

keyvi

**the key value index
optimized for size and speed**

Hendrik Muhs
@HendrikMuhs
hendrik.muhs@gmail.com

key value index

based on finite state (FST)

Opensource (Apache 2.0)

C++(core), Python(binding)

<http://www.keyvi.org>

keyvi in the wild

Browser integrated Search Engine

keyvi powers important parts in the backend



- > 14bn data points (key value pairs)

- > 2.7TB index size

- > 900tsd daily active users (> 10k requests per minute)

- < 60ms average latency

Do we need another key value store?

uses finite state

scales through shared memory

very space efficient

Know the tech you use

Please pick:

B-Trees & Co (*SQL, *DB, MongoDB)

Hash Tables (Redis, Cassandra)

Know the tech you use

Please pick:

B-Trees & Co (*SQL, *DB, MongoDB)

Hash Tables (Redis, Cassandra)

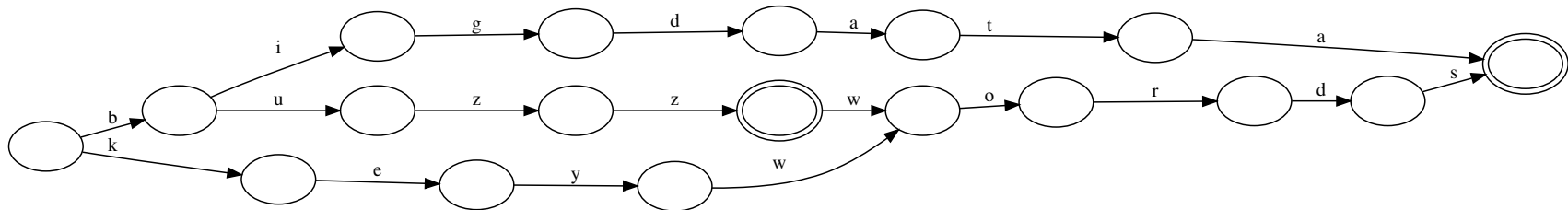
Know the tech you use

heap fragmentation
or garbage collector runs

increasing memory usage

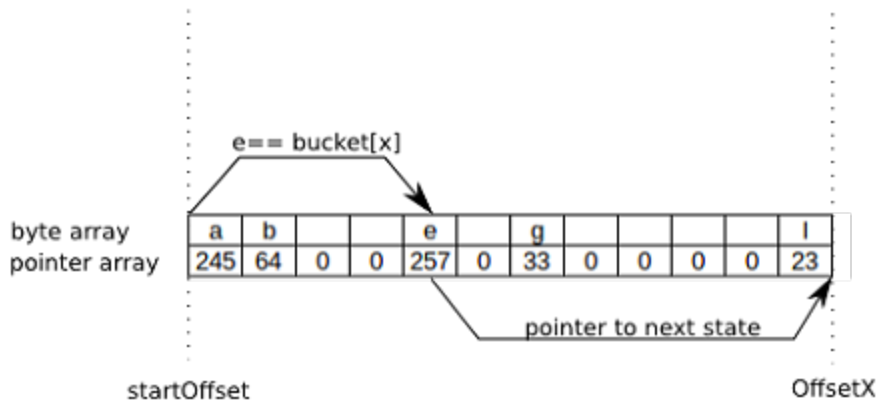
degrading performance

keyvi in a nutshell



finite state algorithm

sparse array persistence



Other FST implemations

Lucene

openfst, nltk, NLP toolkits

Lucene (Elasticsearch, Solr)
termdictionary, suggesters

Solr

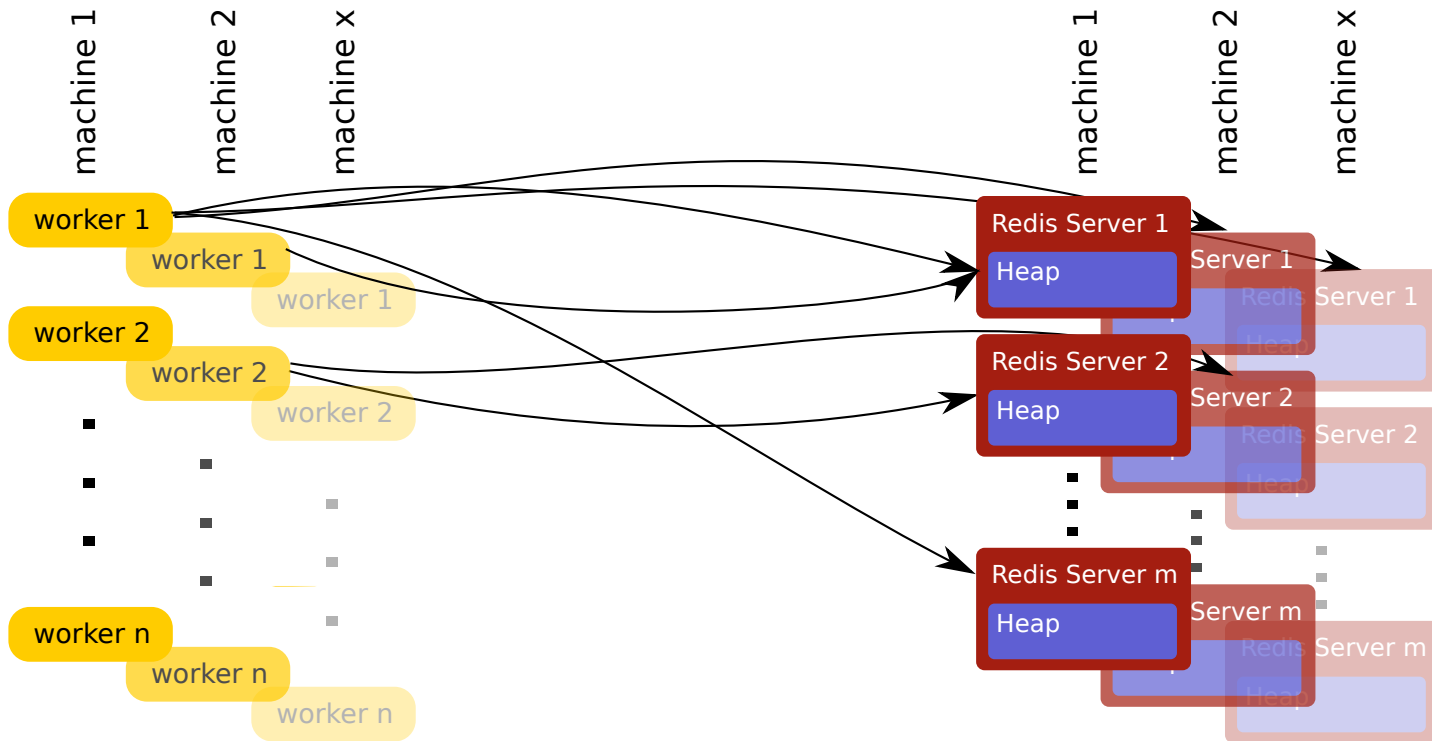
Cliqz Backend Usecase

replacement of **Redis**
single -> multi core
reduced size 5TB -> 2TB
serialized -> direct access

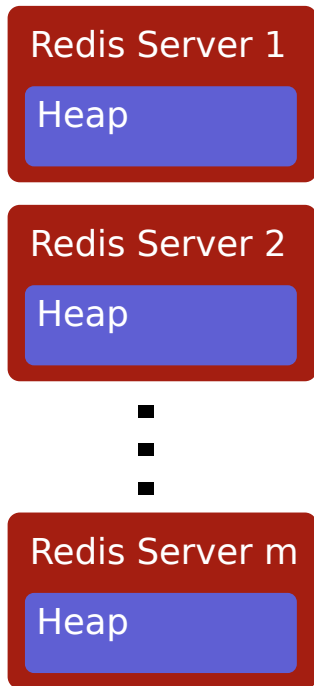
Redis Servers: 3 * 22

keyvi Servers: 3 * 10, later 3

Scaling with Redis



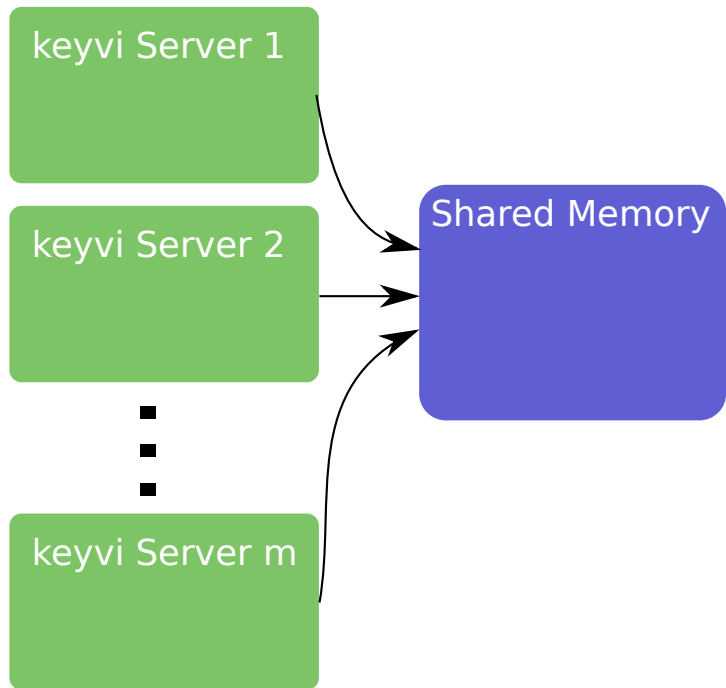
Scaling with Redis 1 Machine



every Redis process has its own heap
data belongs to the process

if Redis dies, reload takes a significant
amount of time

Scaling with keyvi



shared memory:
several processes/threads can read

resilient

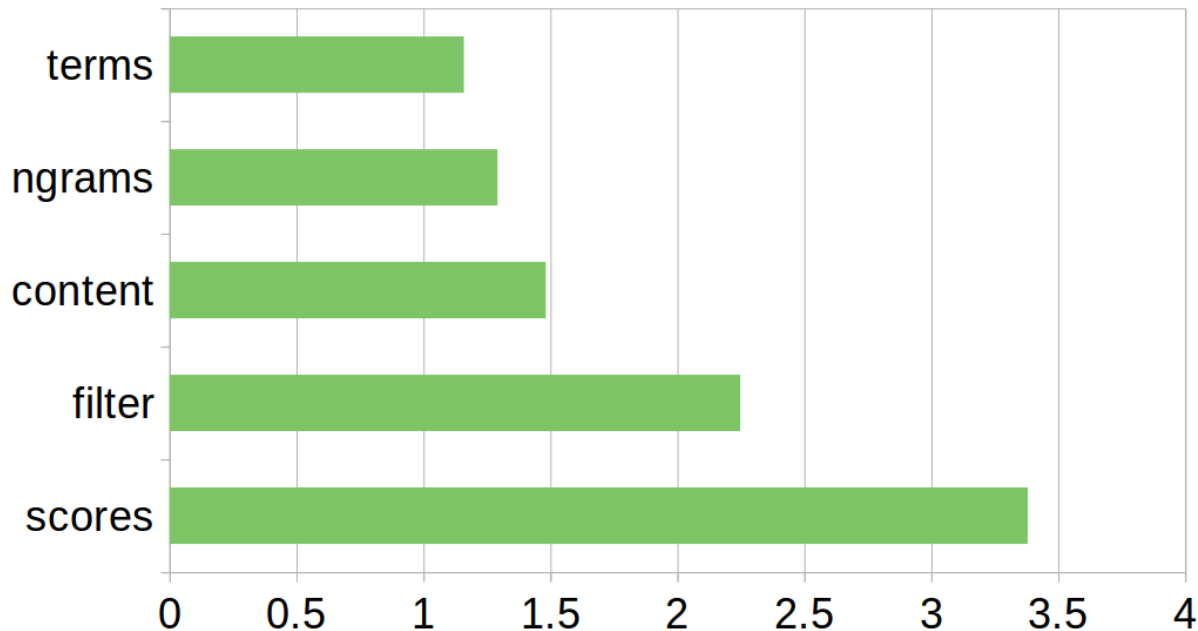
persistence = in-memory

Size Comparison

k/v pairs in million	Redis	keyvi
1	456	385
10	4538	3742
100	45303	36327

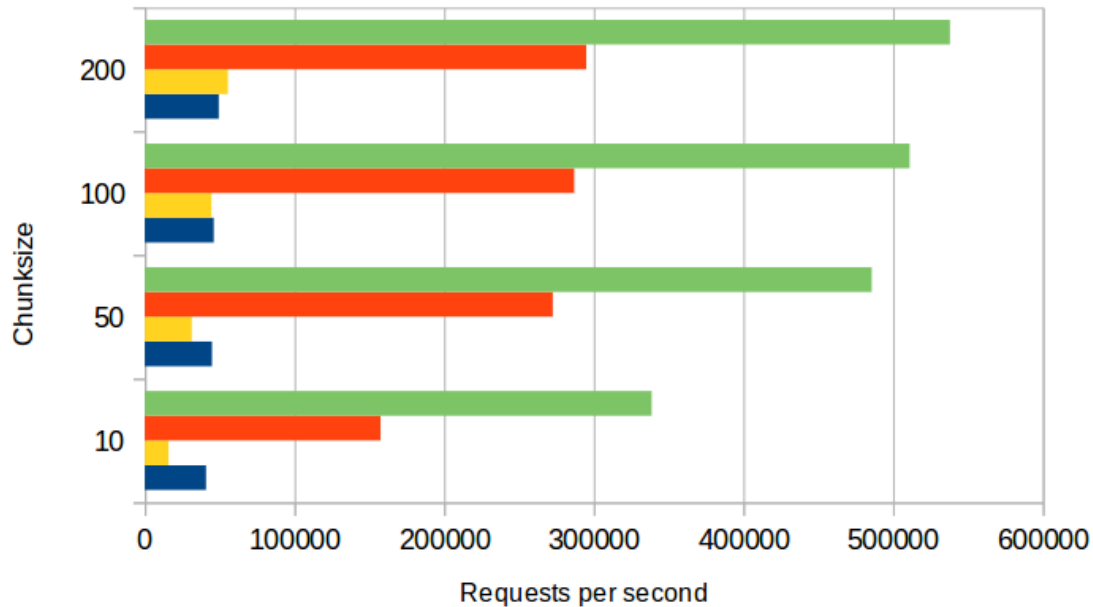
* Size in MB

Compression Ratio per Type



$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

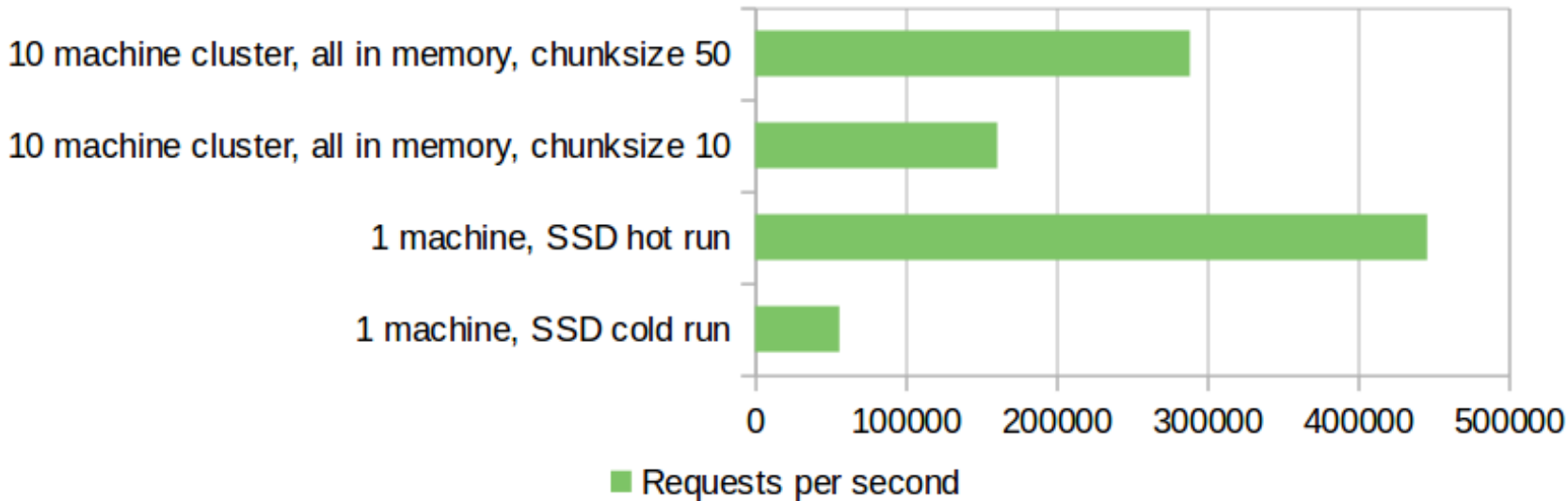
Lookup Benchmark



client/server on the same machine
chunksize == size of (redis) pipeline
host type: r3.8xlarge(AWS)

do your own benchmarks!

keyvi on SSD



Index size 2 TB
SSD tests: 1 * i2.8xlarge
cluster tests: 10 * r3.8xlarge

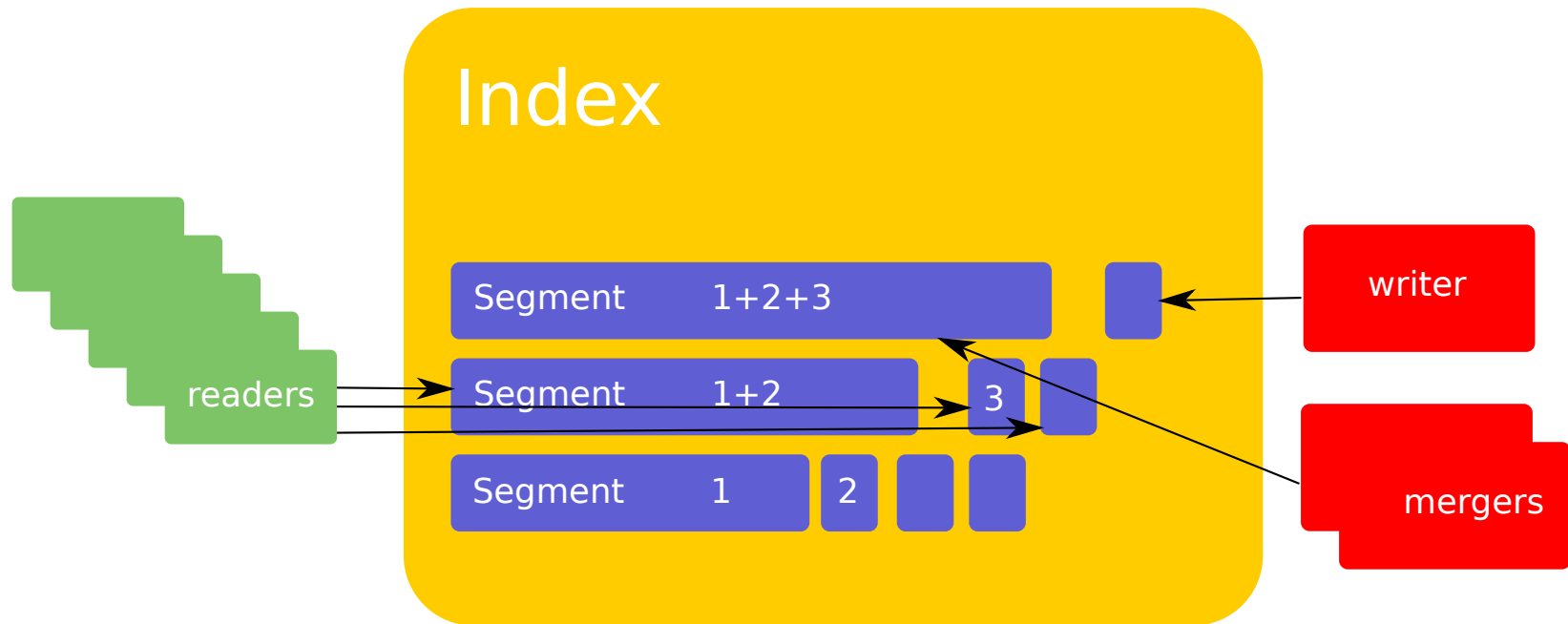
keyvi flavours

in-memory but also disk based

a server but also embedded

A keyvi key value store

Released soon



keyvi Advantages

extremely good locality

no heap fragmentation

compact due to de-duplication built-in



Hands On keyvi

Get started

```
hendrik : bash — Konsole
File Edit View Bookmarks Settings Help
hendrik@hendrik-tp:~$ pip install pykeyvi
Collecting pykeyvi
  Downloading pykeyvi-0.2.2-cp27-cp27mu-manylinux1_x86_64.whl (4.3MB)
    100% |████████████████████████████████████████| 4.4MB 312kB/s
Collecting msgpack-python (from pykeyvi)
Installing collected packages: msgpack-python, pykeyvi
Successfully installed msgpack-python-0.4.8 pykeyvi-0.2.2
You are using pip version 8.1.1, however version 9.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
hendrik@hendrik-tp:~$
```

Enriching Data with keyvi and pySpark

Spark keyvi usecases

enrich (more than once)

free text extraction

filtering

Spark setup

```
In [ ]: from pyspark import SparkFiles
        from pyspark import SparkContext

        sc.addFile("s3n://my_bucket/my_file.kv")
```

```
In [ ]: import pykeyvi

        def my_mapper(key_value):
            key, value = key_value

            # todo: you do not want to do this on every call, to be replaced with some loader:
            # see https://github.com/cliqz-oss/keyvi/blob/master/doc/usage/Using%20pykeyvi%20in%20EMR.md
            d = pykeyvi.Dictionary(SparkFiles.get("my_file.kv"))

            # simple filter
            matched = value['feature'] in d

            # simple exact match
            match = d[value['feature']]

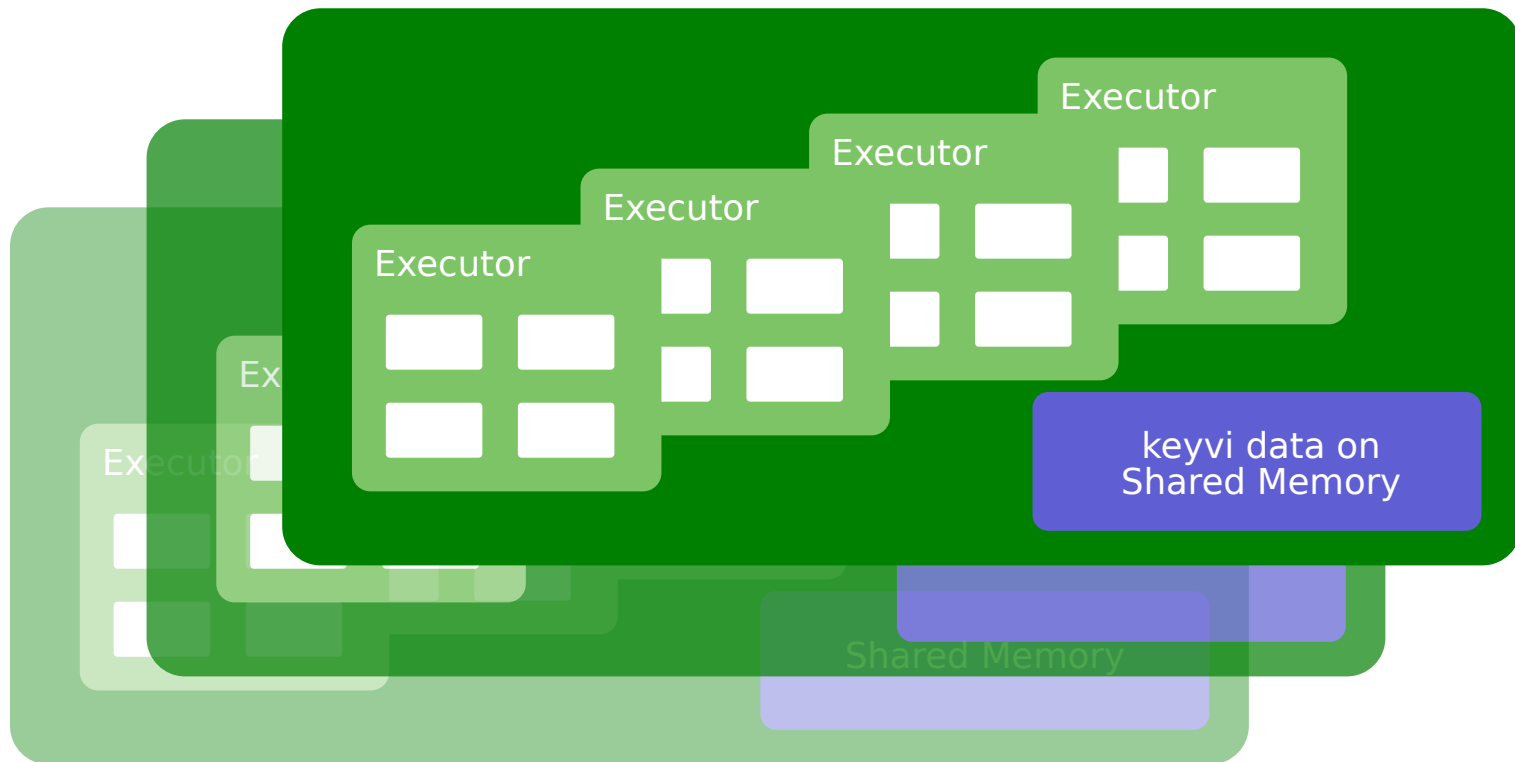
            # Free text (space-tokenized) lookup (leftmost longest, multi token)
            match = d.LookupText[value['feature']]

            # what did we found?
            matched_string = match.GetMatchedString()

            # where did it match?
            start = match.GetStart()
            end = match.GetEnd()

            # get metadata attached to it
            matched_value = match.GetValue()
```

Spark keyvi explained



Almost there!

TakeAways

it's different

it's simple

it's versatile

TakeAways

Try it out!

it's different

it's simple

it's versatile



More infos/material/code at <http://keyvi.org>

Q & A

hendrik.muhs@gmail.com