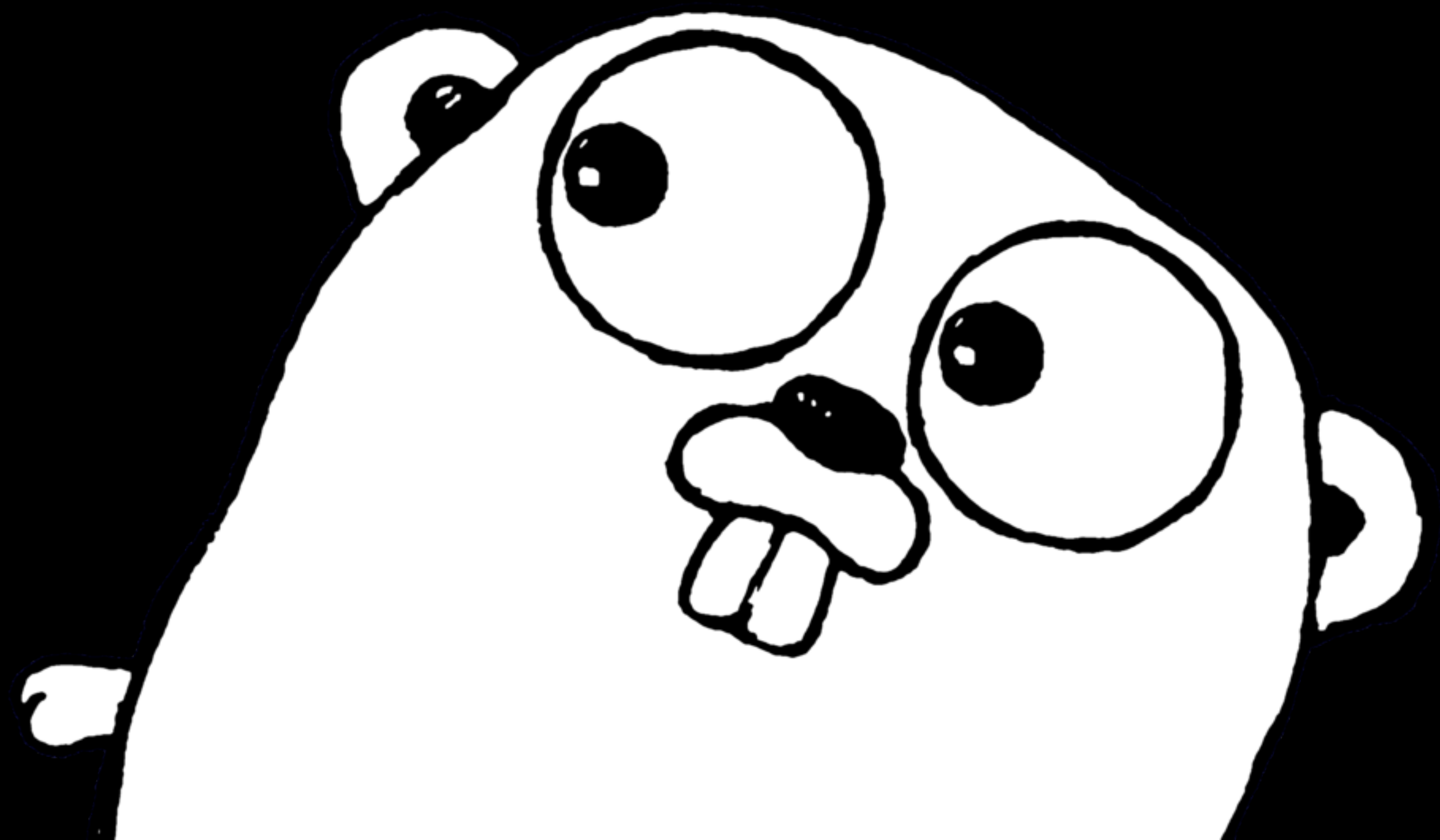
Continuous pprof collection in Go

Profiling thousands of agents running at the Edge

Ozan Sazak - June 2022



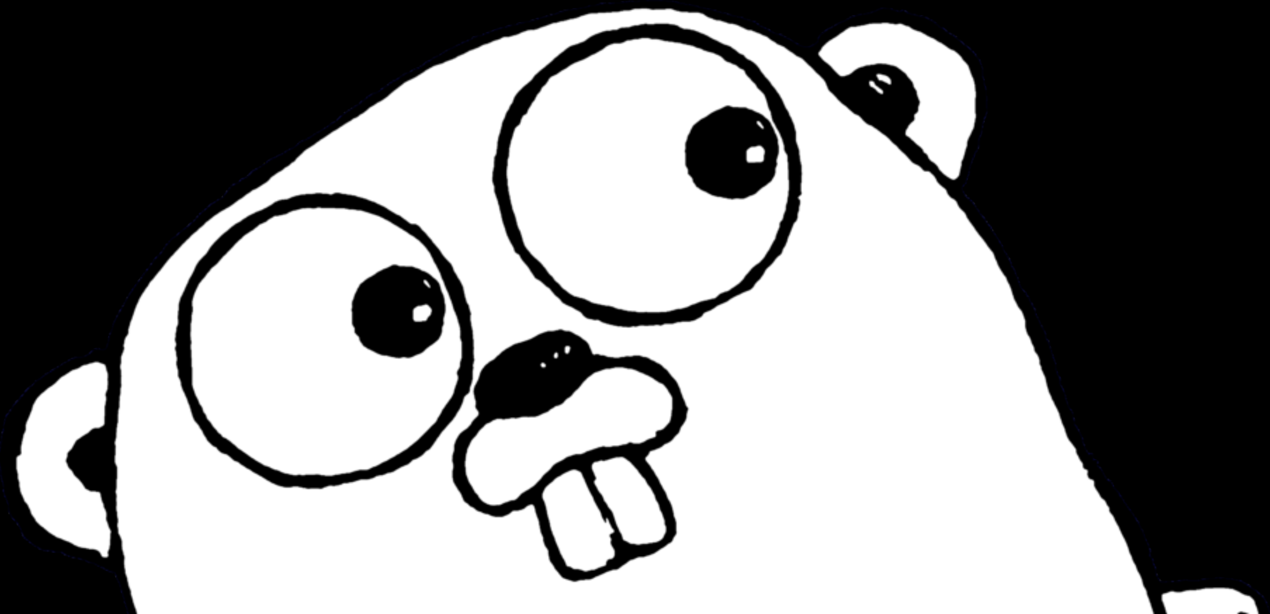
Who am I

- Ozan Sazak
- Senior CS student at METU
- Part-time SWE at Edge Delta

/ozansz

/in/ozan-sazak

sozan@edgedelta.com



Agenda

- What is Edge Delta?
- Why do we need continuous profiling?
- Brief introduction to pprof
- Examples of pprof usage
- How does our continuous profiling work?
- Demo on continuous profiling

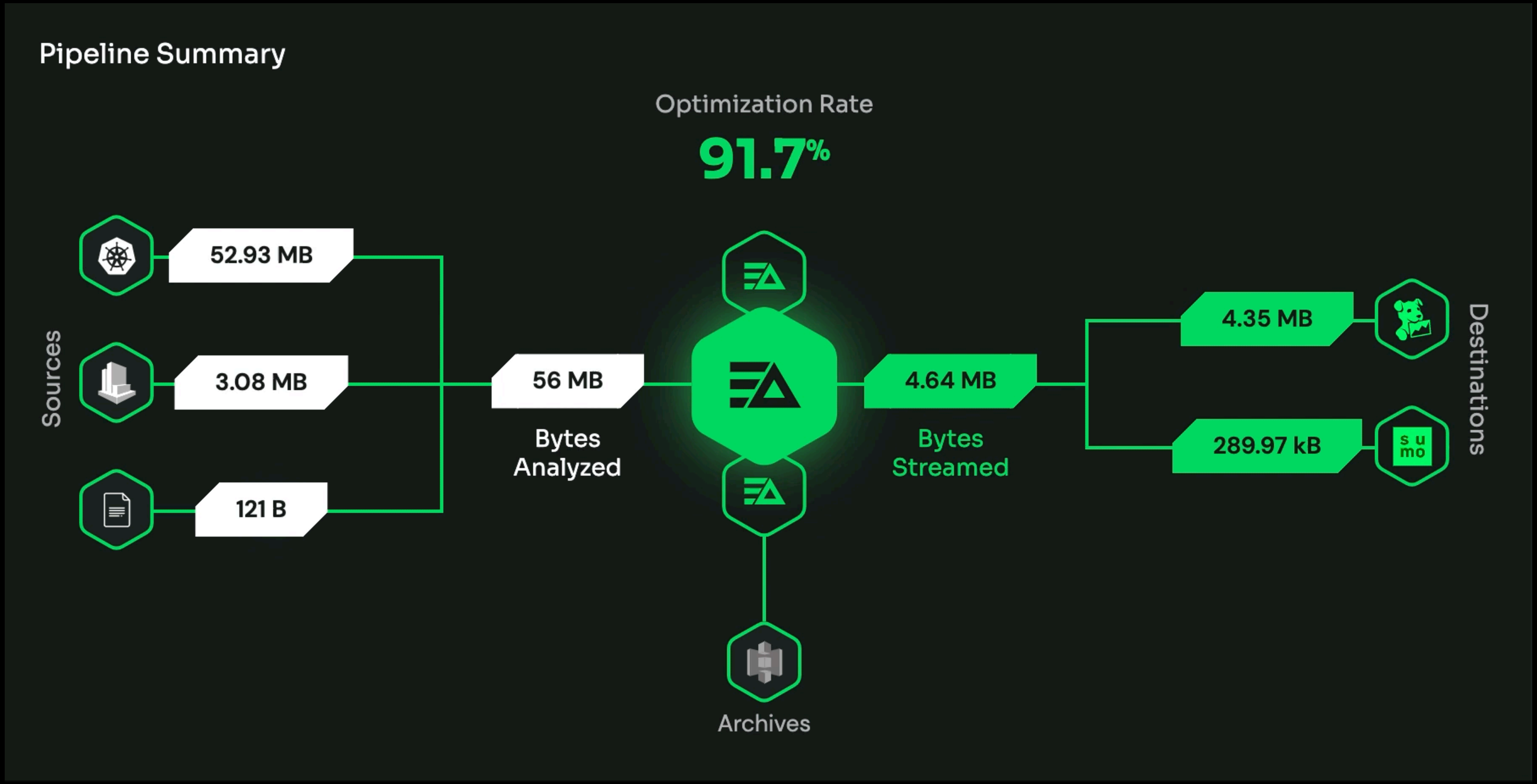
Edge Delta

- Continuous log observation with thousands of agents running on the edge
- Agents preprocess logs and filter/tag findings
- Agents run on customers’ machines with a single YAML config
- (We’re actually using it too, to observe our own services :))

EDGE DELTA

Edge Delta

Multiple input sources, processors, and output integrations through agent config



… why do we need continuous profiling?

- Agents run on customers’ **production** machines
  - They have to be **super performant**
  - It’s **critical** for us to have more visibility on our **bottlenecks**
- We run nightly performance tests before agent releases
  - Gives us insight and confidence over agent performance

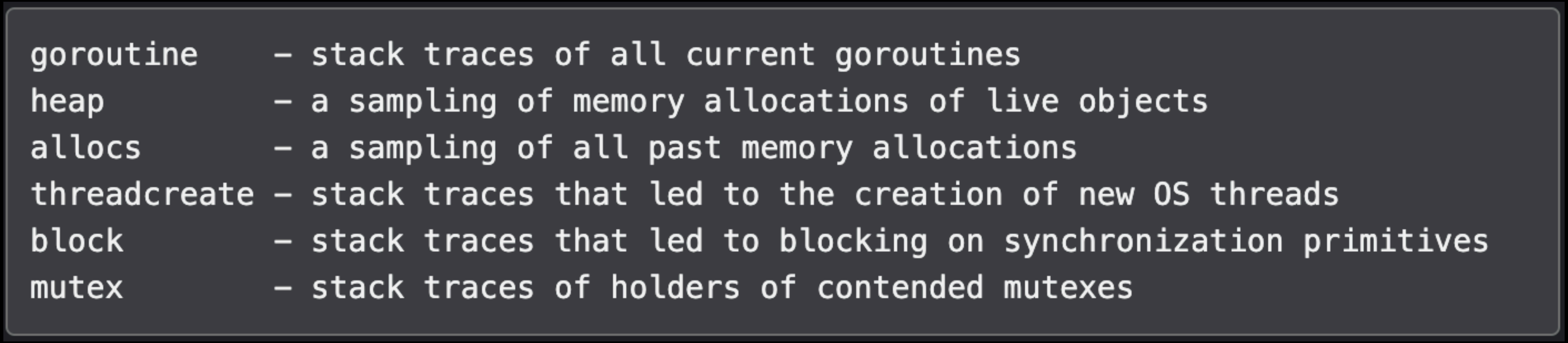
but…

There are always unforeseen cases on customer environment due to,

- Different ways of agent configuration
- Limited resources
- Millions of files
- …

What is profiling?

- Go is often used in programs with high-performance needs
- We also face memory leaks and bottlenecks
- Need to analyze CPU and memory intensive code for optimizations



```
goroutine    - stack traces of all current goroutines
heap         - a sampling of memory allocations of live objects
allocs       - a sampling of all past memory allocations
threadcreate - stack traces that led to the creation of new OS threads
block        - stack traces that led to blocking on synchronization primitives
mutex        - stack traces of holders of contended mutexes
```

The CPU profile is not available as a Profile. It has a special API, the StartCPUProfile and StopCPUProfile functions, because it streams output to a writer during profiling.

Source: pkg.go.dev/runtime/pprof

Different ways to collect profiles

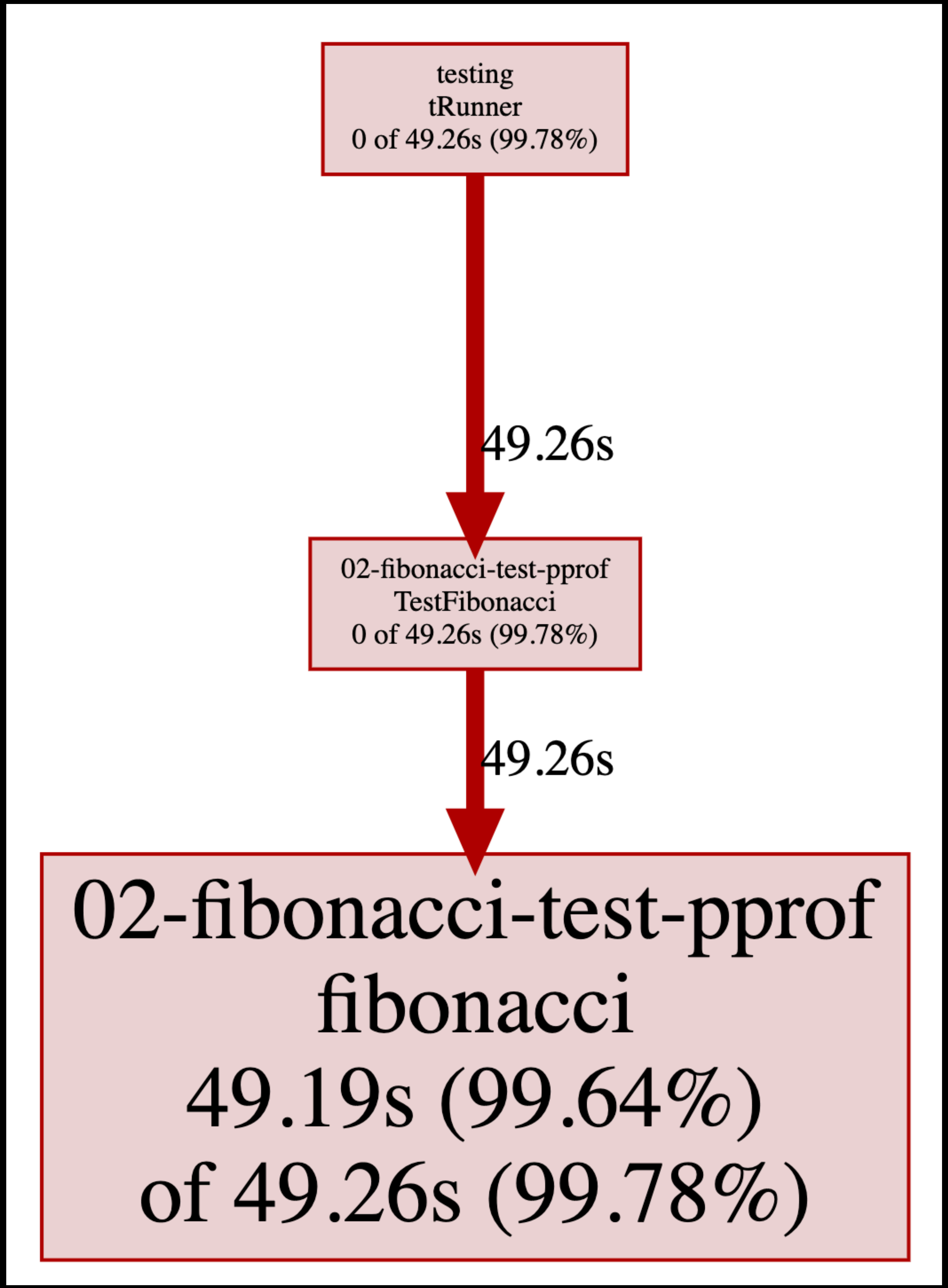
- net/http/pprof
  - Lets us download profiles directly on demand using the default HTTP server
  - /debug/pprof/{goroutine, heap, threadcreate, block, mutex, profile}
- runtime/pprof
  - Lets us generate and save profiles inside the code, in runtime

  -
    ```
    pprof.Lookup("heap")
    ```
- go test
  - Lets us dump profiles automatically after the test runs

`go test` with -cpuprofile

```go
func fibonacci(n uint64) uint64 {
	if n <= 1 {
		return n
	}
	return fibonacci(n-1) + fibonacci(n-2)
}
```

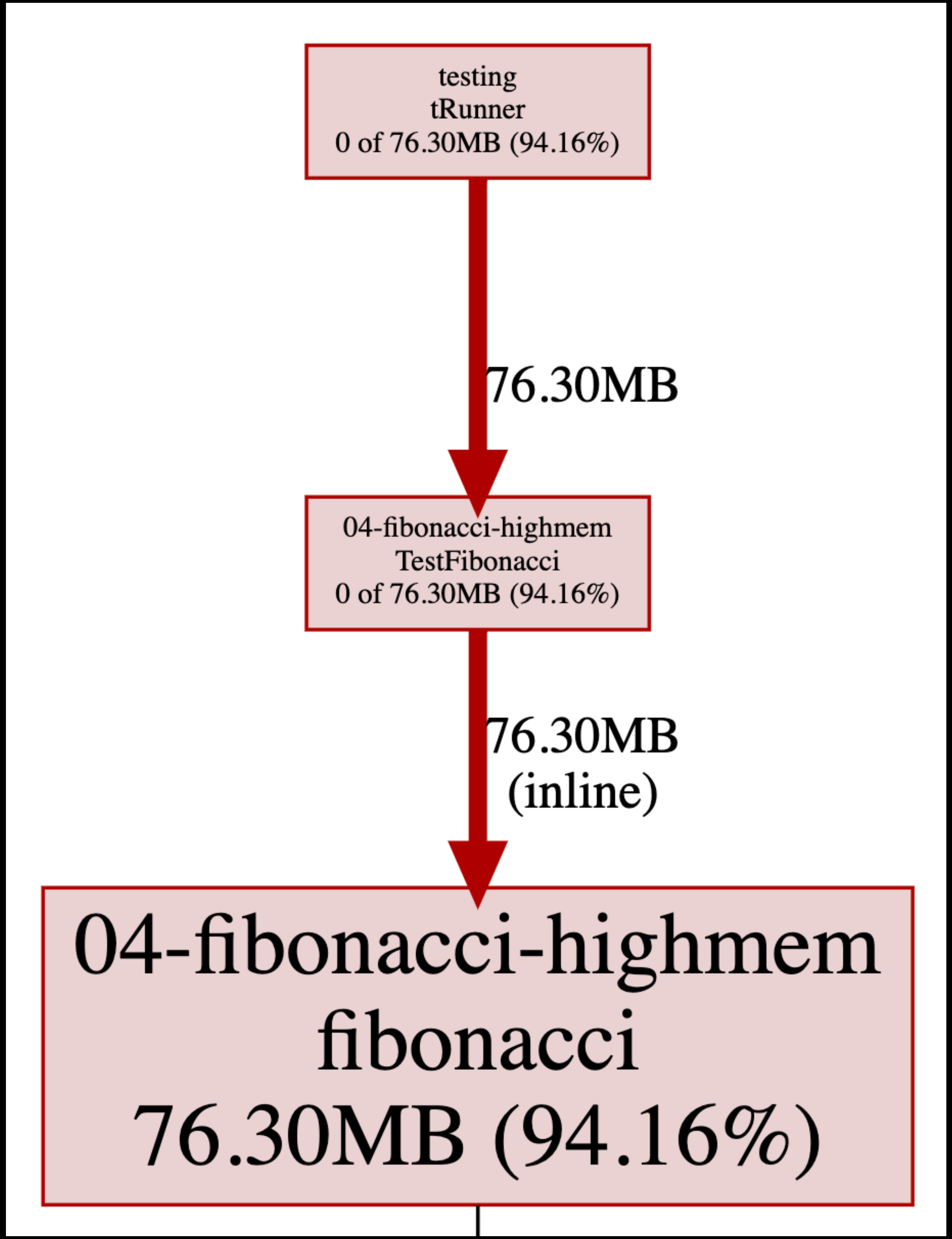
```
❯ go test -cpuprofile cpu.prof -bench .
```



`go test` with -memprofile

```go
func fibonacci(n uint64) uint64 {
    cache := make([]uint64, n+1)
    cache[0] = 0
    cache[1] = 1
    var i uint64
    for i = 2; i <= n; i++ {
        cache[i] = cache[i-1] + cache[i-2]
    }
    return cache[n]
}
```

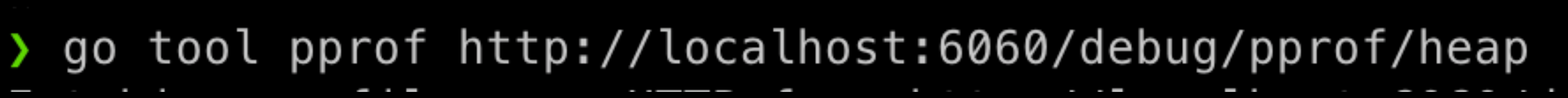
```
❯ go test -cpuprofile cpu.prof -memprofile mem.prof -bench .
```



...with pprof HTTP server

```go
import _ "net/http/pprof"

func main() {
  go func() {
    log.Println(http.ListenAndServe("localhost:6060", nil))
  }()
  // Do some memory and CPU intensive work here...
}
```



```
❯ go tool pprof http://localhost:6060/debug/pprof/heap
```

Some useful commands for `go tool pprof`

- topN: Shows the top N samples is the profile
- web: Generate a graph of the profile data in SVG format and opens it on the browser
- pdf: Generates a PDF file with the same graph as the web command does
- png: Generates a PNG file with the same graph as the web command does
- list func: Shows the source code of the func with the flat and cum metrics side by side

How does continuous profiling work?

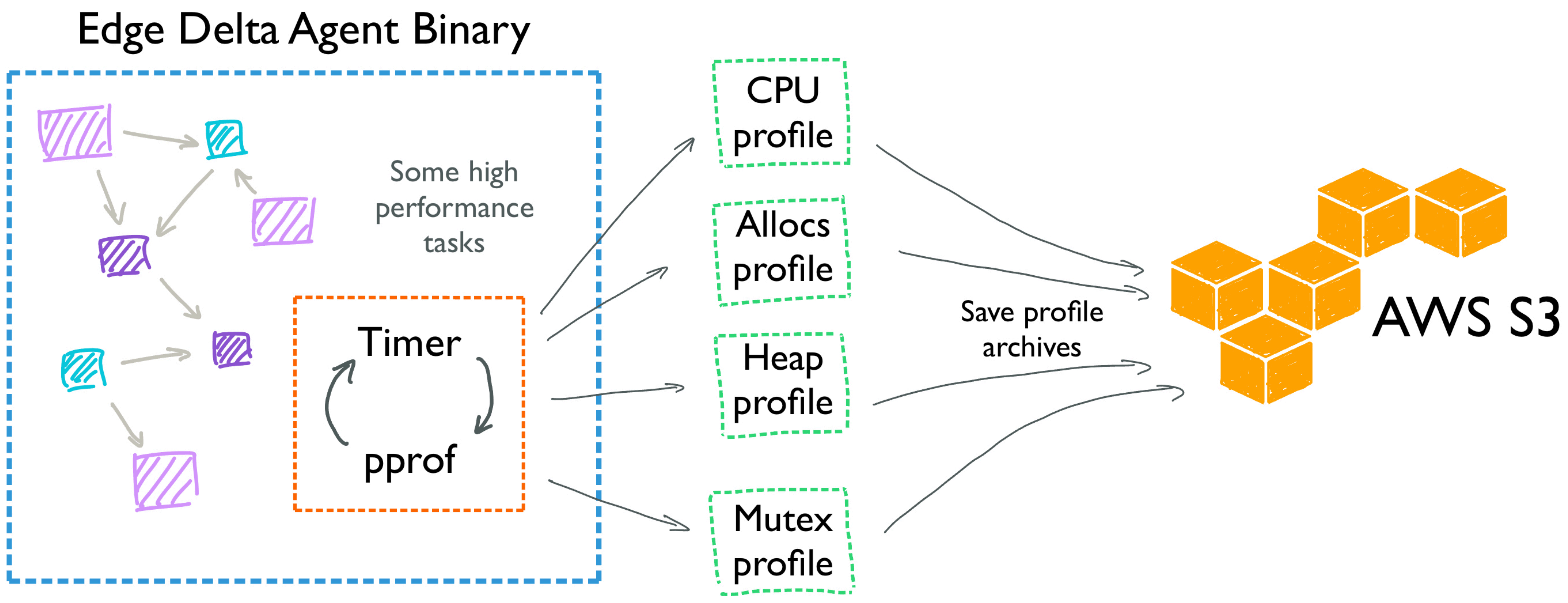
Possible ways to run pprof on runtime

- Run pprof HTTP server, and call specific pprof endpoints
  - The customer need to **expose a port** to internet on production machine
- Use **google/gops** to analyze agent program in runtime
  - The customer need to connect to container, take profile dumps and **share with us**
- Dump profiles through **go test**
  - Not applicable in the runtime

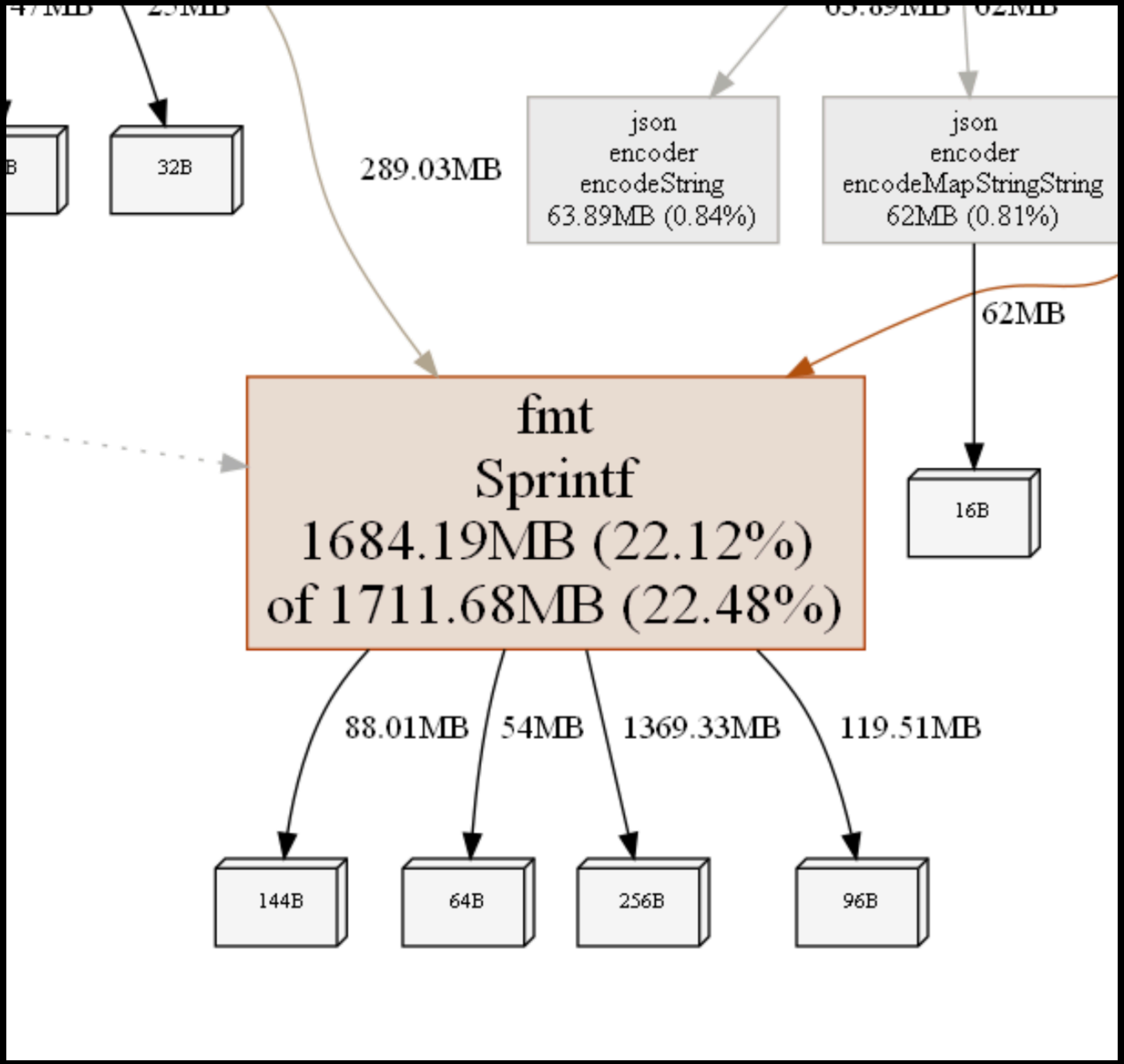
Solution: runtime/pprof

- Supports self-profiling on the agent binary runtime
- The pprof HTTP server actually uses runtime/pprof in background
- Supports various types of profiles

    heap, allocs, goroutine, threadcreate, block, mutex, cpu


Edge Delta Agent Binary
Some high performance tasks
Timer
pprof
CPU profile
Allocs profile
Heap profile
Mutex profile
Save profile archives
AWS S3

It actually works!



We're hiring!

EDGE DELTA

edgedelta.com/company/careers

sozan@edgedelta.com

Thank you for listening!