The Paradox of Alerts

Why deleting 90% of your paging alerts can make your systems better, and how to craft an on call rotation that engineers are happy to join.

@mipsytipsy

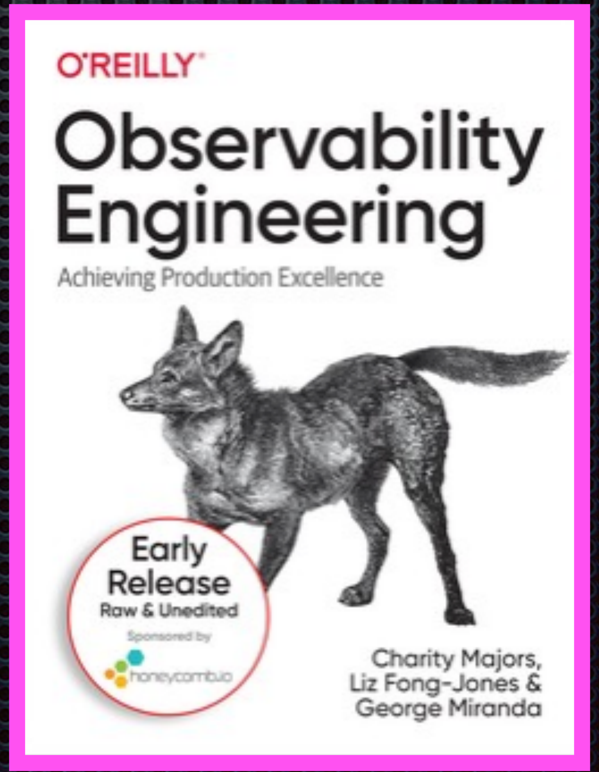
honeycomb.io

@mipsytipsy

engineer/cofounder/CTO

https://charity.wtf

new!



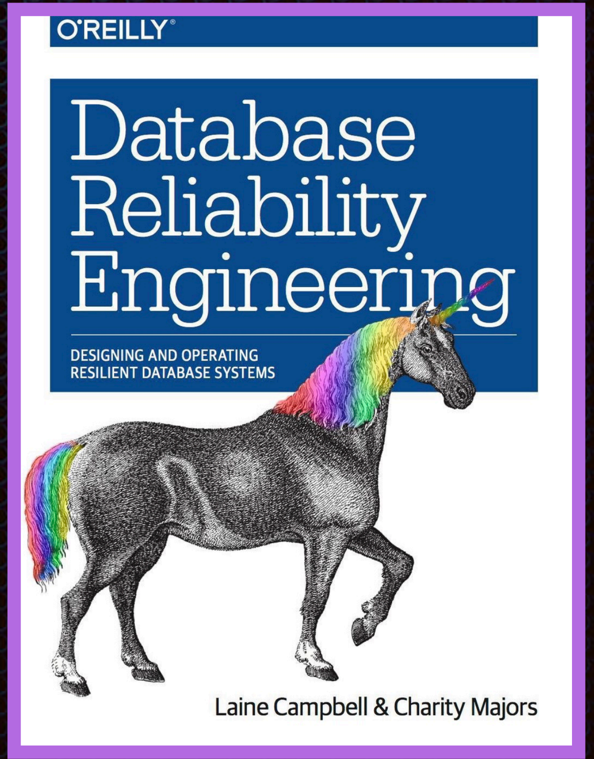
O'REILLY

Observability Engineering

Achieving Production Excellence

Early Release
Raw & Unedited

Charity Majors, Liz Fong-Jones & George Miranda



O'REILLY

Database Reliability Engineering

DESIGNING AND OPERATING RESILIENT DATABASE SYSTEMS

Laine Campbell & Charity Majors

honeycomb.io

“I dread being on call.”

“I became a software engineer with the expectation that other people would be getting paged, not me.”

“I didn’t sign up for this.”

“My time is too valuable to be on call. You want me writing features and delivering user value, not firefighting.”

“On call duty is what burned me out of tech.”

“If you make everybody be on call, we’ll have even fewer mothers (and other marginalized folks) in engineering.”

"Sometimes you just have to buy the happiness of your users with the lifeblood of your engineers."

"I sacrificed MY health and sleep for 10 years of on call duty; now it's YOUR turn."

"You aren't a REAL engineer until you've debugged this live at 3 am."

There are loads of toxic patterns around on call

(posturing, sunk cost fallacies, disrespect for sleep and personal lives, surface fixes, evading responsibility, flappy alerts, over-alerting, lack of training or support, snobbery…)

But it doesn't have to be this way.

We can do so much better. 🥰

I am here to convince you that on call can be:

- Compatible with full adult lives & responsibilities
- Rarely sleep-disturbing or life-impacting
- The sharpest tool in your toolbox for creating alignment
- Something engineers actually enjoy
- Even … ✨volunteer-only✨

On call is a lot of things:

An essential part of software ownership

A social contract between engineers and managers

A set of expert practices and techniques in its own right

A proxy metric for how well your team is performing, how functional your system is, and how happy your users are

A miserable, torturous hazing ritual inflicted on those too junior to opt out ?

😬 😬 😬

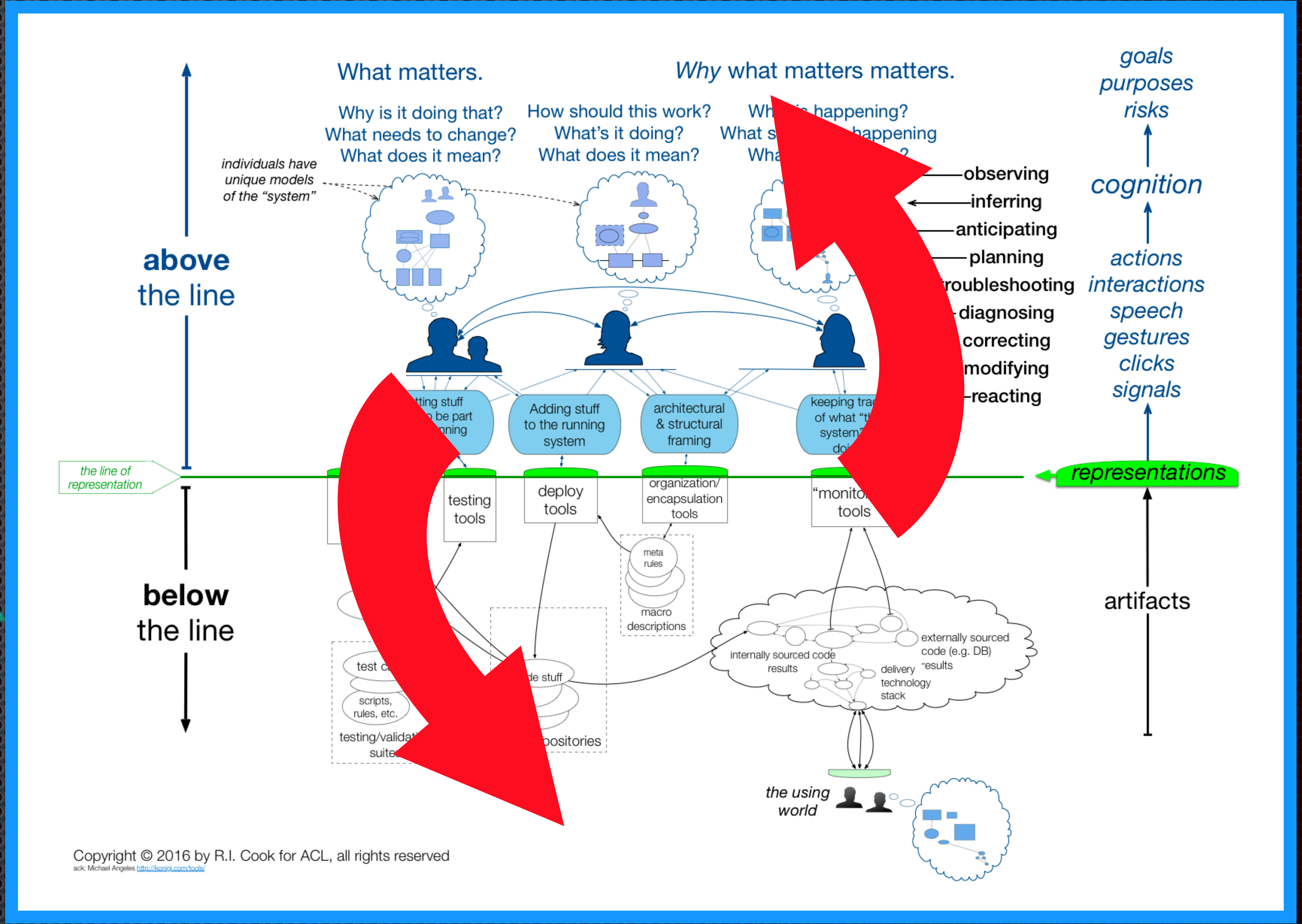
Software is a sociotechnical system

powered by at least two really big feedback loops

Software Ownership

and

Deploys (CI/CD)



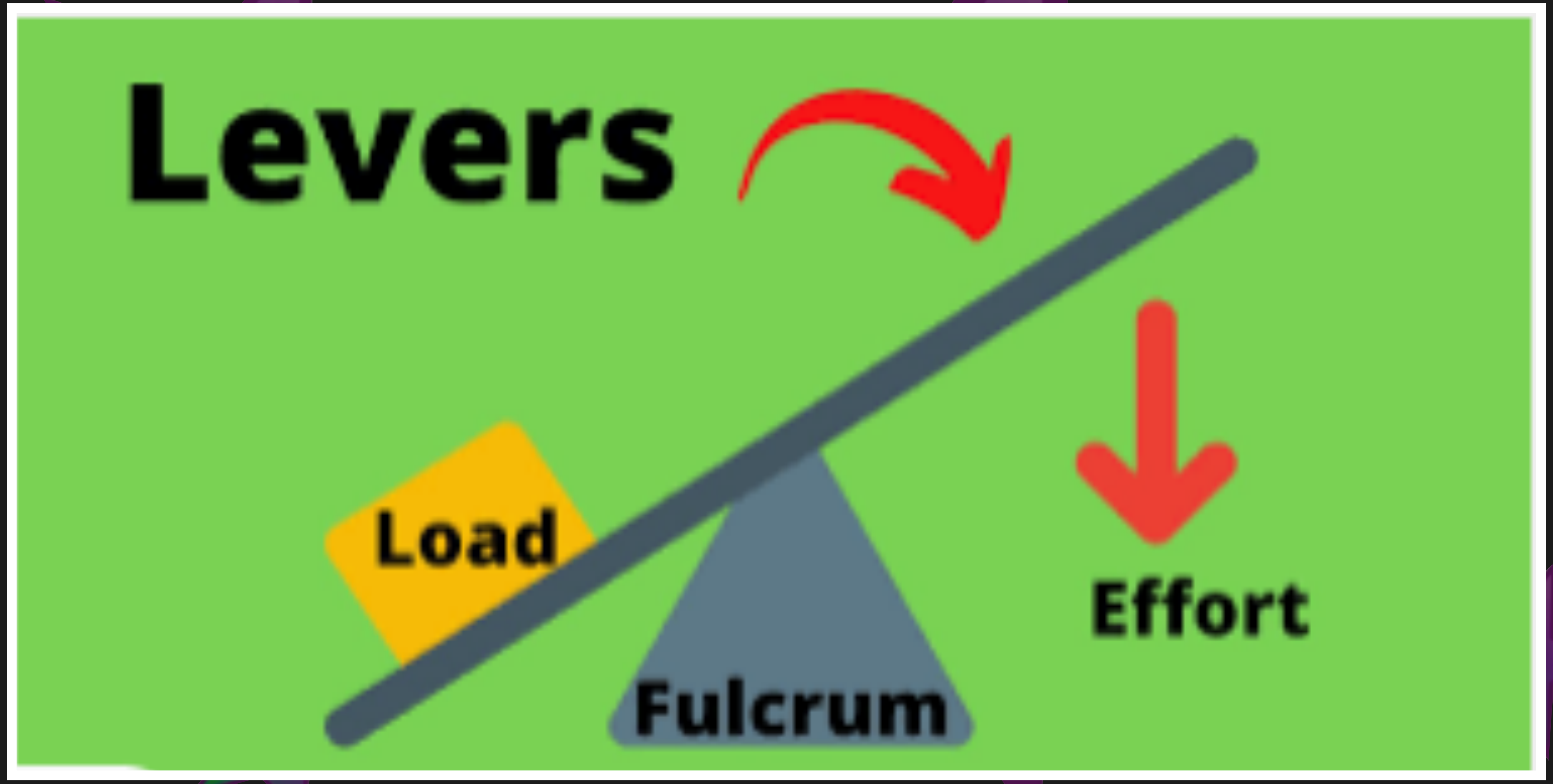
sociotechnical (n)

If you care about high-performing teams, these are the most powerful levers you have.

Software Ownership

and

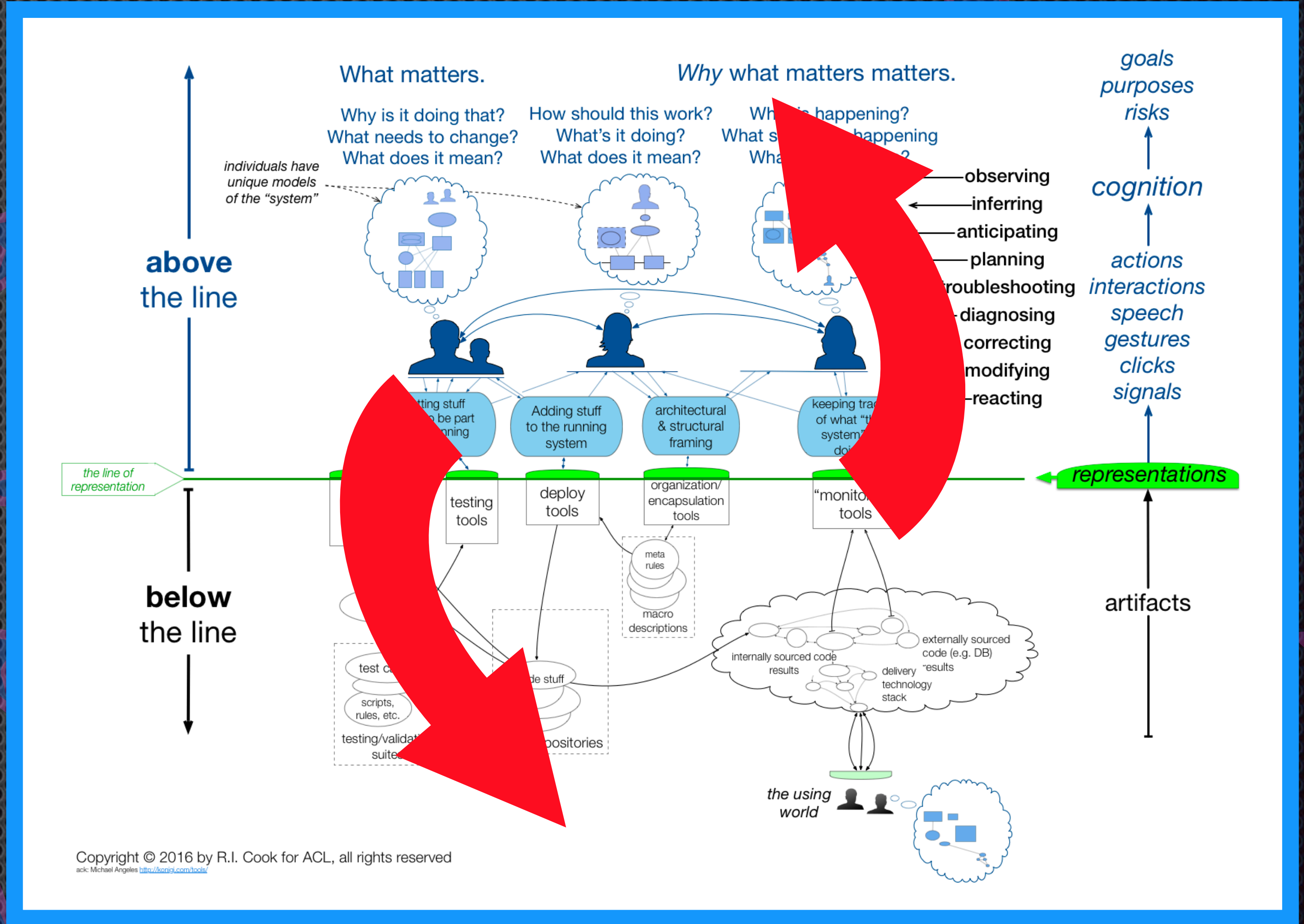
Deploys (CI/CD)



Software Ownership

Putting engineers on call for their own code in production

is how you close the feedback loop

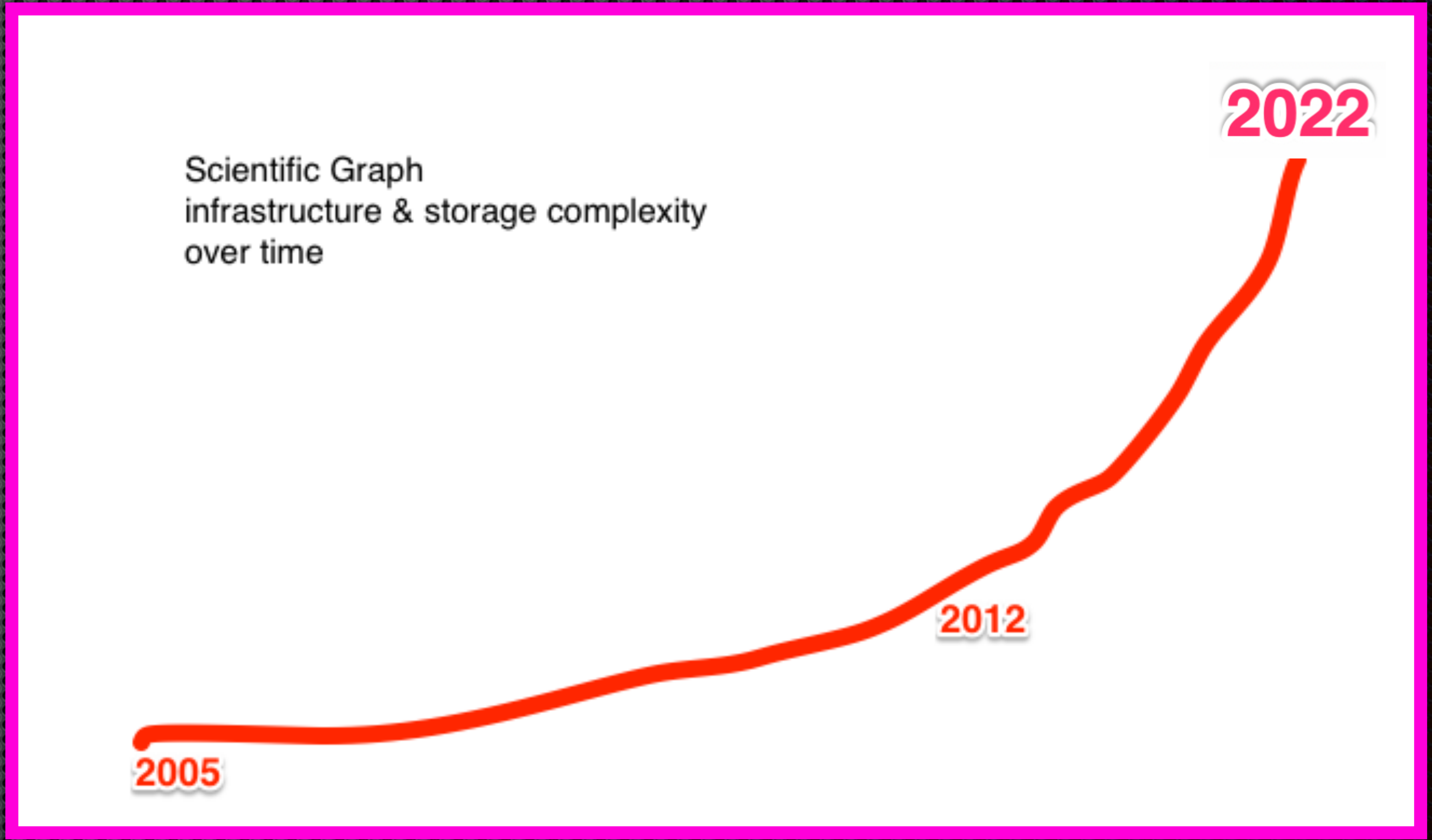


sociotechnical (n)

Software Ownership

is becoming mandatory as complexity has skyrocketed

ephemeral and dynamic,
far-flung and loosely coupled,
partitioned, sharded,
distributed and replicated,
containers, schedulers,
service registries,
polyglot persistence strategies,
autoscaled, redundant failovers,
emergent behaviors,
etc etc etc



Complexity🔥

Who should be on call for their code?

~~Ops teams~~ Any engineers who have code in production.

Is this payback?? 🤔

No!! Yes, ops has always had a streak of masochism.
But this isn't about making software engineers miserable too.
Software ownership is the only way to make things better.
For everyone.

It is possible to have…

an unhealthy system with an easy on call rotation

or a healthy system with a rough on call rotation

What you WANT is to align engineering pain with user pain.

and then you want to track that pain and pay it down.

On call responsibility is a social contract between engineers and managers.

Engineers, your job is to own your code.

Managers, your job is to make sure owning it doesn't suck.

If your manager doesn't take this seriously,

they don't deserve your labor.

Quit your job.

The goal is not, “never get paged.”

Disruptions are part of the job definition. We *can’t* get rid of all the outages and false alarms, and that isn’t the right goal.

It’s way more stressful to get a false alarm for a component you have no idea how to operate, than to comfortably handle a real alarm for something you’re skilled at.

Our targets should be about things we can do because they improve our operational health.

Any engineer who works on a highly-available service should be willing to be woken up a few times/year for their code.

If you're on an 8 person rotation, that's one week every two months

If you get woken up one time every other on call shift,
that's 3x/year and only once every 4 months

This is achievable.

By just about everyone.

It's not even that hard. You just have to care, and do the work.

not-so-thinly veiled engineering classism 🙄

Objections:

"My time is too valuable" Whose isn't? (It will take *you* the *least* time)

"I don't know how to navigate production" Learn! (It's good for you!)

"I have a new baby" Ok fine. Nobody should have **two** alarms.

"We have a follow-the-sun rotation" Lucky you! (ownership still matters)

"I need my sleep / it's stressful." Yeah, it is. (This is how we **fix it**.)

"I just don't want to." There are lots of other kinds of software.

Go work on one.

Let’s make on call fun again!

1. Align engineering pain with user pain, by adopting alerting best practices

2. Track that pain and pay it down

3. Profit!!!

1

Align engineering pain with user pain, by adopting alerting best practices

THAT WHICH DOES NOT KILL US
MAKES US FRAGILE AND FULL OF ONE-OFFS

No Alerting on Symptoms

Delete any paging alerts for symptoms (like "high CPU" or "disk fail"). Replace them with SLOs.

Give up the dream of "predictive alerting." Alert only when users are in pain.

Code should fail fast and hard; architecture should support partial, graceful degradation.

Service-Level Objectives

Moving from symptom-based alerting to SLOs often drops the number of alerts by over 90%.

Alert only on SLO violations and end-to-end checks which correlate directly to real user pain.

Better to spend down an SLO budget than suffer a full outage.

No Flaps

Delete any flappy alerts, with extreme prejudice.

I mean it. No flaps.
They're worse for morale than actual incidents.

Lane Two

Nearly all alerts should go to a non-paging alert queue, for on call to sweep through and resolve first thing in the am, last thing at night. Stuff that needs attention, but not at 2 am.

Prune these too! If it's not actionable, axe it.

You may need to spend some months investing in moving alerts from Lane 1 to Lane 2 by adding resiliency and chaos experiments.

No more than two lanes. There can be only two.

Out of hours alerts

Set a very high bar for what you are going to wake people up for. Actively curate a small list of rock solid e2e checks and SLO burn alerts.

Take every alert as seriously as a heart attack. Track them, graph them, **FIX THEM**.

Paging alerts

Should all come from a single source.
More is messy.

Each paging alert should have a link to documentation describing the check, how it works, and some starter links to debug it. (And there should only be a few!)

Should be tracked and graphed. Especially out-of-hours.

Alerts are *not* all created equal

You have a single source of truth,
or multiple sources of lies.

Better to have twenty daytime paging alerts than one alert paging at 3 a.m.

Better to have fifty or a hundred "lane 2" alerts than one going off at 3 a.m.

Training People

Invest in quality onboarding and training for new people. (We have each new person draw the infra diagram for the next new person ☺️)

Let them shadow someone experienced before going it alone. Give them a buddy.

It's way more stressful to get paged for something you don't know than for something you do. Encourage escalation.

Retro and Resolve

Teach everyone how to hold safe retros, and have them sit in on good ones – safety is learned by absorption and imitation, not lectures.

If anything alerts out of hours, hold a retro. Can this be auto remediated? Can it be diverted to lane two? What needs to be done to fix it for good?

Consider using something like jeli.io to get better over time.

Human SLOs

Nobody should ever have to be on call the night after a bad on call night.

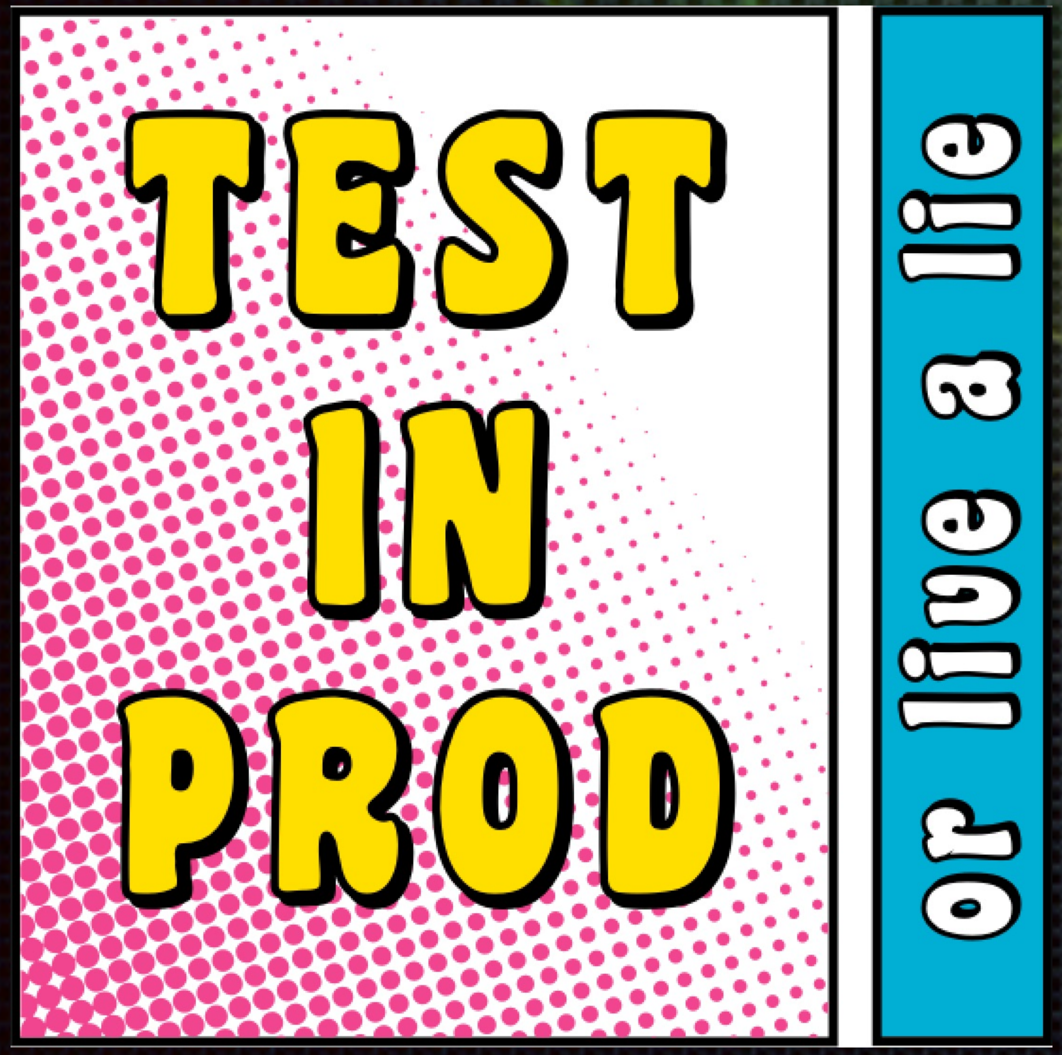
Nobody should feel like they have to ask permission before sleeping in after a rough on call night

If the rate of change exceeds the team's human SLOs, calm the fuck down.

Link: https://www.honeycomb.io/blog/kafka-migration-lessons-learned/

Managers

Managers are bad in the critical path, but it's very good for them to stay in the technical path. On call is great for this.



The ideal solution is for managers to pinch hit and substitute generously.

1

Align engineering pain with user pain, by adopting alerting best practices

2

Track that pain and pay it down

Observability

OBSERVABILITY

EMBRACE THE POWER OF UNKNOWN-UNKNOWNS

Invest in observability. It's not the same thing as monitoring, and you probably don't have it.

To have observability, your tooling must support high-cardinality, high-dimensionality, and explorability.

Links:

https://www.honeycomb.io/blog/observability-101-terminology-and-concepts/

https://www.honeycomb.io/blog/observability-5-year-retrospective/

https://www.honeycomb.io/blog/so-you-want-to-build-an-observability-tool/

Observable Code

MEH-TRICS

Most of us are better at writing debuggable code than observable code, but in a cloud-native world, observability *is* debuggability.

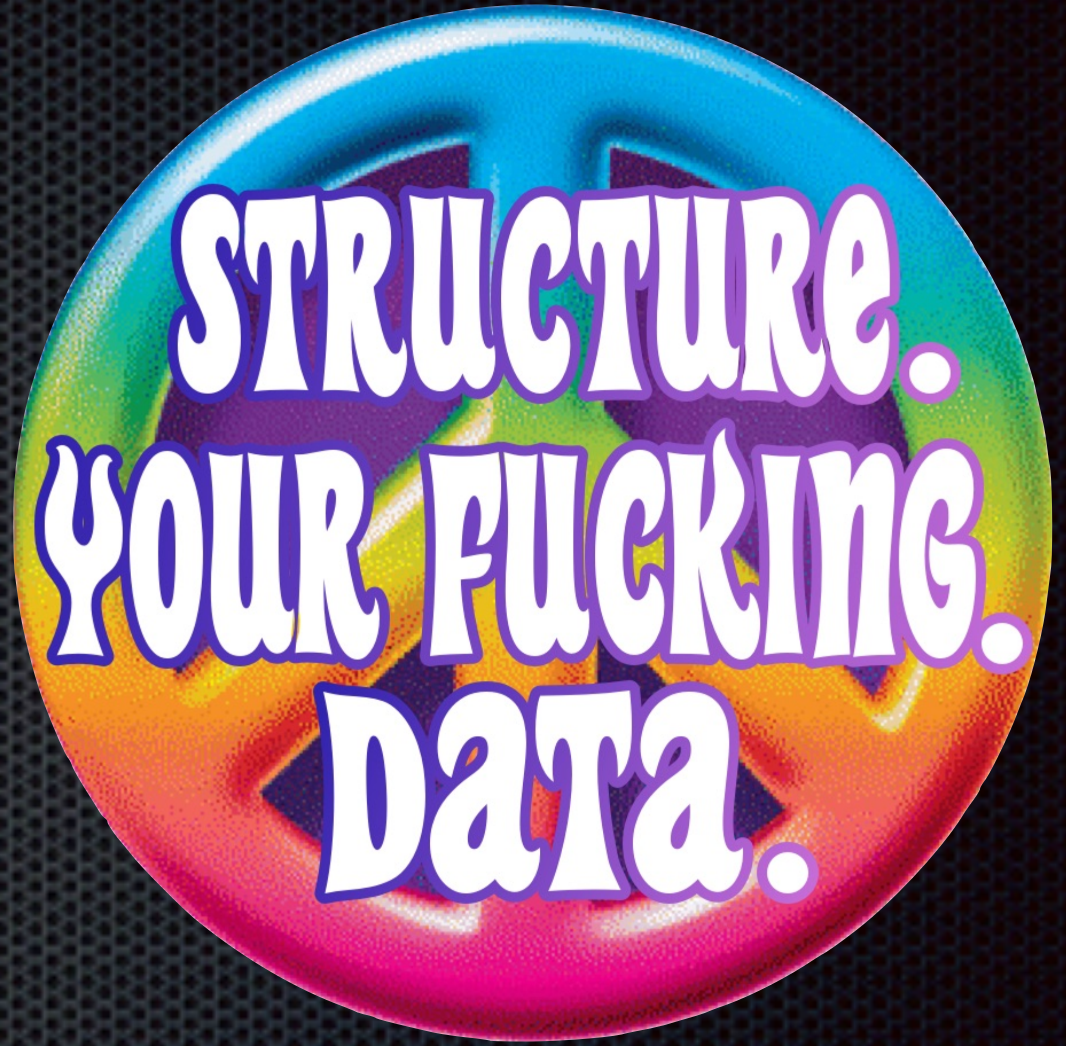
Get a jump on the next 10 years (and evade vendor lock-in) by embracing OpenTelemetry now.

Links:

https://thenewstack.io/opentelemetry-otel-is-key-to-avoiding-vendor-lock-in/

https://www.honeycomb.io/observability-precarious-grasp-topic/

Instrumentation



Metrics and logs cannot give you observability.

Instrument your code for observability,
using arbitrarily-wide structured data blobs
(or “canonical log lines”) and spans.

Links:

https://charity.wtf/2019/02/05/logs-vs-structured-events/

https://stripe.com/blog/canonical-log-lines

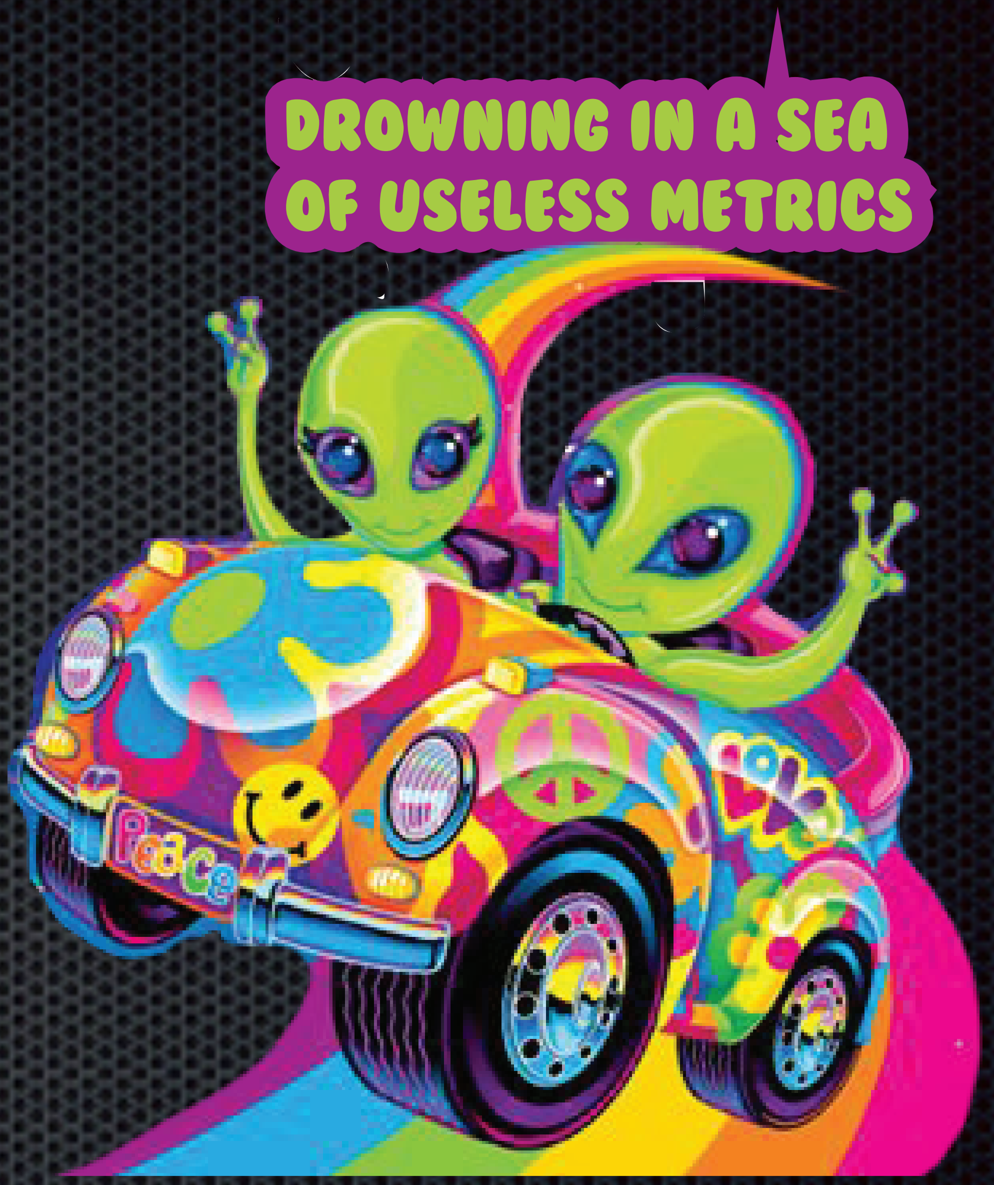
ODD

Practice not just TDD, but ODD –
Observability-Driven Development

Instrument as you go. Deploy fast. Close the loop by inspecting your changes through the lens of your instrumentation, and asking: "is it doing what I expect? does anything else look weird?"

Check your instrumentation after every deploy.
Make it muscle memory.

Shift Debugging Left



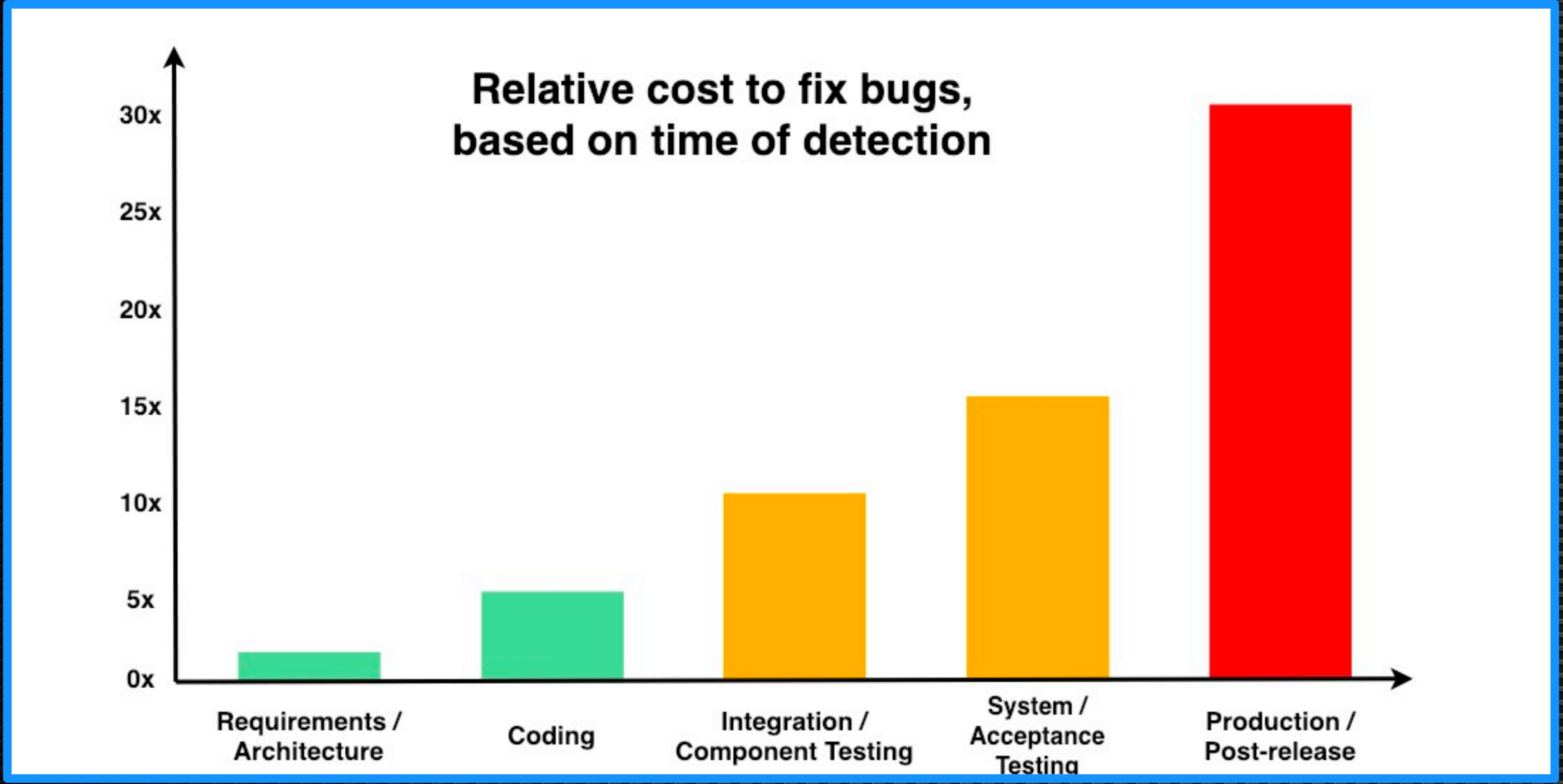
As your team climbs out of the pit of despair, you'll get paged less and less.

In order to stay that way, replace firefighting with active engagement.

Actively inspect and explore production every day. Instrument your code. Look for outliers. Find the bugs before your customers can report them.

Shift Debugging Left

Undeployed software ages like fine milk.



The cost of finding and fixing bugs goes up exponentially with time elapsed since development.

https://deepsource.io/blog/exponential-cost-of-fixing-bugs/

Retro & Resolve

If it's too big to be fixed by on call, get it on the product roadmap.

Make sure engineers have enough time to finish retro action items.

Get better at quantifying and explaining the impact of work that pays down tech debt, increases resiliency, improves dev speed – in retention, velocity, and user & employee happiness alike.

Roadmapping

Reliability work and technical debt are not secondary to features. Treat them just like product work – scope and plan the projects, don’t dig it out of the couch cushions.

Use SLOs (and human SLOs!) to assert the time you need to build a better system.

Being on call gives you the moral authority to demand change.

Test in Production

Stop over-investing in staging and under-investing in prod. Most bugs will only ever be found in prod.

Run chaos experiments (at 3pm, not 3am) to make sure you've fixed it, and consider running them continuously, forever.

If you can't reliably ascertain what's happening within a few minutes of investigating, you need better observability. This is not normal or acceptable.

Qualitative Tracking

Track things you can do, not things you hope don't happen.

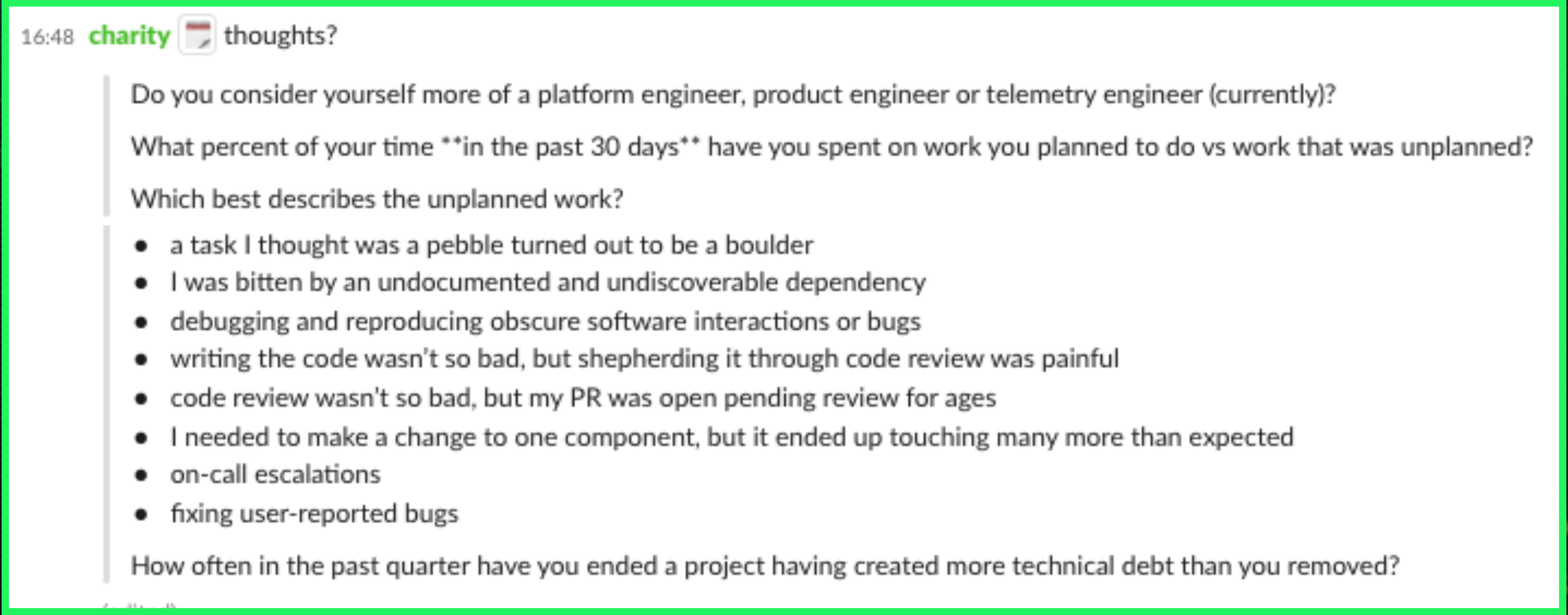
We don't want people to get burned out or have their time abused, but success is not about "not having incidents".

It's about how confident people feel being on call, whether we react in a useful manner, and increasing operational quality and awareness.

Links: https://www.honeycomb.io/blog/tracking-on-call-health/

Ask your engineers

Qualitative feedback over time is the best way to judge eng work.



16:48 charity thoughts?

Do you consider yourself more of a platform engineer, product engineer or telemetry engineer (currently)?

What percent of your time **in the past 30 days** have you spent on work you planned to do vs work that was unplanned?

Which best describes the unplanned work?

- a task I thought was a pebble turned out to be a boulder
- I was bitten by an undocumented and undiscoverable dependency
- debugging and reproducing obscure software interactions or bugs
- writing the code wasn't so bad, but shepherding it through code review was painful
- code review wasn't so bad, but my PR was open pending review for ages
- I needed to make a change to one component, but it ended up touching many more than expected
- on-call escalations
- fixing user-reported bugs

How often in the past quarter have you ended a project having created more technical debt than you removed?

Managers

Managers' reviews should note how well the team is performing, and (more importantly) what trajectory they are on. Be careful with incentives here, but some data is necessary.

Managers who run their teams into the ground should never be promoted.

1. Align engineering pain with user pain, by adopting alerting best practices
2. Track that pain and pay it down
3. Profit!

How well does your team perform?

!= “how good are you at engineering”

High-performing teams

spend the majority of their time solving interesting, novel problems that move the business materially forward.

Lower-performing teams

spend their time firefighting, waiting on code review, waiting on each other, resolving merge conflicts, reproducing tricky bugs, solving problems they thought were fixed, responding to customer complaints, fixing flaky tests, running deploys by hand, fighting with their infrastructure, fighting with their tools, fighting with each other…endless yak shaving and toil.

Build only what you must.

Value good, clean high-level abstractions that let you delegate large swaths of operational burden and software surface area to vendors.

Money is cheap; engineering cycles are not.

Operations

It is easier to keep yourself from falling into an operational pit of doom than it is to dig your way out of one.

Dedicated ops teams may be going the way of the dodo bird, but operational skills are in more demand than ever. Don't under-invest — or underpay for them.

Investments

Decouple your deploys from releases using feature flags.

Get your "run tests and deploy" time down to 15 min or less.

Invest in autodeploys after every merge. Deploy one engineer's changeset at a time.

Invest in progressive deployment

How high-performing is YOUR team?

🔥1 – How frequently do you deploy?

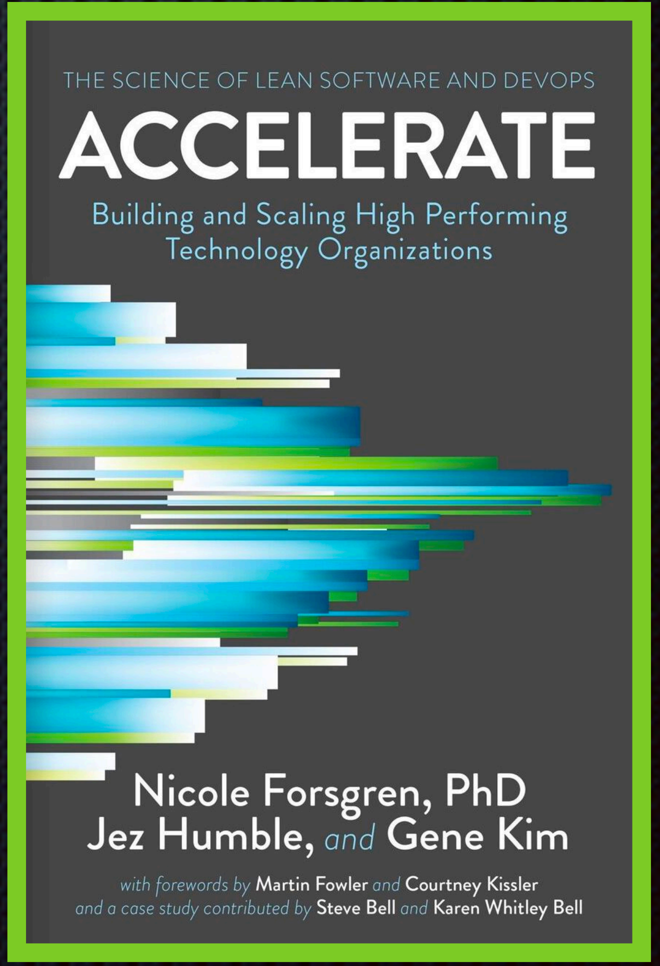
🔥2 – How long does it take for code to go live?

🔥3 – How many of your deploys fail?

🔥4 – How long does it take to recover from an outage?

🔥5 – How often are you paged outside work hours?

ACCELERATE
State of DevOps
2019


THE SCIENCE OF LEAN SOFTWARE AND DEVOPS

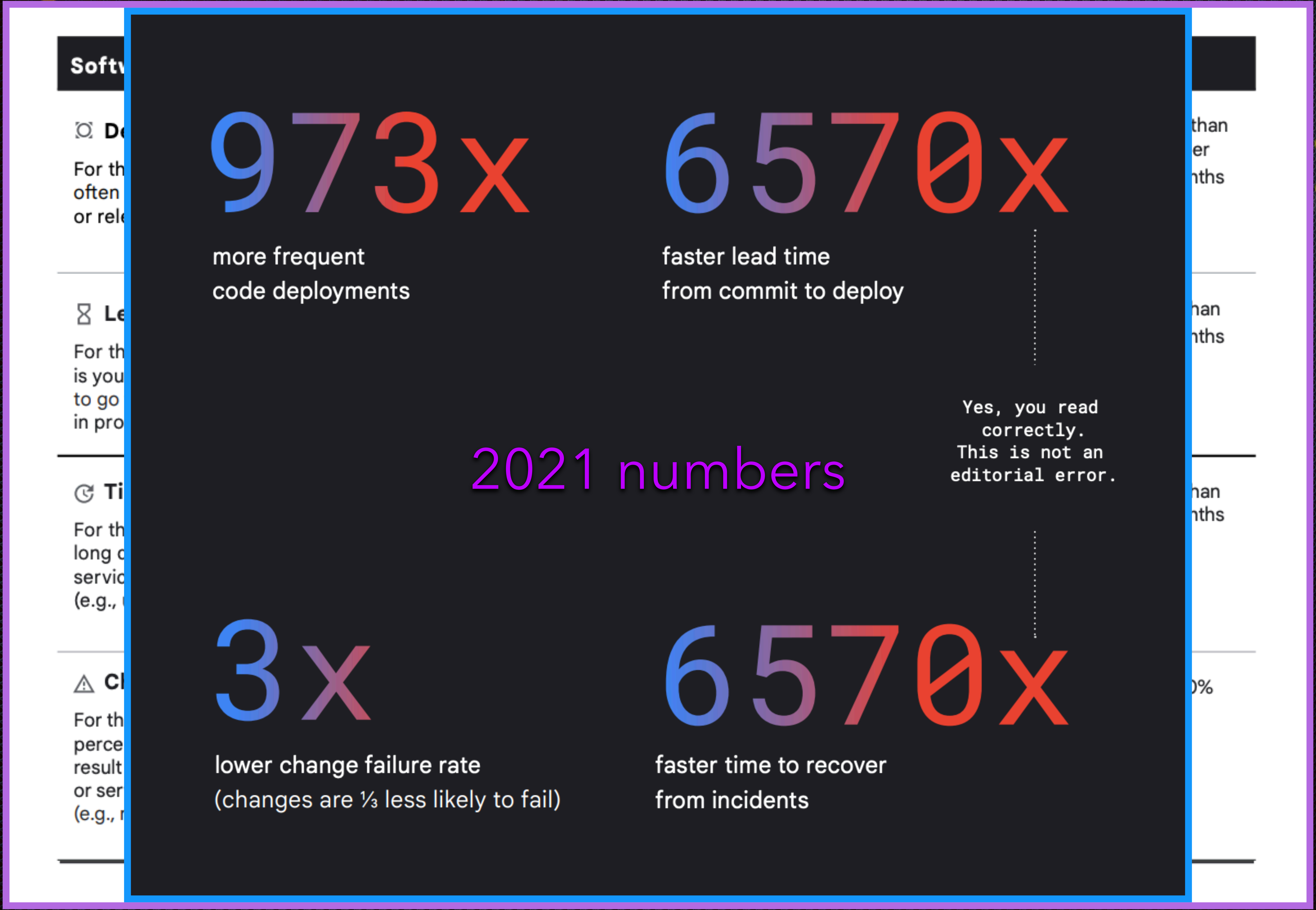
ACCELERATE

Building and Scaling High Performing Technology Organizations

Nicole Forsgren, PhD
Jez Humble, *and* Gene Kim

with forewords by Martin Fowler *and* Courtney Kissler
and a case study contributed by Steve Bell *and* Karen Whitley Bell

There is a wide gap between "elite" teams and the other 75%.



Softw

De

For th
often
or rel

Le

For th
is you
to go
in pro

Ti

For th
long c
servic
(e.g.,

Ch

For th
perce
result
or ser
(e.g., r

than
er
nths

han
nths

han
nths

0%

973x

more frequent
code deployments

6570x

faster lead time
from commit to deploy

2021 numbers

Yes, you read
correctly.
This is not an
editorial error.

3x

lower change failure rate
(changes are ⅓ less likely to fail)

6570x

faster time to recover
from incidents

Work on these things. Track these things. They matter.

Deploy frequency

Time to deploy

Deploy failures

Time to recovery

Out-of-hours alerts

Qualitative polls

Great teams make great engineers. ❤️

Your ability to ship code swiftly and safely has less to do with your knowledge of algorithms and data structures,

and much more to do with the sociotechnical system you participate in.

Technical leadership should focus intensely on constructing and tightening the feedback loops at the heart of their system.

"Technology is the sum of ways in which social groups construct the material objects of their civilizations. The things made are socially constructed just as much as technically constructed. The merging of these two things, construction and insight, is sociotechnology" – wikipedia

sociotechnical (n)

This is not just for “rockstar teams”

The biggest obstacle to operational health is rarely technical knowledge, it’s usually poor prioritization due to lack of hope. Occasionally, it’s shitty management.

It’s way easier to work on a high-performing team with auto-deployments and observability than it is to work on systems without these things.

If your team can write decent tests, you can do this.

So, now that we've done all that…

Now that we have tamed our alerts, and switched to SLOs…

Now that we have dramatically fewer unknown-unknowns…

Now that we have the instrumentation to swiftly pinpoint any cause…

Now that we auto-deploy our changes to production within minutes…

Now that night alerts are vanishingly rare, and the team is conf nt

"I'm still not happy. You said I'd be HAPPY to be on call."

the goodies

If you are on call, you are not to work on features or the roadmap that week.

You work on the system. Whatever has been bothering you, whatever you think is broken … you work on that. Use your judgment. Have fun.

this is your 20% time!!

If you were on call this week, you get next Friday off. Automatically. Always.

When it comes to work, we all want

Autonomy,

Mastery,

Meaning.

On-call can help with these.

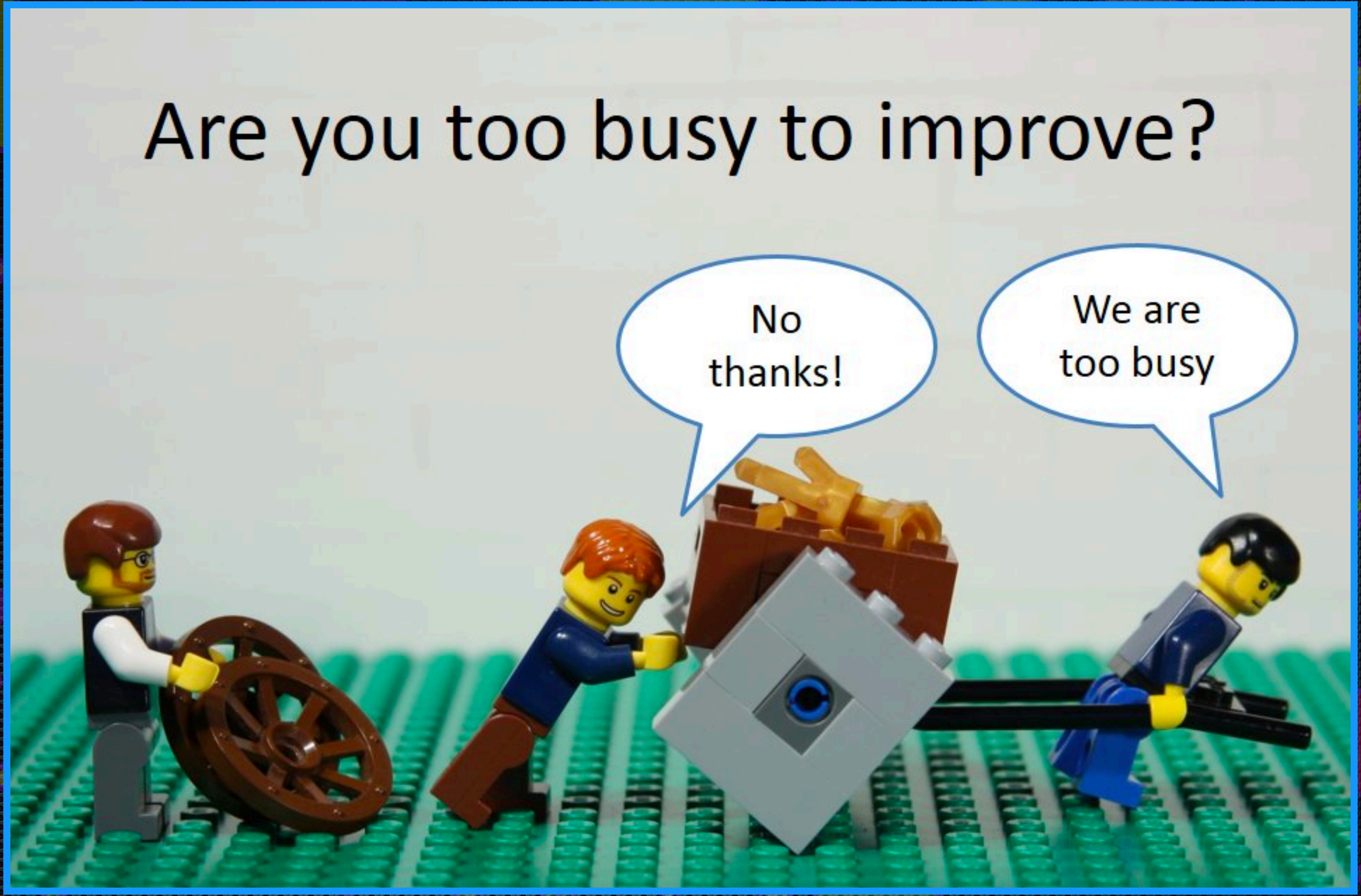
It helps to clarify and align incentives. It makes users truly, directly happy. It increases bonding and teaminess. I don't believe you can truly be a senior engineer unless you're good at on call. The one that we need to work on adding is autonomy…and not abusing people.

I'm of mixed mind about paying people for being on call. I mostly think engineers are like doctors: it's part of the job. With one **big** exception.

If you are struggling to get your engineers the time they need for systems work instead of just cranking features,

you should start paying people a premium every time they are alerted out of hours. Pay them a LOT. Pay enough for finance to complain. Pay them until management is begging you to work on reliability to save them money.

If execs don't care about your people's time and lives, convert it into something they do care about. ✨**Money**✨



The End ☺️

Charity Majors
@mipsytipsy