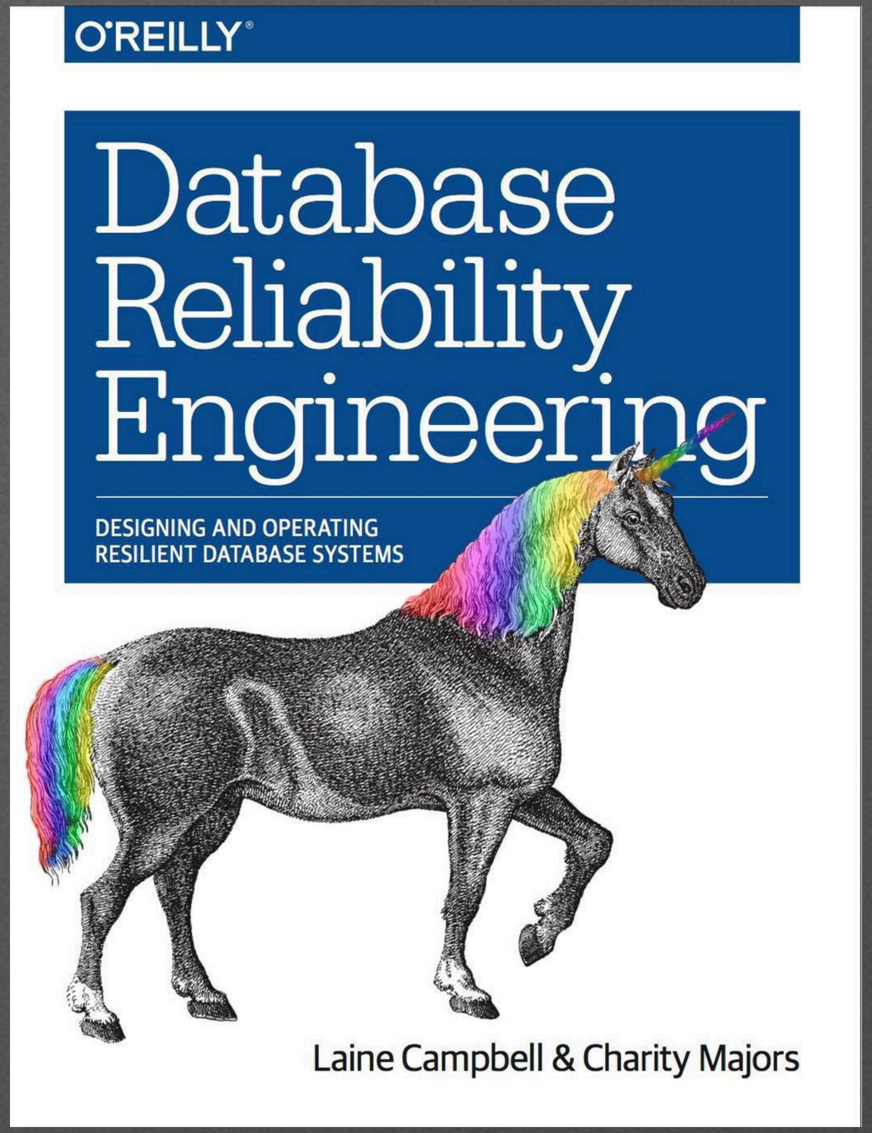
YES, I test in production.

And so should you.

By Charity Majors
@mipsytipsy

SECOND LIFE

honeycomb



O'REILLY

Database Reliability Engineering

DESIGNING AND OPERATING RESILIENT DATABASE SYSTEMS

Laine Campbell & Charity Majors

@mipsytipsy

engineer/cofounder/CEO

"the only good diff is a red diff"

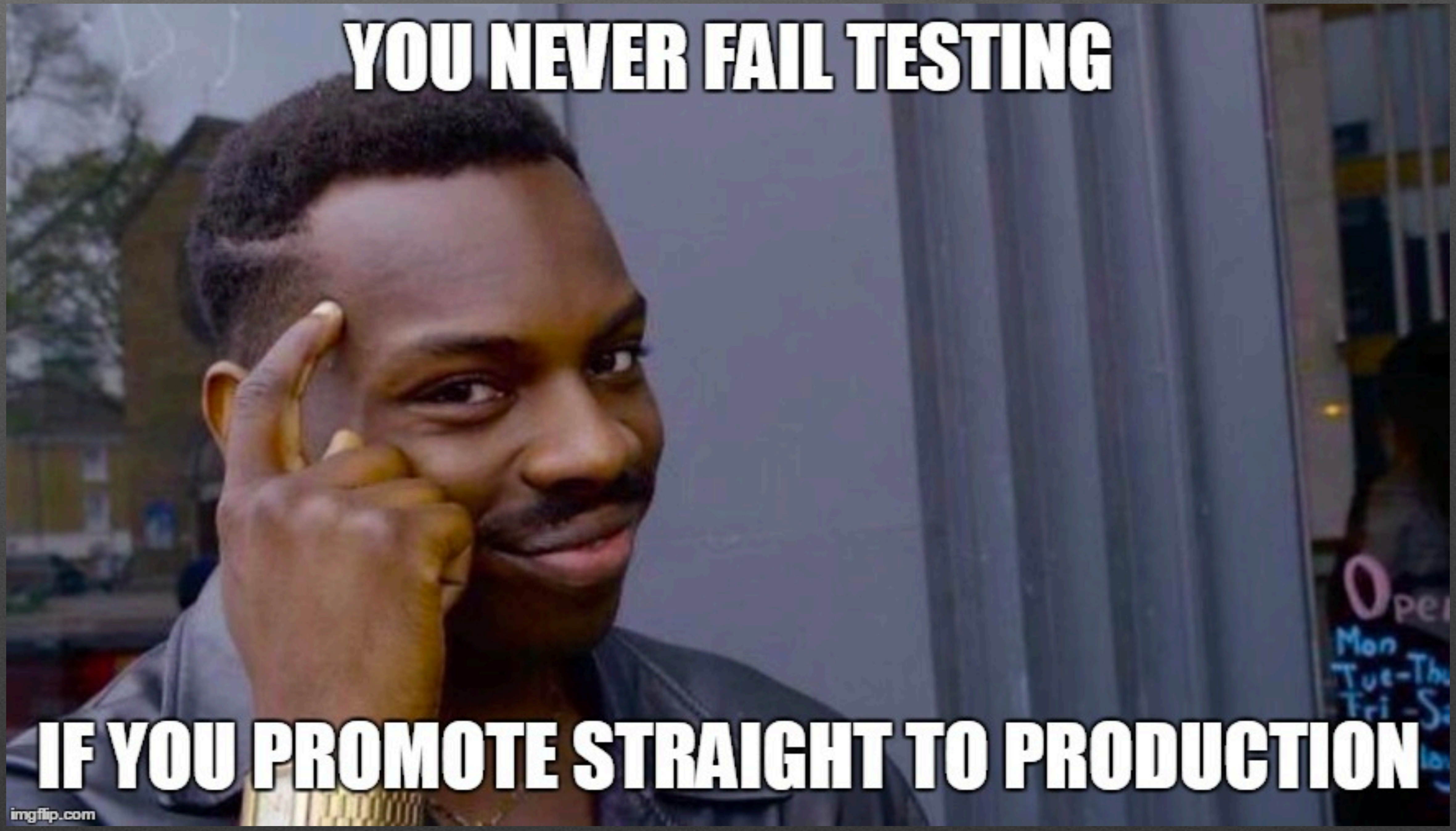
https://charity.wtf

Parse

The Cloud Application Platform

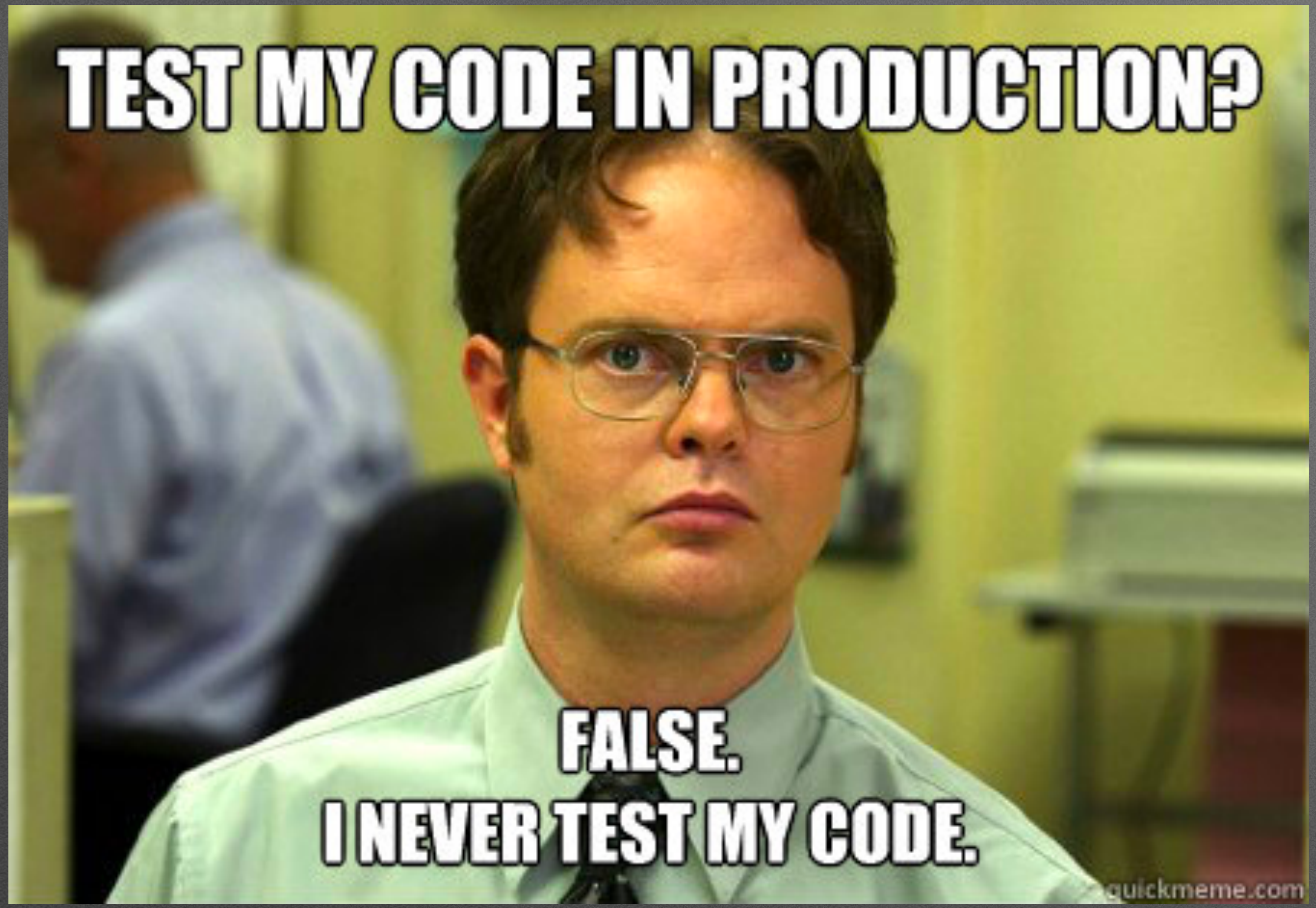
Testing in production has gotten a bad rap.

- Cautionary Tale
- Punch Line
- ~~Serious Strategy~~

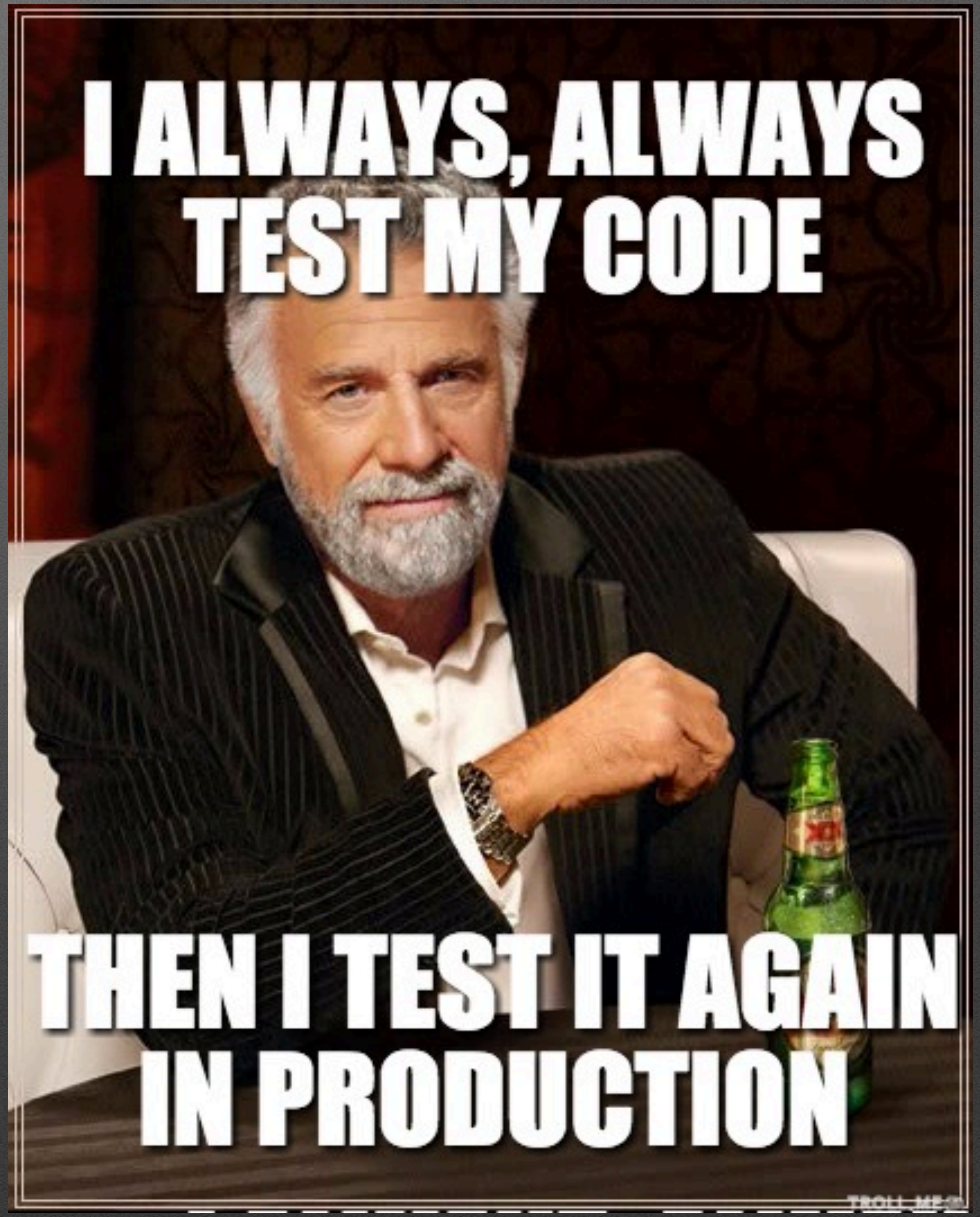


I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO I DO IT IN PRODUCTION
memegenerator.net

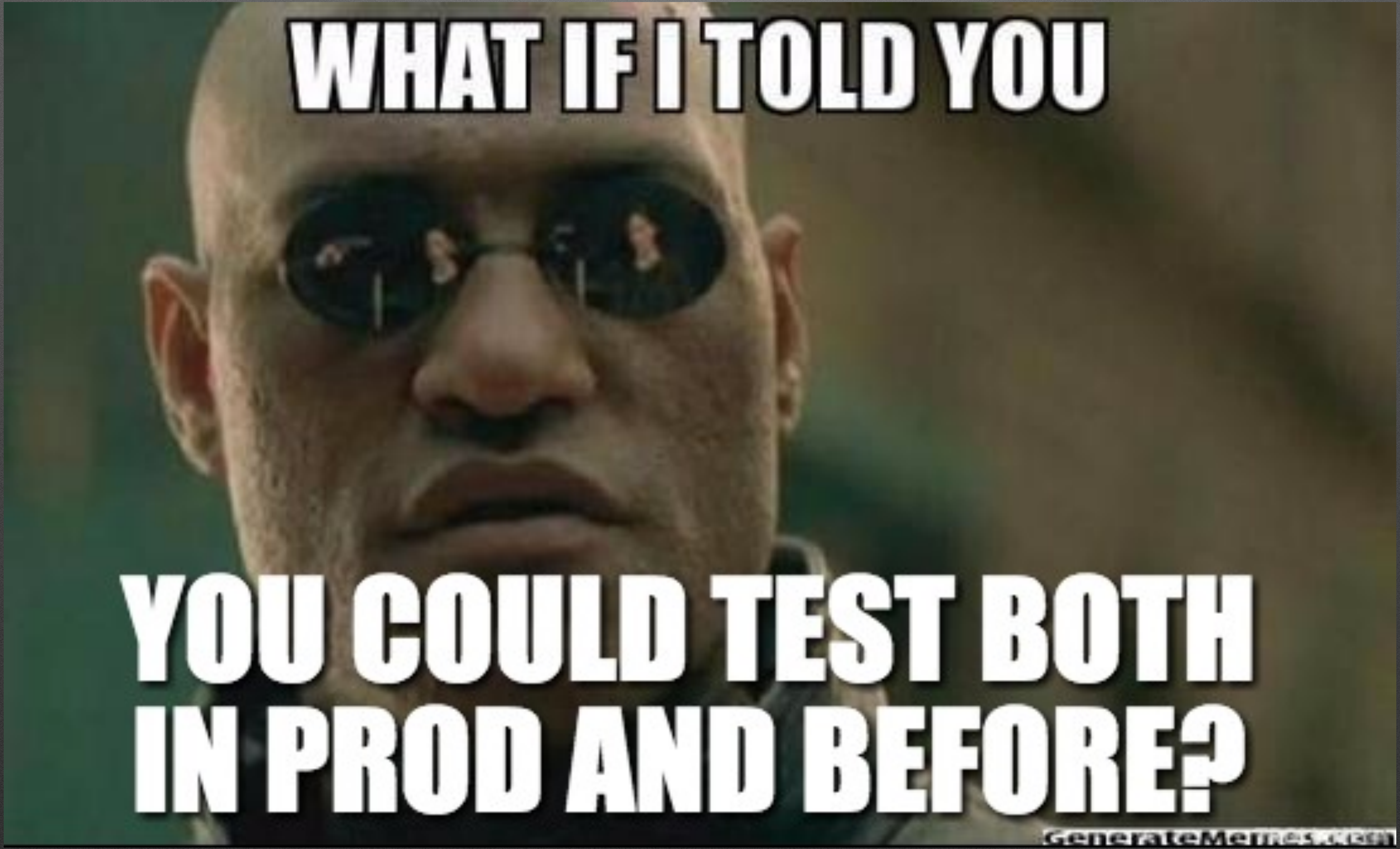
(I blame this guy)

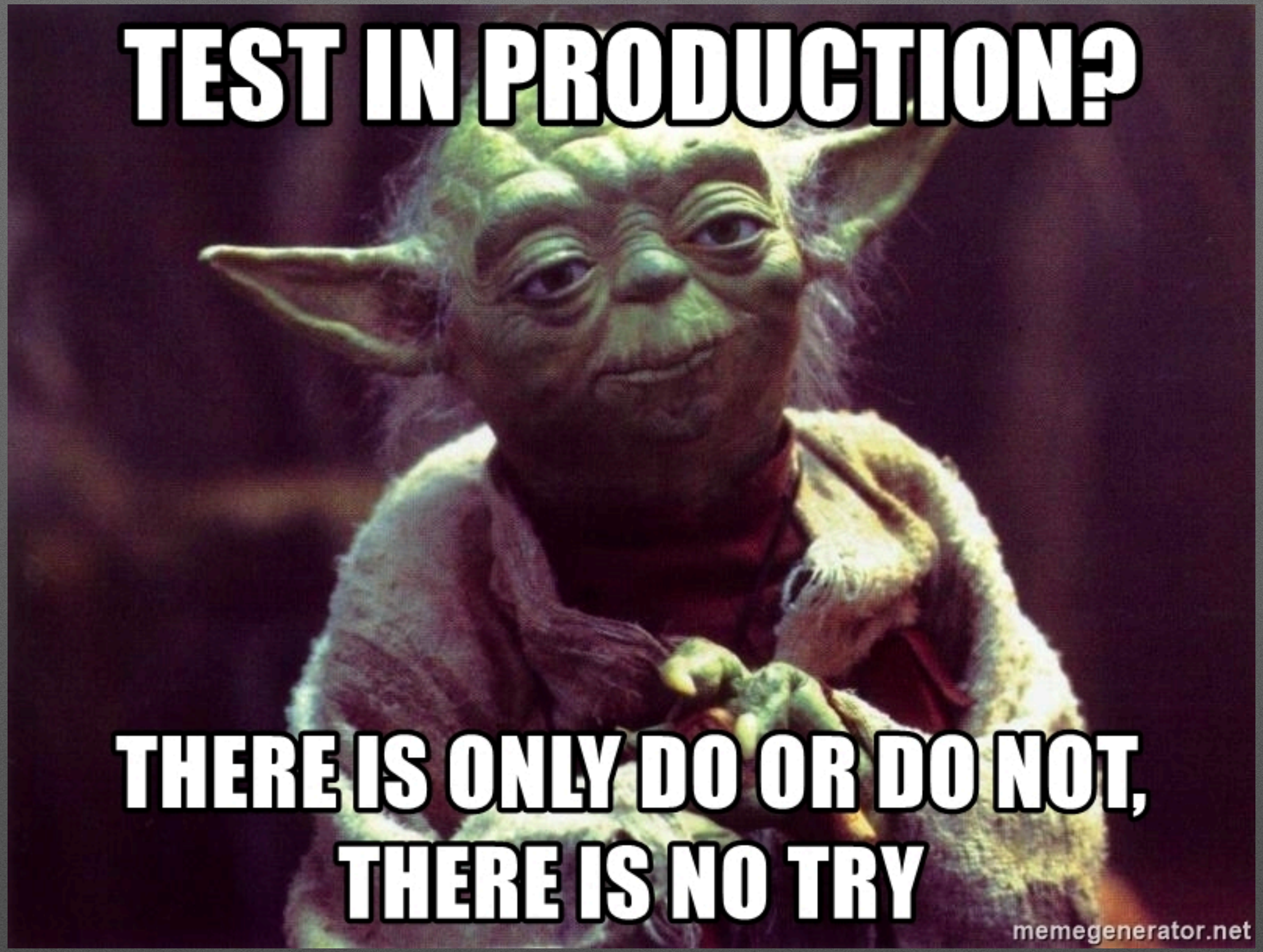


how they think we are


I ALWAYS, ALWAYS TEST MY CODE
THEN I TEST IT AGAIN IN PRODUCTION

how we should be

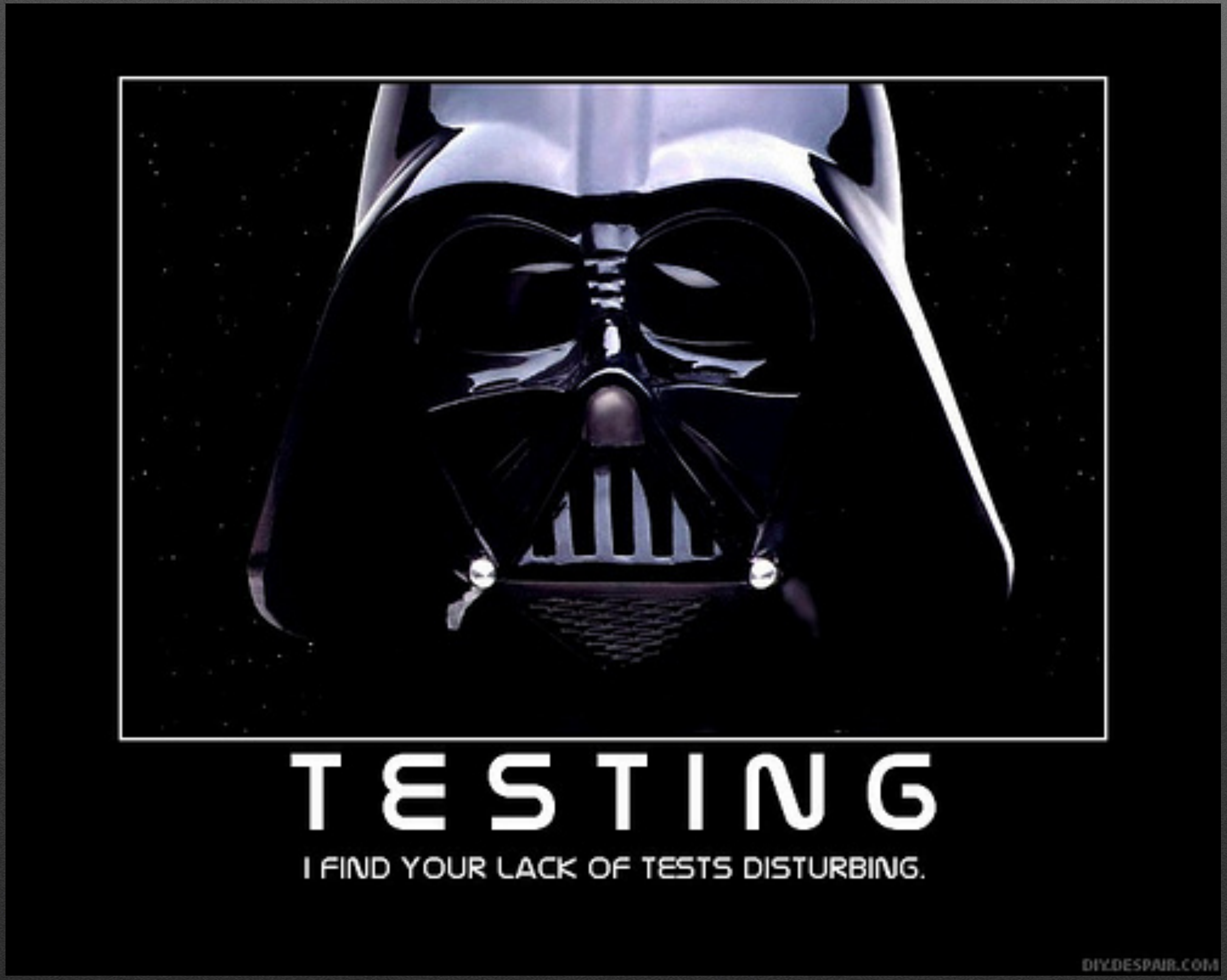

WHAT IF I TOLD YOU
YOU COULD TEST BOTH IN PROD AND BEFORE?
GenerateMemes.com



Test(n): take measures to check the quality, performance, or reliability.

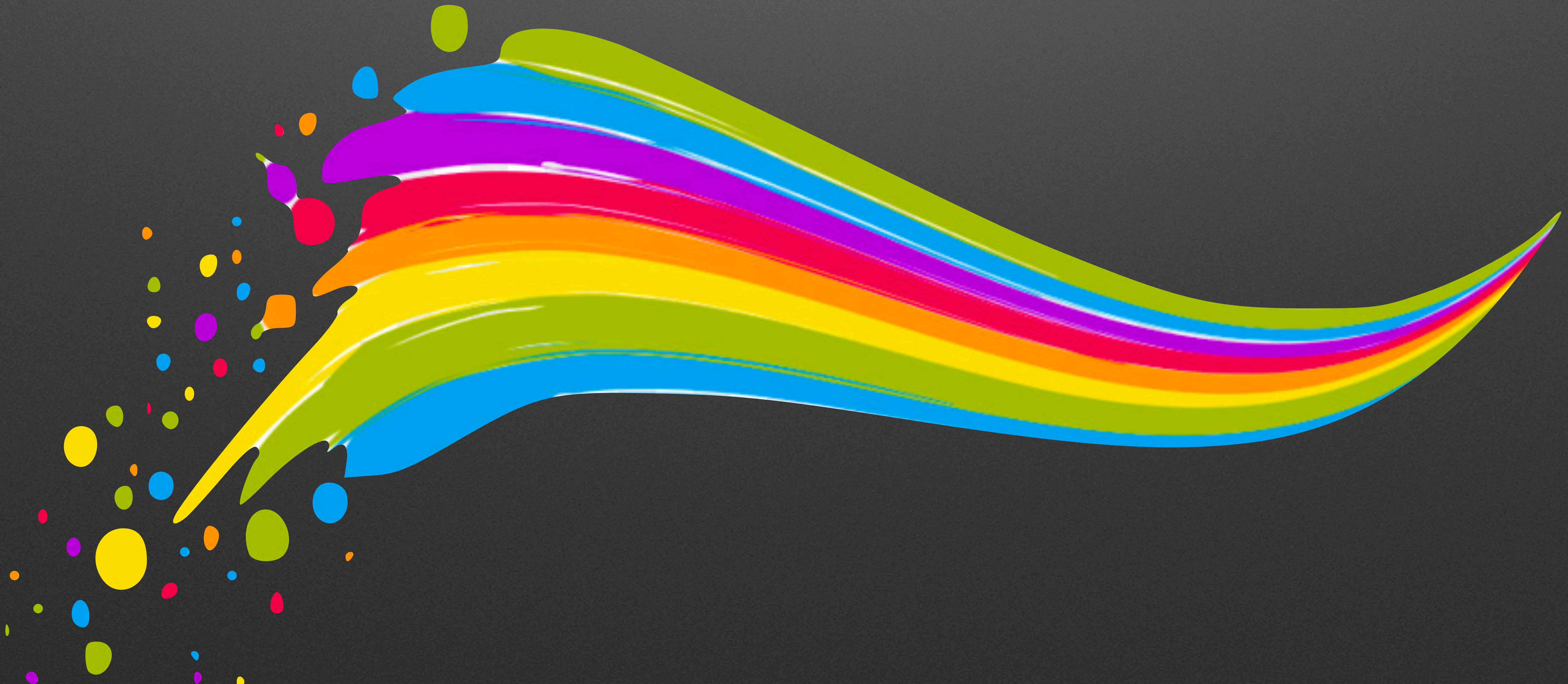
Prod(n): where your users are.

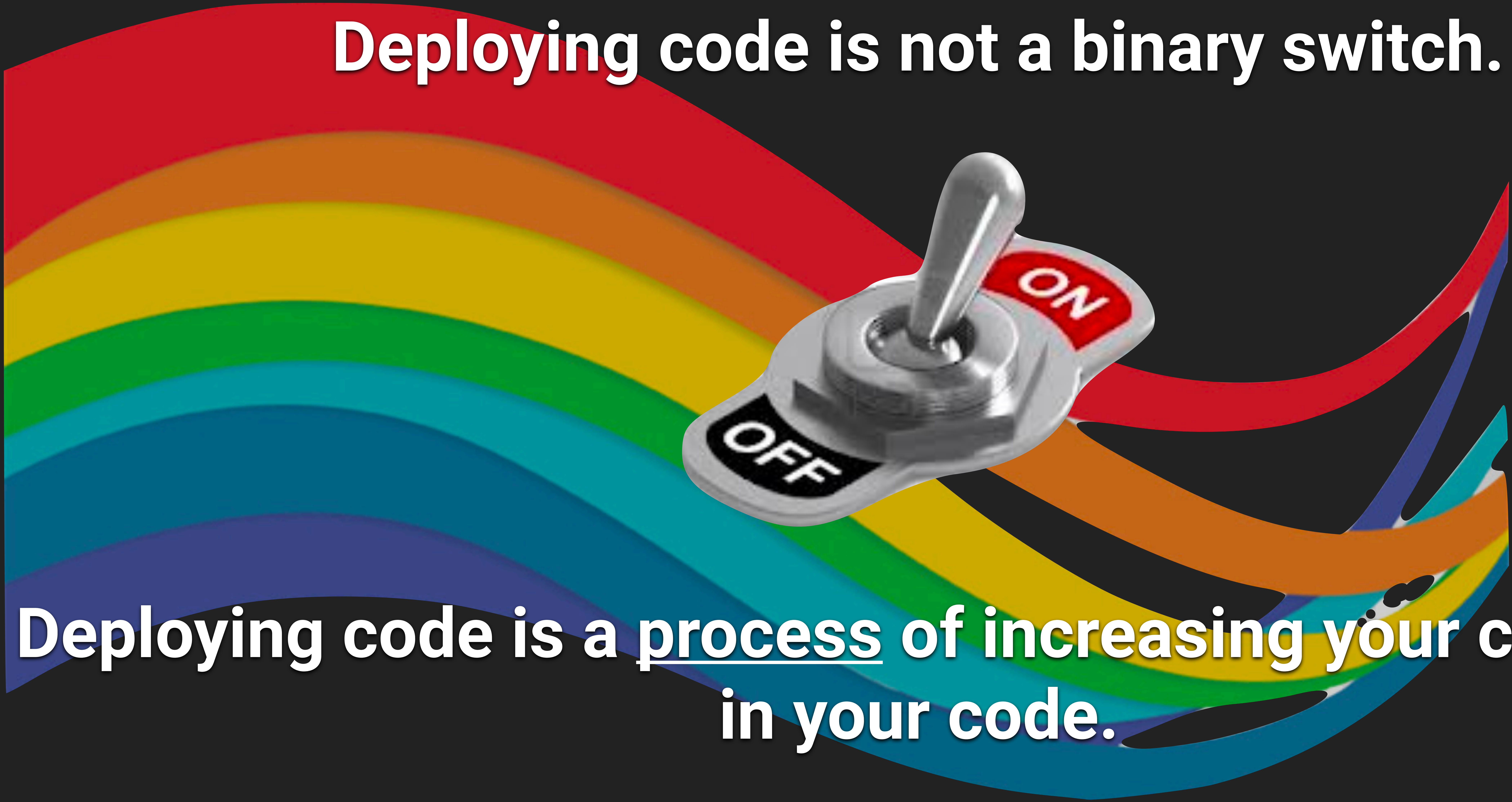
"Testing in production" should not be used as an **excuse** to skimp on testing or spend less.



I am here to tell you how to test *better*, not to help you half-ass it.

Our idea of what the software development lifecycle even looks like is overdue an upgrade in the era of distributed systems.





Deploying code is not a binary switch.

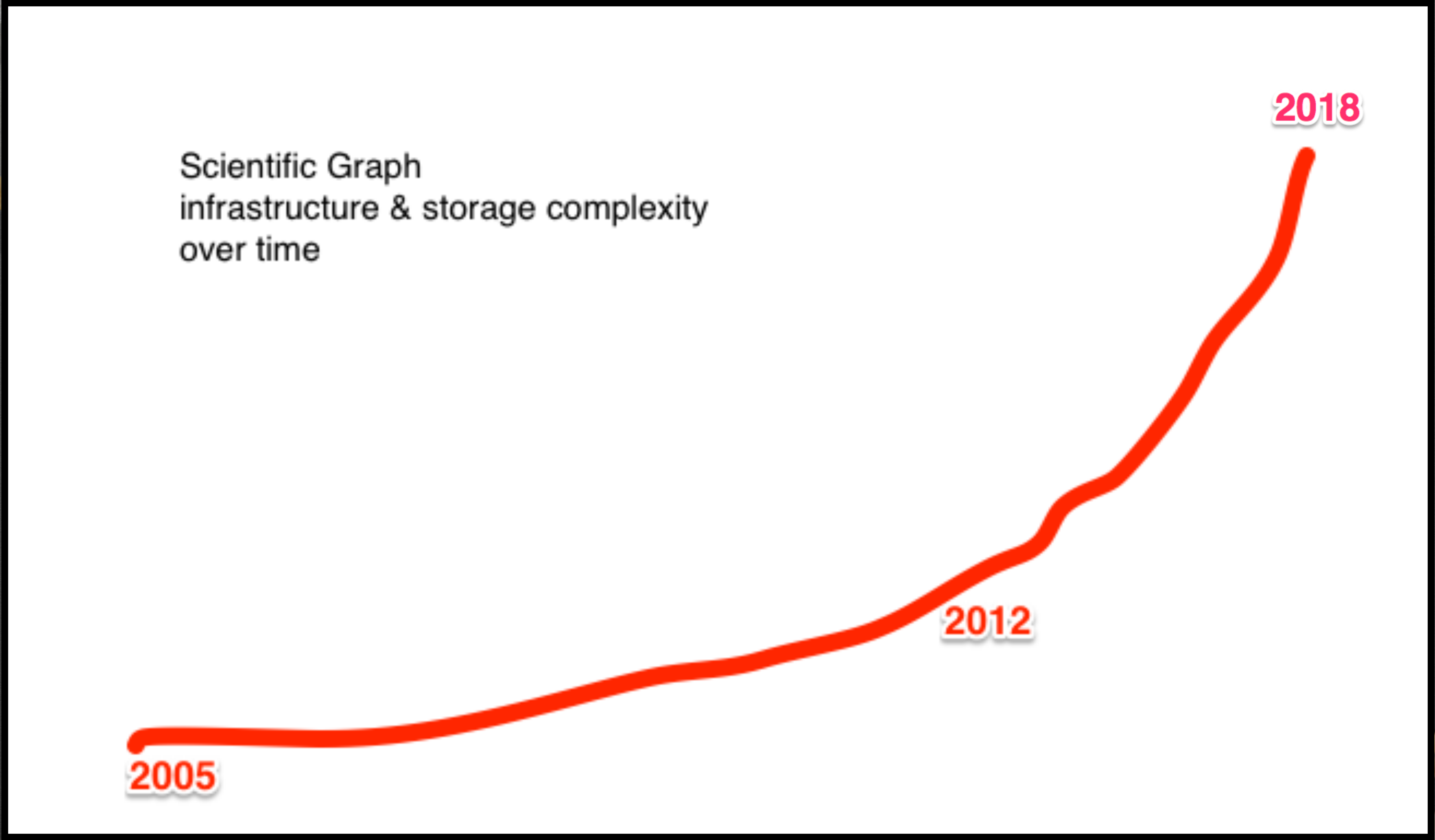
Deploying code is a <u>process</u> of increasing your confidence in your code.

Development
Production
SUCCESS
deploy
ON
OFF

SUCCESS
Development
Observability
Prod ction
VectorStock
ON
OFF

Development
Production
SUCCESS
ON
OFF
VectorStock

why now?

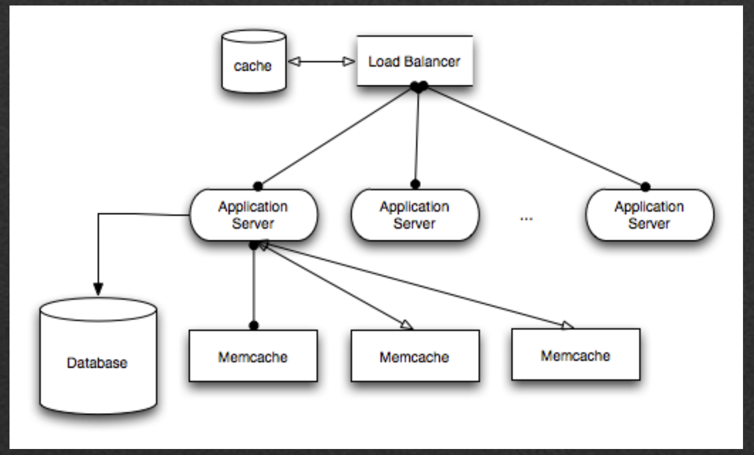


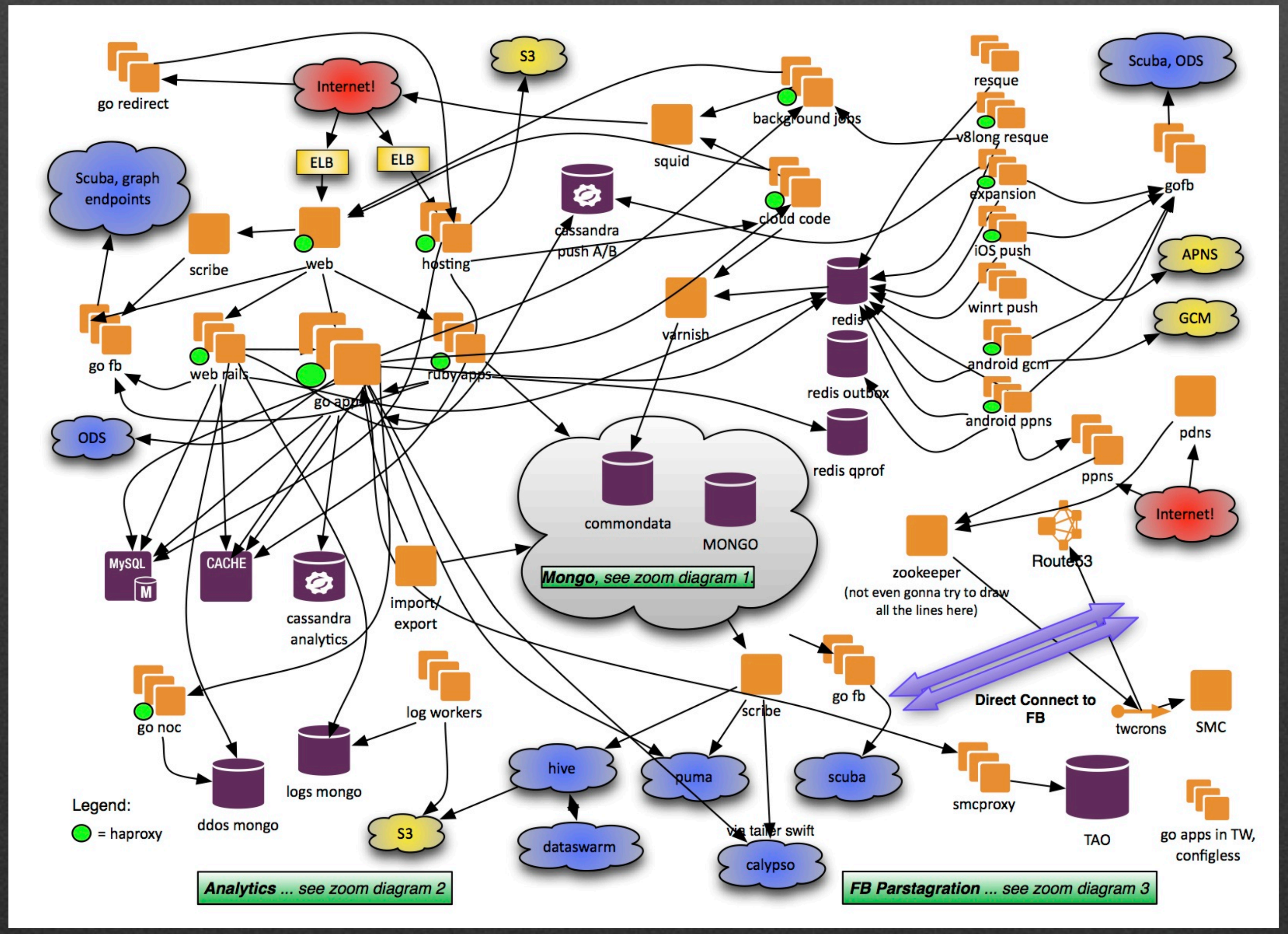
“Complexity is increasing” - Science

LAMP stack => distributed systems

monitoring => observability

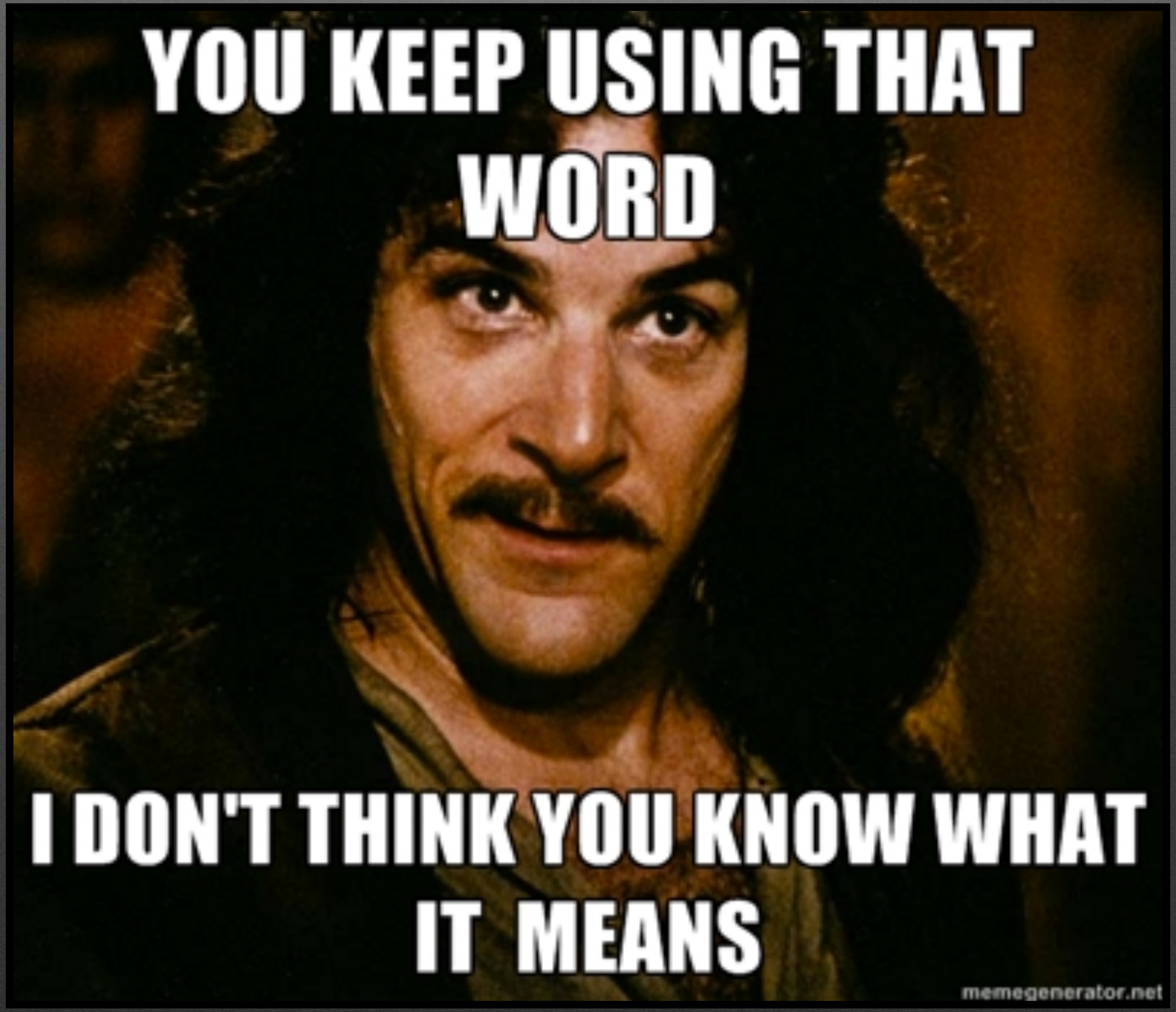
known unknowns => unknown unknowns





Your system is never entirely ‘up’

Many catastrophic states exist at any given time.



why does this matter more and more?

We are all distributed systems engineers now

the unknowns outstrip the knowns
and unknowns are untestable

Distributed systems are particularly hostile to being cloned or imitated (or monitored).

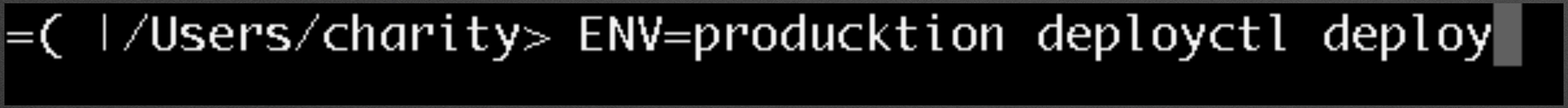
(clients, concurrency, chaotic traffic patterns, edge cases …)

Distributed systems have an infinitely long list of almost-impossible failure scenarios that make staging environments particularly worthless.

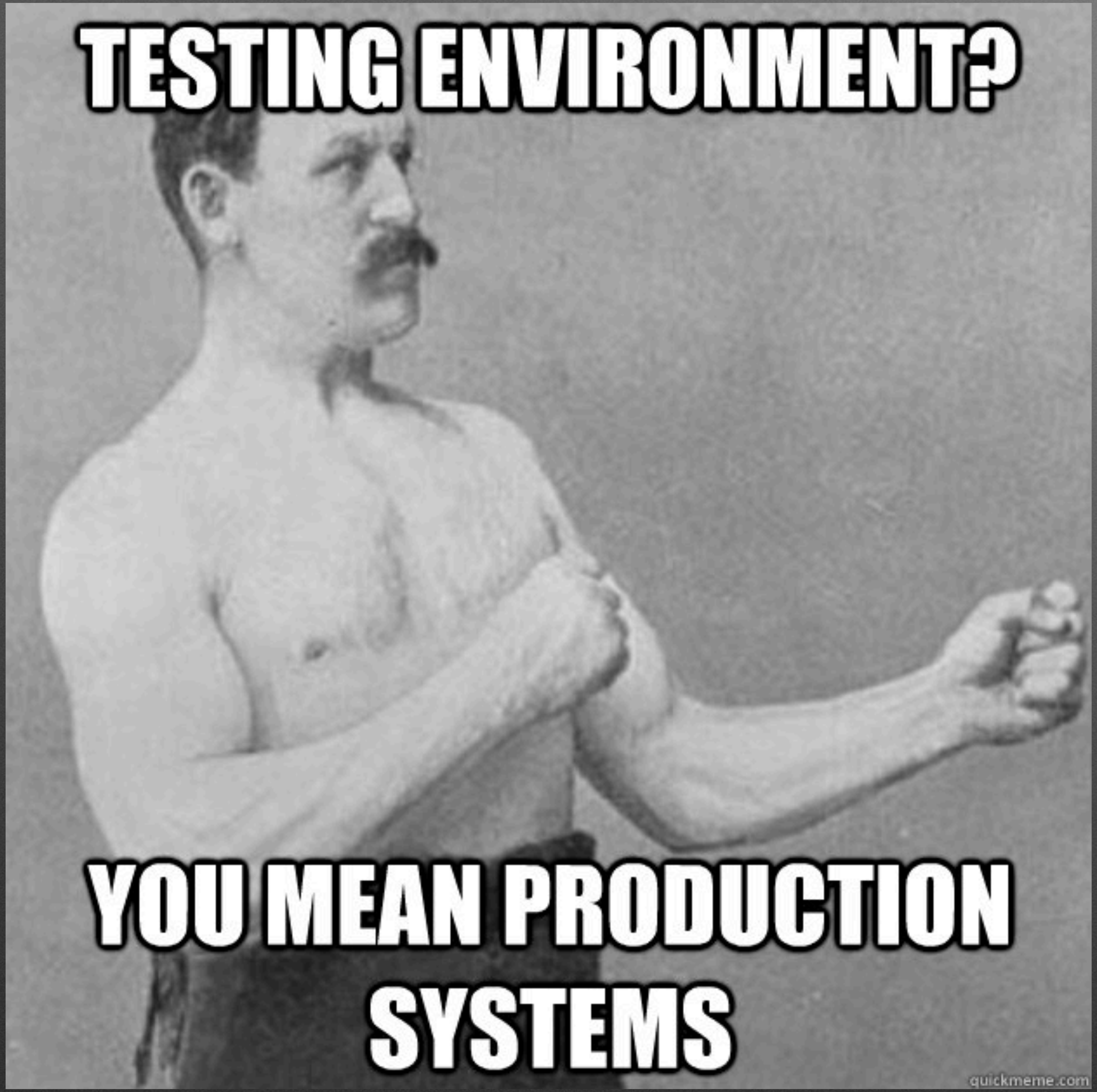
this is a black hole for engineering time

Only production is production.

You can ONLY verify the deploy for any env by deploying to that env

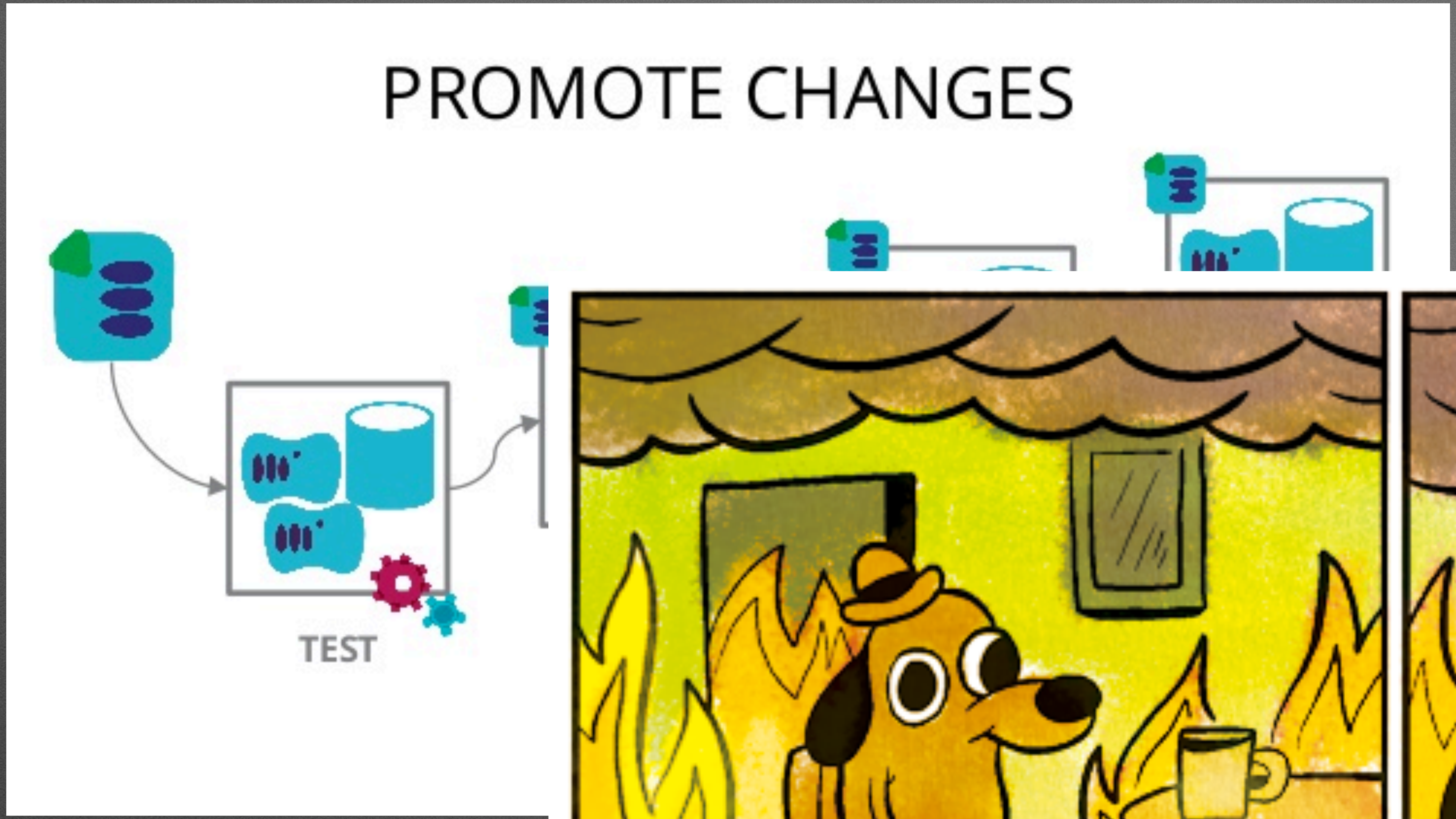


```
=( |/Users/charity> ENV=producktion deployctl deploy
```



1. Every deploy is a *unique* exercise of your process+ code+system

2. Deploy scripts are production code. If you’re using fabric or capistrano, this means you have fab/cap in production. 😳



Staging is not production.

Why do people sink so much time into staging,
when they can't even tell if their own
production environment is healthy or not?

You can catch 80% of the bugs with 20% of the effort. And you should.

That energy is better used elsewhere:

Production.

@caitie's PWL talk: https://youtu.be/-3tw2MYYT0Q

You need to watch your code run with:

Real data
Real users
Real traffic
Real scale
Real concurrency
Real network
Real deploys
Real unpredictabilities.

Staging != Prod

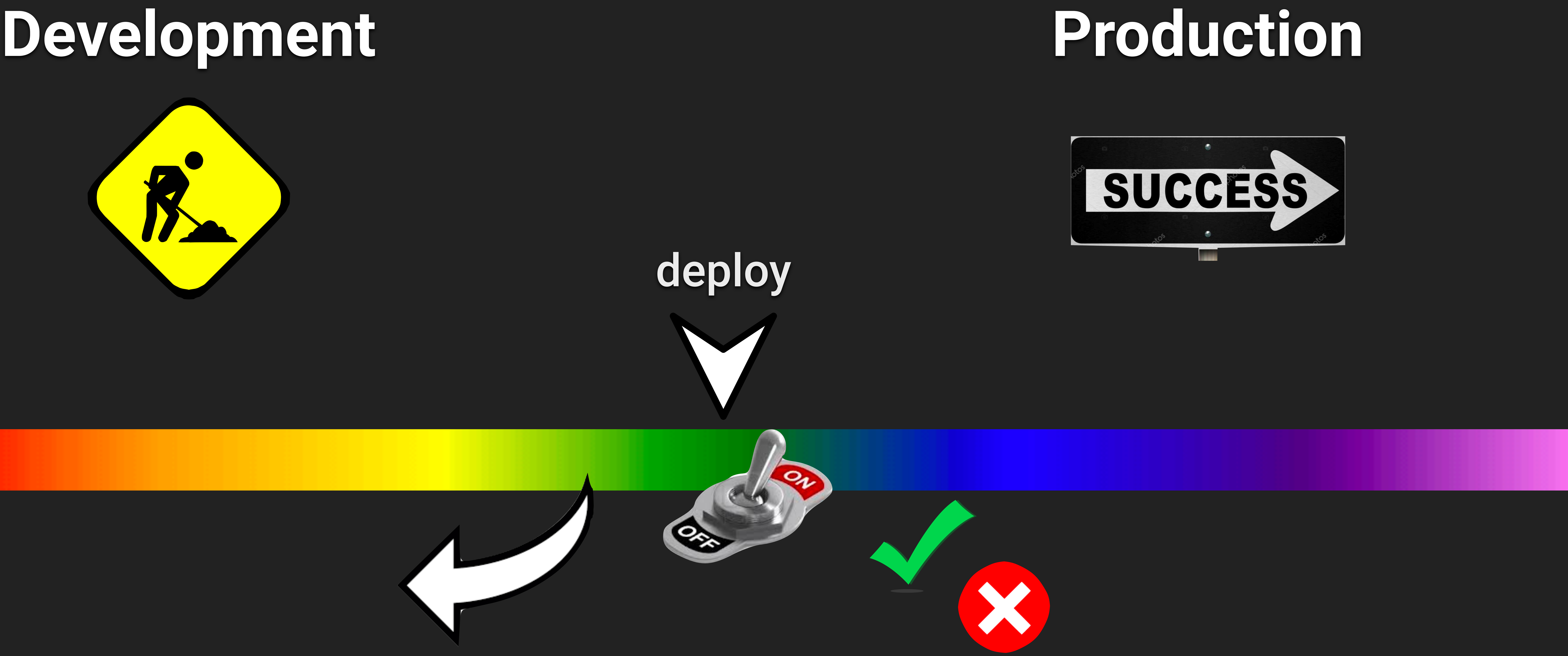
Environmental
differences

Security
of user data

Cost
of duplication

Time/Effort
(diminishing returns)

Uncertainty
of user patterns


Development
Production
SUCCESS
deploy
ON
OFF

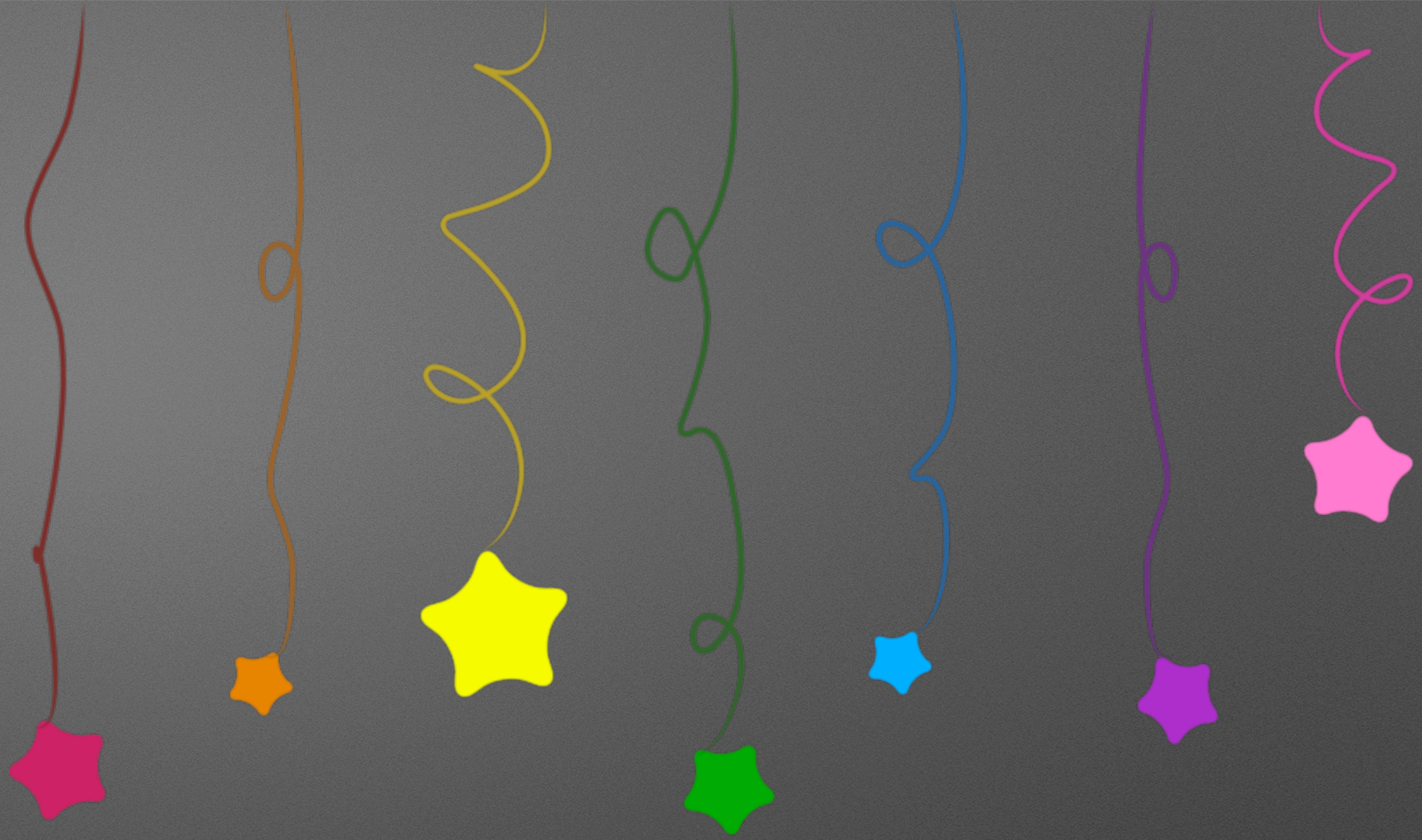
test <u>before</u> prod:

does it work
does my code run
does it fail in the ways i can predict
does it fail in the ways it has previously failed

test <u>in</u> prod:

behavioral tests
experiments
load tests (!!)
edge cases
canaries
weird bugs
data stuff
rolling deploys
multi-region

More reasons:



You are testing DR or chaos engineering
Beta programs where customers can try new features
Internal users get new things first
You have to test with production data
To lower the risk of deployments, you deploy more frequently
You need higher concurrency, etc to retro a bug

test <u>before</u> prod:

does it work
does my code run
does it fail in the ways i can predict
does it fail in the ways it has previously failed

Known unknowns

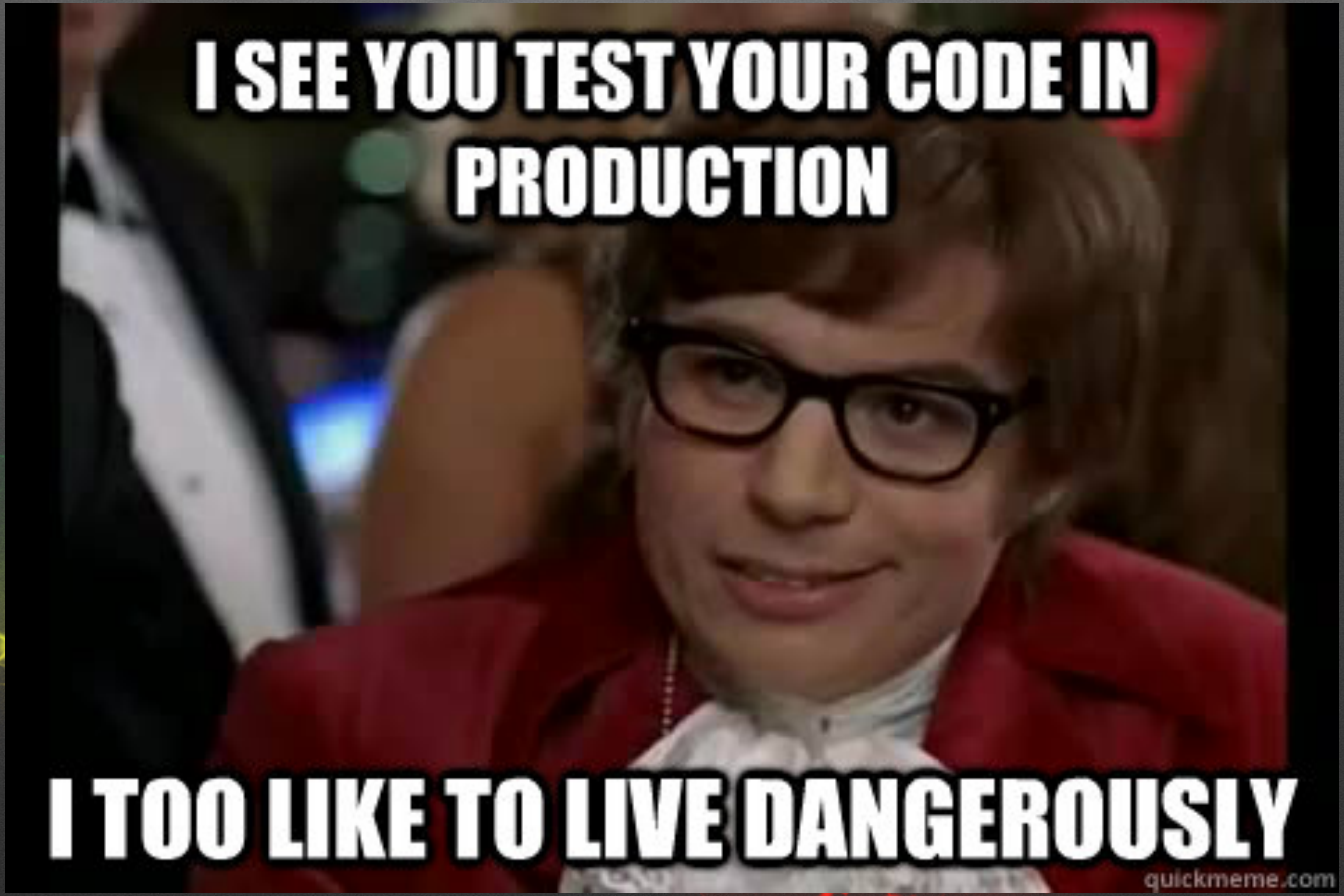
test <u>in</u> prod:

behavioral tests
experiments
load tests (!!)
edge cases
canaries
weird bugs
data stuff
rolling deploys
multi-region

Unknown unknowns (everything else)

test in staging?

meh


I SEE YOU TEST YOUR CODE IN PRODUCTION
I TOO LIKE TO LIVE DANGEROUSLY
quickmeme.com

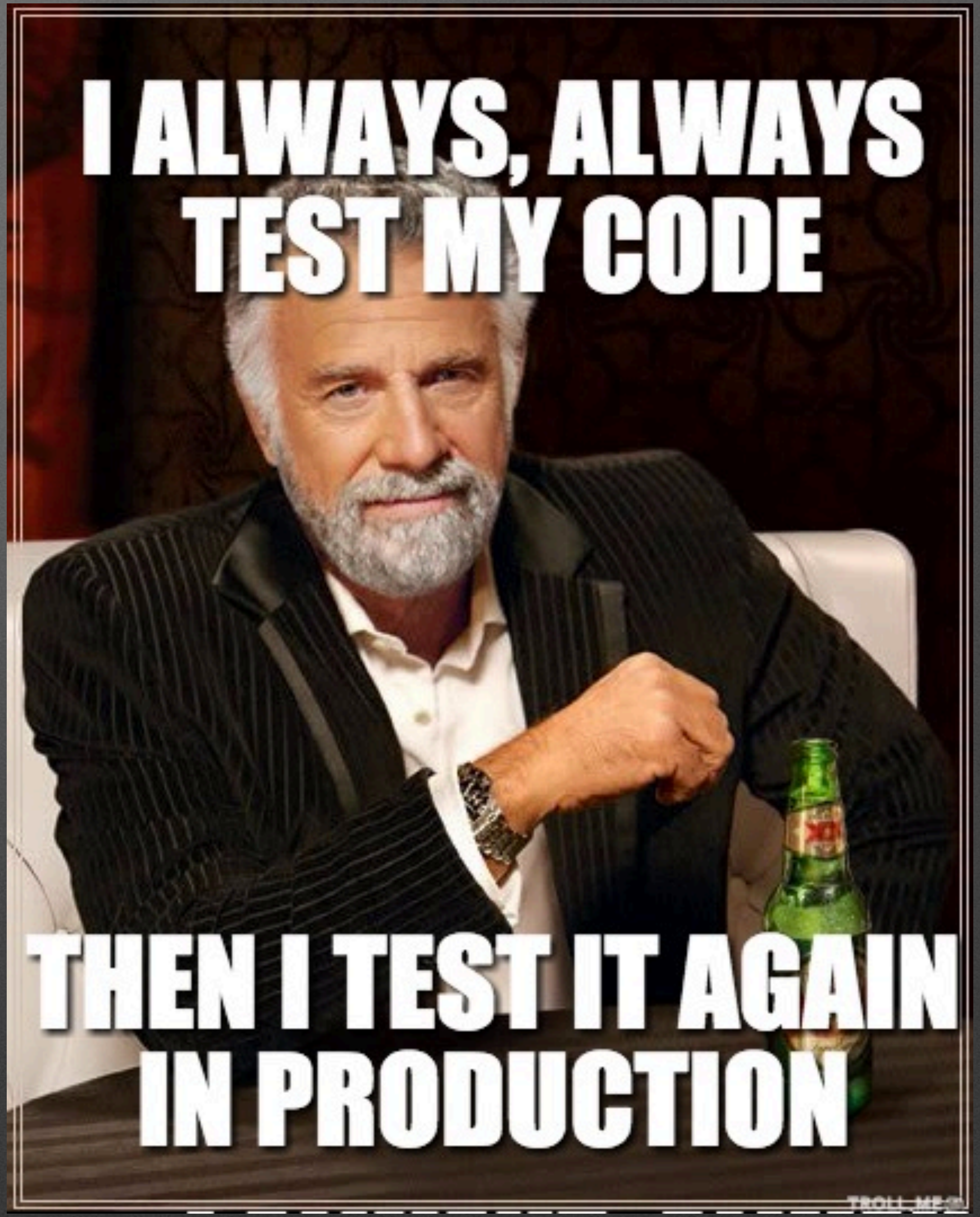
Risks:

Expose security vulnerabilities
Data loss or contamination
Cotenancy risks
The app may die
You might saturate a resource
No rollback if you make a permanent error
Chaos tends to cascade
May cause a user to have a bad experience

also build or use:

feature flags (launch darkly)
high cardinality tooling (honeycomb)
canary canary canaries,
shadow systems (goturbine, linkerd)
capture/replay for databases (apiary, percona)

plz dont build your own ffs


I ALWAYS, ALWAYS TEST MY CODE
THEN I TEST IT AGAIN IN PRODUCTION
TROLL.ME

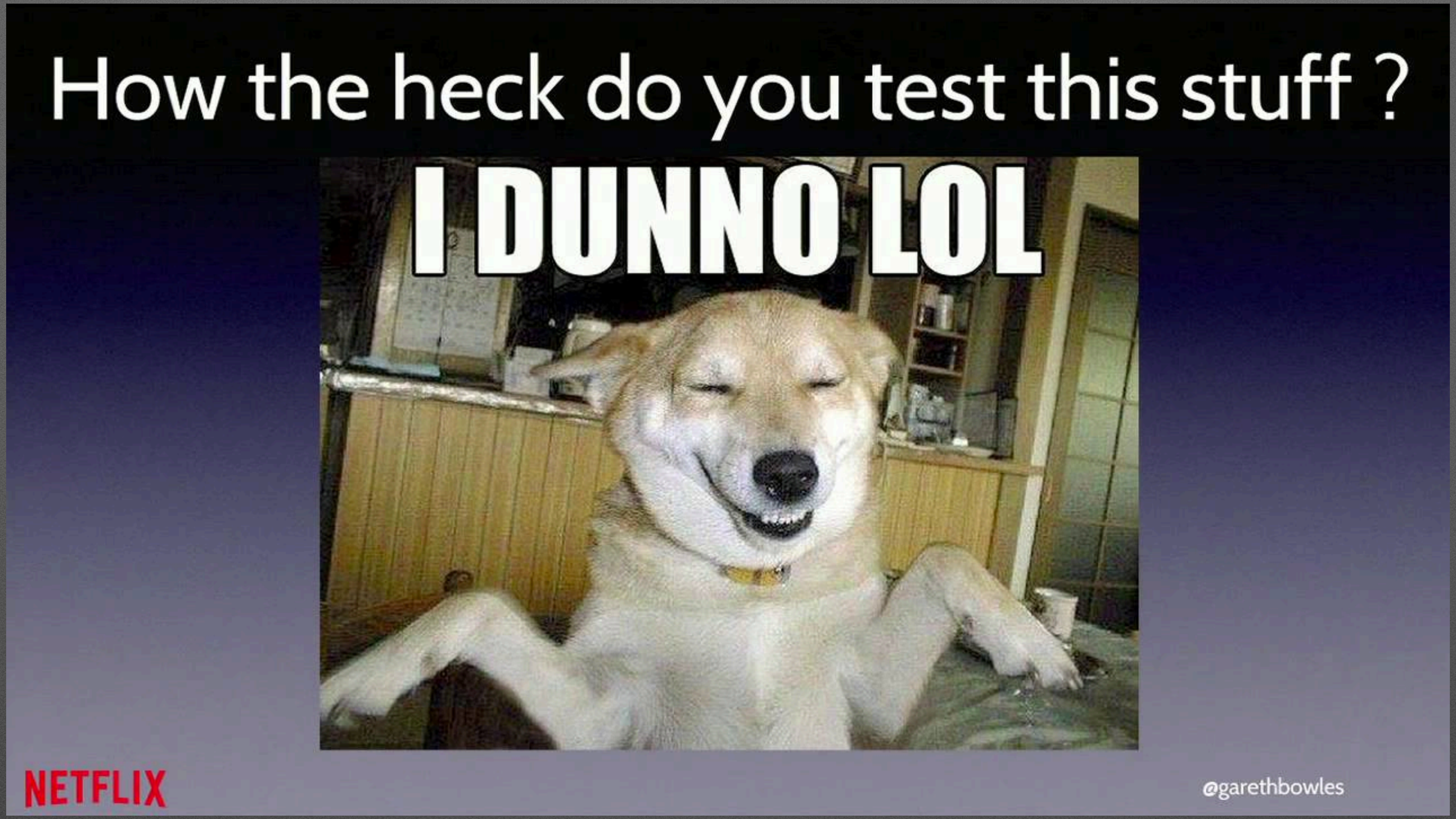
Be less afraid:

Feature flags
Robust isolation
Caps on dangerous behaviors
Auto scaling or orchestration
Query limits, auto throttling
Limits and alarms
Create test data with a clear naming convention
Separate credentials
Be extra wary of testing during peak load hours

Failure is not rare

Practice shipping and fixing lots of small problems

And practice on your users!!



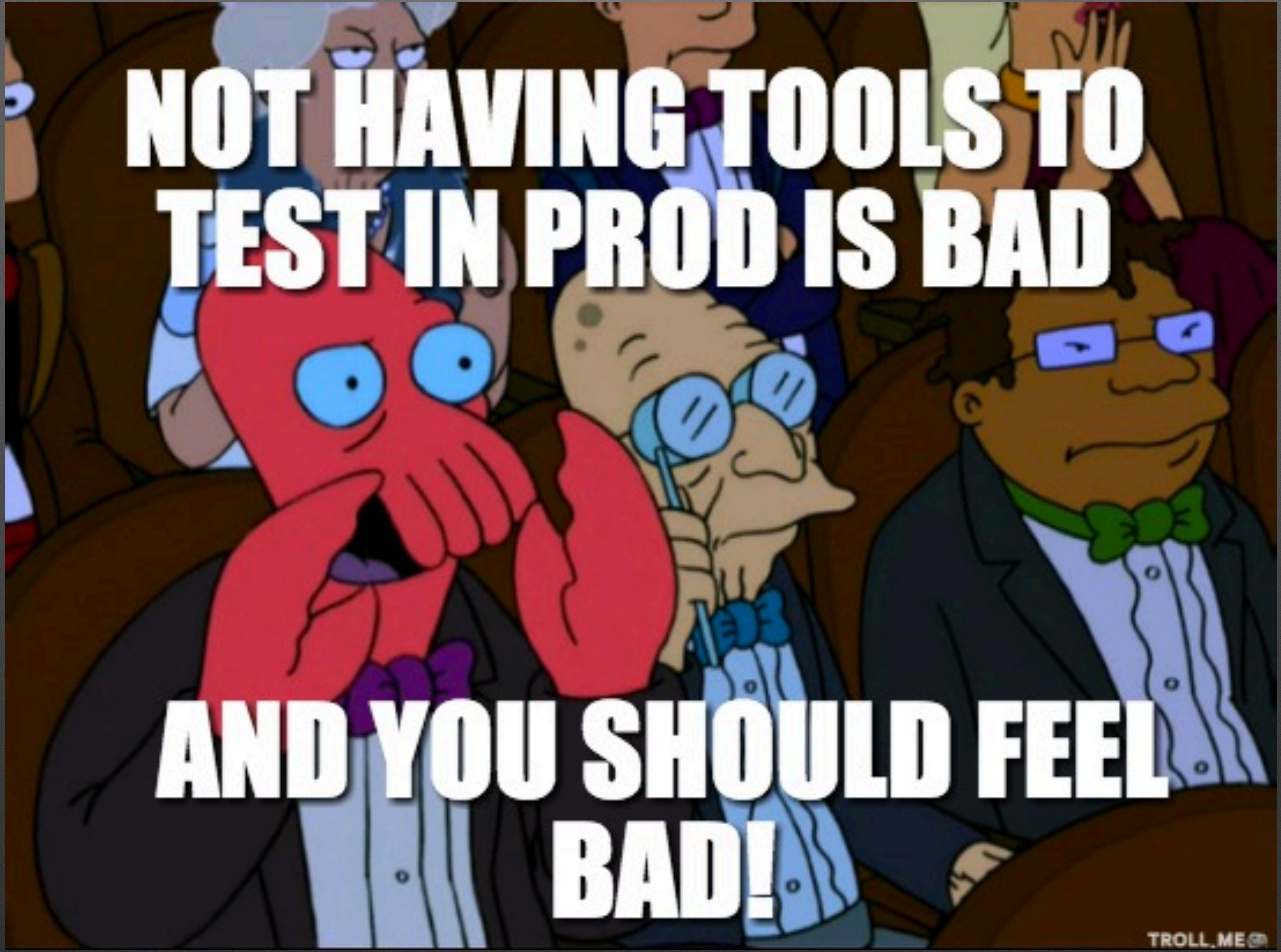
Failure: it’s “when”, not “if”

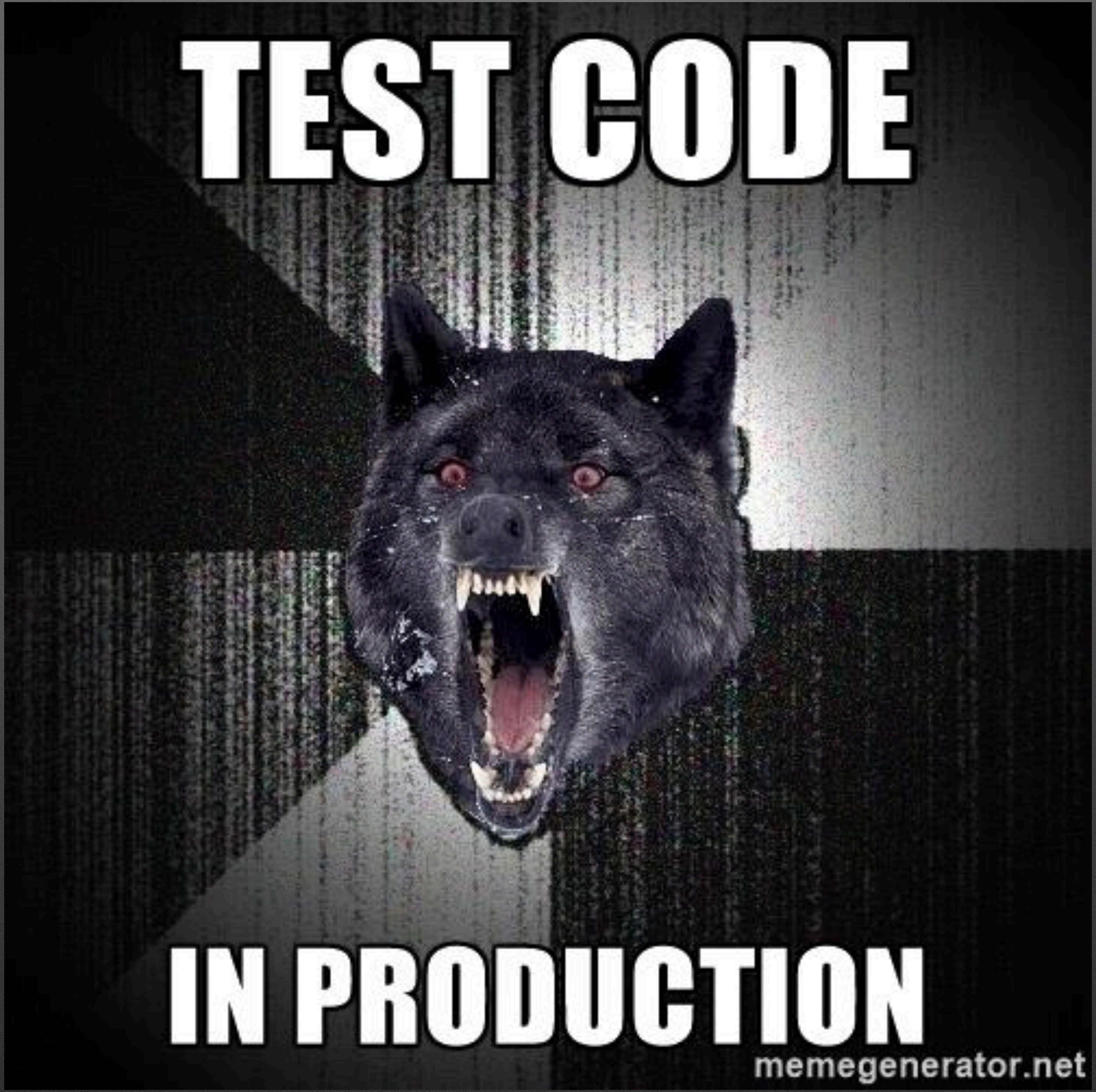
(lots and lots and lots of “when’s”)

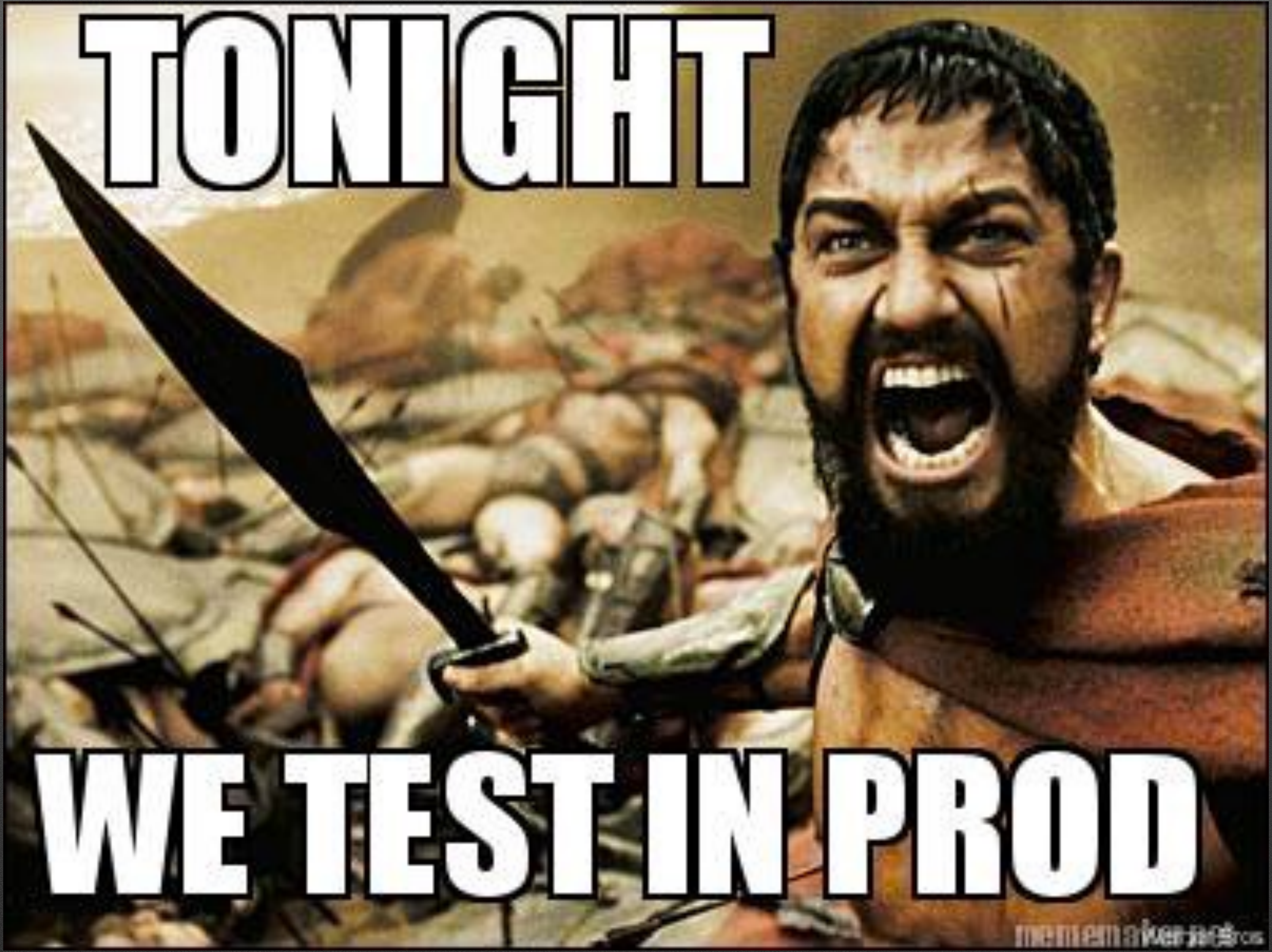
Does everyone ...

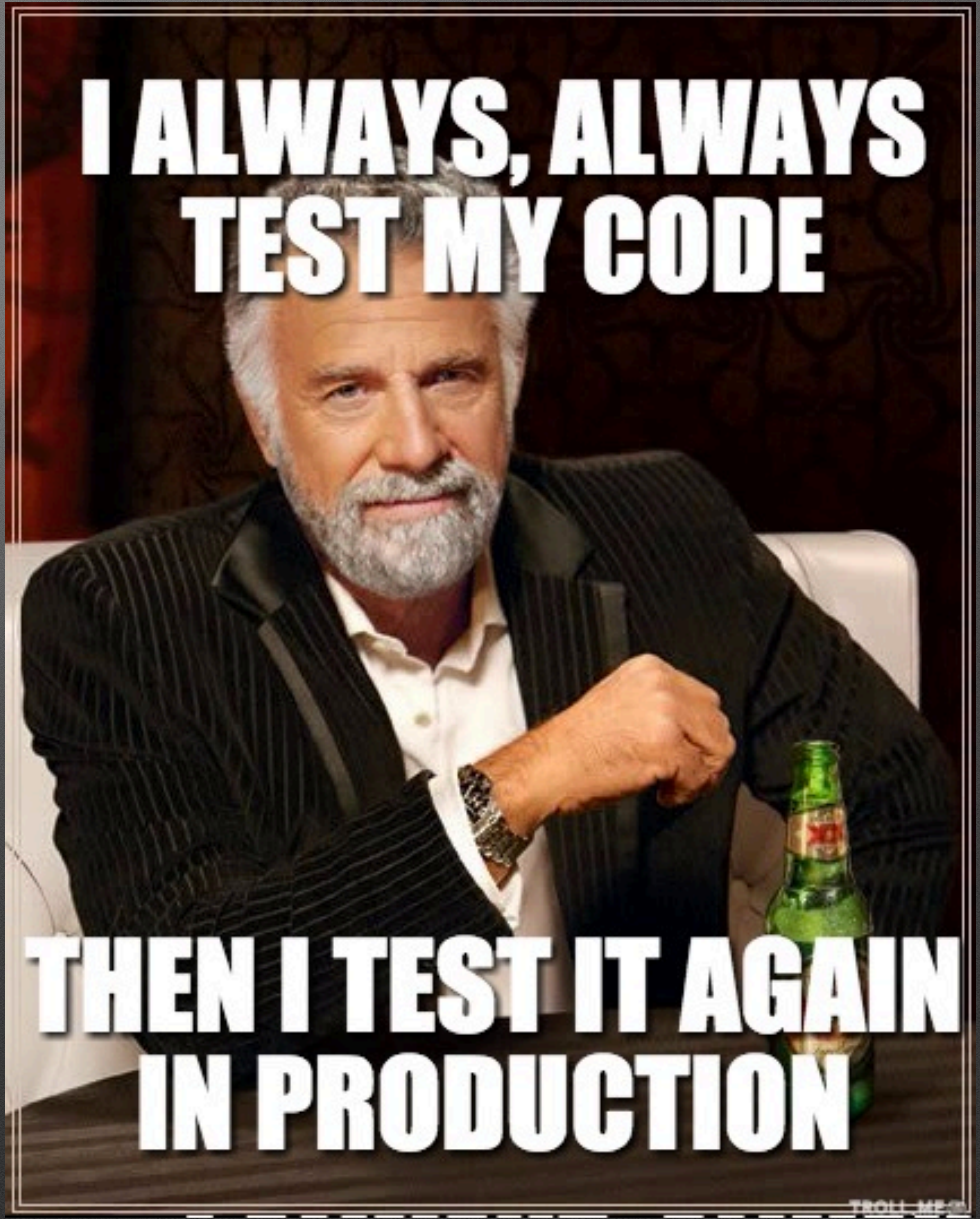
know what normal looks like?
know how to deploy?
know how to roll back?
know how to canary?
know how to debug in production?

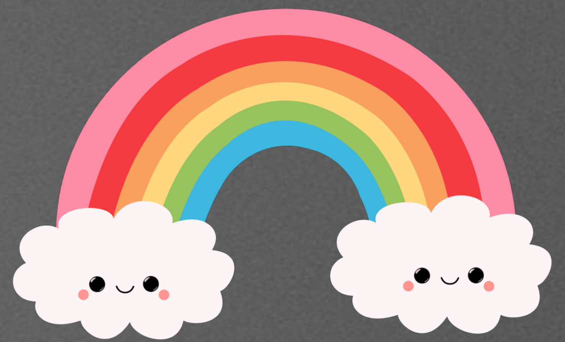
Practice!!~


NOT HAVING TOOLS TO TEST IN PROD IS BAD
AND YOU SHOULD FEEL BAD!
TROLL.ME


TEST CODE
IN PRODUCTION
memegenerator.net


TONIGHT
WE TEST IN PROD

I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO I DO IT IN PRODUCTION
memegenerator.net

I ALWAYS, ALWAYS TEST MY CODE
THEN I TEST IT AGAIN IN PRODUCTION



Charity Majors
@mipsytipsy