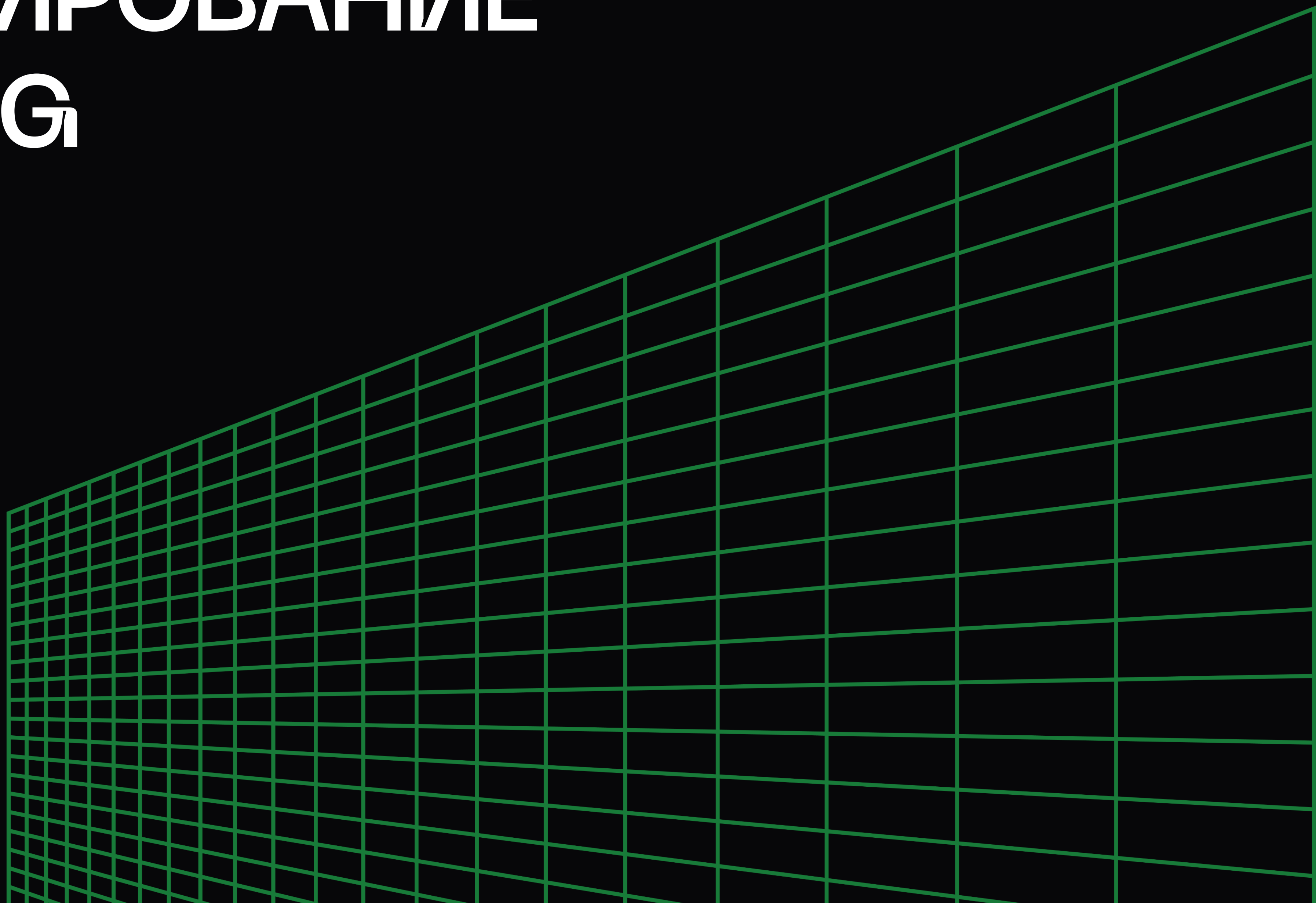


ПРОФИЛИРОВАНИЕ В GOLANG



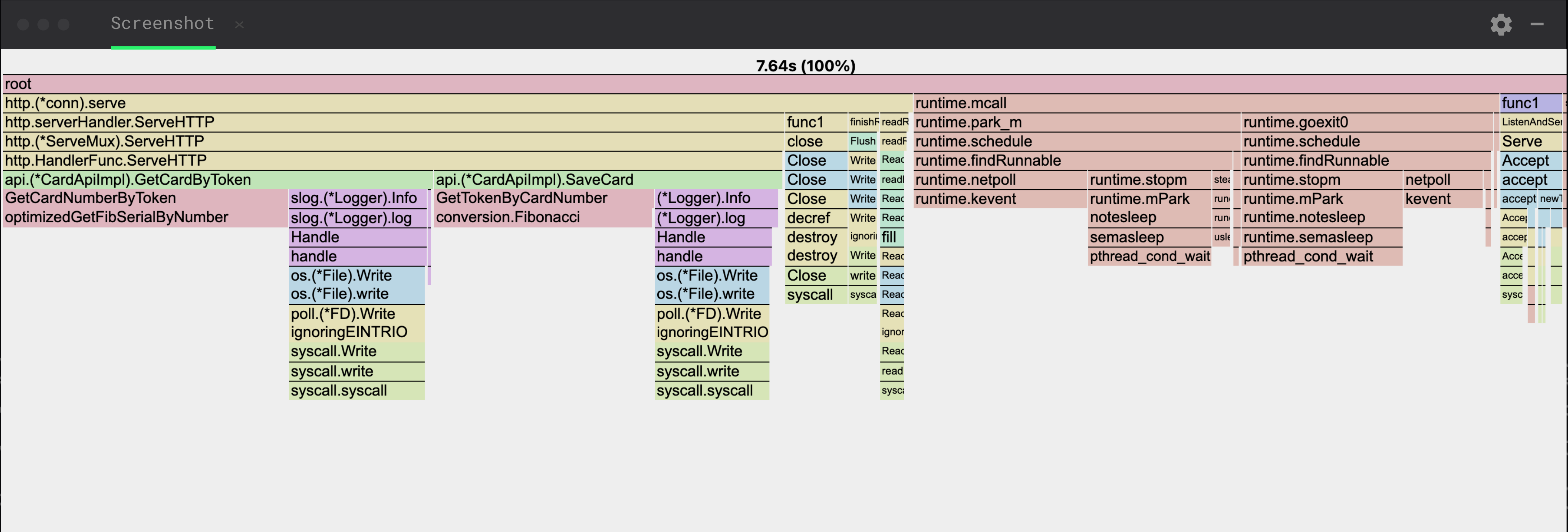
ПЛАН ИНТЕНСИВА

1. Определение и мотивация профилирования
2. Виды профилировщиков
3. Архитектура профилировщика Go
4. Использование профилировщика Go
5. Trace profiler
6. PGO
7. Continuous profiling
8. Примеры профилирования

ОПРЕДЕЛЕНИЕ И МОТИВАЦИЯ ПРОФИЛИРОВАНИЯ

ПРОФИЛИРОВАНИЕ

Динамический сбор метрик программного обеспечения



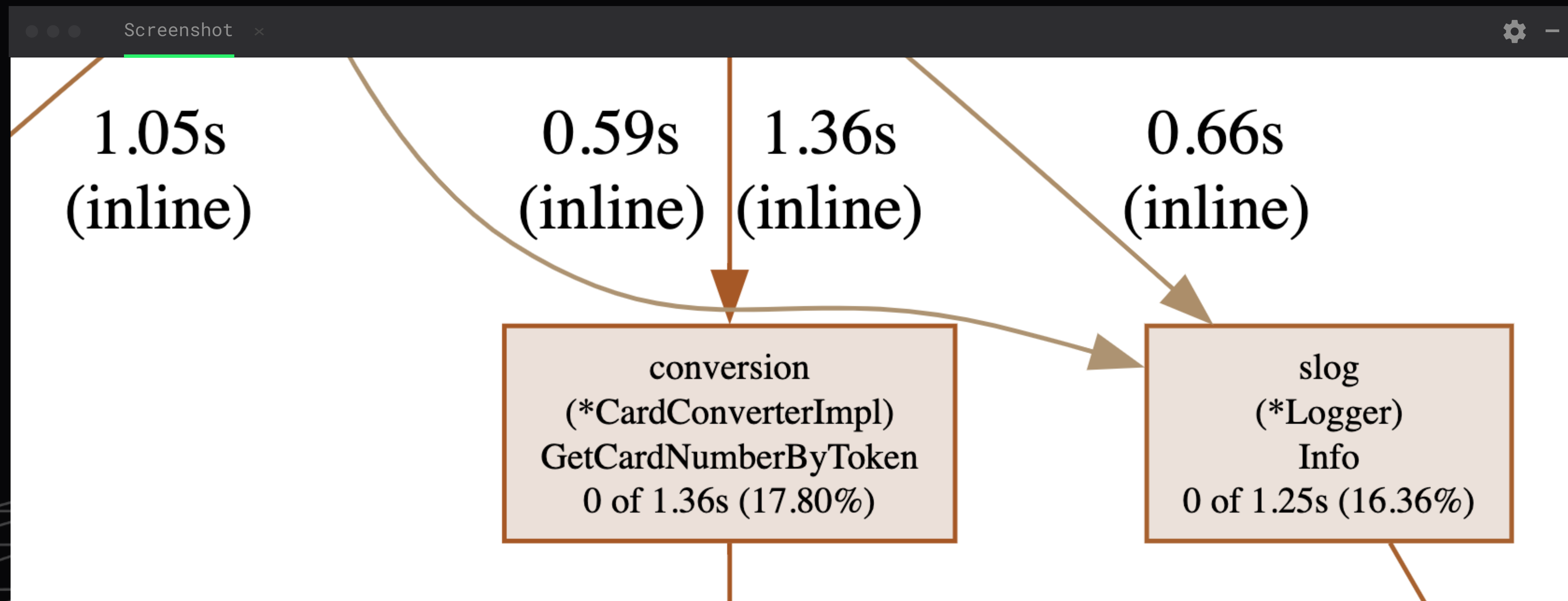
МОТИВАЦИЯ

Выявление узких мест и увеличение масштабируемости



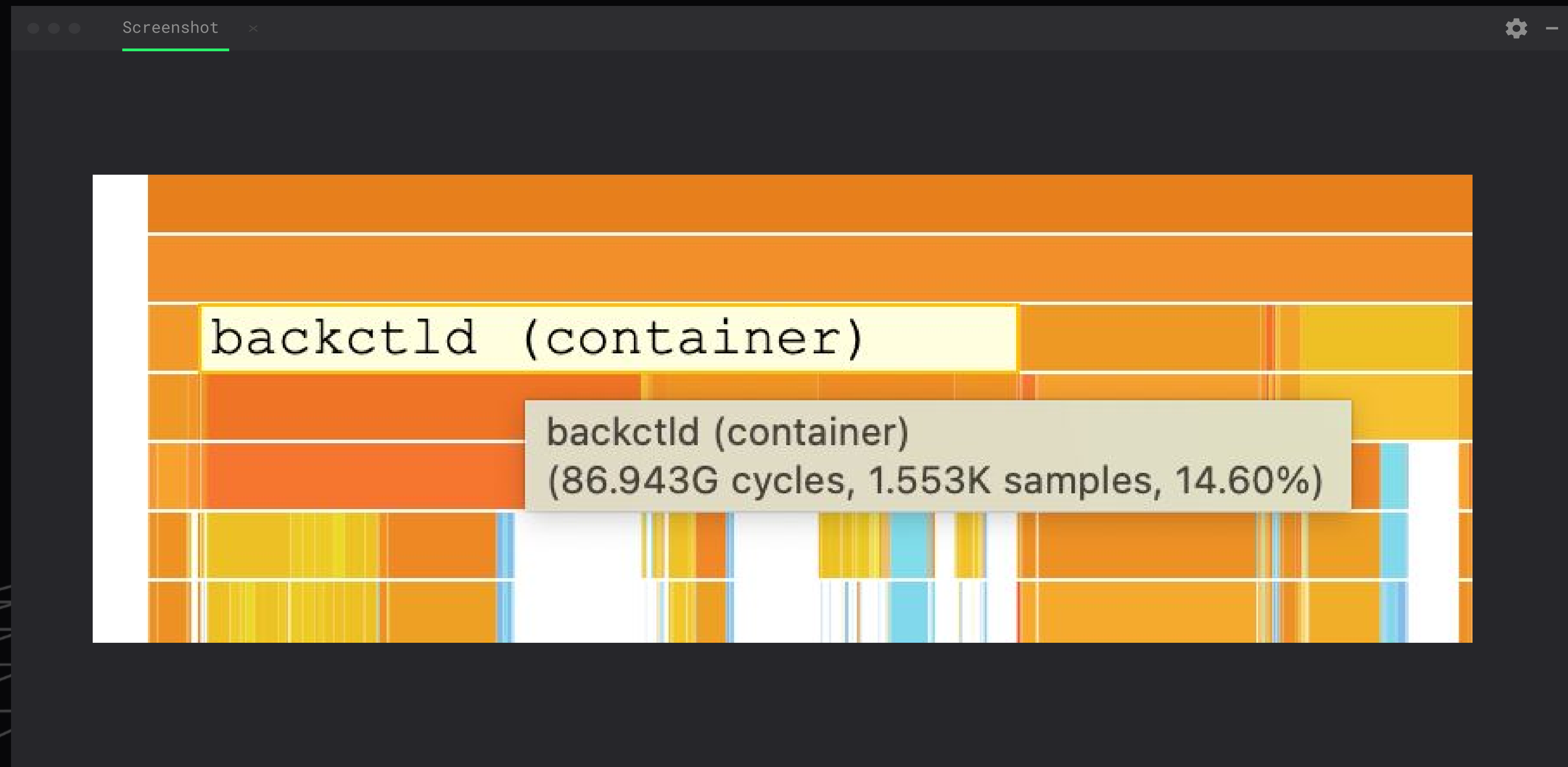
МОТИВАЦИЯ

Повышение производительности, используя данные исполнения при компиляции



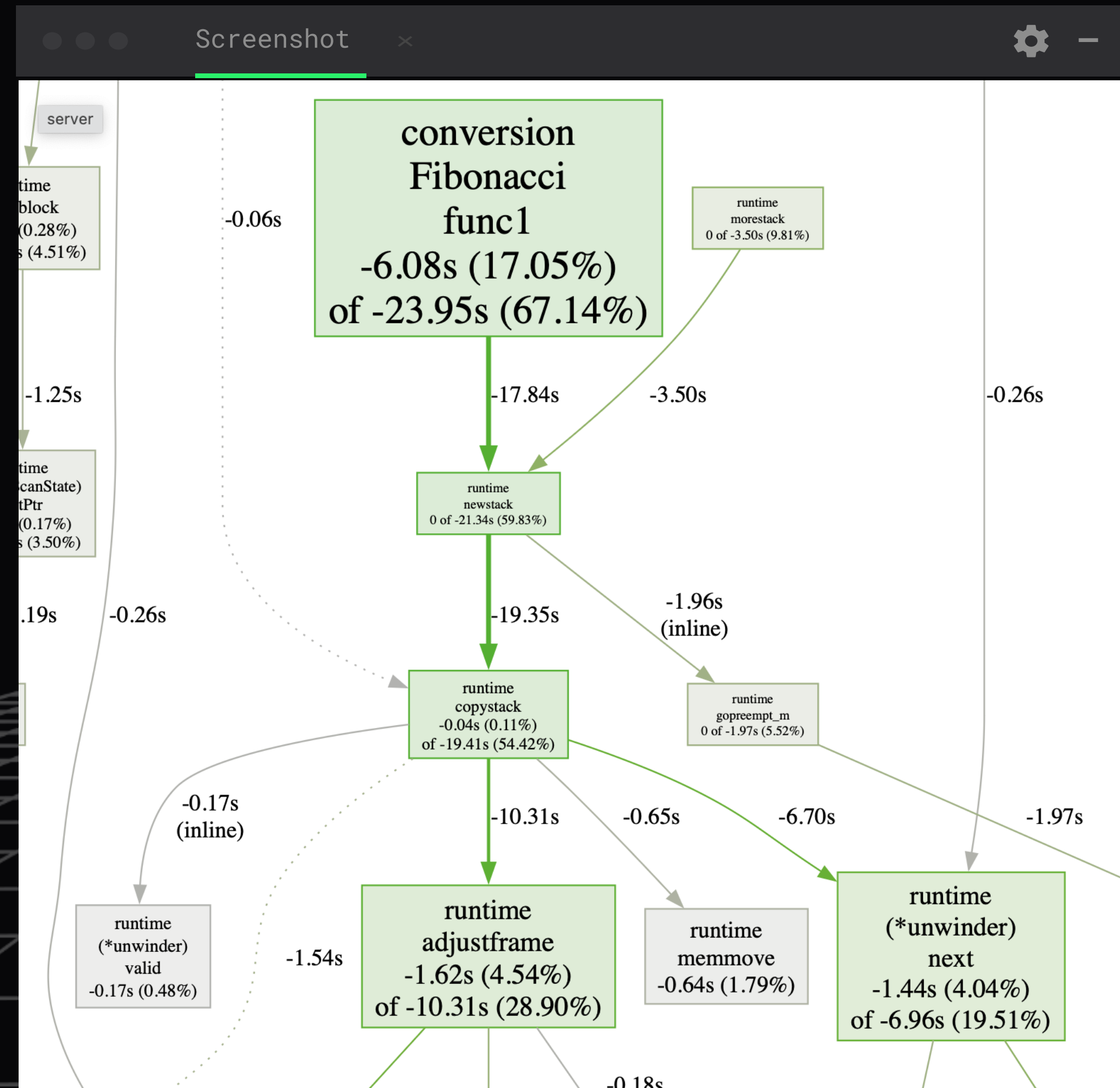
МОТИВАЦИЯ

Утилизация ресурсов того или иного компонента



МОТИВАЦИЯ

Регрессионное тестирование



ВИДЫ ПРОФИЛИРОВЩИКОВ

ИНСТРУМЕНТАЦИЯ (STRUCTED, TRACING)

1. Ручная инструментация
2. Source инструментация
3. Intermediate инструментация

```
63  
64     <-ctx.Done()                                // data[64]++  
65     shutdownCtx, cancel := context.WithTimeout(context.Background(), time.Minute) // data[65]++  
66     defer cancel()                             // data[66]++  
67
```

ИНСТРУМЕНТАЦИЯ (STRUCTED, TRACING)

1. Ручная instrumentation
2. Source instrumentation
3. Intermediate instrumentation
4. Binary instrumentation
5. Runtime instrumentation
(hypervisor, simulator,
interpreter)

ИНСТРУМЕНТАЦИЯ (STRUCTED, TRACING)

Плюсы:

- Позволяет получить
наиболее точную картину

ИНСТРУМЕНТАЦИЯ (STRUCTED, TRACING)

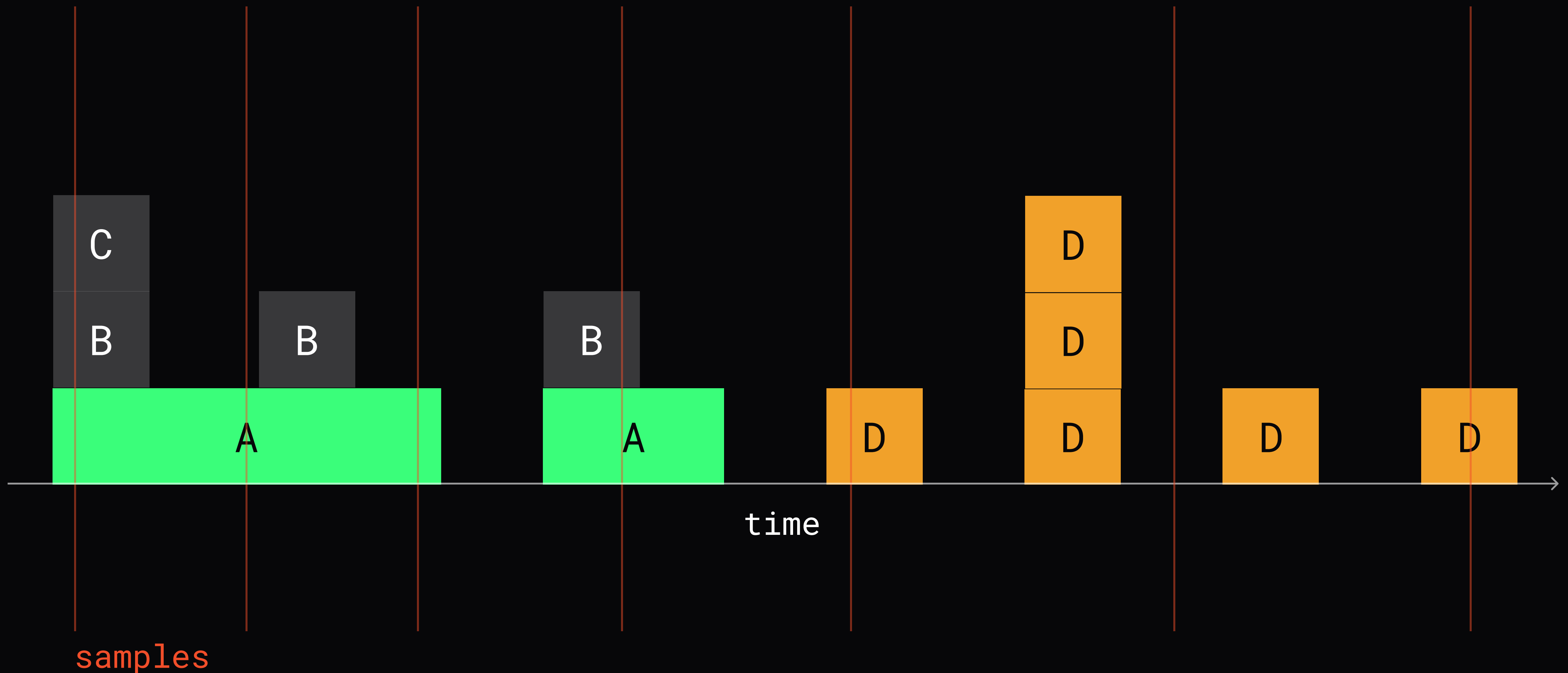
Плюсы:

- Позволяет получить наиболее точную картину

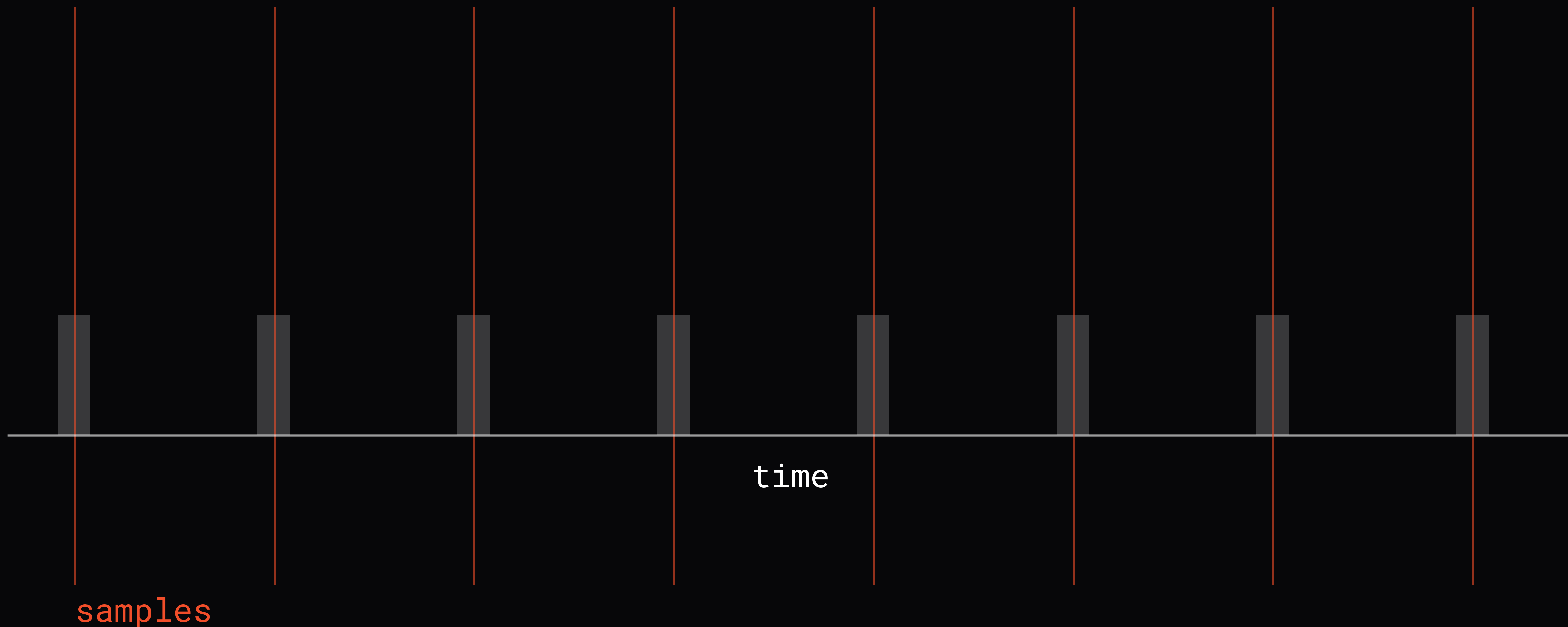
Минусы:

- Из-за агрессивной модификации исходной программы сильно влияет на производительность

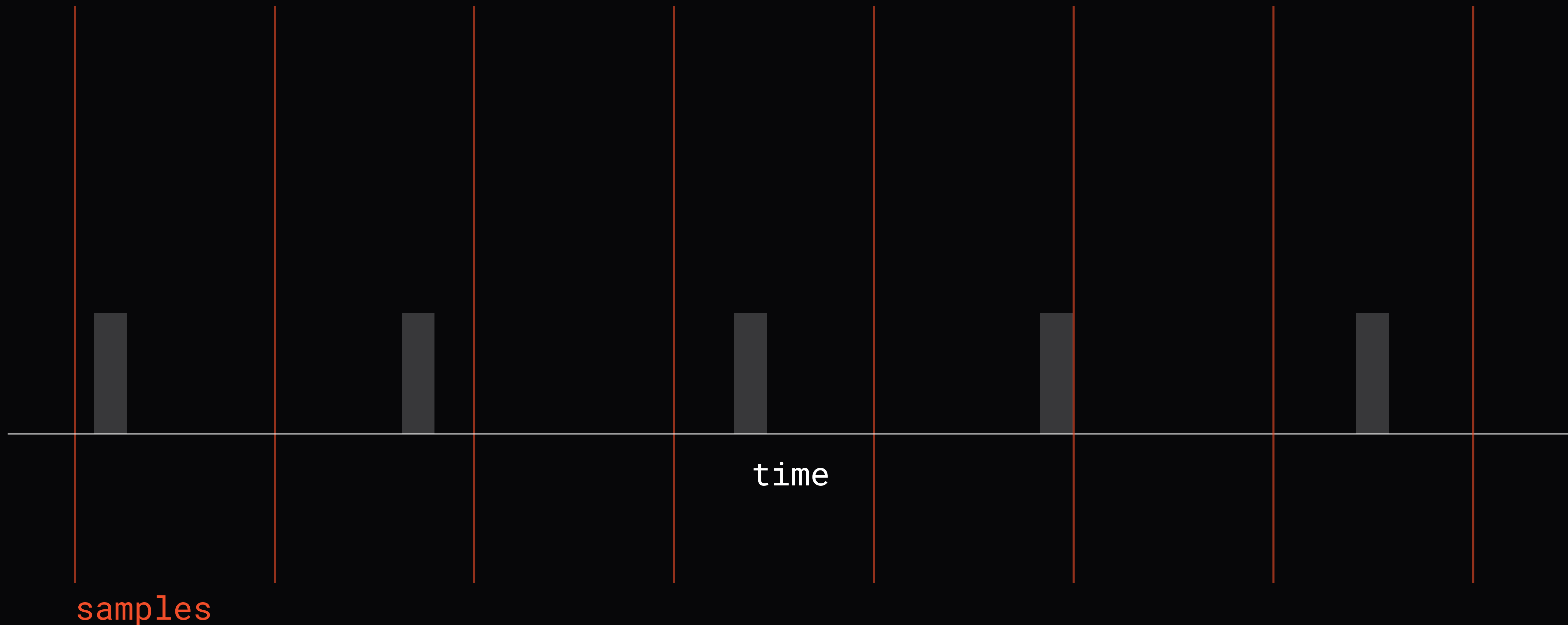
СЭМПЛИРОВАНИЕ



СЭМПЛИРОВАНИЕ



СЭМПЛИРОВАНИЕ



СЭМПЛИРОВАНИЕ

Плюсы:

- Имеет сравнительно
маленькие накладные расходы

СЭМПЛИРОВАНИЕ

Плюсы:

- Имеет сравнительно
маленькие накладные расходы

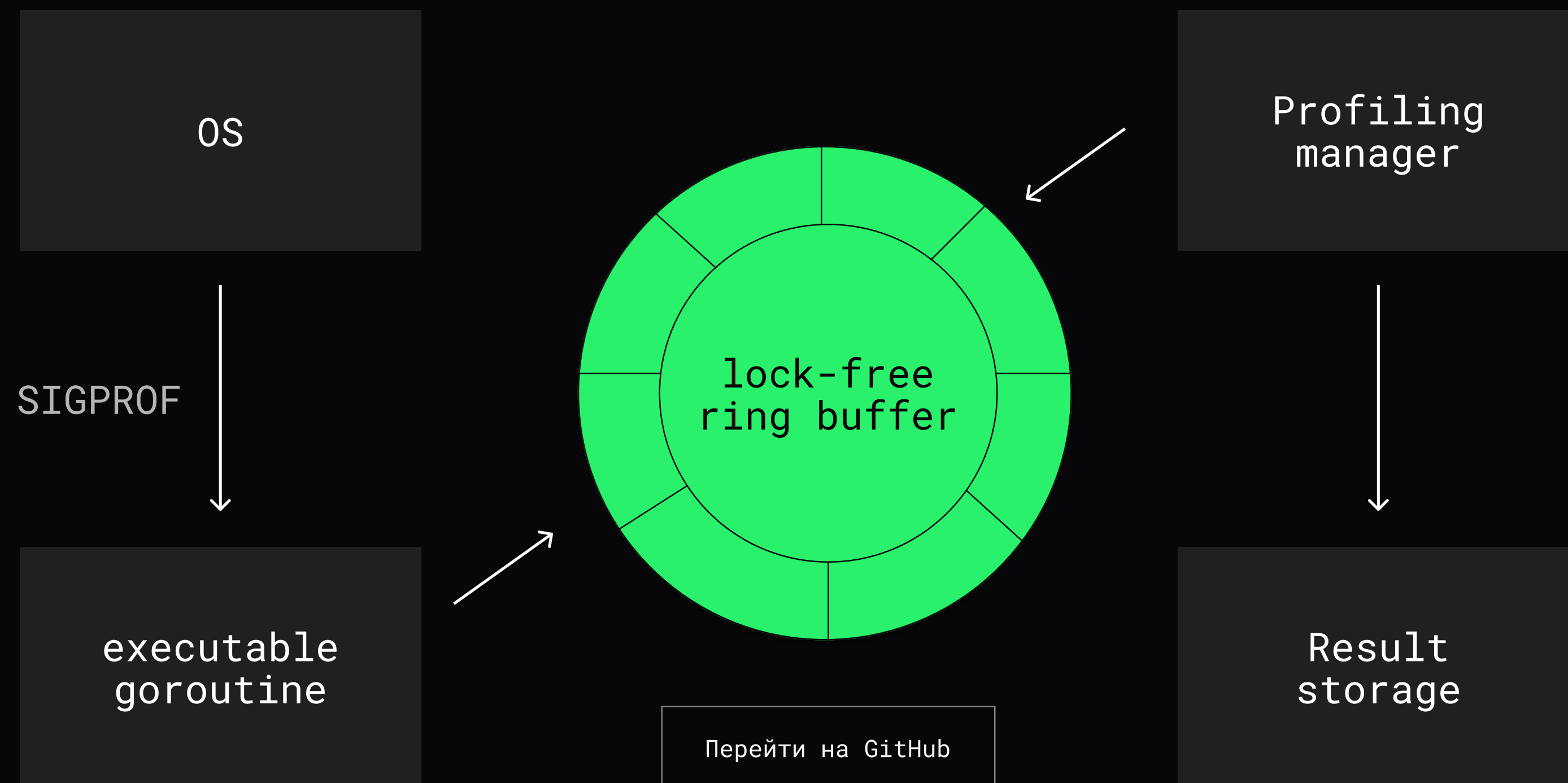
Минусы:

- В зависимости от настройки может
показывать нерепрезентативные результаты

АРХИТЕКТУРА ПРОФИЛИРОВЩИКА GO

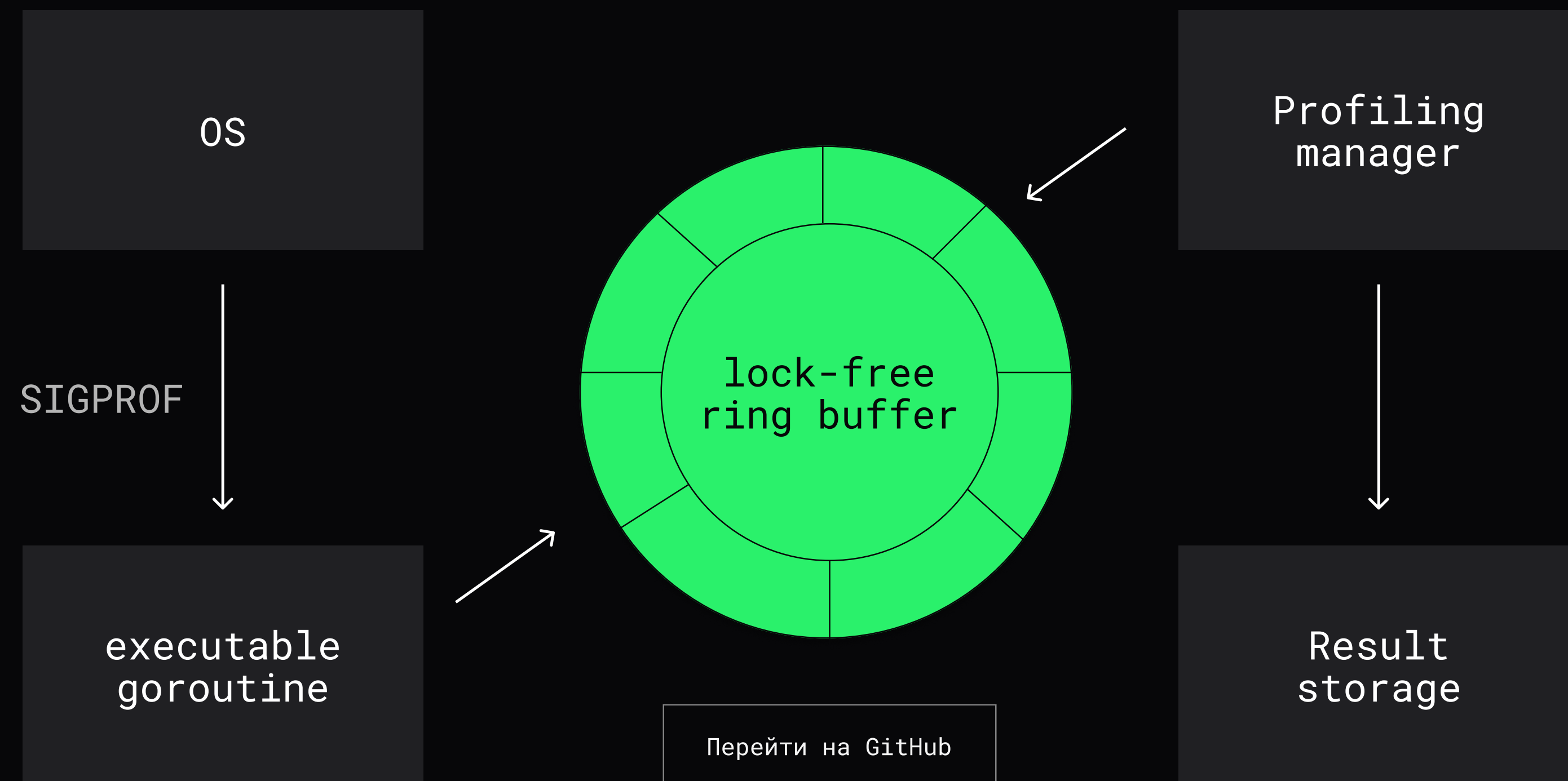
АРХИТЕКТУРА ПРОФИЛИРОВЩИКА

```
// A profBuf is a lock-free buffer for profiling events,  
// safe for concurrent use by one reader and one writer.  
// The writer may be a signal handler running without a user g.  
// The reader is assumed to be a user g.
```



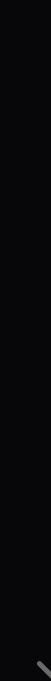
АРХИТЕКТУРА ПРОФИЛИРОВЩИКА

- `cpu`
- `block/mutex`
- `allocs/heap`
- `trace`
- `goroutine`



АРХИТЕКТУРА ПРОФИЛИРОВЩИКА

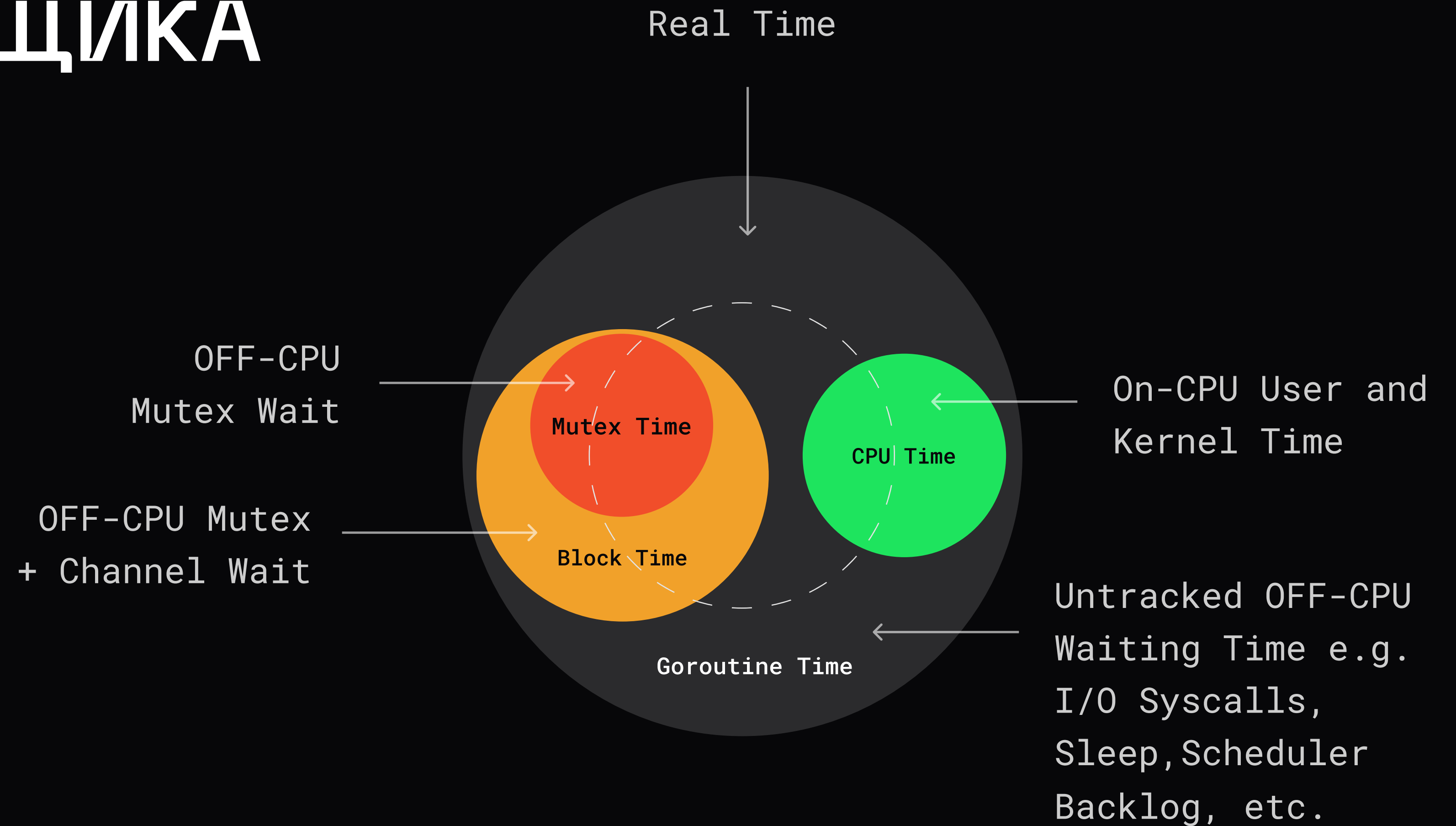
```
Terminal: Optimization × + ▾ ⚙ -  
message Profile {  
    // A description of the samples associated with each Sample.value.  
    // For a cpu profile this might be:  
    //    [["cpu","nanoseconds"]] or [["wall","seconds"]] or [["syscall","count"]]  
    // For a heap profile, this might be:  
    //    [["allocations","count"], ["space","bytes"]],  
    // If one of the values represents the number of events represented  
    // by the sample, by convention it should be at index 0 and use  
    // sample_type.unit == "count".  
    repeated ValueType sample_type = 1;  
    // The set of samples recorded in this profile.  
    repeated Sample sample = 2;  
    // Mapping from address ranges to the image/binary/library mapped  
    // into that address range.  mapping[0] will be the main binary.  
    repeated Mapping mapping = 3;  
    // Locations referenced by samples.  
    repeated Location location = 4;  
    // Functions referenced by locations.  
    repeated Function function = 5;  
    // A common table for strings referenced by various messages.  
    // string_table[0] must always be "".  
    repeated string string_table = 6;  
    // frames with Function.function_name fully matching the following  
    // regexp will be dropped from the samples, along with their successors.  
    int64 drop_frames = 7;    // Index into string table.  
    // frames with Function.function_name fully matching the following  
    // regexp will be kept, even if it matches drop_frames.  
    int64 keep_frames = 8;    // Index into string table.
```



Result
storage

АРХИТЕКТУРА ПРОФИЛИРОВЩИКА

- cpu
- block/mutex
- allocs/heap
- trace
- goroutine



ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА GO

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

Не рекомендуется собирать профили с помощью пакета `runtime`

```
runtime.GoroutineProfile()  
runtime.MutexProfile()  
runtime.MemProfile()  
runtime.CPUProfile()
```

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

```
err = pprof.Lookup(name: "heap").WriteTo(f, debug: 0)
err = pprof.WriteHeapProfile(f)

err = pprof.StartCPUProfile(f)
defer pprof.StopCPUProfile()
```

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

```
func lockProfiles() {  
    profiles.mu.Lock()  
    if profiles.m == nil {  
        // Initial built-in profiles.  
        profiles.m = map[string]*Profile{  
            "goroutine":    goroutineProfile,  
            "threadcreate": threadcreateProfile,  
            "heap":          heapProfile,  
            "allocs":        allocsProfile,  
            "block":         blockProfile,  
            "mutex":         mutexProfile,  
        }  
    }  
}
```

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

```
func registerProfilerEndpoints(mux *http.ServeMux) { 1 usage  👤 Igor Walther
    mux.HandleFunc(🌐"/debug/pprof/", pprof.Index)
    mux.HandleFunc(🌐"/debug/pprof/cmdline", pprof.Cmdline)
    mux.HandleFunc(🌐"/debug/pprof/profile", pprof.Profile)
    mux.HandleFunc(🌐"/debug/pprof/symbol", pprof.Symbol)
    mux.HandleFunc(🌐"/debug/pprof/trace", pprof.Trace)
    mux.Handle(🌐"/debug/pprof/block", pprof.Handler(name: "block"))
    mux.Handle(🌐"/debug/pprof/heap", pprof.Handler(name: "heap"))
    mux.Handle(🌐"/debug/pprof/allocs", pprof.Handler(name: "allocs"))
    mux.Handle(🌐"/debug/pprof/goroutine", pprof.Handler(name: "goroutine"))
    mux.Handle(🌐"/debug/pprof/threadcreate", pprof.Handler(name: "goroutine"))
}
```

PROFILE RATE

```
runtime.SetBlockProfileRate(rate: 1)
runtime.SetMutexProfileFraction(rate: 1)
runtime.SetCPUProfileRate(hz: 1)
runtime.MemProfileRate = 1
```

Из-за накладных расходов некоторые профили по умолчанию отключены

[Перейти на GitHub](#)

```
if rate := MemProfileRate; rate > 0 {
    // Note cache c only valid while m acquired; see #47302
    //
    // N.B. Use the full size because that matches how the GC
    // will update the mem profile on the "free" side.
    if rate != 1 && fullSize < c.nextSample {
        c.nextSample -= fullSize
    } else {
        profilealloc(mp, x, fullSize)
    }
}
mp.mallocing = 0
releasem(mp)
```

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

```
# ----- PROFILING -----  
.PHONY: cpuprof  
cpuprof:  
    (PPROF_TMPDIR=${PPROFDIR} go tool pprof -http :8081 -seconds 20 http://127.0.0.1:8080/debug/pprof/profile)  
  
.PHONY: memprof  
memprof:  
    (PPROF_TMPDIR=${PPROFDIR} go tool pprof -http :8081 http://127.0.0.1:8080/debug/pprof/heap)  
  
.PHONY: traceprof  
traceprof:  
    curl http://localhost:8080/debug/pprof/trace?seconds=20 -o trace.out  
# ----- PROFILING -----
```

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

```
go tool pprof pprof.___6348go_build_card_shielder_go.samples.cpu.001.pb.gz
File: ___6348go_build_card_shielder_go
Type: cpu
Time: Sep 21, 2024 at 12:42pm (MSK)
Duration: 20.13s, Total samples = 7.64s (37.96%)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof)
```

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

```
(pprof) top
Showing nodes accounting for 82.08s, 70.16% of 116.99s total
Dropped 208 nodes (cum <= 0.58s)
Showing top 10 nodes out of 101
```

flat	flat%	sum%	cum	cum%	
27.26s	23.30%	23.30%	84.59s	72.31%	card_shielder/internal/core/conversion.Fibonacci.func1
9.21s	7.87%	31.17%	21.93s	18.75%	runtime.(*stkframe).getStackMap
9s	7.69%	38.87%	9s	7.69%	runtime.adjustpointers
8.41s	7.19%	46.06%	10.46s	8.94%	runtime.pcvalue
5.02s	4.29%	50.35%	5.02s	4.29%	runtime.memmove
5.02s	4.29%	54.64%	5.02s	4.29%	runtime.pthread_kill
4.90s	4.19%	58.83%	9.86s	8.43%	runtime.(*unwinder).resolveInternal
4.67s	3.99%	62.82%	4.69s	4.01%	runtime.pthread_cond_signal
4.51s	3.86%	66.67%	4.51s	3.86%	runtime.usleep
4.08s	3.49%	70.16%	19.19s	16.40%	runtime.(*unwinder).next

```
(pprof)
```

FLAT & CUM

```
func root() { no usages  new *  
    f1()  
    f2()  
    f3()  
}
```

```
func f1() {} 1 usage  new *
```

```
func f2() {} 1 usage  new *
```

```
func f3() {} 1 usage  new *
```

flat = root

cum = root + f1 + f2 + f3

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

```
(pprof) top 20 -cum
Showing nodes accounting for 69.33s, 59.26% of 116.99s total
Dropped 208 nodes (cum <= 0.58s)
Showing top 20 nodes out of 101
```

	flat	flat%	sum%		cum	cum%	
	27.26s	23.30%	23.30%		84.59s	72.31%	card_shielder/internal/core/conversion.Fibonacci.func1
	0	0%	23.30%		63.46s	54.24%	runtime.newstack
	0.10s	0.085%	23.39%		57.98s	49.56%	runtime.copystack
	3.91s	3.34%	26.73%		34.27s	29.29%	runtime.adjustframe
	9.21s	7.87%	34.60%		21.93s	18.75%	runtime.(*stkframe).getStackMap
	0	0%	34.60%		19.35s	16.54%	runtime.systemstack
	4.08s	3.49%	38.09%		19.19s	16.40%	runtime.(*unwinder).next
	0	0%	38.09%		12.29s	10.51%	runtime.gcBgMarkWorker.func2
	0	0%	38.09%		12.29s	10.51%	runtime.gcDrain
	0	0%	38.09%		12.03s	10.28%	runtime.markroot
	0.01s	0.0085%	38.10%		12.02s	10.27%	runtime.markroot.func1

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

```
(pprof) peek Fibonacci
Showing nodes accounting for 116.99s, 100% of 116.99s total
-----+-----
      flat  flat%   sum%        cum   cum%   calls calls% + context
-----+-----
      27.26s 23.30% 23.30%      84.59s 72.31%      0.01s 0.012% | card_shielder/internal/core/conversion.Fibonacci
      57.24s 67.67% | card_shielder/internal/core/conversion.Fibonacci.func1
      0.09s 0.11% | runtime.newstack
      0.09s 0.11% | runtime.asyncPreempt
-----+-----
      0      0% 23.30%      0.45s 0.38%      0.45s 100% | card_shielder/internal/core/conversion.(*CardConverterImpl).GetTokenByCardNumber
      0.44s 97.78% | card_shielder/internal/core/conversion.Fibonacci
      0.01s 2.22% | runtime.makeslice
      0.01s 2.22% | card_shielder/internal/core/conversion.Fibonacci.func1
-----+-----
(pprof)
```

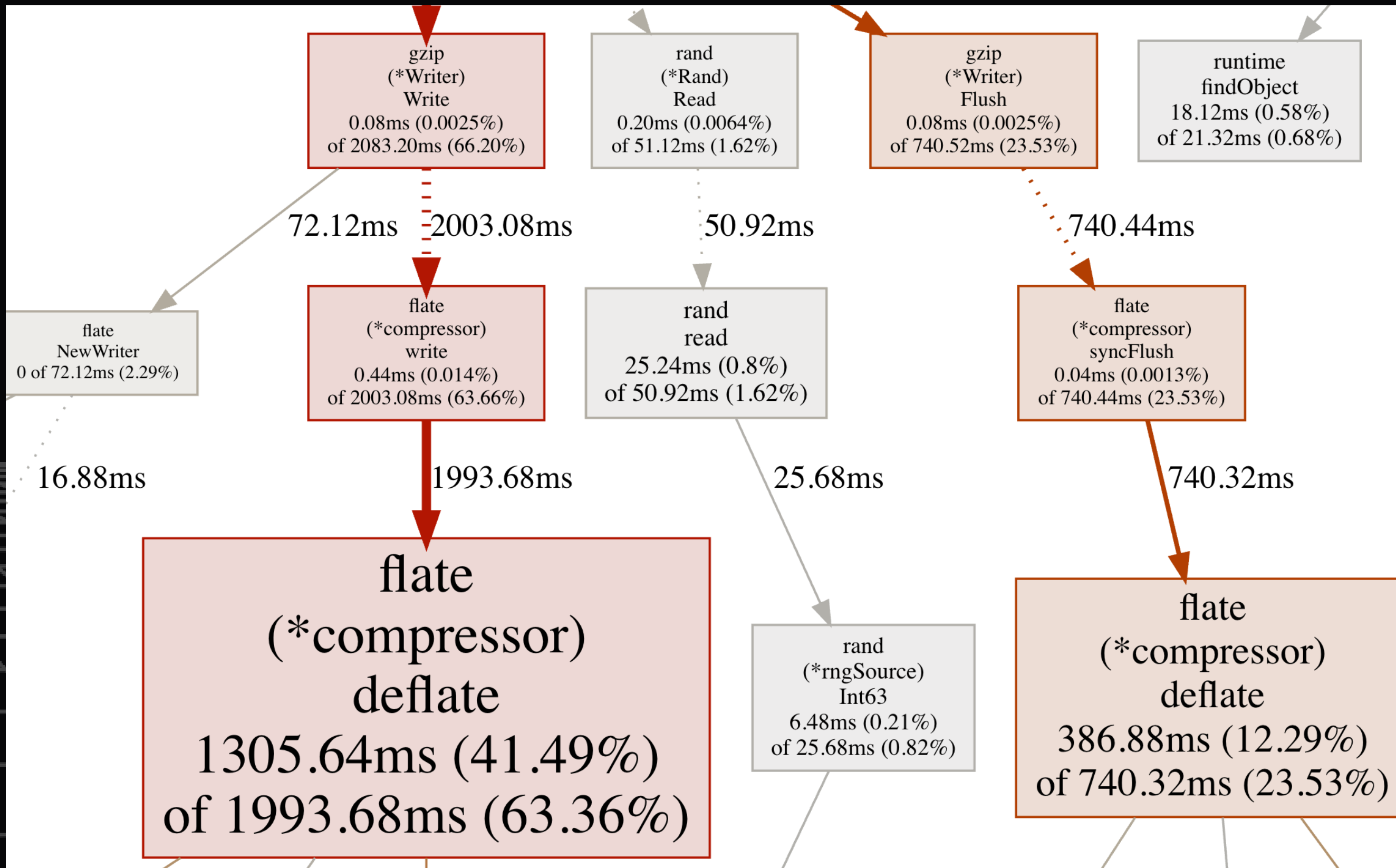
ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

```
(pprof) disasm Fibonacci
Total: 116.99s
ROUTINE ===== card_shielder/internal/core/conversion.Fibonacci
0      450ms (flat, cum) 0.38% of Total
.      . 1000d2c60: MOVD 16(R28), R16                ;optimized_recursive.go:5
.      . 1000d2c64: CMP R16, RSP
.      . 1000d2c68: BLS 33(PC)
.      . 1000d2c6c: MOVD.W R30, -96(RSP)
.      . 1000d2c70: MOVD R29, -8(RSP)
.      . 1000d2c74: SUB $8, RSP, R29
.      . 1000d2c78: MOVD R0, 104(RSP)                ;optimized_recursive.go:24
.      . 1000d2c7c: ADD $1, R0, R2                  ;optimized_recursive.go:6
.      . 1000d2c80: MOVD R2, 32(RSP)
.      . 1000d2c84: MOVD R2, R1
.      . 1000d2c88: ADRP 2383872(PC), R0
.      . 1000d2c8c: ADD $2848, R0, R0
.      440ms 1000d2c90: CALL runtime.makeslice(SB)      ;card_shielder/internal/core/conversion.Fibonacci optimized\_recursive.go:6
```

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

```
-----+-----
(pprof) tree
Showing nodes accounting for 113.51s, 97.03% of 116.99s total
Dropped 208 nodes (cum <= 0.58s)
Showing top 80 nodes out of 101
-----+-----
      flat  flat%   sum%        cum   cum%   calls calls% + context
-----+-----
      27.26s 23.30% 23.30%       84.59s 72.31%           | card_shielder/internal/core/conversion.Fibonacci.func1
                                57.24s 67.67%           | runtime.newstack
-----+-----
                                20.24s 92.29%           | runtime.adjustframe
                                1.69s  7.71%           | runtime.scanframewoker
      9.21s  7.87% 31.17%       21.93s 18.75%           | runtime.(*stkframe).getStackMap
                                7.18s 32.74%           | runtime.pcdatavalue
```

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА



(pprof) web
(pprof) █

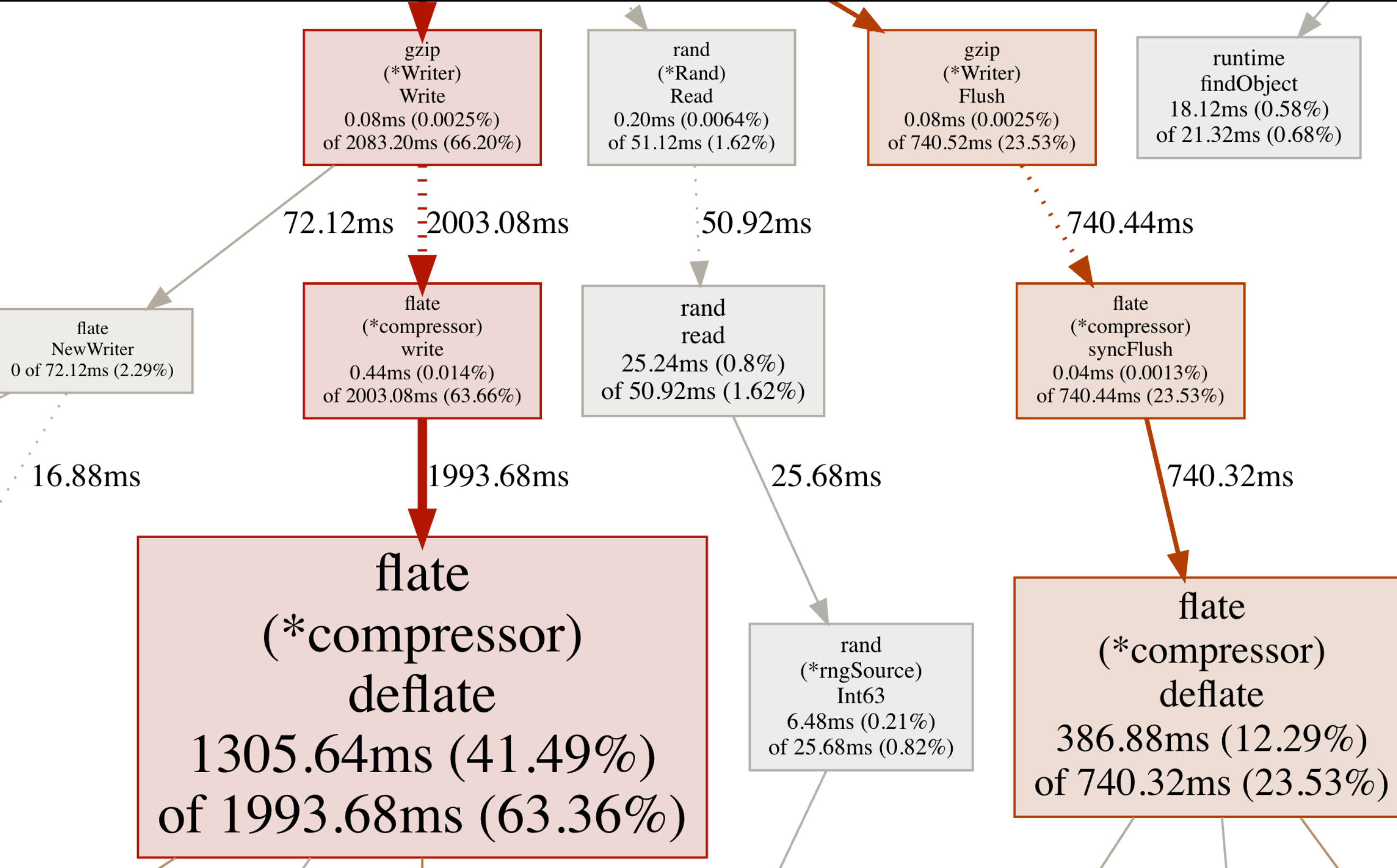
Вершины:

цвет - sum

- красный
- белый
- зеленый

размер - flat

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА



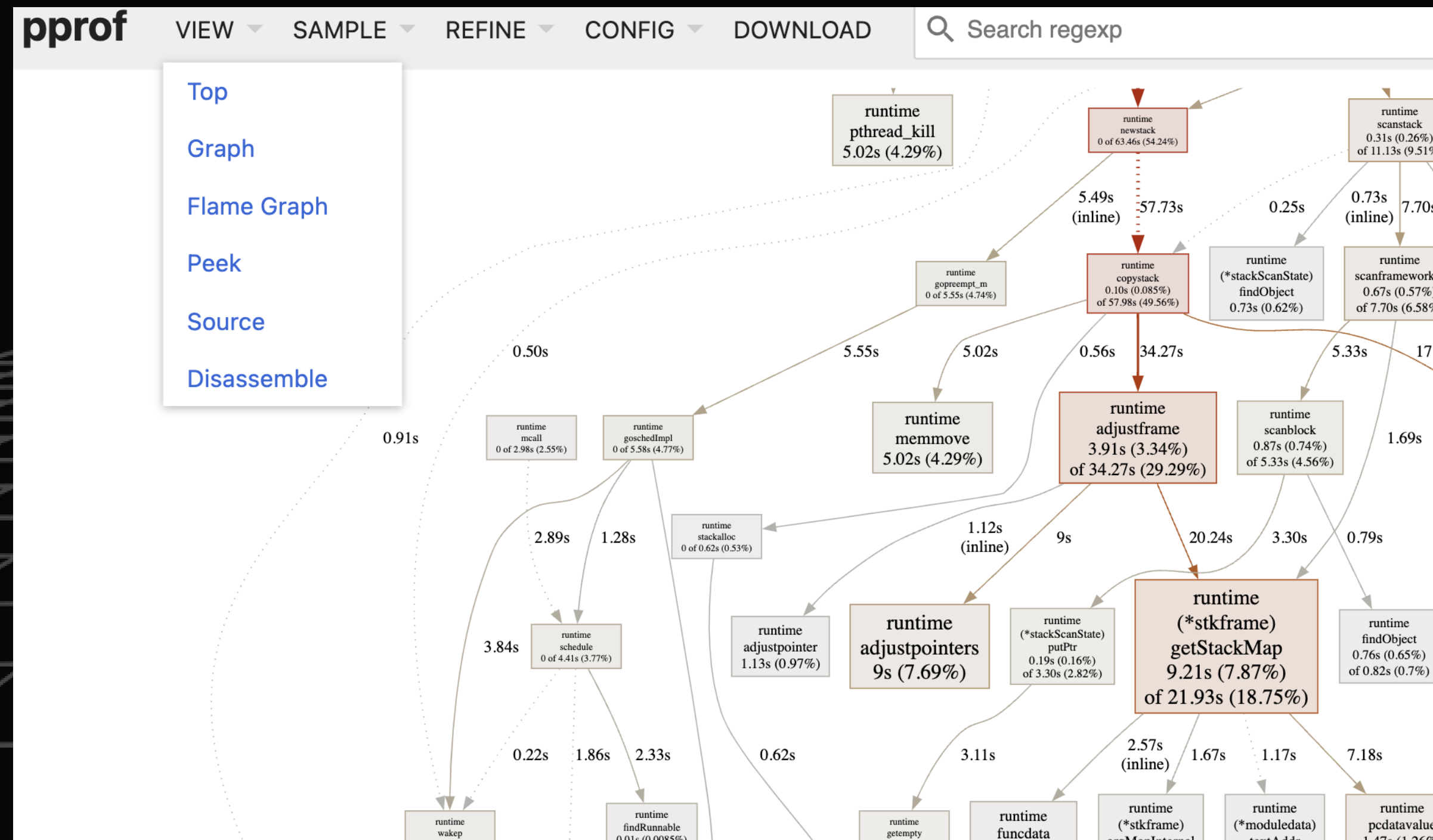
(pprof) web
(pprof) █

Ребра:

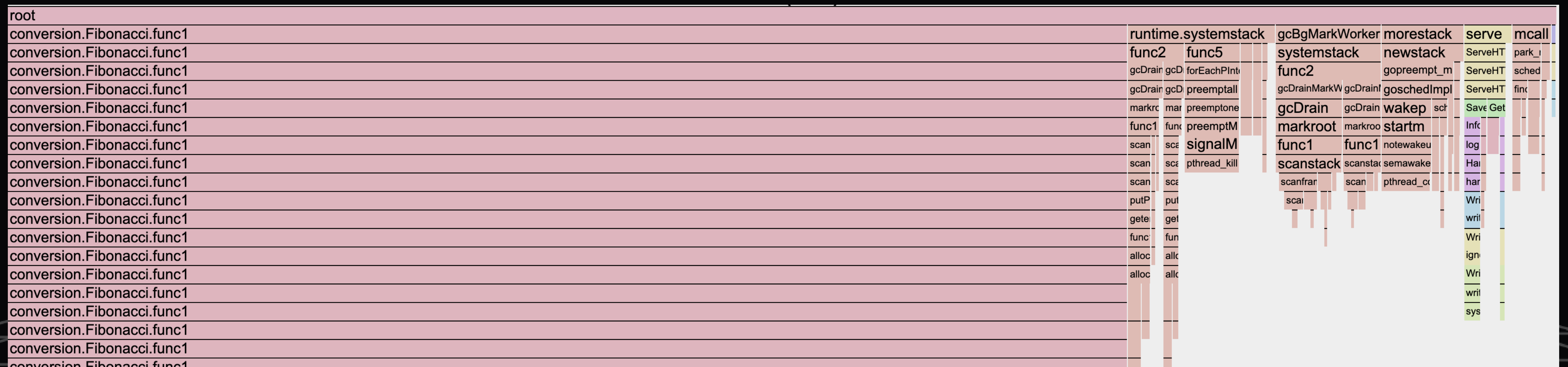
- цвет и размер – ресурсы на пути
- пунктир – пропуск вершин

ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА

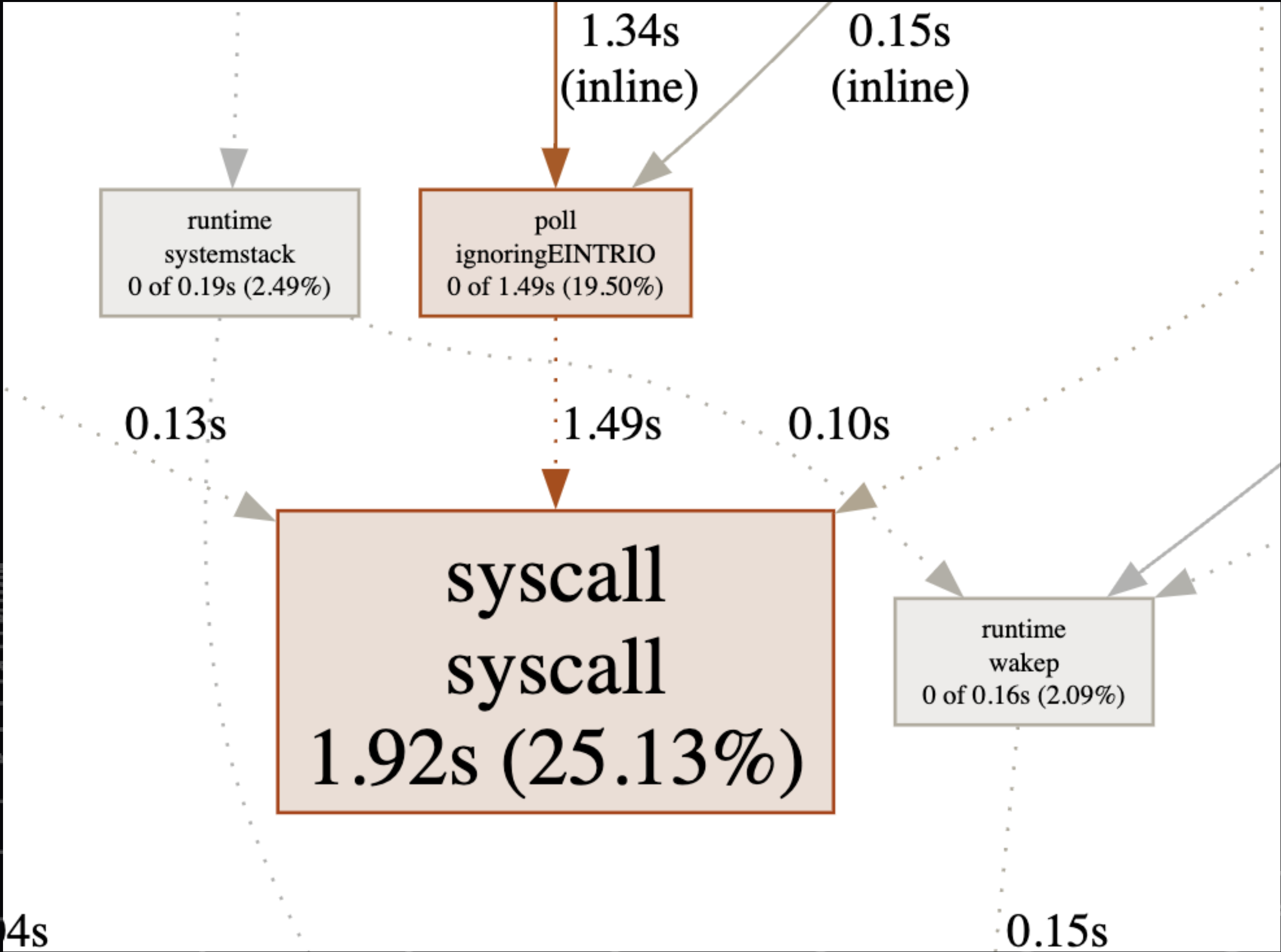
```
go tool pprof -http :8081 ./pprof.___5990go_build_card_shielder_go.samples.cpu.001.pb.gz
Serving web UI on http://localhost:8081
```



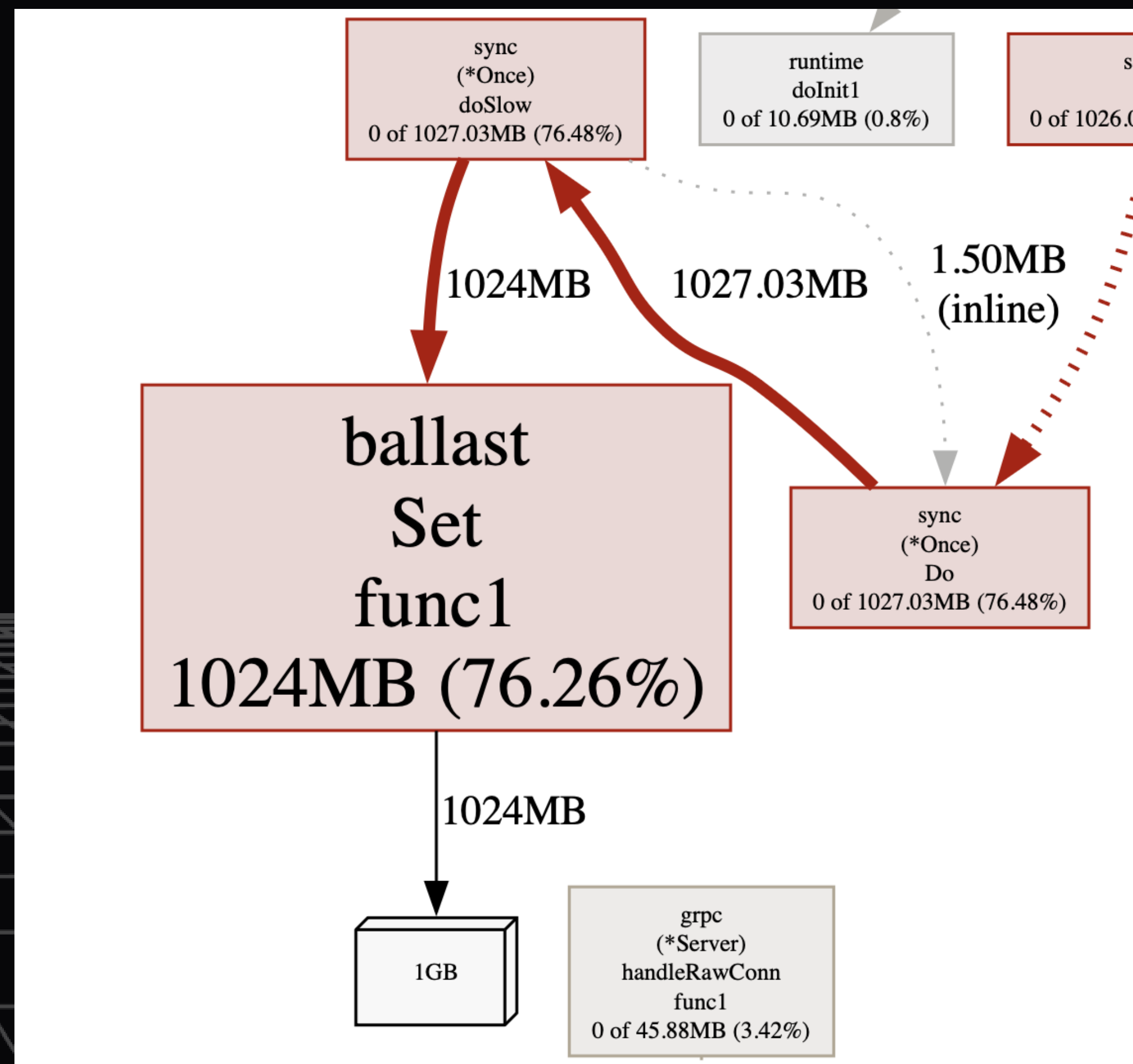
ИСПОЛЬЗОВАНИЕ ПРОФИЛИРОВЩИКА



CPU PROFILING



ALLOCS/HEAP PROFILING

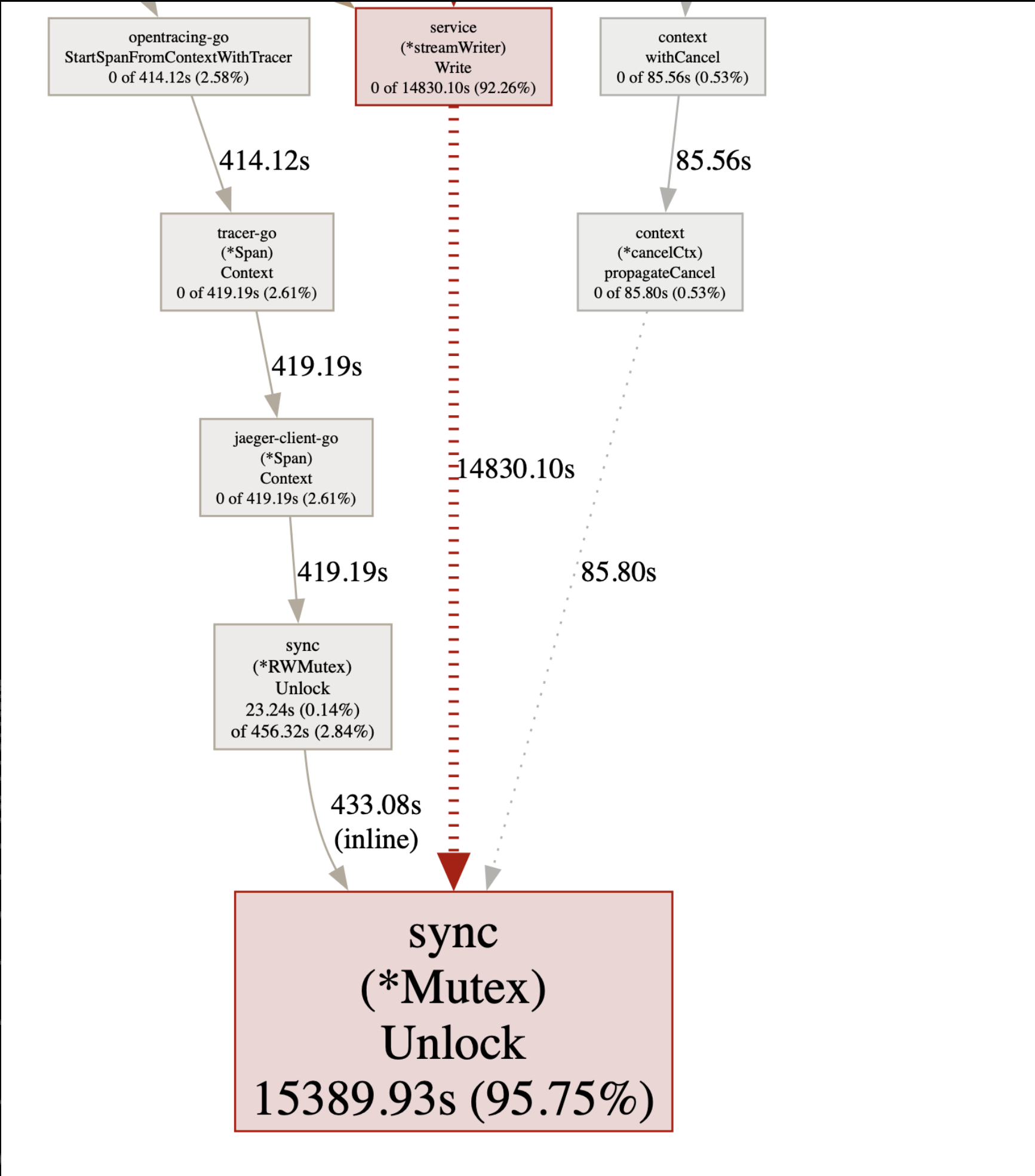


- alloc_objects/count
- alloc_space/bytes
- inuse_objects/count
- inuse_space/bytes

alloc_* - профиль, начиная со старта процесса

inuse_* - профиль, начиная с последней сборки мусора

BLOCK/MUTEX PROFILING



THREADCREATE PROFILE

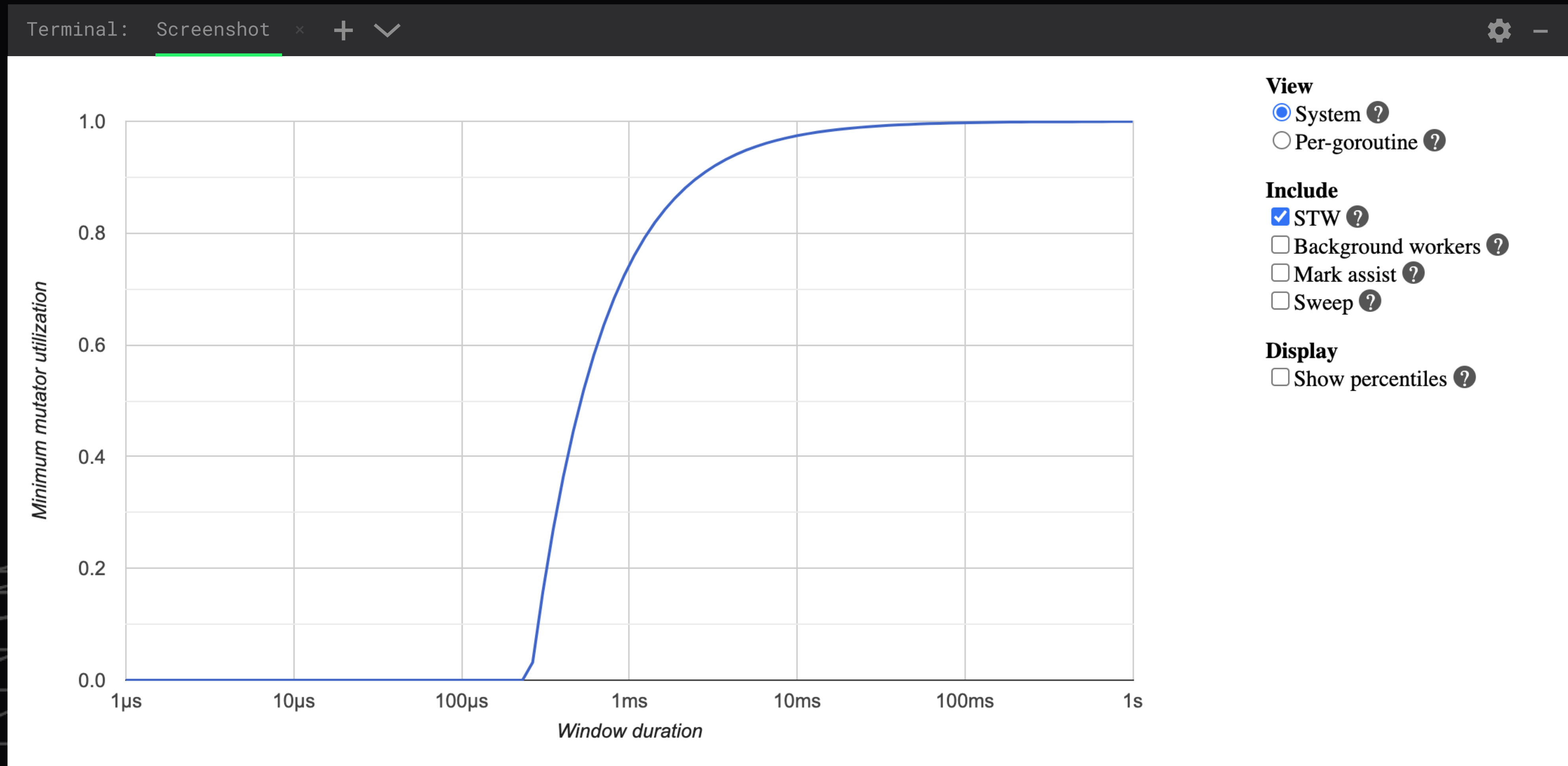
Был частично сломан, используется очень редко для отладки создания большого количества потоков операционной системы

Comment 3:

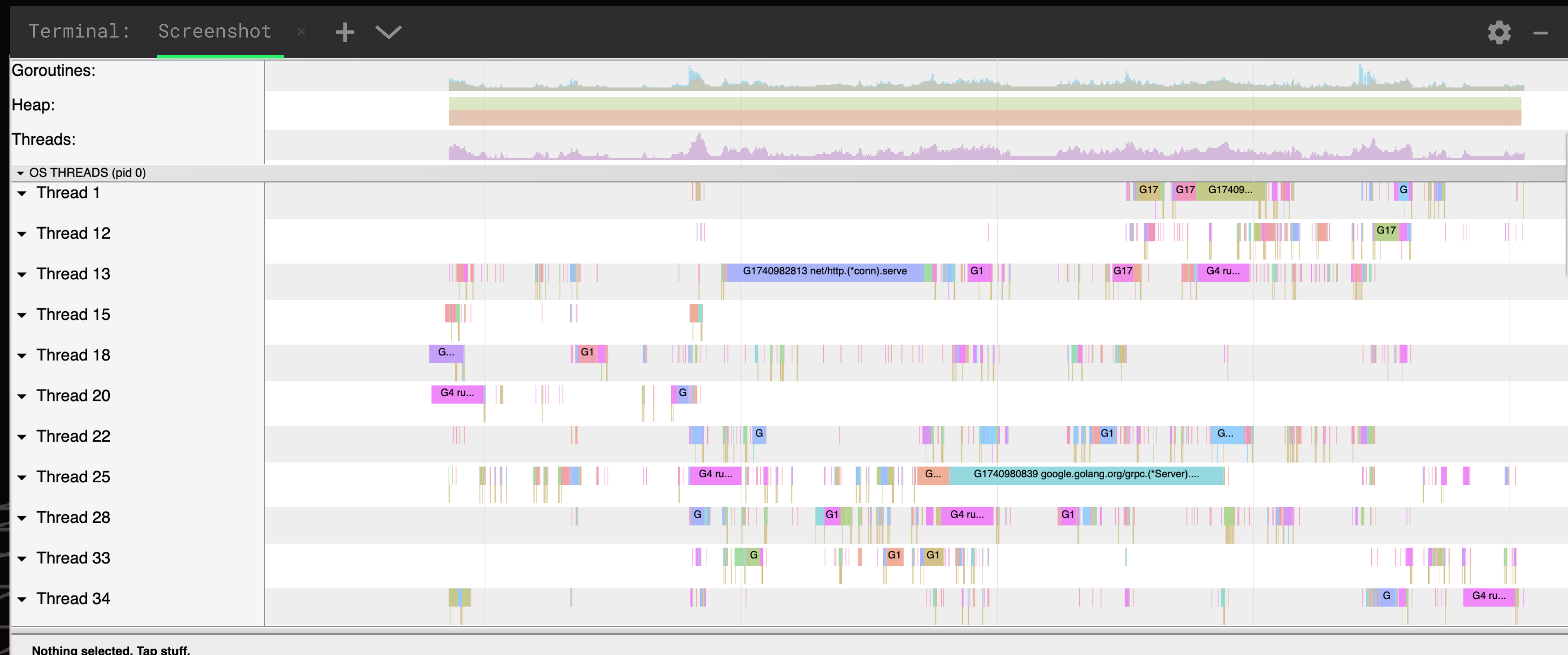
The idea was that if, for example, your program was creating 1000s of threads because they kept getting blocked on cgo DNS lookups, you'd see the cgo DNS call in the profile sample, because it is what blocked and triggered the creation of a new thread.

TRACE PROFILE

TRACE PROFILE



TRACE PROFILE



TRACE PROFILE



GOROUTINE PROFILING

- Статус горутины (idle, runnable, running, syscall, waiting, dead, copystack, preempted)
- Wait reason (IO wait, chan receive, select, trace reader, debug call, finalizer wait...)
- Wait since
- Stack trace
- GoPC
- Lockedm

Terminal: Screenshot × + ▾

Goroutine	Total		Execution time	Block time ()	Block time (chan receive)	Block time (preempted)	Block time (select)	Block time (sync)	Block time (syscall)	Sched wait time	Syscall execution time	Unknown time
1740954373	448.690048ms		857.536μs	0s	0s	0s	447.76192ms	0s	0s	70.592μs	0s	0s
1740966924	421.18368ms		822.528μs	0s	0s	0s	420.281536ms	0s	0s	79.616μs	0s	0s
1740957351	377.73152ms		28.416μs	0s	377.68608ms	0s	0s	0s	0s	17.024μs	0s	0s
1740957346	377.730176ms		841.92μs	0s	0s	0s	376.838976ms	0s	0s	49.28μs	0s	0s
1740928475	375.468672ms		33.472μs	0s	375.415808ms	0s	0s	0s	0s	19.392μs	0s	0s
1740928472	375.4672ms		252.672μs	0s	0s	0s	375.1616ms	0s	0s	52.928μs	0s	0s
1740930712	374.343232ms		3.779456ms	0s	0s	0s	370.444864ms	0s	0s	118.912μs	0s	0s
1740930753	374.338624ms		53.441μs	0s	374.262975ms	0s	0s	0s	0s	22.208μs	0s	0s
1740931946	364.022144ms		770.368μs	0s	0s	0s	363.178305ms	0s	0s	73.471μs	0s	0s
1740975901	354.457664ms		1.200705ms	0s	0s	0s	353.172479ms	0s	0s	84.48μs	0s	0s

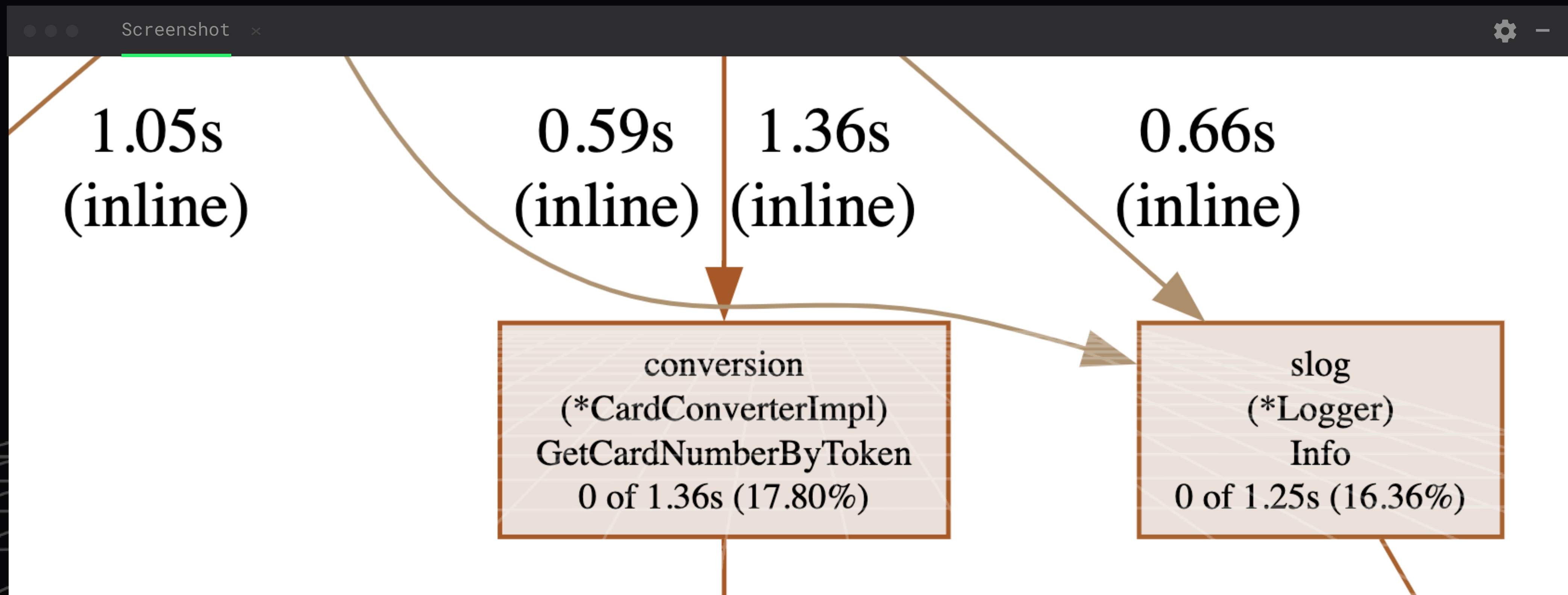
GOROUTINE PROFILING

Terminal: Screenshot				
		these 3 use the same runtime_goroutineProfileWithLabels() func internally		
	runtime.Stack() pprof.Lookup(debug=2)	pprof.Lookup(debug=1)	pprof.Lookup(debug=0)	runtime.GoroutineProfile()
format	text: like panic()	text: pprof format	protocol buffer: pprof format	structured: []StackRecord
individual goroutines	yes	no	no	yes
goid	yes	no	no	no
atomicstatus	yes	no	no	no
waitreason	yes	no	no	no
waitsince	yes	no	no	no
labels	no	yes	yes	no
gopc (pc)	no	no	no	no
gopc (func,file,line)	yes	no	no	no
lockedm	yes	no	no	no
stack trace (pcs)	no	yes	yes	yes
stack trace (func,file,line)	yes	yes	yes	computable
max stack depth	100	32	32	32
overhead per goroutine	O(N) STW (+symbolization)	O(N) STW	O(N) STW	O(N) STW
color legend	good	meh	bad	
	how well it works for continuous profiling			

PGO

PROFILE-GUIDED OPTIMIZATION

-pgo=profile.pb.gz



CONTINUOUS PROFILING

GOROUTINE PROFILING

Сбор профиля

Вы моментально соберете все профили
указанного сервиса `composer-api` на
текущий момент

Собрать профиль

Сбор профиля

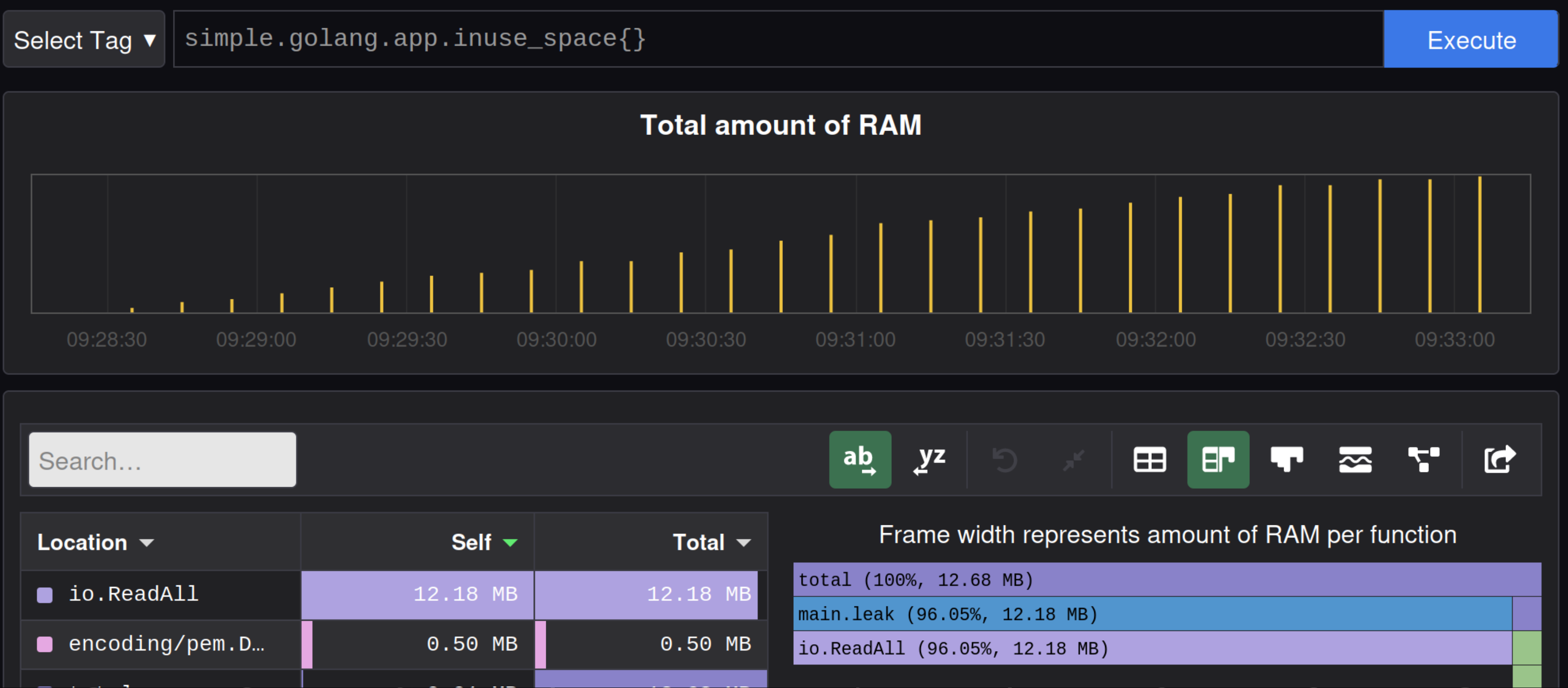
Вы моментально соберете все профили
указанного сервиса `service-name` на
текущий момент

Собрать профиль

CONTINUOUS PROFILING

```
10     "github.com/grafana/pyroscope-go"
11 )
12
13 ► func main() {
14     pyroscope.Start(pyroscope.Config{
15         ApplicationName: "simple.golang.app",
16         ServerAddress:    "http://localhost:4040",
17         Logger:           pyroscope.StandardLogger,
18         ProfileTypes: []pyroscope.ProfileType{
19             pyroscope.ProfileCPU,
20             pyroscope.ProfileAllocObjects,
21             pyroscope.ProfileAllocSpace,
22             pyroscope.ProfileInuseObjects,
23             pyroscope.ProfileInuseSpace,
24         },
25     })
26
27
```

CONTINUOUS PROFILING



ПРИМЕРЫ ПРОФИЛИРОВАНИЯ

ПРИМЕРЫ ПРОФИЛИРОВАНИЯ

```
func ConvertThroughSerialization(src interface{}, dst interface{}) error {  
    serialized, err := json.Marshal(src)  
  
    if err != nil {  
        return err  
    }  
  
    err = json.Unmarshal(serialized, dst)  
  
    if err != nil {  
        return err  
    }  
    return nil  
}
```

ПРИМЕРЫ ПРОФИЛИРОВАНИЯ

```
func transform[T, R any]( no usages  new *
    ctx context.Context,
    in <-chan T,
    out chan<- R,
    f func(e T) R,
) {
    for {
        select {
        case <-ctx.Done():
        case v, ok := <-in:
            if !ok {
                return
            }

            out <- f(v) // ???
        }
    }
}
```

ПРИМЕРЫ ПРОФИЛИРОВАНИЯ

```
func transform[T, R any]( no usages new *
    ctx context.Context,
    in <-chan T,
    out chan<- R,
    f func(e T) R,
) {
    for {
        select {
        case <-ctx.Done():
        case v, ok := <-in:
            if !ok {
                return
            }

            out <- f(v) // ???
        }
    }
}
```

```
func transform[T, R any]( no usages new *
    ctx context.Context,
    in <-chan T,
    out chan<- R,
    f func(e T) R,
) {
    for {
        select {
        case <-ctx.Done():
        case v, ok := <-in:
            if !ok {
                return
            }

            select {
            case <-ctx.Done():
                return
            case out <- f(v): // ok
            }
        }
    }
}
```

УГЛУБЛЕННЫЙ КУРС ПО ОПТИМИЗАЦИЯМ В GOLANG

Научишься выжимать все соки из GO

С помощью основ computer science, низкоуровневого программирования, ассемблера и практических фишек, о которых мало кто рассказывает

длительность

6 недель

Terminal: Difference × + ✓

```
for i := 0; i < len(exampleData); i++ {  
    data := unsafe.Slice(unsafe.StringData(exampleData[i]),  
                          len(exampleData[i]))  
  
    if externalMatchFunction(data) {  
        result = append(result, data)  
    }  
}  
  
return result  
}
```

```
goos: darwin  
goarch: arm64  
pkg: landing_examples  
BenchmarkMatch/fast_match-      135  
BenchmarkMatch/slow_match      2  
PASS  
ok      landing_examples
```