Bitmap Indexes

Marko Kevac
Gophercon Russia 2019

http://bit.ly/bitmapindexes
https://github.com/mkevac/gopherconrussia2019

Contents

1. What indexes are.
2. What is a bitmap index.
3. Where it’s used. Why it’s not.

BORING THEORY

4. Simple implementation in Go. Fight with compiler.
5. Simple implementation in Go assembly.

FUN!

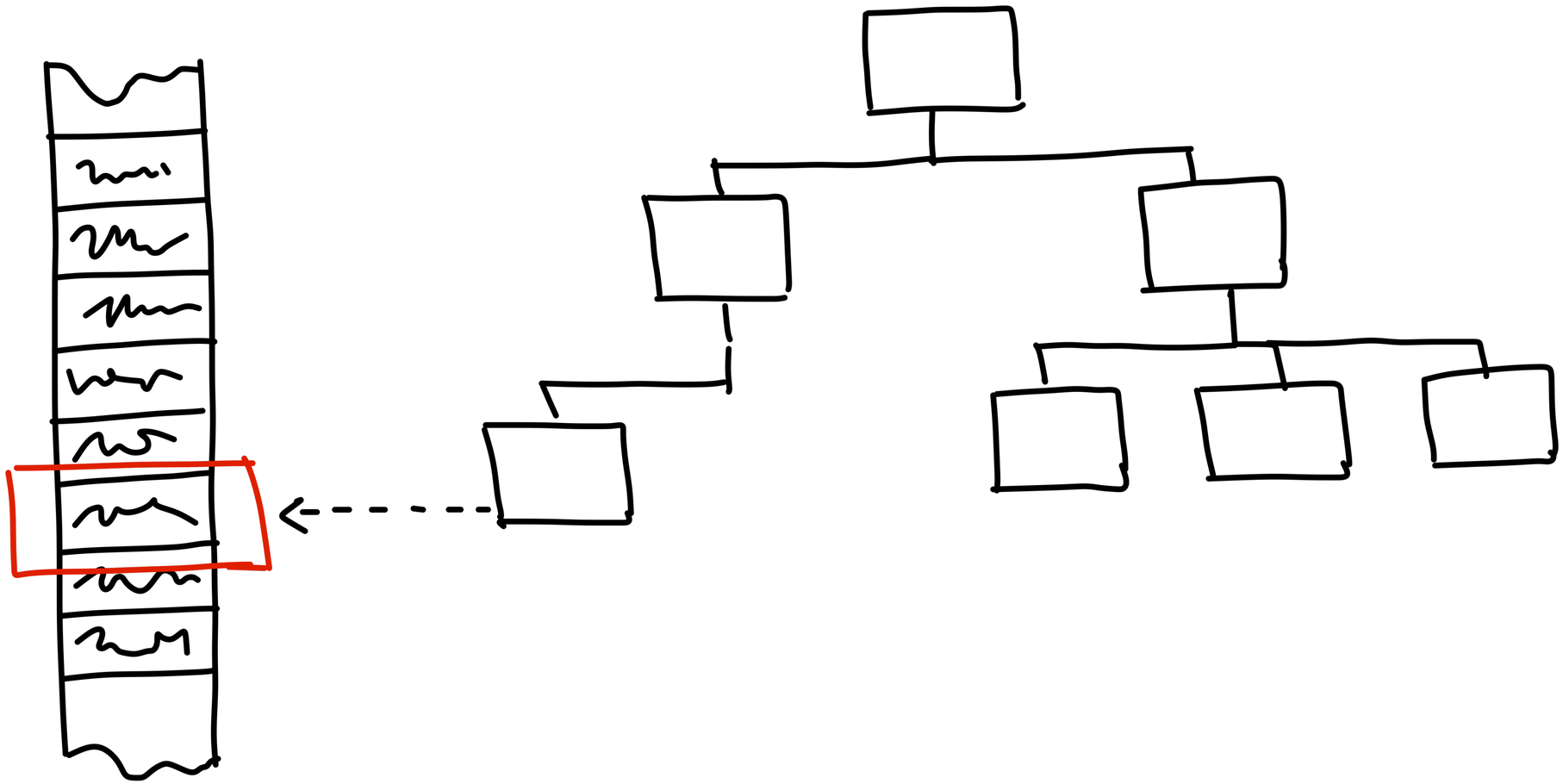
6. Addressing the “problems”.

Also fun!

MEH…

7. Existing solutions.

Indexes



Indexing approaches

Hierarchical division

*-trees
B-trees
R-trees

O(logN)

Indexing approaches

Hash mapping

Hash maps
Reverse indexes

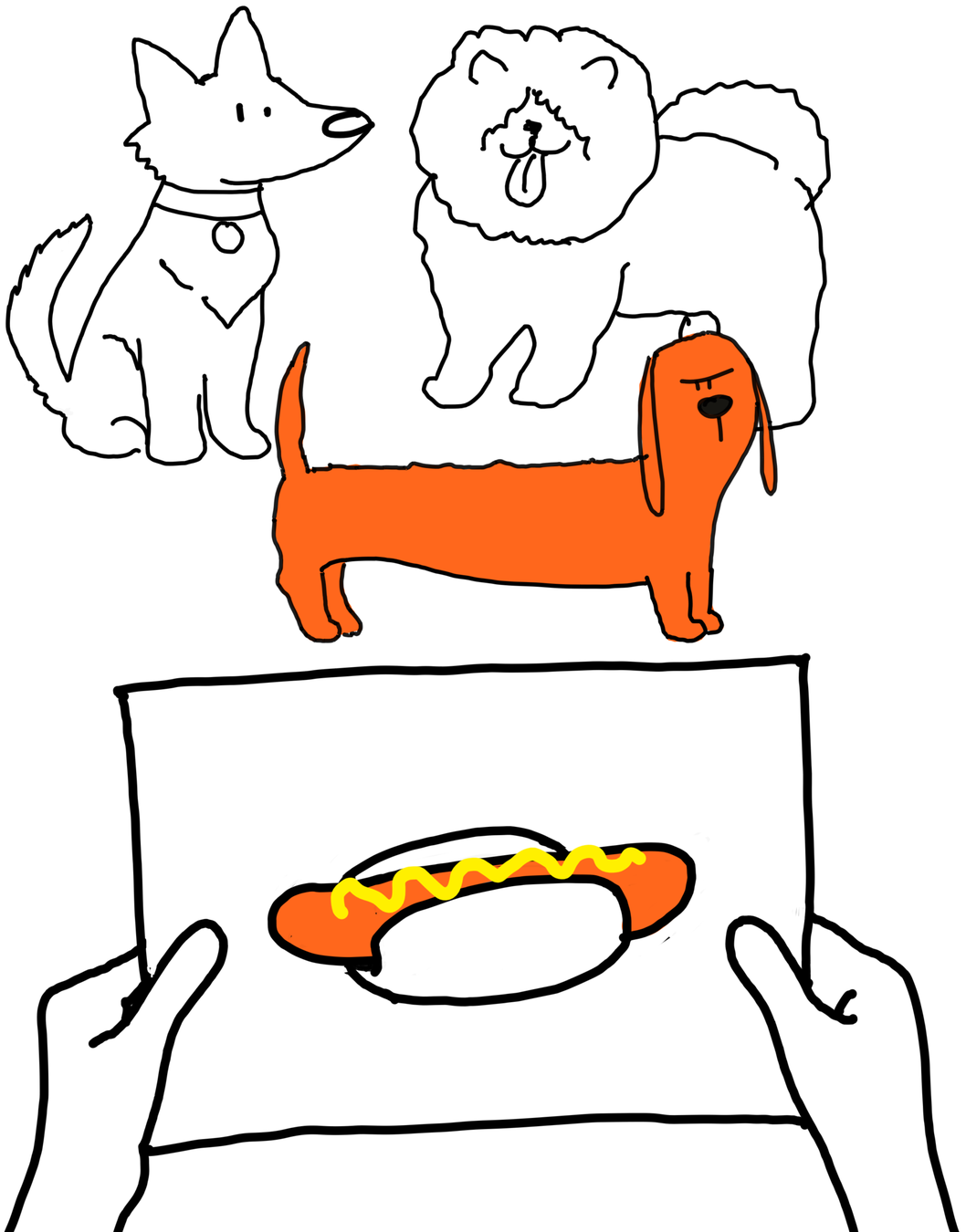
O(1)

Indexing approaches

Instantly knowing

Bloom filters
Cuckoo filters

O(1)



Indexing approaches

Fully using hardware capabilities

Bitmap indexes

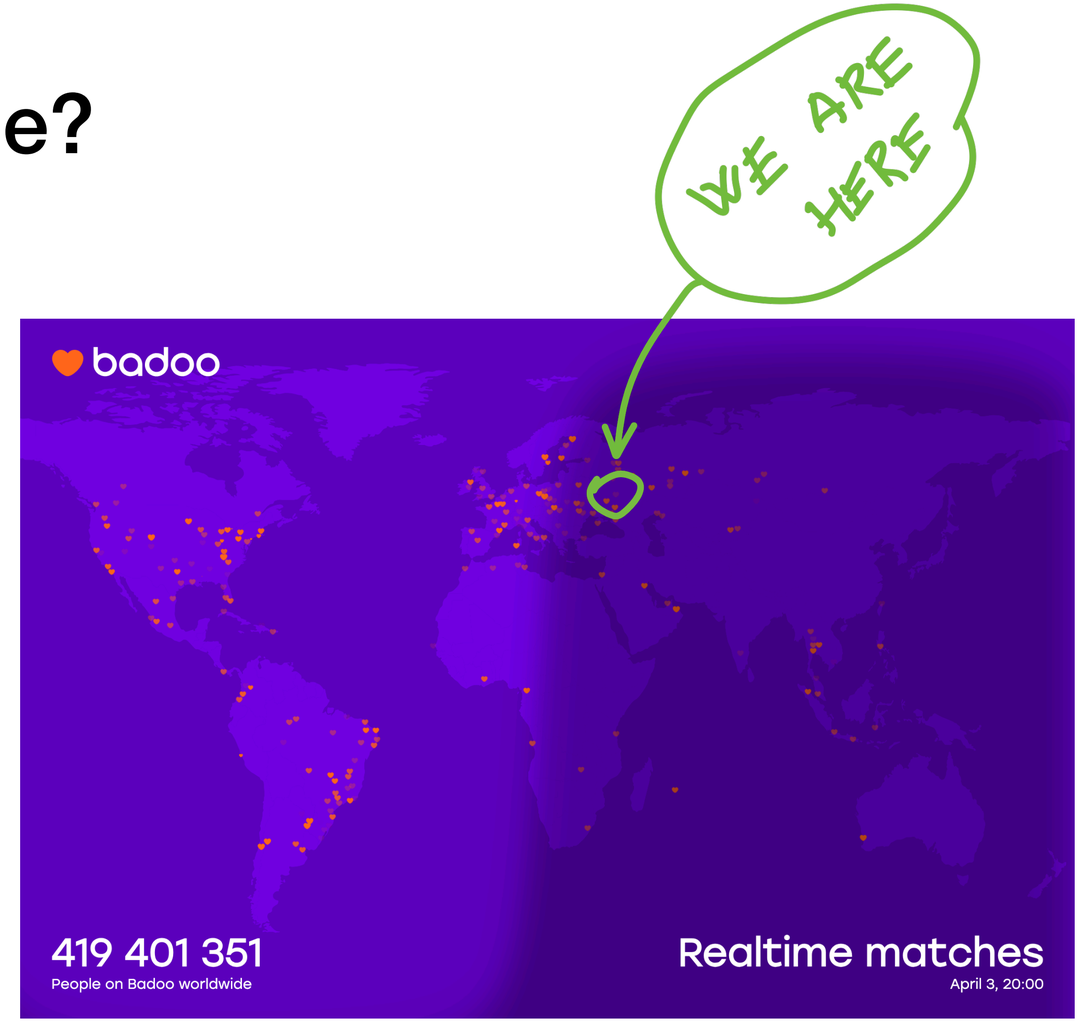
O(N)

Why me?

Marko Kevac
marko@kevac.org

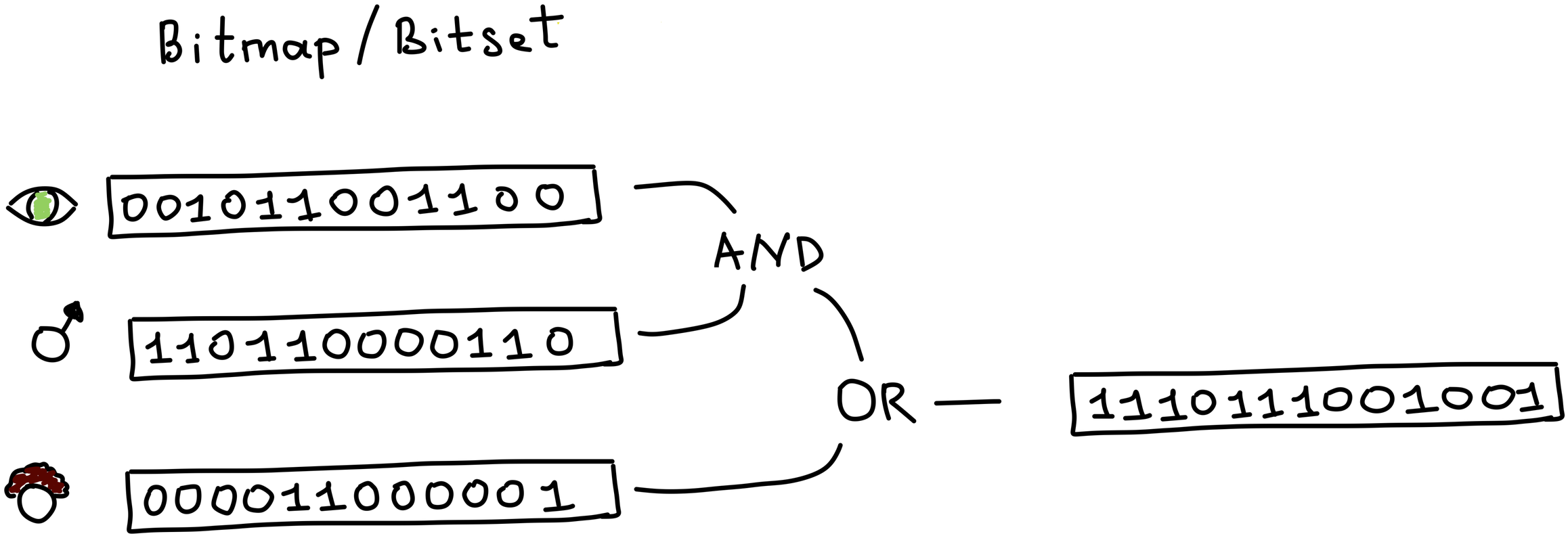
Badoo/Bumble

> 400 000 000 users worldwide



Searching for the best match for you using bitmap indexes since ~ 2010.

What is a Bitmap Index?



Believed to be best for

Searches which combine several queries for different columns that have small cardinality.

“Give me results that have this
and this or this
and definitely not those…”

Small cardinality (few distinct values)

- Eye color
- Gender
- Marital status

Large cardinality

- Distance to the city center
- Number of likes received

Simplest example

Moscow restaurants

- Near Metro
- Has private parking
- Has terrace
- Accepts reservations
- Vegan friendly
- Expensive

0 . Pushkin

1. Turandot

2. Coffee Lovers

3. Yunost

4. Taras Bulba

…

Populating bitmaps

	0	1	2	3	4
near metro	1	0	0	1	1
PRIVATE PARK	1	0	0	0	0
TERRACE	0	1	0	0	1
RESERVATIONS	1	1	0	1	1
VEGAN	0	0	1	1	0
EXPENSIVE	1	1	0	0	0

Give me restaurants that are vegan friendly.

Give me restaurants that have terrace,
accept reservations, but are not expensive.

Give me restaurants that are vegan friendly.

Give me restaurants that have terrace,
accept reservations, but are not expensive.

Vegan friendly

	0	1	2	3	4
near metro	1	0	0	1	1
PRIVATE PARK	1	0	0	0	0
TERRACE	0	1	0	0	1
RESERVATIONS	1	1	0	1	1
VEGAN	0	0	1	1	0
EXPENSIVE	1	1	0	0	0

Give me restaurants that are vegan friendly.

Give me restaurants that have terrace,
accept reservations, but are not expensive.

More complex query

	0	1	2	3	4
near metro	1	0	0	1	1
PRIVATE PARK	1	0	0	0	0
TERRACE	0	1	0	0	1
RESERVATIONS	1	1	0	1	1
VEGAN	0	0	1	1	0
EXPENSIVE	1	1	0	0	0

Terrace AND Reservations

AND

NOT Expensive

More complex query

	0	1	2	3	4
near metro	1	0	0	1	1
PRIVATE PARK	1	0	0	0	0
TERRACE	0	1	0	0	1
RESERVATIONS	1	1	0	1	1
VEGAN	0	0	1	1	0
NOT EXPENS	0	0	1	1	1

Terrace AND Reservations

AND

NOT Expensive

More complex query

	0	1	2	3	4
near metro	1	0	0	1	1
PRIVATE PARK	1	0	0	0	0
TERRACE	0	1	0	0	1
RESERVATIONS	1	1	0	1	1
VEGAN	0	0	1	1	0
NOT EXPENS	0	0	1	1	1

Terrace AND Reservations

AND

NOT Expensive

Bitmap index usage

Oracle DB 

MySQL

PostgreSQL

Tarantool

Redis

MongoDB

ElasticSearch

Bitmap index usage

Oracle DB		Old fashioned
MySQL		Proposal https://dev.mysql.com/worklog/task/?id=1524
PostgreSQL		Internal
Tarantool		Simple
Redis		https://redis.io/commands/bitfield
MongoDB		Proposal https://jira.mongodb.org/browse/SERVER-1723
ElasticSearch		Internal

https://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems#Indexes

Bitmap index usage

Oracle DB	✓	Old fashioned
MySQL	–	Proposal
PostgreSQL	–	Internal
Tarantool	~	Simple
Redis	–	
MongoDB	–	Proposal
ElasticSearch	–	Internal
Pilosa	✓✓✓	Amazing

Implementation in Go

```
const restaurants = 65536
const bitmapLength = restaurants / 8 // 8192 bytes

var (
  nearMetro       = make([]byte, bitmapLength)
  privateParking  = make([]byte, bitmapLength)
  terrace         = make([]byte, bitmapLength)
  reservations    = make([]byte, bitmapLength)
  veganFriendly   = make([]byte, bitmapLength)
  expensive       = make([]byte, bitmapLength)
)
```

http://bit.ly/bitmapindexes

```
func fill(r *rand.Rand, b []byte, probability float32)

fill(r, nearMetro, 0.1)
fill(r, privateParking, 0.01)
fill(r, terrace, 0.05)
fill(r, reservations, 0.95)
fill(r, veganFriendly, 0.2)
fill(r, expensive, 0.1)

func indexes(a []byte) []int
```

http://bit.ly/bitmapindexes

Give me restaurants with a terrace that accept reservations, but are not expensive.

terrace AND reservations AND (NOT expensive)

http://bit.ly/bitmapindexes

Give me restaurants with a terrace that accept reservations, but are not expensive.

~~**terrace AND reservations AND (NOT expensive)**~~

terrace AND reservations ANDNOT expensive

```
func and(a []byte, b []byte, res []byte) {
 for i := 0; i < len(a); i++ {
  res[i] = a[i] & b[i]
 }
}

func andnot(a []byte, b []byte, res []byte) {
 for i := 0; i < len(a); i++ {
  res[i] = a[i] & ^b[i]
 }
}
```

http://bit.ly/bitmapindexes

```
func BenchmarkSimpleBitmapIndex(b *testing.B) {
  _, _, terrace, reservations, _, expensive := initData()

  resBitmap := make([]byte, bitmapLength)

  b.ResetTimer()

  for i := 0; i < b.N; i++ {
    andnot(terrace, expensive, resBitmap)
    and(reservations, resBitmap, resBitmap)
  }
}
```

```
name
SimpleBitmapIndex-12
```

```
time/op
10.8µs ± 0%
```

http://bit.ly/bitmapindexes

```
$ go test -run=None -bench=BenchmarkSimpleBitmapIndex$ -cpuprofile=cpu.out

$ pprof -http=:8080 ./simple.test ./cpu.out
```

```
         .          .                 for i := 0; i < b.N; i++ {
         .      680ms                         andnot(terrace, expensive, resBitmap)
         .          .  10fd7da:                       MOVQ 0x20c8(SP), AX                 simple_test.go:55
         .          .  10fd7e2:                       MOVQ AX, 0(SP)                      simple_test.go:55
         .          .  10fd7e6:                       MOVQ 0x90(SP), CX                   simple_test.go:55
         .          .  10fd7ee:                       MOVQ CX, 0x8(SP)                    simple_test.go:55
         .          .  10fd7f3:                       MOVQ 0x98(SP), DX                   simple_test.go:55
         .          .  10fd7fb:                       MOVQ DX, 0x10(SP)                   simple_test.go:55
         .          .  10fd800:                       MOVQ 0x20d8(SP), BX                 simple_test.go:55
         .          .  10fd808:                       MOVQ BX, 0x18(SP)                   simple_test.go:55
         .          .  10fd80d:                       MOVQ 0xb8(SP), SI                   simple_test.go:55
         .          .  10fd815:                       MOVQ SI, 0x20(SP)                   simple_test.go:55
         .          .  10fd81a:                       MOVQ 0xc0(SP), DI                   simple_test.go:55
         .          .  10fd822:                       MOVQ DI, 0x28(SP)                   simple_test.go:55
         .          .  10fd827:                       LEAQ 0xc8(SP), R8                   simple_test.go:55
         .          .  10fd82f:                       MOVQ R8, 0x30(SP)                   simple_test.go:55
         .          .  10fd834:                       MOVQ $0x2000, 0x38(SP)              simple_test.go:55
         .          .  10fd83d:                       MOVQ $0x2000, 0x40(SP)              simple_test.go:55
         .      680ms  10fd846:                       CALL github.com/mkevac/gopherconrussia2019/simple.andnot(SB) simple_test.go:55

         .      470ms                         and(reservations, resBitmap, resBitmap)
         .          .                 }
```

http://bit.ly/bitmapindexes

```
$ go build -gcflags="-m -m"
# github.com/mkevac/gopherconrussia2019/simple
./simple.go:27:6: cannot inline and: unhandled op FOR
./simple.go:71:6: cannot inline andnot: unhandled op FOR
```

cmd/compile: for-loops cannot be inlined #14768

https://github.com/golang/go/issues/14768

```
func andInlined(a []byte, b []byte, res []byte) {
  i := 0

loop:
  if i < len(a) {
    res[i] = a[i] & b[i]
    i++
    goto loop
  }
}
```

```
$ go build -gcflags="-m -m" 2>&1 | grep andInlined

./simple.go:33:6: can inline andInlined as: func([]byte, []byte,
[]byte) { i := 0; loop: ; if i < len(a) { res[i] = a[i] & b[i]; i++;
goto loop } }
```

```
name
SimpleBitmapIndex-12
SimpleBitmapIndexInlined-12
```

```
time/op
10.8µs ± 0%
8.88µs ± 0%
```

~ 2 µs

http://bit.ly/bitmapindexes

Bounds check elimination

```
70            .          .
71            .       20ms           func andnot(a []byte, b []byte, res []byte) {
72         130ms      130ms                   for i := 0; i < len(a); i++ {
73         450ms      450ms                           res[i] = a[i] & ^b[i]
            90ms       90ms  10fcf9f:                         MOVB R10, 0(BX)(AX*1)                  simple.go:73
                             ⋮
            10ms       10ms  10fcfab:                         MOVZX 0(R9)(AX*1), R10                 simple.go:73
            70ms       70ms  10fcfb0:                         CMPQ DI, AX                            simple.go:73
               .          .  10fcfb3:                         JAE 0x10fcfdb                          simple.go:73
           120ms      120ms  10fcfb5:                         MOVZX 0(R8)(AX*1), R11                 simple.go:73
            30ms       30ms  10fcfba:                         NOTL R11                               simple.go:73
            30ms       30ms  10fcfbd:                         ANDL R11, R10                          simple.go:73
           100ms      100ms  10fcfc0:                         CMPQ SI, AX                            simple.go:73
               .          .  10fcfc3:                         JB 0x10fcf9f                           simple.go:73
               .          .  10fcfc5:                         JMP 0x10fcfd1                          simple.go:73
               .          .  10fcfc7:                         MOVQ 0x10(SP), BP                      simple.go:73
               .          .  10fcfcc:                         ADDQ $0x18, SP                         simple.go:73
               .          .  10fcfd0:                         RET                                    simple.go:73
               .          .  10fcfd1:                         MOVQ SI, CX                            simple.go:73
               .          .  10fcfd4:                         CALL runtime.panicIndex(SB)            simple.go:73
               .          .  10fcfd9:                         UD2                                    simple.go:73
               .          .  10fcfdb:                         MOVQ DI, CX                            simple.go:73
               .          .  10fcfde:                         CALL runtime.panicIndex(SB)            simple.go:73
               .          .  10fcfe3:                         UD2                                    simple.go:73

74            .          .                   }
75            .          .           }
```

http://bit.ly/bitmapindexes

```go
func andNoBoundsCheck(a []byte, b []byte, res []byte) {
  if len(a) != len(b) || len(b) != len(res) {
    return
  }

  for i := 0; i < len(a); i++ {
    res[i] = a[i] & b[i]
  }
}
```

```
name                                        time/op
SimpleBitmapIndex-12                        10.8µs ± 0%
SimpleBitmapIndexInlined-12                 8.88µs ± 0%
SimpleBitmapIndexInlinedAndNoBoundsCheck-12 8.33µs ± 0%
```

~ 500ns

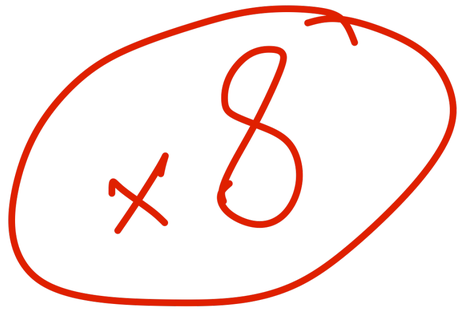
http://bit.ly/bitmapindexes

Bigger batches

```
const restaurants = 65536
const bitmapLength = restaurants / (8 * 8) // 8192 bytes (1024 elements)

nearMetro      = make([]uint64, bitmapLength)

func fill(r *rand.Rand, b []uint64, probability float32)

func indexes(a []uint64) []int

func andNoBoundsCheck(a []uint64, b []uint64, res []uint64) {
  if len(a) != len(b) || len(b) != len(res) {
    return
  }

  for i := 0; i < len(a); i++ {
    res[i] = a[i] & b[i]
  }
}
```

http://bit.ly/bitmapindexes

```
name                                                time/op
SimpleBitmapIndex-12                                10.8µs ± 0%
SimpleBitmapIndexInlined-12                         8.88µs ± 0%
SimpleBitmapIndexInlinedAndNoBoundsCheck-12         8.33µs ± 0%

name                                                  time/op
BiggerBatchBitmapIndexNoBoundsCheck-12                1.06µs ± 0%
```



SIMD

Single Instruction Multiple Data

16-byte chunks (SSE)
32-byte chunks (AVX, AVX2)
64-byte chunks (AVX512)

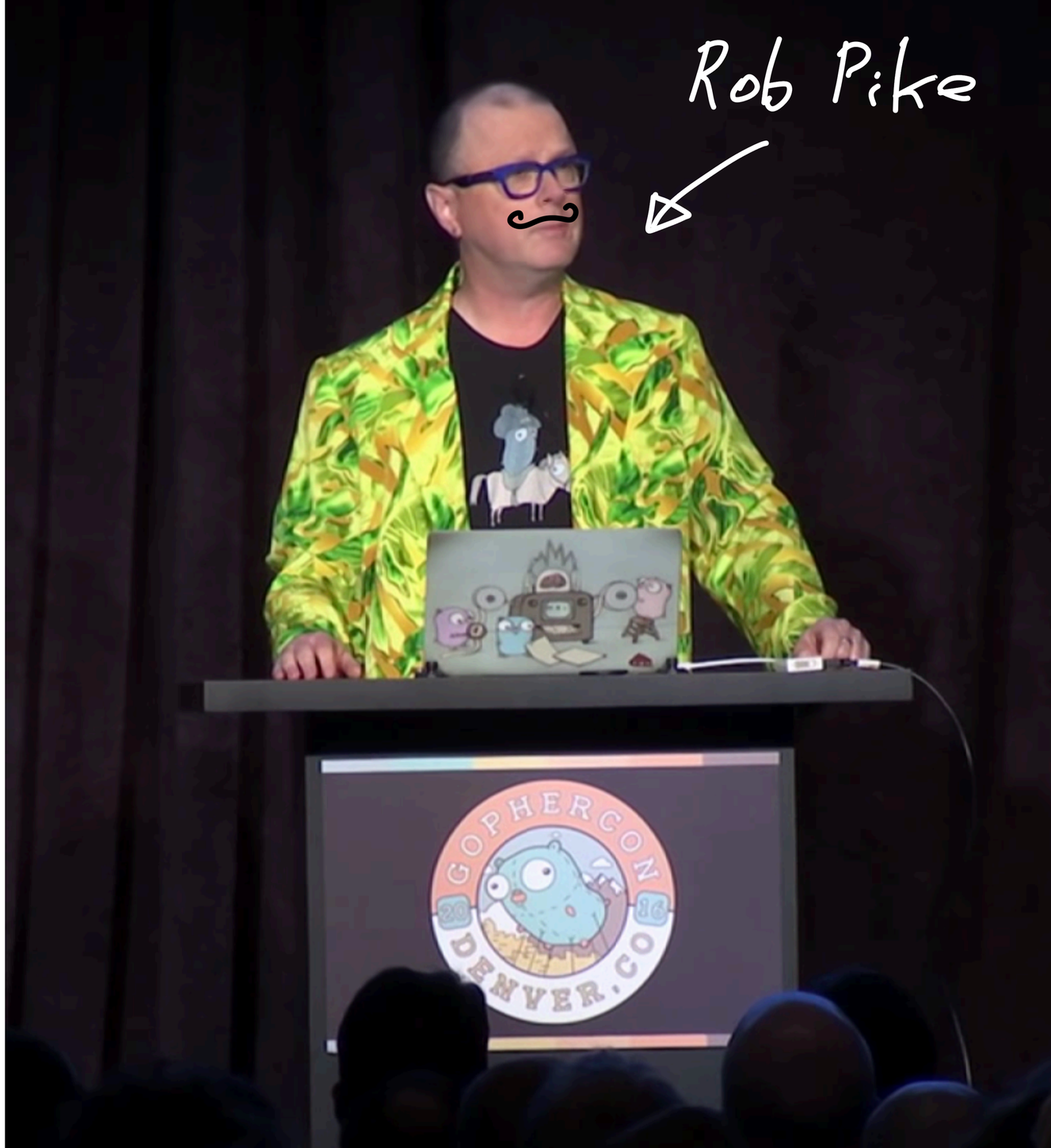
Vectorization

Go assembly

Not the real assembly, more like IRL.
Platform independent.

https://www.youtube.com/watch?v=KINIAgRpkDA

https://habr.com/ru/company/badoo/blog/317864/ (rus)

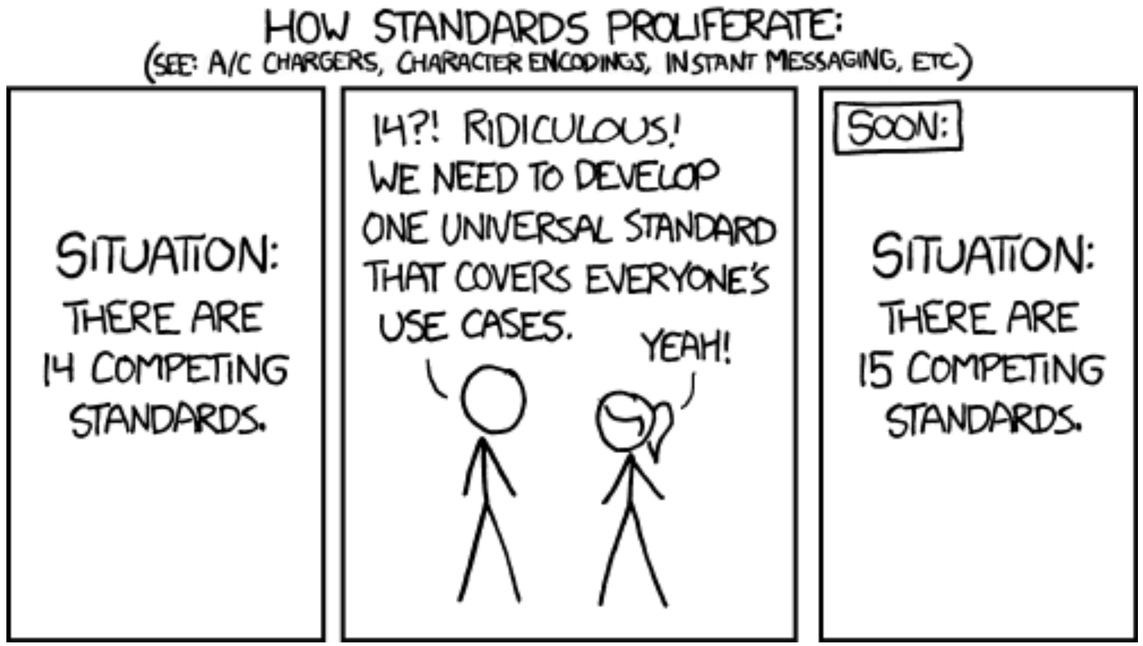


Go assembly

Not the real assembly, more like IRL.
Platform independent.

~~AT&T~~ ~~Intel~~ Plan9 !

No fun :-(



https://xkcd.com/927/

https://github.com/Maratyszcza/PeachPy

avo

https://github.com/mmcloughlin/avo

Avo example

```
// +build ignore

package main

func main() {
  TEXT("Add", NOSPLIT, "func(x, y uint64) uint64")
  Doc("Add adds x and y.")
  x := Load(Param("x"), GP64())
  y := Load(Param("y"), GP64())
  ADDQ(x, y)
  Store(y, ReturnIndex(0))
  RET()
  Generate()
}
```

```
//go:generate go run asm.go -out add.s -stubs stub.go
```

Avo example

```
// +build ignore

package main

func main() {
  TEXT("Add", NOSPLIT, "func(x, y uint64) uint64")
  Doc("Add adds x and y.")
  x := Load(Param("x"), GP64())
  y := Load(Param("y"), GP64())
  ADDQ(x, y)
  Store(y, ReturnIndex(0))
  RET()
  Generate()
}
```

```
//go:generate go run asm.go -out add.s -stubs stub.go
```

add.s

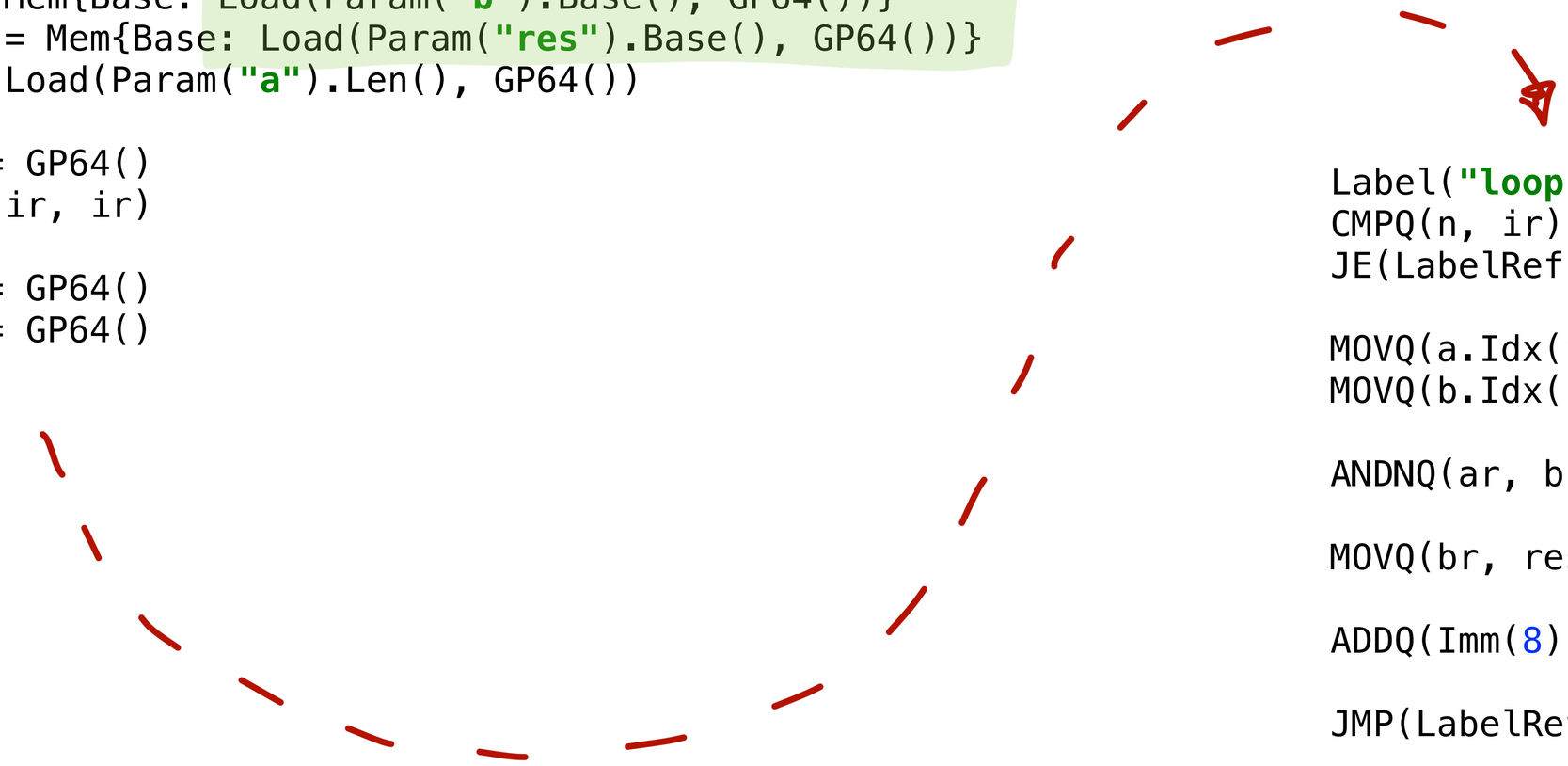
```
// Code generated by command: go run asm.go -out add.s -stubs stub.go. DO NOT EDIT.

#include "textflag.h"

// func Add(x uint64, y uint64) uint64
TEXT ·Add(SB), NOSPLIT, $0-24
   MOVQ x+0(FP), AX
   MOVQ y+8(FP), CX
   ADDQ AX, CX
   MOVQ CX, ret+16(FP)
   RET
```

stub.go

```
// Code generated by command: go run asm.go -out add.s -stubs stub.go. DO NOT EDIT.

package add

// Add adds x and y.
func Add(x uint64, y uint64) uint64
```

Scalar version

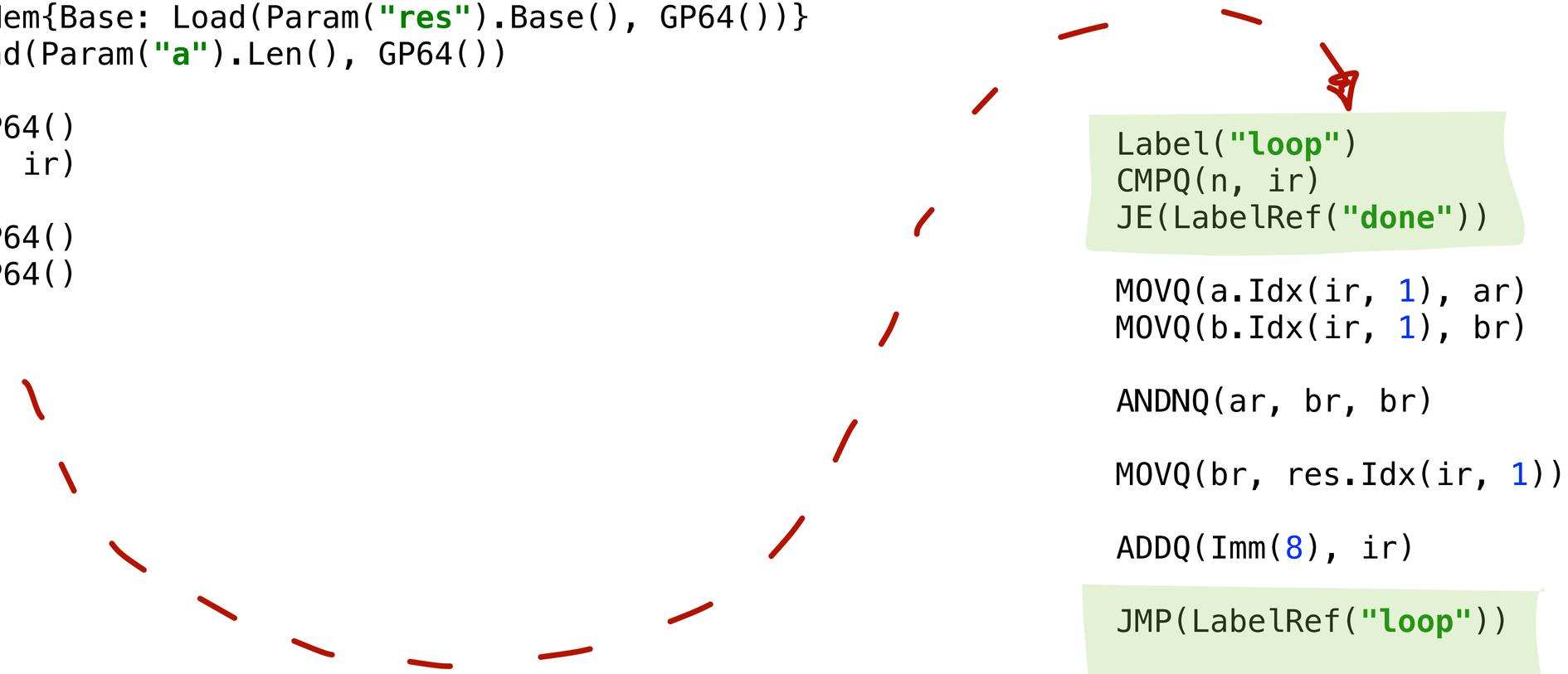
```
TEXT("andnotScalarFaster", NOSPLIT, "func(a []byte, b []byte, res []byte)")

a := Mem{Base: Load(Param("a").Base(), GP64())}
b := Mem{Base: Load(Param("b").Base(), GP64())}
res := Mem{Base: Load(Param("res").Base(), GP64())}
n := Load(Param("a").Len(), GP64())

ir := GP64()
XORQ(ir, ir)

ar := GP64()
br := GP64()
```



```
Label("loop")
CMPQ(n, ir)
JE(LabelRef("done"))

MOVQ(a.Idx(ir, 1), ar)
MOVQ(b.Idx(ir, 1), br)

ANDNQ(ar, br, br)

MOVQ(br, res.Idx(ir, 1))

ADDQ(Imm(8), ir)

JMP(LabelRef("loop"))

Label("done")
RET()
Generate()
```

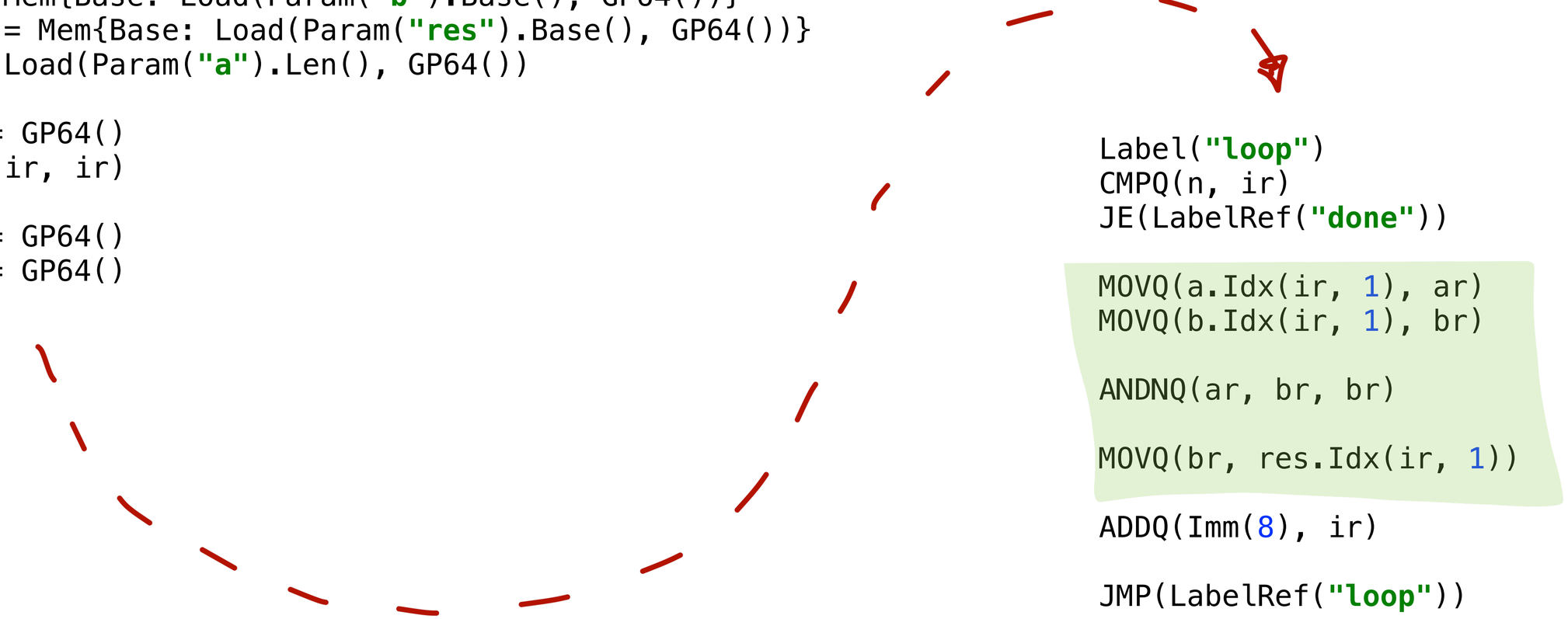
Scalar version

```
TEXT("andnotScalarFaster", NOSPLIT, "func(a []byte, b []byte, res []byte)")

a := Mem{Base: Load(Param("a").Base(), GP64())}
b := Mem{Base: Load(Param("b").Base(), GP64())}
res := Mem{Base: Load(Param("res").Base(), GP64())}
n := Load(Param("a").Len(), GP64())

ir := GP64()
XORQ(ir, ir)

ar := GP64()
br := GP64()

Label("loop")
CMPQ(n, ir)
JE(LabelRef("done"))

MOVQ(a.Idx(ir, 1), ar)
MOVQ(b.Idx(ir, 1), br)

ANDNQ(ar, br, br)

MOVQ(br, res.Idx(ir, 1))

ADDQ(Imm(8), ir)

JMP(LabelRef("loop"))

Label("done")
RET()
Generate()
```



Scalar version

```
TEXT("andnotScalarFaster", NOSPLIT, "func(a []byte, b []byte, res []byte)")

a := Mem{Base: Load(Param("a").Base(), GP64())}
b := Mem{Base: Load(Param("b").Base(), GP64())}
res := Mem{Base: Load(Param("res").Base(), GP64())}
n := Load(Param("a").Len(), GP64())

ir := GP64()
XORQ(ir, ir)

ar := GP64()
br := GP64()
```



```
Label("loop")
CMPQ(n, ir)
JE(LabelRef("done"))

MOVQ(a.Idx(ir, 1), ar)
MOVQ(b.Idx(ir, 1), br)

ANDNQ(ar, br, br)

MOVQ(br, res.Idx(ir, 1))

ADDQ(Imm(8), ir)

JMP(LabelRef("loop"))

Label("done")
RET()
Generate()
```

```
// func andnotScalarFaster(a []byte, b []byte, res []byte)
TEXT ·andnotScalarFaster(SB), NOSPLIT, $0-72
  MOVQ a_base+0(FP), AX
  MOVQ b_base+24(FP), CX
  MOVQ res_base+48(FP), DX
  MOVQ a_len+8(FP), BX
  XORQ BP, BP

loop:
  CMPQ  BX, BP
  JE    done
  MOVQ  (AX)(BP*1), SI
  MOVQ  (CX)(BP*1), DI
  ANDNQ SI, DI, DI
  MOVQ  DI, (DX)(BP*1)
  ADDQ  $0x08, BP
  JMP   loop

done:
  RET
```

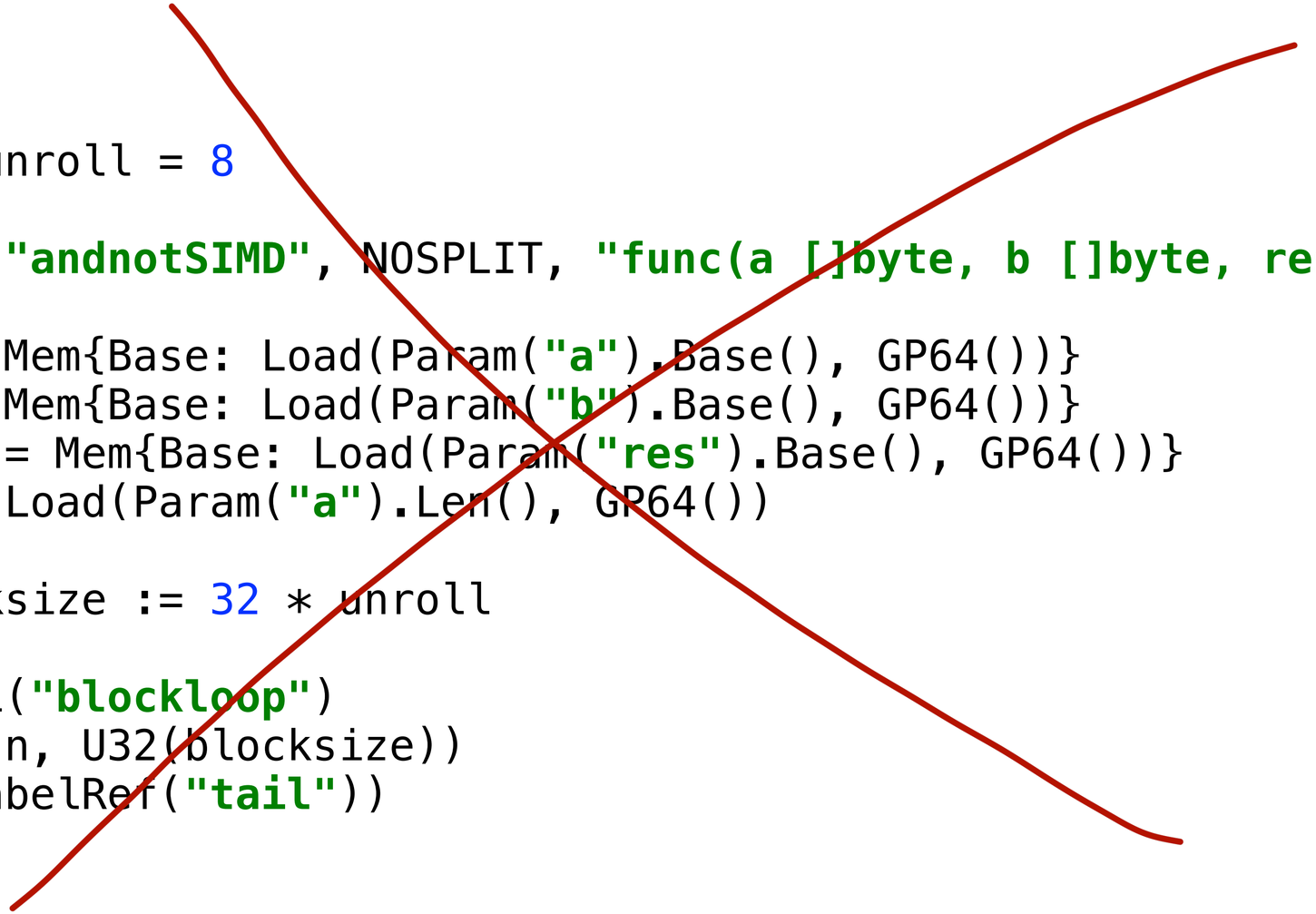
http://bit.ly/bitmapindexes

```
name                                          time/op
SimpleBitmapIndex-12                          10.8µs ± 0%
SimpleBitmapIndexInlined-12                   8.88µs ± 0%
SimpleBitmapIndexInlinedAndNoBoundsCheck-12   8.33µs ± 0%

name                                               time/op
BiggerBatchBitmapIndexNoBoundsCheck-12             1.06µs ± 0%

name                                    time/op
SimpleScalarFasterBitmapIndex-8         1.04µs ± 0%
```

Inlining assembly code is not possible: https://github.com/golang/go/issues/26891

Vector (SIMD) version

```
var unroll = 8

TEXT("andnotSIMD", NOSPLIT, "func(a []byte, b []byte, res []byte)")

a := Mem{Base: Load(Param("a").Base(), GP64())}
b := Mem{Base: Load(Param("b").Base(), GP64())}
res := Mem{Base: Load(Param("res").Base(), GP64())}
n := Load(Param("a").Len(), GP64())

blocksize := 32 * unroll

Label("blockloop")
CMPQ(n, U32(blocksize))
JL(LabelRef("tail"))
```

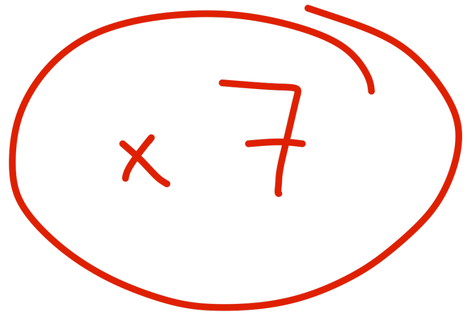


```
// Load b.
bs := make([]VecVirtual, unroll)
for i := 0; i < unroll; i++ {
  bs[i] = YMM()
}

for i := 0; i < unroll; i++ {
  VMOVUPS(b.Offset(32*i), bs[i])
}

// The actual operation.
for i := 0; i < unroll; i++ {
  VPANDN(a.Offset(32*i), bs[i], bs[i])
}

for i := 0; i < unroll; i++ {
  VMOVUPS(bs[i], res.Offset(32*i))
}
```

8

```
ADDQ(U32(blocksize), a.Base)
ADDQ(U32(blocksize), b.Base)
ADDQ(U32(blocksize), res.Base)
SUBQ(U32(blocksize), n)
JMP(LabelRef("blockloop"))

Label("tail")
RET()
```

```
name                                                time/op
SimpleBitmapIndex-12                                10.8µs ± 0%
SimpleBitmapIndexInlined-12                         8.88µs ± 0%
SimpleBitmapIndexInlinedAndNoBoundsCheck-12         8.33µs ± 0%

name                                                      time/op
BiggerBatchBitmapIndexNoBoundsCheck-12                    1.06µs ± 0%

name                                       time/op
SimpleSIMDBitmapIndex-8                     154ns ± 0%
SimpleScalarFasterBitmapIndex-8            1.04µs ± 0%
```



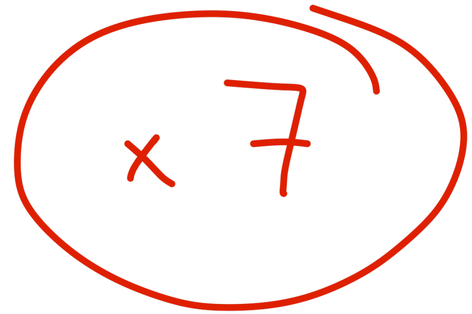
http://bit.ly/bitmapindexes

```
name                                                time/op
SimpleBitmapIndex-12                                10.8µs ± 0%
SimpleBitmapIndexInlined-12                         8.88µs ± 0%
SimpleBitmapIndexInlinedAndNoBoundsCheck-12         8.33µs ± 0%

name                                                      time/op
BiggerBatchBitmapIndexNoBoundsCheck-12                    1.06µs ± 0%

name                                      time/op
SimpleSIMDBitmapIndex-8                    154ns ± 0%
SimpleScalarFasterBitmapIndex-8           1.04µs ± 0%
```

AVX 512

PREFETCHING

JIT



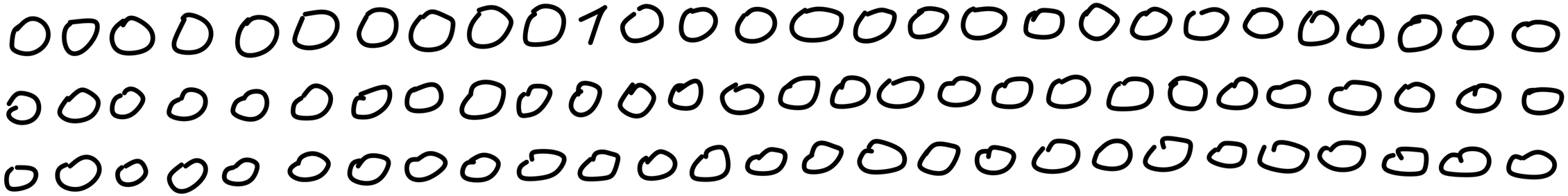
http://bit.ly/bitmapindexes

Bitmap Index “Problems”

1. High cardinality problem
2. High throughput problem
3. Non-trivial queries

High cardinality problem

Representation leads to sparsely populated bitmaps.



Different representation.

Bit per value 000100000000 cardinality 10 value 3

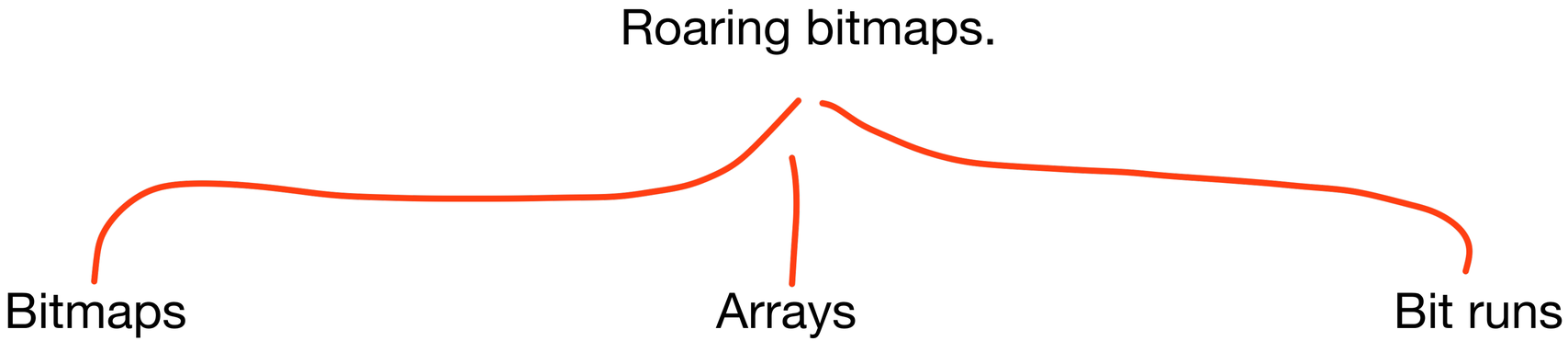
Binary representation 0011 4 bit width

Compression.

PWAH
EWAH

Run-length encoding 01014

https://crd-legacy.lbl.gov/~kewu/ps/LBNL-54673.pdf

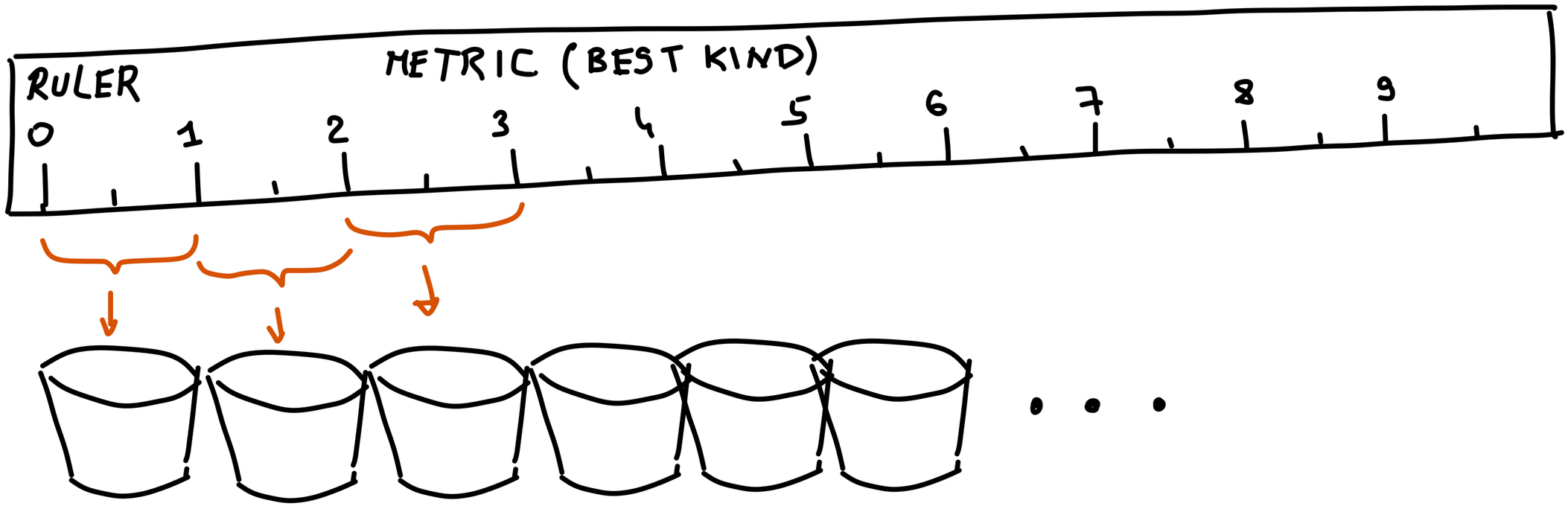


https://www.roaringbitmap.org/

https://arxiv.org/pdf/1603.06549.pdf

Apache Lucene
Apache Spark
InfluxDB
Netflix Atlas
Bleve

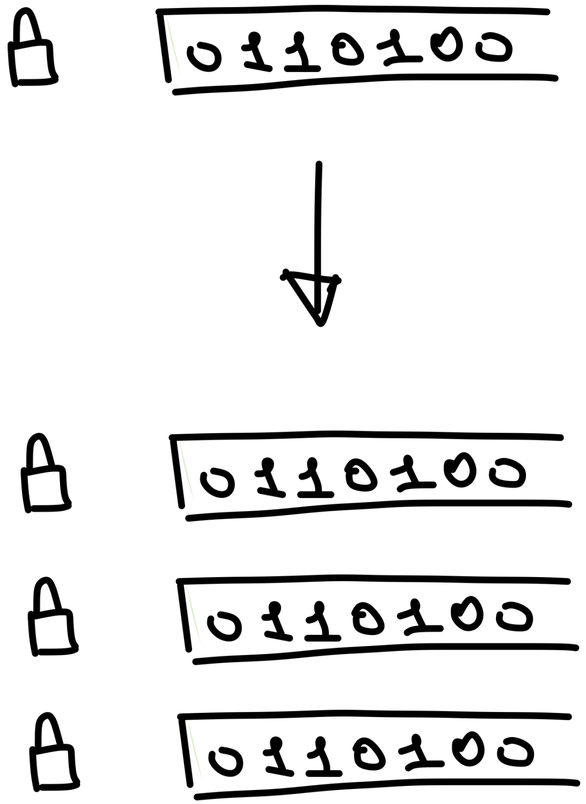
Binning.



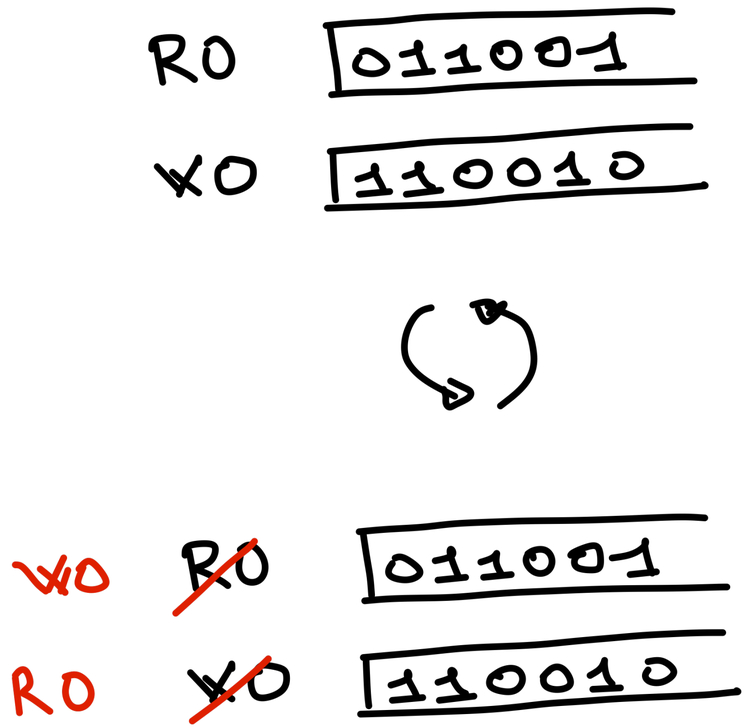
Height (float, (0, inf), inf cardinality) -> Binned height (uint, (50, 250), cardinality 200)

High throughput problem

Sharding



Versioning



Non-trivial queries: range queries

Straightforward solution

“Give me hotel rooms that cost from 200 to 300 dollars a night.”

“Give me hotel rooms that cost 200 dollars a night.”
OR
“Give me hotel rooms that cost 201 dollars a night.”
OR
“Give me hotel rooms that cost 202 dollars a night.”
OR
“Give me hotel rooms that cost 203 dollars a night.”
…
OR
“Give me hotel rooms that cost 300 dollars a night.”

Non-trivial queries: range queries

Binning

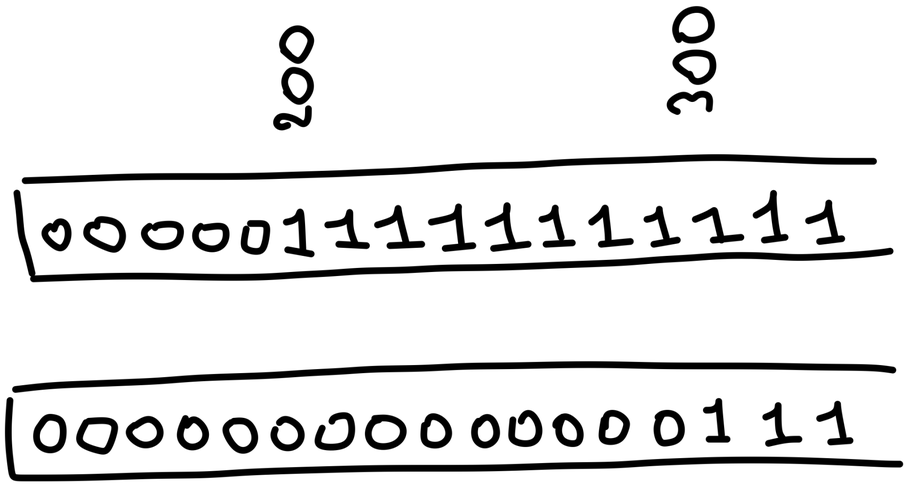
“Give me hotel rooms that cost from 200 to 300 dollars a night.”

“Give me hotel rooms that cost 200-250 dollars a night.”
OR
“Give me hotel rooms that cost 250-300 dollars a night.”

Non-trivial queries: range queries

Range-encoded bitmaps

“Give me hotel rooms that cost from 200 to 300 dollars a night.”



“Give me hotel rooms that cost <= 300 dollars a night.”

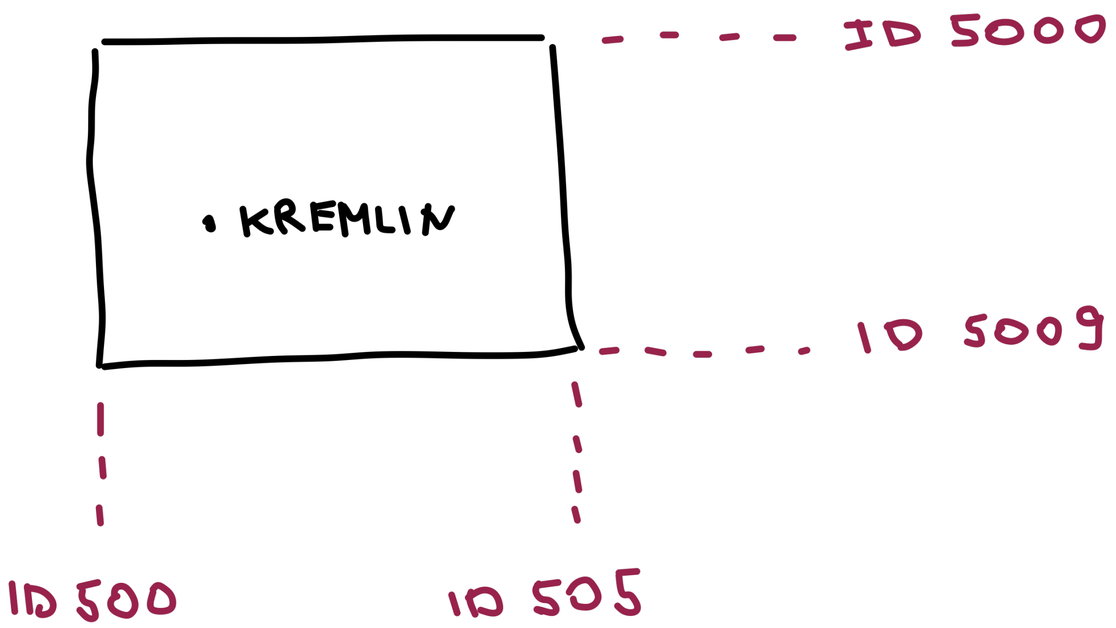
AND
NOT

“Give me hotel rooms that cost <= 199 dollars a night.”

http://www.dabi.temple.edu/~vucetic/cis616spring2005/papers/P4%20p215-chan.pdf

Non-trivial queries: geo queries

Google S2 (Official site, Gophercon Russia 2018 talk)
Badoo Geotri (April 1 article on Habr)




```
WHERE ID VERTICAL
IN (5000,5009) AND
ID HORIZONTAL
IN (500,505)
```

Ready solutions?

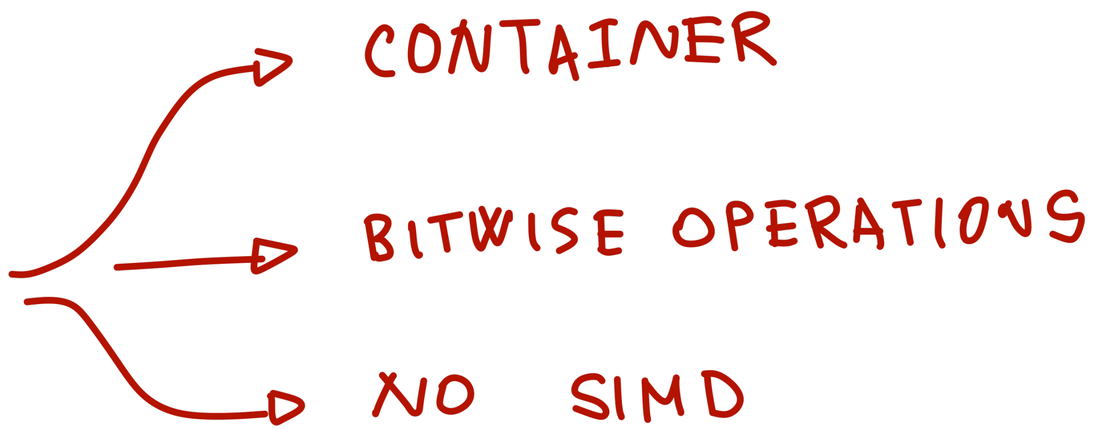
1. Roaring bitmaps

2. Pilosa

3. Maybe mainstream DBMS with get bitmap index support?

Ready solutions?

1. Roaring bitmaps

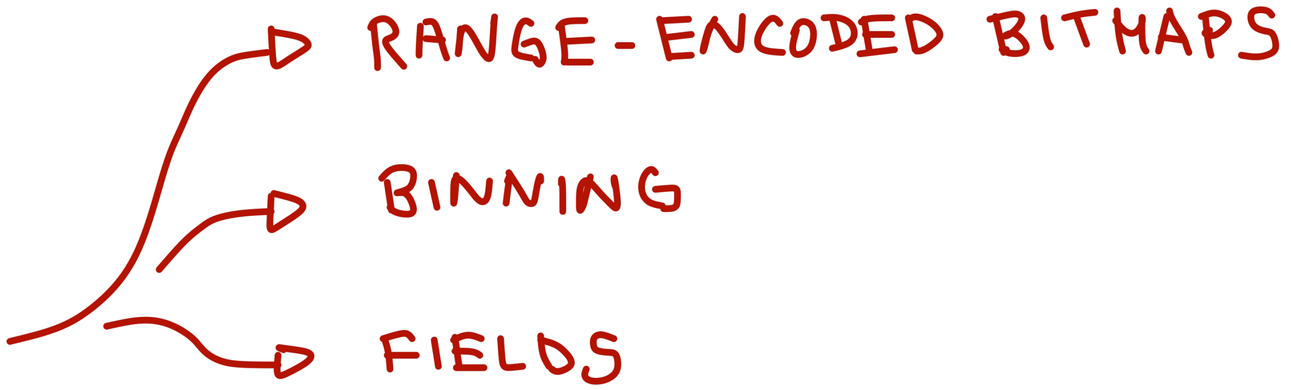


2. Pilosa

3. Maybe mainstream DBMS with get bitmap index support?

Ready solutions?

1. Roaring bitmaps
2. Pilosa
3. Maybe mainstream DBMS with get bitmap index support?



Pilosa example

```
client = pilosa.DefaultClient()

// Retrieve the schema
schema, err = client.Schema()

// Create an Index object
myindex = schema.Index("myindex")

terraceField = myindex.Field("terrace")
reservationsField = myindex.Field("reservations")
expensiveField = myindex.Field("expensive")

// make sure the index and the field exists on the server
err = client.SyncSchema(schema)

fill(r, terraceField, 0.05)
fill(r, reservationsField, 0.95)
fill(r, expensiveField, 0.1)

response, err = client.Query(myindex.Intersect(
  terraceField.Row(0),
  myindex.Not(expensiveField.Row(0)),
  reservationsField.Row(0)))
```

Pilosa example

```
client = pilosa.DefaultClient()

// Retrieve the schema
schema, err = client.Schema()

// Create an Index object
myindex = schema.Index("myindex")

terraceField = myindex.Field("terrace")
reservationsField = myindex.Field("reservations")
expensiveField = myindex.Field("expensive")

// make sure the index and the field exists on the server
err = client.SyncSchema(schema)

fill(r, terraceField, 0.05)
fill(r, reservationsField, 0.95)
fill(r, expensiveField, 0.1)

response, err = client.Query(myindex.Intersect(
  terraceField.Row(0),
  myindex.Not(expensiveField.Row(0)),
  reservationsField.Row(0)))
```

Pilosa example

```
client = pilosa.DefaultClient()

// Retrieve the schema
schema, err = client.Schema()

// Create an Index object
myindex = schema.Index("myindex")

terraceField = myindex.Field("terrace")
reservationsField = myindex.Field("reservations")
expensiveField = myindex.Field("expensive")

// make sure the index and the field exists on the server
err = client.SyncSchema(schema)

fill(r, terraceField, 0.05)
fill(r, reservationsField, 0.95)
fill(r, expensiveField, 0.1)

response, err = client.Query(myindex.Intersect(
  terraceField.Row(0),
  myindex.Not(expensiveField.Row(0)),
  reservationsField.Row(0)))
```

Pilosa example

```
client = pilosa.DefaultClient()

// Retrieve the schema
schema, err = client.Schema()

// Create an Index object
myindex = schema.Index("myindex")

terraceField = myindex.Field("terrace")
reservationsField = myindex.Field("reservations")
expensiveField = myindex.Field("expensive")

// make sure the index and the field exists on the server
err = client.SyncSchema(schema)

fill(r, terraceField, 0.05)
fill(r, reservationsField, 0.95)
fill(r, expensiveField, 0.1)

response, err = client.Query(myindex.Intersect(
  terraceField.Row(0),
  myindex.Not(expensiveField.Row(0)),
  reservationsField.Row(0)))
```

Pilosa example

```
client = pilosa.DefaultClient()

// Retrieve the schema
schema, err = client.Schema()

// Create an Index object
myindex = schema.Index("myindex")

terraceField = myindex.Field("terrace")
reservationsField = myindex.Field("reservations")
expensiveField = myindex.Field("expensive")

// make sure the index and the field exists on the server
err = client.SyncSchema(schema)

fill(r, terraceField, 0.05)
fill(r, reservationsField, 0.95)
fill(r, expensiveField, 0.1)

response, err = client.Query(myindex.Intersect(
  terraceField.Row(0),
  myindex.Not(expensiveField.Row(0)),
  reservationsField.Row(0)))
```

Pilosa example

```
client = pilosa.DefaultClient()

// Retrieve the schema
schema, err = client.Schema()

// Create an Index object
myindex = schema.Index("myindex")

terraceField = myindex.Field("terrace")
reservationsField = myindex.Field("reservations")
expensiveField = myindex.Field("expensive")

// make sure the index and the field exists on the server
err = client.SyncSchema(schema)

fill(r, terraceField, 0.05)
fill(r, reservationsField, 0.95)
fill(r, expensiveField, 0.1)

response, err = client.Query(myindex.Intersect(
  terraceField.Row(0),
  myindex.Not(expensiveField.Row(0)),
  reservationsField.Row(0)))
```

```
$ go run pilosa.go
2019/03/30 20:41:12 filling the data...
2019/03/30 20:41:47 finished filling the data
2019/03/30 20:41:47 got 2796 columns
```

Ready solutions?

1. Roaring bitmaps

2. Pilosa

3. Maybe mainstream DBMS will get bitmap index support?

Closing words

- Bitmap indexes
- Performance optimizations

Thank you

http://bit.ly/bitmapindexes
https://github.com/mkevac/gopherconrussia2019