

# WHAT I WISH I KNEW BEFORE SCALING UBER TO 1,000 SERVICES

MATT RANNEY

U B E R



# WHAT I WISH I KNEW BEFORE SCALING UBER TO 1,000 SERVICES

MATT RANNEY

U B E R







**As of April 2016:**

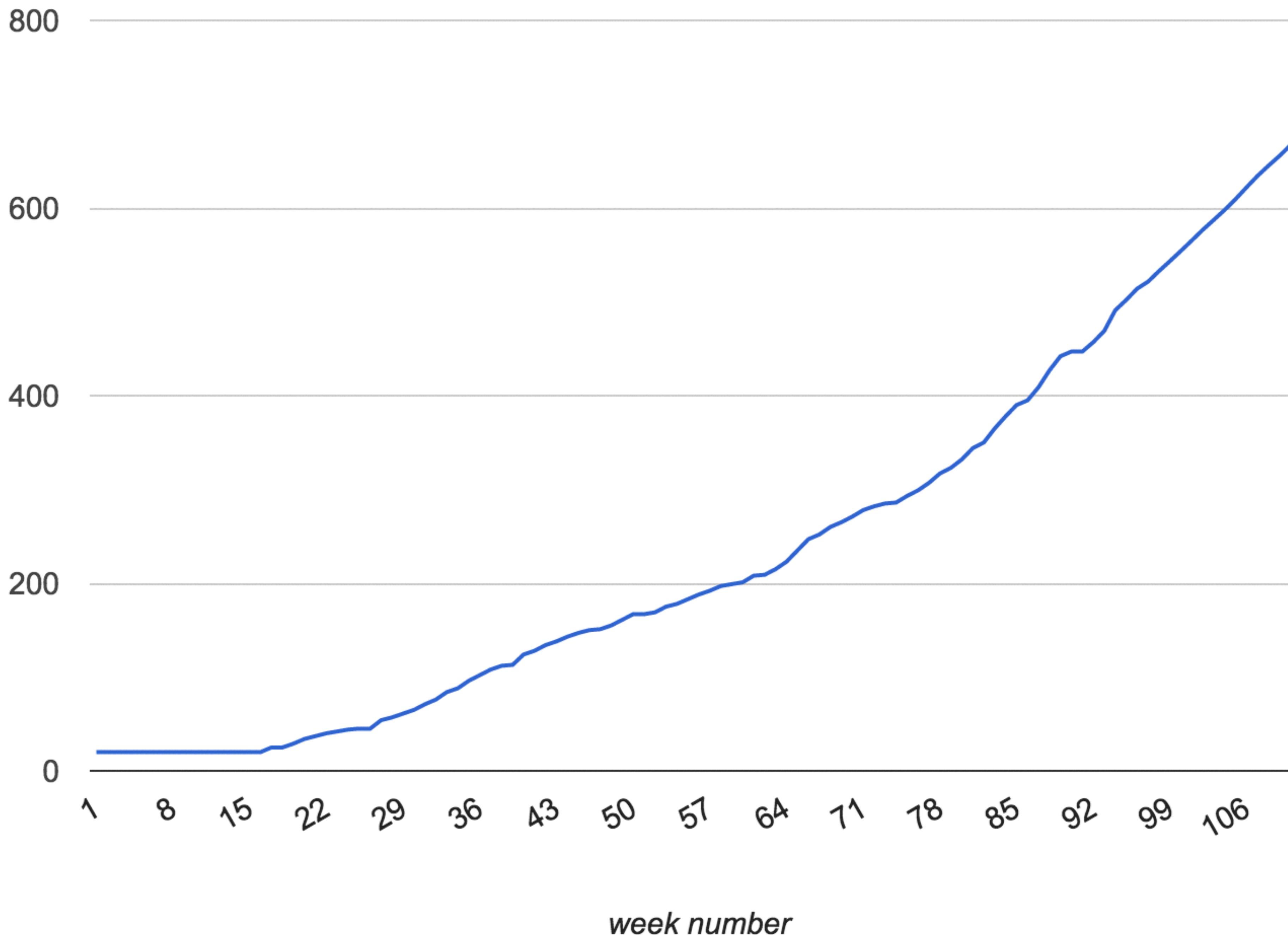
**Uber Cities Worldwide: 400+**

**Countries: 70**

**Employees: 6,000+**

# LIFE LESSONS

# total services



# **MICROSERVICES**

**Immutable?**  
**Append Only?**

# WHY MICROSERVICES?

Move and Release Independently  
Own your Uptime  
Use the “Best” tool for the job

# **WHAT ARE THE COSTS?**

Now you have a distributed system  
Everything is an RPC  
What if it breaks?

# LESS OBVIOUS COSTS

Everything is a tradeoff

You can build around problems

Might trade complexity for politics

You get to keep your biases

**pre-history**    **PHP (outsourced)**

**Dispatch**    **Node.JS, moving Go**

**Core Services**    **Python, moving to Go**

**Maps**    **Python and Java**

**Data**    **Python and Java**

**Metrics**    **Go**

# LANGUAGES

Hard to share code

Hard to move between teams

WIWIK: Fragments the culture

# **RPC**

**HTTP/REST gets complicated**

**JSON needs a schema**

**RPCs are slower than PCs**

**WIWIK: servers are not browsers**

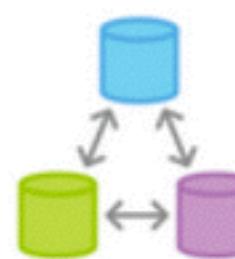
# HOW MANY REPOS

Many is good

One is good

Many is bad

One is bad

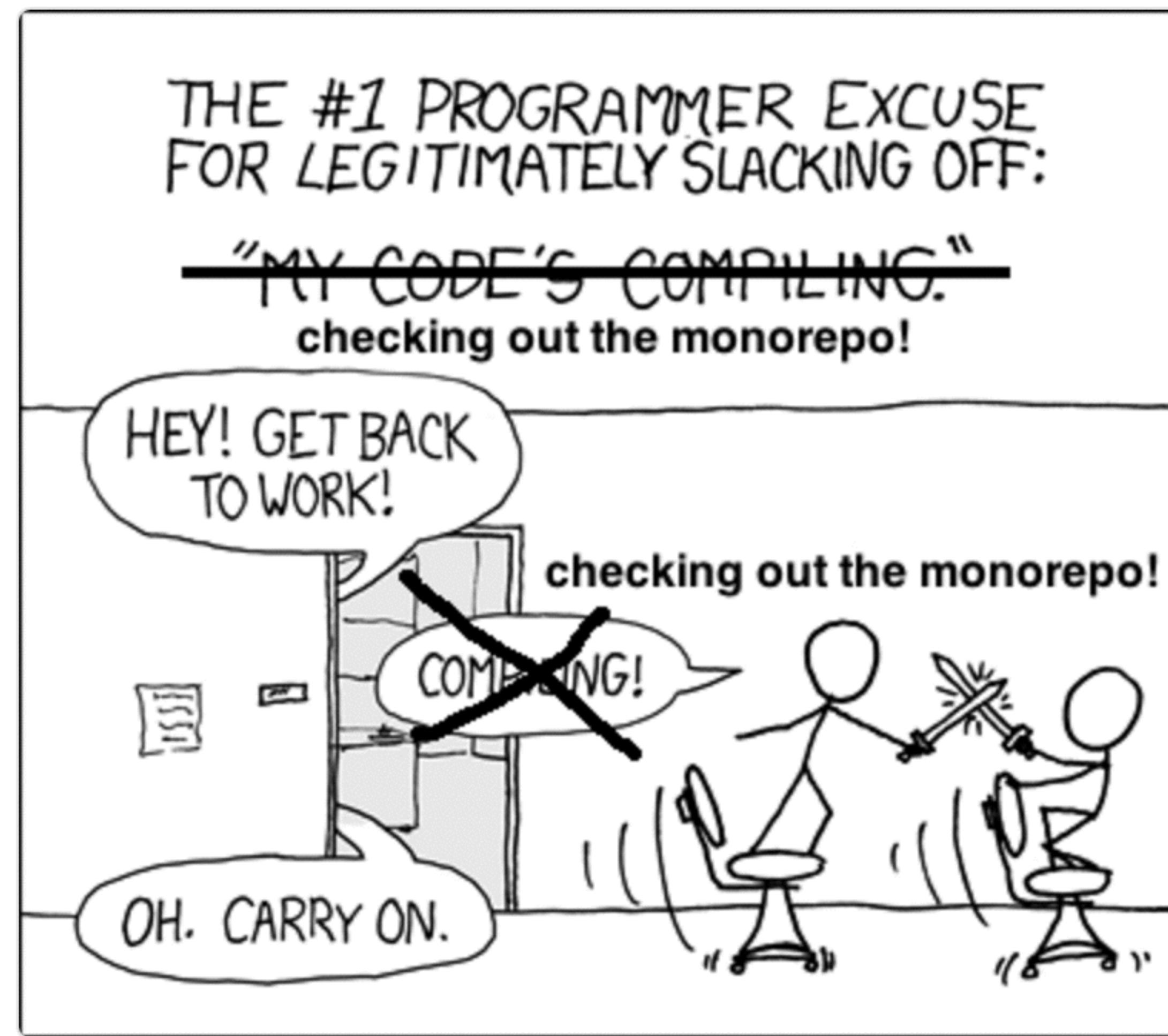


monorepi  
@monorepi



Following

here I updated the XKCD comic for you



# APRIL 2016

```
mjr:~$ perl -ne '$c++; $p++ if /personal/; $conf++ if /config/; END { print "$c total\n$p personal\n$conf config\n";}' all_repos
7005 total
1074 personal
374 config
```

# MAY 2016

```
mjr:~$ perl -ne '$c++; $p++ if /personal/; $conf++ if /config/; END { print "$c total\n$p personal\n$conf config\n";}' all_repos
8263 total
1137 personal
407 config
```

# OPERATIONAL

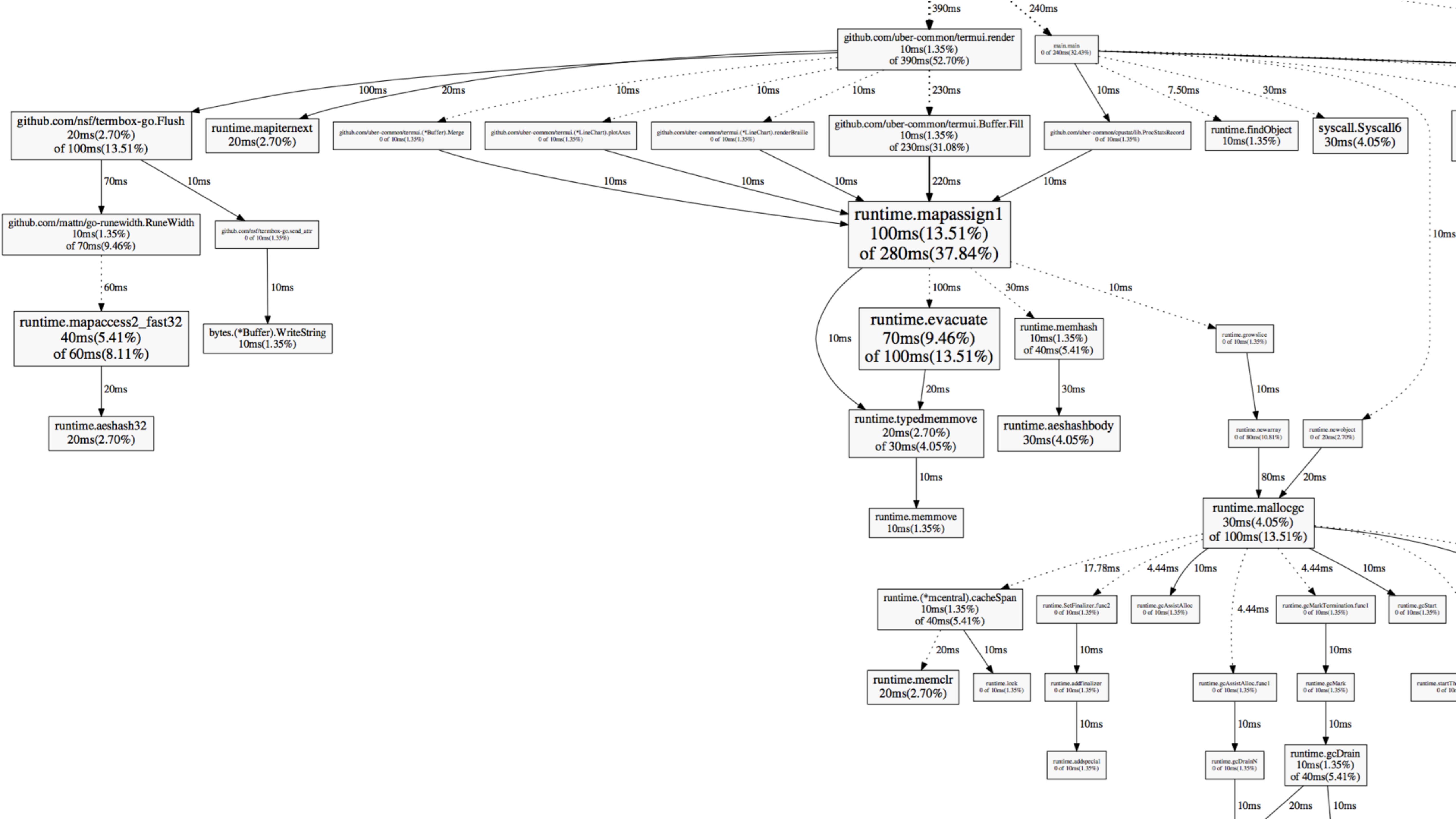
What happens when things break?

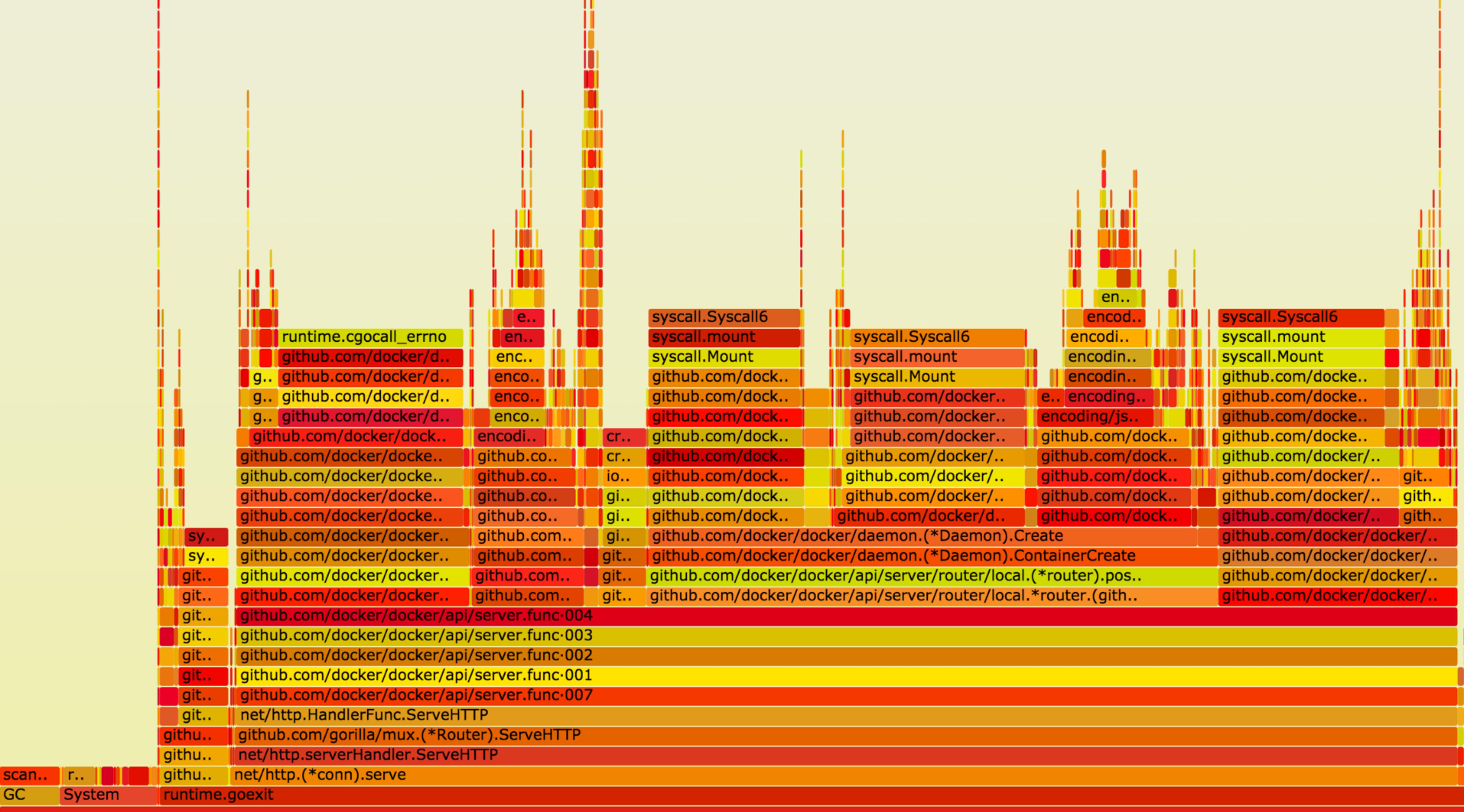
Can other teams release your service?

Understand a service in the larger context

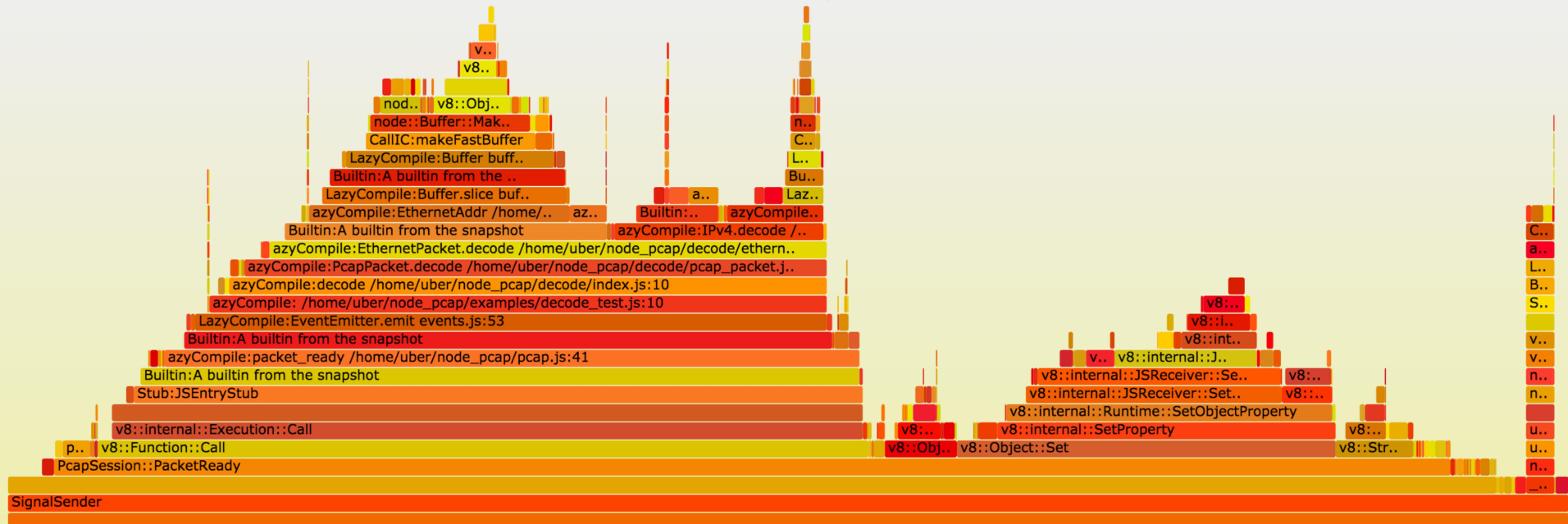
# **PERFORMANCE**

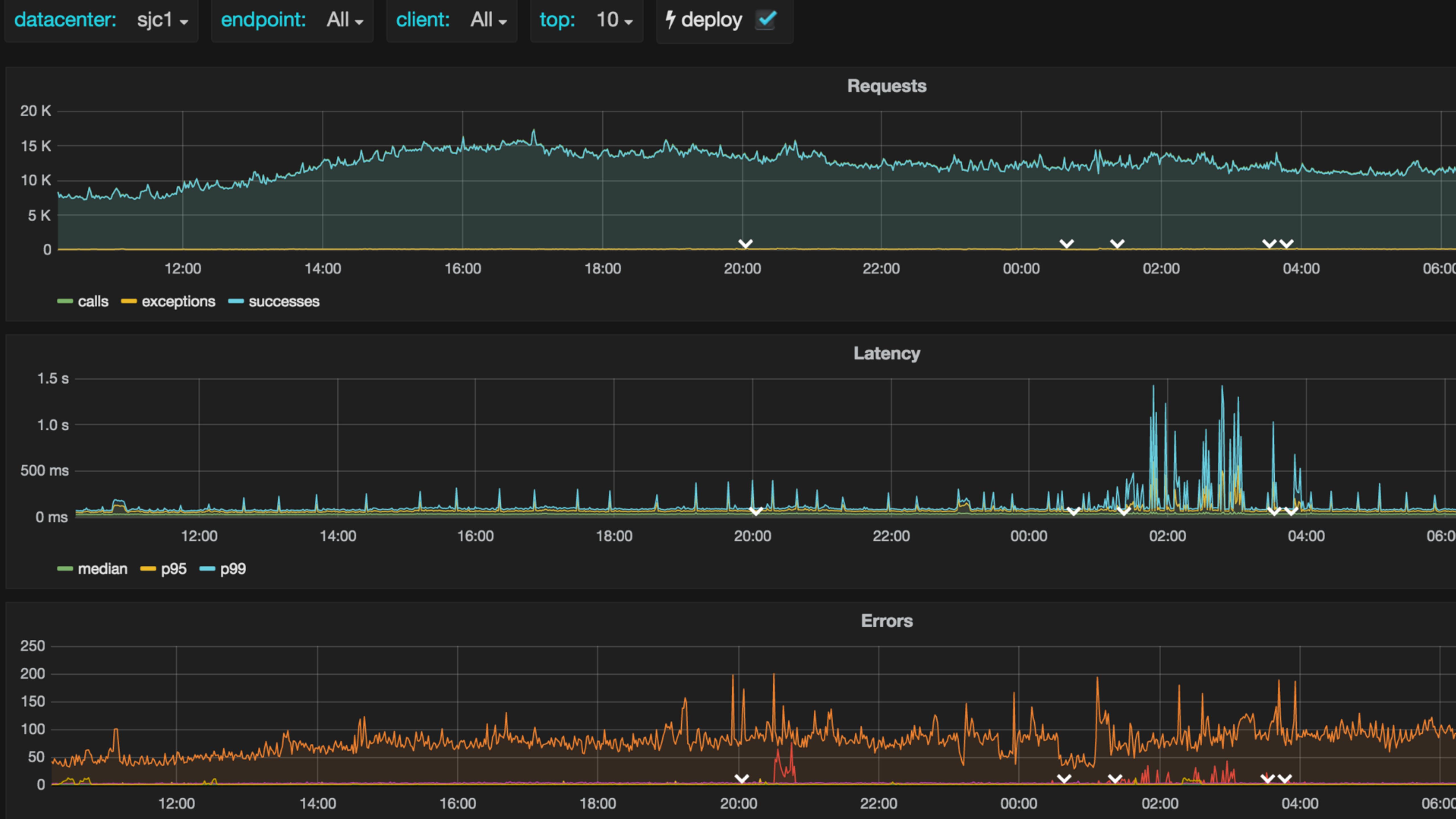
Depends on language tools





### Flame Graph





# PERFORMANCE

Doesn't matter until it does

Probably want at least simple perf requirements

WIWIK: “good” not required, but “known” is

# FANOUT

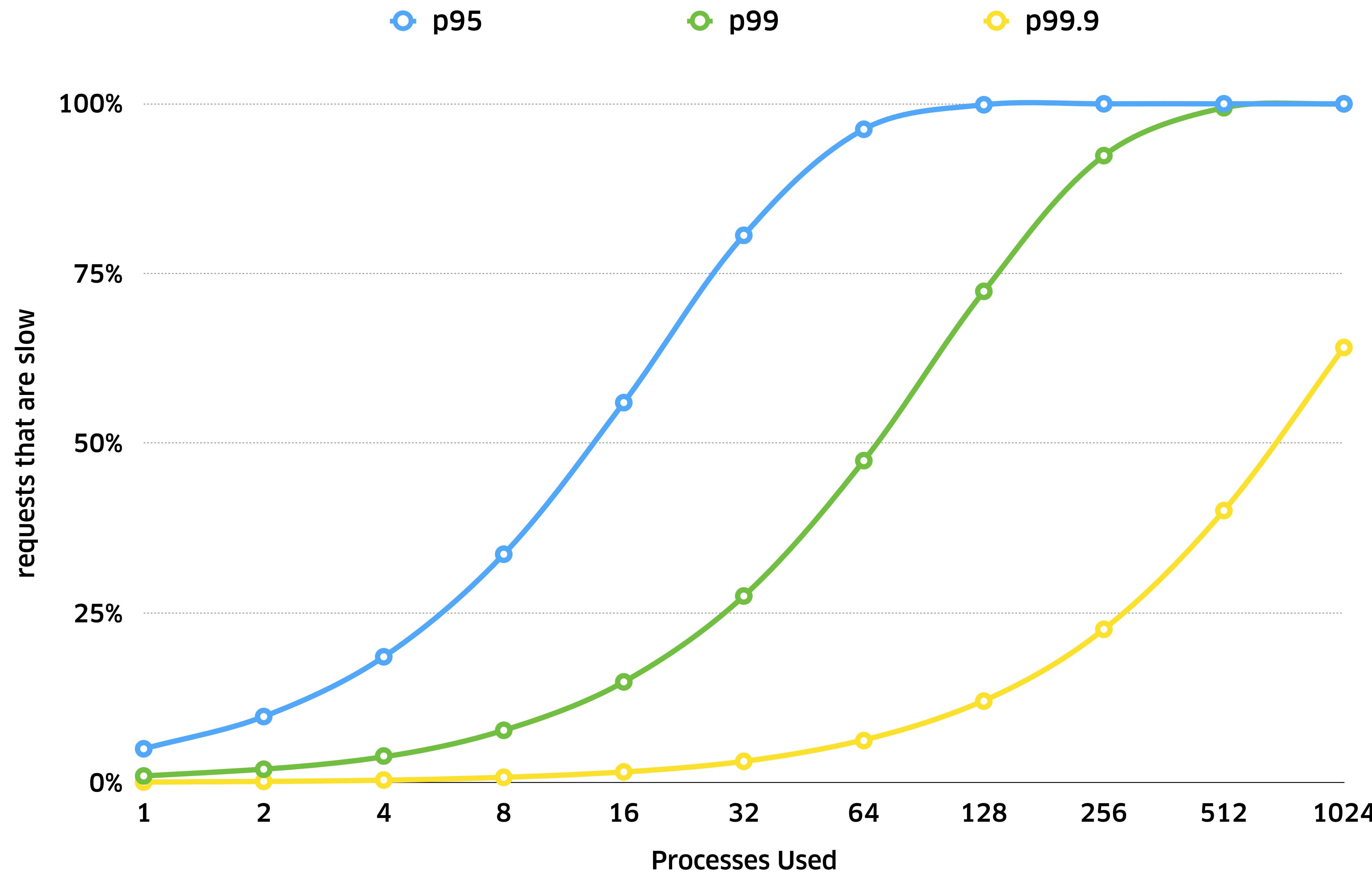
overall latency  $\geq$  latency of slowest

1ms avg, 1000ms p99

use 1: 1% at least 1000ms

use 100: 63% at least 1000ms

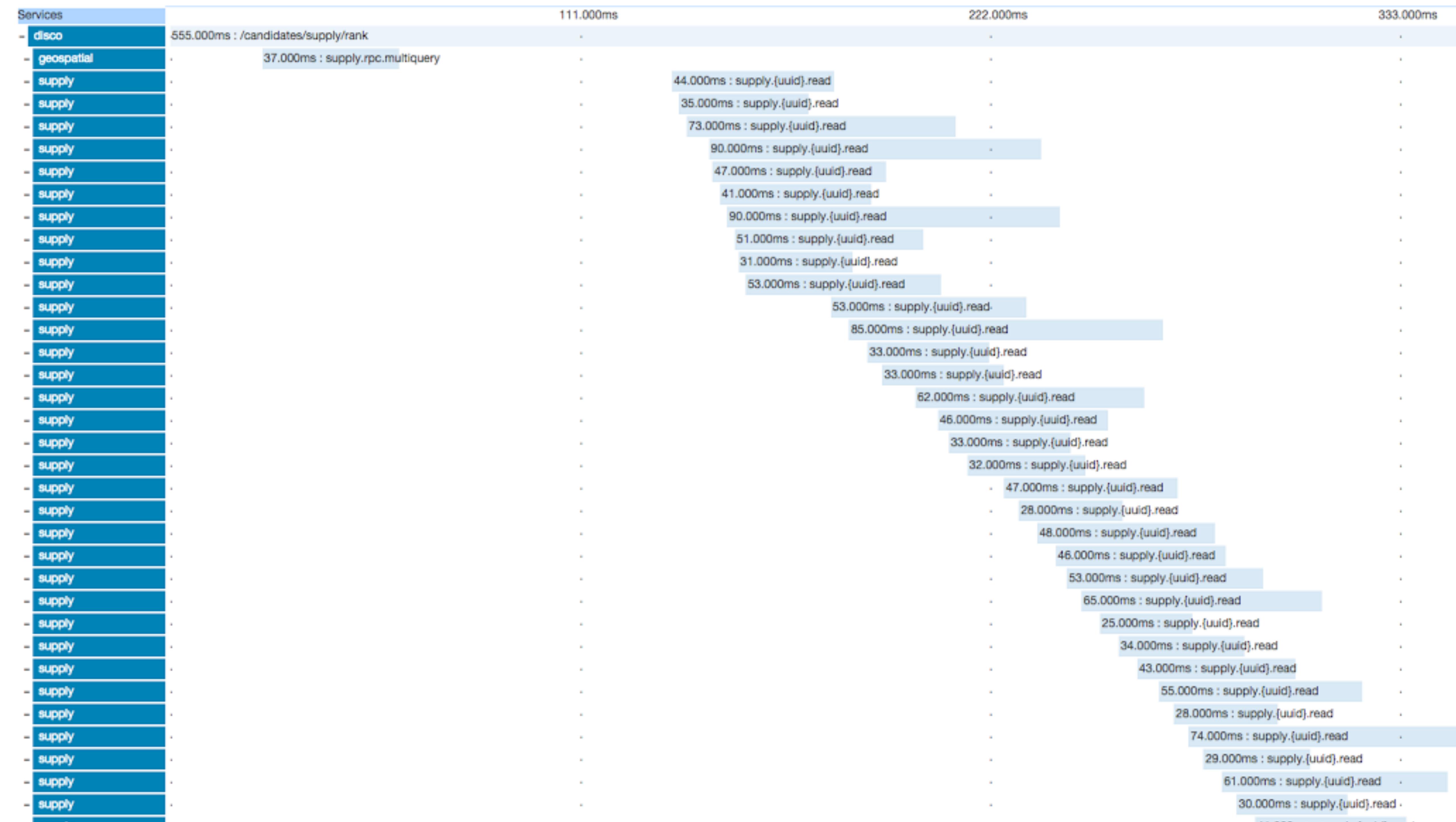
$$1.0 - 0.99^{100} = 0.634 = 63.4\%$$



# **TRACING**

**Lots of ways to get this  
Best way to understand fanout**

Services		756.000ms	1.512s	2.268s	3.024s	3.780s
- rtapi	3.775s : /riders/:rideruuid/pickup	.	.	.	.	.
- passport	. 3.000ms : resolveregion	.	.	.	.	.
- cn	. 3.000ms : resolveregion	.	.	.	.	.
- on	. 162.000ms : getclient	.	.	.	.	.
- halyard	.	58.000ms : gettreatmentresult	.	.	.	.
- optic	.	62.000ms : /client/:uuid/ping	.	.	.	.
- geospatial	.	6.000ms : supply.rpc.multiquery	.	.	.	.
- paxon	.	3.000ms : /eyeball/	.	.	.	.
- ueta	.	33.000ms : /v2/eta/predict-many	.	.	.	.
- onedirection	.	4.000ms : /fitted_multi	.	.	.	.
- onedirection	.	3.000ms : /fitted_multi	.	.	.	.
- ueta	.	32.000ms : /v2/eta/predict-many	.	.	.	.
- ultron	.	4.000ms : /classify	.	.	.	.
- ultron	.	3.000ms : /classify	.	.	.	.
- api	.	3.085s : verifypaymentprofile	.	.	.	.
- demand	.	.	.	.	230.000ms : /client/:uuid/jo	.
- optic	.	.	.	.	.	8.000ms : /cli
- optic	.	.	.	.	.	100.000ms : /
- demand	.	.	.	.	.	45.000ms : /
- trident	.	.	.	.	.	55.000m
- on	.	.	.	.	.	6.000m
- passport	.	.	.	.	.	44.0



Services	1.515s	3.031s	4.546s	6.062s
- accountmgmt	7.577s : accountmgmtservice::getallmerchants			
- accountmgmt	58.104ms : sql select			
- accountmgmt	57.771ms : mysqlDb::select			
- accountmgmt	180.370ms : sql select			
- accountmgmt	180.120ms : mysqlDb::select			
- accountmgmt	5.316ms : sql select			
- accountmgmt	4.976ms : mysqlDb::select			
- accountmgmt	1.848ms : sql select			
- accountmgmt	766μ : mysqlDb::select			
- accountmgmt	1.048ms : sql select			
- accountmgmt	600μ : mysqlDb::select			
- accountmgmt	1.070ms : sql select			
- accountmgmt	783μ : mysqlDb::select			
- accountmgmt	940μ : sql select			
- accountmgmt	624μ : mysqlDb::select			
- accountmgmt	1.130ms : sql select			
- accountmgmt	791μ : mysqlDb::select			
- accountmgmt	2.553ms : sql select			
- accountmgmt	814μ : mysqlDb::select			
- accountmgmt	751μ : sql select			
- accountmgmt	495μ : mysqlDb::select			
- accountmgmt	956μ : sql select			
- accountmgmt	734μ : mysqlDb::select			
- accountmgmt	722μ : sql select			
- accountmgmt	493μ : mysqlDb::select			
- accountmgmt	698μ : sql select			
- accountmgmt	469μ : mysqlDb::select			
- accountmgmt	692μ : sql select			
- accountmgmt	479μ : mysqlDb::select			
- accountmgmt	669μ : sql select			
- accountmgmt	455μ : mysqlDb::select			
- accountmgmt	702μ : sql select			
- accountmgmt	475μ : mysqlDb::select			
- accountmgmt	719μ : sql select			

# TRACING

Probably want sampling  
WIWIK: cross-lang context propagation

# LOGGING

Need consistent, structured logging

Multiple languages makes this hard

Logging floods can amplify problems

WIWIK: Accounting

## README.md

[godoc](#)[reference](#)[build](#)[passing](#)[coverage](#)

98%

Fast, structured, leveled logging in Go.

## Structure

Zap takes an opinionated stance on logging and doesn't provide any `printf`-style helpers. Rather than

```
logger.Printf("Failed to fetch URL %s (attempt %v), sleeping %s before retry.", url, tryNum, sleepFor), zap  
encourages the more structured
```

```
logger.Info("Failed to fetch URL.",  
    zap.String("url", url),  
    zap.Int("attempt", tryNum),  
    zap.Duration("backoff", sleepFor),  
)
```

This is a bit more verbose, but it enables powerful ad-hoc analysis, flexible dashboarding, and accurate message bucketing. In short, it helps you get the most out of tools like ELK, Splunk, and Sentry. All log messages are JSON-serialized, though PRs to support other formats are welcome.

## Performance

For applications that log in the hot path, reflection-based serialization and string formatting are prohibitively expensive — they're CPU-intensive and make many small allocations. Put differently, using `encoding/json` and `fmt.Println` to log tons of `interface{}`s makes your application slow.

Log a message using a logger that already has 10 fields of context:

Library	Time	Bytes Allocated	Objects Allocated
⚡ zap	231 ns/op	0 B/op	0 allocs/op
logrus	8035 ns/op	3438 B/op	61 allocs/op
go-kit	6790 ns/op	2486 B/op	48 allocs/op
log15	20709 ns/op	3543 B/op	69 allocs/op

Log a static string, without any context or `printf`-style formatting:

Library	Time	Bytes Allocated	Objects Allocated
⚡ zap	223 ns/op	0 B/op	0 allocs/op
standard library	562 ns/op	32 B/op	2 allocs/op
logrus	2765 ns/op	1336 B/op	26 allocs/op
go-kit	1092 ns/op	624 B/op	13 allocs/op
log15	5513 ns/op	1351 B/op	23 allocs/op

# LOAD TESTING

Need to test against production

Without breaking metrics

Preferably all the time

WIWIK: all systems need to handle “test” traffic

# **FAILURE TESTING**

**WIWIK: people won't like it**

# MIGRATIONS

Old stuff still has to work  
What happened to immutable?  
WIWIK: mandates are bad

# **OPEN SOURCE**

**Build/buy tradeoff is hard**

**Commoditization**

**WIWIK: this will make people sad**

# POLITICS

Services allow people to play politics

Company > Team > Self

# **TRADEOFFS**

**Everything is a tradeoff  
Try to make them intentionally**

# THANKS