keyvi

the key value index

optimized for size and speed

key value index

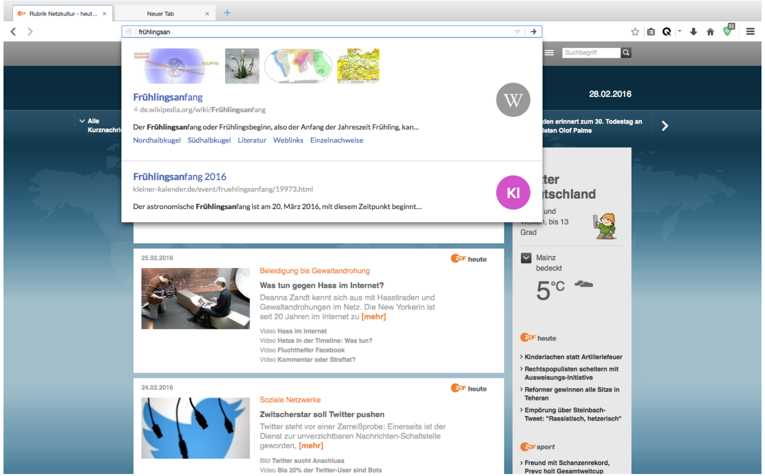
based on finite state
(FST, immutable data structure)

Opensource (Apache 2.0)

written in C++(core), Python(binding)

keyvi @CLIQZ

Search in the Browser



Available on the App Store

ANDROID APP ON Google play

keyvi in numbers @CLIQZ

1.8bn URLs

13bn data points (key value pairs)

2.5TB index size

60ms average latency

6k requests per minute

99.9% availability

keyvi History @CLIQZ

(→ 10/2014)
to slow/heavy
no need for full-text search
did not fit our model

keyvi History @CLIQZ

elastic

redis

(→ 10/2014)
to slow/heavy
no need for full-text search
did not fit our model

(4/2014 →)
simple
space efficient
server side scripting

keyvi History @CLIQZ

redis

keyvi

(→ 09/2015)
capacity problems
single threaded
heap based
single-point of failure

keyvi History @CLIQZ

redis

(→ 09/2015)
capacity problems
single threaded
heap based
single-point of failure

keyvi

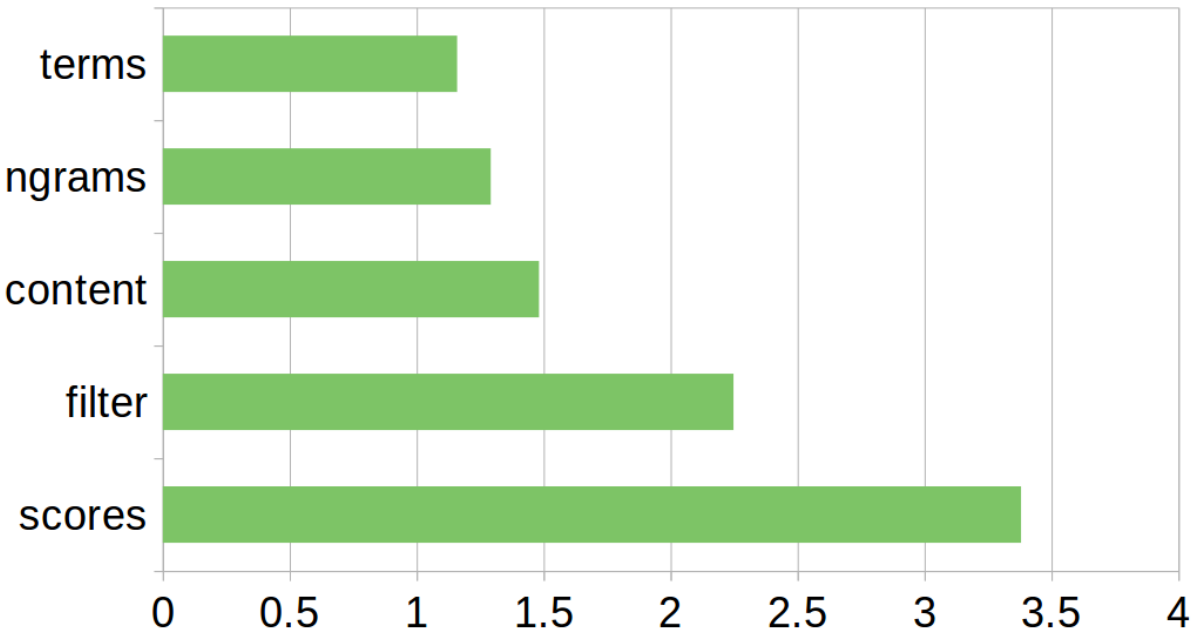
(05/2015 →)
massive size reduction
multi threaded/process
shared memory
more reliable

Size Comparison

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

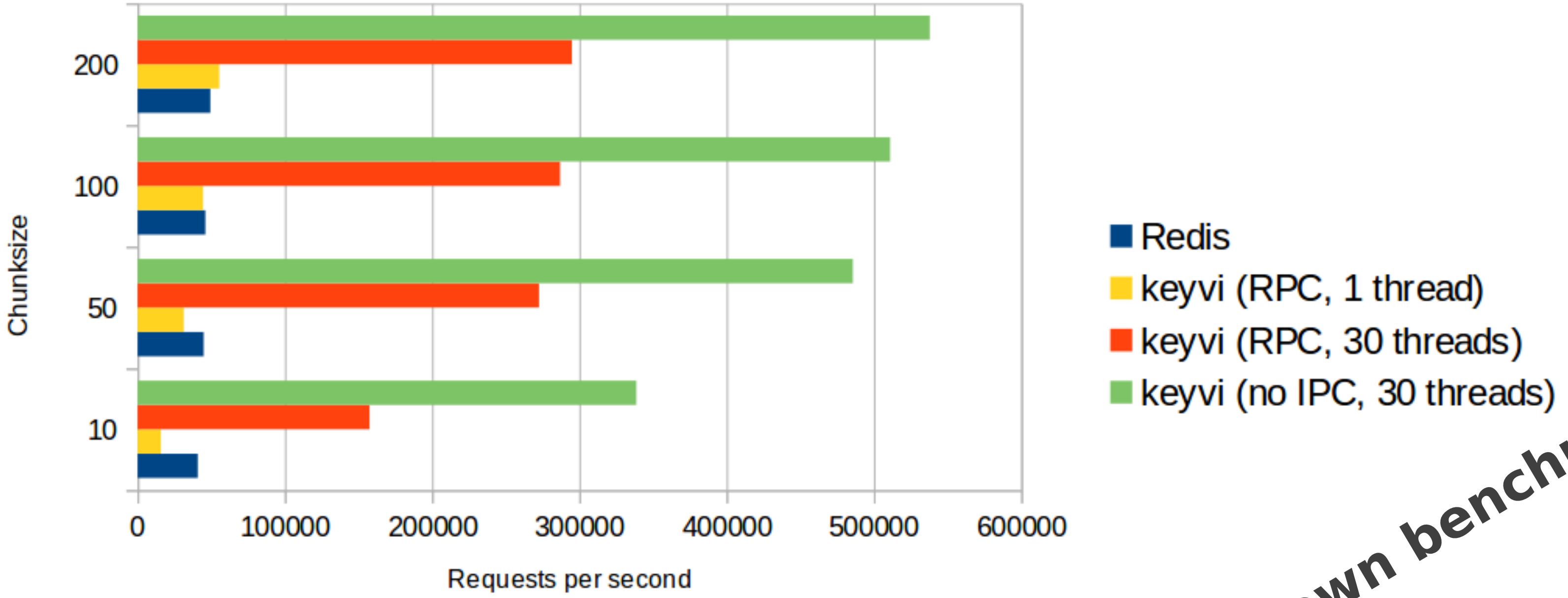
* Size in MB

Compression Ratio per Type



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

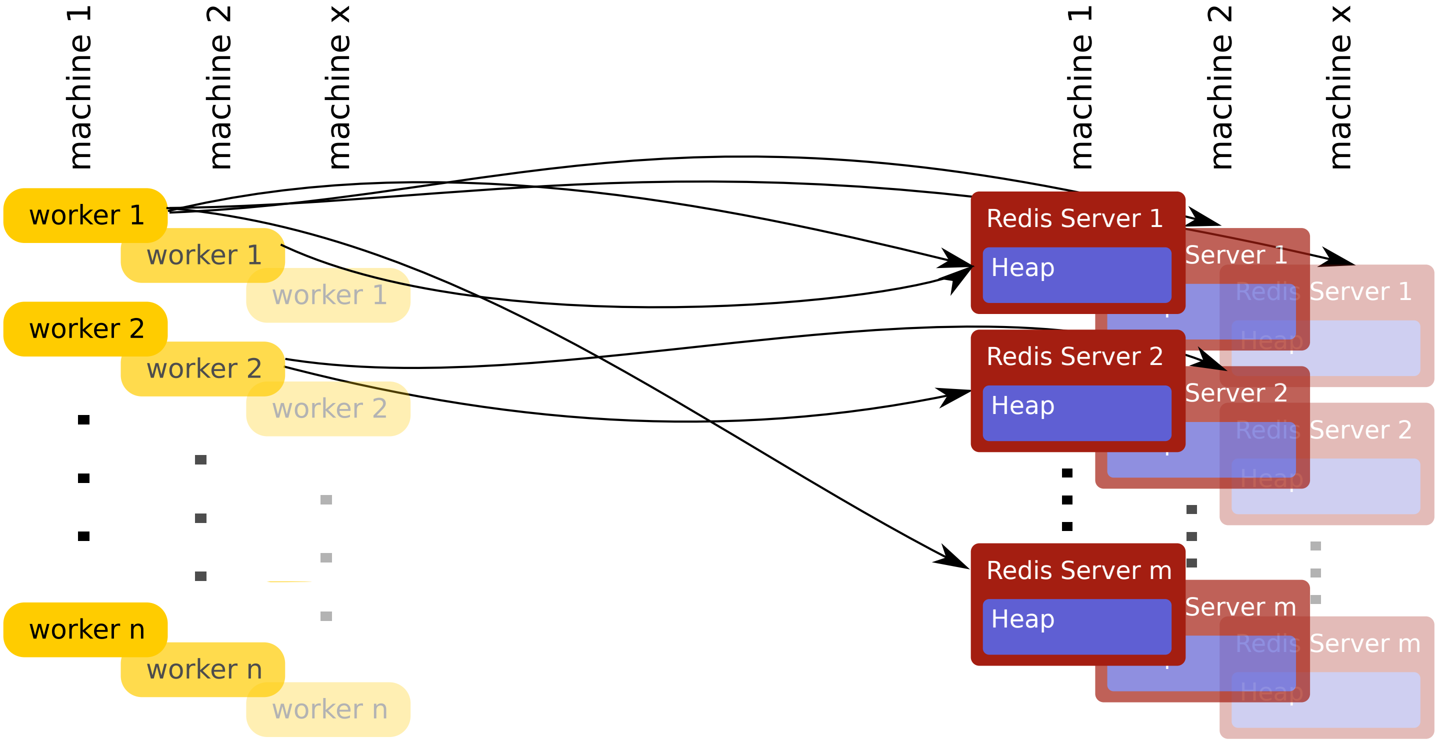
Lookup Benchmark



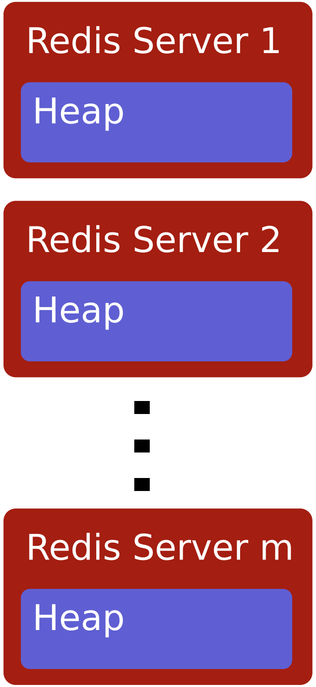
do your own benchmarks!

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

Scaling with Redis



Scaling with Redis 1 Machine

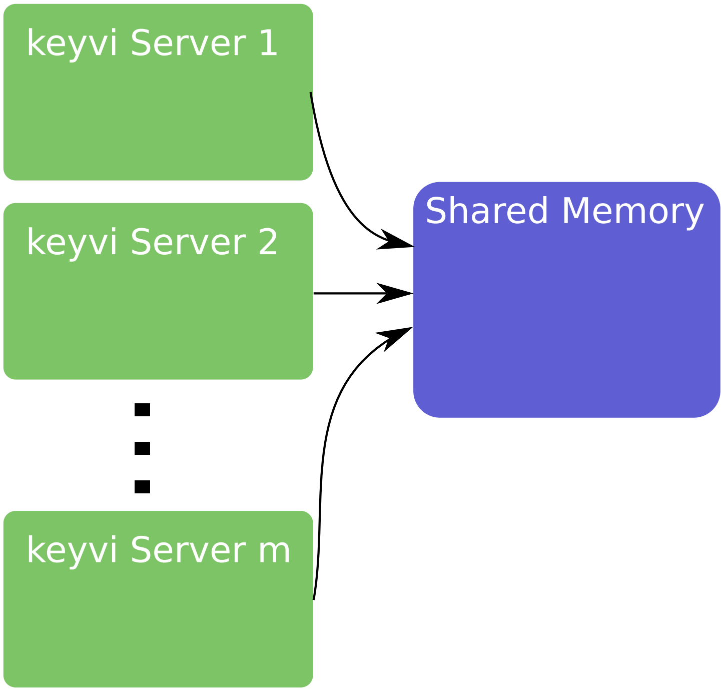


every Redis process has its own heap

data access is local

if Redis dies, data has to be reloaded
(can take a significant amount of time)

Scaling with keyvi



keyvi is shared memory

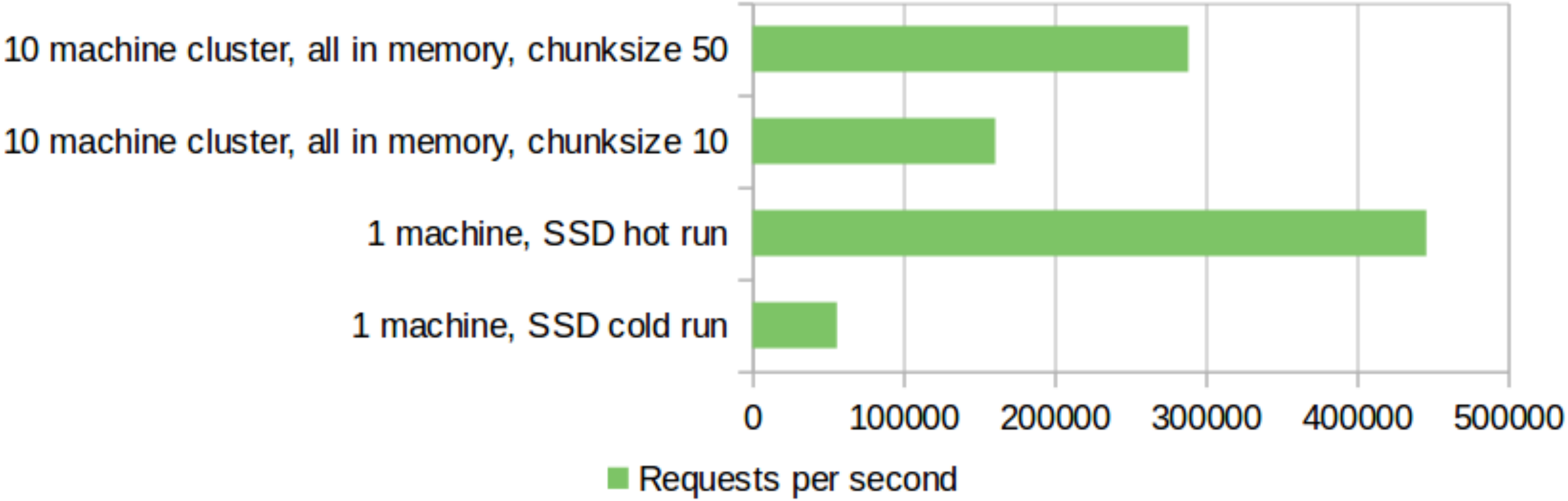
several processes/threads can read

a crash effects 1 process

no deserialization/loading

no need for IPC/networking if local

keyvi on SSD

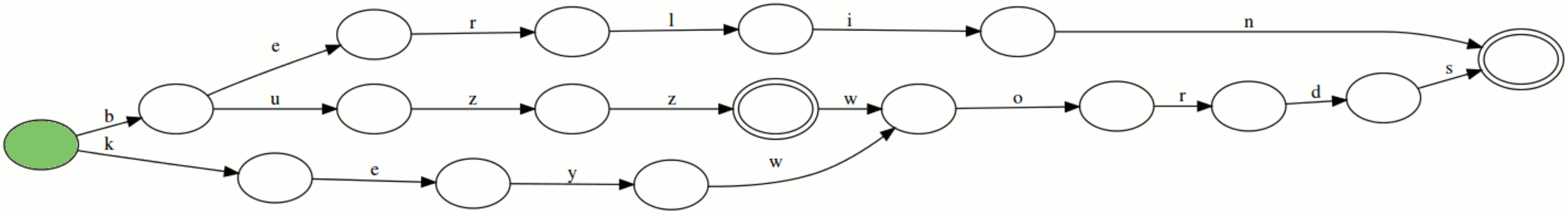


Index size 2 TB
SSD tests: 1 * i2.8xlarge
cluster tests: 10 * r3.8xlarge

Finite State in keyvi

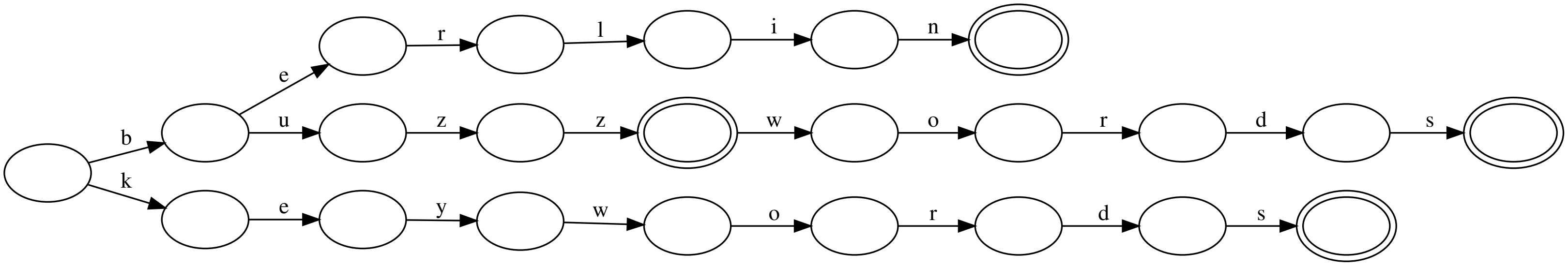
Under the Hood: FST

An FST consisting of: berlin, buzzwords, buzz, keywords



Under the Hood: a Trie for Comparison

An FST consisting of: berlin, buzzwords, buzz, keywords



FST Incremental Construction

does not fit in memory

  -> we need all outgoing transitions before persisting

    -> (external) sort upfront

minimize on the fly

  -> hashtable with low overhead, bounded, apply LRU

stream input and output

Compact Persistence

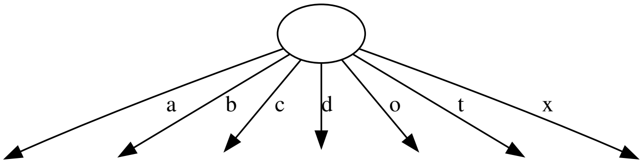
store nodes as compact as possible

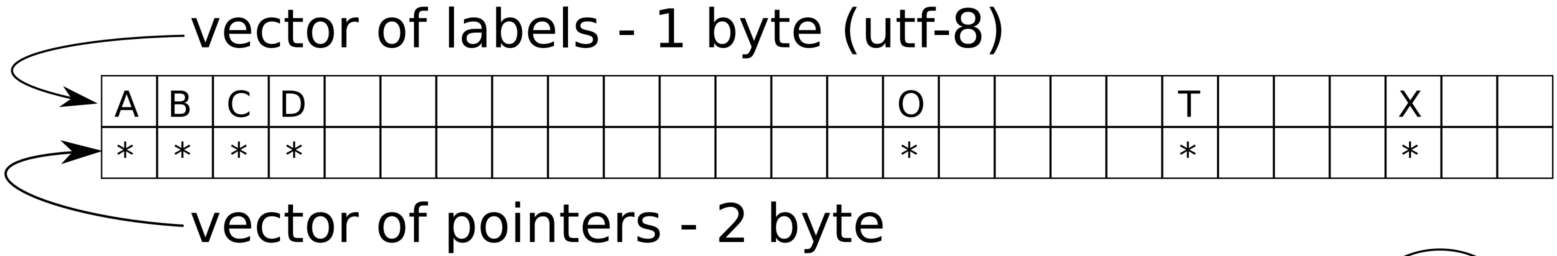
-> use clever bit magic

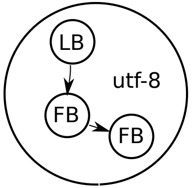
Lucene: list of small byte representations

Keyvi: sparse array packing

Sparse Array







utf-8 handling

Interleaving of States

A		C	D											O					T				X		
*		*	*											*					*				*		

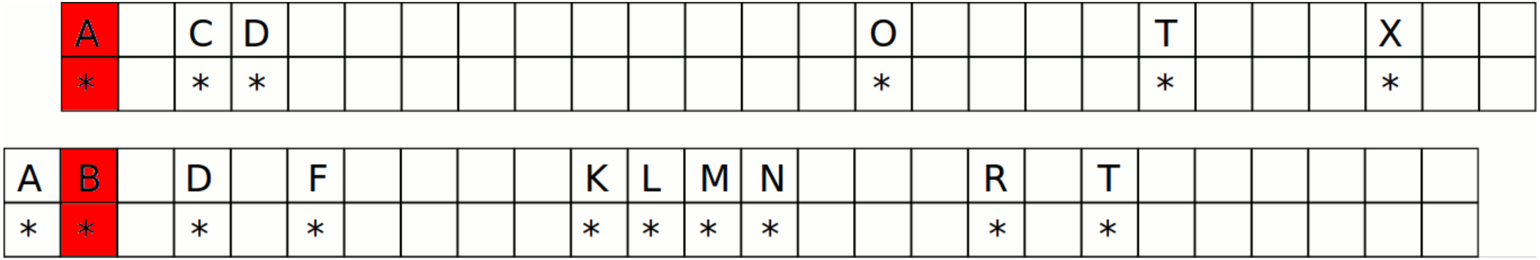
A	B		D		F					K	L	M	N				R		T						
*	*		*		*					*	*	*	*				*		*						

Interleaving of States

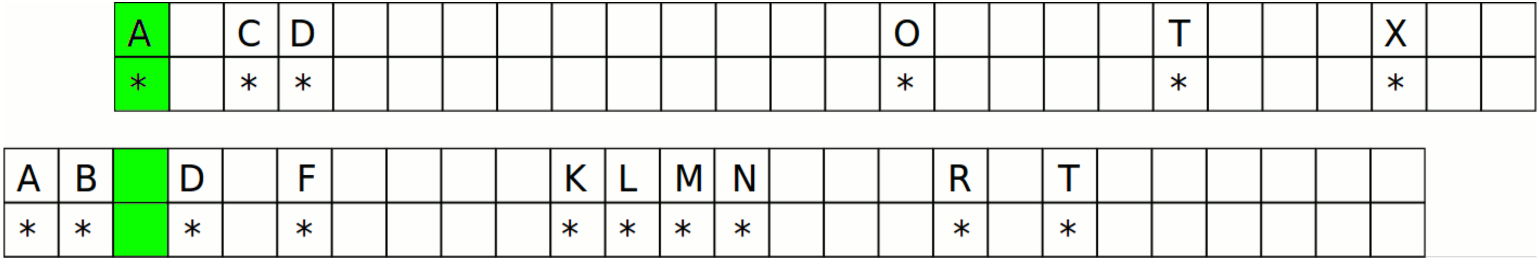
A		C	D											O					T				X		
*		*	*											*					*				*		

A	B		D		F					K	L	M	N				R		T						
*	*		*		*					*	*	*	*				*		*						

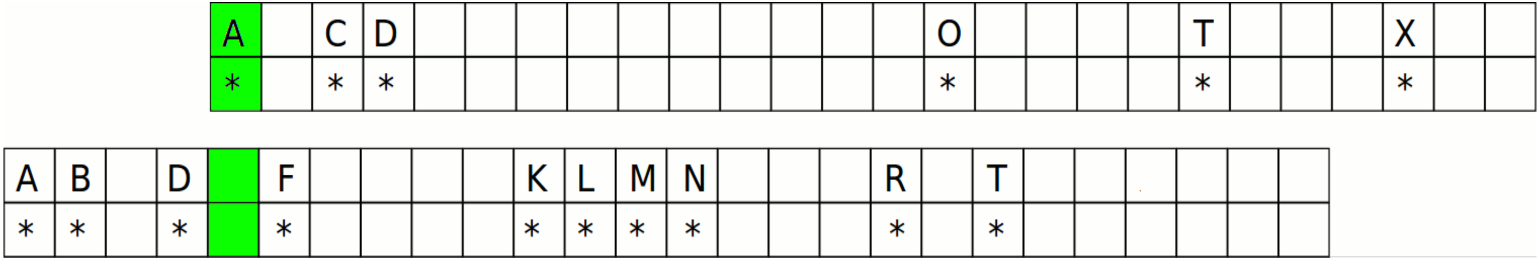
Interleaving of States



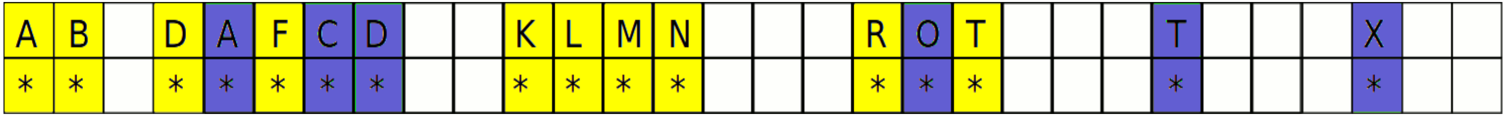
Interleaving of States



Interleaving of States



Interleaving of States



Sparse Array Tricks

fast bit vector comparison (intrinsics)

variable length encoding

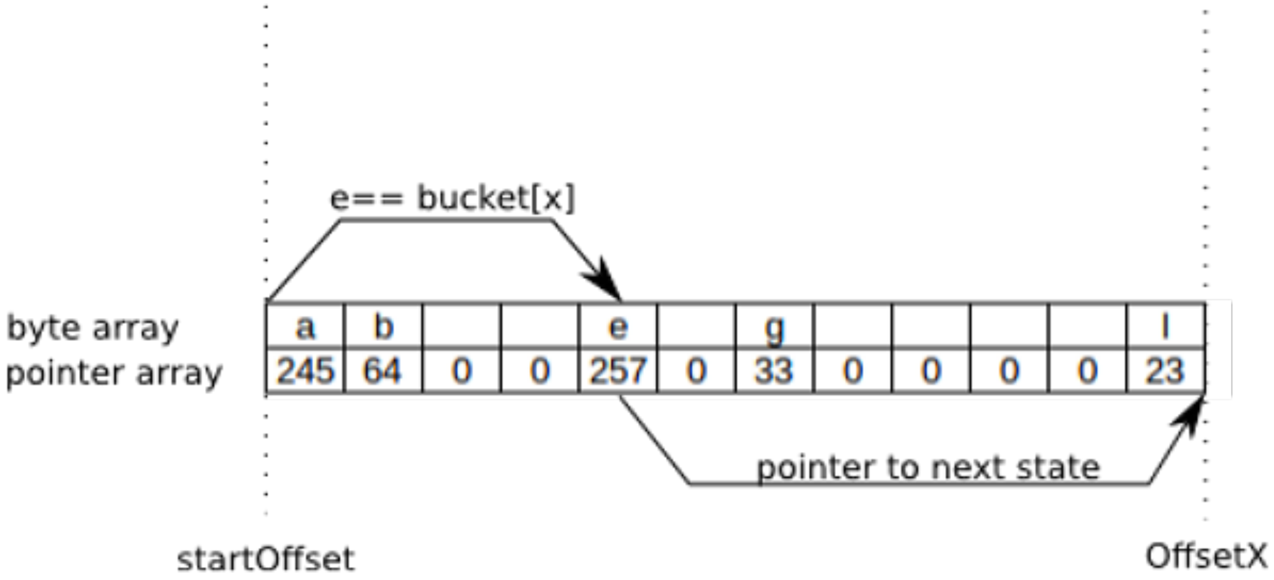
relative offsets

sliding windows

...

Keyvi Lookup

exact match == n * access in both arrays



Storing Values

offset in a separate buffer stored as part of the finite state

supported values: key-only, ints, strings, json

- special subtype: int with inner weights

json store: msgpack, compression (snappy, zlib)

Almost all "real time" people end up
actually being perfectly ok with "fast enough",
and very very few people are really _hard_ real time.

(Linus Torvalds, 1998)

Updates and Realtime

Simple Updates @CLIQZ

Delta Update t-2

Delta Update t-1

Delta Update t (as needed: hourly, daily, on-demand)

Main Index / Master Segment (monthly)

under development: keyvi merger

More Usecases

exact matching
- entity recognition, e.g. in pySpark

approximate matching:
- close/near match e.g. for Geo applications
- scoring based: Levenshtein & Co

completion matching:
- prefix, multi-word, approximate

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

Q & A

hendrik.muhs@gmail.com

Benchmark FST