We're already doing microservices

Contents

→ InstaMike Pains :tired_face"...

→ Bounded Contexts 

→ What are these 👾 things?

→ 👾 Gotchas 😛

→ Must Win.

Detailed Contents(REMOVE THIS SLIDE)

→ Mike Pains 😫...

→ Pains: Gem updates

→ Pains: Scaling

→ Pains: Deploying

→ Bounded Contexts

→ Devise Context -> Auth-er ('Cookie' Service)

Detailed Contents(REMOVE THIS SLIDE)

→ Introducing Microservices 👾

→ Microservices Flavors

→ Workers to Work-er-s 🤓

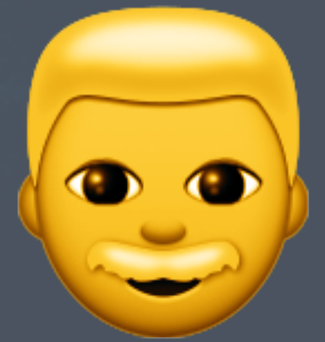
→ Caveats

→ Must Win

BASED ON A

TRUE STORY

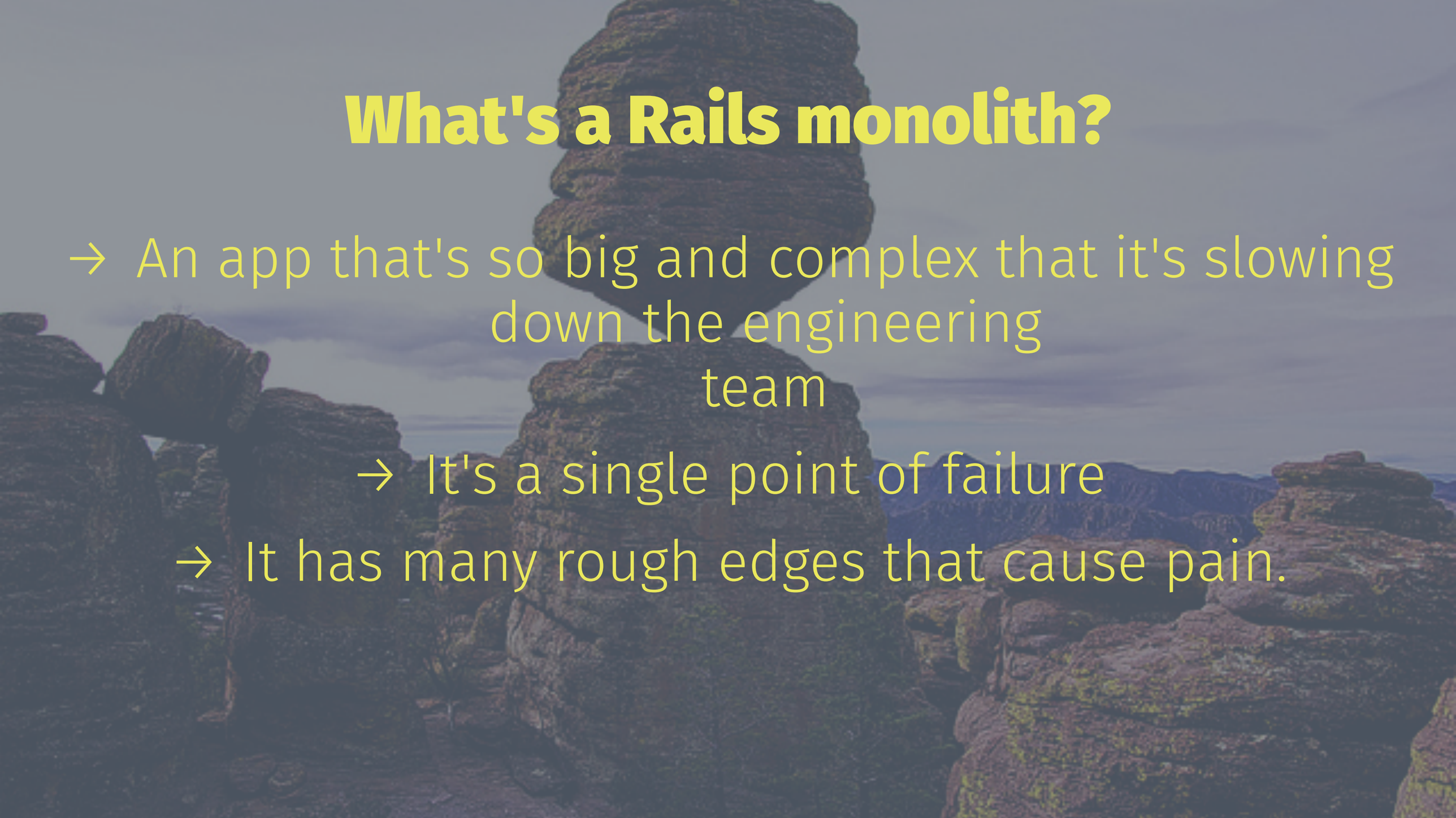
InstaMike



BASED ON A

TRUE STORY

MustWin ♥ 

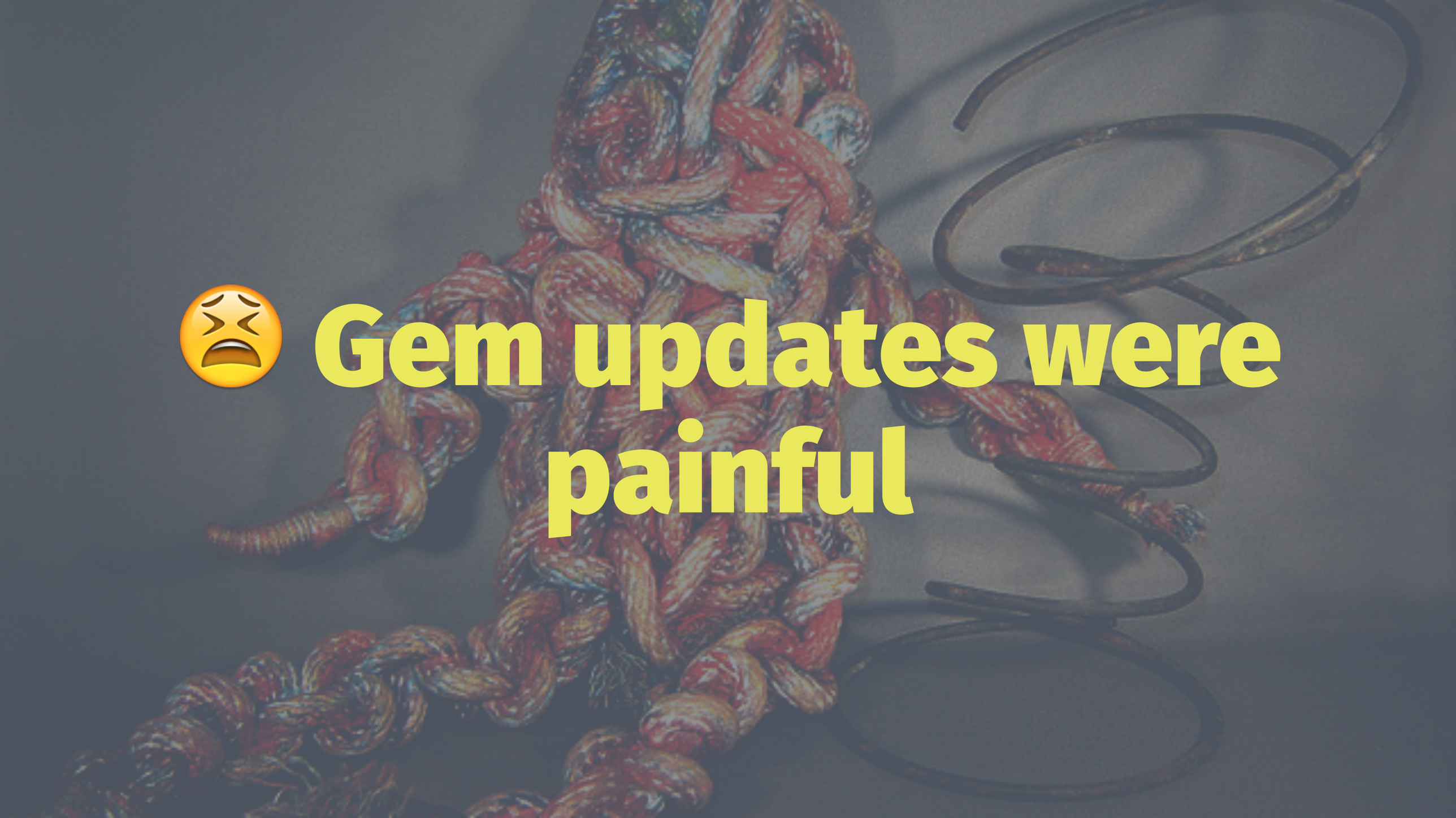


What's a Rails monolith?

→ An app that's so big and complex that it's slowing down the engineering team

→ It's a single point of failure

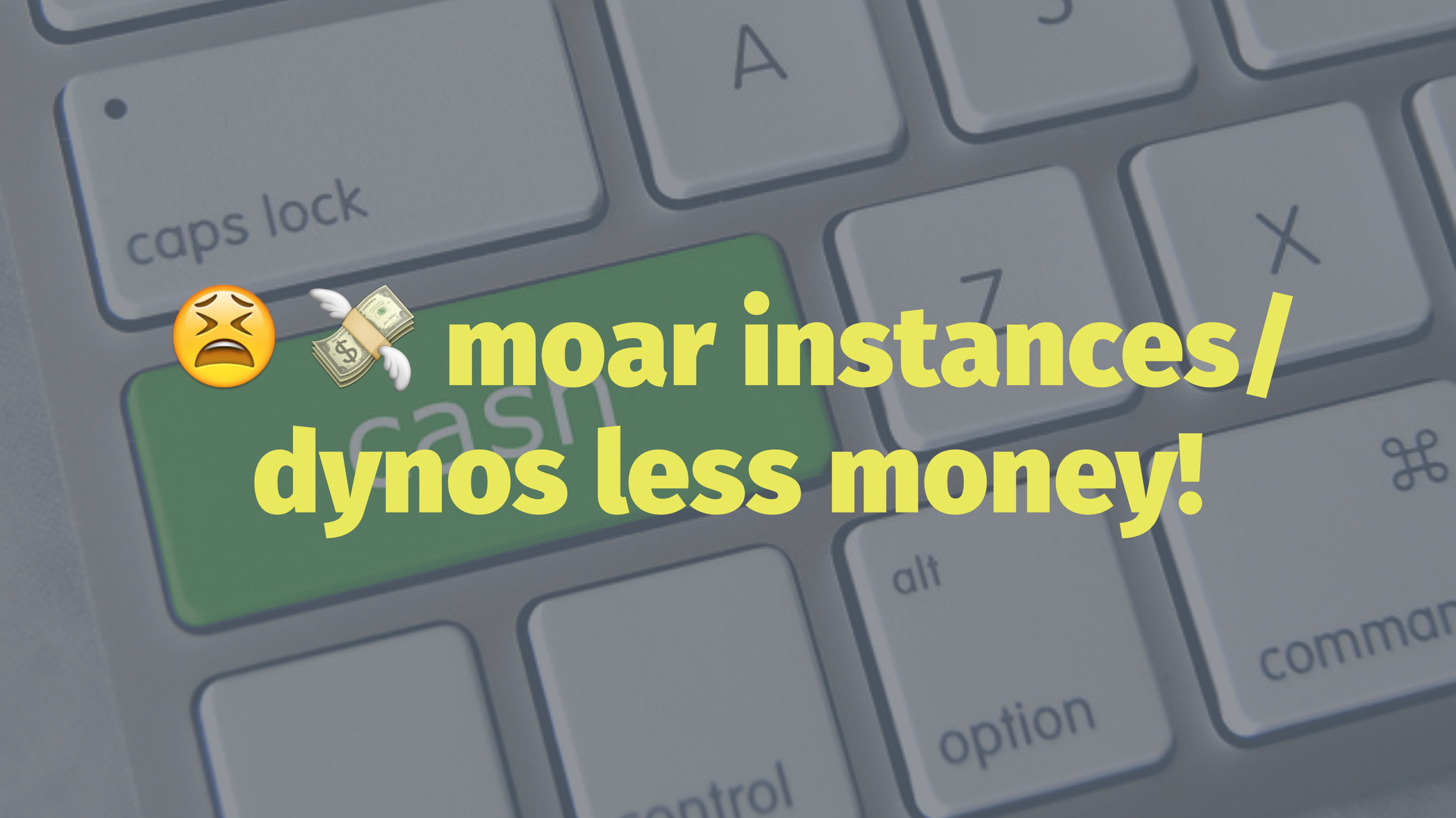
→ It has many rough edges that cause pain.


😫 Gem updates were painful

😫 Gem updates were painful

```
Bundler could not find compatible versions for gem "tilt":
  In Gemfile:
    sass-rails (= 3.2.6) ruby depends on
      tilt (~> 1.3) ruby

    slim (>= 0) ruby depends on
      tilt (2.0.0)
```



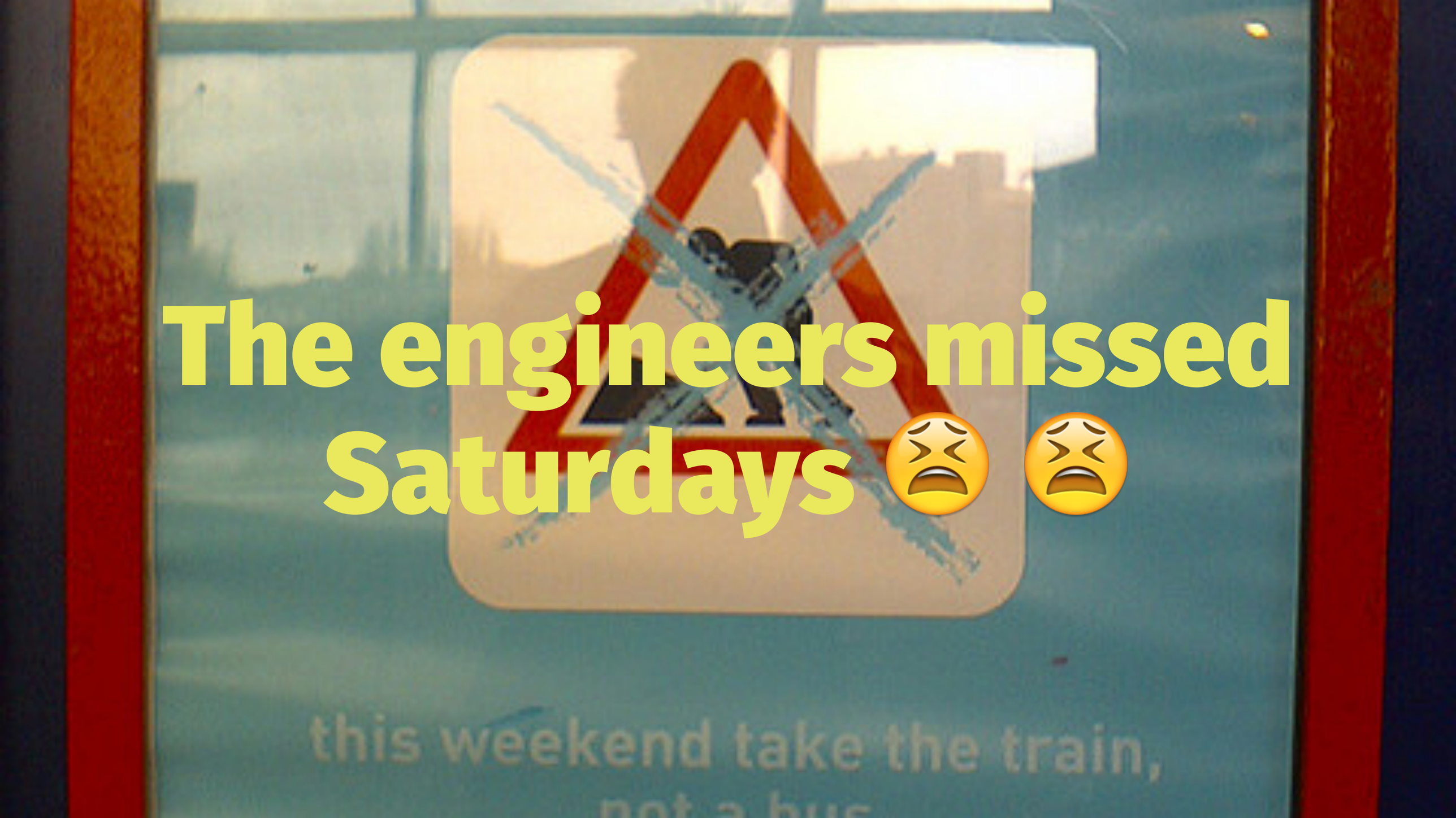
😫💸 moar instances/ dynos less money!

😫 💸 moar instances for just one endpoint

→ InstaMike had a user/images route in their app. It accounted for
98% of it's traffic.

→ They kept having to buy more boxes just for this one endpoint.

→ It was 💰 💰 💸


The engineers missed
Saturdays 😫 😫
this weekend take the train,

Saturday was the only "safe day" to deploy 😫 😫

→ If everything lives inside one app, then the whole team needs to be in sync while doing releases.

→ For InstaMike that meant that all releases had to happen on Saturday Morning when the users were less active.

→ It was 😫

How did they fix InstaMike? 🍱 Divide and conquer!

1. Identify what things belong together
2. Isolate the parts
3. Scale those parts

Bounded Contexts FTW! 

"A specific responsibility enforced by explicit boundaries"

A small set of of models that operate almost independently of the rest of the app.

Bounded Contexts FTW! 

```
$ tree libs/
libs/
├── analytics
│   ├── ab_groups.rb
│   └── generate_report.rb
├── photos
│   ├── filters.rb
│   ├── resize.rb
│   └── upload.rb
└── user
    ├── friends.rb
    └── user.rb
```

Bounded Contexts FTW! 

→ Rails Engines are bounded contexts

```
module Blorgh
  class Engine < ::Rails::Engine
  end
end
```

Devise Context -> Auth-er

→ In a nutshell what does devise do? Setup a session cookie.

→ That's it.

```ruby
insert.code.for.session.setting.here
```

Devise Context -> Auth-er

→ Sessions can be shared across apps

→ Even on apps that aren't written on ruby

`insert.gorailsyourself.example.here`

The Image Context

→ It uses http for communication

→ Yes there is GEMFORIMAGES

→ Using Imgix or Imgur api is faster

→ But there are so many open source implementations that you can just pick one and go.

The Image Context -> Imag-er

```
insert.httpcalls.example.here
```

ETL -> Bottle-er

And talking about cool stuff, what about I told you
don't need to run a
script every night to update your app reports?

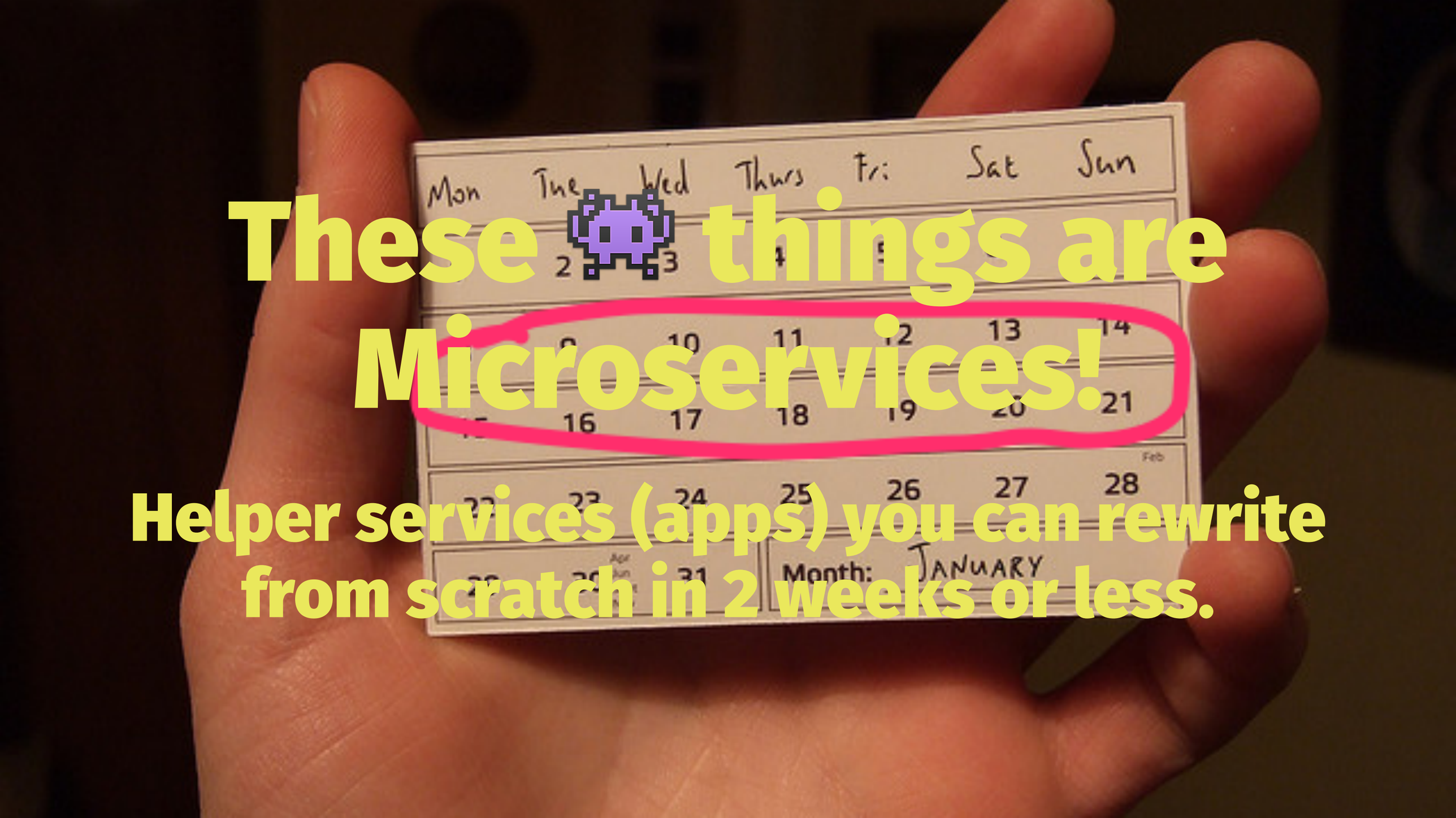
Postgres internally has a Write Ahead Log, which is a
stream of events
that is used for replication.

There is this really awesome project called Bottled

ETL -> Bottle-er

```
insert.example.here
```

What are these things?


These things are Microservices!
Helper services (apps) you can rewrite from scratch in 2 weeks or less.

These 👾 things are Microservices!

→ Auth Sharing services

→ Http Services

→ Database sharing services

→ Messaging Queue services

Microservices 👾 can:

→ Be written in any language

→ Talk other services via Rest, Redis, RabbitMQ, Cookies, or smoke signals

→ Live inside or outside your app's firewall

→ Can scale individually

If your app uses Resque, you're already using services (but not microservices)

To get them to be 👾:

→ Make them smaller. Their dependencies must also be able to be rewriten in 2 weeks.- Enable them to scale individually (Some implementations already allow you to do this).

→ Stop them loading the whole app inside them

👾 Microservices aren't perfect

There are many gotchas for getting Microservices right:

→ Distributed systems are hard. (Pick speed, consistency or availability)

→ Debugging across multiple services can be tricky

→ Creating development & production environments becomes harder

Three quick tips before wrapping up:

→ RabbitMQ or any other messaging service makes it easier to keep state consistent. HTTP loses messages very easily.

→ Try to minimize state as much as you can. Replication is very costly

→ Check the references (Specially Sam Newman's book)

Must Win builds ❤ apps

I'm Gonzalo Maldonado and I'm a lead engineer at Must Win.
We build ❤ apps. (Sometimes with 👾 Microservices)

Check us out at

MustWin.com

References

→ Sam Newman's Building Microservices

→ High Scalability blog

→ Continuous Deployment

→ Ruby Rogues

→ https://speakerdeck.com/elg0nz/5-ways-we-screwed-up-microservices

→ Eric Evan's Domain Driven Design

→ Mustwin.com

Image Credits

→ https://flic.kr/p/ymPxN

→ https://flic.kr/p/62rNAB

→ https://flic.kr/p/t1um7