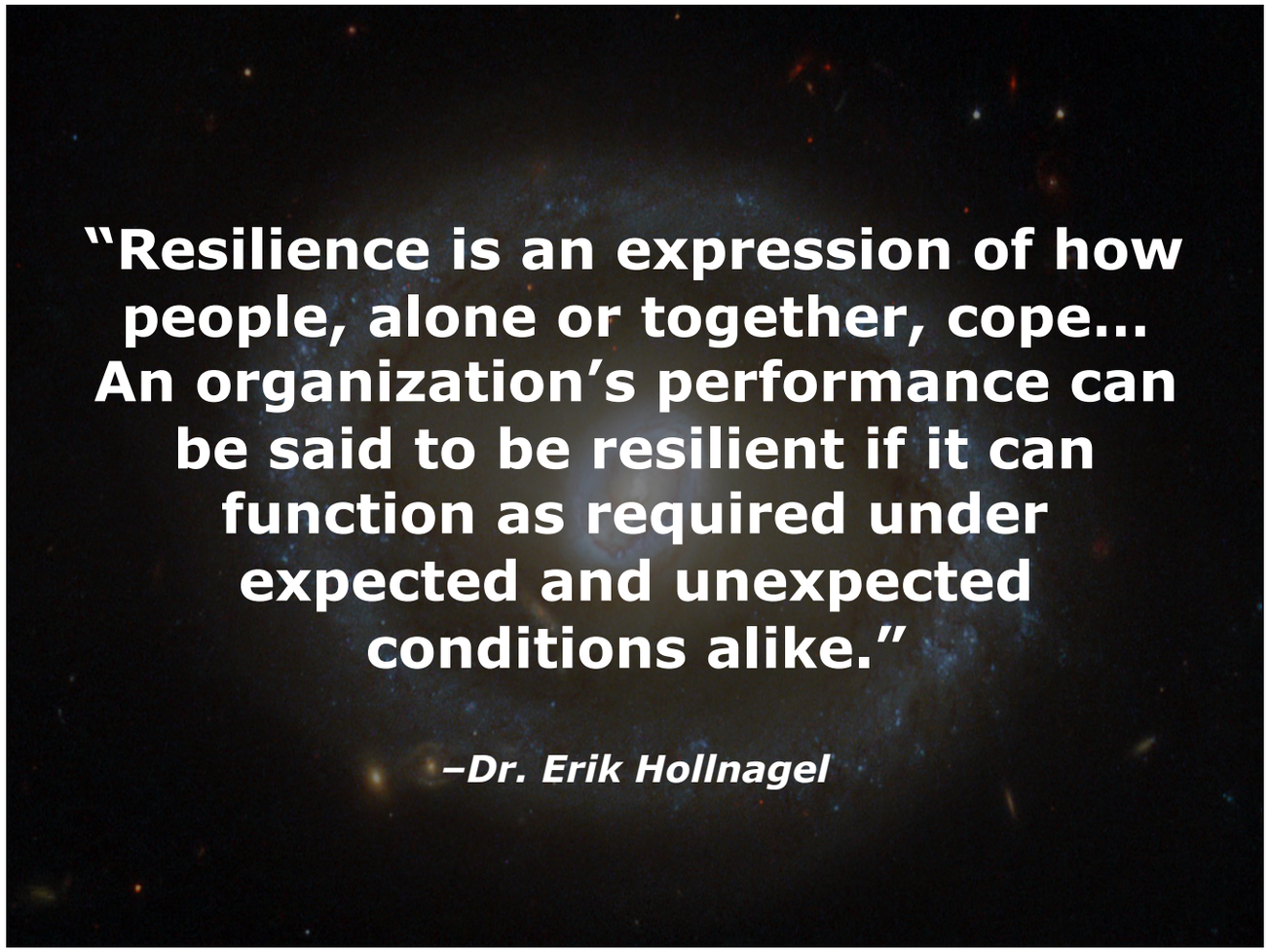
Resilience Engineering Mythbusting

Will Gallego, Sr. Software Engineer
@wcgallego

fastly

My background - retro’s/postmortems/incident reviews. Keeping blame aware here, assuming best intent.

Resilience Engineering has been a practiced field for a while now before we in software engineering got a handle on that ball - in nuclear power plants, medicine, and transportation to name just a few. When we say “Resilience Engineering”, it’s with a specific idea in mind. So what do we mean by engineering “resilience”?



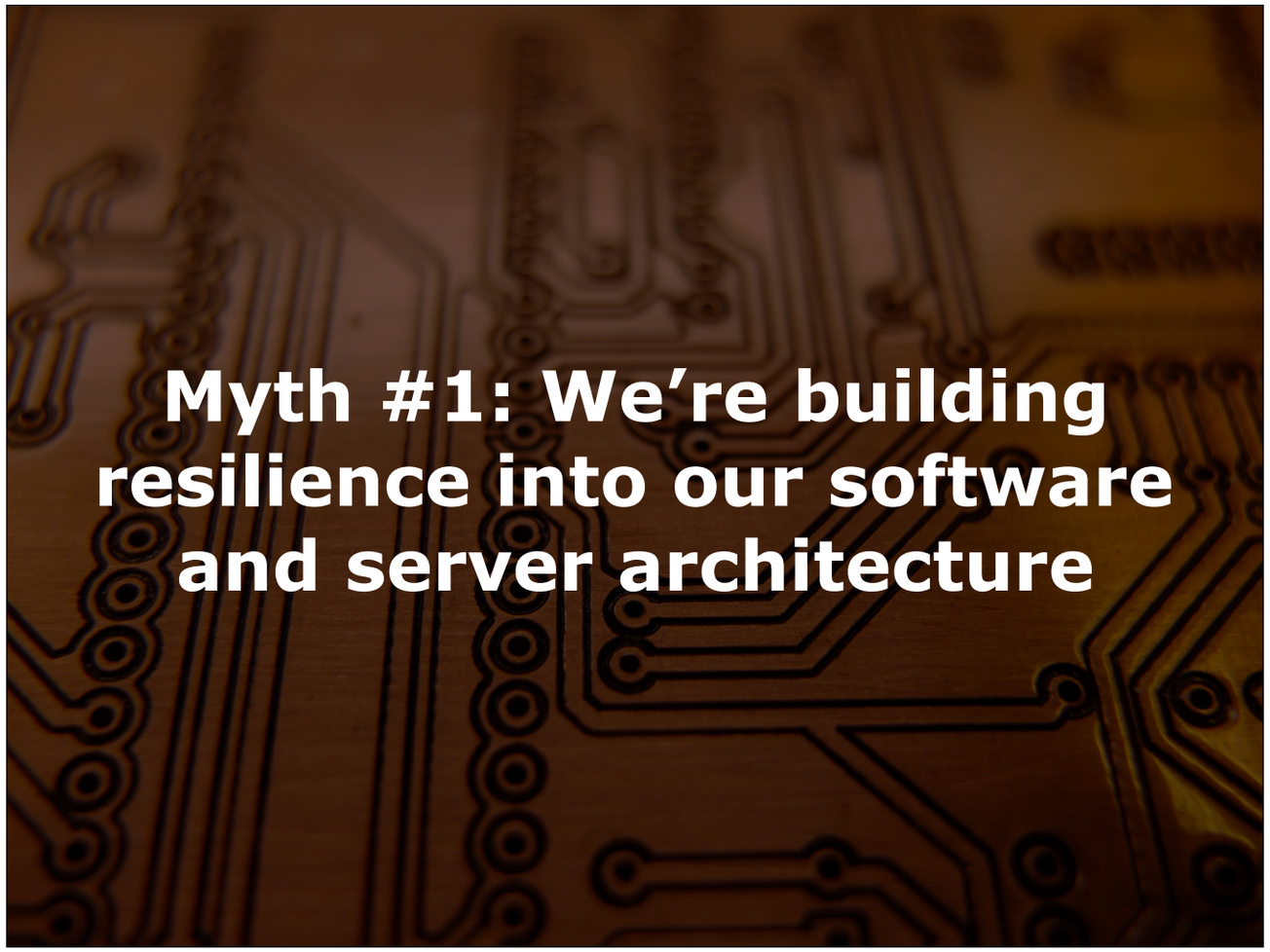
Thorough, but verbose. This is a lot to keep in mind!

Can it adapt?

Short hand, Easier to remember. Is it able to..

- make decisions? (this is not the same as conditional logic)
- Change course (or decide to keep the course!)?
- Can it use assumptions, hints, clues, and expertise to plot a path forward?

Only then can something be called “resilient”



Usually mean robustness or reliability. Our code and our infrastructure don’t have the ability to make judgment calls or to think around things that haven’t been planned already, only predetermined instructions

Any SaaS company that tells your their product will make your business “resilient” is trying to sell you something

Robustness

Can the system perform within predetermined expected boundaries

- Caching of static assets
- Automatic failover of databases
- Load balancer removing nodes from pool

Reliability

Can we trust the system to perform within expected bounds

- **Assessment of known boundaries**
- **Confidence in system safety**
- **Understanding limitations**

Resilience

Can it adapt when the capacity to work is exceeded

- **Going "off script" from a runbook**
- **An operator terminating a process consuming heavy resources**
- **Escalating a page**

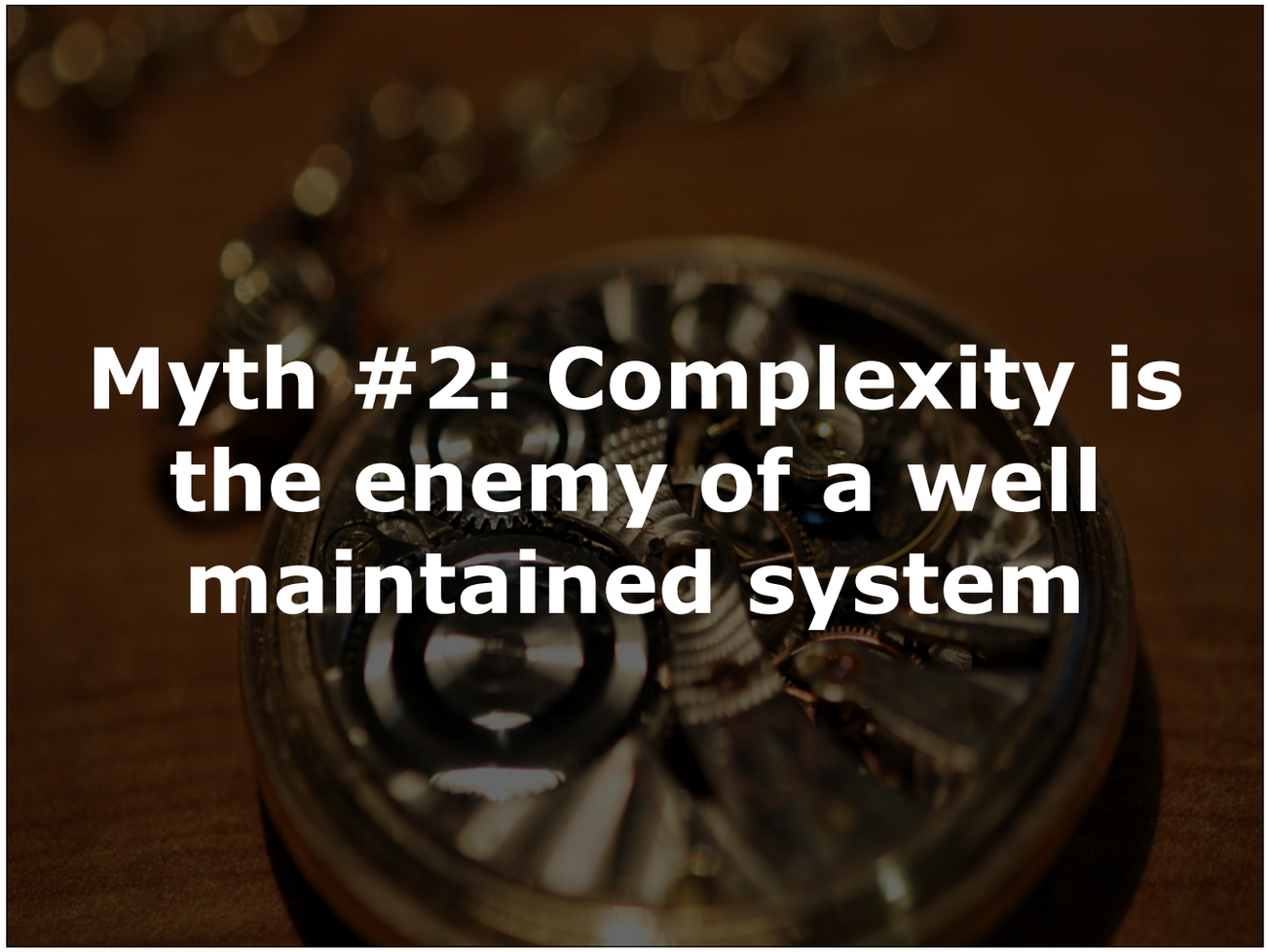
Bringing back to what we want to focus on today, resilience!
Resilient - Who are the people making large and small adjustments to keep it performing when the above two are not met or may not be met in the future?

Our servers can’t display resilience …but our systems can

Systems can be because people are part of systems - interacting, directing them, and making decisions that computers can’t to alter previously laid out designs.
Isn’t this just pedantry? Because Resilience Engineering exists external to the tech industry, with detailed research and studies that we should adopt more thoroughly, without which:

- Chaos Experiments don’t have insight
- Incident management misses flexibility/critical decision making in handling crises
- Retro’s will tend towards blame and shallow bug fixes, missing deeper understanding

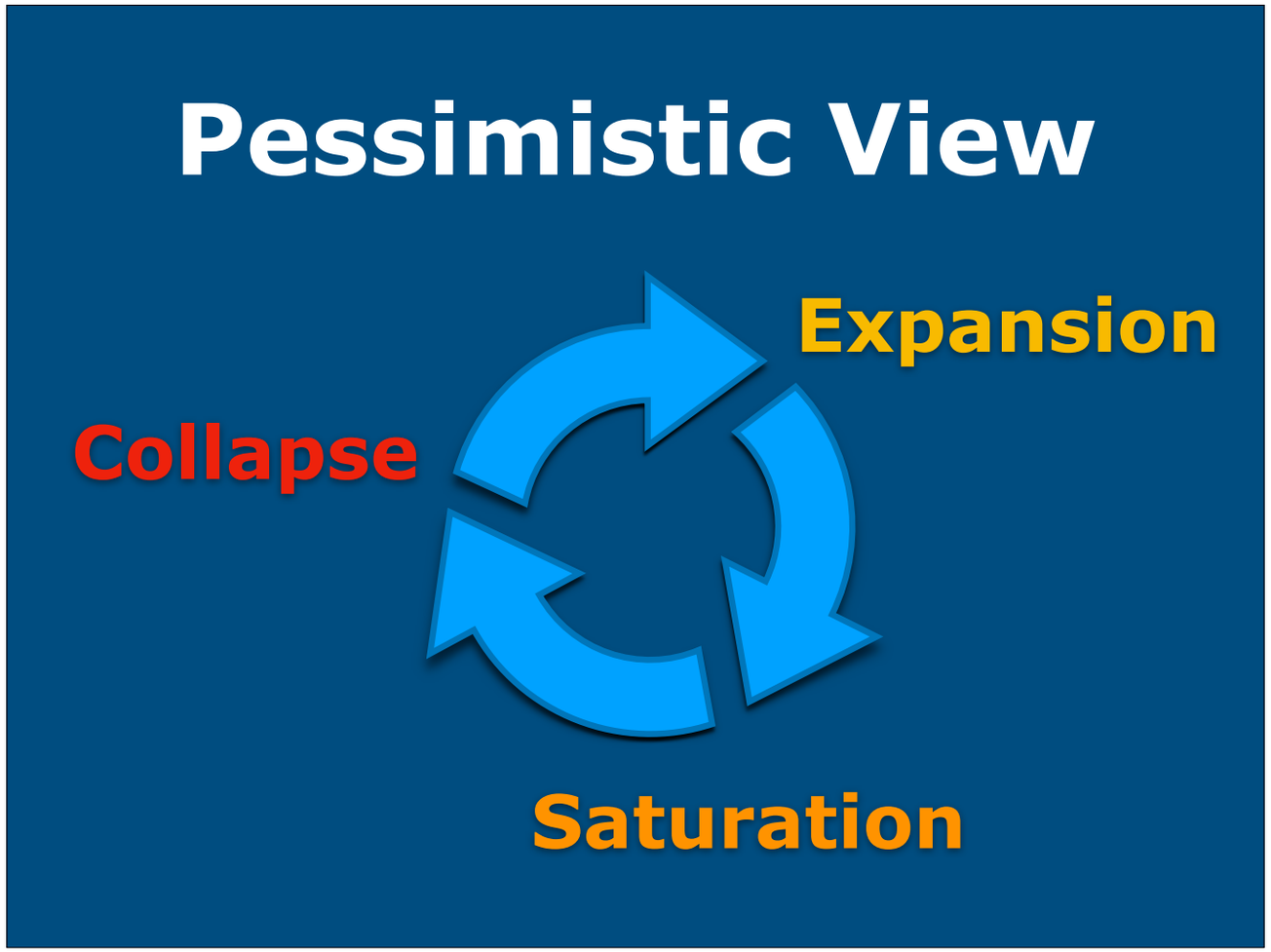
Without an understanding of these concepts, we can’t execute upon these well, we can’t respond effectively, and our systems become brittle



We’re constantly looking to reduce complexity in systems, and for good reason. They’re hard to reason about and there’s a tendency towards. Sometimes we build enough functionality, though, that our systems need to maintain some level of complexity to function.

How do we cope with complexity?

If complexity is therefore required in our system, without a way to simplify further, we need to figure out how to raise our level of expertise to meet the requirements of maintaining a complex system



Systems trapped in cycle of expansion, saturation, and collapse. Catastrophe is inevitable. Continually trying to be better, faster, cheaper, which increases complexity and costs for such increase with little compensation.

Assumption of codebases gaining bloat, alerts going stale, noise from our observability tools. “It got too complex, so we couldn’t maintain it and it exploded”

Optimistic View

Systems require self monitoring

Anticipate and Learn before collapse

Experts begin to build an understanding of a system to avoid the costliness of letting a system exceed capacity before requiring intervention.

Resilience Engineering is **practicing this continuous effort** to be self aware of our system, to make changes when judgment demands, and continuously seek to expand expertise to handle the unknown. This investment is *uncertain* - we don't know when we'll need it (and often if we'll actually need it!)

CE looks to increase understanding of how a system or group of interacting systems work. Sometimes we find vulnerabilities to fix, but that’s a nice side effect.

Also, not creating chaos, but revealing it. The chaos already exists in a system.

I’ve seen these errors before, I know what to expect (or have an idea around it). I have an “intuition” about how we can approach a new unforeseen problem. That’s what we’re looking to gain. This “intuition”.

For Chaos Engineering to work, you need to understand the system

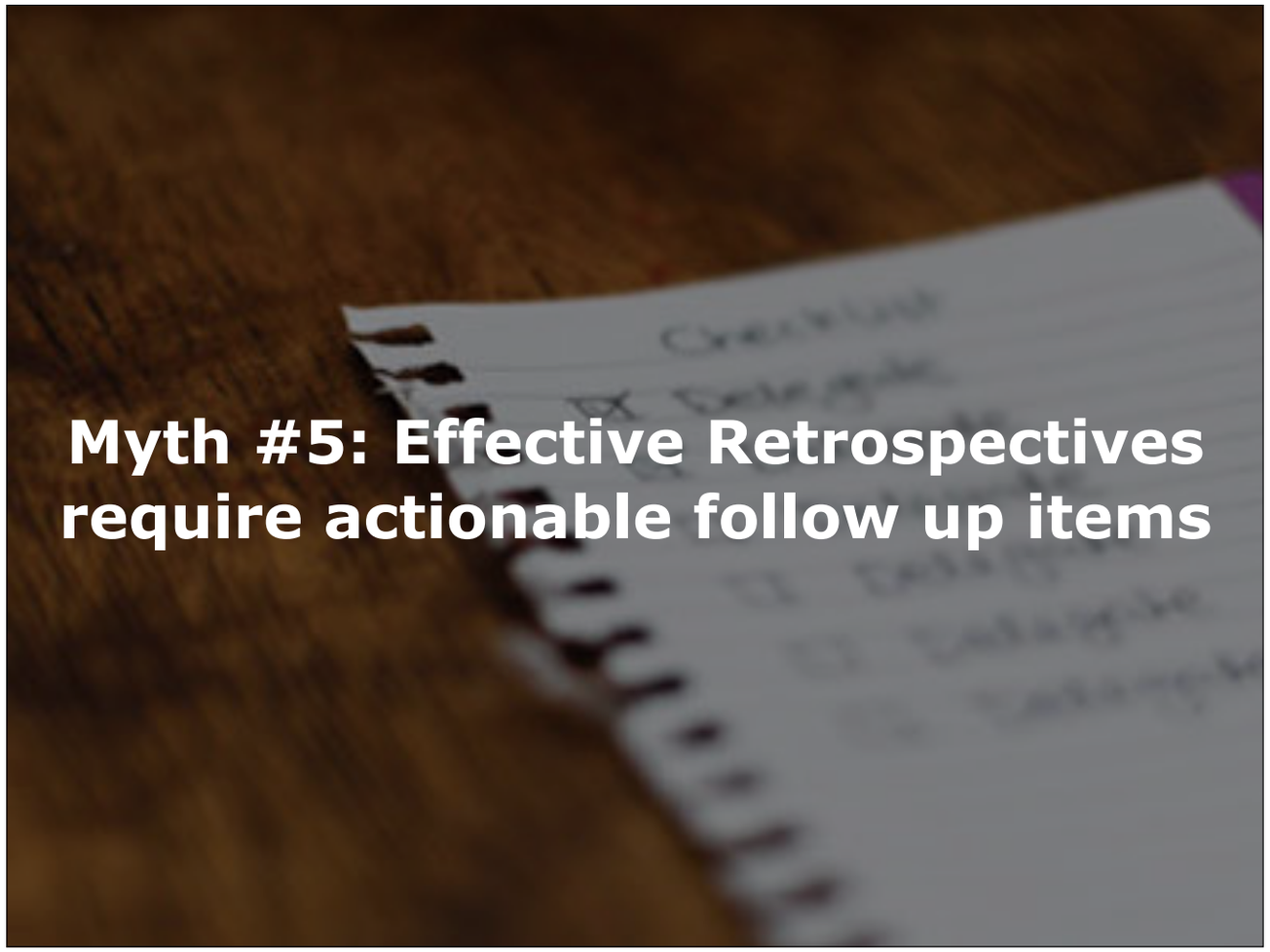
Myth #4: Always follow best practices to ensure system availability

Systems fail for all sorts of reasons, even when we do things that seem right. Might not be a fit for your situation, or need tweaking
Best practices are not 100% universal. It is often handwaving from “I saw a blog post/conf talk/etc and we need those best practices’

Same applies to runbooks, wiki's, guides, etc. "This worked last time" doesn't guarantee it'll work again this time. Situations are not always the same, documentation grows stale and systems *change*. Resilience means making good judgment calls by continuously observing our systems and making a decision, sometimes disagreeing with our past selves - and our "best practices".

Best Practices don’t guarantee safe actions

If it goes right, then you were ok to be “risky”. If it goes wrong, though? “You should have followed best practices!”. This is constructing the value of a decision after the fact, instead of understanding the decision making properties during the incident.



Time for learning. Errors will route around fixes. Need to develop understanding of problems. Retro's should be transparent and open, shared between teams.

It's tempting to graph every thing we can, to have dashboards, SaaS products, and more paging alerts. We want to be *sure* we catch everything and to make sure this *never happens again*. This is a tendency especially seen in incident reviews look for "root cause". We want *answers*! And we want to feel like we have control by coming up with concrete deliverables

How do we know our Retrospectives are effective?

Difficulty to fully judge. Want #’s to know reviews are “worthwhile” or how to improve upon them. Assessing the skill level of your engineers, their ability to learn and grow from outages. It’s slow, it’s gradual, and it’s almost invisible in the short term. Focus on confidence your team has in the system and who understands it.

Alternate question - how do you judge how much your engineers learn at say…a conference?

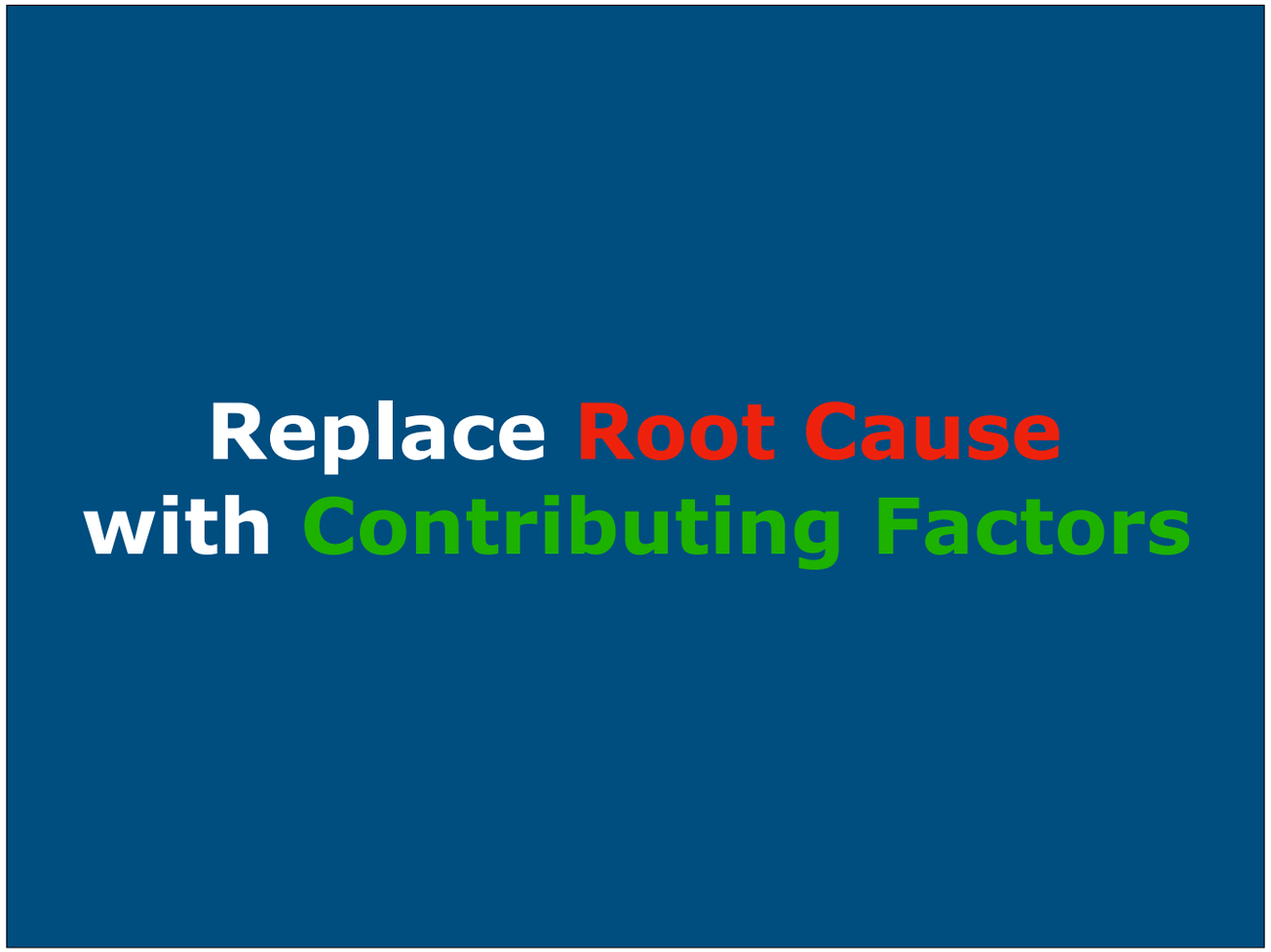
See this often as the objective for retro's. Our systems are complex and it's tempting to simplify resulting from a single event. Solve that, and the problem will never happen again!

Retro's that highlight root cause lead to shallow answers for complex problems. They often misattribute or lead to blameful thinking (stopping at most political convenient "root cause"). With minimal effort, you can find an event or events preceding a root cause. And one before that. In short, it's root cause all the way down. Events interact with one another and emergent phenomena crop up that don't exist with events separately.

The “Five Whys” Strategy is equally fallacious

We repeatedly ask “Why” 5 times to get to our “answer”.

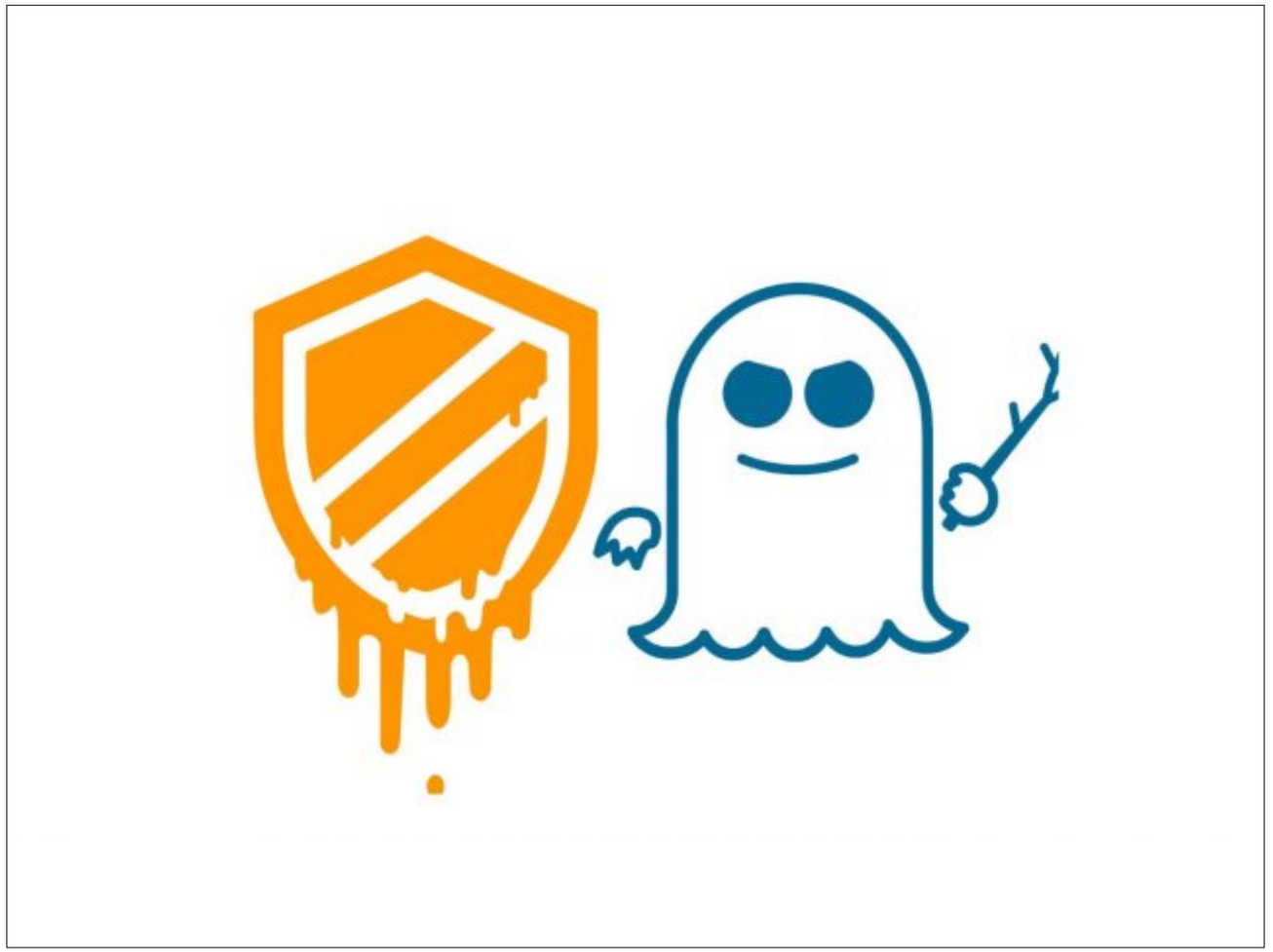
- same problems with linear causality, ignoring interactions/emergent phenomena
- Ask team to individually list and get different sets of “Why’s”.
- What’s the root cause of “success”, the one thing that made it go “right”?



Replacement? “Contributing Factors”. Understand motivations people have for decision making and why those decisions looked optimal at the time. Failure will route around our best efforts, but deeper inspection into problem statements is how we get folks leveled up. That leveling up is where we build expertise that allow systems to act with resilience.

We like to measure how “safe” we are by noting how often/badly things go wrong. Often seen with MttR and MtBF. The recency, frequency, and severity occludes:

- failures are constantly happening
- Dark Debt
- Can be gamed! Severity and decision as to what constitutes a failure to “measure” are subjective



(Spectre/Meltdown story) Hardware vulnerabilities in speculative execution to read memory from other programs.

How long should code last until it’s considered “safe”? This code lasted 25+ years before it was found out to be a vulnerability

Share expertise instead of numbers

Ask “obvious” questions. Experts share with folks new to a topic, question what is considered “standard”.

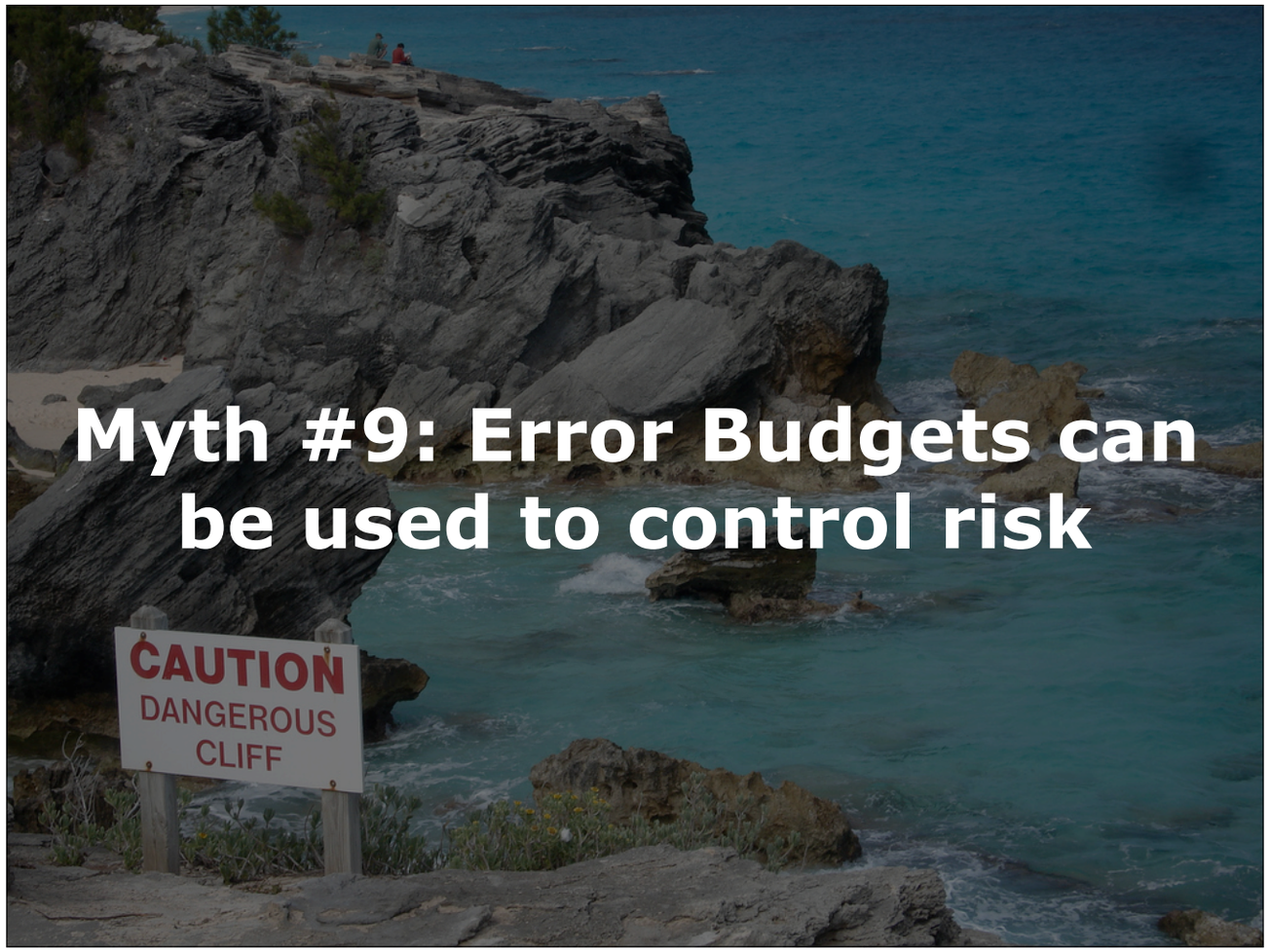
Can be fun, for sure, but severity not directly correlated with learning.

Outcome bias!

Tendency to smooth edges, even in most transparent environments.

Look to
close calls

Small incidents give us a chance to get really good at doing the introspection, looking through details
Close calls can produce significant learning as well



Not strong on Error Budgets. (Briefly discuss SLO/SLI/SLA/error budget)

Dangerous CSS story (CSS took down website)

Safety-I (protective)

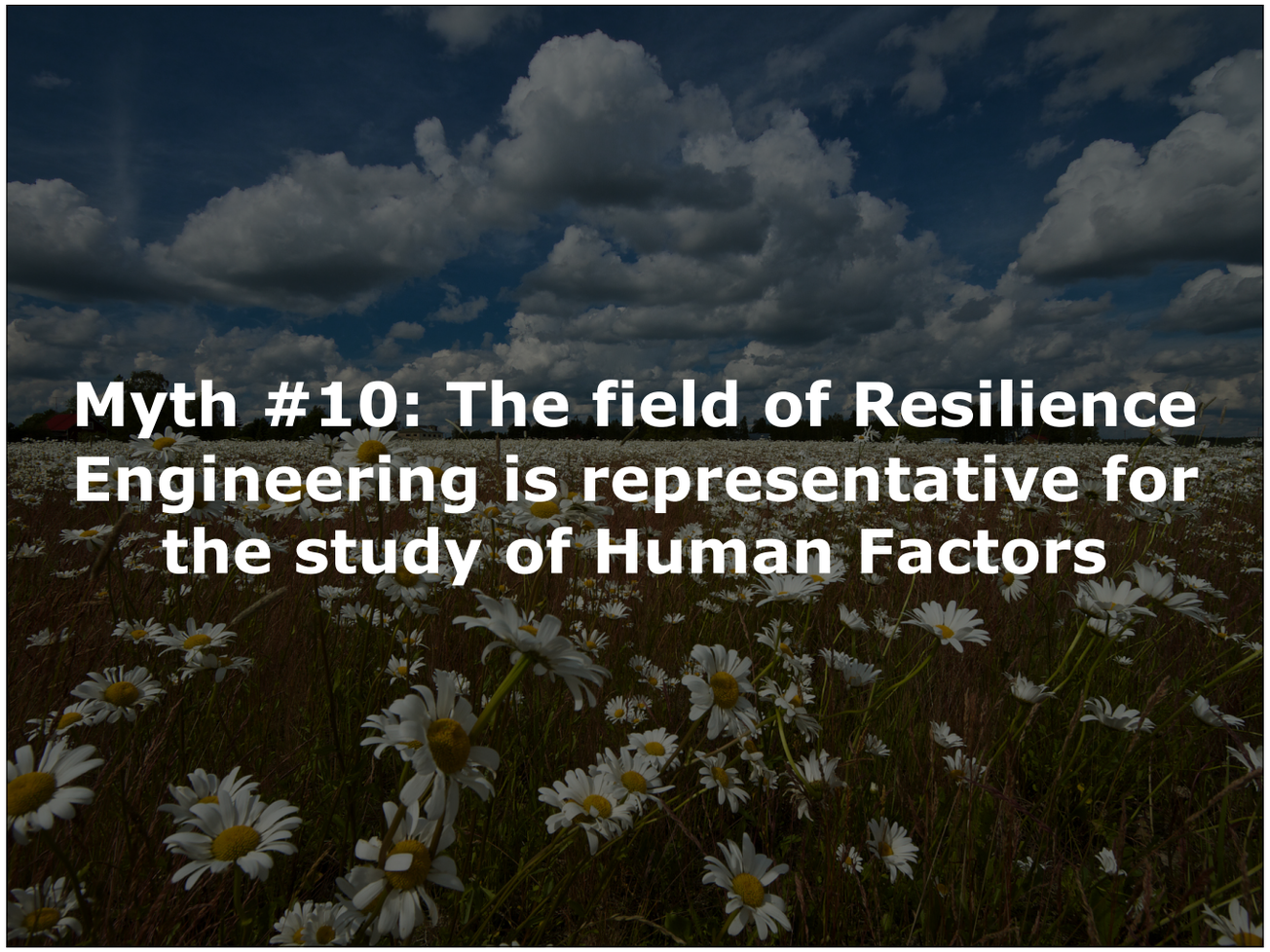
vs

Safety-II (productive)

Safety-I - Look at the errors, react to them

Safety-II - Look at errors, but also where we succeed and the normal, every day events too! Forward thinking (how systems perform, not just how they remain safe). How are we getting *good* at what we do by being “risky”?

You don’t know much about risk by counting errors



Human factors represented by subset of people who tend to look like me. Need to increase diversity of our representation if we want to improve the practice and actually study the interactions of people.

Takeaways

"The ETTO Principle"
Dr. Erik Hollnagel

"How Complex Systems Fail"
Dr. Richard Cook

"The Field Guide to Understanding 'Human Error'"
Dr. Sydney Dekker

Can’t plan for everything

Get experts to
share their expertise

How do they know they need to share it?

Embrace Uncertainty

It’s scary! We can’t put numbers to everything. We can’t prevent all failures or have 100% uptime.

Get comfortable because you *have* to.

Thanks!

@wcgallego

Intro - https://www.flickr.com/photos/11746801@N04/7069259807
Solar system - https://www.flickr.com/photos/pagedooley/16245781464
Space - https://www.flickr.com/photos/gsfc/14411386334
Circuit - https://www.flickr.com/photos/aureobcn/8573429757
Stopwatch - https://www.flickr.com/photos/casualcapture/5386695002
Beakers - https://www.flickr.com/photos/kingway-school/5876407905
Roots - https://www.flickr.com/photos/76988831@N03/6903898695
Caution - https://www.flickr.com/photos/e-coli/3888542890
Supernova - https://www.flickr.com/photos/ernenn/5273529766
Cliffs - https://www.flickr.com/photos/dancehallone/4822678938
Field of flowers - https://www.flickr.com/photos/k3ntfin/7479480266