High-level Intro to Benchmarking and Profiling 

Håvard Anda Estensen, Vipps

Agenda

- Why profile your system?
- Benchmarking vs. Profiling
- Benchmark functions
- Profile systems
- Dos and Don’ts
- Do you even need this?!

Why Profile our System?

- We want our system to be fast 🚀
  - Amazon: +100ms => -1% revenue
  - Our program is only part of the system
- Don’t guess what the performance should be, measure it!

“The First Rule of Program Optimization:
Don't do it”

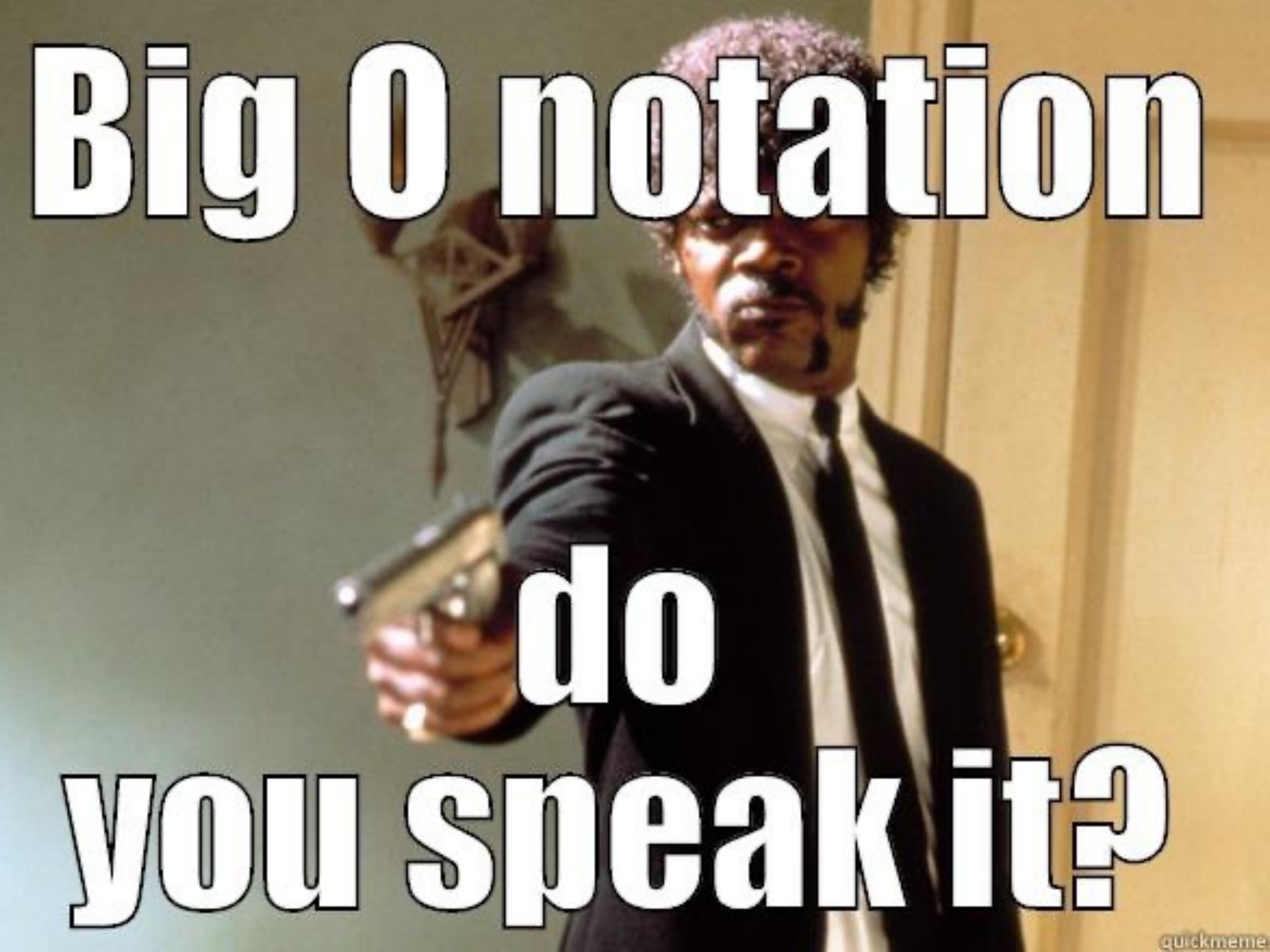
“The Second Rule of Program Optimization
(for experts only!):
Don't do it yet.”

– Michael A. Jackson

Benchmarking vs. Profiling

- Benchmarking measure the execution time of a function
- Profiling is about finding performance bottlenecks

Benchmark Functions


Big O notation
do
you speak it?
quickmeme

```
$ time wc -w sherlock.txt
 1095695 sherlock.txt
wc -w sherlock.txt
        0.02s user
        0.00s system
        95% cpu
        0.027 total
```

```go
func ConcatStrings(words []string) string {
    var str string
    for _, word := range words {
        str += word
    }
    return str
}
```

```go
func BenchmarkConcatString(b *testing.B) {
    words := make([]string, 100)
    for i := range words {
        words[i] = RandStringBytes(length)
    }
    b.Run("concat string", func(b *testing.B) {
        for i := 0; i < b.N; i++ {
            ConcatStrings(words)
        }
    }
}
```

```
$ go test -bench=ConcatStrings
BenchmarkConcatStrings/concat_strings-12
103142
10387 ns/op
```

```
b.Run(“concat string”, func(b *testing.B) {
    b.ReportAllocs()
    …
}
```

```
$ go test -bench=ConcatStrings
BenchmarkConcatStrings/concat_strings-12
    103142
    10387 ns/op
    53488 B/op
    99 allocs/op
```

```
b.Run(“concat string”, func(b *testing.B) {
    b.ReportAllocs()
    …
}
```

```
$ go test -bench=ConcatStrings
BenchmarkConcatStrings/concat_strings-12
    110922
    10566 ns/op
    53488 B/op
    99 allocs/op
```

```
$ go get golang.org/x/perf/cmd/benchstat
$ go test -bench=BenchmarkConcatString -count=8 >> old.txt
$ benchstat old.txt
name                          time/op
ConcatString/concat_string-12   11.3µs ± 9%
```


THIS IS FINE

But why are we concatenating 100 strings? Is this a realistic use-case for our program?

Is this our performance bottleneck?

CPU Profile

- Interrupt program every 10ms and record collection of stack traces
- Save profile to disk
- If the program runs for less than 10ms the file will be empty 🙃

```
import _ “net/http/pprof"

            or

import "runtime/pprof"

            or

"github.com/pkg/profile"

func main() {
    defer profile.Start(profile.CPUProfile, profile.ProfilePath(“.”))
    …
}
```

```
“github.com/pkg/profile"

func main() {
    defer profile.Start(profile.CPUProfile, profile.ProfilePath(“.”))
    …
}

$ go tool pprof -http=:8080 mem.pprof
```

```
import _ “net/http/pprof”
```

```
$ go tool pprof -http=:8080 http:/localhost:8080/debug/pprof/profile
```

Demo: leftpad

You pay for memory allocation more than once. The first time when you allocate it. And then every time the garbage collector runs.

“Reduce/Reuse/Recycle” 💚

–@bboreham

Stack vs. Heap

- Each thread has its own stack
- Threads shares heap
- Coordination is expensive
- GC reclaim stuff on the heap
- Conclusion: Try to allocate on the stack

```
defer profile.Start(profile.TraceProfile, profile.ProfilePath(“.”)).Stop()
```

```
$ go tool trace trace.out
```

Dos and Don’ts

- Use idle and dedicated hw
- Throttle CPU and turn off power saving
- Run benchmarks multiple times
- Use ~~complicated math~~ benchstat to measure ~~variance~~ noise
- Regression benchmarks are useful
- Don’t do it!

Who Should Optimize?

- Do you have enough RAM?
- Can you avoid creating a distributed system?
- Does you hourly report take two hours to run?
- Every optimization comes at a cost of time and most often complexity

“Don’t optimize, but also don’t be dumb”

–Damian Gryski

Conclusion

- Don’t spend time prematurely optimizing
- Profile with Idle CPU on same machine
- Preferably multiple times with perflock to avoid 🔥
- Heap allocation is expensive, stack allocation is cheap
- Spend time on improving your bottleneck

Resources

- Code for the presentation
  https://github.com/estensen/go-perf
- Optimizing Go Code Without a Blindfold
  https://www.youtube.com/watch?v=oE_vm7KeV_E
- Two Go Programs, Three Different Profiling Techniques
  https://www.youtube.com/watch?v=nok0aYiGiYA
- Profiling and optimizing Go web applications
  https://artem.krylysov.com/blog/2017/03/13/profiling-and-optimizing-go-web-applications/

@estensen on Keybase