OfflineFirst Apps With PouchDB and CouchDB

Lorna Mitchell, IBM

OfflineFirst Apps

Offline is not an error condition

"Offline capability is a key characteristic of modern Progressive Web Applications"
- http://offlinefirst.org

Being Offline

Could include:

- being in a place without internet infrastructure
- being without a data package on your phone
- being on the tube or a plane

Being Offline

Could include:

- being in a place without internet infrastructure
- being without a data package on your phone
- being on the tube or a plane
- getting the train from Huddersfield to ... really anywhere

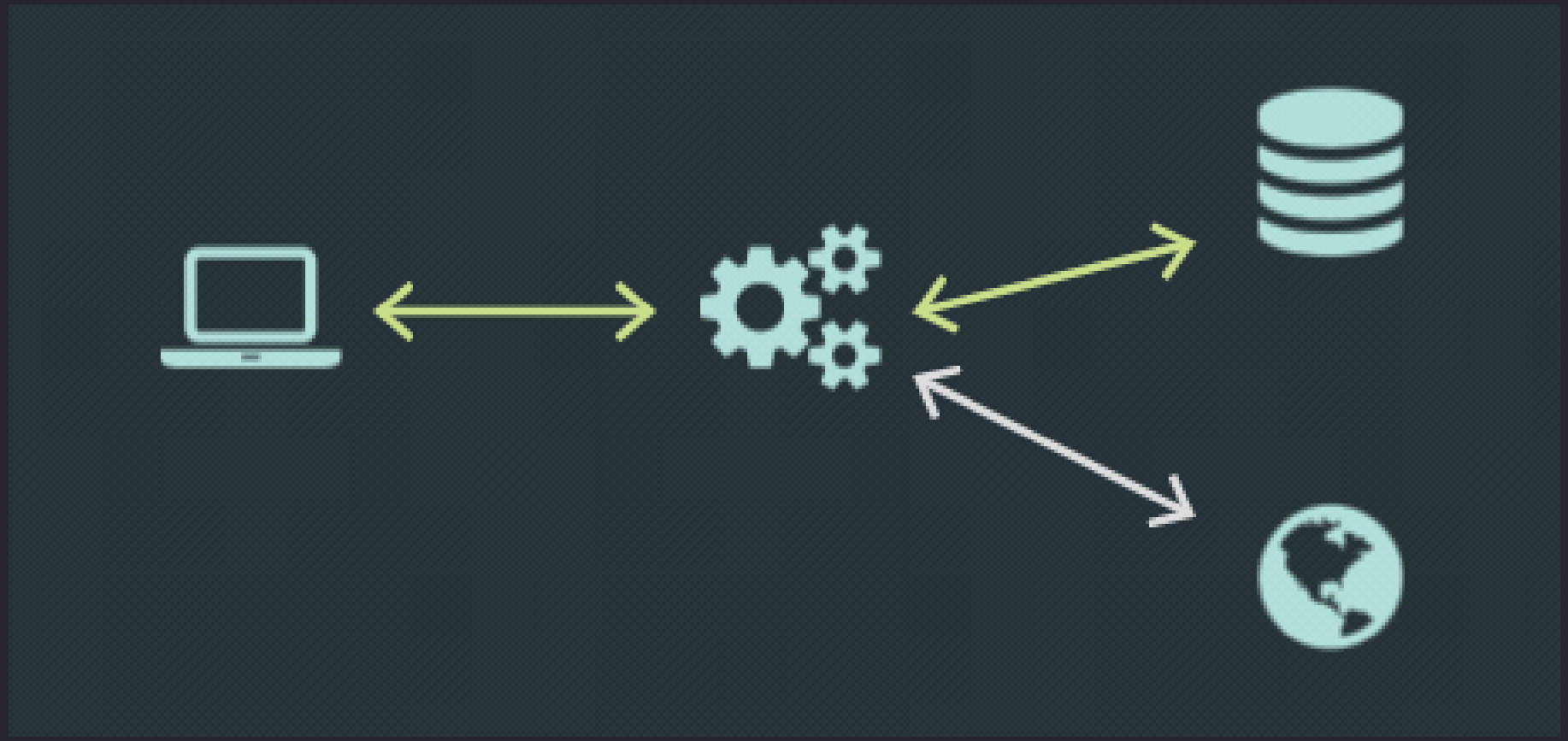
OfflineFirst

OfflineFirst means failing gracefully

You'll also hear PWA which is a Progressive Web App, covering more than just network failures

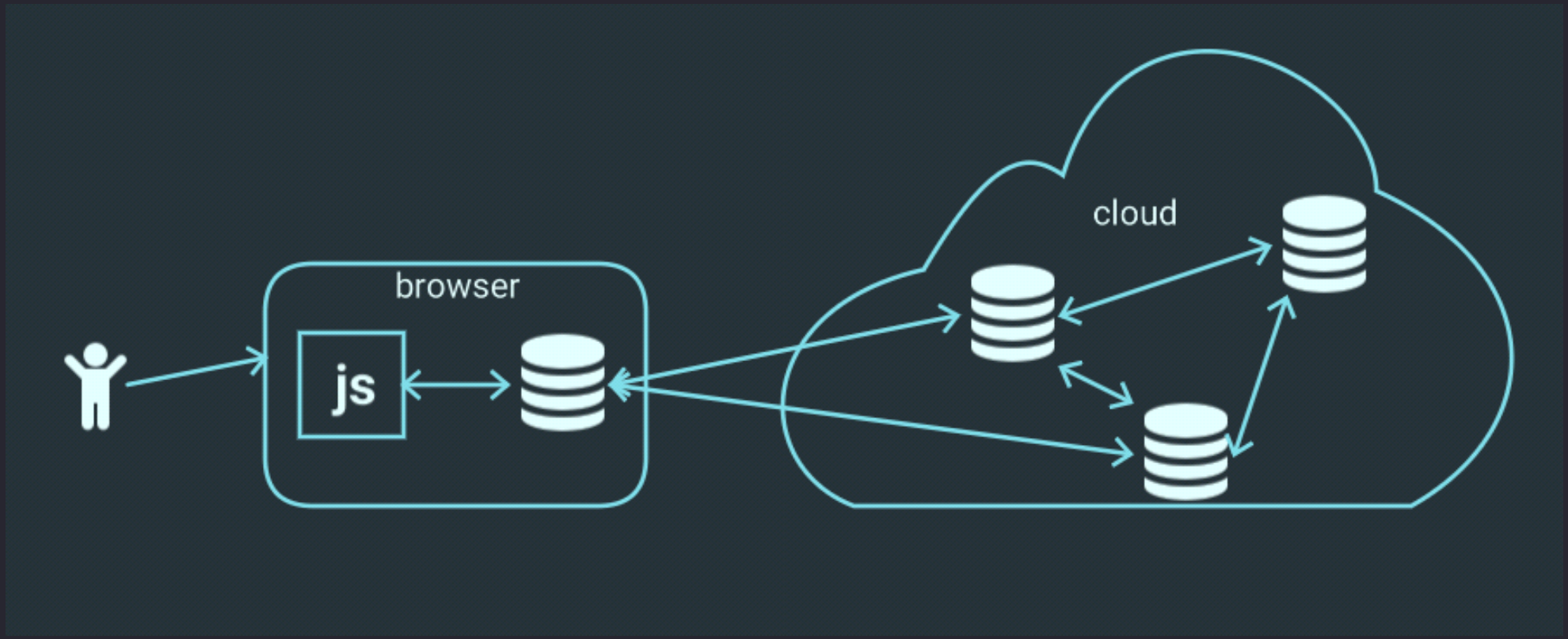
Achieving OfflineFirst: Code

Service Worker caches on first page load



Achieving OfflineFirst: Data

Client-side app and storage, background sync



PouchDB and CouchDB

- **https://pouchdb.com/**
  - A database that your client-side javascript can use
  - Can also sync to CouchDB (-compatible) databases

- **https://couchdb.apache.org**
  - A NoSQL document database
  - Best replication on the planet (probably)
  - HTTP API = good support in all languages

Example App: Shopping List

- Client-side JavaScript with PouchDB
- Works locally
- If connected, syncs to Cloudant/CouchDB
- Code here:
  https://github.com/lornajane/robust-shopping-list

PouchDB in Action

In `index.html`:

```
<script src="/js/pouchdb-6.1.2.min.js"></script>
<script src="/js/shopping.js"></script>
```

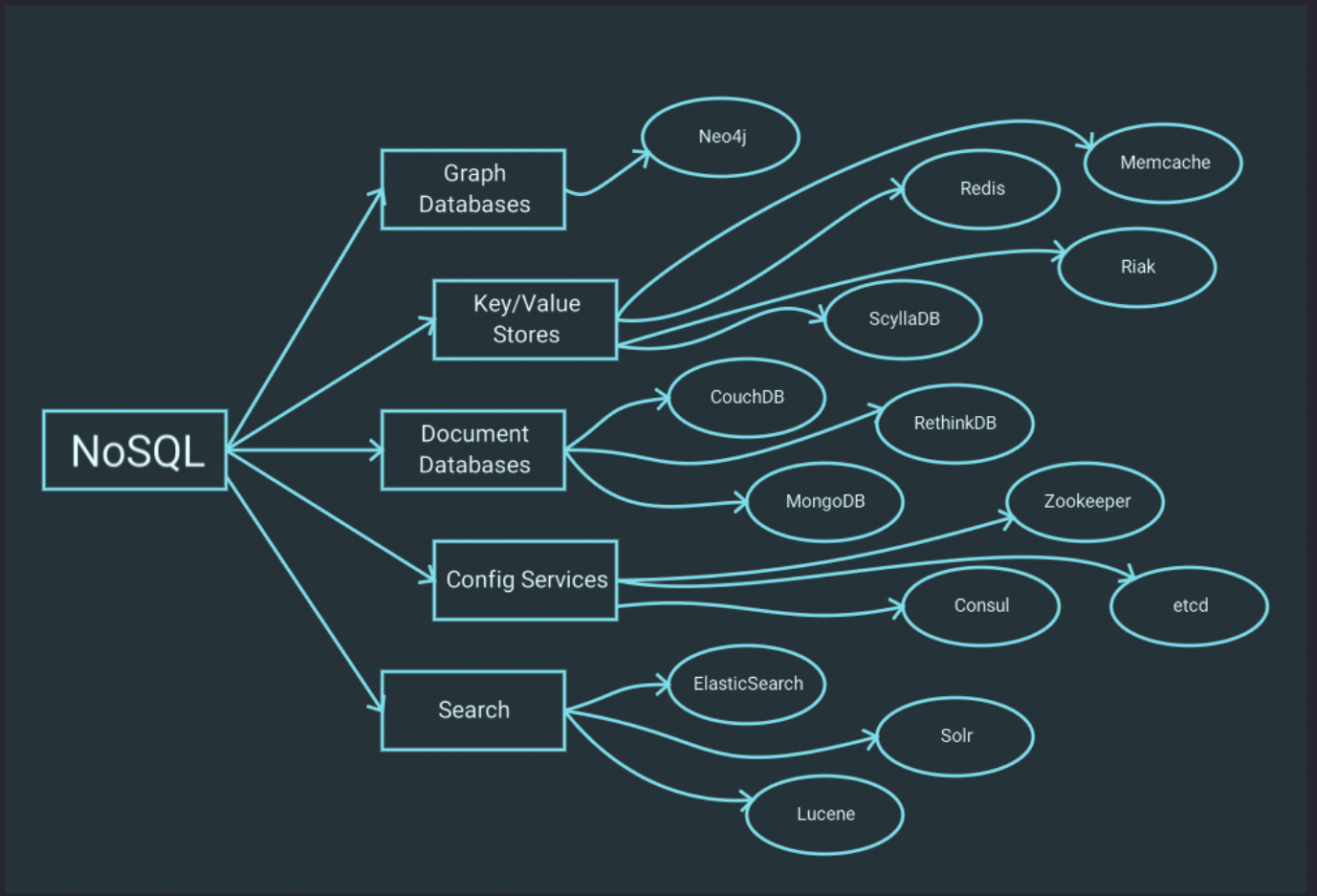
`shopping.js` is where my client-side JavaScript lives

PouchDB in Action

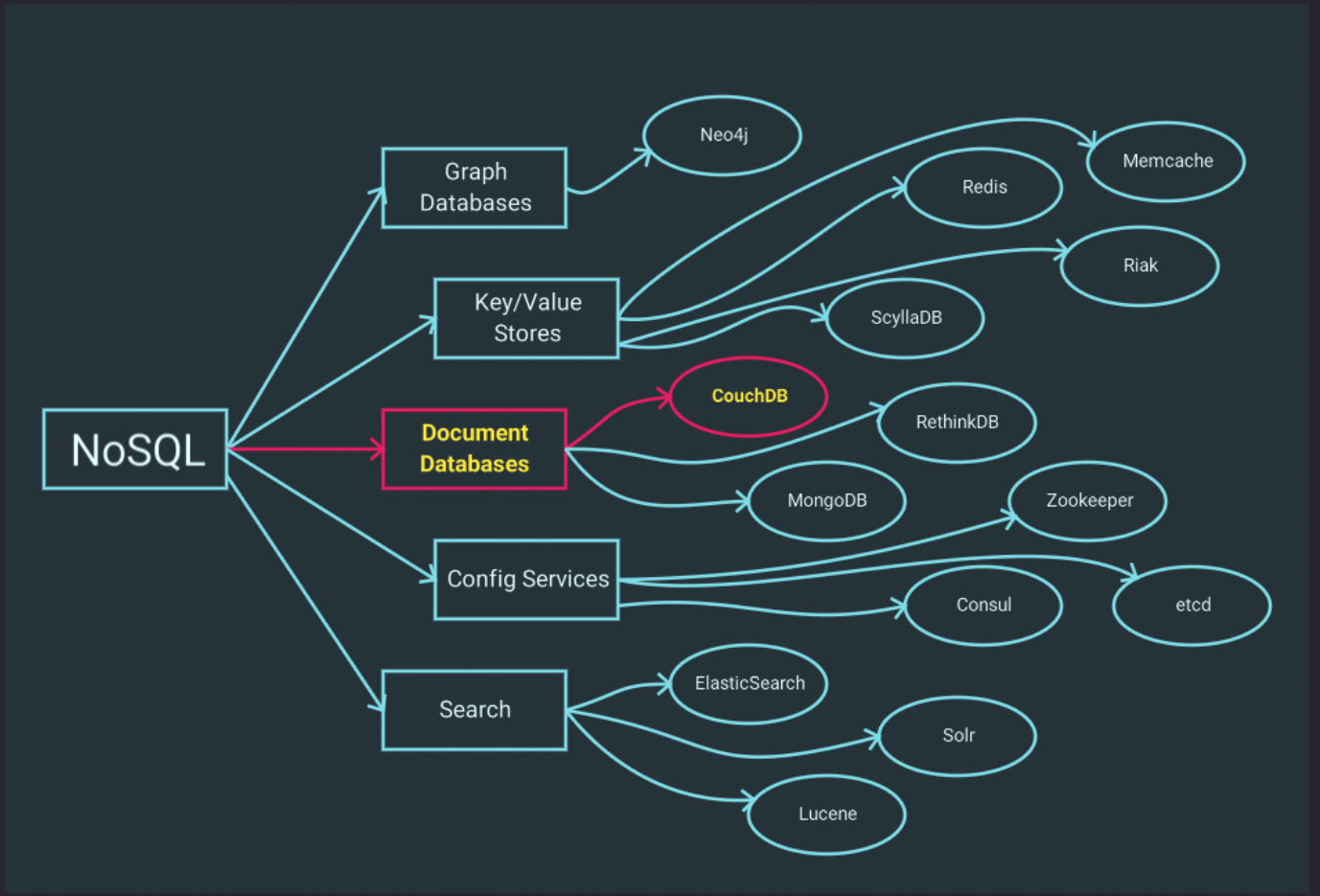
```
var db = new PouchDB('shopping');
var remoteDB = new PouchDB('http://localhost:5984/shopping');
window.onload = function() {
    db.sync(remoteDB, { live: true, retry: true }
    ).on('change', function (change) {
        return getItemList().then(function (contents) {
            document.getElementById('itemList').innerHTML = con
        })
    }).on('active', function (info) {
        return getItemList().then(function (contents) {
            document.getElementById('itemList').innerHTML = con
        });
    });
```

NoSQL Document Database

NoSQL Document Database

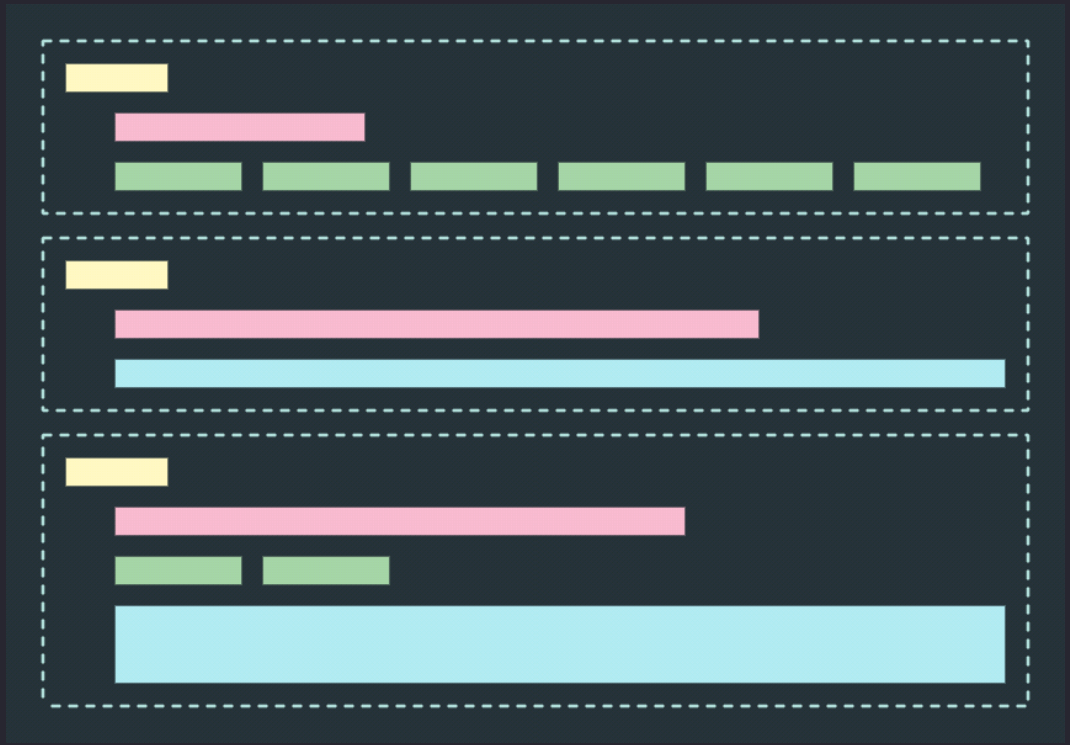


NoSQL Document Database



Document Databases

Store collections of schemaless documents



Document Databases

Choose a document database if:

- the records you store don't have the same structure as one another
- you need to change data structures without downtime
- you like high availability

Data Design for PouchDB

- data structure: include nested data/array, omit empty fields
- identifiers: pick a meaningful ID where appropriate
- beware updating/appending data: these cause conflicts

CouchDB

Cluster Of Unreliable Commodity Hardware

- HTTP API
- JSON data format
- Performant views use JavaScript and MapReduce
- Ad-hoc queries with a JSON structure using Mango

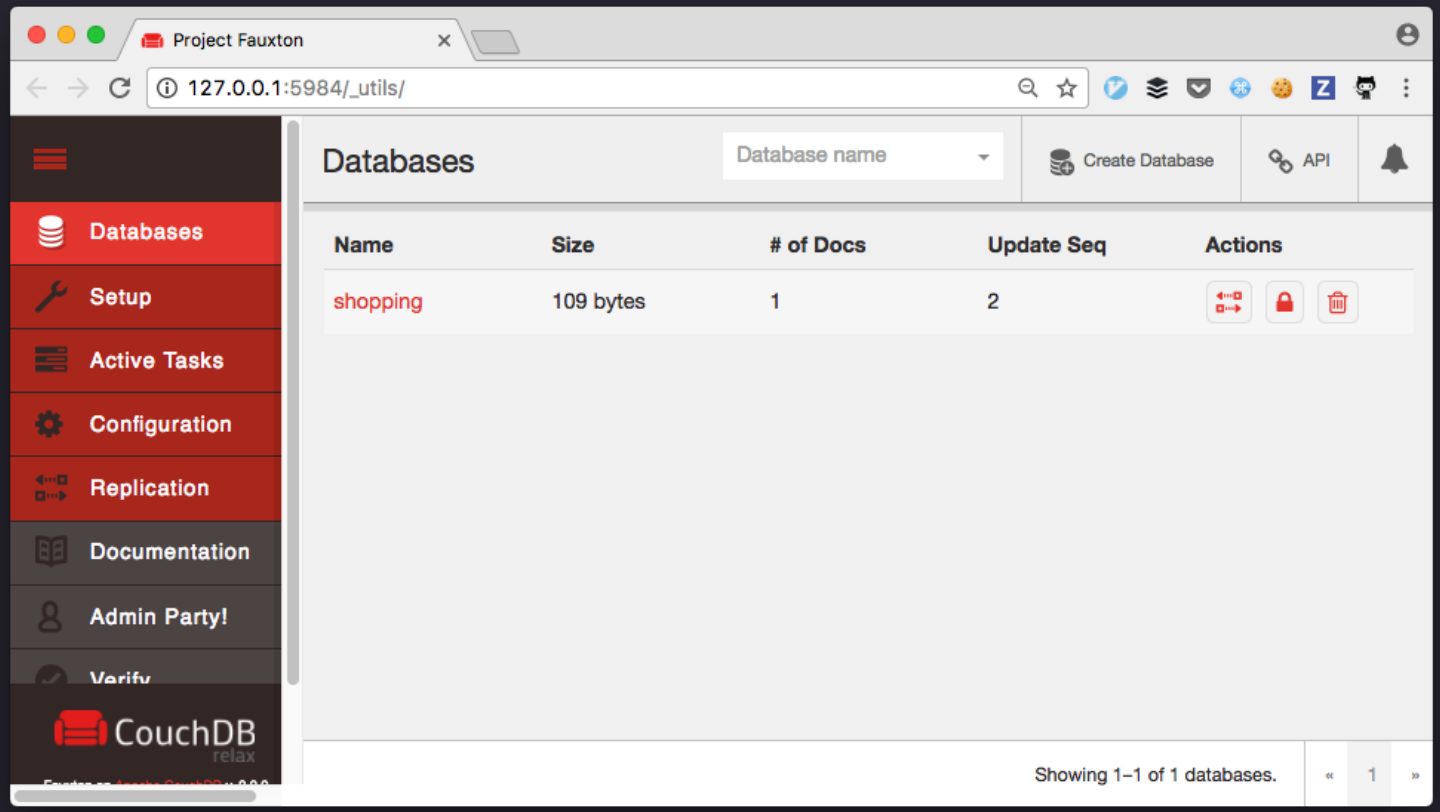
Curl and Not-Curl

- love `curl`? (https://curl.haxx.se/)
    - try `jq` (https://stedolan.github.io/jq/)
- hate `curl`? Try one of these
    - http-console https://github.com/cloudhead/http-console
    - Postman https://www.getpostman.com/
- for more, try this HTTP Tools post (and comments): http://lornajane.net/posts/2017/http-tools-roundup

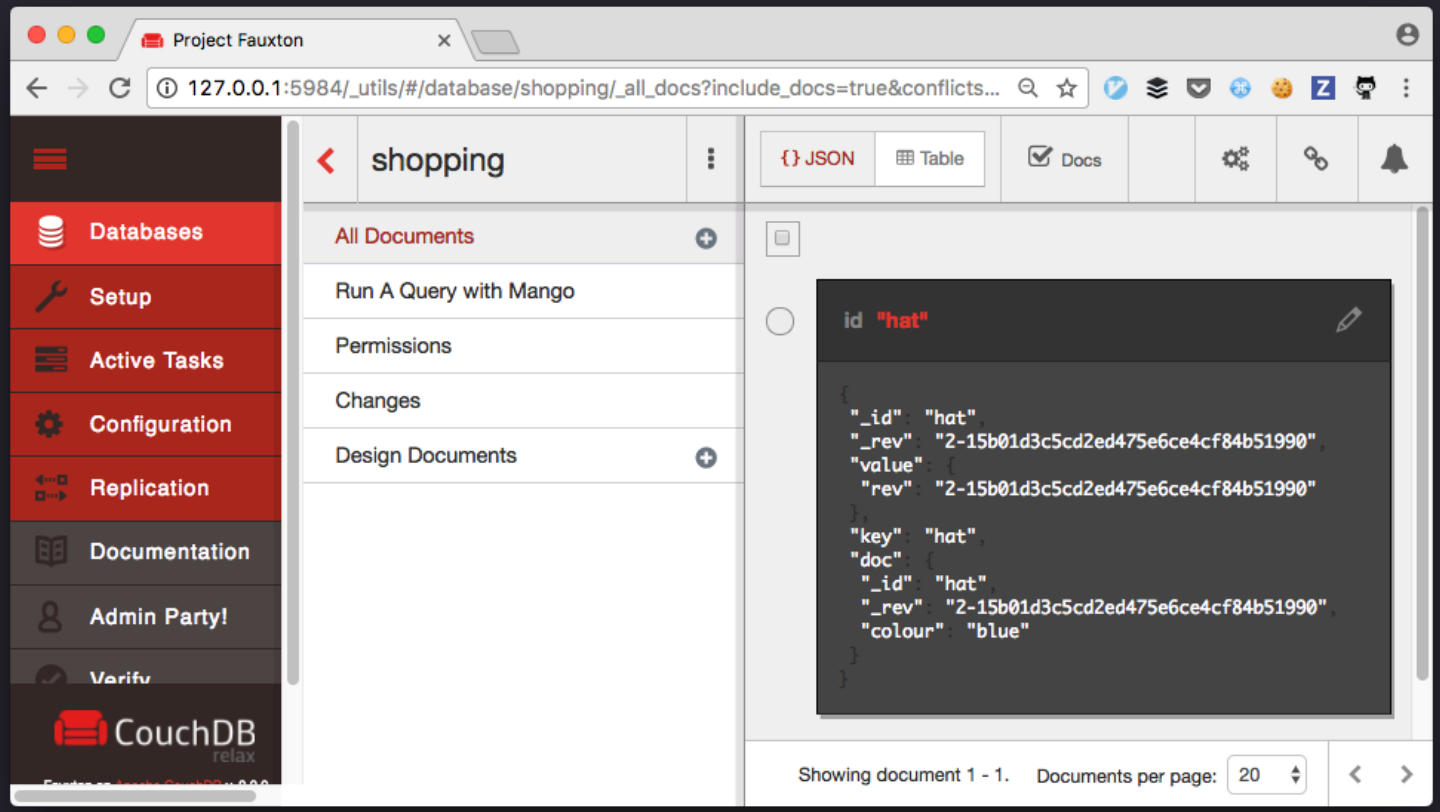
Fauxton

Friendly web interface

Fauxton


Project Fauxton
127.0.0.1:5984/_utils/
Databases
Database name
Create Database
API
Databases
Setup
Active Tasks
Configuration
Replication
Documentation
Admin Party!
Verify
CouchDB
relax
Name
Size
# of Docs
Update Seq
Actions
shopping
109 bytes
1
2
Showing 1–1 of 1 databases.
1

Fauxton



@lornajane

CouchDB: Lovely Doc DB

I could stop here:

- JSON format
- HTTP interface and nice web UI
- Scales well
- Modern, performant document database

Changes Feed

A feed containing all database changes.

```
GET /_changes
```

Replication

Replication

- Replication can be in either direction - or both
- Can be one-off, or continuous
- Other CouchDB-compatible storage also exists
    - e.g. PouchDB, a JavaScript implementation

Conflicts

Change docs in both places, replicate again:

```
87bf-bluemix.cloudant.com:443/shopping> GET /hat?conflicts=true
```

```
{
        _id: '123',
        _rev: '4-ecbc38075f9a8535c123e523519613b9',
        item: 'cheese',
        _conflicts: [ '3-0bb689d59034fb769d99dcf697ae2de7' ]
}
```

CouchDB will always choose the same "winning" doc

Conflicts

Fetch the "losing" doc(s) with `?rev=` parameter

```
87bf-bluemix.cloudant.com:443/shopping> GET /123?rev=3-0bb689d5903
```

```
{
        _id: 123,
        _rev: '3-0bb689d59034fb769d99dcf697ae2de7',
        item: 'cheddar cheese'
}
```

CouchDB doesn't store old revisions forever

Mango

Mango: CouchDB Queries

Mango is a mongo-like query language, useful for ad-hoc querying

It is a JSON structure containing:

- **Selector:** the criteria to match records on
- **Fields:** which fields to return
- **Sort:** what order you'd like that in (use with **Skip**)
- **Limit:** how many records (default = 25)

Mango: Example Query

Use a query like this with the `_find` endpoint

```
{
    "selector": {
        "Year": {"$eq": "2012"}
    },
    "fields": ["Quarter", "Product line"],
    "limit": 5
}
```

Mango: Example Query

```
$ curl -X POST -H Content-Type:application/json \
http://localhost:5984/products/_find --data @mango.json

{"warning":"no matching index found, create an index to optimize q
"docs":[
{"Quarter":"Q1 2012","Product line":"Mountaineering Equipment"},
{"Quarter":"Q1 2012","Product line":"Mountaineering Equipment"},
{"Quarter":"Q1 2012","Product line":"Mountaineering Equipment"},
{"Quarter":"Q1 2012","Product line":"Mountaineering Equipment"},
{"Quarter":"Q1 2012","Product line":"Mountaineering Equipment"}
]}
```

Mango: Indexes

Describe the index in JSON, then use the `_index` endpoint

```
{
"index": {
    "fields": ["Year"]
},
"name": "Year"
}
```

Mango: Indexes

```
$ curl -X POST -H Content-Type:application/json \
http://localhost:5984/products/_index --data @index.json
```

```
{
    "result": "created",
    "id": "_design/e9b54f2ac34b8823ccbe8aaf6f406d464f50f521",
    "name": "Year"
}
```

Check which indexes are used by putting `_explain` where the `_find` normally goes!

Views

Views

- Written in Javascript
- Use MapReduce
- The map results are stored
- Can be used either for filtering, or for aggregation

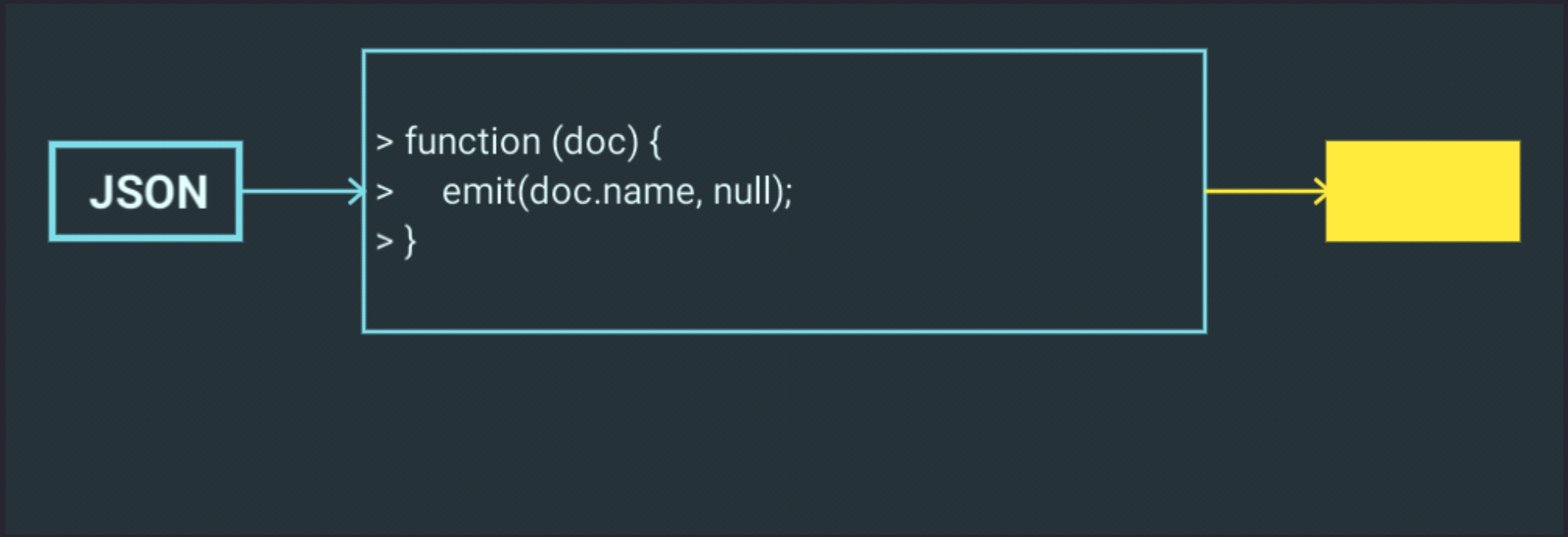
MapReduce Primer: Map

- Examine each document, "emit" 0+ keys/value pairs
- Scales well because each document is independent
- To filter a collection of documents, use map step only

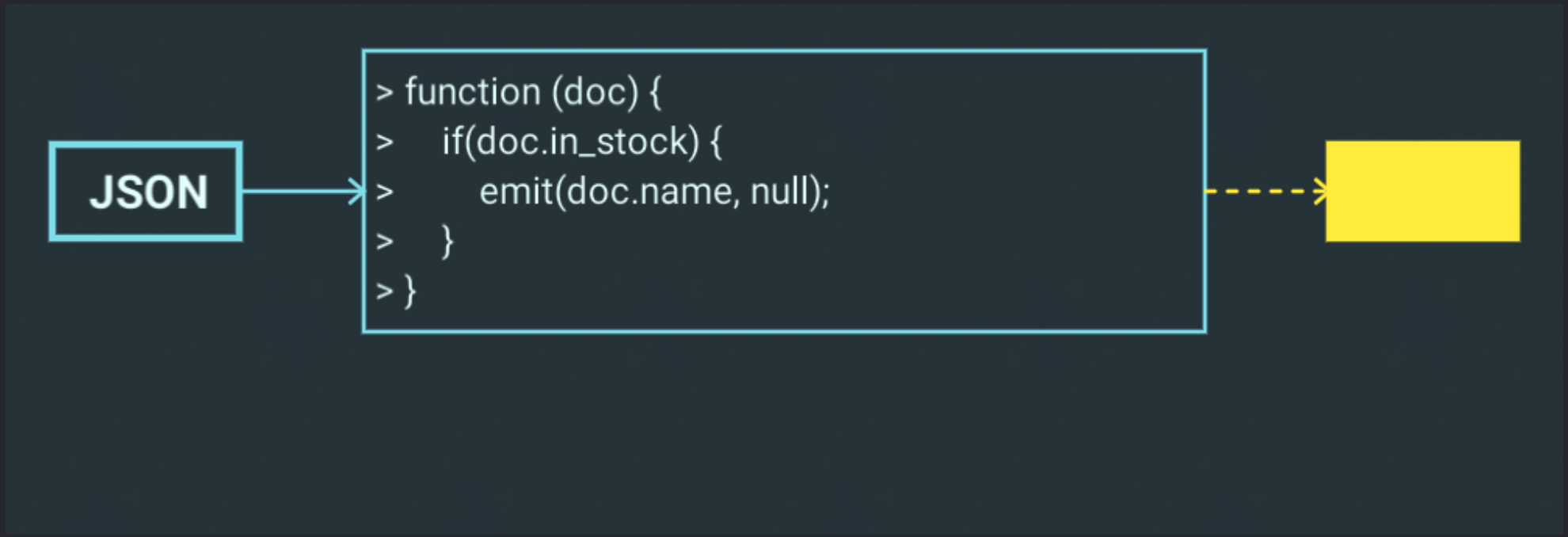
MapReduce Primer: Map

```
> {
>     product_name: shoe77,
>     departments: [outdoor, sports, womens],
>     in_stock: true,
>     date_added: 2017-03-16
> }
```

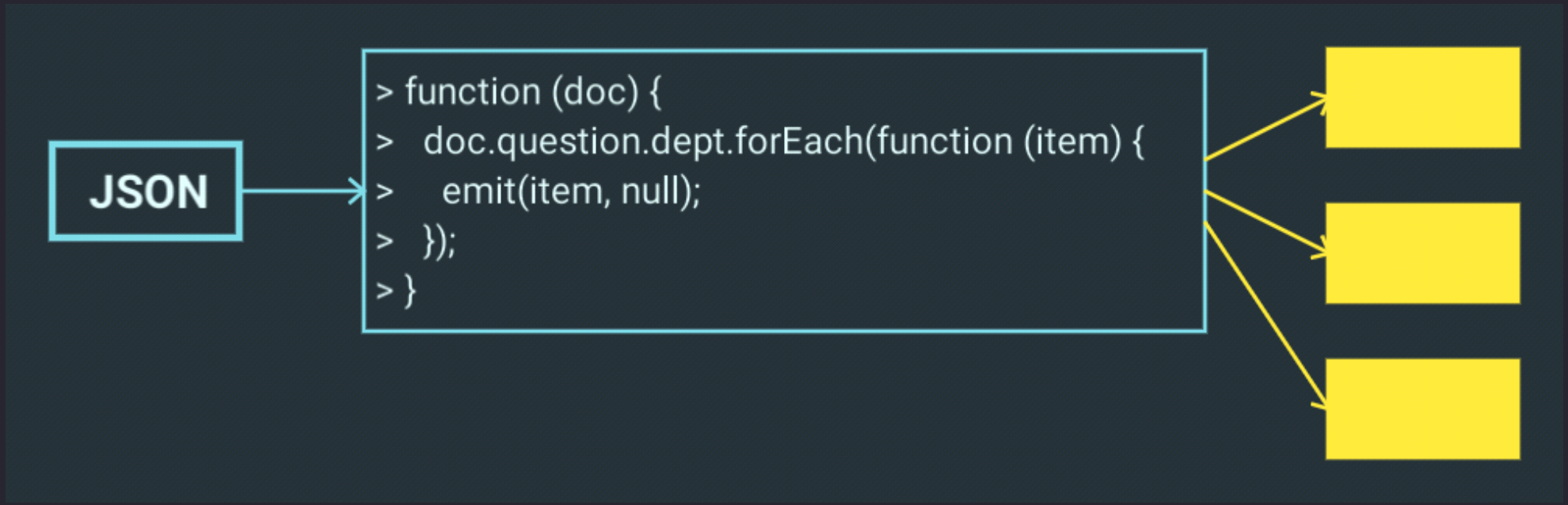
MapReduce Primer: Map



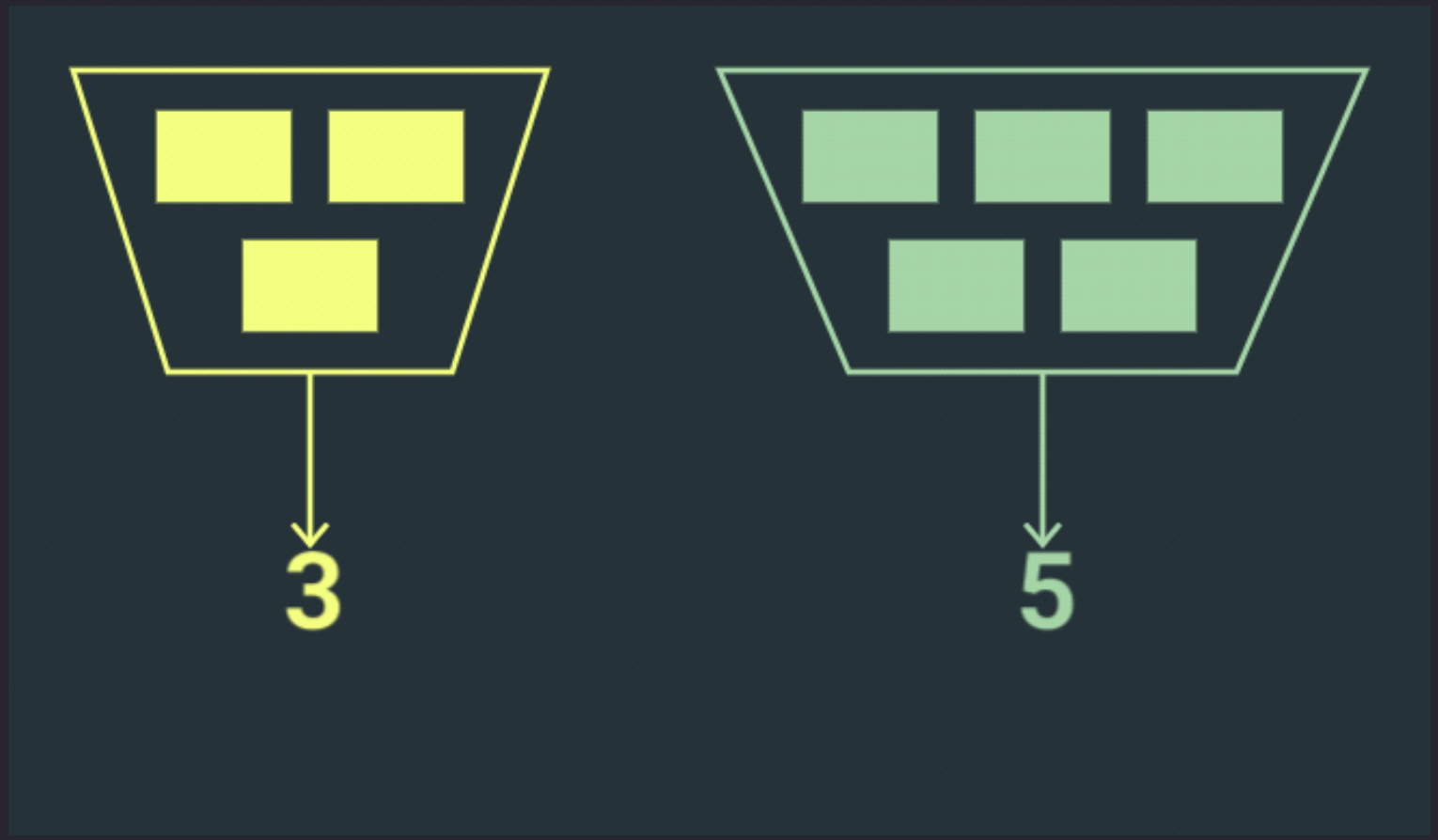
MapReduce Primer: Map



MapReduce Primer: Map



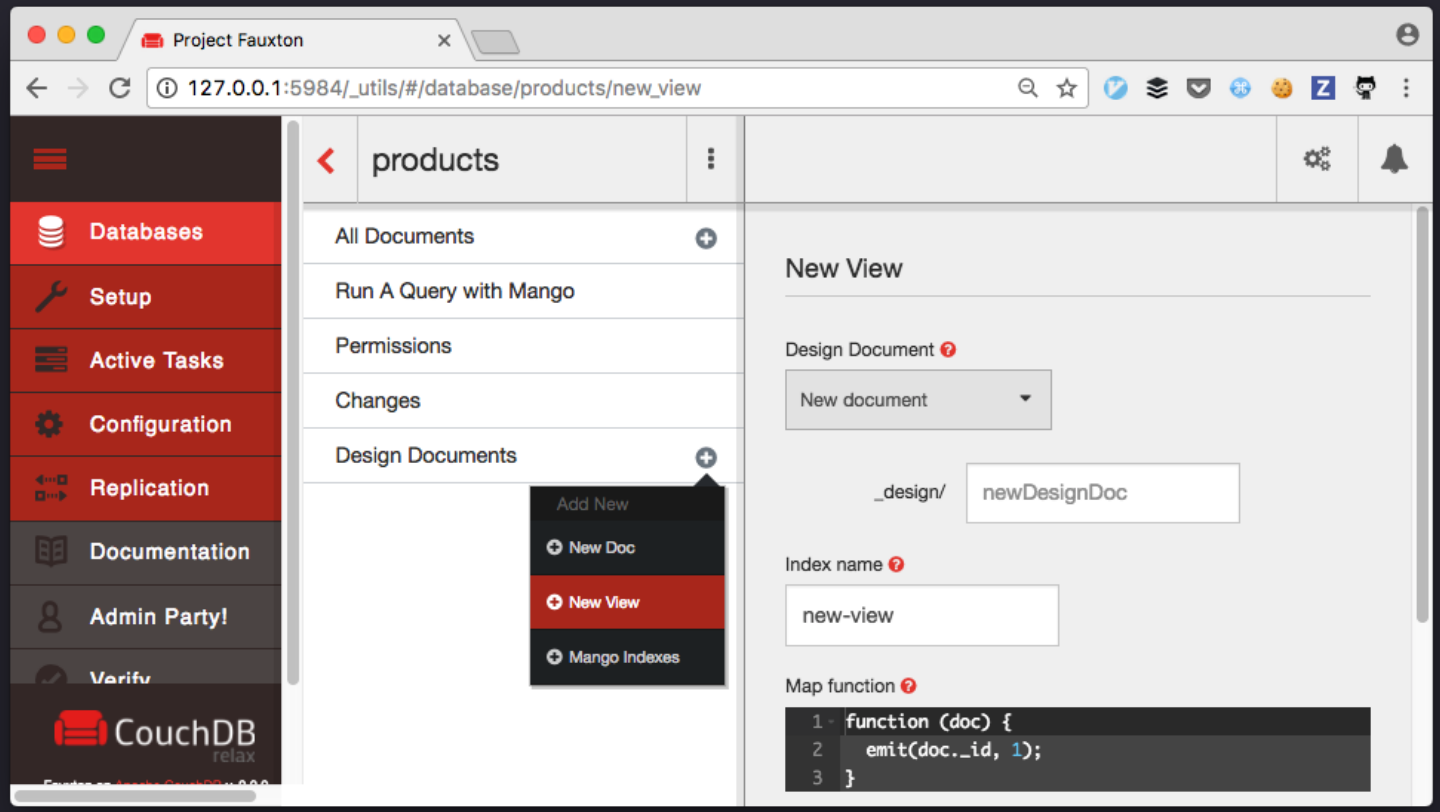
MapReduce Primer: Reduce



MapReduce Primer: Reduce

- "Reduce" values in batches with the same key
- CouchDB has useful built in functions for most things
- Use reduce step when you want aggregate data
    - (SQL equivalent: a query with `GROUP BY`)

Views Example



OfflineFirst Apps With PouchDB and CouchDB

@lornajane

Example Apps

Ready-made shopping list examples are available:

- VanillaJS and PouchDB (a more detailed example)
- Polymer and PouchDB
- React and PouchDB
- Vue.js and PouchDB
- React Native and PouchDB

https://github.com/ibm-watson-data-lab/shopping-list

Resources

- https://lornajane.net
- https://github.com/lornajane/robust-shopping-list
- https://github.com/ibm-watson-data-lab/shopping-list
- https://offlinefirst.org
- http://hood.ie/