Intro

Q. *Why do we love Ruby?*

A.

"**Ruby** is designed to be **human-oriented**.
It **reduces the burden** of programming.
It tries to push jobs back to machines. You can accomplish **more tasks** with less work, in smaller yet **readable code**."

- Matz

01001000011101010110
11010110000101101110
0111011001110011
01001101011000010110
00110110100001101001
0110111001100101

Human

vs

Machine

Q. *Why do I love MongoDB?*

A.

“**MongoDB** is designed to be **human-oriented**.
It **reduces the burden of programming**.
It tries to push jobs back to machines. You can accomplish **more tasks** with less work, in smaller yet **readable code**.”

- Banker (indebted to Matz)

MongoDB (is) for Rubyists*

* *and all human-oriented programmers*

1. SQL (or No)

Key/Value Stores

Dynamo, Voldemort, Redis, Memcached (keyword: “stores”)

Column-Oriented

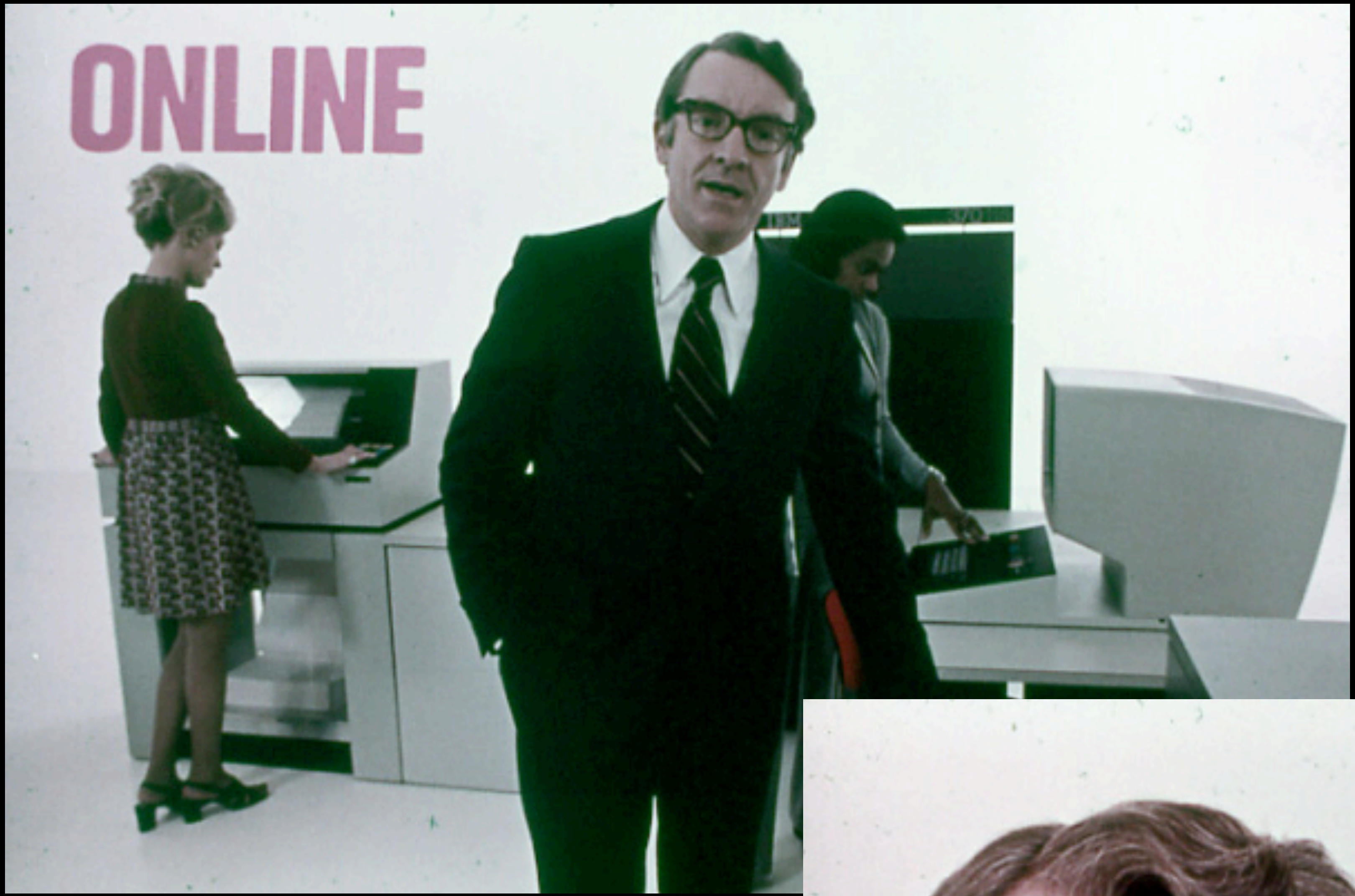
Cassandra, BigTable

Document Databases

MongoDB, CouchDB

RDBMS

- Transactional
- Normalized
- Rigorous
- Queryable (deeply)


ONLINE

Data Base

Why build MongoDB?

Fast & Queryable

Key-Value

MongoDB

Relational

MongoDB

open-source

high-performance

built for scale

document-oriented

schema-free

high performance

relaxed acid

document model

memory-mapped

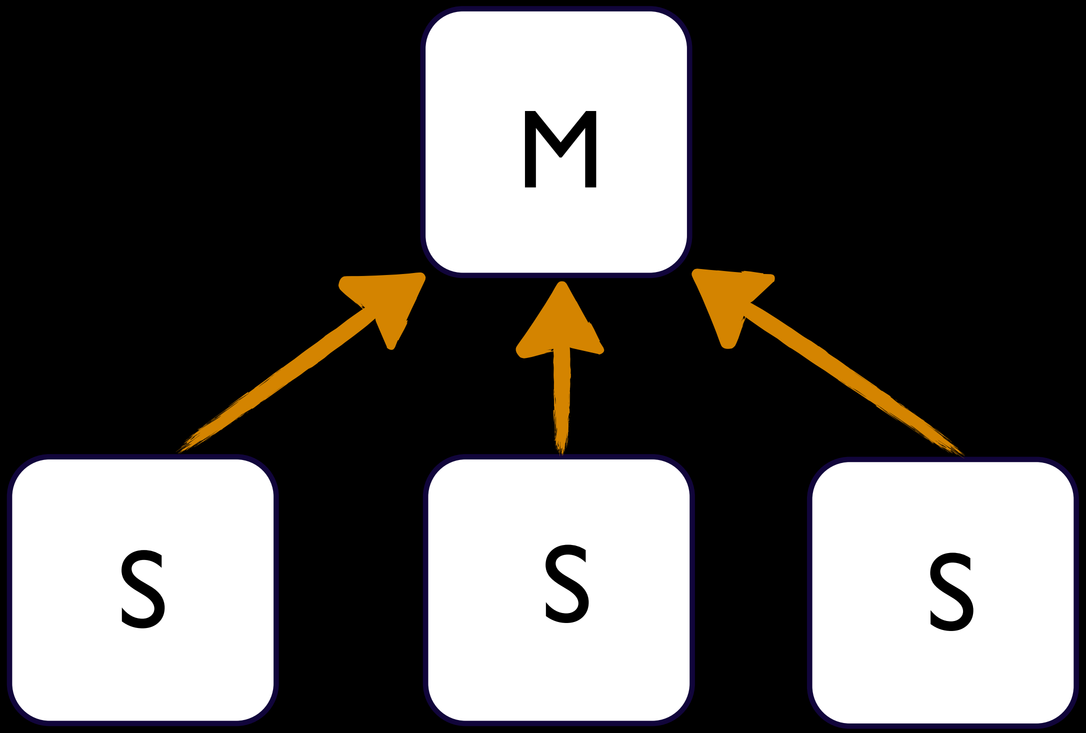
built for scale

master-slave

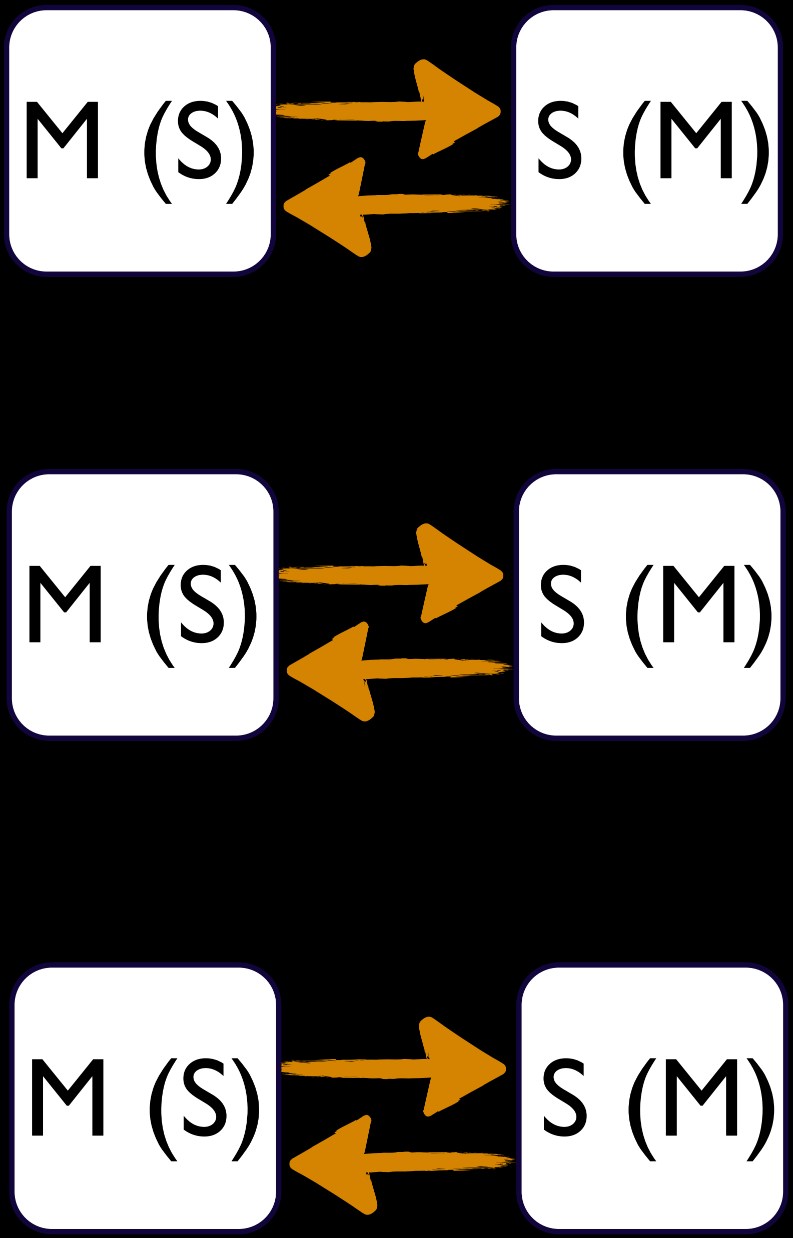
replica pairs

sharding

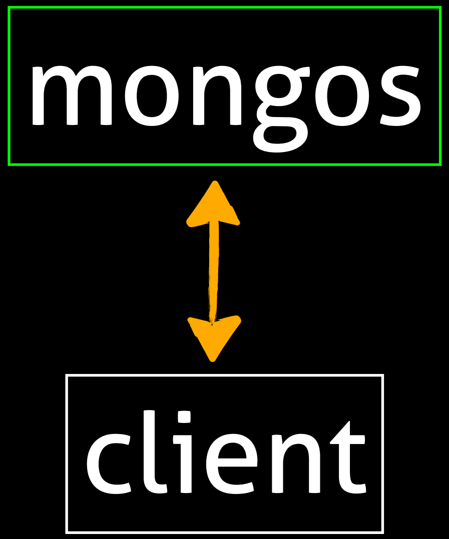
master-slave



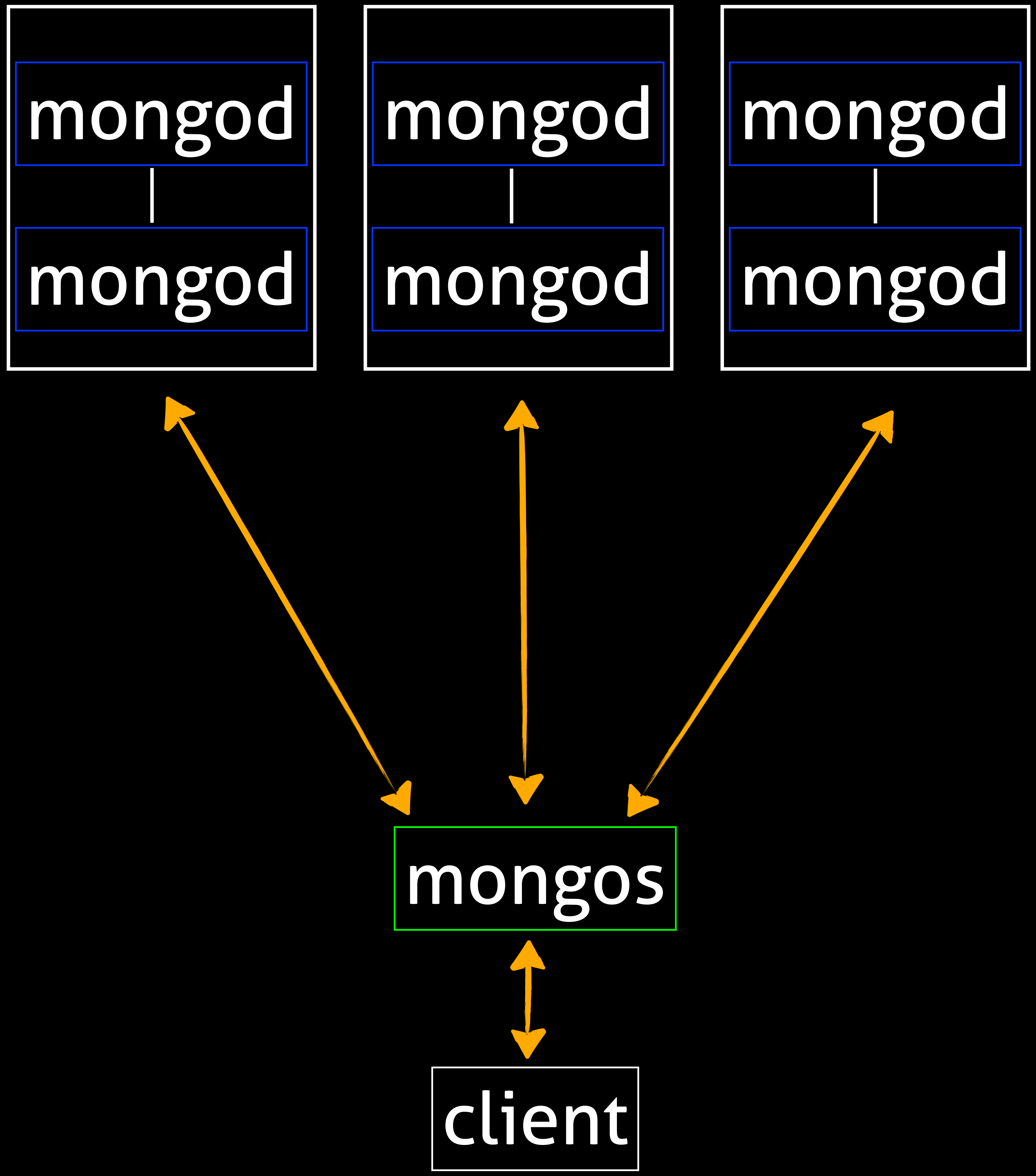
replica-pairs

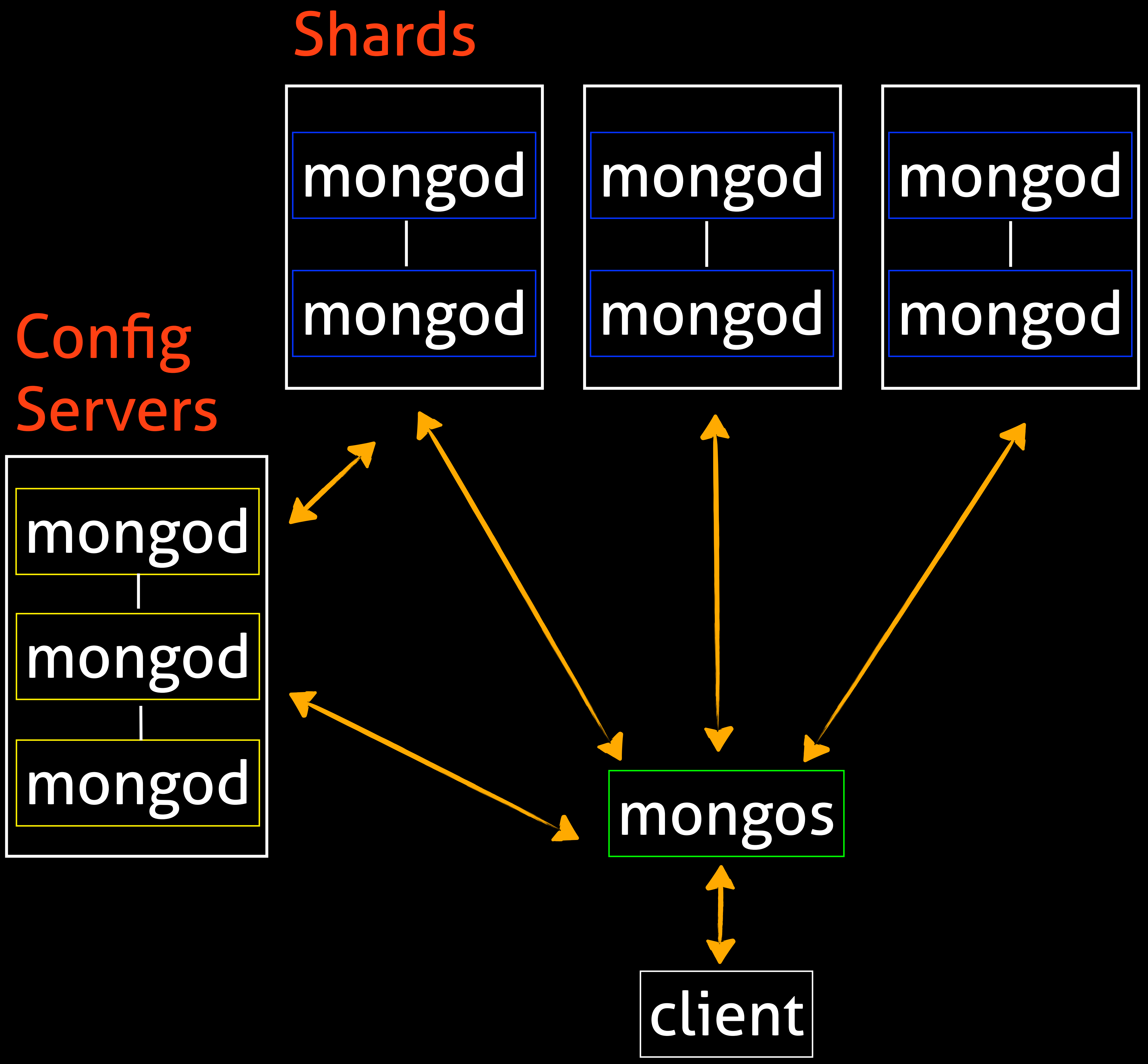


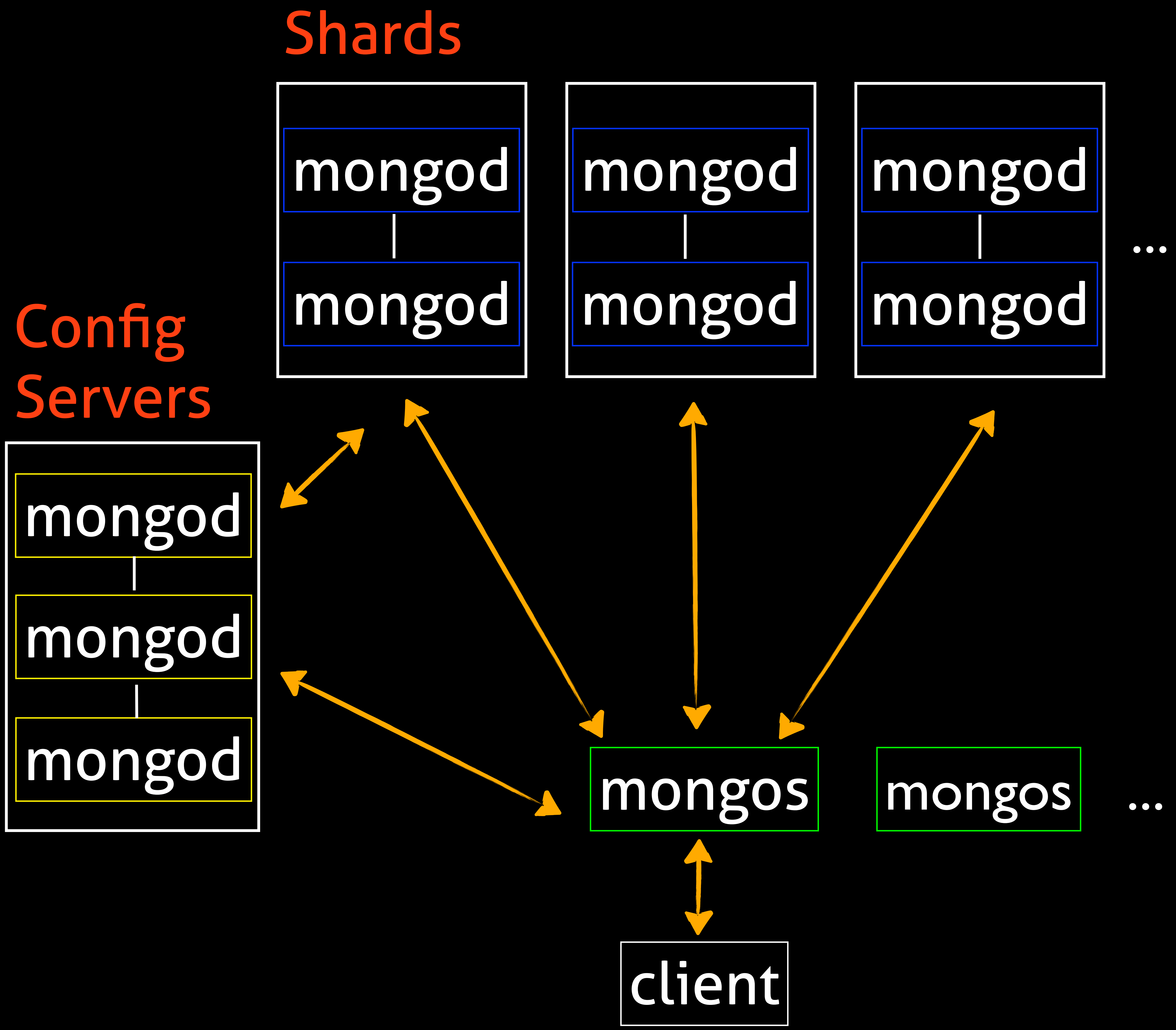
client


mongos
client

Shards




Shards
mongod
mongod
mongod
mongod
mongod
mongod
Config Servers
mongod
mongod
mongod
mongos
client


Shards
mongod
mongod
mongod
mongod
mongod
mongod
...
Config Servers
mongod
mongod
mongod
mongos
mongos
...
client

document-oriented

```
{:sku    => ‘637636’,
 :name => ‘Linen tailored pant’,
 :about => [{:title => ‘fabric & care’,
                   :content => [‘Dry clean’,
                                     ‘Imported’]},
                  {:title => ‘overview’,
                   :content => [‘Tailored fit’,
                                     ‘Yarn dyed’]}
                 ]
}
```

schema-free

```
{:sku    => ‘637636’,
 :note  => ‘Added this with no migration!”
 :name => ‘Linen tailored pant’,
 :about => [{:title => ‘fabric & care’,
                  :content => [‘Dry clean’,
                                    ‘Imported’]},
                 {:title => ‘overview’,
                  :content => [‘Tailored fit’,
                                    ‘Yarn dyed’]}
                ]
```

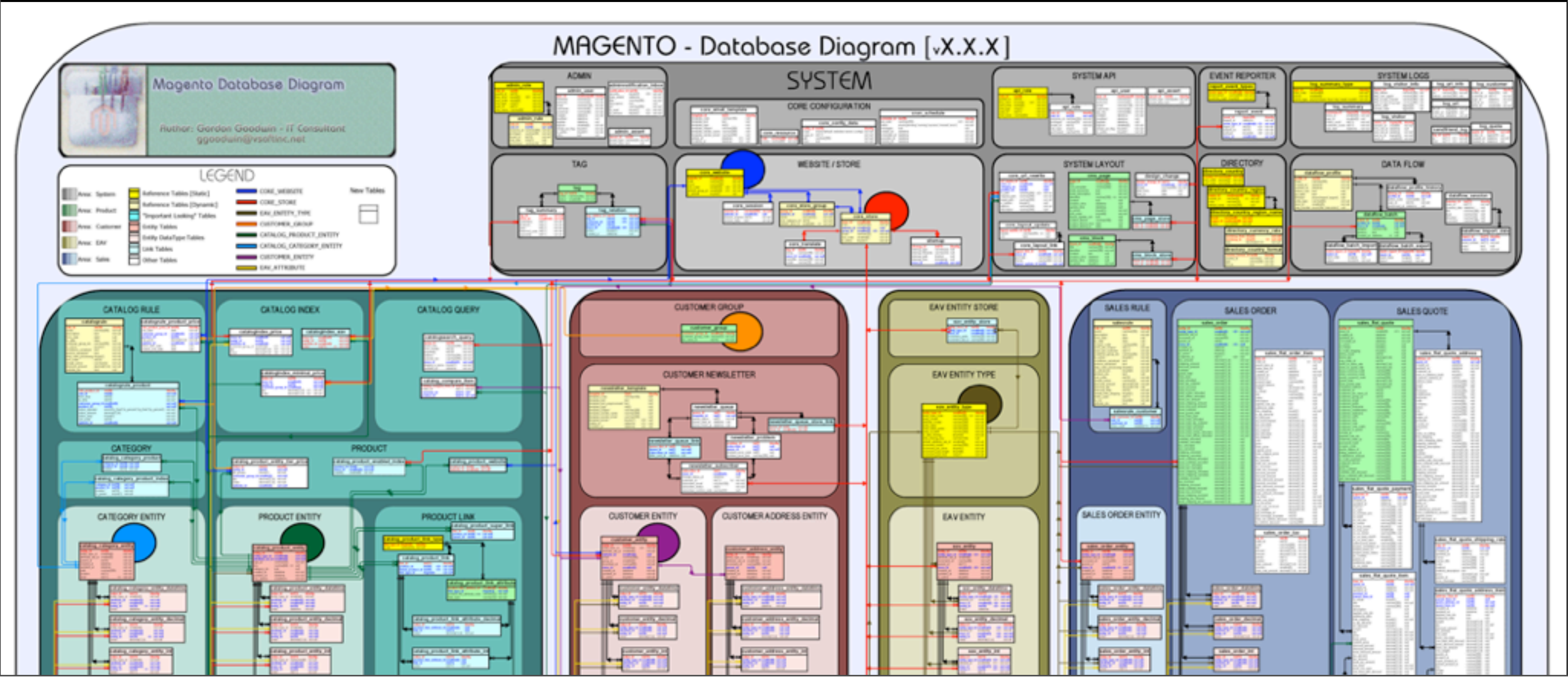
Q. *How many times have you typed this?*

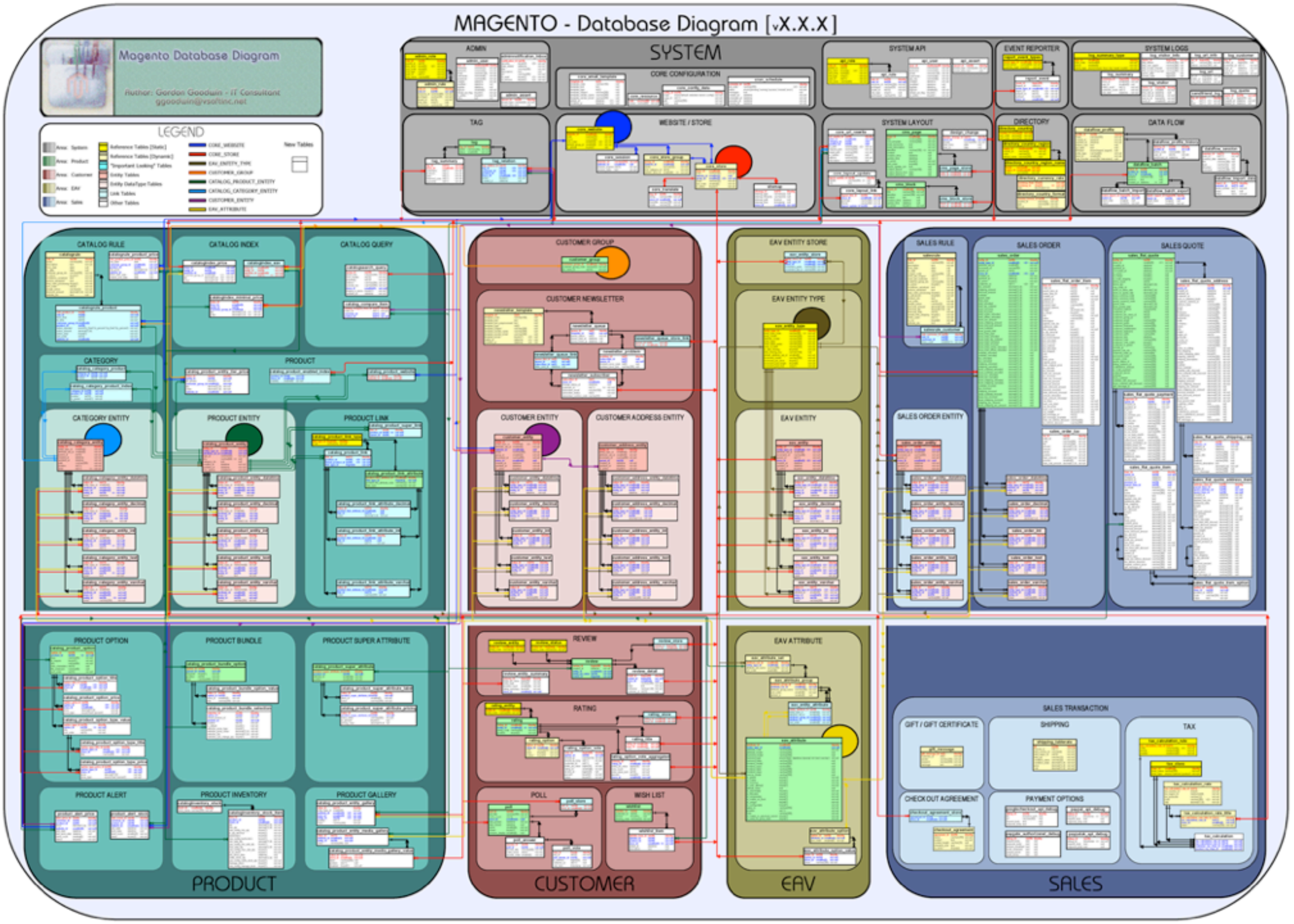
Q. *How many times have you typed this?*

```
rake db:migrate
```

Q. *How do you make an RDBMS dynamic?*

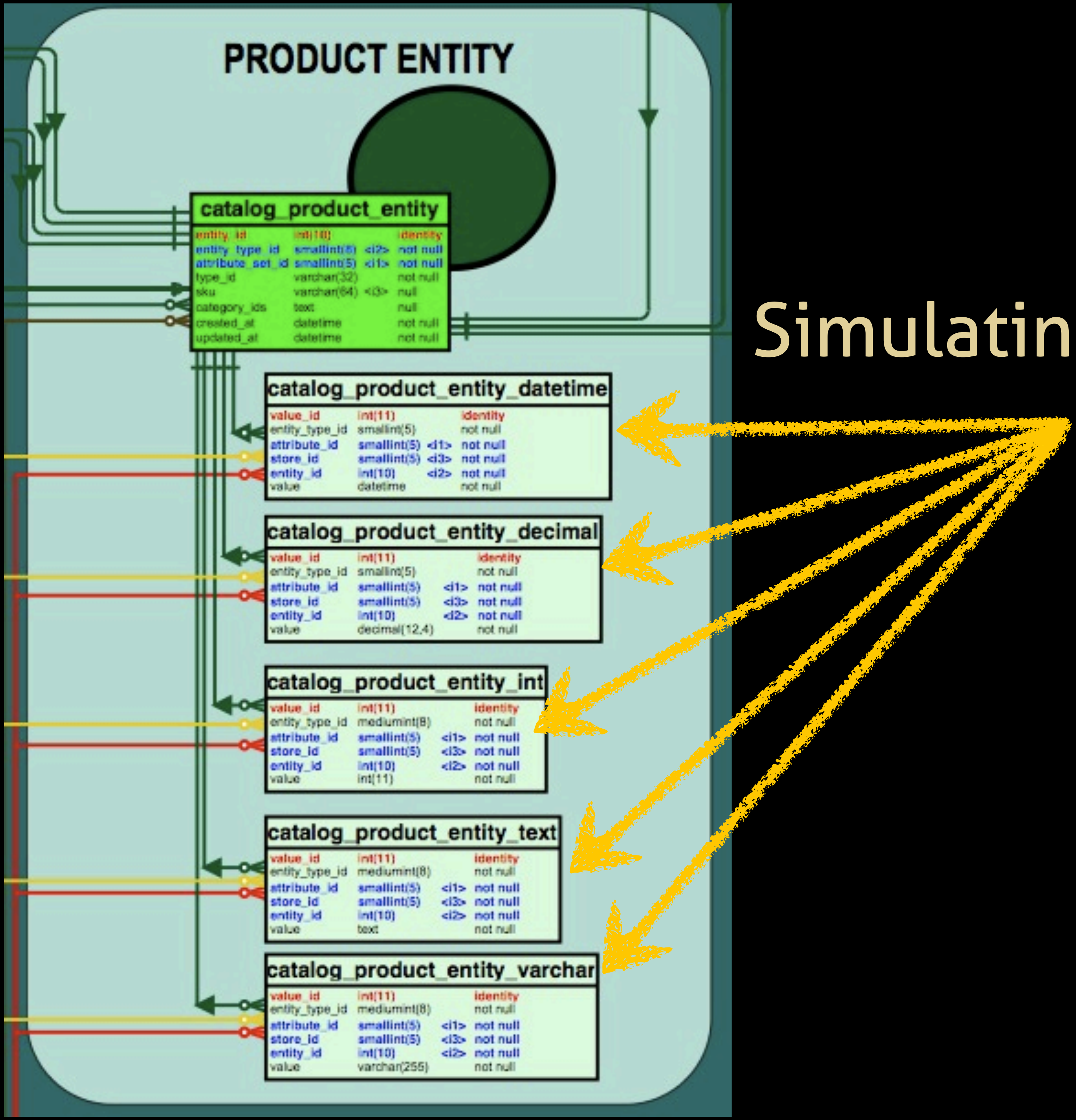
A. *Hundreds of little tables.*





Tuesday, December 15, 2009

Simulating a flexible schema



catalog_product_entity

entity_id	int(10)		identity
entity_type_id	smallint(5)	<i2>	not null
attribute_set_id	smallint(5)	<i1>	not null
type_id	varchar(32)		not null
sku	varchar(64)	<i3>	null
category_ids	text		null
created_at	datetime		not null
updated_at	datetime		not null

catalog_product_entity_datetime

value_id	int(11)		identity
entity_type_id	smallint(5)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	datetime		not null

catalog_product_entity_decimal

value_id	int(11)		identity
entity_type_id	smallint(5)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	decimal(12,4)		not null

catalog_product_entity_int

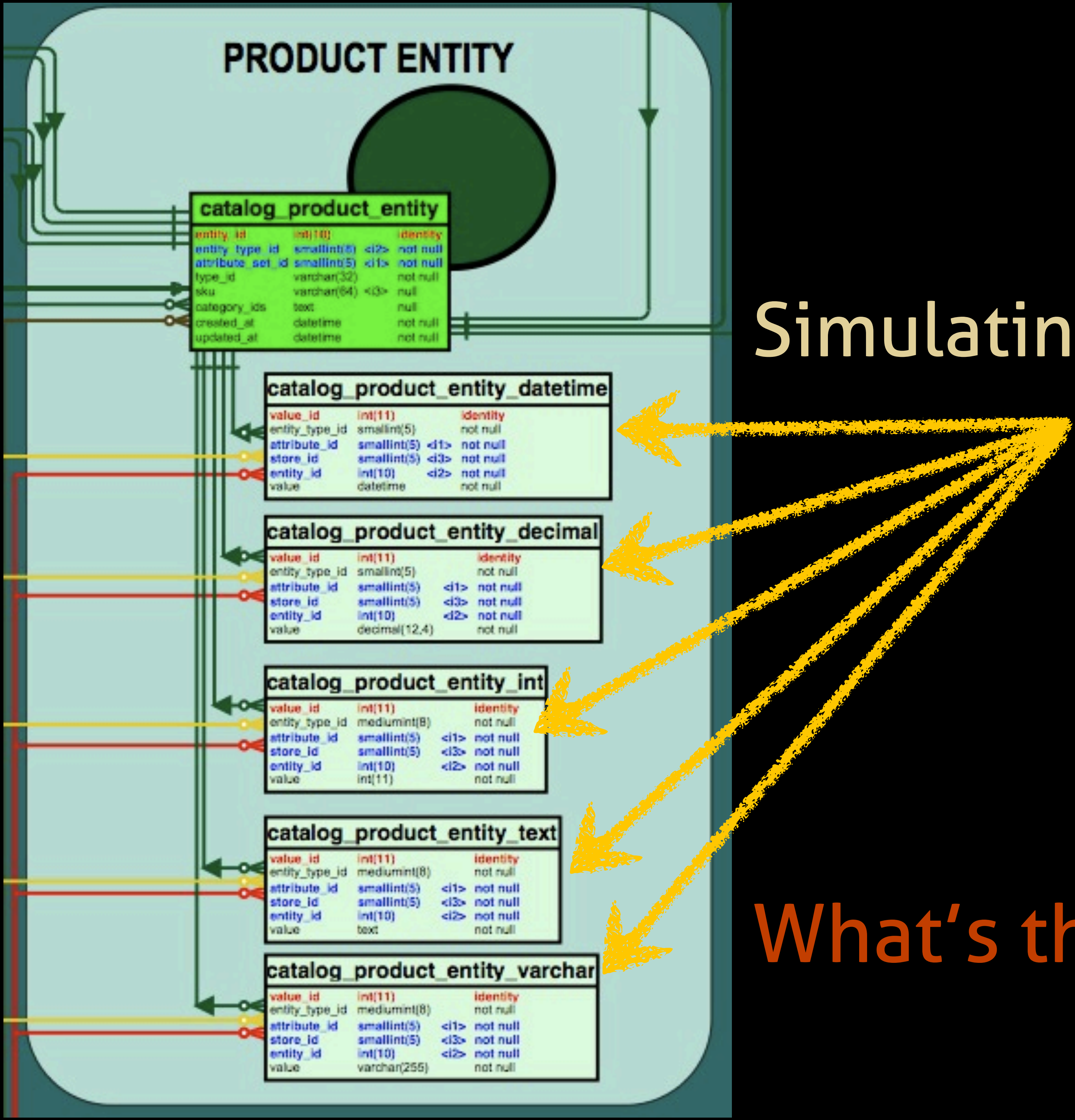
value_id	int(11)		identity
entity_type_id	mediumint(8)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	int(11)		not null

catalog_product_entity_text

value_id	int(11)		identity
entity_type_id	mediumint(8)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	text		not null

catalog_product_entity_varchar

value_id	int(11)		identity
entity_type_id	mediumint(8)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	varchar(255)		not null



catalog_product_entity

entity_id	int(10)		identity
entity_type_id	smallint(5)	<i2>	not null
attribute_set_id	smallint(5)	<i1>	not null
type_id	varchar(32)		not null
sku	varchar(64)	<i3>	null
category_ids	text		null
created_at	datetime		not null
updated_at	datetime		not null

catalog_product_entity_datetime

value_id	int(11)		identity
entity_type_id	smallint(5)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	datetime		not null

catalog_product_entity_decimal

value_id	int(11)		identity
entity_type_id	smallint(5)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	decimal(12,4)		not null

catalog_product_entity_int

value_id	int(11)		identity
entity_type_id	mediumint(8)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	int(11)		not null

catalog_product_entity_text

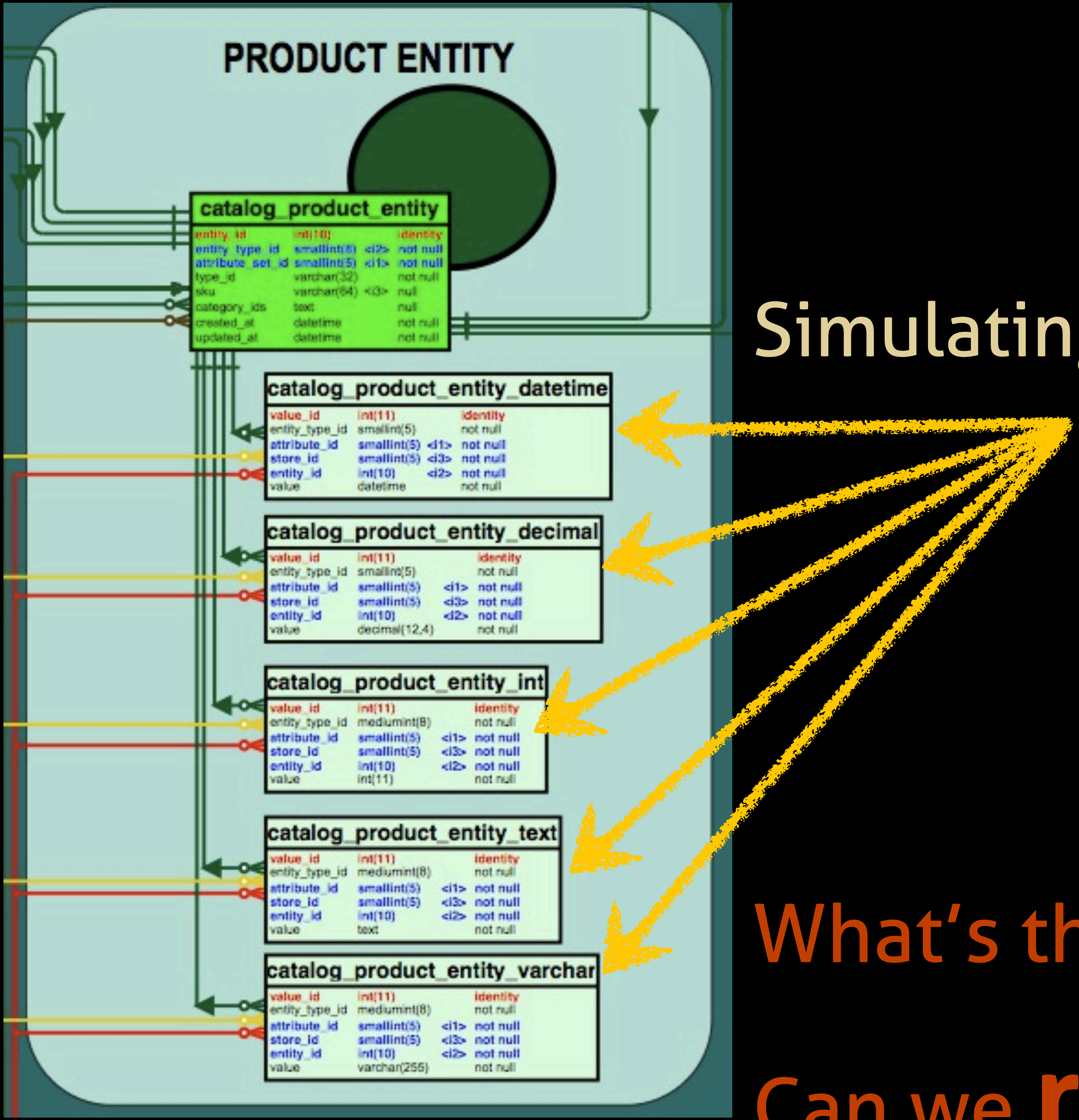
value_id	int(11)		identity
entity_type_id	mediumint(8)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	text		not null

catalog_product_entity_varchar

value_id	int(11)		identity
entity_type_id	mediumint(8)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	varchar(255)		not null

Simulating a flexible schema

What's the **join** like?



catalog_product_entity

entity_id	int(10)		identity
entity_type_id	smallint(8)	<i2>	not null
attribute_set_id	smallint(5)	<i1>	not null
type_id	varchar(32)		not null
sku	varchar(64)	<i3>	null
category_ids	text		null
created_at	datetime		not null
updated_at	datetime		not null

catalog_product_entity_datetime

value_id	int(11)		identity
entity_type_id	smallint(5)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	datetime		not null

catalog_product_entity_decimal

value_id	int(11)		identity
entity_type_id	smallint(5)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	decimal(12,4)		not null

catalog_product_entity_int

value_id	int(11)		identity
entity_type_id	mediumint(8)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	int(11)		not null

catalog_product_entity_text

value_id	int(11)		identity
entity_type_id	mediumint(8)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	text		not null

catalog_product_entity_varchar

value_id	int(11)		identity
entity_type_id	mediumint(8)		not null
attribute_id	smallint(5)	<i1>	not null
store_id	smallint(5)	<i3>	not null
entity_id	int(10)	<i2>	not null
value	varchar(255)		not null

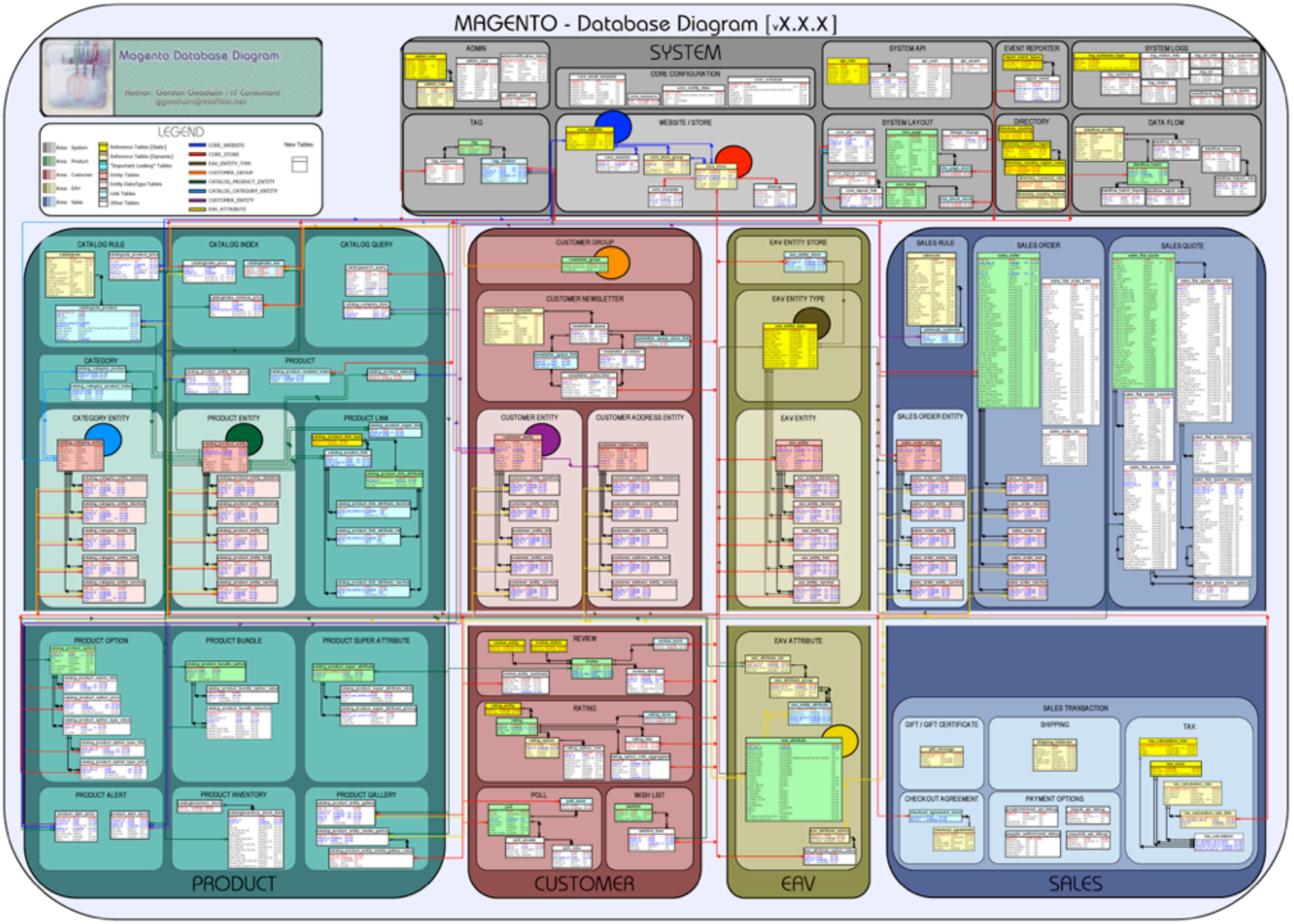
Simulating a flexible schema

What's the **join** like?

Can we **reason** about it?

Q. *What % of these tables could be better represented as documents?*

Machine


MAGENTO - Database Diagram [vX.X.X]
Magento Database Diagram
Author: Gordon Goodwin - IT Consultant
ggoodwin@vsoftinc.net
LEGEND
SYSTEM
ADMIN
CORE CONFIGURATION
SYSTEM API
EVENT REPORTER
SYSTEM LOGS
TAG
WEBSITE / STORE
SYSTEM LAYOUT
DIRECTORY
DATA FLOW
CATALOG RULE
CATALOG INDEX
CATALOG QUERY
CATEGORY
PRODUCT
CATEGORY ENTITY
PRODUCT ENTITY
PRODUCT LINK
PRODUCT OPTION
PRODUCT BUNDLE
PRODUCT SUPER ATTRIBUTE
PRODUCT ALERT
PRODUCT INVENTORY
PRODUCT GALLERY
PRODUCT
CUSTOMER GROUP
CUSTOMER NEWSLETTER
CUSTOMER ENTITY
CUSTOMER ADDRESS ENTITY
REVIEW
RATING
POLL
WISH LIST
CUSTOMER
EAV ENTITY STORE
EAV ENTITY TYPE
EAV ENTITY
EAV ATTRIBUTE
EAV
SALES RULE
SALES ORDER
SALES QUOTE
SALES ORDER ENTITY
SALES TRANSACTION
GIFT / GIFT CERTIFICATE
SHIPPING
TAX
CHECKOUT AGREEMENT
PAYMENT OPTIONS
SALES

human

```
{:sku    => ‘637636’,
 :name => ‘Linen tailored pant’,
 :about => [{:title => ‘fabric & care’,
                     :content => [‘Dry clean’,
                                          ‘Imported’]},
                    {:title => ‘overview’,
                     :content => [‘Tailored fit’,
                                          ‘Yarn dyed’]}
                   ]
}*
```

*Admittedly simplified, but not far-fetched.

2. Ruby to MongoDB

Anatomy of an **Insert**

```ruby
require 'mongo'

@connection = Mongo::Connection.new
@db         = @connection.db('commerce')
@coll       = @db.collection('users')


```

Connection

mongod (or mongos)

Database

a database

Collection

a schema-free table

```ruby

document = {:first_name => 'Yukihiro',
            :last_name  => 'Matsumoto',
            :username   => 'matz',
            :dob        => Time.utc(1965, 4, 14),
            :languages  => ['ruby', 'perl', 'c'],
            :updates => [{:text => 'hacking on garbage collection',
                          :date => Time.now},
                         {:text => '1.9 uses os threads',
                          :date => Time.utc(2007, 1, 1)}]
           }

@users.save(document)
```

ObjectID Generation

BSON Serialization

Fire and Forget

Dynamic Queries*

*like sql

```ruby

@users.find({:username => 'matz'})


@users.find({:first_name => /^Y/})

```

```
@users.find({:dob => {'$lte' => Time.utc(1970, 1, 1)}})

@users.find({:languages => {'$in' => ['ruby', 'perl']}})

@users.find({'updates.date' => {'$gt' => Time.utc(2009)}})
```

b-tree

Indexes*

*up to 40 per collection

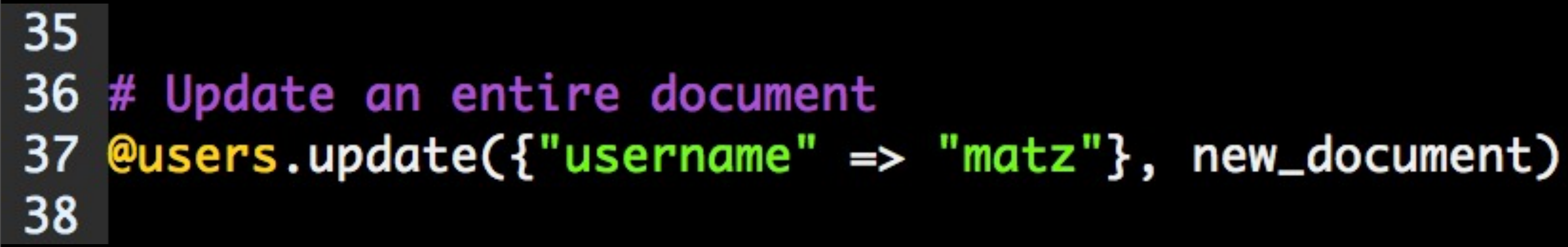
```ruby

# Ascending index on first_name (string)
@users.create_index([:first_name, 1])


# Ascending index on languages (array)
@users.create_index([:languages, 1])


# Descending index on dob (date)
@users.create_index([:dob, -1])


# Unique index on username
@users.create_index([:username, 1], true)

```

```
# Compound-key indexes (and, this case, nested)
@users.create_index(['updates.votes', 1],
                    ['updates.created_at', -1])
```

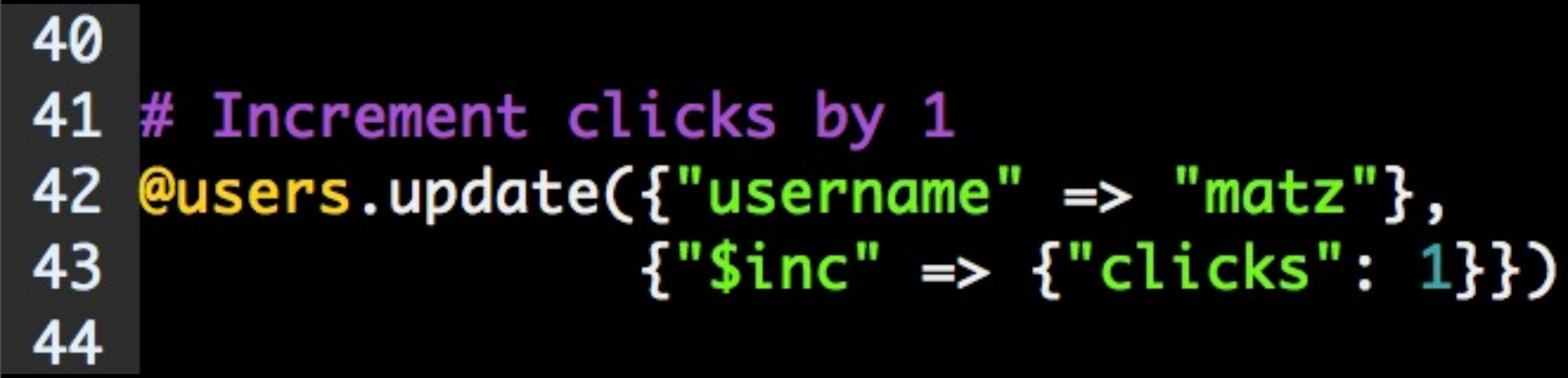
Flexible & *Fast*
Updates*

*and **upserts**, too

```ruby

# Update an entire document
@users.update({"username" => "matz"}, new_document)

```



```
# Update an entire document
@users.update({"username" => "matz"}, new_document)
```



```
# Increment clicks by 1
@users.update({"username" => "matz"},
              {"$inc" => {"clicks": 1}})
```

```ruby

# Update an entire document
@users.update({"username" => "matz"}, new_document)

```

```ruby

# Increment clicks by 1
@users.update({"username" => "matz"},
              {"$inc" => {"clicks": 1}})

```

```ruby

# Push on another status update
@users.update({"username" => "matz"},
         {"updates" =>
           {"$push" =>
             {"text" => "human-readable",
              "date" => Time.now

```

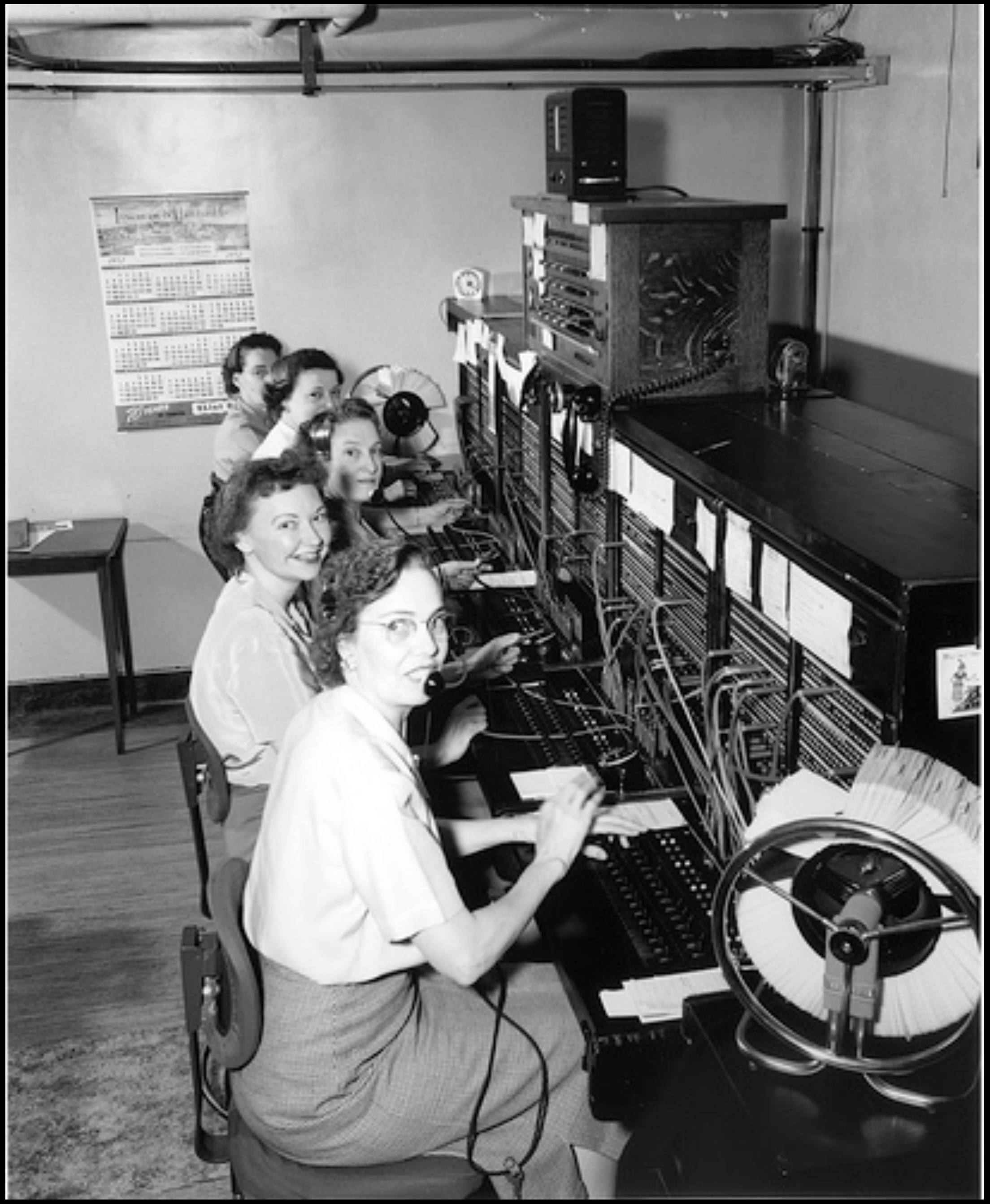
```ruby

# Upserts
@users.update({"username" => 'matz'}, document,
                  :upsert => true)

```

Elegant Operators*

*document keywords

for **queries**

$ne
$in
$nin
$mod
$all
$size
$exists



for **updates**

$inc
$set
$push
$pushAll
$pop
$pull
$pullAll

for everything else:

Javascript*

*yes, MongoDB speaks JS

group
where
map-reduce

3. Design Patterns

One to

y Many Many Many Ma

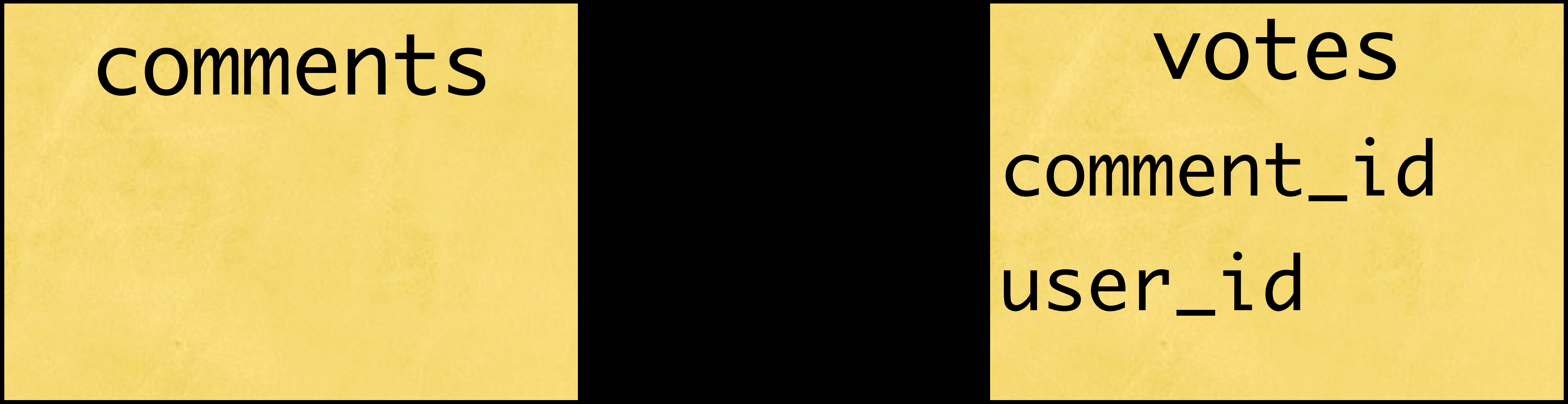
any Many Many Many

ny Many Many Many M

Many Many Many Man

comments has_many votes

relation



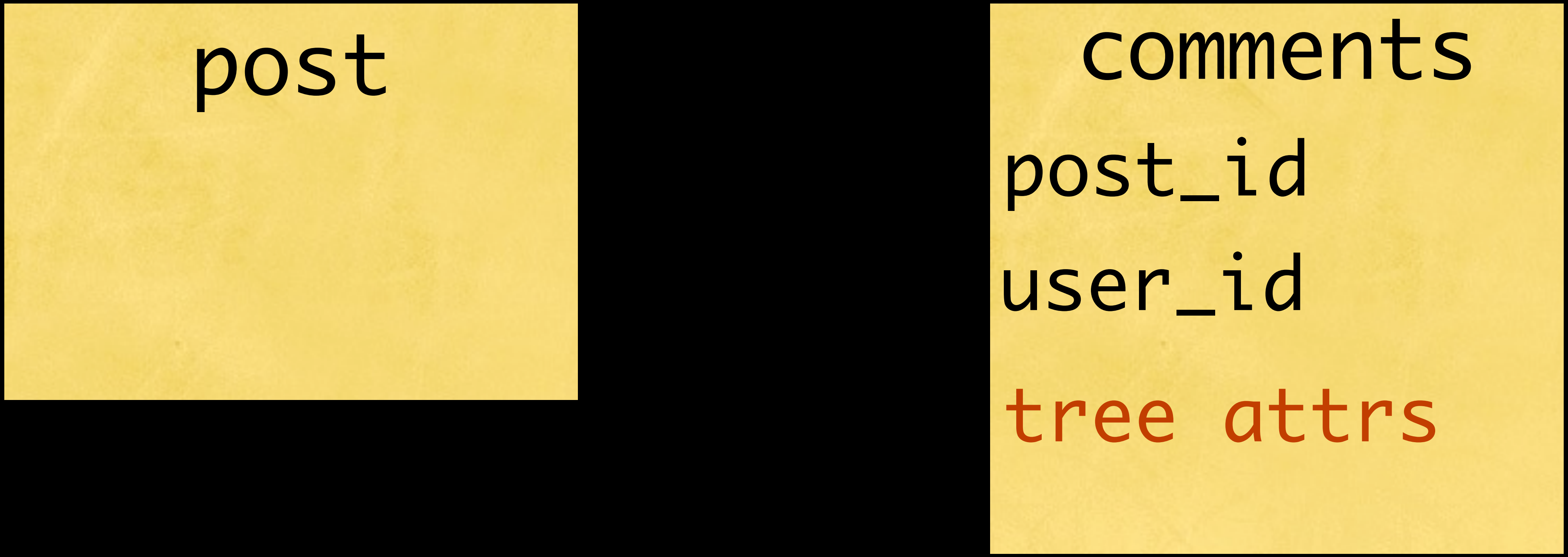
document

```

comment = {:text => "lmao!",
           :date => Time.now,
           :voters => ['48f8fc0001',
                       '48f8fc0002',
                       '48f8fc0003'],
           :votes  => 3
          }

```

```

# Atomic update
Comment.update({"_id" => comment_id, "voters" => {"$ne" => voter_id}},
               {"voters" => {"$push" => voter_id}, "votes" => {"$inc" => 1}}

```

post has_many comments

relation



1. embedded document

```
{:title    => ‘a life unexamined’,
 :comments => [
   {:author => ‘socrates’,
    :text     => ‘is not worthwhile’},
   {:author => ‘epicurus’,
    :text     => ‘leads to bliss’}
                         ]
}
```

2. embedded & nested

```
{:comments => [
   {:author => ‘socrates’,
    :text      => ‘is not worthwhile’,
    :comments => [
      {:author => ‘epicurus’,
       :text      => ‘leads to bliss’}]},
                  ]
}
```

3. normalized

```
[{:author  => ‘socrates’,
  :text      => ‘is not worthwhile’,
  :post_id => ‘4c4fa6d000002’},
 {:author  => ‘epicurus’,
  :text      => ‘leads to bliss’,
  :post_id => ‘4c4fa6d000002’}
]
```

Embed relationships

tightly-bound concepts

tradeoffs

Break out first-class docs

independent concepts

any Many Many Many

any Many Many Many M

Many Many Many Man

to

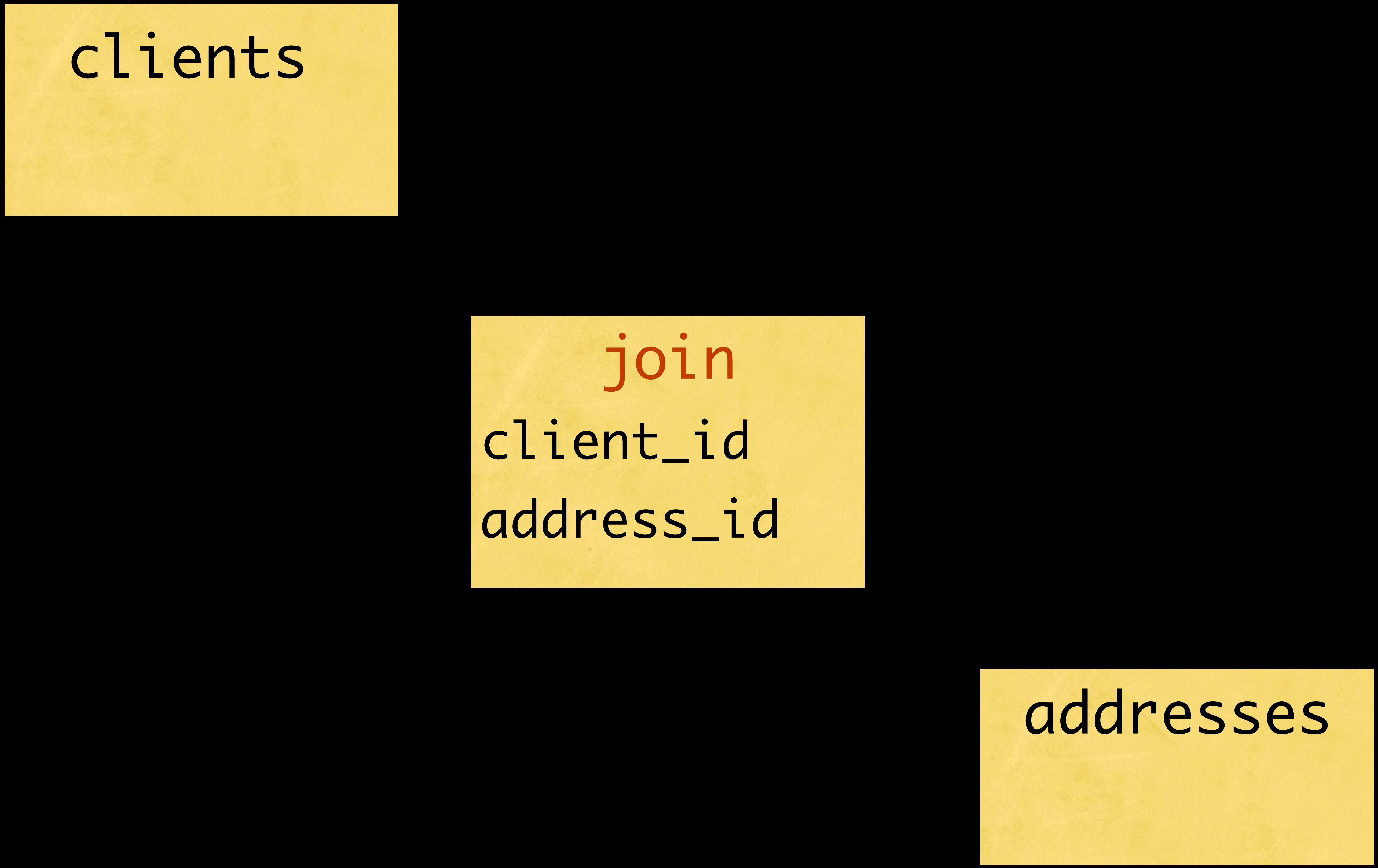
y Many Many Many Man

ny Many Many Many

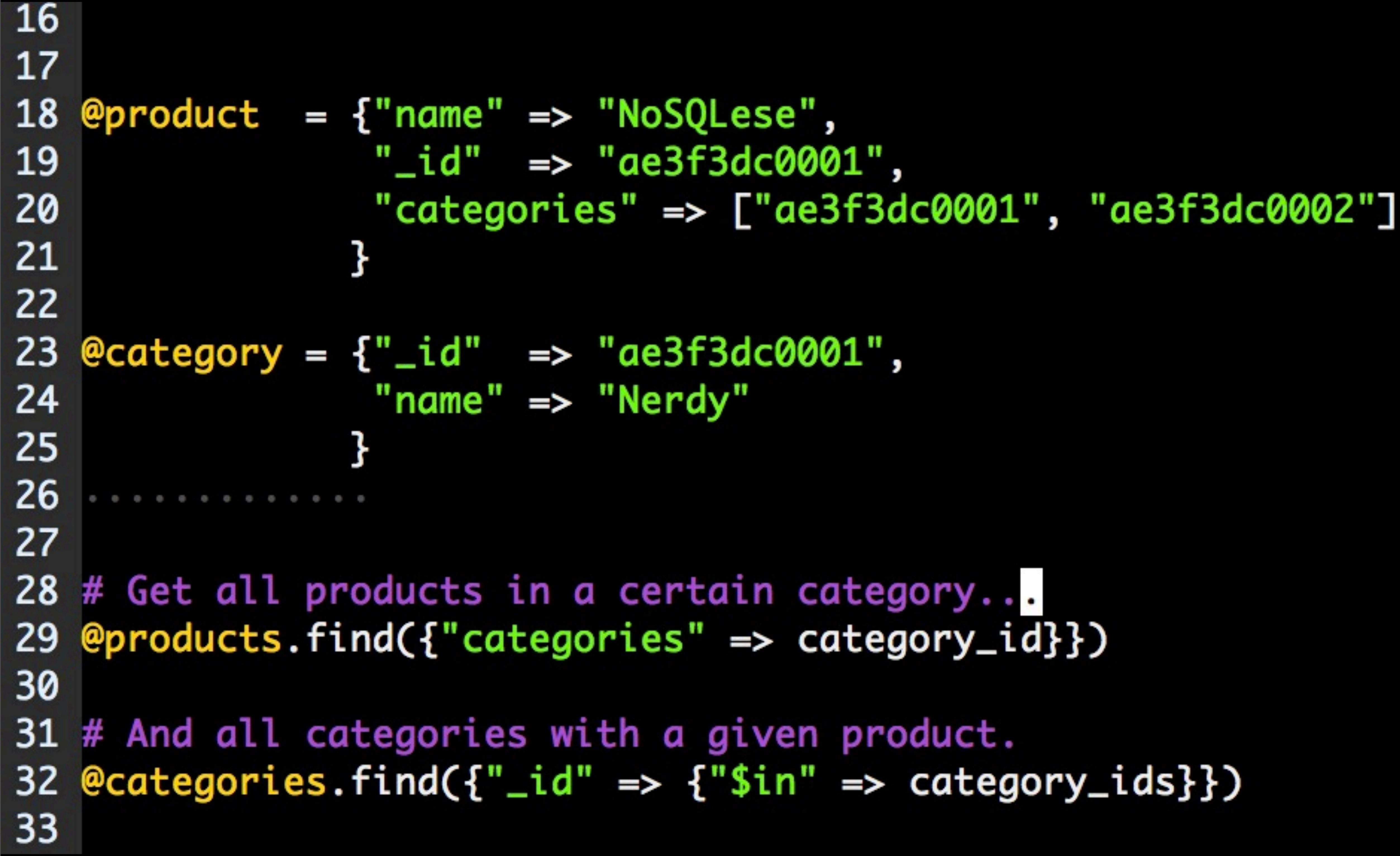
ny Many Many Many M

Many Many Many Many

relation



document



```
@product  = {"name" => "NoSQLese",
             "_id"  => "ae3f3dc0001",
             "categories" => ["ae3f3dc0001", "ae3f3dc0002"]
            }

@category = {"_id"  => "ae3f3dc0001",
             "name" => "Nerdy"
            }
·············

# Get all products in a certain category...
@products.find({"categories" => category_id}})

# And all categories with a given product.
@categories.find({"_id" => {"$in" => category_ids}})

```

DenOrmalization

```
{ :username  => 'socrates',
  :text      => '...be as you wish',
  :user_id   => '4c4fa6d000002' }
```

Exercises for the coder

Capped Collections

GridFS for images, videos, music, large binary objects

MongoDB JS Shell
MongoMapper

MongoDB JS Shell
MongoMapper

Pre-compiled binaries
Thorough documentation
Multi-language support

Q. *What is MongoDB good for?*

A. the web real-time

logging

analytics

A. the web

real-time

  logging

  analytics

clear path to scalability

comprehensible data models

speed

A.

the web
real-time
  logging
  analytics

clear path to scalability
comprehensible data models
speed

human-oriented programmers

google groups: mongodb-user

freenode: #mongodb

docs & download: mongodb.org

github.com/banker/newsmonger

newsmonger.heroku.com

twitter.com/hwaet

kyle@10gen.com

inspired by slideshare.net/timanglade/tin