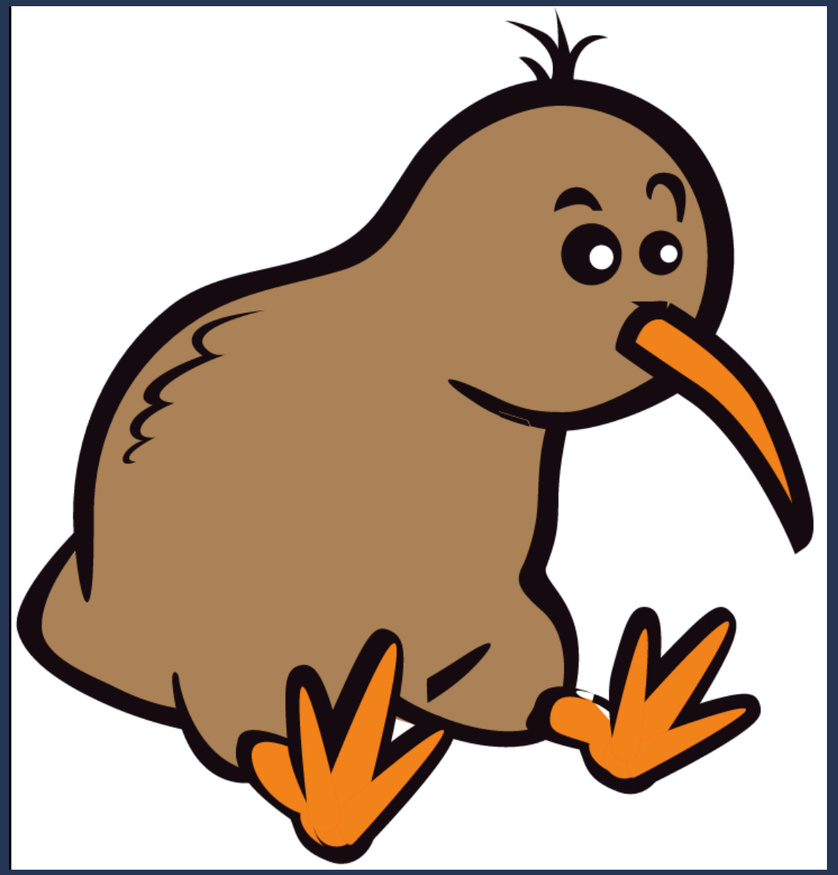
Uptime: Building Resilient Services in Go

Blake Caldwell
http://blakecaldwell.net
https://twitter.com/blakecaldwell

Fog Creek Software



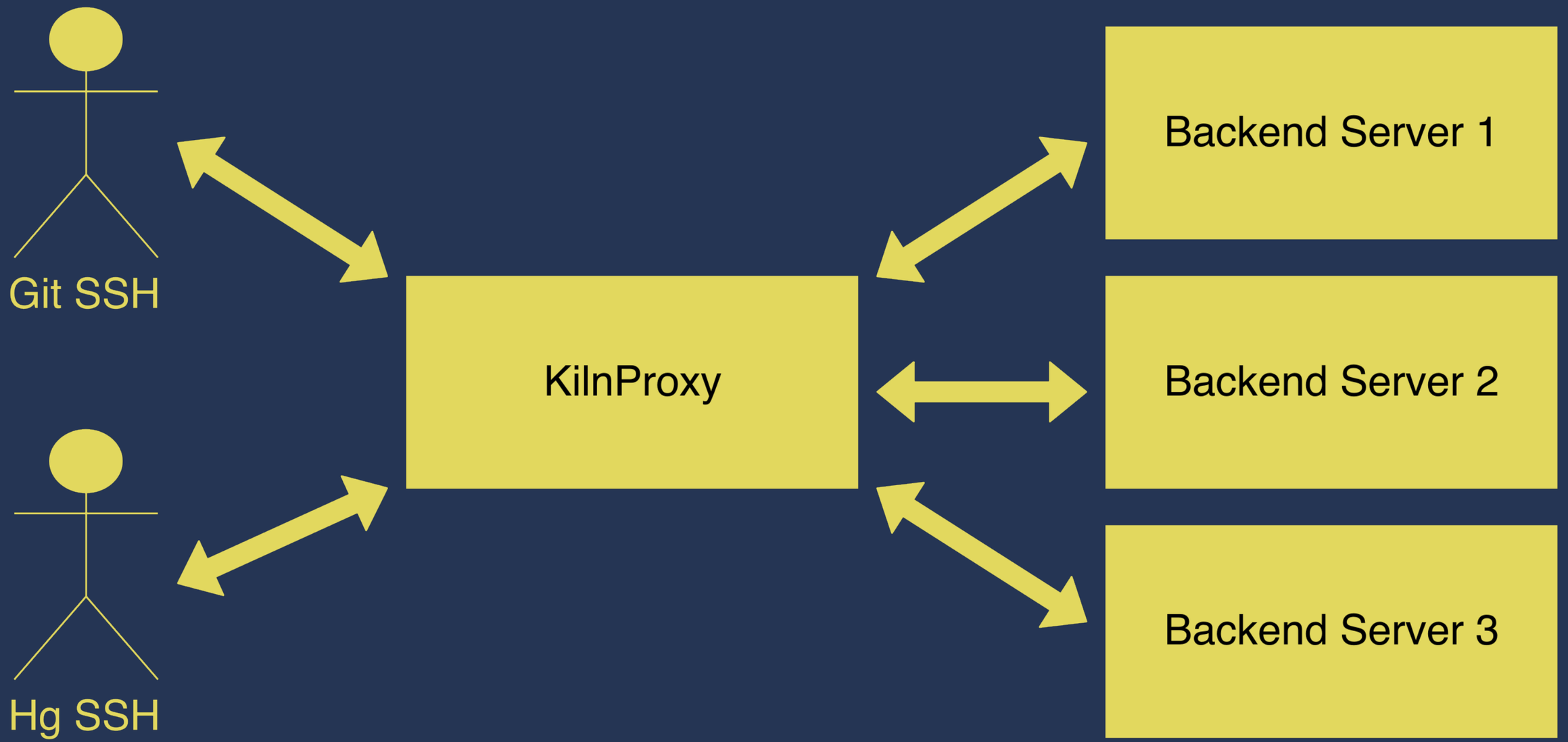
Git & Hg Source Code Hosting

Kiln

2014 Google I/O: What's Go?


Solution Found!
*Now to find the problem

Kiln's SSH Reverse Proxy


Git SSH
Hg SSH
KilnProxy
Backend Server 1
Backend Server 2
Backend Server 3

Why Rewrite?

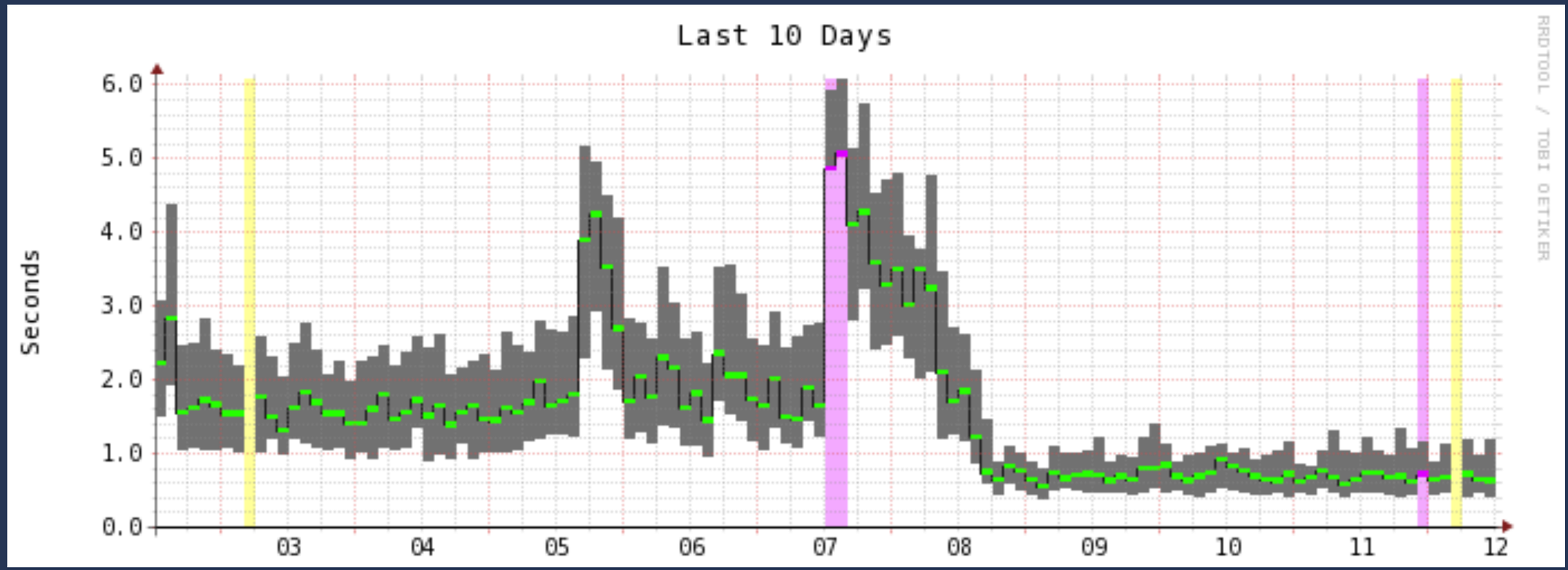
Tons of Concurrency

1. Accepts SSH connection (1 goroutine)
2. Authenticates via public/private key
3. Connects to backend server
4. Proxies STDIN, STDOUT, and STDERR (3 goroutines)

How'd It Go?

KilnProxy: From Python to Go

Clone times halved!



Resiliency: The Process

Careful Coding:
Error Handling & Clean-up

Handle All Errors

```
resourceA, err := OpenResourceA()
if err != nil {
    return nil, err
}
defer resourceA.Close()
```

What About `nil`?

Back To Our Example:

```
resourceA, err := OpenResourceA()
if err != nil {
    return nil, err
}
defer resourceA.Close()
```

Be Careful!

```
// can return nil, and that's not an error!
resourceA, err := OpenResourceA()
if err != nil {
    return nil, err
}
defer resourceA.Close()     // panic on nil????
```

Not Necessarily.

One Solution:

```
// can return nil, and that's not an error!
resourceA, err := OpenResourceA()
if err != nil {
    return nil, err
}
defer func(){
    if resourceA != nil {
        resourceA.Close()
    }
}()
```

Gross.

Make deferred Methods nil-Safe

```
// Don't forget: resource might be nil!
func (resource *Resource) Close() {
    if resource != nil {
        // .. clean up
    }
}
```

Much Better!

```
// can return nil, and that's not an error!
resourceA, err := OpenResourceA()
if err != nil {
    return nil, err
}
defer resourceA.Close()     // will never panic!
```

Careful Coding:
Channels

Channel Axioms [1]

1. A send to a nil channel **blocks forever**
2. A receive from a nil channel **blocks forever**
3. A send to a closed channel **panics**
4. A receive from a closed channel returns the zero value immediately

---

[1] http://dave.cheney.net/2014/03/19/channel-axioms

Careful Coding:

Panics!

You Can Recover From Panics

- ... but you shouldn't always do so!
- Only recover if you're sure it's okay
- Panic recovery is for current goroutine
- At very least, log the stack trace

Careful Coding:
Avoid Race Conditions!

Go's Race Detector:

- Reports when variable access is not synchronized
- Crashes with a full stack trace, including the read and write goroutines
- Should be used in unit tests, development, and testing environments

Race Detector Output:

```
==================
WARNING: DATA RACE
Read by goroutine 5:
  main.func·001()
      race.go:14 +0x169

Previous write by goroutine 1:
  main.main()
      race.go:15 +0x174

Goroutine 5 (running) created at:
  time.goFunc()
      src/pkg/time/sleep.go:122 +0x56
  timerproc()
      src/pkg/runtime/ztime_linux_amd64.c:181 +0x189
==================
```

Enable Race Detection:

```
$ go test -race mypkg    // to test the package
$ go run -race mysrc.go  // to run the source file
$ go build -race mycmd   // to build the command
$ go install -race mypkg // to install the package
```

Careful Coding: Implement Timeouts

Network Timeouts:

- network dial timeout
- network connection inactivity timeout
- total connection timeout

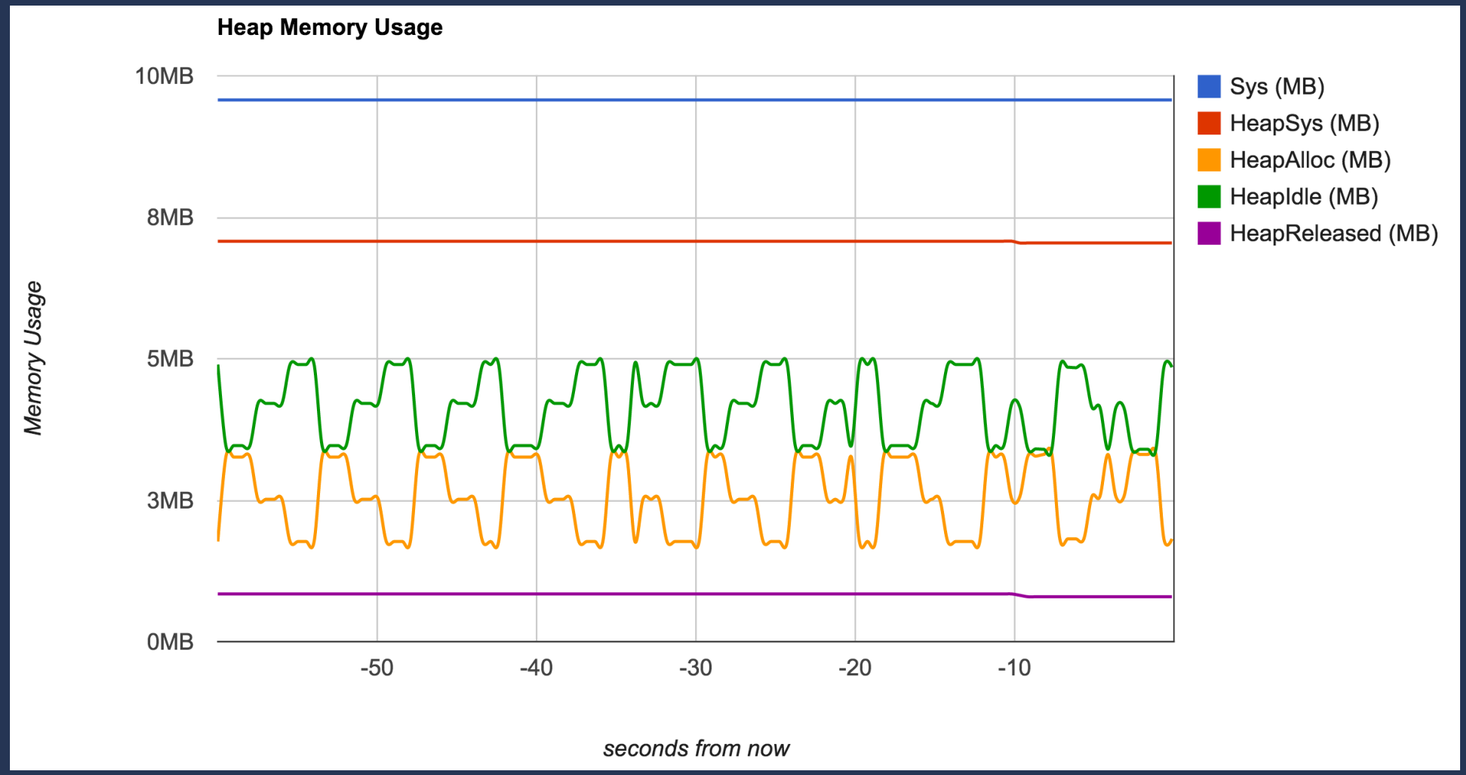
TEST ALL THE THINGS!

Know

Your Service

Know Your Service:
How Does It Use Memory?

Profile It!



http://github.com/fogcreek/profiler

What to Watch

- How much memory does the service use when idle?
- How much memory per connection?
- Does the system reclaim memory that's no longer used?
- What's the garbage collector doing? `GODEBUG=gctrace=1`
- Where is memory allocated? (PPROF)

Know Your Service:

PPROF

PPROF Inspects Your Running Process

- Blocking profile
- Goroutine count and full stacktraces
- Heap profile
- Stacktraces that lead to thread creations

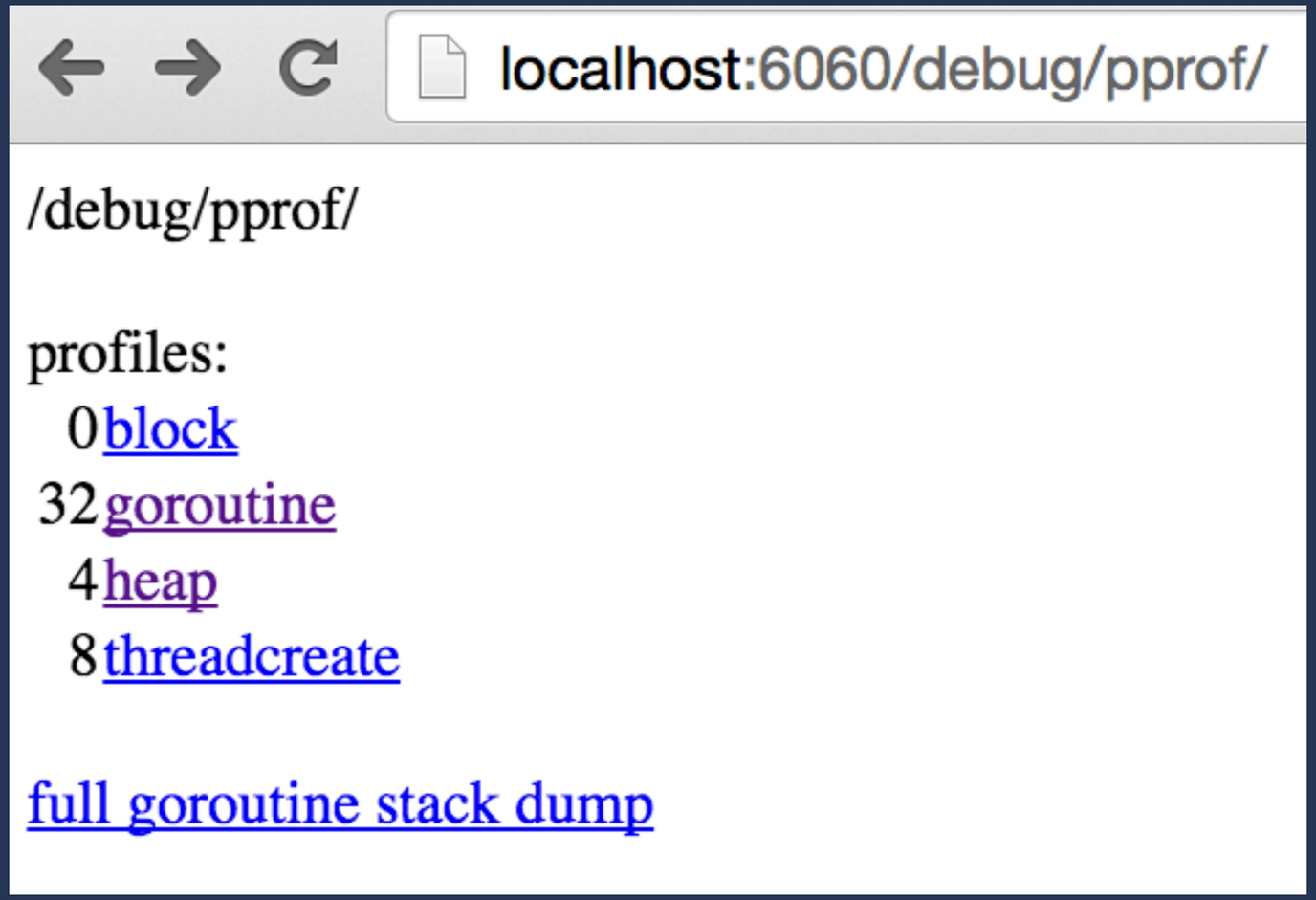
Enabling PPROF:

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

func main() {
    http.ListenAndServe(":6060", nil)

    // ...
}
```

PPROF Main Page



Don't leak goroutines!

Use PPROF To Tell You:

- How many goroutines when nobody is connected?
- How many goroutines per connection?
- Are all goroutines cleaned up after all connections close?

PPROF: Goroutine Page

```
goroutine profile: total 13
1 @ 0x1e5ba6 0x1e594d 0x1e1a35 0xf3edf 0xf404e 0xd24f1 0xd3c5d 0xd462a 0xd2047 0x42e21
#       0x1e5ba6        runtime/pprof.writeRuntimeProfile+0xd6  /Users/blake/go-1.4/src/runtime/pprof/pprof.go:540
#       0x1e594d        runtime/pprof.writeGoroutine+0x9d       /Users/blake/go-1.4/src/runtime/pprof/pprof.go:502
#       0x1e1a35        runtime/pprof.(*Profile).WriteTo+0xd5   /Users/blake/go-1.4/src/runtime/pprof/pprof.go:229
#       0xf3edf         net/http/pprof.handler.ServeHTTP+0x35f  /Users/blake/go-1.4/src/net/http/pprof/pprof.go:169
#       0xf404e         net/http/pprof.Index+0x15e              /Users/blake/go-1.4/src/net/http/pprof/pprof.go:181
#       0xd24f1         net/http.HandlerFunc.ServeHTTP+0x41     /Users/blake/go-1.4/src/net/http/server.go:1265
#       0xd3c5d         net/http.(*ServeMux).ServeHTTP+0x17d    /Users/blake/go-1.4/src/net/http/server.go:1541
#       0xd462a         net/http.serverHandler.ServeHTTP+0x19a  /Users/blake/go-1.4/src/net/http/server.go:1703
#       0xd2047         net/http.(*conn).serve+0xb57            /Users/blake/go-1.4/src/net/http/server.go:1204

1 @ 0x1a665 0x1a6d8 0x1d30f 0x1d09d 0xf04f9 0x42cb 0x1a3a3 0x42e21
#       0xf04f9 sync.(*WaitGroup).Wait+0x169    /Users/blake/go-1.4/src/sync/waitgroup.go:132
#       0x42cb  main.main+0x7b                  /Users/blake/Documents/Development/Personal/src/github.com/wblakecaldwell/sump/server/main.go:112
#       0x1a3a3 runtime.main+0xf3               /Users/blake/go-1.4/src/runtime/proc.go:63

1 @ 0x1a665 0x1a6d8 0x1a4fe 0x42e21
#       0x1a665 runtime.gopark+0x105            /Users/blake/go-1.4/src/runtime/proc.go:131
#       0x1a6d8 runtime.goparkunlock+0x48       /Users/blake/go-1.4/src/runtime/proc.go:136
#       0x1a4fe runtime.forcegchelper+0xce      /Users/blake/go-1.4/src/runtime/proc.go:99

1 @ 0x1a665 0x1a6d8 0x13d2c 0x42e21
#       0x1a665 runtime.gopark+0x105            /Users/blake/go-1.4/src/runtime/proc.go:131
#       0x1a6d8 runtime.goparkunlock+0x48       /Users/blake/go-1.4/src/runtime/proc.go:136
#       0x13d2c runtime.bgsweep+0xbc            /Users/blake/go-1.4/src/runtime/mgc0.go:98

1 @ 0x1a665 0x1a6d8 0x1332a 0x42e21
#       0x1a665 runtime.gopark+0x105            /Users/blake/go-1.4/src/runtime/proc.go:131
#       0x1a6d8 runtime.goparkunlock+0x48       /Users/blake/go-1.4/src/runtime/proc.go:136
#       0x1332a runtime.runfinq+0xba            /Users/blake/go-1.4/src/runtime/malloc.go:727
```

PPROF:

From the Command Line

What Are Your GoRoutines Doing?

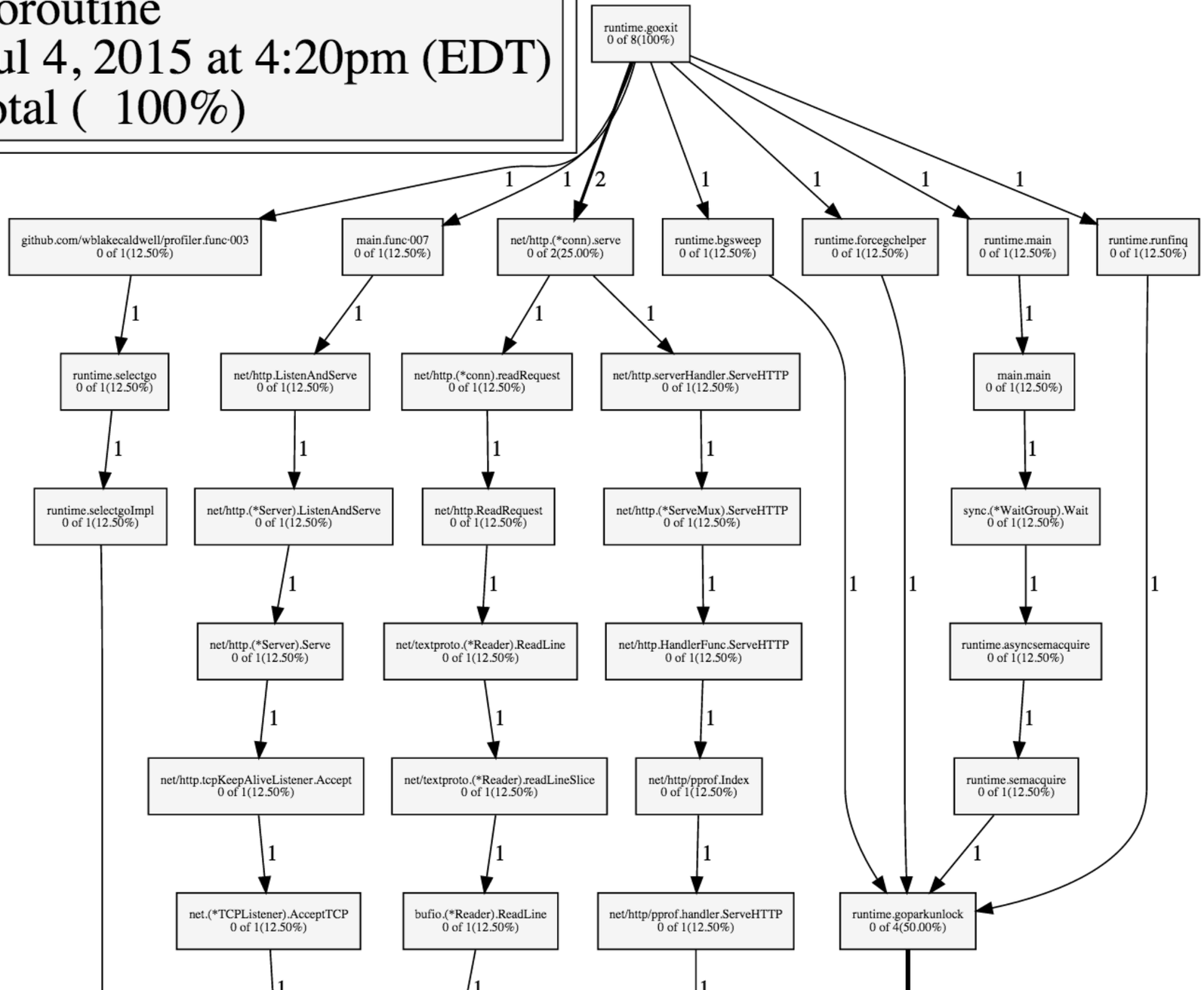
```
$ go tool pprof ./server http://localhost:6060/debug/pprof/goroutine

(pprof) top5

11 of 11 total (  100%)
Showing top 5 nodes out of 49 (cum >= 1)
      flat  flat%   sum%        cum   cum%
         9 81.82% 81.82%          9 81.82%  runtime.gopark
         1  9.09% 90.91%          1  9.09%  runtime.notetsleepg
         1  9.09%   100%          1  9.09%  runtime/pprof.writeRuntimeProfile
         0     0%   100%          1  9.09%  bufio.(*Reader).ReadLine
         0     0%   100%          1  9.09%  bufio.(*Reader).ReadSlice

(pprof) web
```

File: server
Type: goroutine
Time: Jul 4, 2015 at 4:20pm (EDT)
8 of 8 total (100%)



Who's Allocating Heap Memory?

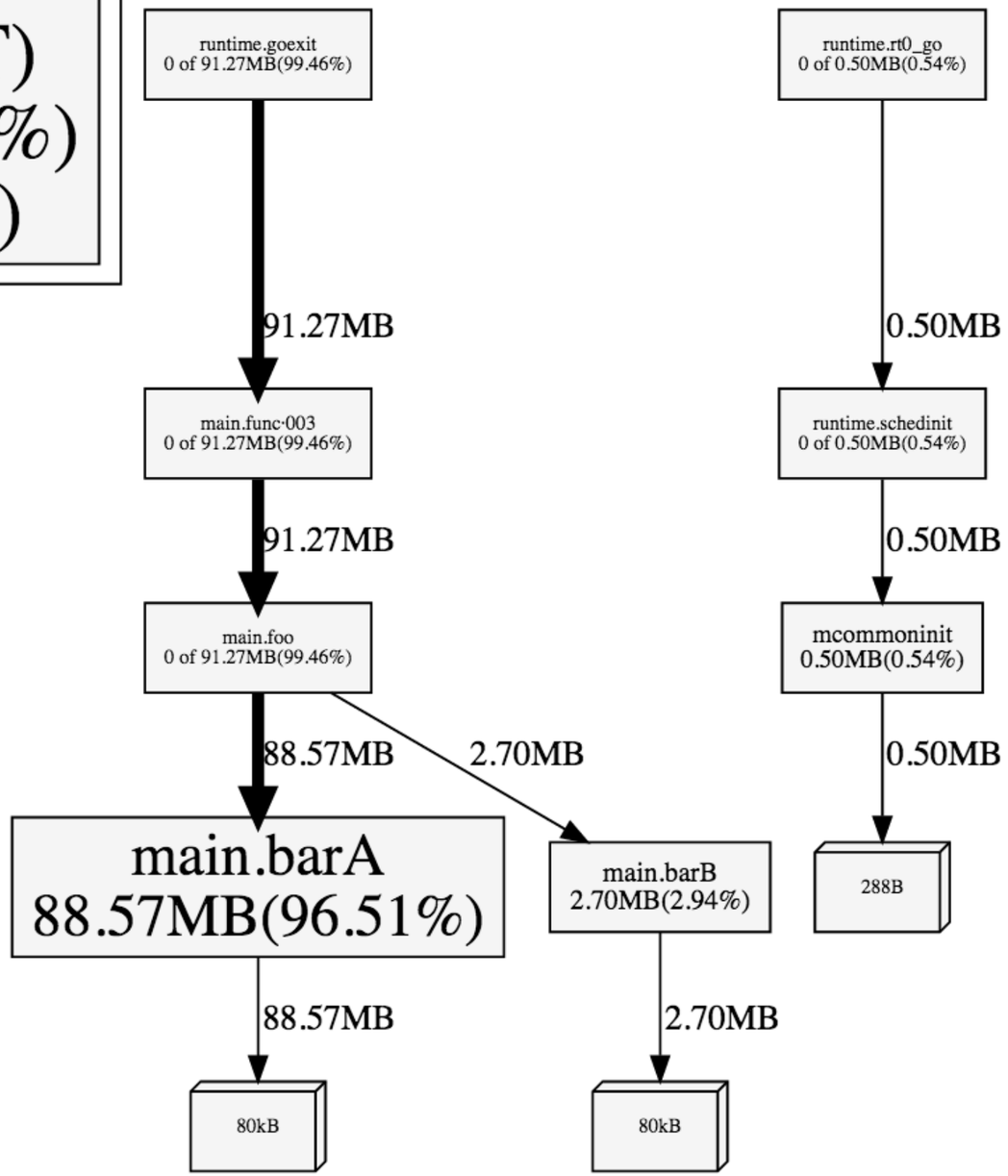
```
$ go tool pprof ./server http://localhost:6060/debug/pprof/heap

(pprof) top5

2362.41kB of 2362.41kB total (  100%)
Dropped 28 nodes (cum <= 11.81kB)
      flat  flat%   sum%        cum   cum%
 1850.27kB 78.32% 78.32%  1850.27kB 78.32%  github.com/wblakecaldwell/profiler.func·002
  512.14kB 21.68%   100%   512.14kB 21.68%  mcommoninit
         0     0%   100%  1850.27kB 78.32%  runtime.goexit
         0     0%   100%   512.14kB 21.68%  runtime.rt0_go
         0     0%   100%   512.14kB 21.68%  runtime.schedinit

(pprof) web
```

File: server
Type: inuse_space
Time: Jul 4, 2015 at 4:29pm (EDT)
91.77MB of 91.77MB total (100%)
Dropped 1 node (cum <= 0.46MB)



Know Your Service:
Watch It Run

/info Endpoint

```
{
    Version: "1.0.275-b244a2b9b8-20150202.163449",
    StartTimeEpochSecs: 1430515329,
    CurrentTimeEpocSecs: 143117131,
    Uptime: "167h10m2s"
}
```

Managing Service Version

```
Version: "1.0.275-b244a2b9b8-20150202.163449"
```

Which is:

```
<major>.<minor>.<commit#>-<Git SHA>-<date>.<time>
```

Managing Service Version

Version is stored in a global variable, set by your build script

In code:

```
var ServiceVersion string
```

Build script:

```
$ go build -ldflags \
        "-X main.ServiceVersion \
            1.0.275-b244a2b9b8-20150202.163449" \
        kilnproxy
```

Keep Good Logs!

- Create a semi-unique string per request
- Use this request string as a prefix in all log entries
- Always log at least the start and end of a request

Who's Currently Connected?

/connections Endpoint

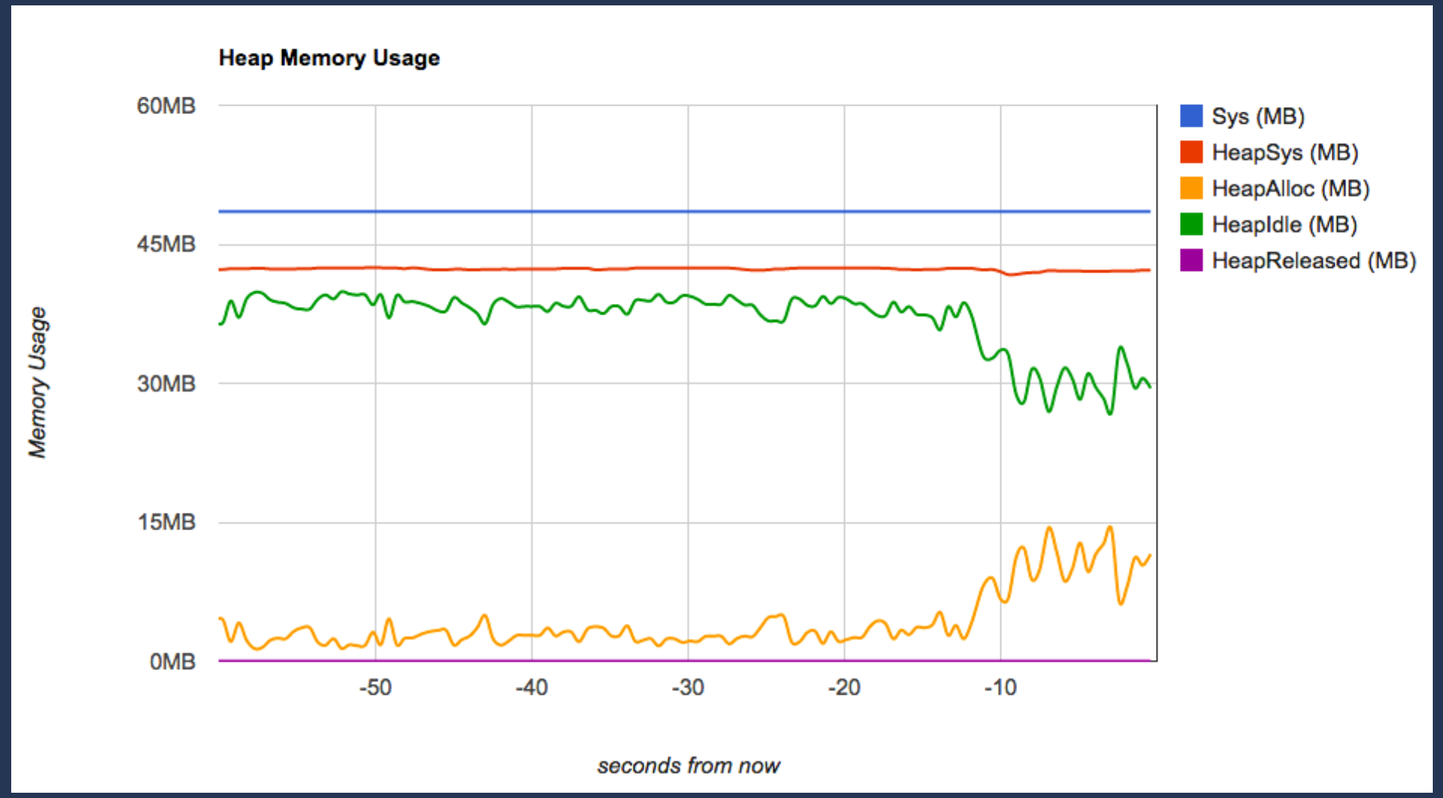

```
{
    "CurrentUserCount":1,
    "CurrentlyAuthenticatedUsers":
    [
        {"Account":"aviato",
        "Name":"Erlich Bachman",
        "PublicKeyName":"Build Server",
        "SessionKey":"106abc0c",
        "SessionDuration":"25m4s"
        }
    ]
}
```

Drain and Die

Game Day.

KilnProxy Using 40MB more memory than normal!!!

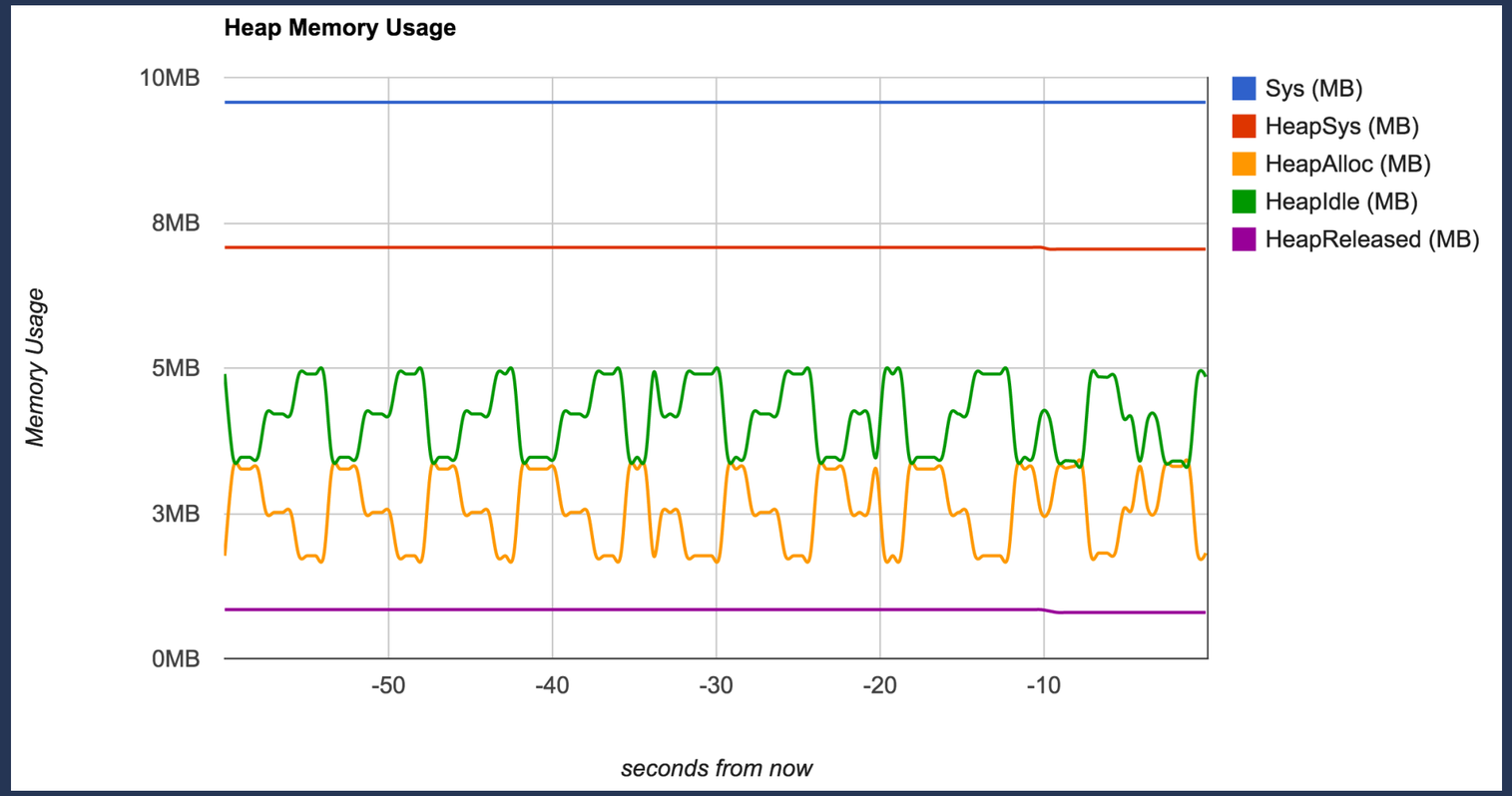
Profiler Tells Me:
This memory is still in use



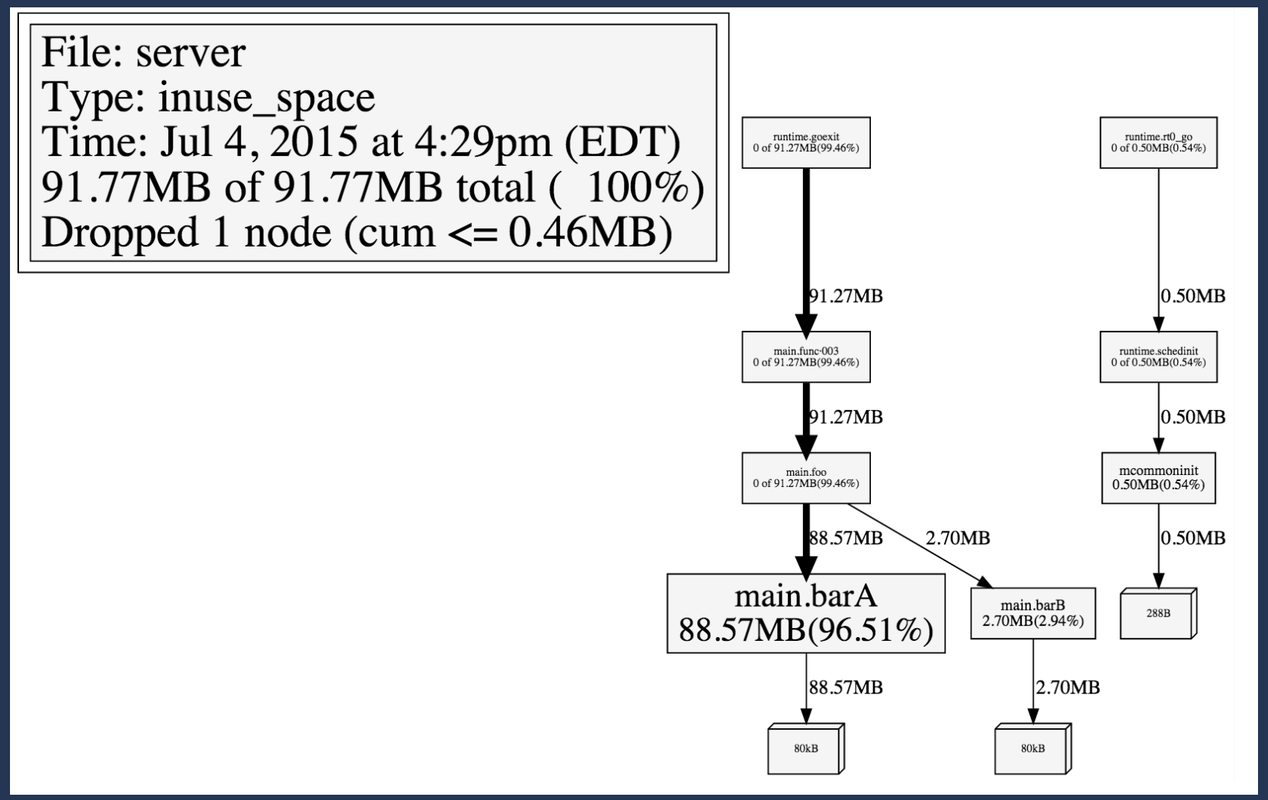
/connections Tells Me:
Initech is connected 10 times


```
{
    "CurrentUserCount":25,
    "CurrentlyAuthenticatedUsers":
    [
        {"Account":"initech", "Name":"Peter Gibbons", ...
        {"Account":"initech", "Name":"Peter Gibbons", ...
        {"Account":"initech", "Name":"Peter Gibbons", ...
        {"Account":"initech", "Name":"Peter Gibbons", ...
        {"Account":"initech", "Name":"Peter Gibbons", ...
        ...
    ]
}
```

Dev Profiling Told Me:
Each connection uses 4MB of memory



Dev PPROF Told Me:
Most of that 4MB is SSH internals



Wolfram Alpha Tells Me:
4MB x 10 = 40MB

We Contact Initech.

Timeouts Make Sure That:

Their connections will be closed

Prod Profiling Told Me:
This memory will be reclaimed

So.....

Uptime: Preserved.

Thanks!

Blake Caldwell
http://blakecaldwell.net
https://twitter.com/blakecaldwell

Memory Profiler: http://github.com/fogcreek/profiler

Credit: Images

- **Thinking Man**: خالد منتصف - License: CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)
- **Race car**: Newlin Keen, Kollins Communications, Inc