Observability is due for a version change: are you ready for it?



@mipsytipsy

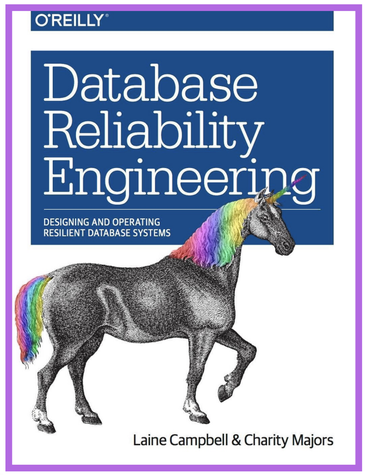
honeycomb.io

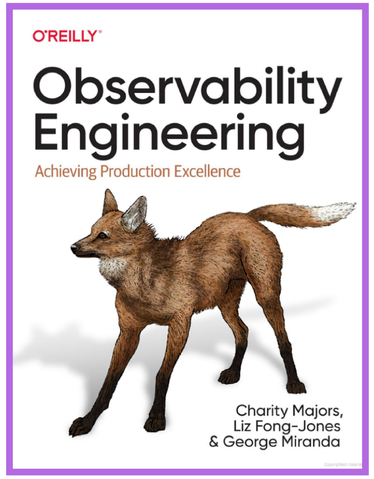


@mipsytipsy

engineer/cofounder/CTO

https://charity.wtf

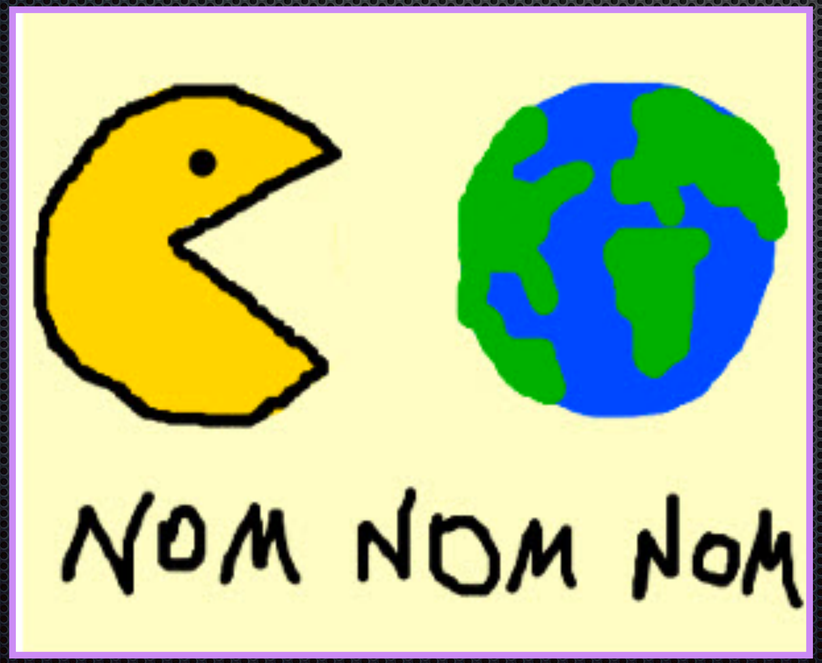


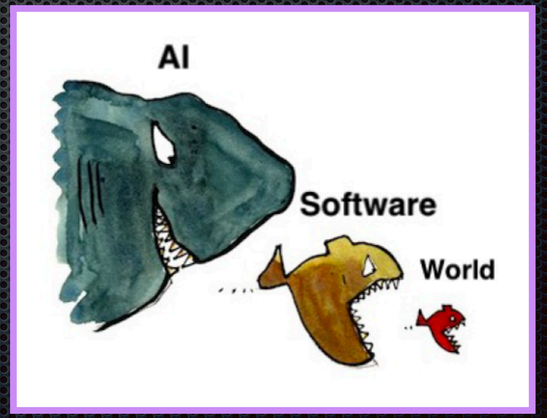


honeycomb.io

Every company is now a technology company

Engineering efficiency and execution
are no longer niche concerns





It really pays to be on a high-performing team.

High-performing teams get to spend most of their time working on interesting, novel problems that move the business materially forward.

The **team** is the smallest viable unit of software ownership. Individuals don't own software.



How do we build high-performing teams?

“By hi all th artest
peopl eatest
engineers -Googlers
we can ou nds on”

NO.



“How well does your team perform?” != “How good are you at engineering?”

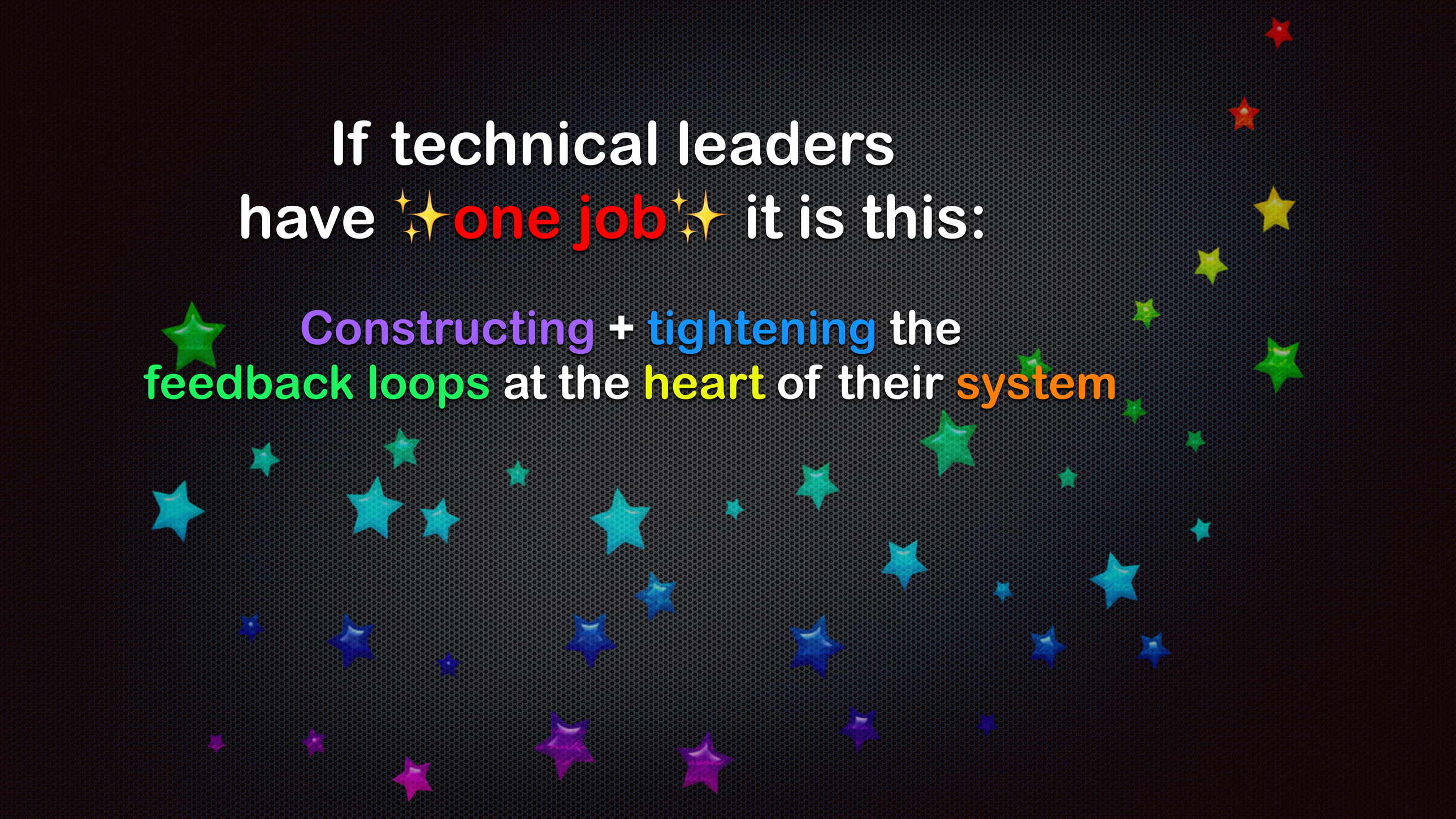
Your ability to ship code swiftly + safely
has less to do with your knowledge
of algorithms & data structures,

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



If technical leaders have ✨one job✨ it is this:

Constructing + tightening the feedback loops at the heart of their system



Modern software development practices

Engineers own their code in production

Practice observability-driven development

Test in production

Separate deploys from releases using feature flags

Continuous deployment (or at least delivery)

Modern software development
best practices are ✨ALL✨ about:

FAST FEEDBACK LOOPS

Get your code into production
as fast as possible after writing it.

When it comes to software,

speed is safety.



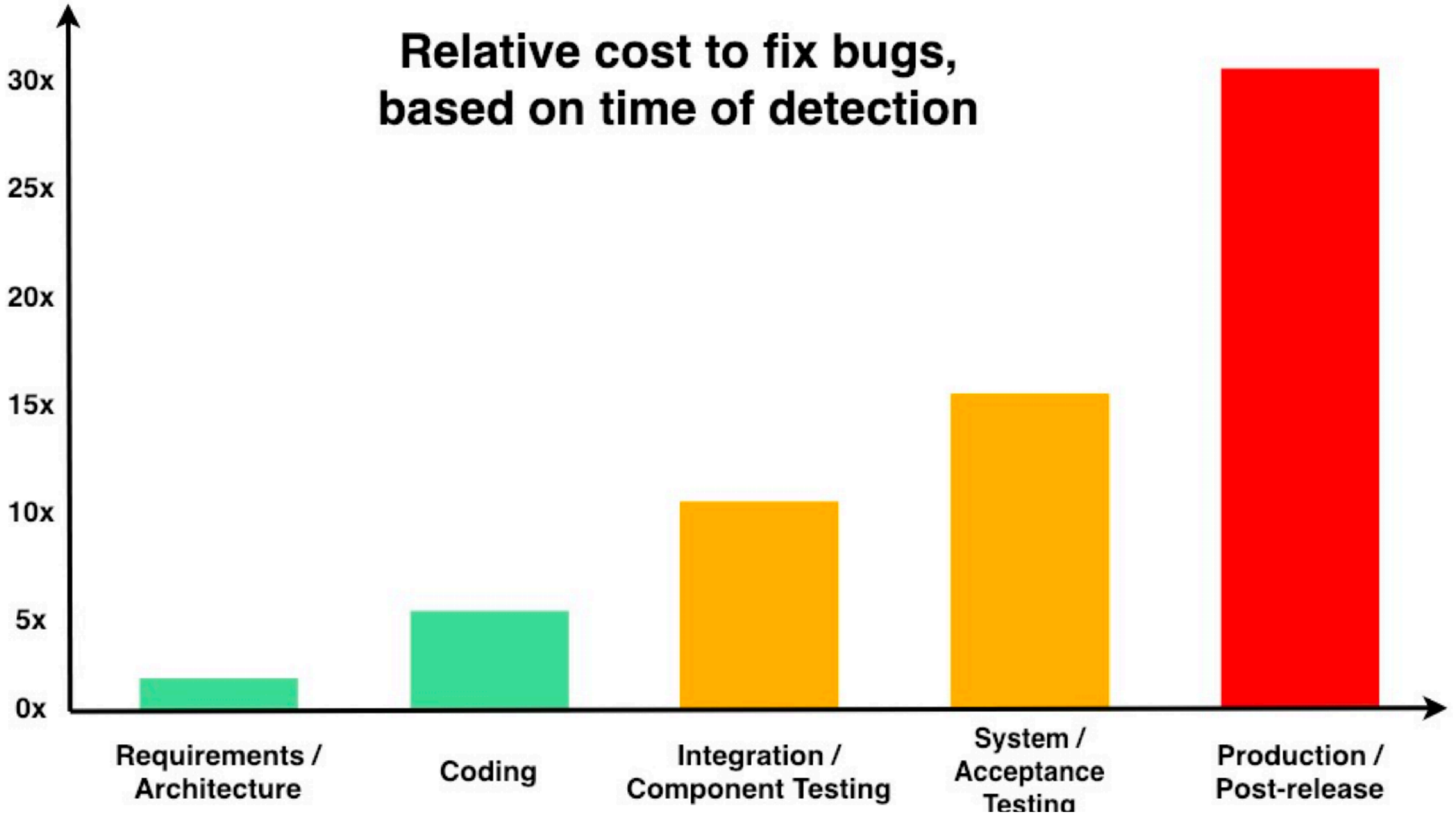


Getting code into production fast is the

key feedback loop

that everything else proceeds from.

The cost of finding and fixing bugs goes up exponentially from the moment you write them.



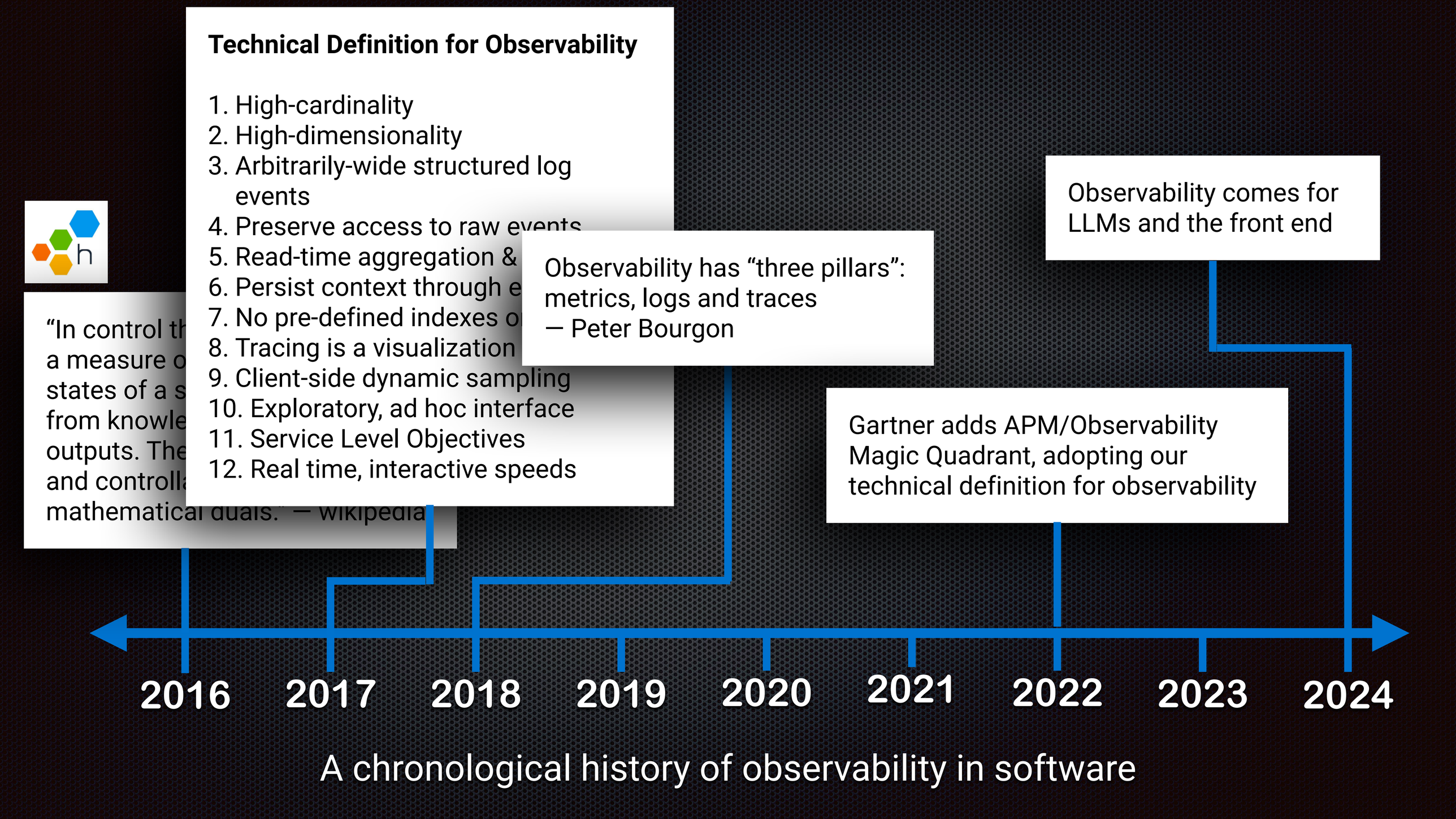


Your ability to move swiftly, with confidence, is grounded in the quality of your observability.

But what does that even *mean*, these days??

(Great question! 🙋)





A chronological history of observability in software

Observability is a ✨property✨ of complex systems.

But … how then to account for the enormous step function in value, usability, cost model, etc between generations of tooling?

Observability

1.0 → 2.0

“Three pillars:”
metrics, logs, traces

Single source of truth:
wide structured logs

Observability 1.0

Data:	Metrics, logs and traces, captured separately
Source of truth:	Many. APM, RUM, logging, tracing, metrics, analytics…
Interface:	Static dashboards
Debugging:	Debug based on intuition, scar tissue from past outages, and guesswork
Alerts:	Page on symptoms appearing in metrics
Cost:	Pay to store your data many times

Observability 2.0

Data:	Wide, rich structured logs (aka events or spans), with high cardinality and high dimensionality
Source of truth:	One
Interface:	Exploratory, interactive; no dead ends
Debugging:	Follow the trail of breadcrumbs. It’s in the data.
Alerts:	Page on customer pain via SLOs
Cost:	Pay to store your data once

You have observability only if you have…

1. High-cardinality
2. High-dimensionality
3. Arbitr ily-wide structur log event
4. Preserve ss t events
5. Read-time a ion & querying
6. Persist co gh execution
7. No pre- d ind or schemas
8. Trac a visualizat er time
9. Client-side dynamic sampling
10. Exploratory, ad hoc interface
11. Service Level Objectives
12. Real time, interactive speeds

If you have three pillars, and many tools:

Observability 1.0

If you have a single source of truth:

Observability 2.0

How the data gets stored

OBSERVABILITY 1.0

- Metrics
- Logs
- Traces
- APM
- RUM
- …

- Tracing is an entirely different tool

- Siloed tools, with no connective tissue or only a few, predefined connective bits

- Write-time aggregation

OBSERVABILITY 2.0

- Arbitrarily-wide structured data blobs

- Single source of truth

- Tracing is just visualizing over time

- It’s just data. Treat your data like data.

- Read-time aggregation; raw events

There are only three types of data:

1. The metric
2. Unstructured logs (strings)
3. Structured logs

RUM tools are built on top of metrics to understand browser user sessions.

APM tools are built on top of metrics to understand application performance.





Tiny, fast, and cheap
Each metric is a single number,
with some tags appended
Stored in TSDBs

NO context.
NO high cardinality.
NO data structures.
NO ability to correlate or dig deeper
Only basic static dashboards.
EXTREMELY limited.

Metrics are the right tool for summarizing vast quantities of data, and aggregating it in a way so it can cheaply age out and lose fidelity over time.

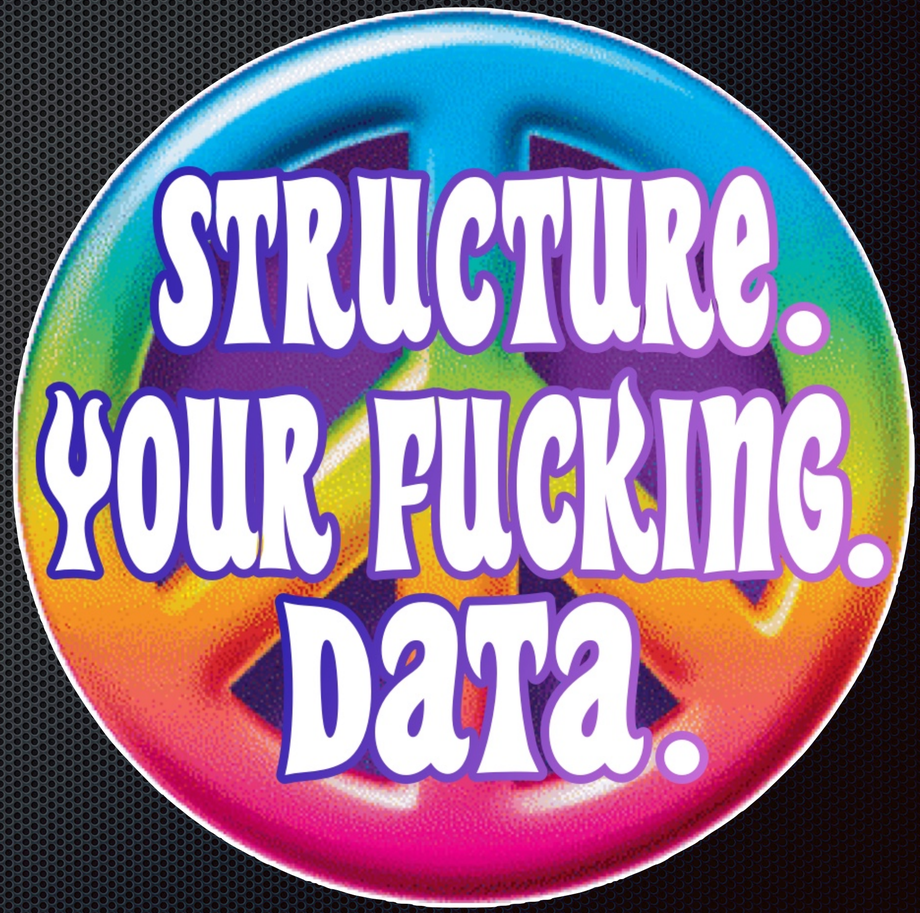
They are not equipped to help you introspect and understand your software.

To understand our systems, we turn to logs. Even unstructured logs are more powerful than metrics, because they preserve SOME context and connective dimensions.

Unstructured Logs

However, you have to know what you're looking for in order to find it.

And the only thing you can do is string search, which is sllooooooowwwww.



Who uses it, and how?

OBSERVABILITY 1.0

- About MTTR, MTTD, and reliability
- Usually a checklist item before shipping code to production — “how will we monitor this?”

- An “ops concern”

- Static dashboards

- No support for structured data

OBSERVABILITY 2.0

- Underpins the entire software development lifecycle.

- Part of the development process

- Exploratory, open-ended interface

- High cardinality
- High dimensionality

Observability 1.0 is about how you ✨operate✨ software

It is traditionally focused more on bugs, errors, MTTR, MTTD, reliability, monitoring, and performance.

Observability 2.0 is about how you ✨develop✨ software

It is what underpins the entire software development lifecycle, allowing you to hook up tight feedback loops and move swiftly, with confidence.

How you interact with production

OBSERVABILITY 1.0

- You deploy your code and wait to get paged. 🤞
- Your job is done when you commit your code and tests pass
- Your world is broken up into two very different universes, Dev & Prod

OBSERVABILITY 2.0

- You practice Observability-Driven Development
- Your job isn’t done until you’ve verified it works in production.
- These worlds are porous and overlapping
- You are in constant conversation with your code. 💜

How you debug

OBSERVABILITY 1.0

- You flip from dashboard to dashboard, pattern-matching with your eyeballs
- You lean heavily on intuition, past experience, and a rich mental model of the system
- The best debuggers are always the engineers who have been there the longest and seen the most.
- Search-first

OBSERVABILITY 2.0

- You form a hypothesis, ask a question, consider the results, and ask another based on the answer.
- You don’t have to guess. You follow the trail of breadcrumbs to the answers, every time.
- The best debuggers are the people who are the most curious.
- Analysis-first

The cost model

OBSERVABILITY 1.0

- You pay to store your data again and again and again and again, multiplied by the number of tools
- Cost goes up (at best) linearly, driven by the number of custom metrics you define
- Cost for individual metrics can spike massively and unpredictably
- Keeping costs under control requires ongoing investment from engineering

OBSERVABILITY 2.0

- You pay to store your data ✨once✨
- You can store infinite “custom metrics”, appended to your events
- Powerful, surgical options for controlling costs via head-based or tail-based dynamic sampling

Why does observability 1.0 cost so much?

Because you have to pay for so many different tools / pillars, your costs rise at a multiplier of your traffic (5x? 7x?)

Because so many of those tools are built on metrics

Because of the high overhead of ongoing engineering labor to manage costs and billing data

Because of the dark matter of lost engineering cycles.

https://www.honeycomb.io/blog/cost-crisis-observability-tooling

Envelope math: cost of a custom metric

`request.Latency`

5 hosts, 4 endpoints, 2 status codes as a Count metric: **40 custom metrics**

1000 hosts, 5 methods, 20 handlers, 63 status codes as a Count metric: **6.3M custom metrics**

1000 hosts, 5 methods, 20 handlers, 63 status codes as a histogram using defaults (max, median, avg, 95pct, count): **31.5M custom metrics**

1000 hosts, 5 methods, 20 handlers, 63 status codes as a histogram using defaults (max, median, avg, 95pct, count) plus distribution (99pct, 99.5pct, 99.9pct, 99.99pct: **63M custom metrics**

A DataDog acct comes with 100-200 free custom metrics, and costs 10 cents for every 100 over. 63M custom metrics costs you $63,000/month for `request.Latency`

The cost model

OBSERVABILITY 1.0

- Ballooning costs are baked in to the 1.0 model. 🙁
- As your bill goes up, the value you get out of your tools actually *goes down*.
- Metrics and unstructured logs both suffer from opaque, bursty billing and degrade in punishing ways

OBSERVABILITY 2.0

- Your costs go up as your traffic goes up and as you add more spans for finer-grained inspection
- As your bill goes up, the value you get out of your tools goes up too.
- Costs effectively nothing to widen structured data & add more context

With observability 1.0 tools, as costs go up, the value you get out goes down.

Ballooning costs are baked into observability 1.0. ☹️

Observability 2.0 isn't "cheap", but its costs are predictable and aligned with engineering value.

Observability 2.0 is faster, cheaper, and simpler to use.

The way you are doing it NOW is the hard way.

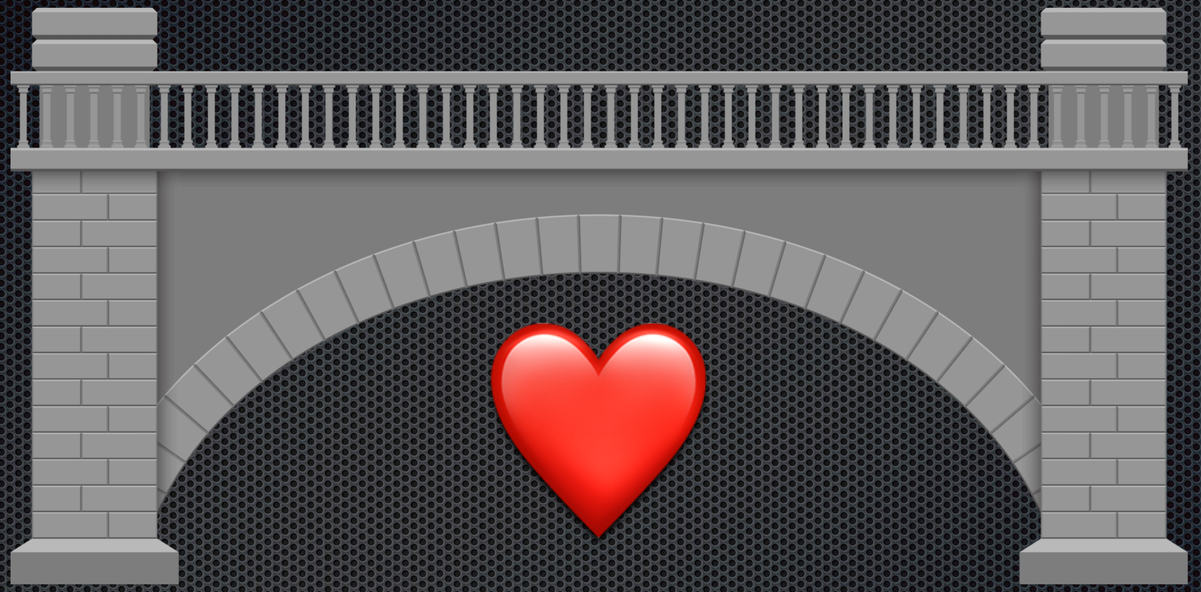
We have learned to be insanely clever when it comes to wringing every last bit of utility out of metrics and unstructured logs.

What if we didn’t have to work that hard?

What if it was all just … data.



Structured logs are the bridge to the future.



Metrics are a bridge to our past.

Metrics aren’t *completely* useless; they still have their place! (In infrastructure 😛.)

Envelope math: cost of a structured log

`request.Latency`

With structured logs, you should be able to capture each of these dimensions

```
endpoint
method
status_code
handler
hostname
```

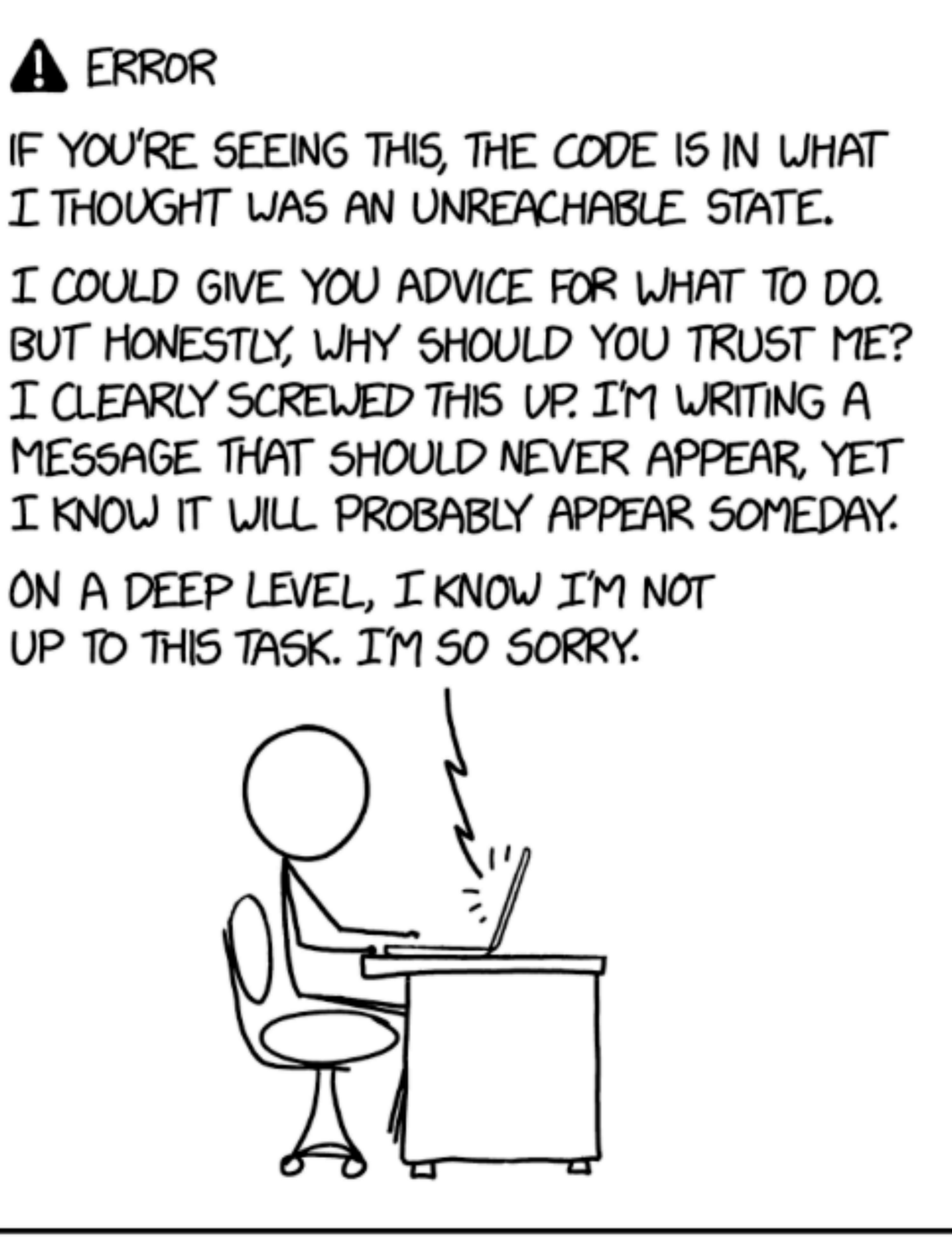
and just slice and dice, break down and group by any combination of dimensions.

AND SO MUCH MORE ...

```
app_id
user_id
shopping_cart_id
build_id
… etc
```

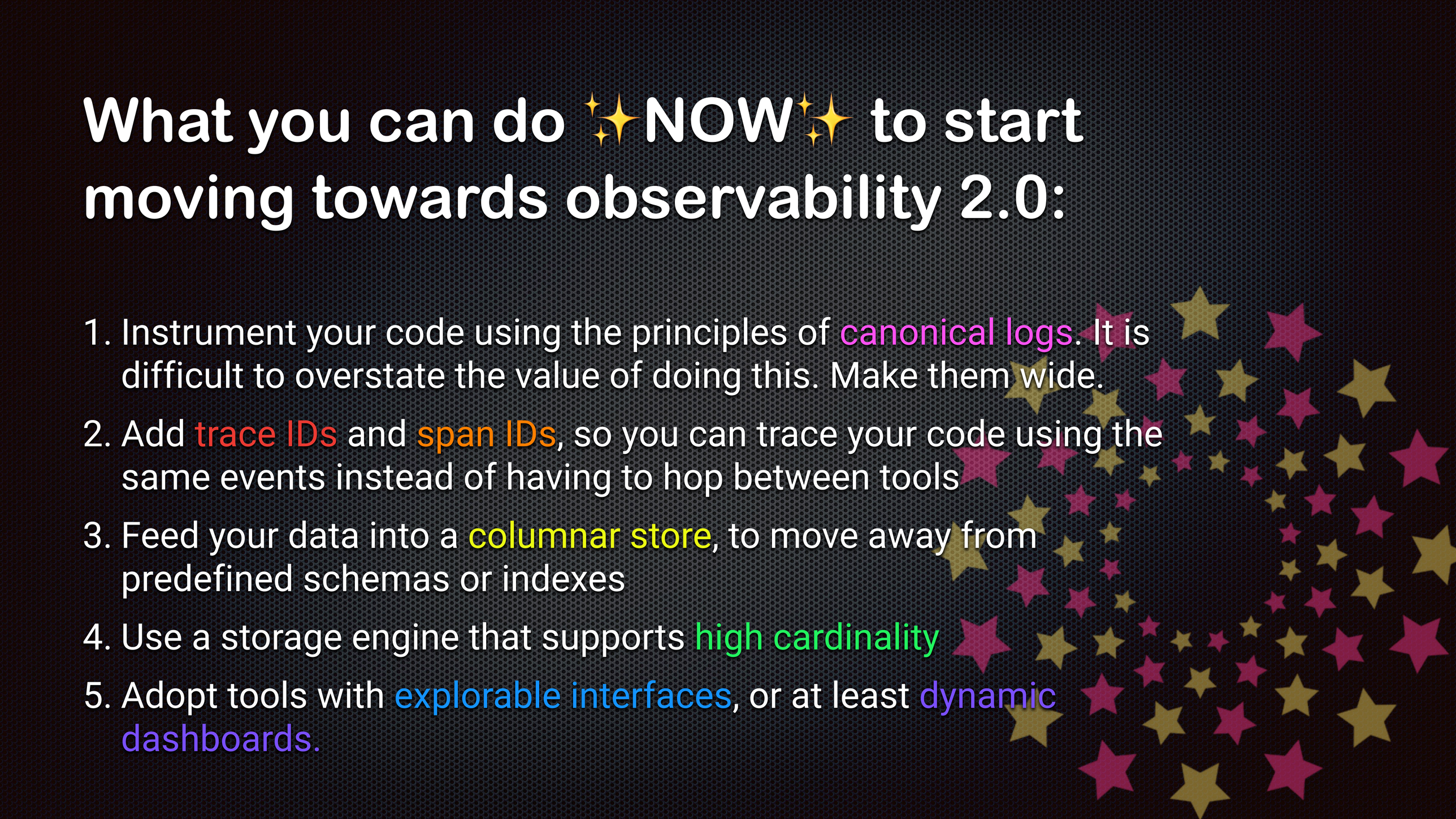
You build better systems by building software this way.

You become a better engineer by building software this way.



What you can do ✨NOW✨ to start moving towards observability 2.0:

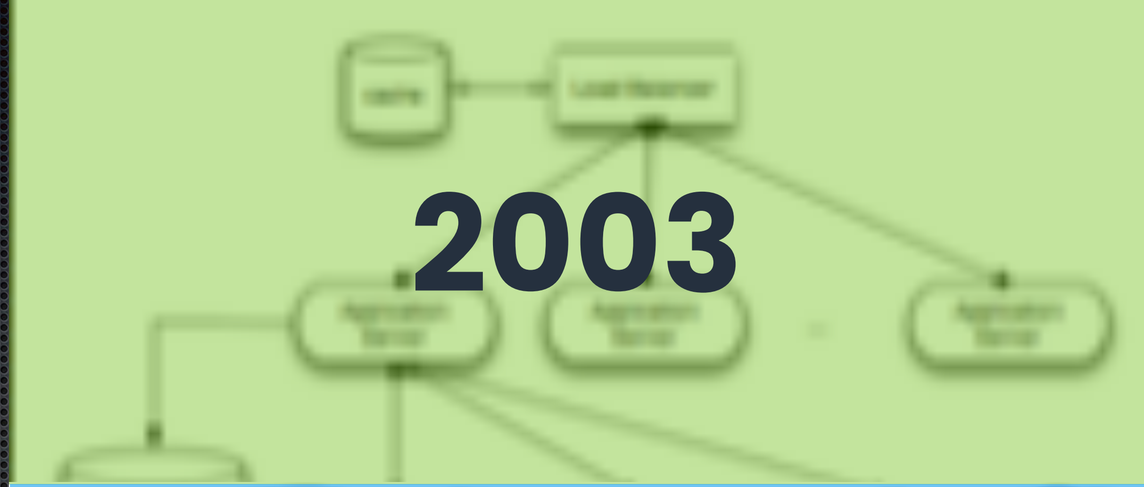
1. Instrument your code using the principles of canonical logs. It is difficult to overstate the value of doing this. Make them wide.
2. Add trace IDs and span IDs, so you can trace your code using the same events instead of having to hop between tools
3. Feed your data into a columnar store, to move away from predefined schemas or indexes
4. Use a storage engine that supports high cardinality
5. Adopt tools with explorable interfaces, or at least dynamic dashboards.

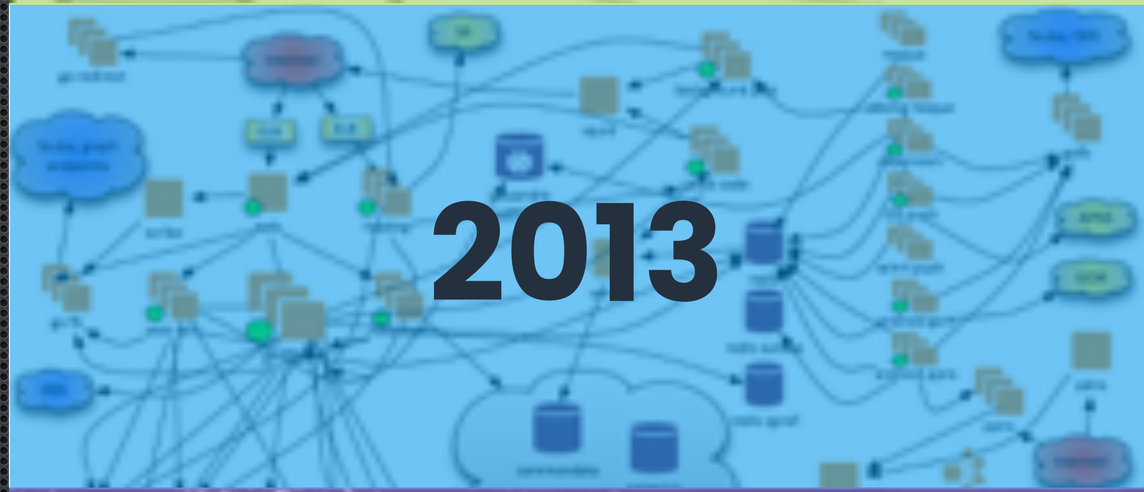


What got us here won’t get us there.

We used to be able to reason about our architecture. Not anymore.

Now we have to instrument for observability, or we are screwed.







Here’s the dirty little secret:

The next generation of systems won’t be built and run by burned-out, exhausted people, or command-and-control teams just following orders.

It can’t be done.

Our systems have become too complicated…too hard.
The shit that can be done on autopilot will be automated out of existence.

Writing code is not the hard part.

It never has been. The hard part of software is understanding it, maintaining it, extending it, scaling it, operating it, migrating it, refactoring it, crafting the right level of abstractions, instrumenting it, reasoning about it.

We can no longer hold a model of these systems in our heads and reason about them, or intuit the solution.

Those who try will lose.

Our systems are emergent and unpredictable. Runbooks and canned playbooks won’t work; it takes your full creative self.



Observability 2.0 advances the craft of software engineering.

We are trying to make it faster and safer to bring change to the world.

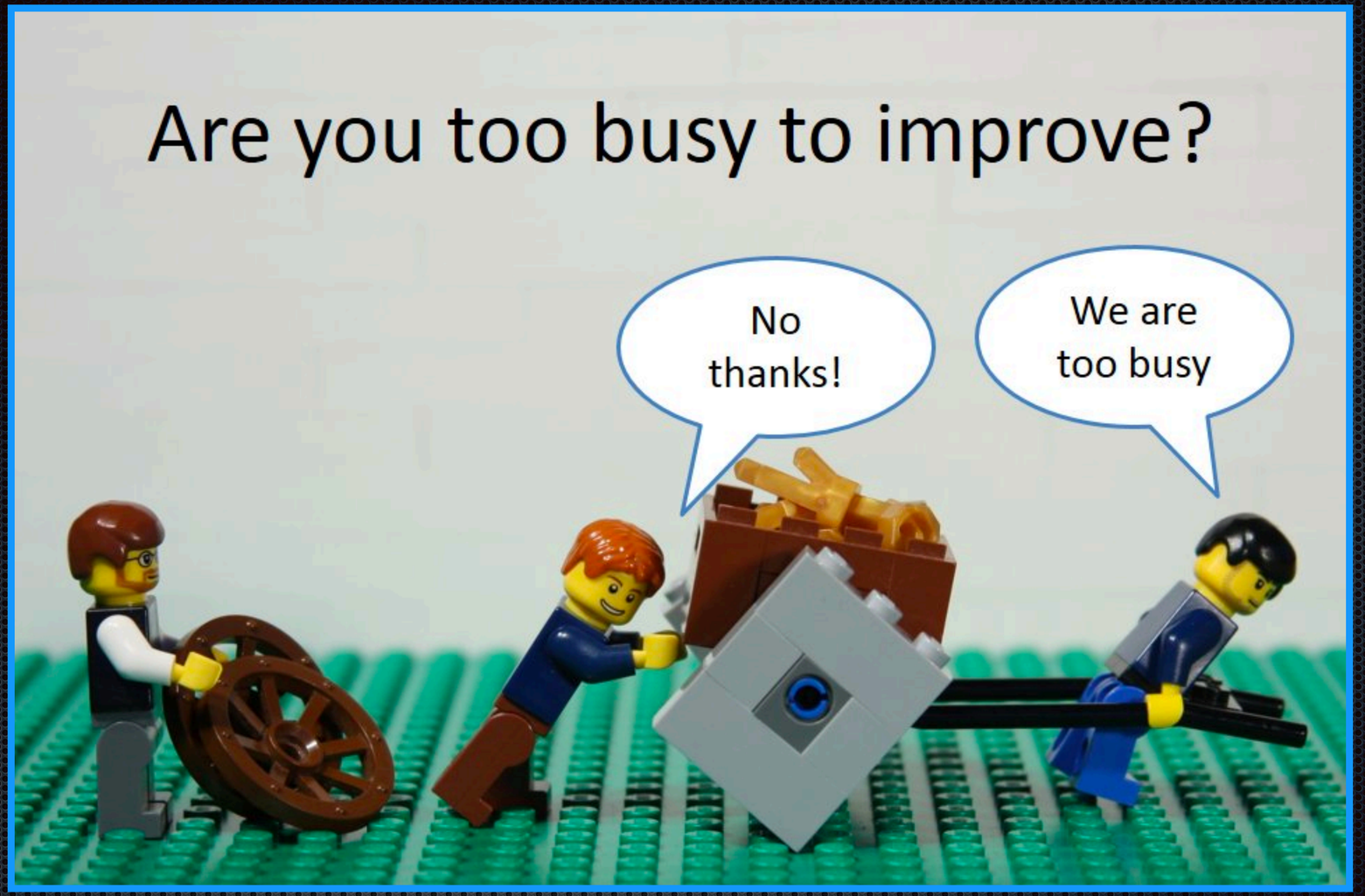
We are trying to make this a humane profession.

The biggest obstacle between us and a better world, is when we don’t believe one is actually possible.

Demand more from your tools. Demand more from your vendors.

Everyone writes code. Everyone owns their code in production,
And everybody deserves the tools to do it efficiently and well.





The End ☺️

Charity Majors
@mipsytipsy

