

Welcome!

MongoDB schema design

PHP NorthWest - Manchester, UK - Apr 3rd, 2012

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

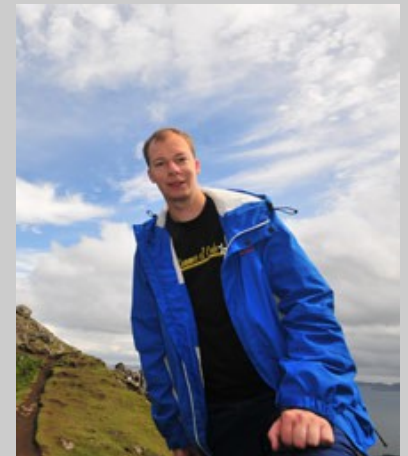
[@derickr](https://twitter.com/derickr)

[e-mail me](mailto:derick@10gen.com)

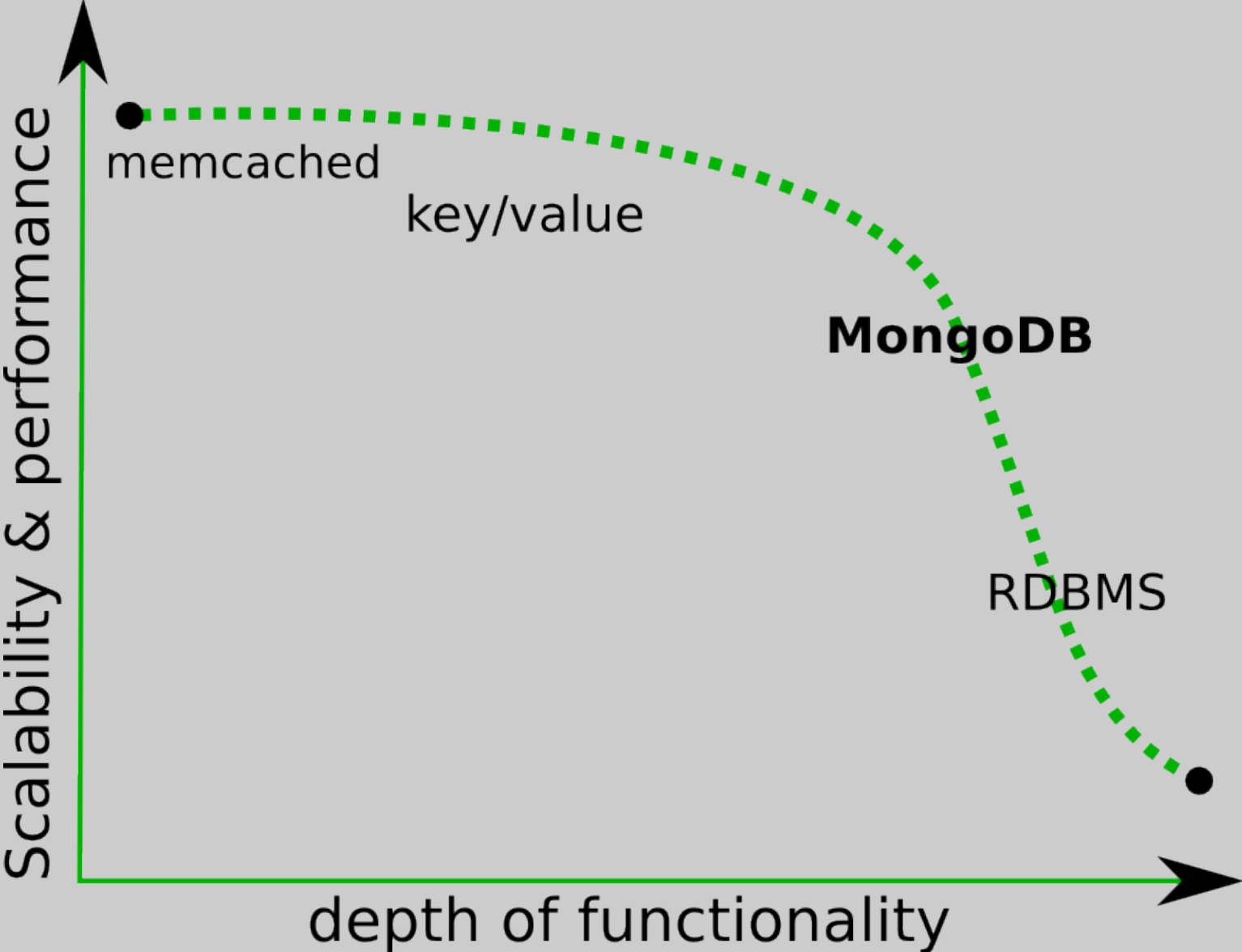


Derick Rethans

- Dutchman living in London
- PHP MongoDB driver maintainer for 10gen (the company behind MongoDB)
- Author of Xdebug
- Author of the `mcrypt`, `input_filter`, `dbus`, `translit` and `date/time` extensions



Database landscape



NoSQL



Key/value



Cassandra



Column



sones

Graph



CouchDB
relax



Document

Terminology

- JSON 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

Simple document:

```
{
  "_id" : ObjectId("4cb4ab6d7addf98506010001"),
  "handle" : "derickr",
  "name" : "Derick Rethans"
}
```

Document with embedded documents:

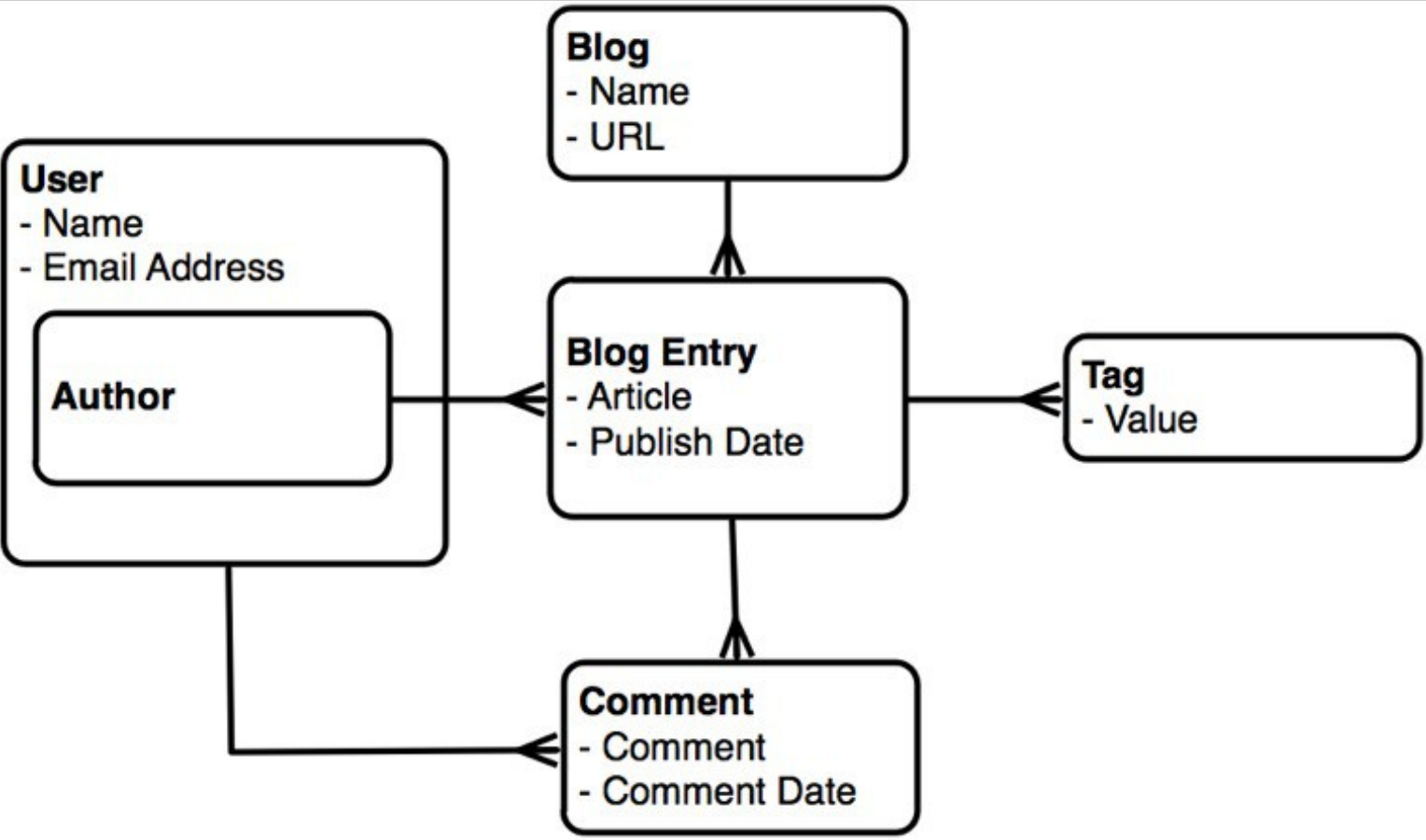
```
{
  "_id" : "derickr",
  "name" : "Derick Rethans",
  "talks" : [
    { "title" : "Profiling PHP Applications",
      "url" : "http://derickrethans.nl/talks/profiling-phptour.pdf",
    },
    { "title" : "Xdebug",
      "url" : "http://derickrethans.nl/talks/xdebug-phpbcn11.pdf",
    }
  ]
}
```

- 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)

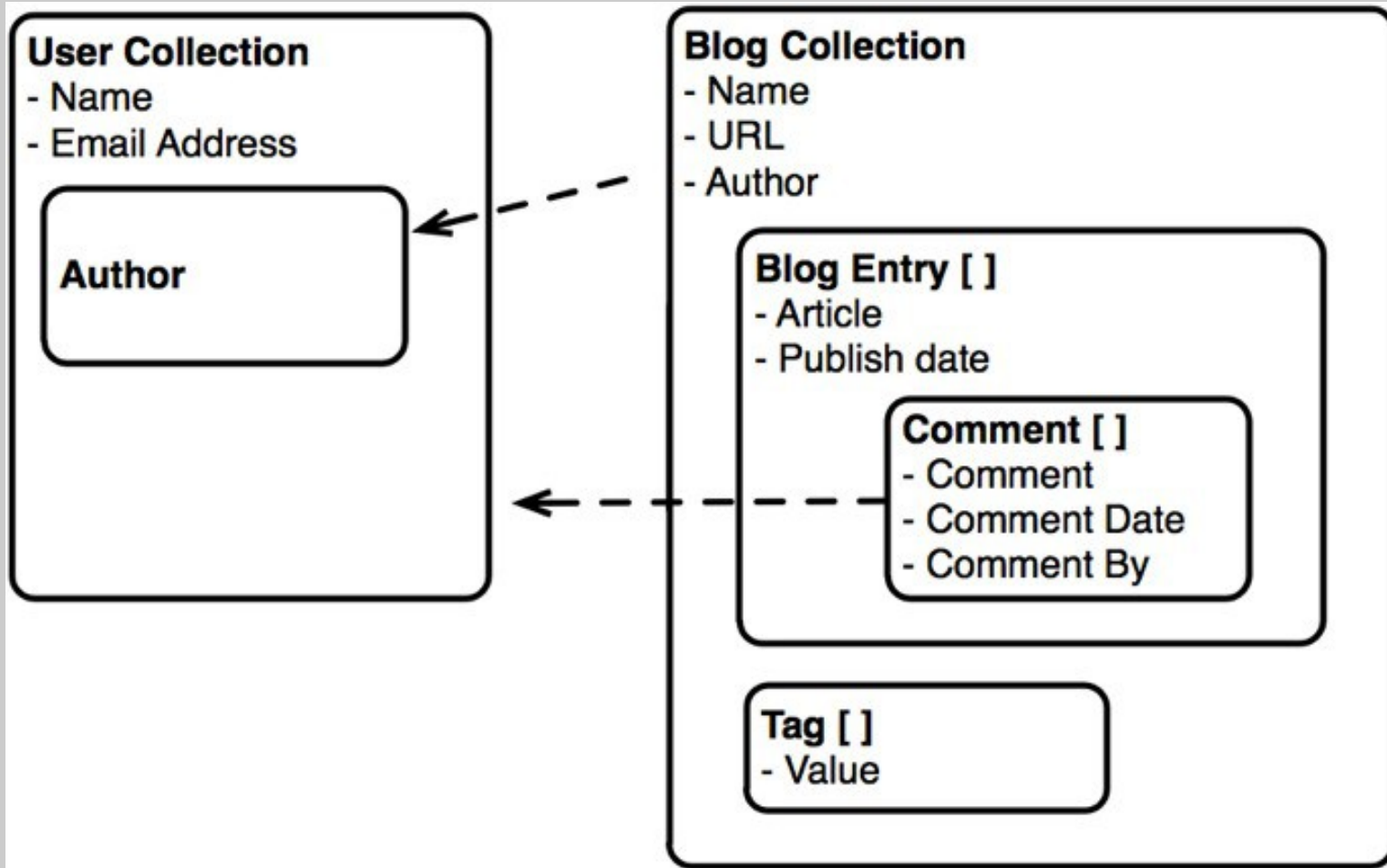
Goals:

- 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



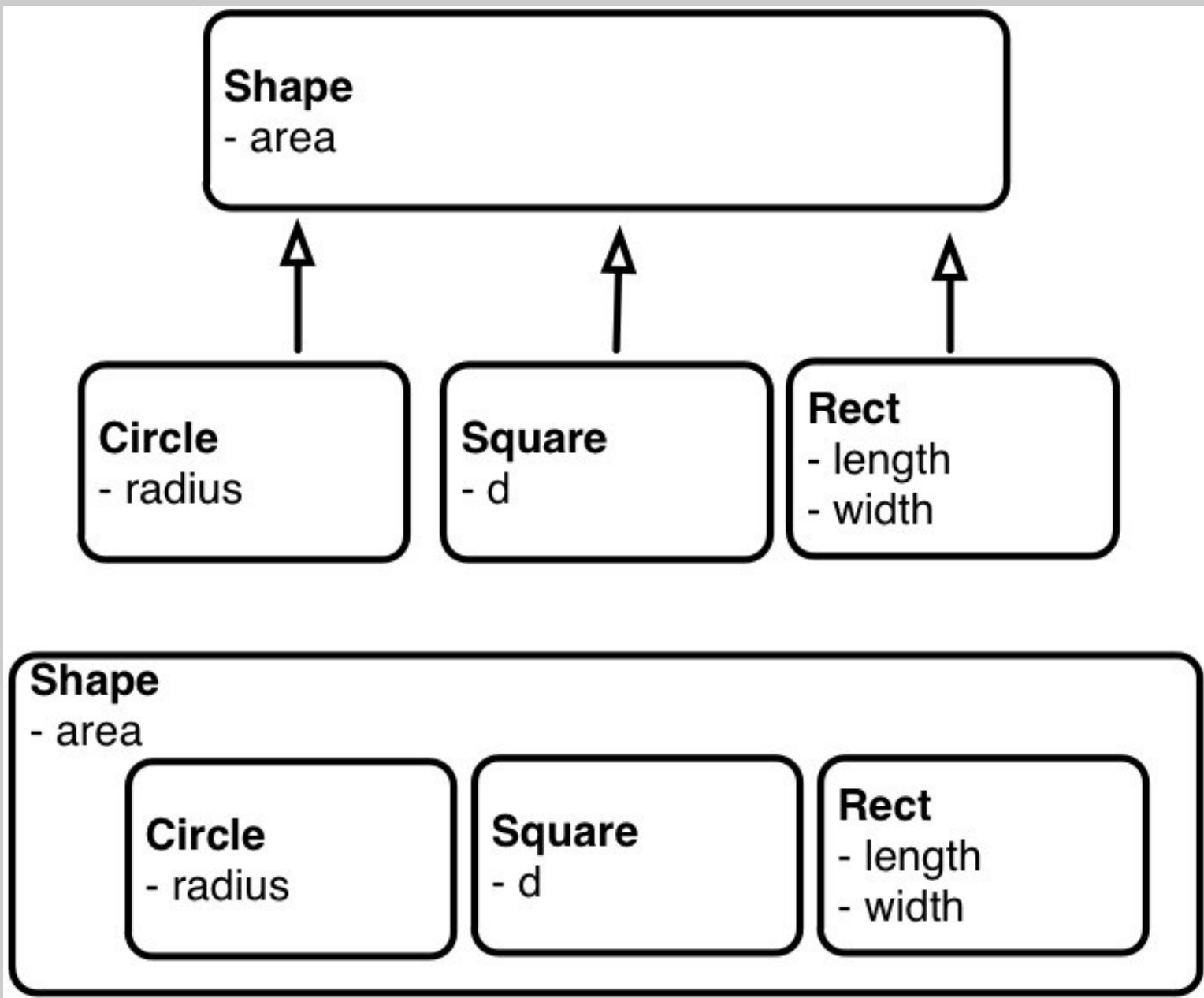
Schema considerations

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

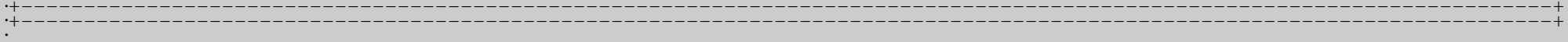
Considerations

- No Joins
- Document writes are atomic

Inheritance



Single table inheritance - RDBMS



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}
```

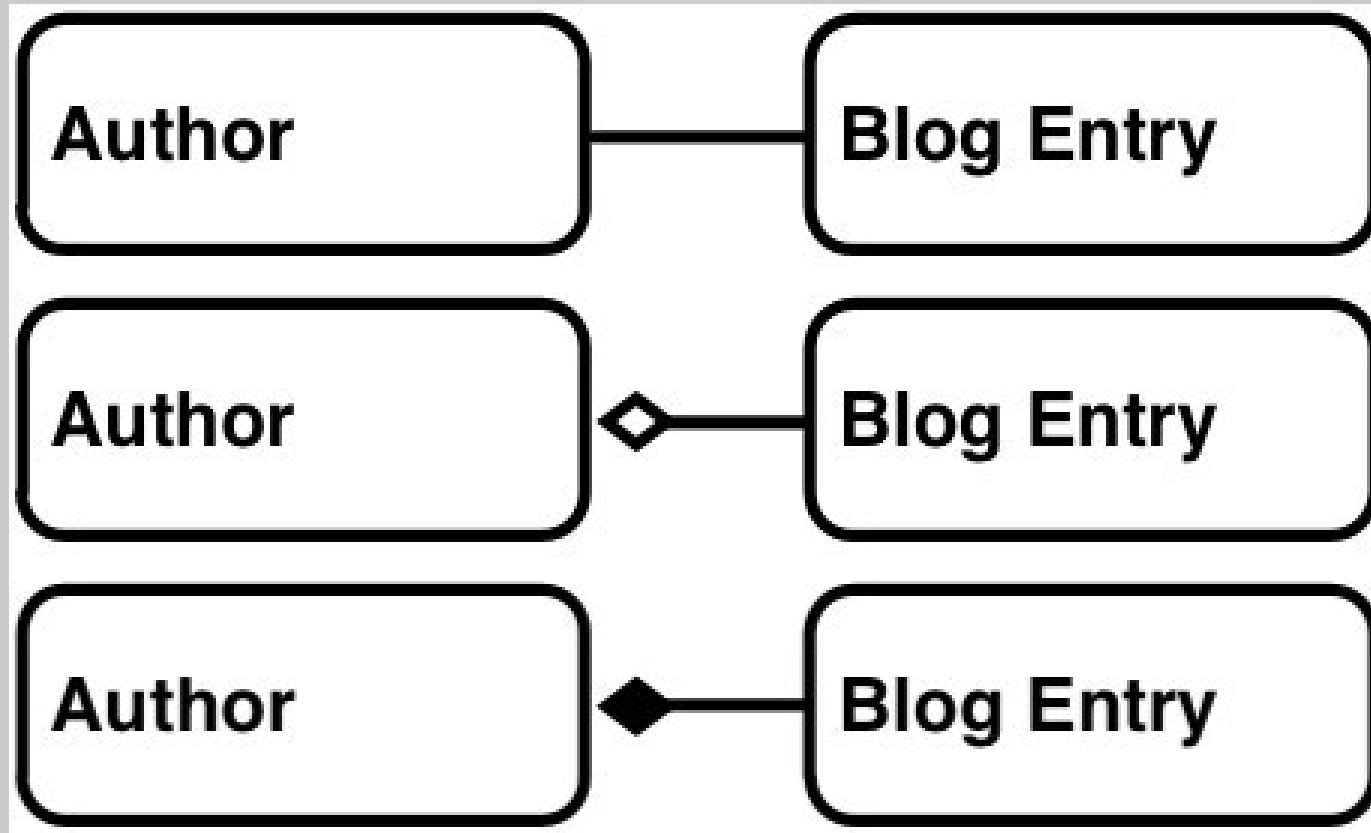
Inheritance

- 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



Embedded Array / Array Keys

- slice operator to return subset of array
- some queries harder e.g find latest comments across all documents

```
blogs: {
  author: "Hergé",
  date: "Tue Mar 28 2012 12:41:29 GMT",
  comments: [
    {
      author: "Kyle",
      date: "Tue Mar 28 2012 12:41:54 GMT",
      text: "great book"
    }
  ]
}
```


One to Many (1:n)

Embedded Tree

- single document
- natural
- hard to query

```
blogs: {
  author: "Hergé",
  date: "Tue Mar 28 2012 12:41:29 GMT",
  comments: [
    {
      author: "Kyle",
      date: "Tue Mar 28 2012 12:41:54 GMT",
      text: "great book"
      replies: [
        { author: "Derick", ... }
      ]
    }
  ]
}
```

One to Many (1:n)

Normalised (with two collections)

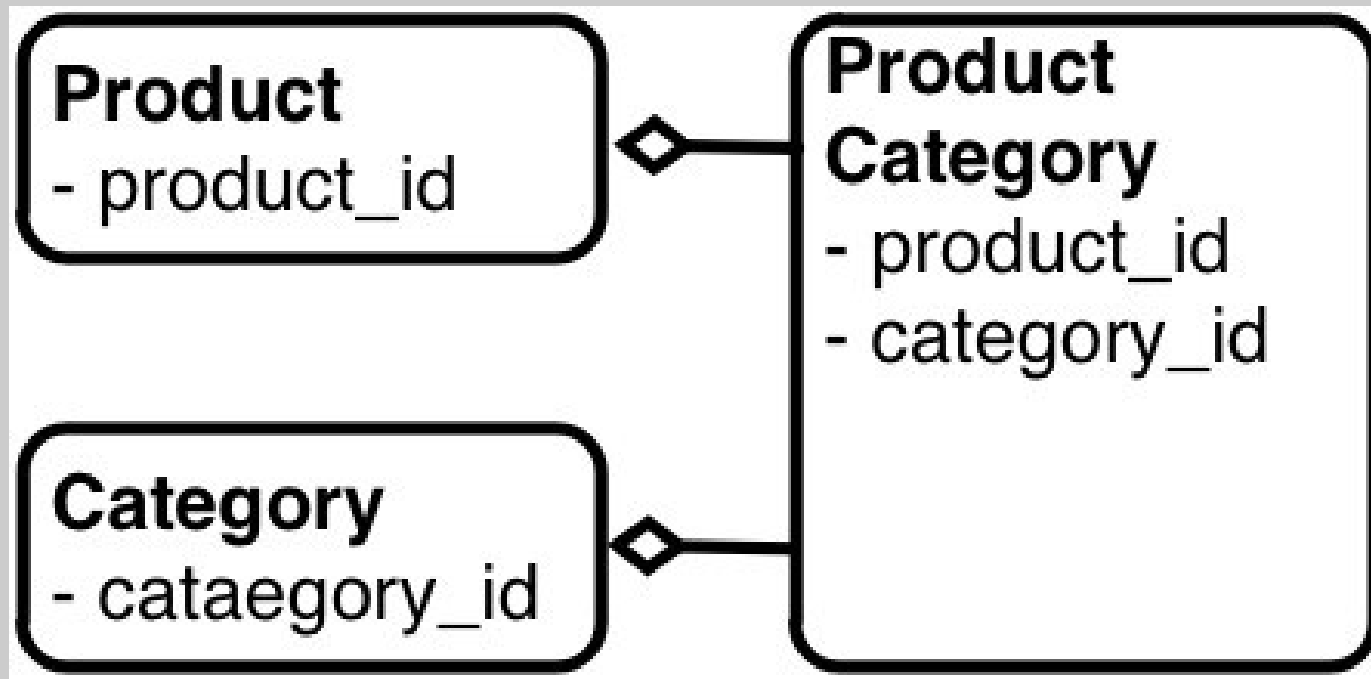
- most flexible
- more queries

```
blogs: {
  author: "Hergé",
  date: "Tue Mar 28 2012 12:41:29 GMT",
  comments: [
    { comment: "1" }
  ]
}
```

```
comments: {
  _id: "1",
  author: "Kyle",
  date: "Tue Mar 28 2012 12:41:54 GMT",
},
{
  _id: "2",
  author: "Derick",
  in_reply_to: "1",
}
```

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 elephant", category_ids: [ 4, 7 ] }
{ _id: 11, name: "Pink elephant", 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 elephant):

```
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 elephant", category_ids: [ 4, 7 ] }  
{ _id: 11, name: "Pink elephant", category_ids: [ 4, 8 ] }
```

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

All categories for a given product (pink elephant):

```
product = db.products.find( { category_ids: 4 } );  
db.categories.find( { _id: { $in: product.category_ids } } );
```

All products for a given category (toys):

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

Many to Many (n:m) - alternative 2

```
products:
{ _id: 10, name: "Blue elephpant" }
{ _id: 11, name: "Pink elephpant" }

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):

```
category = db.categories.find( { category_ids: 4 } );
db.products.find( { _id: { $in: category.product_ids } } );
```

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?

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 }  
  ]  
}
```

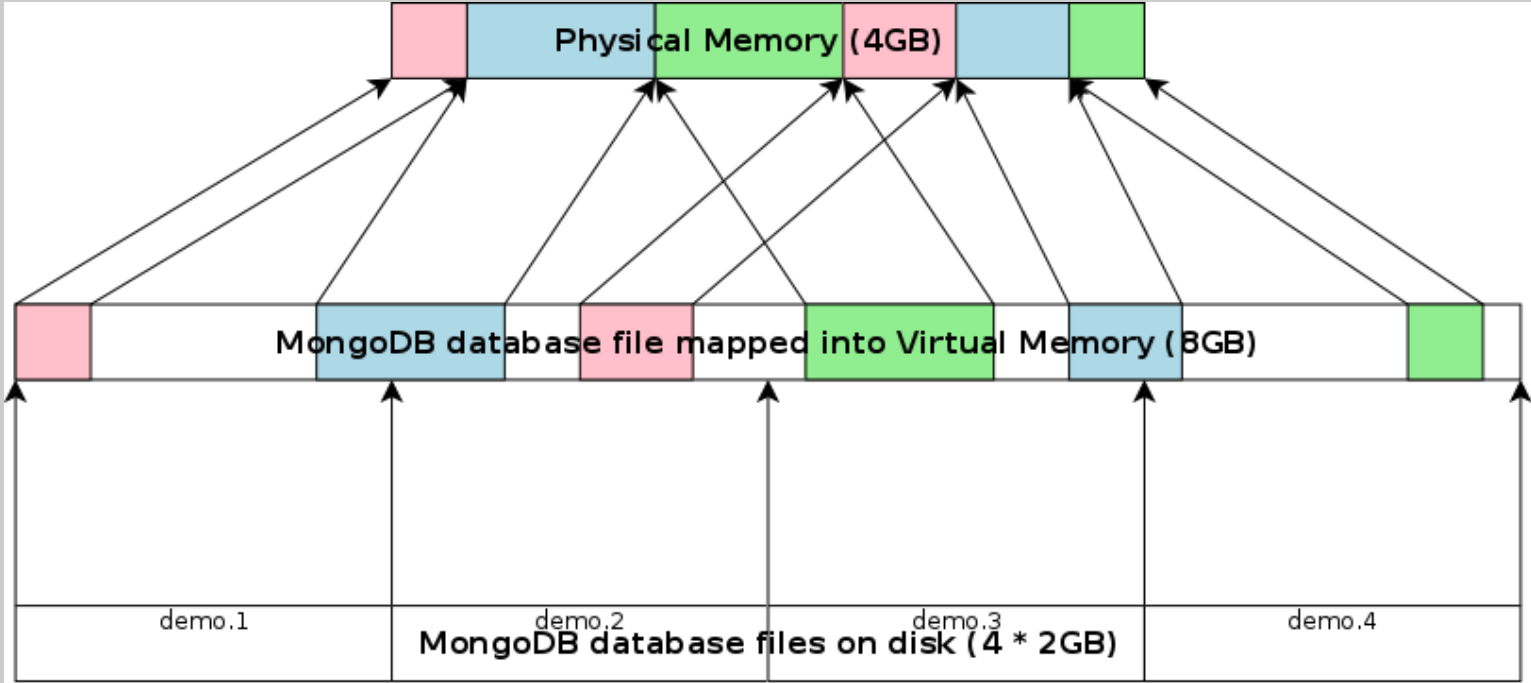
Define and document your keys!

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 write!" },
    { ts: 1332942118, author: "...", comment: "..." },
    ...
  ]
},
{
  _id: 42,
  page: 1,
  comments: [
    { ts: 1332942261, author: "Derick", comment: "blah blah" },
    { ts: 1332942482, author: "...", comment: "..." },
  ]
}
```

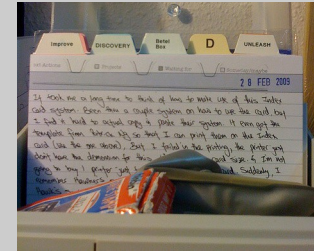
A bit more work when updating, but a lot easier to retrieve

How MongoDB stores data



Indexes

- Just like a relational database, MongoDB also benefits from indexes.
- Every collection has (automatically) an index on `_id`.
- Indexes can be on single or multiple fields.
- `MongoCursor->explain()`.



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

$c->insert( array( '_id' => 'ele1', 'name' => 'Jumbo' ) );
$c->insert( array( '_id' => 'ele2', 'name' => 'Tantor' ) );

var_dump( $c->find( [ '_id' => 'ele1' ] )->explain() );
?>
```

Indexes

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

Indexes

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

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' ) ] )
```

Helps you with finding POIs (pubs!) in a 2D space

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

$c->ensureIndex( array( 'l' => '2d' ) );
$c->insert([ 'name' => 'Betsy Smith', 'l' => [ -0.193, 51.537 ] ]);
$c->insert([ 'name' => 'London Tavern', 'l' => [ -0.202, 51.545 ] ]);

$closest = $m->demo->command( [
    'geoNear' => 'pubs',
    'near' => [ -0.198, 51.538 ],
    'spherical' => true,
] );

foreach ( $closest['results'] as $res ) {
    printf( "%s: %.2f km\n", $res['obj']['name'], $res['dis'] * 6378 );
}
?>
```

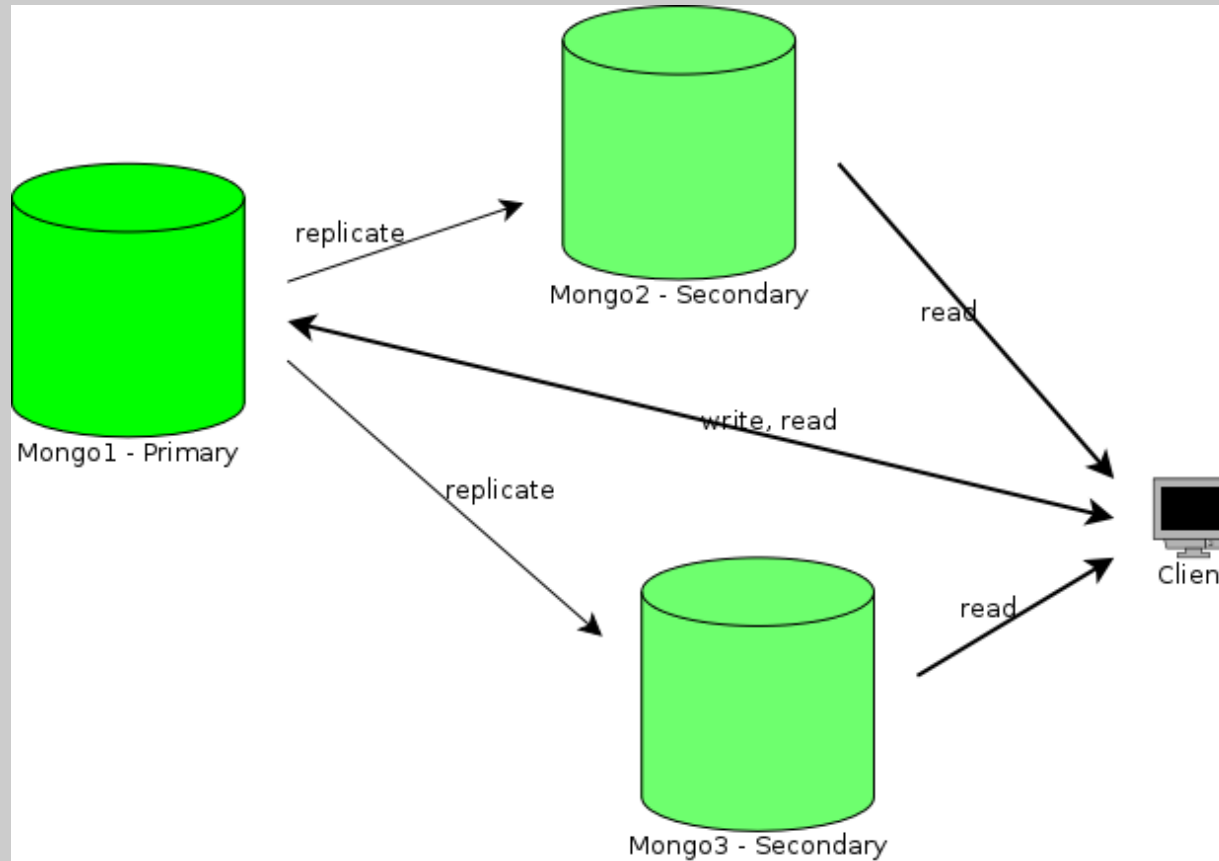
More about 2D indexes

A 2d geo index wants two elements. With the first being the longitude (x) and the second one being the latitude (y) (as in GeoJSON).

The following are all equivalent:

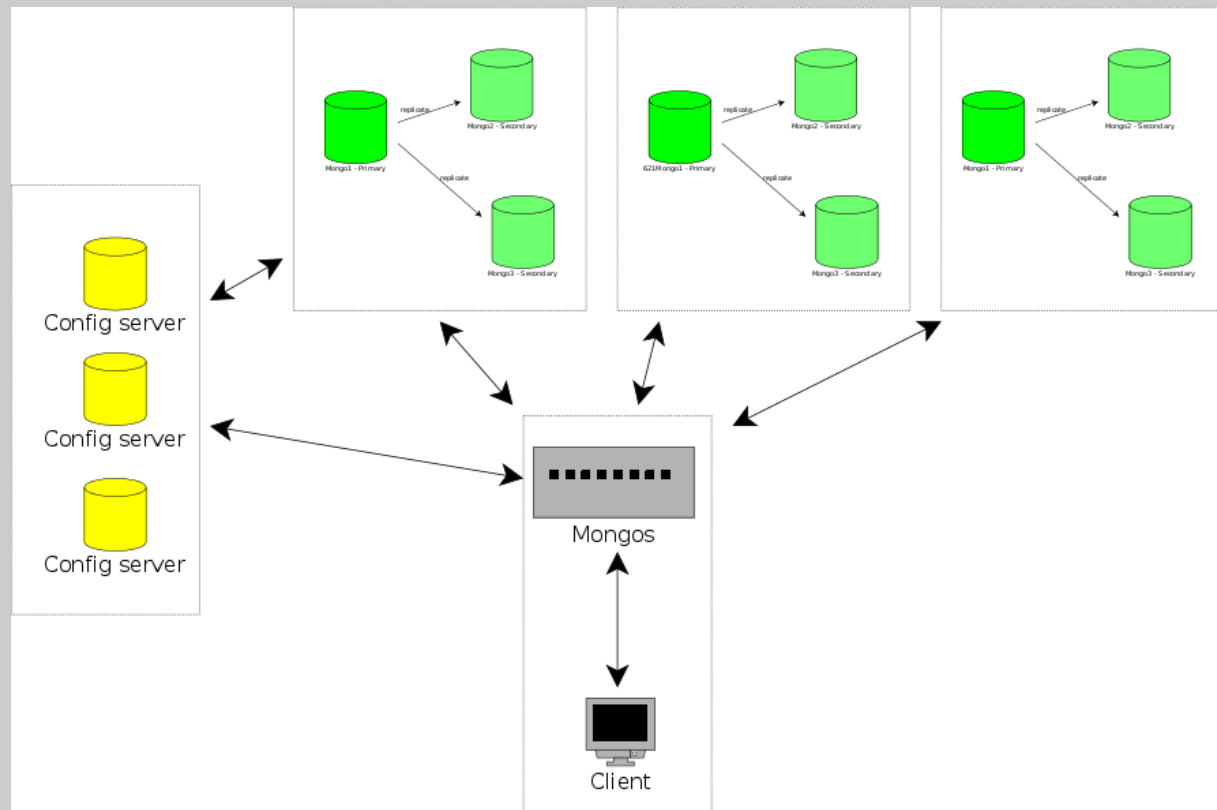
```
$myCol->insert( [
  _id: 42, loc: [ 6.43, 52.1233 ]
] );
$myCol->insert( [
  _id: 42, loc: { long: 6.43, lat: 52.1233 }
] );
$myCol->insert( [
  _id: 42, loc: { latitude: 6.43, longitude: 52.1233 }
] );
```


Replication



- Failover/Availability
- Scaling reads
- Primaries, secondaries and arbiters
- Odd number to prevent split brain

Sharding



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

Who uses MongoDB?



MongoDB UK

- Annual one day conference dedicated to the open source database MongoDB.
- June 20th, 2012 at the Mermaid Conference & Events Centre
- Early Bird (\$50) ends May 23rd

Mongo UGs

- MongoDB London (Apr 24th): bi-monthly
- Office Hours (Apr 4th): bi-weekly near Old Street

- Slides: http://derickrethans.nl/talks/::-:talk_id::-:
- Contact me: Derick Rethans: @derickr, derick@10gen.com
- Feedback: e-mail me

