Hands-On MongoDB

ZendCon 2012 - Santa Clara, US - Oct 22nd, 2012

Derick Rethans - derick@10gen.com - twitter: @derickr

http://joind.in/6861

10gen | the MongoDB company

mongoDB

About Me

Derick Rethans

- Dutchman living in London
- One of the PHP MongoDB driver maintainers
- Author of the PHP debugger tool Xdebug, PHP's Date/Time/Timezone support, and a bunch of other PHP extensions
- I ♥ maps

10gen | the MongoDB company

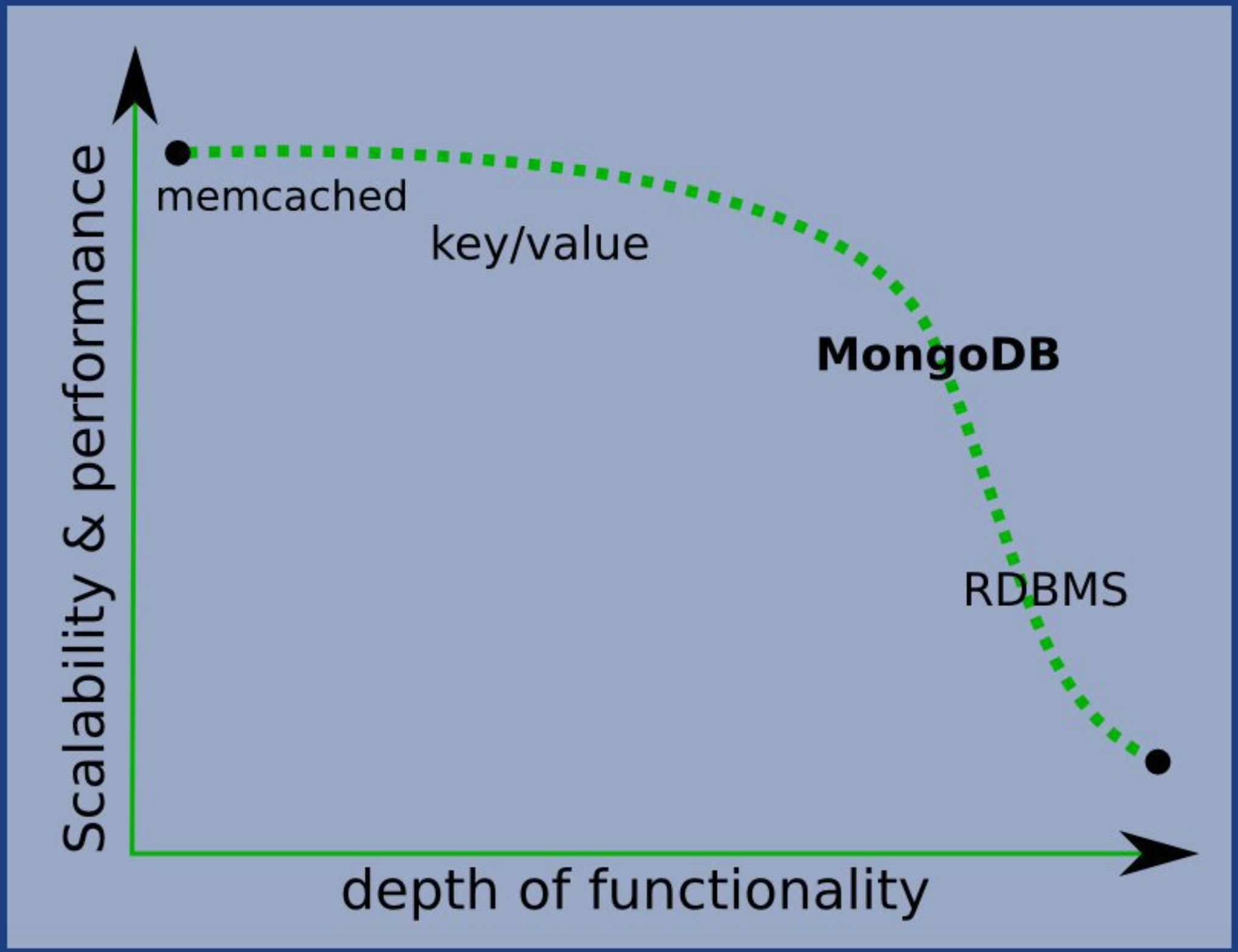
mongoDB

Today's Topics

- Introduction into MongoDB with PHP
- Schema Design
- Indexes
- Practical Example of Schema Design and Indexes
- Replication
- Sharding
- Questions and Answers

Introduction

Database landscape



Terminology

- **Document**: the data (row)
- **Collection**: contains documents (table, view)
- **Index**
- **Embedded Document** (~join)

Documents

- Stored as BSON (Binary JSON)
- Can have embedded documents
- Have a unique ID (the `_id` field)
- Are **schemaless**

Document with embedded documents:

```
{
  "_id" : "derickr",
  "name" : "Derick Rethans",
  "talks" : [
    { "title" : "Profiling PHP Applications",
      "url" : "http://derickrethans.nl/talks/profiling-phptour.pdf",
    },
    { "title" : "Xdebug",
      "url" : "http://derickrethans.nl/talks/xdebug-phpbcn11.pdf",
    }
  ]
}
```

The MongoDB PHP extension

- Maintained and supported by 10gen
- Can be installed with pecl: `pecl install mongo`
- Add `extension=mongo.so` to `php.ini`

10gen | the MongoDB company

Connecting to MongoDB

No databases or collections have to do be explicitly created:

```
<?php
$m = new Mongo();
$database = $m->demo;
$collection = $database->testCollection;
?>
```

Different connection strings:

- `new Mongo("mongodb://localhost");`
- `new Mongo("mongodb://localhost:29000");`
- `new Mongo("mongodb://mongo.example.com");`

Inserting a document

```php
<?php
$document = [
  "_id" => "derickr",
  "name" => "Derick Rethans",
  "talks" => [
    [
      "title" => "Profiling PHP Applications",
      "url" => "http://derickrethans.nl/talks/profiling-phptour.pdf",
    ],
    [
      "title" => "Xdebug",
      "url" => "http://derickrethans.nl/talks/xdebug-phpbcn11.pdf",
    ]
  ]
];

$m = new Mongo();
var_dump( $m->demo->talks->insert( $document ) );
?>
```

Result:

```
boolean true
```

mongoDB supports many types

- null
- boolean
- integer (both 32-bit and 64-bit, MongoInt32, MongoInt64)
- double
- string (UTF-8)
- array
- associative array ("object" for JavaScript)
- MongoId (In the form of `4cb4ab6d7addf98506010000`)
- MongoDate
- MongoBinData

IDs

- Every document needs a unique ID
- ID consists of: 4 bytes timestamp, 3 byte machine id, 2 bytes PID and 3 bytes counter: `507e6384 44670a 3e6e 000000`

```php
<?php
$m = new Mongo();
$c = $m->demo->articles;

$c->insert( [ 'name' => 'Derick', '_id' => 'derickr' ] );

$d = array( 'name' => 'Derick' );
$c->insert( $d );
var_dump( $d );
?>
```

Result:

```
array (size=2)
  'name' => string 'Derick' (length=6)
  '_id' =>
    object(MongoId)[30]
      public '$id' => string '5084512344670afd0b00004a' (length=24)
```

Checking for errors

So far, we have not checked for whether inserts worked.

"Safe" inserts:

```
<?php
$m = new Mongo;
$c = $m->demo->talks;

try {
    $c->insert(
        array( '_id' => 'derickr' ), // document
        array( 'safe' => true )      // options
    );
} catch ( Exception $e ) {
    echo $e->getMessage(), "\n";
}
?>
```

Result:

```
localhost:27017: E11000 duplicate key error index: demo.talks.$_id_  dup key: { : "
```

Checking for errors (2)

Safeness levels:

- Confirm change recorded in memory: `array( 'safe' => true );`
- Confirm inclusion in journal: `array( 'j' => true );`
- Confirm committed to disk: `array( 'fsync' => true );`
- Confirm replication: `array( 'safe' => true, 'w' => 2 );`

```
<?php
$m = new Mongo();
$c = $m->demo->talks;

$c->insert(
    array( '_id' => 'mongodb' ),       // document
    array( 'safe' => true, 'w' => 2 ) // options
);
?>
```

From 1.3:

```
<?php
$m = new Mongo('mongodb://localhost/?safe=true');
?>
```

Querying (1)

```
<?php
$m = new Mongo();
// First "found" item:
$record = $m->demo->talks->findOne();

// Search on the _id field being 'derickr'
$record = $m->demo->talks->findOne( array( '_id' => 'derickr' ) );

// Search on the array element key 'title' in the 'talks' field
$record = $m->demo->talks->findOne(
  array(
    'talks.title' => 'Xdebug'
  )
);
?>
```

Querying (2)

```
<?php
$m = new Mongo();

// Find with 'projection'
$record = $m->demo->talks->findOne(
  array( 'talks.title' => 'Xdebug' ),
  array( 'name' => true, 'talks.url' => true )
);
?>
```

Advanced querying operators

Besides testing for exact matches, mongoDB also has operators. Operators start with a '$', so make sure to use single quotes!

Array:

- $in (value needs to be one of the elements of the given array)
- $nin (not in)
- $all (value needs to match all elements of the given array)
- $size (exact size of array)
- $elemMatch (element in an array matches the specified match expression)

Others:

- $type (check for type number, see http://www.mongodb.org/display/DOCS/Advanced+Queries#AdvancedQueries-%24type)

Querying: matching array elements

```
<?php
$m = new Mongo;
$c = $m->demo->continent;
$r = $c->find( array(
    'cities.population' => array( '$gte' => 11000000 ),
    'cities.dem' => array( '$gte' => 100 ),
) );

foreach( $r as $c ) {
    echo $c['name'], "\n";
    foreach( $c['cities'] as $city ) {
        echo '- ', $city['name'], ': ', $city['population'], ' - ', $city['dem'], "
    }
}
?>
```

Result:

```
South America
- Buenos Aires: 13076300 - 131
- São Paulo: 10021295 - 769
Asia
- Karachi: 11624219 - 38
- Moscow: 10381222 - 144
- İstanbul: 11174257 - 39
```

Querying: matching array elements ($elemMatch)

```
<?php
$m = new Mongo;
$c = $m->demo->continent;
$r = $c->find( array(
    'cities' => array(
        '$elemMatch' => array(
            'population' => array( '$gte' => 11000000 ),
            'dem' => array( '$gte' => 100 ),
        )
    )
) );

foreach( $r as $c ) {
    echo $c['name'], "\n";
    foreach( $c['cities'] as $city ) {
        echo '- ', $city['name'], ': ', $city['population'], ' - ', $city['dem'], "
    }
}
?>
```

Result:

```
South America
- Buenos Aires: 13076300 - 131
- São Paulo: 10021295 - 769
```

Querying and looping

Finding multiple documents is done with the `find()`, and returns an iterator in the form of a `MongoCursor` object.

```
<?php
$m = new Mongo();
$c = $m->demo->talks;

$cursor = $c->find( [ 'talks.title' => [ '$exists' => true ] ] );

foreach ( $cursor as $r )
{
        echo $r['_id'], "\n";
}
?>
```

Result:

```
derickr
```

Cursor methods

```
<?php
$m = new Mongo();
$c = $m->demo->cities;

$cursor = $c->find( array( 'country_code' => 'RU' ) )
            ->sort( array( 'name' => 1 ) )
            ->limit( 5 )->skip( 25 );

foreach ( $cursor as $r ) {
    var_dump( $r['name'] );
}
?>
```

Result:

```
string 'Al'met'yevsk' (length=16)
string 'Amursk' (length=6)
string 'Anapa' (length=5)
string 'Angarsk' (length=7)
string 'Anna' (length=4)
```

Modifying documents

```php
<?php
$m = new Mongo;
$c = $m->demo->presentations;
$c->remove();

$c->insert( array(
  '_id' => 'mongo-tut-zendcon12',
  'name' => 'Hands-On MongoDB'
) );

$c->update(
  array( 'name' => 'Hands-On MongoDB' ), // criteria
  array( '$set' => array( 'lastviewed' => time() ) ) // modifiers
);

/* document is now:
array(
  '_id' => 'mongo-tut-zendcon12',
  'name' => 'Hands-On MongoDB'
  'lastviewed' => 1350296073,
) */
?>
```

Updating documents

Update only updates the **first** document it finds by default.

You can set an option to get all matching documents to be updated

```
<?php
$m = new Mongo;
$c = $m->demo->products;
$c->drop();

$c->insert( [ '_id' => 'blue-elephpant',  'generation' => 1, 'price' => 7.5 ] );
$c->insert( [ '_id' => 'pink-elephpant',  'generation' => 2, 'price' => 8.0 ] );
$c->insert( [ '_id' => 'green-elephpant', 'generation' => 3, 'price' => 7.5 ] );

$c->update(
    [ 'generation' => [ '$lte' => 2 ] ],  // criteria
    [ '$dec' => [ 'price' => 2.0 ] ],     // update spec
    [ 'multiple' => true ]                // options: multiple
);
?>
```

Upserting documents

upsert: if the record(s) do not exist, insert one.

```
<?php
$m = new Mongo;
$c = $m->demo->elephpants; $c->drop();

function birthDay( $c, $name )
{
    $c->update(
        array( 'name' => $name ),                    // criteria
        array( '$inc' => array( 'age' => 1 ) ),      // update spec
        array( 'upsert' => true )                    // options
    );
    echo $c->findOne( array( 'name' => 'Santon' ) )['age'], "\n";
}

birthDay( $c, 'Santon' );
birthDay( $c, 'Santon' );
?>
```

Result:

```
1
2
```

Indexes

```php
<?php ini_set('xdebug.var_display_max_depth', 1);
$m = new Mongo;
$c = $m->demo->elephpants;
$c->drop();

$c->insert( [ '_id' => 'ele1', 'name' => 'Jumbo' ] );
$c->insert( [ '_id' => 'ele2', 'name' => 'Tantor' ] );
$c->insert( [ '_id' => 'ele3', 'name' => 'Stampy' ] );

var_dump( $c->find( [ 'name' => 'Jumbo' ] )->explain() );
?>
```

Result:

```
array (size=15)
  'cursor' => string 'BasicCursor' (length=11)
  'isMultiKey' => boolean false
  'n' => int 1
  'nscannedObjects' => int 3
  'nscanned' => int 3
  'nscannedObjectsAllPlans' => int 3
  'nscannedAllPlans' => int 3
  'scanAndOrder' => boolean false
  'indexOnly' => boolean false
  'nYields' => int 0
  'nChunkSkips' => int 0
  'millis' => int 0
  'indexBounds' =>
    array (size=0)
```

Indexes

```php
<?php ini_set('xdebug.var_display_max_depth', 1);
$m = new Mongo;
$c = $m->demo->elephpants;
$c->drop();

$c->ensureIndex( [ 'name' => 1 ] );

$c->insert( [ '_id' => 'ele1', 'name' => 'Jumbo' ] );
$c->insert( [ '_id' => 'ele2', 'name' => 'Tantor' ] );
$c->insert( [ '_id' => 'ele3', 'name' => 'Stampy' ] );

var_dump( $c->find( [ 'name' => 'Jumbo' ] )->explain() );
?>
```

Result:

```
array (size=15)
  'cursor' => string 'BtreeCursor name_1' (length=18)
  'isMultiKey' => boolean false
  'n' => int 1
  'nscannedObjects' => int 1
  'nscanned' => int 1
  'nscannedObjectsAllPlans' => int 1
  'nscannedAllPlans' => int 1
  'scanAndOrder' => boolean false
  'indexOnly' => boolean false
  'nYields' => int 0
  'nChunkSkips' => int 0
  'millis' => int 0
```

More about indexes

Compound indexes:

```
$myCol->ensureIndex( [ _id => 1, ts => -1 ] )
```

Indexes can be made on nested fields:

```
$myCol->ensureIndex( [ "article.title" => -1 ] )
```

Indexes can be made on array values:

```
$myCol->insert( [ '_id' => 42, 'values' => [ 'fourty', 'two' ] ] );
$myCol->ensureIndex( [ 'values' => 1 ] );
```

Searching with regexp: ^:

```
$myCol->find( [ 'name' => new MongoRegex( '/^tan/i' ) ] )
```

Commands

- Are run on the database
- Do **not** return a cursor
- Return one document
- Some commands have helpers in the drivers and shell

Distinct

```php
<?php
$m = new Mongo; $c = $m->demo->pubs; $c->drop();

$c->insert( [ 'name' => 'Betsy Smith',   'city' => 'London' ] );
$c->insert( [ 'name' => 'London Tavern', 'city' => 'London' ] );
$c->insert( [ 'name' => 'Lammars',       'city' => 'Manchester' ] );
$c->insert( [ 'name' => 'Weatherspoons', 'city' => 'Coventry' ] );

$r = $m->demo->command( [
    'distinct' => 'pubs',
    'key' => 'city',
    'query' => [ 'name' => [ '$ne' => 'Weatherspoons' ] ]
] );
var_dump( $r['values'] );
?>
```

Result:

```
array (size=2)
  0 => string 'London' (length=6)
  1 => string 'Manchester' (length=10)
```

Aggregation Framework

New in 2.2

Powerfull framework for running multiple operations in a pipeline, mostly replacing M/R.

Operations are:

- `$match`: for matching documents, à la `find()`.
- `$project`: for "rewriting" documents, but more powerful with computed expressions: adding fields (`$add`), conditions (`$ifNull`), string functions (`$toLower`), etc
- `$unwind`: hands out array elements each at a time
- `$group`: aggregates items into buckets defined by key
- `$sort`
- `$limit` / `$skip`

Aggregation example

```
<?php
$m = new Mongo();
$c = $m->demo->cities;

$res = $c->aggregate( [
    [ '$match' => [ 'name' => 'Atlanta' ] ],
    [ '$unwind' => '$alternatenames' ],
    [ '$project' => [
        'name' => '$alternatenames',
        'meta' => [
            'elevation' => '$dem',
            'population' => 1,
            'timezone' => 1,
        ],
        'location' => 1,
        '_id' => 0,
    ] ],
    [ '$sort' => [ 'name' => 1 ] ]
] );

var_dump( $res );
?>
```

Result:

```
array (size=37)
  0 =>
    array (size=3)
      'location' =>
```

Any questions?

Schema Design

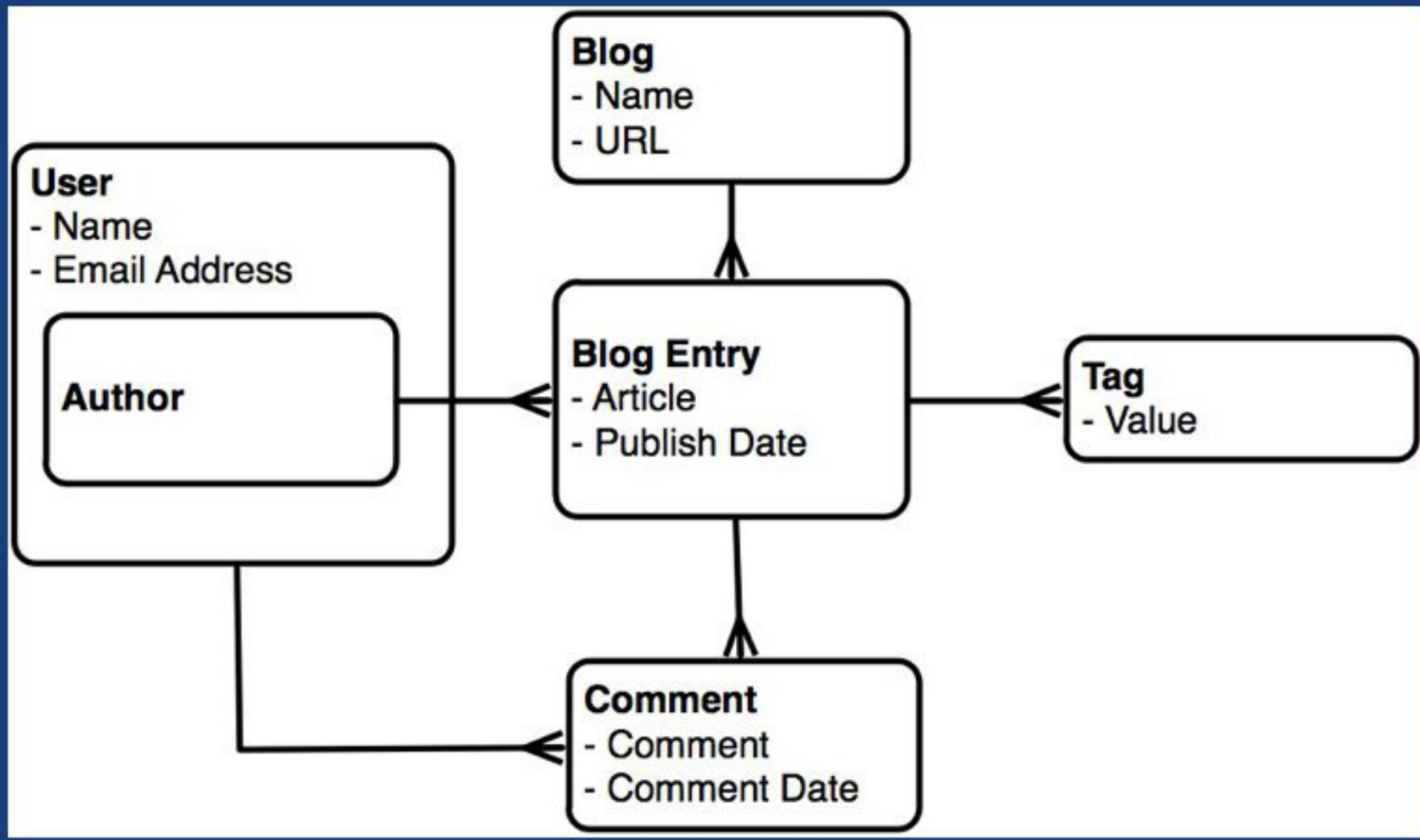
RDBMS: Normalisation

- 1970 E.F.Codd introduces 1st Normal Form (1NF)
- 1971 E.F.Codd introduces 2nd and 3rd Normal Form (2NF, 3NF)
- 1974 Codd & Boyce define Boyce/Codd Normal Form (BCNF)
- 2002 Date, Darween, Lorentzos define 6th Normal Form (6NF)

Goals:

- Avoid anomalies when inserting, updating or deleting
- Minimize redesign when extending the schema
- Make the model informative to users
- Avoid bias towards a particular style of query

Blog in a RDBMS



Blog in MongoDB

User Collection
- Name
- Email Address

Blog Collection
- Name
- URL
- Author

Entry Collection
- Text
- Publish Date

Comments[]
- Comment
- Comment Date
- Comment Author

Tags[]

Schema considerations

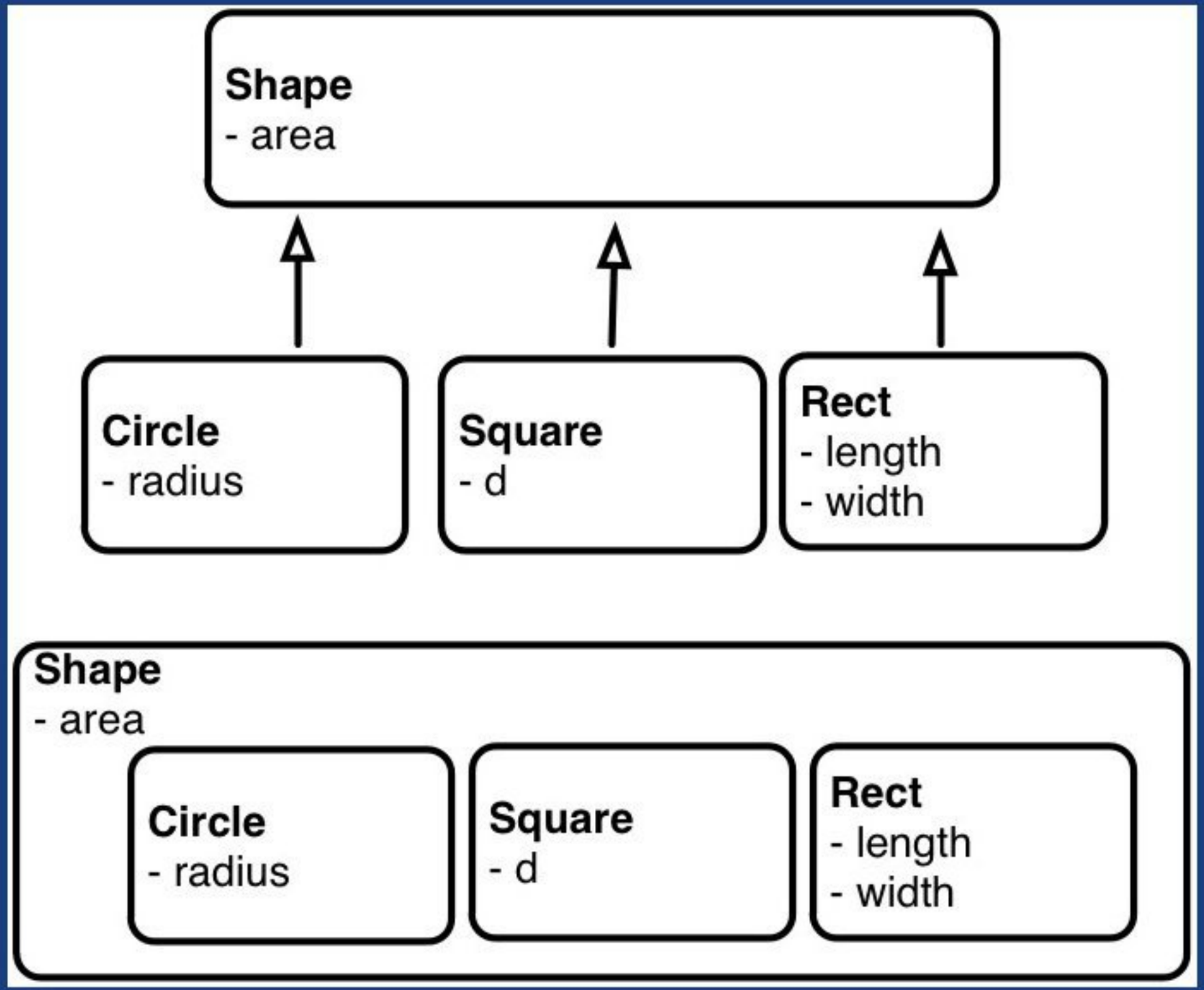
Access Patterns?

- Read / Write Ratio
- Types of updates
- Types of queries
- Data life-cycle

Considerations

- No Joins
- Document writes are atomic

Inheritance



Single table inheritance - RDBMS

id	type	area	radius	d	length	width
1	circle	3.14	1			
2	square	4		2		
3	rectangle	10			5	2

Single table inheritance - MongoDB

```
{ _id: "1", type: "circle", area: 3.14, radius: 1}

{ _id: "2", type: "square", area: 4, d: 2}

{ _id: "3", type: "rectangle", area: 10, length: 5, width: 2}
```

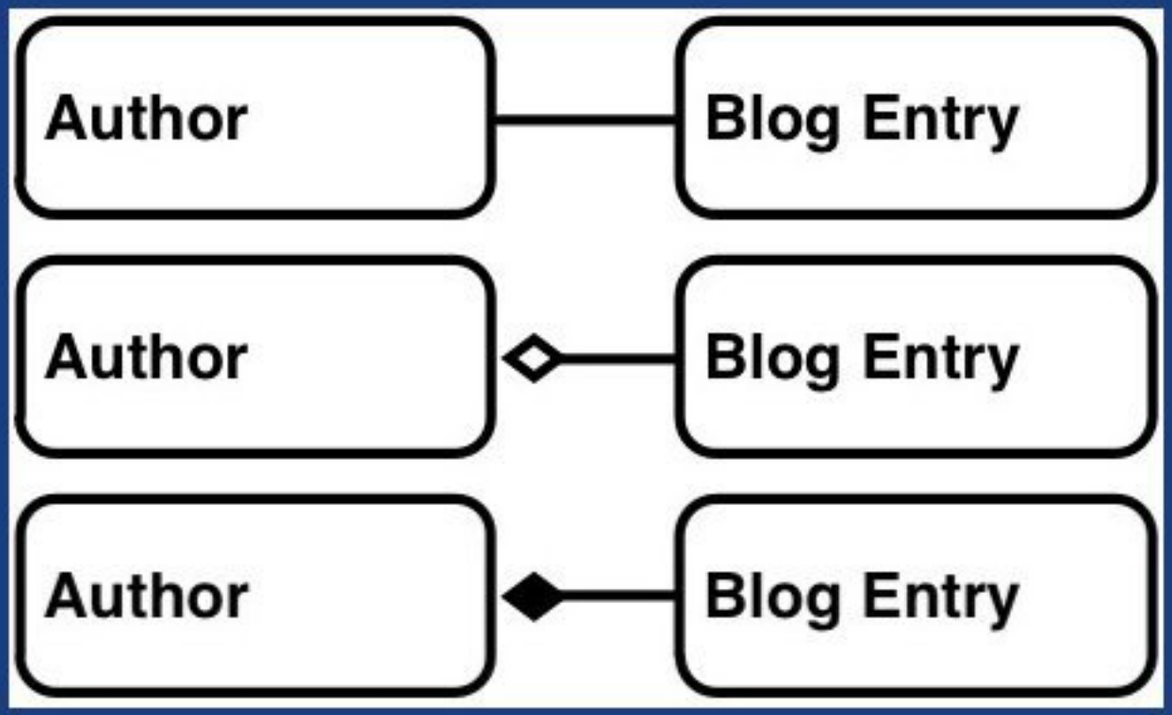
Inheritance

- Simple to query across sub-types
- Indexes on specialized values will be small

One to Many (1:n)

One to Many relationships can specify:

- degree of association between objects
- containment
- life-cycle



Embedding Comments and Views

```
{
  name: "Derick's MongoDB Tour Wrap-up",
  date: "2012-10-01",
  comments: [
    { name: "Adam", text: "It was great having you in Sou..." },
    { name: "Jake", text: "I don't think you can ever re..." },
  ],
  views: [
    { time: 1350297458, ip: "192.168.42.101" },
    { time: 1350297729, ip: "192.168.42.103" },
    { time: 1350298912, ip: "192.168.42.102" },
  ]
},
{
  name: "What is PHP doing?",
  date: "2012-07-13",
  comments: [
    { name: "Chris", text: "The .gdbinit trick was somethi..." },
  ],
  views: [
    { time: 1350297458, ip: "192.168.42.101" },
  ]
}
```

Tips and hints: array keys

Don't do:

```
temperature: {
  _id: 42,
  points: [
    { 1332942067: 17.3 },
    { 1332942118: 17.5 }
  ]
}
```

instead, do:

```
temperature: {
  _id: 42,
  points: [
    { ts: 1332942067, temp: 17.3 },
    { ts: 1332942118, temp: 17.5 }
  ]
}
```

Define and document your keys!

Tree structure

- single document
- natural
- hard to query

```
blogs: {
  author: "sambeau",
  date: "84 days ago",
  comments: [
    {
      author: "ElliotH",
      date: "84 days ago",
      text: "It's also missing the necessary credit to OpenStreetMap's
          contributors; we look forward to working with Apple to get that on
          there.",
      replies: [
        {
          author: "sambeau",
          date: "83 days ago",
          text: "I do think that this is a very classy move by the OSM people,
              no ranting blog post or 'Apple stole our stuff', welcoming people
              presents a much better image of the project."
        }
      ]
    }
  ]
}
```

Linking Articles and Views

By splitting views out into their own collection

Article collection:

```
{
  _id: 4,
  name: "Derick's MongoDB Tour Wrap-up",
  date: "2012-10-01",
  comments: [
    { name: "Adam", text: "It was great having you in Sou..." },
    { name: "Jake", text: "I don't think you can ever re..." },
  ],
},
{
  _id: 7,
  name: "What is PHP doing?",
  date: "2012-07-13",
  comments: [
    { name: "Chris", text: "The .gdbinit trick was somethi..." },
  ]
}
```

Views collection:

```
{ article_id: 4, time: 1350297458, ip: "192.168.42.101" },
{ article_id: 4, time: 1350297729, ip: "192.168.42.103" },
{ article_id: 4, time: 1350298912, ip: "192.168.42.102" },
{ article_id: 7, time: 1350297458, ip: "192.168.42.101" },
```

Tips and hints: pre-page results

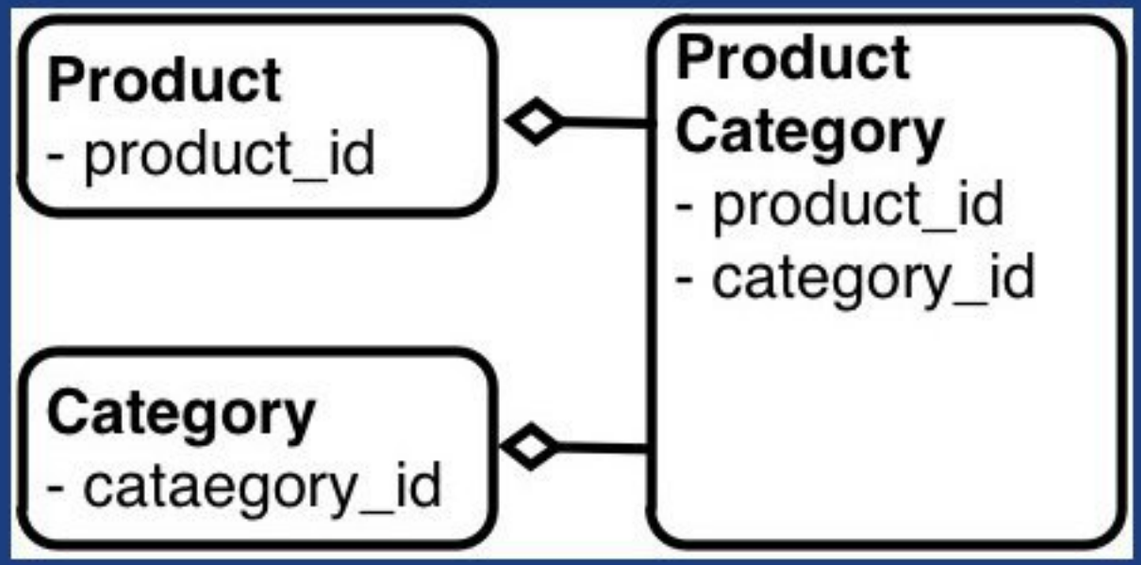
```
article: { _id: 42, title: "Xdebug saves you time!" }
```

```
comments:
{
  _id: 42,
  page: 0,
  comments: [
    { ts: 1332942067, author: "Derick", comment: "It also costs a lot of time to wr
    { ts: 1332942118, author: "...", comment: "..." },
    ...
  ]
},
{
  _id: 42,
  page: 1,
  comments: [
    { ts: 1332942261, author: "Derick", comment: "blah blah" },
    { ts: 1332942482, author: "...", comment: "..." },
  ]
}
```

A bit more work when updating, but a lot easier to retrieve

Many to Many (n:m)

- products can be in many categories
- category can have many products



Many to Many (n:m)

```
products:
{ _id: 10, name: "Blue elephpant", category_ids: [ 4, 7 ] }
{ _id: 11, name: "Pink elephpant", category_ids: [ 4, 8 ] }

categories:
{ _id: 4, name: "toys",             product_ids: [ 10, 11 ] }
{ _id: 8, name: "everything pink", product_ids: [ 11 ] }
```

All categories for a given product (pink elephpant):

```
db.categories.find( { product_ids: 11 } );
```

All products for a given category (toys):

```
db.products.find( { category_ids: 4 } );
```

Updates need to be done in two collections

Many to Many (n:m) - alternative 1

```
products:
{ _id: 10, name: "Blue elephpant", category_ids: [ 4, 7 ] }
{ _id: 11, name: "Pink elephpant", category_ids: [ 4, 8 ] }

categories:
{ _id: 4, name: "toys"            }
{ _id: 8, name: "everything pink"}
```

All categories for a given product (pink elephpant):

```
product = db.products.find( { _id: 11 } );
db.categories.find( { _id: { $in: product.category_ids } } );
```

All products for a given category (toys):

```
db.products.find( { category_ids: 4 } );
```

Embedding versus Linking

Embedding

- Simple data structure
- Limited to 16MB
- Larger documents
- How often do you update?
- Will the document grow and grow?

Linking

- More complex data structure
- Unlimited data size
- More, smaller documents
- What are the maintenance needs?

Any questions?

Indexes

Indexes: What's in store?

- What is indexing?
- Understanding different types of indexes
- Working with indexes in MongoDB

Indexes are the single biggest tunable performance factor in MongoDB.

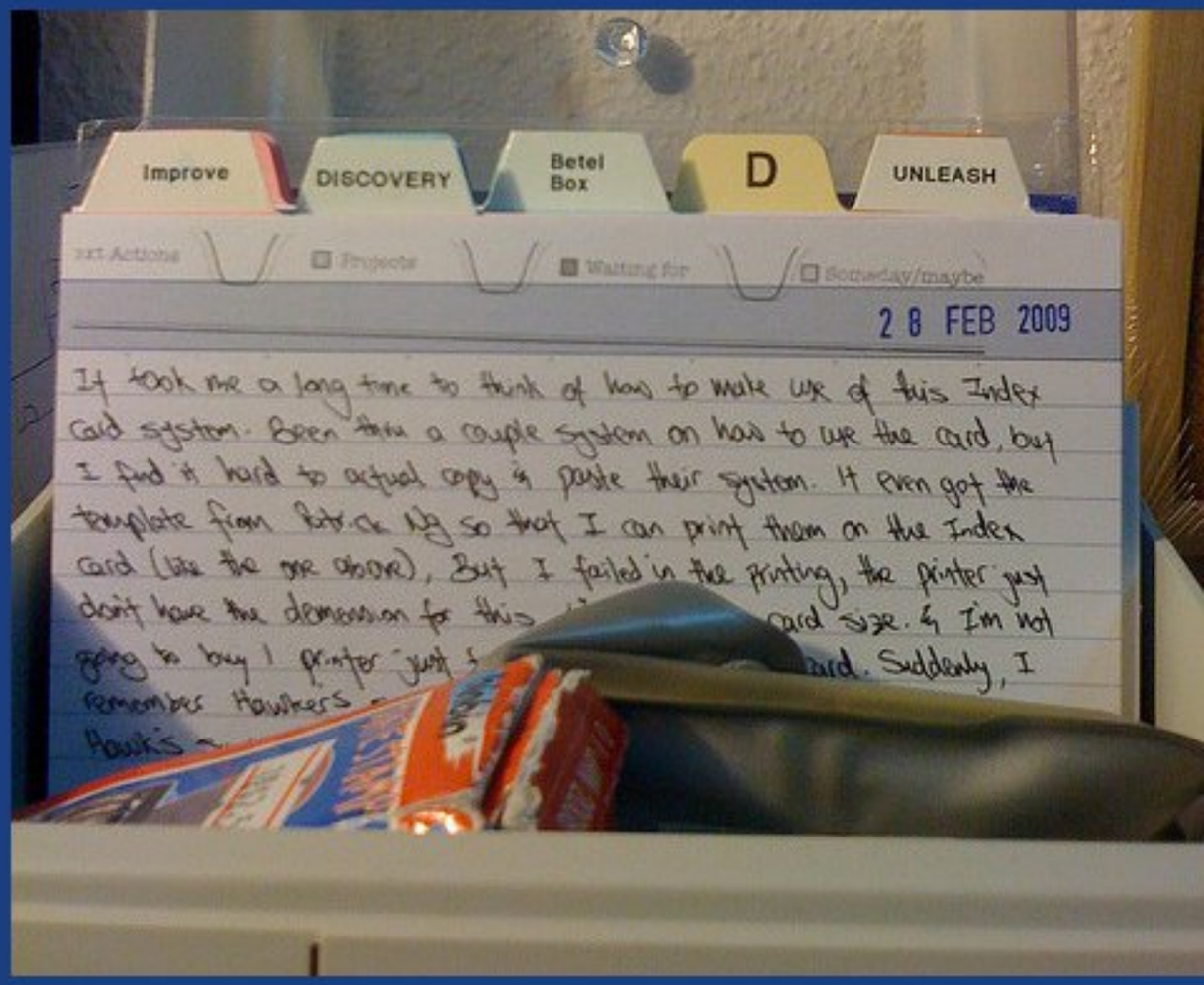
Absent or suboptimal indexes are the most common avoidable MongoDB performance problem.

So what problem do indexes solve?

Let's start with a map...

How do you find a city?

- Searching through the whole map is going to take a while...
- You probably want to come up with a smarter solution.
- Let's create an index!



Creating Indexes

The `_id` field is always indexed. Additional indexes can be created with `ensureIndex`:

```
// Create an index on the name attribute
db.cities.ensureIndex( { name: 1 } );

// Create a compound index on country code and feature code
db.cities.ensureIndex( { country_code: 1, feature_code: 1 } );

// Create an index on the area field of timezone:
db.cities.ensureIndex( { 'timezone.area': 1 } );
```

Let's imagine a simple index

City	Longitude	Latitude
Aabenraa	55.04434	9.41741
Aachen	50.77664	6.08342
...	...	...
London	51.50853	-0.12574
Londonderry County Borough	54.99721	-7.30917
Londrina	-23.31028	-51.16278
...	...	...
Zwijndrecht	51.8175	4.63333
Zwolle	52.5125	6.09444
Żyrardów	52.0488	20.44599
Zyryanovsk	49.72654	84.27318
Żywiec	49.68529	19.19243

How do you find a large city in a country?

Let's imagine a compound index

Country Code	Population	Name
AD	15853	les Escaldes
AD	20430	Andorra la Vella
...	...	...
GB	435791	Edinburgh
GB	447047	Sheffield
GB	455123	Leeds
GB	468945	Liverpool
GB	610268	Glasgow
GB	984333	Birmingham
GB	7556900	London
...	...	...

```
db.cities.ensureIndex( { country_code: 1, population: 1 } );
```

Let's look at the query plan

```
db.cities.find( { country_code: 'GB', population: { $gte: 500000 } } ).explain();
```

```
{
        "cursor" : "BtreeCursor country_code_1_population_1",
        "nscanned" : 4,
        "nscannedObjects" : 4,
        "n" : 4,
        "millis" : 0,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "isMultiKey" : false,
        "indexOnly" : false,
        "indexBounds" : {
                "country_code" : [ [ "GB", "GB" ] ],
                "population" : [ [ 500000, 1.7976931348623157e+308 ] ]
        }
}
```

```
{ "name" : "Glasgow", "country_code" : "GB", "population" : 610268 }
{ "name" : "Birmingham", "country_code" : "GB", "population" : 984333 }
{ "name" : "City of London", "country_code" : "GB", "population" : 7556900 }
{ "name" : "London", "country_code" : "GB", "population" : 7556900 }
```

Let's look at the query plan when we try to sort

```
db.cities.find( { country_code: 'GB', population: { $gte: 500000 } } )
         .sort( { population: -1 }).explain();
```

```
{
        "cursor" : "BtreeCursor country_code_1_population_1 reverse",
        "nscanned" : 4,
        "nscannedObjects" : 4,
        "n" : 4,
        "millis" : 0,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "isMultiKey" : false,
        "indexOnly" : false,
        "indexBounds" : {
                "country_code" : [ [ "GB", "GB" ] ],
                "population" : [ [ 1.7976931348623157e+308, 500000 ] ]
        }
}
```

```
{ "name" : "London", "country_code" : "GB", "population" : 7556900 }
{ "name" : "City of London", "country_code" : "GB", "population" : 7556900 }
{ "name" : "Birmingham", "country_code" : "GB", "population" : 984333 }
{ "name" : "Glasgow", "country_code" : "GB", "population" : 610268 }
```

How do you find a large city on a mountain?

Let's imagine another compound index

Country	Name	Population	DEM
AD	les Escaldes	15853	1033
AD	Andorra la Vella	20430	1037
...	...	...	...
ET	Addis Ababa	2757729	2405
ET	Ādīgrat	65000	2451
...	...	...	...
MX	Mexico City	12294193	2240
...	...	...	...

```
db.cities.ensureIndex( { population: 1, dem: 1 } );
```

Let's look at the query plan

```
db.cities.find( { population: { $gte: 2000000 }, dem: { $gte: 1654 } } ).explain();
```

```
{
        "cursor" : "BtreeCursor population_1_dem_1",
        "nscanned" : 127,
        "nscannedObjects" : 6,
        "n" : 6,
        "millis" : 3,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "isMultiKey" : false,
        "indexOnly" : false,
        "indexBounds" : {
                "population" : [ [ 2000000, 1.7976931348623157e+308 ] ],
                "dem" : [ [ 1654, 1.7976931348623157e+308 ] ]
        }
}
```

```
{ "name" : "Johannesburg", "population" : 2026469, "dem" : 1767 }
{ "name" : "Nairobi", "population" : 2750547, "dem" : 1684 }
{ "name" : "Addis Ababa", "population" : 2757729, "dem" : 2405 }
{ "name" : "Kabul", "population" : 3043532, "dem" : 1798 }
{ "name" : "Bogotá", "population" : 7102602, "dem" : 2582 }
{ "name" : "Mexico City", "population" : 12294193, "dem" : 2240 }
```

Let's have a quick look on how index look-ups work

```
db.cities.find( { population: { $gte: 2000000 }, dem: { $gte: 1654 } } );
```

	...	...	...
	Shijiazhuang	1992474	77
	Medellín	1999979	1500
	Almaty	2000900	793
	Zhengzhou	2014125	104
	Al Başrat al Qadīmah	2015483	6
	Kowloon	2019533	92
	Johannesburg	2026469	1767
	Dalian	2035307	33
⇒	...	...	...

```
n = 1 - nscanned = 6
```

Sorting by descending population

```
db.cities.find( { population: { $gte: 2000000 }, dem: { $gte : 1654 } } )
         .sort( { population: -1 } ).explain();
```

```
{
        "cursor" : "BtreeCursor population_1_dem_1 reverse",
        "nscanned" : 127,
        "nscannedObjects" : 6,
        "n" : 6,
        "millis" : 1,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "isMultiKey" : false,
        "indexOnly" : false,
        "indexBounds" : {
                "population" : [ [ 1.7976931348623157e+308, 2000000 ] ],
                "dem" : [ [ 1.7976931348623157e+308, 1654 ] ]
        }
}
```

```
{ "name" : "Mexico City", "population" : 12294193, "dem" : 2240 }
{ "name" : "Bogotá", "population" : 7102602, "dem" : 2582 }
{ "name" : "Kabul", "population" : 3043532, "dem" : 1798 }
{ "name" : "Addis Ababa", "population" : 2757729, "dem" : 2405 }
{ "name" : "Nairobi", "population" : 2750547, "dem" : 1684 }
{ "name" : "Johannesburg", "population" : 2026469, "dem" : 1767 }
```

Sorting by descending elevation

```
db.cities.find( { population: { $gte: 2000000 }, dem: { $gte : 1654 } } )
         .sort( { dem: -1 } ).explain();
```

```
{
        "cursor" : "BtreeCursor population_1_dem_1",
        "nscanned" : 127,
        "nscannedObjects" : 6,
        "n" : 6,
        "scanAndOrder" : true,
        "millis" : 1,
...
```

- Batch and sort results in memory
- No streaming (with getMore) of results
- 32 MB data limit for sorting

Querying by just the DEM

```
db.cities.find( { dem: { $gte : 1654 } } ).explain();
```

```
{
        "cursor" : "BasicCursor",
        "nscanned" : 22840,
        "nscannedObjects" : 22840,
        "n" : 614,
        "millis" : 26,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "isMultiKey" : false,
        "indexOnly" : false,
        "indexBounds" : {
        }
}
```

MongoDB can not use only the second part of a compound index (`{ population: 1, dem: 1 }`).

Sorting by descending elevation - take 2

```
db.cities.ensureIndex( { dem: -1 } );
db.cities.find( { population: { $gte: 2000000 }, dem: { $gte : 1654 } } )
         .sort( { dem: -1 } ).explain();
```

```
{
        "cursor" : "BtreeCursor population_1_dem_1",
        "nscanned" : 127,
        "nscannedObjects" : 6,
        "n" : 6,
        "scanAndOrder" : true,
        "millis" : 1,
...
```

MongoDB can only use one index at the same time.

Indexes

- MongoDB uses B-trees for indexes
- Many of the principles that apply to an RDBMS also apply to MongoDB
- e.g., indexes require additional work on inserts, updates and deletes
- A collection can have 64 indexes at most
- MongoDB can only use one index per query (with a few exceptions)

Indexes (2)

Be aware of redundant indexes. Order is important. With an index on `{ dem: 1, population: 1 }`:

- You **don't** need an index on `{ dem: 1 }`.
- You **do** need an index on `{ population: 1 }`.
- The index can be used for sorting on **dem** and then **population**, but not on **population** and then **dem**.
- Range search work best if used on the last field only (remember range on **population** and **dem**).

Index options

INDEX

MongoDB has a few options for indexes as well

Unique indexes

Unique indexes prevent multiple documents to match the same index key:

```
> db.cities.ensureIndex( { country_code: 1, name: 1 }, { unique: true } );
```

```
E11000 duplicate key error index:
demo.cities.$country_code_1_name_1  dup key: { : "AR", : "Mercedes" }
```

Remove the offending key and try again:

```
> db.cities.remove( { 'country_code': 'AR', name: 'Mercedes' } );

> db.cities.ensureIndex( { country_code: 1, name: 1 }, { unique: true } );

E11000 duplicate key error index:
demo.cities.$country_code_1_name_1  dup key: { : "AR", : "San Pedro" }
```

You can force the index with **dropDups** (This could delete a lot of data!):

```
> db.cities.ensureIndex( { country_code: 1, name: 1 },
                         { unique: true, dropDups : true } );
```

Sparse Indexes

Index on `admin1_code`:

A sparse index doesn't contain documents where the field-to-be-indexed is NULL:

```
db.cities.ensureIndex( { admin1_code: 1 }, { sparse: true } );
```

Sparse indexes can have unexpected results

```
> db.cities.find( { admin1_code: { $exists: true } } ).count();
22816
> db.cities.find( { admin1_code: { $exists: false } } ).count();
0
> db.cities.count();
22840
```

Let's look at `explain()`:

```
> db.cities.find( { admin1_code: { $exists: false } } ).explain();
{
        "cursor" : "BtreeCursor admin1_code_1",
        "nscanned" : 0,
        "nscannedObjects" : 0,
        "n" : 0,
...
```

A sparse index doesn't contain links to documents without the indexed field.

Covered Indexes

MongoDB can return data from the **index only**:

```
db.cities.ensureIndex( { country_code: 1, name: 1 } );
db.cities.find(
        { country_code: 'NL' }, { country_code: 1, name: 1, _id: 0 }
).explain();
```

```
{
        "cursor" : "BtreeCursor country_code_1_name_1",
        "nscanned" : 256,
        "nscannedObjects" : 256,
        "n" : 256,
        "millis" : 1,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "isMultiKey" : false,
        "indexOnly" : true,
...
```

- An index needs to be used in the first place
- Make sure to provide a projection
- Remove `_id` explicitly from the projection

Indexes on multi-value fields

Example:

```
{ "name" : "Arnhem", "alternatenames" : [ "Arnem", "Arnheim", "Arnhem", ... ] }
```

Create as a normal index:

```
db.cities.ensureIndex( { alternatenames: 1 } );
```

The `isMultiKey` field is set to `true`:

```
db.cities.find( { alternatenames: 'Arnheim' } ).explain();
{
    "cursor" : "BtreeCursor alternatenames_1",
    "nscanned" : 1,
    "nscannedObjects" : 1,
    "n" : 1,
    "millis" : 0,
    "nYields" : 0,
    "nChunkSkips" : 0,
    "isMultiKey" : true,
...
```

How does MongoDB pick an index?

- MongoDB does not keep statistics
- MongoDB just tries all the possible indexes
- When it has found the fastest one, it remembers it
- MongoDB will re-try all possible plans once in a while
- Indexes can be "hinted":

```
db.cities.ensureIndex( { population: 1, dem: 1 } );
db.cities.ensureIndex( { dem: -1 } );
db.cities.find( { population: { $gte: 2000000 }, dem: { $gte : 1654 } })
         .sort( { dem: -1 } )
         .hint( { dem: -1 } );
```

```
{
        "cursor" : "BtreeCursor dem_-1",
        "nscanned" : 614,
        "nscannedObjects" : 614,
        "n" : 6,
        "millis" : 2,
...
```

Indexes with $or

```
db.cities.ensureIndex( { country_code: 1 } );
db.cities.find( { $or: [{country_code: 'NL'}, {country_code: 'BE'}] } ).explain();
```

```
{
    "clauses" : [
        {
            "cursor" : "BtreeCursor country_code_1",
            "nscanned" : 256,
            "nscannedObjects" : 256,
            "n" : 256,
...
            "indexBounds" : { "country_code" : [ [ "NL", "NL" ] ] }
        },
        {
            "cursor" : "BtreeCursor country_code_1",
            "nscanned" : 184,
...
            "indexBounds" : { "country_code" : [ [ "BE", "BE" ] ] }
        }
    ],
    "nscanned" : 440,
    "nscannedObjects" : 440,
    "n" : 440,
    "millis" : 3
}
```

Back to the map...

Geospatial indexes (2d indexes)

MongoDB supports indexes on two dimensional fields:

```
db.cities.ensureIndex( { location: '2d' } )
```

```
db.cities.find( { location: { $near: [ -0.088, 51.489 ] } } );
```

```
db.cities.find( { location: { $near: [ -0.088, 51.489 ] } } ).explain();
{
    "cursor" : "GeoSearchCursor",
    "nscanned" : 100,
    "nscannedObjects" : 100,
    "n" : 100,
...
```

- By default only 100 items, but changable with `limit()`
- You can also add `maxDistance`:

```
db.cities.find( { location: { $near: [ -0.08, 51.48 ], $maxDistance: 0.1 } } );
```

Geospatial indexes (geoNear command)

```
db.runCommand( { geoNear: 'cities', near: [ -0.088, 51.489 ], num: 10 } );
```

```
{
    "ns" : "demo.cities",
    "near" : "0110111010111010111010011111101110111010110010011011",
    "results" : [
        { "dis" : 0.024097918997296273, "obj" : { "name" : "City of London", "ascii
        { "dis" : 0.02958403792587928,  "obj" : { "name" : "Brixton", "asciiname" :
    ],
    "stats" : {
        "time" : 0,
        "btreelocs" : 0,
        "nscanned" : 85,
        "objectsLoaded" : 26,
        "avgDistance" : 0.05905622408516078,
        "maxDistance" : 0.09879506691618341
    },
    "ok" : 1
}
```

- Includes distance
- Limited to "one document"

Geospatial indexes (spherical)

Since MongoDB 1.8 a spherical model is also supported:

```
db.cities.find( { location: { $nearSphere: [ -0.088, 51.489 ] } } );
```

```
db.runCommand( { geoNear: 'cities', near: [ -0.088, 51.489 ], spherical: true } );
```

- Location needs to be: **longitude, latitude** in a field.
- For 50°N and 13°E:
  - `[13, -50]`
  - `{ lon : 13, lat : -50 }`
  - `{ lat : -50, lon : 13 }`

Geospatial indexes (spherical)

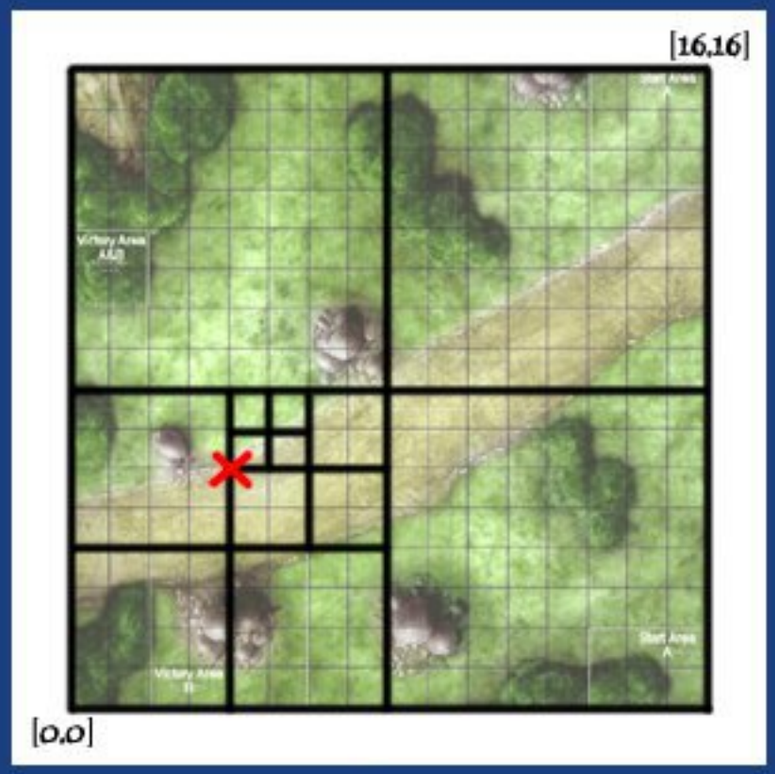
Find all cities within 150 km from London, in France:

```
db.runCommand( {
    geoNear: 'cities',
    near: [ -0.088, 51.489 ],
    spherical: true,
    query: { country_code: 'FR' },
    maxDistance: 150 / 6378
} ).results.
   forEach(function(obj) {
        print( obj.obj.country_code, '-', obj.obj.name, ': ', obj.dis * 6378 );
   } );
```

```
FR - Outreau :  146.16198019887855
FR - Boulogne-sur-Mer :  146.92634969912072
FR - Calais :  147.59429539150375
```

- Distances are in radians, so convert
- Results are limited to one document (16MB)

Geohashes



Geohash so far: 00110**00**

```
db.cities.ensureIndex( { location: '2d', bits: 4 } )
```

- Default `bits` is 26, about 0.3 meter resolution (*½ * 40000 / 2^26*)

When can indexes not be used?

- Many sorts of negations (`$ne`, `$not`, `$nin`).
- Tricky aritmethic (`$mod`).
- Most regular expressions (`/ondo/`), unless anchored to the start of the string (`/^Lon/`).
- JavaScript expressions in `$where` clauses, such as `where dem > population`. In that case, try to pre-compute.
- map/reduce

Any questions?

Practical Example

OpenStreetMap

- "Wikipedia for Map **Data**"
- Licensed under the Open Database License (ODbL 1.0):

  You are free to copy, distribute, transmit and adapt our maps and data, as long as you **credit OpenStreetMap** and its contributors. If you alter or build upon our maps or data, you may distribute the result only under the same licence.

- Rendered map: http://openstreetmap.org/
- A lot of data is not rendered, but is available.

The Data

- Nodes (Lat/Lon point)

```
<node id='459517295' lat='50.0100766' lon='8.3162402' user='WoGo' timestamp='200
```

- Ways (Ordered interconnection of nodes, ie: roads)
- Areas (Closed ways, ie: buildings, squares and landuse)

```
<way id='76174399' user='Derick Rethans' uid='37137' timestamp='2010-09-06T08:30
    <nd ref='898861293'/>
    <nd ref='898861305'/>
    <nd ref='898861298'/>
    <nd ref='898861315'/>
    <nd ref='898861293'/>
    ...
</way>
```

- Tags (Describe an element)

```
<tag k='addr:housenumber' v='375'/>
<tag k='addr:street' v='Kilburn High Road'/>
<tag k='amenity' v='pub'/>
<tag k='building' v='yes'/>
<tag k='name' v='North London Tavern'/>
```

Tagging features (1)

Tagging features (2)

addr:housenumber: 29
addr:street: Saint Martin's Lane
amenity: pub
building: yes
building:levels: 4
name: The Chandos

Tagging features (3)

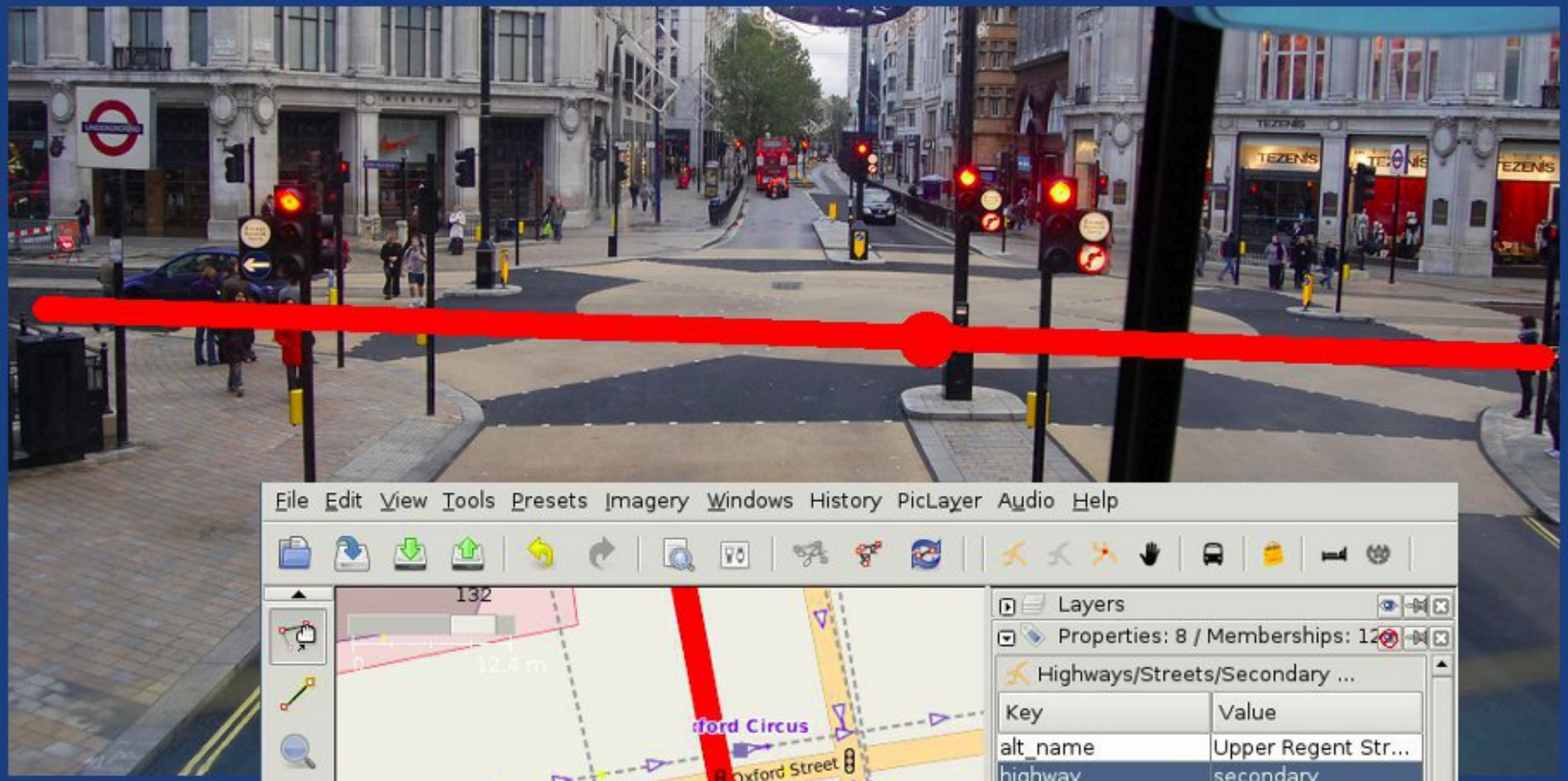
amenity: cafe
name: Pret A Manger
website: http://www.pret.com

Tagging features (4)

highway: secondary
name: Saint Martin's Lane
oneway: yes

natural: tree

Editing tags



Using OpenStreetMap data

Process:

- Use XAPI, API or extracts to fetch data
- Parse XML file with PHP into a DB (MongoDB)
- Query database
- Show data
- WIN!

Fetching OSM data

```
wget http://download.geofabrik.de/osm/europe/great_britain/england.osm.bz2
```

(Or pick a smaller country from `http://download.geofabrik.de/osm/europe/`)

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="Osmosis 0.39">
  <bound box="51.26000,-0.56300,51.68000,0.28000" origin="http://www.openstreetmap.
  <node id="44" version="3" timestamp="2011-09-08T20:53:43Z" uid="15867" user="matt
  <node id="47" version="2" timestamp="2011-09-08T20:53:43Z" uid="15867" user="matt
  <node id="52" version="3" timestamp="2011-09-08T21:13:19Z" uid="15867" user="matt
...
  <node id="1571982070" version="1" timestamp="2011-12-31T20:37:18Z" uid="545369" u
    <tag k="name" v="Charing Cross"/>
    <tag k="railway" v="subway_entrance"/>
    <tag k="source:name" v="survey"/>
    <tag k="source:railway" v="survey"/>
  </node>
...
  <way id="74" version="5" timestamp="2010-12-12T22:43:15Z" uid="508" user="Welshie
    <nd ref="196101"/>
    <nd ref="196100"/>
    <nd ref="196331"/>
    <tag k="abutters" v="retail"/>
    <tag k="highway" v="primary"/>
    <tag k="maxspeed" v="30 mph"/>
    <tag k="name" v="Ballards Lane"/>
    <tag k="ref" v="A598"/>
  </way>
```

Importing OSM into MongoDB

From `http://www.openstreetmap.org/browse/way/4442243`:

```
Way:     Brondesbury Road (4442243)
Tags:    highway = secondary
         name = Brondesbury Road
         ref = B451
         source:ref = OS OpenData StreetView
```

or a little more organised:

```
{
    '_id': 'w4442243',
    'name' : 'Brondesbury Road',
    'tags': {
        'highway': 'secondary',
        'ref' : 'B451',
        'source_ref' : 'OS OpenData StreetView',
    }
}
```

Importing OSM into MongoDB (2)

```
{
    '_id': 'w4442243',
    'name' : 'Brondesbury Road',
    'tags': {
        'highway': 'secondary',
        'ref' : 'B451',
        'source_ref' : 'OS OpenData StreetView',
    }
}
```

Querying all secondary roads:

```
db.poi.find( { 'tags.highway' : 'secondary' } );
```

Find all roads:

```
db.poi.find( { 'tags.highway' : { $exists : true } );
```

Importing OSM into MongoDB (3)

We need an index for that:

```
db.poi.ensureIndex( { "tags.amenity" : 1 } );
```

And the new "explain" output:

```
db.poi.find( { 'tags.amenity' : 'pub' } ).explain();
{
    "cursor" : "BtreeCursor tags.amenity_1",
    "isMultiKey" : false,
    "n" : 3895,
    "nscannedObjects" : 3895,
    "nscanned" : 3895,
    ...
```

Importing OSM into MongoDB (4)

```
{
    '_id': 'w4442243',
    'name' : 'Brondesbury Road',
    'tags': {
        'highway': 'secondary',
        'ref' : 'B451',
        'source_ref' : 'OS OpenData StreetView',
    }
}
```

```
db.poi.ensureIndex( { "tags.amenity" : 1 } );
```

```
db.poi.ensureIndex( { "tags.highway" : 1 } );
```

```
db.poi.ensureIndex( { "tags.ref" : 1 } );
```

```
db.poi.ensureIndex( { "tags.???" : 1 } );
```

- Lots of indexes make things slow
- Limited to 64 indexes

Tags as a key/value combination

```
$doc = array(
    'name' => 'A440',
    'tags' => [
        'highway' => 'secondary',
        'oneway' => 'yes'
    ]
);
$db->poi->ensureIndex( [ 'name' => 1 ] );
$db->poi->ensureIndex( [ 'tags.highway' => 1 ] );
$db->poi->ensureIndex( [ 'tags.oneway' => 1 ] );

// Road with name=Strand
$db->poi->find( [ 'name' => 'Strand' ] );
// All roads
$db->poi->find( [ 'tags.highway' => [ '$exists' =>  1 ] ] );
```

- Lots of indexes are required
- Easy to extract data from

Tags as dictionary

```
$doc = array(
    'tags' => [
        [ 'k' => 'name', 'v' => 'A440' ],
        [ 'k' => 'highway', 'v' => 'secondary' ],
        [ 'k' => 'oneway', 'v' => 'yes' ]
    ]
);
$db->poi->ensureIndex( [ 'tags.v' => 1, 'tags.k' => 1 ] );

// Road with name=Strand
$db->poi->find( [
    'tags' => [ '$elemMatch' => [ 'k' : 'name', 'v' : 'Strand' ] ]
] );
// All roads
$db->poi->find( [ 'tags.k' => 'highway' ] );
```

- Two indexes required:
  - one for key/value combinations
  - one for "all roads" question (on `tags_indexed.k`)
- Good for finding key/value combinations

Tags as concatenated strings

```
$doc = array(
    'tags' => [
        'name=A440',
        'highway=secondary',
        'oneway=yes'
    ]
);
$db->poi->ensureIndex( [ 'tags' => 1 ] );

// Road with name=Strand
$db->poi->find( [ 'tags' => 'name=Strand' ] );
// All roads
$db->poi->find( [ 'tags' => new MongoRegex( '/^highway=/' ) ] );
```

- One index required
- Good for finding key/value combinations
- Good enough for doing the "all roads" question

Any questions?

Replication

Replication

- Replication, elections and configuration
- High availability scenarios

Replication

Use cases

- High Availability (auto-failover)
- Read Scaling (extra copies to read from)
- Backups
  - Online, Delayed Copy (fat finger)
  - Point in Time (PiT) backups

- Use (hidden) replica for secondary workload
  - Analytics
  - Data-processing
  - Integration with external systems (f.e. Solr)

Types of outages

Planned

- Hardware upgrade
- O/S or file-system tuning
- Relocation of data to new file-system / storage
- Software upgrade

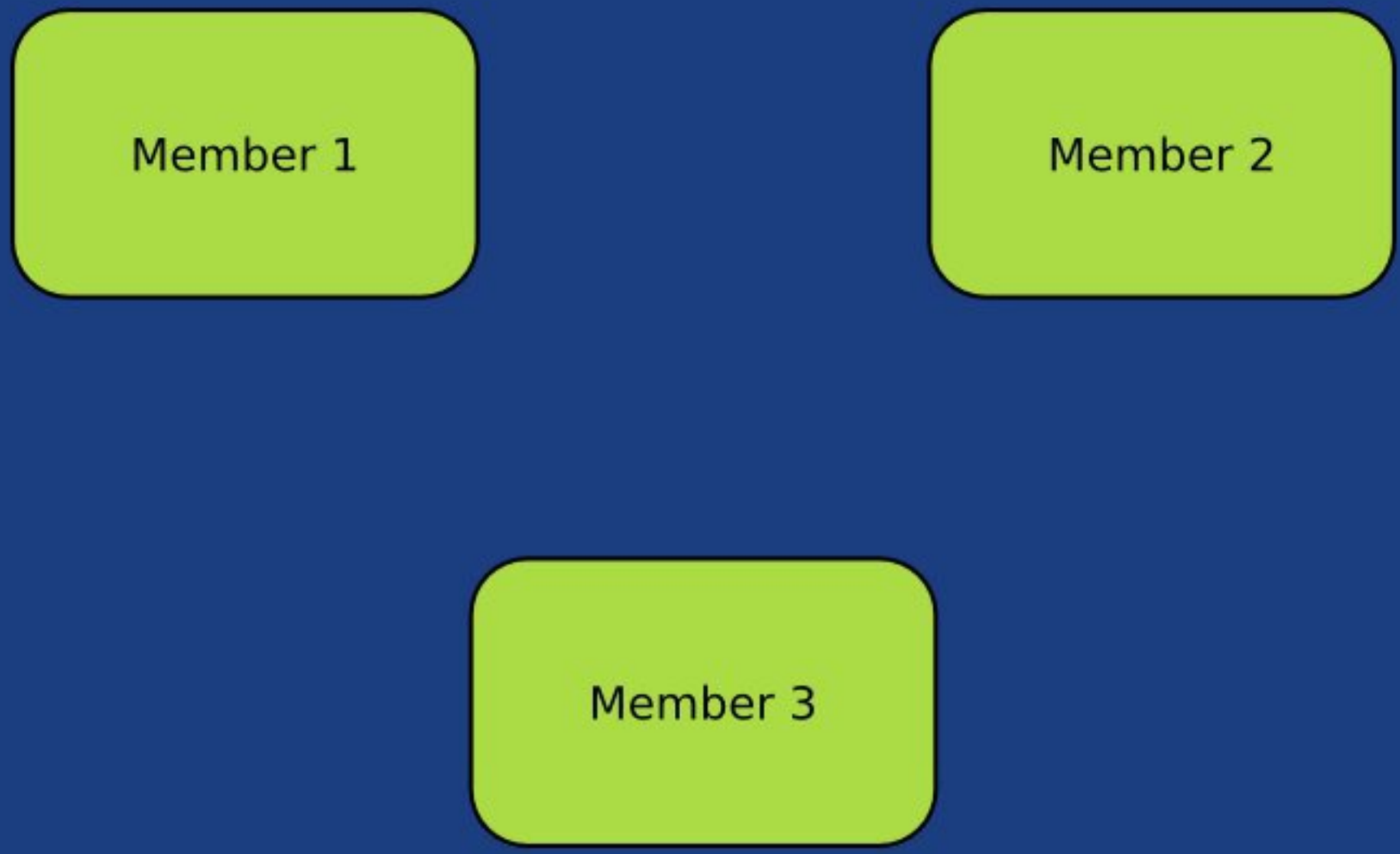
Unplanned

- Hardware failure
- Data center failure
- Region outage
- Human error
- Application corruption

Replicaset features

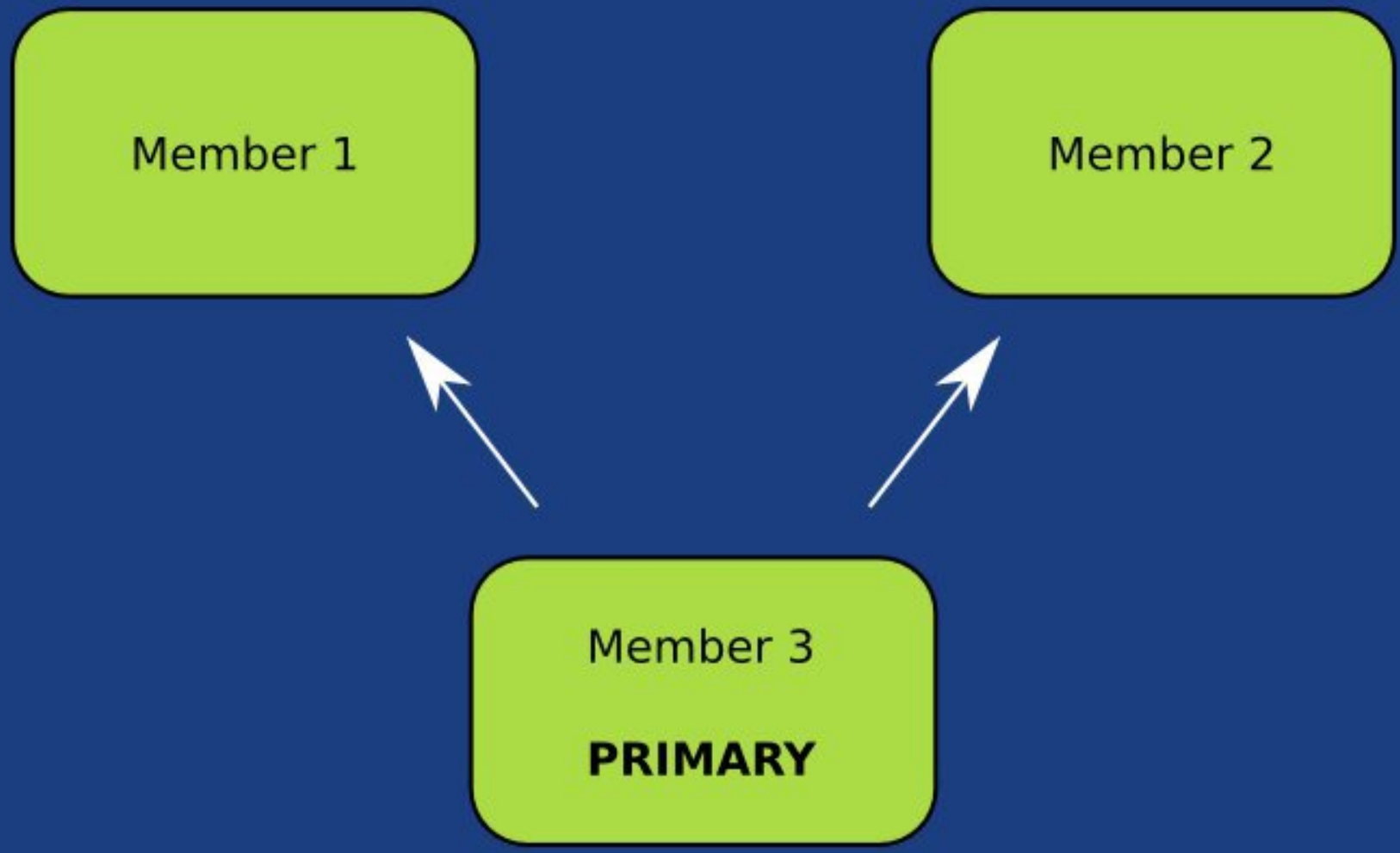
- A cluster of N servers
- All writes to primary
- Reads can be to primary (default) or a secondary
- Any (one) node can be primary
- Consensus election of primary
- Automatic failover
- Automatic recovery

How MongoDB replication works



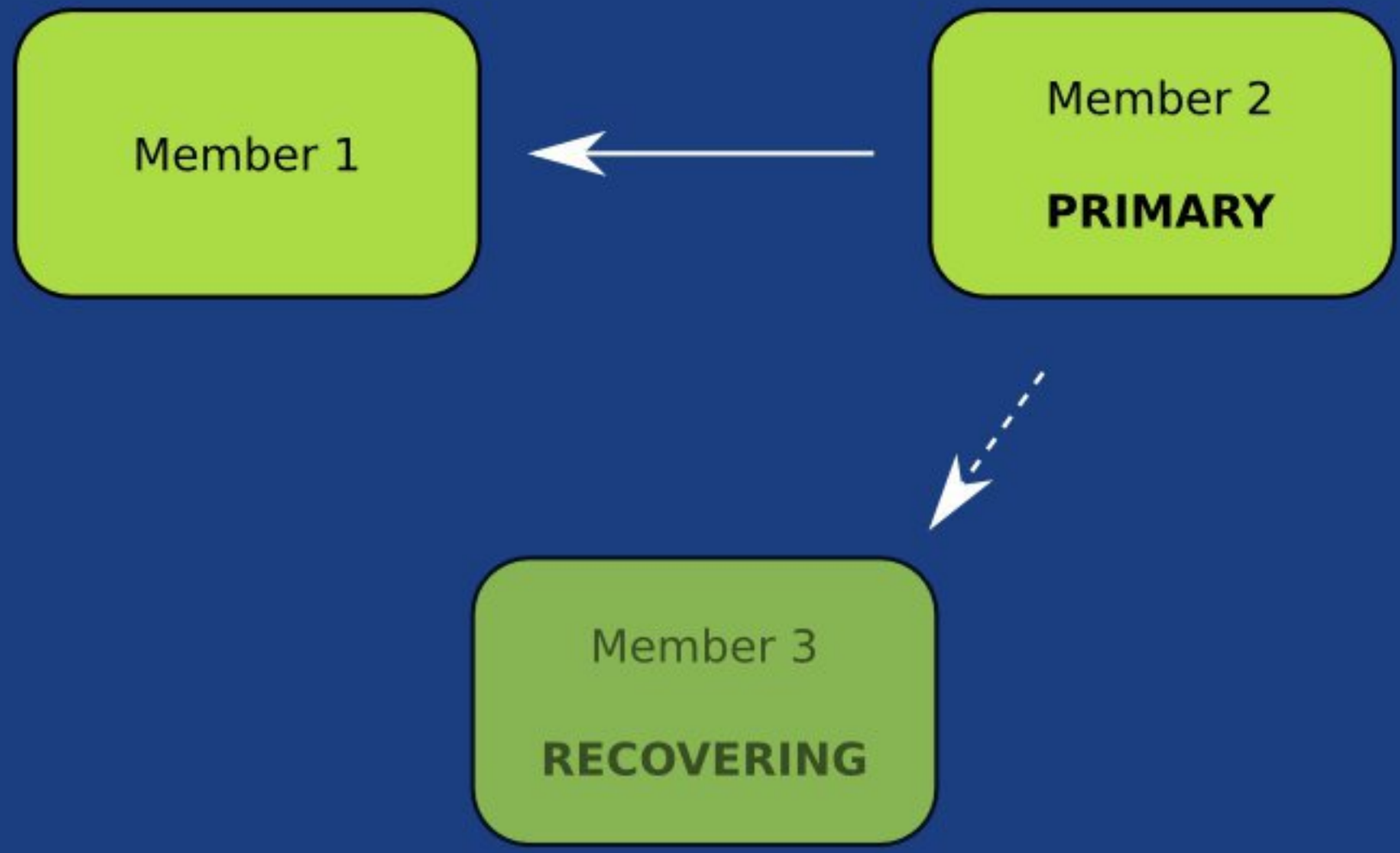
- Set is made up of 2 or more nodes
- It makes most sense to have an **odd** number of nodes

How MongoDB replication works



- Election establishes the PRIMARY
- Data replication from PRIMARY to SECONDARY

How MongoDB replication works



- Automatic recovery

How does replication work?

- Change operations are written to the oplog
  - The oplog is a capped collection (fixed size)
    - Must have enough space to allow new secondaries to catch up after copying from a primary
    - Must have enough space to cope with any applicable slaveDelay
- Secondaries query the primary's oplog and apply what they find
- All replicas contain an oplog

Setting up replication

Starting the daemons (2 data, 1 arbiter):

```
mongod -f /etc/mongodb.conf --dbpath=~/repl-slave0 --port 13000 --replSet seta
mongod -f /etc/mongodb.conf --dbpath=~/repl-slave1 --port 13001 --replSet seta
mongod -f /etc/mongodb.conf --dbpath=~/repl-arb --port 13002 --replSet seta --rest
```

Configure the set:

```
mongo whisky:13000

> db.adminCommand( {
  replSetInitiate: {
    _id: 'seta',
    members: [
     { _id: 0, host: 'whisky:13000', tags: { 'dc': 'west', 'use': 'accounting' } },
     { _id: 1, host: 'whisky:13001', tags: { 'dc': 'east', 'use': 'reporting' } },
    ]
  }
} );

PRIMARY> rs.addArb('whisky:13002');
```

Managing a replica set

- `rs.conf()`: get current configuration
- `rs.initiate(cfg)`: initiate replica set
- `rs.reconfig(cfg)`: reconfigure a replica set
- `rs.add("hostname:port")`: add a new member
- `rs.addArb("hostname:port")`: add a new arbiter
- `rs.remove("hostname:port")`: remove a member

Priorities

Priorities

- Priority, floating point number between 0 and 100
- Used during an election:
  - Most up to date
  - Highest priority
  - Less than 10s behind failed Primary
- Allows weighting of members during failover

Priorities

- Assuming all members are up to date
- Members A or B will be chosen first
  - Highest priority
- Members C or D will be chosen when:
  - A and B are unavailable
  - A and B are not up to date
- Member E is never chosen
  - priority:0 means it cannot be elected

Write concerns

Safeness levels:

- Confirm replication: `array( 'safe' => true, 'w' => 2 );`

```
<?php
$m = new Mongo( 'mongodb://localhost:13000/?replSet=seta' );
$c = $m->demo->talks;

$c->insert(
    array( '_id' => 'mongodb' ),       // document
    array( 'safe' => true, 'w' => 2 ) // options
);
?>
```

- `w != 0` automatically enables "getLastError"
- `w > 1` means that the extra nodes have read from the oplog

Write concerns

- Safe mode:
```
$c->insert( [ 'text' => 'example' ], [ 'w' => 1 ] );
```
- Safe mode:
```
$c->insert( [ 'text' => 'example' ], [ 'safe' => true ] );
```
- Two nodes:
```
$c->insert( [ 'text' => 'example' ], [ 'w' => 2 ] );
```
- Majority of nodes:
```
$c->insert( [ 'text' => 'example' ], [ 'w' => "majority" ] );
```
- Tagged nodes:
```
$c->insert( [ 'text' => 'example' ], [ 'w' => "allDCs" ] );
```

Tagging


PSP
AA 2438
PALM SPRINGS CA
10NOV
DFW
AA 753
DALLAS FT WORTH TX
10NOV
SKD 1425
JFK
B6 82
527304834
3016UA-715543
UA 7002
UA 6496
SEA
LAX
United. #1 in
-time arriv

Tagging

- Control over where data is written to.
- Each member can have one or more tags:

```
tags: {dc: "stockholm"}
tags: {dc: "stockholm",
       ip: "192.168",
       rack: "row3-rk7"}
```

- Replica set defines rules for where data resides
- Rules can change without change application code

Tagging example

```
seta:SECONDARY> rs.config();
{
  "_id" : "seta", "version" : 5,
  "members" : [
   { "_id": 0, "host": "w:13000", "tags": { "dc": "west",  "use": "accounting" } },
   { "_id": 1, "host": "w:13001", "tags": { "dc": "east",  "use": "reporting" } },
   { "_id": 2, "host": "w:13002", "arbiterOnly": true }
   { "_id": 3, "host": "w:13003", "tags": { "dc": "east",  "use": "accounting" } },
   { "_id": 4, "host": "w:13004", "tags": { "dc": "north", "use": "reporting" } },
  ],
  settings : {
    getLastErrorModes: {
      allDCs:  { "dc": 3 },
      allUses: { "use": 2 }
    }
  }
}
```

From PHP, enforce the write tags by:

```
$c->insert( array( … ), array( 'safe' => 'allUses' ) );
```

Read preferences

Allows the driver to select between candidate servers from a specific set

- primary (default)
- primary_preferred (`Mongo::RP_PRIMARY_PREFERRED`)
- secondary
- secondary_preferred (default when old slaveOkay is used)
- nearest (within 15ms of closest)

- You can restrict the candidates with read preference tags
- Only the candidates within 15ms ping time from the fastest are used
- Random (per-query) among the matched sets

Read preferences

```
<?php
$m = new Mongo(
        'mongodb://localhost:13000/?' .
        'replicaSet=poiset&readPreference=nearest&readPreferenceTags=dc:asia'
);
$c = $m->phpunit->test;
$c->find(...);
?>
```

or:

```
<?php
$m = new Mongo( 'mongodb://localhost:13000/?replicaSet=poiset' );
$c = $m->phpunit->test;
$c->setReadPreference( Mongo::RP_SECONDARY, array( 'dc' => 'europe' ) );
$c->find(...);
?>
```

Read preferences

```
mongodb://mongo1.local,mongo3.local/?replicaSet=demo ⏎
        &readPreference=secondary&readPreferenceTags=dc:europe
```

Servers (after matching tags):

```
mongo2.local - secondary - 25ms - dc:europe,server:small
mongo3.local - secondary -  4ms - dc:europe,server:big
mongo4.local - secondary - 17ms - dc:europe,server:big
```

Eventual consistency

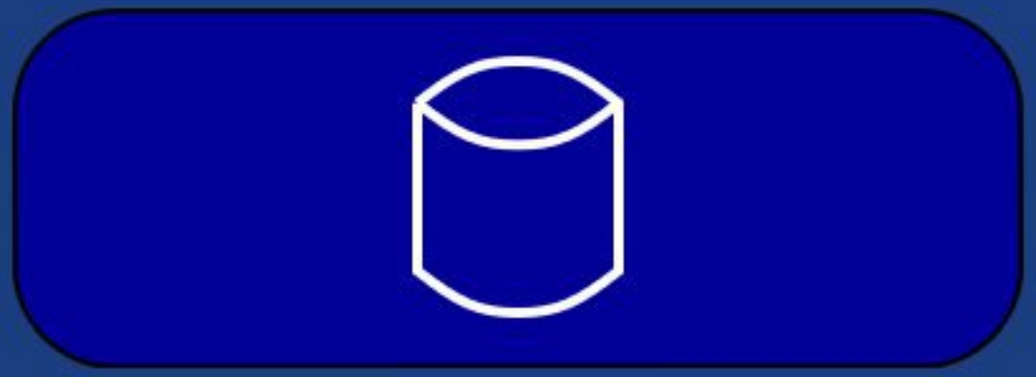
- Read Preference / Slave Okay
  - driver will always send writes to Primary
  - driver will send read requests to Secondaries
- Warning!
  - Secondaries may be out of date
  - Not applicable for all applications

Replicaset wrap-up

- Reads from Primary are always consistent
- Reads from Secondaries are eventually consistent
- Automatic failover if a Primary fails
- Automatic recovery when a node joins the set
- Full control of where writes occur
- Full control of where reads occur

High Availability Scenarios

A single node



- Will have downtime
- Human intervention required if node crashes
- Journal is required

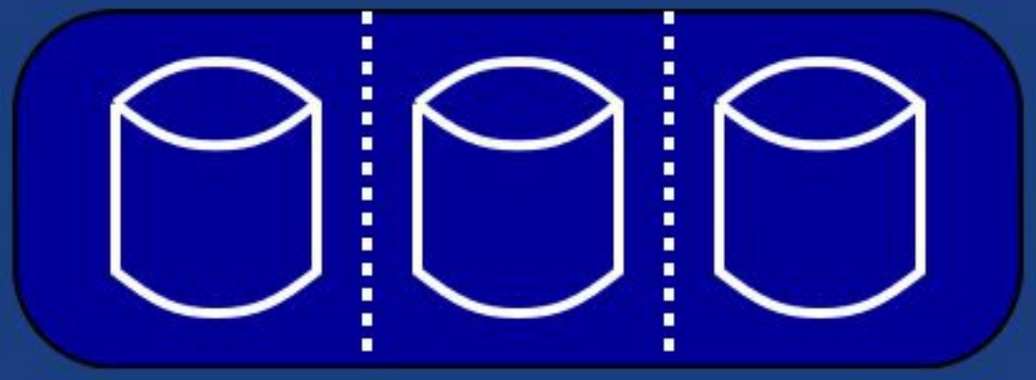
Replicaset 1

- Single datacentre
- Single switch and power
- One node failure
- Automatic recovery of single node crash
- Points of failure:
  - Power
  - Network
  - Datacentre

Replicaset 2

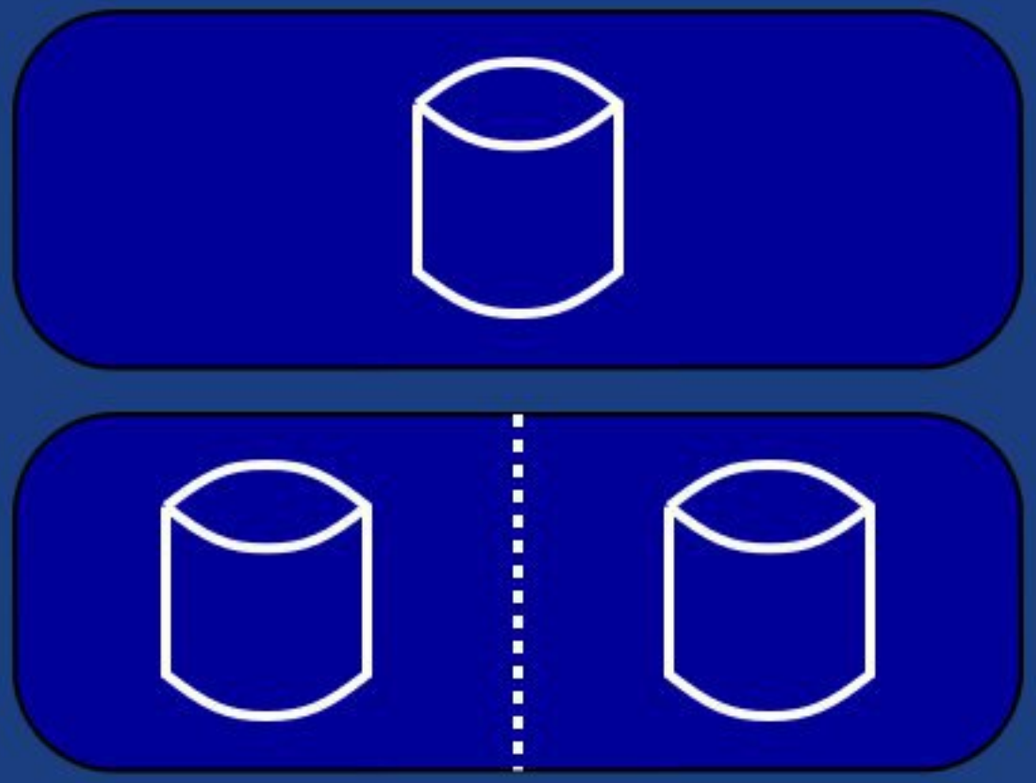
- Single datacentre
- Multiple power/network zones
- Automatic recovery of single node crash
- w=2 not viable as losing one node means no writes
- Points of failure:
  - Datacentre
  - Two node failure

Replicaset 3



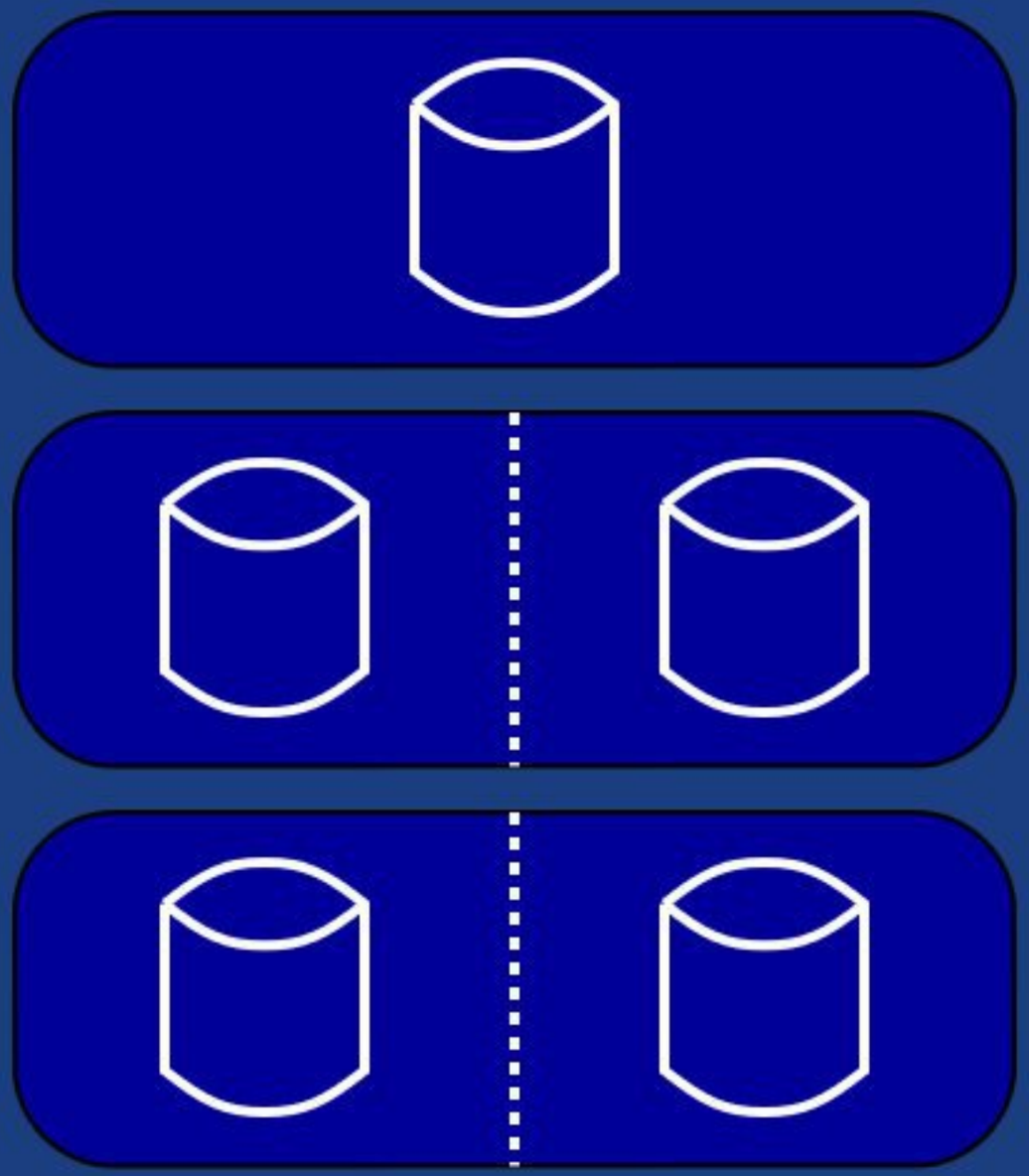
- Single datacentre
- Multiple power/network zones
- Automatic recovery of single node crash
- w=2 viable as 2 out of 3 nodes available
- Points of failure:
  - Datacentre
  - Two node failure

Replicaset 4



- Multi datacentre
- DR node for safety
- Can't do multi datacentre durable write safely since only one node in distant datacentre

Replicaset 5



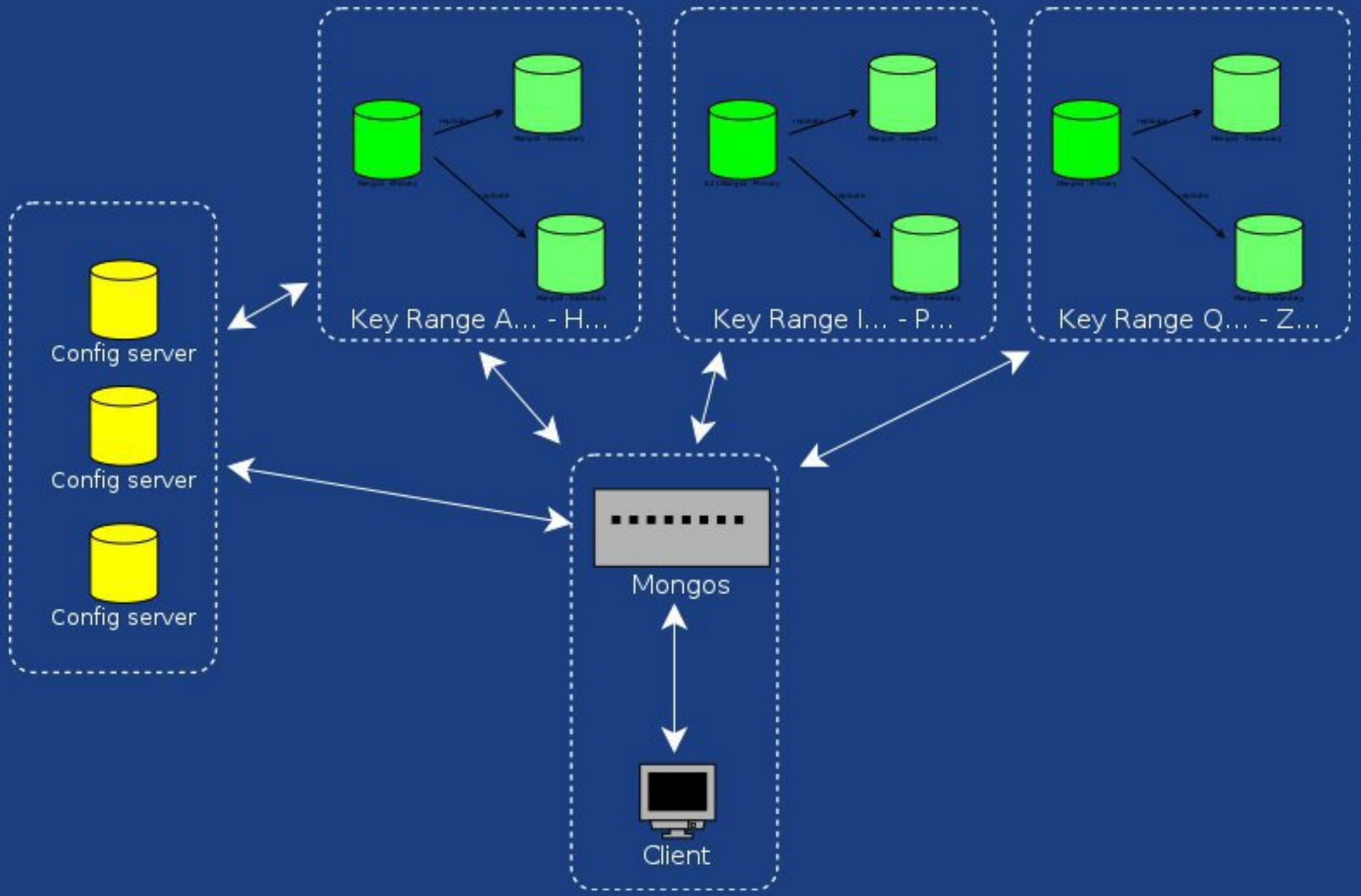
- Three datacentres
- Can survive full datacentre loss
- Can do `w = { dc : 2 }` to guarantee write in two datacentres

Typical deployments

Use	Set size	Data protection	High Availability	Notes
	1	No	No	Must use --journal to protect against crashes
✔	2	Yes	No	On loss of one member, set becomes read-only
✔	3	Yes	Yes - 1 failure	On loss of one member, two surviving nodes can elect new primary
	4	Yes	Yes - 1 failure	On loss of two members, two surviving nodes are read-only
✔	5	Yes	Yes - 2 failures	On loss of two members, three surviving nodes can elect new primary

Sharding

Sharding



- Scaling writes and reads
- Config servers, router (`mongos`) and replica sets

Sharding: How does it work?

- Sharding happens on a collection level
- The sharding key determines how data is distributed

```
db.runCommand( {
        shardcollection: "demo.cities",
        key: { name: 1 }
} );
```

- Data is stored in chunks, each with a maximum size of 64MB
- A chunk contains data in the form of a subsection of the shard key
- **mongos** makes sure that chunks are distributed
- Chunk meta-data is stored in the **config servers**
- Each shard is a replica-set

Sharding: mongos

- Acts just like a `mongod` towards clients
- You can have as many as you want
- Lightweight, so can run on the web servers
- Caches meta-data from config servers
- Routes queries to the correct shards:
  - Insertions directly to one shard
  - Queries to one, a few, or all shards

Sharding: config server

- Three of them
- Changes use two phase commit
- If any are down, meta-data goes read-only
- System is online, as long as one is up

Any questions?

Resources

- Slides: http://derickrethans.nl/talks/mongo-tut-zendcon12
- Contact me: Derick Rethans: @derickr, derick@10gen.com
- Feedback: http://joind.in/6861

10gen | the MongoDB company

mongoDB