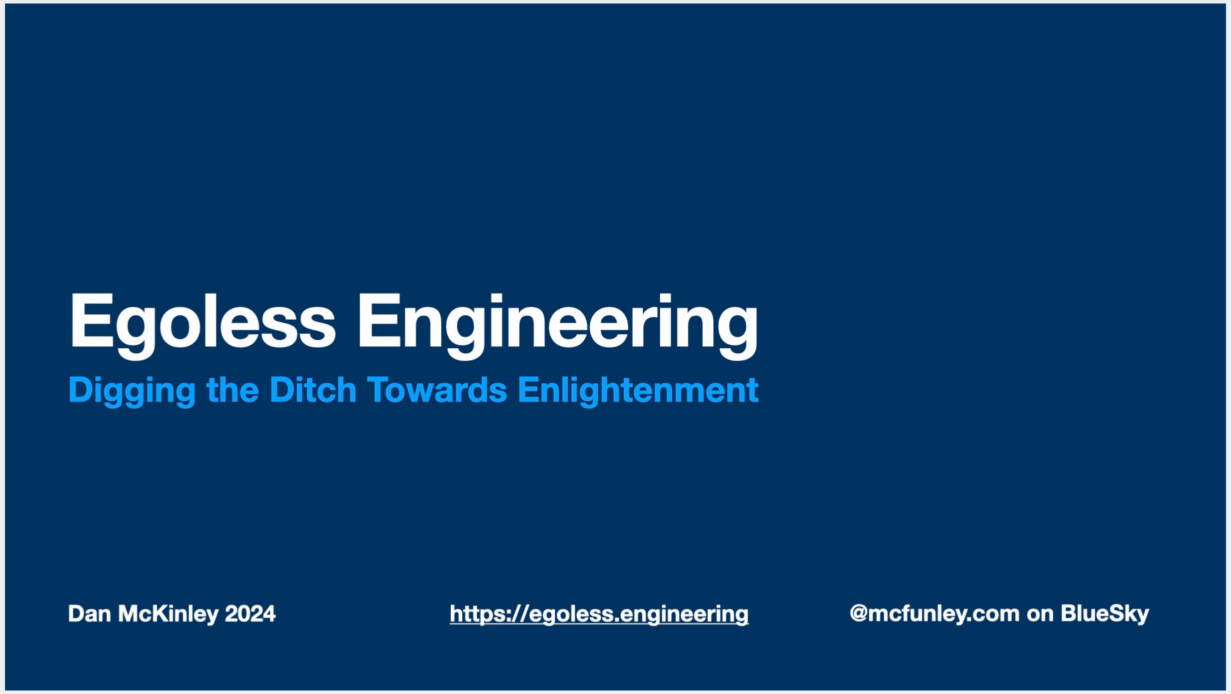
Egoless Engineering

Like many of you, I was raised in the background radiation of Calvinist thought. I expected little but redemptive hard labor, before presumably one day dying in a mine. I also read Hackers & Painters at an impressionable age and was kind of a jerk about it for a while. This talk is about how despite this, I got better.

The real urtext of my tradition was two people standing on stage explaining how although they have different jobs, they are friends, and try to help each other. It was mindblowing at the time and led to many subsequent revelations, and ultimately a blissful state of psychic death. It turns out misery is a shitty proxy metric.

Here are some other talks and my website.

@mcfunley.com

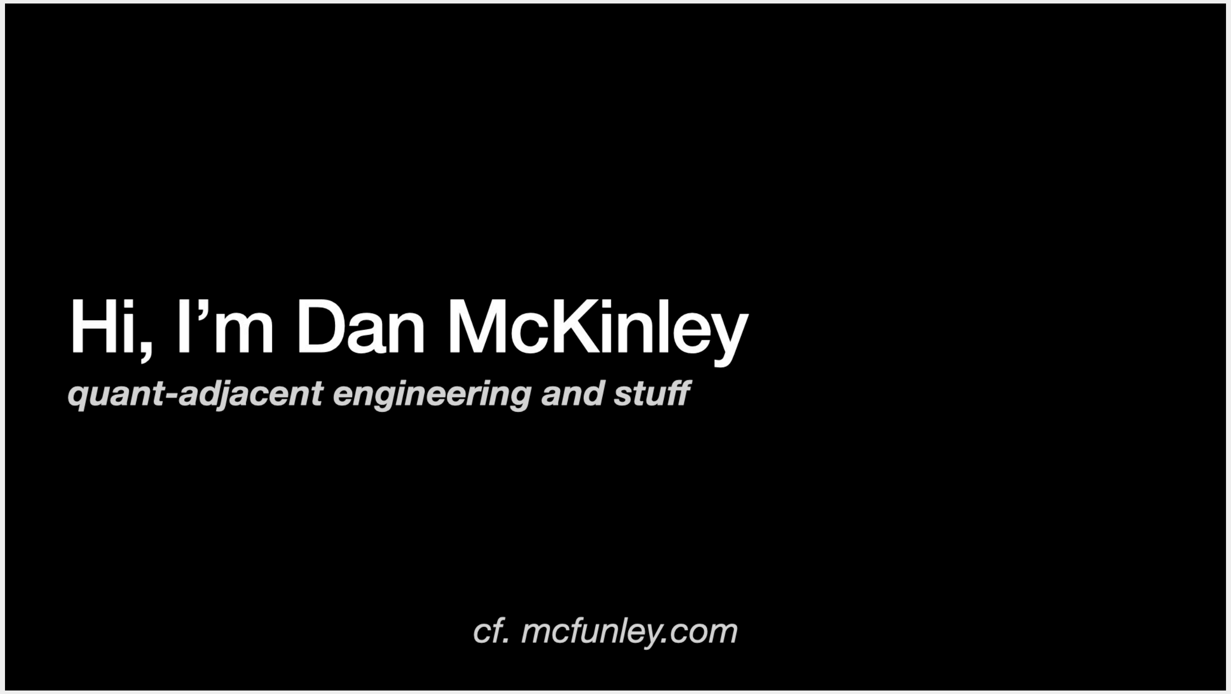


Egoless Engineering

Digging the Ditch Towards Enlightenment

Dan McKinley 2024 https://egoless.engineering @mcfunley.com on BlueSky

Hey everyone
#



Hi, I’m Dan McKinley

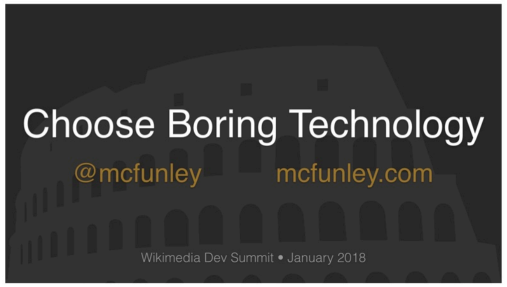
quant-adjacent engineering and stuff

cf. mcfunley.com

I’m Dan McKinley. It’s nice to talk to all of you.

I’ve done engineering since the turn of the century, and led small teams and biggish orgs.
#

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



Choose Boring Technology

@mcfunley mcfunley.com

Wikimedia Dev Summit • January 2018

Data Driven Products Now!

Dan McKinley @mcfunley mcfunley.com

April, 2016

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

I’ve spun the wheel enough times that I’ve gotten a few stretches of my career when things were going great. Nothing gold can stay though, and those moments were fleeting.

Welcome to my latest talk in my very-slowly-moving series about trying to rekindle that magic.
#

At most places I've worked, things haven't been going great. I'm not really complaining. I was in a position of responsibility at those places, and I hope I did more good than harm.

I'm going to talk through some stories about that. My hope with this is to do more good than harm.

Maybe y'all can create more fulfilling places to work, and hire me to work there.
#

What this talk is about

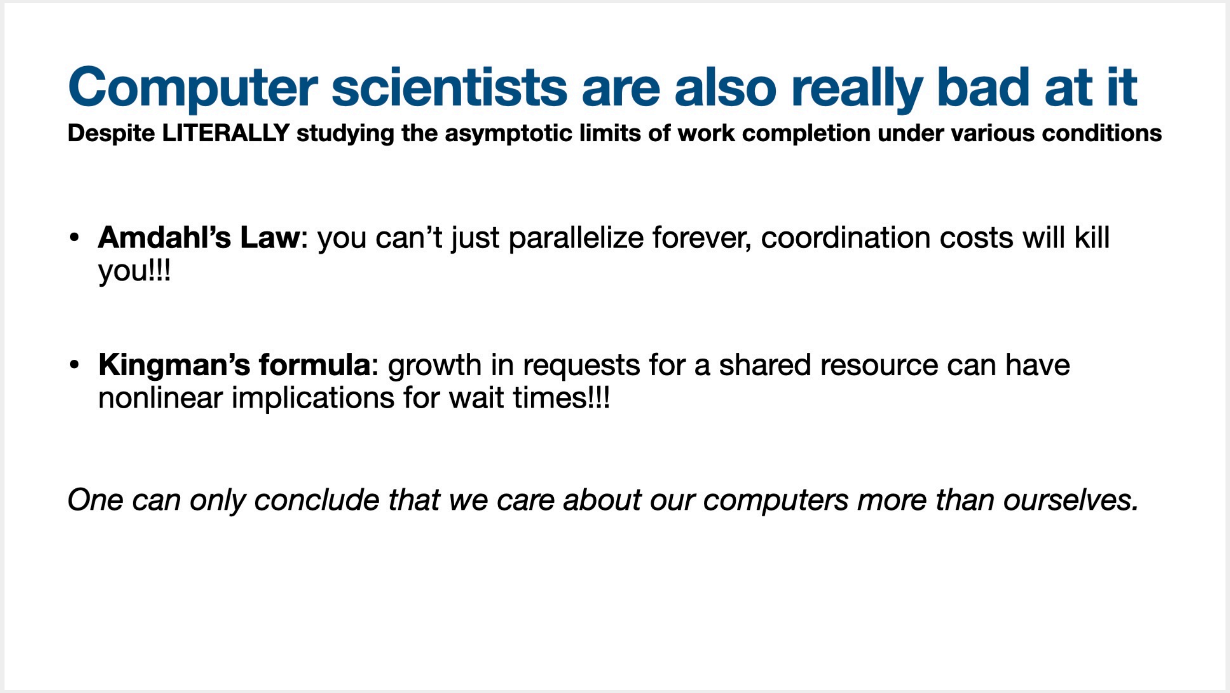
So I have noticed similarities when things are going poorly, and would like to talk about them.
#

~Everyone honks up by diffusing responsibility poorly

One big problem everywhere is that once you have two employees, you have to divide responsibility somehow.

How you do that is really important. It's a move that has both immediate and second-order consequences which are mostly awful to contemplate.

So most companies don't contemplate it that much.
#



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.

Computer scientists like me start their career by realizing that they accidentally joined a math major that isn't a science, and isn't about computers. It's about how abstract work gets done and stuff.

And the trauma of this initial realization is so severe that we spend our entire careers refusing to apply anything we learned outside the domain of computers.

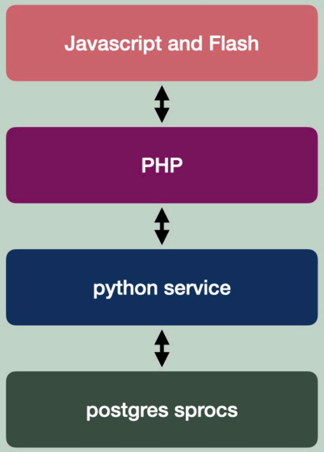
Things might be better if we tried to think about work as much as we think about the computers. Later in this talk I will try, despite how much it hurts me to do that.
#

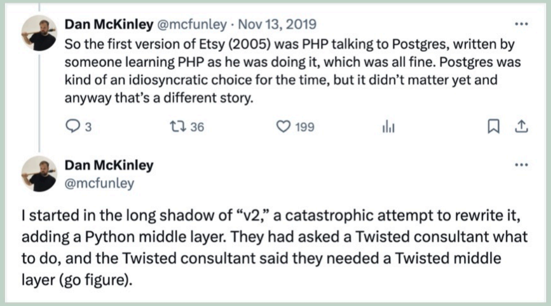
Sad Stories

I'd like to talk through the dynamics I've seen at different places. I'm going to anonymize it somewhat because even really successful companies suffer from pathology, and my intent is to learn rather than just shitpost.
#

I worked for a startup

The first job I want to talk about was a startup I worked for, which ultimately was really successful. I was in the first 20 employees there.
#





Dan McKinley @mcfunley · Nov 13, 2019

So the first version of Etsy (2005) was PHP talking to Postgres, written by someone learning PHP as he was doing it, which was all fine. Postgres was kind of an idiosyncratic choice for the time, but it didn't matter yet and anyway that's a different story.

3 36 199

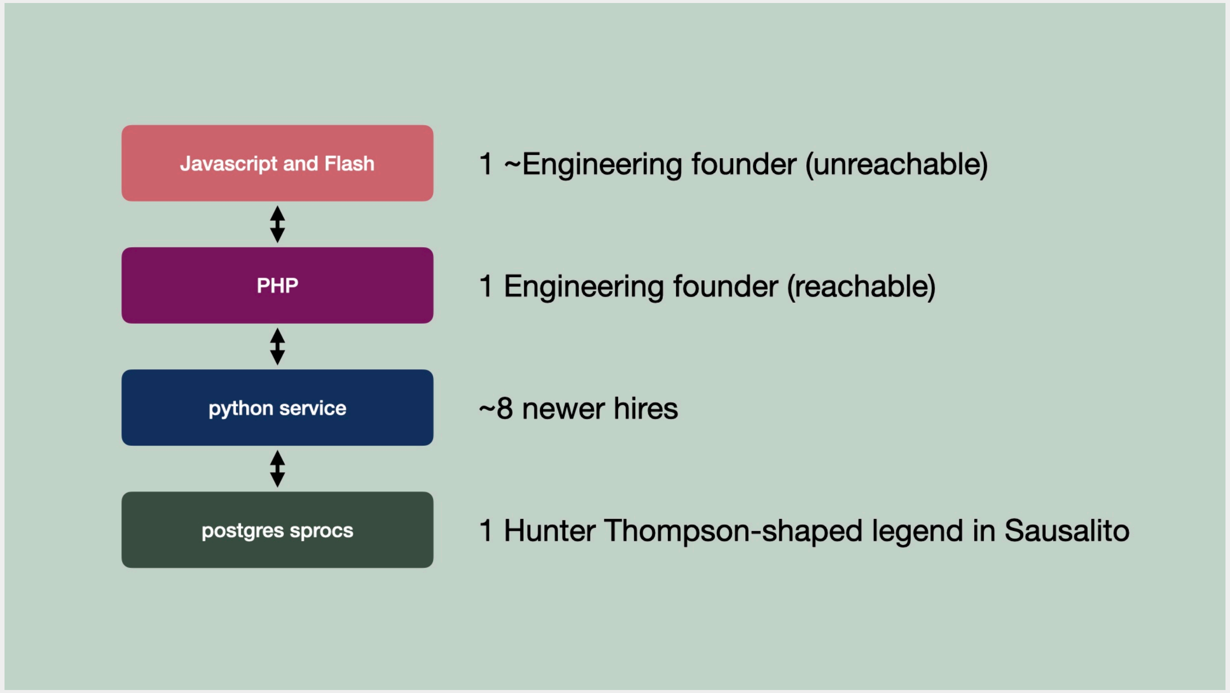
Dan McKinley
@mcfunley

I started in the long shadow of "v2," a catastrophic attempt to rewrite it, adding a Python middle layer. They had asked a Twisted consultant what to do, and the Twisted consultant said they needed a Twisted middle layer (go figure).

You'd think this would be a really scrappy group but no, at that point they had already managed to set themselves up like IBM in the 1970s or something.

I was hired to work on a python middle layer which should not have existed in the first place. In this misguided attempt to scale, they added a Python middle layer. The theory was that web requests would be served faster if they progressed through more network hops.

This is a bad theory.
#


Javascript and Flash
PHP
python service
postgres sprocs
1 ~Engineering founder (unreachable)
1 Engineering founder (reachable)
~8 newer hires
1 Hunter Thompson-shaped legend in Sausalito

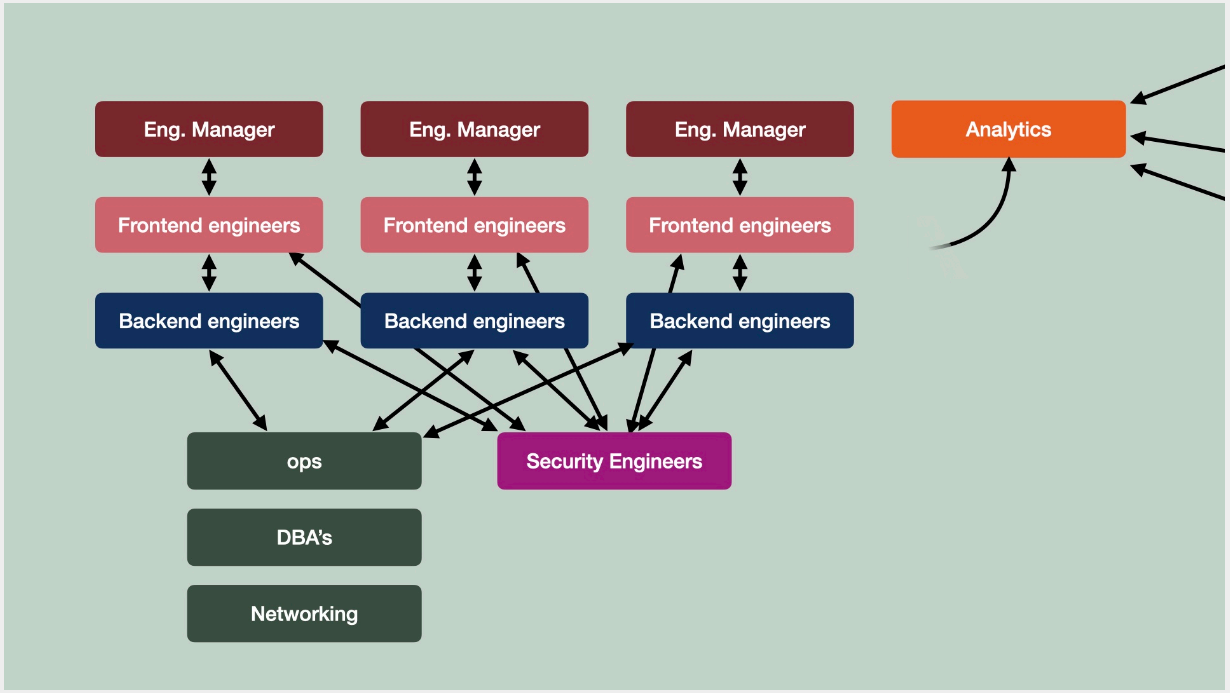
So to launch any feature at this company, you needed to get a database person to do some DDL and write some stored procedures. You needed to get a python person to write some python that didn't achieve much. You needed a PHP person to write some frontend code. And so on.

As a result of all of this, the company released zero features for two whole years and everyone got fired.

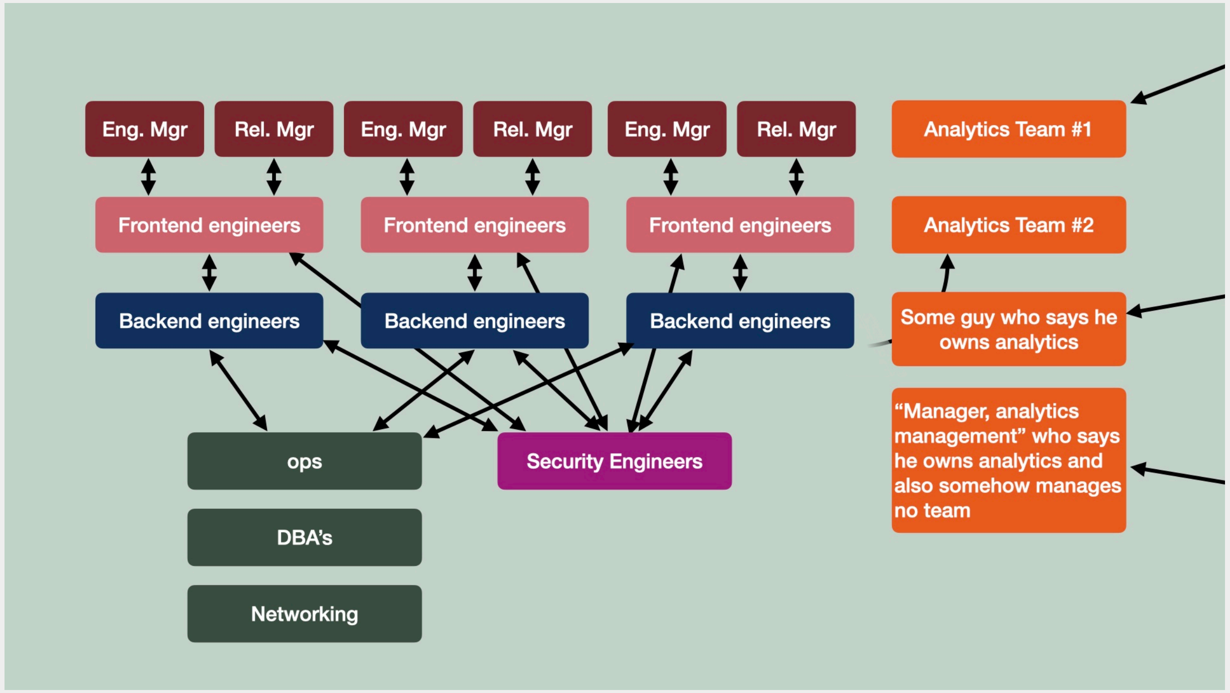
(But not me, I survived by complaining.)
#

I worked for a B2B service provider

Much later on I worked at a fairly mature company that had more than a thousand people and was selling a b2b product.
#

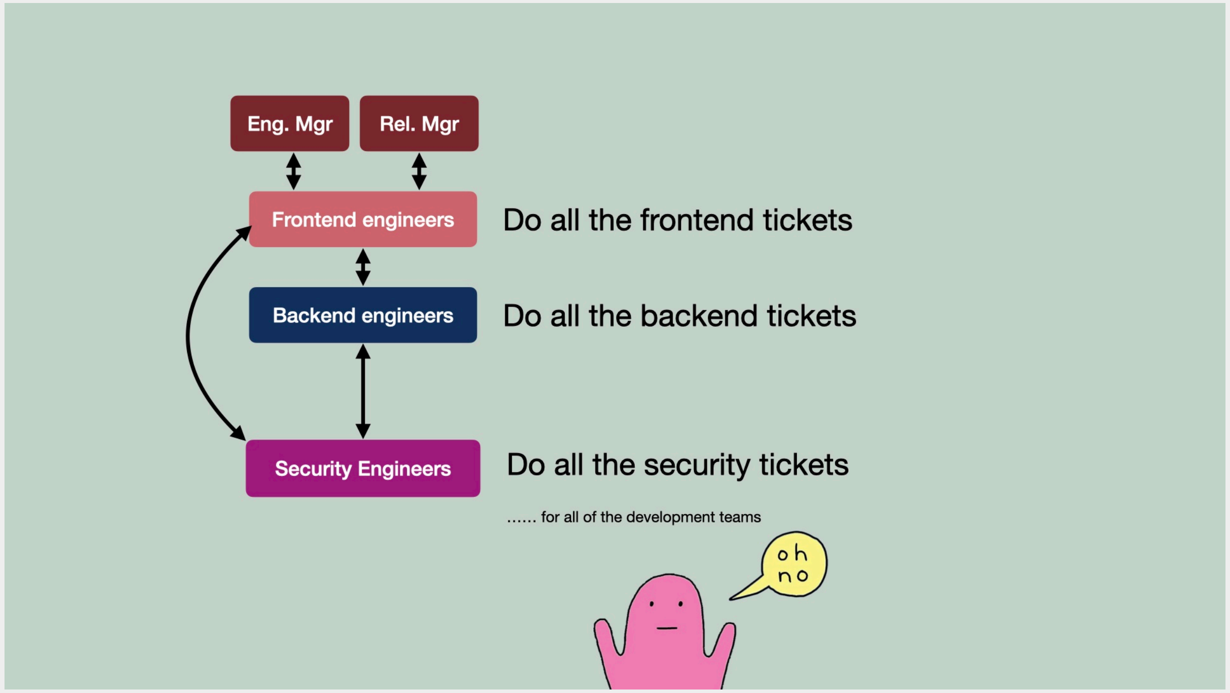


They'd managed to avoid serializing absolutely everything, in that they'd divided into feature teams. And they had shared services teams like ops, and DBA's, and so on. That seemed reasonable at first, or at least not different than most places.
#



But something interesting had happened where the roles just kept dividing, like a bacteria colony out of control or something.

They split engineering management in half and created a role called "release manager." To … manage releases, I guess? It was never entirely clear to me why. At some point things just came unglued.
#


Eng. Mgr
Rel. Mgr
Frontend engineers
Do all the frontend tickets
Backend engineers
Do all the backend tickets
Security Engineers
Do all the security tickets
...... for all of the development teams
oh no

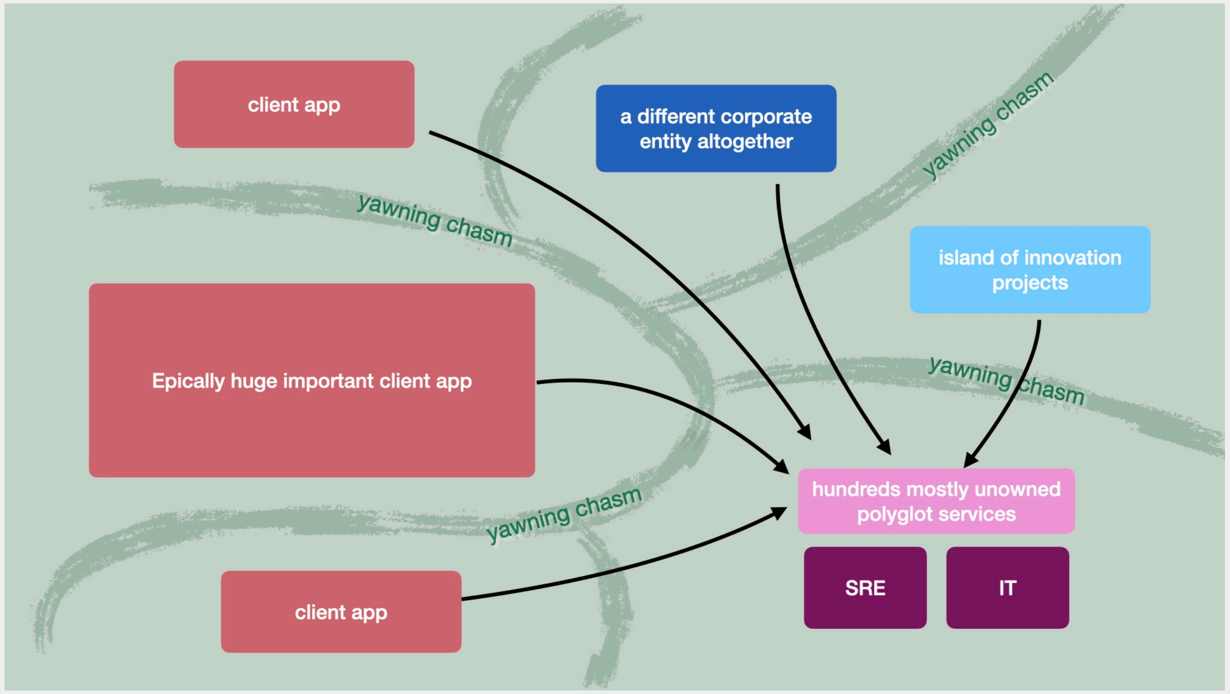
And they chased that dragon into some very dark corners.

A big mistake they made in my view was that they conflated roles with work a little too much. Frontend engineers did all of the frontend work and none of the backend work, and vice versa.

That was survivable, but taking that policy to the security team and making the security team do all of the security work was not. I'll come back to this.
#

I worked for a native client company

And then finally I worked at a company that mostly made native client applications. There was initially one big hit client app, but by the 2020s a bigger, disjointed set of efforts had grown up in parallel.
#


client app
a different corporate entity altogether
yawning chasm
yawning chasm
island of innovation projects
Epically huge important client app
yawning chasm
yawning chasm
hundreds mostly unowned polyglot services
client app
SRE
IT

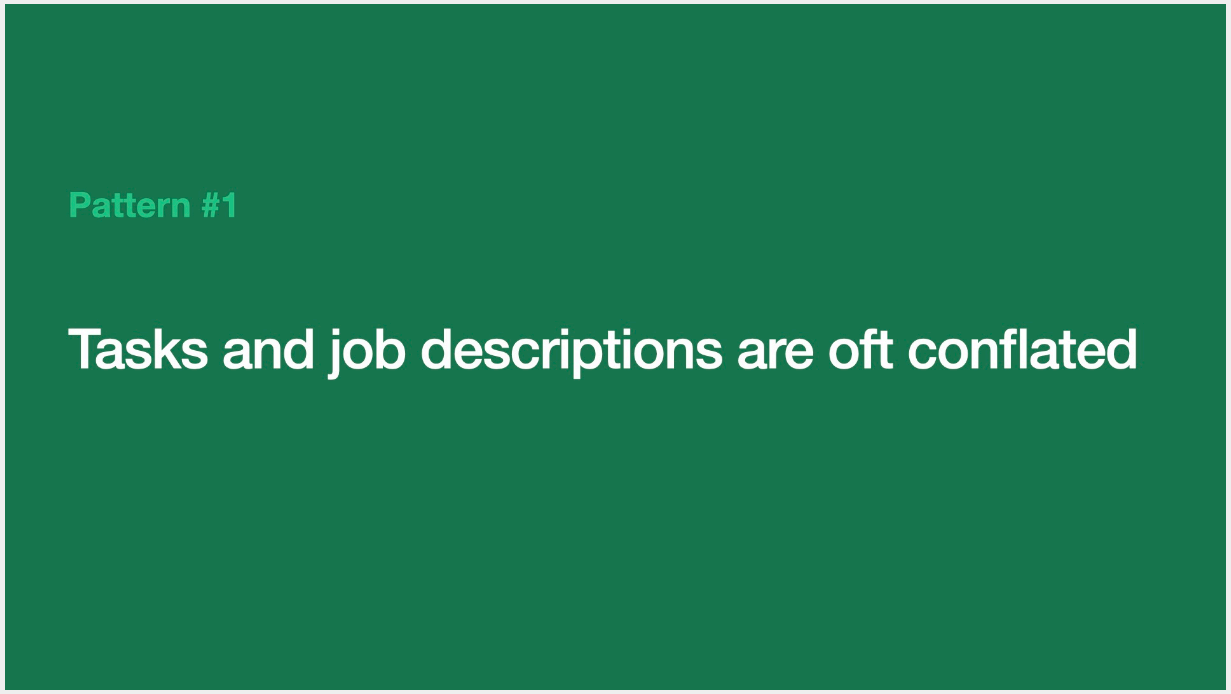
The product portfolio as a whole had a very weak supervisory structure. The products didn't coordinate in any meaningful way on tech stacks or other decisions, since they all independently reported up to a CEO that made no attempt to coordinate them.

But despite that, an attempt was made to have a shared operations function. This presented a lot of difficulty because the operations group wasn't involved in architecture decisions.

They wouldn't have had enough time to do that anyway, since they were busy with hundreds of services that dev teams had moved on from years before. It's a tale as old as time.
#

Generalizing the Hellscape

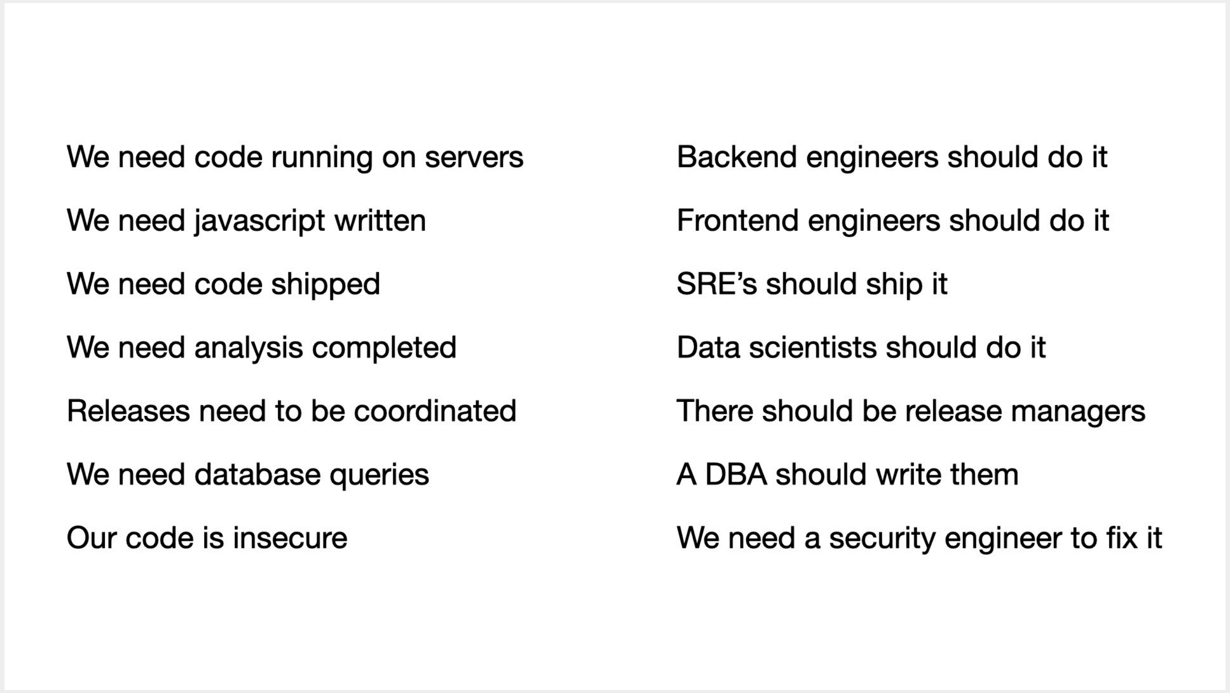
So that was a high-level overview of how all struggling companies are unique. But they're also all the same. I've spent many nights pondering the mistakes these places have made, and I'd like to inflict my observations on you now.
#



Pattern #1

Tasks and job descriptions are oft conflated

One big thing that a lot of people love to do is create new role types. For any new thing a company wants to do, the tendency is to put up a new job description.
#

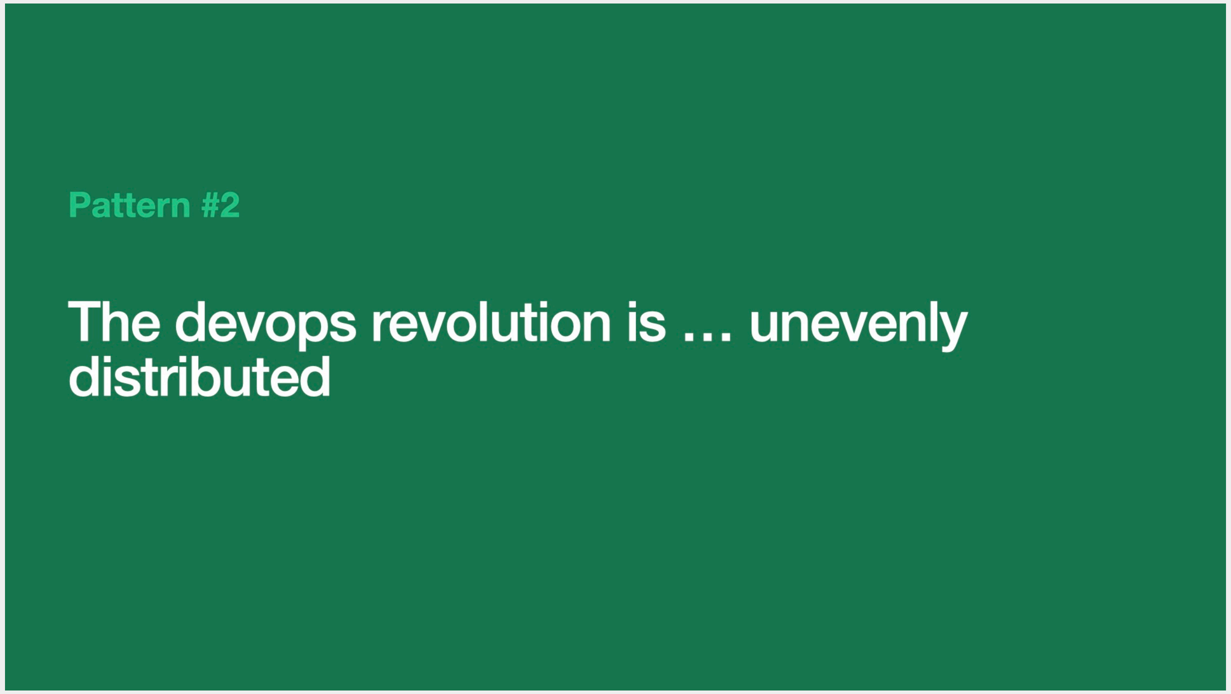


We need code running on servers	Backend engineers should do it
We need javascript written	Frontend engineers should do it
We need code shipped	SRE's should ship it
We need analysis completed	Data scientists should do it
Releases need to be coordinated	There should be release managers
We need database queries	A DBA should write them
Our code is insecure	We need a security engineer to fix it

I think a lot of people notice this and chafe at it when the role is for the new hotness. For example, every company wants to rub some AI on their stuff now, so they are putting up job descriptions for AI engineers.

If you're an engineer interested in AI sitting in such a company, you're annoyed that they're doing this (and potentially paying that person more than you) when you could easily rub some AI on some stuff.

This is just a salient example of a dumb mistake that happens in many, many more dimensions all the time.
#



Pattern #2

The devops revolution is … unevenly distributed

A mundane example of that is the ops and development divide (or these days maybe the divide between SRE and development).

That divide has been very pronounced at nearly every place I've worked.
#

See you know how to *say* "devops"
But you don't know how to *do* the devops
And that's really the most important part of the devops
Anybody can just say "devops"

And that's despite every single one of those companies claiming that they quote-unquote "do devops."

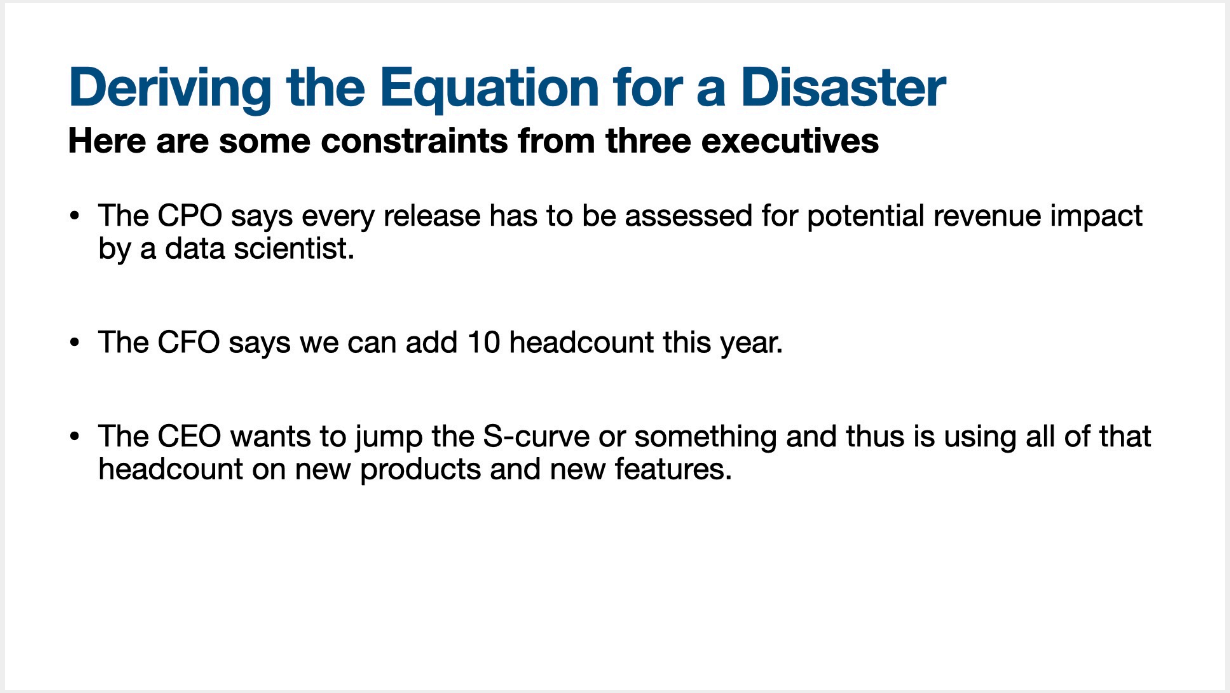
Devops as a portmanteau was meant to communicate something about breaking down the divide, and building empathy between these two groups. I have worked hard to cultivate this, personally, but rarely found it in the wild in reality.
#

Pattern #3

Strict division of labor feels obvious, but is wrong, and also sucks

Once people have subdivided work, they naturally try to arrange people into assembly lines. If you have AI work, hand it over to the AI person. If you have ops work, give it to the ops person.

That's the kind of thing that feels obvious to leaders, but I think it's wrong. And sometimes it's wrong in a mathematically provable way. Let's dig in.
#

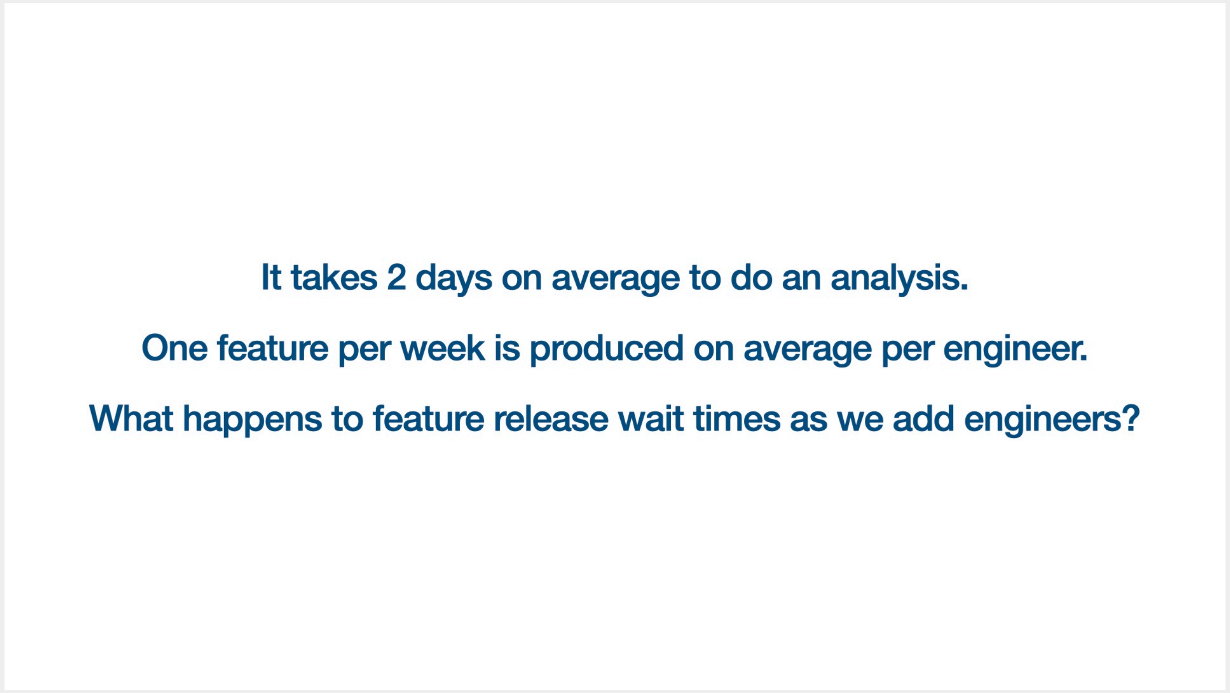


Deriving the Equation for a Disaster

Here are some constraints from three executives

- The CPO says every release has to be assessed for potential revenue impact by a data scientist.
- The CFO says we can add 10 headcount this year.
- The CEO wants to jump the S-curve or something and thus is using all of that headcount on new products and new features.

Here’s a very typical set of executive constraints. They might say: we have to measure everything. We aren’t going to hire more data scientists. We’re going to focus on building new products and only hire new engineers for that.
#



It takes 2 days on average to do an analysis.

One feature per week is produced on average per engineer.

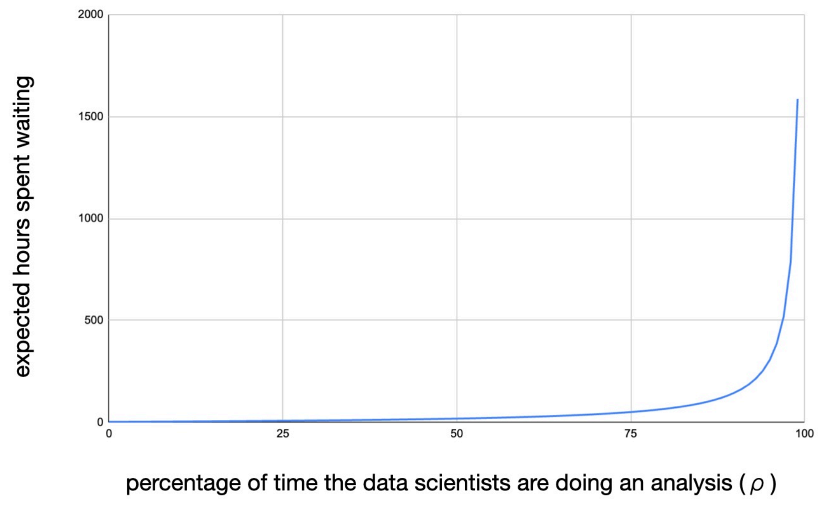
What happens to feature release wait times as we add engineers?

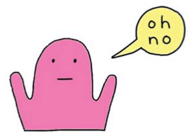
Those are inputs to a very simple equation.

Engineers produce work that needs to be analyzed at a rate, and the analysis takes a finite amount of time.

Remember the premise is that we are trying to go faster by adding engineers.
#

$$\text{wait time for analysis} = \left(\frac{\rho}{1 - \rho}\right)\left(\frac{0.5^2 + 0.25^2}{2}\right) \times 16 \text{ hours}$$





But, whoops! We made everything go slower, actually.

With that simple setup, it doesn’t take very long at all for things to get extremely out of hand. We added a few engineers and wound up with a nonlinear increase in how long people need to wait before they can ship things.

By the time the data scientists are fully busy, the individual feature waiting for analysis is absolutely doomed.

(Source for the math is here.)
#

It’s so much worse though

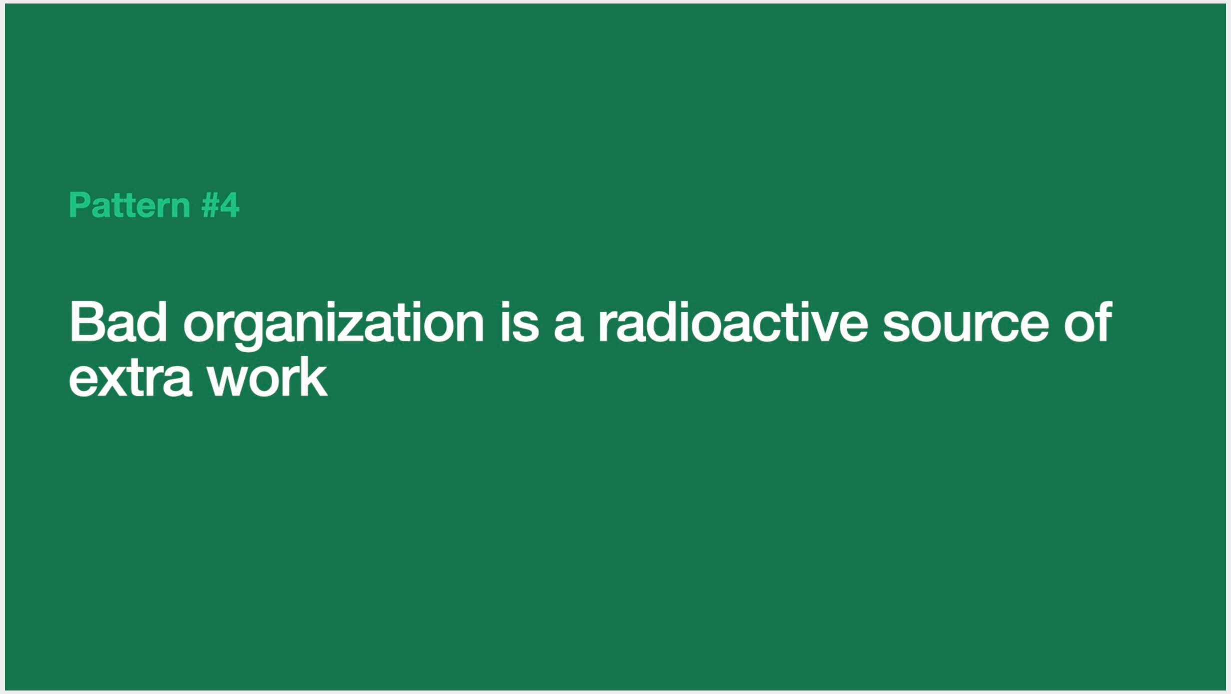
Since the data scientists are dying,

- You can only interact with them by submitting tickets.
- Nobody can explain their feature 100% correctly in a ticket.
- 120 hours later is a bad time to be told you asked the question wrong.
- They’ve also got no time to make tools or datasets you might use to do the work yourself.

Executives are:

- Typically graph and equation blind.
- Well-adapted to insisting on fundamentally contradictory goals.

Of course, organizational dynamics are stacked against analyzing this problem in a clearheaded way. Everyone thinks the other groups are sandbagging. We come into petty conflicts that obscure the physics of the situation.
#

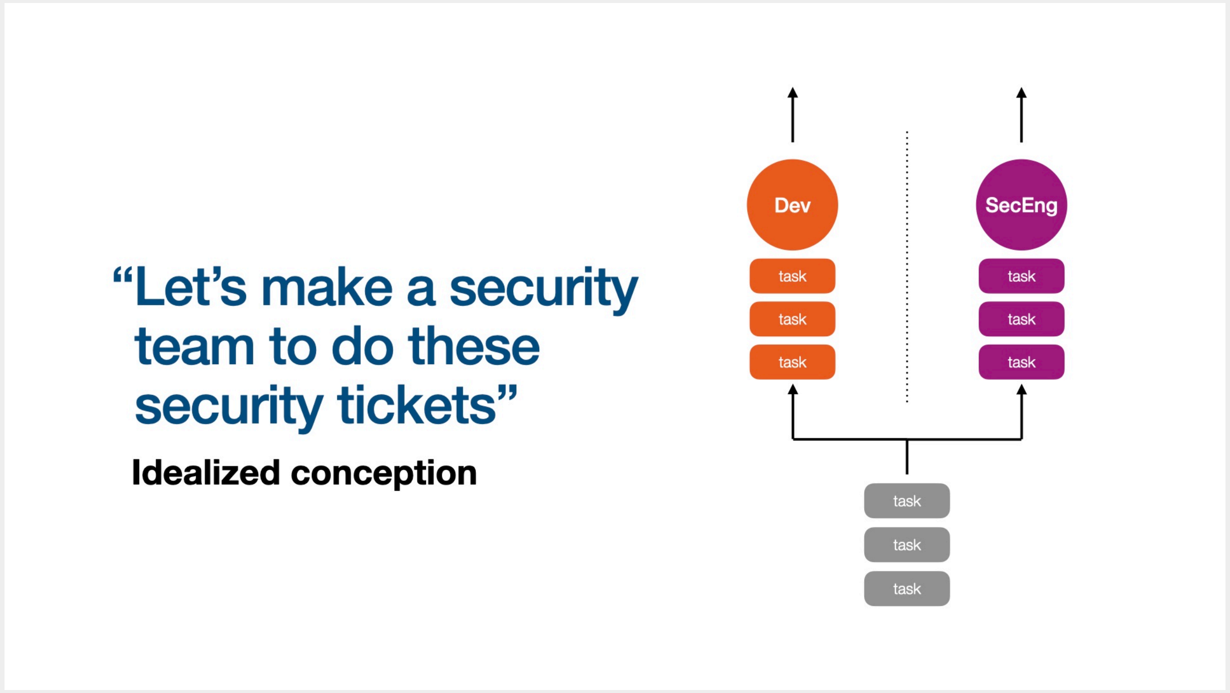


Pattern #4

Bad organization is a radioactive source of extra work

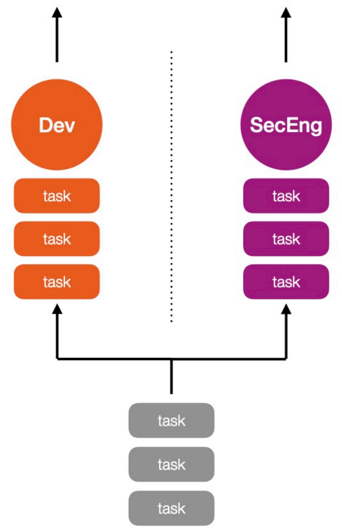
That kind of setup is one in which ticketing things can generate more work than you’d have if people had time to talk to each other.

That’s a specific case of a general issue. Poorly factored organizational boundaries create work.
#



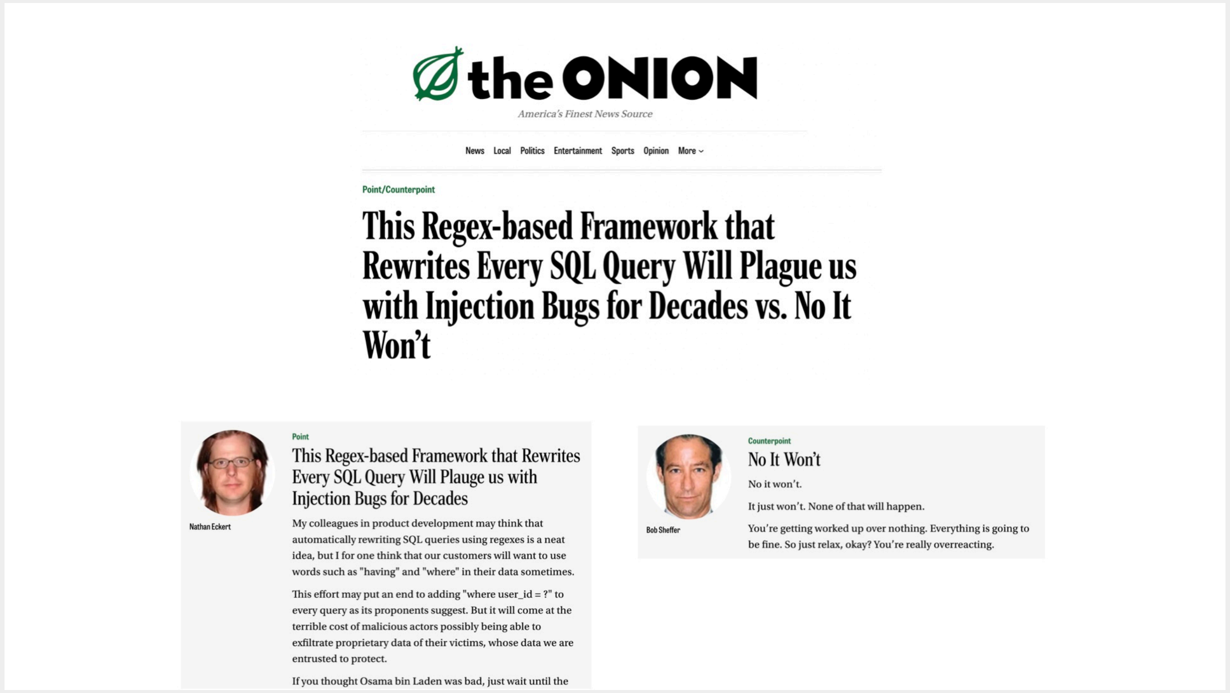
“Let’s make a security team to do these security tickets”

Idealized conception



Earlier I talked about a company that tried to use its security team to do all of the security remediation work.

By adding a queue for security tickets, 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. #


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

Point

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

Nathan Eckert

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

Counterpoint

No It Won't

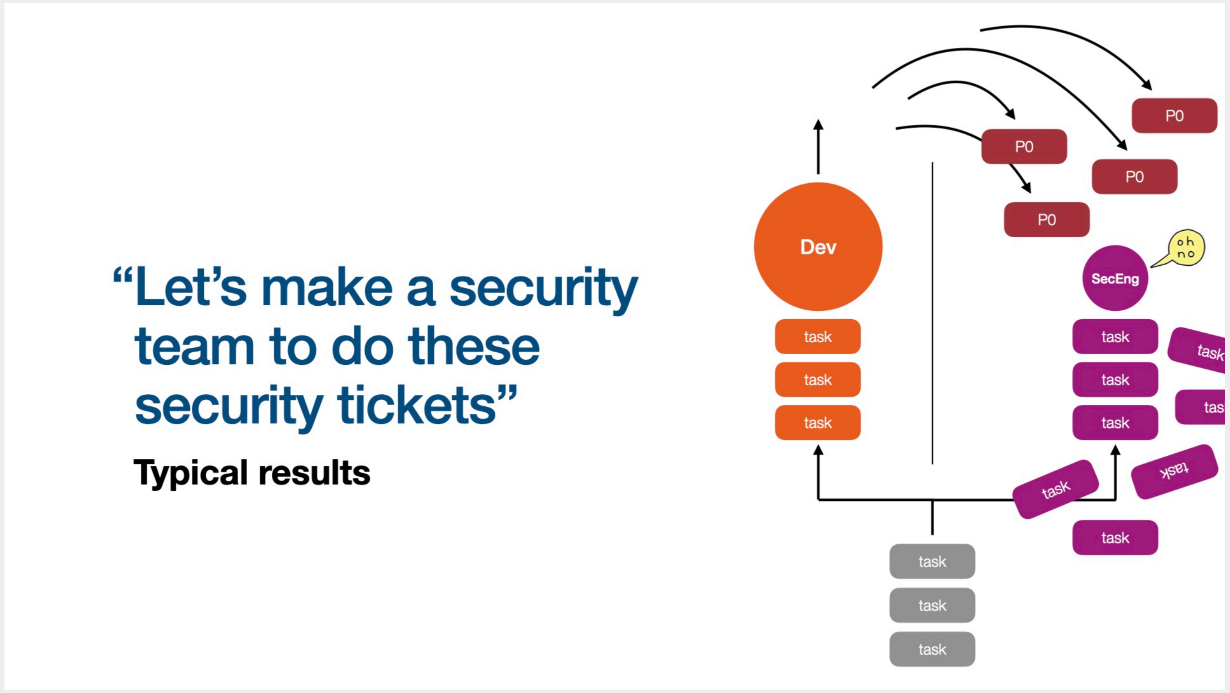
Bob Sheffer

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 with that is the horrible things development can get up to when it's not encouraged to worry about security.
#



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

Typical results

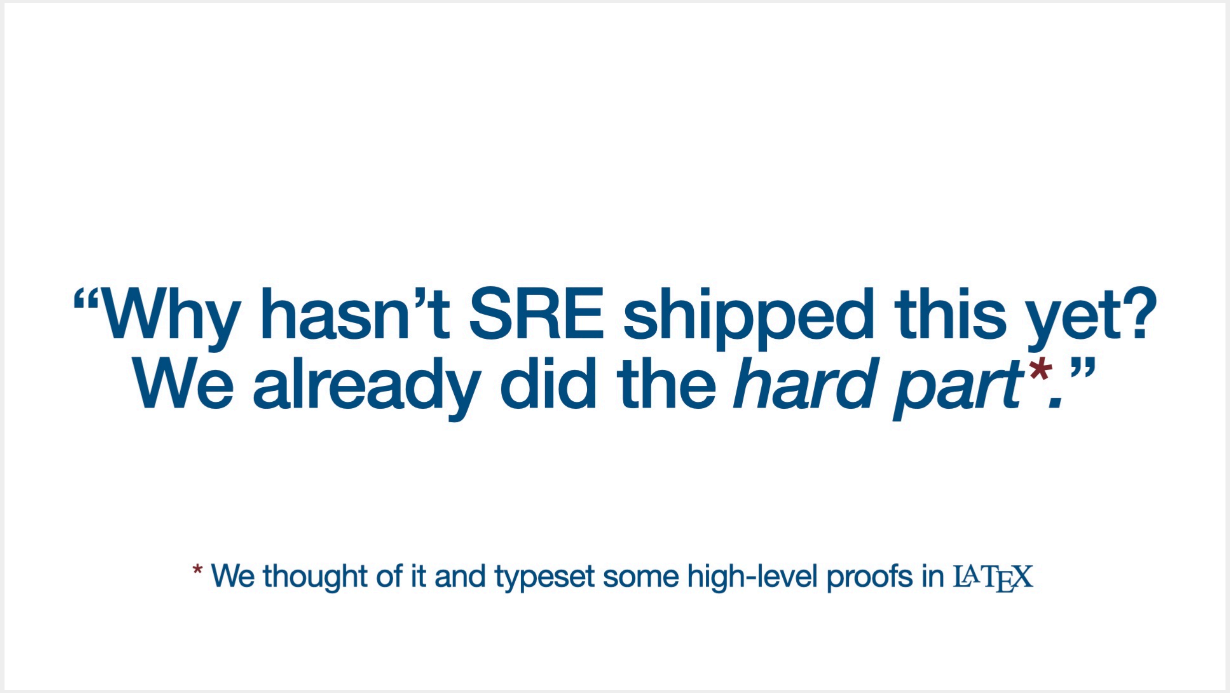
We can generate extra security work with our engineering activity if we erect a wall between those things.

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 #5

Humans have an impressive capacity for chauvinism

An undercurrent of all of this is how people conceive of themselves, and how they are to one another.
#

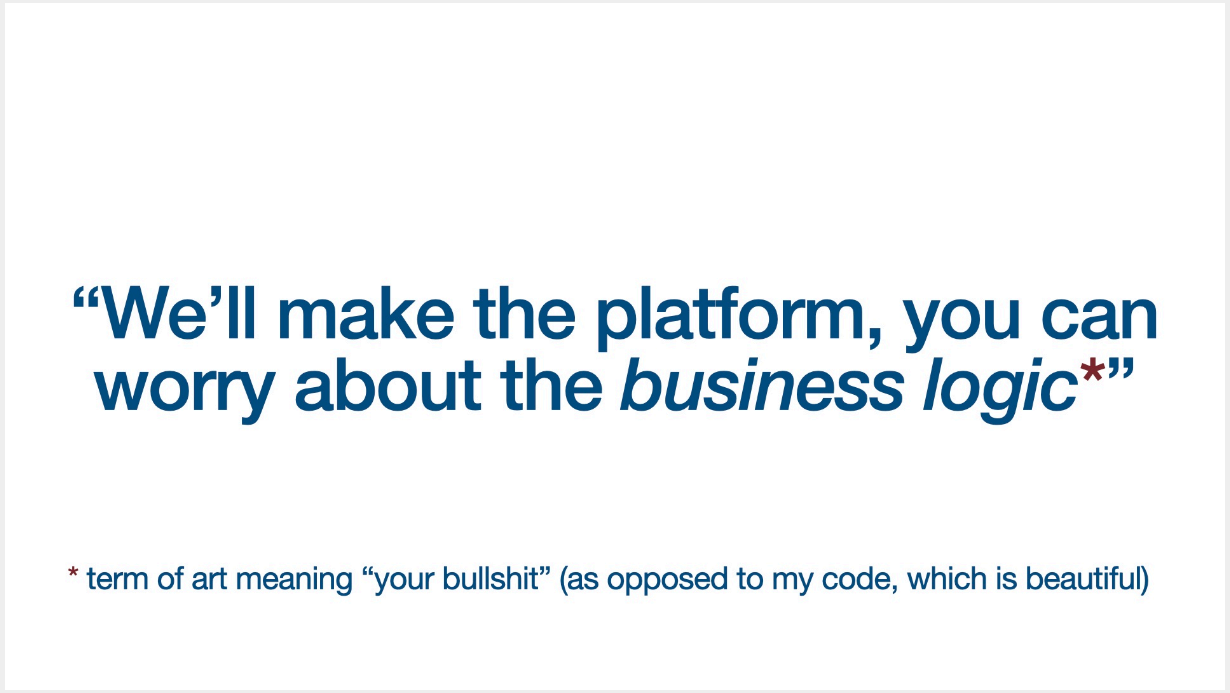


“Why hasn’t SRE shipped this yet? We already did the *hard part**.”

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

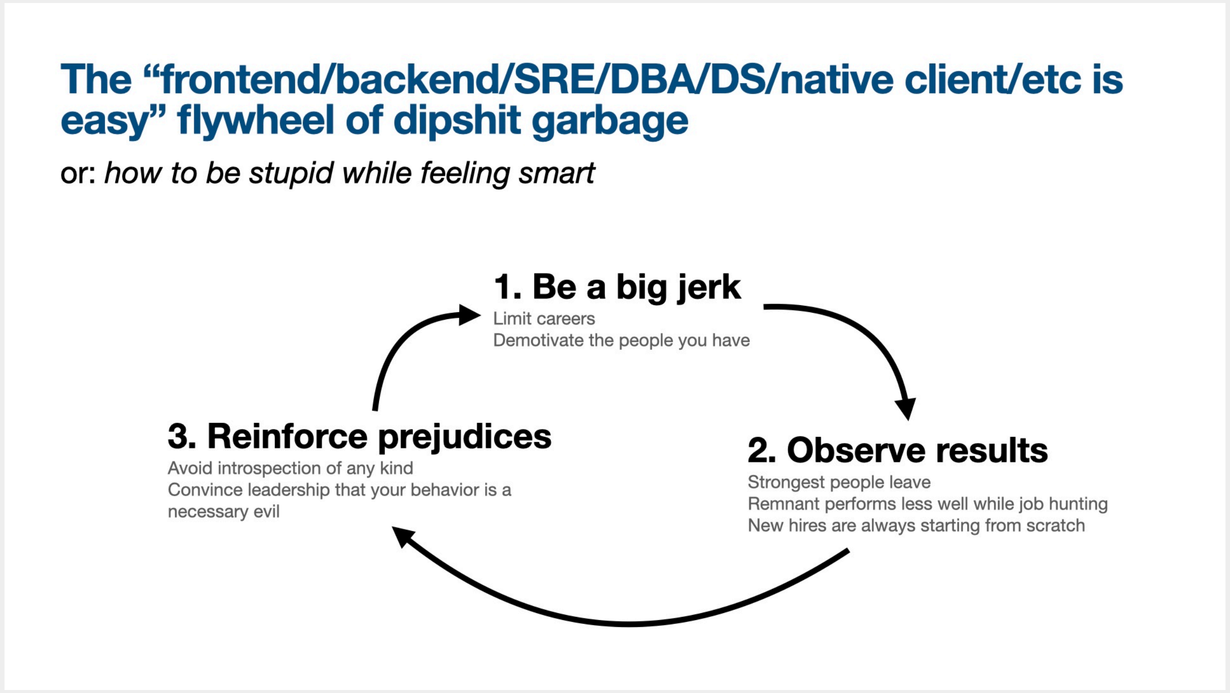
There can be a general tendency among some of us to imagine that other peoples’ jobs are easy compared to ours.

Obviously it’s not a majoritarian view that working on the server is so easy “it’s not clear it’s even technical,” but I can report that some crazies out there 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)

Those vibes can absolutely run in the other direction though, and thinking that product work is less technical might be even more common. Platform teams are full of folks who aren’t very worried about what the company they work for even does. #



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

or: *how to be stupid while feeling smart*

1. Be a big jerk
Limit careers
Demotivate the people you have

2. Observe results
Strongest people leave
Remnant performs less well while job hunting
New hires are always starting from scratch

3. Reinforce prejudices
Avoid introspection of any kind
Convince leadership that your behavior is a necessary evil

Although the results of these tendencies are very stupid, punting the offenders into the sun usually eludes us a solution. That’s because this behavior can be coded as “smart” and “valuable” to outside observers.

While a CEO may not endorse the behavior, they imagine that it gets results. Except it doesn’t. Driving other workers out reduces total throughput. Obviously.

The biggest problem with brilliant jerks is not necessarily even that they’re jerks. When you take a systemic perspective, they aren’t even brilliant! They’re dumb!
#

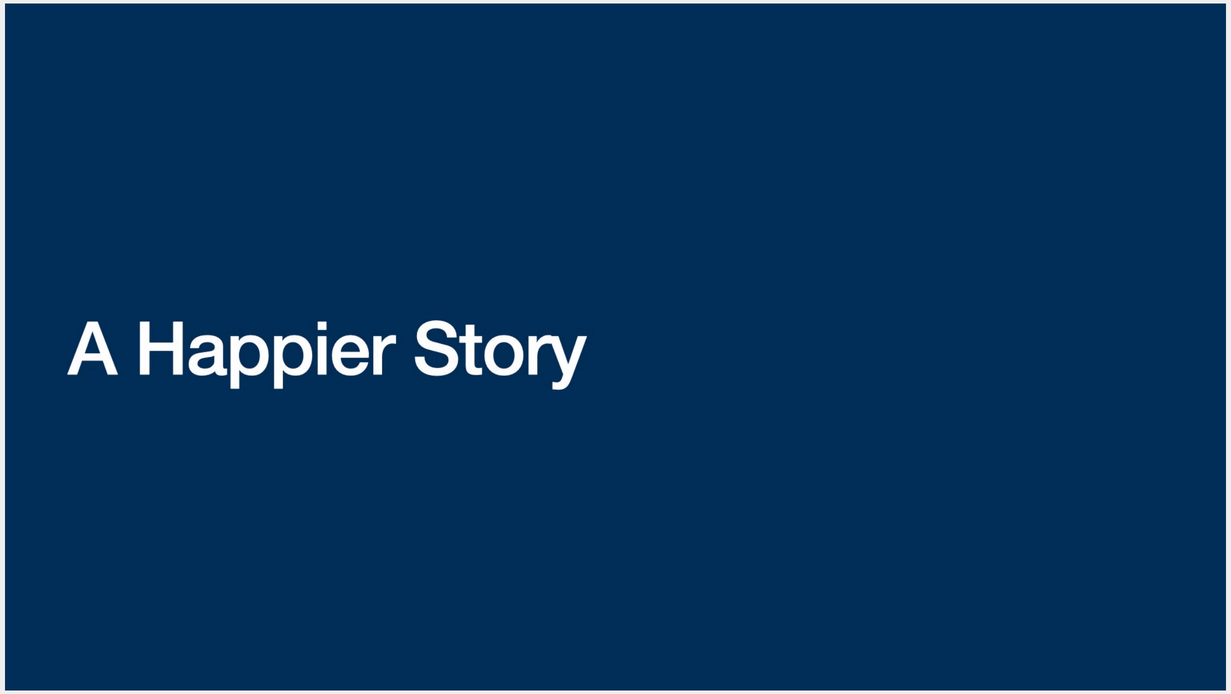
Parochialism & Ego

Hell is empty, these devils are here

Recurring themes throughout every chapter of this for me are parochialism and ego, the yin and yang of dysfunction. Both of these have positive and negative spins.

Parochialism might be deference, and not wanting to step on someone else’s toes. Or it might be lack of curiosity. Ego might be pride in your work, or it might be territoriality, or dismissiveness in the abilities of others.

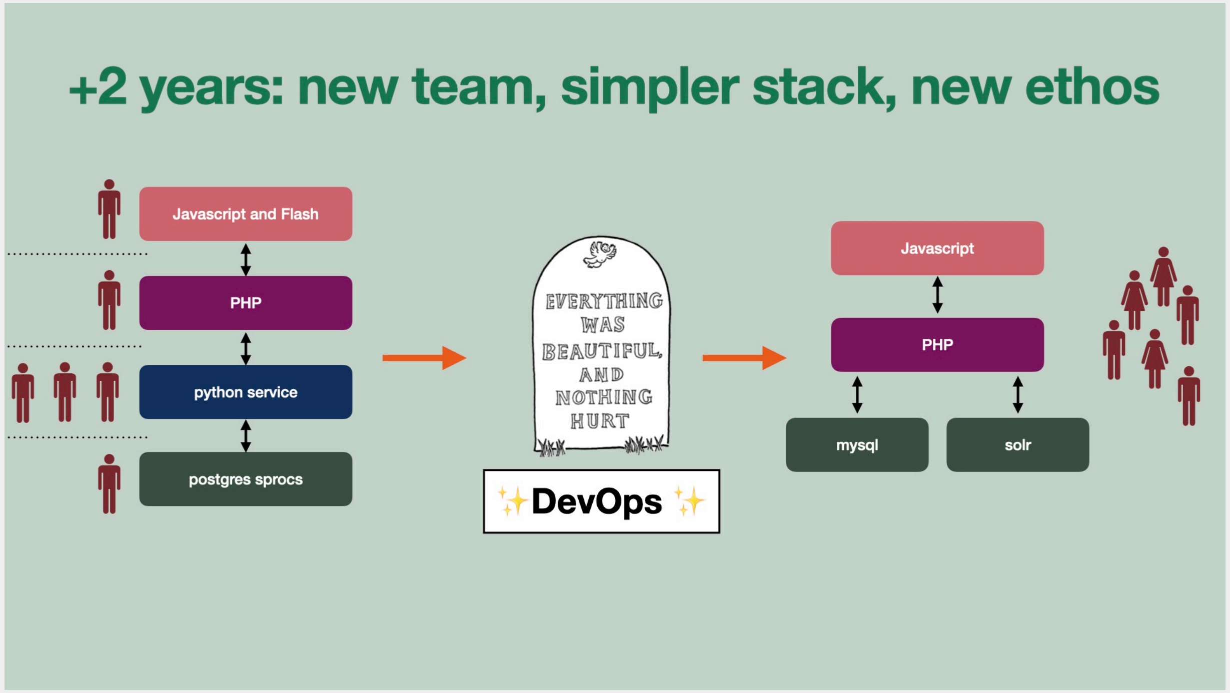
These are the primal urges that lead to the breakdowns in empathy that fill our lives with pain.
#



A Happier Story

Thank you for allowing me to trauma dump for a while. Like I said at the beginning of this talk, I’m motivated to make things better, not just complain.

I’ve spent a long time thinking about how the most functional engineering team I ever worked on got that way, and I want to give you a vignette from those times.
#



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

The startup I worked at fought its way from horizontally layered fiefdoms to a simpler architecture and a smaller team. We imported some leaders that took devops very seriously, as they had come up with the idea in the first place.

The most important bit that we took from that movement was that we should tear down existing barriers between roles. So after about two years of creative destruction pretty much everybody was pitching in on pretty much everything.
#

That approach was ultimately successful, and we found infrastructure stability and gained some ability to ship software again. We crawled out the other end of the scaling tunnel clean.
#

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.

That, combined with several CEO changes due to some other chaos, morphed the company into an engineering-led organization.

As an engineering team we found ourselves reaping an infrastructure dividend that gave us a bit of free time, and we had an unusual level of permission from the rest of the company to dictate how that time was used.
#

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

That’s a rare situation, and I think we knew it at the time. Ideas were thrown around about how we should use our extra capacity.
#

Proposition 1: Boil the Ocean

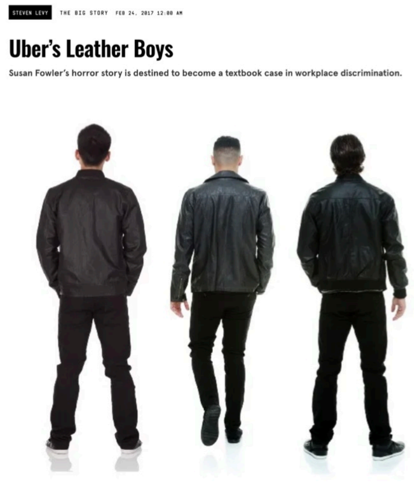
1. Feature freeze lmao
2. Embark on five-year rewrite into ~~Scala~~ ~~Golang~~ ~~Clojure~~ Swift.
3. *[Get fired early in year two and be legends.]*

I've been part of several other teams that found themselves in this situation and decided to rewrite various things they hated. I think this is the most common outcome.

But we had already tried this, so 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

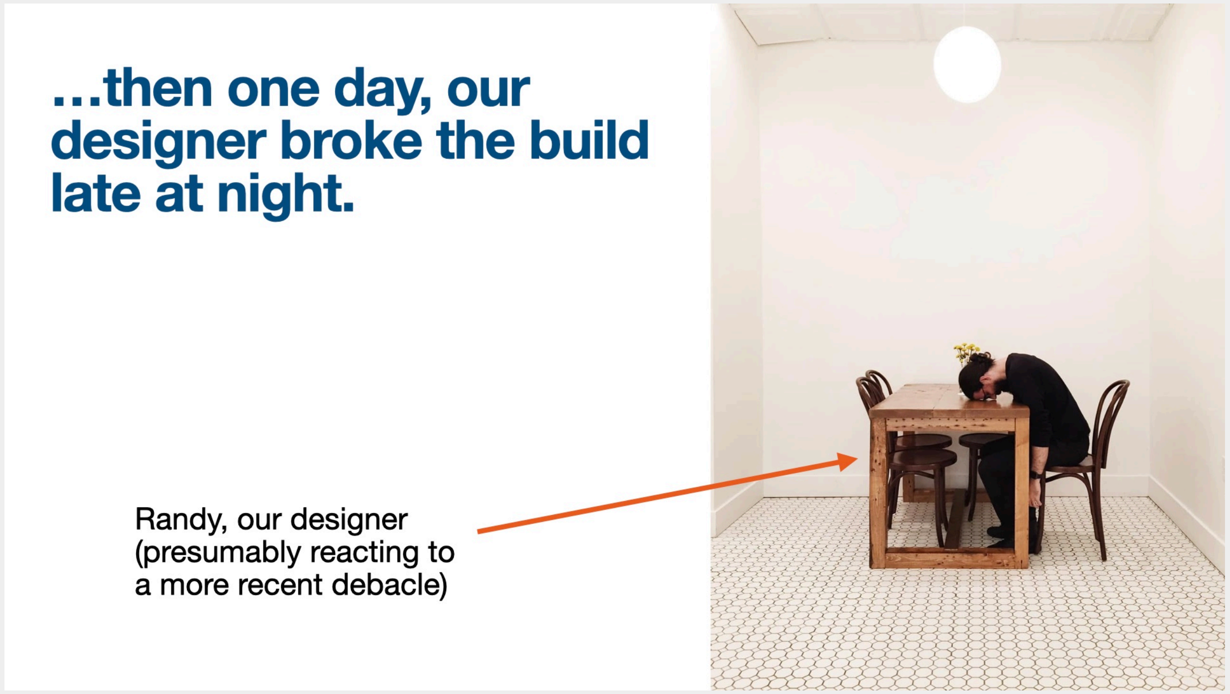
Uber's Leather Boys

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 and strut. We did a bit of that, but not as a primary strategy.
#

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

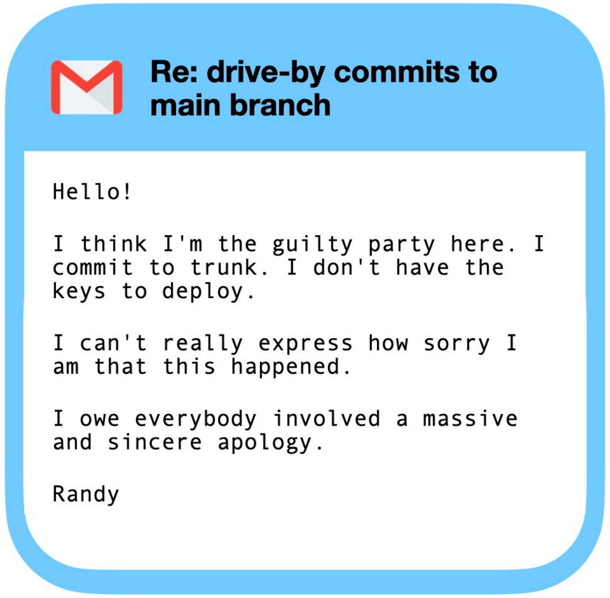
None of that felt right. It just wasn't us. So we didn't do anything drastic for a while.
#



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

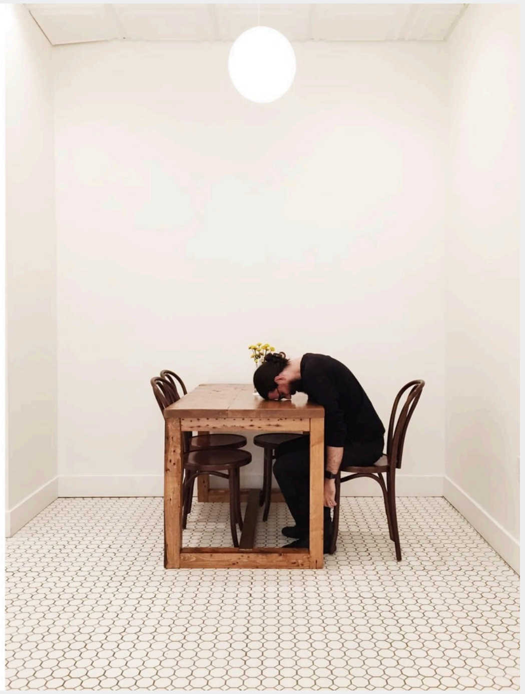
Randy, our designer (presumably reacting to a more recent debacle)

Then one day, our designer broke the build in the middle of the night. Everyone came in the next day and couldn’t work until they figured out what had happened.
#



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 really apologetic about the whole thing. An apology email thread ensued.

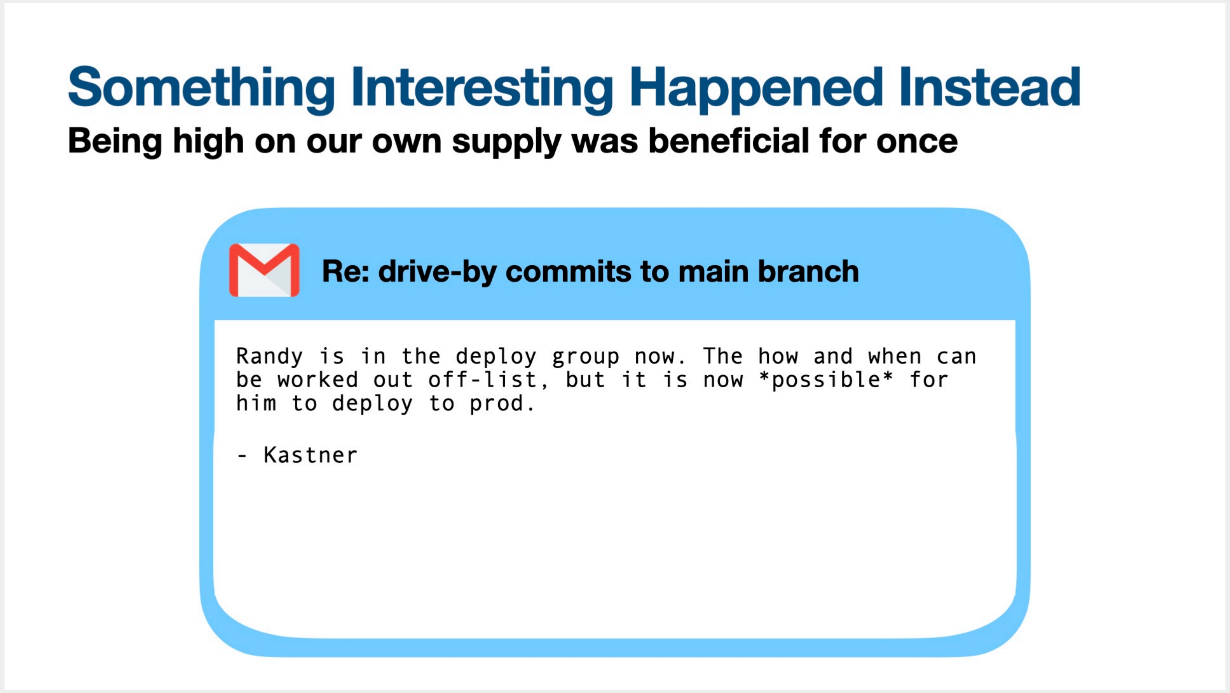
#

"An Engineer should review all of Randy's non-trivial code changes."
"The issue is that Randy doesn't run tests, and wouldn't know how to fix test failures."
"Randy should focus on design, and Engineers should focus on functionality and code quality."

The temptation was there to be huge jerks, and rebuild fiefdoms.

Perfectly reasonable people thought that maybe we should put more of a wall between designers and the css source code.

It was suggested that designers should just design, and coders should do all of the coding.
#



Something Interesting Happened Instead

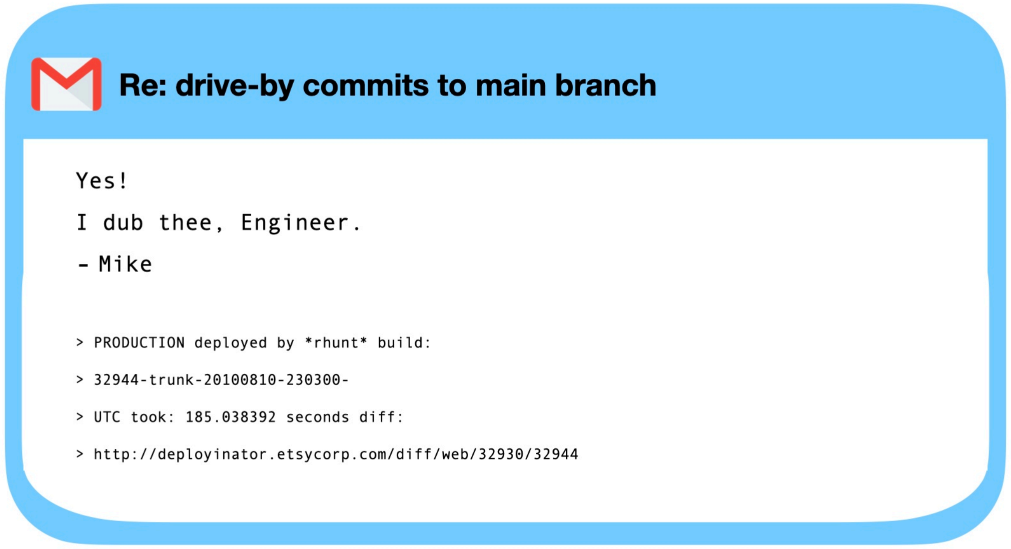
Being high on our own supply was beneficial for once

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 a miracle happened. There was an inspired person who decided to do the opposite. They just yolo'd giving the designer the deploy keys.

I've come to understand this event as being entirely contingent, but it was also very special.
#



Re: drive-by commits to main branch

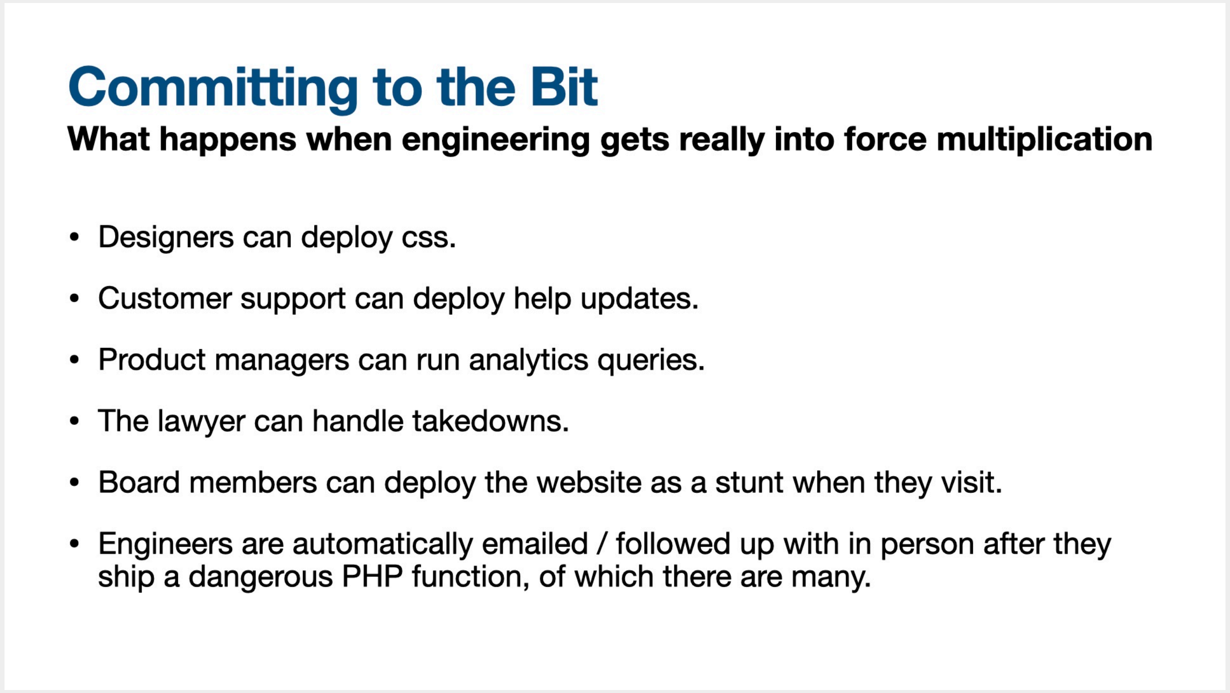
```
Yes!
I dub thee, Engineer.
- Mike

> 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.

We spent our free time building 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. Nothing bad happened.
#

Proposition 3: Empower Everybody Else

That was 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.

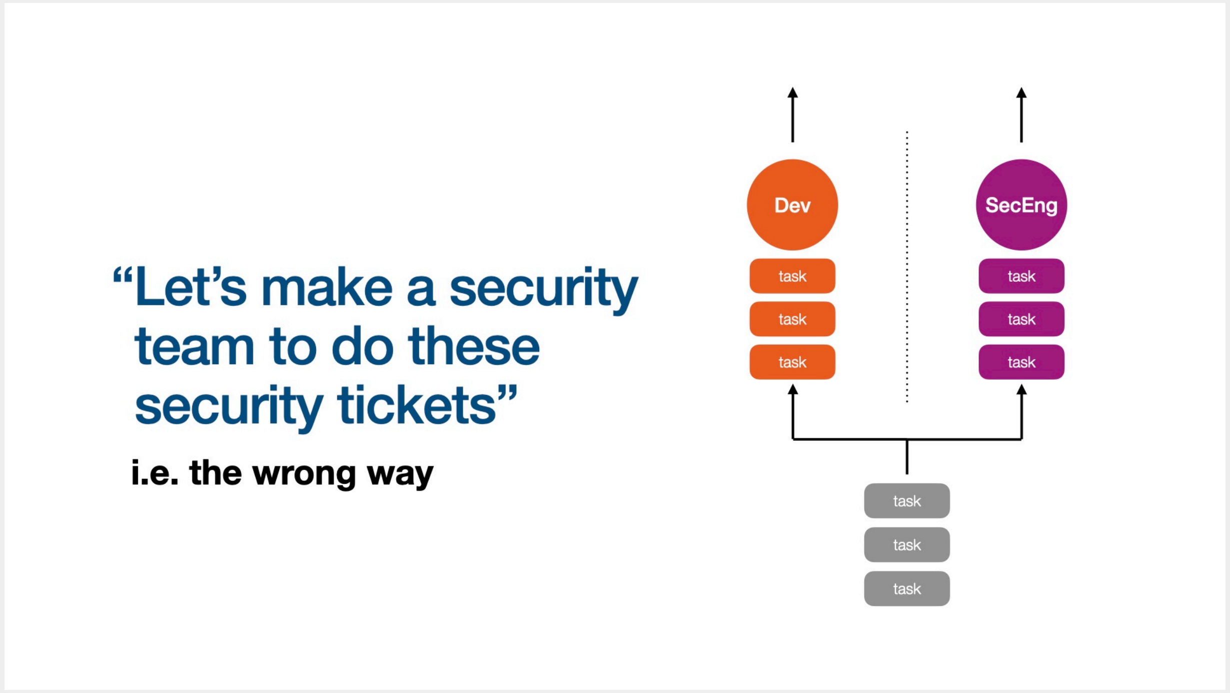
We did the thing we did with designers with tons of other groups, in technical and nontechnical disciplines.
#

What Executing Well Felt Like

That was the process of getting from a place of not executing to executing really well. I want to spend a bit of time talking about what good felt like.
#

Domain experts, not domain owners

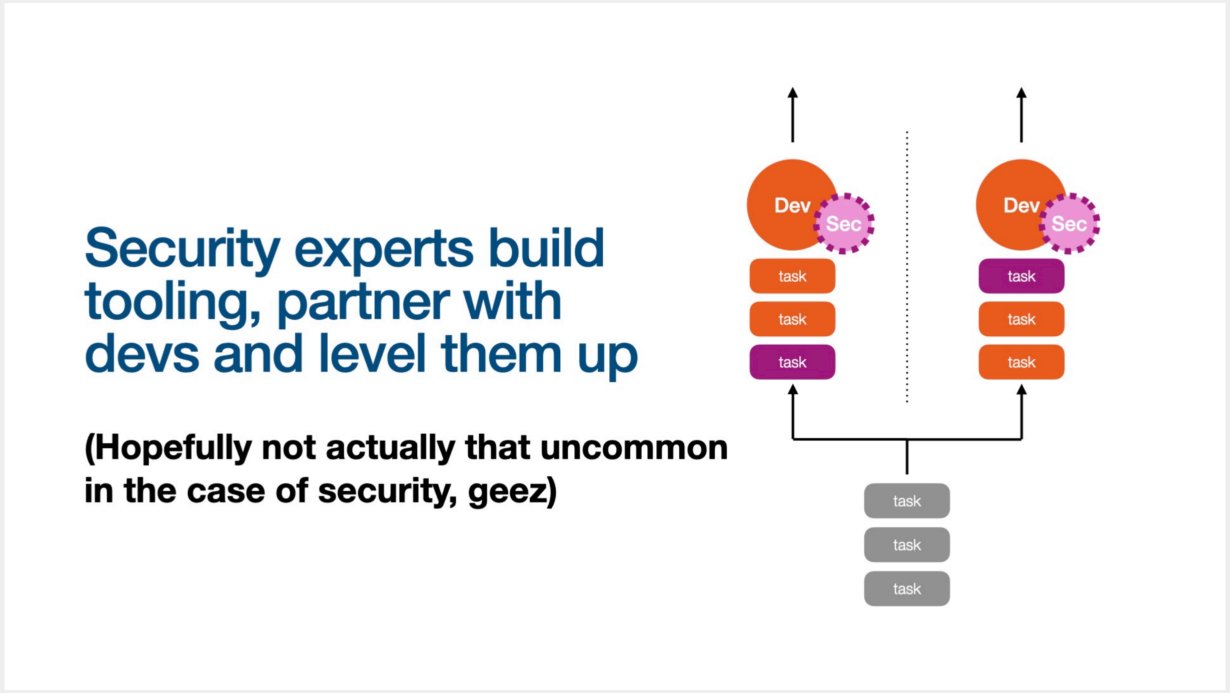
We tried really hard to have domain experts, but never really domain owners.
#



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

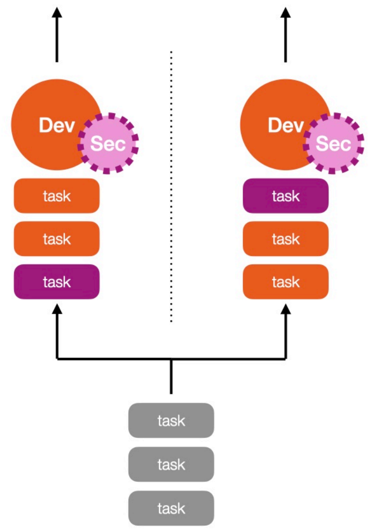
i.e. the wrong way

Let’s go back and consider the company I worked at where the security team was responsible for all of the security tickets, again.
#



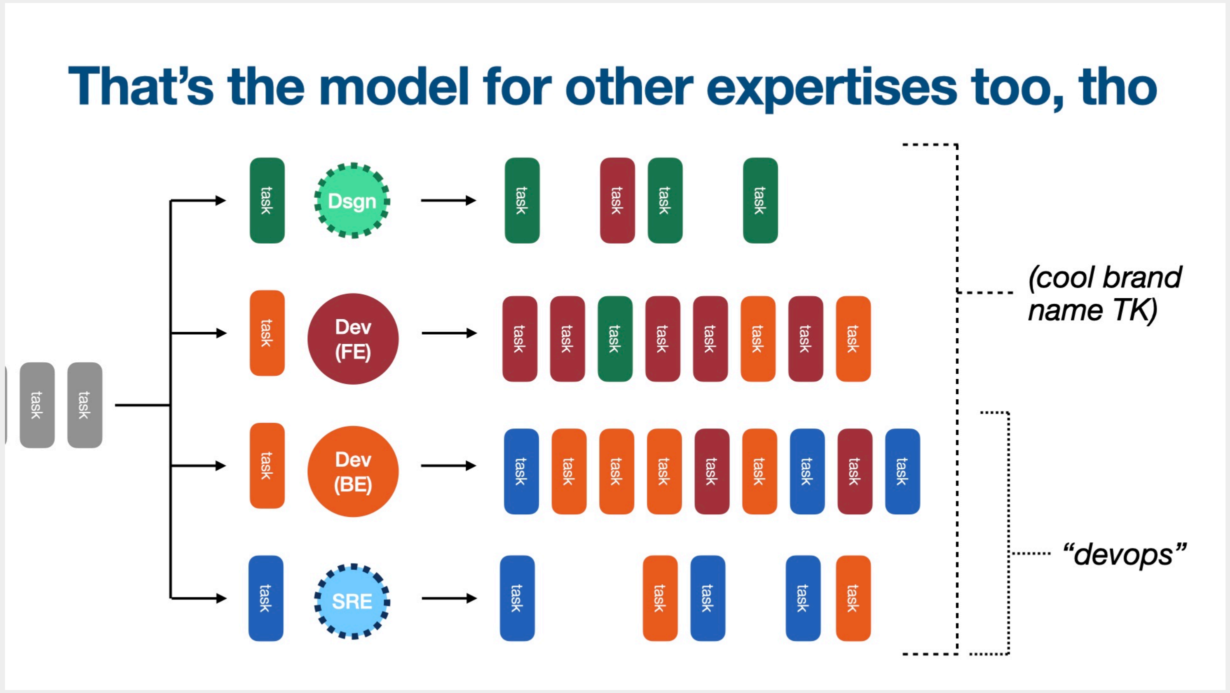
Security experts build tooling, partner with devs and level them up

(Hopefully not actually that uncommon in the case of security, geez)



The way we approached that when we were executing well was that everyone was responsible for security. The security team was the set of domain experts focused on leveling everyone up in that regard, and making sure we all got to a secure place.

(I think this is thankfully not that uncommon when it comes to security as a discipline.)
#



That's the model for other expertises too, tho

But it's a lot less common in my experience to buy into that model for other disciplines.

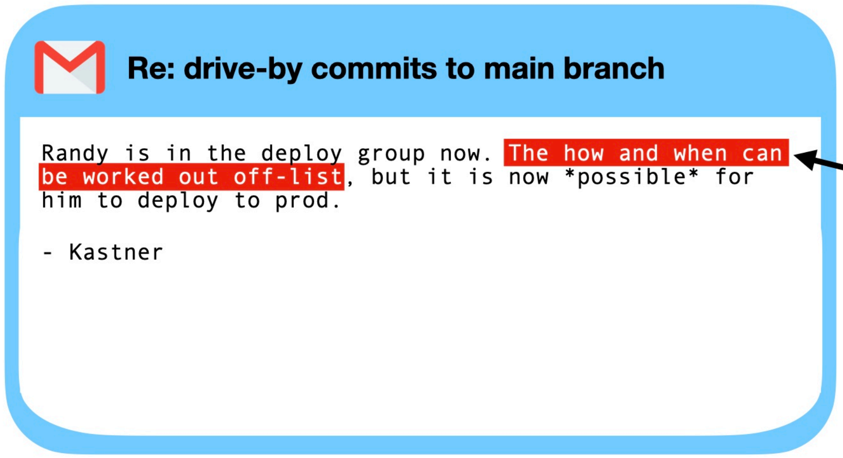
We had a word for the idea that SRE and developers should share tasks, it was "devops." But like I said nearly all teams I've worked on since have not actually done this.

And many teams I've worked on have also failed to extrapolate, and have had frontend or client engineers strictly siloed from backend engineers. They get up to building entire GraphQL monstrosities to avoid talking to each other.

You want someone that's an expert in frontend work, but I think you probably do not want people with "frontend" and "backend" in their title. #

Make your discipline more accessible

This should be part of your job!



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
```

(Lots of tooling work happened off camera)

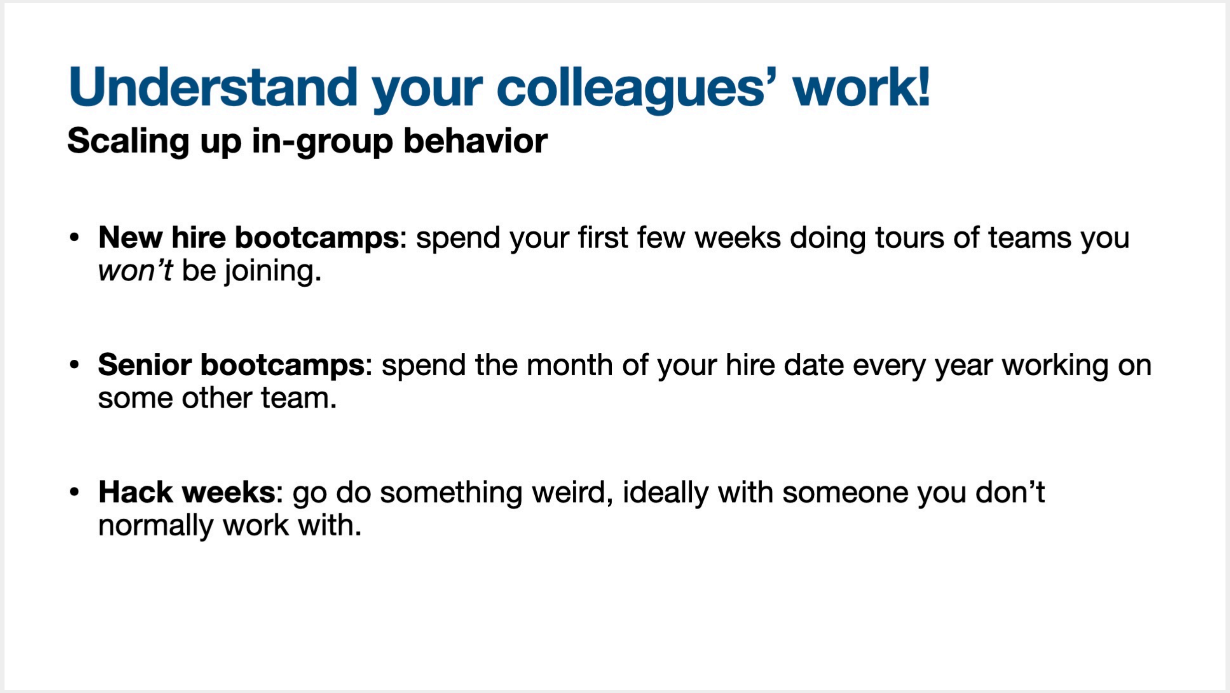
Domain experts, not domain owners.

An important thing that distinguishes a expert as distinct from an owner is that you encourage the expert to devote some of their time to helping others work in their discipline.
#

Proactive cross-pollination

That we needed to give people slack time like that to sustain the level of overall teamwork was a general principle.

There were other ways in which we didn’t just wait for the streams to cross themselves. We injected sustaining energy into this system very intentionally.
#



Understand your colleagues' work!

Scaling up in-group behavior

- **New hire bootcamps**: spend your first few weeks doing tours of teams you *won't* be joining.
- **Senior bootcamps**: spend the month of your hire date every year working on some other team.
- **Hack weeks**: go do something weird, ideally with someone you don't normally work with.

Human beings are limited by their in-group.

You feel psychological safety within your in group. You feel like you can experiment and learn new things within your in group. You can exercise curiosity about the people in your in-group and what they do.

So we invested time in making people's in-groups larger. Bootcamps were one of those ways: we forced people to work on other teams occasionally.

Hack weeks were another institution that achieved similar ends.
#

Intentional team values

A decade later, I can articulate what our organization’s principles were. I can explain the higher-level reasons we were doing bootcamps, or code reviews, or a million other things. This is a victory of being intentional and vocal about what our team values were.
#

We are a team that…

- **Digs ditches**. Nobody is too good for any task.
- **Returns shopping carts (even if nobody's watching)**. We leave things better than we found them.

It was important to us to articulate that elitism as an attitude was poison. To the contrary, we do windows here. If a ditch needs digging, our tech leads will line up to dig it.

We also needed to create the strong expectation that you'd leave things in a better state as you messed with other people's stuff. That made people overjoyed to have you messing with their stuff.
#

Parting Thoughts

Let me leave you with a few things.
#

We are living in the age of narcissistic personality disorder
(screw this dumbass)

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.

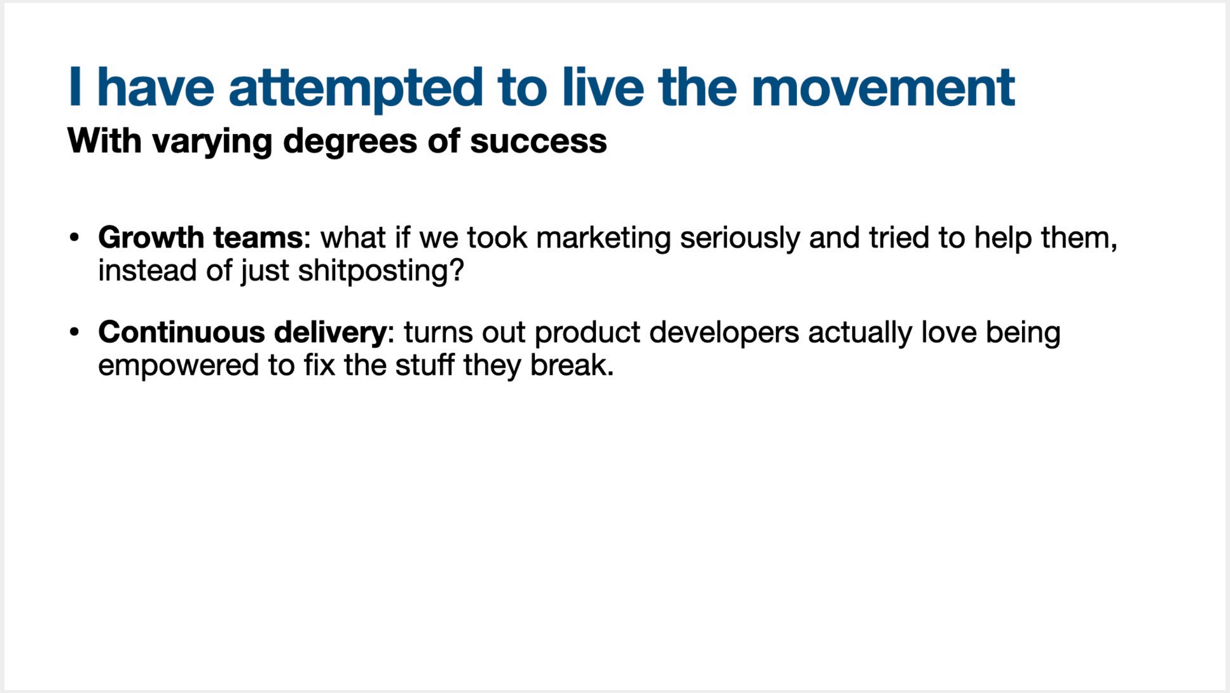
At a certain genetic equilibrium, NPD is evolutionarily adaptive, so these people are out there. But in tech we've grown a tendency to see it as useful rather than parasitic. The industry believes in assholes.

Valuing others around you should not be a radical act, but that's where things are unfortunately.
#

~~Parochialism & Ego~~

It shouldn’t be a radical act because results are better if we can cooperate. And life is better without cult leaders.

How do we tear down parochialism and ego?
#



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.

I've tried. I've got some notes about trying.

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, like it was the 90s. 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.
#

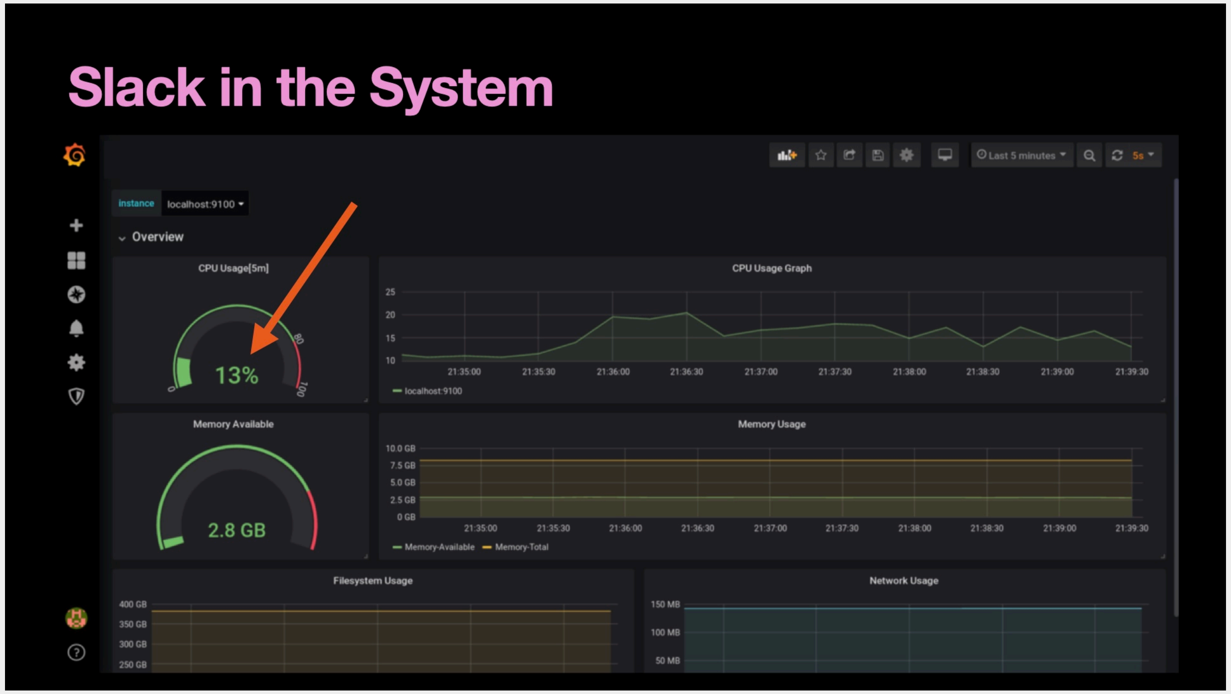
Permission

The missing ingredient in those situations was just permission. Mainly what I needed to do was give people the permission they needed to be curious, and to cooperate.

I don’t think you can grass roots that, beyond what I’m trying to do here by making the idea more popular.

It’s on leaders to value cooperation and to reward curiosity.
#

Slack in the System

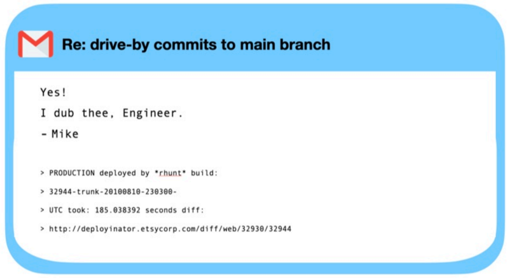


In addition to permission, there has to be slack in the system. This requires persistent commitment.
When you start skipping the bootcamps, they'll die out as a practice.

We don't let our machines run at 100% utilization. But again, we have a tendency to love our computers more than ourselves.
#

Leadership’s values trickle down

The choice is yours



Re: drive-by commits to main branch

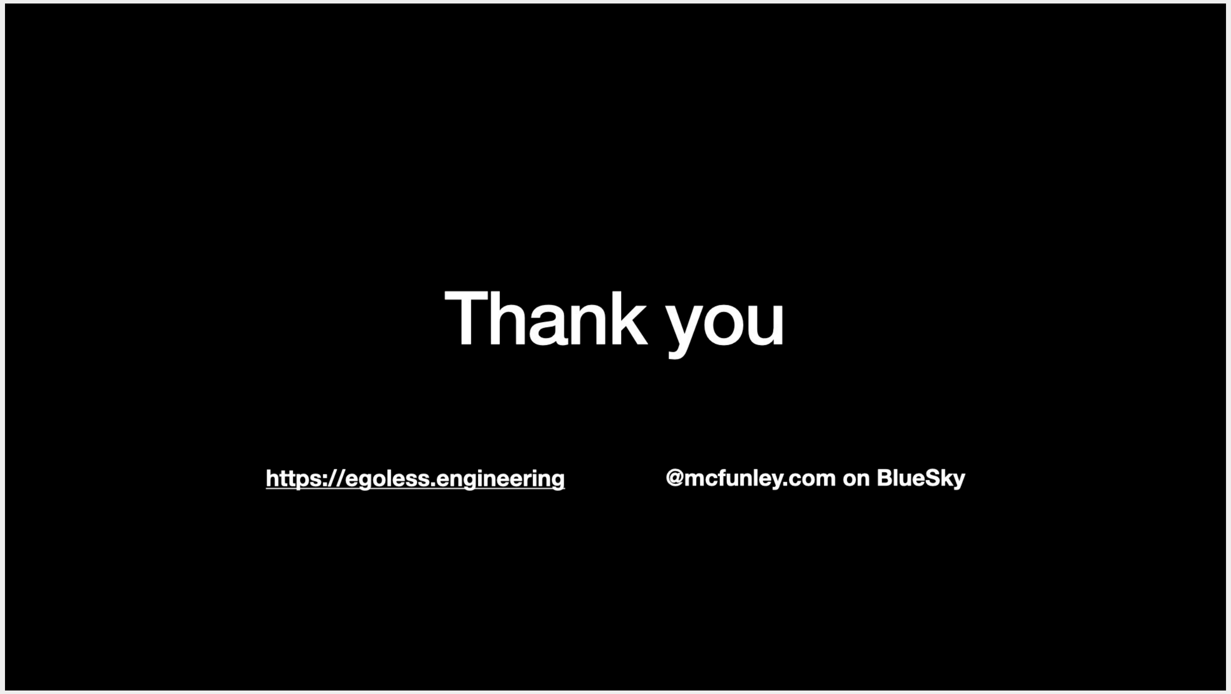
```
Yes!
I dub thee, Engineer.
- Mike

> PRODUCTION deployed by *rhunt* build:
> 32944-trunk-20100810-230300-
> UTC took: 185.038392 seconds diff:
> http://deployinator.etsycorp.com/diff/web/32930/32944
```

The things we value and reward as leaders trickle down. Leaders can reward our worst or our best impulses.

Most CEO’s I’ve worked for have tried to kill feel-good programs like bootcamps and hack weeks, but I’ve never worked for a CEO that tried to end feel-bad programs like mandatory code review.

None of them would admit this, but I think there’s an industry instinct that misery gets results. I think this is mistaken. Misery is a shitty proxy metric for results.

#



Thank you

https://egoless.engineering

@mcfunley.com on BlueSky

#

@mcfunley.com

Thanks to many folks who have inspired me over the years but particularly Coda Hale, Kellan Elliot-McCrea, Camille Fournier, and Marc Hedlund.

CHECK OUT MY OTHER CLUBS

Generated with keynote-export.