ResinDB

A search-centric document database

Definition of a document database

- The core mechanism of any database is that of a **key/value** store.
- In a document store terms (i.e. key/value pairs) are grouped into documents.
- A document (a serialized business entity or graph) can be viewed upon as a **dictionary of key/value**, or a nested dictionary of dictionaries of string/object, much like a JSON document**.**
- A document database may be defined as a key/value store where the value is a document and the (primary) key is optional, a **key/document store**.
- A key/value store can respond to lookups by key. *I.e. what value did I store with this key?* The query is thus composed of a key.
- Stores and databases alike may **index the values** within a certain scope (where the scope might be a key, column, range or no scope i.e. there is a global scope) to be able to respond to lookups by value.
- Document databases instead keep an inverted index where **values are mapped not to keys but to documents** to be able to respond to document lookups. *I.e. what documents did I store that has this value in this column or field?* The query is thus composed of a document.

An inverted index

To fit inside an inverted index, this document...


```
{
   "label":  "universe",
   "description": "totality of planets, stars, galaxies, intergalactic space, or all matter or all energy"
}
```


...will be transformed into the following terms (and their count) *:

label/**universe** (1)

description/**totality of planets, stars, galaxies, intergalactic space, or all matter or all energy** (1)

The terms in a full-text search inverted index

label/**universe** (1)

description/**totality** (1)

description/**of** (1)

description/**planets** (1)

description/**stars** (1

description/**galaxies** (1)

description/**intergalactic** (1)

description/**space** (1)

description/**or (2)**

description/**all (2)**

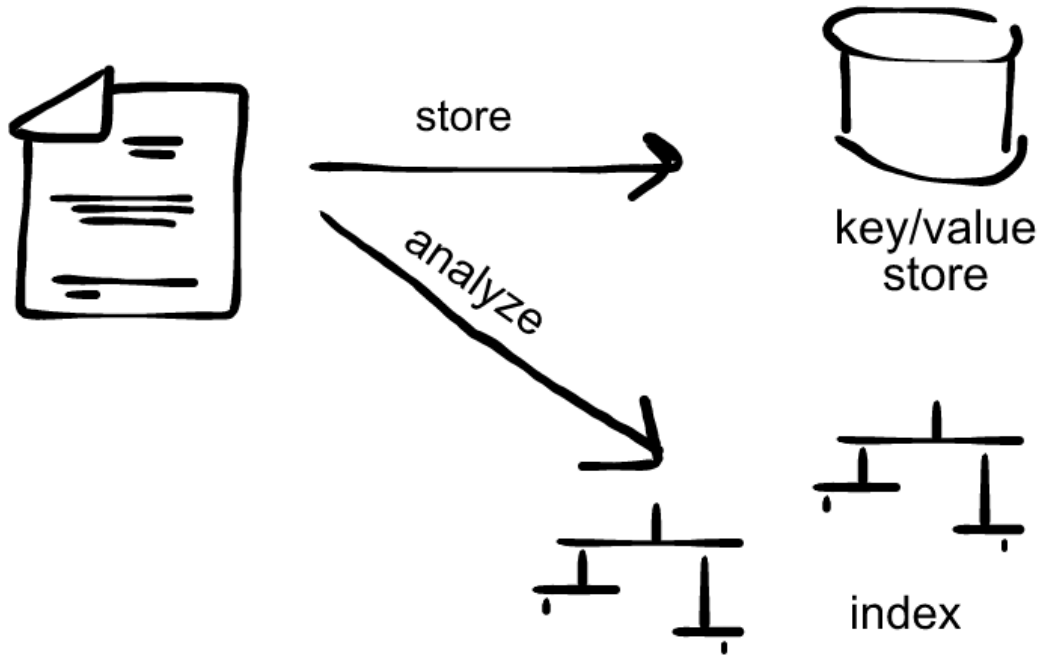
description/**matter** (1)

description/**energy** (1)

Conceptual model

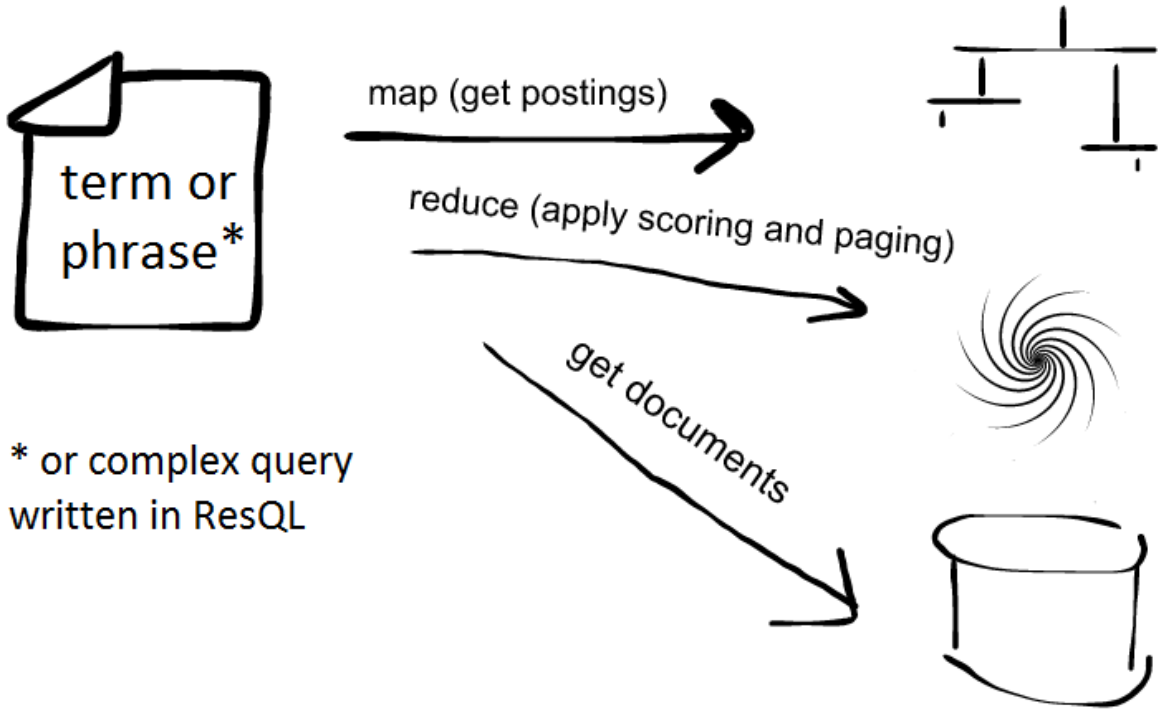
Write

Column-based indexing,
row-based compression



Read

Document-based querying,
snapshot and disk-based reading



* or complex query
written in ResQL

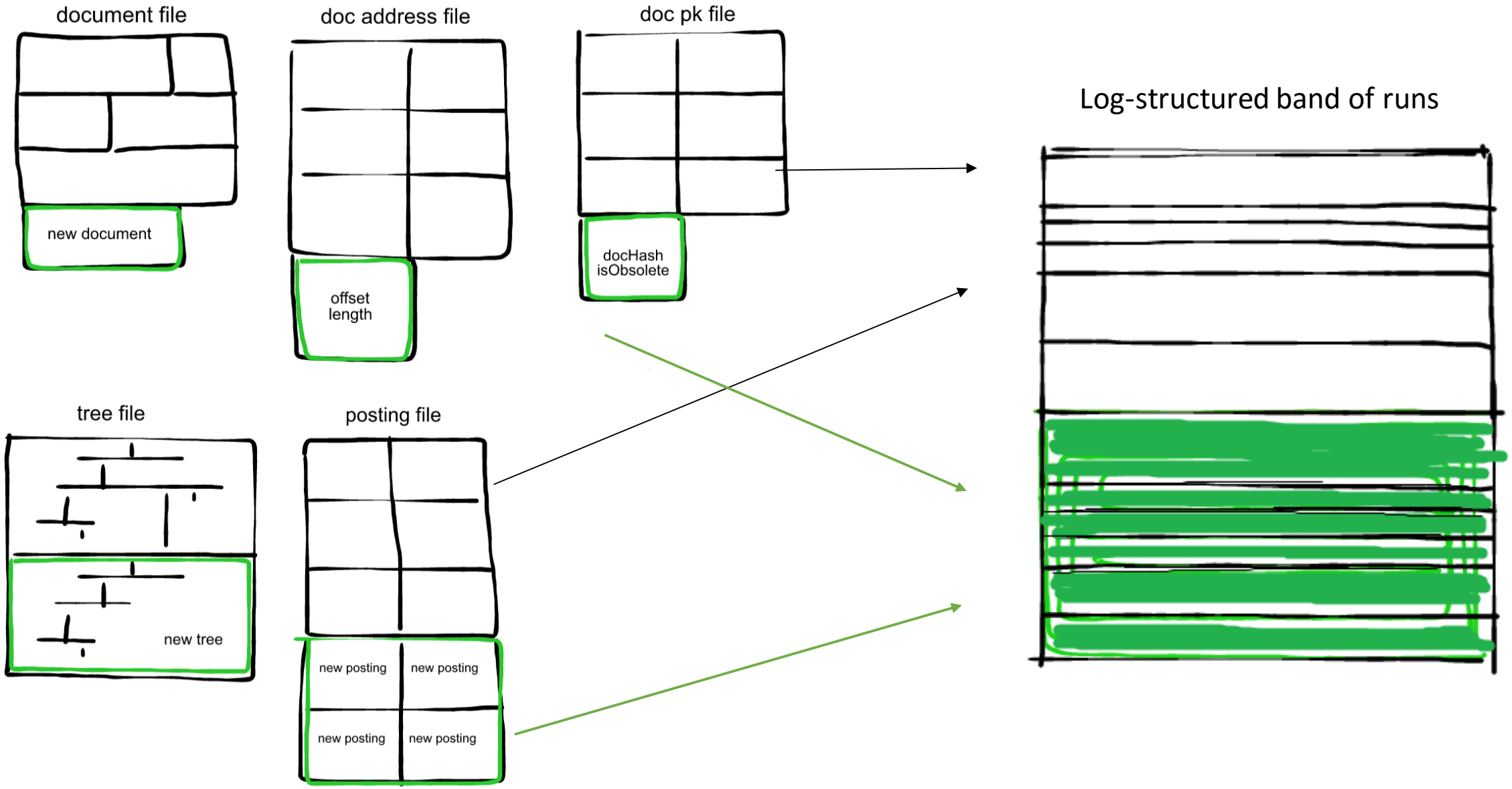
Writing/reading with LcrsTrie and LcrsNode

Resin's default index data structure is a binary character trie. It is **represented in memory** and during indexing by the LcrsTrie **and on disk** and at the time of query by the LcrsNode.

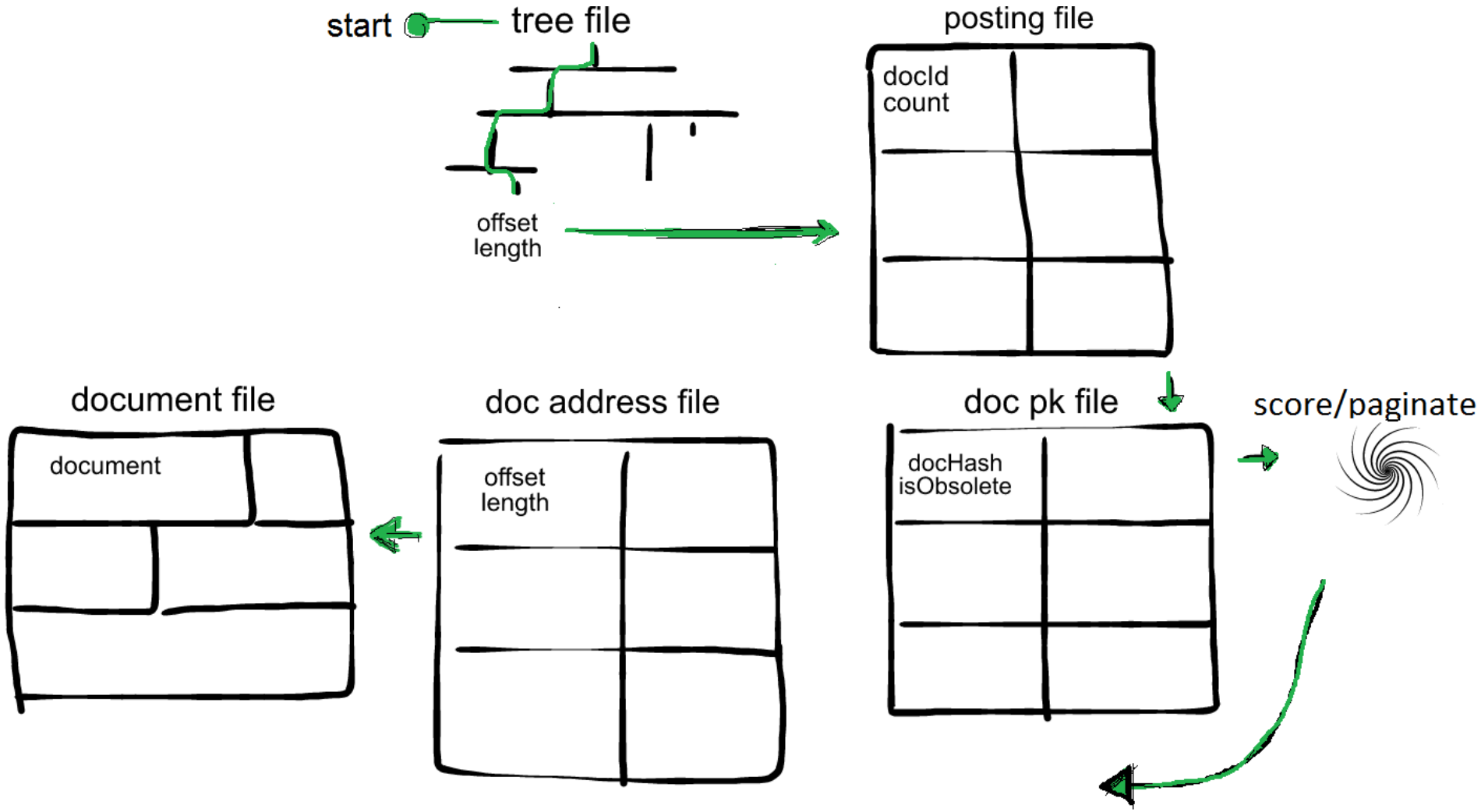
The LcrsNode offer the same binary search capabilities as a in-memory binary search tree but without having to load anything into memory except for the current tree node.

A index lookup is a sequential forward-only read of a bitmap.

Write model


document file
new document
doc address file
offset
length
doc pk file
docHash
isObsolete
tree file
new tree
posting file
new posting
new posting
new posting
new posting
Log-structured band of runs

Read model


start
tree file
offset
length
posting file
docId
count
document file
document
doc address file
offset
length
doc pk file
docHash
isObsolete
score/paginate

Vector space model and map/reduce

Query: "What is a cat?"

Parse into document: [what,is,a,cat]

Scan index: what

Scan index: is

Scan index: a

Scan index: cat

Found documents:

[(i), (have), a, cat],

[what, (if), (i), (am), a, cat]

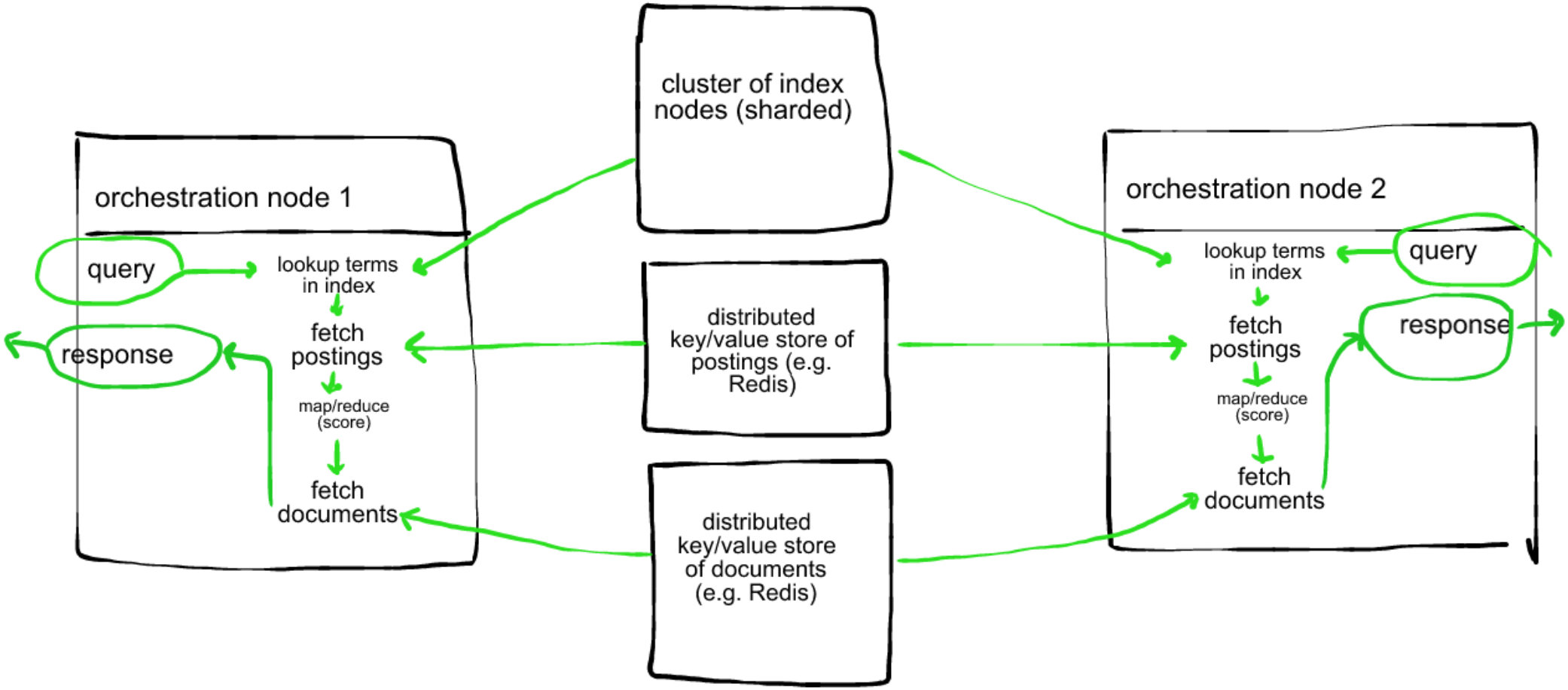
Normalize to fit into 4-dimensional space:
[what,is,a,cat]
[null, null, a, cat],
[what, null, a, cat]

Give each word a weight (tf-idf):
[0.2, 0.1, 0.1, 3],
[0, 0, 0.1, 3],
[0.2, 0, 0.1, 3]

Map the query and the documents in vector space, sort by the documents' (Euclidean) distance from the query document, paginate and as a final step, **fetch documents from the filesystem.**

Distributed model

Resin over gRPC



.Net Core/C#

How a indie developer keeps pace, feature- and performance-wise, with DocumentDb, RocksDB and Bigtable.

;)