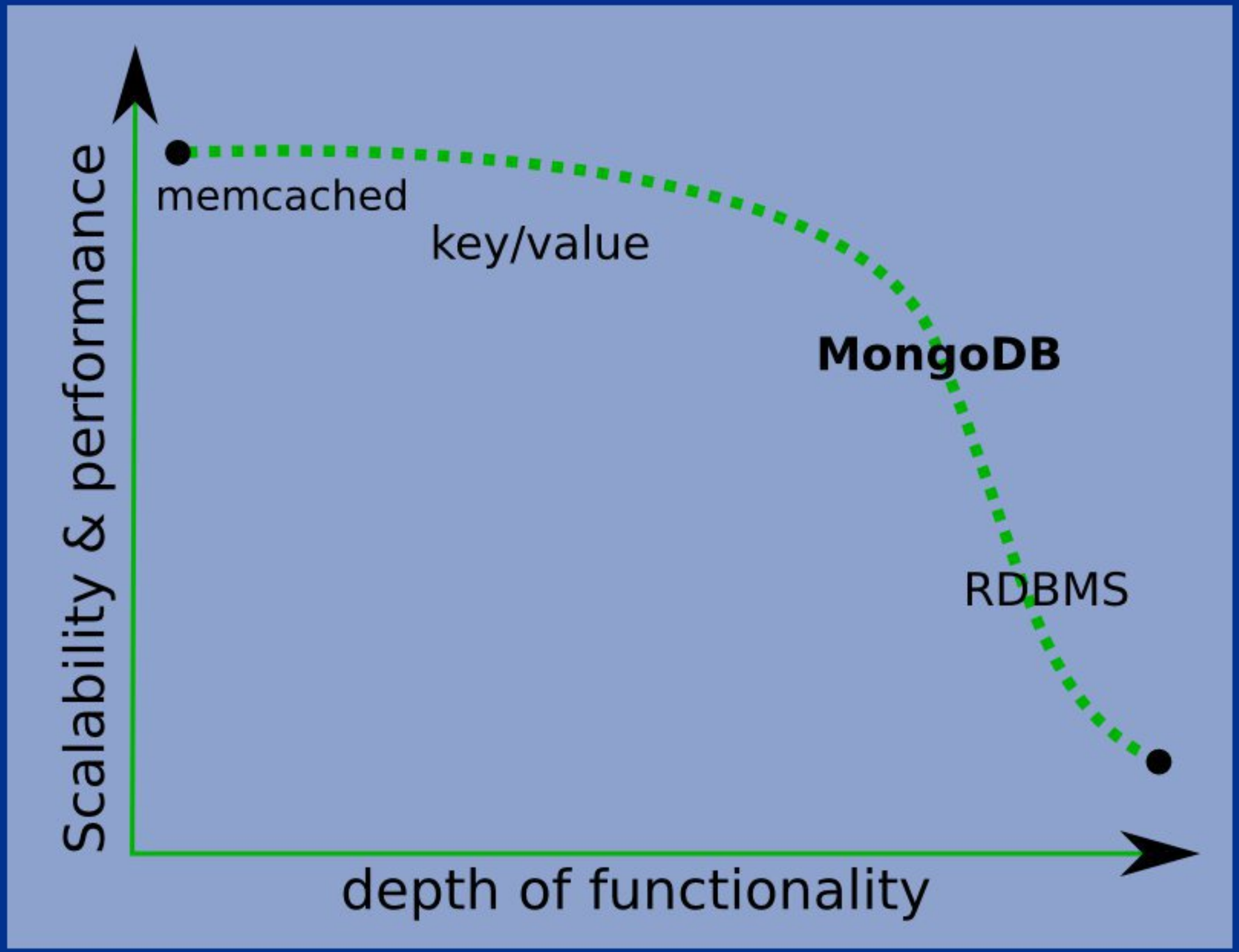
MongoDB Introduction

Derick Rethans - derick@10gen.com
@derickr
http://joind.in/7866

10gen | the MongoDB company

mongoDB

Database landscape



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

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",
    }
  ]
}
```

Connecting to MongoDB

No databases or collections have to do be explicitly created:

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

Different connection strings:

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

Inserting a document

```
<?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 MongoClient();
var_dump( $m->demo->talks->insert( $document ) );
?>
```

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
$m = new MongoClient();
$c = $m->demo->articles;

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

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

Querying (one document)

```php
<?php
$m = new MongoClient();
// 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 (with projection)

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

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

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 MongoClient();
$c = $m->demo->talks;

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

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

Result:

```
derickr
```

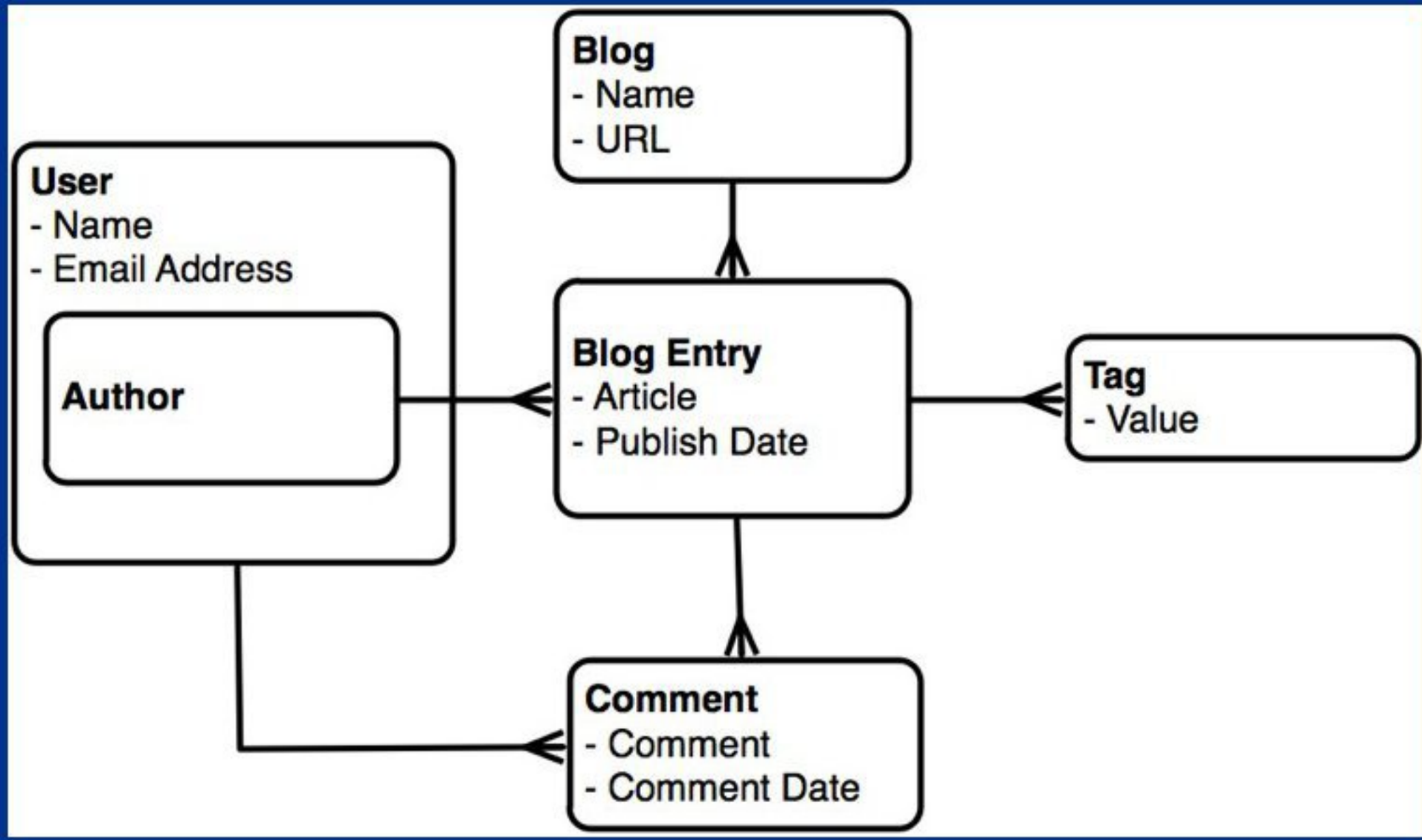
Cursor methods

```php
<?php
$m = new MongoClient();
$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'] );
}
?>
```

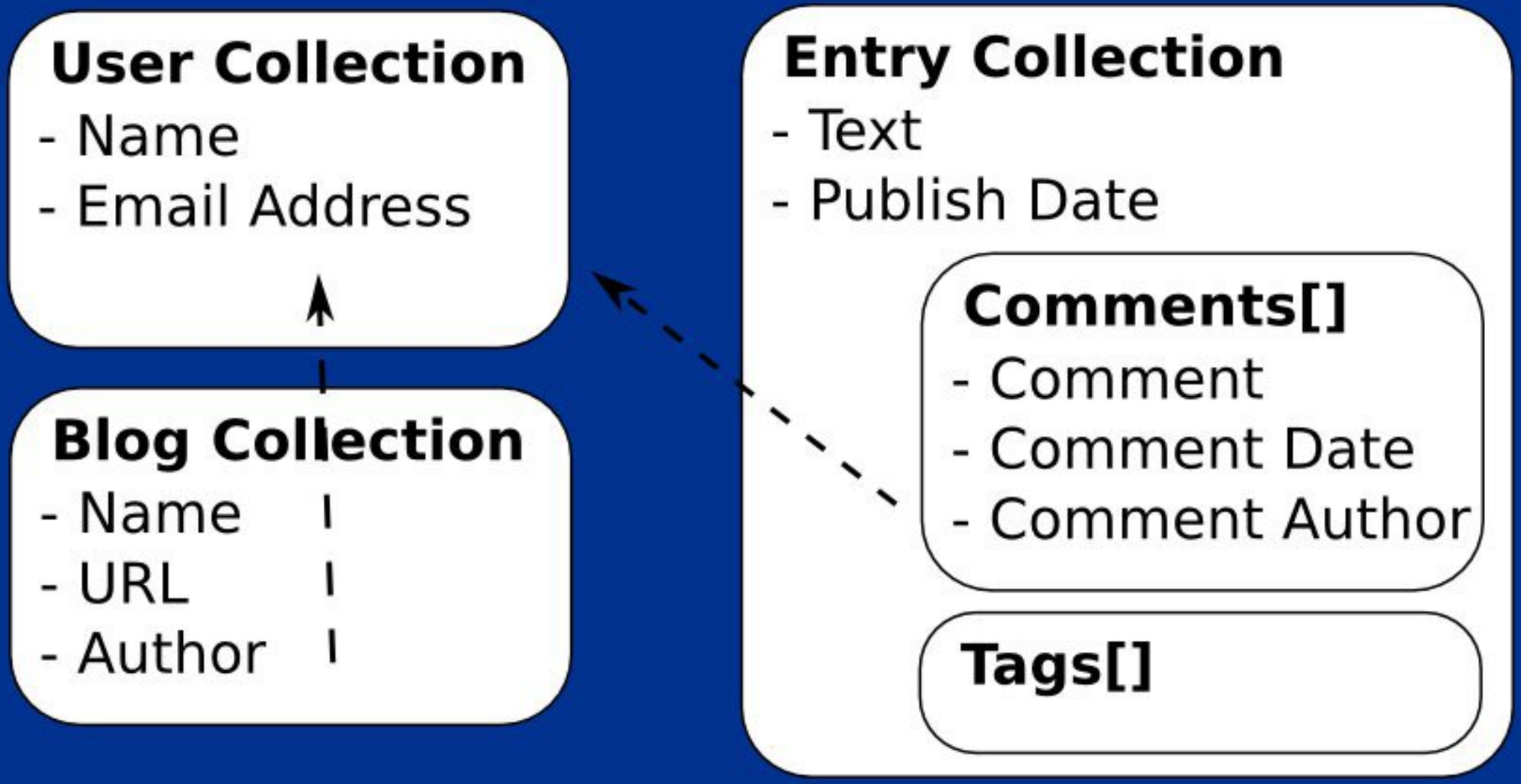
Result:

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

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

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}
```

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

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

```
{
    '_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

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" },
  ]
}
```

Linking Articles and Views

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" },
```

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?

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

```
…
{
        "_id" : "n1996641853",
        "tags" : [
                "amenity=restaurant",
                "cuisine=japanese",
                "name=Kaiseki"
        ]
}
{
        "_id" : "n1618702957",
        "tags" : [
                "amenity=fast_food",
                "cuisine=kebab",
                "name=Pita Durum"
        ]
}
…
```

We want all different cuisines, with their usage count

Aggregation example

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

$res = $m->demo->poiConcat->aggregate( [
    /* First we match all documents that have a cuisine tag: */
    [ '$match' => [ 'tags' => new MongoRegex( '/^cuisine=/' ) ] ],
] );

var_dump( $res['result'] );
?>
```

Result:

```
array (size=219)
  0 =>
    array (size=4)
      '_id' => string 'n75838522' (length=9)
      'type' => int 1
      'loc' =>
        array (size=2)
          0 => float 1.0853358
          1 => float 51.2781096
      'tags' =>
        array (size=6)
          0 => string 'amenity=restaurant' (length=18)
          1 => string 'cuisine=arab;north_african' (length=26)
          2 => string 'name=Azouma' (length=11)
          3 => string 'postal_code=CT1 1NH' (length=19)
```

Aggregation example

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

$res = $m->demo->poiConcat->aggregate( [
    [ '$match' => [ 'tags' => new MongoRegex( '/^cuisine=/' ) ] ],
    /* Then we unwind all the tags */
    [ '$unwind' => '$tags' ],
] );

var_dump( $res['result'] );
?>
```

Result:

```
array (size=960)
  0 =>
    array (size=4)
      '_id' => string 'n75838522' (length=9)
      'type' => int 1
      'loc' =>
        array (size=2)
          0 => float 1.0853358
          1 => float 51.2781096
      'tags' => string 'amenity=restaurant' (length=18)
  1 =>
    array (size=4)
      '_id' => string 'n75838522' (length=9)
      'type' => int 1
```

Aggregation example

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

$res = $m->demo->poiConcat->aggregate( [
    [ '$match' => [ 'tags' => new MongoRegex( '/^cuisine=/' ) ] ],
    [ '$unwind' => '$tags' ],
    /* Match again to get rid of all the useless tags */
    [ '$match' => [ 'tags' => new MongoRegex( '/^cuisine=/' ) ] ],
] );

var_dump( $res['result'] );
?>
```

Result:

```
array (size=227)
  0 =>
    array (size=4)
      '_id' => string 'n75838522' (length=9)
      'type' => int 1
      'loc' =>
        array (size=2)
          0 => float 1.0853358
          1 => float 51.2781096
      'tags' => string 'cuisine=arab;north_african' (length=26)
  1 =>
    array (size=4)
      '_id' => string 'n31315473' (length=9)
```

Aggregation example

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

$res = $m->demo->poiConcat->aggregate( [
    [ '$match' => [ 'tags' => new MongoRegex( '/^cuisine=/' ) ] ],
    [ '$unwind' => '$tags' ],
    [ '$match' => [ 'tags' => new MongoRegex( '/^cuisine=/' ) ] ],
    /* Now we group on the tags */
    [ '$group' => [ '_id' => '$tags' ] ],
] );

var_dump( $res['result'] );
?>
```

Result:

```
array (size=42)
  0 =>
    array (size=1)
      '_id' => string 'cuisine=vegetarian' (length=18)
  1 =>
    array (size=1)
      '_id' => string 'cuisine=vegan' (length=13)
  2 =>
    array (size=1)
      '_id' => string 'cuisine=tea_shop' (length=16)
  3 =>
    array (size=1)
```

Aggregation example

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

$res = $m->demo->poiConcat->aggregate( [
    [ '$match' => [ 'tags' => new MongoRegex( '/^cuisine=/' ) ] ],
    [ '$unwind' => '$tags' ],
    [ '$match' => [ 'tags' => new MongoRegex( '/^cuisine=/' ) ] ],
    /* Now we group on the tags, and add the count */
    [ '$group' => [ '_id' => '$tags', 'count' => [ '$sum' => 1 ] ] ],
] );

var_dump( $res['result'] );
?>
```

Result:

```
array (size=43)
  0 =>
    array (size=2)
      '_id' => string 'cuisine=vegetarian' (length=18)
      'count' => int 1
  1 =>
    array (size=2)
      '_id' => string 'cuisine=vegan' (length=13)
      'count' => int 1
  2 =>
    array (size=2)
      '_id' => string 'cuisine=tea_shop' (length=16)
```

Aggregation example

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

$res = $m->demo->poiConcat->aggregate( [
    [ '$match' => [ 'tags' => new MongoRegex( '/^cuisine=/' ) ] ],
    [ '$unwind' => '$tags' ],
    [ '$match' => [ 'tags' => new MongoRegex( '/^cuisine=/' ) ] ],
    [ '$group' => [ '_id' => '$tags', 'count' => [ '$sum' => 1 ] ] ],
    /* and at last we sort by count */
    [ '$sort' => [ 'count' => -1 ] ],
] );

var_dump( $res['result'] );
?>
```

Result:

```
array (size=43)
  0 =>
    array (size=2)
      '_id' => string 'cuisine=indian' (length=14)
      'count' => int 54
  1 =>
    array (size=2)
      '_id' => string 'cuisine=burger' (length=14)
      'count' => int 37
  2 =>
    array (size=2)
```

Indexes are the single biggest tunable performance factor in MongoDB.

Creating Indexes

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

```
<?php
$m = new MongoClient;
$c = $m->demo->cities;

// Create an index on the name attribute
$c->ensureIndex( array( 'name' => 1 ) );

// Create a compound index on country code and feature code
$c->ensureIndex( array( 'country_code' => 1, 'feature_code' => 1 ) );

// Create a unique index on the area field of timezone:
$c->ensureIndex( array( 'timezone.area' => 1 ) );
```

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 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 }
```

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 uses indexes in order from left to right

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)

Indexes are used from left to right.

```
$c->ensureIndex( [ 'country_code' => 1, 'timezone' => 1, 'asciiname' => 1 ] )
```

✔ `$c->find(['country_code' => 'GB']);`

✔ `$c->find(['country_code' => 'GB', 'timezone' => 'Europe/London']);`

✔ `$c->find(['country_code' => 'GB']).sort(['timezone' => 1]);`

✘ `$c->find(['country_code' => 'GB', 'asciiname' => 'London']);`

✘ `$c->find(['country_code' => 'GB']).sort(['asciiname' => 1]);`

✘ `$c->find(['timezone' => 'Europe/London']);`

✘ `$c->find(['asciiname' => 'Europe/London']);`

✔ `$c->find(['country_code' => 'GB']).sort(['timezone' => 1, 'asciiname' => 1]);`

✘ `$c->find(['country_code' => 'GB']).sort(['timezone' => 1, 'asciiname' => -1]);`

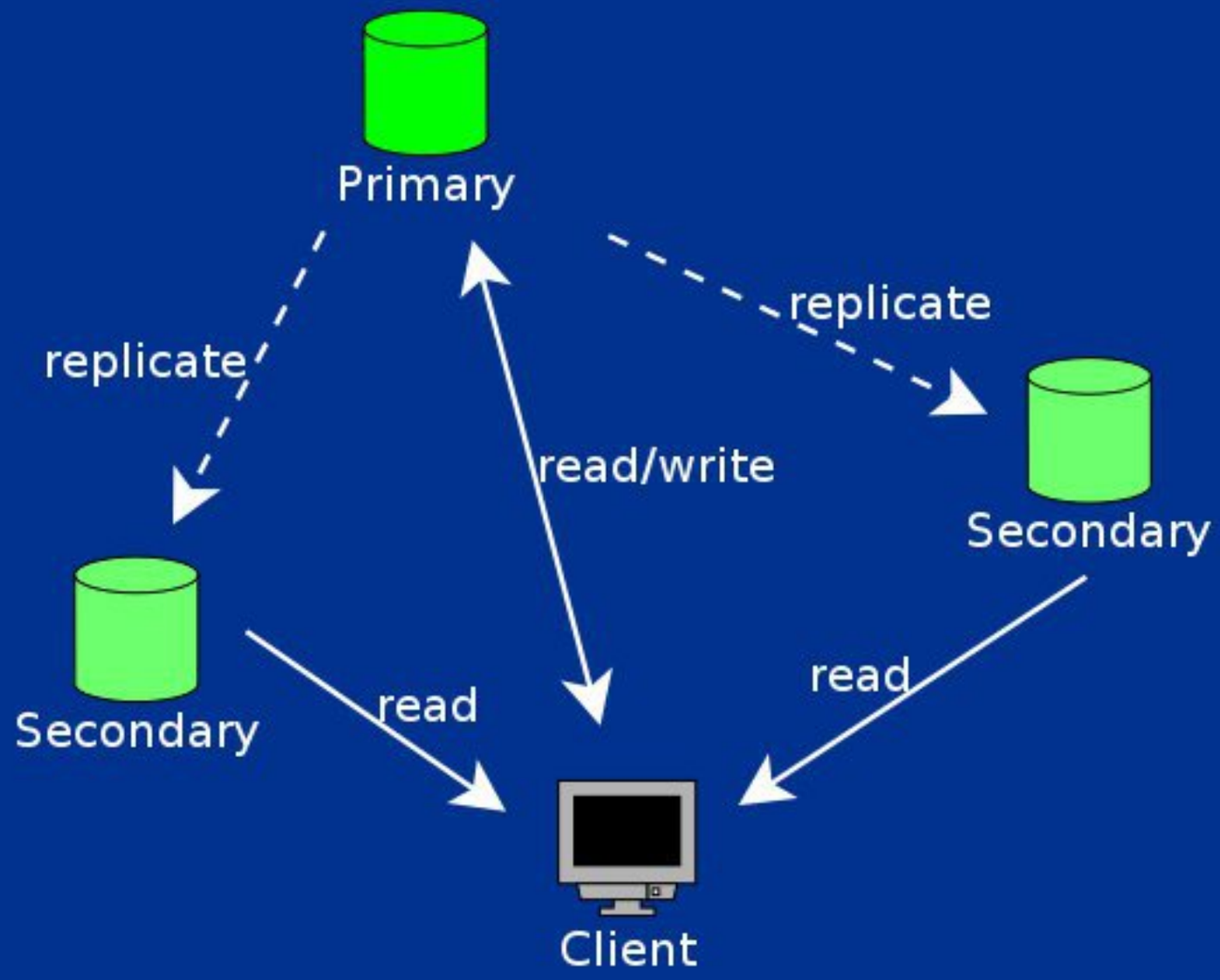
✘ `$c->find(['country_code' => 'GB']).sort(['timezone' => -1, 'asciiname' => 1]);`

✔ `$c->find(['country_code' => 'GB']).sort(['timezone' => -1, 'asciiname' => -1]);`

When can indexes not be used?

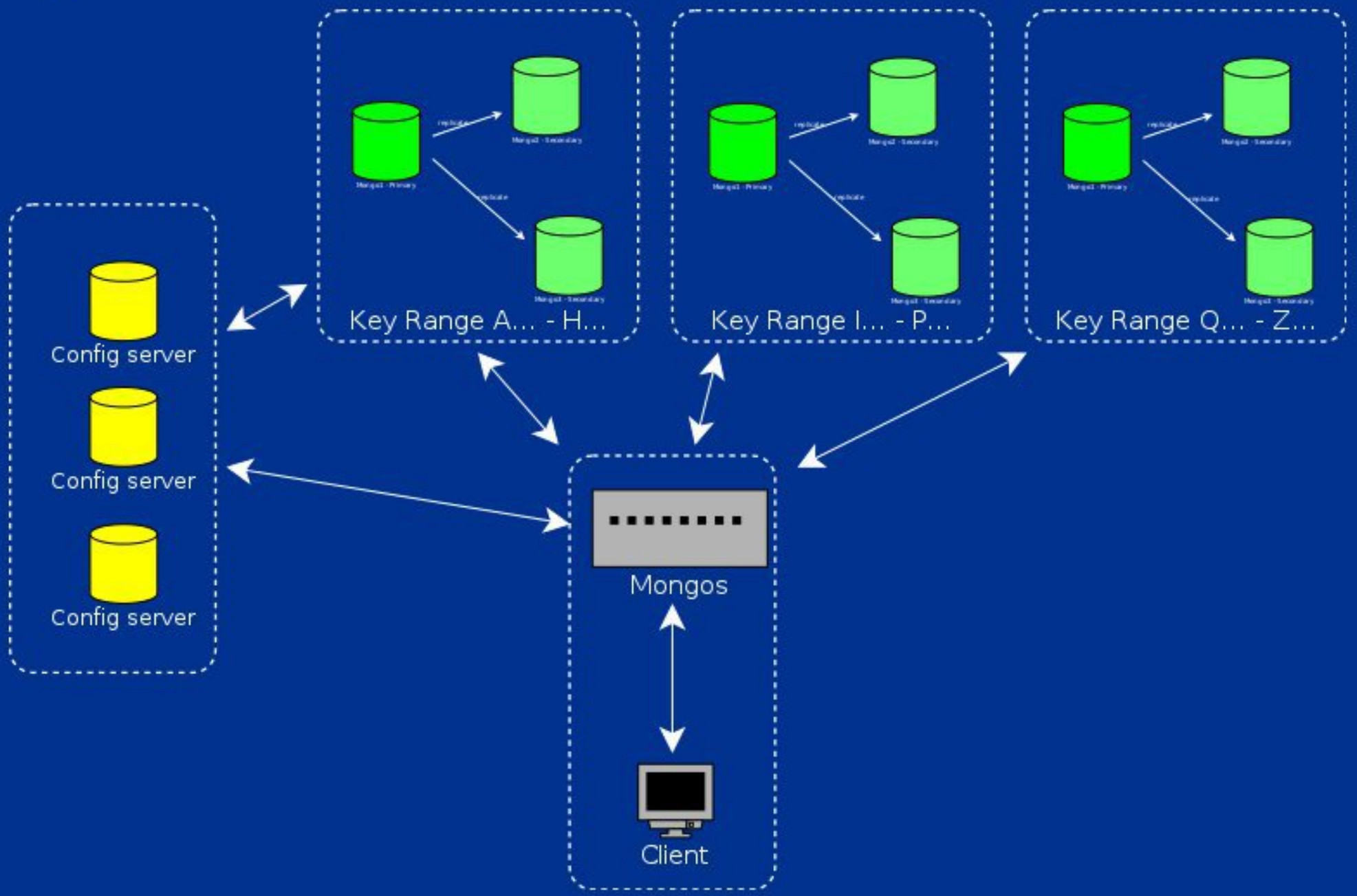
- Many sorts of negations (`$ne`, `$not`, `$nin`).
- Tricky aritmethic (`$mod`).
- Most regular expressions (`/ondo/`), unless anchored to the start of the string (`/^Lon/`).
- JavaScript expressions in `$where` clauses, such as `where dem > population`. In that case, try to pre-compute.
- map/reduce

Replication



- Failover/High Availability
- Scaling reads
- Primaries, secondaries and arbiters

Sharding



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

Wrap-up

- MongoDB is a document store
- Schemaless doesn't mean "no design necessary"
- It's fast and easy to scale
- No knobs

Resources

- Slides and Feedback: http://joind.in/7866
- Contact me: Derick Rethans: @derickr, derick@10gen.com

10gen | the MongoDB company

mongoDB