Writing a High Performance Database in Go

Two Meanings of "Database"

Database Server

Database Library

You may never write a database but...

HOW WE ACCESS DATA AFFECTS US ALL!

Why write a database in Go?

Things that need to be pretty fast

Things that need to be *really* f*cking fast

Things that need to be pretty fast

User Management

Schema Management

Query Parsing

Backup / Recovery

Bulk Data Insertion

etc...

Things that need to be *really* f*cking fast

Things that need to be pretty fast

User Management

Schema Management

Query Parsing

Backup / Recovery

Bulk Data Insertion

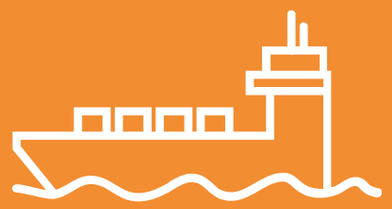
etc...

Things that need to be *really* f*cking fast

Query Execution

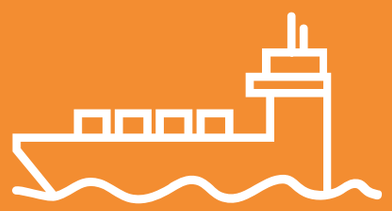
There’s more to databases than speed

There’s more to databases than speed



Easy Deployment

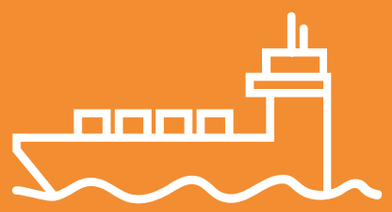
There’s more to databases than speed



Easy Deployment

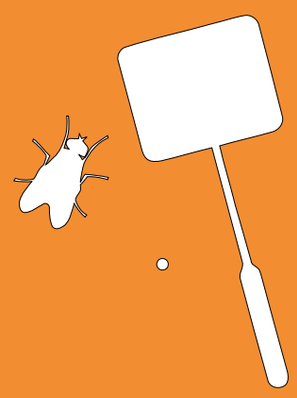
User friendly API

There’s more to databases than speed



Easy Deployment

User friendly API



Simple debugging

How do you make the fast parts fast?

Option #1: CGO

Pro: Integrate with tons of existing libraries

Con: Overhead incurred with each C function call

LuaJIT

Easy to integrate, good community

Half the speed of C, weird caveats

LLVM

Really, really fast

Really, really complicated

The point isn’t to just use C

The point is that C is an option

Option #2: Pure Go

Bolt

Basics of Bolt

Pure Go port of LMDB

Memory-mapped B+tree

MVCC, ACID transactions

Zero copy reads

Batch Work Together

Bolt Batch Benchmarks

Batch Size	Performance
1	Baseline
10	**9x** Baseline
100	**45x** Baseline
1000	**90x** Baseline

Disclaimer: YMMV

Transaction Coalescing

Use a channel to stream changes

Group changes into single transaction

Either all changes commit or rollback

Encoding Matters!

Encoding Performance

JSON	Baseline
gogoprotobuf	**20x** JSON
Cap’n Proto	**60x** JSON

Disclaimer: YMMV

Encoding Performance

See also: Albert Strasheim's "Serialization in Go" Talk

http://www.slideshare.net/albertstrasheim/serialization-in-go

https://github.com/cloudflare/goser

Here’s a crazy idea...

Direct map to your data file

Map a struct to a []byte

```
// Create a byte slice with the same size as type T.
var value = make([]byte, unsafe.Sizeof(T{})

// Map a typed pointer from the byte slice and update it.
var t = (*T)unsafe.Pointer(&value[0])
t.ID = 123
t.MyIntValue = 20

// Insert value into database.
db.Update(func(tx *bolt.Tx) error {
    return tx.Bucket(“T”).Put([]byte(“123”), value)
})
```

Map a []byte to a struct

```
// Start a read transaction.
db.View(func(tx *bolt.Tx) error {
    c := tx.Bucket("T").Cursor()

    // Iterate over each value in the bucket.
    for k, v := c.First(); k != nil; k, v = c.Next() {
        var t = (*T)unsafe.Pointer(&value[0])

        // ... do something with "t" ...
    }

    return nil
})
```

Pros:

No encoding/decoding

Insert 100k values/sec

Read 20M values/sec

Cons:

Fixed struct layout

Machine specific endianness

People will think you’re crazy

Your CPU can do **3 billion** operations per second so USE IT!

How to think about performance optimization

Hierarchy of Need

Self-actualization

Esteem

Love/Belonging

Safety

Physiological

Hierarchy of Need

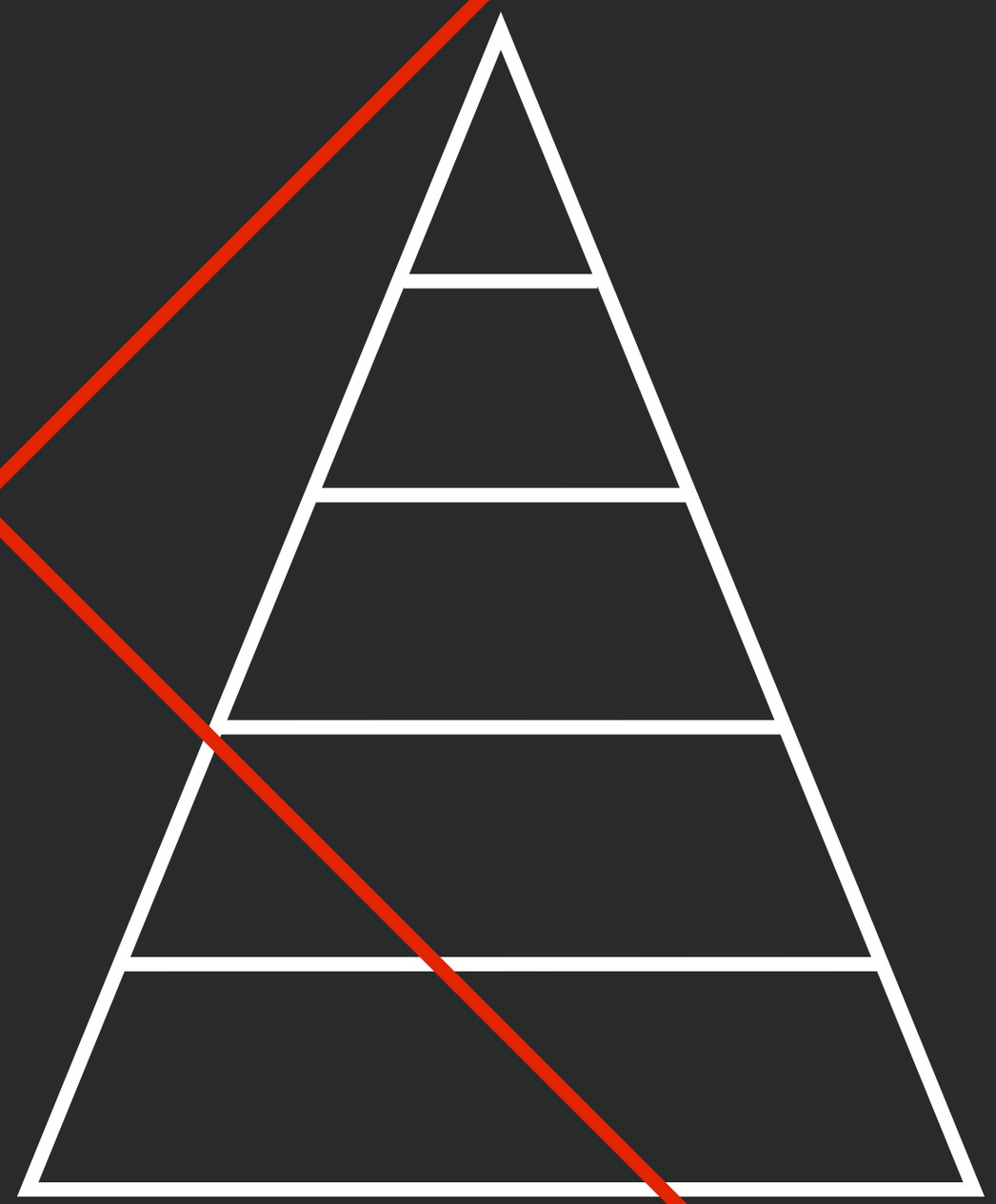
Self-actualization

Esteem

Love/Belonging

Safety

Physiological



Hierarchy of **SPEED**

Memory Access

Mutexes

Memory Allocation

Disk I/O

Network I/O

Go can be extremely fast... if you know how to optimize it!

Questions

@benbjohnson