GOEXPERIMENT=arena

Vale a pena usar **arenas**?

O que eu venho fazendo?

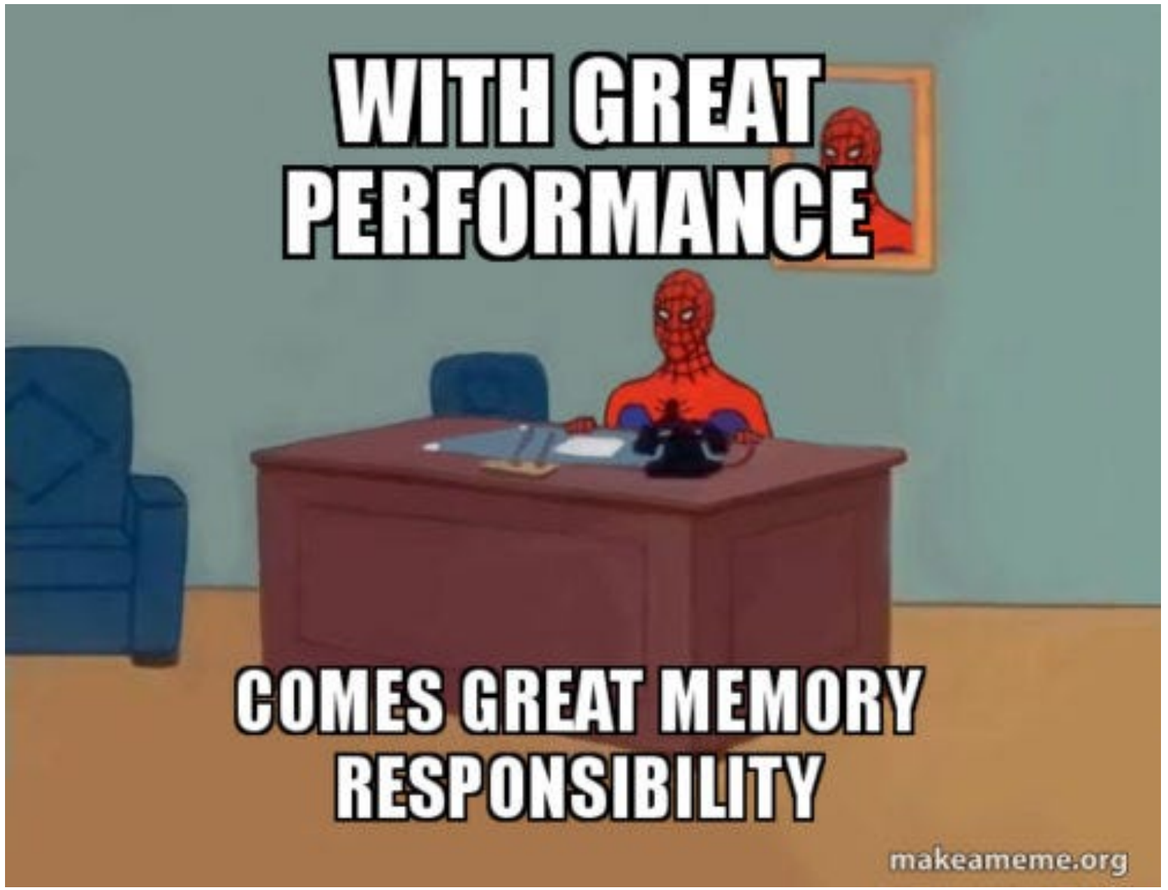
stone

MANNING

Packt


Me, who forgot a
free() in a loop
Malloc
Memory: 104390 MB

Memória não é infinita



AGENDA

1 - Memory Layout

2 - Garbage Collector

3 - Go Memory Arena

4 - Case Study

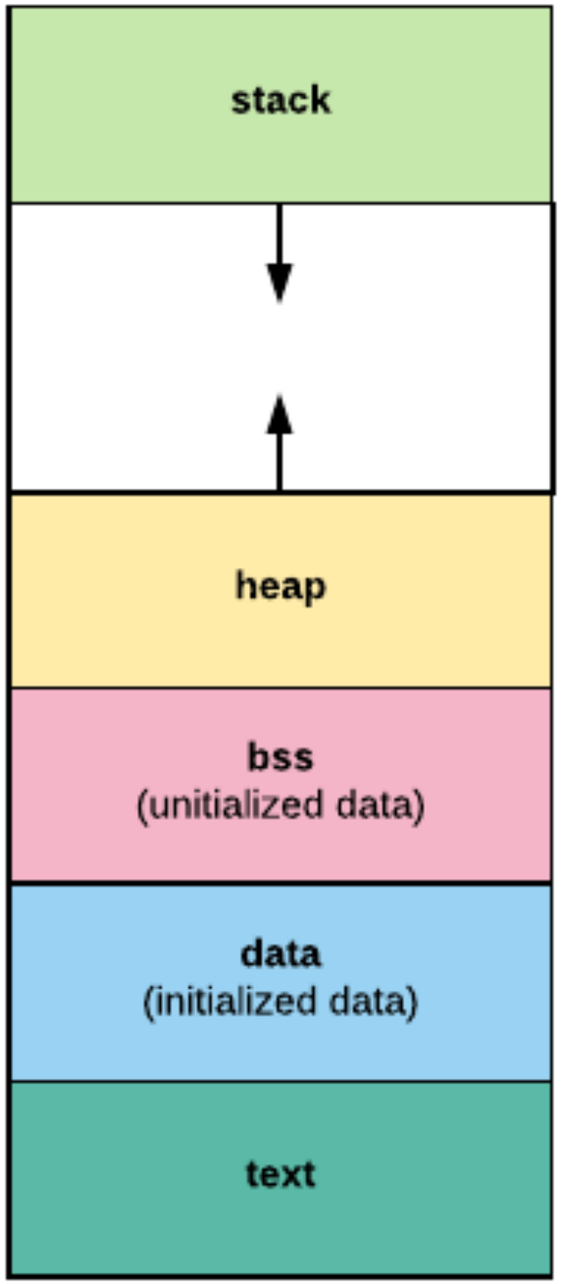
5 - Guidelines

1 – Memory Layout

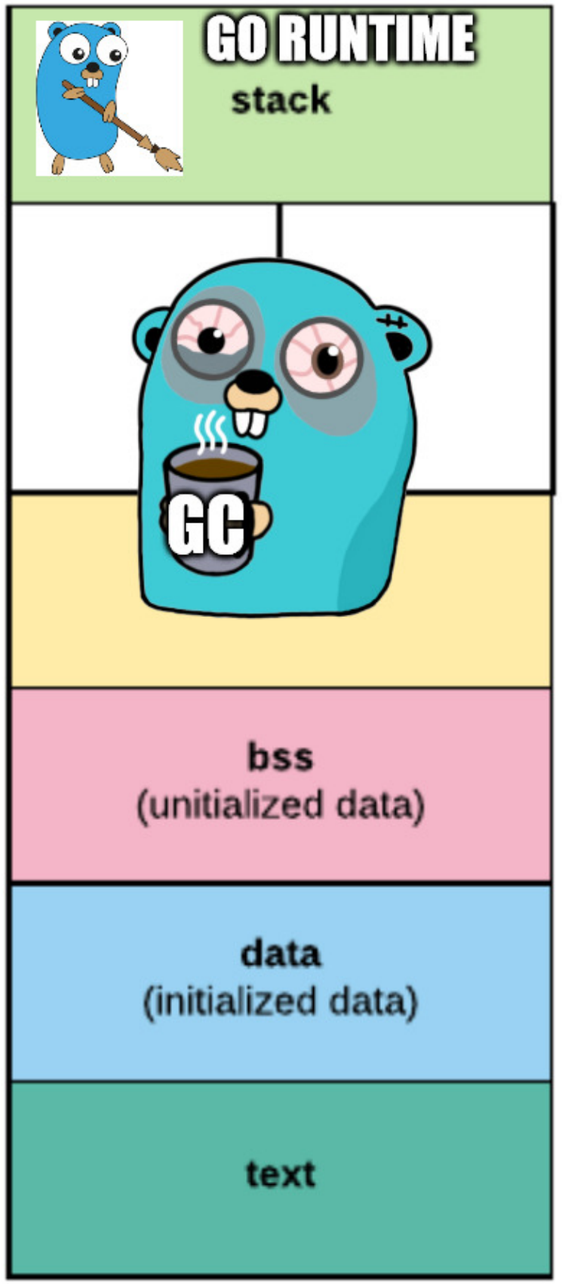
Memory Layout

» Known size objects -> Runtime managed

» Unknown size objects -> GC Managed



GC vs. Runtime



2 – Garbage Collector

Maior foco no programa e menos na memória

Evitar acidentes

Responsabilidades

- Rastrear alocações na heap
- Liberar alocações que não são mais necessárias
- Manter as alocações em uso

Memory inference

O que liberar?

Memory inference

» Tracing GC

» Reference Counting GC

Tracing GC

Tracing Garbage Collection is a form of automatic memory management that consists of determining which objects should be deallocated ("garbage collected") by tracing which objects are *reachable* by a chain of references from certain "root" objects, and considering the rest as "garbage" and collecting them.

The Journey of Go's Garbage Collector

https://go.dev/blog/ismmkeynote

Timeline

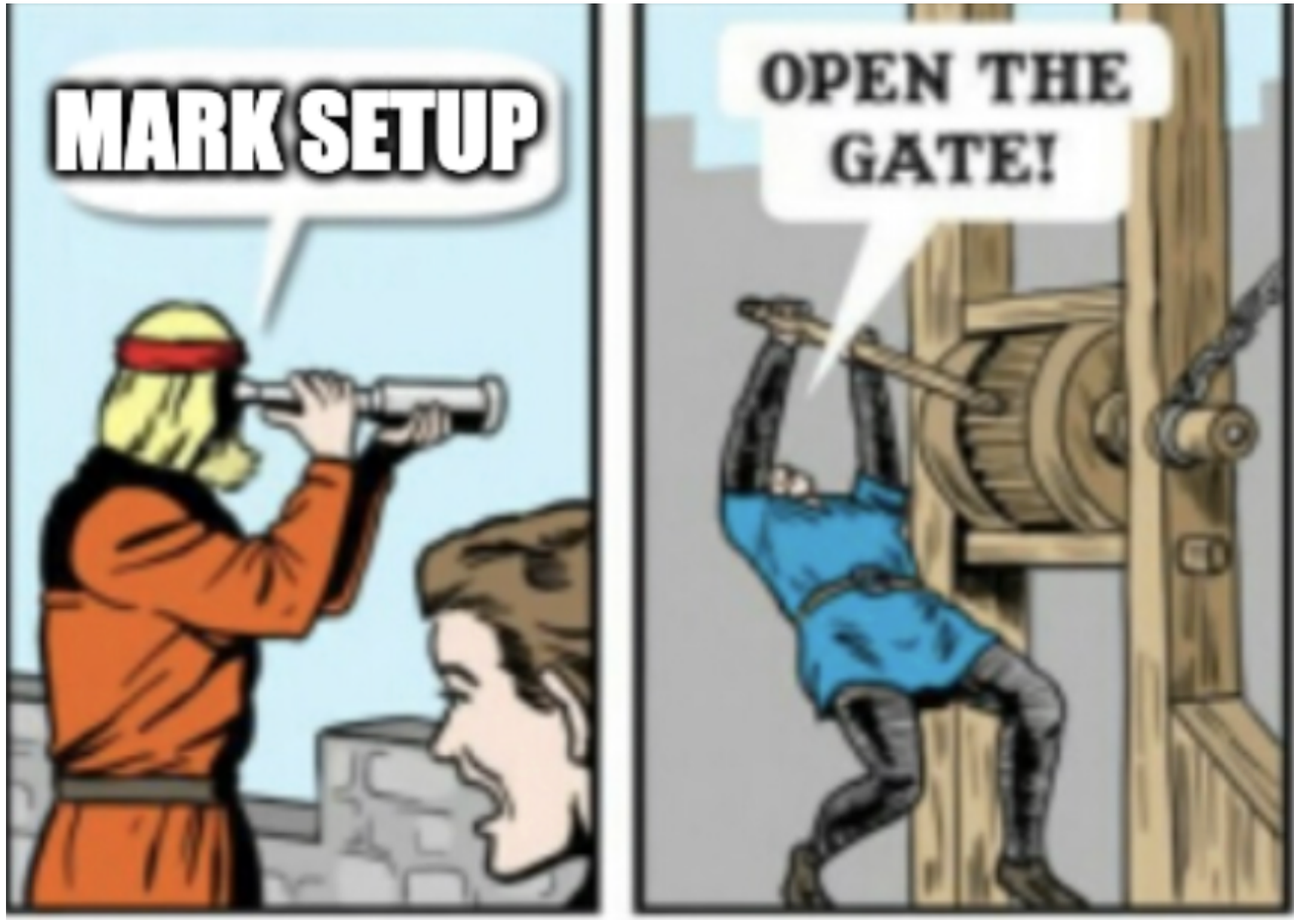
- 1.0 - Mark and Sweep (300ms)
- 1.5 - Tricolor, Concurrent, Mark and Sweep (10ms)
- latest - Optimizations (0,5ms)

GC Phases

- Mark setup - STW
- Marking - Concurrent
- Mark Termination - STW

GC Phases - Mark Setup

- Liga Write Barrier
- Todas as goroutines devem estar paradas.



GC Phases - Marking

» Procura pelos ponteiros nas `roots` para a heap

» Percorre o grafo da heap desses ponteiros

» Marca os valores na heap como em uso

GC Phases - Concurrent Marking

Concurrent:

- 25% of the available CPU for itself
- Até +5% com mark assist

GC Phases – Mark Assist

GC Phases - Mark Termination

- Turn off the Write Barrier
- Clean up tasks
- Next collection goal is calculated

Sweep

Freeing Heap Memory

Fine tuning 

GOGC (env var)

Default GOGC=100

Próxima execução quando o tamanho da heap aumentar 100%

Heap Goal

Se a coleta terminar com 10MB na heap a próxima irá executar quando a heap chegar a 10MB de novas alocações.

GC Pacer

Quando começar uma nova coleta?

Collection Cadence

Estima quanto tempo irá levar para finalizar a coleta.

 + GC = + Latency 

» Rouba capacidade da CPU (no mínimo 25%)

» STW Latency (0% for work)

Resumo do estado atual da GC

- 25% CPU durante a coleta
- Heap 2x live heap (ou max heap)
- 2 STW com tempos menores que 0,5ms por ciclo de coleta
- Em steady state temos poucos Mark Assists

Ajude o GC a te ajudar ℹ️

P1 - IdentifIque e remova alocações desnecessárias

P2:

- Mantenha a heap tão pequena quanto possível
- Encontre o pace ideal
- Fique dentro dos objetivos para cada coleta
- Minimize o tempo da coleta, STW e Mark Assists

Problema

Alocar um grande quantidade de objetos em uma operação em um período de tempo bem definido.

GOGC=off ?  

OOM 

Soft limit?

GOMEMLIMIT (env var)

Latency vs. memory footprint

No manual management

Memory Region

- AED Free Storage Package (1967) - zones
- PL/I (1976) - AREA data type
- C (1990) - Arenas
- Apache HTTP Server - Pools
- Postgres - Memory Context

https://en.wikipedia.org/wiki/Region-basedmemorymanagement

3 – Go Memory Arena

proposal: arena: new package providing memory arenas

`https://github.com/golang/go/issues/51317`

This proposal is on hold indefinitely due to serious API concerns. The GOEXPERIMENT=arena code may be changed incompatibly or removed at any time, and ***we do not recommend its use in production****.*

Objetivos

» Alocar de uma vez uma região de memória (chunk) para nossos objetos.

» Poder desalocar todo essa região de uma vez só.

» Não introduzir os problemas da gestão manual.

Como usar?

```
GOEXPERIMENT=arena
```

```
import "arena"
```

Arena - crash course

- Criar uma arena
- Fazer operações usando arena para alocar memória onde for necessário.
- Liberar a arena.

Criando uma nova Arena

```
func NewArena() Arena

--------------

mem := arena.NewArena()
```

Incluindo objetos na Arena

```
func New[T any](a *Arena) *T
```

```
mem := arena.NewArena()
p := arena.New[Person](mem)
```

Colocando objetos na heap

```
func Clone[T any](s T) T

--------------

mem := arena.NewArena()
p1 := arena.New[Person](mem) // arena-allocated
p2 := arena.Clone(p1) // heap-allocated
```

Liberando a memória

```
func (a *Arena) Free()
```

```
mem.Free()
```

Criando slices

```
func MakeSlice[T any](a *Arena, len, cap int) []T
```

```
mem := arena.NewArena()
slice := arena.MakeSlice[string](mem, 100, 100)
```

Address sanitizer

```
type T struct {
    Num int
}

func main() {
    mem := arena.NewArena()
    o := arena.New[T](mem)
    mem.Free()
    o.Num = 123 // 🙅🏼
}
```

Address sanitizer (Asan)

go run -asan main.go

```
accessed data from freed user arena 0x40c0007ff7f8
```

Diferença na abordagem

Alocando na heap -> O(N) deallocs

Alocando na arena -> O(1) dealloc

Performance?

Performance não se adivinha, se mede.

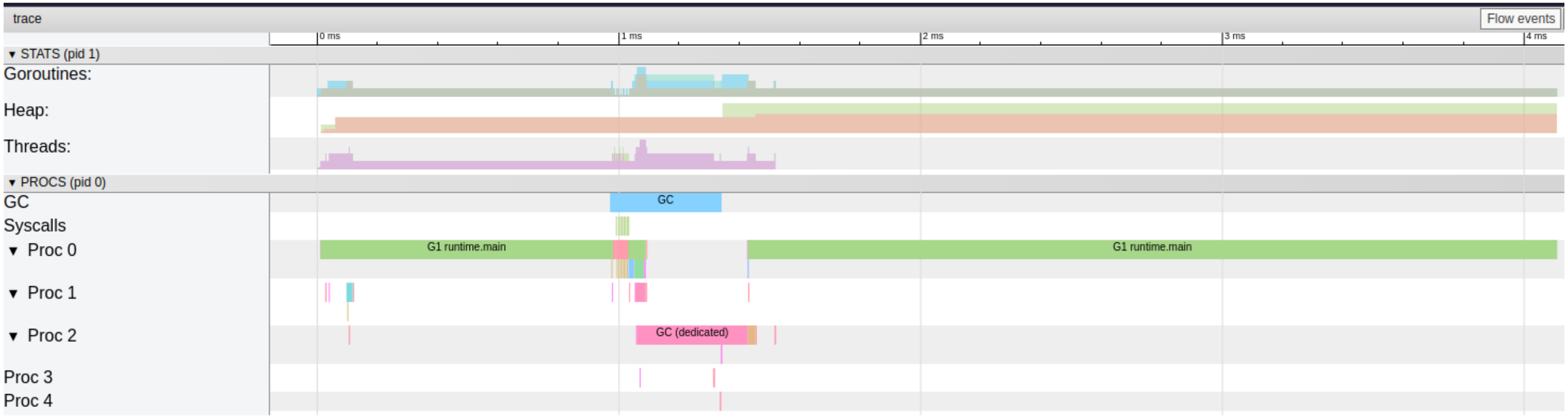
Vanilla Code

```
slice := make([]CustomStruct, 0)
for i := 0; i < 100_000; i++ {
    slice = append(slice, CustomStruct{
        Name: "Alex",
        Age:  123,
    })
}
```

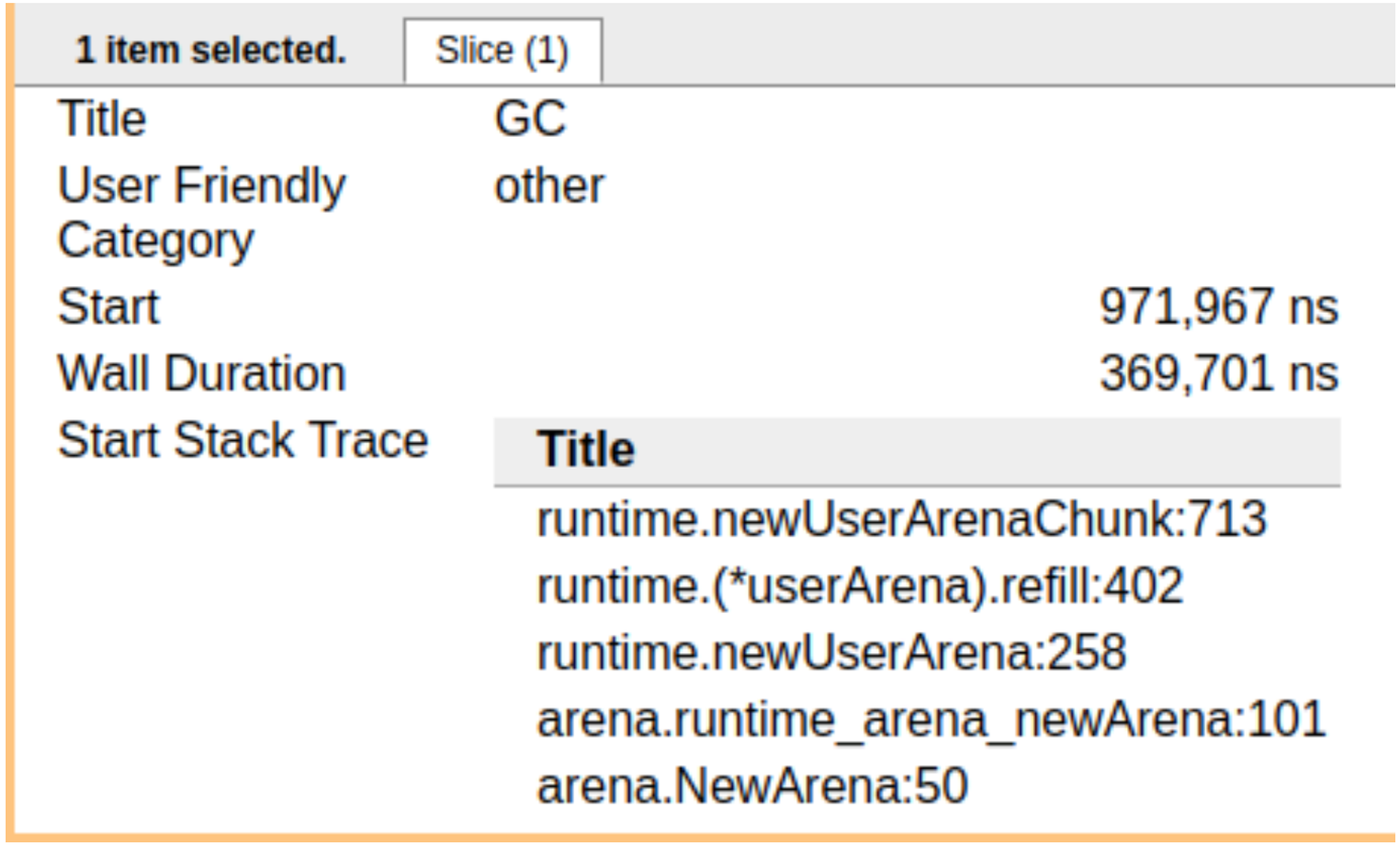
Arena Code

```
mem := arena.NewArena()
defer mem.Free()

slice := arena.MakeSlice[CustomStruct](mem, 100_000, 100_000)

for i := 0; i < 100_000; i++ {
    c := arena.New[CustomStruct](mem)
    c.Name = "Alex"
    c.Age = 123
    slice[i] = *c
}
```

Arena


trace
Flow events
0 ms
1 ms
2 ms
3 ms
4 ms
▾ STATS (pid 1)
Goroutines:
Heap:
Threads:
▾ PROCS (pid 0)
GC
Syscalls
▾ Proc 0
▾ Proc 1
▾ Proc 2
Proc 3
Proc 4
GC
G1 runtime.main
G1 runtime.main
GC (dedicated)

Arena


1 item selected.
Slice (1)
Title
GC
User Friendly Category
other
Start
971,967 ns
Wall Duration
369,701 ns
Start Stack Trace
Title
runtime.newUserArenaChunk:713
runtime.(*userArena).refill:402
runtime.newUserArena:258
arena.runtime_arena_newArena:101
arena.NewArena:50

Trace - Vanilla

Trace - Vanilla

4 items selected. Slices (4)

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
GC	2,763,703 ns	2,763,703 ns	690,926 ns	4

Selection start	622,416 ns
Selection extent	3,719,647 ns

Benchmark

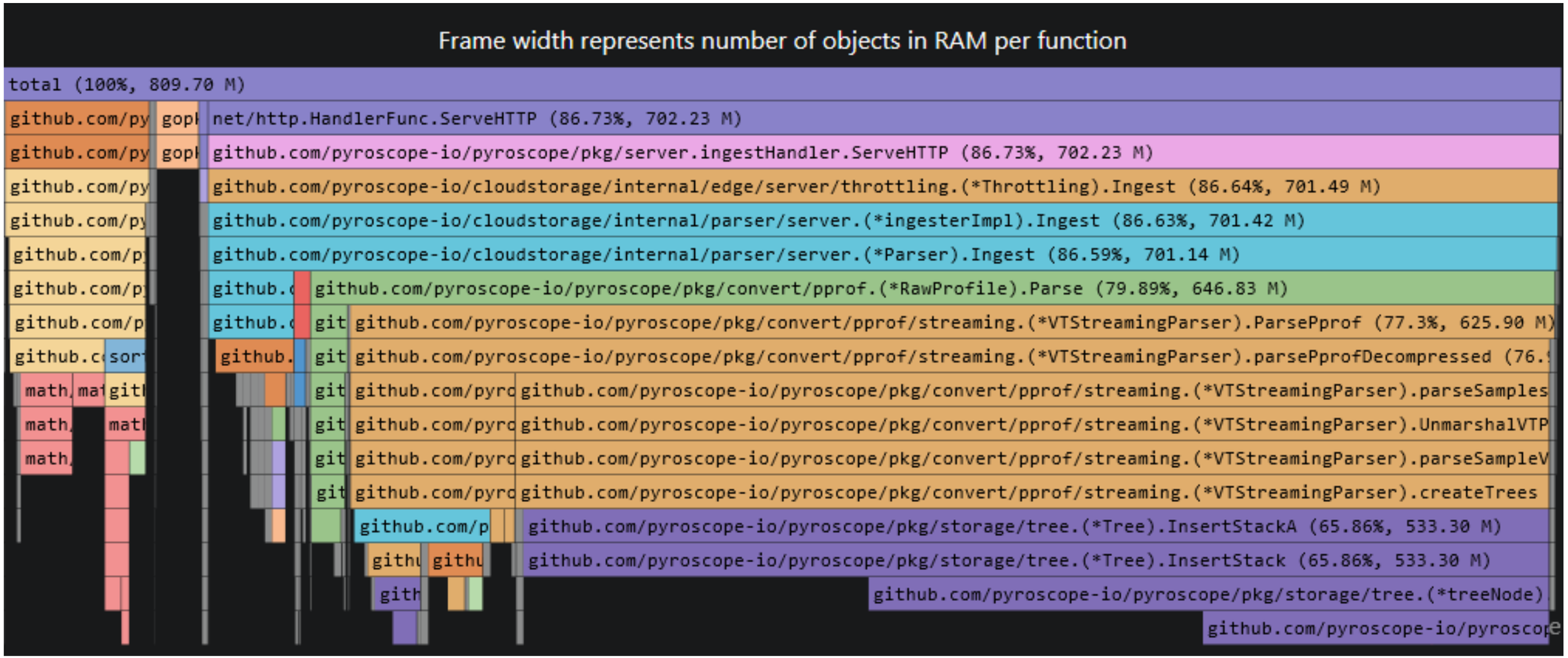
```
BenchmarkVanilla-8          196    6240066 ns/op 14077976 B/op        29 allocs/op
BenchmarkArena-8            487    2433707 ns/op 4811826 B/op          3 allocs/op
```

Benchmark

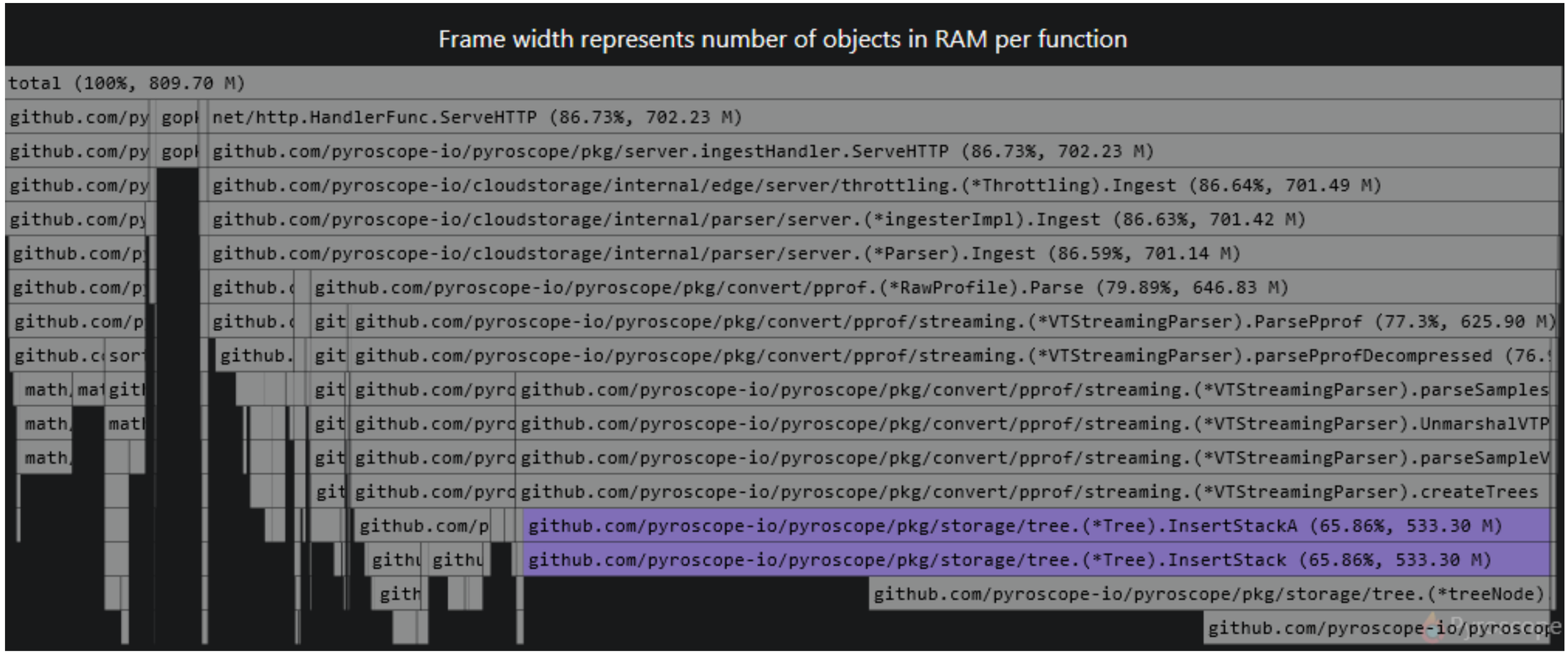
Nome	Iterações	ns/op	Byte/op	Alocações
Vanilla	196	6 240 066	14 077 976	29
Arena	487	2 433 707	4 811 826	3

4 – Case Study: Pyroscope

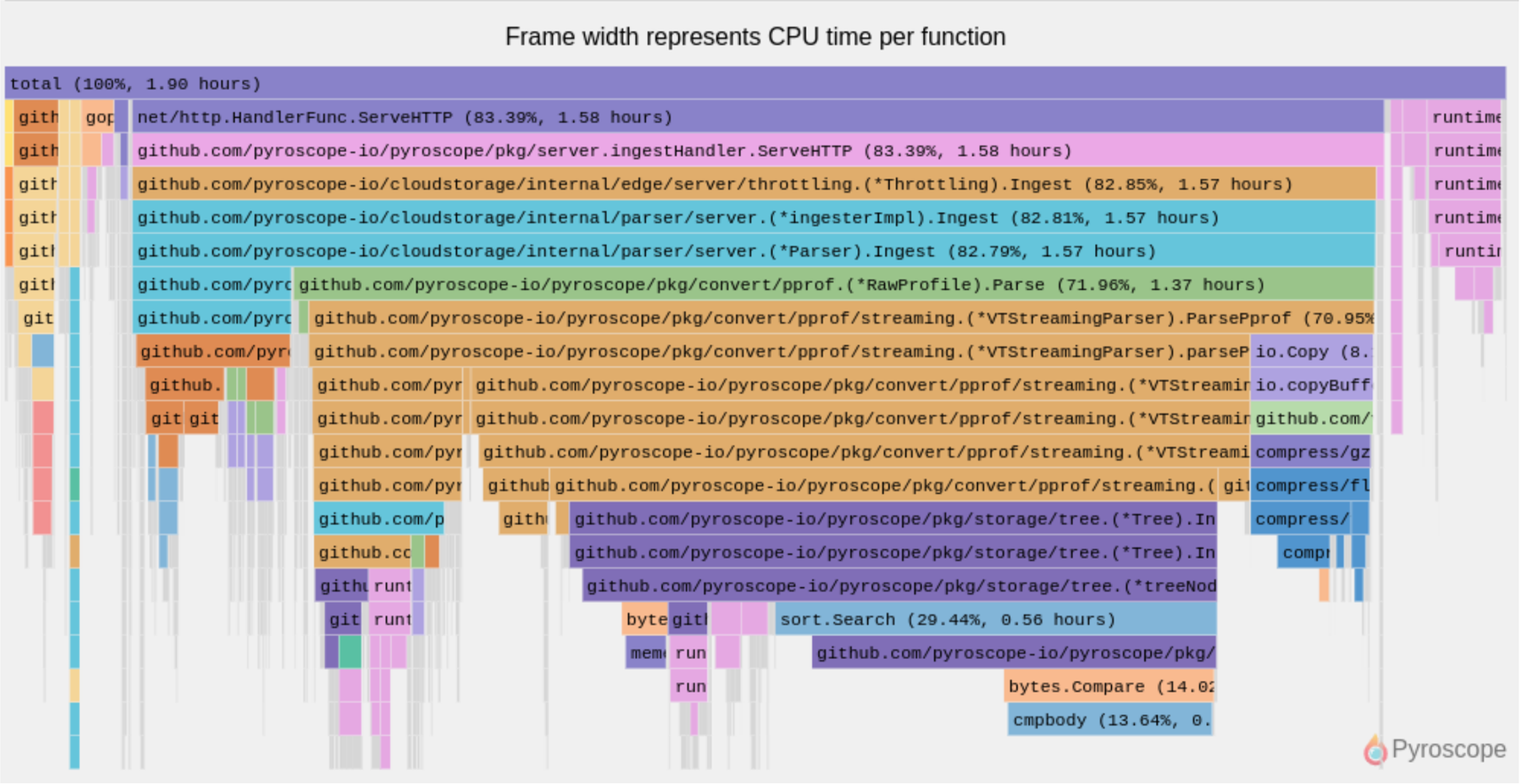
Allocations profile

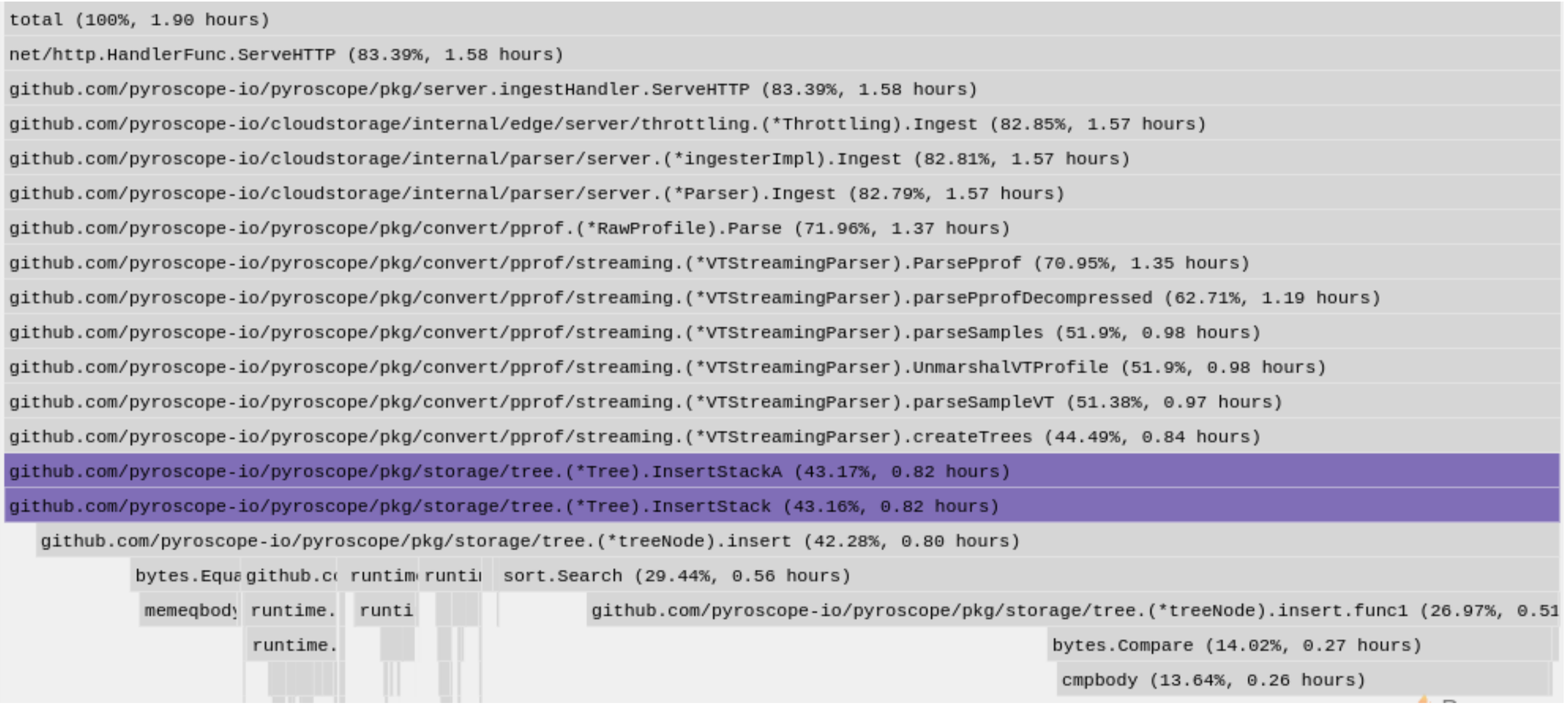


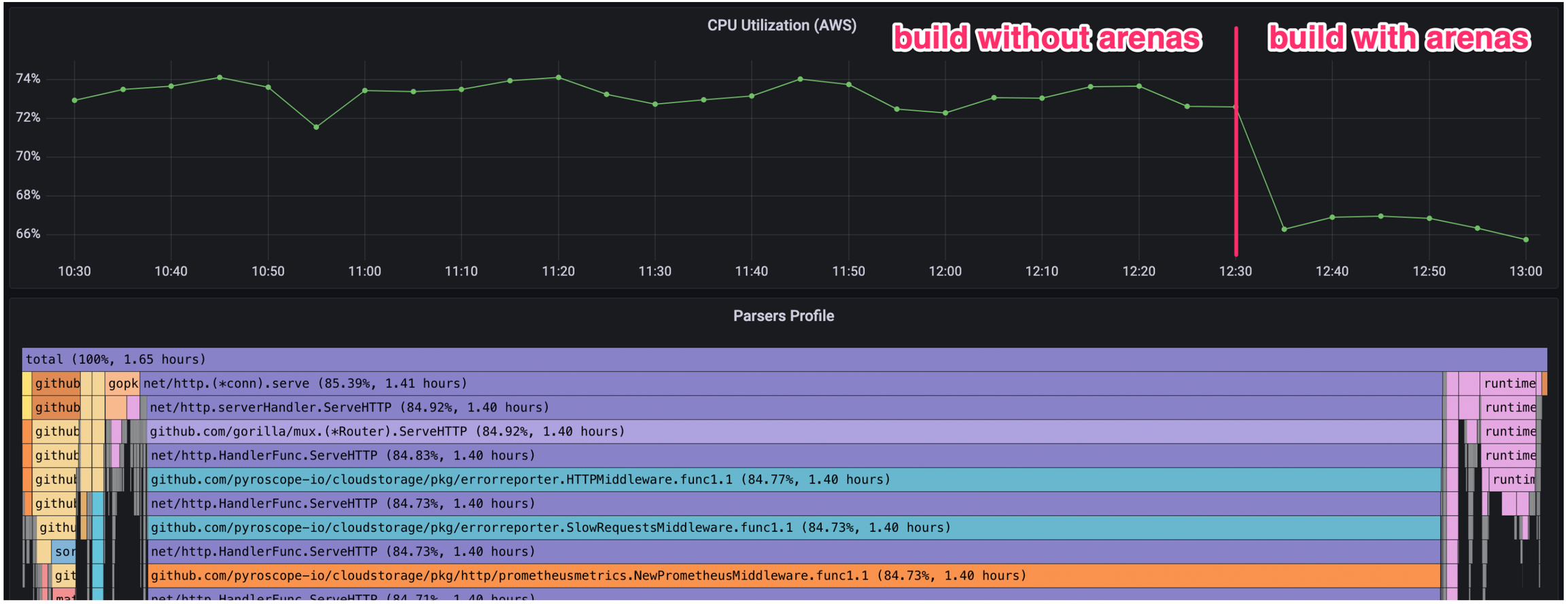
Allocations profile



CPU


Frame width represents CPU time per function
total (100%, 1.90 hours)
net/http.HandlerFunc.ServeHTTP (83.39%, 1.58 hours)
github.com/pyroscope-io/pyroscope/pkg/server.ingestHandler.ServeHTTP (83.39%, 1.58 hours)
github.com/pyroscope-io/cloudstorage/internal/edge/server/throttling.(*Throttling).Ingest (82.85%, 1.57 hours)
github.com/pyroscope-io/cloudstorage/internal/parser/server.(*ingesterImpl).Ingest (82.81%, 1.57 hours)
github.com/pyroscope-io/cloudstorage/internal/parser/server.(*Parser).Ingest (82.79%, 1.57 hours)
github.com/pyroscope-io/pyroscope/pkg/convert/pprof.(*RawProfile).Parse (71.96%, 1.37 hours)
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreamingParser).ParsePprof (70.95%
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreamingParser).parseP
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreamin
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreamin
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreami
github.com/pyroscope-io/pyroscope/pkg/storage/tree.(*Tree).In
github.com/pyroscope-io/pyroscope/pkg/storage/tree.(*Tree).In
github.com/pyroscope-io/pyroscope/pkg/storage/tree.(*treeNod
sort.Search (29.44%, 0.56 hours)
github.com/pyroscope-io/pyroscope/pkg/
bytes.Compare (14.02
cmpbody (13.64%, 0.
io.Copy (8.
io.copyBuff
compress/gz
compress/fl
compress/
runtime
Pyroscope


total (100%, 1.90 hours)
net/http.HandlerFunc.ServeHTTP (83.39%, 1.58 hours)
github.com/pyroscope-io/pyroscope/pkg/server.ingestHandler.ServeHTTP (83.39%, 1.58 hours)
github.com/pyroscope-io/cloudstorage/internal/edge/server/throttling.(*Throttling).Ingest (82.85%, 1.57 hours)
github.com/pyroscope-io/cloudstorage/internal/parser/server.(*ingesterImpl).Ingest (82.81%, 1.57 hours)
github.com/pyroscope-io/cloudstorage/internal/parser/server.(*Parser).Ingest (82.79%, 1.57 hours)
github.com/pyroscope-io/pyroscope/pkg/convert/pprof.(*RawProfile).Parse (71.96%, 1.37 hours)
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreamingParser).ParsePprof (70.95%, 1.35 hours)
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreamingParser).parsePprofDecompressed (62.71%, 1.19 hours)
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreamingParser).parseSamples (51.9%, 0.98 hours)
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreamingParser).UnmarshalVTProfile (51.9%, 0.98 hours)
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreamingParser).parseSampleVT (51.38%, 0.97 hours)
github.com/pyroscope-io/pyroscope/pkg/convert/pprof/streaming.(*VTStreamingParser).createTrees (44.49%, 0.84 hours)
github.com/pyroscope-io/pyroscope/pkg/storage/tree.(*Tree).InsertStackA (43.17%, 0.82 hours)
github.com/pyroscope-io/pyroscope/pkg/storage/tree.(*Tree).InsertStack (43.16%, 0.82 hours)
github.com/pyroscope-io/pyroscope/pkg/storage/tree.(*treeNode).insert (42.28%, 0.80 hours)
bytes.Equa
github.c
runtim
runti
sort.Search (29.44%, 0.56 hours)
memeqbody
runtime.
runti
github.com/pyroscope-io/pyroscope/pkg/storage/tree.(*treeNode).insert.func1 (26.97%, 0.51
runtime.
bytes.Compare (14.02%, 0.27 hours)
cmpbody (13.64%, 0.26 hours)


CPU Utilization (AWS)
build without arenas
build with arenas
74%
72%
70%
68%
66%
10:30
10:40
10:50
11:00
11:10
11:20
11:30
11:40
11:50
12:00
12:10
12:20
12:30
12:40
12:50
13:00
Parsers Profile
total (100%, 1.65 hours)
net/http.(*conn).serve (85.39%, 1.41 hours)
net/http.serverHandler.ServeHTTP (84.92%, 1.40 hours)
github.com/gorilla/mux.(*Router).ServeHTTP (84.92%, 1.40 hours)
net/http.HandlerFunc.ServeHTTP (84.83%, 1.40 hours)
github.com/pyroscope-io/cloudstorage/pkg/errorreporter.HTTPMiddleware.func1.1 (84.77%, 1.40 hours)
net/http.HandlerFunc.ServeHTTP (84.73%, 1.40 hours)
github.com/pyroscope-io/cloudstorage/pkg/errorreporter.SlowRequestsMiddleware.func1.1 (84.73%, 1.40 hours)
net/http.HandlerFunc.ServeHTTP (84.73%, 1.40 hours)
github.com/pyroscope-io/cloudstorage/pkg/http/prometheusmetrics.NewPrometheusMiddleware.func1.1 (84.73%, 1.40 hours)

Oportunidades

gRPC

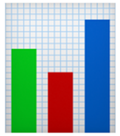
Em gRPC existem muitos pequenos objetos sendo criados durante o encoding e decoding das mensagens.

`https://protobuf.dev/reference/cpp/arenas/`

JSON decode/encode

Arena artesanal: valyala / **fastjson** (15x~ faster than stdlib)

5 – Guidelines

Data driven? 

Não? Você ta chutando.

Several Allocations? 

Não? Você provavelmente vai usar mais memória para fazer a mesma coisa.

Same small structure?

yes? sync.Pool

Hot Path? 

Não? Otimização prematura.

Vale a pena usar?

Vale demais! porém...

- Não é trivial.
- Experimente combinações de GOGC e GOMEMLIMIT
- Talvez você nunca precise usar. (e ta tudo bem)
- Sua codebase que lida com a arena é muito delicada (leia-se possíveis crashes e memory leaks)

🚨 Experimental API 🚨

- Pode desaparecer
- Ser descontinuada
- Gerar retrabalho pelas mudanças de API

🙌 Obrigado! 🙌

x.com/alextrending