APACHECON
North America
Montréal, Canada
24-27 September, 2018

CouchDB
relax

# Another
# 10 Common Misconceptions about Apache CouchDB

Joan Touzet ❦ https://atypical.net/ ❦ `wohali`

## Just like last time…

This presentation is a bit dry.

But these problems keep coming up on mailing lists, IRC, and in discussion with developers & operators.

Today, I'd rather inform than entertain.

## The Platform

JAMES BOW PHOTOGRAPHER (20170119); TRANSIT TORONTO COLLECTION (2017)

## 1. CouchDB = MongoDB

- The "original" NoSQL (…*but we were provably first!*)
- Document-oriented structure
- Map-Reduce
- Streaming changes feeds

## CouchDB ≠ MongoDB

“My party line on Mongo vs. Couch is that on the surface they might look similar (database, documents, JSON-ish), but when you look at implementation, at every of the 100,000 decisions you have to make when building such a thing, Mongo went one way, and we went another.”

*– Jan Lehnardt, VP Apache CouchDB*

## CouchDB ≠ MongoDB

<u>MongoDB</u>

- Binary protocol
- BSON (binary)
- Speed
- Features

<u>CouchDB</u>

- HTTP API
- JSON
- Durability (append only)
- Scalability

## CouchDB ≠ Couchbase

### Couchbase

- **No longer** compatible with CouchDB or PouchDB!
- Frankenproduct of Membase + CouchDB fork
- Commercial product

### CouchDB

- Replication is our killer feature!
- Does one thing well. Plays great with Redis, Apache Spark, etc.
- Apache-licensed OSS

## 2. Installing CouchDB?

“CouchDB is hard to install.”

“Erlang? Ancient JavaScript? Feh.”

## Installing CouchDB!

Packages from Apache repositories now available! (See docs.couchdb.org)

- `apt install couchdb`
- `yum install couchdb`

64-bit Windows installer also available (for development)

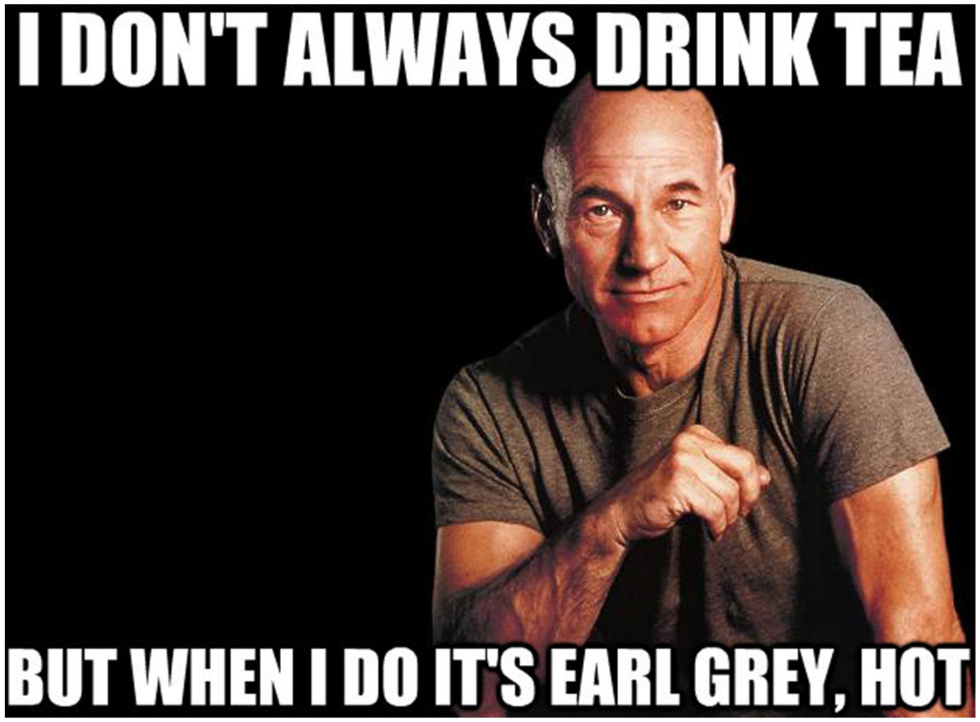
- Please don't run CouchDB on Windows in production!

macOS installer available (for development)

FreeBSD ports tree now has CouchDB 2.2.0, too.

Docker image available as `apache/couchdb` or `couchdb`

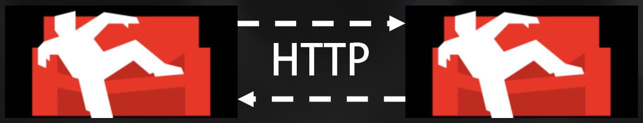
## Replication

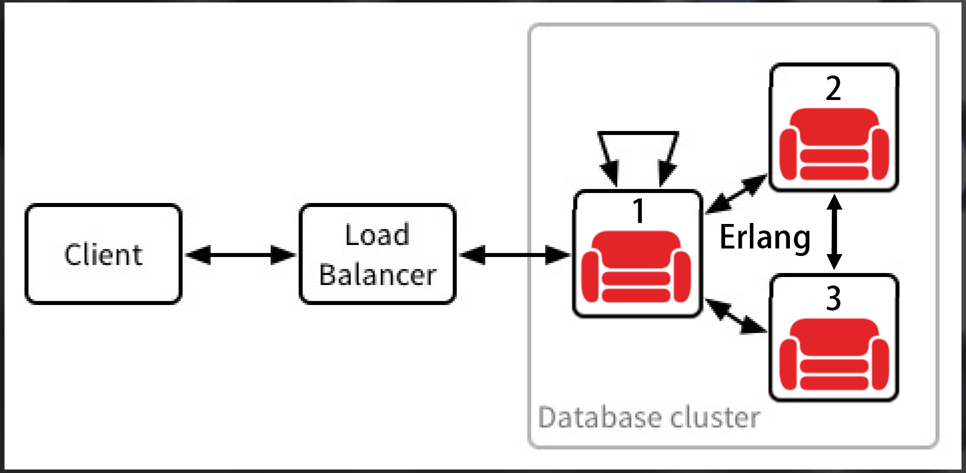


https://imgur.com/gallery/RdzjQWe

## 3. Scaling via replication

Yes … but not in the way you think!





CouchDB 1.x

CouchDB 2.x

## What does this mean?

CouchDB 2.x has native clustering functionality

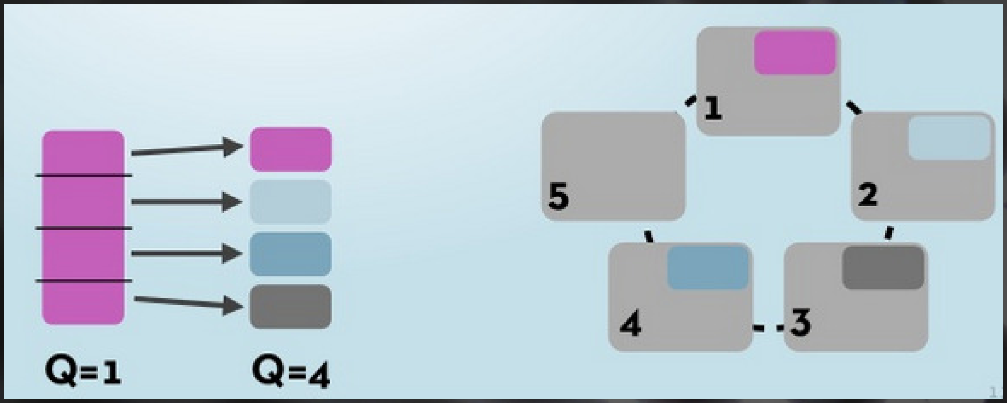
"Internal replication" is optimized for this process

CouchDB 2.x shards the database for optimization

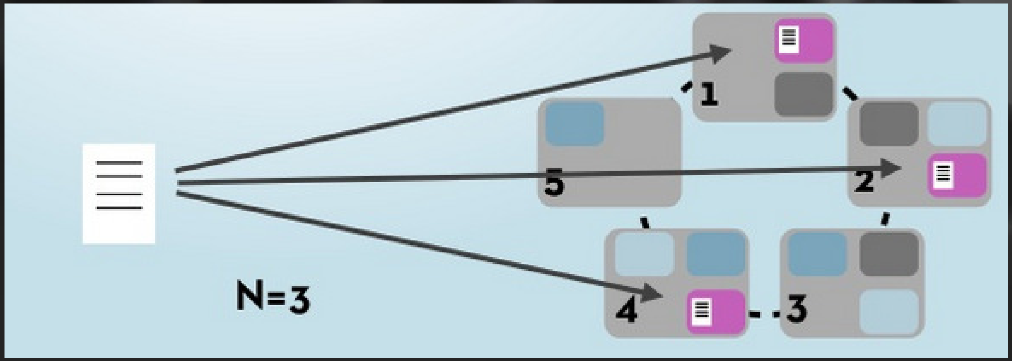
CouchDB has **no leader election** or "**global coordinator**"!

# Database / View Sharding

*q* = # of shards
(default: 8)
(4 here for a small picture)



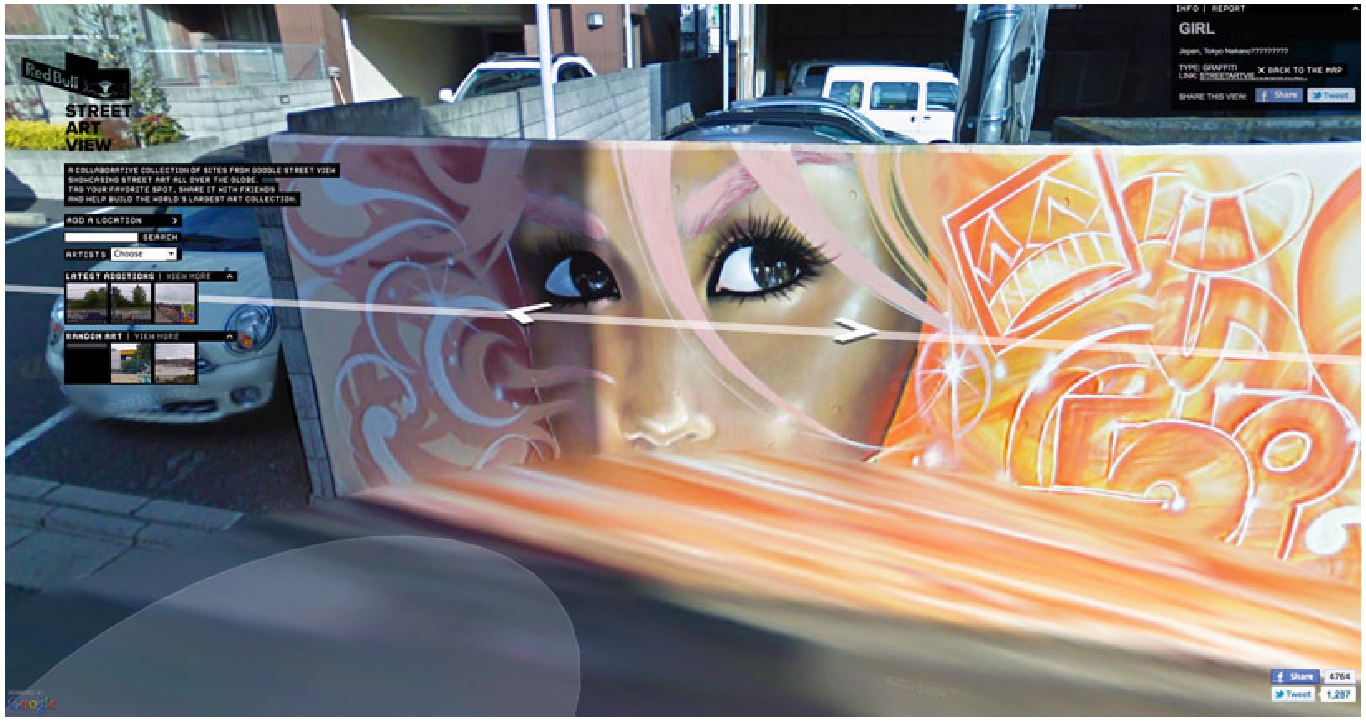
*n* = number of replicas
(default: 3)



## Deployment Recommendations

1. Keep all nodes in the same AZ / data centre / rack
2. Stick with the defaults (*q*=8, *n*=3) unless you're really small (1 node) or really big (>50GB JSON DB)
3. Use HAProxy for your load balancer, it's the best!

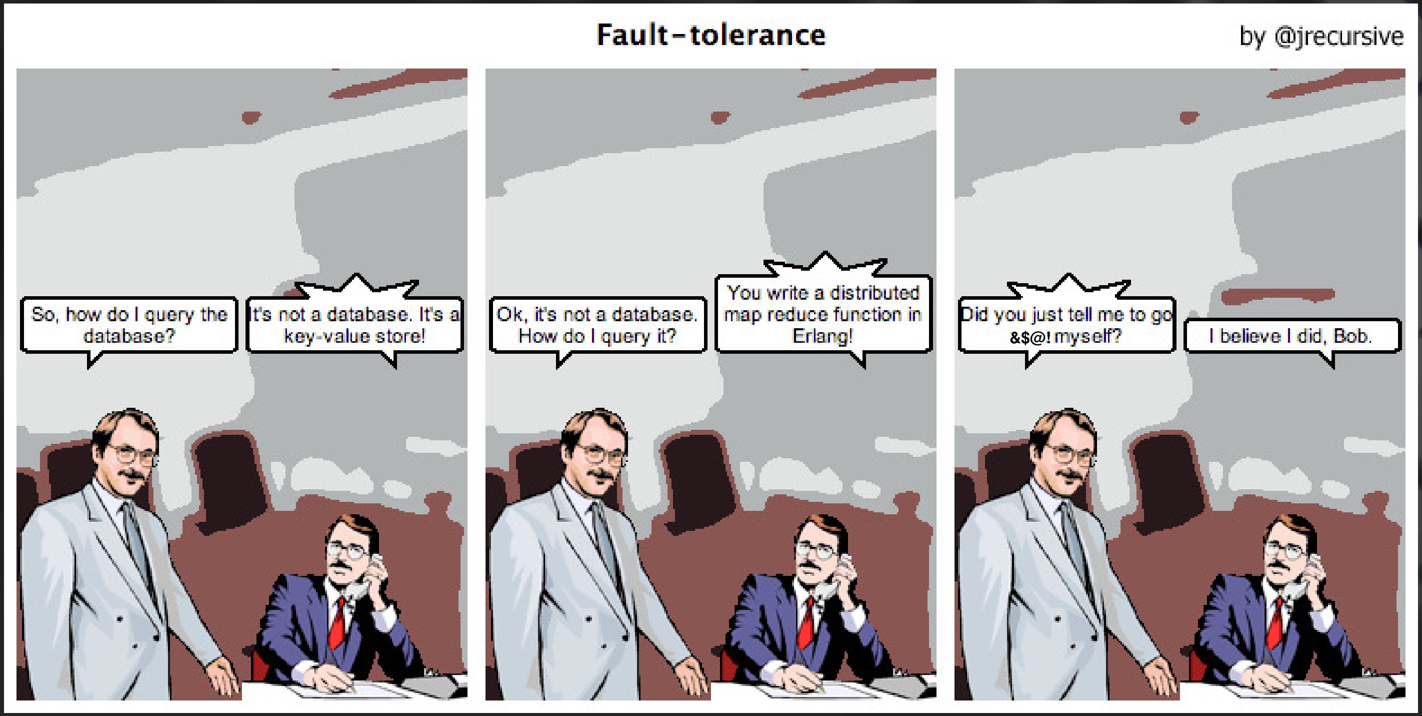
## Document Indexes & Views



A COLLABORATIVE COLLECTION OF SITES FROM GOOGLE STREET VIEW
SHOWCASING STREET ART ALL OVER THE GLOBE.
TAG YOUR FAVORITE SPOT, SHARE IT WITH FRIENDS
AND HELP BUILD THE WORLD'S LARGEST ART COLLECTION.

Graffiti, as captured by Google Earth, Tokyo, Japan

# 4. MapReduce is (still) hard.


Fault-tolerance
by @jrecursive
So, how do I query the database?
It's not a database. It's a key-value store!
Ok, it's not a database. How do I query it?
You write a distributed map reduce function in Erlang!
Did you just tell me to go &$@! myself?
I believe I did, Bob.

## 4. MapReduce is (still) hard.

Now, you have three easier, fantastic options!

1. Mango
2. Full-text Search† (Apache Lucene powered)
3. Geospatial Search†

†Provided by 3rd-party add-ons, requires recompile.

## What is Mango?

Declarative, JSON-based query language

Designed to meet ≥75% of all your querying needs

Inspired by a well-known NoSQL competitor's query…

*Actually* the same Map-Reduce implementation underneath!

## Introduction to Mango

A. Prototype your query.

B. Make an index to speed it up.

C. Check & use your index in your query.

## Mango selectors are powerful.

```
{ "zagat.rating": { "$gt": 18 } }

{ "michelin.stars": { "$exists": true } }

{"cuisine": { "$all": ["Malaysian","Singaporean"] }}
```

…and <u>everything</u> is specified in the selector at query time!

## In JavaScript...

```
{ "zagat.rating": { "$gt": 18 } }
```

```
if (doc.zagat &&
    doc.zagat.rating &&
    doc.zagat.rating === int(doc.zagat.rating)) {
  if (doc.zagat.rating > 18) {
    return(doc._id, null);
  }
}
```

## A. Prototype your query.

```
$ curl -H "Content-type: application/json" -X POST \
    http://localhost:5984/mydb/_find \
    -d '{"selector": { "food": "chili" }}' | jq .

{
  "docs": [
    {
      "_id": "b",
      "_rev": "1-0f07c7dbc9a29f0d0c2729f9c61f5411",
      "name": "Chris",
      "food": "chili"
    }
  ],
  "bookmark": "g1AAAAAyeJzLYWBgYMpgSmHgKy5JLCrJTq2MT8lPzkzJBYozJoEkOGASEKEsAE8ZDXs",
  "warning": "no matching index found, create an index to optimize query time"
}
```

## B. Make an index.

```
$ curl
    -H "Content-type: application/json" \
    -X POST \
    http://localhost:5984/mydb/_index \
    -d '{"index": { "fields": ["food"] }, "ddoc": "food", "type": "json"}'
```

```
{
  "result": "created",
  "id": "_design/food",
  "name": "f9aed20d8e363a7066bfd32ee016b6280163b99a"
}
```

## C. Check & use your index.

```
$ curl -H "Content-type: application/json" -X POST \
    http://localhost:5984/mydb/_explain \
    -d '{"selector": { "food": "chili" }, "use_index": "food"}' | jq .
```

```
{
  "dbname": "abc",
  "index": { "ddoc": "_design/food", ... },
  "selector": { "food": { "$eq": "chili" } },
  "opts": { "use_index": [ "food" ], ... },
  "limit": 25, "skip": 0, "fields": "all_fields", ... }
}
```

## C. Check & use your index.

```
$ curl -H "Content-type: application/json" -X POST \
    http://localhost:5984/mydb/_find \
    -d '{"selector": { "food": "chili" }, "use_index": "food"}' | jq .

{
  "docs": [
    {
      "_id": "b",
      "_rev": "1-0f07c7dbc9a29f0d0c2729f9c61f5411",
      "name": "Chris",
      "food": "chili"
    }
  ],
  "bookmark": "..."
}
```

## Mango Pro Tips

1. Index on all the fields you use in your selector.
2. Index use is automatic, but double-check `/{db}/_explain` before going into production!
3. Avoid `$in` and `$regex` unless absolutely necessary.
   - These operators are always a full db/index scan! That means they're **slow**!
   - If you really need this, look into the Lucene-powered full-text search add-on.
4. Mango indexes still use design documents.
   - check out `_design/food` after trying this example!
5. Use selectors for replication instead of JavaScript filters - way faster!

~~4. MapReduce is still hard.~~

4. Mango is easy.

## 5. “Cool, attachments!”

Large attachments can create performance issues, especially for replication.

- Replication of entire database will be held up by big attachments
  - This is also true for node-to-node internal cluster replication!
- Large files can rapidly eat available disk space
- >1GB attachments are not a first-order design scenario.

*Repeat!*

Attachments are not available to views or Mango.

You wouldn’t store video files as BLOBs in Oracle, would you?

## New Recommendations

- Use Couch doc `_id` or GUIDs to tag large assets
- Stash them in S3, B2, Dropbox, NextCloud, etc.
- If you must use them: ≤16 MB total per Couch doc.
- Upgrade to CouchDB ≥2.2.0 (see bug #745)

## DB & Document Design

Toronto City Hall

## 6. (Ab)using the primary index

CouchDB 1.x:

"I put my document type in the document's `_id`.
"Then I just use sub-range queries on `/{db}/_all_docs`…"

```
GET /{db}/_all_docs?startkey=type_###&endkey=type_###
```

## New Recommendation

2.x: Use Mango partial indexes!

- Index only contains matching docs
- Can further narrow scope at query time meaningfully
- You **must** add the `use_index` parameter at query time

Example `/{db}/_index` :

```
{
  "index": {
    "partial_filter_selector": {
      "type": "account",
      "status": {
        "$ne": "archived"
      }
    }
  },
  "fields": [ … ]
}
```

## 7. Deleting Documents

"I upload sensor data, process it, then delete it."

In other words, CouchDB as ersatz message queue

## Doc Deletion Options

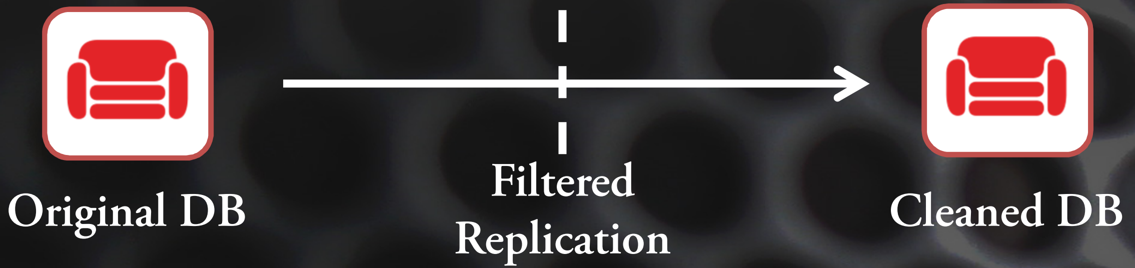
1. Rolling Databases:

June 2018 July 2018 August 2018

   - Write/read only from the database you need
   - When done, archive or delete as necessary
   - Pick your own appropriate time interval

## Doc Deletion Options

2. Replicate-to-remove:



– Filter out deleted documents during replication
– Swap DB when done. Opportunity to re-shard if desired!
– Do this with a single command using: https://github.com/neighbourhoodie/couchdb-continuum

## Doc Deletion Options

3. Maybe CouchDB isn't right for you…

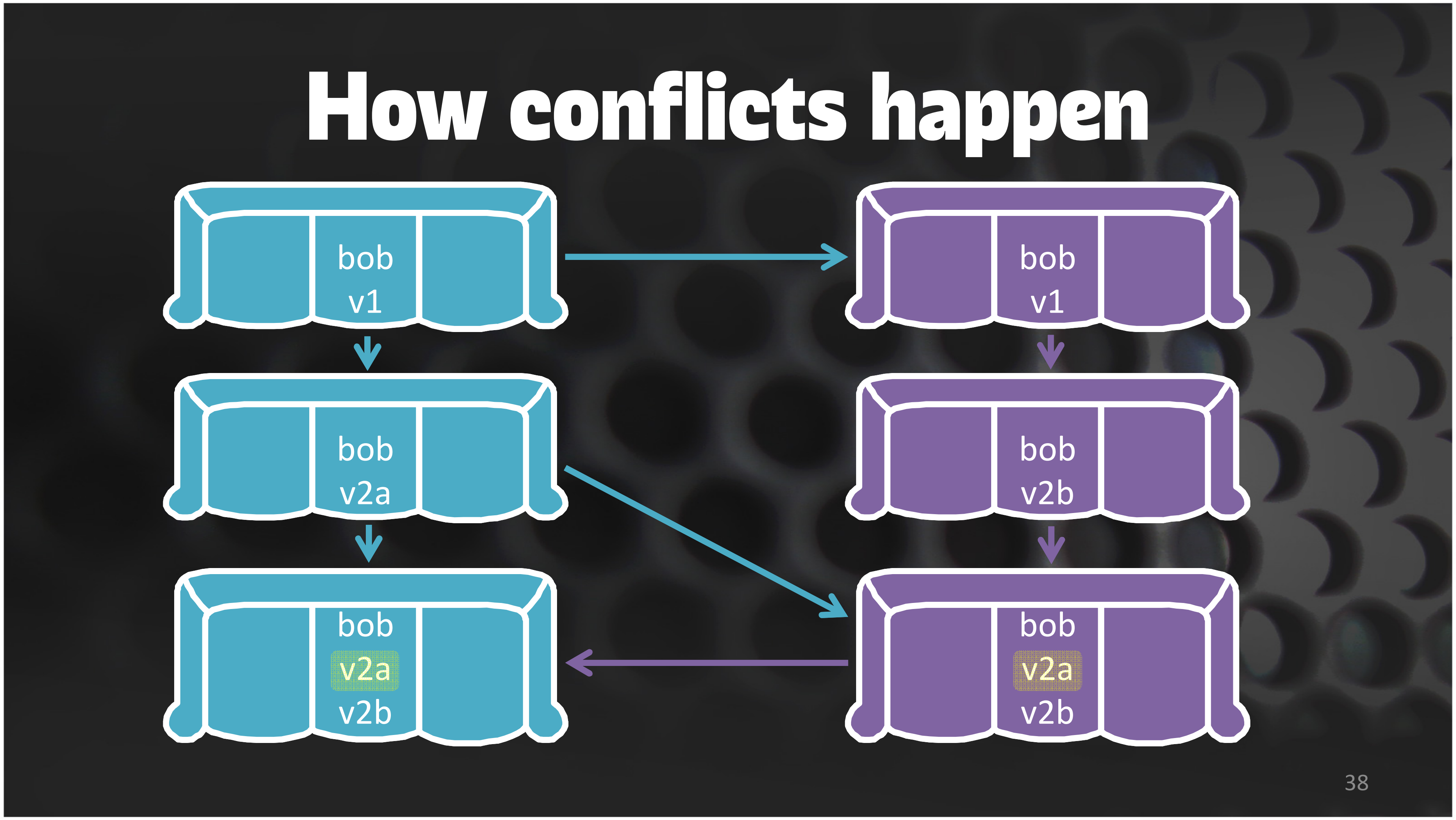
   – Consider a time-series database (like OpenTSDB)
   – Consider a true message queue (like RabbitMQ)

4. Clustered purge (CouchDB ≥2.3.0) may help (but is not a panacea, read the docs on release)

## 8. “Conflicts? What are those?”

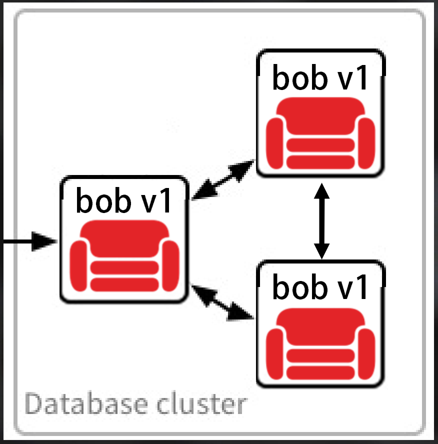
“I write a document. I never check for conflicts.”

“I write a document, if I get a conflict, I just write it again.”

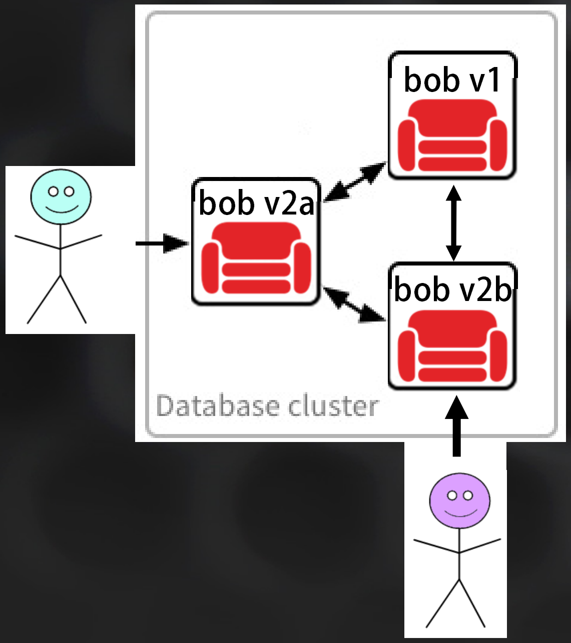


## How conflicts happen

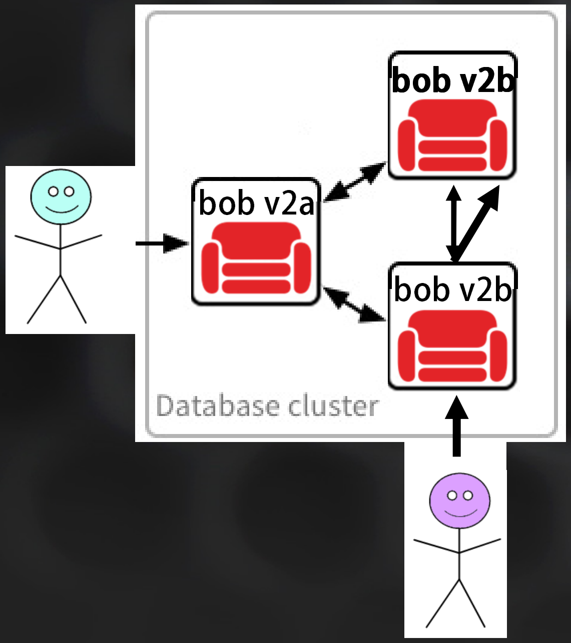
## How conflicts also happen



# How conflicts also happen

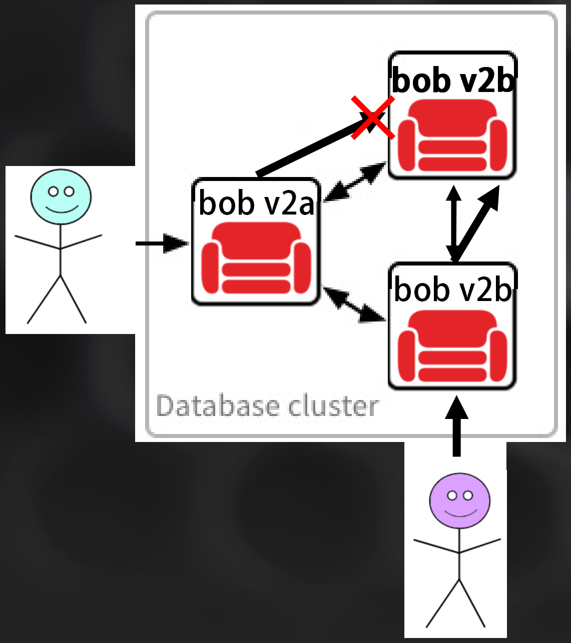


# How conflicts also happen



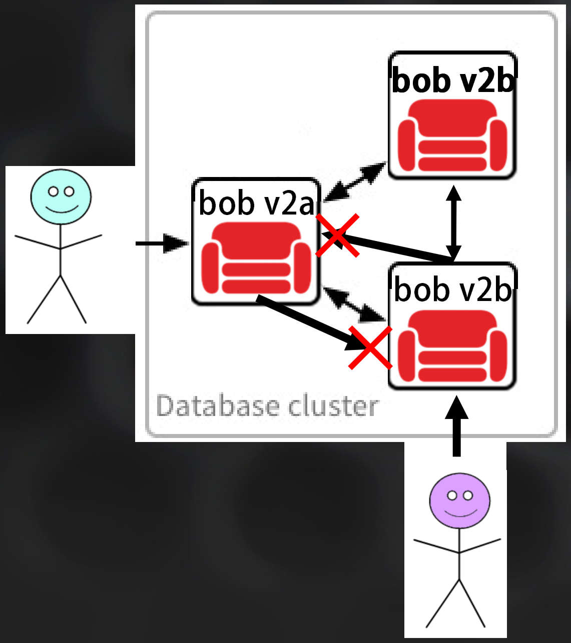
00:01.001

## How conflicts also happen



00:01.002

## How conflicts also happen

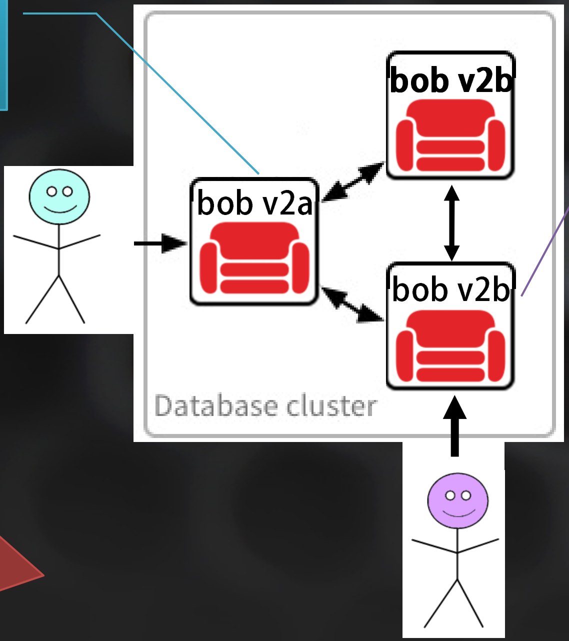


00:01.003

## How conflicts also happen

copies = 1
$n$ = 3
Quorum NG

copies = 2
$n$ = 3
Quorum OK



Quorum:

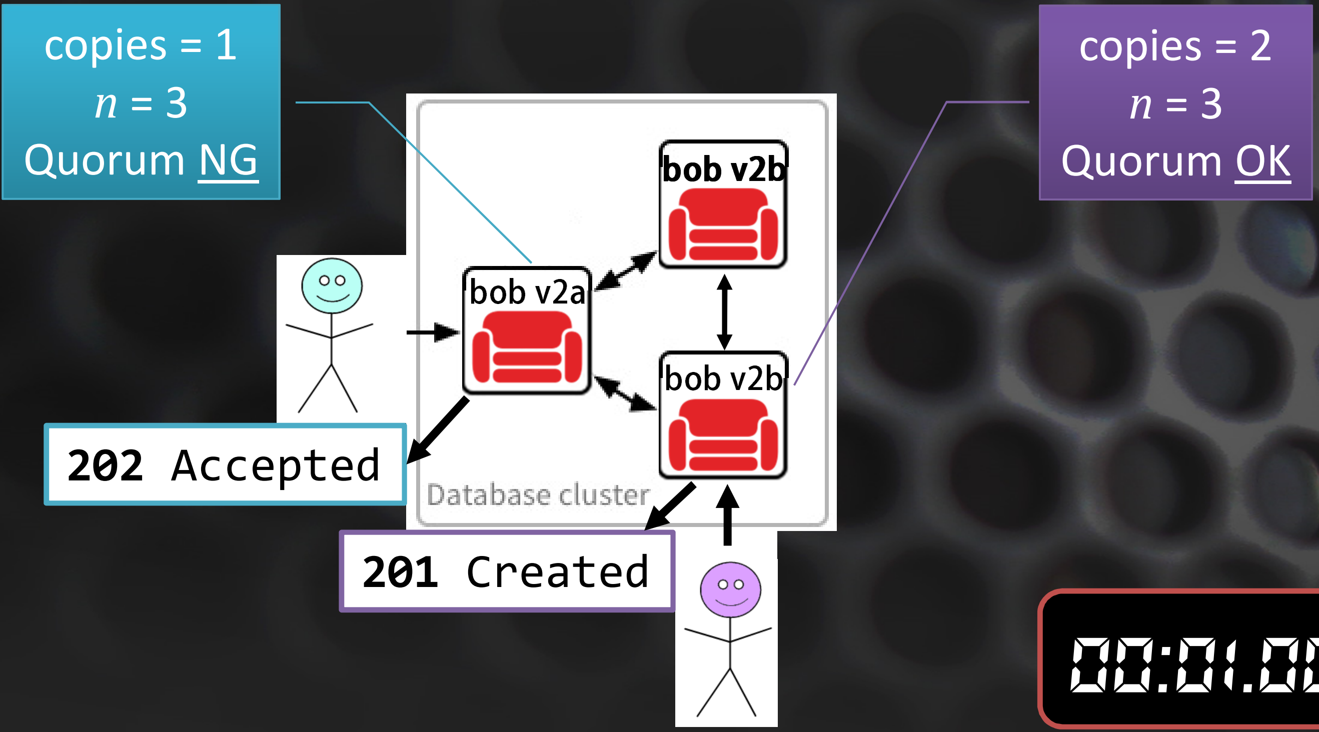
$\geq \frac{n+1}{2}$ copies

00:01.004

## How conflicts also happen

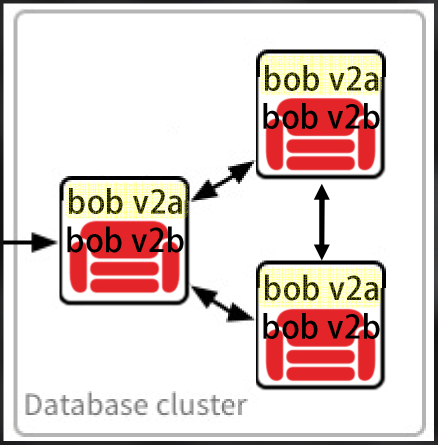
copies = 1
$n$ = 3
Quorum NG

copies = 2
$n$ = 3
Quorum OK



00:01.009

# How conflicts also happen



bob v2a
“arbitrarily”
wins!

00:01.010

## How to detect & resolve conflicts

#1 Best option:

- Listen for `201 Created` vs. `202 Accepted`
  - Check your library code: many libraries **don't differentiate!!**

- Whoever receives a `202` must decide what to do!

- Best if automatic winner selection is **NOT OK**.

## How to detect & resolve conflicts

Second best option:

- Look for conflicts in a system cleanup script
  - Use Mango with selector `{"conflicts": true}` (CouchDB ≥2.2.0)
- Cleanup script must decide what to do!
- Best if merging the documents can be done later.

## How to detect & resolve conflicts

Third best option:

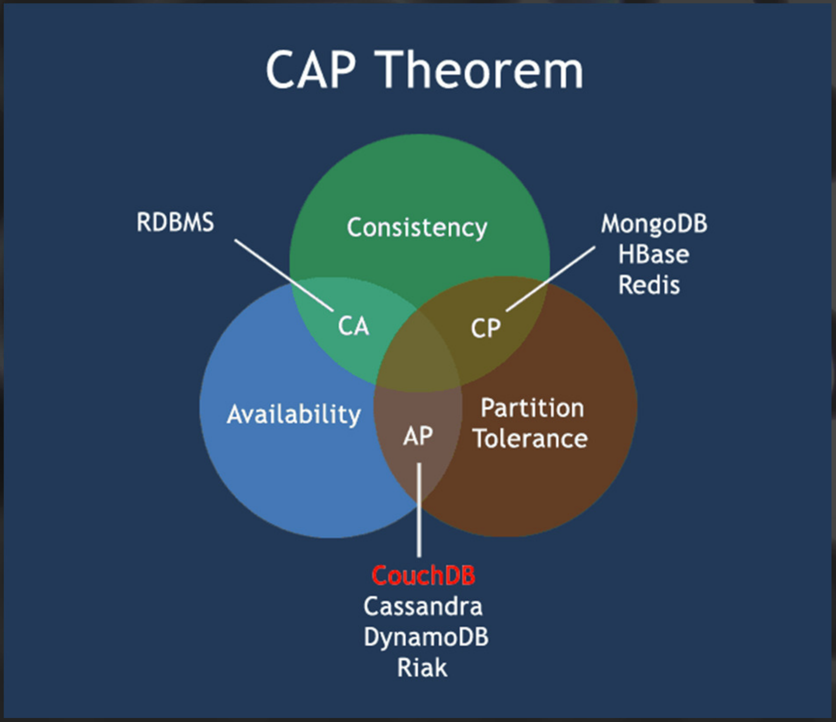
- Look for conflicts in a system cleanup script
  - Use Mango with selector `{"conflicts": true}` (CouchDB ≥2.2.0)
- Cleanup script just deletes losing document
- Best if automatic winner selection is OK.

## 10. Counting with `_rev` / `seq`

CouchDB 1.x:

"`_rev` always increments by 1, right?"

"DB sequence numbers give me absolute document ordering!"



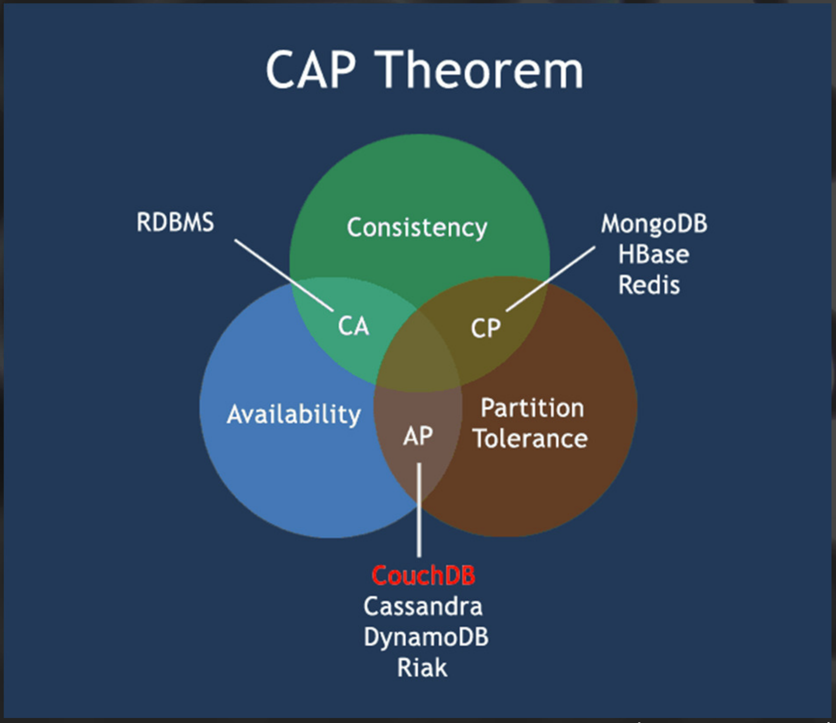
w3resource.com (CC BY-NC-SA 3.0)

You didn't even realize that I completely skipped #9!

## 9. Counting with `_rev` / `seq`

CouchDB 2.x clustering means developers must think more about the implications of distributed systems.

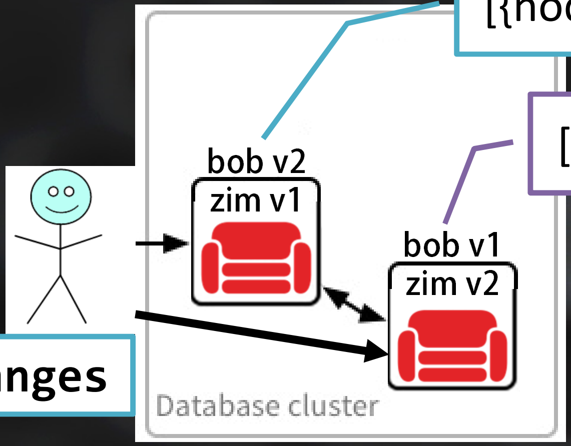
`_rev` / `seq` now include information about the cluster state at the time of generation.



*w3resource.com (CC BY-NC-SA 3.0)*

## 9. Counting with `_rev` / `seq`

$q = 2$



```
GET /db/_changes
```

[{node1, 00-79, 0}, {node2, 80-FF, 1}]

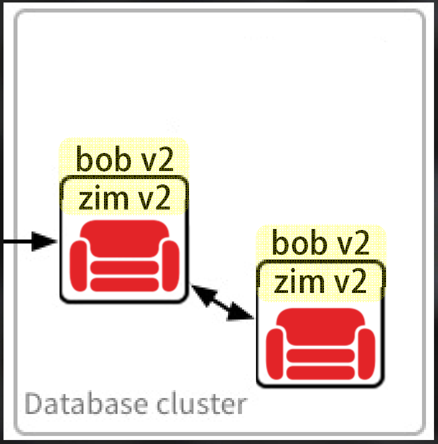
[{node1, 80-FF, 1}, {node2, 00-79, 0}]

Decoded for explanation

00:07.00

Either response is OK – and intuitive!

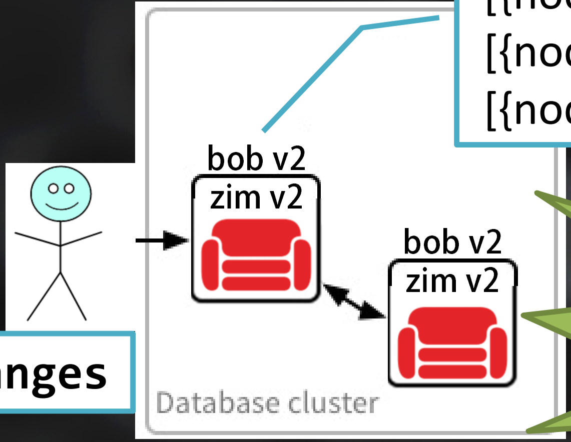
## 9. Counting with `_rev` / `seq`



00:07.20

# 9. Counting with `_rev` / `seq`

[{node1, 00-79, 1}, {node1, 80-FF, 1}]
[{node1, 00-79, 1}, {node2, 80-FF, 1}]
[{node2, 00-79, 1}, {node1, 80-FF, 1}]
[{node2, 00-79, 1}, {node2, 80-FF, 1}]



GET /db/_changes

Any one of these 4 responses is possible & correct!

00:07.30

## 9. Responsibly using `seq` values

1. `GET /{db}/_changes` one line at a time
2. Process each row **idempotently.**
   – That means apply the change independent of other rows, or their ordering
3. Periodically store the `seq/last_seq` value of the last row you processed
4. If you crash, restart: `GET /{db}/_changes?seq={value}`

## 9. Responsibly using `_rev` values

CouchDB is eventually consistent.

Absolute document ordering is not a guarantee.

**Remember**: Compaction, and internal or external replication, can and will remove intermediate document revs!

Last ditch option: *q*=`1`, *n*=`1` (no clustering, not scaleable)

## What about SQL SEQUENCE?

Again, CouchDB is eventually consistent.

CouchDB does not provide a guaranteed, globally unique, monotonically increasing sequence number.

Use UUIDs instead.

- `GET /_uuids` is convenient!

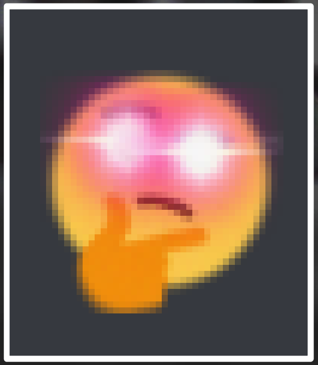
# Operations

Toronto Highway RESCU Operations

## 10. “Monitoring? CouchDB?”

“I monitor the host, but not CouchDB itself.

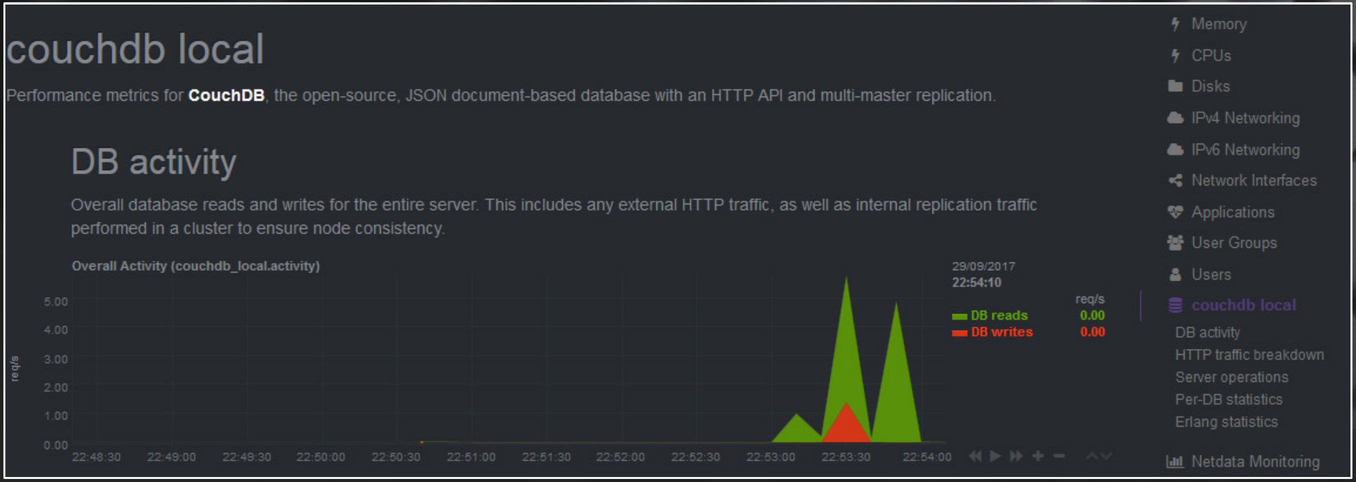
“It is self-healing, right?”



## Easy option: NetData

Per-node web service, fixed RAM & CPU usage

Can feed into most back-ends



couchdb local

Performance metrics for **CouchDB**, the open-source, JSON document-based database with an HTTP API and multi-master replication.

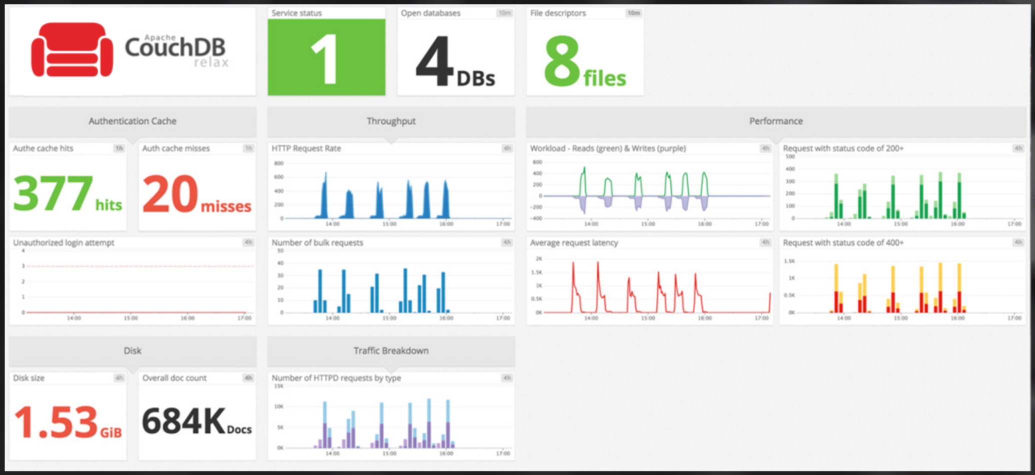
DB activity

Overall database reads and writes for the entire server. This includes any external HTTP traffic, as well as internal replication traffic performed in a cluster to ensure node consistency.

http://my-netdata.io/

## Easy option: Datadog

Datadog has native CouchDB integration:



https://docs.datadoghq.com/integrations/couch/

## Easy option: AWS CloudWatch

Neighbourhoodie releases AWS CloudWatch:

https://github.com/neighbourhoodie/aws-couchwatch

ALv2 *of course!*

## Monitoring CouchDB

Per-node endpoints you should track & graph:

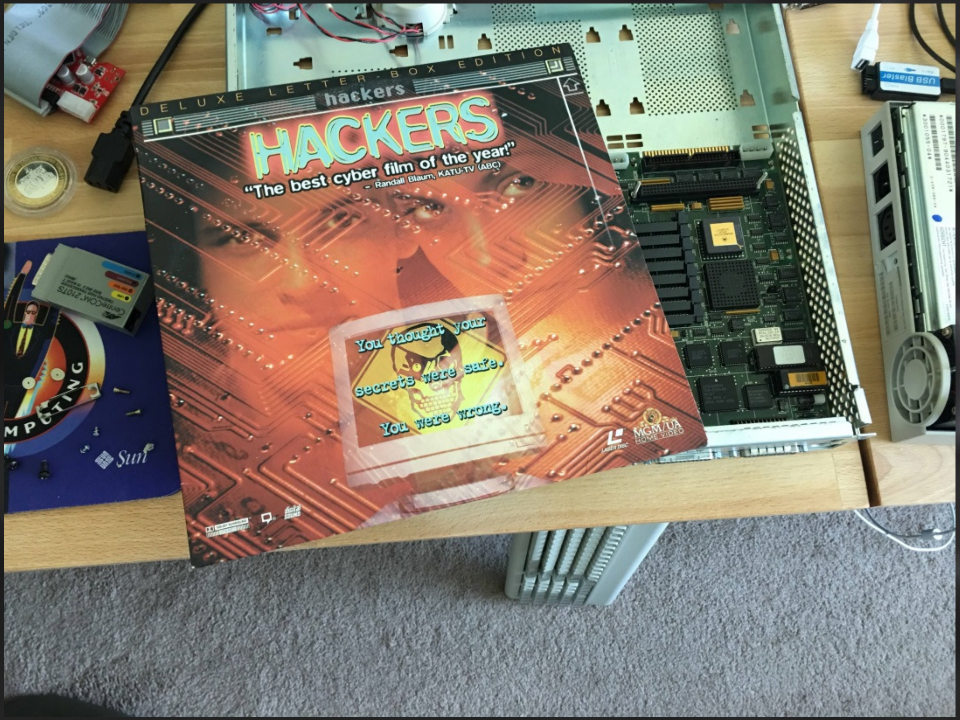
`GET /_node/_local/_stats`

– CouchDB specific data

`GET /_node/_local/_system`

– Erlang and OS-level data

## See Inside the Couch


DELUXE LETTER-BOX EDITION
hackers
HACKERS
"The best cyber film of the year!"
- Randall Blaum, KATU-TV (ABC)
You thought your secrets were safe. You were wrong.
MGM/UA HOME VIDEO
USB Blaster
Sun

…more to come soon from Neighbourhoodie!

## Thank you for listening!

Joan Touzet ❦ https://atypical.net/ ❦ `wohali`