Implementing a key-value store

for fun (and profit?)

André Silva @ ShiftForward

19th January 2017

Implementing a 

for fun (and profit?)

André Silva @ ShiftForward

19th January 2017

What's a key-value store?

What's a key-value store?

- `put(key, value)`

What's a key-value store?

- `put(key, value)`
- `get(key) -> value`

What's a key-value store?

- `put(key, value)`
- `get(key) -> value`
- `delete(key)`

What's a key-value store?

```
Map[String, String]
```

What's a key-value store?

`Map[String, String]`

- In-memory
- No persistence
- Cannot be shared

Why are key-value stores useful?

- Simple
- Fast
- You can build upon them
  - SQL
  - Graphs

Why would I want to implement one?

Why would I want to implement one?

Why would I want to implement one?

⚠️ Please don't roll your own database! ⚠️

Let's do this!

Goals

Goals

- Low latency

Goals

- Low latency
- High throughput

Goals

- Low latency
- High throughput
- Handle datasets larger than memory

Goals

- Low latency
- High throughput
- Handle datasets larger than memory
- Crash friendly

Goals

- Low latency
- High throughput
- Handle datasets larger than memory
- Crash friendly
- Simple design

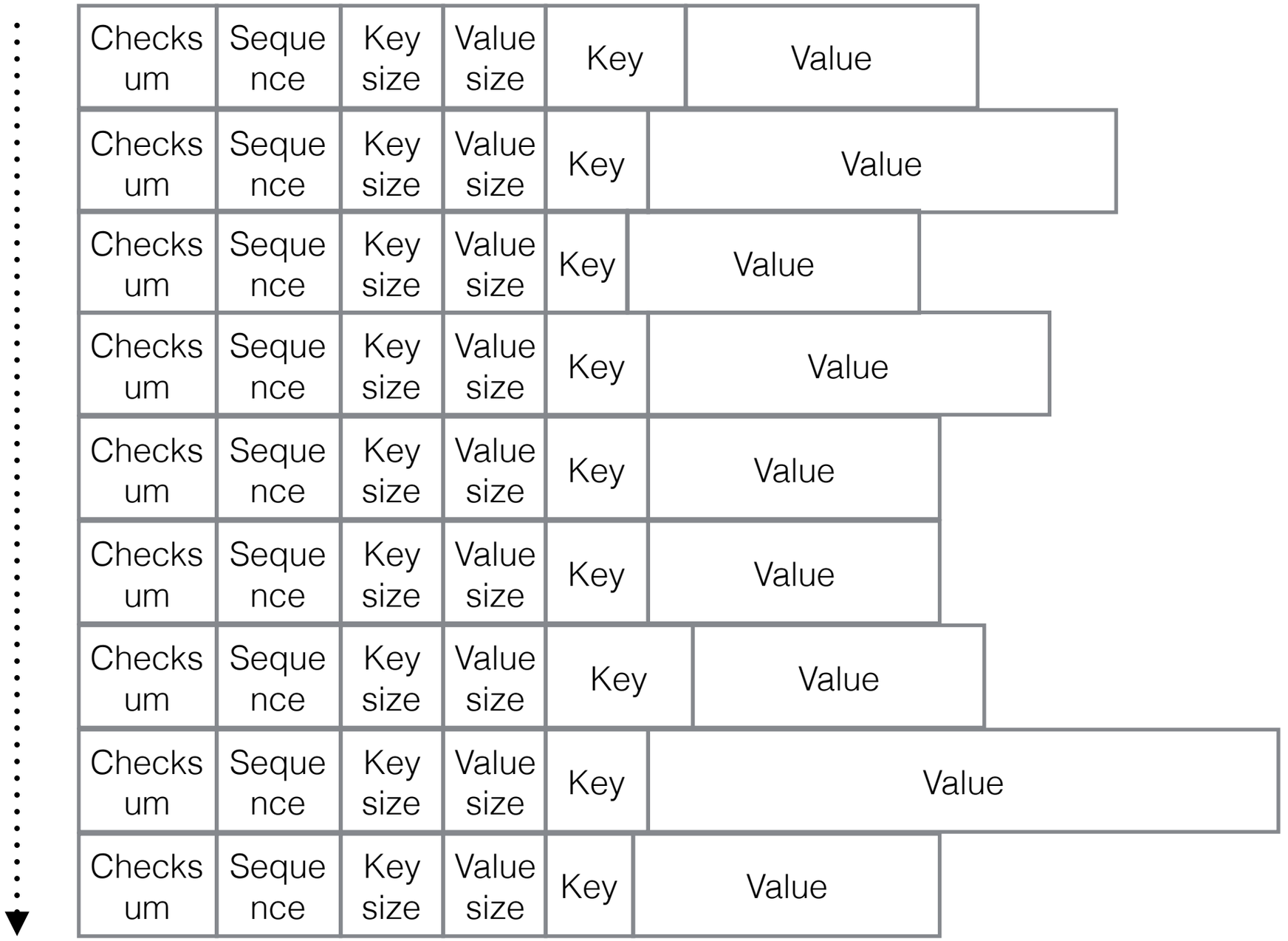
Basic design

- Log-structured hash table
- Two main components:
  - An in-memory **index** (`HashMap`)
  - A **log** file(s)

Log entry

Checksum	Sequence	Key size	Value size	Key	Value

Log



Database

- A directory with multiple files
- At any moment only one file is "active"
- When a size threshold is reached we create a new "active" file
- Append-only

Database

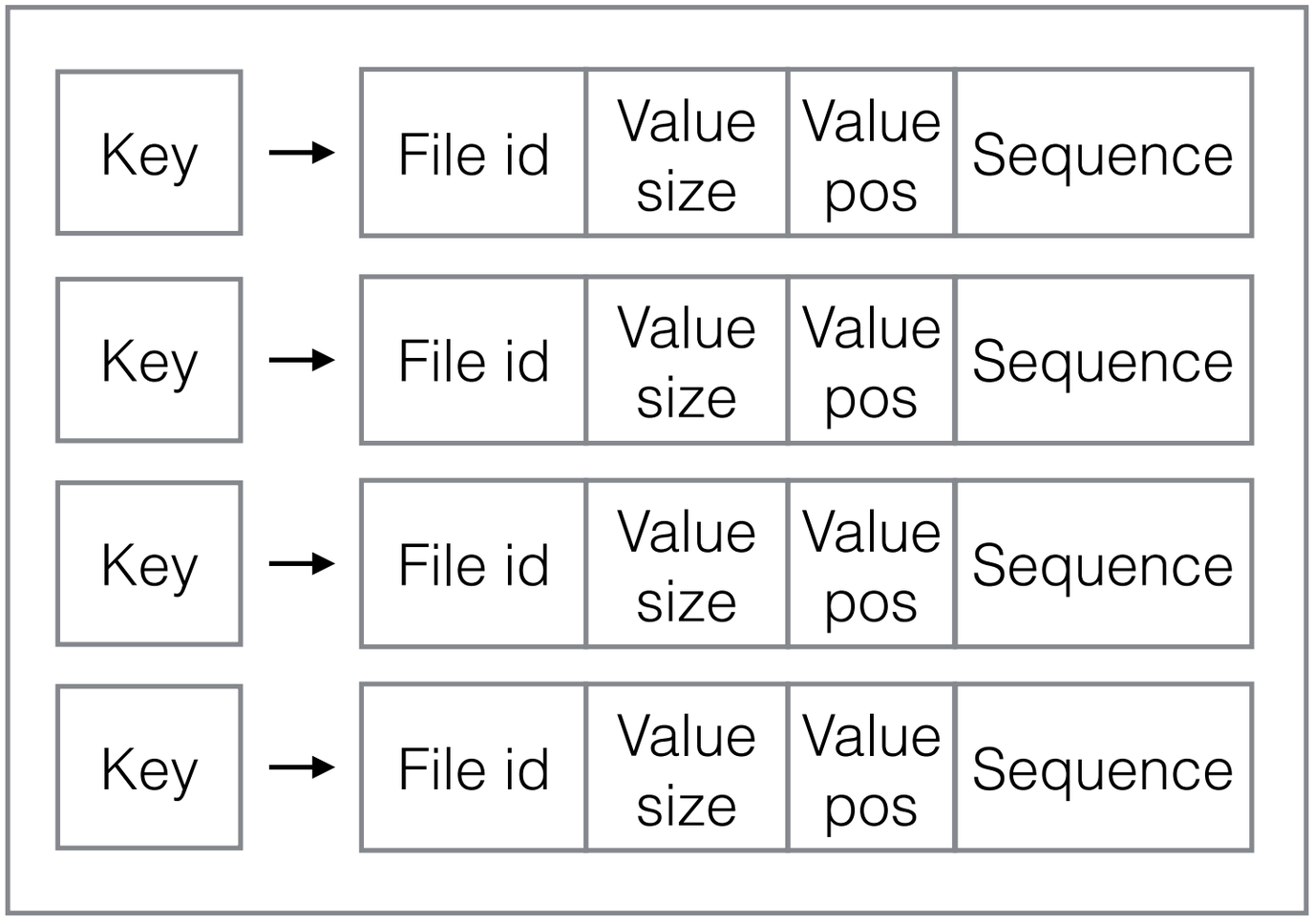
```
$ ls -1 test.db
0000000000.cask.data
0000000000.cask.hint
0000000001.cask.data
0000000001.cask.hint
0000000002.cask.data
0000000002.cask.hint
0000000003.cask.data
cask.lock
```

Database

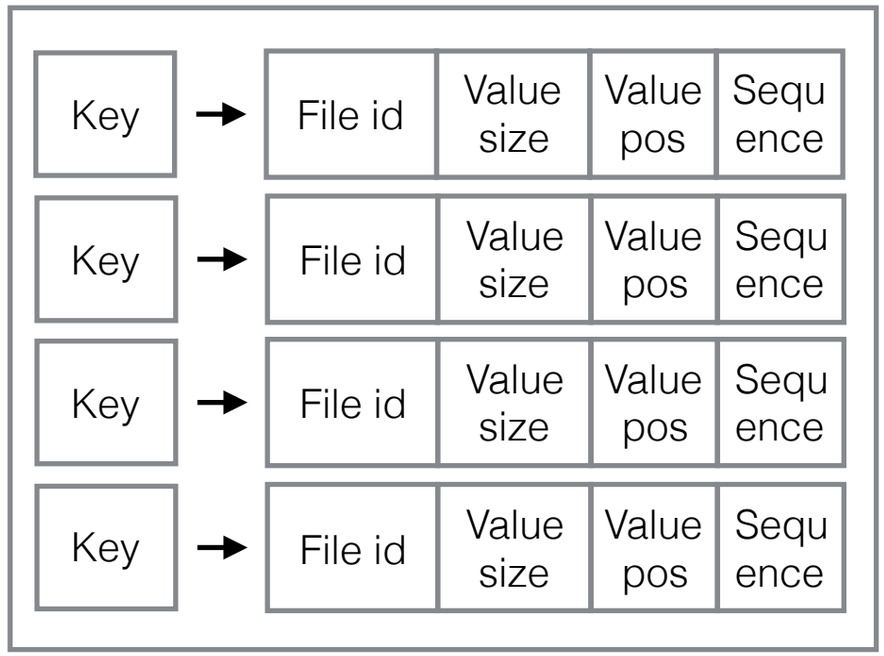
```
$ ls -1 test.db
0000000000.cask.data
0000000000.cask.hint
0000000001.cask.data
0000000001.cask.hint
0000000002.cask.data
0000000002.cask.hint
0000000003.cask.data
cask.lock
```

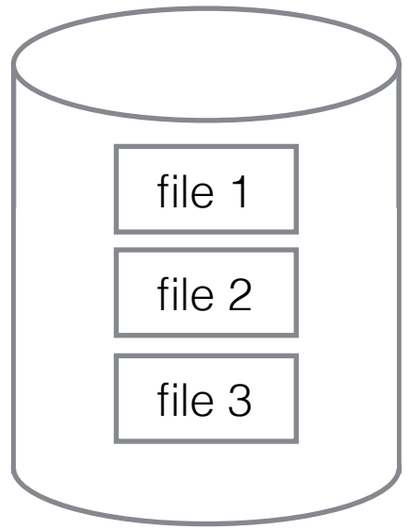
Active file

Index


Key
File id
Value size
Value pos
Sequence
Key
File id
Value size
Value pos
Sequence
Key
File id
Value size
Value pos
Sequence
Key
File id
Value size
Value pos
Sequence

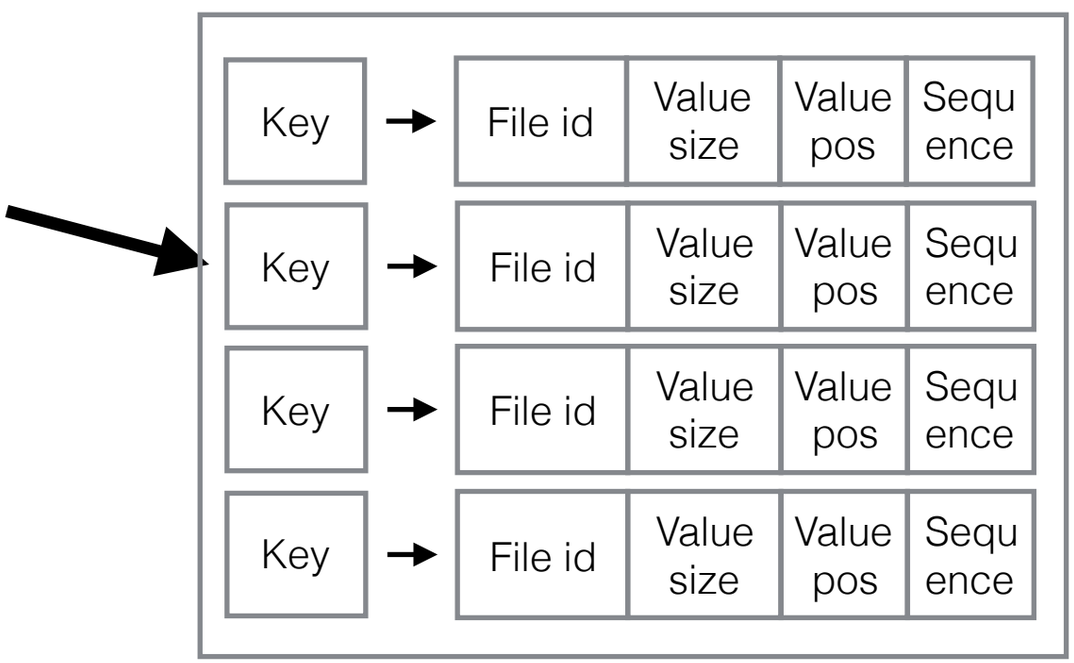
get(key)

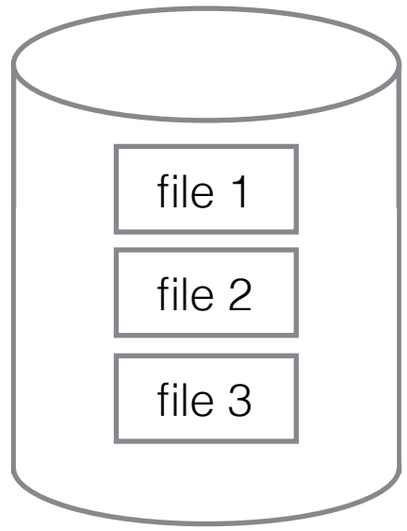




Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value

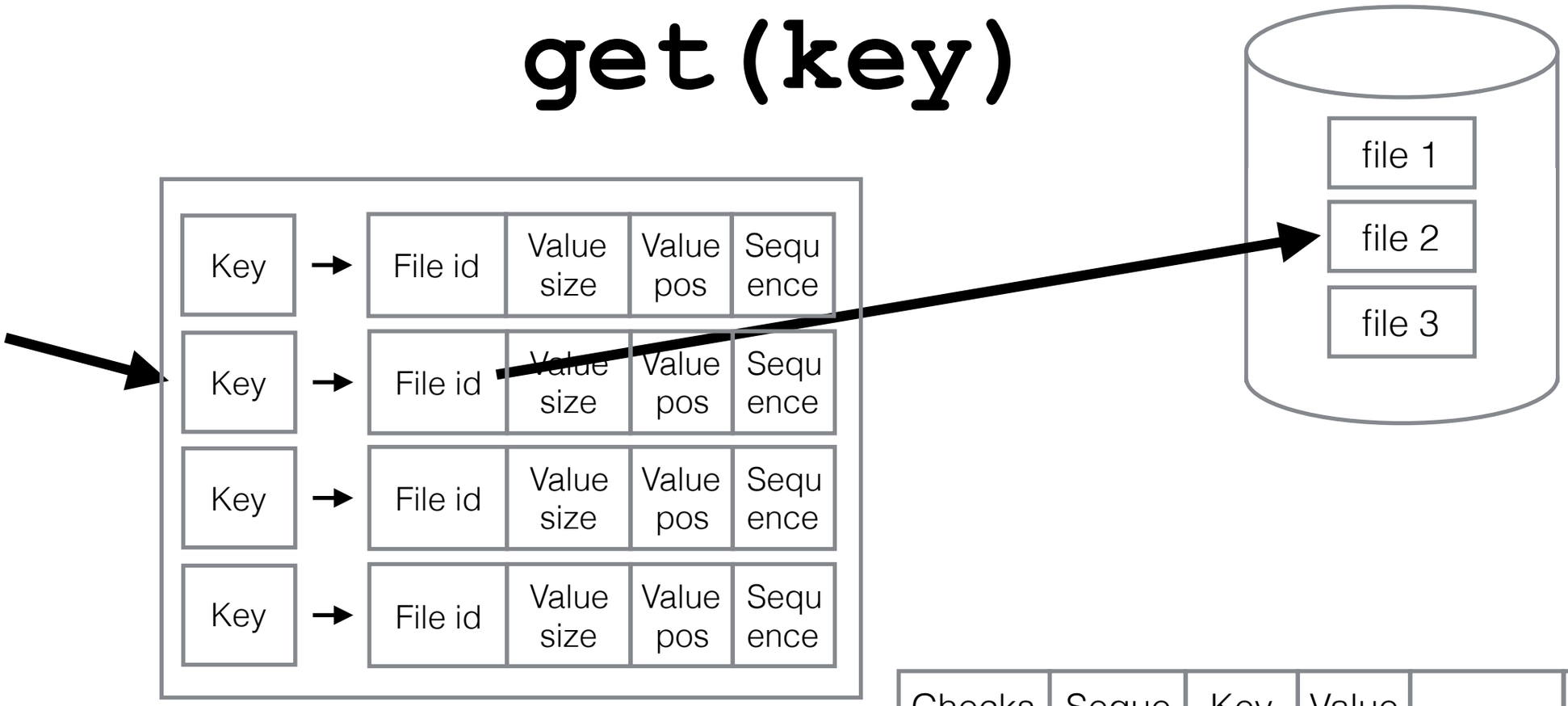
get(key)



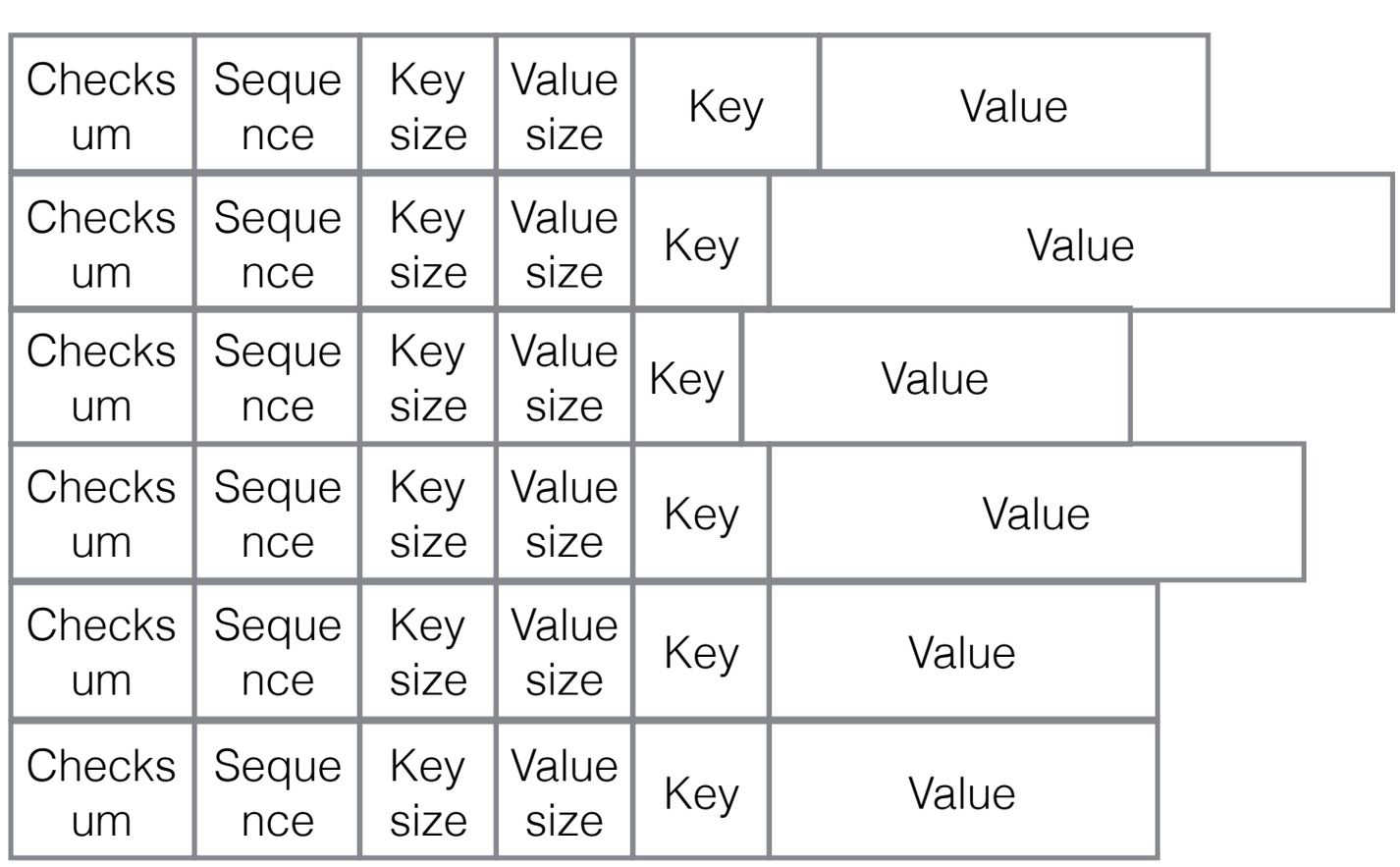


Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value

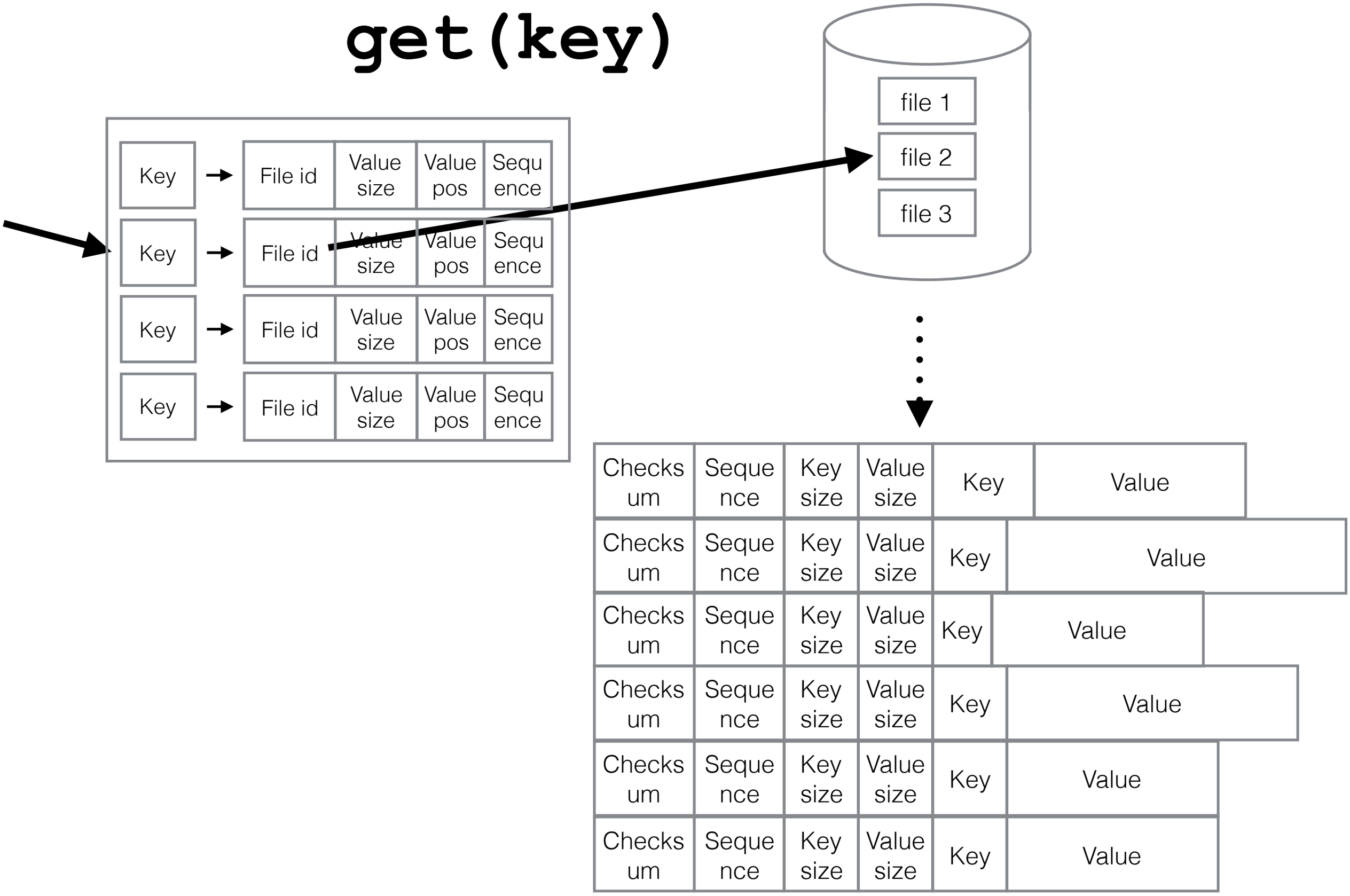
get(key)


Key
File id
Value size
Value pos
Sequ ence
Key
File id
Value size
Value pos
Sequ ence
Key
File id
Value size
Value pos
Sequ ence
Key
File id
Value size
Value pos
Sequ ence

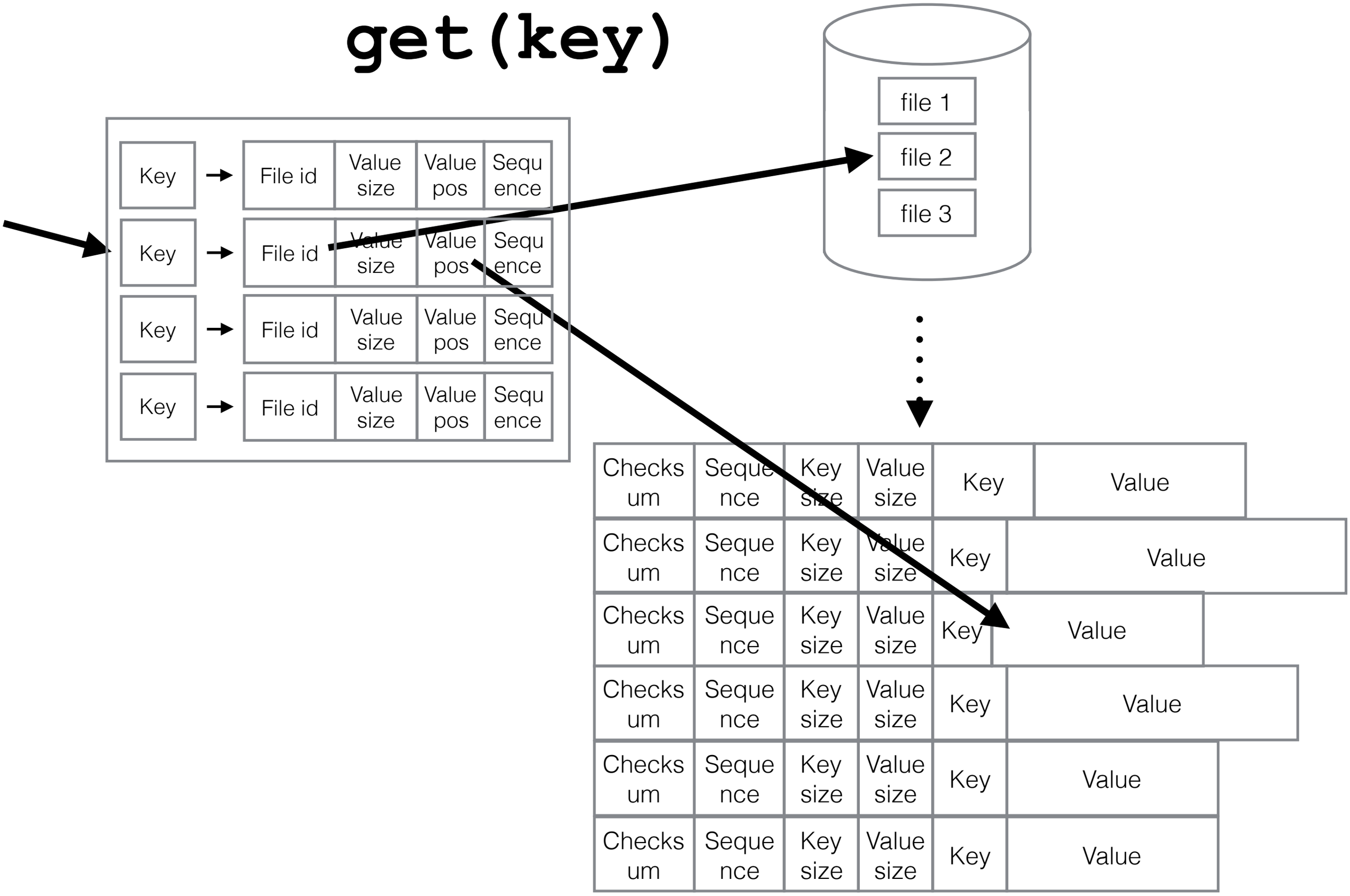
file 1
file 2
file 3


Checksum
Sequence
Key size
Value size
Key
Value
Checksum
Sequence
Key size
Value size
Key
Value
Checksum
Sequence
Key size
Value size
Key
Value
Checksum
Sequence
Key size
Value size
Key
Value
Checksum
Sequence
Key size
Value size
Key
Value
Checksum
Sequence
Key size
Value size
Key
Value

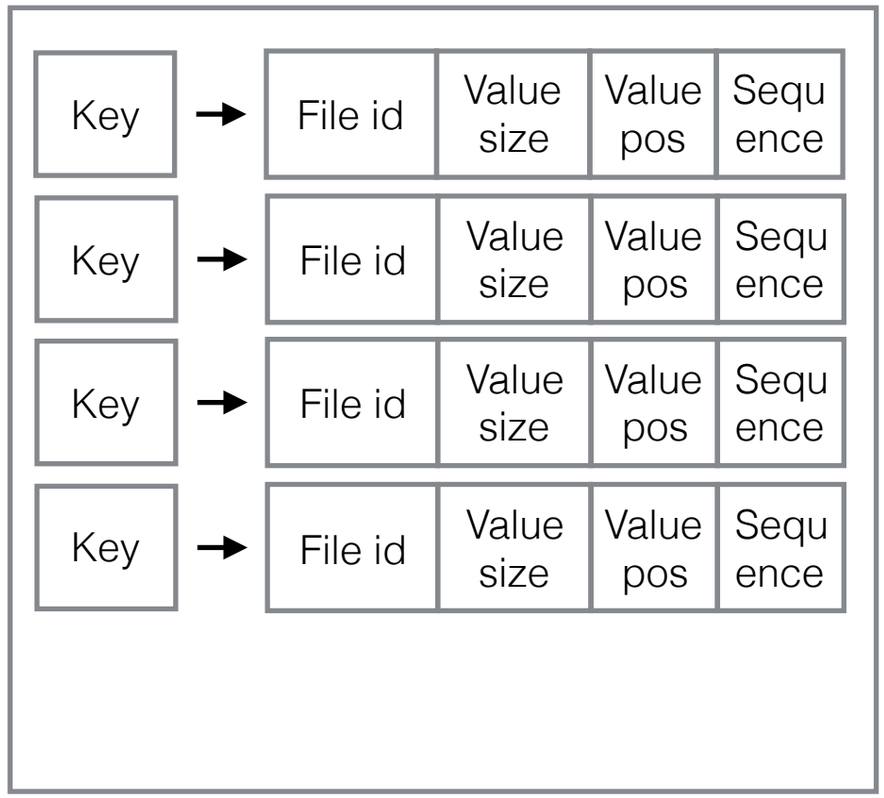
get(key)

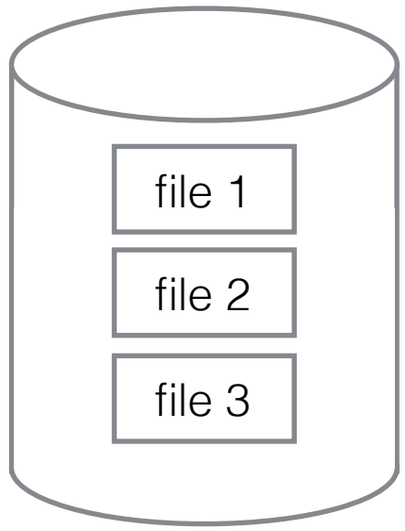

Key
File id
Value size
Value pos
Sequence
Key
File id
Value size
Value pos
Sequence
Key
File id
Value size
Value pos
Sequence
Key
File id
Value size
Value pos
Sequence
file 1
file 2
file 3
Checksum
Sequence
Key size
Value size
Key
Value
Checksum
Sequence
Key size
Value size
Key
Value
Checksum
Sequence
Key size
Value size
Key
Value
Checksum
Sequence
Key size
Value size
Key
Value
Checksum
Sequence
Key size
Value size
Key
Value
Checksum
Sequence
Key size
Value size
Key
Value

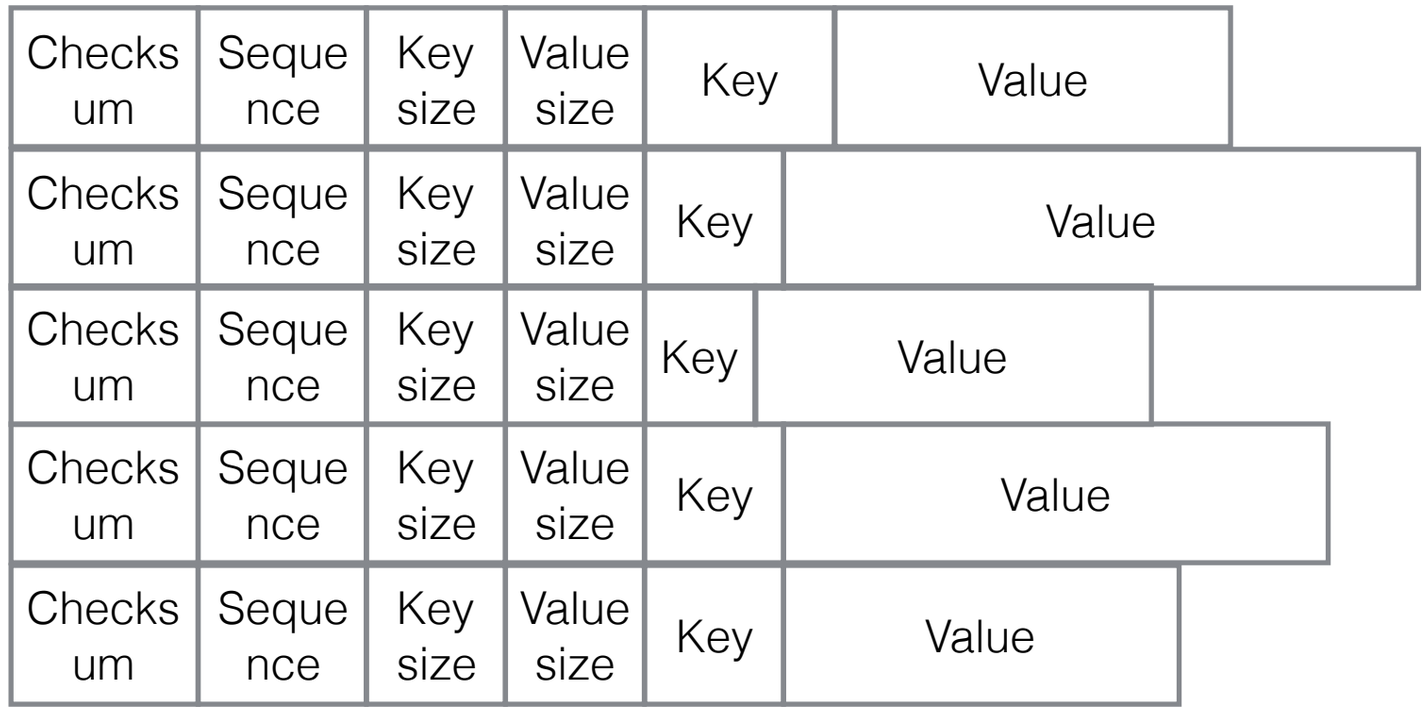
get(key)


Key
File id
Value size
Value pos
Sequ ence
Key
File id
Value size
Value pos
Sequ ence
Key
File id
Value size
Value pos
Sequ ence
Key
File id
Value size
Value pos
Sequ ence
file 1
file 2
file 3
Checks um
Seque nce
Key size
Value size
Key
Value
Checks um
Seque nce
Key size
Value size
Key
Value
Checks um
Seque nce
Key size
Value size
Key
Value
Checks um
Seque nce
Key size
Value size
Key
Value
Checks um
Seque nce
Key size
Value size
Key
Value
Checks um
Seque nce
Key size
Value size
Key
Value

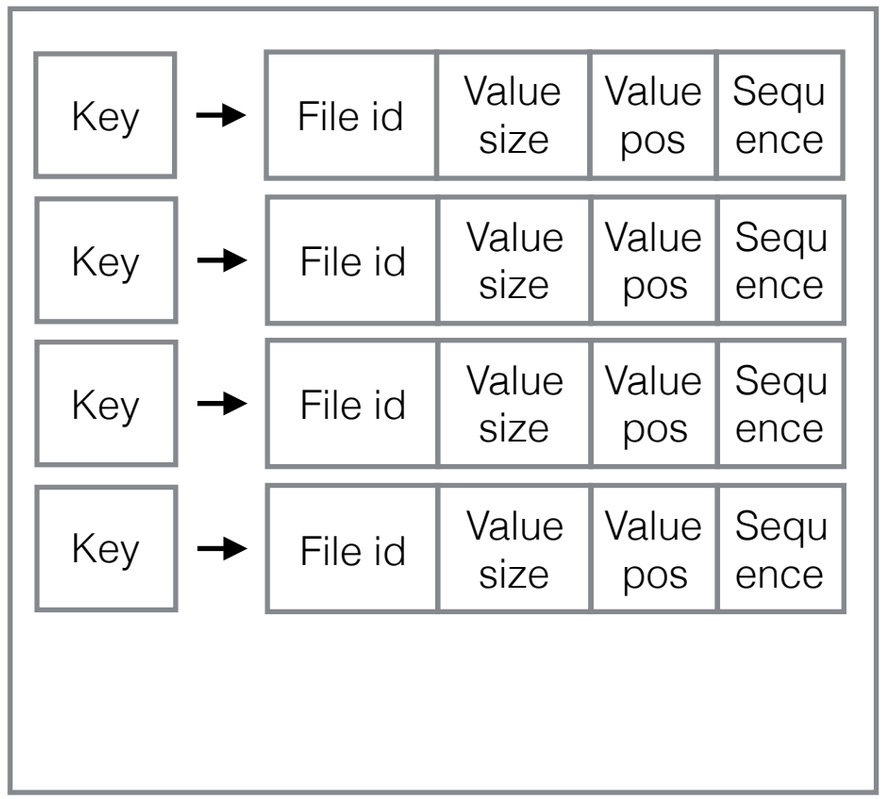
set(key, value)

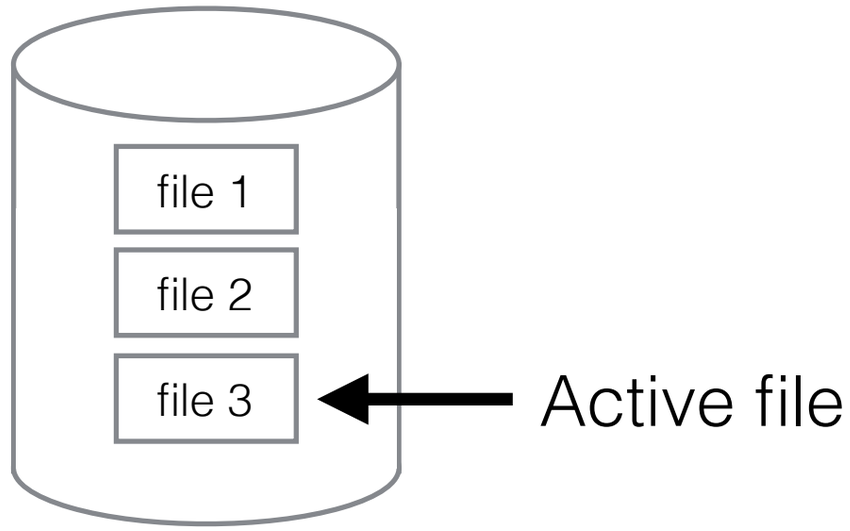






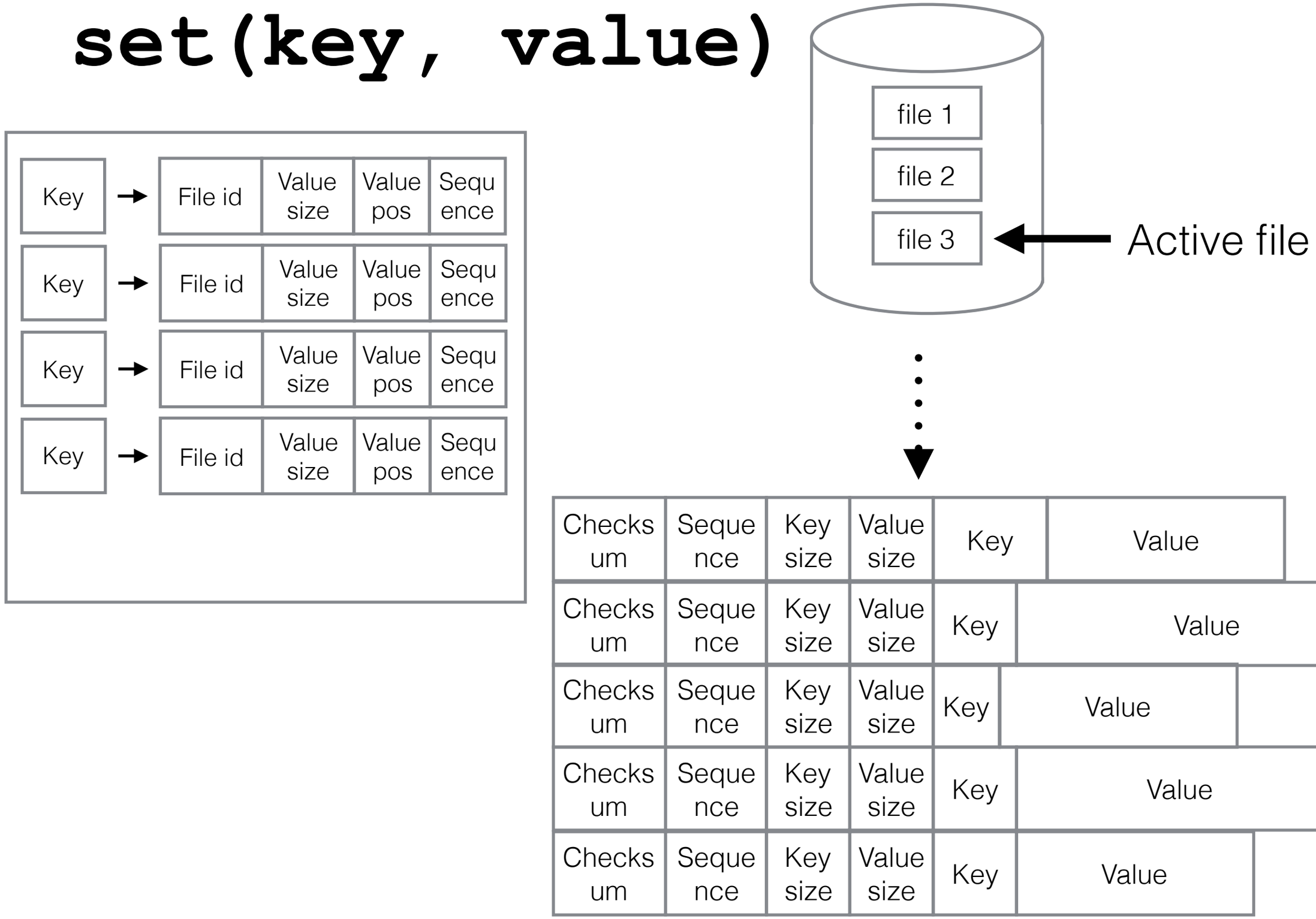
set(key, value)



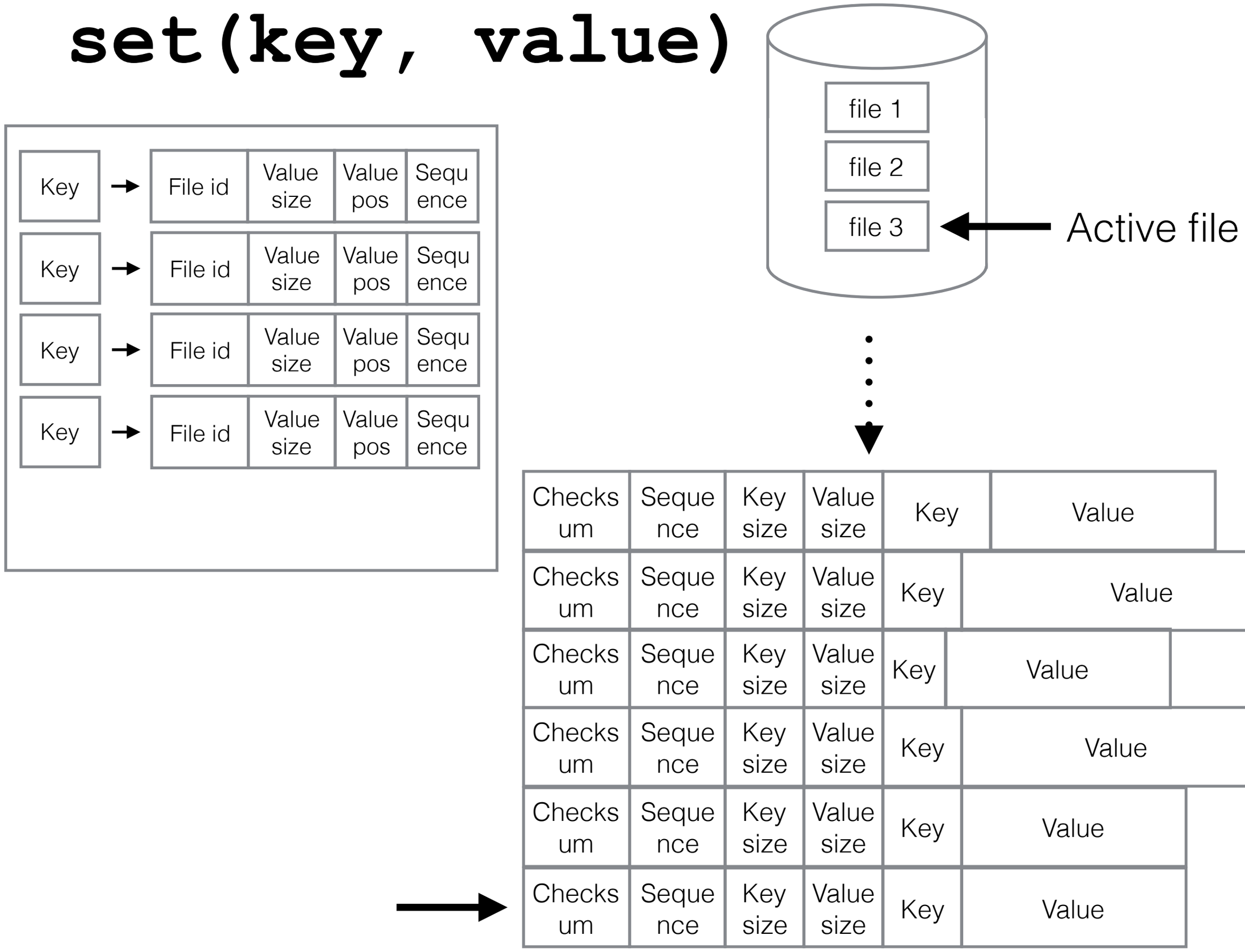


Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value

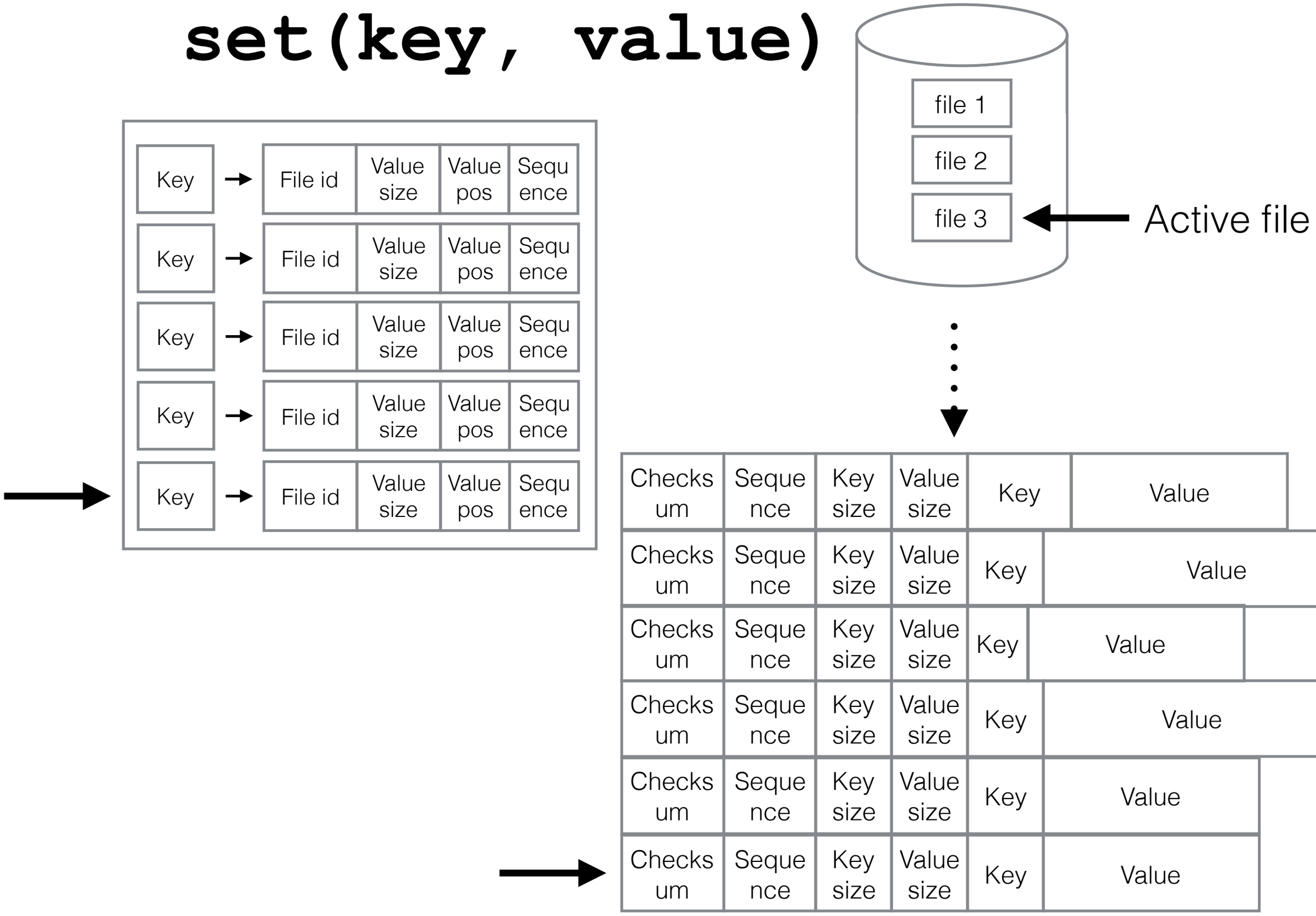
set(key, value)



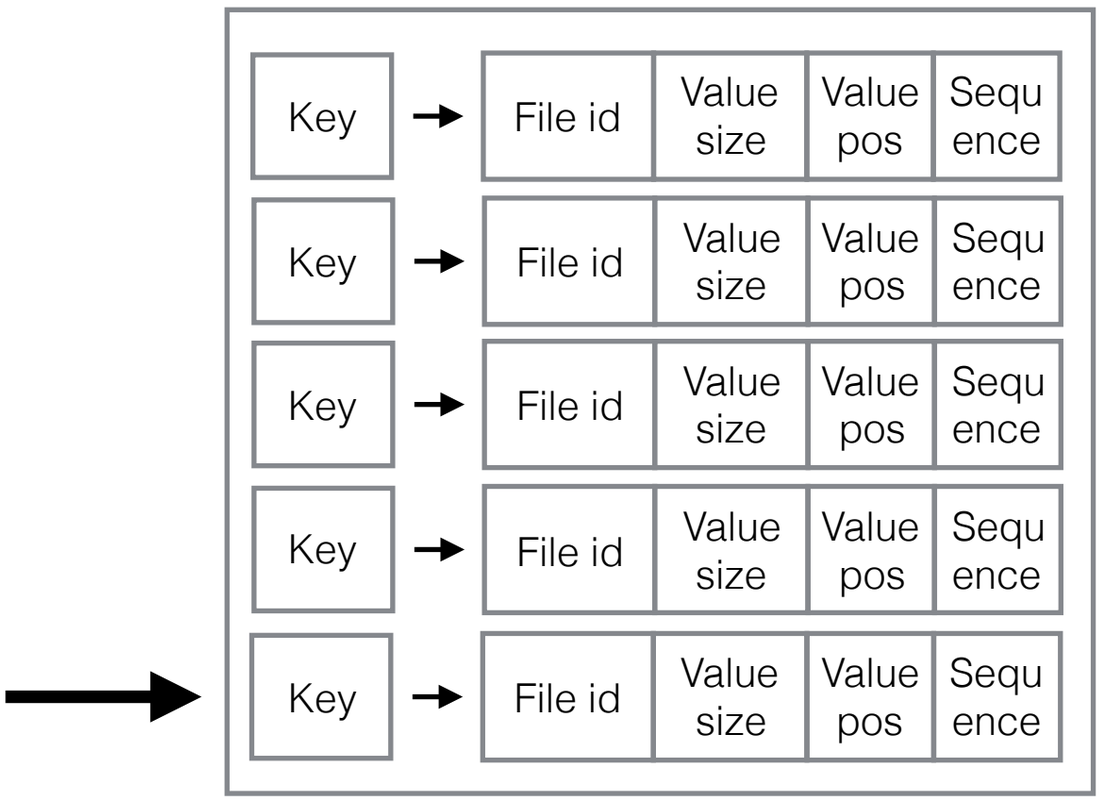
set(key, value)

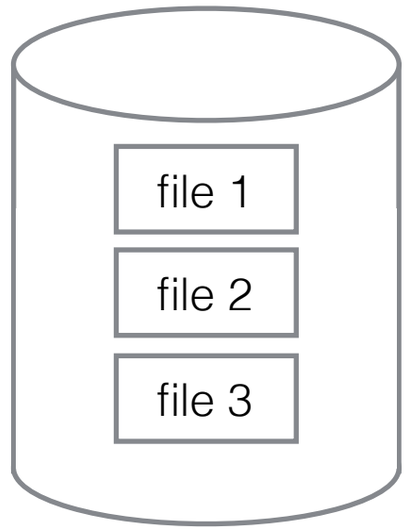


set(key, value)



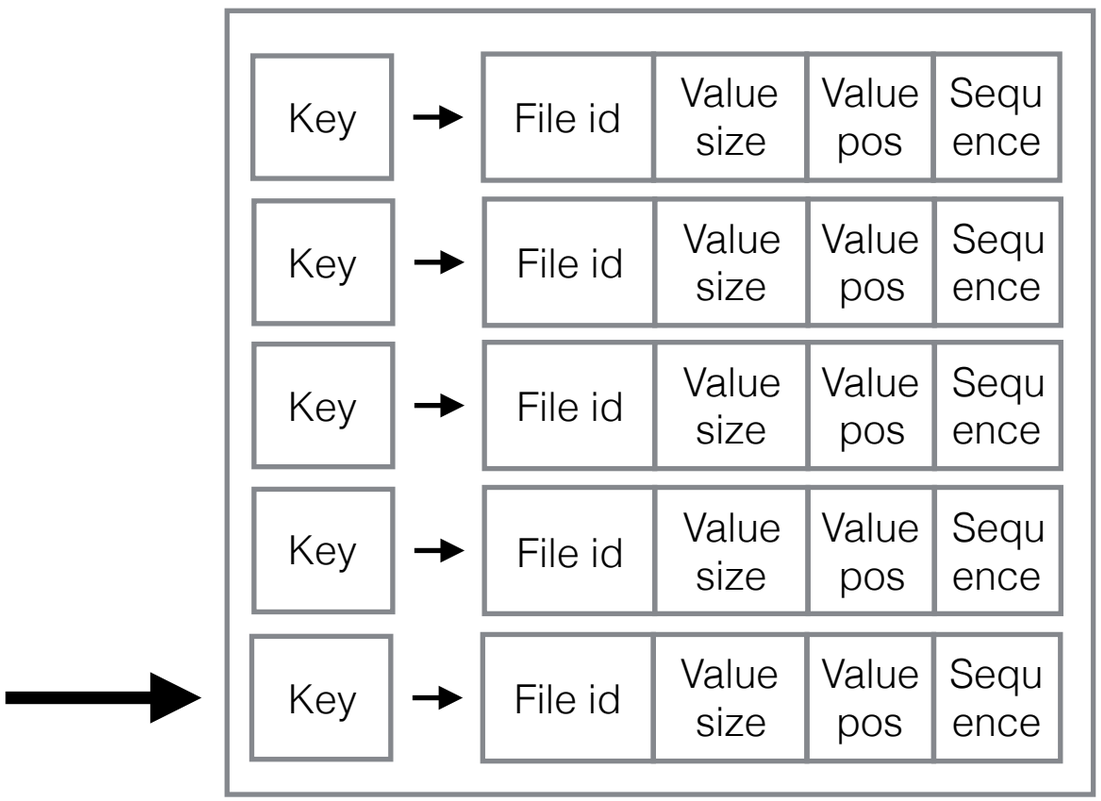
delete(key)

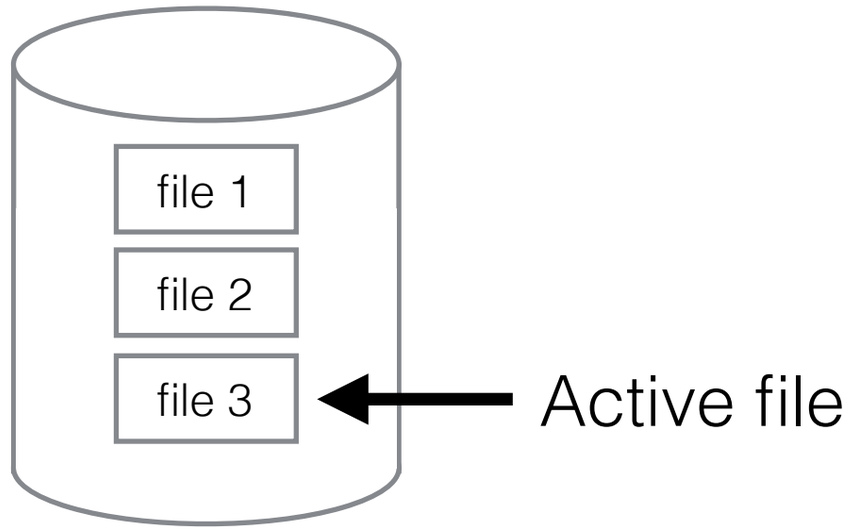




Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value

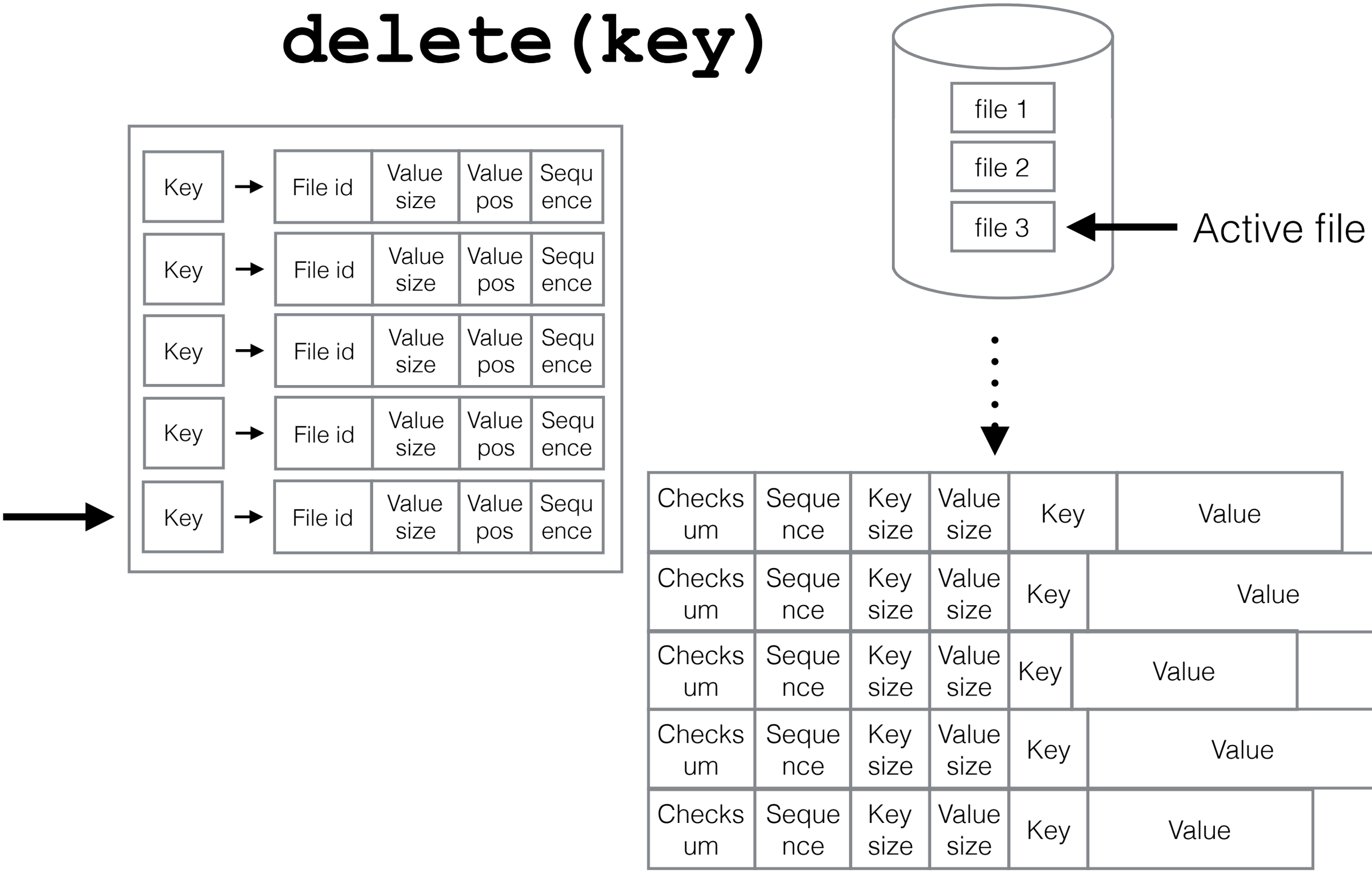
delete(key)



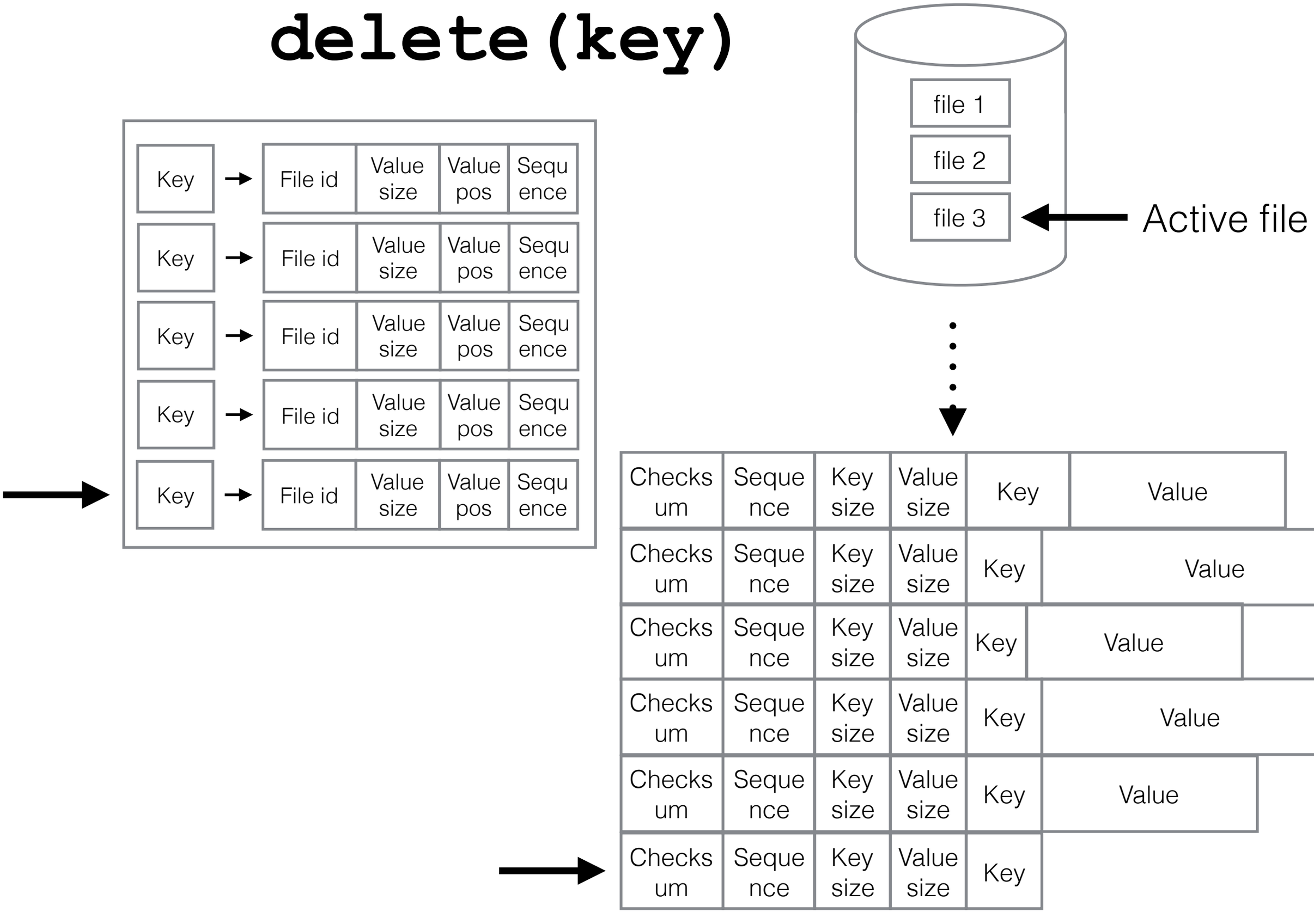


Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value
Checksum	Sequence	Key size	Value size	Key	Value

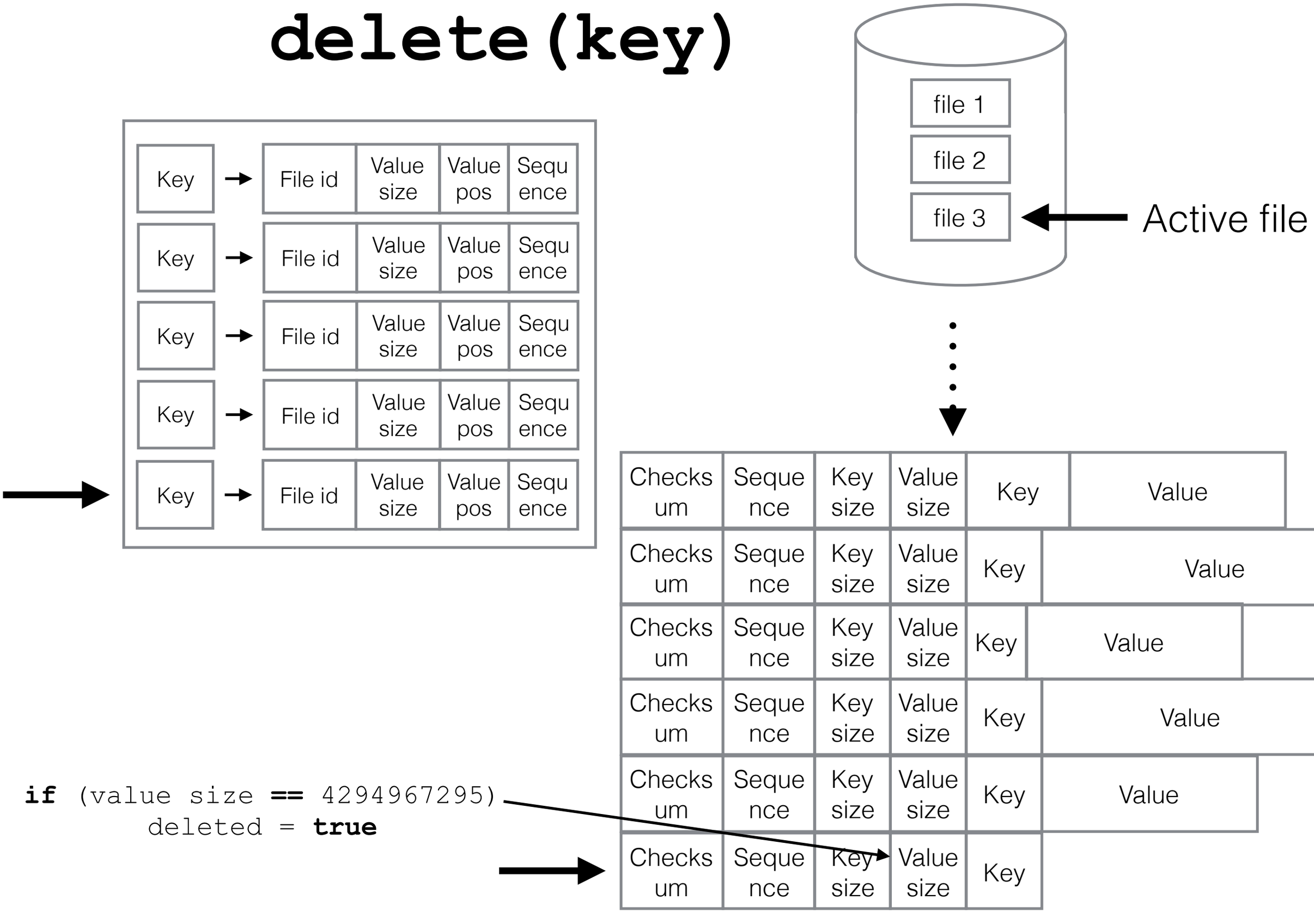
delete(key)


Key
File id
Value size
Value pos
Sequence
file 1
file 2
file 3
Active file
Checksum
Sequence
Key size
Value size
Key
Value

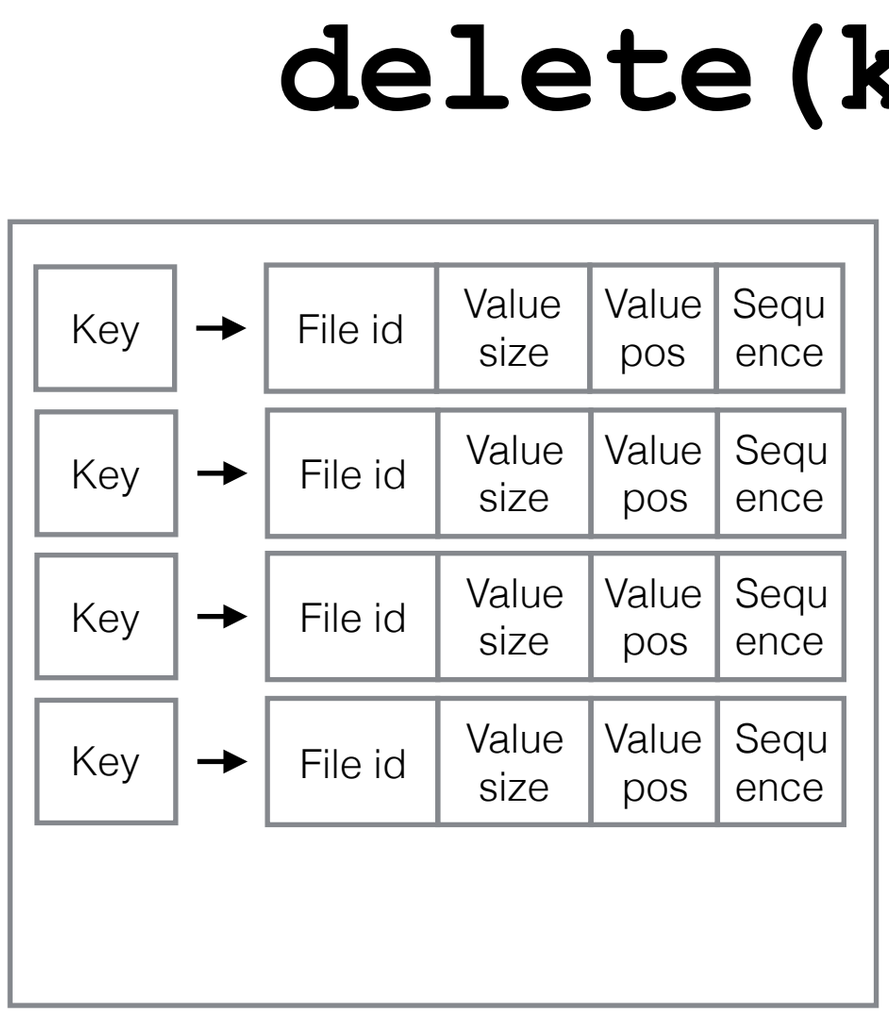
delete(key)

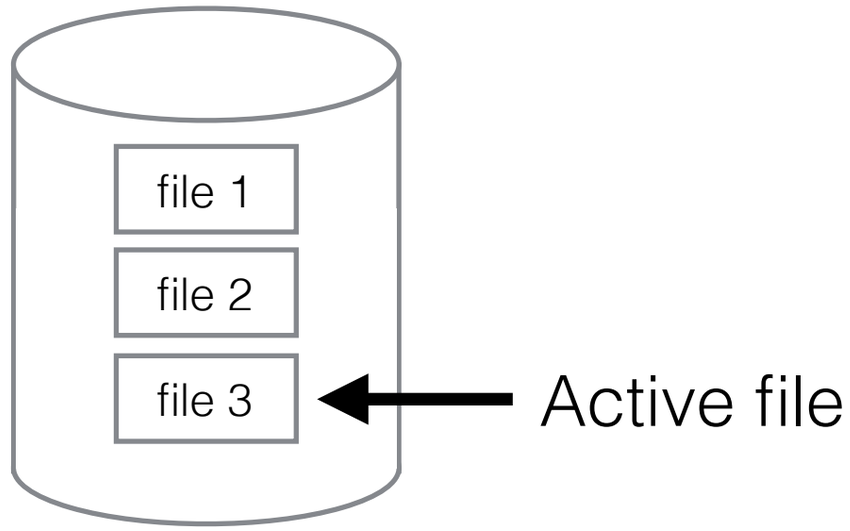

Key
File id
Value size
Value pos
Sequence
file 1
file 2
file 3
Active file
Checksum
Sequence
Key size
Value size
Key
Value

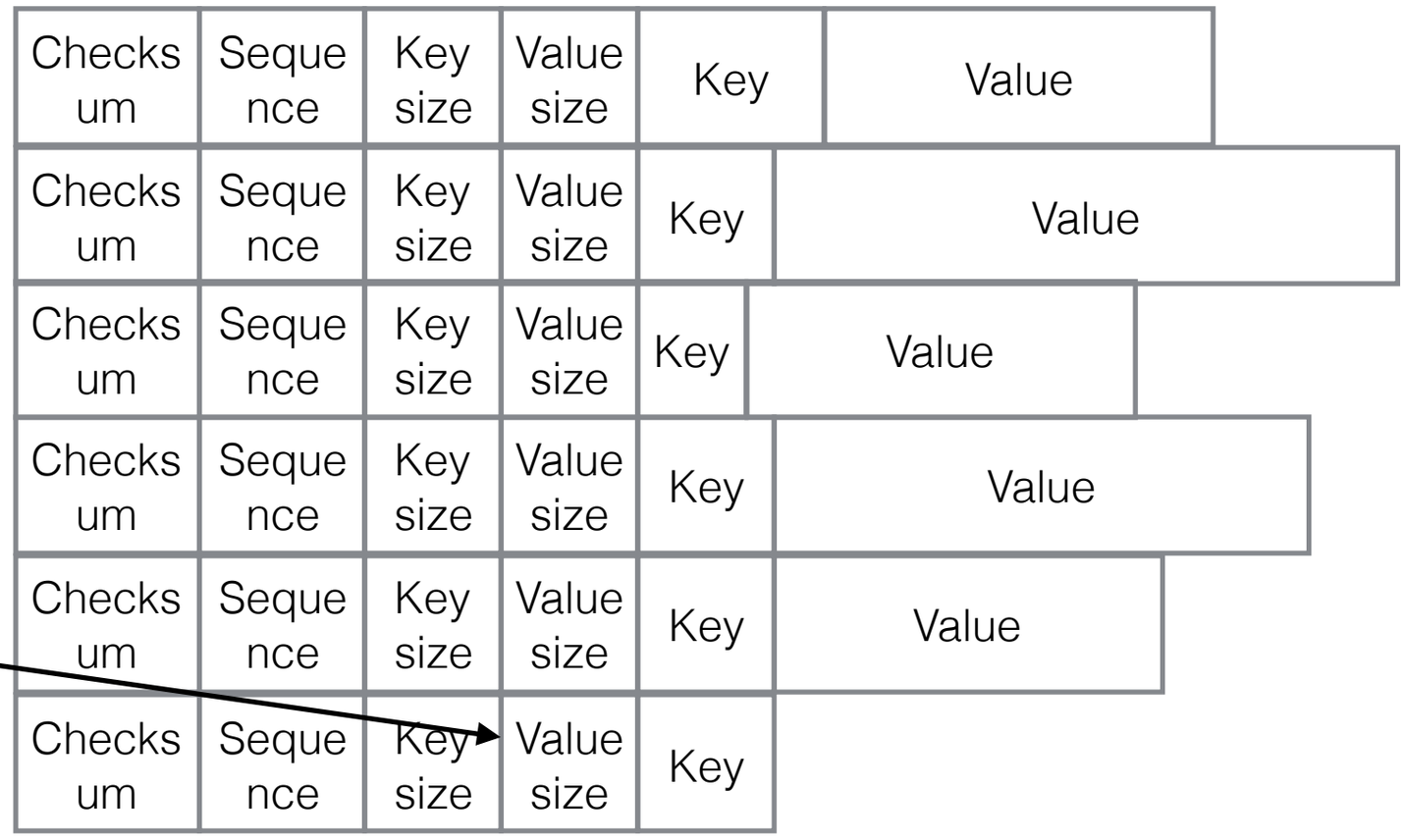
delete(key)


Key
File id
Value size
Value pos
Sequence
file 1
file 2
file 3
Active file
Checksum
Sequence
Key size
Value size
Key
Value
if (value size == 4294967295)
deleted = true

delete(key)








```
if (value size == 4294967295)
        deleted = true
```

Analysis

- Predictable
    - At most one disk seek for reads
    - No seeking for writes
- OS's filesystem read-ahead cache friendly
- Append-only is crash friendly

Startup

Startup

- What happens when we open the key-value store again?

Startup

- What happens when we open the key-value store again?
    - Must read the complete log to recreate index

Startup

```
for file in data_files {
  for entry in file {
    let index_entry = index.get(entry.key)
    if (!index_entry && !entry.deleted)
      index.insert(entry)
    else if (index_entry.sequence <= entry.sequence) {
      if (entry.deleted) index.delete(entry.key)
      else index.insert(entry)
    }
  }
}
```

Except it will fill all your disks

Garbage collection

- We need to **delete dead entries** from the log

Garbage collection

- We need to **delete dead entries** from the log

"hello"	"world"
"foo"	"bar"
"hello"	"baz"
"key"	"value"
delete	"key"

(simplified log)

Garbage collection

- We need to **delete dead entries** from the log

"hello"	"world"
"foo"	"bar"
"hello"	"baz"
"key"	"value"
delete	"key"

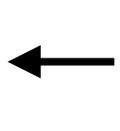
← dead

(simplified log)

Garbage collection

- We need to **delete dead entries** from the log

"hello"	"world"
"foo"	"bar"
"hello"	"baz"
"key"	"value"
delete	"key"

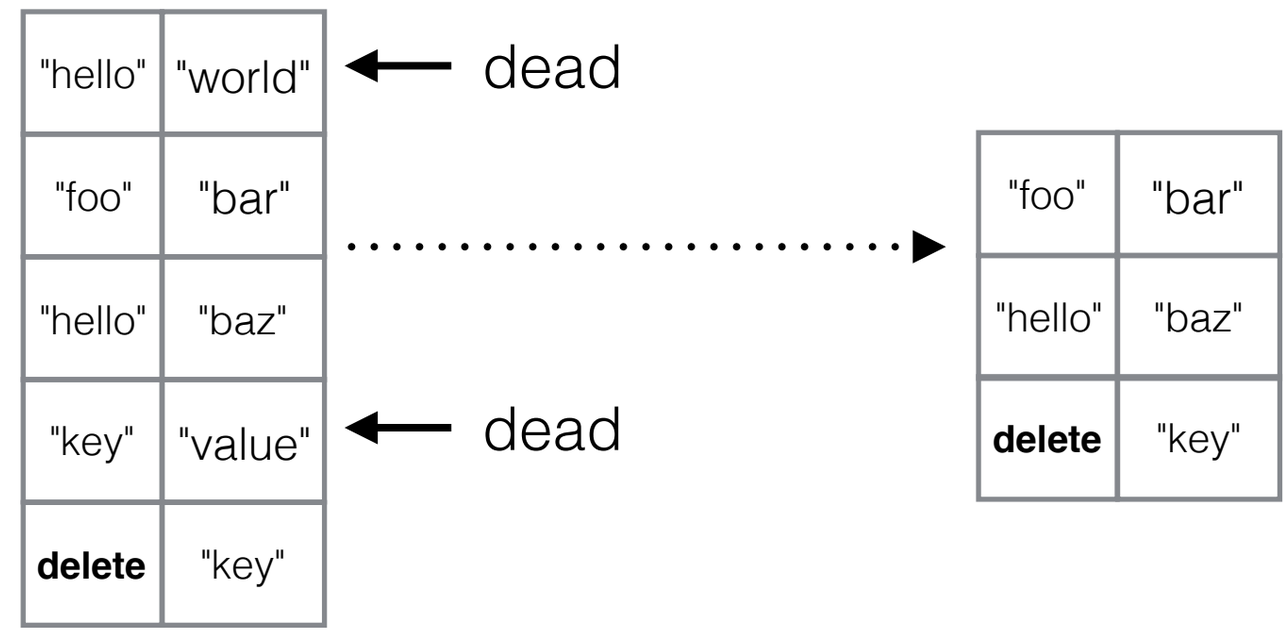

dead


dead

(simplified log)

Garbage collection

- We need to **delete dead entries** from the log



(simplified log)

Garbage collection

```
let deletes = HashMap::new()
for entry in file {
  let index_entry = index.get(entry.key)
  if (index_entry && index_entry.sequence == entry.sequence) {
    log.write(entry)
  }
  else if (!index_entry && entry.deleted) {
    let delete = deletes.get(entry.key)
    if (!delete || delete < entry.sequence)
      deletes.insert(entry.key, entry.sequence)
  }
}

for (key, sequence) in deletes {
  log.write(Entry::deleted(sequence, key))
}
```

Garbage collection

```
let deletes = HashMap::new()
for entry in file {
  let index_entry = index.get(entry.key)
  if (index_entry && index_entry.sequence == entry.sequence) {
    log.write(entry)
  }
  else if (!index_entry && entry.deleted) {
    let delete = deletes.get(entry.key)
    if (!delete || delete < entry.sequence)
      deletes.insert(entry.key, entry.sequence)
  }
}

for (key, sequence) in deletes {
  log.write(Entry::deleted(sequence, key))
}
```

Garbage collection

- Background thread
- Creates a new file with only "live" entries
- Updates all the entries in the index to point to the new file
- Delete the old file

Where is the garbage?

- Keep statistics for the amount of "garbage" in each file
  - Dead entries
  - Dead bytes
  - Fragmentation
    - `dead_entries / total_entries`

All the keys must be in-memory

What's the index size?

- Overhead per key: ~ 40 bytes
- Lets assume keys are UUID strings: 32 bytes
- Given 4GB of memory we can store
  - ~ 55 million keys

What's the index size?

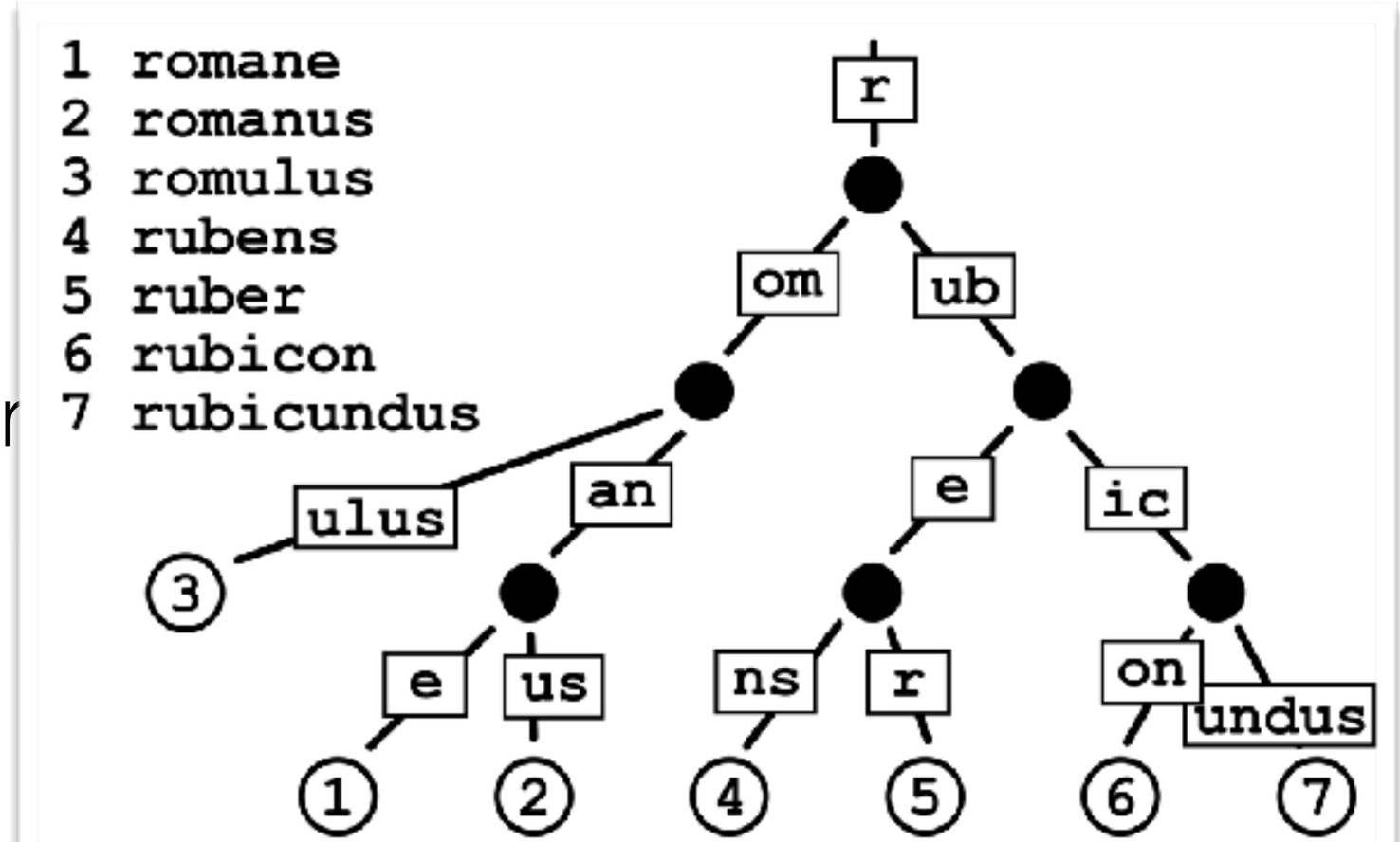
- Overhead per key: ~ 40 bytes
- Lets assume keys are UUID strings: 32 bytes
- Given 4GB of memory we can store
  - ~ 55 million keys

We can do better

- If keys are **"hierarchical"**
  - DNS names
  - IP addresses
- Use a **radix trie** for the **index**

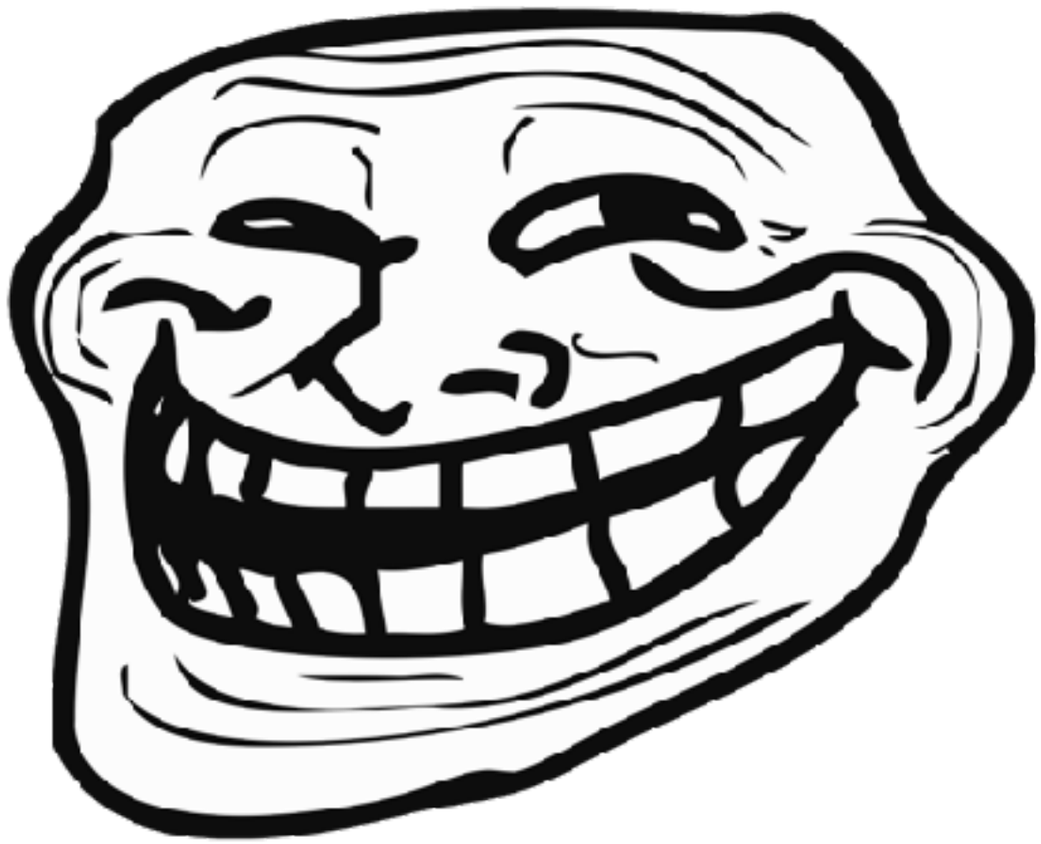
We can do better

- If keys are **"hierarchical"**
  - DNS names
  - IP addresses
- Use a **radix trie** for



What's the index size?

What's the index size?



Just kidding

Demo time

Going further

- Multiple logs to increase write throughput
- Parallel garbage collection
- Ordering
  - Log-structured merge-tree (LevelDB)

References

- https://github.com/andrebeat/cask
  - **Rust** implementation
- https://github.com/basho/bitcask
  - Original implementation in **C** and **Erlang**
- http://basho.com/wp-content/uploads/2015/05/bitcask-intro.pdf

Questions?

Speedup startup