Badger: Fast Key-Value DB in Go

Manish R Jain, Dgraph Labs
Apr 14, 2018
Gopher China, Shanghai

Dgraph Labs

- Fast, Distributed graph database.
- Sparse data sets.
- Lots of relationships.

https://dgraph.io

What is Badger?

- Badger is an embedded key-value database, written in Go.
- Licensed under Apache 2.0.

```
go get github.com/dgraph-io/badger/...
```

Current Status

- Closing v2.0.
- Close to 3500 Github starts.
- 42 contributors.
- Used by Dgraph, Go-IPFS, 0-stor, Sandglass.

Serving 300TB (and growing) at Usenet Express

Basic Operations

Set a key-value

```
func set() error {
    fmt.Println("\nRunning SET")
    return db.Update(func(txn *badger.Txn) error {
        if err := txn.Set([]byte("foo"), []byte("bar")); err != nil {
            return err
        }
        fmt.Println("Set foo to bar")
        return nil
    })
}
```

Get a key-value

```
func get() error {
    fmt.Println("\nRunning GET")
    return db.View(func(txn *badger.Txn) error {
        item, err := txn.Get([]byte("foo")) // handle err
        if err != nil {
            return err
        }
        val, err := item.Value() // handle err
        if err != nil {
            return err
        }
        fmt.Printf("The value is: %s\n", val)
        return nil
    })
}
```

Iterate key-values

```
func iterate() error {
    fmt.Println("\nRunning ITERATE")
    return db.View(func(txn *badger.Txn) error {
        opts := badger.DefaultIteratorOptions
        it := txn.NewIterator(opts)
        defer it.Close()

        for it.Rewind(); it.Valid(); it.Next() {
            k := it.Item().Key()
            v, err := it.Item().Value() // handle err
            if err != nil {
                return err
            }
            fmt.Printf("key=%s, value=%s\n", k, v)
        }
        return nil
    })
}
```

Run the code

```
func main() {
    opt := badger.DefaultOptions
    opt.Dir = "/tmp/db"
    opt.ValueDir = opt.Dir
    var err error
    db, err = badger.Open(opt)
    if err != nil {
        panic(err)
    }
    defer db.Close()
    fmt.Println("DB opened")
    set()
    get()
    iterate()
    fmt.Println("DB done")
}
```

Run

Badger != replacement for Go map

Motivation and Outcome

Cgo is not Go

Some people, when confronted with a problem, think
“I know, I’ll use cgo.”

Now they have two problems.

->Cgo is not Go, Dave Cheney

RocksDB

- Great write throughput.
- Okay read throughput.

Cons:

- Required Cgo.

BoltDB

- Pure Go.
- Great read throughput.

Cons:

- Bad write throughput.

Why build it?

- Go native key-value DB for Dgraph.
- No compromise in read-write performance.
- Avoid Cgo.

What did we spend?

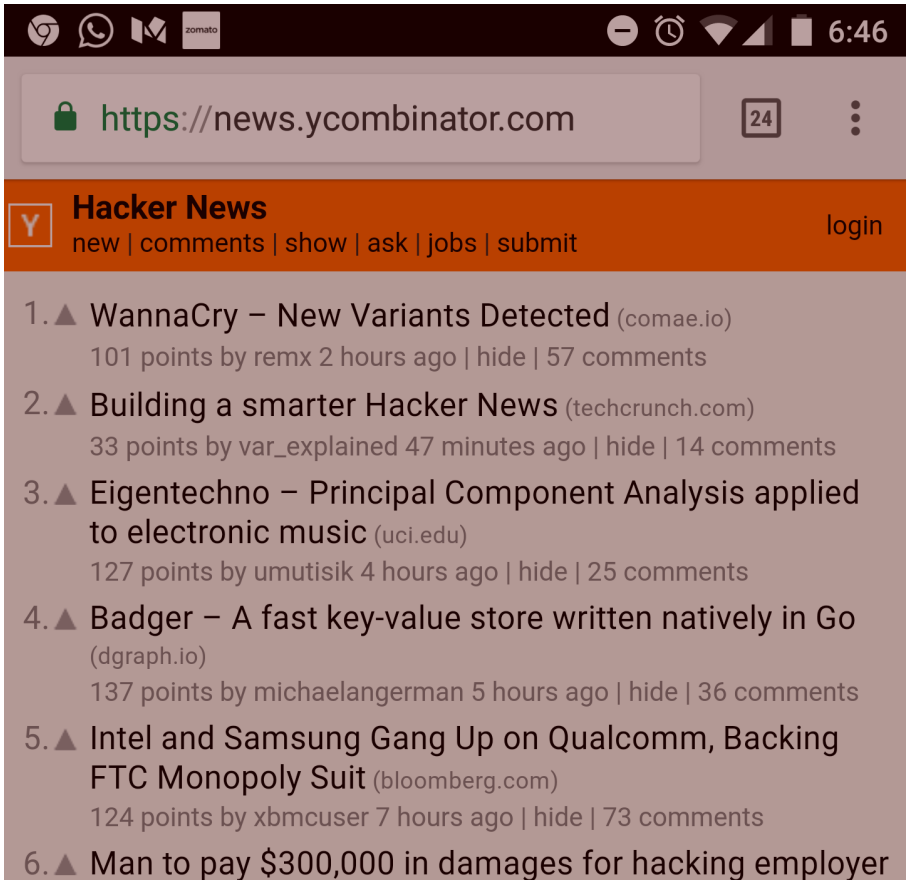
- Spent a few months.
- Built with <1 full-time gopher.
- Aka, the power of Go!

What did we gain?

- A faster key-value DB for Go.
- Ability to run Go profilers all the way down to disk.
- Clean Go code (no C).

Launch Reception

- Within 12 hours of blog post release
- First page of HN for a day.
- 355 points, 96 comments.
- 1250 Github stars in 4 days.



(bbc.com)
22 points by happy-go-lucky 2 hours ago | hide | 14 comments

7. ▲ Why We Are Self-Publishing the Aviary Cookbook
(medium.com)
31 points by wnm 2 hours ago | hide | 6 comments

8. ▲ Makers of Crowdfunded “Gravity Blanket” Withdraw Unsupported Medical Claims (statnews.com)
30 points by simon_acca 4 hours ago | hide | 26 comments

9. ▲ Startup School 12: Alan Kay, Part II [video]
(startupschool.org)
44 points by sama 4 hours ago | hide | 13 comments

10. ▲ A Mathematician’s Lament (2002) [pdf] (maa.org)
89 points by Tomte 8 hours ago | hide | 16 comments

11. ▲ Data Science Workflow: Overview and Challenges
(acm.org)

Recruiters loved it!

- Got 3 different emails from 3 different recruiters...

Recruiters loved it!

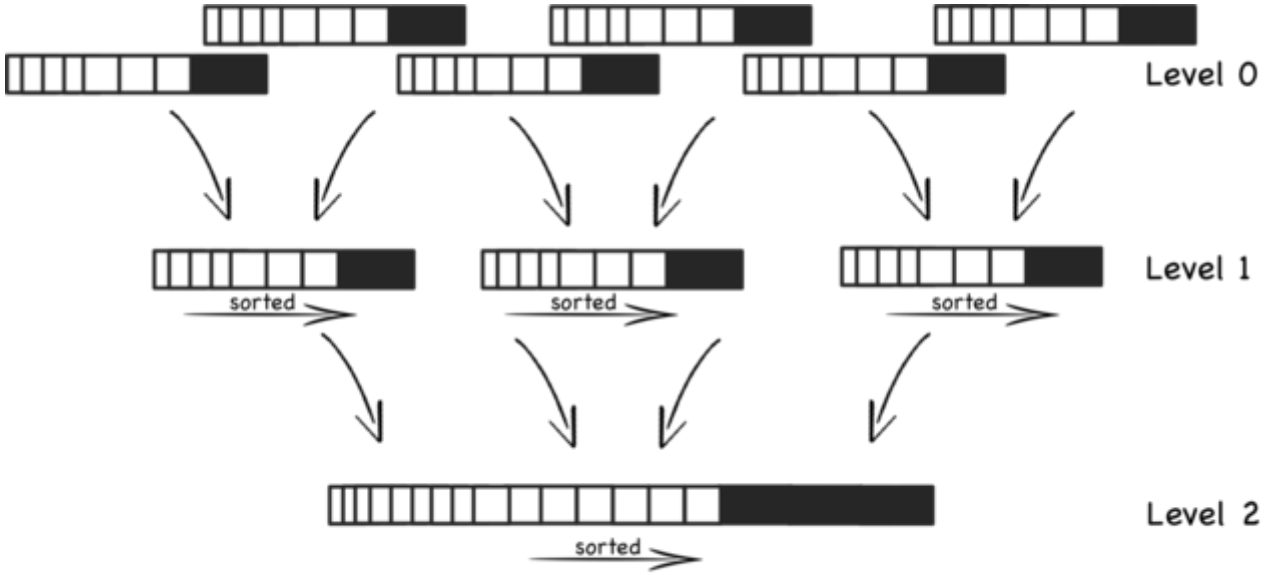
- For jobs in the same company.

Design

Two common Trees

- LSM trees
- B+ trees

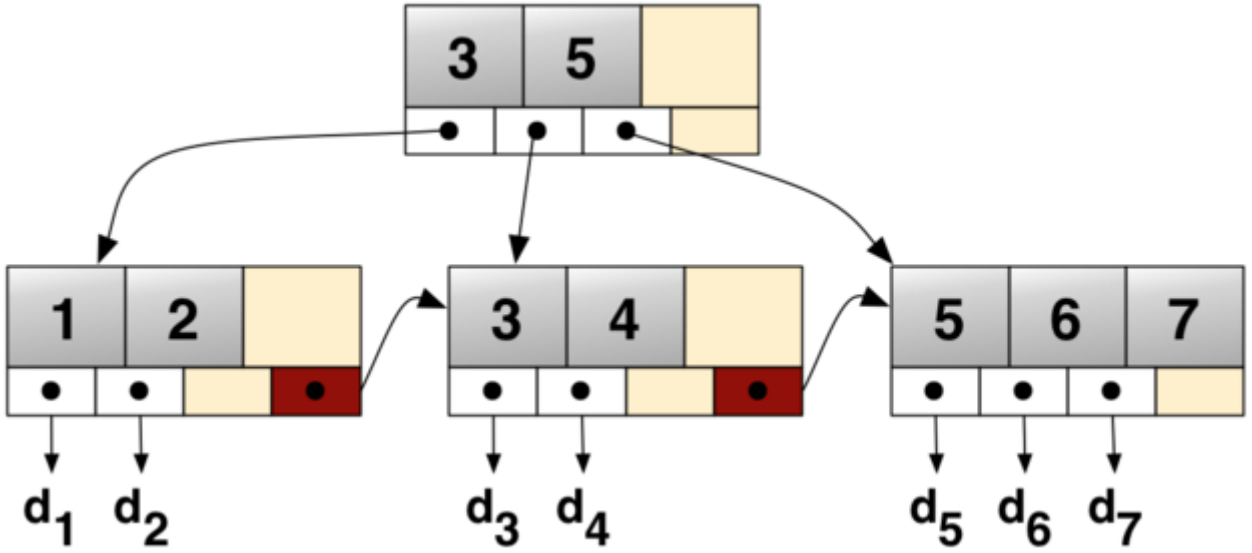
LSM Trees



Compaction continues creating fewer, larger and larger files

- More levels
- High write throughput
- High read latency
- Example: RocksDB

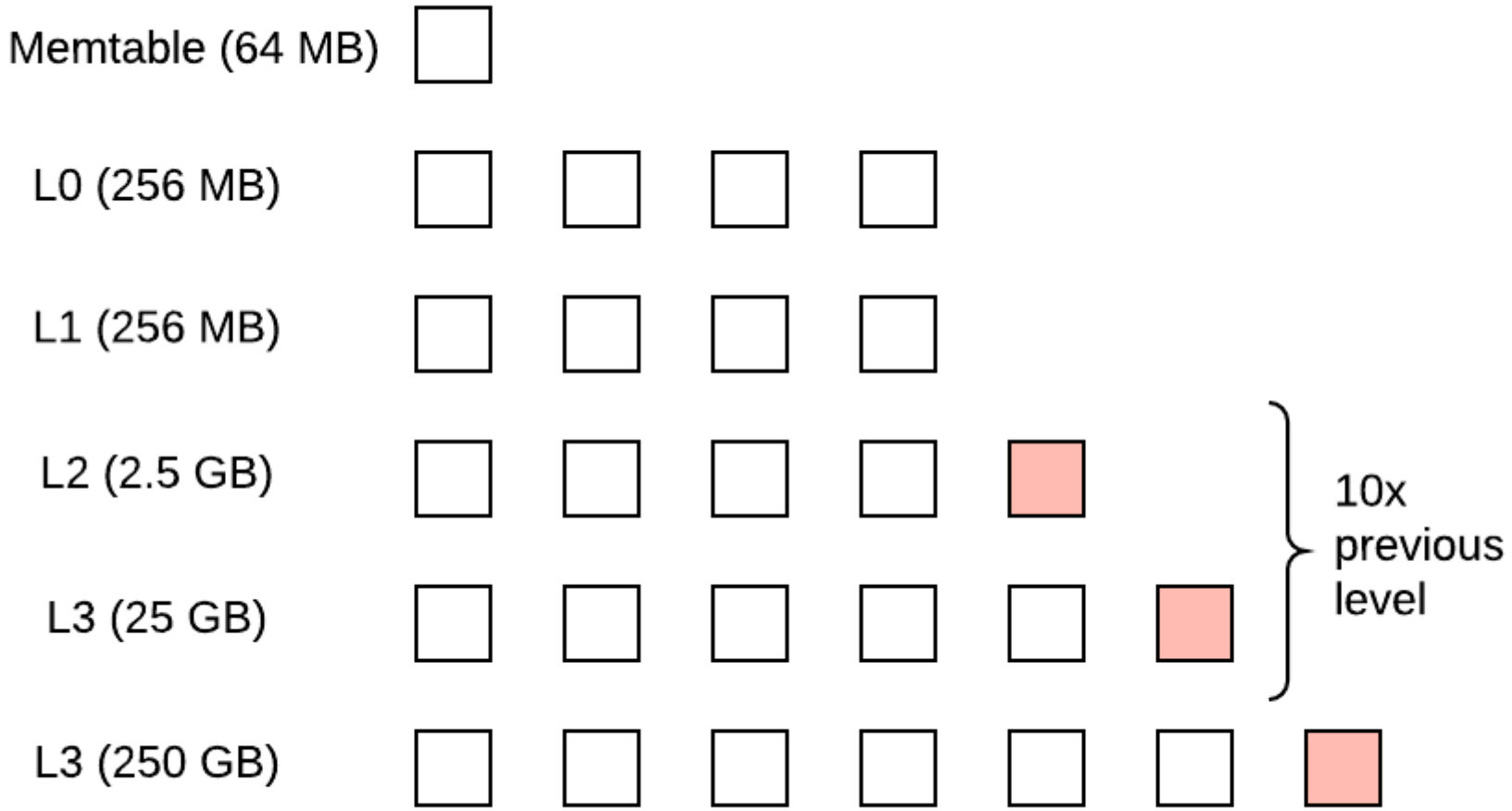
B+ Trees



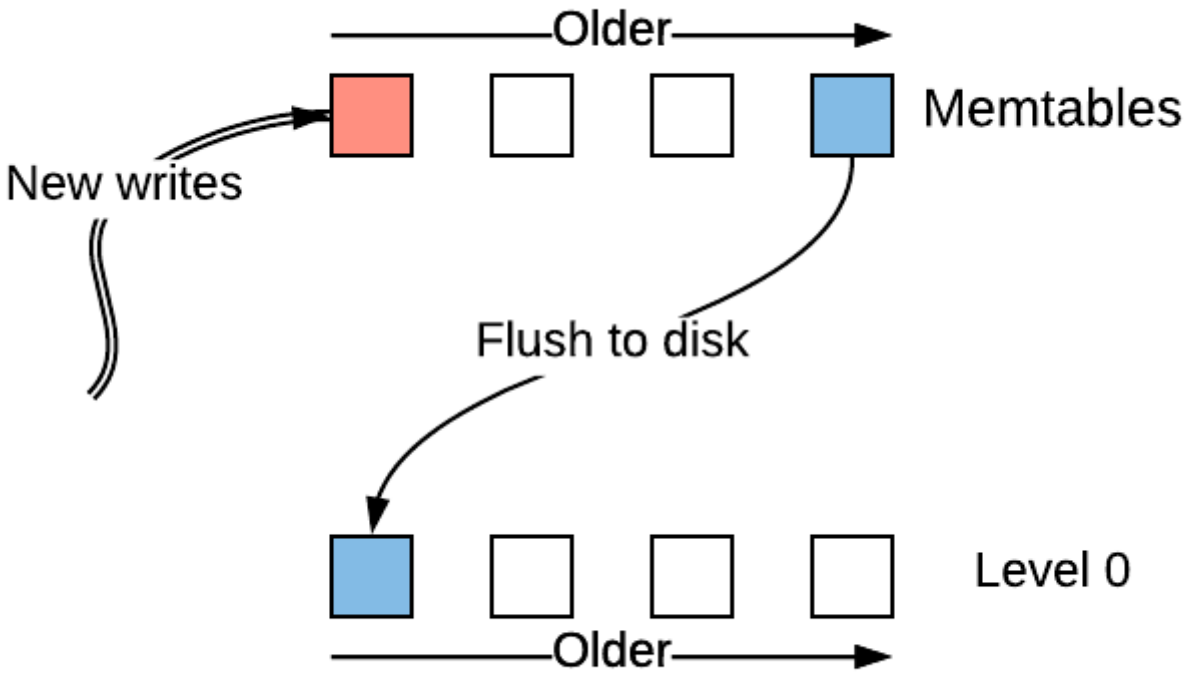
- Fewer levels
- Low write throughput
- Low read latency
- Example: BoltDB

Badger is based on LSM trees.

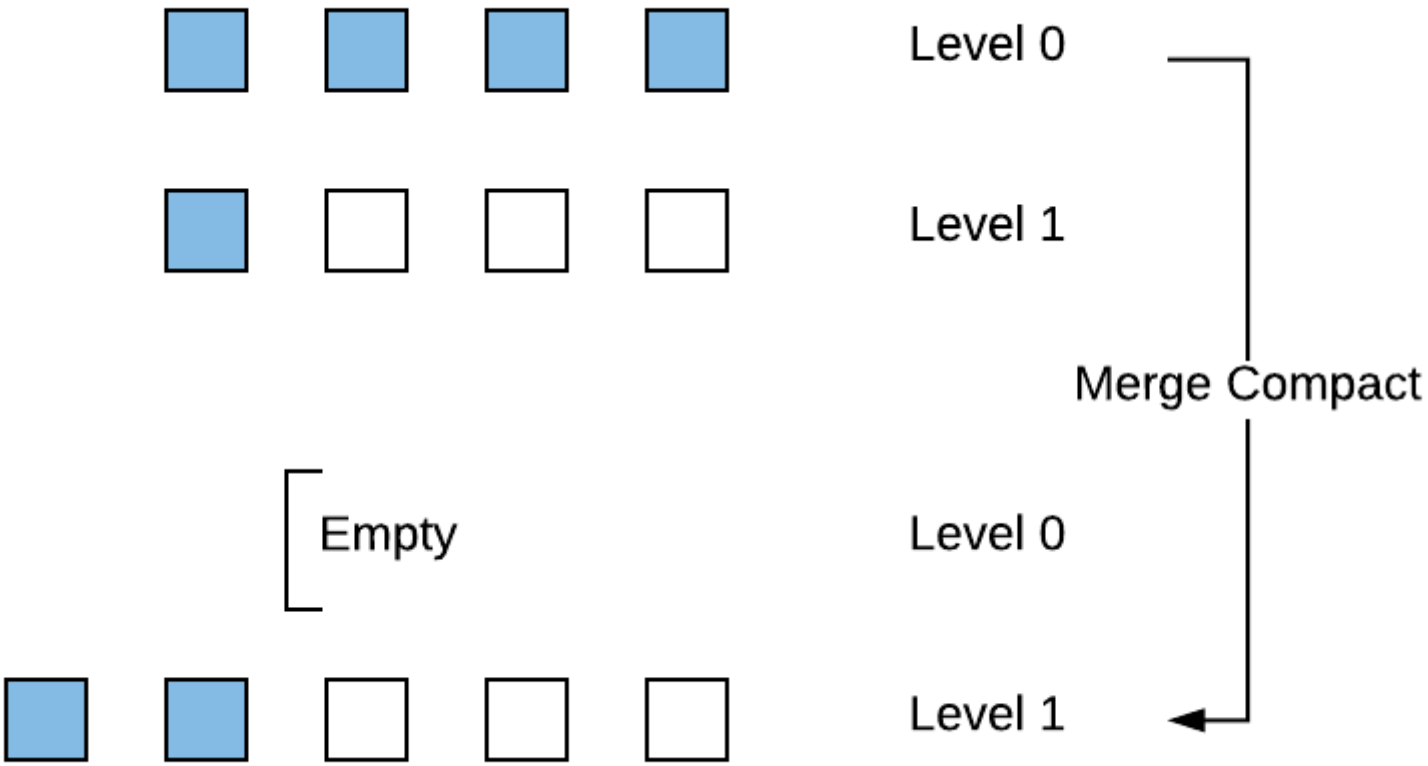
LSM Trees


Memtable (64 MB)
L0 (256 MB)
L1 (256 MB)
L2 (2.5 GB)
L3 (25 GB)
L3 (250 GB)
10x
previous
level

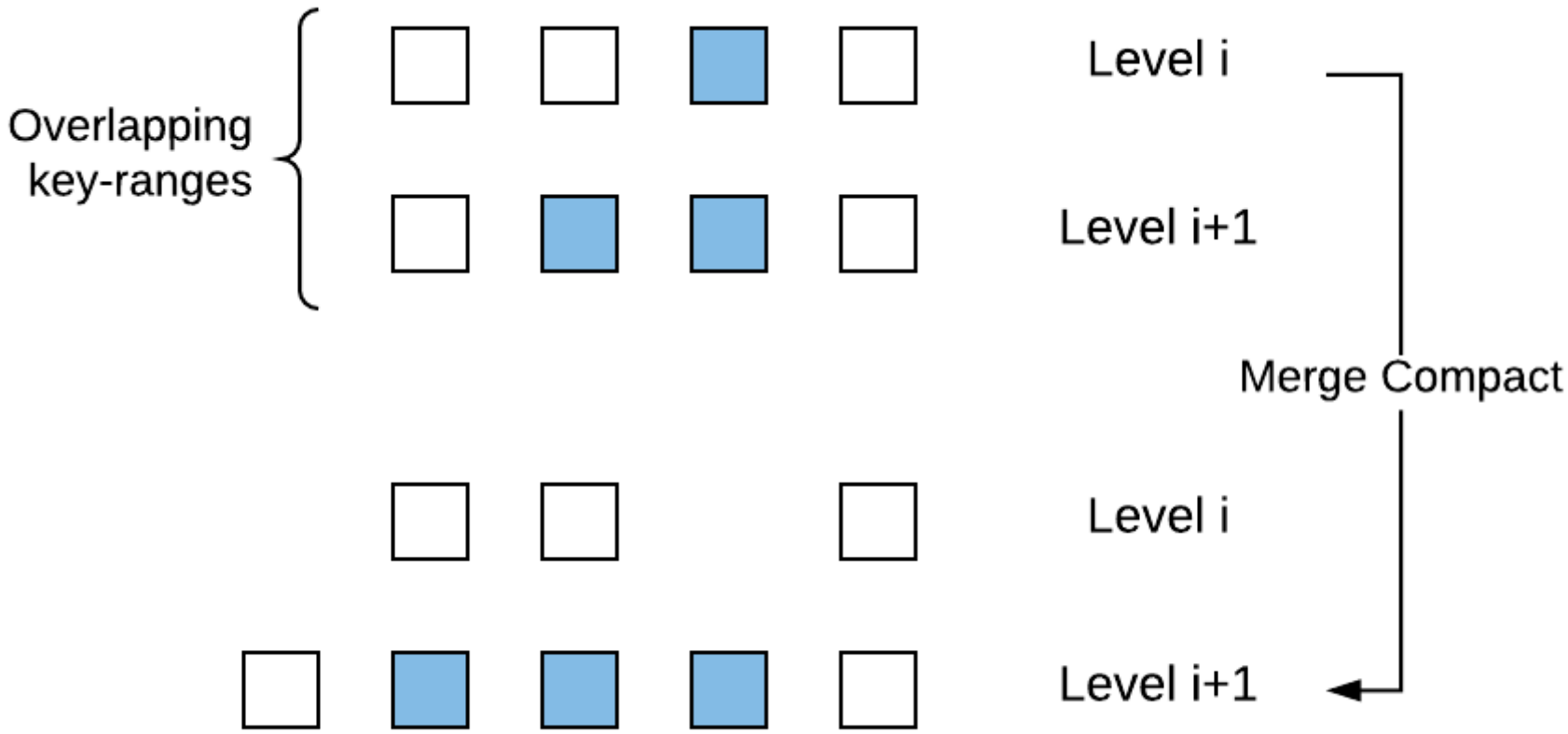
Writes in LSM trees: Memtable to L0



Writes in LSM trees: L0 to L1



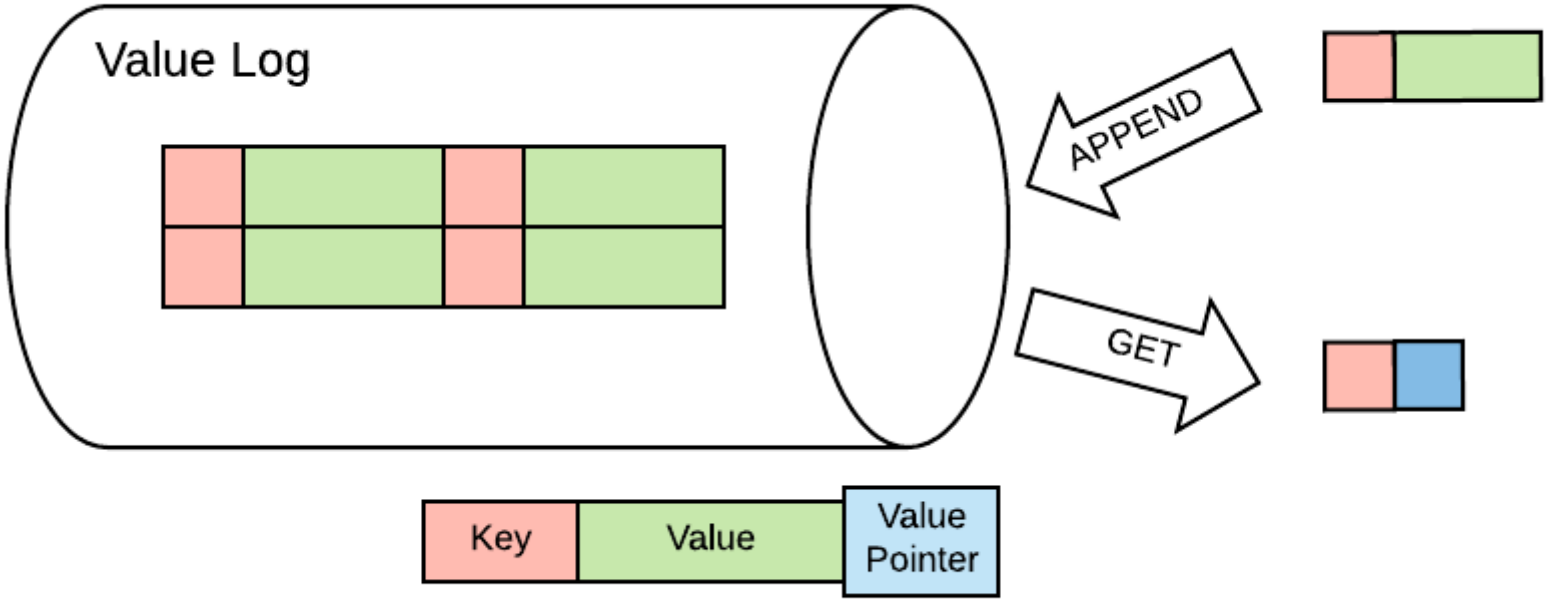
Writes in LSM trees: Li to Li+1



What makes Badger unique?

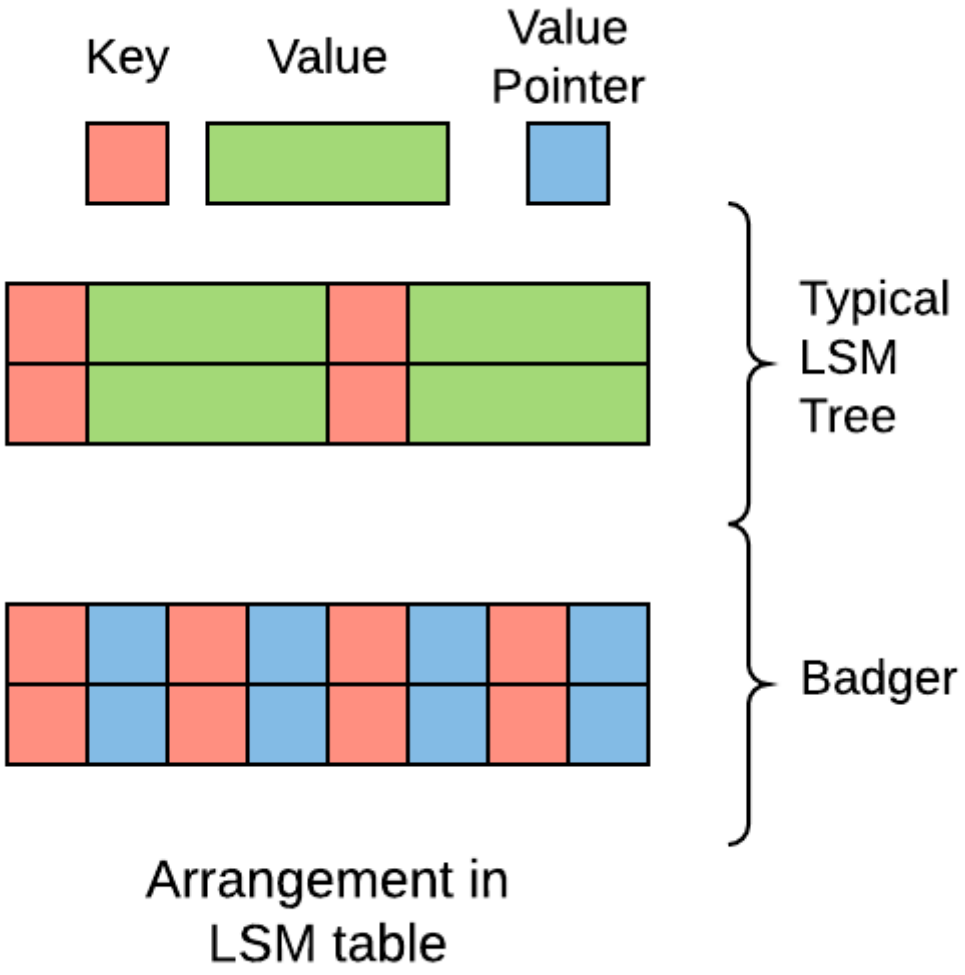
- Based on WiscKey paper by Uni Wisconsin-Madison.
- Separates keys from values.
- Stores keys in LSM tree.
- Stores value in value log.

Write to Value Log

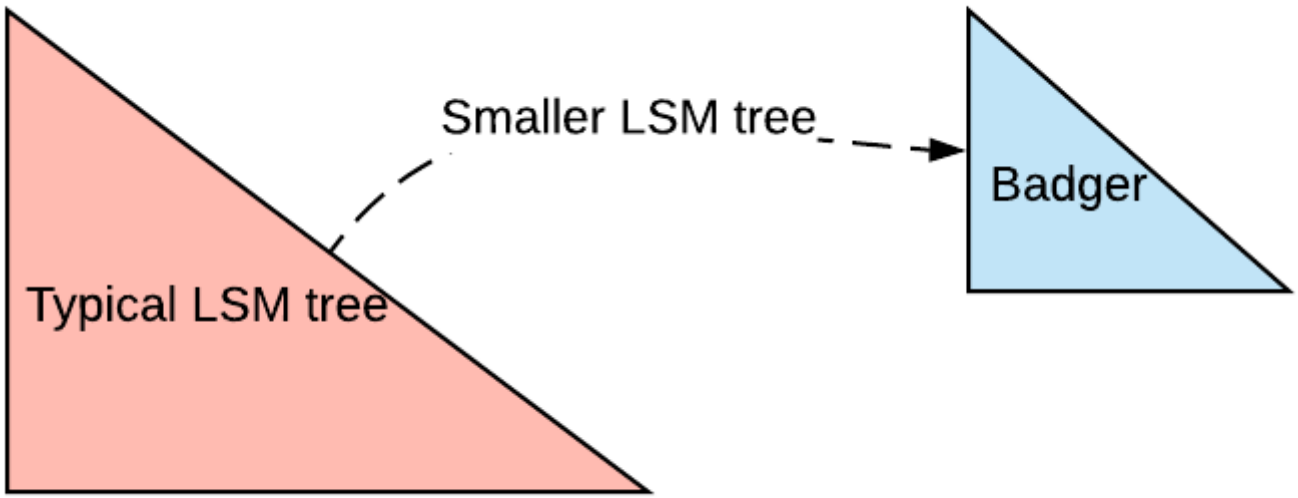


- Write value, get pointer.

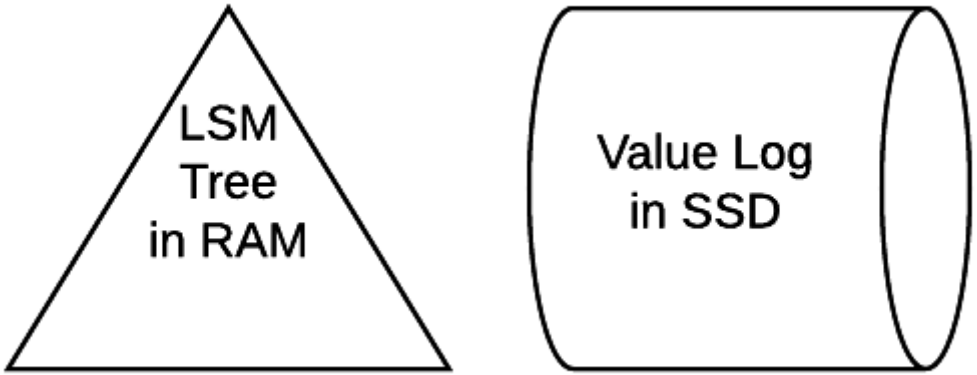
More keys per table



Smaller LSM tree



Typical Badger setup



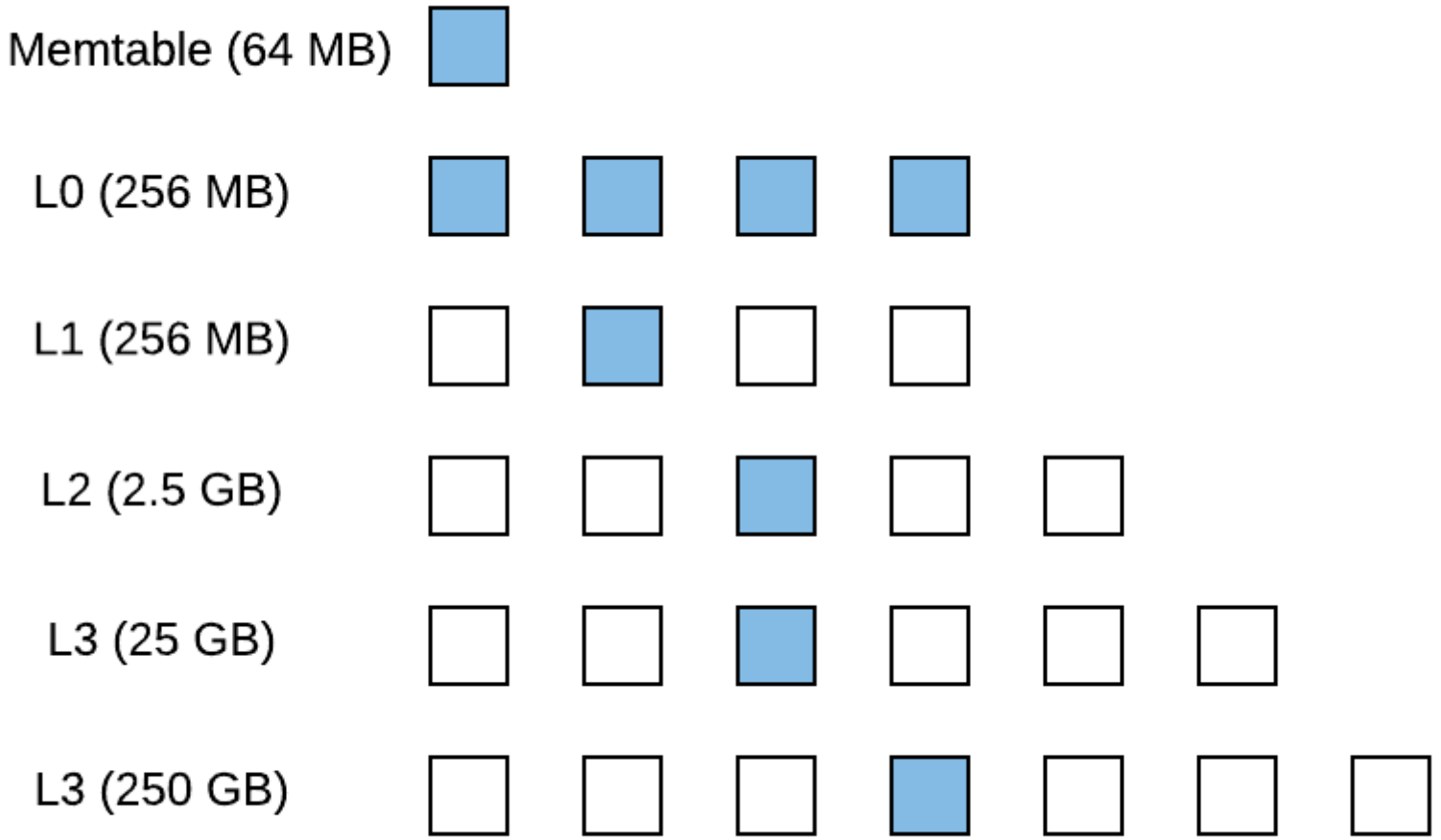
Advantages of smaller LSM tree

- Can be kept in RAM.
- Low read amplification (fewer lookups).
- Low write amplification (fewer compactions).
- $\propto$ Number of keys.

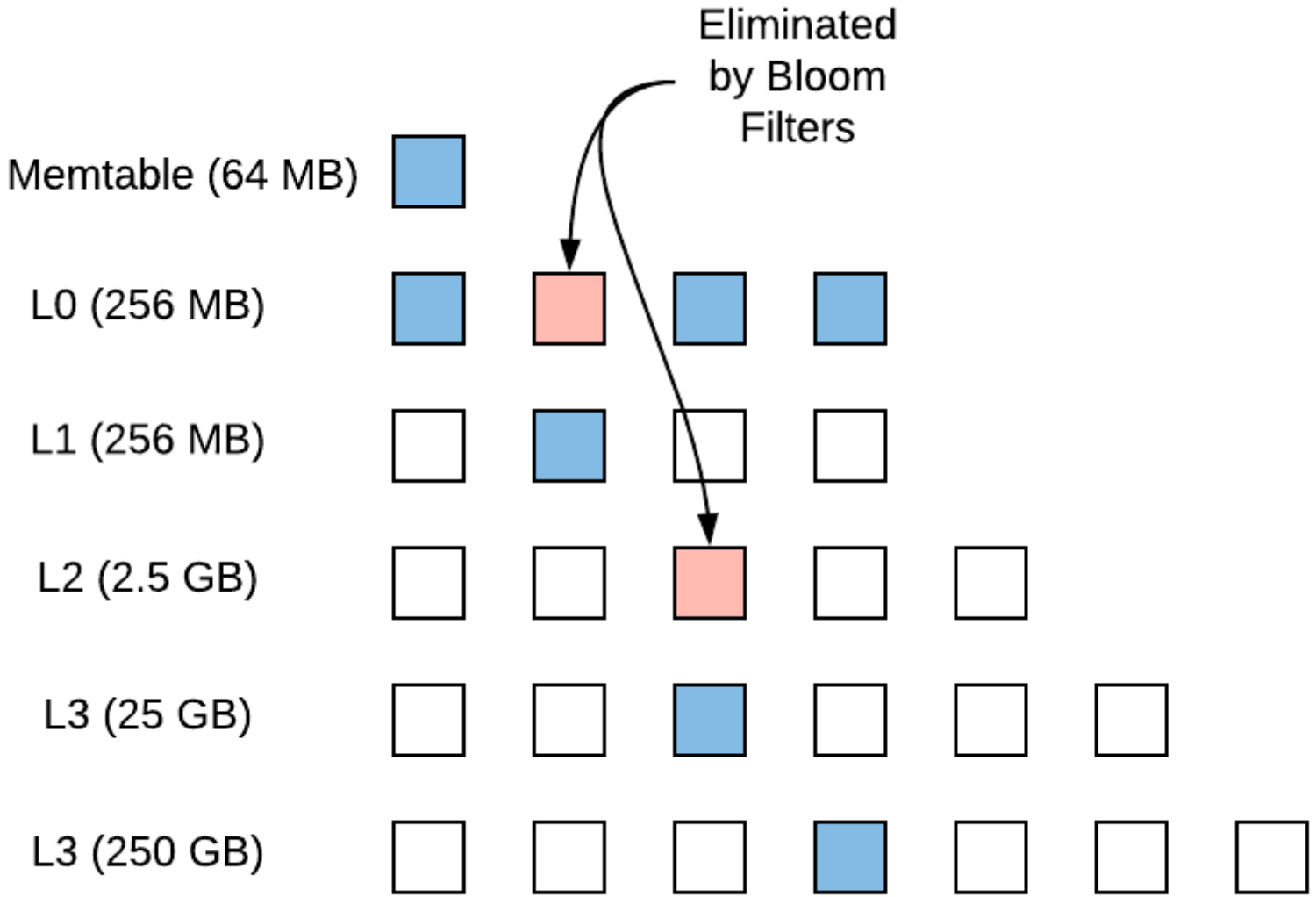
Usenet Express

- Hundreds of terabytes of data.
- Few gigabytes of LSM tree.

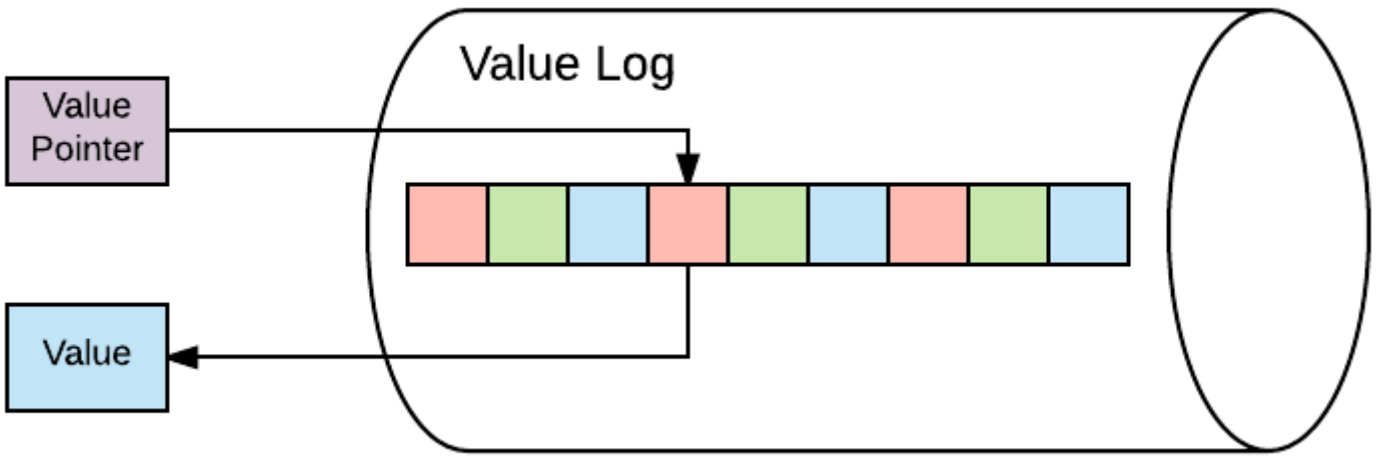
Reads in Badger: LSM tree



Reads in Badger: Bloom Filters



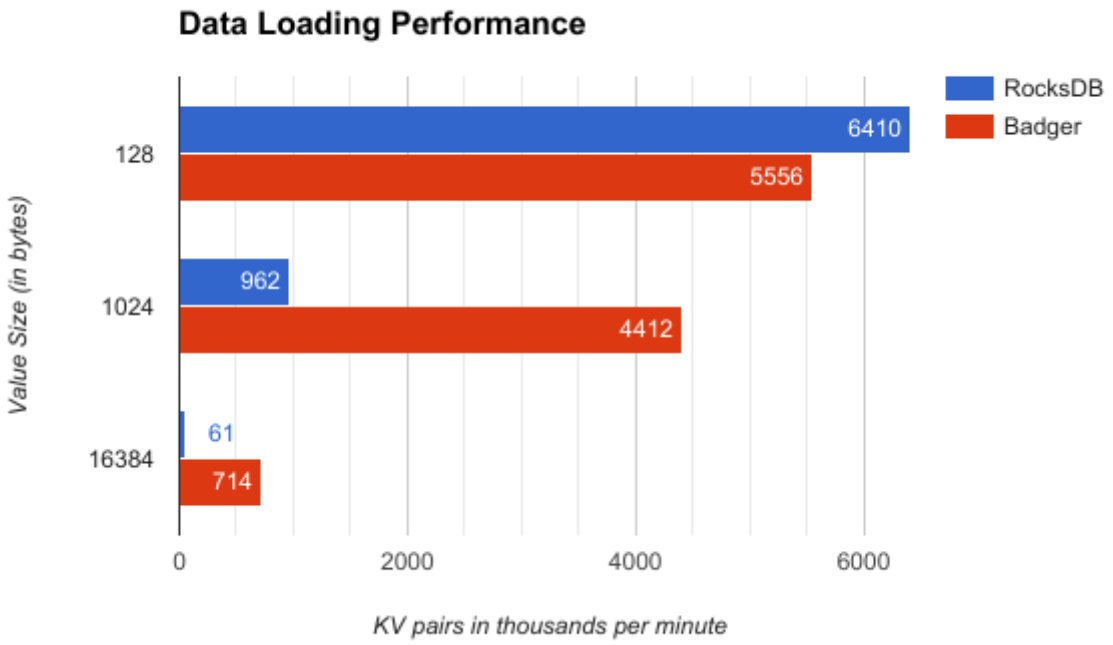
Reads in Badger: Value Log



- Once key found in LSM tree, read from value log.

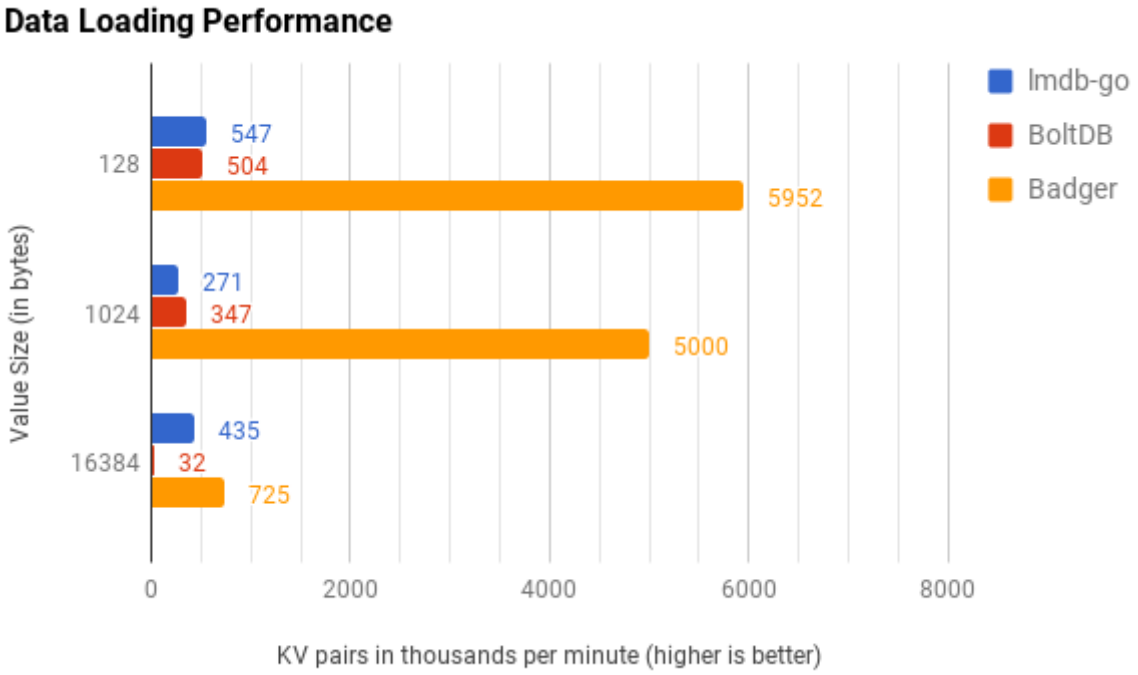
Badger is FAST-er

Data Loading: Badger vs Go-RocksDB



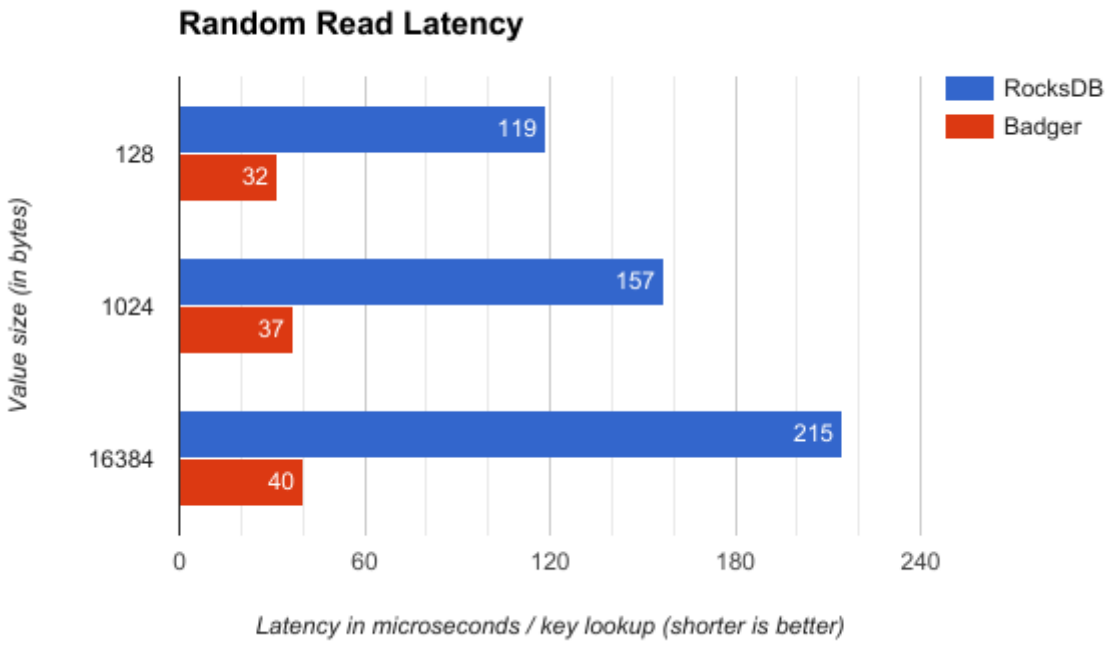
- As value size increases, Badger's becomes 11.7x faster.

Data Loading: Badger vs BoltDB



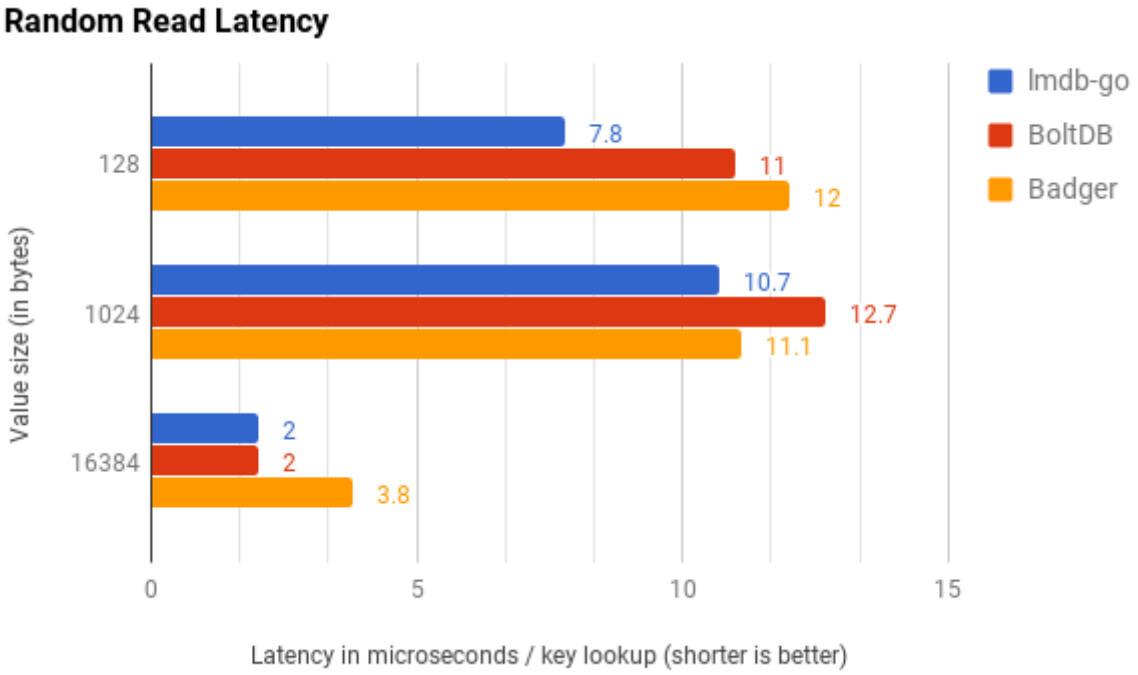
- 11x - 22x faster than BoltDB on all value sizes.

Random Reads: Badger vs Go-RocksDB



- Random Get latency is 3.7x - 5.3x lower than RocksDB.

Random Reads: Badger vs BoltDB



- Random Get latency is slightly better or worse, depending on value size.

Various other benchmarks

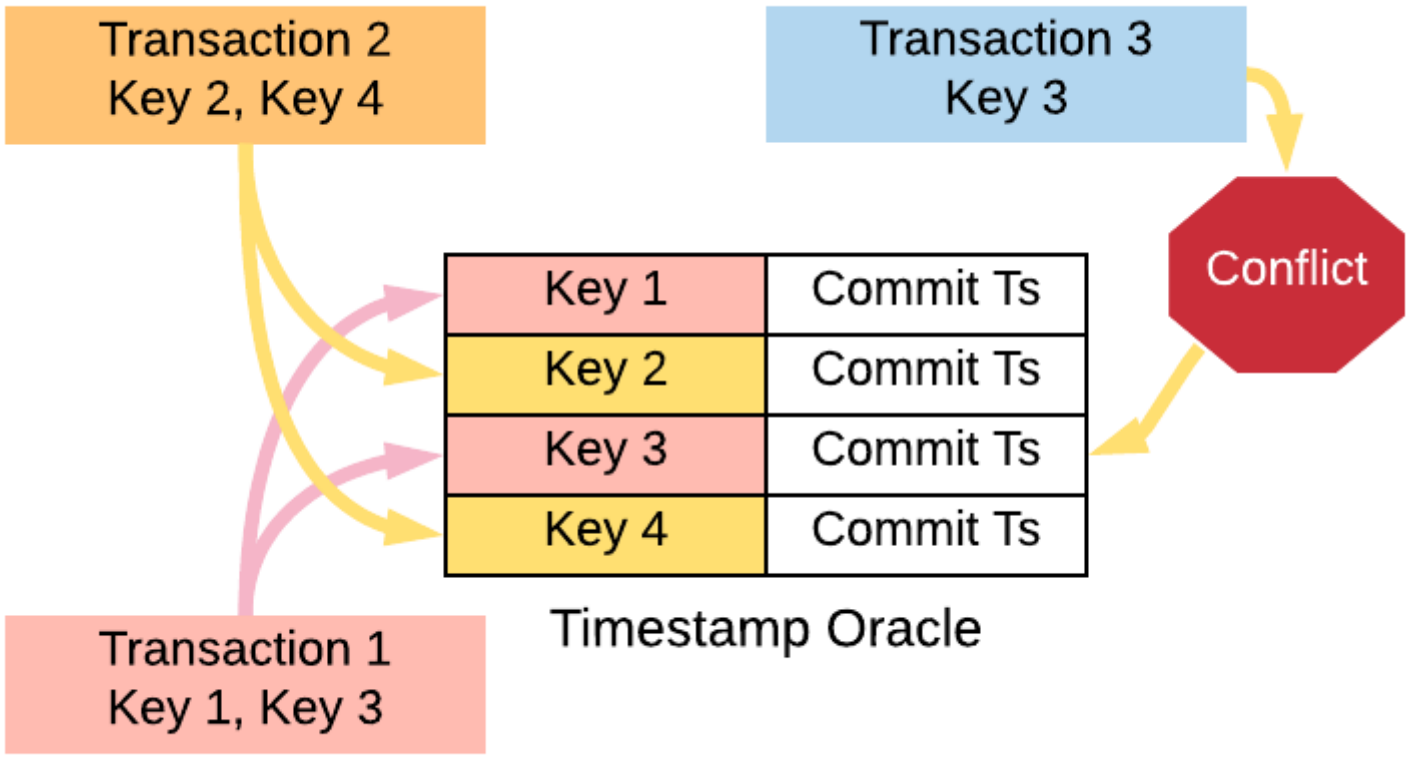
- Range iteration latency, etc.
- Can be found on https://blog.dgraph.io/
- Benchmarking code is open sourced.

github.com/dgraph-io/badger-bench (https://github.com/dgraph-io/badger-bench)

Features

Concurrent Transactions

- Badger uses Oracle to achieve concurrent lock-free transactions.



Concurrent Writes

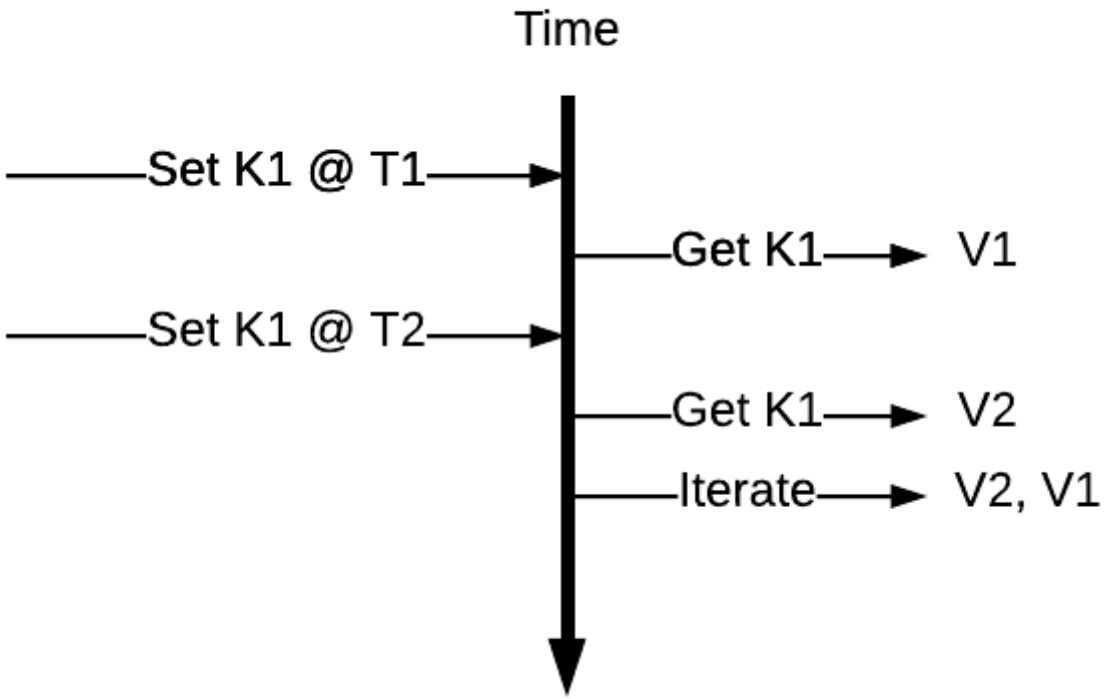
- Batch up writes from multiple transactions.
- Amortize cost of disk write.
- No wait -> Smart Batching.

Smart Batching in Go

```
reqs := make([]*request, 0, 10)
for {
    var r *request
    select {
    case r = <-db.writeCh:
    case <-lc.HasBeenClosed():
        goto closedCase
    }

    for {
        reqs = append(reqs, r)
        reqLen.Set(int64(len(reqs)))

        select {
        case r = <-db.writeCh:
        case pendingCh <- struct{}{}:
            goto writeCase
        case <-lc.HasBeenClosed():
            goto closedCase
        }
    }
```

```
closedCase:
    close(db.writeCh)
    for r := range db.writeCh {
        reqs = append(reqs, r)
    }

    pendingCh <- struct{}{}
    writeRequests(reqs)
    return

writeCase:
    go writeRequests(reqs)
    reqs = make([]*request, 0, 10)
    reqLen.Set(0)
}
```

Multi Version Concurrency Control



- Badger stores multiple versions of the key.
- Provides direct access to the versions, via iterate.

Crash Resilience

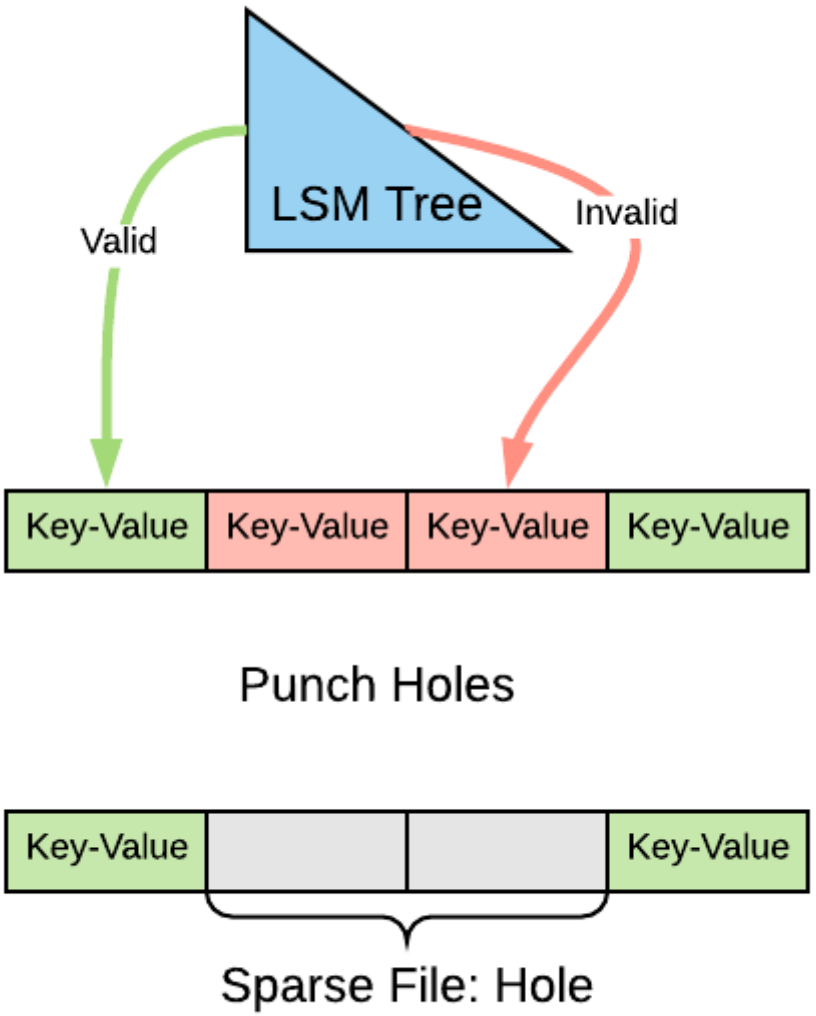
- LSM Memtables can be lost to crashes.
- Can be recovered from value log on restart.

Value Log Garbage Collection

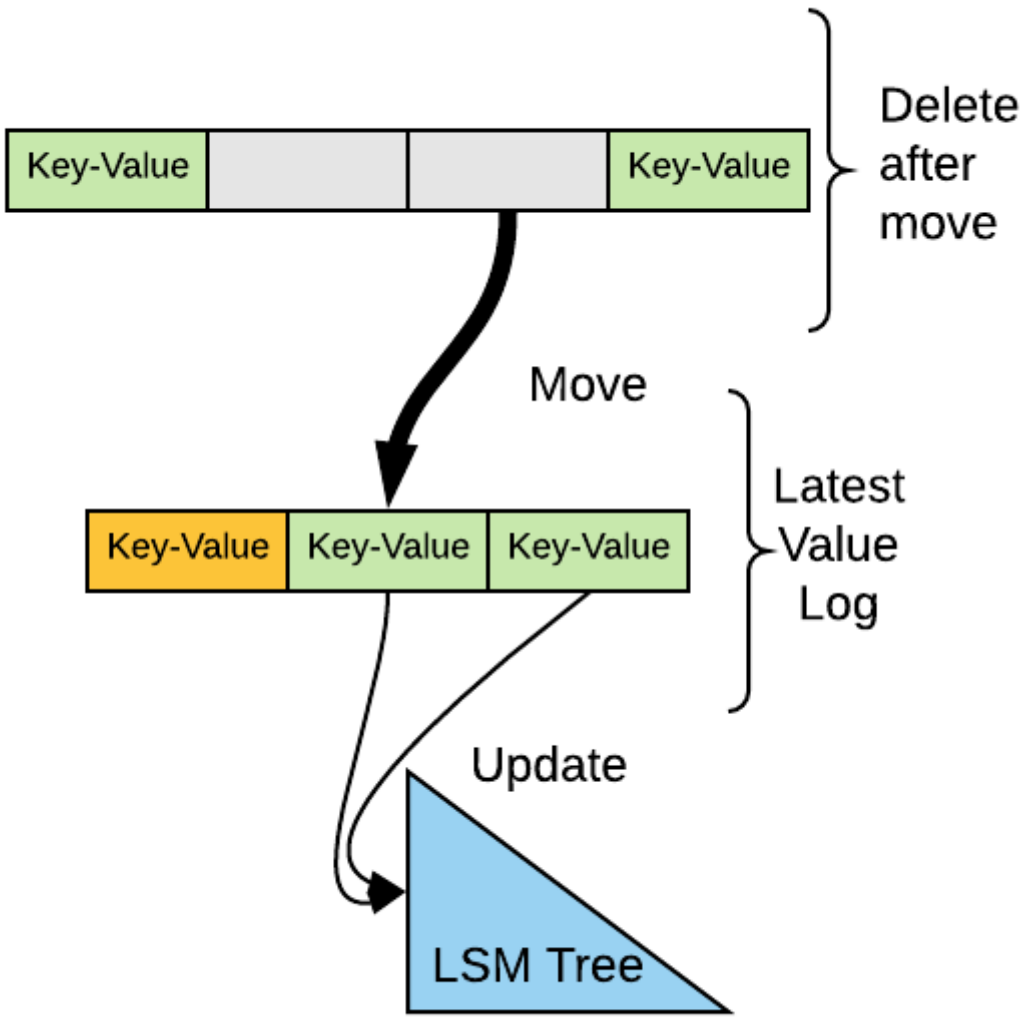
Why?

- Value log would keep growing with every Set.
- Older versions of keys can be deleted.
- Corresponding values can be deleted from value log.

Stage 1: Punch Holes (v2.0 in Linux)



Stage 2: Move entries, Delete log



Dealing with Qu-err-key file systems.

Would a file delete reclaim space in the file system?

Delete, no reclaim

- No

```
if err := t.fd.Truncate(0); err != nil {
    // This is very important to let the FS know
    // that the file is deleted.
    return err
}
```

- Truncate the file before deleting.

Would closing a file sync its contents to disk?

Close, no-sync

- No

```
if err := lf.fd.Sync(); err != nil {
    return errors.Wrapf(err, "Unable to sync value log: %q", lf.path)
}
if err := lf.fd.Close(); err != nil {
    return errors.Wrapf(err, "Unable to close value log: %q", lf.path)
}
```

- Explicitly sync file before closing.

Can a new synced file be lost?

Create, no-found

- Yes

```
f, err := os.Open(dir)
if err != nil {
    return errors.Wrapf(err, "While opening directory: %s.", dir)
}
err = f.Sync()
closeErr := f.Close()
if err != nil {
    return errors.Wrapf(err, "While syncing directory: %s.", dir)
}
return errors.Wrapf(closeErr, "While closing directory: %s.", dir)
```

- Sync a directory just like you would sync a file.

Can a crash add garbage data to end of file?

Crash, no-clean

- Yes.
- Add checksums to know when to truncate a file.

Who should use Badger?

Don't use Badger if...

- You no Go! (C++, Java)
- You have a single-threaded sequential workload.
- You have a small, read-only workload.
- All your data can fit in memory easily.

Use Badger if...

- You Go!
- You want to avoid Cgo.
- You want a performant read-write workload.
- You access data concurrently (many goroutines accessing data).
- You need 3-dimensional access.

Future Work

- Encryption at rest.
- Others? (tell us what you need)

Work on Badger and Dgraph. Come join us!

github.com/dgraph-io/badger (https://github.com/dgraph-io/badger)

github.com/dgraph-io/dgraph (https://github.com/dgraph-io/dgraph)

Careers at Dgraph (https://dgraph.io/about.html)

Talk to us on Wechat

Thank you

Manish R Jain, Dgraph Labs
Apr 14, 2018
Gopher China, Shanghai
manish@dgraph.io (mailto:manish@dgraph.io)
@manishrjain (http://twitter.com/manishrjain)