FOSDEM Go Devroom, February 2025

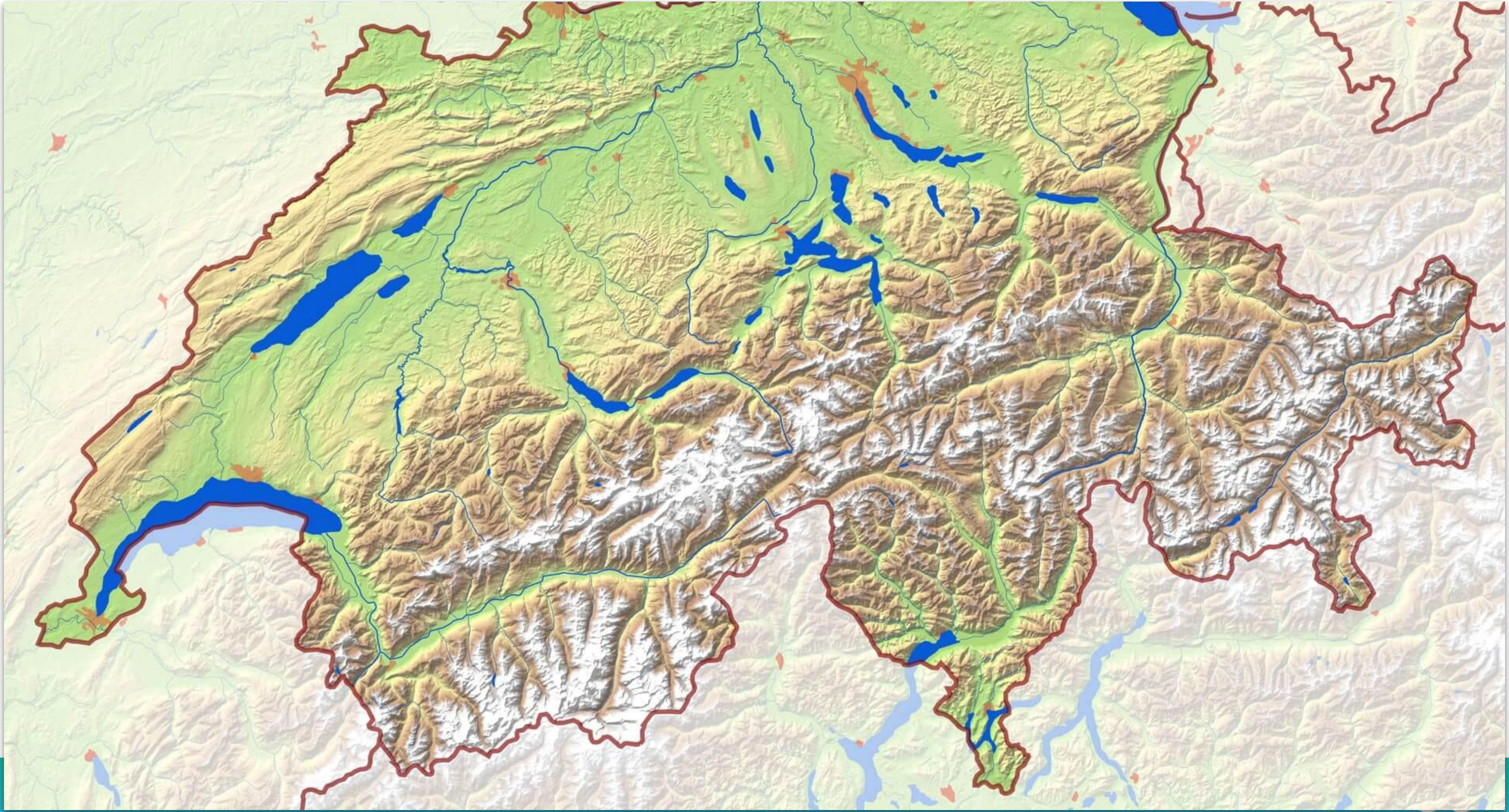
Swiss Maps

Bryan Boreham

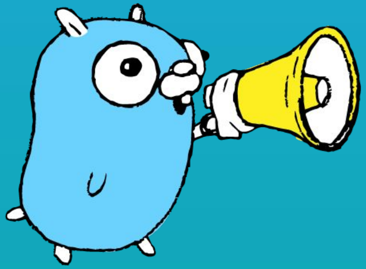
Grafana Labs

@bboreham@grafana.social

@bboreham.bsky.social



Overview



- Who am I?
- What am I talking about?
- Why am I talking about this?
- How does it work?
- When does it not work?

"Swiss Map" is a new map implementation in Go 1.24

Hi, I'm Bryan Boreham

I work at Grafana Labs, mostly on:

Prometheus Mimir loki Tempo

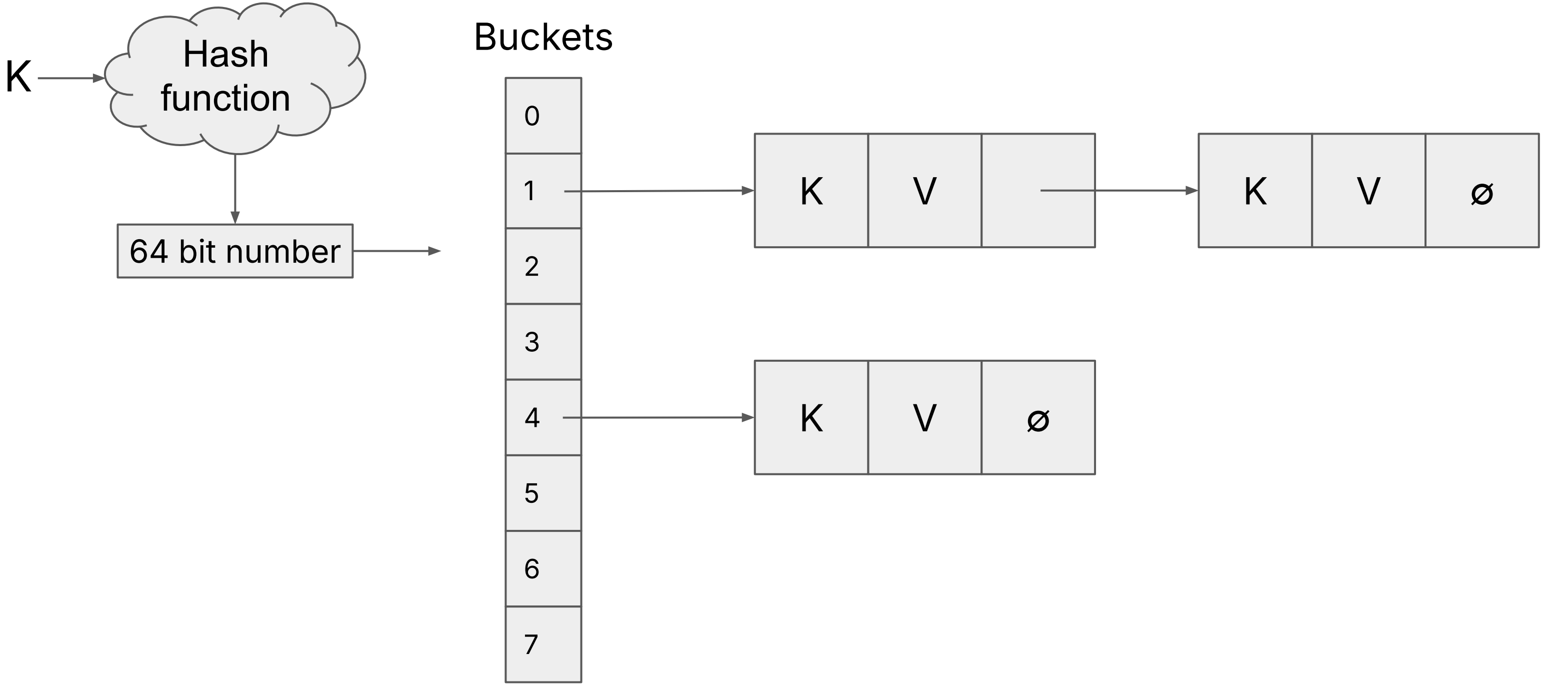
My inspiration for this talk

Abseil's Open Source Hashtables: 2 Years In - CppCon 2019

A Go map

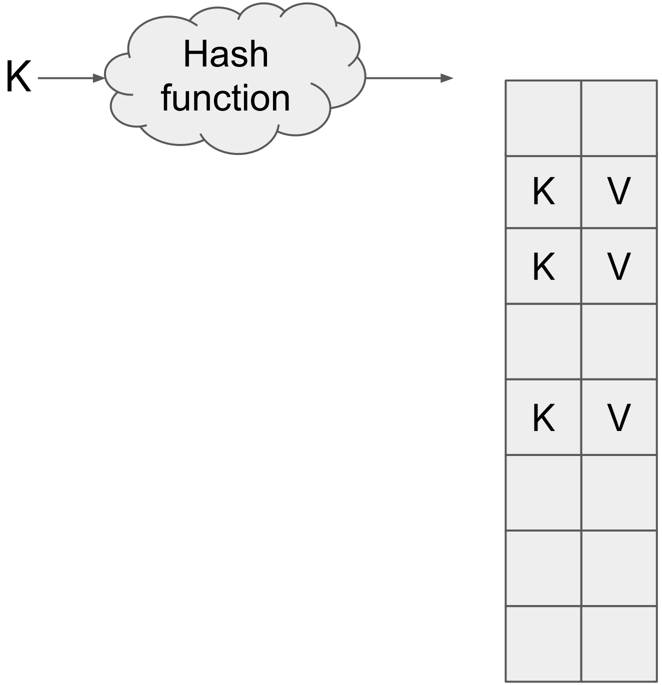
Construct:	`m := map[string]int{}`
Insert:	`m["route"] = 66`
Lookup:	`v = m[k]`
Delete:	`delete(m, k)`
Iterate:	`for k, v := range m`
Size:	`len(m)`

Classical hash map with chaining



This is called "open hashing"

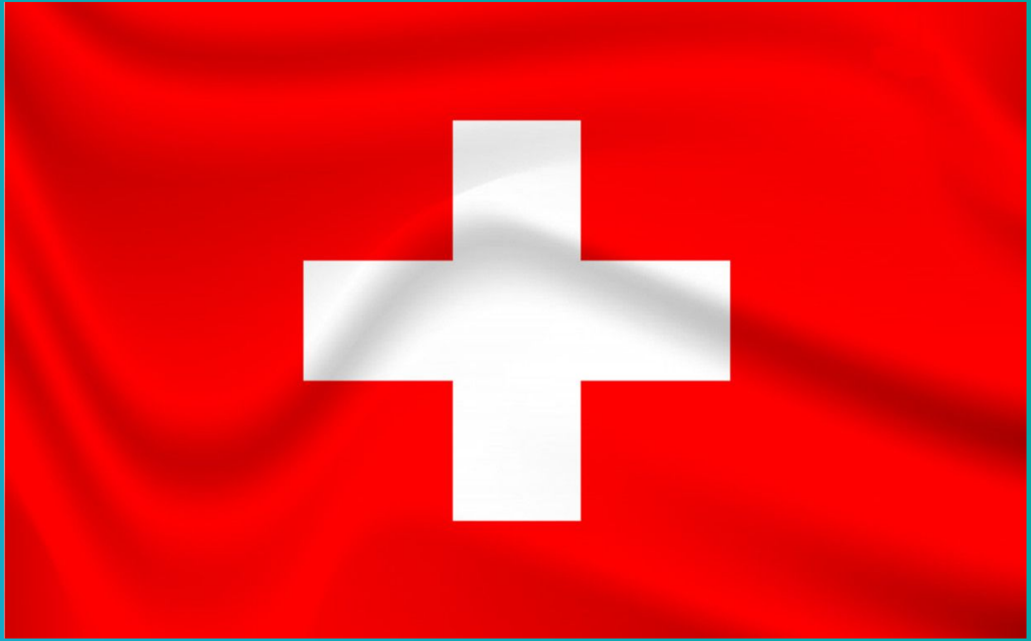
Classical hash map with probing



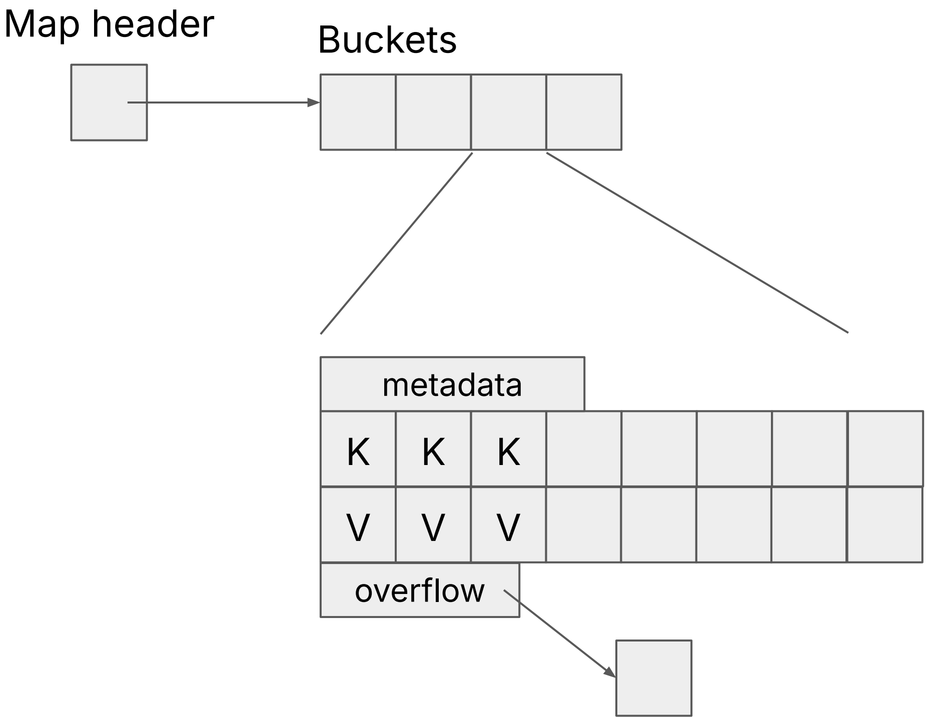
This is called "closed hashing"

Closed Hashing

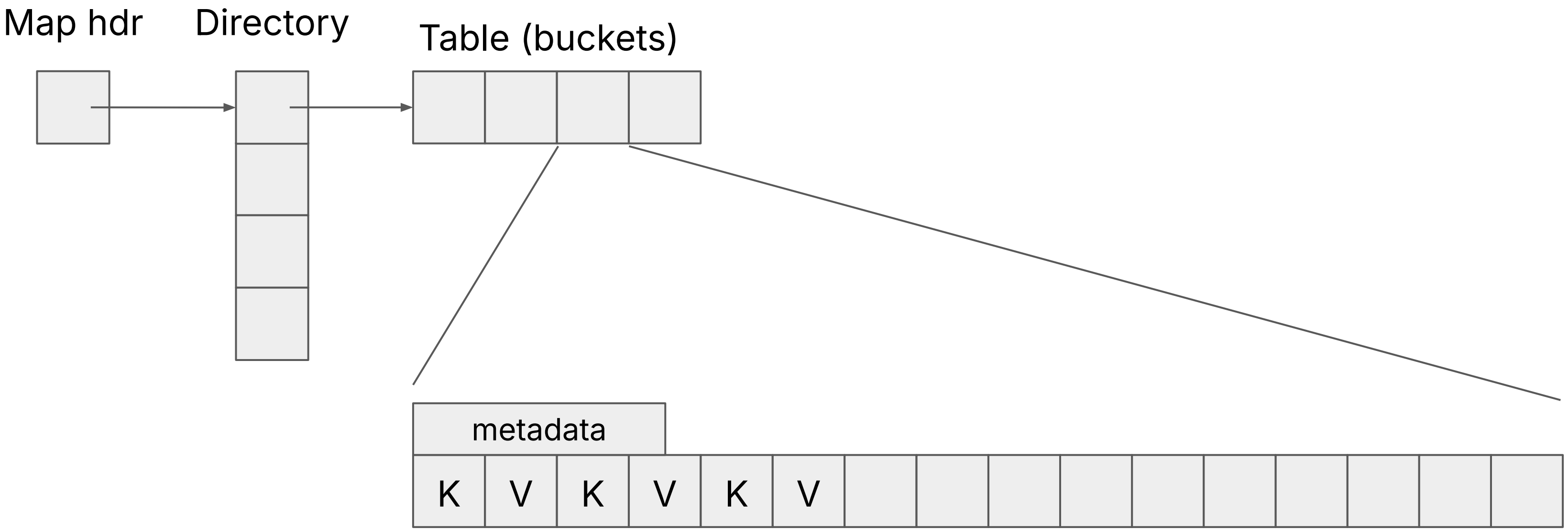
CH



Go 1.23 map (before Swiss Maps)

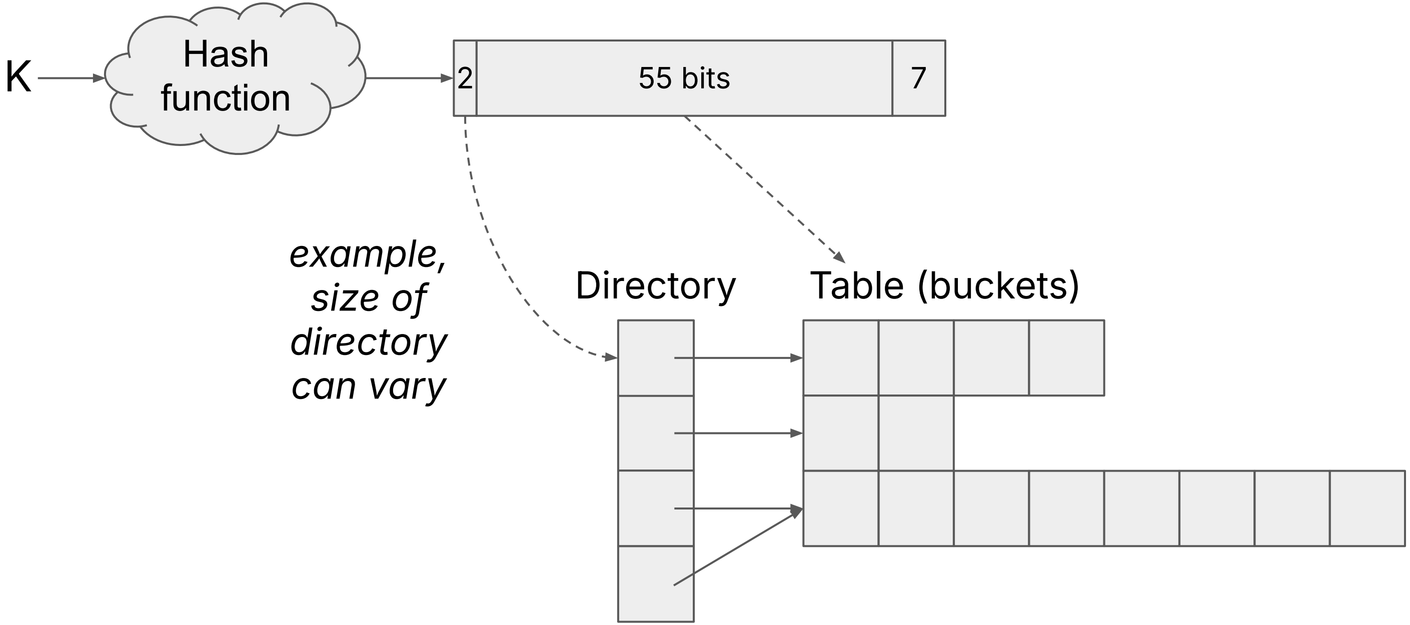


Go 1.24 map (Swiss Map)



What was that metadata?

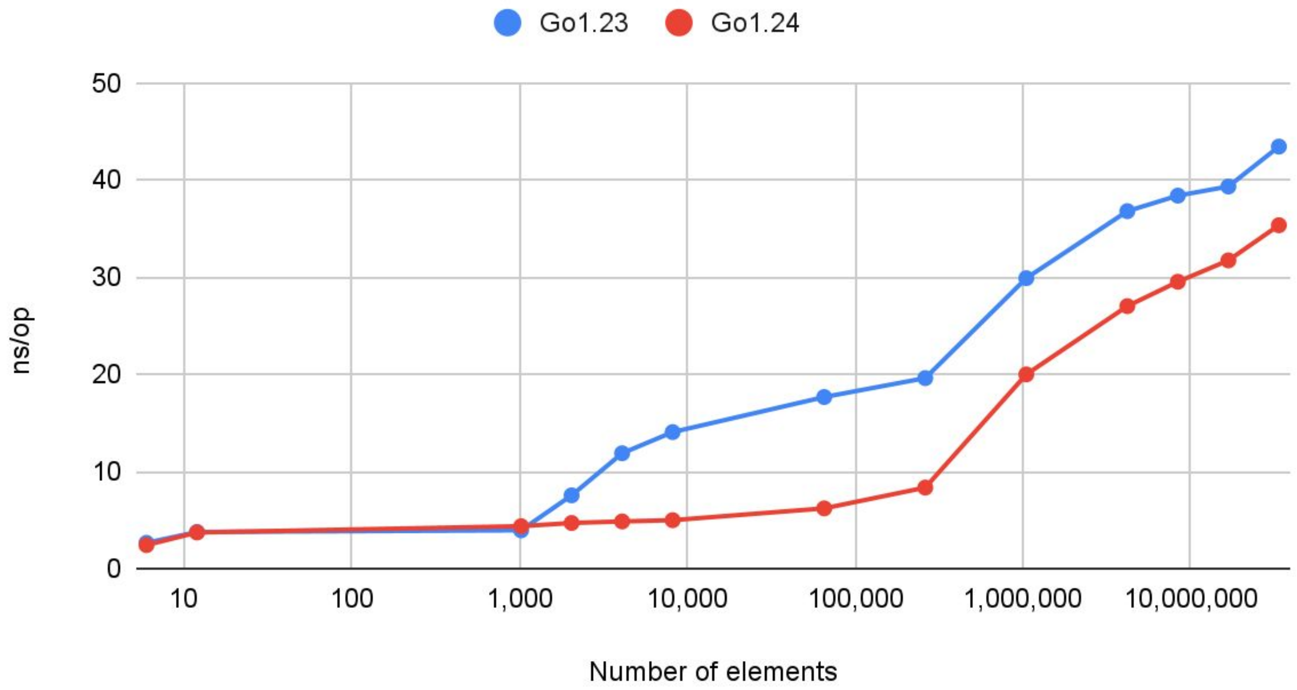
Go 1.24 map: more detail



example, size of directory can vary

How Does It Perform?

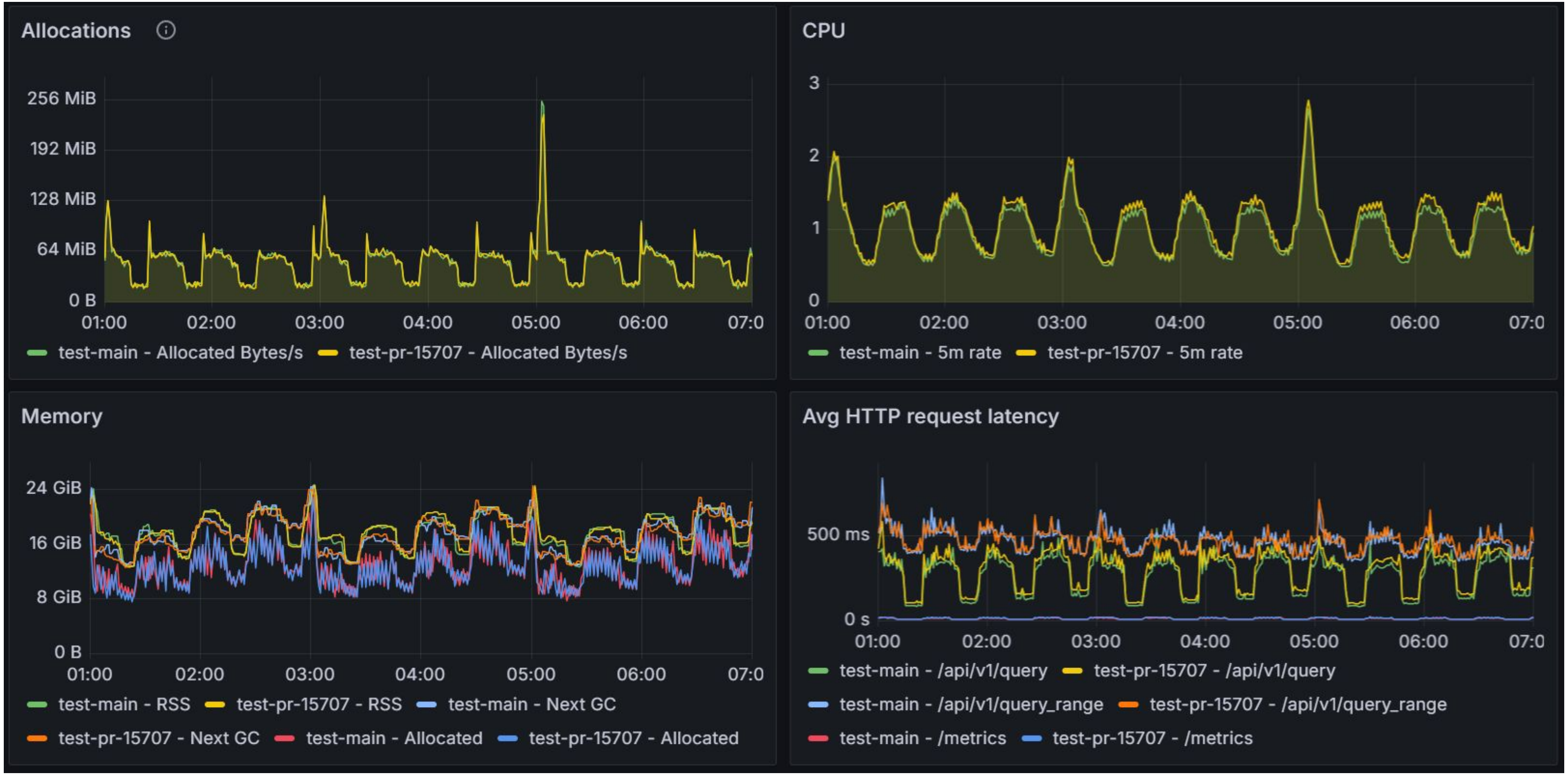
Speed: map[int64]int64 Lookup - Hit



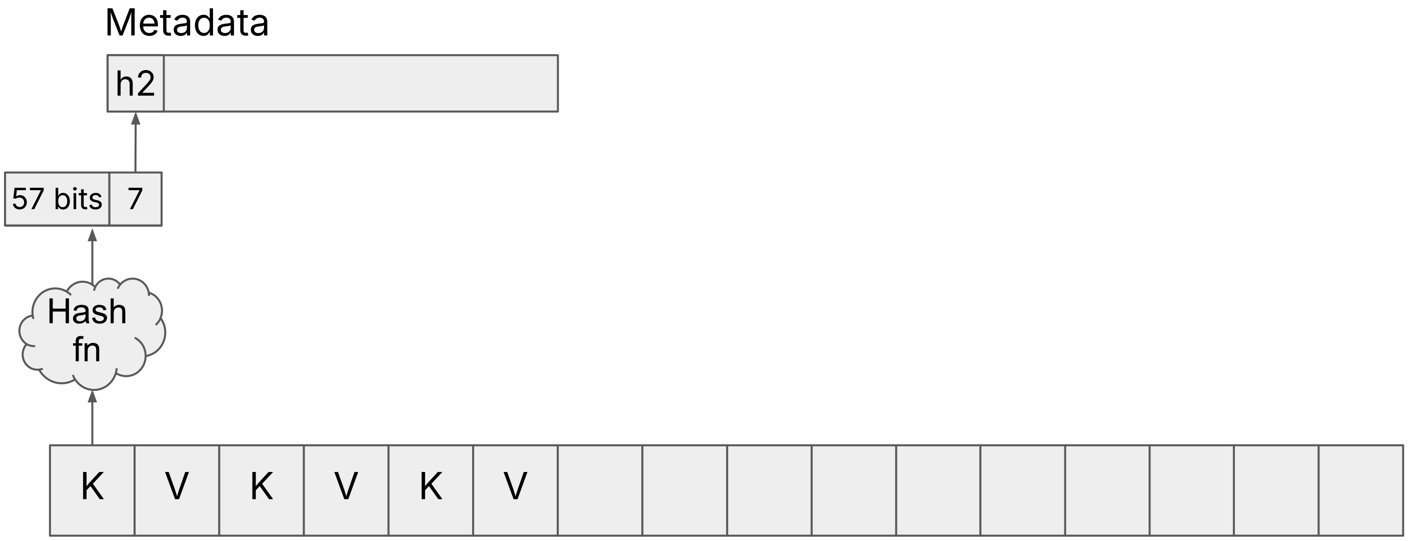
BenchmarkMapAccessHit, on Intel® Core™ i7-14700K

Lower is better

Running Prometheus with 6M series

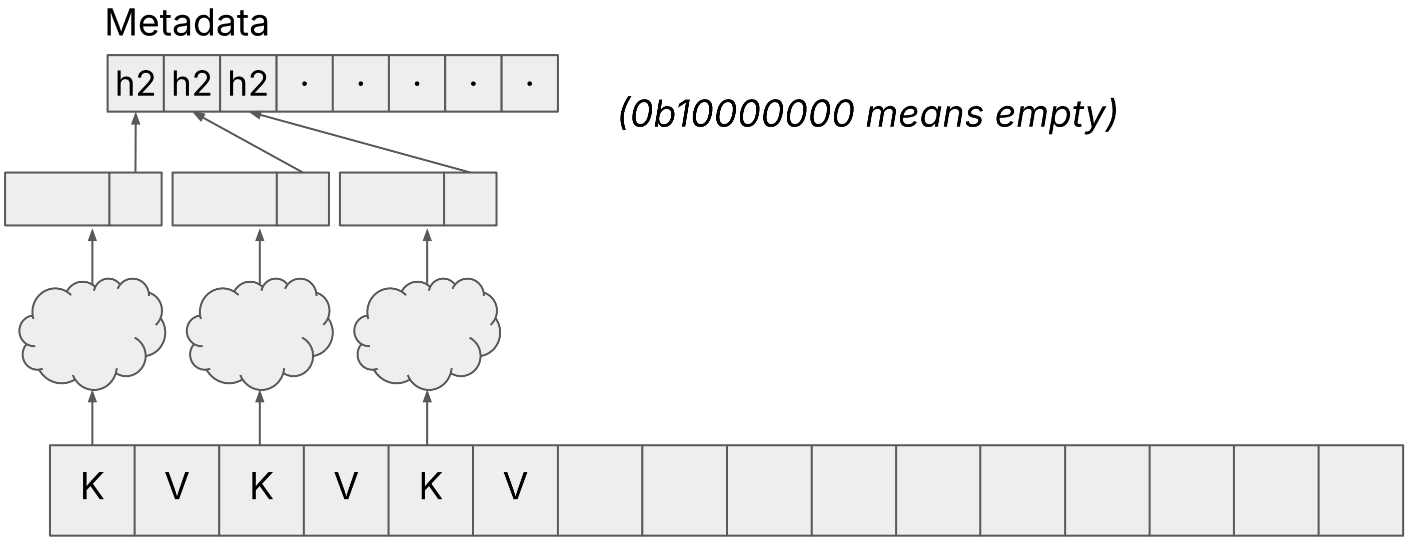


Bucket metadata



Not to scale

Control word has 1 byte per element



(0b10000000 means empty)

Example: finding an element

Metadata from bucket (0b10000000 == 0x80 is empty)

42	03	05	80	80	80	80	80

Key to find, multiplied by 0x0101010101010101

03	03	03	03	03	03	03	03

Xor

41	00	06	83	83	83	83	83

Subtract 0x0101010101010101 then AND with NOT

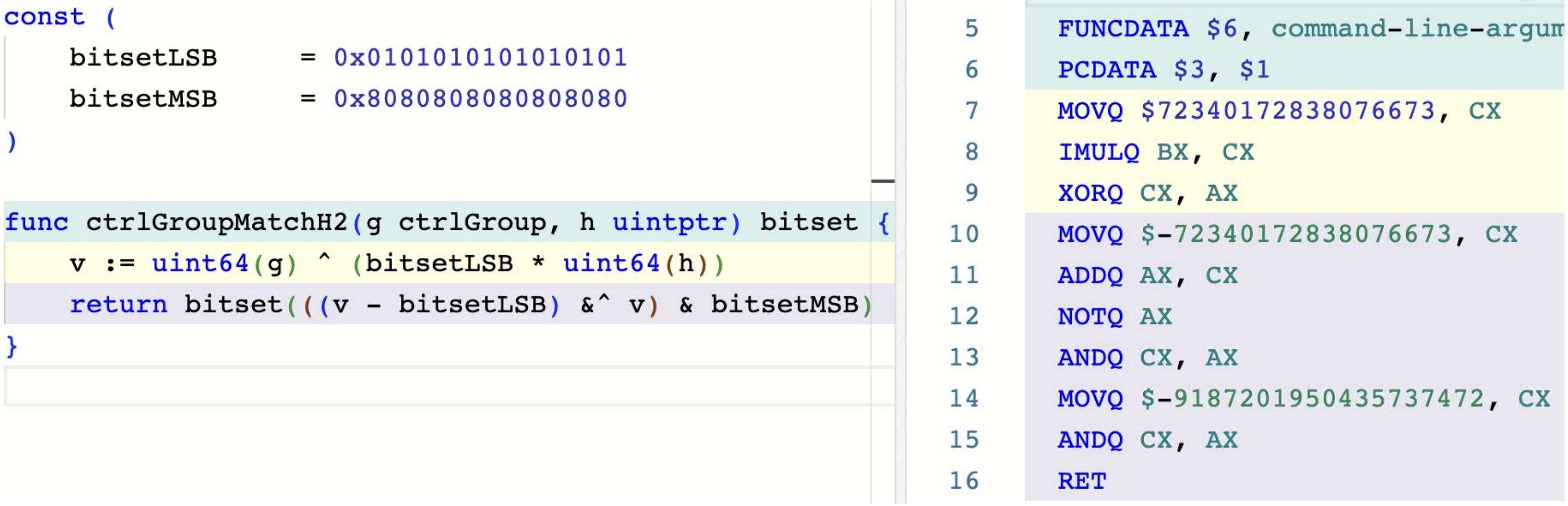
3E	FF	01	00	00	00	00	00

Mask top bit

00	10	00	00	00	00	00	00

Watch carefully.

Compiled code



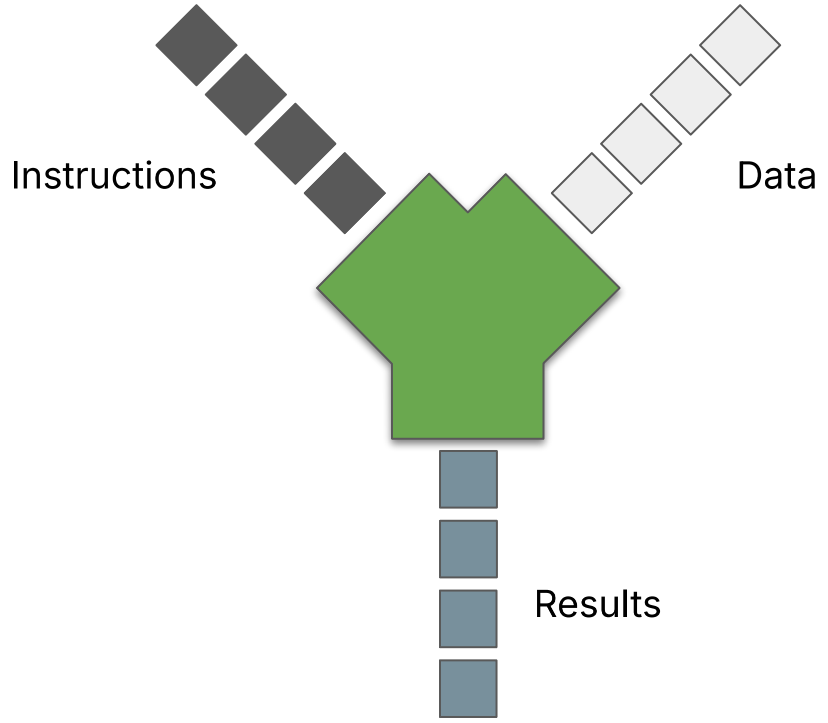
```go
const (
    bitsetLSB     = 0x0101010101010101
    bitsetMSB     = 0x8080808080808080
)

func ctrlGroupMatchH2(g ctrlGroup, h uintptr) bitset {
    v := uint64(g) ^ (bitsetLSB * uint64(h))
    return bitset(((v - bitsetLSB) &^ v) & bitsetMSB)
}
```

```
FUNCDATA $6, command-line-argum
PCDATA $3, $1
MOVQ $72340172838076673, CX
IMULQ BX, CX
XORQ CX, AX
MOVQ $-72340172838076673, CX
ADDQ AX, CX
NOTQ AX
ANDQ CX, AX
MOVQ $-9187201950435737472, CX
ANDQ CX, AX
RET
```

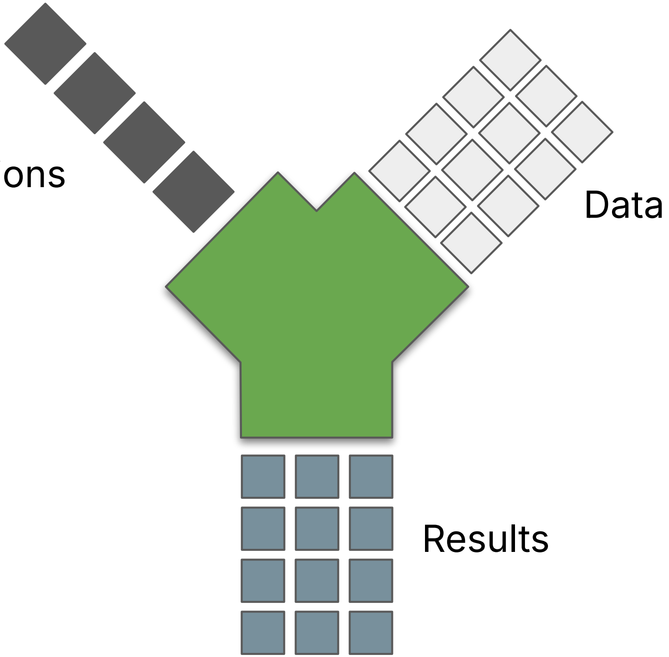
https://godbolt.org/z/KE88ccKGz

SIMD?

Single Instruction Single Data



Single Instruction Multiple Data



GOAMD64=v3 code generated

```
(gdb) disassemble <internal/runtime/maps.ctrlGroupMatchH2> *
    movq     %rcx,%xmm0
    pshufb   %xmm15,%xmm0
    movq     %rsi,%xmm1
    pcmpeqb  %xmm1,%xmm0
    pmovmskb %xmm0,%esi
```

* actually this is inlined

…/cmd/compile/internal/ssagen/intrinsics.go *

```
addF("internal/runtime/maps", "ctrlGroupMatchH2",
    [...]
    if buildcfg.GOAMD64 >= 2 {
        // Broadcast h2 into each byte of a word.
        broadcast := s.newValue1(ssa.OpAMD64PSHUFBbroadcast, types.TypeInt128, hfp)
        // Compare each byte of the control word with h2.
        eq := s.newValue2(ssa.OpAMD64PCMPEQB, types.TypeInt128, broadcast, gfp)
        // Mask: each output bit is equal to the sign bit each input byte.
        out := s.newValue1(ssa.OpAMD64PMOVMSKB, types.Types[types.TUINT16], eq)
```

* edited to fit slide

Finding an element, SIMD version

Metadata from bucket (0b10000000 == 0x80 is empty)

42	03	05	80	80	80	80	80

```
pshufb   %xmm15,%xmm0
```

03	03	03	03	03	03	03	03

```
pcmpeqb  %xmm1,%xmm0
```

00	11	00	00	00	00	00	00

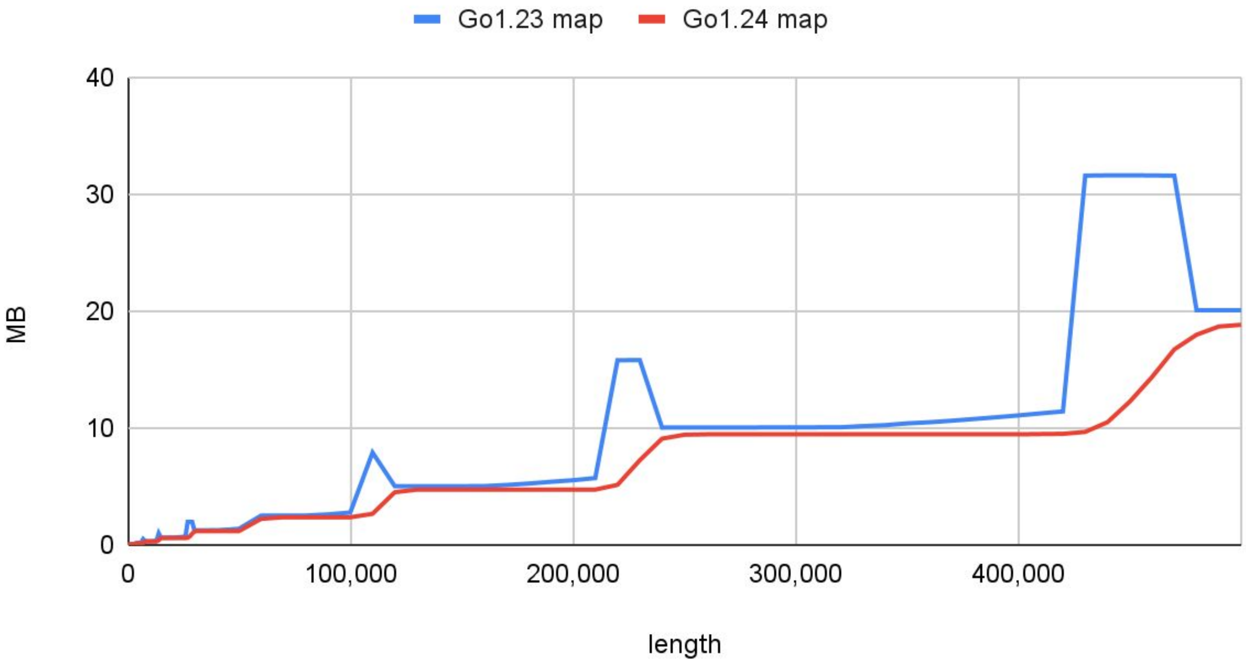
```
pmovmskb %xmm0,%esi
```

00	00	00	00	00	00	00	02

0b00000010

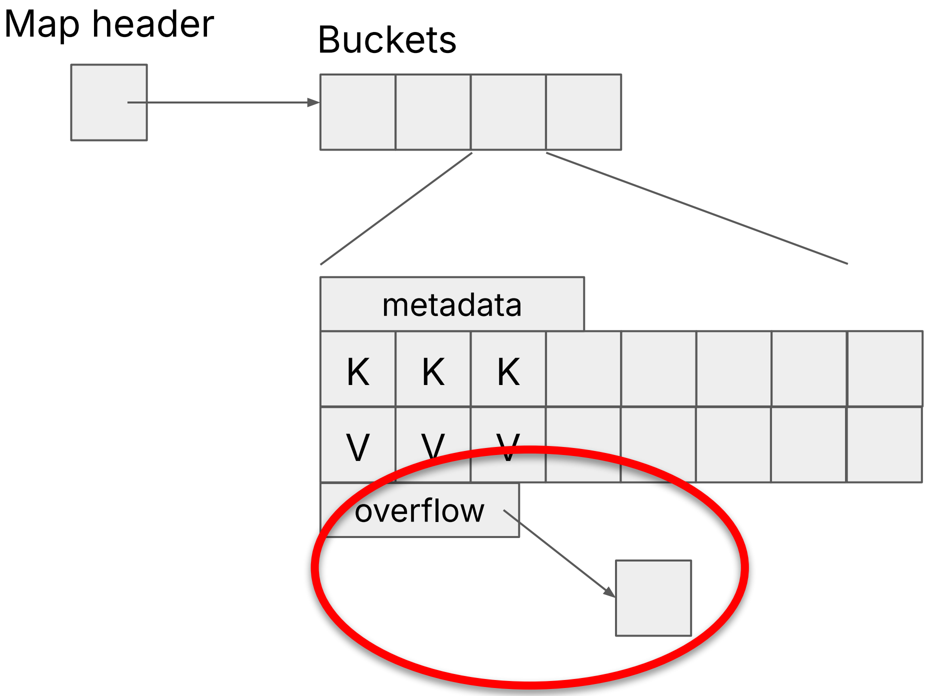
Back to the benchmarks

Memory size: with `make(map[int64]int64)`

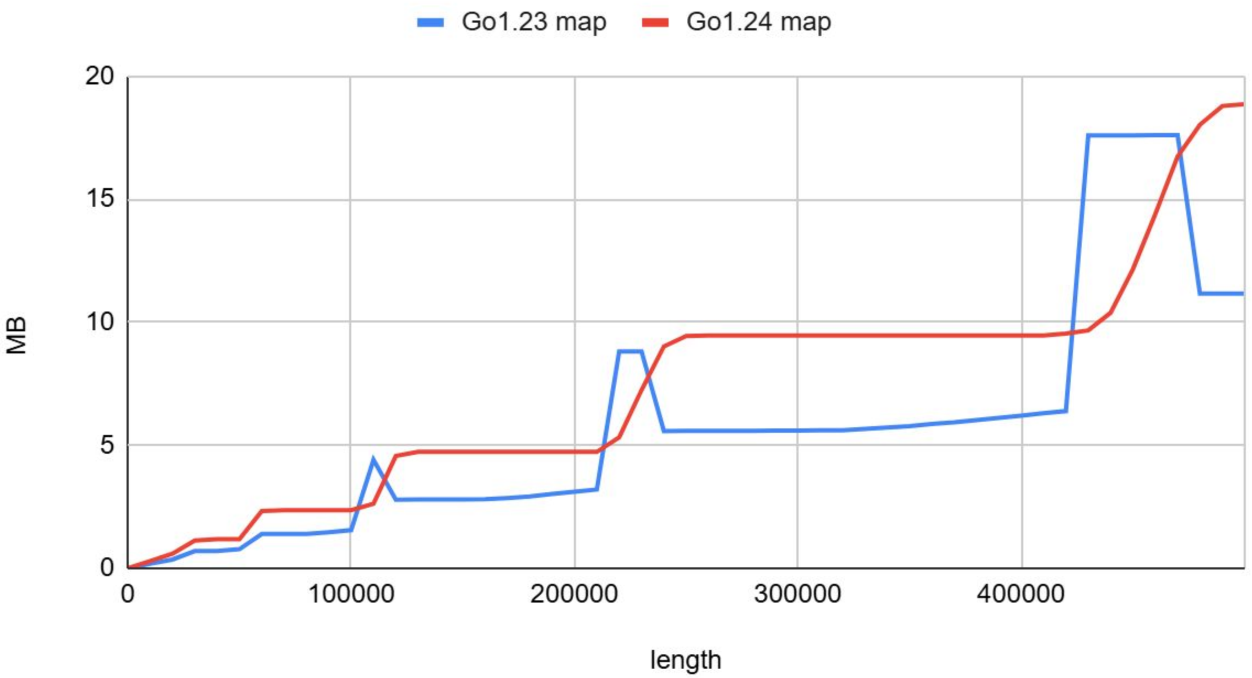


Lower is better

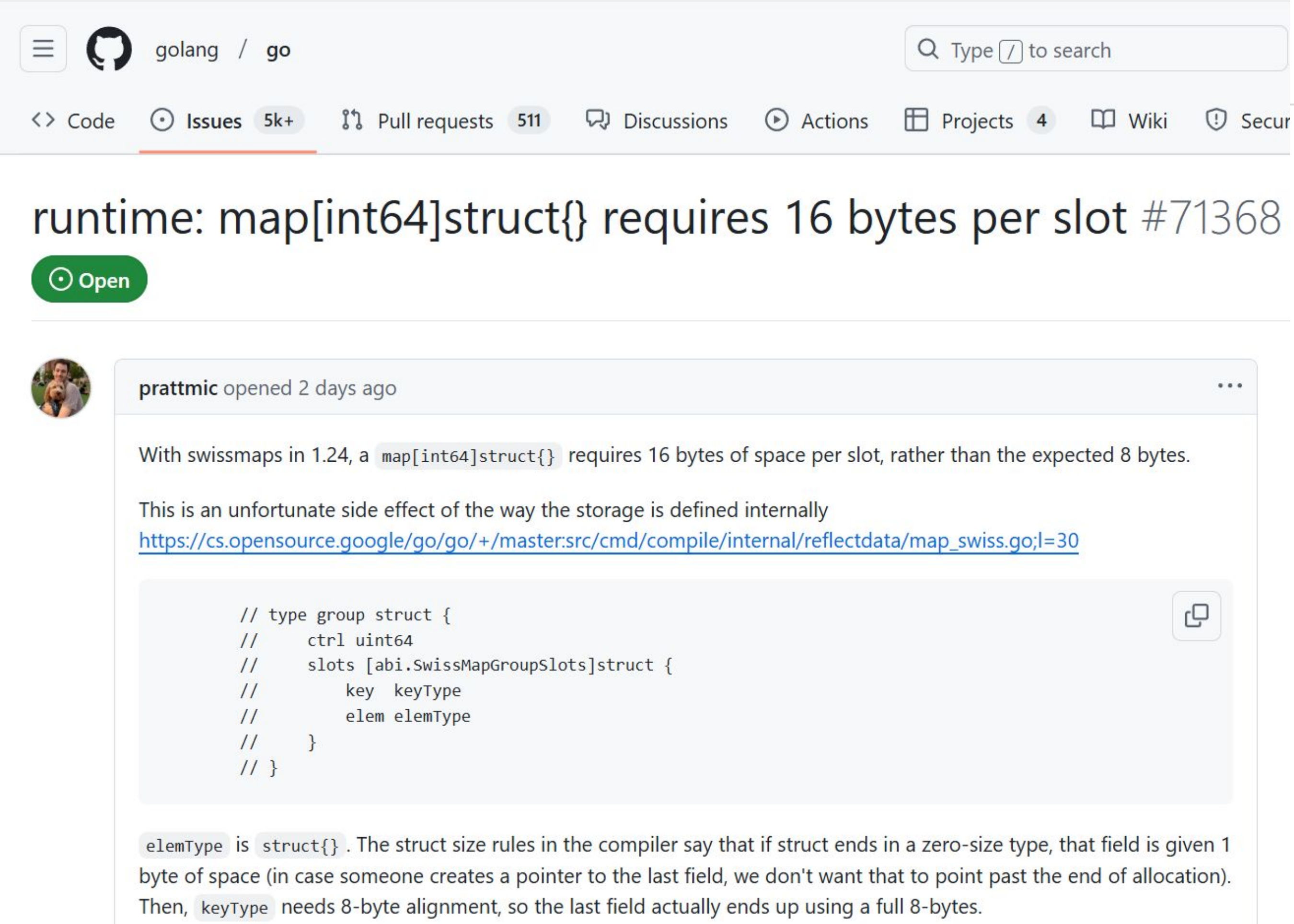
Go 1.23 map (recap)



Memory size: with `make(map[int64]struct{})`



Lower is better



runtime: map[int64]struct{} requires 16 bytes per slot #71368

prattmic opened 2 days ago

With swissmaps in 1.24, a `map[int64]struct{}` requires 16 bytes of space per slot, rather than the expected 8 bytes.

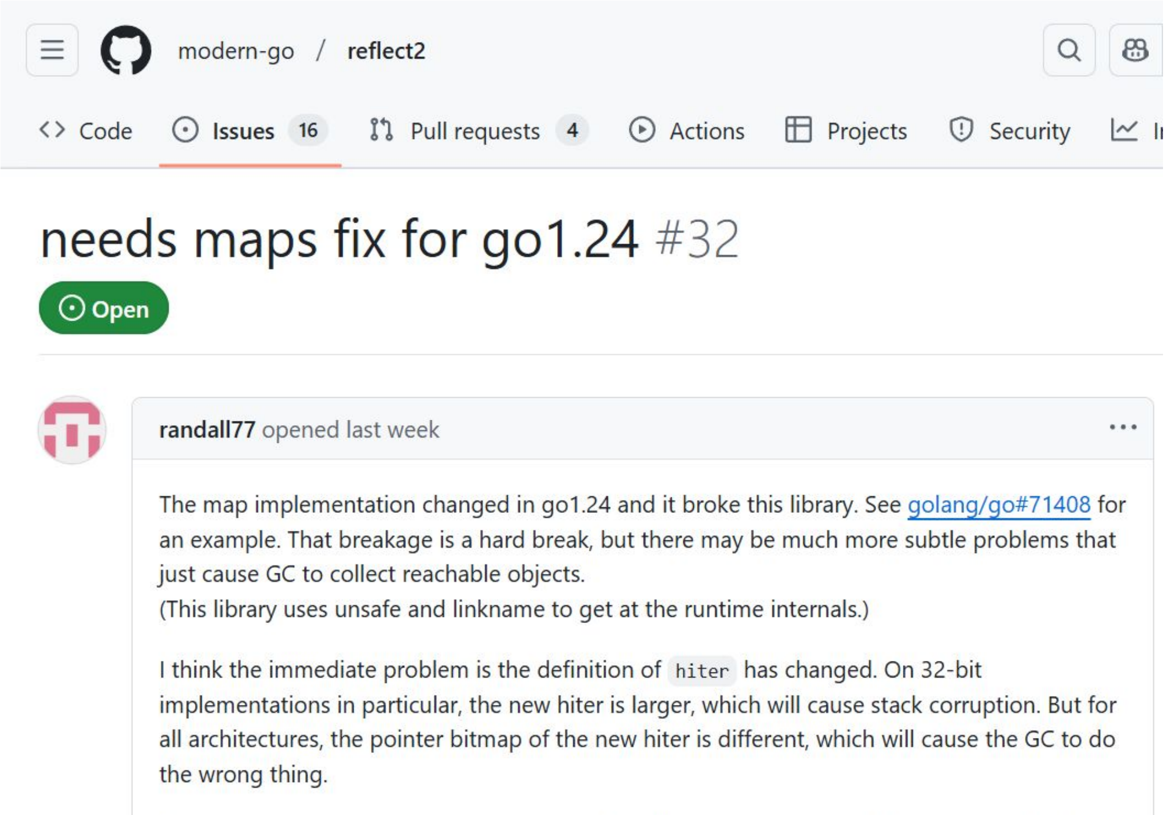
This is an unfortunate side effect of the way the storage is defined internally https://cs.opensource.google/go/go/+/master:src/cmd/compile/internal/reflectdata/map_swiss.go;l=30

```
	// type group struct {
	//     ctrl uint64
	//     slots [abi.SwissMapGroupSlots]struct {
	//         key  keyType
	//         elem elemType
	//     }
	// }
```

`elemType` is `struct{}`. The struct size rules in the compiler say that if struct ends in a zero-size type, that field is given 1 byte of space (in case someone creates a pointer to the last field, we don't want that to point past the end of allocation). Then, `keyType` needs 8-byte alignment, so the last field actually ends up using a full 8-bytes.

https://go.dev/issue/71368

Unsafe access (reflect2, json-iterator, ...)



needs maps fix for go1.24 #32

randall77 opened last week

The map implementation changed in go1.24 and it broke this library. See golang/go#71408 for an example. That breakage is a hard break, but there may be much more subtle problems that just cause GC to collect reachable objects.
(This library uses unsafe and linkname to get at the runtime internals.)

I think the immediate problem is the definition of `hiter` has changed. On 32-bit implementations in particular, the new hiter is larger, which will cause stack corruption. But for all architectures, the pointer bitmap of the new hiter is different, which will cause the GC to do the wrong thing.

Credit for Swiss Maps goes to

Alkis Evlogimenos
Jeff Dean
Jeffrey Lim
Matt Kulukundis
Roman Perepelitsa
Sam Benzaquen
Sanjay Ghemawat
Shaindel Schwartz

Michael Pratt
Keith Randall
Cherry Mui
ZhangYunHao
thepudds
Peter Mattis

...plus more, I'm sure

(not to me)

Questions?

Links:

https://abseil.io/about/design/swisstables

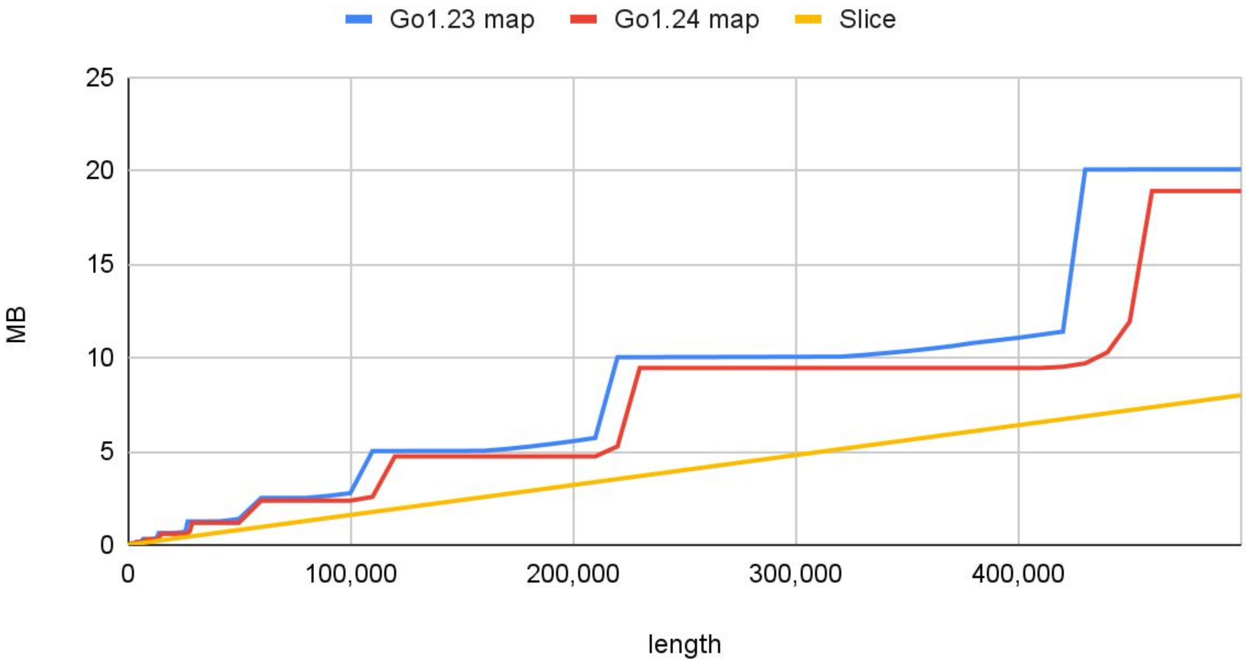
Videos:

GopherCon 2016: Inside the Map Implementation - Keith Randall

CppCon 2017: Designing a Fast, Efficient, Cache-friendly Hash Table - Matt Kulukundis

Swiss Guard by Mary Pollard; running Gopher by Ramya Anand.
Original Gopher design by Renee French.

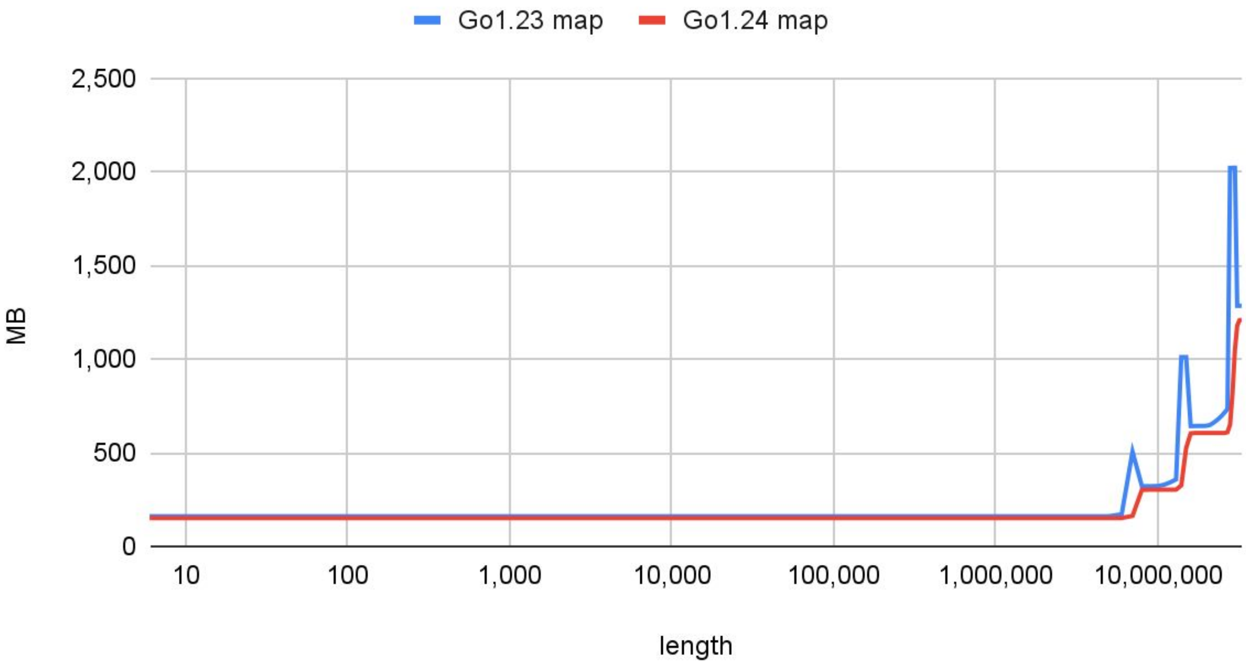
Memory size: with `make(map[int64]int64, size)`



Measured using Go runtime profiler.

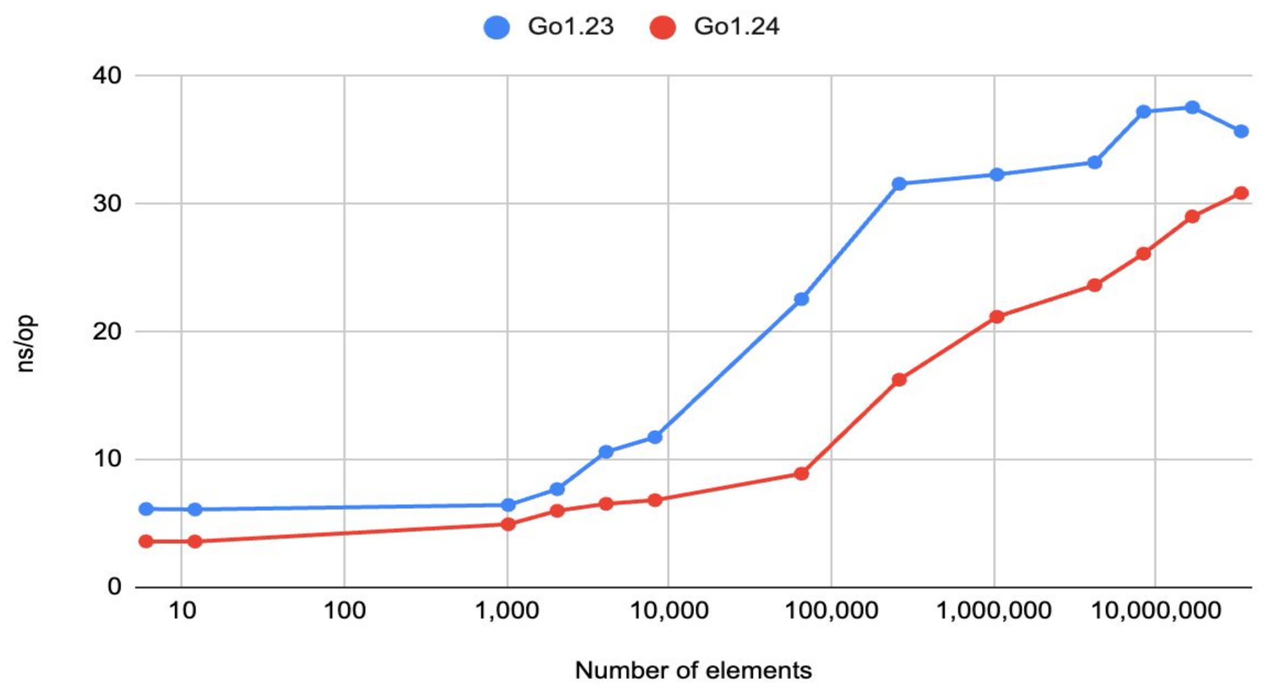
Lower is better

Memory size: make(map[int64]int64, 4194304)



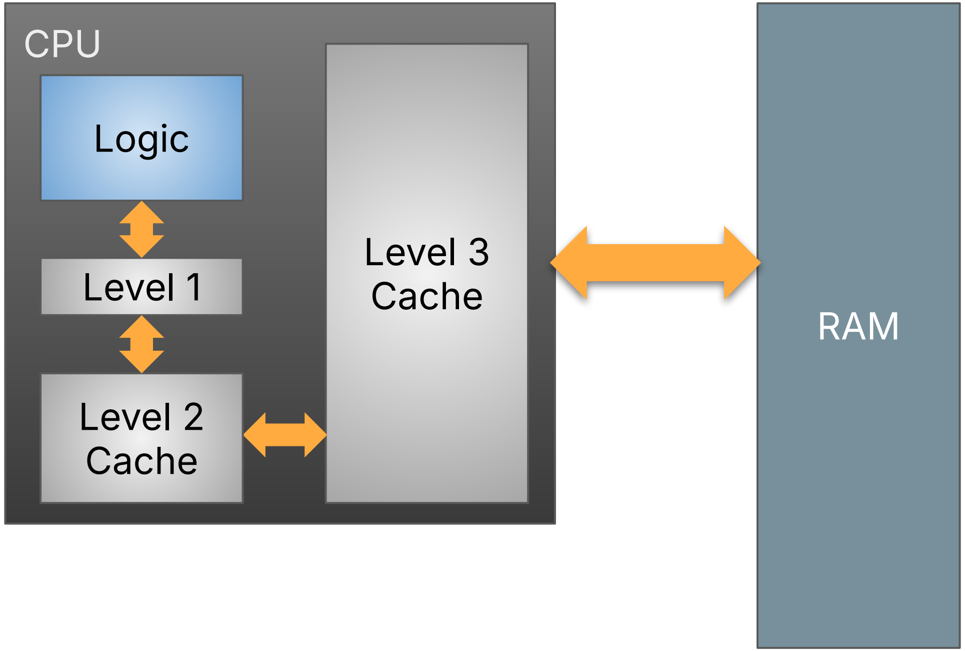
Lower is better

Speed: map[int64]int64 Miss: make(map[int64]int64, 4M)



Lower is better

CPU Memory Architecture



(Not to scale)

Requirements of a map

In the Go specification:

A map is an unordered group of elements of one type, called the element type, indexed by a set of unique *keys* of another type, called the key type. - https://go.dev/ref/spec

In the Go blog:

"in general they offer fast lookups, adds, and deletes" - https://go.dev/blog/maps

Special features of Go maps

Can't take the address of an element.

"For an operand `x` of type `T`, the address operation `&x` generates a pointer of type `*T` to `x`. The operand must be *addressable*, that is, either a variable, pointer indirection, or slice indexing operation; or a field selector of an addressable struct operand; or an array indexing operation of an addressable array."

Can modify the map during iteration.

"If a map entry that has not yet been reached is removed during iteration, the corresponding iteration value will not be produced. If a map entry is created during iteration, that entry may be produced during the iteration or may be skipped."

Quoting https://go.dev/ref/spec

Special features of Go maps

Can't provide your own hash function.

Can't take the address of an element.

Can modify the map during iteration.

Quoting https://go.dev/ref/spec