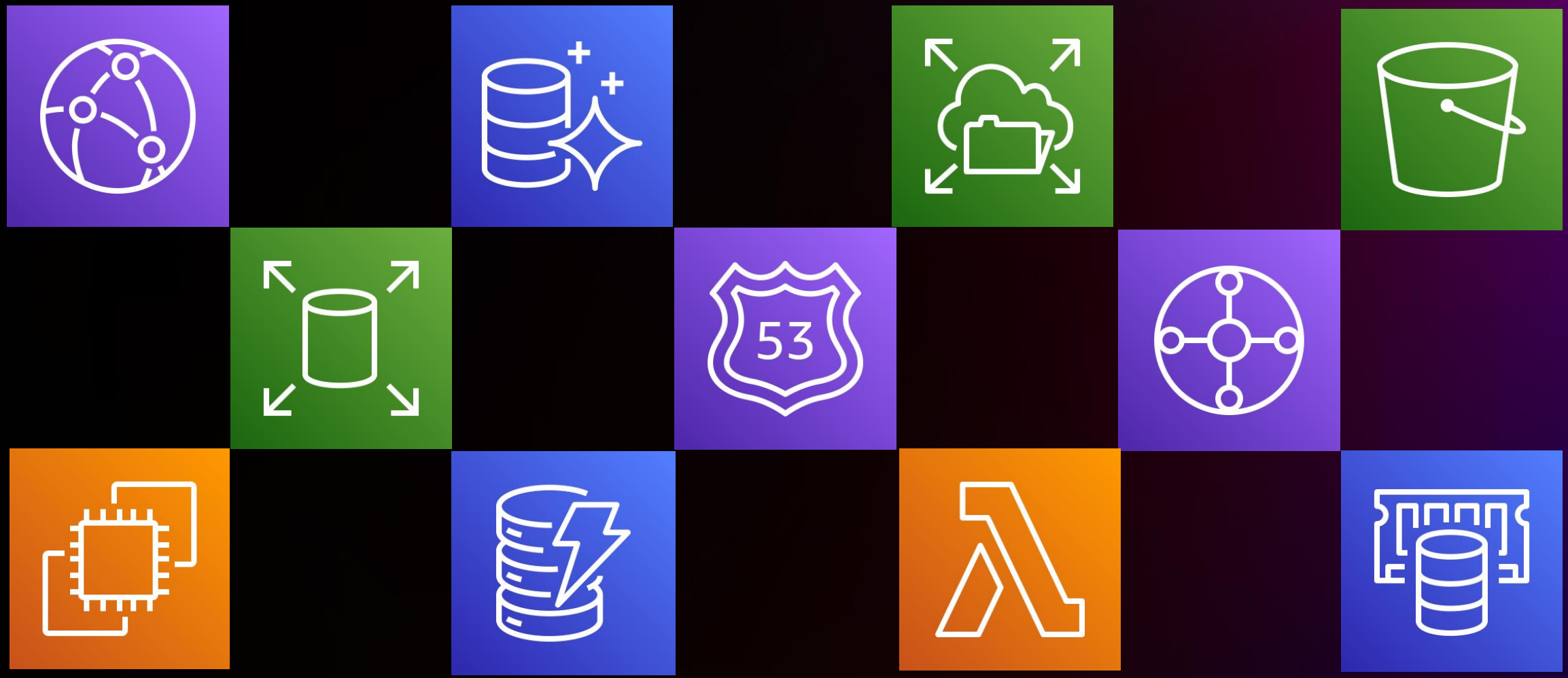

AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

ARC310

Beyond five 9s: Lessons from our highest available data planes

Colm MacCárthaigh
VP/Distinguished Engineer
Amazon Web Services

What is a data plane?



The Amazon Builders' Library

- First-hand articles from engineers who built AWS services:
  - Caching
  - Retries
  - Fairness
  - Safety
  - More …

https://aws.amazon.com/builders-library/

Beyond five 9s

Updated

Rethinking the nines model

Simplicity and survival of the fittest

Testing as time travel

Building technical fearlessness

Beyond five 9s

Updated

Compartmentalization and blast radius

Constant-work and minimizing change

Operational safety

Beyond five 9s

Updated

Rethinking the nines model

Compartmentalization and blast radius

Simplicity and survival of the fittest

Constant-work and minimizing change

Testing as time travel

Operational safety

Technical fearlessness

Rethinking the nines model

The traditional nines model

- 99% available = 3.65 days per year of impact
- 99.9% available = 8.77 hours per year of impact
- 99.99% available = 52.6 minutes per year of impact
- 99.999% available = 5.26 minutes per year of impact

The traditional nines model

Great for measuring overall business impact and value

Low-fidelity measurement

Not so great for systems design

99.9

99%

Some problems with the nines model

What about partitioned and cellular systems?

How do you pick a good time duration? Modern services can go years without an issue

99.9

99%

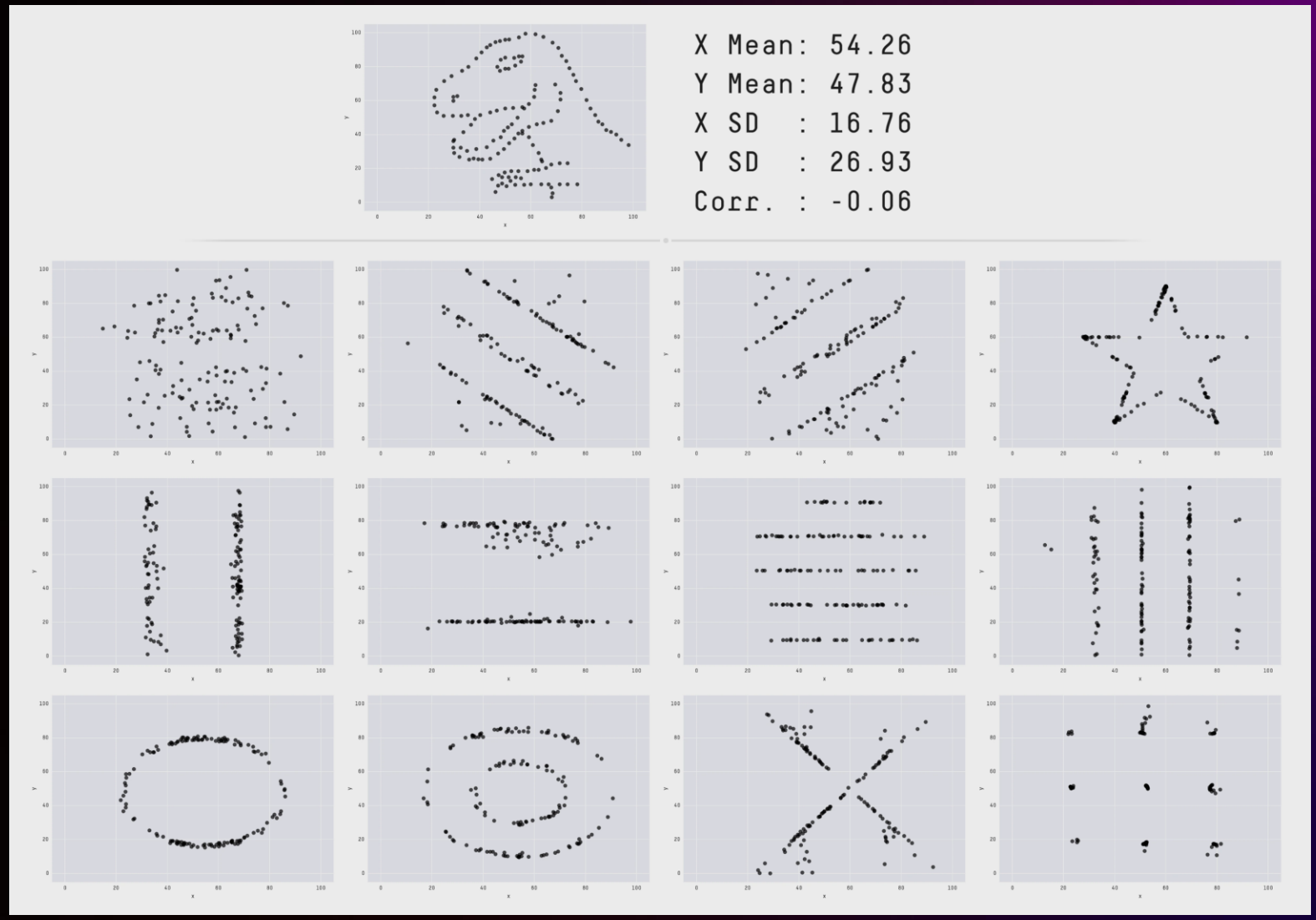
Beyond five 9s

“Never trust summary statistics alone; always visualize your data”

- Alberto Cairo

The Datasaurus Dozen

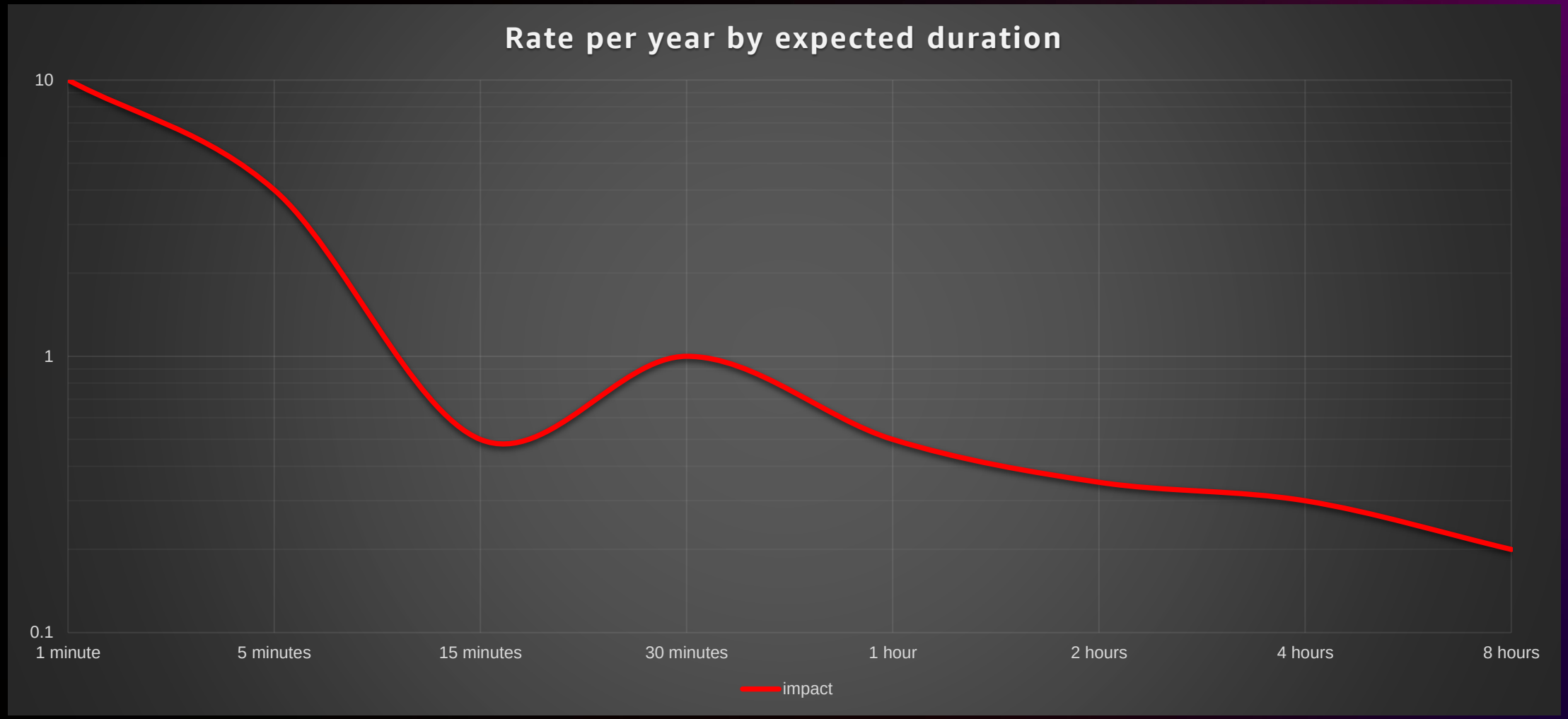
- Justin Matejka, George Fitzmaurice



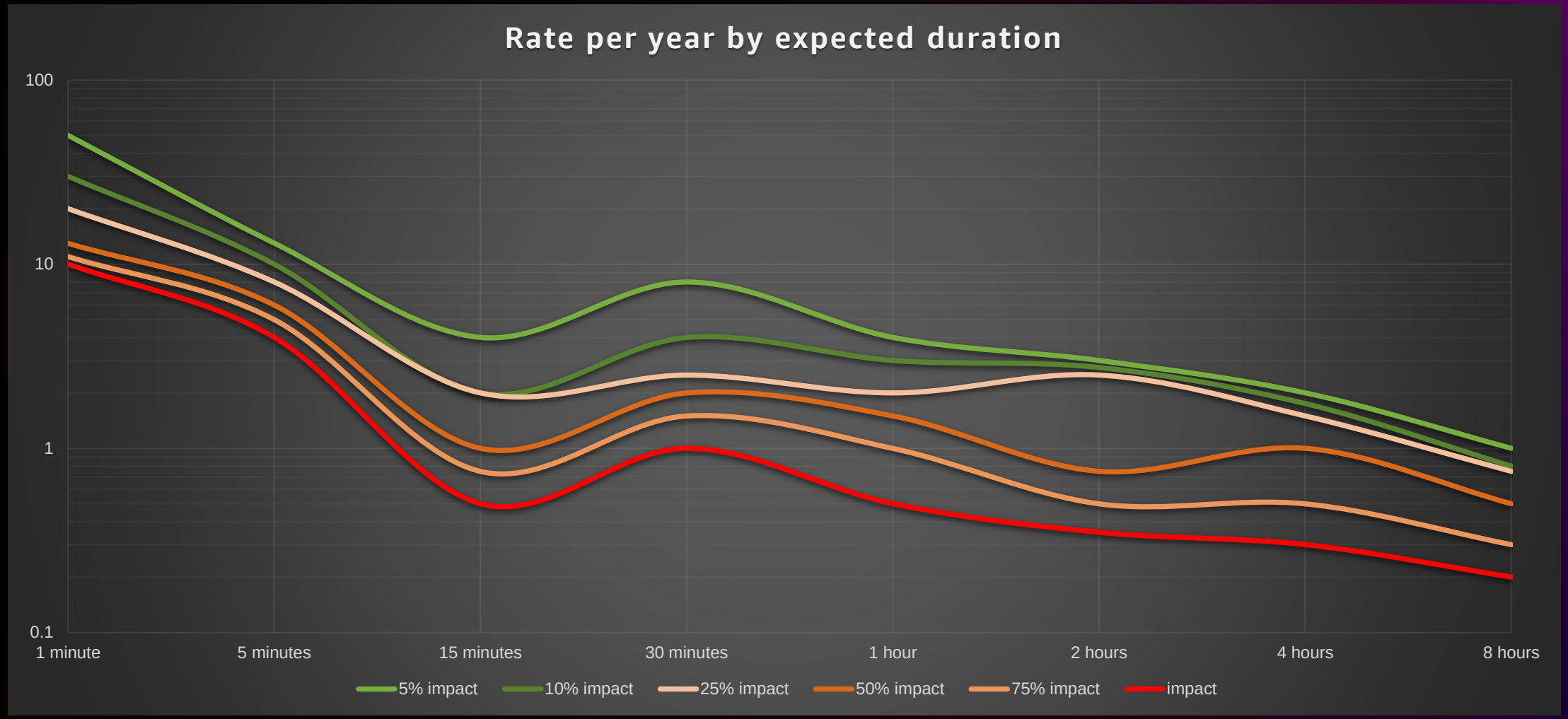
What really matters

- How long could an interruption last?
- Recovery Time Objective (RTO)
- How often could it happen?
- What is the **R**ate and **E**xpected **D**uration? (RED)

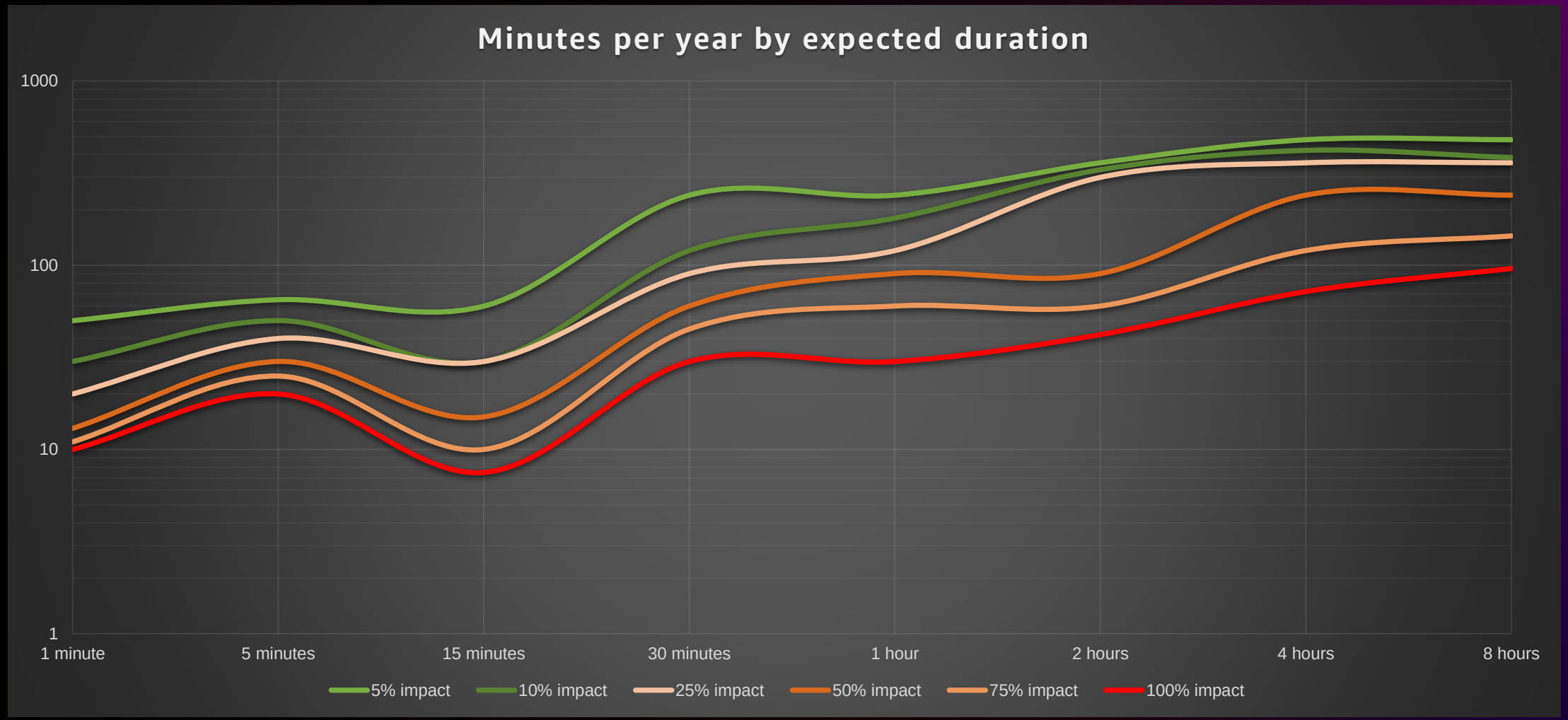
What really matters



What really matters



What really matters



Rate and expected duration

- Simpler to talk about and to reason about
- Uncovers the “lumpiness”
- Focuses on what kind of resilience plans are needed
- Highlights where operational processes matter

Rate and expected duration

- Cloud services
- On-premises databases
- On-premises appliances
- Low-level physical infrastructure

Rate and expected duration

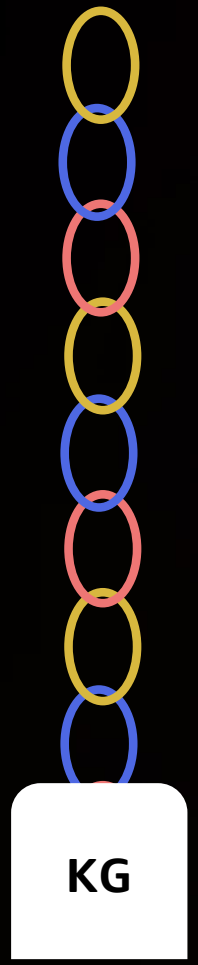
- Measuring historical data is the gold standard
- But even guesses are useful
- Just asking the question can be clarifying
- Feynman’s Appendix F

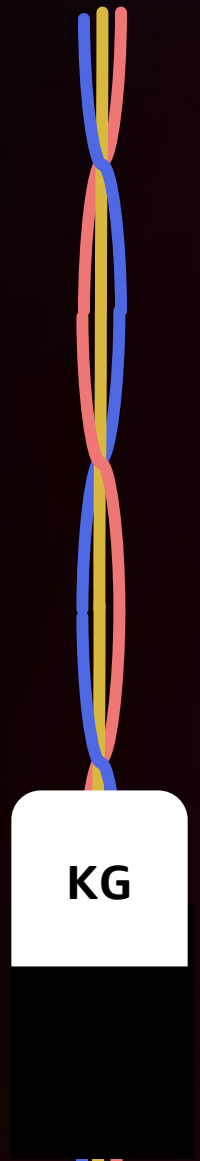
What else is wrong with nines?

What else is wrong with nines?

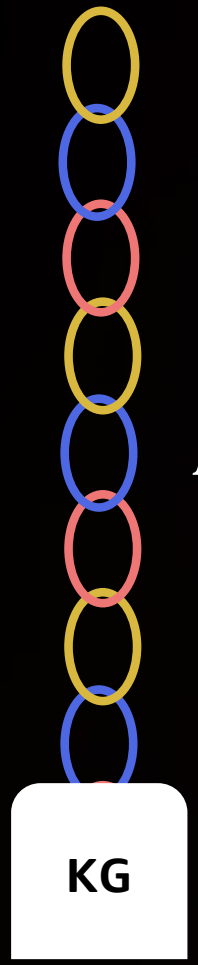
- The nines model comes from mechanical and civil engineering
- Failure mode and effects analysis (FMEA)
- Physical components have different failure characteristics, and usually no recovery

The traditional nines model

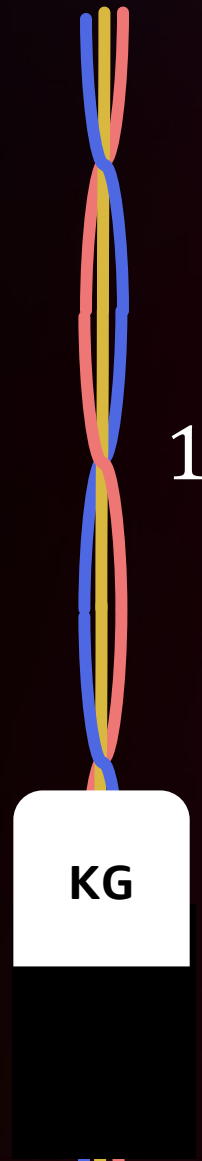




The traditional nines model

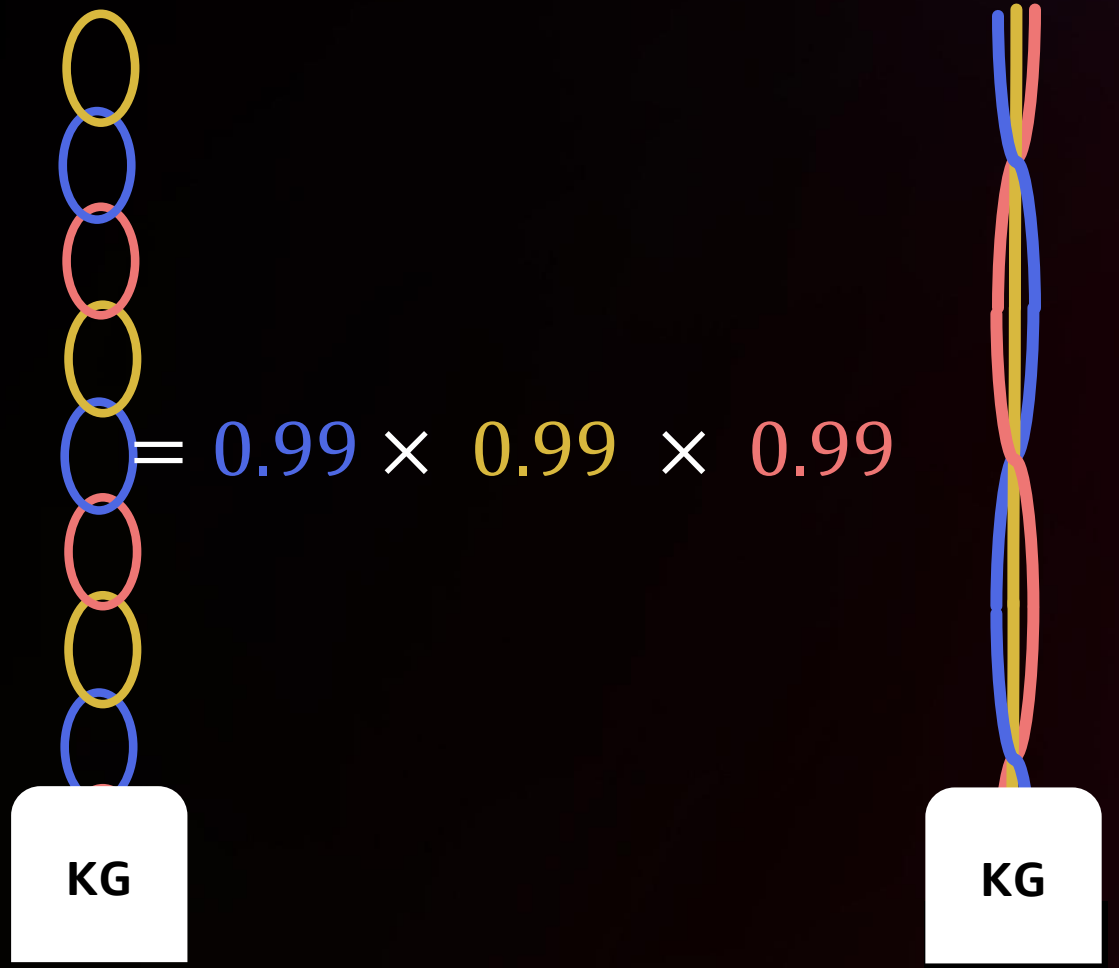


$$N_1 \times N_2 \times N_2 \ldots$$



$$1 - (1 - N_1) \times (1 - N_2) \ldots$$

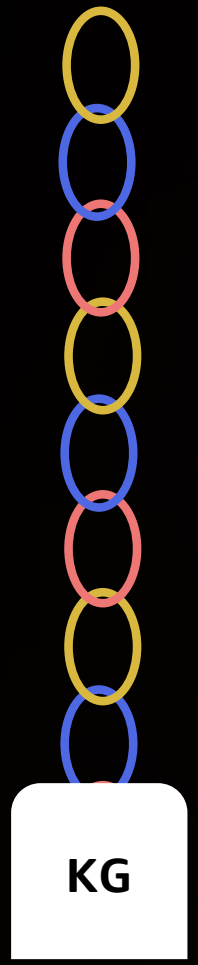
The traditional nines model



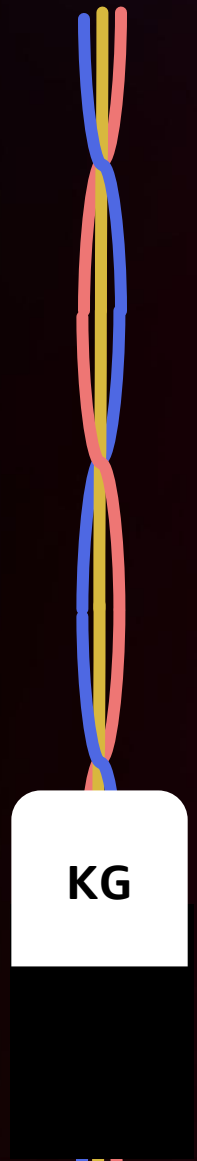
$= 0.99 \times 0.99 \times 0.99$

$$1 - ((1 - 0.99) \times (1 - 0.99) \times (1 - 0.99))$$

The traditional nines model



0.97

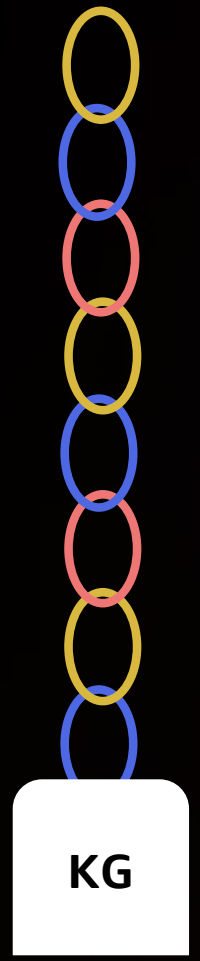


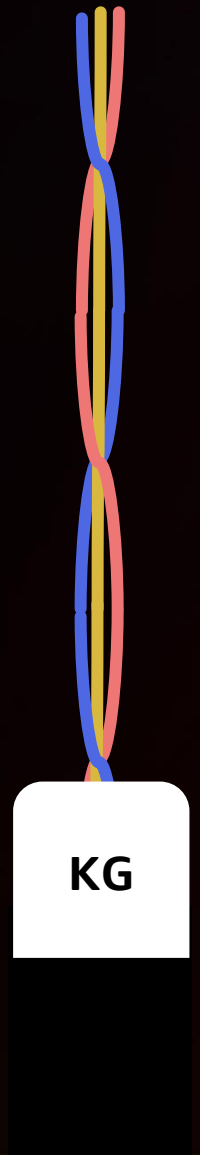
0.999999

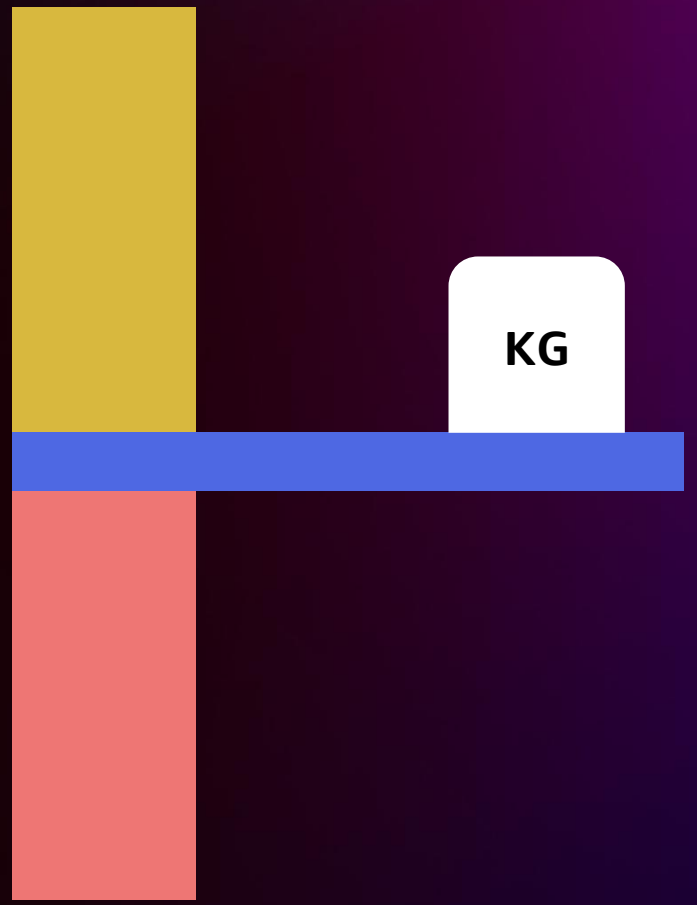
Case study: Using multiple DNS providers

Other DNS provider

Chains, braids, and cantilevers







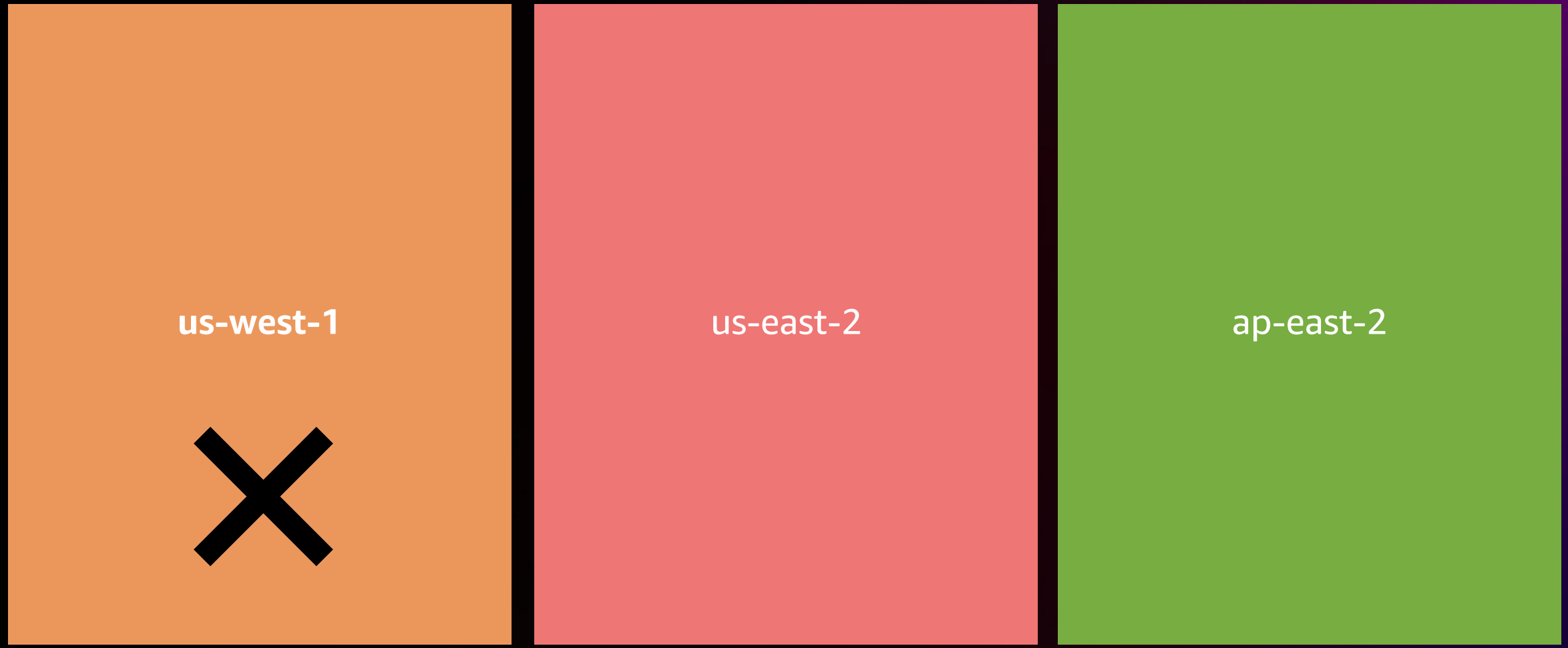
Takeaways so far

- Focus on the rate and estimated duration of issues
- Whole system “end-to-end” testing is critical
- Whole lifecycle “birth-to-rebirth” testing is critical

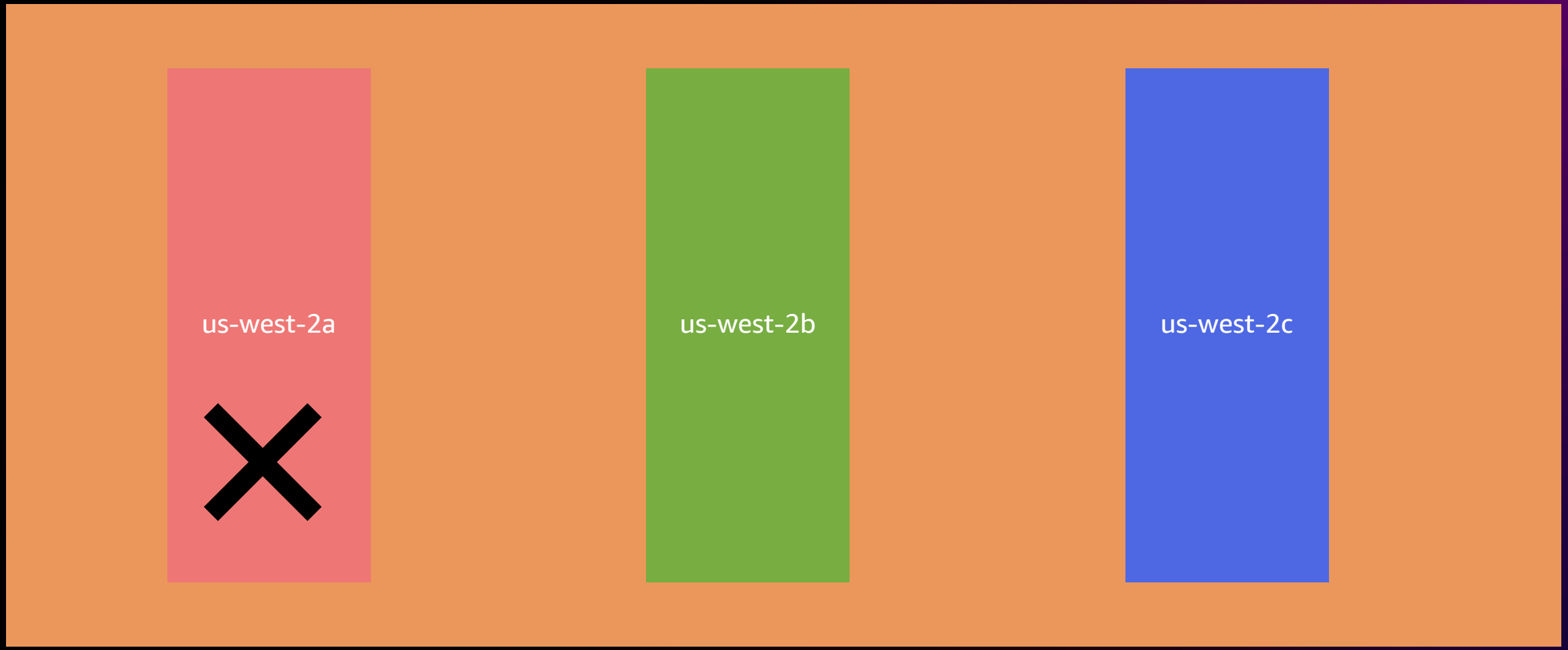
Compartmentalization and blast radius

Regional isolation

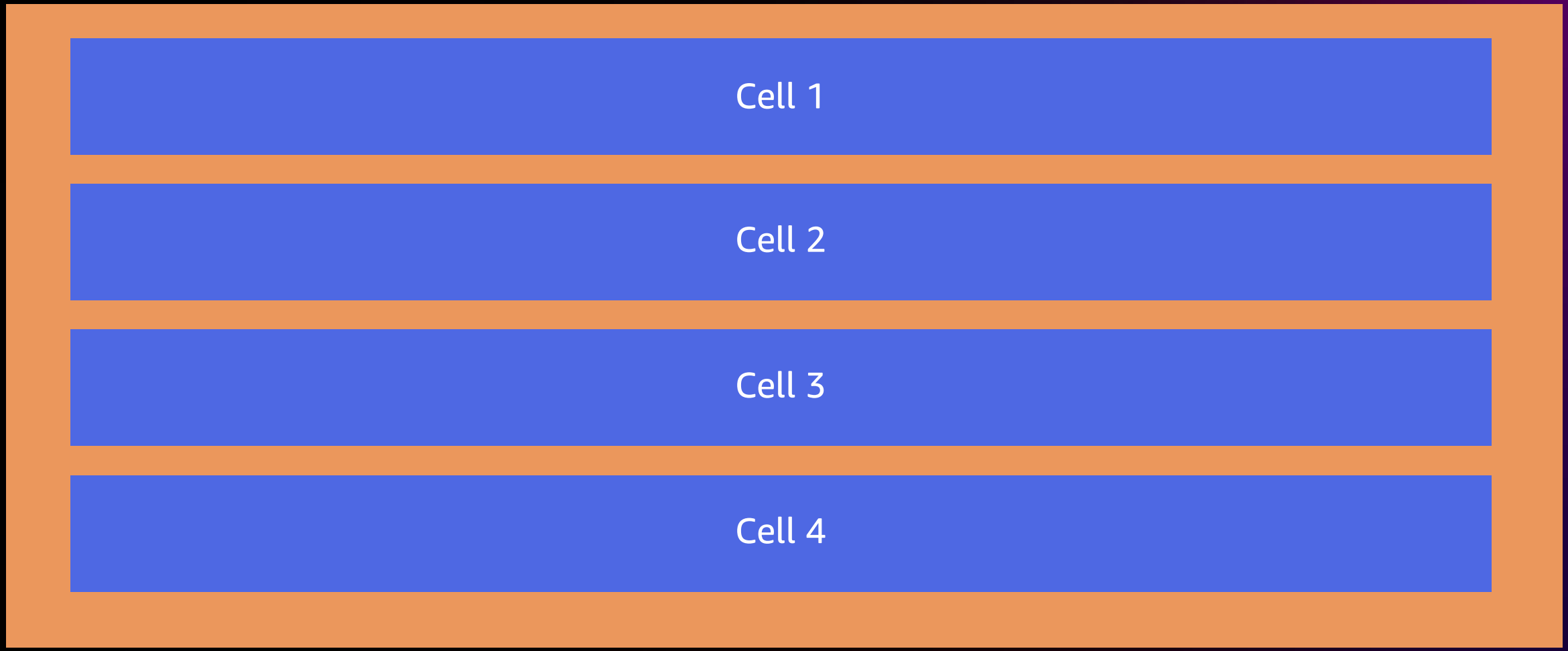
Regional isolation



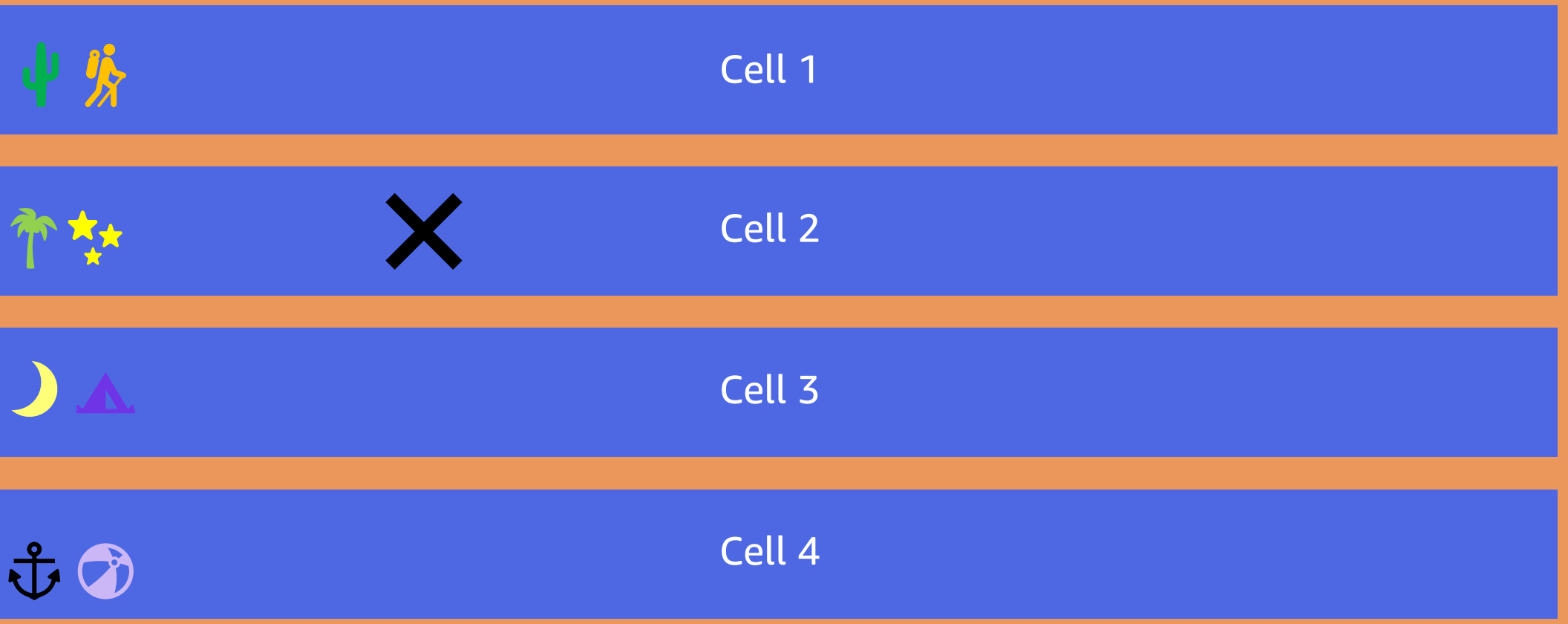
Zonal isolation



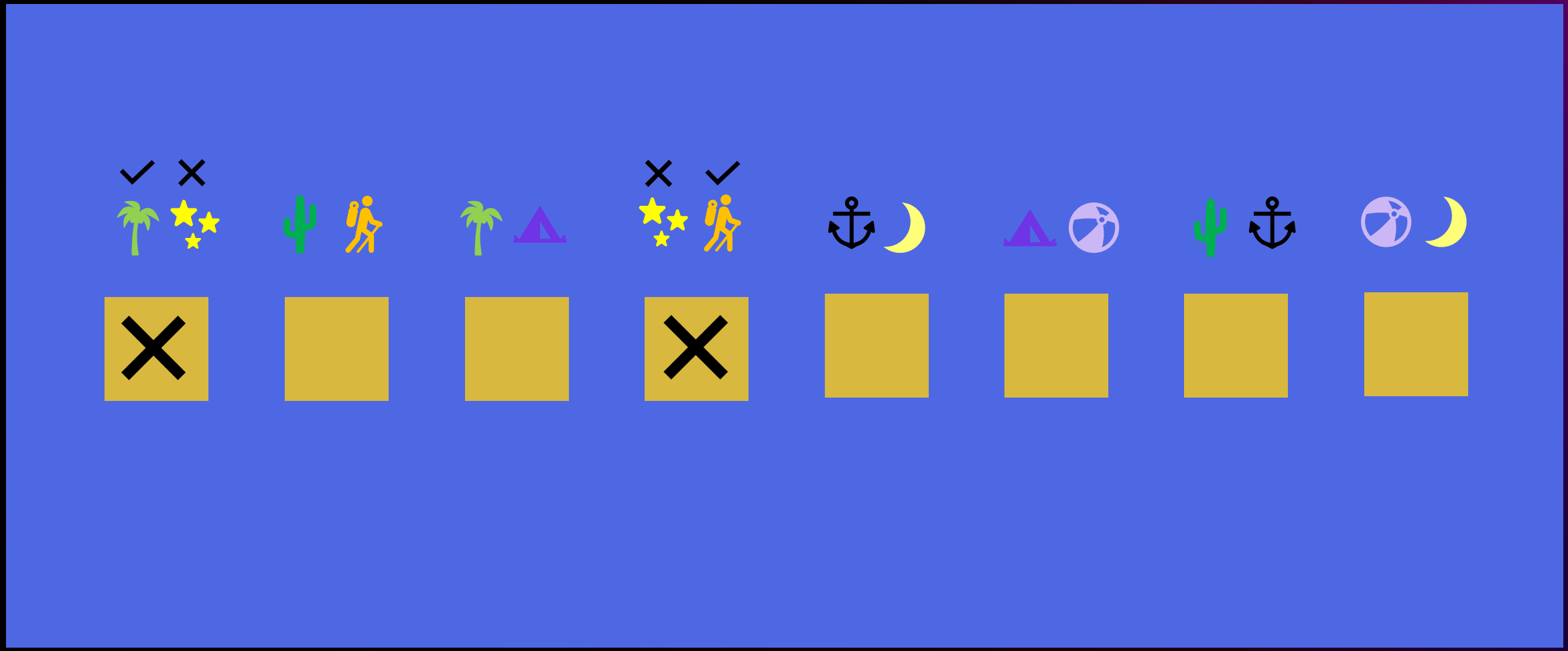
Cellular isolation



Cellular Isolation

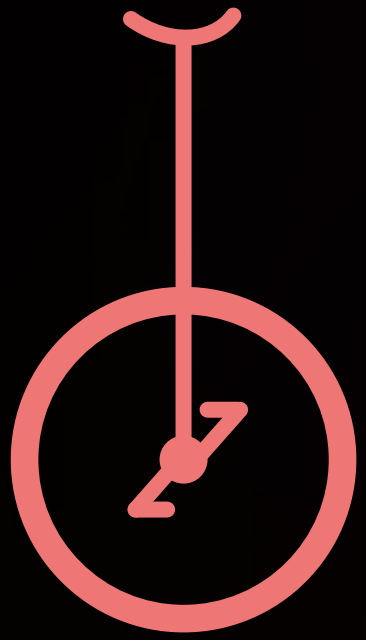


Shuffle sharding



Simplicity and survival of the fittest

Which is simple?

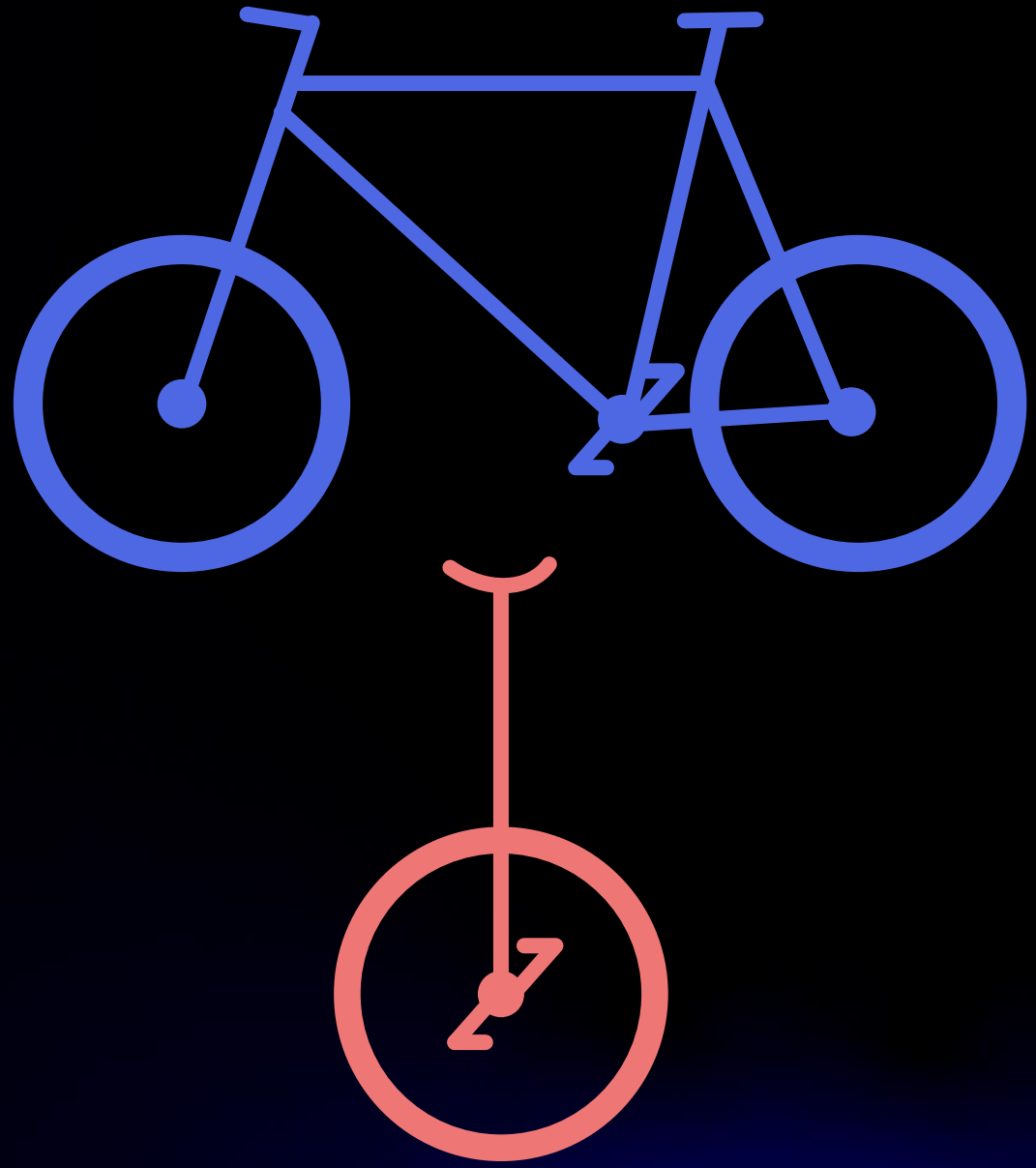


Which is simple?

It’s not the “fewest moving parts,” it’s more like “fewest needed parts”

Simplicity takes into account all of the trade-offs a design must make

Most important factor is usability

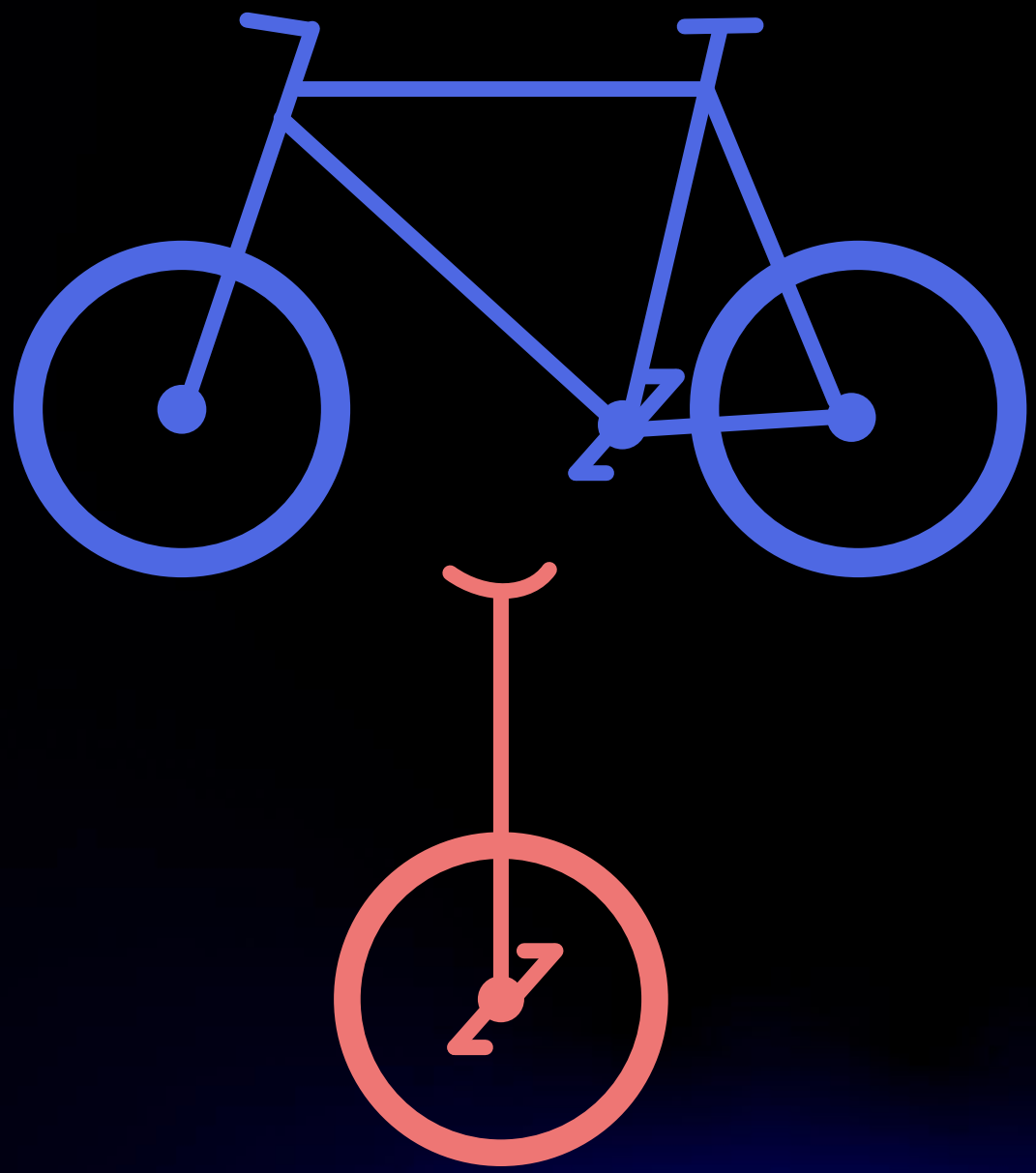


Survival of the fittest

Over time, systems evolve more than they are designed

More faults are eliminated over time

The best and most well-worn patterns are reused over and over

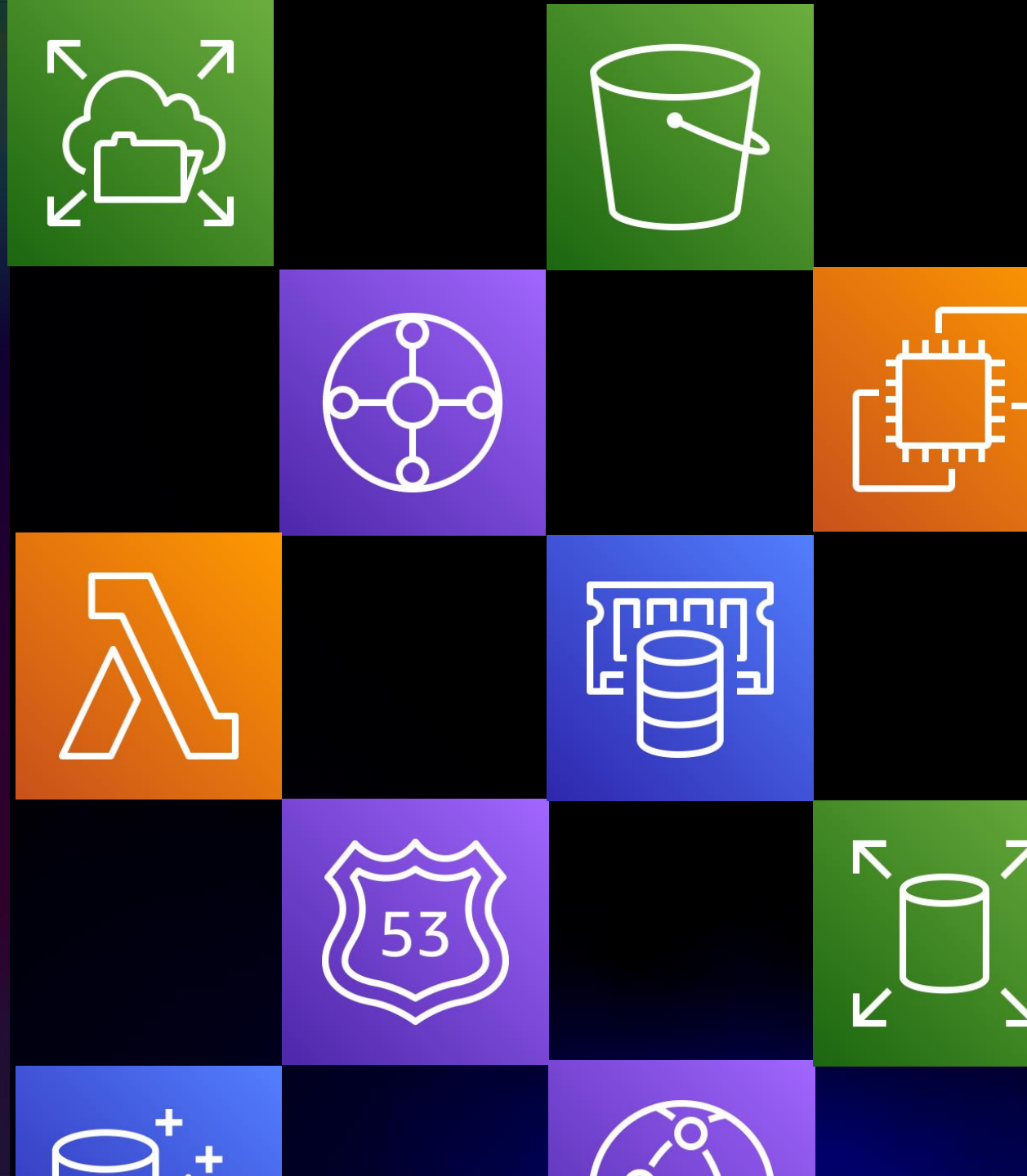


Survival of the fittest

Even the first original AWS services still contain original code from the launch versions

“Respect what came before” vs. NIH

Our Corrections of Error (COE) process feeds back into design iterations



Constant work and minimizing change

Constant work

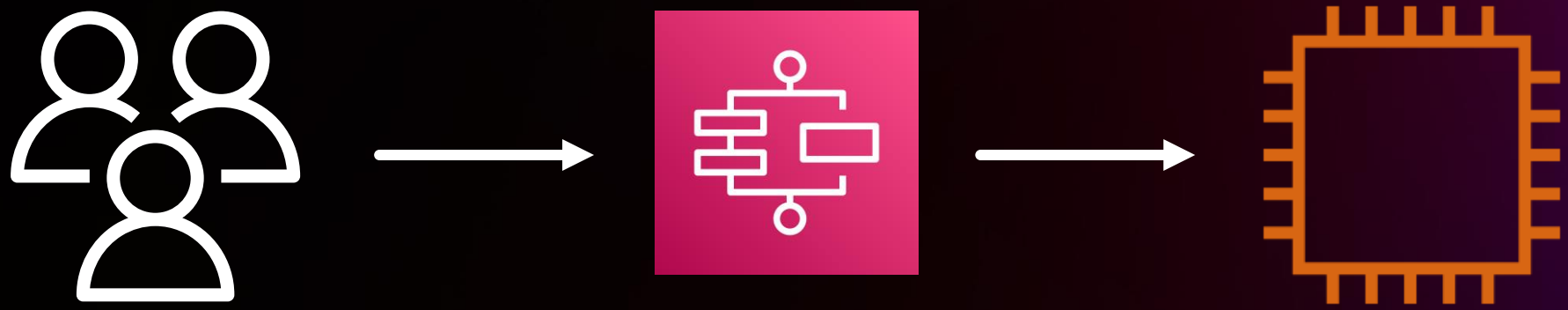
Risk is proportionate to rates of change in systems

Example: a spike in load can slow down a system, which can cause knock-on and cascading effects

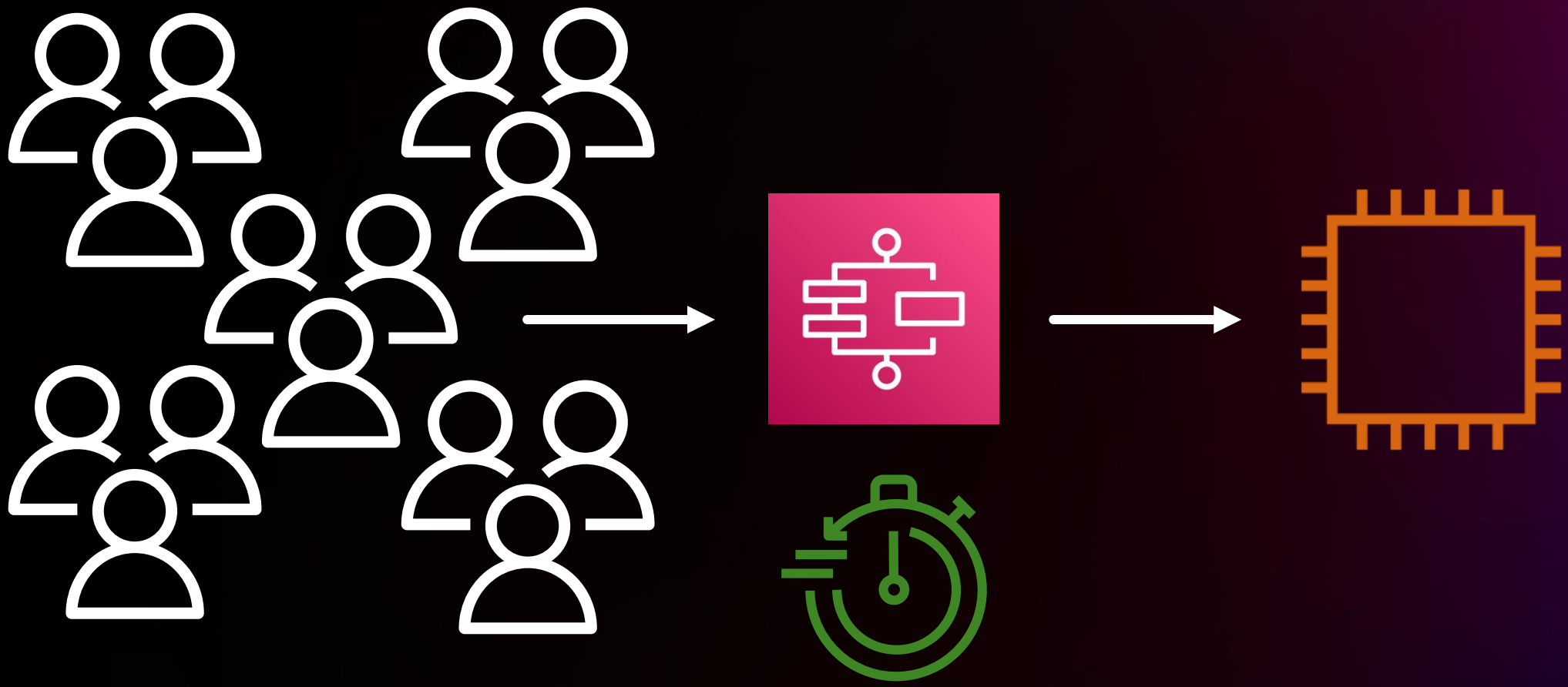
Reducing dynamism in systems is a great way to make them simpler

A counter-intuitive solution is to run the system at “maximum” load all the time, every time

Constant work

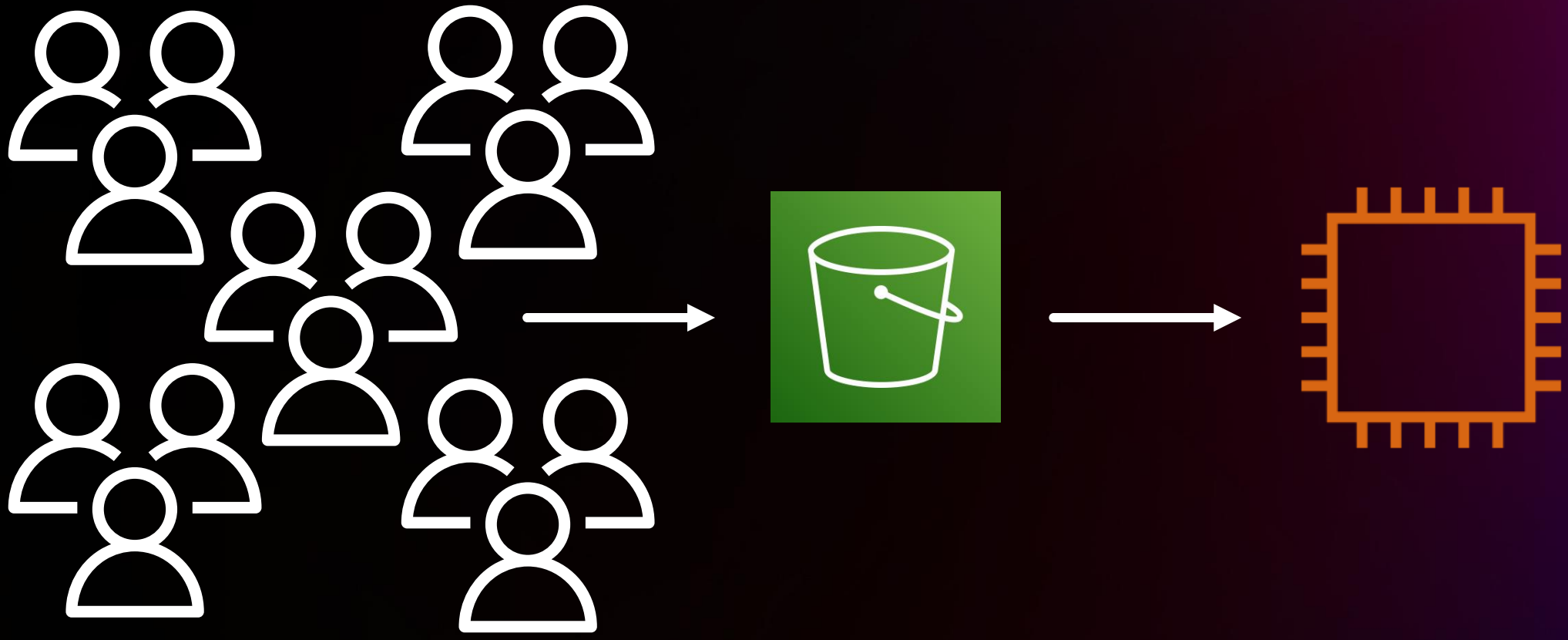


Constant work



Constant work

Constant work



Testing as time travel

“There is no compression algorithm for experience„

Andy Jassy

Testing as time travel

1000s of unit tests

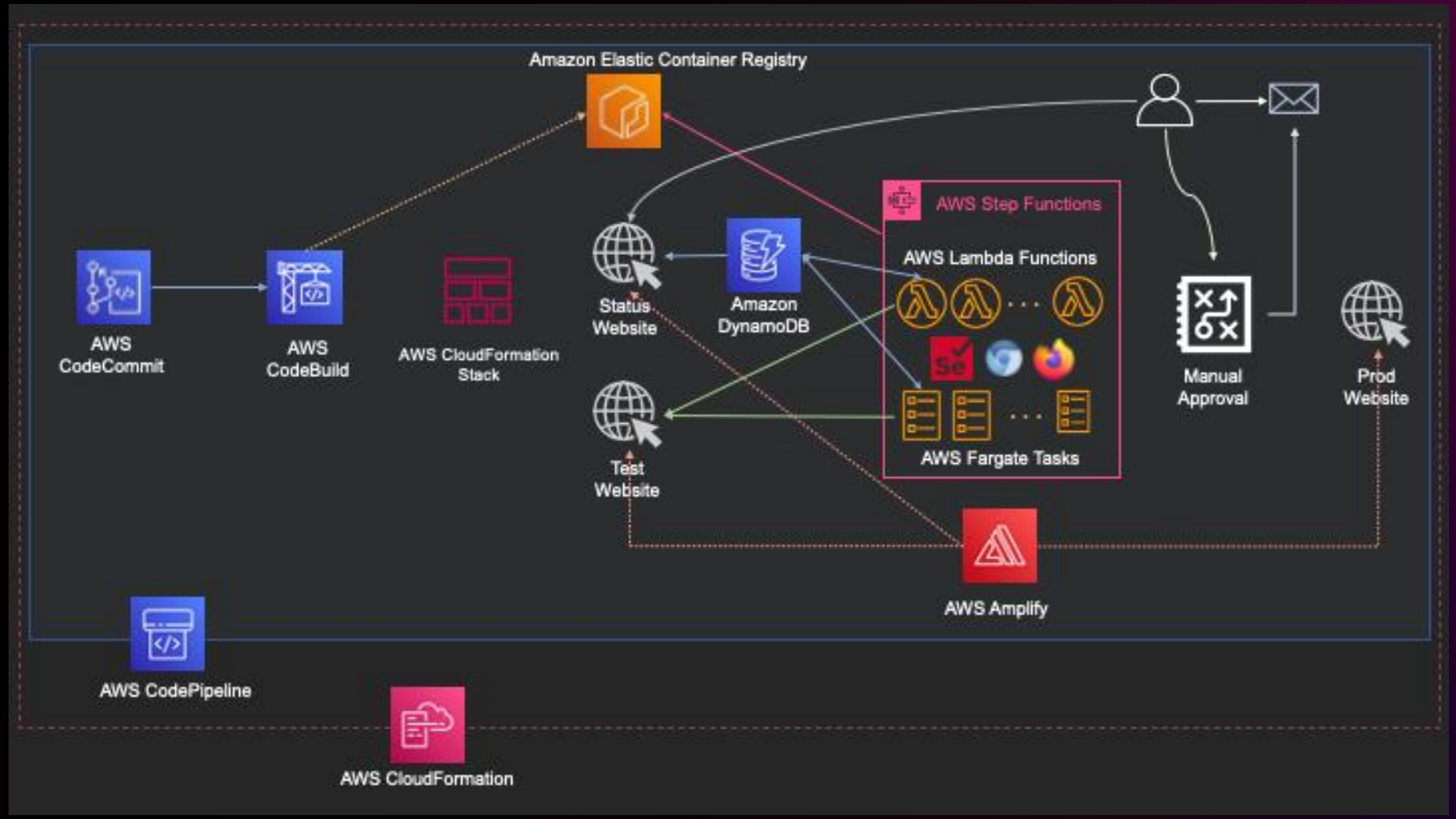
100s of integration tests

Pre-prod environments

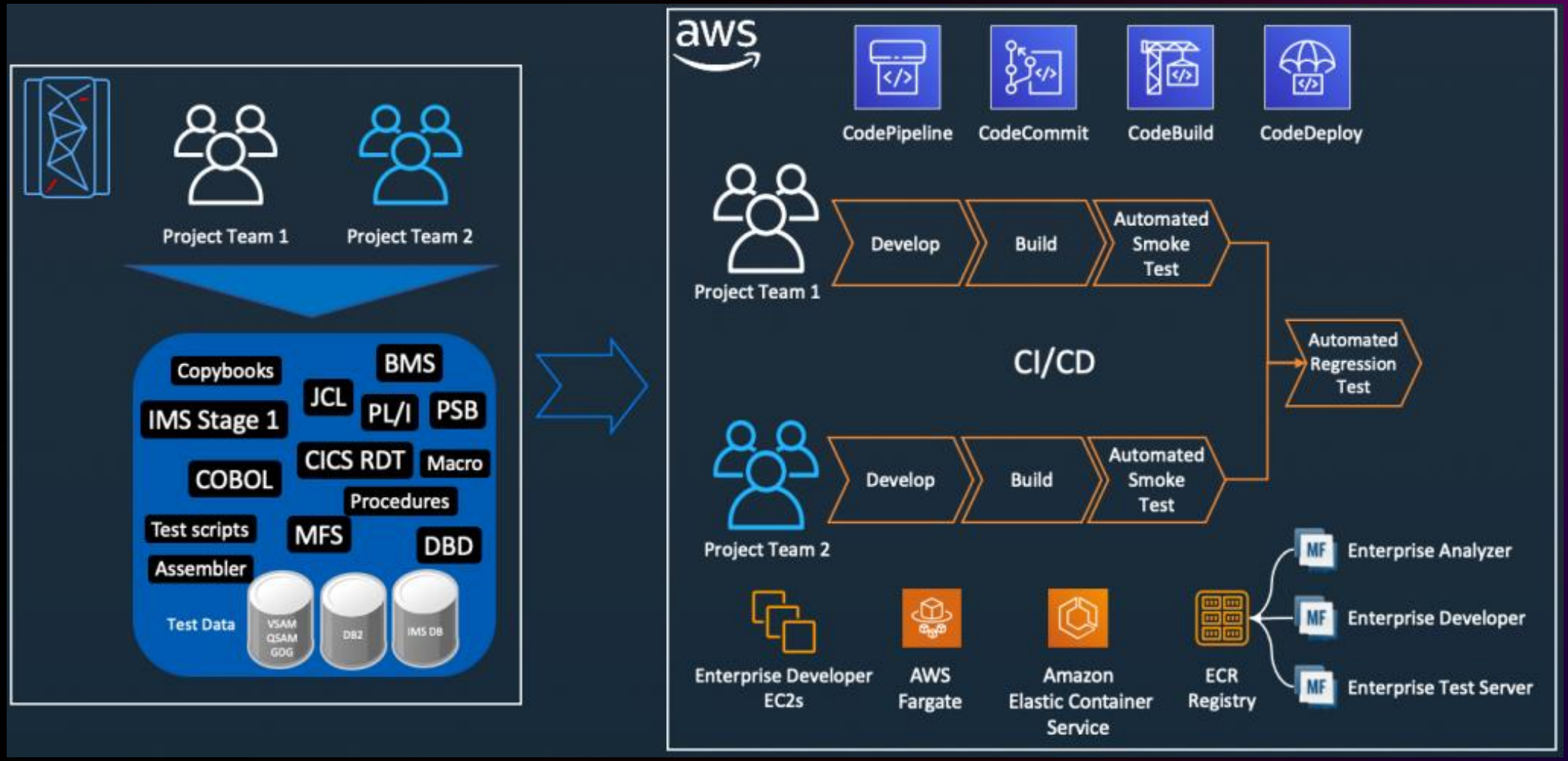
Roll forward and rollback testing

```
Running s2n_hash_test.c                                  ... PASSED       161 tests
Running s2n_rc4_test.c                                   ... PASSED     28742 tests
Running s2n_map_test.c                                   ... PASSED     57418 tests
Running s2n_override_openssl_random_test.c               ... PASSED        13 tests
Running s2n_handshake_test.c                             ... PASSED       232 tests
Running s2n_ecc_test.c                                   ... PASSED        30 tests
Running s2n_stuffer_test.c                               ... PASSED       701 tests
Running s2n_3des_test.c                                  ... PASSED     27248 tests
Running s2n_stuffer_hex_test.c                           ... PASSED       319 tests
Running s2n_pem_rsa_dhe_test.c                           ... PASSED        50 tests
Running s2n_aes_test.c                                   ... PASSED     54189 tests
Running s2n_fragmentation_coalescing_test.c              ... PASSED        64 tests
Running s2n_aes_sha_composite_test.c                     ... PASSED    196450 tests
Running s2n_malformed_handshake_test.c                   ... PASSED        82 tests
Running s2n_hmac_test.c                                  ... PASSED       392 tests
Running s2n_record_test.c                                ... PASSED    179817 tests
Running s2n_client_extensions_test.c                     ... PASSED       225 tests
Running s2n_self_talk_test.c                             ... PASSED  43000462 tests
Running s2n_self_talk_alpn_test.c                        ... PASSED  64500710 tests
Running s2n_drbg_test.c                                  ... PASSED   1000155 tests
Running s2n_random_test.c                                ... PASSED 204480831 tests
Running s2n_cbc_verify_test.c                            ... PASSED   8641805 tests
Running s2n_aead_aes_test.c                              ... PASSED  29115201 tests
```

Testing as time travel



Testing as time travel



Testing as time travel

With chaos engineering, Region-scale simulation, automated reasoning, and formal verification, we're going much further

We can prove that code is correct for any possible set of inputs

Getting easier and easier

Testing as time travel

Use instrumentation and observability for full life-cycle testing

Every sensor you add can provide a lifetime of value

More data as the service grows

Operational safety

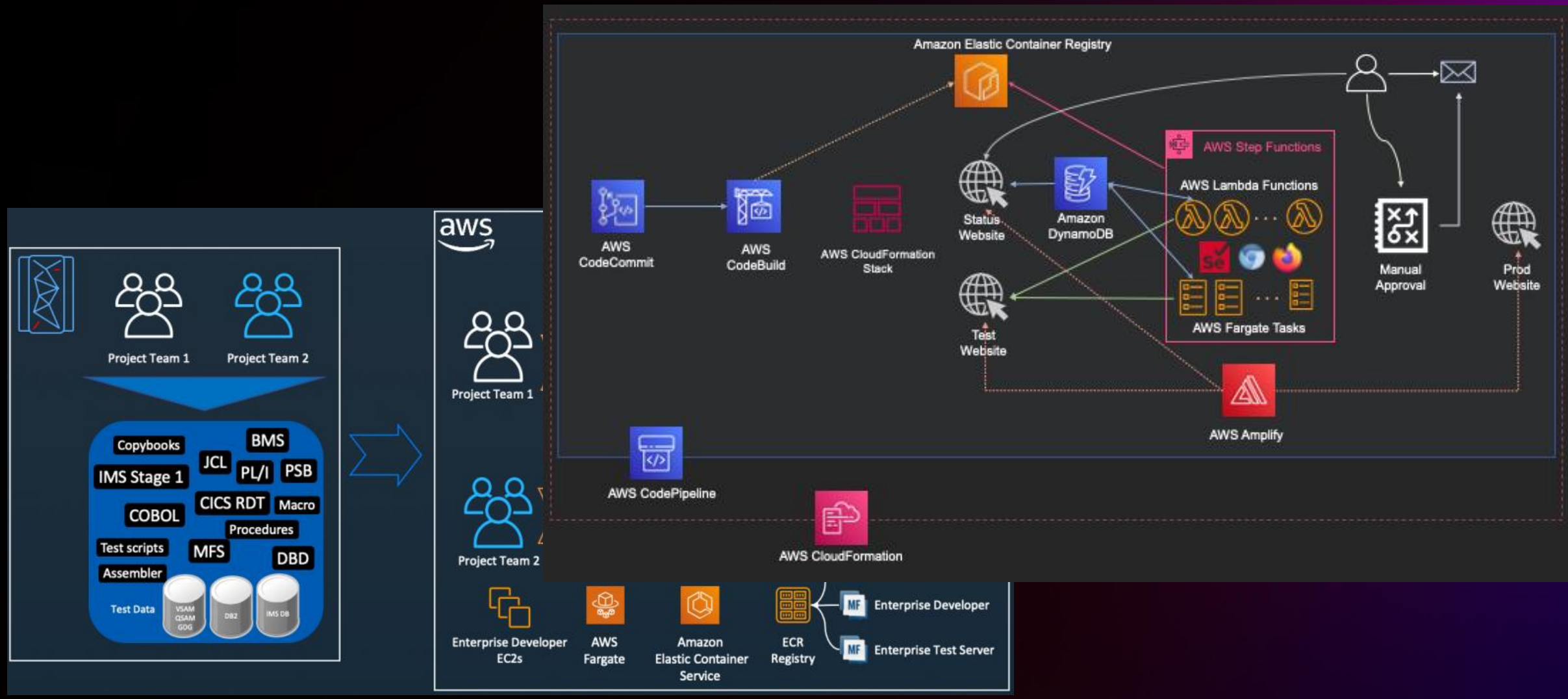
Operational safety

Web services are … services

Live, running, operational concerns

Most issues are triggered by people making changes

Deployment safety



Deployment safety

New code means risk, so we are incredibly paranoid about deploying it

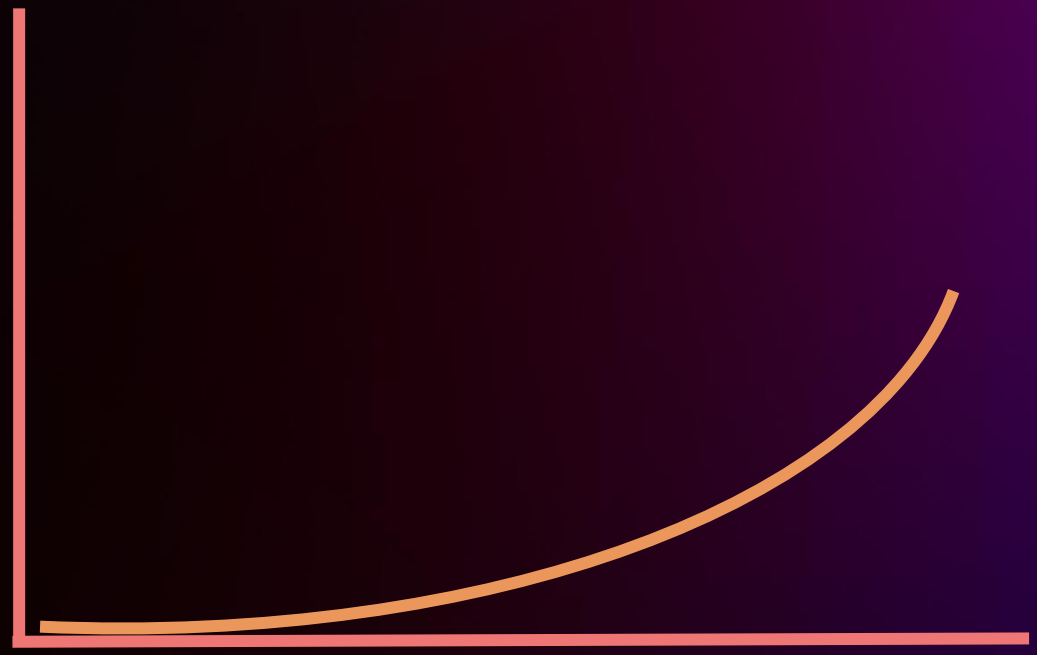
CI/CD staged deployment process

Promotion testing and monitoring at every stage, with automated rollback

Fast and reliable rollback

Deployment safety

1. Code-review
2. Check-in
3. Pre-Production
4. One Box
5. One Availability Zone
6. One Region
7. Onward …



Cattle vs. pets

Most systems at AWS are beyond a scale that can be managed by hand

We use Auto Scaling groups, VPCs, subnets, security groups, and more as units of abstraction

Our deployment systems clone infrastructure between Regions

Cattle vs. pets

AWS operators don’t have access to all AWS Regions

Services such as AWS Snowball and AWS Snowmobile are designed to be disconnected for periods of time

Bastioned systems: limited access for recovery via “bastions,” with record and notification processes

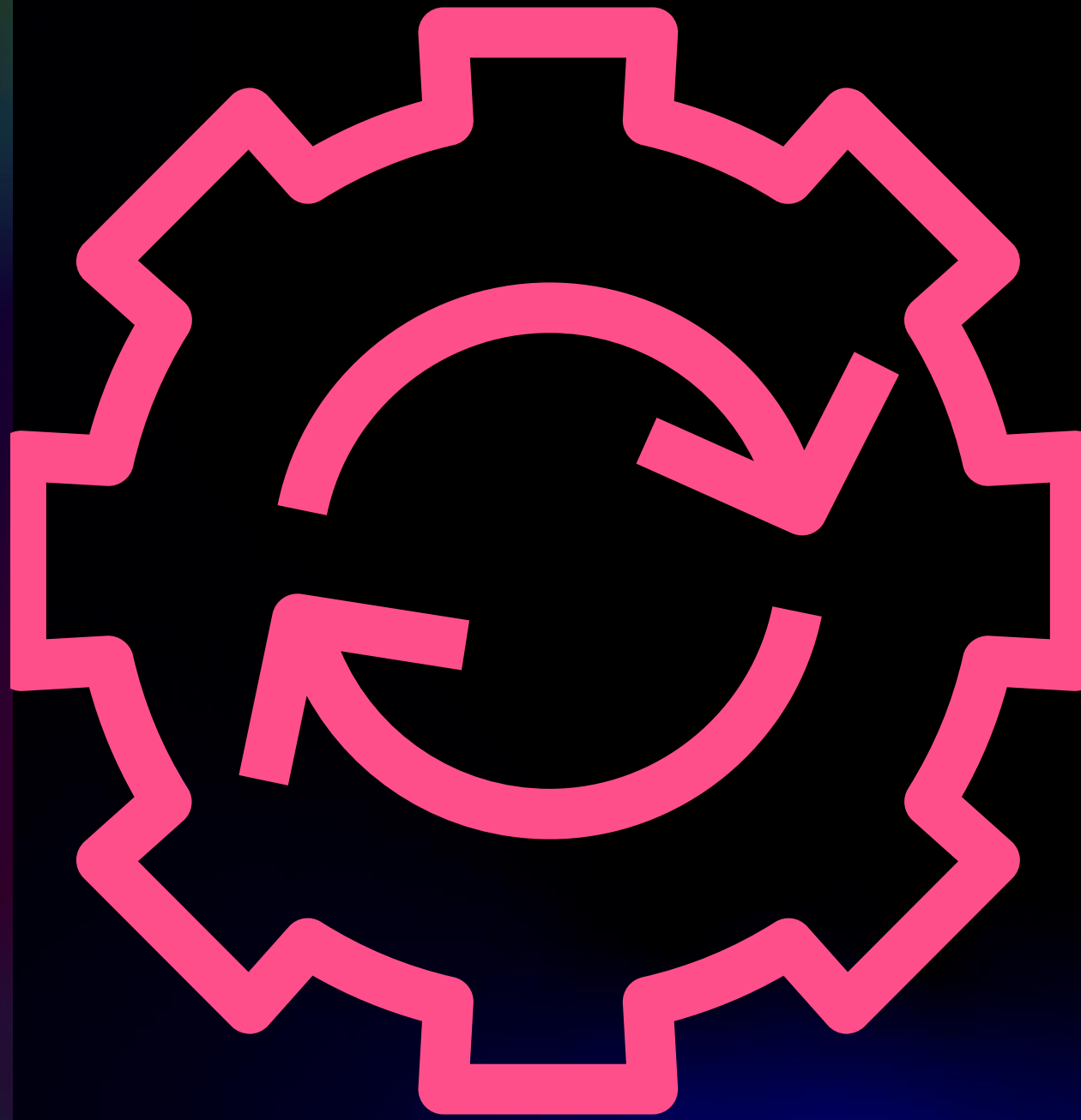
Hermetic systems: systems with no general purpose, interactive, or administrative access

Automation first

“Automation first” is the only scalable answer

“Automation first” is also the safer option

Every lesson learned can be stored as code, not just good intentions



Technical fearlessness

“Culture eats strategy for breakfast „

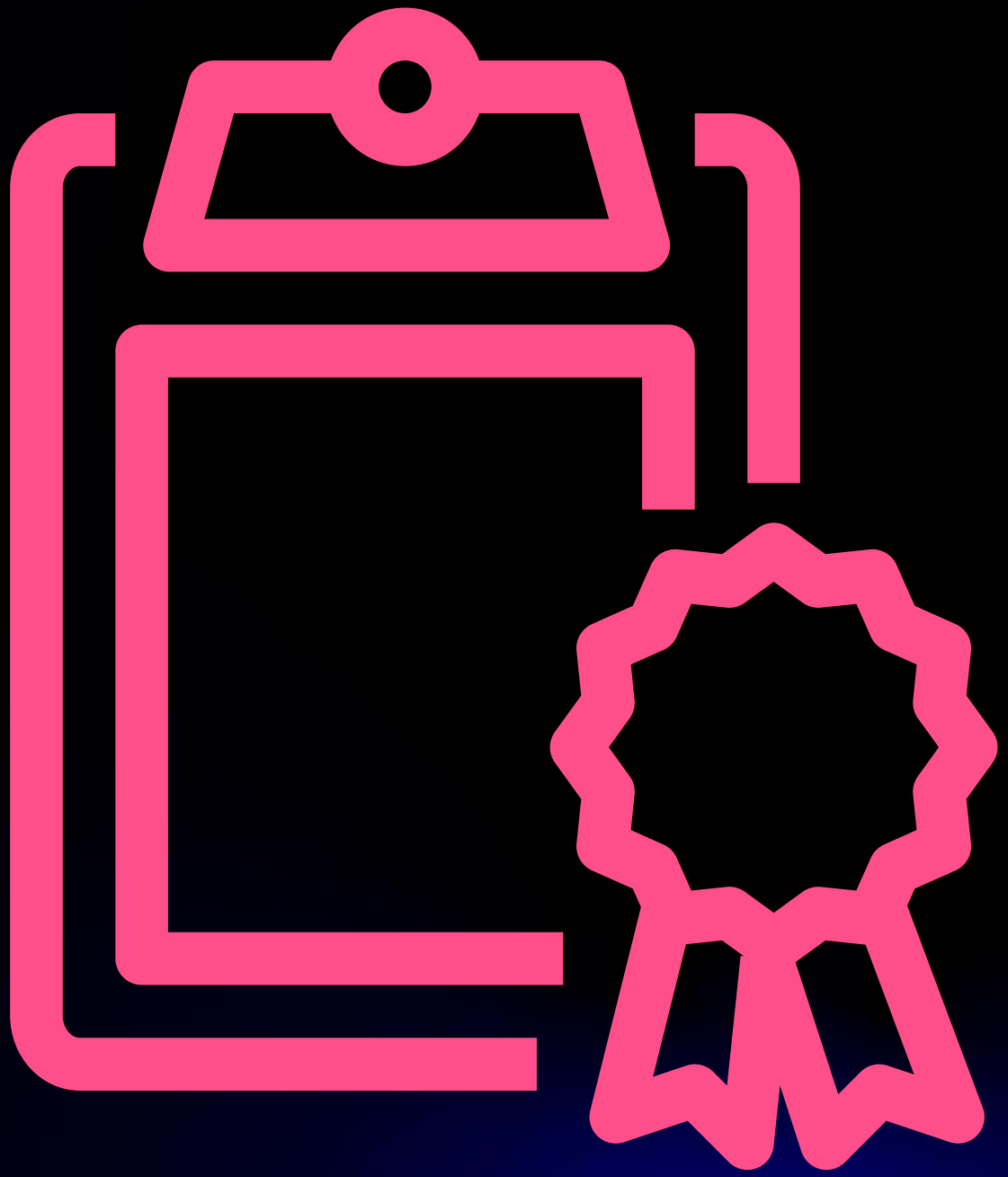
Peter Drucker

Technical fearlessness

High standards demand elite teams, and elite teams insist on high standards

Quality isn’t something that is budgeted explicitly

Quality is embedded in the very approach

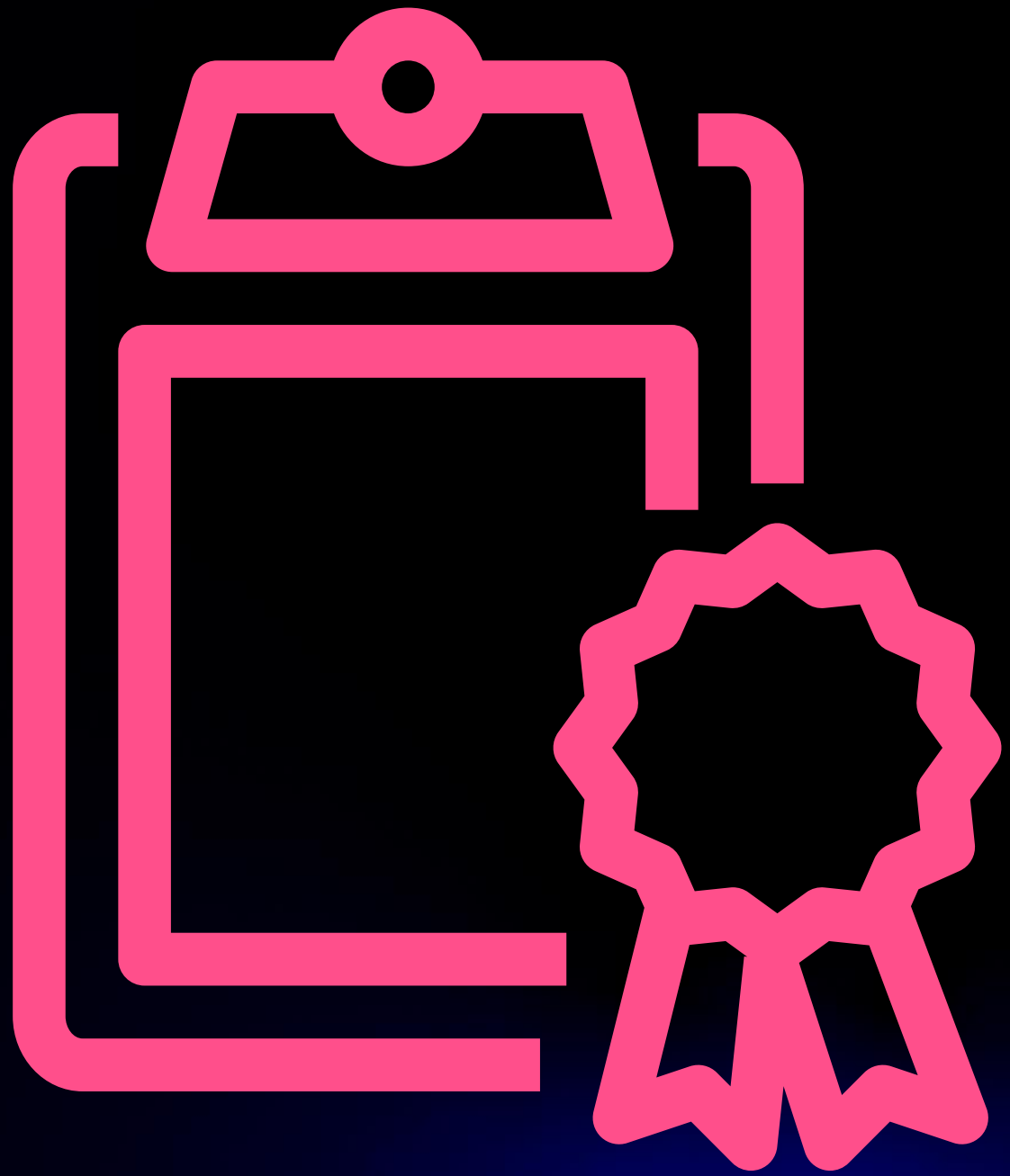


Technical fearlessness

No “dumb questions”

Any person can be elite with the right support

Critiques and criticism broadly welcomed

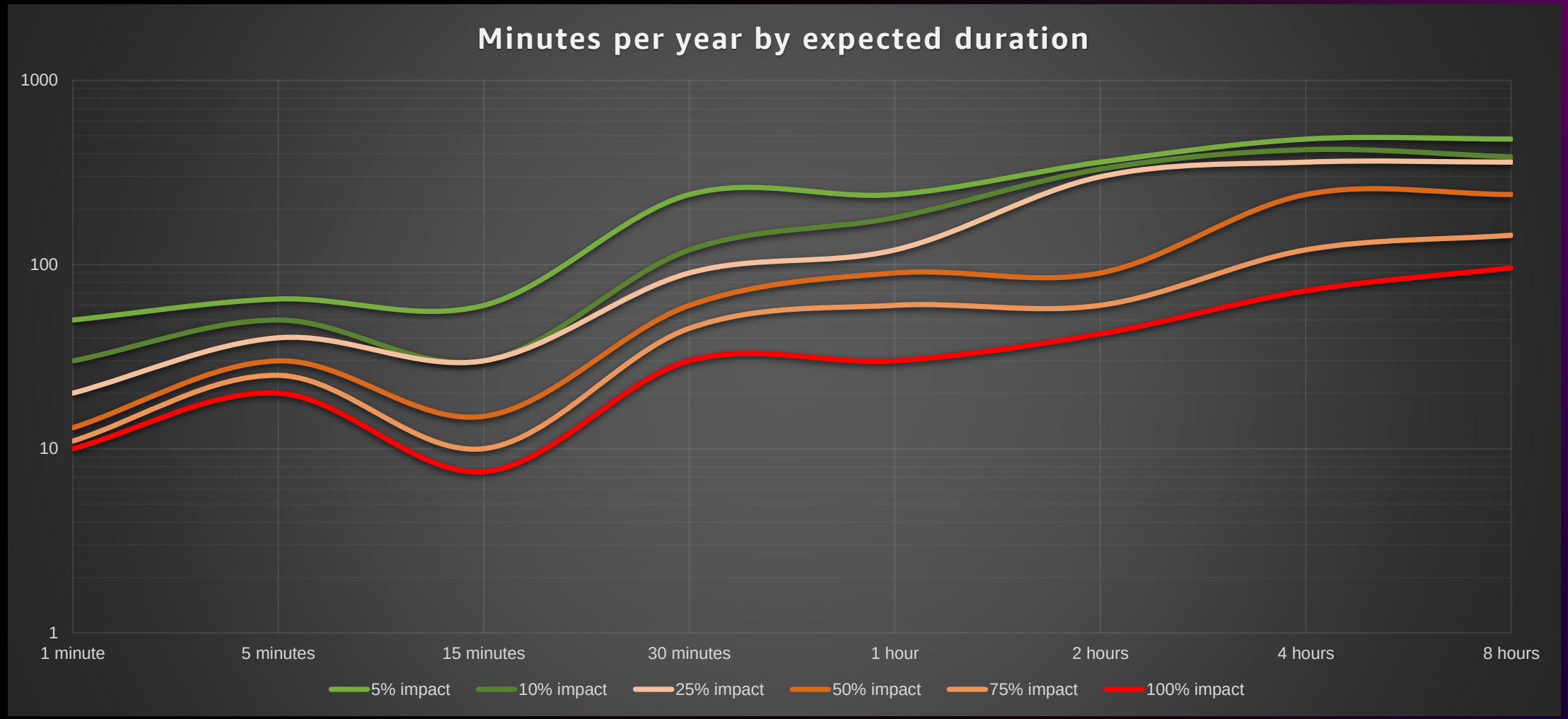


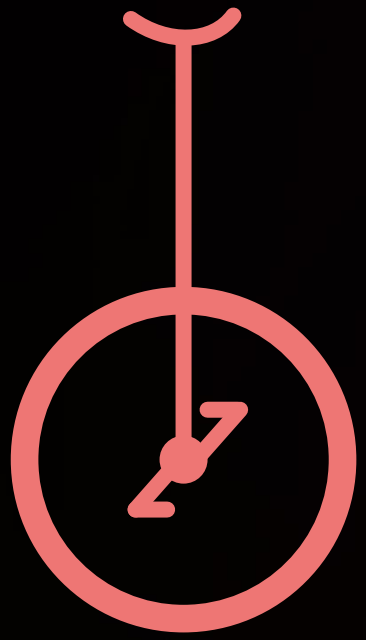
Technical fearlessness

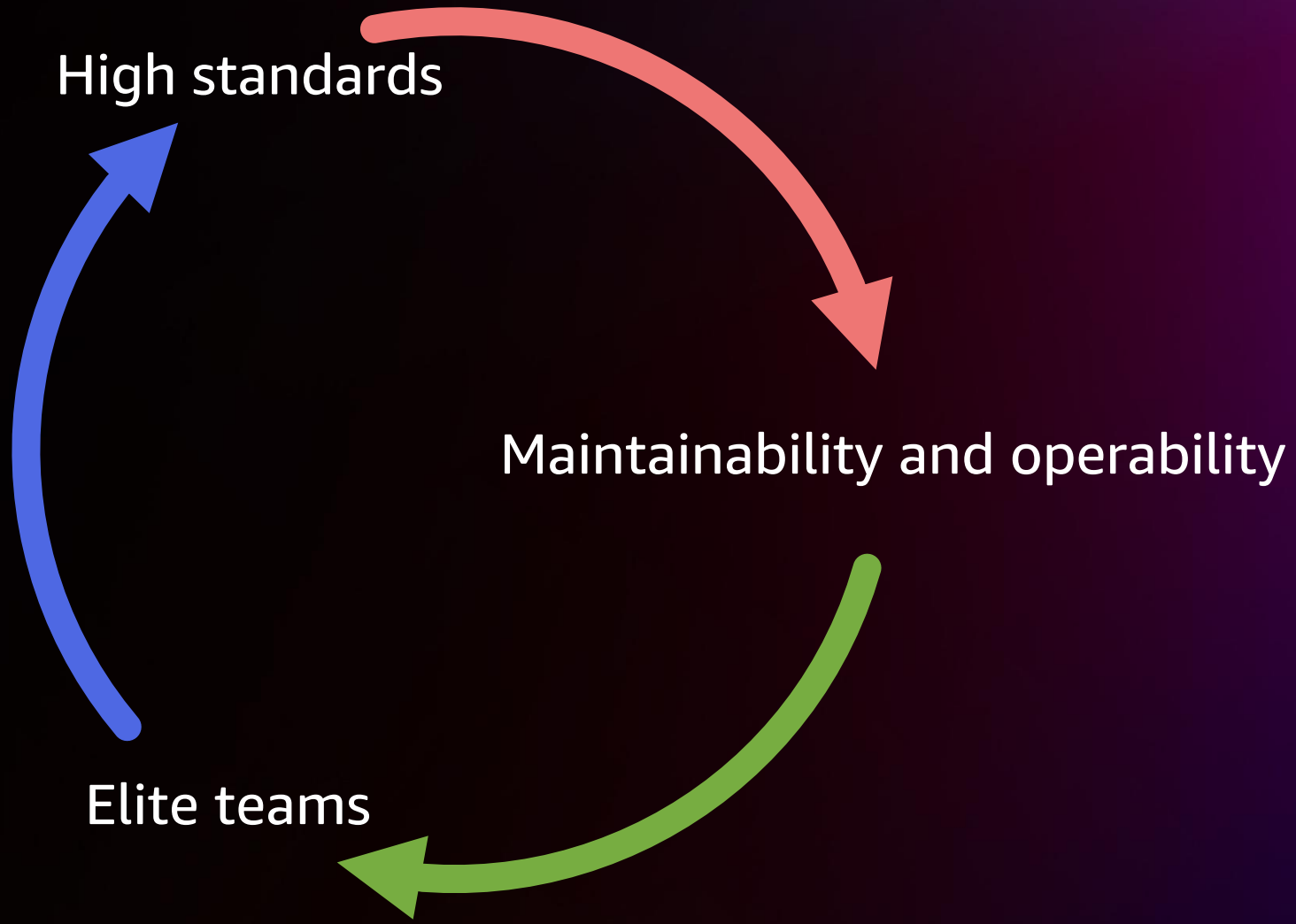
Technical fearlessness

Technical fearlessness

Takeaways







Thank you!

Colm MacCárthaigh
colmmacc@amazon.com

Please complete the session survey in the **mobile app**