Go execution tracer

Gophercon Brazil 2017

André Carvalho

$ whoami

- Developer @ globo.com / tsuru
- Interested in all things performance, tracing and systems programming
- @andresantostc
- https://andrestc.com

Agenda

- The tool basics
  - What is the Go execution tracer
  - How to collect trace data
  - How to analyze trace data
- Using the tracer
  - to improve working code
  - to investigate a strange behavior

Go execution tracer

- Gives insight into the execution of a Go program
  - What are my goroutines doing when not in CPU?
- Instruments the go runtime
  - Captures events in nanosecond precision
  - Data is not aggregated/sampled!
- Available since go 1.5

Go execution tracer

- Events
  - Goroutines creation/start/end/block/unblock
  - Network
  - Syscalls
  - Memory allocation
  - Garbage collection

Collecting traces

- Three ways to collect traces
  - trace.Start / trace.Stop
  - go test -trace=trace.out
  - debug/pprof/trace handler

Collecting traces

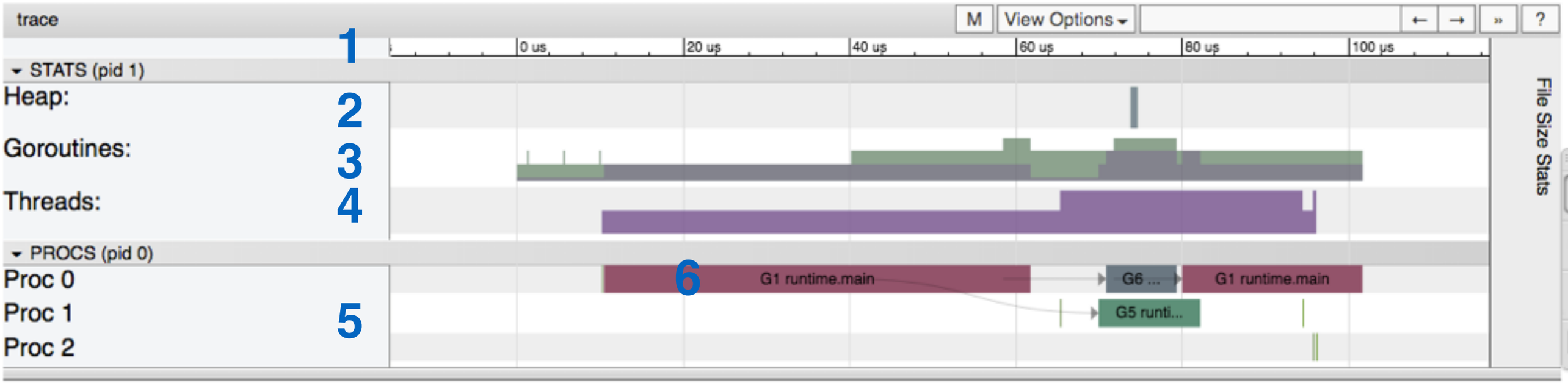
- Writes the tracing output to os.Stderr
- go run main.go 2> trace.out
- go tool trace trace.out

```go
package main

import (
    "os"
    "runtime/trace"
)

func main() {
    trace.Start(os.Stderr)
    defer trace.Stop()
    // create new channel of type int
    ch := make(chan int)

    // start new anonymous goroutine
    go func() {
        // send 42 to channel
        ch <- 42
    }()
    // read from channel
    <-ch
}
```

Trace

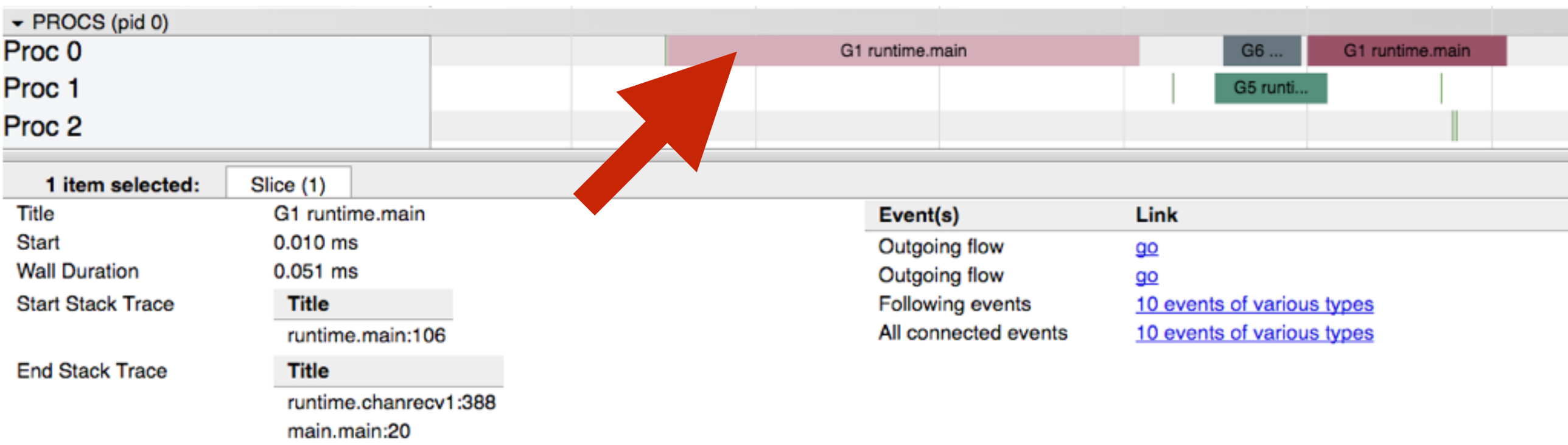
View trace
Goroutine analysis
Network blocking profile
Synchronization blocking profile
Syscall blocking profile
Scheduler latency profile

View Trace



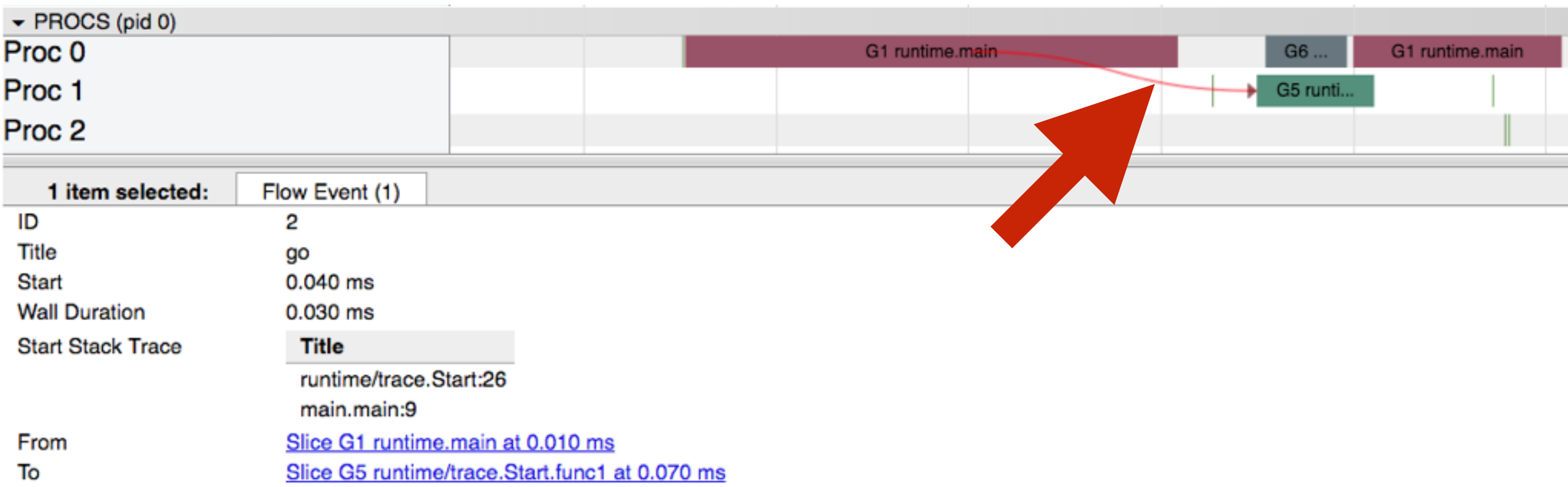
1. Timeline
2. Heap usage
3. goroutines
4. OS threads
5. Virtual Processors
6. goroutine + events

View Trace



Selecting a goroutine

View Trace



Selecting an event

Tracing web applications

```go
package main

import (
    "net/http"
    _ "net/http/pprof"
)

func main() {
    http.Handle("/hello", http.HandlerFunc(helloHandler))

    http.ListenAndServe(":8080", http.DefaultServeMux)
}

func helloHandler(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("hello world!"))
}
```

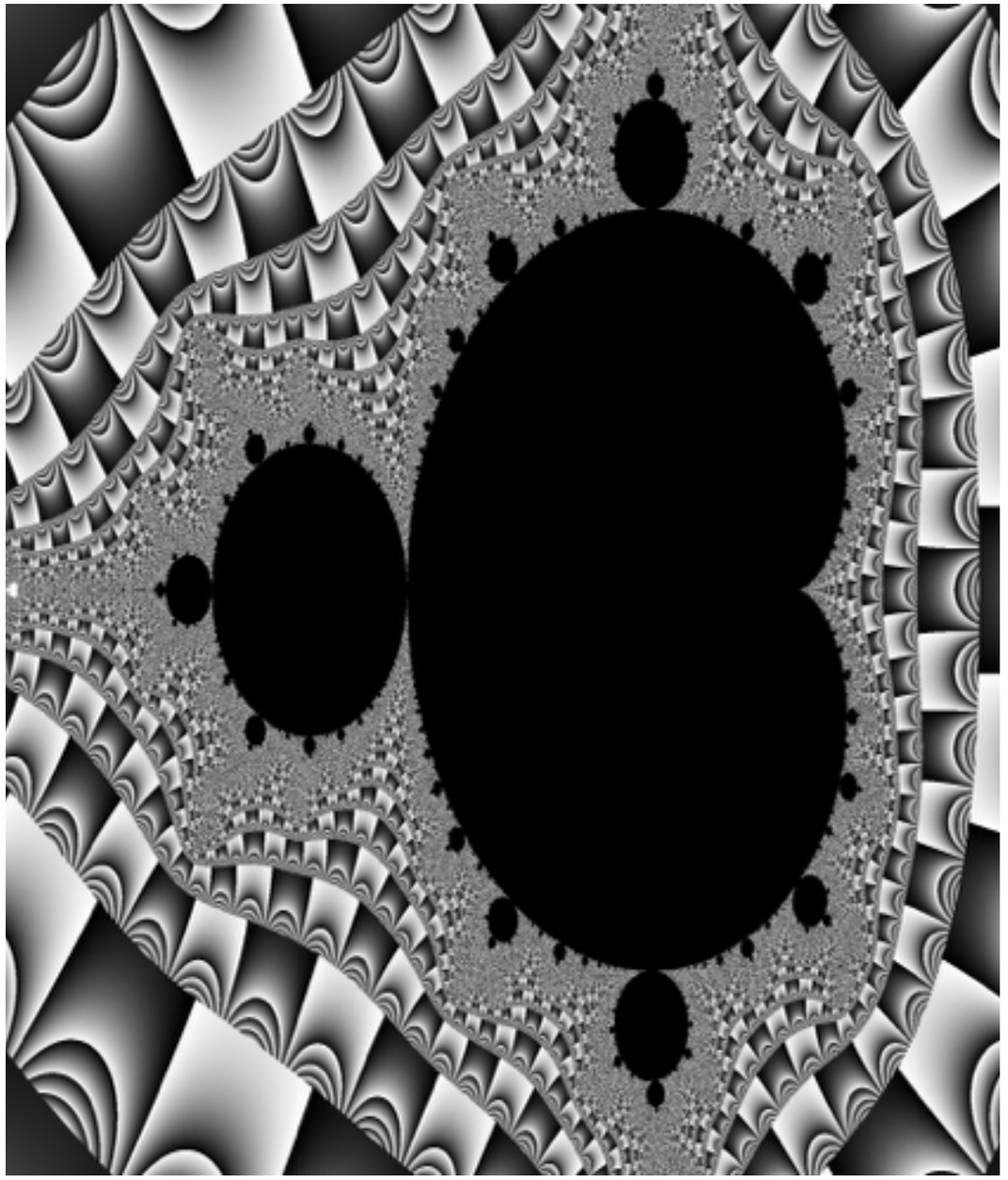
https://github.com/andrestc/go-tracing/tree/master/07-web

Using the tracer to improve working code

Mandelbrot

CPU intensive calculations to figure out each pixel's color

https://github.com/campoy/mandelbrot/



Pixel

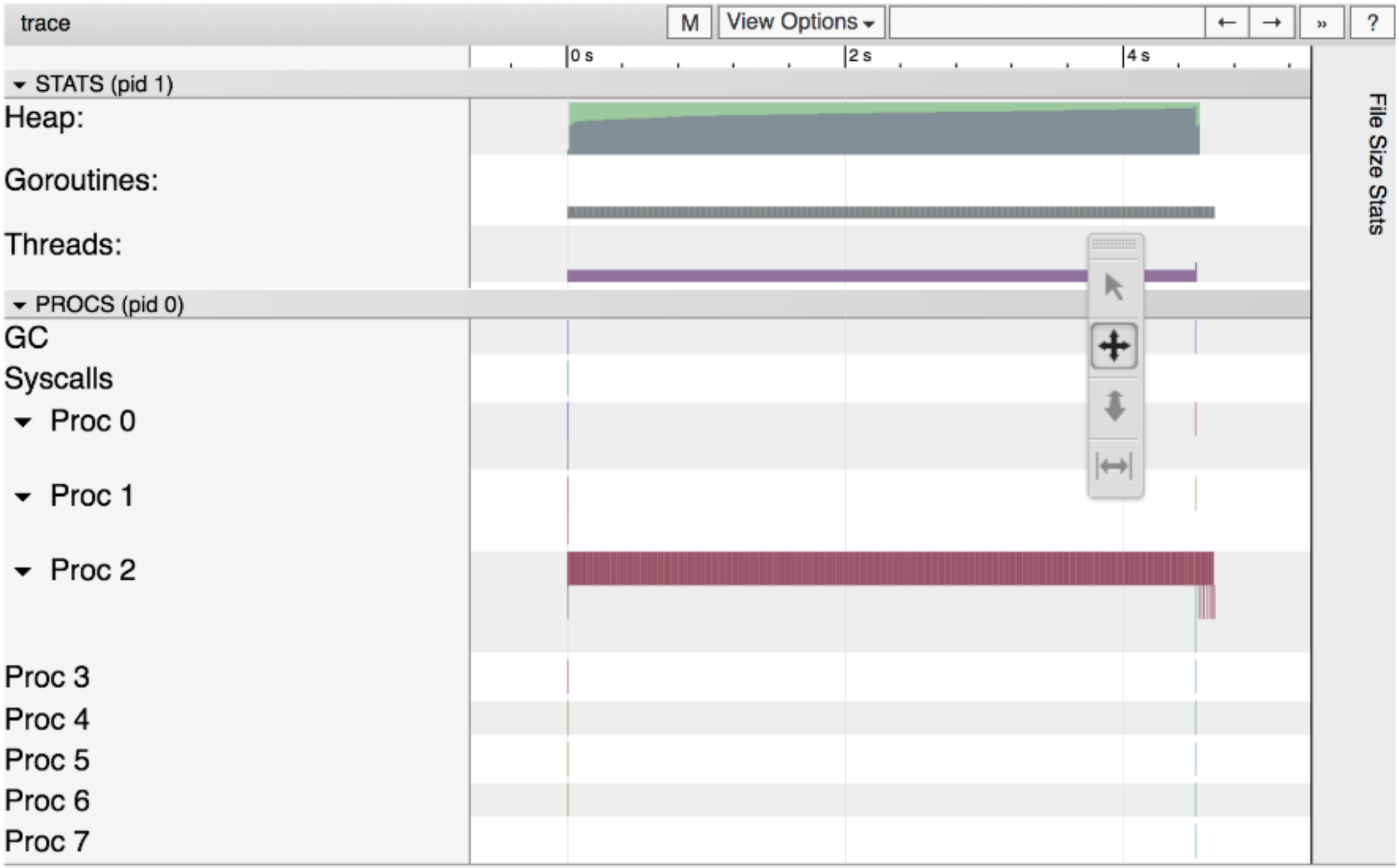
```
func pixel(i, j, width, height int) color.Color {
        const complexity = 4

        xi := norm(i, width, -1.0, 2)
        yi := norm(j, height, -1, 1)

        const maxI = 1000
        x, y := 0., 0.

        for i := 0; (x*x+y*y < complexity) && i < maxI; i++ {
                x, y = x*x-y*y+xi, 2*x*y+yi
        }

        return color.Gray{uint8(x)}
}
```

Sequential

```
func createSeq(width, height int) image.Image {
        m := image.NewGray(image.Rect(0, 0, width, height))
        for i := 0; i < width; i++ {
                for j := 0; j < height; j++ {
                        m.Set(i, j, pixel(i, j, width, height))
                }
        }
        return m
}
```

```
$ time ./mandelbrot
./mandelbrot  4.44s user 0.01s system 99% cpu 4.466 total
```

Sequential

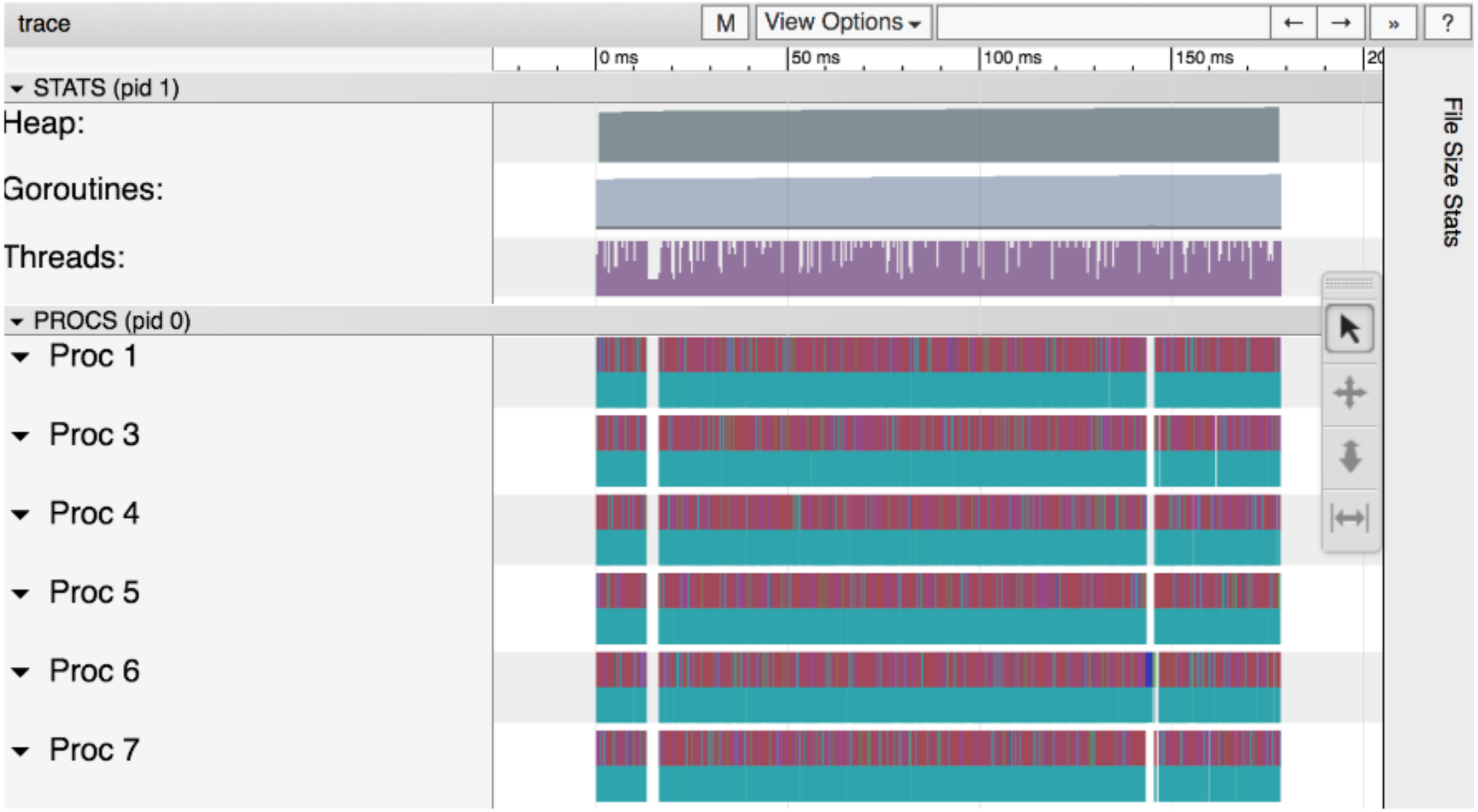

trace
M
View Options
0 s
2 s
4 s
STATS (pid 1)
Heap:
Goroutines:
Threads:
PROCS (pid 0)
GC
Syscalls
Proc 0
Proc 1
Proc 2
Proc 3
Proc 4
Proc 5
Proc 6
Proc 7
File Size Stats

One goroutine per pixel

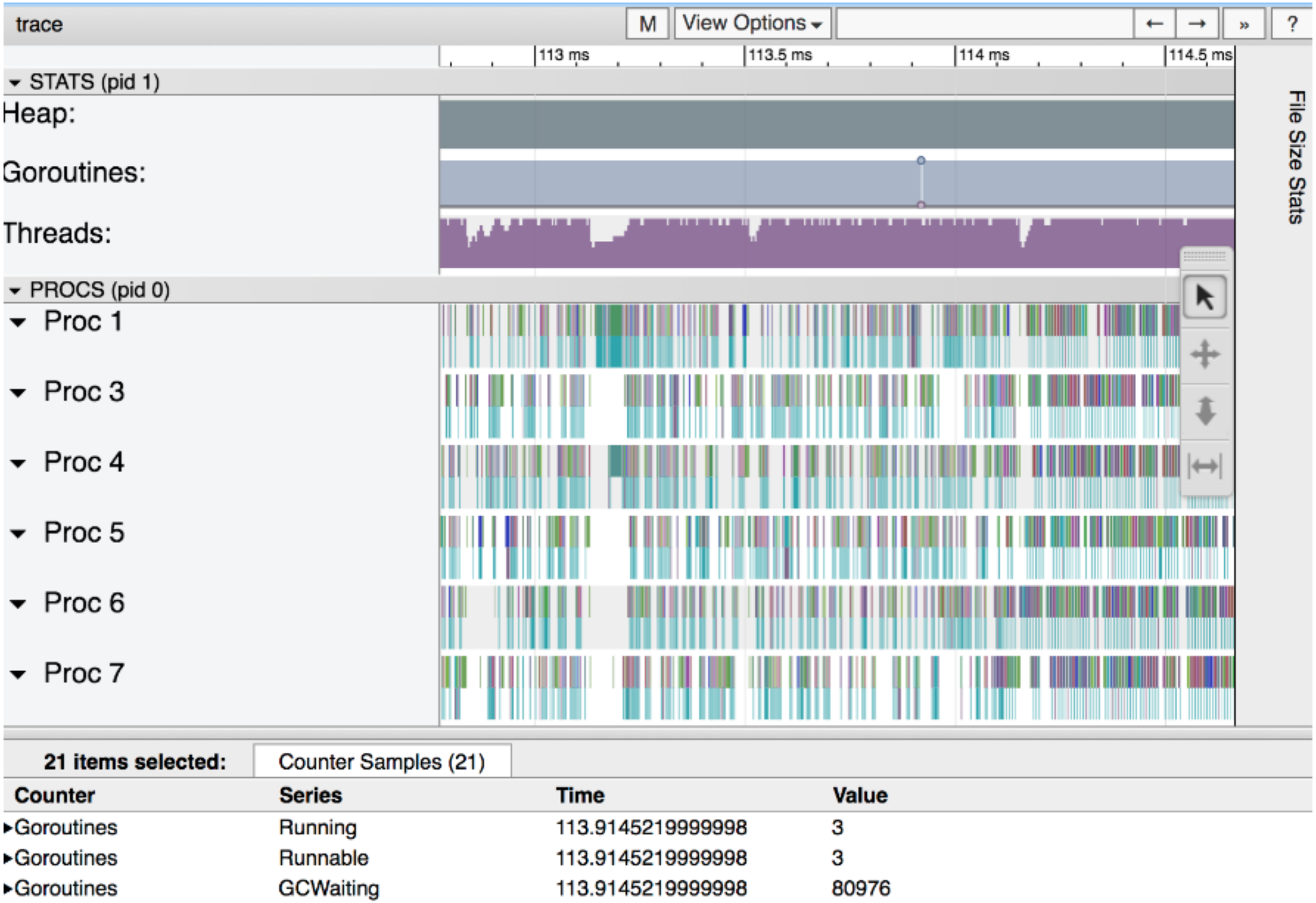
```
func createPixel(width, height int) image.Image {
        m := image.NewGray(image.Rect(0, 0, width, height))
        var w sync.WaitGroup
        w.Add(width * height)
        for i := 0; i < width; i++ {
                for j := 0; j < height; j++ {
                        go func(i, j int) {
                                m.Set(i, j, pixel(i, j, width, height))
                                w.Done()
                        }(i, j)
                }
        }
        w.Wait()
        return m
}
```

```
$ time ./mandelbrot
./mandelbrot  13.70s user 1.53s system 471% cpu 3.234 total
```

One goroutine per pixel



One goroutine per pixel



One goroutine per column

```
func createCol(width, height int) image.Image {
        m := image.NewGray(image.Rect(0, 0, width, height))
        var w sync.WaitGroup
        w.Add(width)
        for i := 0; i < width; i++ {
                go func(i int) {
                        for j := 0; j < height; j++ {
                                m.Set(i, j, pixel(i, j, width, height))
                        }
                        w.Done()
                }(i)
        }
        w.Wait()
        return m
}
```

```
$ time ./mandelbrot
./mandelbrot  4.90s user 0.02s system 499% cpu 0.985 total
```

trace

M | View Options | ← | → | » | ?

110 ms | 120 ms | 130 ms | 140 ms

File Size Stats

STATS (pid 1)

Heap:

Goroutines:

Threads:

PROCS (pid 0)

GC

Syscalls

Proc 0

i1116 m... | G1117 m... | G1118 m... | G1119 m... | G1120 m... | G1121 m... | G1122 r

Proc 1

1300 main.... | G1301 main.... | G1302 main.... | G1303 main.... | G1304 main

Proc 2

in.... | G1207 m... | G1208 m... | G1209 m... | G1210 m... | G1211 main.... | G1212 m...

Proc 3

G1554 m... | G1555 m... | G1556 m... | G1557 m... | G1558 m... | G1559 m... | G1560 m... | G156

Proc 4

G... | G... | G | G58 | G1048 m... | G1049 m... | G1050 m... | G1051 m... | G1052 m...

Proc 5

G982 m... | G983 m... | G984 m... | G985 m... | G986 m... | G98... | G98... | G98... | G99...

Proc 6

G94... | G... | G101... | G... | G... | G94... | G

Proc 7

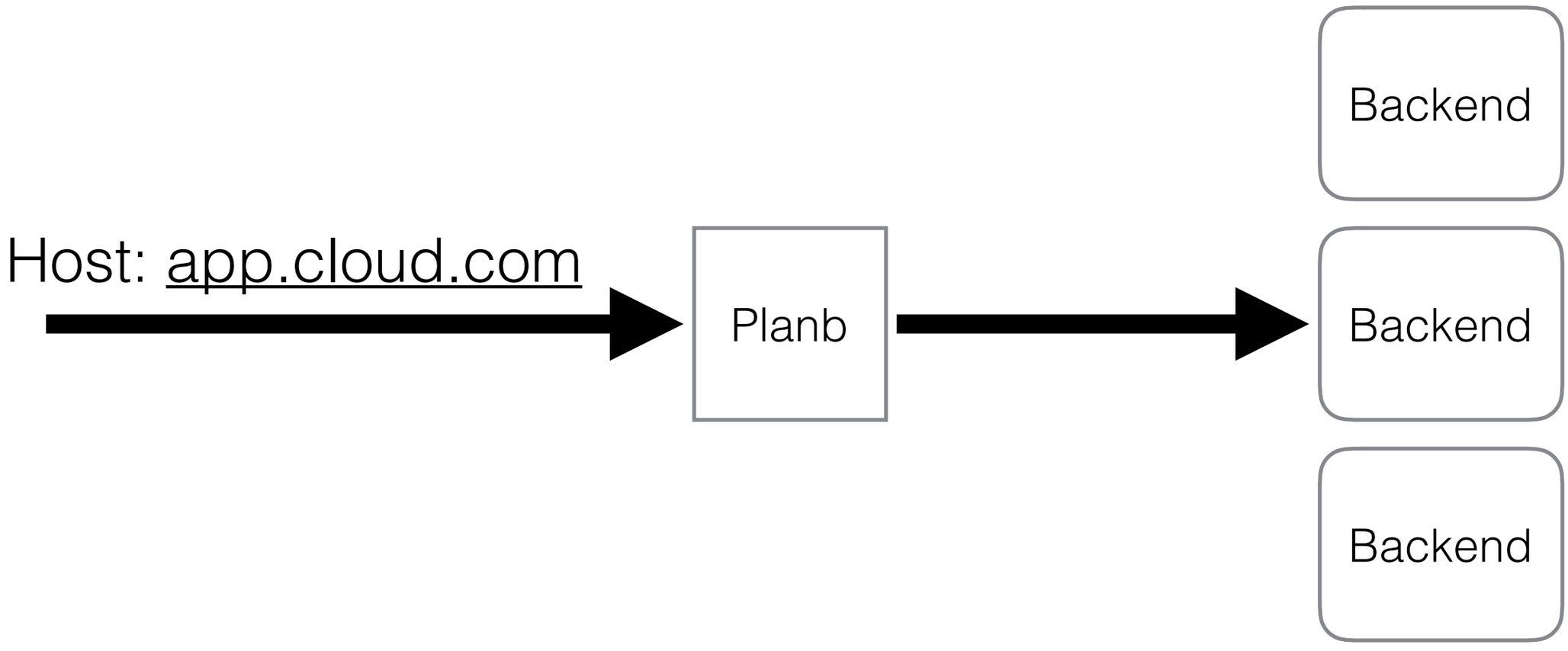
28 main.... | G1429 main.... | G1430 main.... | G1431 main.... | G1432 main.... | G1433 main.... | G1563

Debugging a strange test behavior

Planb

- github.com/tsuru/planb
  - reverse proxy based of hipache
- One of tsuru's key components
- Used extensively on globo.com

Planb



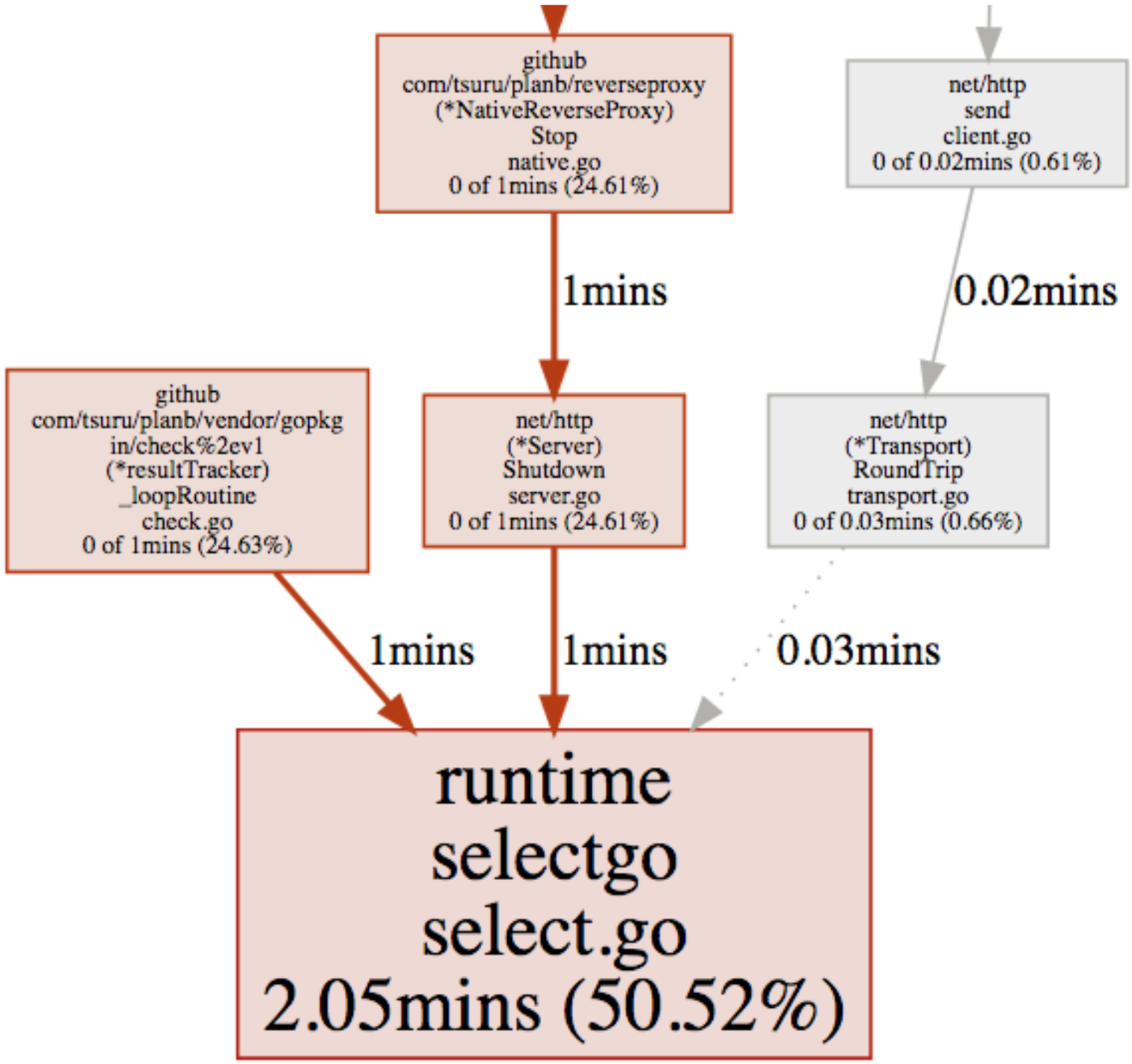
```
$ go test -check.v -check.f TestRoundTripStressWithTimeout
PASS: reverseproxy_test.go:763: S.TestRoundTripStressWithTimeoutBackend 0.012s
PASS: reverseproxy_test.go:763: S.TestRoundTripStressWithTimeoutBackend 0.011s
OK: 2 passed
PASS
ok      github.com/tsuru/planb/reverseproxy     0.044s
```

```
$ go test -check.v -check.f TestRoundTripStressWithTimeout
PASS: reverseproxy_test.go:763: S.TestRoundTripStressWithTimeoutBackend 60.013s
PASS: reverseproxy_test.go:763: S.TestRoundTripStressWithTimeoutBackend 0.014s
OK: 2 passed
PASS
ok      github.com/tsuru/planb/reverseproxy     60.045s
```

```
763  func (s *S) TestRoundTripStressWithTimeoutBackend(c *check.C) {
764      router := &noopRouter{dst: "http://127.0.0.1:23771"}
765      rp := s.factory()
766      err := rp.Initialize(ReverseProxyConfig{Router: router})
767      c.Assert(err, check.IsNil)
768      addr, listener := getFreeListener()
769      go rp.Listen(listener, nil)
770      defer rp.Stop()
771      defer listener.Close()
772      u := fmt.Sprintf("http://%s/", addr)
773      wg := sync.WaitGroup{}
774      nConnections := 50
775      for i := 0; i < nConnections; i++ {
776          wg.Add(1)
777          go func() {
778              defer wg.Done()
779              req, err := http.NewRequest("GET", u, nil)
780              c.Assert(err, check.IsNil)
781              req.Host = "myhost.com"
782              rsp, err := http.DefaultClient.Do(req)
783              c.Assert(err, check.IsNil)
784              defer rsp.Body.Close()
785              c.Assert(rsp.StatusCode, check.Equals, 503)
786          }()
787      }
788      done := make(chan bool)
789      go func() {
790          wg.Wait()
791          close(done)
792      }()
793      select {
794      case <-done:
795      case <-time.After(time.Minute):
796          c.Fatal("timeout out after 1 minute")
797      }
798  }
```

Let’s trace it!

```
$ go test -trace=trace.out -check.v -check.f TestRoundTripStressWithTimeout
```


github
com/tsuru/planb/reverseproxy
(*NativeReverseProxy)
Stop
native.go
0 of 1mins (24.61%)
net/http
send
client.go
0 of 0.02mins (0.61%)
1mins
0.02mins
github
com/tsuru/planb/vendor/gopkg
in/check%2ev1
(*resultTracker)
_loopRoutine
check.go
0 of 1mins (24.63%)
net/http
(*Server)
Shutdown
server.go
0 of 1mins (24.61%)
net/http
(*Transport)
RoundTrip
transport.go
0 of 0.03mins (0.66%)
1mins
1mins
0.03mins
runtime
selectgo
select.go
2.05mins (50.52%)

```
2487  func (srv *Server) Shutdown(ctx context.Context) error {
2488      atomic.AddInt32(&srv.inShutdown, 1)
2489      defer atomic.AddInt32(&srv.inShutdown, -1)
2490
2491      srv.mu.Lock()
2492      lnerr := srv.closeListenersLocked()
2493      srv.closeDoneChanLocked()
2494      for _, f := range srv.onShutdown {
2495          go f()
2496      }
2497      srv.mu.Unlock()
2498
2499      ticker := time.NewTicker(shutdownPollInterval)
2500      defer ticker.Stop()
2501      for {
2502          if srv.closeIdleConns() {
2503              return lnerr
2504          }
2505          select {
2506          case <-ctx.Done():
2507              return ctx.Err()
2508          case <-ticker.C:
2509          }
2510      }
2511  }
```

net/http.(*conn).serve N=50

Goroutine	Total time, ns	Execution time, ns	Network wait time, ns	Sync block time, ns	Blocking syscall time, ns	Scheduler wait time, ns	GC sweeping time, ns	GC pause time, ns
432	60036424682	20108	60035899328	0	0	505246	0	7890133
49	14275321	568388	1062813	11064154	0	1579966	0	0
149	14227546	496880	658666	10361225	0	2710775	0	0
152	14210924	222502	2766080	7996838	0	3225504	0	0
147	14159859	528756	1350353	10095634	0	2185116	0	0

https://github.com/golang/go/issues/21204

Workaround

```
@@ -763,7 +763,7 @@ func (s *S) TestRoundTripStress(c *check.C) {
763 763  func (s *S) TestRoundTripStressWithTimeoutBackend(c *check.C) {
764 764          router := &noopRouter{dst: "http://127.0.0.1:23771"}
765 765          rp := s.factory()
766     -        err := rp.Initialize(ReverseProxyConfig{Router: router})
    766 +        err := rp.Initialize(ReverseProxyConfig{Router: router, ReadTimeout: time.Second})
767 767          c.Assert(err, check.IsNil)
768 768          addr, listener := getFreeListener()
769 769          go rp.Listen(listener, nil)
@@ -1041,7 +1041,7 @@ func baseBenchmarkServeHTTP(rp ReverseProxy, b *testing.B) {
```

```
$ go test -check.v -check.f TestRoundTripStressWithTimeout
PASS: reverseproxy_test.go:763: S.TestRoundTripStressWithTimeoutBackend 1.013s
PASS: reverseproxy_test.go:763: S.TestRoundTripStressWithTimeoutBackend 0.013s
OK: 2 passed
PASS
ok      github.com/tsuru/planb/reverseproxy     1.055s
```

https://github.com/tsuru/planb/pull/35

Conclusions

- Go execution tracer helps understand concurrency
  - Complements other tools (eg memory/cpu profile)
- Not much documentation available
  - Opportunity for contributions

Reference

- Using the go tracer to speed up fractal rendering
  - Just for func #22
- Go Execution Tracer (Design Doc)

Obrigado!

Slides and notes in english
will be available at
https://andrestc.com