

# ResinDB

A search-centric document database

# A document database

- The core mechanism of a database is that of a **key/value** store.
- The principal difference between a key/value store and a document store is that in a document store terms (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 key/value store can respond to lookups by key (i.e. what value did I store with this key?). The query is composed of a key.
- Key/value stores and databases alike can **index the values** of a certain key to be able to respond to lookups by value.
- Document databases 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 contain this content?)

# An inverted index

## A document:

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

## The postings \*:

label/**universe** (1)

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

\* posting = a term reference, (a count) + a reference to the original document

# 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)

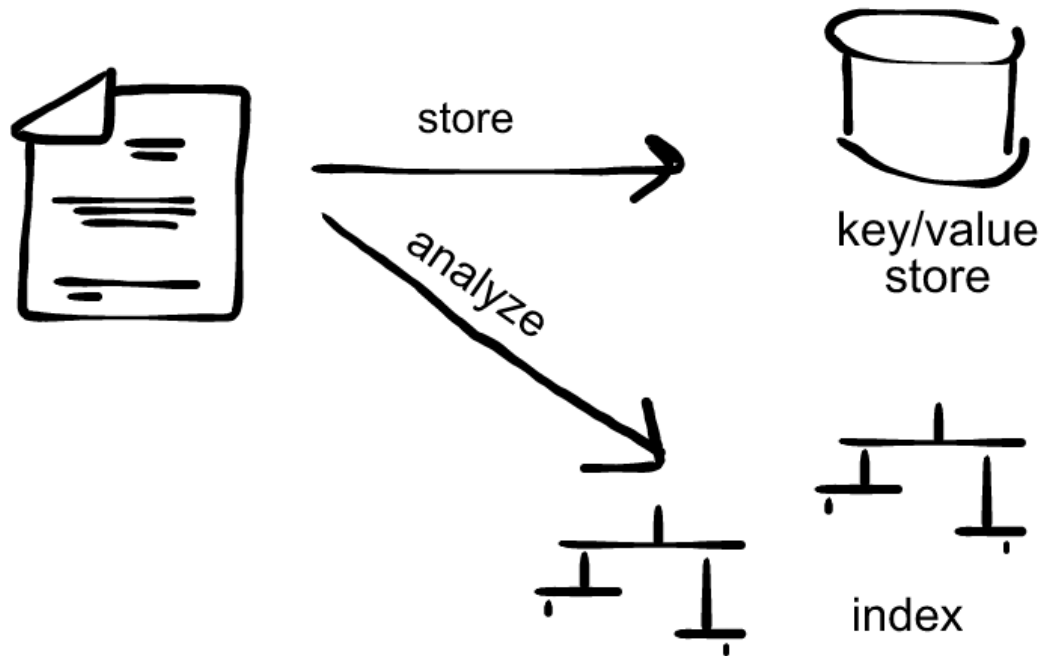
description/**matter** (1)

description/**energy** (1)

# Conceptual model

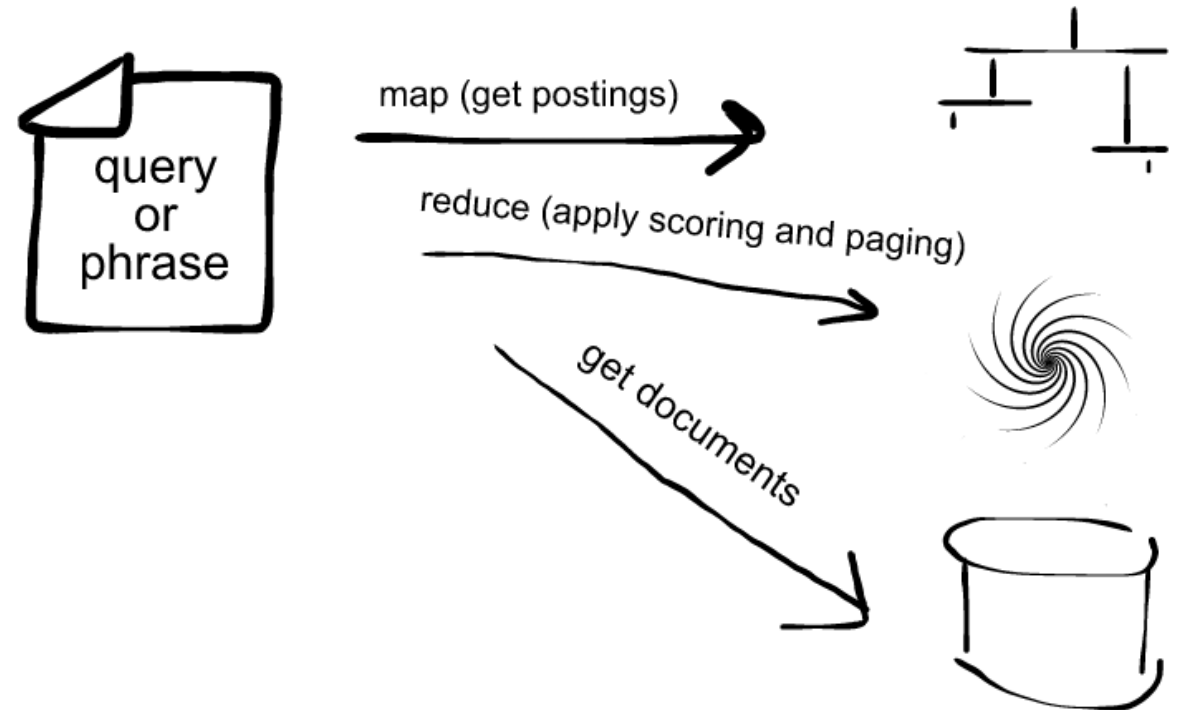
## Write

Column-based indexing,  
row-based compression

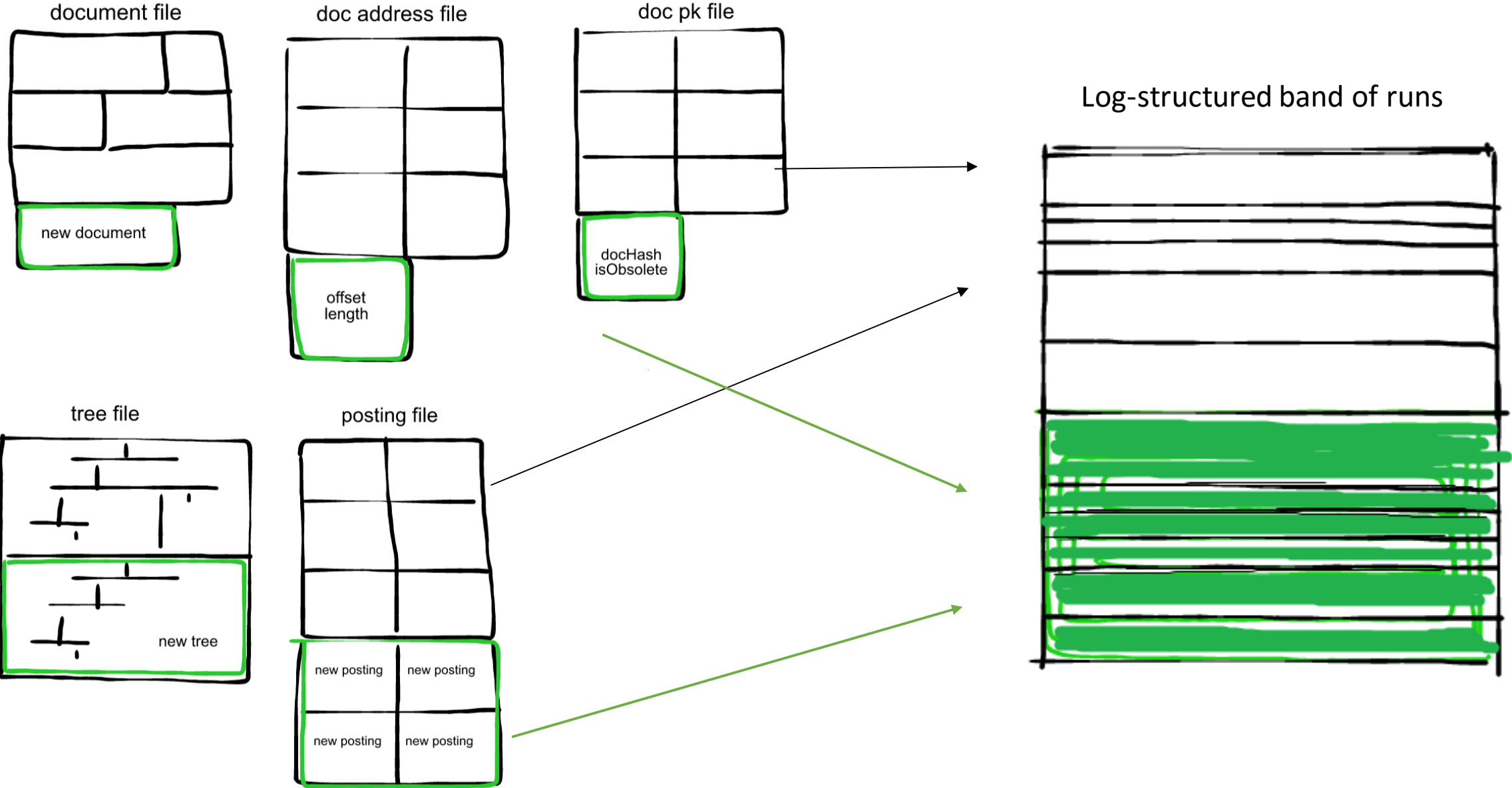


## Read

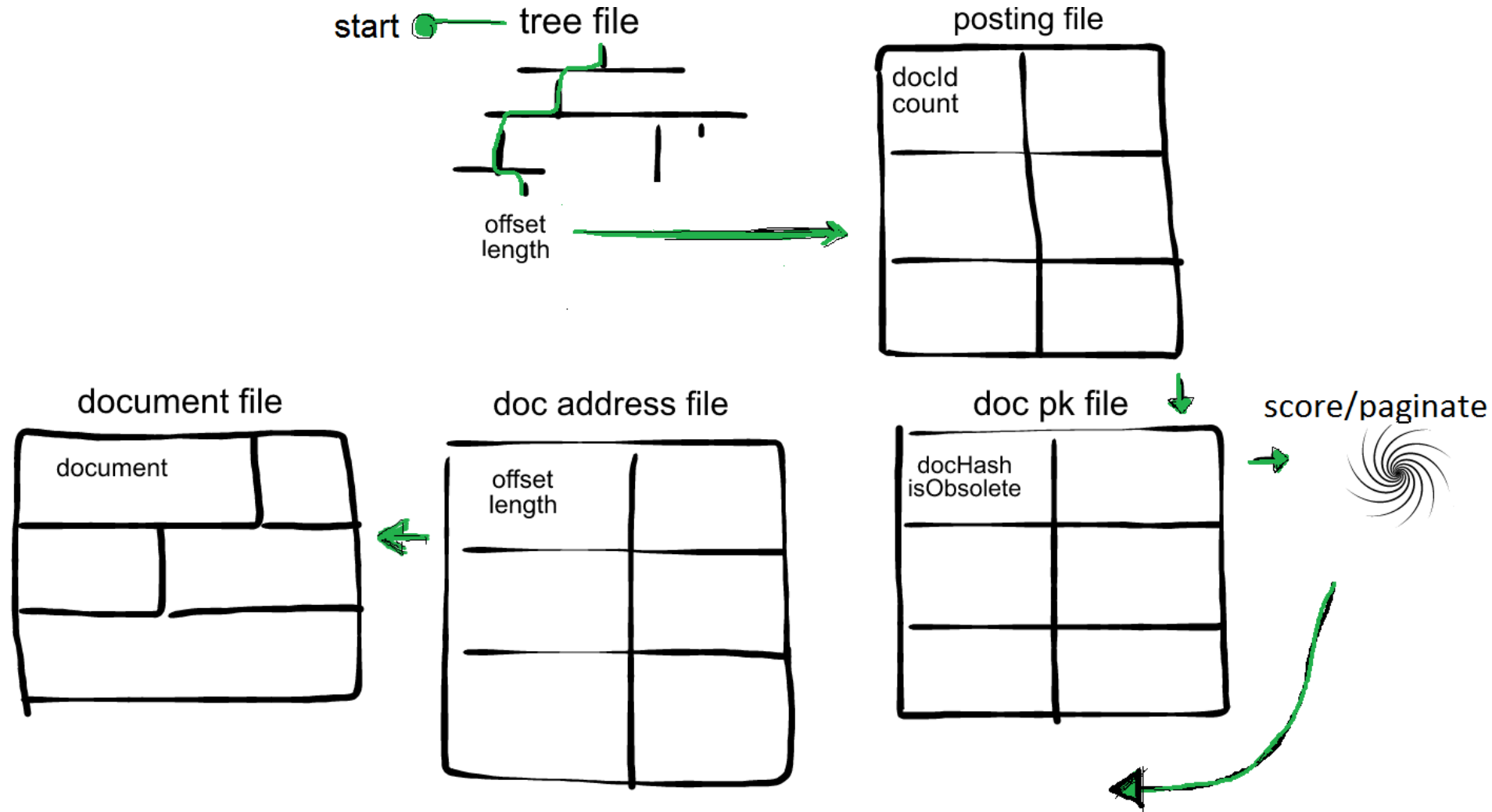
Term-based reading



# Write model



# Read model



# Vector space model

**Query:** "What is a cat?"

**Create query 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:**

[null, null, a, cat],  
[what, null, a, cat]

**Score the documents:**

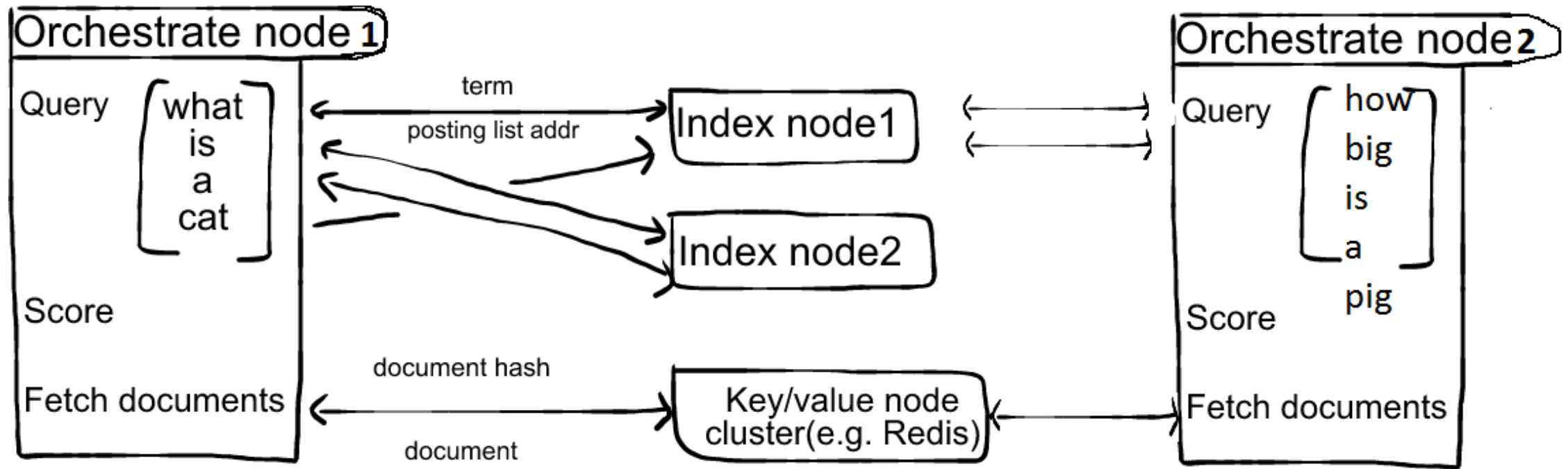
[0, 0, 0.1, 3],  
[0.2, 0, 0.1, 4]

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



# Left-child-right-sibling character trie

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 on-disk tree traversal is a sequential forward-only read of a bitmap.

# .Net Core/C#

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