Egoless Engineering

Digging the Ditch Towards Enlightenment

Dan McKinley 2024

https://egoless.engineering

@mcfunley.com on BlueSky

Hi, I'm Dan McKinley

quant-adjacent engineering and stuff

cf. mcfunley.com

I have done engineering and led small teams and biggish orgs

I once worked on an engineering team that was executing very well

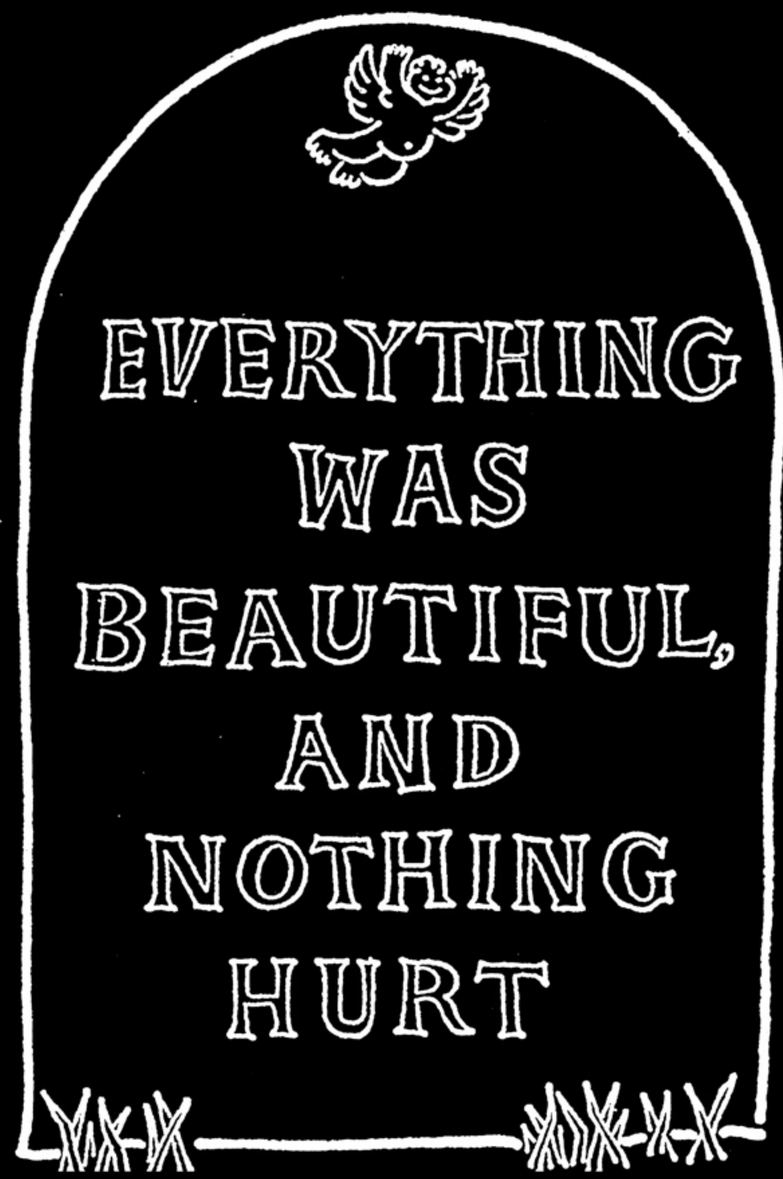
it was pretty chill I wouldn't mind doing it again

welcome to my latest talk in my series trying to rekindle the magic

nothing lasts forever

I am interested in pathology, catastrophe, malfeasance,
and fiascos of all kinds.

.. which is lucky because
There are at least two sources of selection bias:
I was hired into those places because they had problems and knew that, and wanted my help
and second, If I have a motto guiding my career, it's "well this might be funny."



I am not complaining, exactly. And since I was in a position of responsibility at these places I'll add that hopefully, I did more good than harm.
My hope with this talk is that maybe y'all can create more fulfilling places to work, and hire me to work there.

What this talk is about

I have noticed similarities when things are going poorly, and would like to talk about them.
Maybe y’all can create more fulfilling places to work, and hire me to work there

~Everyone honks up by diffusing responsibility poorly

Of course immediately once you have two employees, you have to divide responsibility somehow
But how you effect this is massively important
It’s a move with both immediate and second-order consequences
Most companies are relatively thoughtless about it

Computer scientists are also really bad at it

Despite LITERALLY studying the asymptotic limits of work completion under various conditions

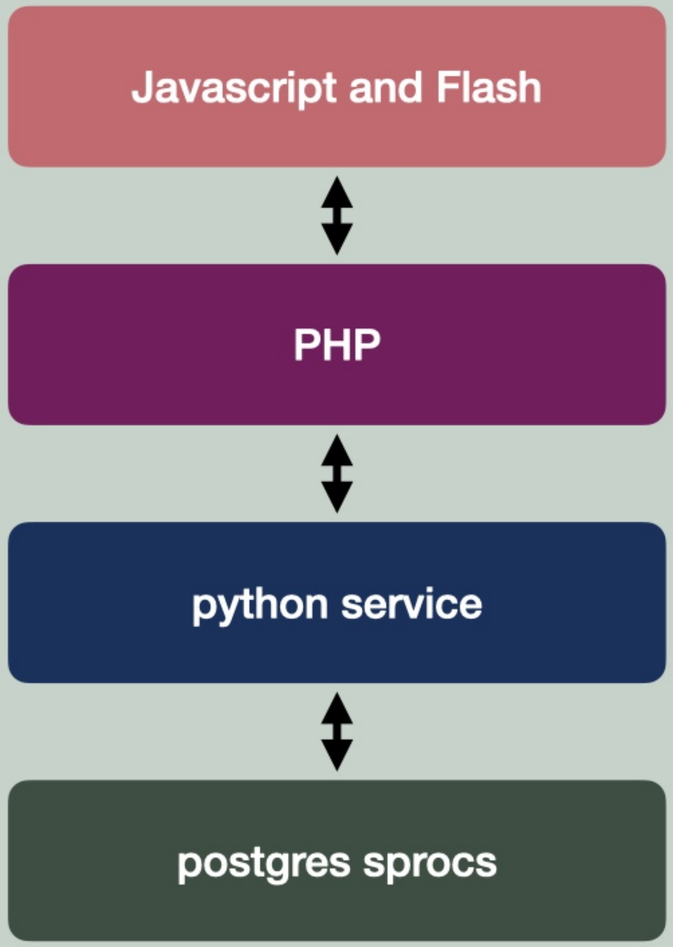
- **Amdahl's Law**: you can't just parallelize forever, coordination costs will kill you!!!
- **Kingman's formula**: growth in requests for a shared resource can have nonlinear implications for wait times!!!

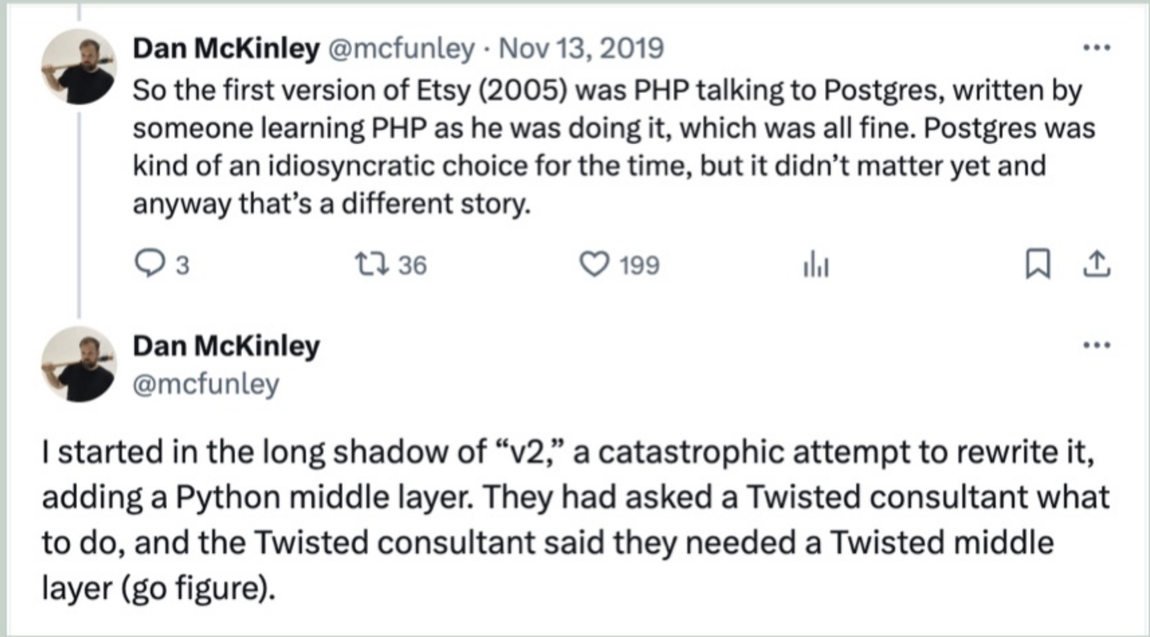
One can only conclude that we care about our computers more than ourselves.

Sad Stories

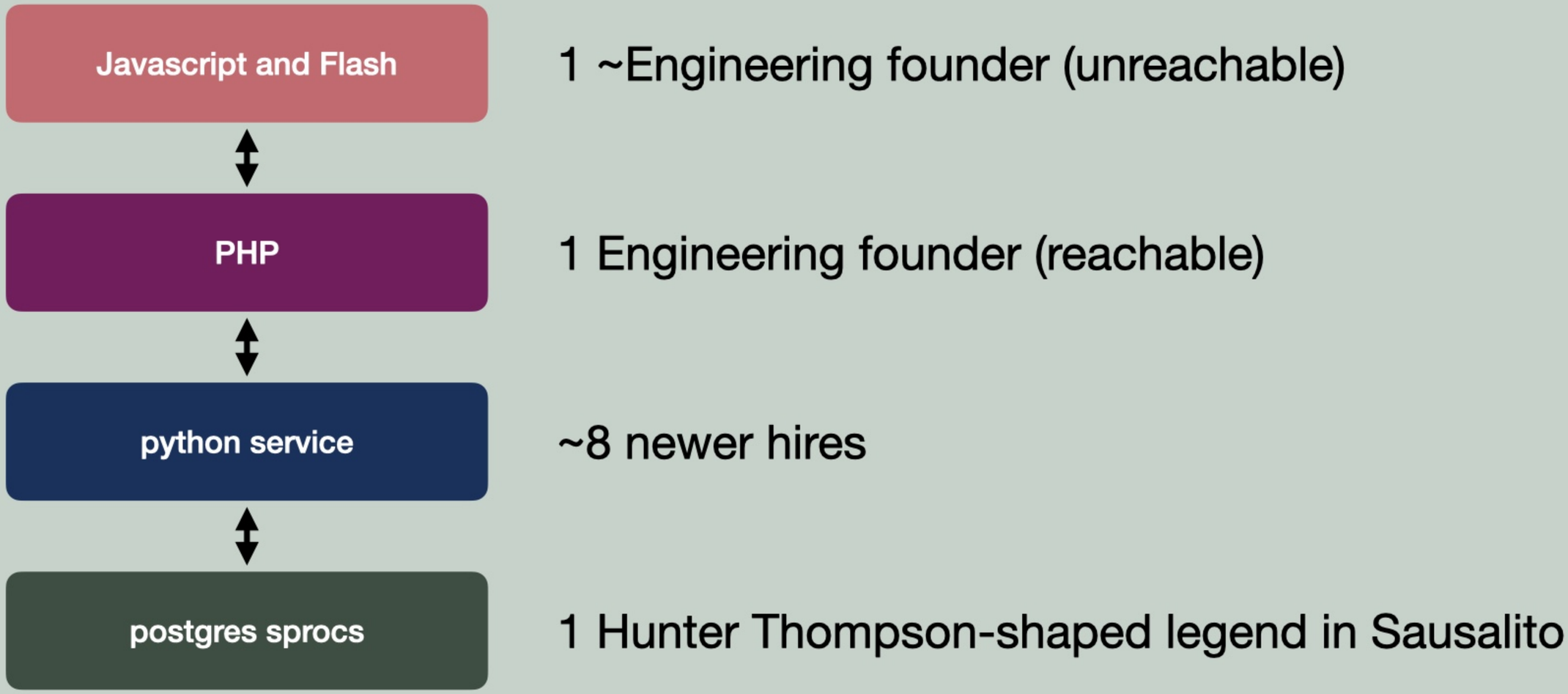
Even really successful companies are like this, so when I relate these things I'm going to anonymize them somewhat.
I won't dwell too long on this
My point is to learn from situations where things are going to shit, not to shitpost

I worked for a startup





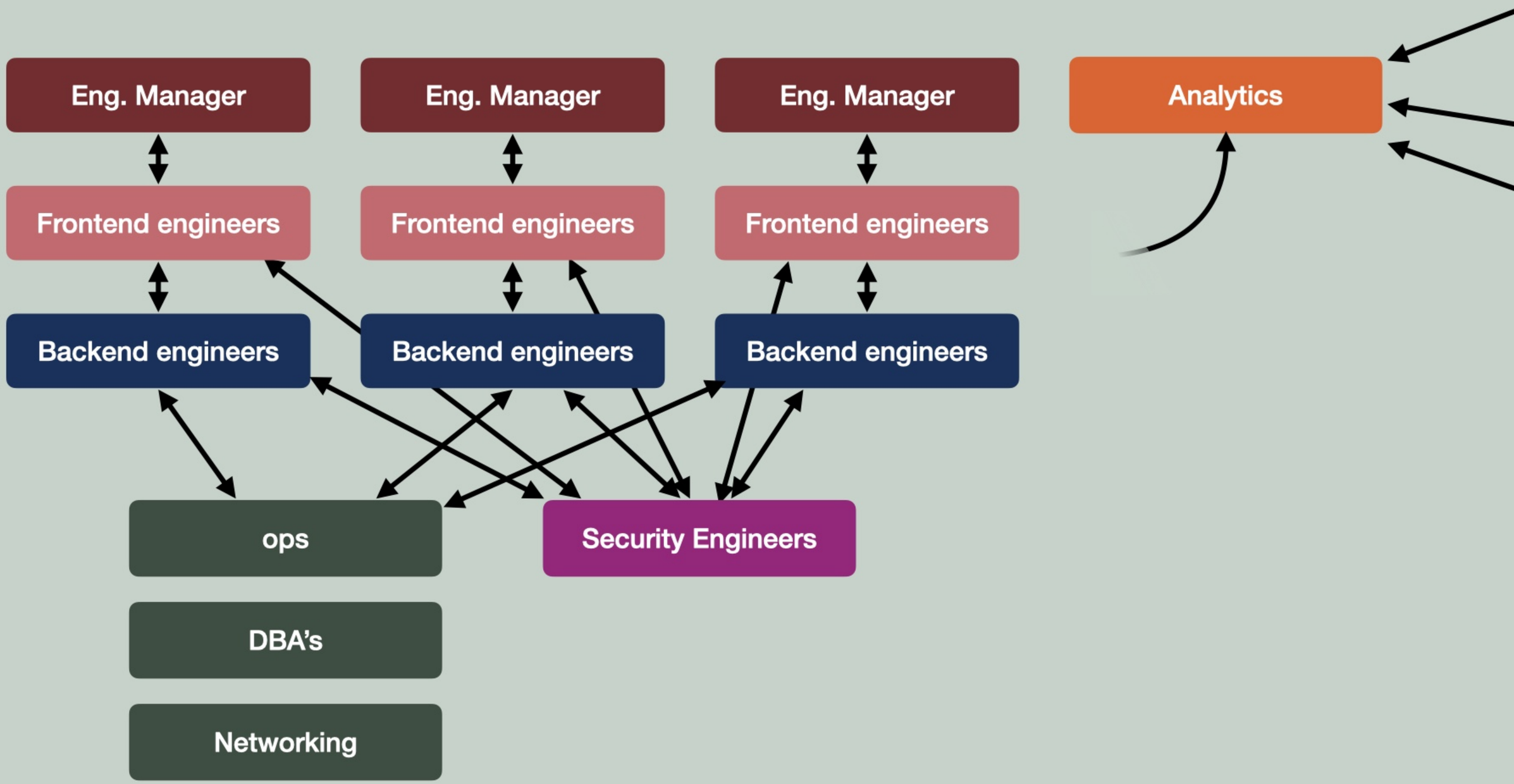
You'd think this would be scrappy but no, they had already managed to try to set themselves up like IBM in the 1970s or something
I was hired to work on a python middle layer
In a misguided attempt at scaling, they added a Python middle layer.
We will make things go faster by adding network hops
People keep trying this to this day, I am not sure why
(Architectural version of this whole talk, in microcosm, with the same results)



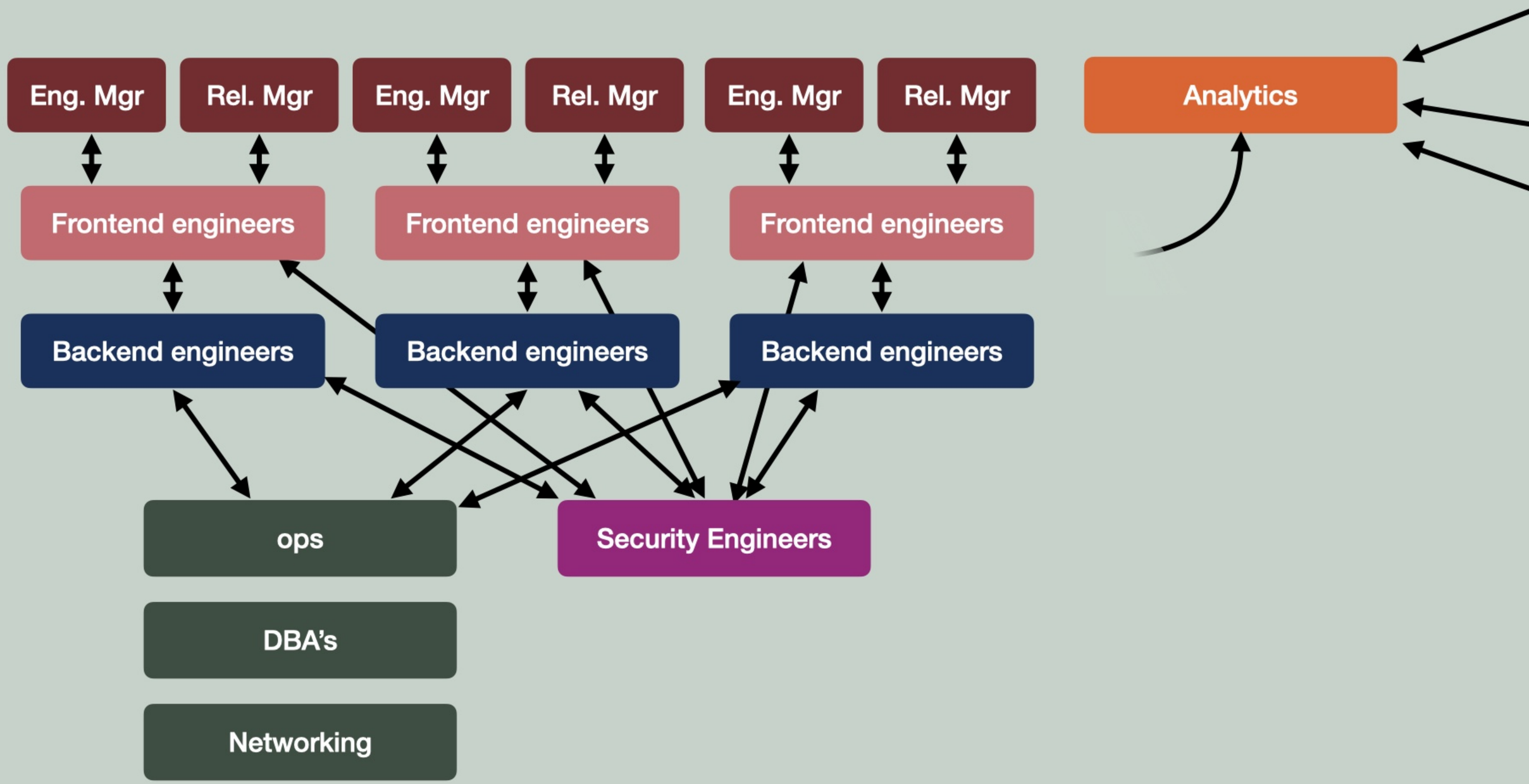
To launch any feature,
You needed to get a database person to do some DDL and write some sprocs
You needed to get a python person to write some bullshit
You needed a PHP person to write some bullshit
You needed either a JS or Flash person to write some bullshit

I worked for a B2B service provider

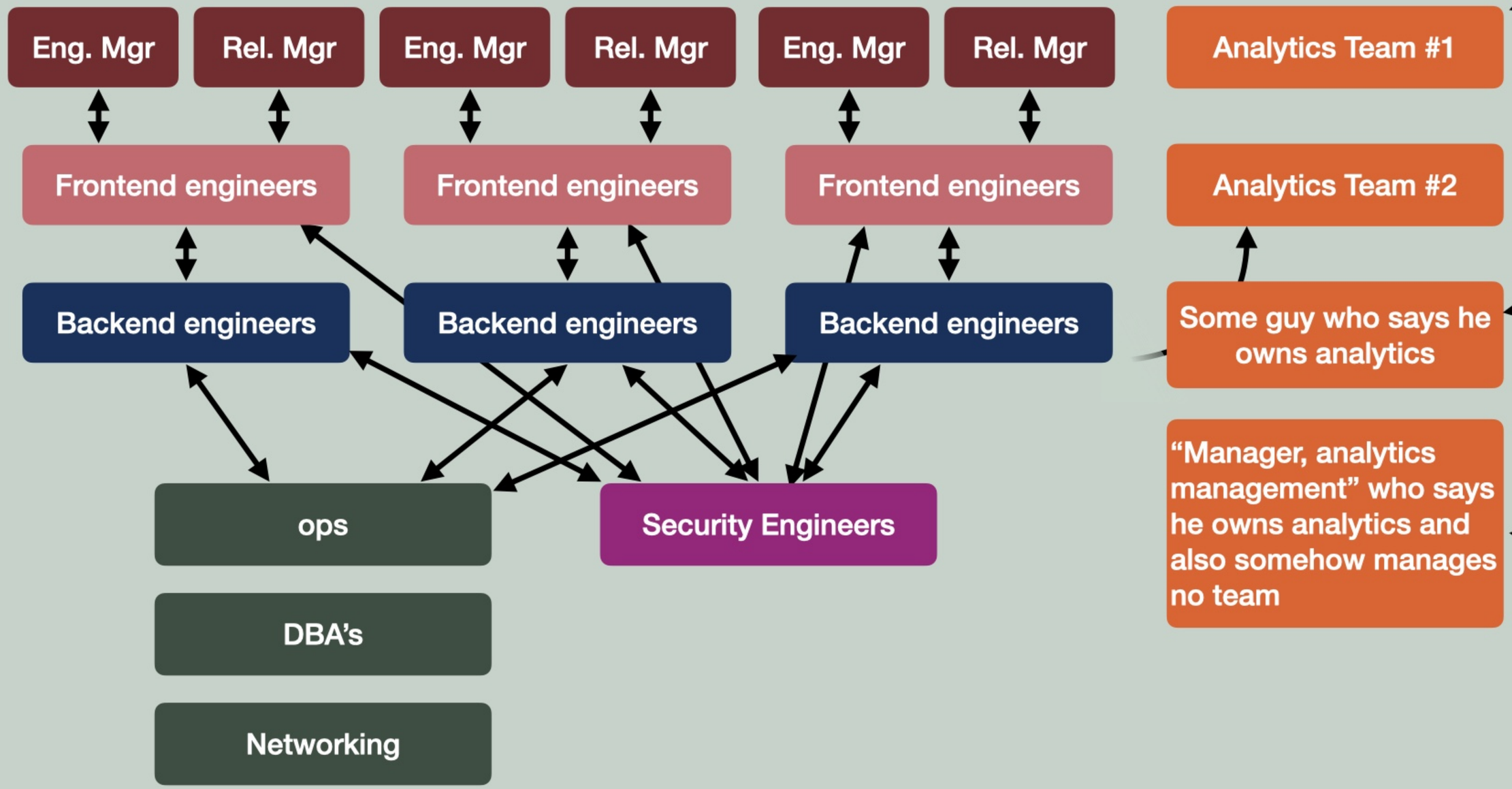
sort of managed to not



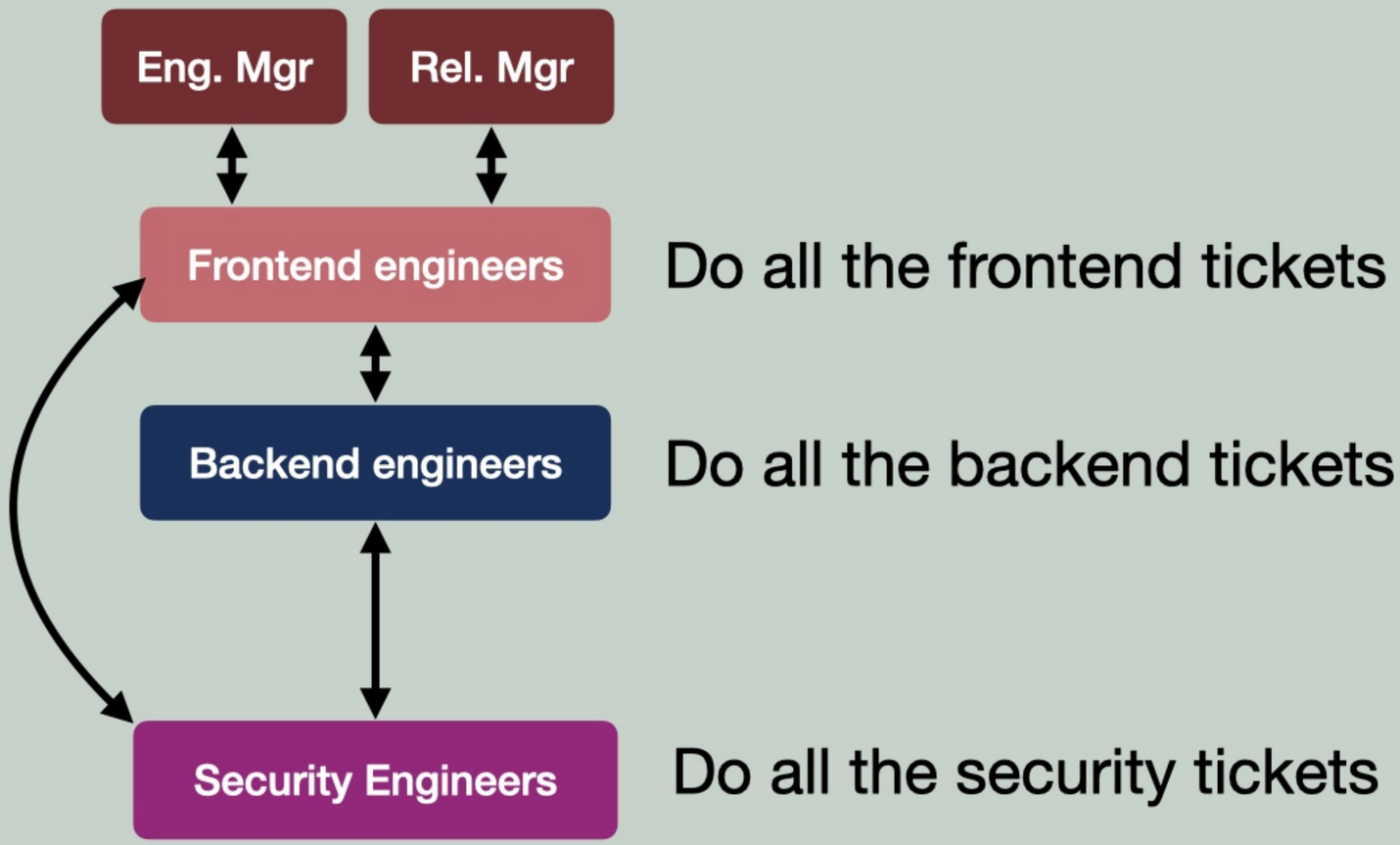
seems sorta reasonable at first



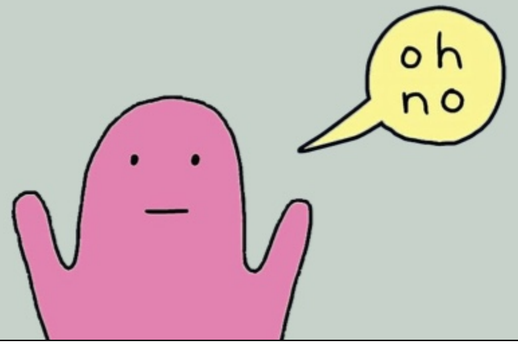
but at some point things had come unglued
eng managers had been split in two, so every team had an engineering manager and a release manager. the release manager was there to ... manage releasing software, I guess



but at some point things started to come unglued
eng managers had been split in two, so every team had an engineering manager and a release manager. the release manager was there to ... manage releasing software, I guess

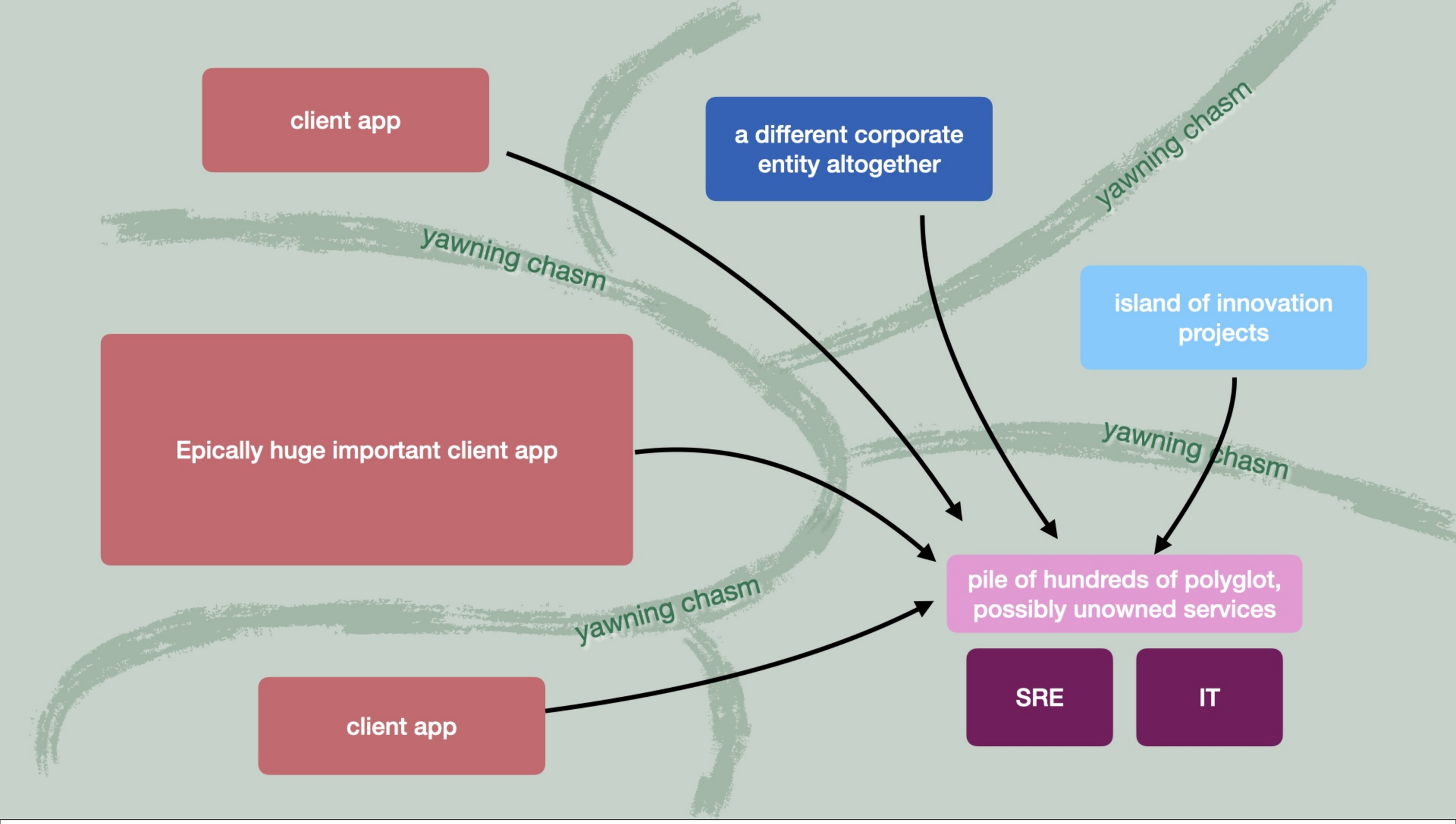


…… for all of the development teams



I worked for a native client company

There was initially one big hit client app, but by the 2020s a bigger, disjointed set of efforts had grown up in parallel.



The product portfolio as a whole had a very weak supervisory structure.
They didn't coordinate in any meaningful way on tech stacks or other shared decisions, since they all independently reported up to a CEO that made no attempt to coordinate them.
But despite that, an attempt was made at having a shared operations function.
This presented a lot of difficulty because the operations function wasn't adequately involved in architecture decisions, which were largely left to individual engineers.
They wouldn't have had enough time to do that anyway, since they were busy with hundreds of services that dev teams may have moved on from years before.
The end result of this was that everyone was overworked, and the quality of service was more or less unknown.
It's a tale as old as time.

Generalizing the Hellscape

Pattern #1

Tasks and job descriptions are oft conflated

- “A thing must be done” often leads to “this should be a job description”
  - “We need database queries written, so the DBA should do it”
  - “We need javascript and css, so we need a frontend engineer role”
  - “We need security, we should have some security people do our security”
  - “We need code on servers, someone outside of product development should do that for us as a service”
  - “We need releases to happen, so release managers or SRE’s should do them”
  - We need analysis to be done -> we need data scientists to do it

Pattern #2

Serialized development feels obvious, but is wrong, and also sucks

Seems obvious that you're only as fast as your slowest shared resource

Less obvious (to your boss): adding people can compound in suprising ways
implication of queueing theory is that adding people to client teams reduces the work getting done, because there's more contention for the shared team. they are typically last to get funding though.

It makes everyone mad, and is slower:
Poorly-factored handoffs rapidly devolve into ticket queues as people get busy

- often can't actually explain work in tickets
- three days later is a bad time to hear that you've made the wrong request
- everyone is mad

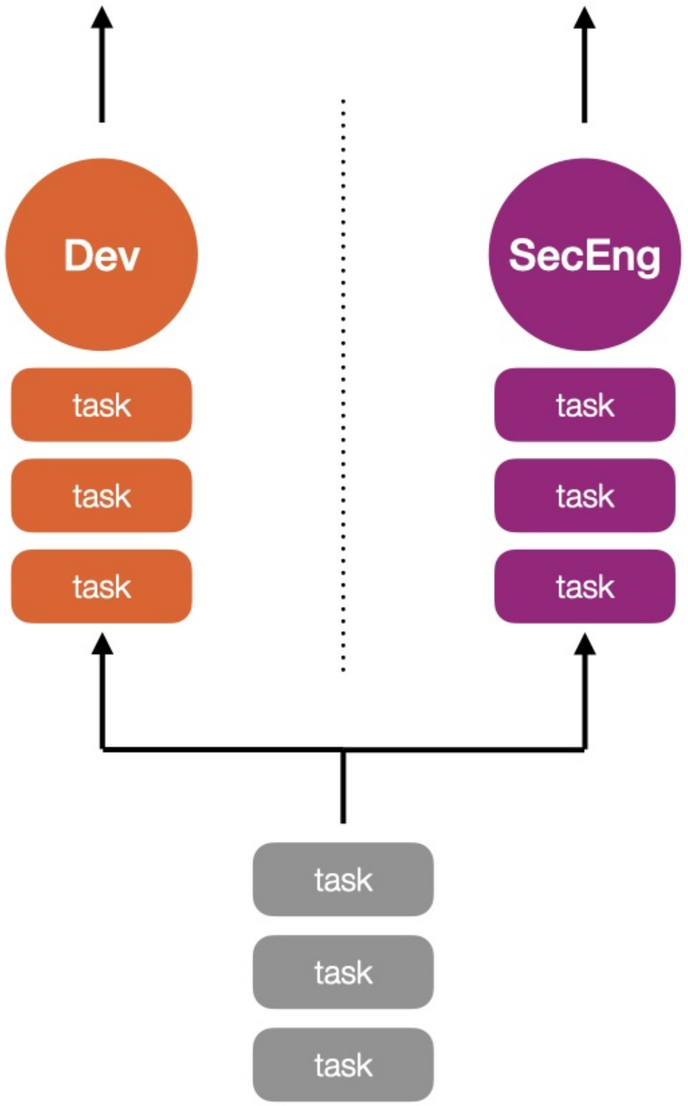
Pattern #3

Bad organization is a radioactive source of extra work

Tickets generating more work than a quick conversation is a specific case of a general issue, which is that poorly factored organizational boundaries generate work that wouldn’t otherwise exist.

“Let’s make a security team to do these security tickets”

Idealized conception



Easy example with which to understand this was at the company I talked about earlier, that tried to use its security team to do all of the security remediation work.

By adding a queue for security tickets and workers to serve them, they were hoping for a result like this. There’d be the same total volume of work getting through the system, and fewer context switches for the development queue.

the ONION

America's Finest News Source

News Local Politics Entertainment Sports Opinion More

Point/Counterpoint

This Regex-based Framework that Rewrites Every SQL Query Will Plague us with Injection Bugs for Decades vs. No It Won't

Nathan Eckert

Point

This Regex-based Framework that Rewrites Every SQL Query Will Plauge us with Injection Bugs for Decades

My colleagues in product development may think that automatically rewriting SQL queries using regexes is a neat idea, but I for one think that our customers will want to use words such as "having" and "where" in their data sometimes.

This effort may put an end to adding "where user_id = ?" to every query as its proponents suggest. But it will come at the terrible cost of malicious actors possibly being able to exfiltrate proprietary data of their victims, whose data we are entrusted to protect.

If you thought Osama bin Laden was bad, just wait until the

Bob Sheffer

Counterpoint

No It Won't

No it won't.

It just won't. None of that will happen.

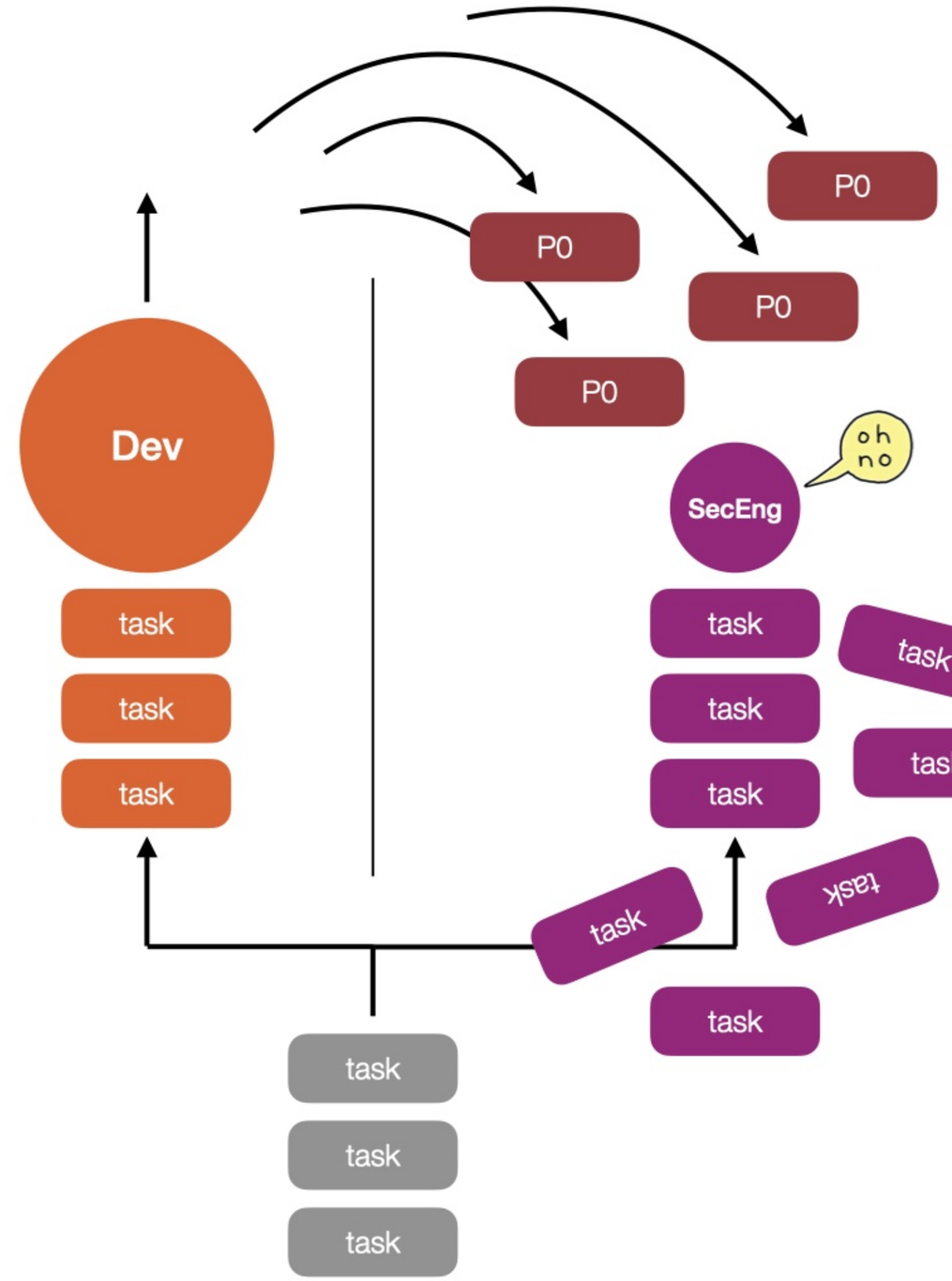
You're getting worked up over nothing. Everything is going to be fine. So just relax, okay? You're really overreacting.

The problem is what development can get up to when they aren't encouraged to be paranoid about security.

Compare:
Security team scrambling to keep up with bugs in a regex-driven sql rewriting framework, vs.
not doing anything remotely like this
If the framing of your security team is that engineers don't need to worry about security, this is what inevitably happens

“Let’s make a security team to do these security tickets”

Typical results



So you wind up generating a lot of extra security work with your engineering activity, after erecting a wall between them.

If you take the queueing analogies too seriously you might expect your overall development throughput to slow down, since the security queue can’t keep up. But in practice this mostly doesn’t happen. The weak point in this setup is the collective will to do security work, so it just doesn’t get done. The security queue overflows and explodes.

On a long enough timescale of course something horrible precipitates, but that’s a problem for future you. In the here and now it’s only a minor humanitarian catastrophe which we can ignore.

Pattern #4

Humans have an impressive capacity for chauvinism

“Why hasn’t SRE shipped this yet? We already did the *hard part**.”

* We thought of it and typeset some high-level proofs in LaTeX

There can be a general tendency among some of us to imagine that other peoples’ jobs are easy compared to ours

Obviously it’s not a majoritarian view that working on the server is so easy “it’s not clear it’s even technical,” but some crazies do hold this view

“We’ll make the platform, you can worry about the *business logic**”

* term of art meaning “your bullshit” (as opposed to my code, which is beautiful)

But these vibes can certainly run in the other direction

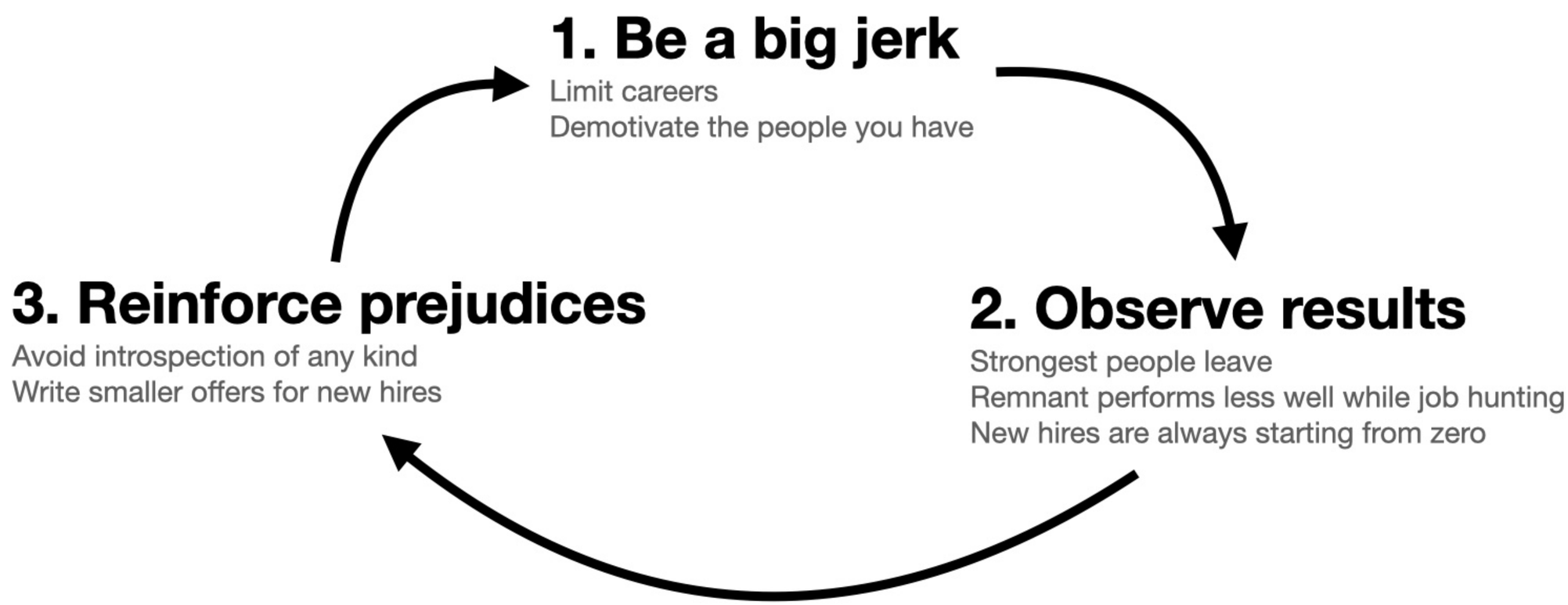
“Google pays backend engineers more than frontend and fastracks them for promotion… almost all backend software engineers were men. Almost all female software engineers, however, were frontend engineers.”

- 2017 article re: a lawsuit Google paid $118M to settle

Backend vs. Frontend very common across the industry
Imposes or enforces career limitations / de-facto classes of engineers
demographically skewed in terms of impact
vicious flyweel
career limiting for people you have, they’ll leave
can’t hire at higher levels
never come in contact with experienced practitioners

The “frontend/backend/SRE/DBA/DS/native client/etc is easy” flywheel of dipshit garbage

how to be stupid while feeling smart



Especially poisonous because although the results of this are very stupid, the behavior of the perpetrators can be coded as “smart” and “valuable” to outside observers

Pattern #5

The devops revolution is … unevenly distributed

(I'll say more about this in a bit, but devops is ...)

The pre-devops age never actually ended
every place i worked claimed to get it
but every place had people with "SRE" titles being sysadmins from 1990
devops is here but unevenly distributed

devops was about empathy

Parochialism & Ego

Hell is empty, these devils are here

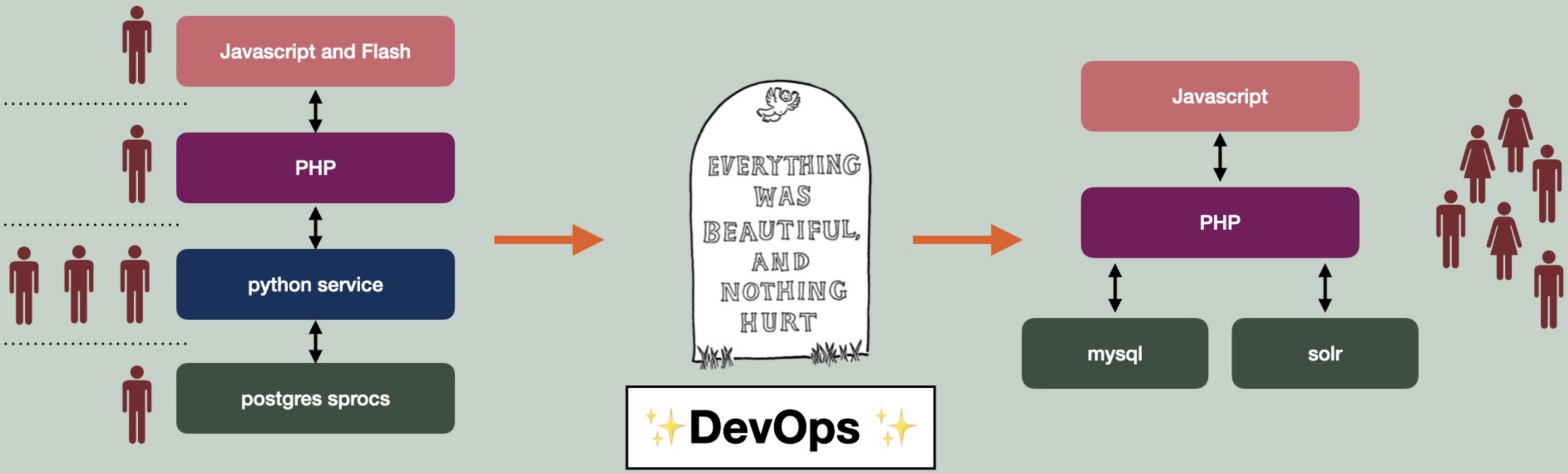
The yin and yang of dysfunction. A recurring theme throughout every chapter of this.
parochialism (lack of curiosity, excess deference) and
ego (being dismissive of others' abilities, or their jobs, or thinking that your job is just to complicated for those simpletons)

Breakdowns in empathy fill our lives with pain.

A Happier Story

Foo bar

+2 years: new team, simpler stack, new ethos



Fiefdoms on the left, everybody more or less pitching in on everything on the right
We imported a bunch of engineering leaders from Flickr who were part of the devops movement
A lot of creative destruction went on, which was tumultuous at the time

that was ultimately successful, and that combined with some other chaos morphed the company into an engineering-led org

After rebooting the team, and after years of backbreaking labor, we crawled out the other end of the tunnel victorious.

You know an engineering team with power is a little like the mule with a spinning wheel.
No one knows how they got it, and danged if they know how to use it.

What should we do with our ~~spinning wheel~~ new mandate?

Proposition 1: Boil the Ocean

1. Feature freeze lmao
2. Embark on five-year rewrite into ~~Scala~~ ~~Golang~~ ~~Clojure~~ Swift.
3. *[Get fired early in year two and be legends.]*

This had actually already happened once or twice, as I mentioned before, so although it had its proponents it wasn't popular

Proposition 2: Dress Like Big Dorks

"Uber decided to buy the SREs leather jackets, and gave fittings to all. But after the company learned that it had enough men's sizes to qualify for a discount but would have to pay retail for the women's jackets, it decided to give jackets only to male SREs."

STEVEN LEVY THE BIG STORY FEB 24, 2017 12:00 AM

Uber's Leather Boys

Susan Fowler's horror story is destined to become a textbook case in workplace discrimination.

the other thing people seem to do in this situation is buy merch
we did a bit of that but not to this extent, and not as a primary strategy

Neither of these felt right, so we continued with business as usual

None of that felt right, though, it just wasn’t us. So we didn’t do anything drastic for a while.

…then one day, our designer broke the build late at night.

Randy, our designer (presumably reacting to a more recent debacle)

Re: drive-by commits to main branch

Hello!

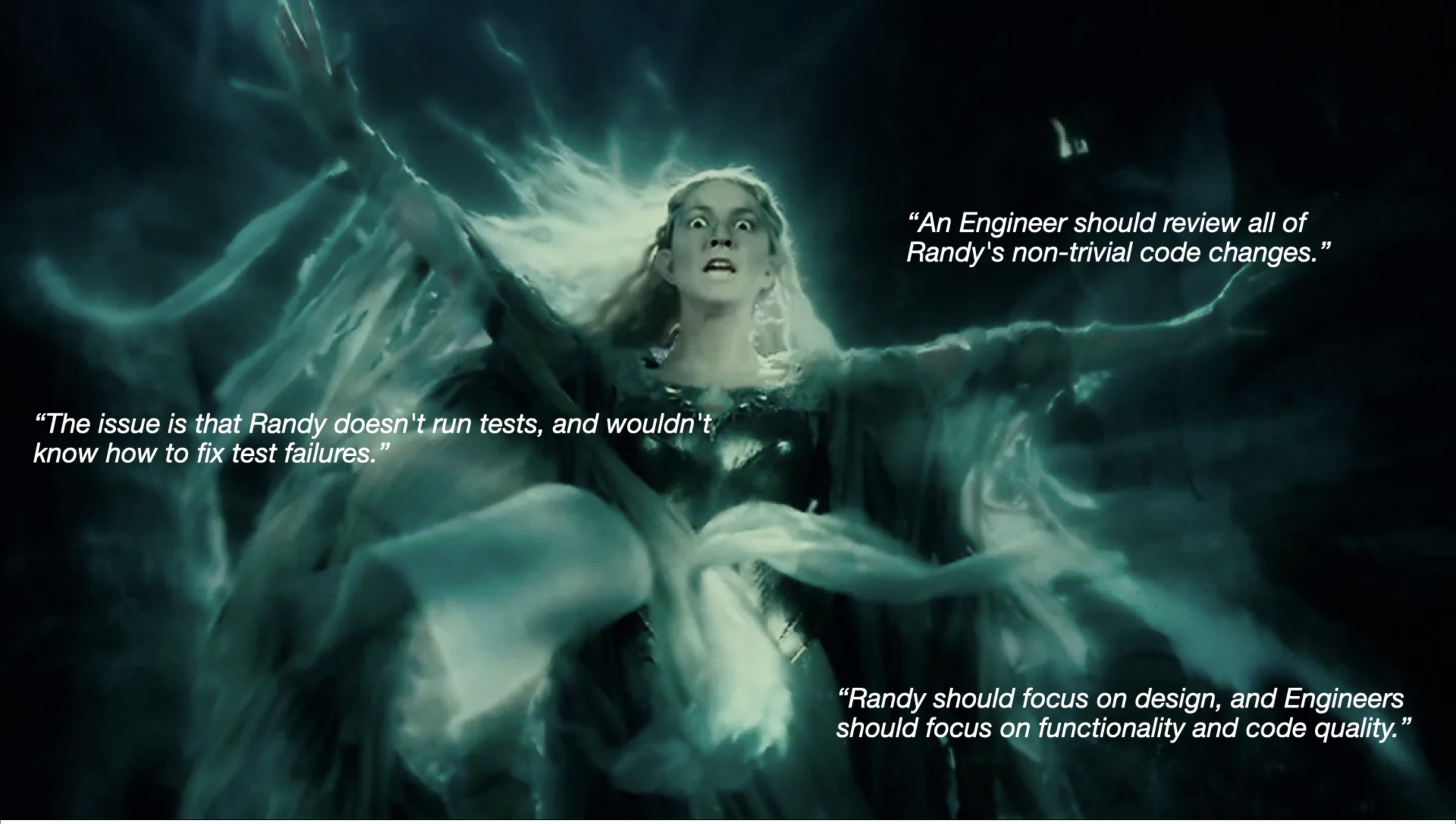
I think I'm the guilty party here. I commit to trunk. I don't have the keys to deploy.

I can't really express how sorry I am that this happened.

I owe everybody involved a massive and sincere apology.

Randy

He was obviously very apologetic about the whole thing



And the temptation was there to be huge jerks, and some folks did dip their toes in those waters.
This presented the possibility that maybe we should put up a wall between designers and the css source code.

Something Interesting Happened Instead

Being high on our own supply was beneficial for once

Re: drive-by commits to main branch

```
Randy is in the deploy group now. The how and when can
be worked out off-list, but it is now *possible* for
him to deploy to prod.

- Kastner
```

But instead an inspired person just yolo leaned into it, and gave the designer the deploy keys.

I've come to understand this event as being entirely contingent. I've seen lots of teams that reached this moment and made different choices, and we weren't smarter or better or anything. We were just taking the devops zeitgeist hilariously seriously at that moment in time.

Re: drive-by commits to main branch

Yes!

I dub thee, Engineer.

- Mike

> PRODUCTION deployed by *rhunt* build:

> 32944-trunk-20100810-230300-

> UTC took: 185.038392 seconds diff:

> http://deployinator.etsycorp.com/diff/web/32930/32944

And it was great. Designers started shipping code and we did everything we needed to in terms of monitoring, test suites, et cetera to make that safe for them to do. Everyone rejoiced and got shit done.

Proposition 3: Empower Everybody Else

That got us to an epiphany: the thing we were actually going to do was use our rare and precious organizational power, and the free time that came with it, to lift up other teams and make them more effective.

Committing to the Bit

What happens when engineering gets really into force multiplication

- Designers can deploy css.
- Customer support can deploy help updates.
- Product managers can run analytics queries.
- The lawyer can handle takedowns.
- Board members can deploy the website as a stunt when they visit.
- Engineers are automatically emailed / followed up with in person after they ship a dangerous PHP function, of which there are many.

There were a lot of epiphanies in that era but they all kind of felt like that one.
That was the process of getting from a place of not executing to executing really well.

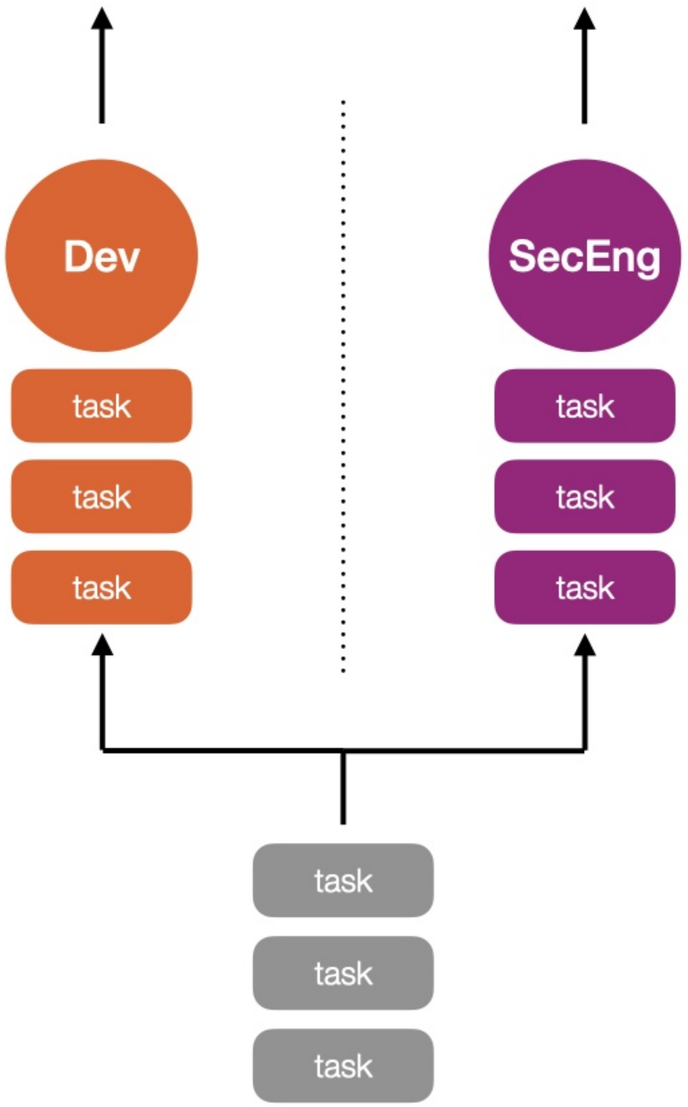
What Executing Well Felt Like

Domain experts, not domain owners

- Make part of your job making other people plausible within your discipline
  - not to do all of the security tickets, build stuff to make others' code more secure
  - not to do all of the analysis, make sound analysis possible by others
  - not to do all the deploys, but to make sure there are safety nets that make them safe for everyone
  -

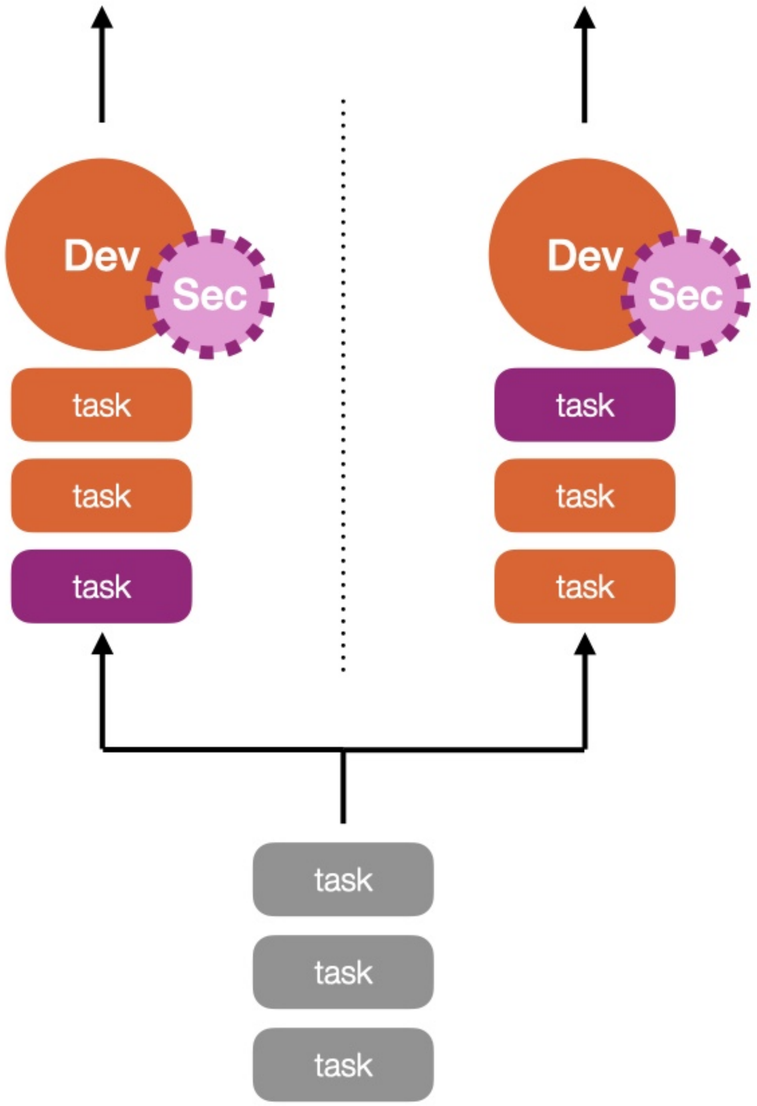
“Let’s make a security team to do these security tickets”

i.e. the wrong way



TKTK

TKTK



- Great security orgs level everyone else up

focus on unblocking everyone else
grease the wheels for others making progress

Lesson #2

Part of your job is making your discipline more accessible

Lesson #3

New roles/titles should cost someone a finger (metaphorically)

Obviously do it, but only when you have no other choice
On the whole, I think separating frontend and backend engineers as roles was a mistake the industry made

consider handoffs and role specialization to be smells.
These are eventually necessary
But avoid them until you just can’t

Proactive cross-pollination

bootcamps
senior bootcamps and onboarding bootcamps
cross all the streams
teach everyone a little bit of everything

What works?

- understand your colleague's work! Care about it!
- Genuine / radical curiosity about what other people do
-

- the "in-group" scaled way up

Lesson #1

Understand your colleages' work! Care about it!

how do you care about it? well,

Intentional team values

- What it felt like:
  - psychological safety in changing someone else’s stuff
  - expectation that you’d leave things in a better state

- Think about your team values and talk about them (we are a team that does windows. We are a team that returns shopping carts to the corral even if nobody's watching)
  Elitism is poison - we do windows

- Never use GraphQL

"code as craft:" meant blue collar. we do windows. we are not above ditchdigging.

cultivate and reward curiosity

GraphQL: Not Today, Satan

Lots of tech out there directly promises that you'll never need to talk to engineering again, or you won't need to talk to some other team of other engineers again.

I'm picking on GraphQL which, to be fair, doesn't do that overtly in its own marketing materials. However so many teams I've been around have tried to use it to avoid writing queries, knowing what a database index is, or talking to their local DBA's and SRE's. This always goes poorly, for reasons that transcend the team dynamics.

Revise visual

Experience Our AI Powered CDP

You'll love never talking to your engineers again

INTUITIVE INTERFACE
No Coding Necessary
And also you can put a JSON blob in a text box buried in settings!

DEPLOYMENT SIDECHANNEL FTW
Add our tracker and ship arbitrary javascript of dubious quality to your change password page today!

CANNOT FAIL, CAN ONLY BE FAILED
Feel the rush of chasing the ETL dragon as your application evolves.

GraphQL (when conceived of as a device for factoring out backend development) is just one tool in the class of tools that make similar false promises.

Right now your CMO is being pitched on some crapware AI CDP that they're told will let them never talk to you again, once you do all the work to integrate it, of course.

This is a trap, and I think it's one that great teams know how to avoid.

Parting Thoughts

~~Parochialism & Ego~~

Inside and outside of tech the world is rife with cult leaders who lack the gene for humility - bullies who inspire and encourage cruelty in their followers to no end other nihilism.
Valuing others around you should not be a radical act, but that’s where things are heading.

I have attempted to live the movement

With varying degrees of success

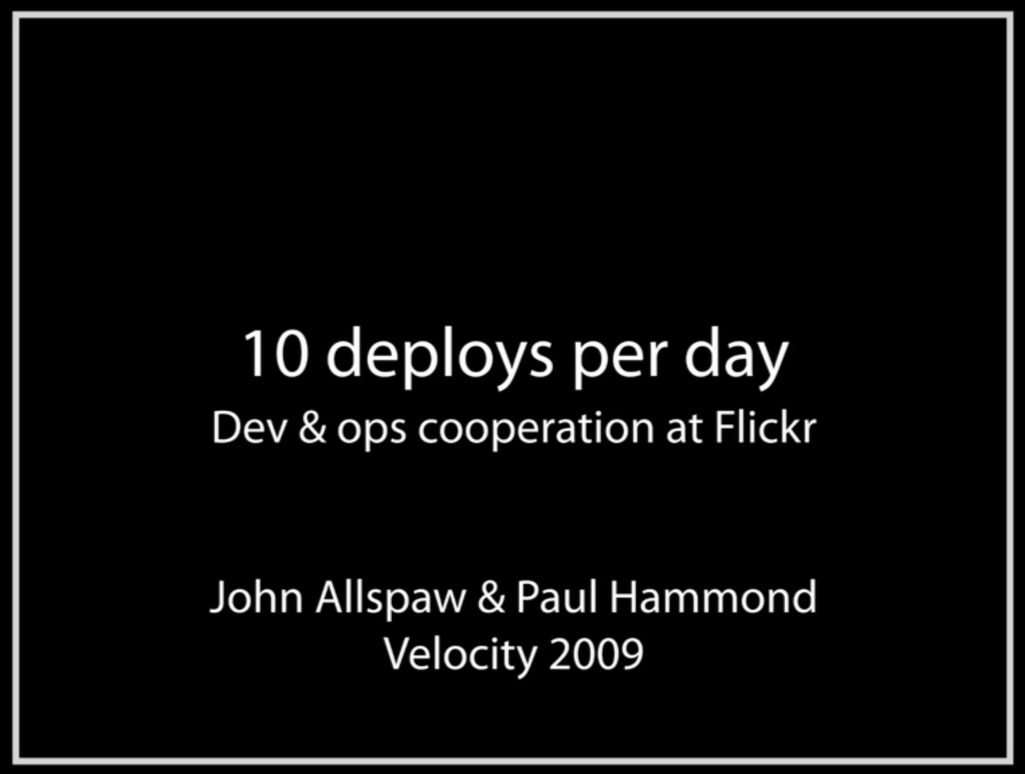
- **Growth teams**: what if we took marketing seriously and tried to help them, instead of just shitposting?
- **Continuous delivery**: turns out product developers actually love being empowered to fix the stuff they break.

Yes, growth hacking is a bad word and everything. But I have started growth teams twice and the one weird trick to making marketing start to work is to … try to help? It's crazy.

I inherited a situation where SRE was doing all of the deploys for the company, truly partying like it was 1999. I pushed to switch that to developers shipping their own code. There was some hesitation, the worry being that developers would break stuff and not fix it. But the better angels of our nature prevailed, engineers largely resolved their own incidents right away. Product engineers actually asked to be handed the pagers! It was heartwarming.

- Why don’t more people try this? what do people need to do it?
  - Parochialism and ego need the investment of active energy to be destroyed
  - (Permission, it’s on us as leaders to spot this stuff)
  - Slack in the system. Requires persistent commitment. Skip the bootcamps, it’ll die out
  - I’ve made the case in a few places that it’s penny-wise and pound foolish to
  - We don’t let our machines run at 100% utilization. But again, we love our computers more than ourselves
- Code as craft - thing people always asked me about. Pov important. Our pov was that it was that the world needed ditchdiggers too. It’s us!

The Urtext of my Tradition



In the industry zeitgeist, devops went through the "this is a thing we need, so it's a role" arc and you eventually saw job listings for devops engineers. Then eventually it was a tool you were buying, and the tool would let you rub some devops on things. Then it went out of fashion.
But the entire idea was that the roles should exist with a blurry line of responsibility.
This was just two friends of mine from Flickr standing on stage saying: "hey I'm John, hey I'm Paul, we are friends and cooperate."
Working at a high-functioning eng org. meant just taking this extremely seriously, initially between dev and ops.
But eventually we took that premise seriously in an omni-directional sort of way and that's where the magic came from.

DevOps at 15:

What if DevOps, but for everybody?

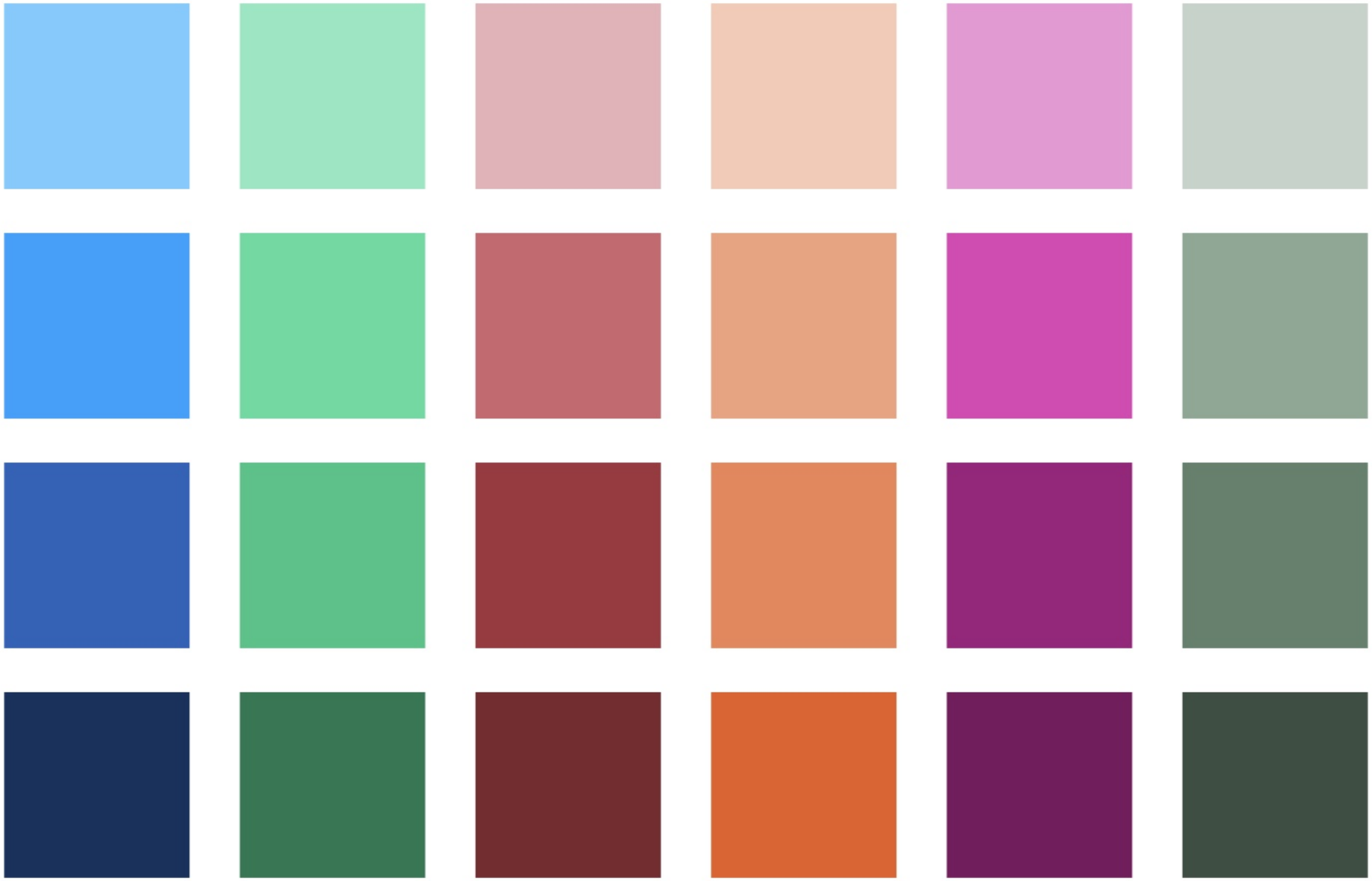
That was 15 years ago now, which was long enough in the past that they used square slides.
Most places I've worked were improved by taking the spirit of DevOps seriously, and in my opinion they could have used more of it.
The industry as a whole would be improved by the same.

Parking Lot

Key Results

- Number of features shipped over two years rounds to zero.
- Pointless Python middle layer rewritten twice, though.
- Treaded water on scaling, which was much more than nothing.
- Engineering founders, the entire ops team, and most engineers fired (but not me).
- Ultimately it was fine because the business was unkillable and we got better (more on that later).

Make sure to credit https://codahale.com/work-is-work/ in header of talk page with the disclaimer about wikipedia



In addition to
SRE's were also tasked with doing deploys, which was
Which was surprising in that it's arrangement from the early 2000s, which is kind of surprising if you understand "SRE" to be some post-devops evolution