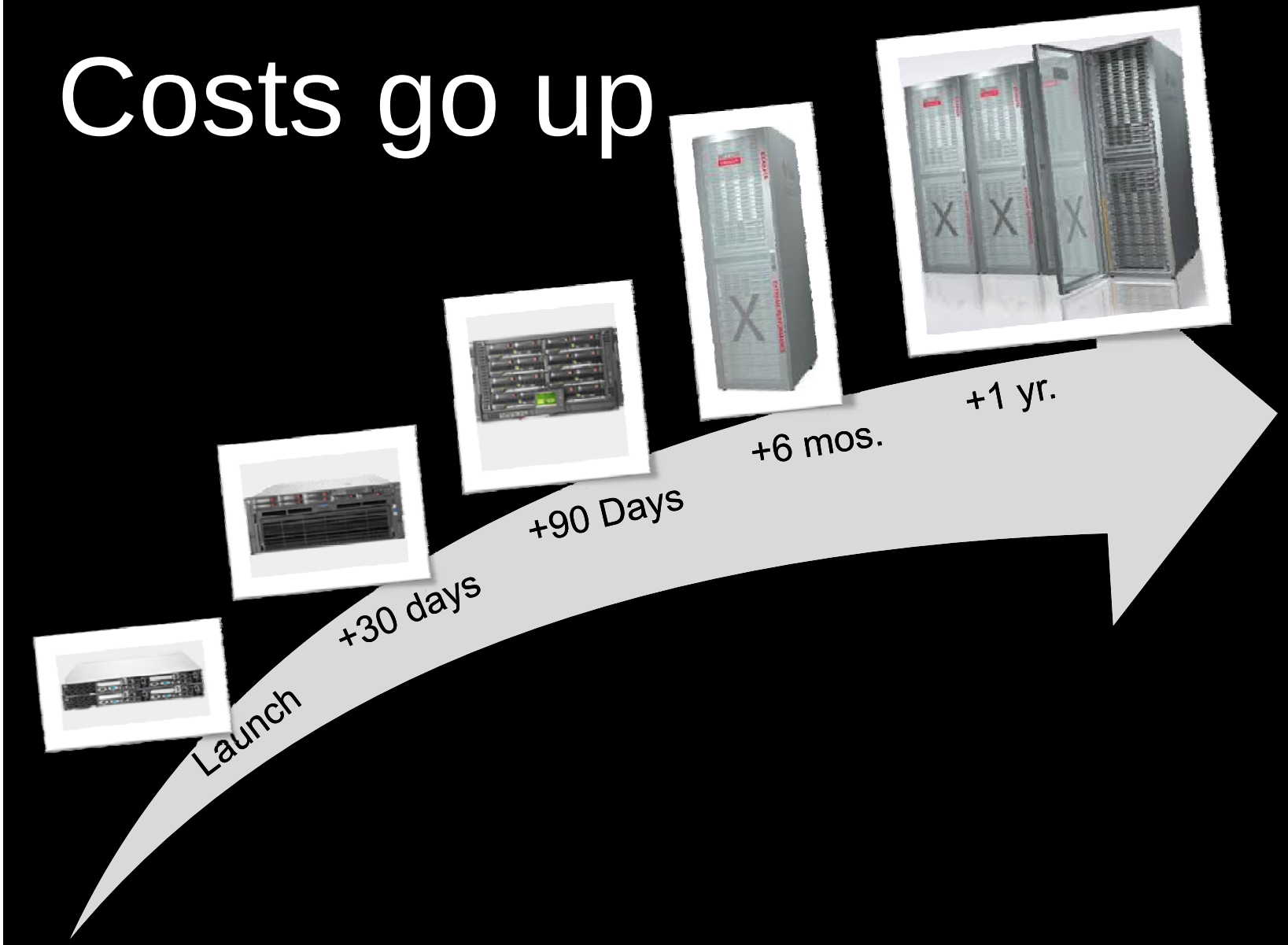
- Infrastructure costs grows faster than revenue



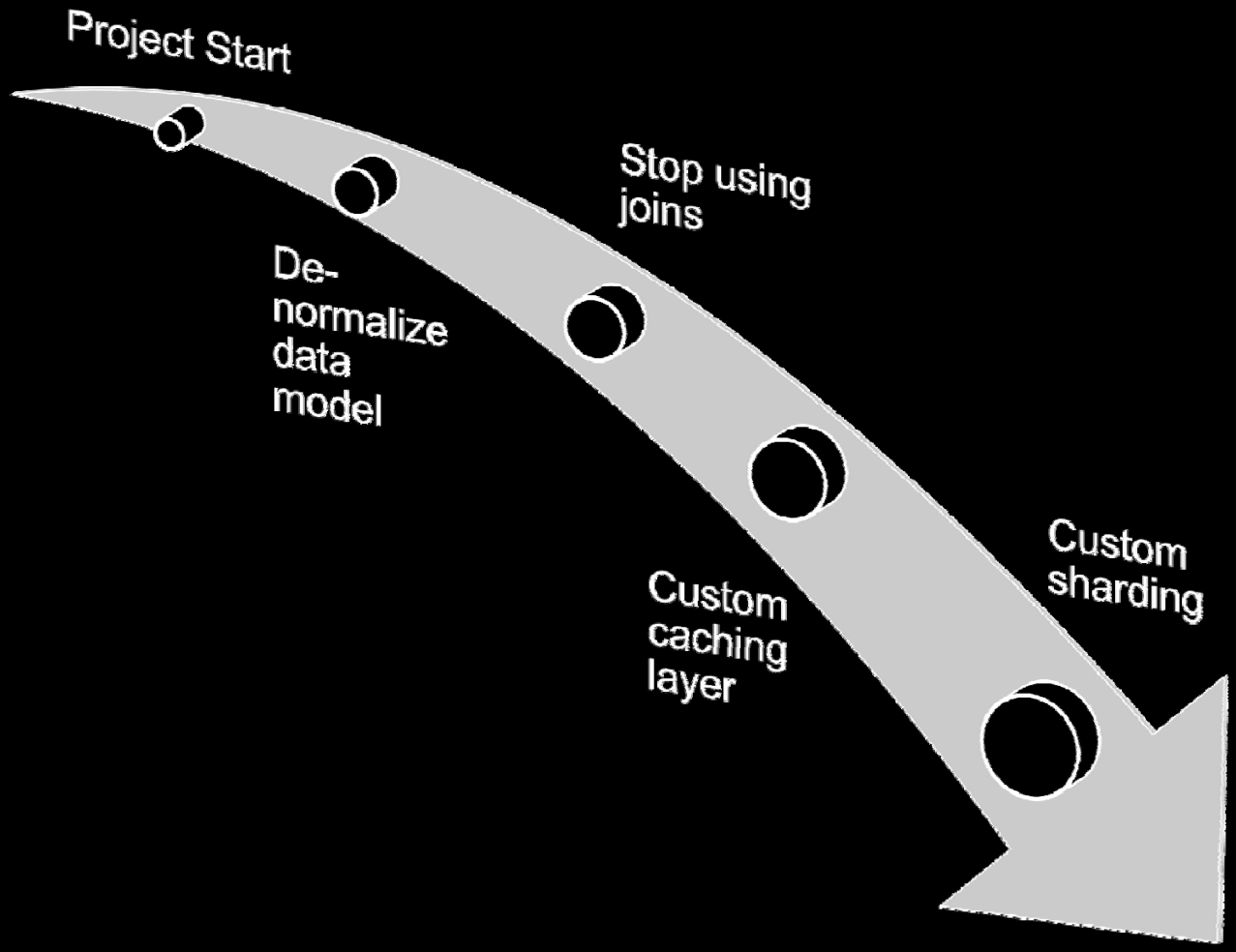
- Development timeline slows down



Costs go up



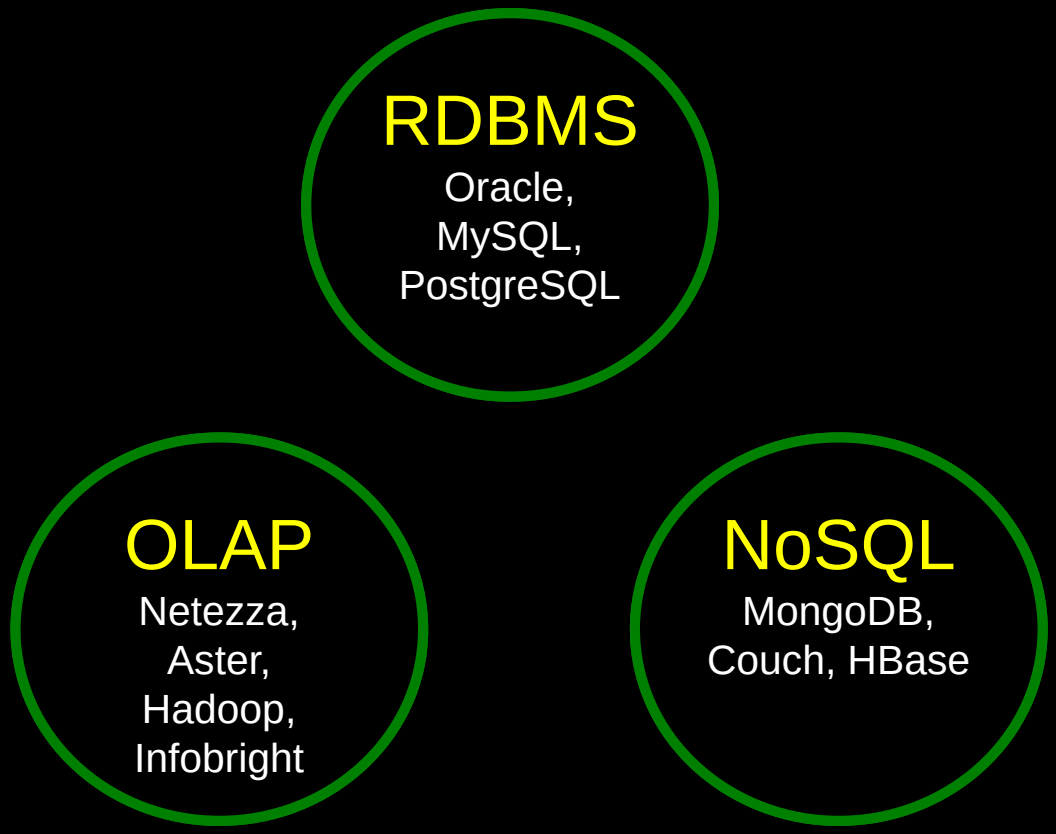
Productivity goes down

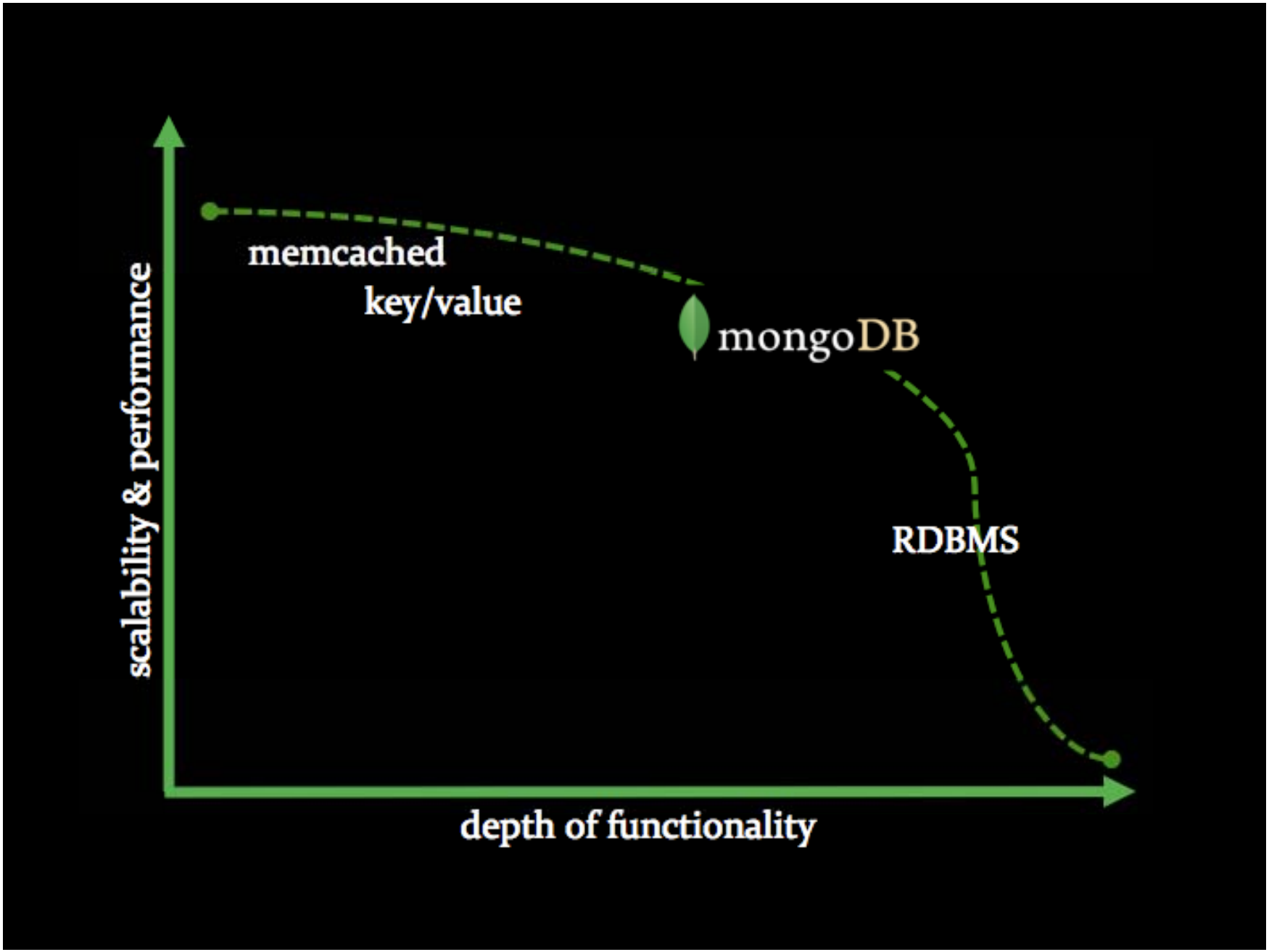


Summary

- We use relational databases because they make our developers productive
- But relational databases don’t scale up well
- Our hardware costs go up exponentially while revenue growth is linear
- And we start taking away those features that made our developers productive

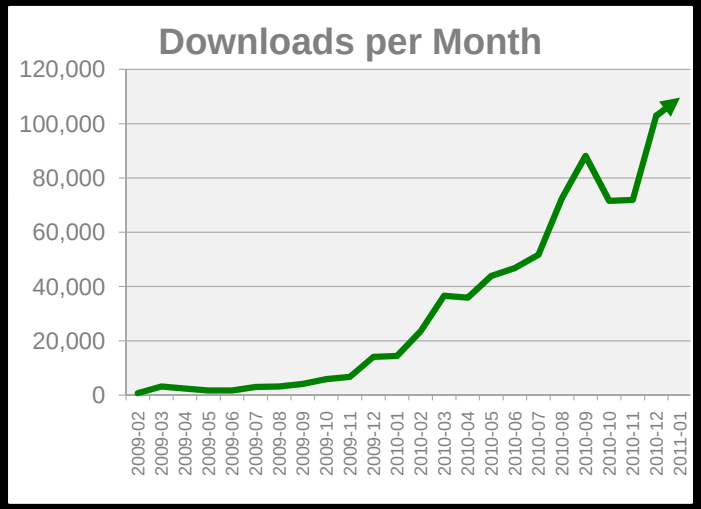
Database Evolution



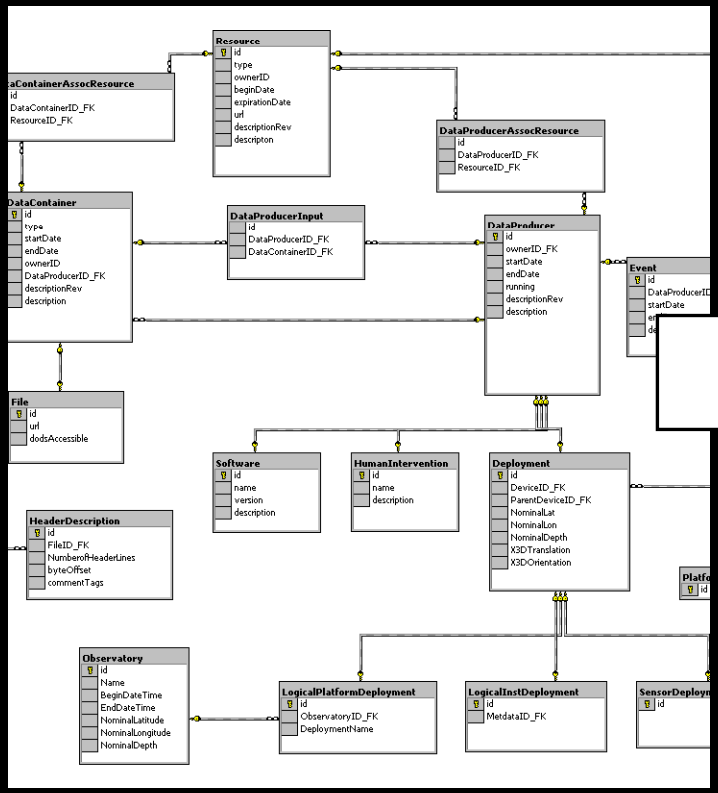

scalability & performance
memcached
key/value
mongoDB
RDBMS
depth of functionality

MongoDB

- **Easy to start**
  - Open source
  - Drivers in every major language
- **Easy to develop**
  - Schemaless document model
  - Flexible query language
  - Secondary indexes
- **Easy to scale**
  - Built in sharding
  - Asynchronous replication



Tables to Documents




```
{
  title: 'MongoDB',
  contributors: [
    { name: 'Eliot Horowitz',
      email: 'eliot@10gen.com' },
    { name: 'Dwight Merriman',
      email: 'dwight@10gen.com'
}
  ],
  model: {
    relational: false,
    awesome: true
  }
}
```

DATA AS DOCUMENTS

Terminology

RDBMS	Mongo
Table, View	Collection
Row(s)	JSON Document
Index	Index
Join	Embedded Document
Partition	Shard
Partition Key	Shard Key

Documents

Blog Post Document

```
p = { author: “roger”,
      date: new Date(),
      text: “Spirited Away”,
      tags: [“Tezuka”, “Manga”]}

> db.posts.save(p)
```

Querying

```
>db.posts.find()

  { _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
    author : "roger",
    date : "Sat Jul 24 2010 19:47:11 GMT-0700
(PDT)",
    text : "Spirited Away",
    tags : [ "Tezuka", "Manga" ] }
```

Notes:

- _id is unique, but can be anything you'd like

Secondary Indexes

Create index on any Field in Document

```
// 1 means ascending, -1 means descending

>db.posts.ensureIndex({author: 1})

>db.posts.find({author: 'roger'})

{ _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
  author : "roger",
  ... }
```

Query Operators

- Conditional Operators
  - –$all, $exists, $mod, $ne, $in, $nin, $nor, $or, $size, $type
  - –$lt, $lte, $gt, $gte

```
// find posts with any tags
> db.posts.find( {tags: {$exists: true }} )

// find posts matching a regular expression
> db.posts.find( {author: /^rog*/i } )

// count posts by author
> db.posts.find( {author: ‘roger’} ).count()
```

Atomic Operations

- $set, $unset, $inc, $push, $pushAll, $pull, $pullAll, $bit

```
> comment = { author: “fred”,
                  date: new Date(),
                  text: “Best Movie Ever”}

 > db.posts.update( { _id: “...” },
                          {$push: {comments: comment}} );
```

Nested Documents

```
{  _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
   author : "roger",
   date : "Sat Jul 24 2010 19:47:11 GMT-0700
(PDT)",
   text : "Spirited Away",
   tags : [ "Tezuka", "Manga" ],
   comments : [
     {
         author : "Fred",
         date : "Sat Jul 24 2010 20:51:03 GMT-0700
(PDT)",
         text : "Best Movie Ever"
```

Map Reduce

```
// count tag occurance
> map = function() {
            this.tags.forEach( function(tag) {
                  emit( tag, {count:1} )
            });
      };
> reduce = function( key, values ) {
            var total = 0;
            values.forEach( function(value) {
                  total += value.count;
            });
            return { count: total };
      };
> out = db.posts.mapReduce(m,r);
```

Map Reduce

```
> db[out.result].find()
{ _id: ‘Tezuka’, value : { count: 3 } }
{ _id: ‘Manga’, value : { count: 4 } }
```

Indexes

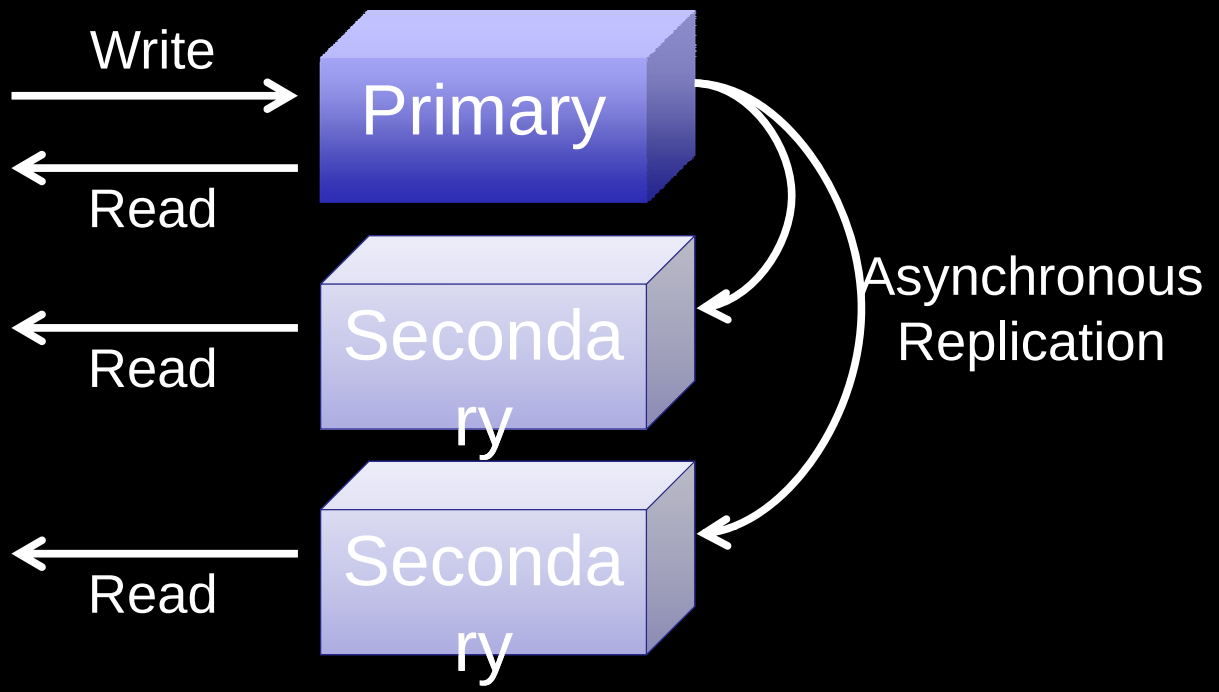
// Index nested documents
> db.posts.ensureIndex(“comments.author”:1)
➢db.posts.find({‘comments.author’:’Fred’})

// Index on tags
> db.posts.ensureIndex(tags: 1)
> db.posts.find({ tags: ’Manga’ })

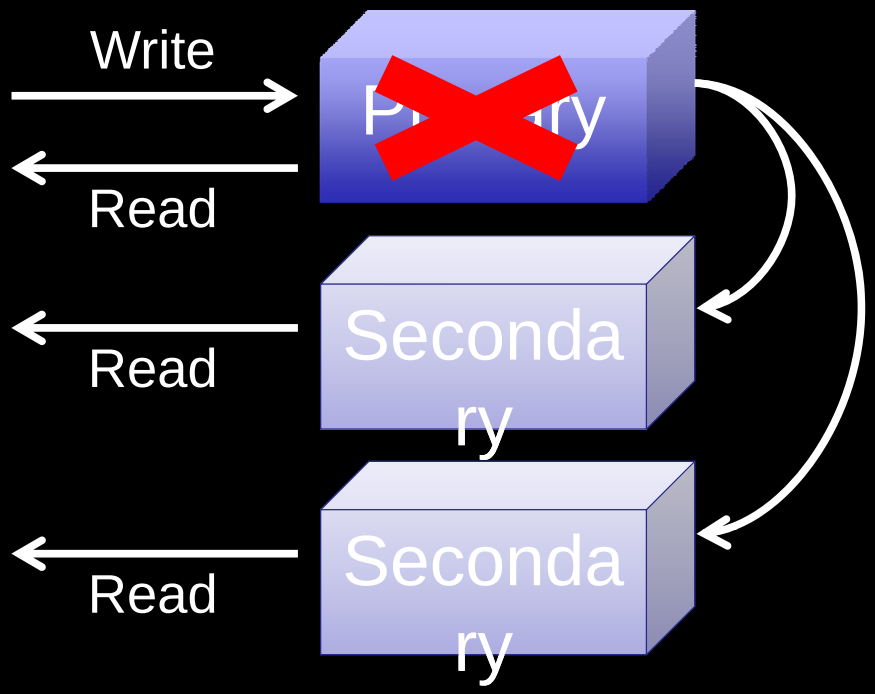
// geospatial index
> db.posts.ensureIndex(“author.location”: “2d”)
> db.posts.find(“author.location” : { $near : [22,42] })

DEPLOYMENT & SCALING

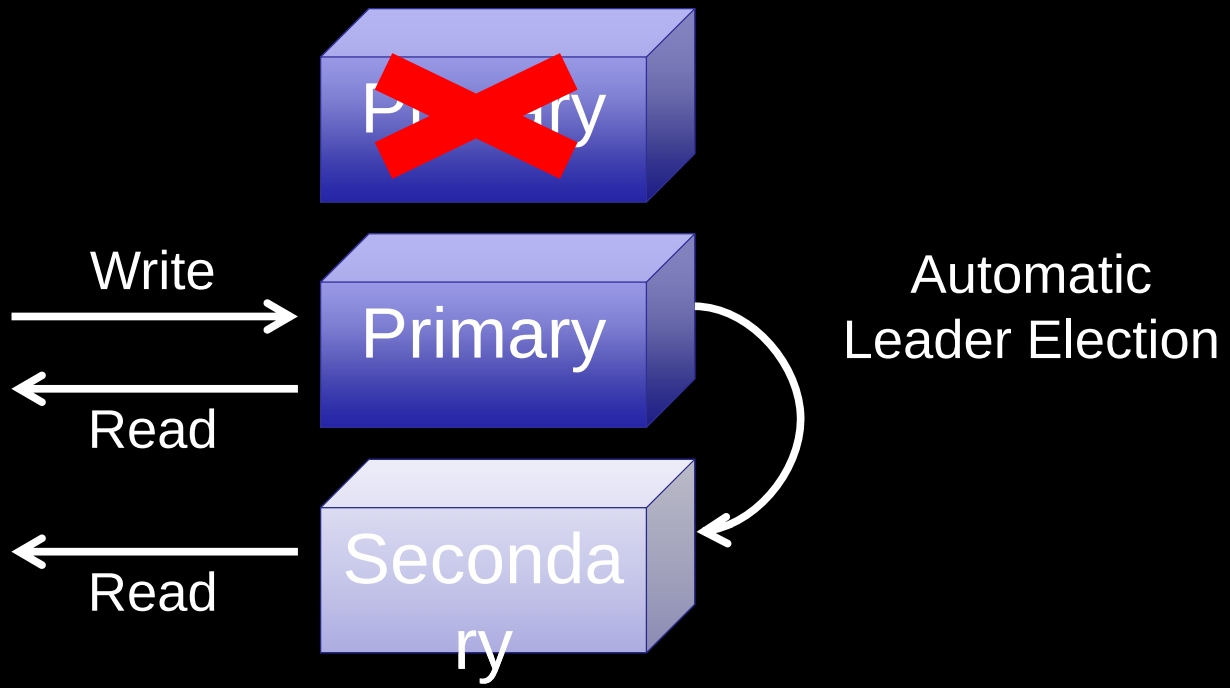
Replica Sets



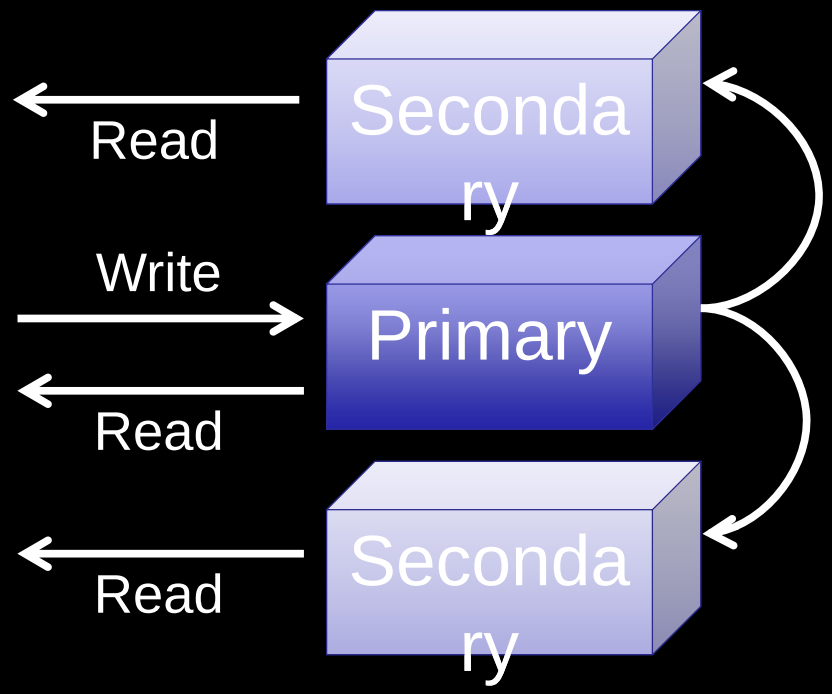
Replica Sets



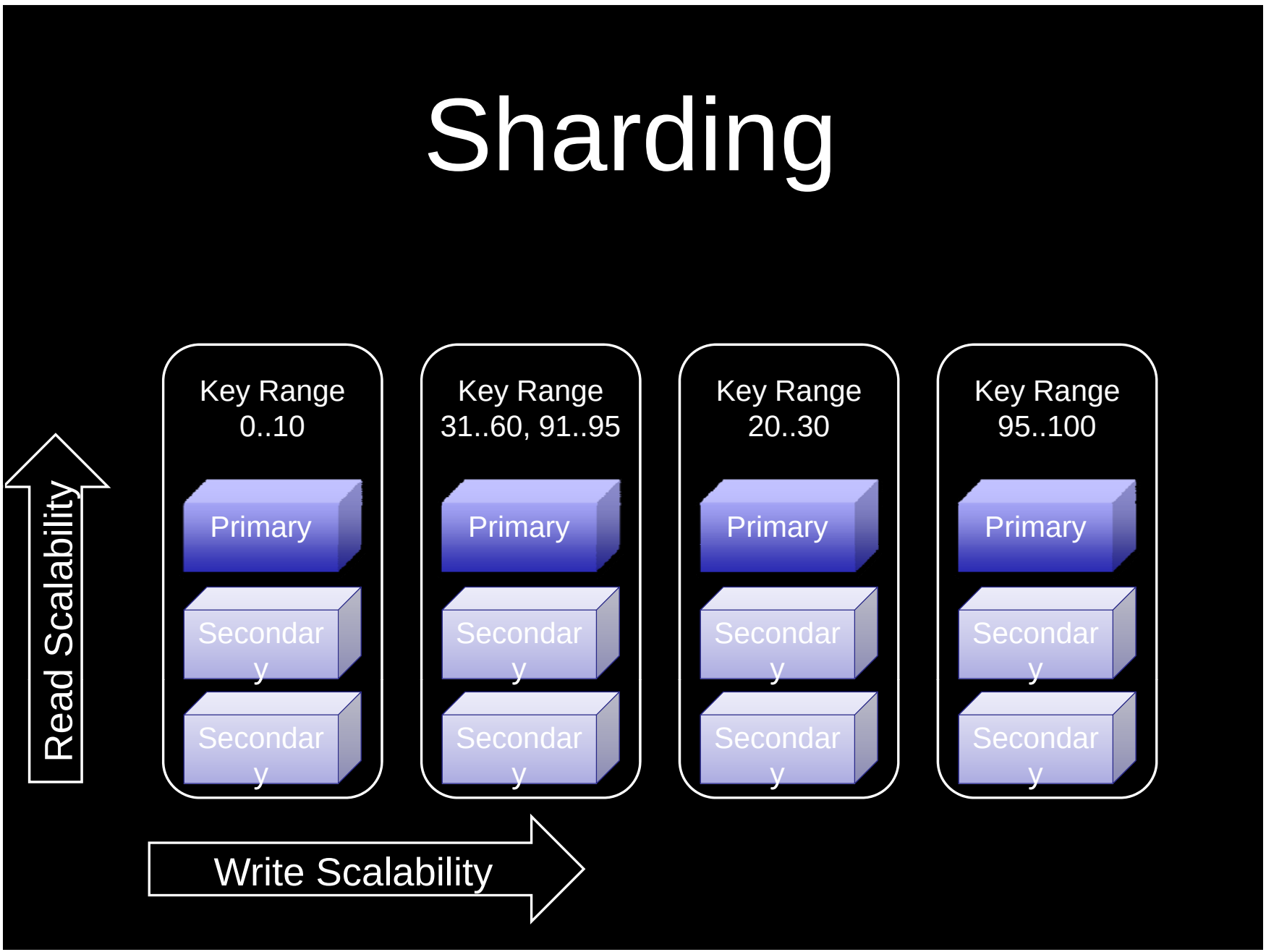
Replica Sets



Replica Sets



Sharding


Key Range
0..10
Key Range
31..60, 91..95
Key Range
20..30
Key Range
95..100
Primary
Secondary
Secondary
Primary
Secondary
Secondary
Primary
Secondary
Secondary
Primary
Secondary
Secondary
Read Scalability
Write Scalability

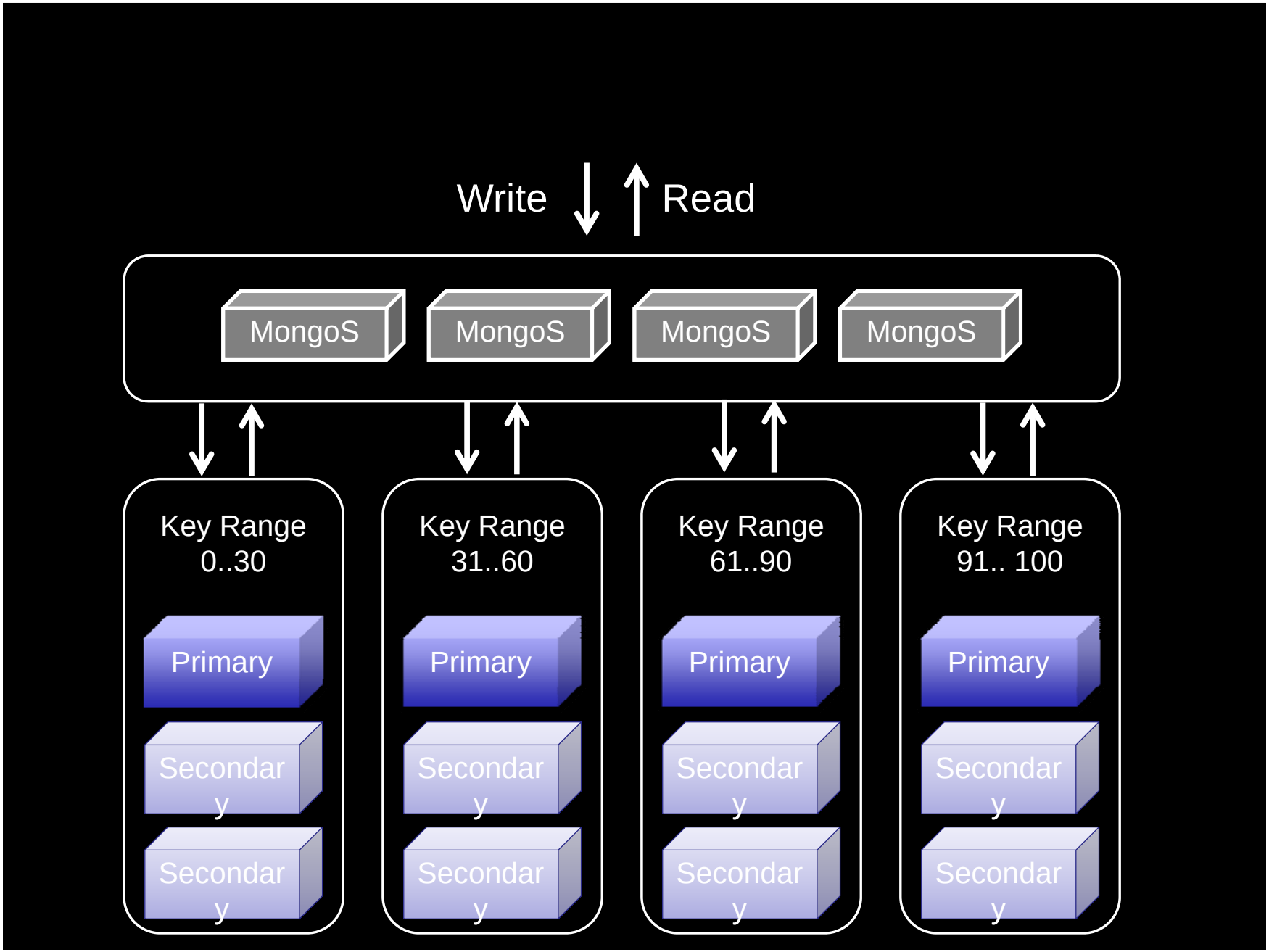

Write
Read
MongoS
MongoS
MongoS
MongoS
Key Range
0..30
Key Range
31..60
Key Range
61..90
Key Range
91.. 100
Primary
Primary
Primary
Primary
Secondar
y
Secondar
y
Secondar
y
Secondar
y
Secondar
y
Secondar
y
Secondar
y
Secondar
y

Photo Meta-Data

Problem:

- Business needed more flexibility than Oracle could deliver

Solution:

- Used MongoDB instead of Oracle

Results:

- Developed application in one sprint cycle
- 500% cost reduction compared to Oracle
- 900% performance improvement compared to Oracle

Customer Analytics

Problem:

- Deal with massive data volume across all customer sites

Solution:

- Used MongoDB to replace Google Analytics / Omniture options

Results:

- Less than1week to build prototype and prove business case
- Rapid deployment of new features (1/day, 1/week)

Online Dictionary

Problem:

- MySQL could not scale to handle their 5B+ documents

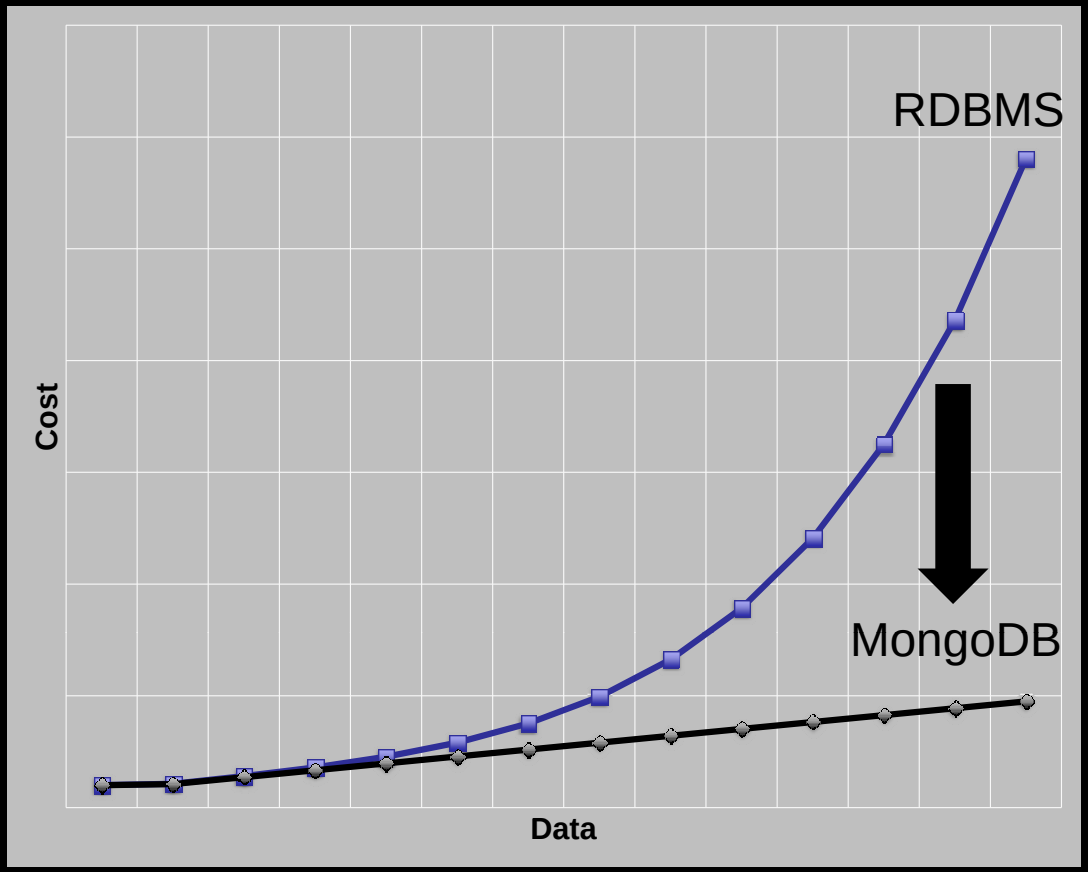
Solution:

- Switched from MySQL to MongoDB

Results:

- Massive simplification of code base
- Eliminated need for external caching system
- 20x performance improvement over MySQL

Summary



mongoDB

- Easy to start
- Easy to develop
- Easy to scale