mongoDB

Building your first app

Ross Lawley – ross@10gen.com

twitter: @RossC0

10gen | the MongoDB company

Hello

I'm Ross Lawley

Work for 10gen

Help maintain pymongo

Maintain MongoEngine

I love opensource and agile methodologies

twitter: RossC0

http://github.com/rozza

A talk of two halves

http://www.flickr.com/photos/53366513@N00/4312672217

Origins of MongoDB

Before 10gen

Dwight Merriman and Eliot Horowitz

Double Click & Shopwiki

-30 **billion** ads a day

-Built multiple database caching layers

Scaling RDMS kills productivity

Project start

Denormalize

Stop using joins

Custom caching layer

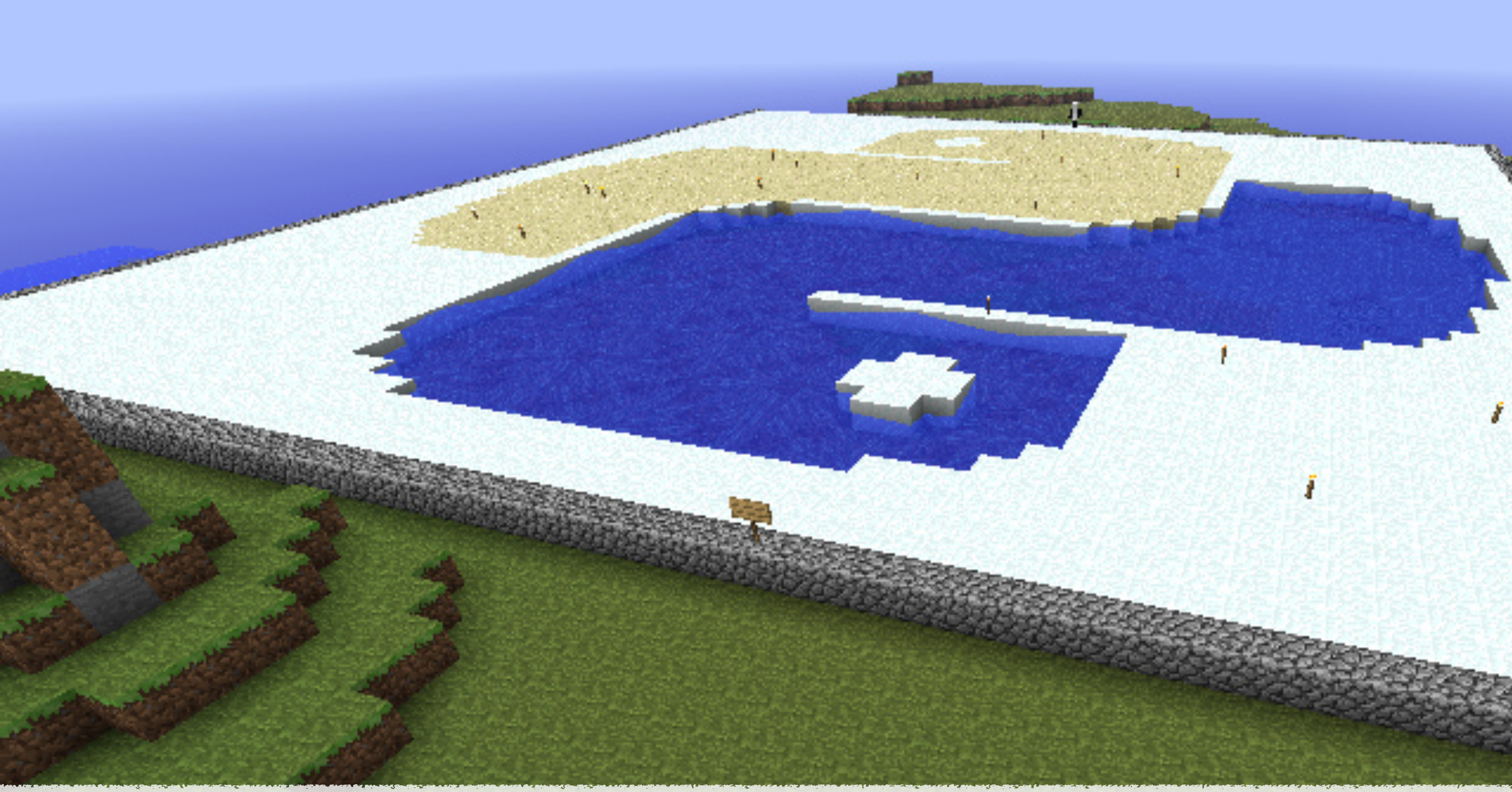
Custom sharding

2007 10gen formed

Originally to create a PAAS service

MongoDB is only three years old

0.8 February 2009 First standalone release
1.0 August 2009 Simple, but used in production
1.2 December 2009 map/reduce, external sort index building
1.4 March 2010 Background indexing, geo
1.6 August 2010 Sharding, replica sets
1.8 March 2011 Journalling, sparse/covered indexes
2.0 September 2012 Compact, concurrency
2.2 July 2012 Concurrency, aggregation framework



MongoDB and Python

python support

Project mongo-python-driver

Configuration Matrix			single_server	master_slave	replica_set
with-extensions	legacy-release	2.4			
		2.7			
		3.2			
		jython			
		pypy			
	stable-release	2.4			
		2.7			
		3.2			
		jython			
		pypy			
	unstable-release	2.4			
		2.7			
		3.2			
		jython			
		pypy			
without-extensions	legacy-release	2.4			
		2.7			
		3.2			
		jython			
		pypy			

A Document database

```
{   _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
    author : "Ross",
    date : ISODate("2012-07-05T10:00:00.000Z"),
    text : "About MongoDB...",
    tags : [ "tech", "databases" ],
    comments : [{
        author : "Tim",
        date : ISODate("2012-07-05T11:35:00.000Z"),
        text : "Best Post Ever!"
    }],
    comment_count : 1
}
```

In Python

```
{   '_id' : ObjectId("4c4ba5c0672c685e5e8aabf3"),
    'author' : "Ross",
    'date' : datetime.datetime(2012, 7, 5, 10, 0),
    'text' : "About MongoDB...",
    'tags' : [ "tech", "databases" ],
    'comments' : [{
            'author' : "Tim",
            'date' : datetime.datetime(2012, 7, 5, 11, 35),
            'text' : "Best Post Ever!"
    }],
    'comment_count' : 1
}
```

Getting Started

```
// Create a connection
import pymongo
conn = pymongo.Connection('mongodb://localhost:27017')

// Connect to a database
db = conn.tutorial
```

```
// Or via a dictionary lookup
db = conn['tutorial']

// Files for the db don't exist until you add data
```

Adding data

```
// Add some data
db.my_collection.save({"Some": "data"})

// Insert - better, explicit
db.my_collection.insert({"Hello": "Florence!"})

// Find data
db.my_collection.find()
<pymongo.cursor.Cursor at 0x25df850>

// Return first that matches
db.my_collection.find_one()
{'_id': ObjectId('4ff4a5b0bb69331891000000'),
 'Hello': 'Florence!'}
```

BSON

BSON { 01010100 11101011 10101110 01010101 }

BSON *[bee · sahn]*, short for Binary JSON, is a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays. BSON also contains extensions that allow representation of data types that are not part of the JSON spec. For example, BSON has a Date type and a BinData type.

BSON can be compared to binary interchange formats, like Protocol Buffers. BSON is more "schema-less" than Protocol Buffers, which can give it an advantage in flexibility but also a slight disadvantage in space efficiency (BSON has overhead for field names within the serialized data).

BSON was designed to have the following three characteristics:

1. **Lightweight**

   Keeping spatial overhead to a minimum is important for any data representation format, especially when used over the network.

2. **Traversable**

   BSON is designed to be traversed easily. This is a vital property in its role as the primary data representation for MongoDB.

3. **Efficient**

   Encoding data to BSON and decoding from BSON can be performed very quickly in most languages due to the use of C data types.

specification | implementations | FAQ | discussion

http://bsonspec.org

Finding data

```
// Query by example - pass in a dict
db.my_collection.find({"score": 60})

// Operators $gt, $gte, $lt, $lte, $ne, $nin,
// $regex, $exists, $not, $or..
db.my_collection.find({"score": {"$gte": 60,
                                 "$lte": 70})

// Sorting (1 ascending, -1 descending)
db.my_collection.find().sort({"name": 1})

// Paginating
db.my_collection.find().skip(5).limit(5)
```

Updating data

```
// Updating - beware! Replaces the document
db.my_collection.update({"_id": 123},{"score": 80})

// Use atomic updates.
db.my_collection.update({}, {"$set": {"score": 80}})

// Multi flag to update more than one
db.my_collection.update({}, {"$set": {"x":"y"},
                             multi=True)

// Upserts
db.my_collection.update({"_id": 123},{"score": 80},
                             upsert=True)
```

Indexes

```
// Single field indexes
db.scores.ensure_index('score')

// Compound indexes
db.scores.ensure_index([
    ('score', pymongo.ASCENDING),
    ('name', pymongo.DESCENDING)]
)

// Geo indexes
db.places.create_index([("loc", GEO2D)])
```

Query plan

```
db.scores.find().explain()

{u'cursor': u'BasicCursor',
 u'indexBounds': {},
 u'indexOnly': False,
 u'isMultiKey': False,
 u'millis': 1,
 u'n': 3000,
 u'nChunkSkips': 0,
 u'nYields': 0,
 u'nscanned': 3000,
 u'nscannedObjects': 3000,
 u'scanAndOrder': False,
 u'server': u'lucid64:27017'}
```

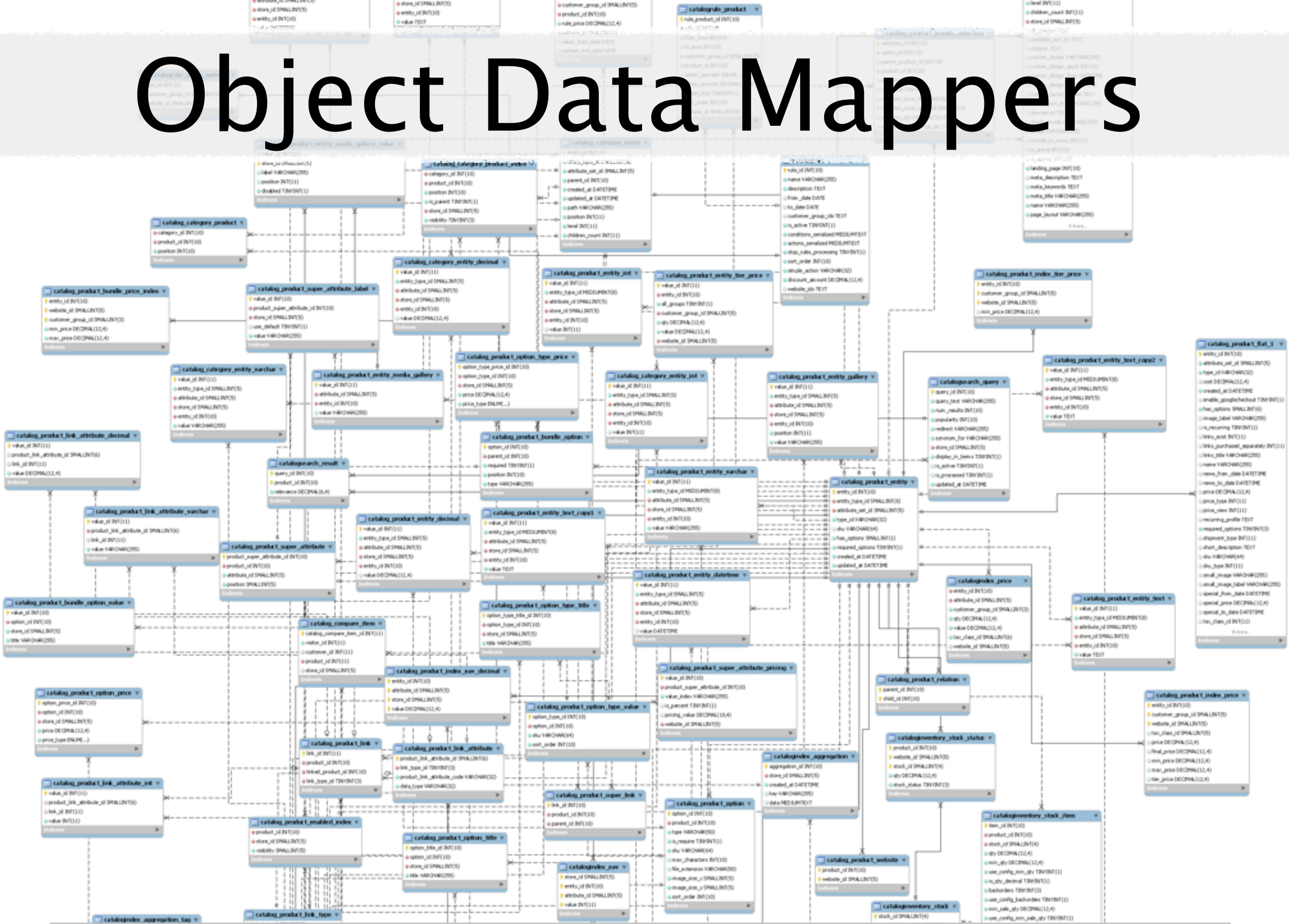
Gridfs

```
// Store files in mongoDB
import gridfs
fs = gridfs.GridFS(db)

// Save file to mongo
my_image = open('my_image.jpg', 'r')
file_id = fs.put(my_image)

// Read file
fs.get(file_id).read()
```

Object Data Mappers



http://www.magento-exchange.com/magento-database/magento-1-4-database-er-diagram-for-catalog-and-product-tables/

Why?

Documents schema in code

Data validation

Enforce schema when required

Can DRY up code..

Lots of options

Humongolus - pythonic and lightweight ORM

MongoKit - ORM-like layer on top of PyMongo

Ming - Developed by SourceForge

MongoAlchemy - Inspired by SQLAlchemy

MongoEngine - Inspired by the Django ORM

Minimongo - lightweight, pythonic interface

Learn by doing

Pymongo Tutorial

http://api.mongodb.org/python/current/tutorial.html

Europython Workshop:

http://github.com/rozza/demos

Tutorial

http://docs.mongodb.org/manual/tutorial/write-a-tumblelog-application-with-flask-mongoengine/

Contents

Write a Tumblelog Application with Flask and MongoEngine

Introduction

This tutorial describes the process for creating a basic tumblelog application using the popular Flask Python web-framework in conjunction with the *MongoDB* database.

The tumblelog will consist of two parts:

1. A public site that lets people view posts and comment on them.
2. An admin site that lets you add and change posts.

This tutorial assumes that you are already familiar with Flask and have a basic familiarity with MongoDB and have *installed MongoDB*. This tutorial uses MongoEngine as the Object Document Mapper (ODM,) this component may simplify the interaction between Flask and MongoDB.

> **Where to get help:** If you're having trouble going through this tutorial, please post a message to mongodb-user or join the IRC chat in *#mongodb* on irc.freenode.net to chat with other MongoDB users who might be able to help.

Replication

http://www.flickr.com/photos/10335017@N07/4570943043

High availability

Single master system - Primary always consistent

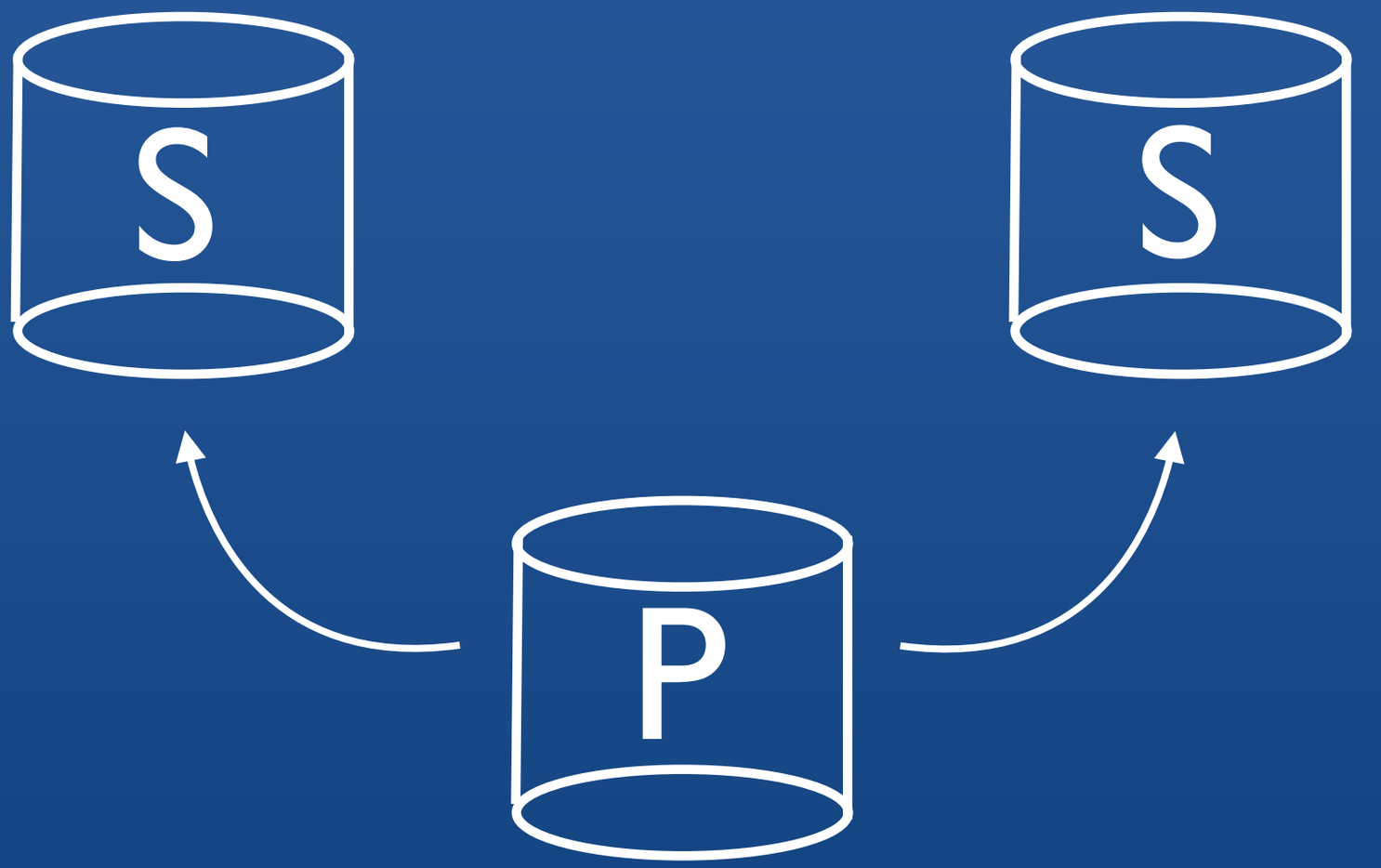
Automatic failover if a Primary fails

Automatic recovery when a node joins the set

Full control over writes using write concerns

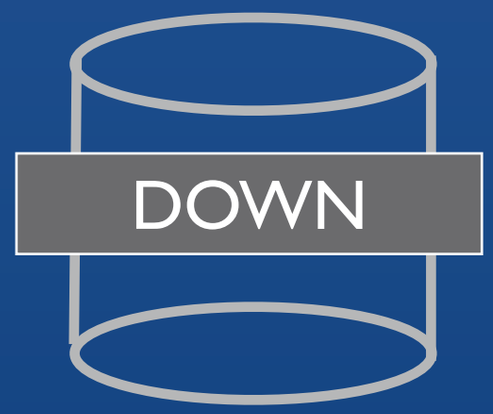
Easy to administer and manage

Replica set is made up of 2 or more nodes



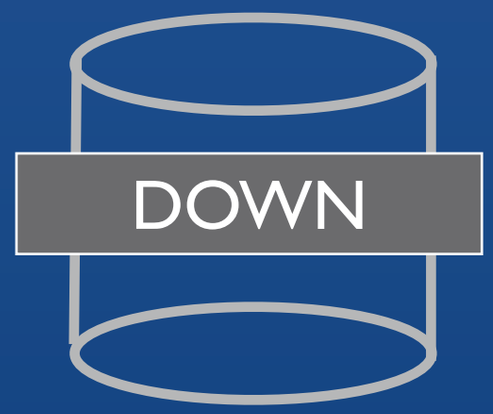
Election establishes the PRIMARY

Data replication from PRIMARY to SECONDARY



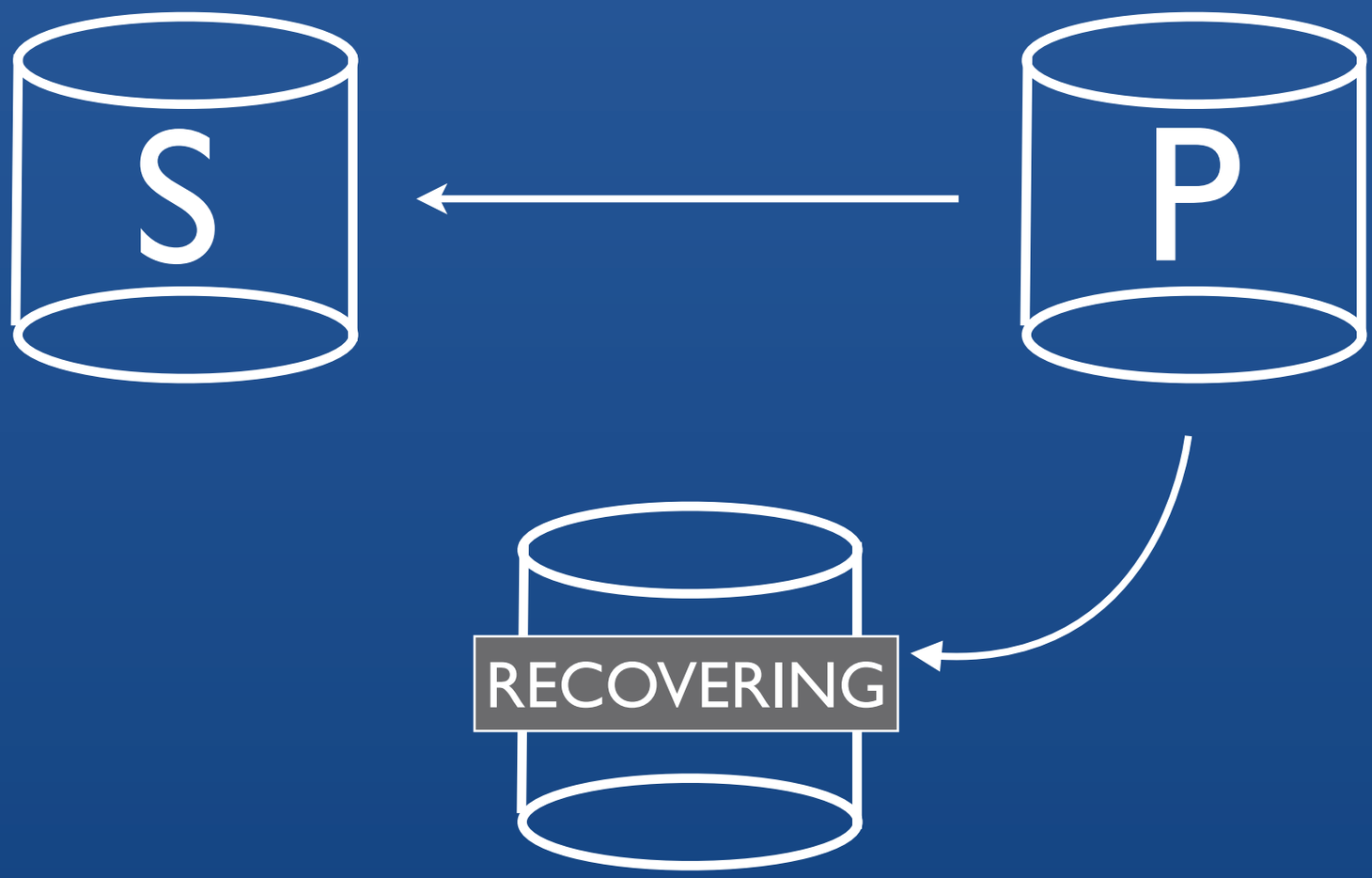
PRIMARY may fail

Automatic election of new PRIMARY if majority exists

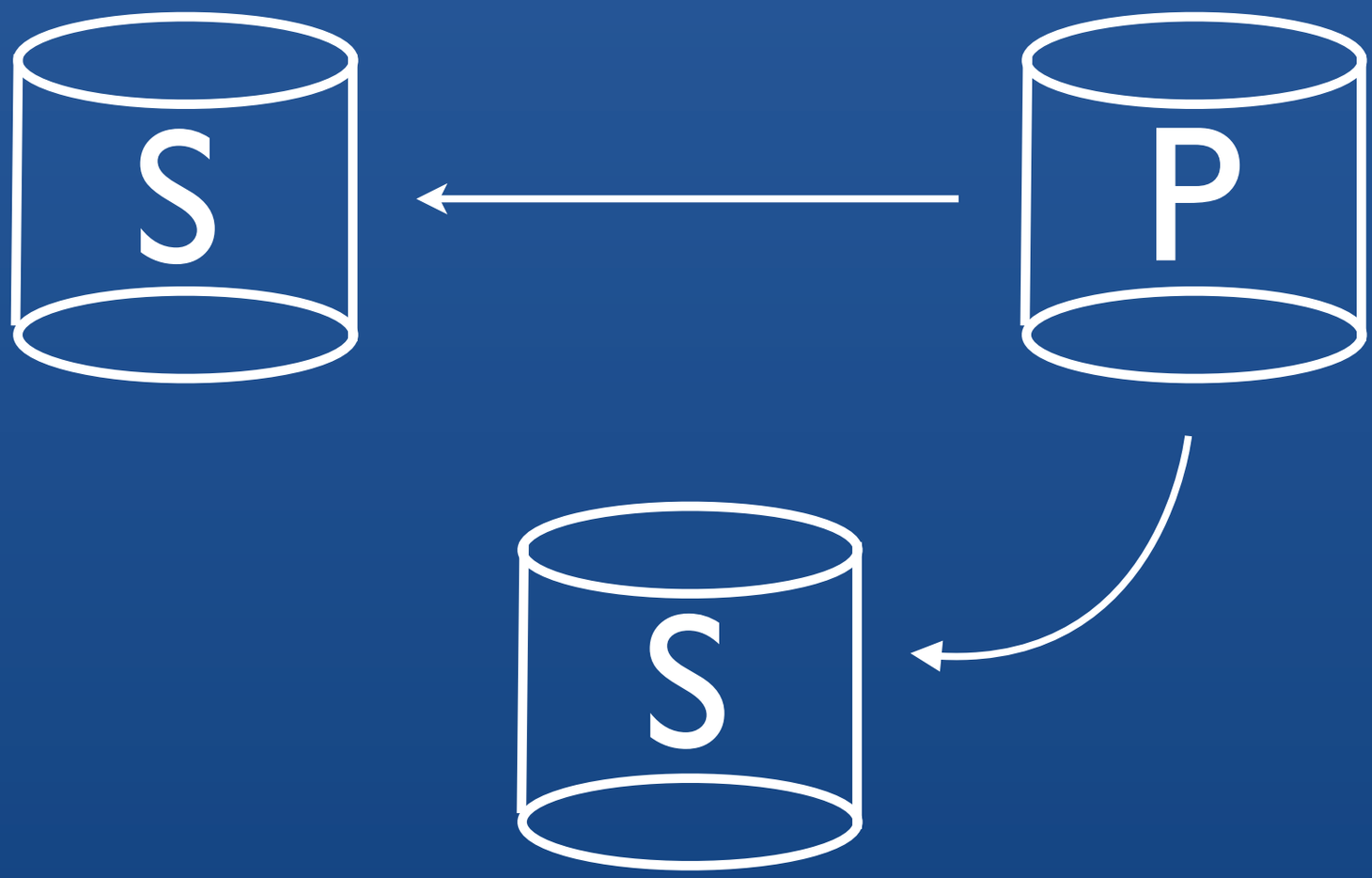


New PRIMARY elected

Replica set re-established



Automatic recovery



Replica set re-established

Advanced features

Durability via write concerns

- On a connection, database, collection and query level
- Tag nodes and direct writes to specific nodes / data centres

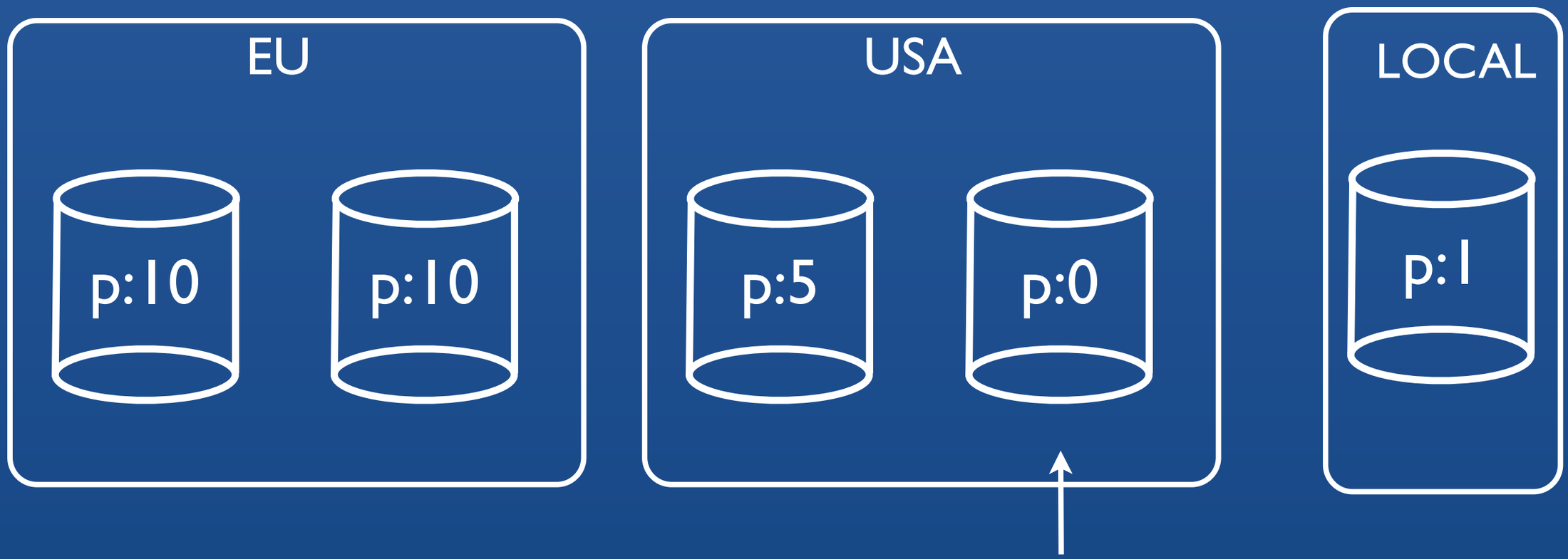
Prioritisation

- Prefer specific nodes to be primary
- Ensure certain nodes are never primary

Scaling reads

- Not applicable for all applications
- Secondaries can be used for backups, analytics, data processing

Example Durable Setup



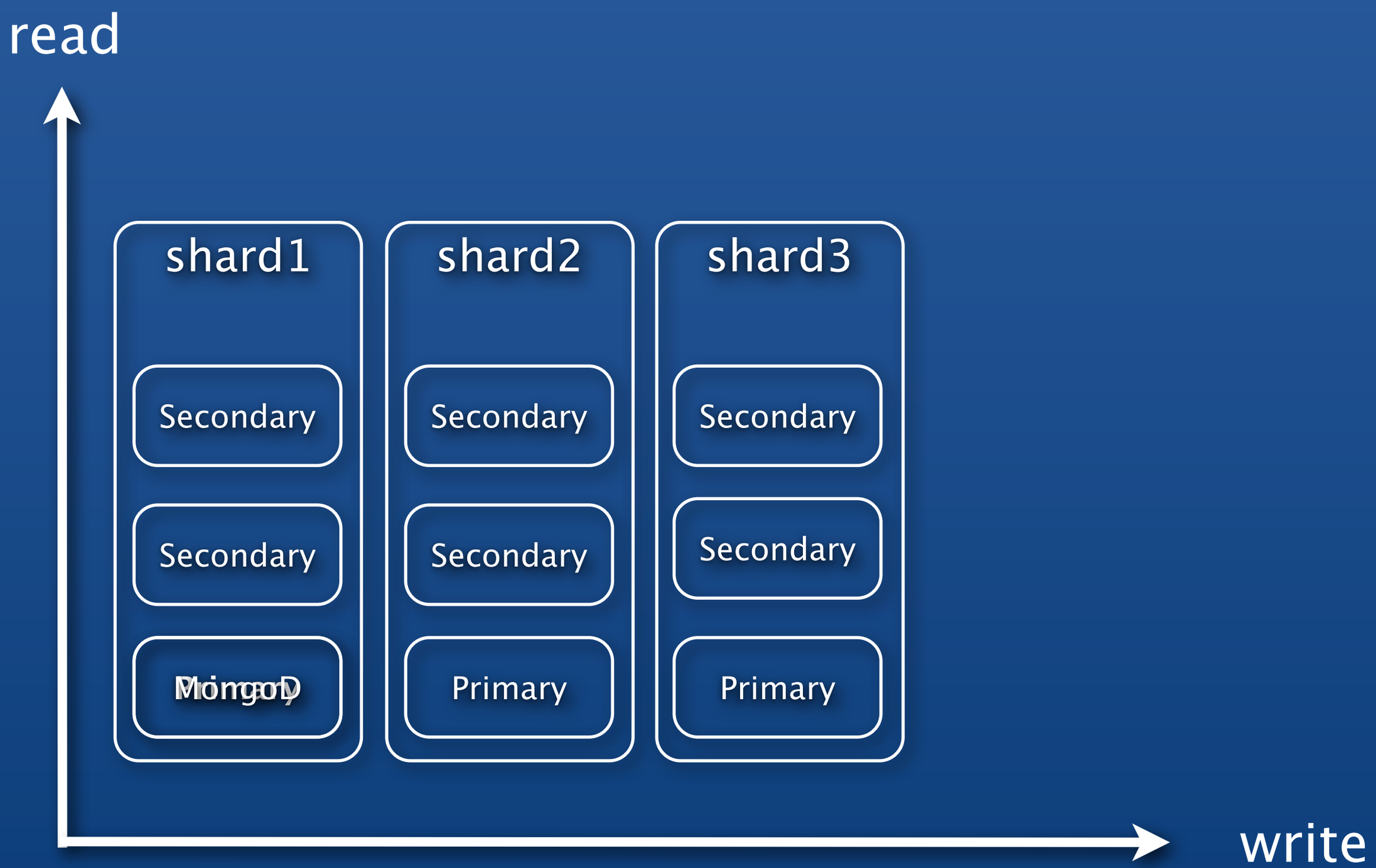
Primary Data Centre

Backups / Analytics Server

Sharding

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

Horizontal scale out



MongoDB Sharding

Automatic partitioning and management

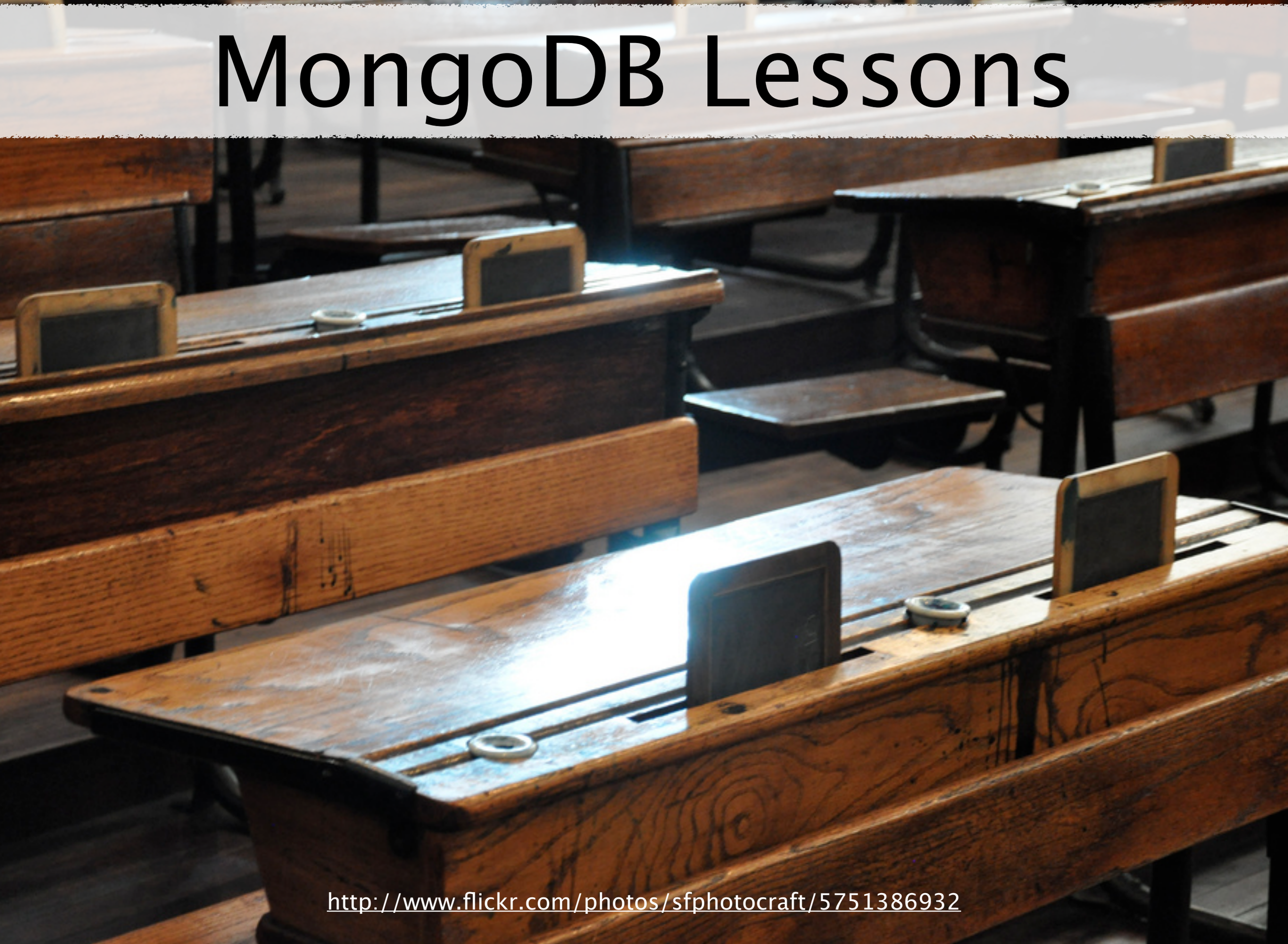
Range based

Convert to sharded system with no downtime

Fully consistent

Durable and Scaled

MongoDB Lessons



http://www.flickr.com/photos/sfphotocraft/5751386932

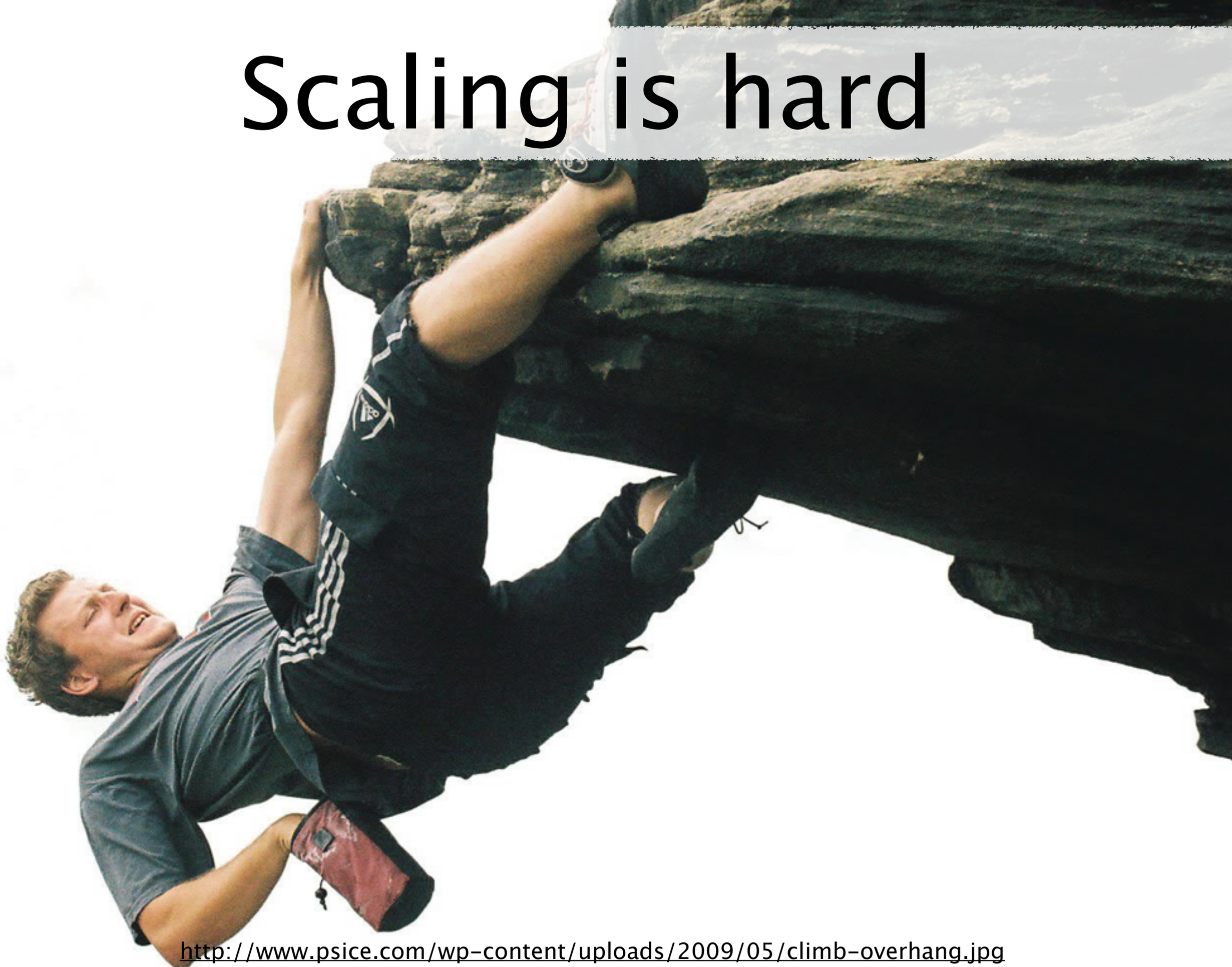
MongoDB is Web scale

http://www.xtranormal.com/watch/6995033/mongo-db-is-web-scale

No magic solution

http://burstownurseries.co.uk

Scaling is hard



http://www.psice.com/wp-content/uploads/2009/05/climb-overhang.jpg

Don't be premature

http://www.sustainabletechnolog.com/2008/12/23/data-center-cooling-efficiency/125

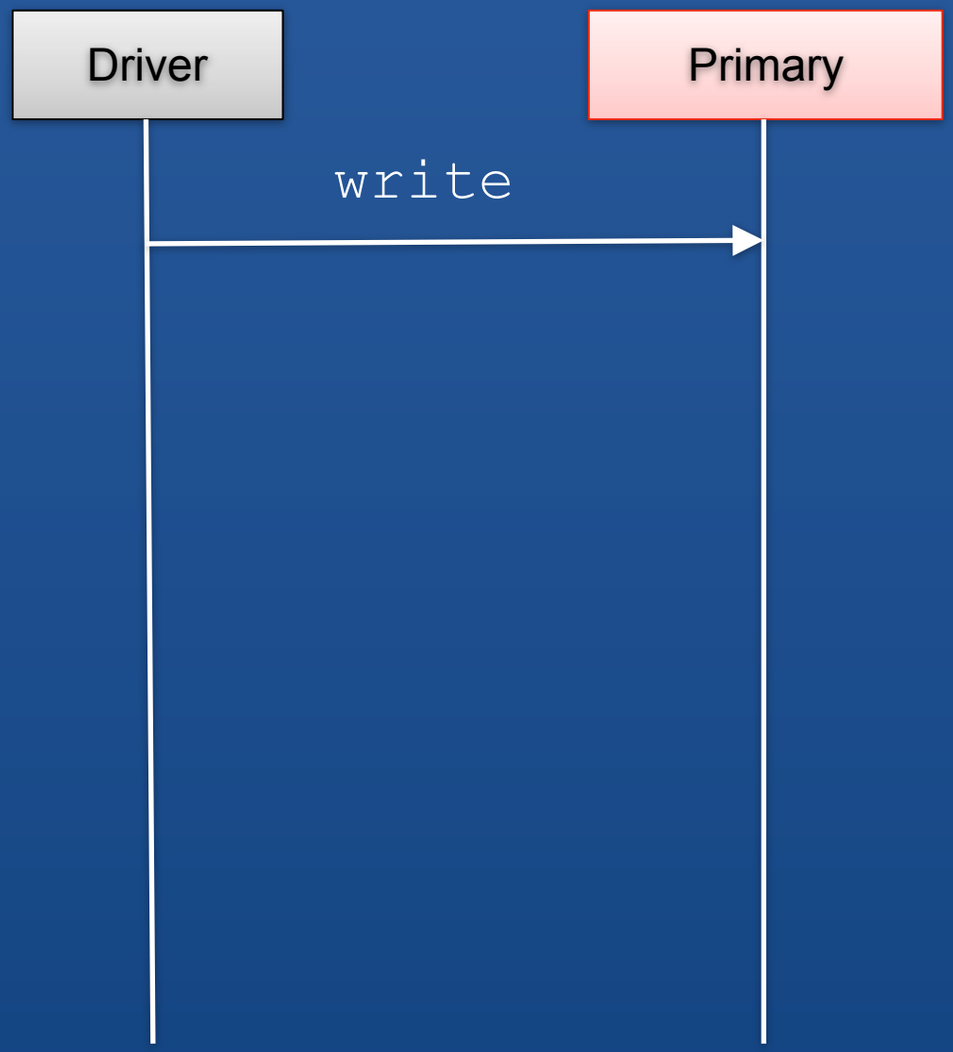
You will lose all your data

http://www.flickr.com/photos/andie-no-uta/5465642092

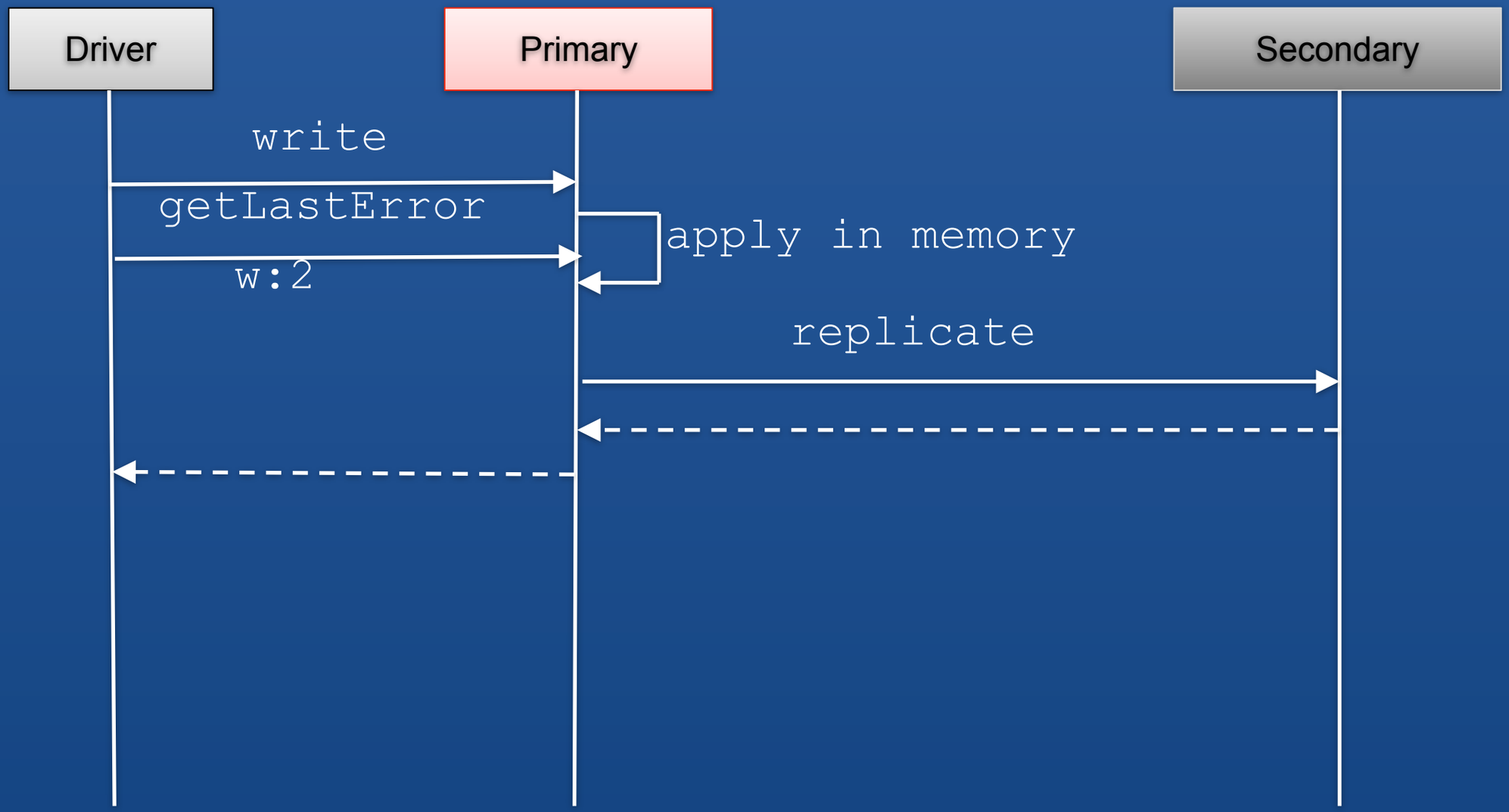
Fire and Forget Writes

http://www.flickr.com/photos/brenduro/5632572311/

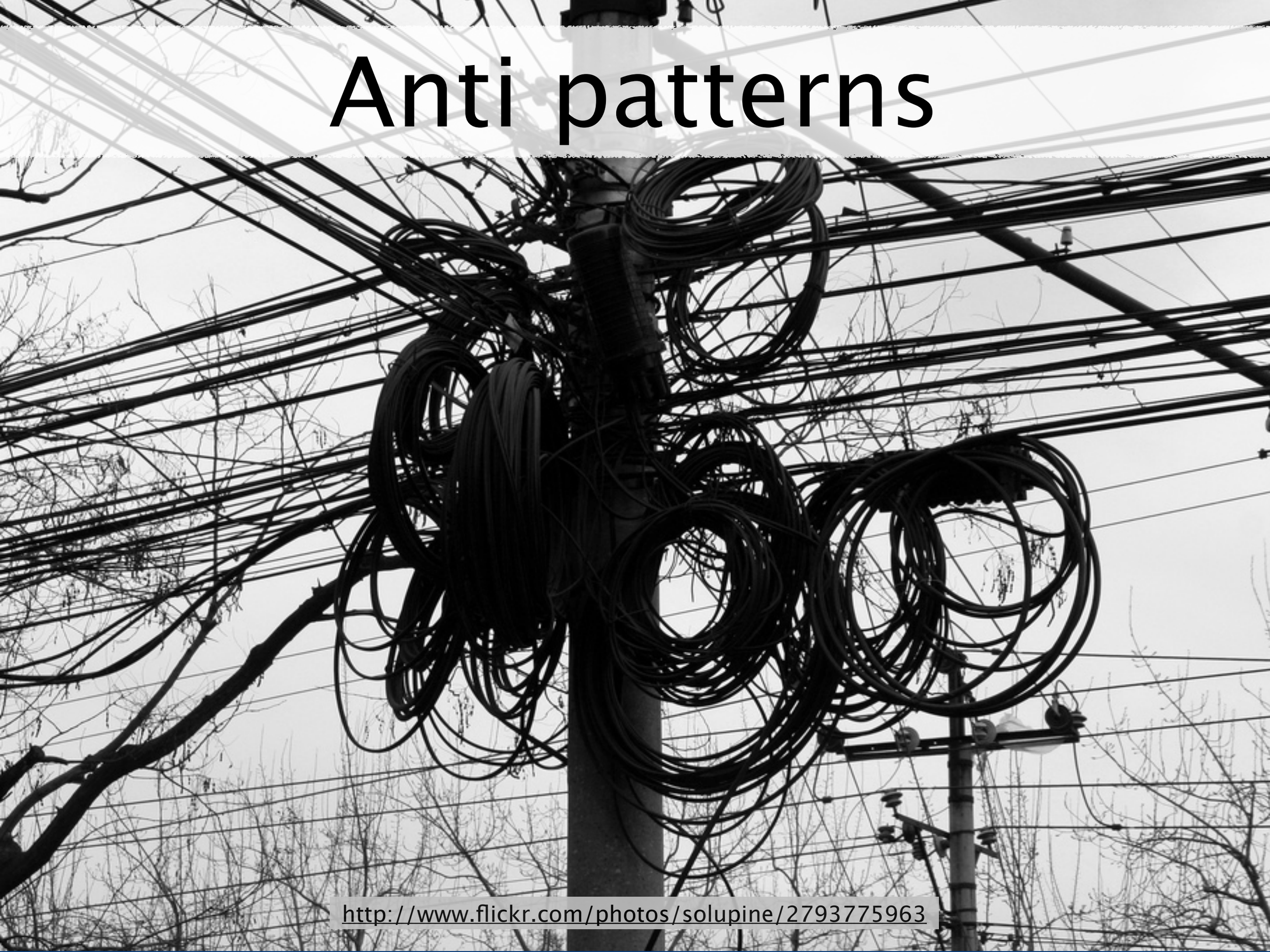
Write concerns



Write concerns



Anti patterns



http://www.flickr.com/photos/solupine/2793775963

Schema-less != Chaos

http://www.flickr.com/photos/redskyy/3246393916

Bad things

One size fits all collections are bad

Unbounded arrays smell and don't perform

Arrays that store all the data

References everywhere

Massive embedded tree structures

Just because you can..

http://www.coolest-gadgets.com/20120513/flying-hovercraft-uber-rich/

Use the right tool

http://www.flickr.com/photos/paultcowan/4644667373

Best practices

http://www.flickr.com/photos/solupine/2793775963

Prove it

Design schema upfront for large scale

Everything scales well with no data

Prove the schema works based on your usecase

Performance test

Questions

http://www.flickr.com/photos/9550033@N04/5020799468