Lamboozling Attackers

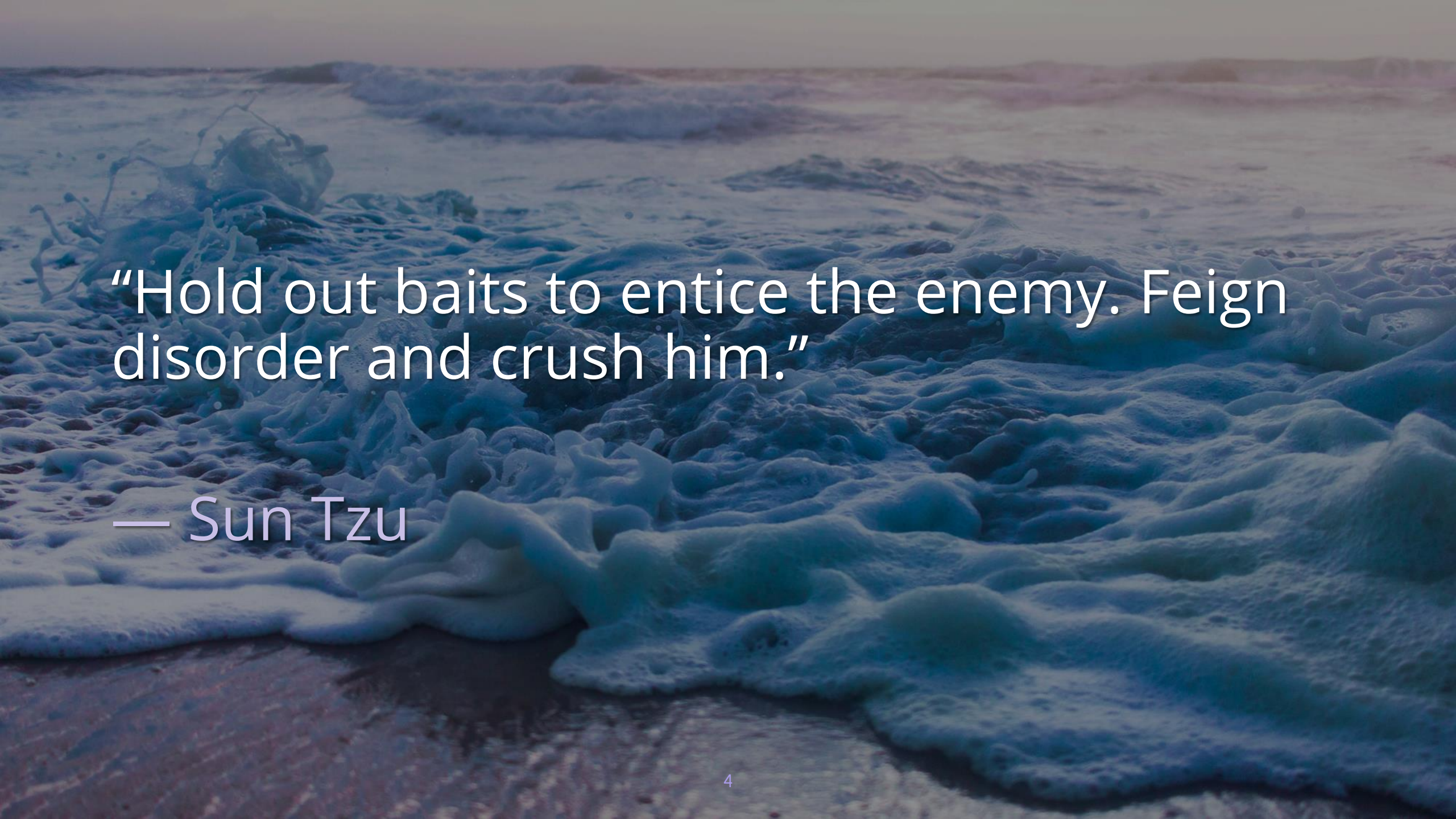
Kelly Shortridge (@swagitda_)
Ryan Petrich (@rpetrich)

Summercon 2022

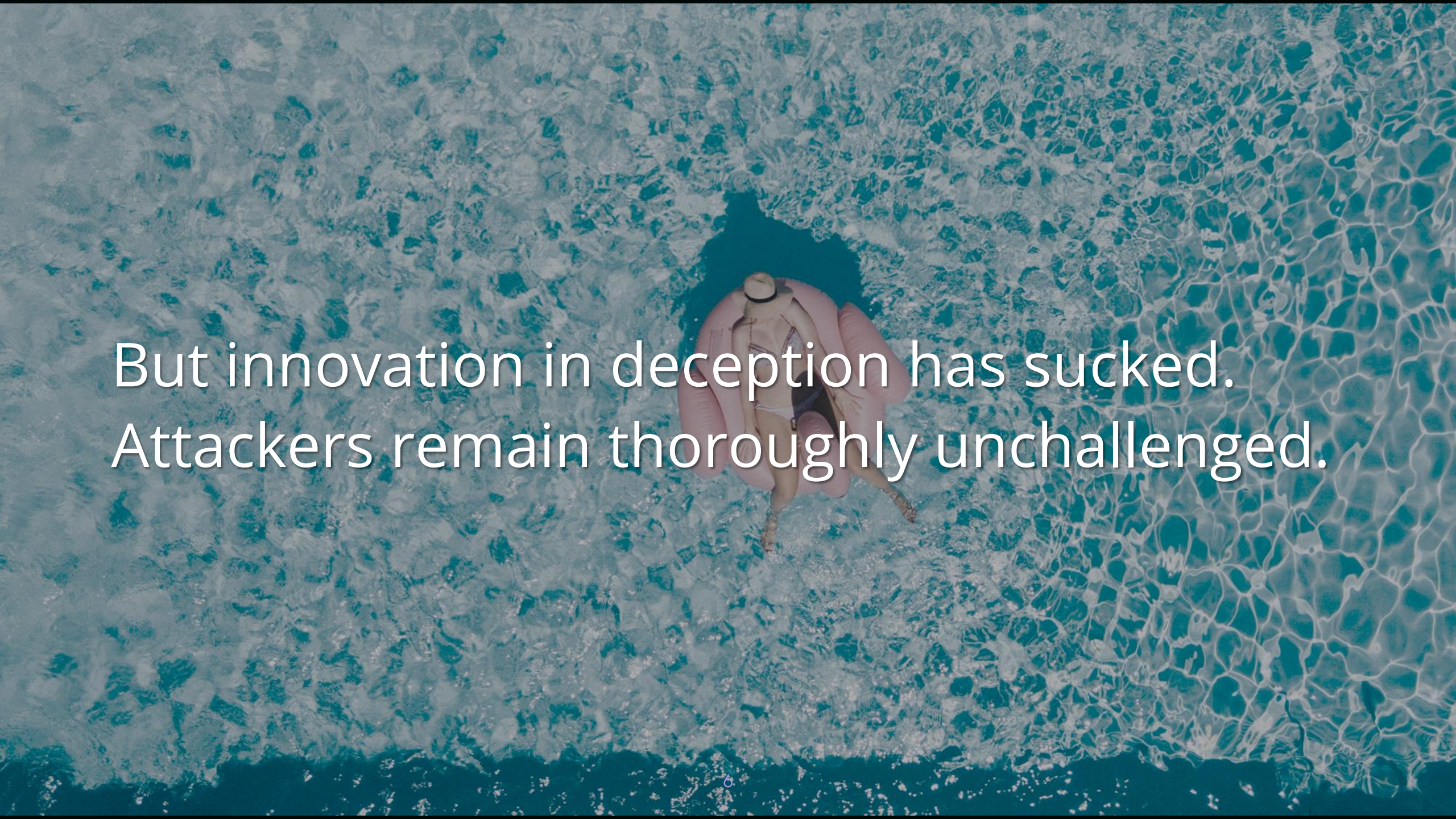
Hi, I'm Kelly

fastly


Hi, I'm Ryan


“Hold out baits to entice the enemy. Feign disorder and crush him.”
— Sun Tzu

Deception is a powerful resilience tactic


But innovation in deception has sucked.
Attackers remain thoroughly unchallenged.


How do we build better deception systems given our goals, constraints, and tradeoffs?

The answer is a new generation of deception systems: deception *environments*

I. Exploiting attacker brains

II. The sucky status quo

III. Deux ex modern computing

IV. Designing deception environments

V. Harvesting potential

VI. Future opportunities


I. Exploiting attacker
brains (for fun & profit)

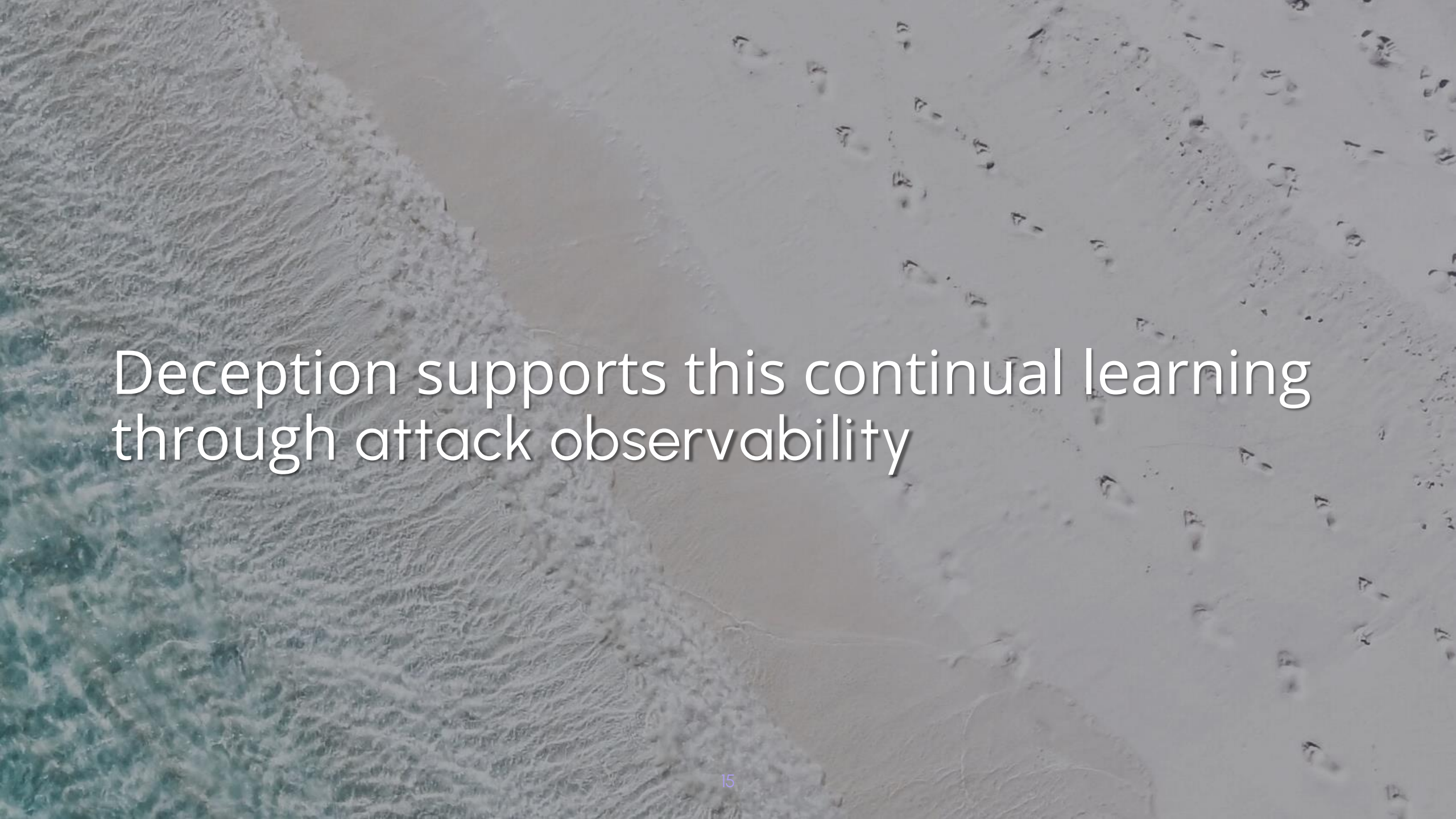
Attackers (plural noun):

humans whose objectives are met by accessing, destabilizing, stealing, or otherwise leveraging other humans' computers without consent

Goal: anticipate this type of negative shock when dev-ing & ops-ing systems

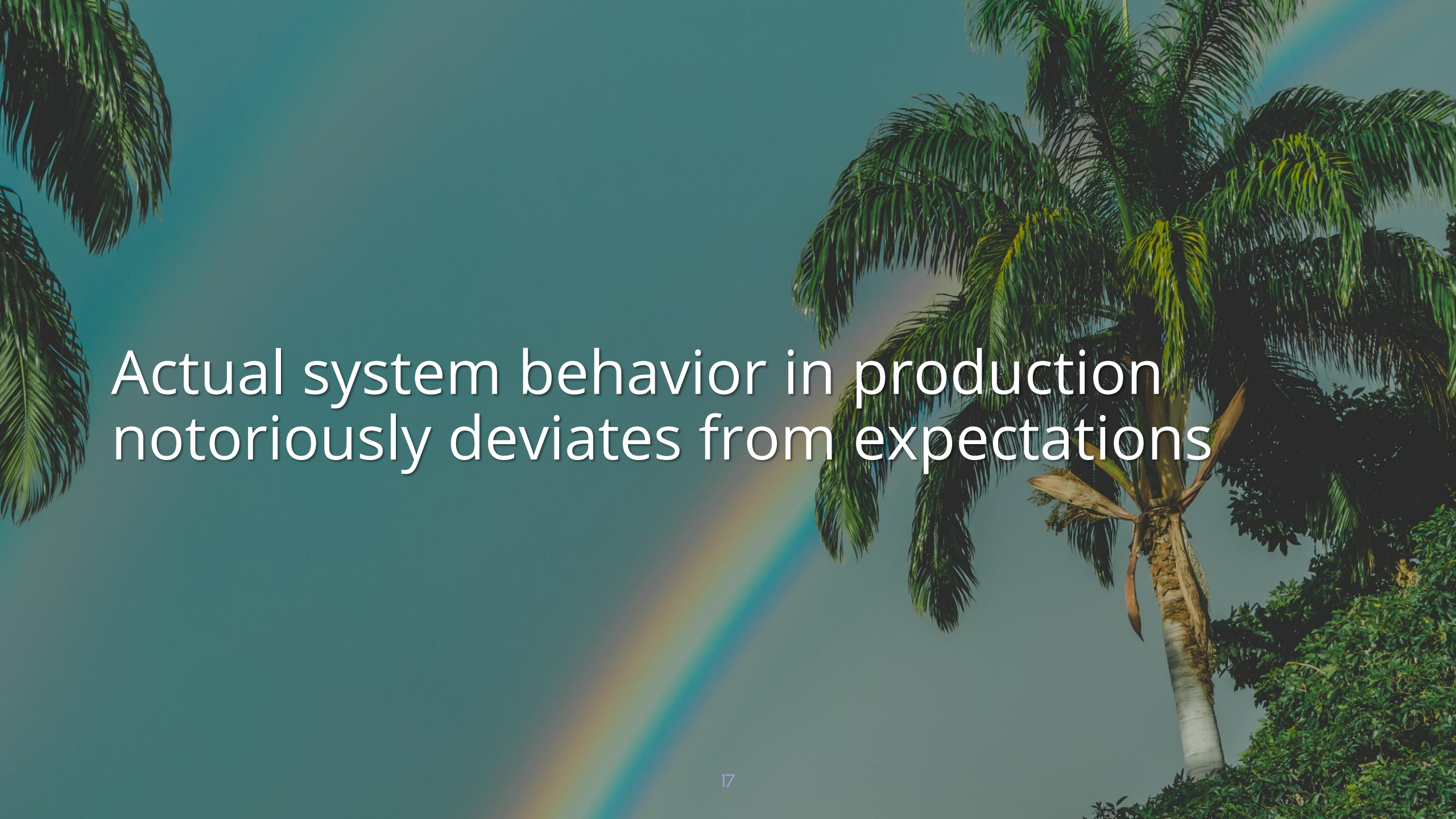
1. Collect relevant info about attackers
2. Implement anticipatory mechanisms that impede the success of attack ops

Sustaining resilience in complex systems requires a continual learning capacity



Deception supports this continual learning through attack observability

Attack Observability: observing the interaction between attackers & systems


Actual system behavior in production
notoriously deviates from expectations

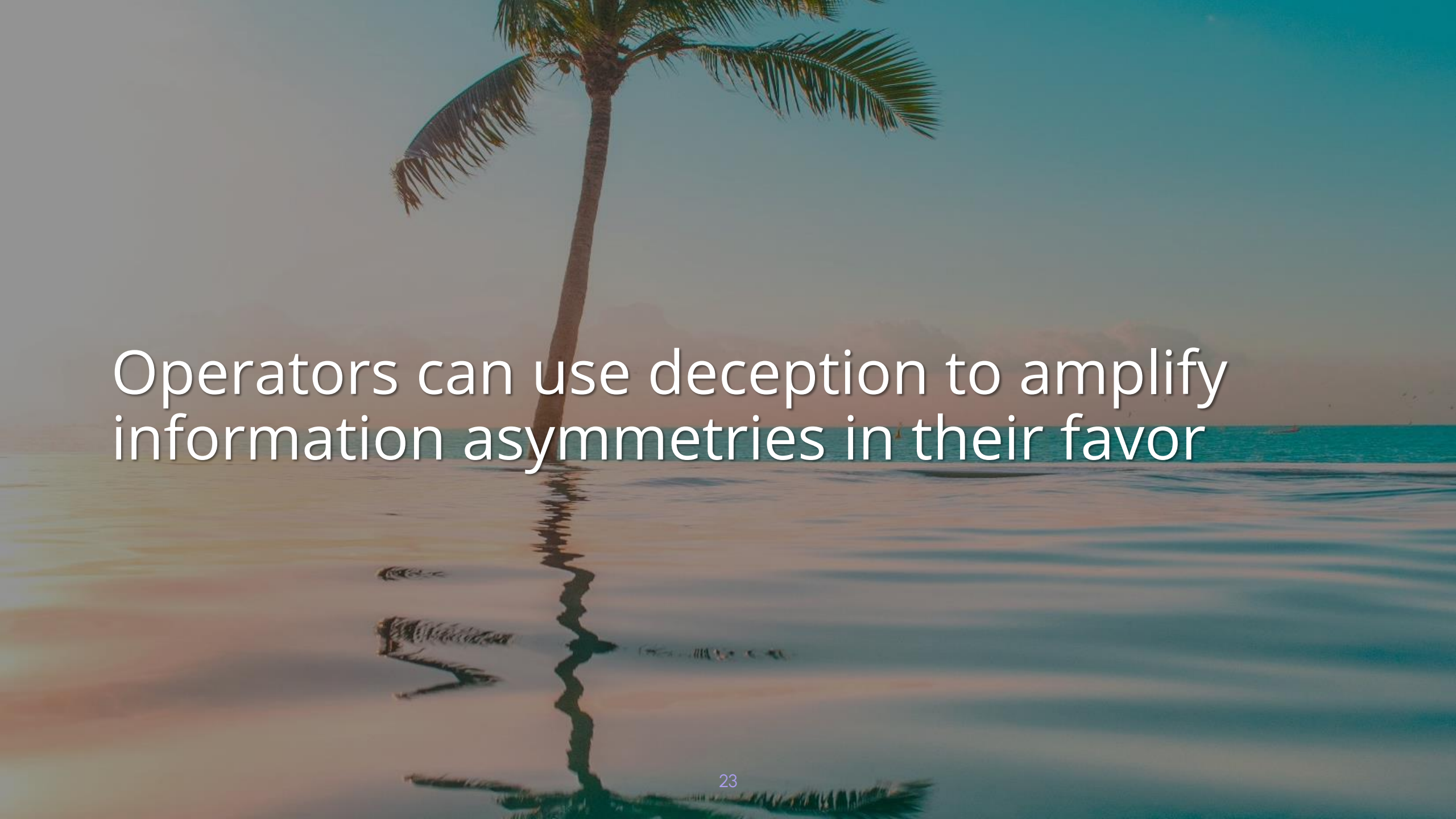
You may have beliefs about attacker behavior, but does it match reality?


To understand attackers, we need to understand how humans learn & decide

Human learning & decision-making are tightly coupled == exploit opportunity

Information asymmetry leads to core advantages for one “side” of the game

Each side chooses a plan based on pre-existing beliefs + learned experience



Operators can use deception to amplify information asymmetries in their favor

Make attacker experiences unreliable; poison the attacker’s learning process

Deception systems help exacerbate info asymmetry in two dimensions...

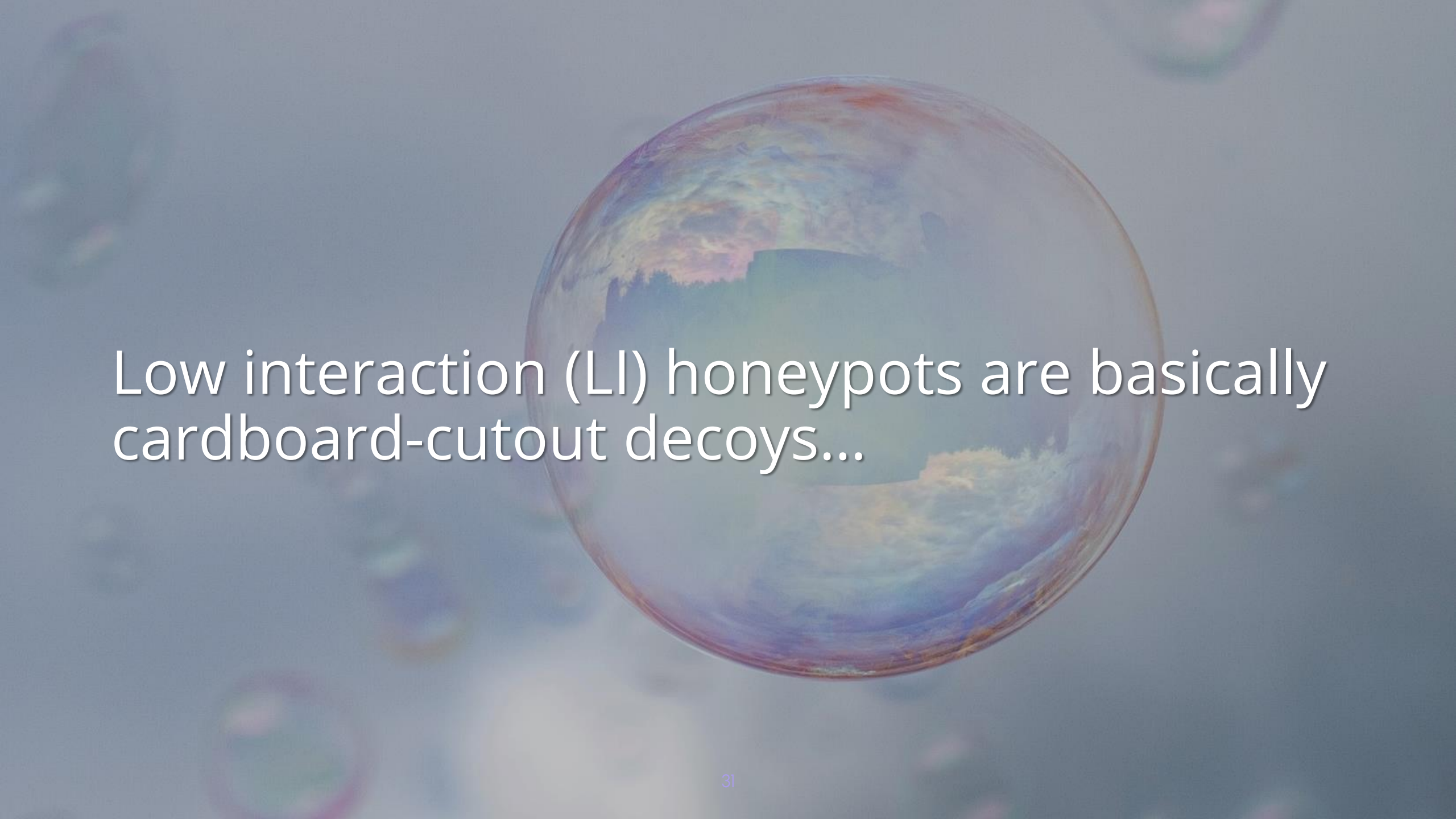
1. Expose real-world data on attackers' thought processes (increasing the value of info for operators)

2. Manipulating info to disrupt attackers' abilities to learn & make decisions (reducing the value of info for attackers)


II. The sucky status quo

Honeypots are the status quo for the art of deception and never really grew up…

What kinds of honeypots are there?



Low interaction (LI) honeypots are basically cardboard-cutout decoys…

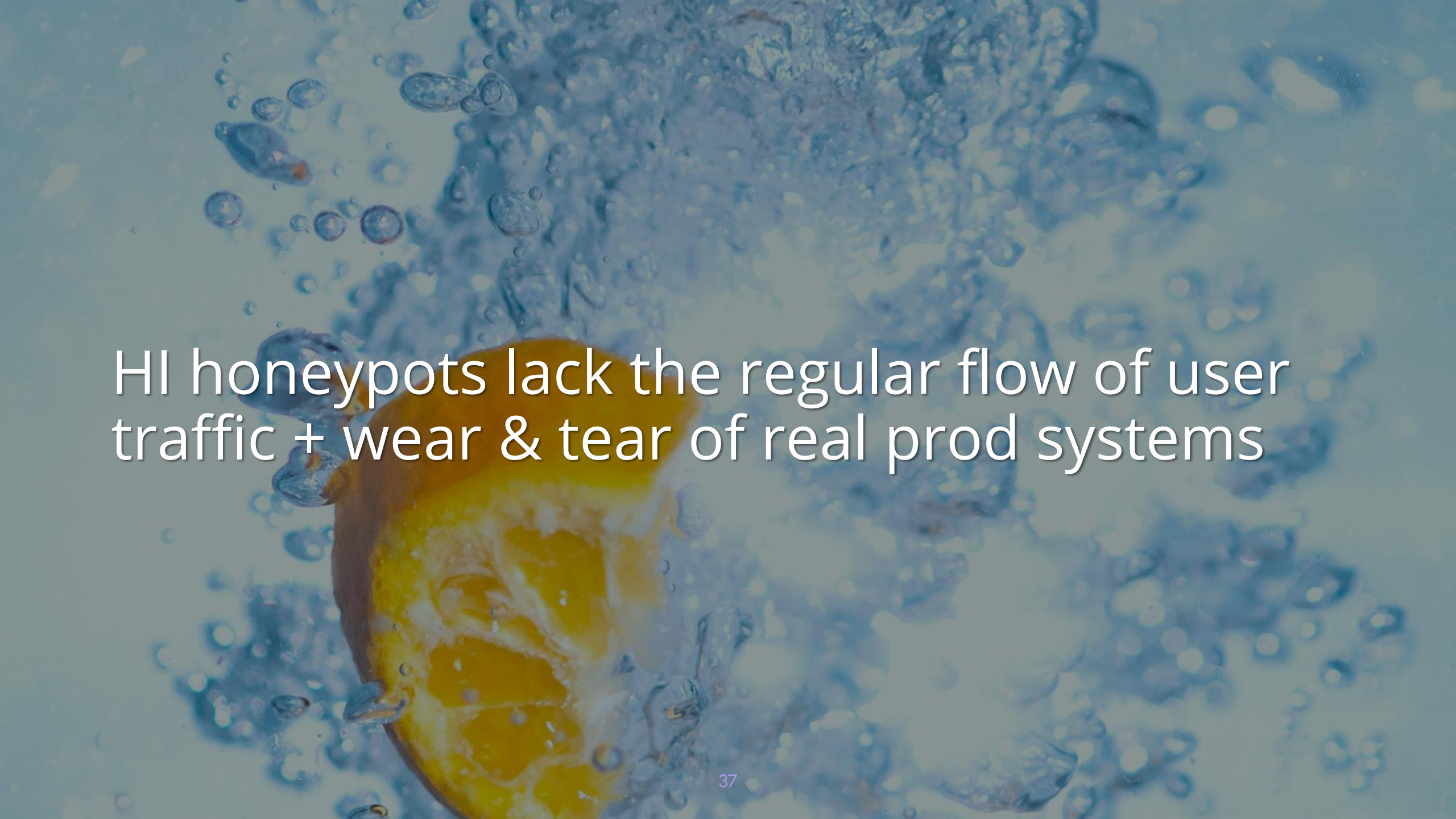
Medium interaction (MI) honeypots imitate a specific system without meaningful depth

High interaction (HI) honeypots are just vulnerable copies of services…

LI & MI honeypots are ineffectual af at deceiving attackers so we can dismiss them

Even HI honeypots are unconvincing to attackers with a modicum of experience

“Does the system feel real?” (no)
“Does it lack activity?” (yes)



HI honeypots lack the regular flow of user traffic + wear & tear of real prod systems

P.S. a fundamental flaw of honeypots is that they’re controlled by infosec people...

III. Deux ex modern computing

We really *need* a new generation of deception given its potential for resilience

Deception Environments are this new gen and differ both in design & ownership

Attackers have expertise in attacking *systems* – so no wonder the status quo fails

Deception *environments* (DEs) are possible with new types of computing + new owners

Goal of traditional honeypots = frequency of scanning tools or exploiting known vulns

DEs observe attacker behavior through all operational stages + experiment on them

What parts of modern infra help lower costs & improve deception design efficacy?

Cloud computing – the ability to provision fully isolated infra with little expense

Deployment automation + defining infra declaratively decreases ops overhead

Virtualization advancements: isolation, observability, denser computing

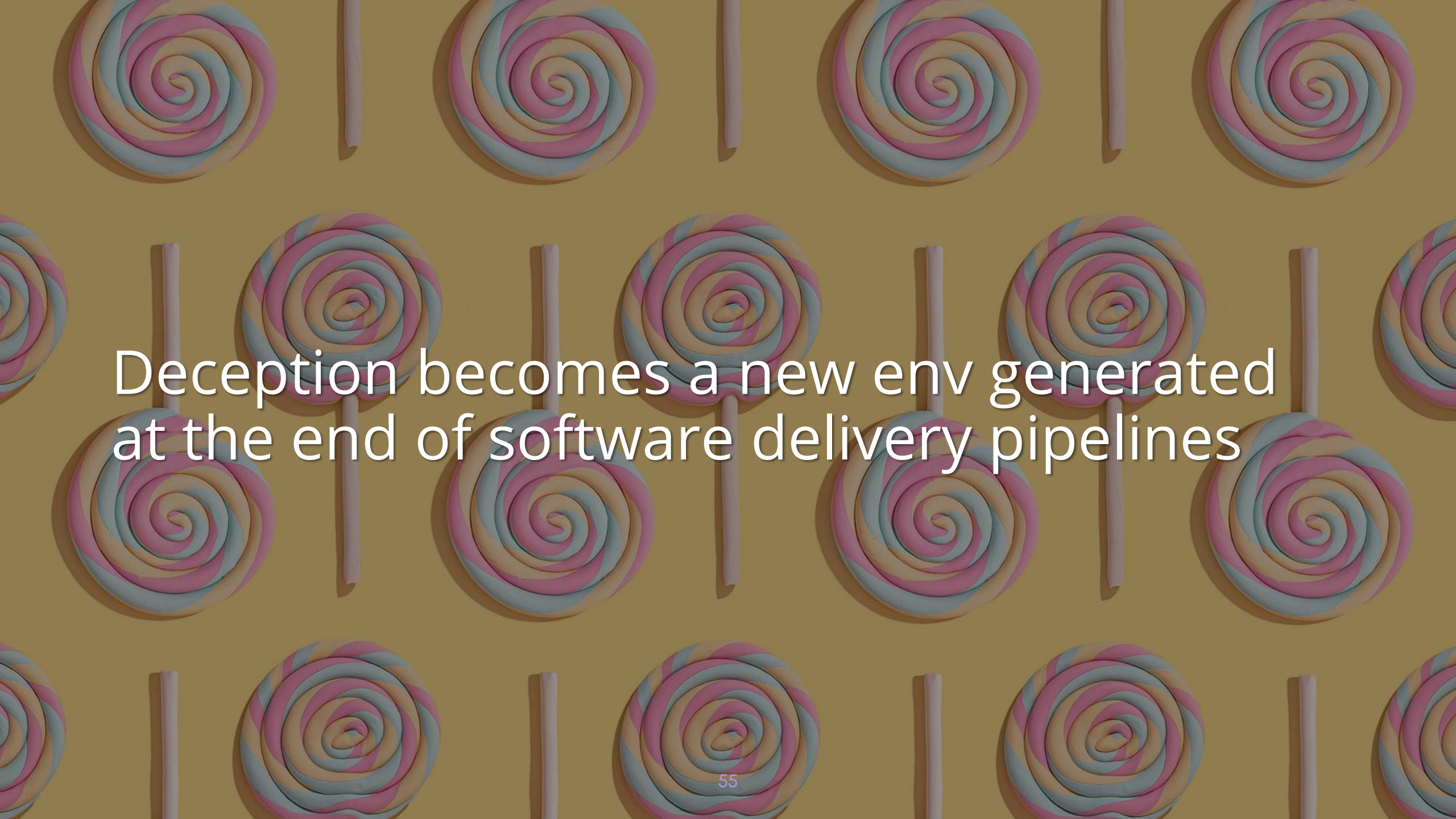
SDN proliferation enables isolated network topology dedicated to attackers

Ownership should be based on systems design expertise, not security expertise

SWEs can repurpose deployment templates to build unique, powerful deception envs


IV. Designing deception environments

DE design philosophy: repurpose the design, assets, & templates of a real system

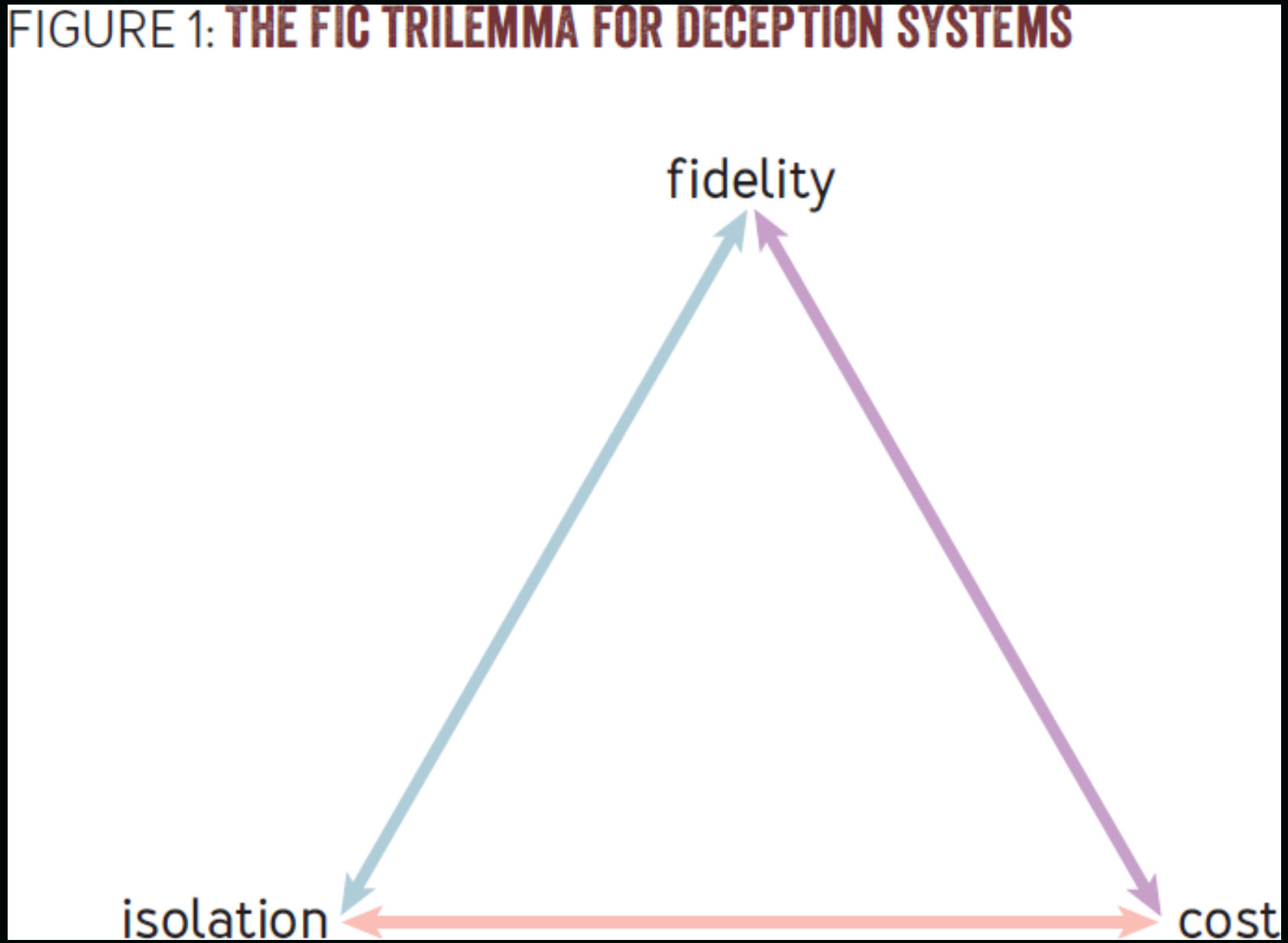

Deception becomes a new env generated
at the end of software delivery pipelines

We can preempt attacker skepticism by designing a DE that feels “lived in”

Starting with the design of a real prod system == realism + more relevant insights

The F.I.C. trilemma: fidelity, isolation, cost

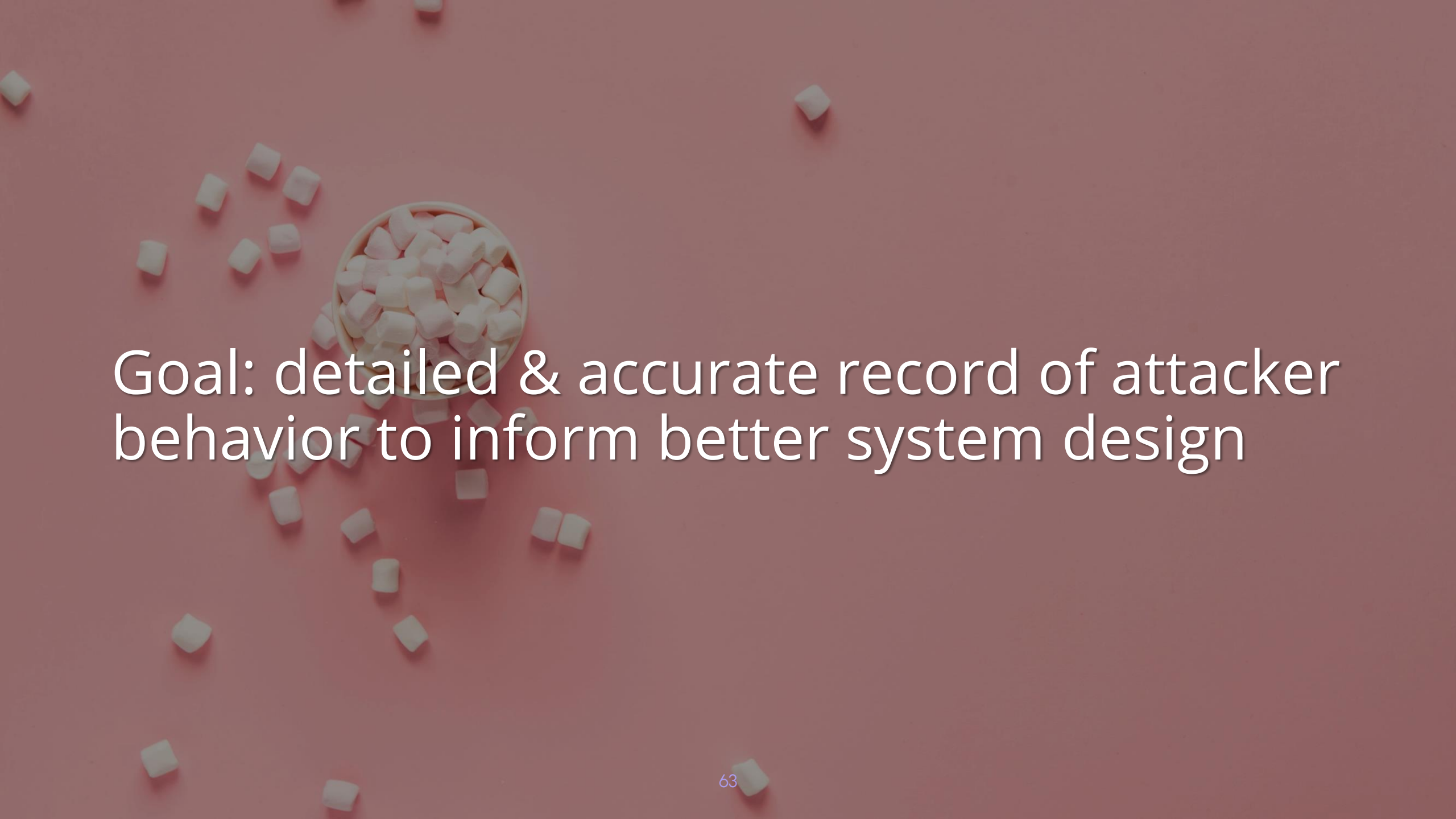
FIGURE 1: THE FIC TRILEMMA FOR DECEPTION SYSTEMS



Fidelity: credibility to attackers and ability to support attack observability

Attackers expect to see things like a service running, prod-like traffic, coordinating with other services, orchestration, monitoring...

Deception systems need sufficient depth to stimulate extended attacker activity

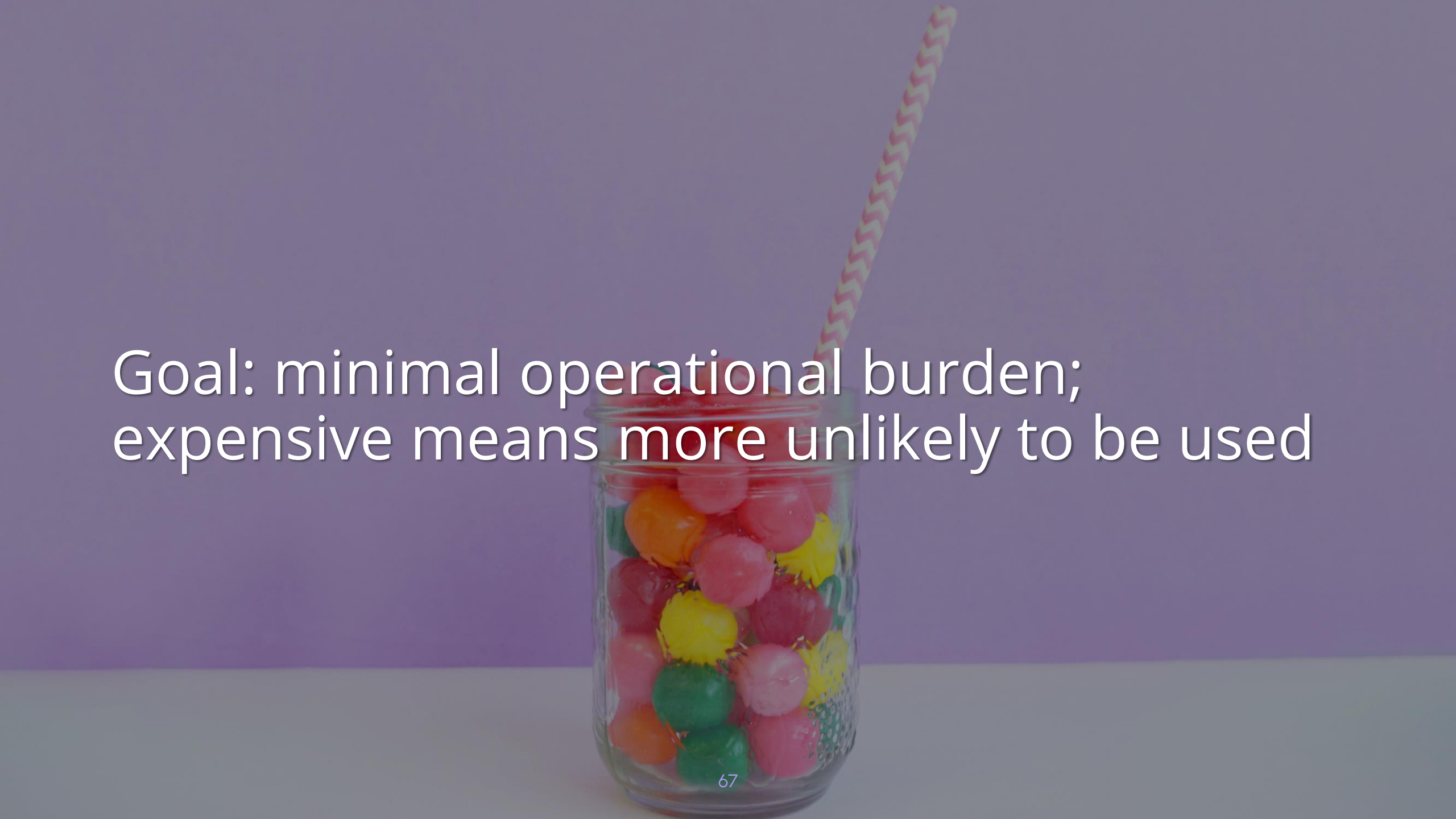


Goal: detailed & accurate record of attacker behavior to inform better system design

Isolation: degree to which the deception system is isolated from the real env or data

Goal: not jeopardizing availability of the real system or data privacy

Cost: computing infra + operational overhead required to deploy & maintain



Goal: minimal operational burden; expensive means more unlikely to be used

FIGURE 2: EXAMPLE DECEPTION SYSTEMS MAPPED TO THE FIC TRILEMMA

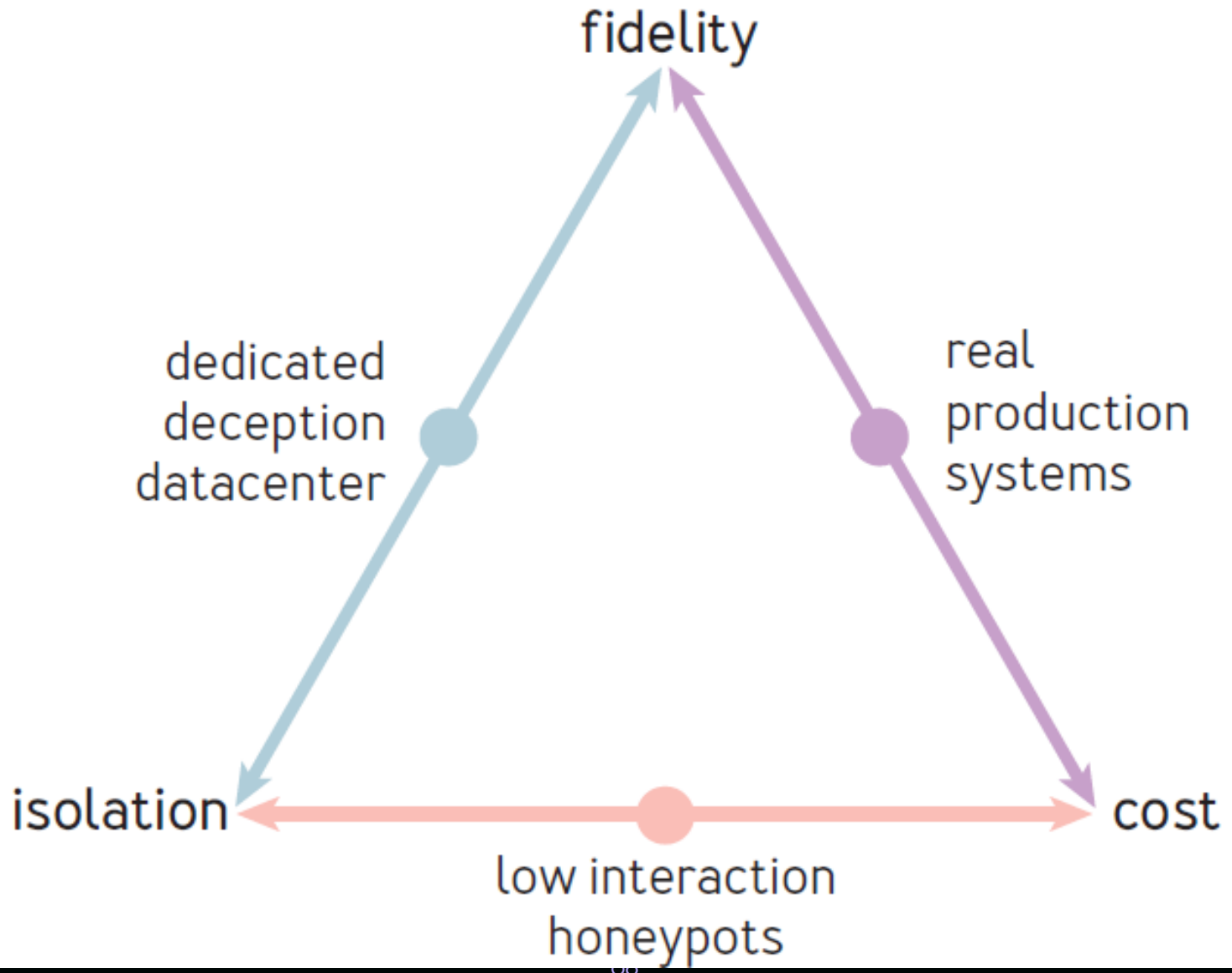
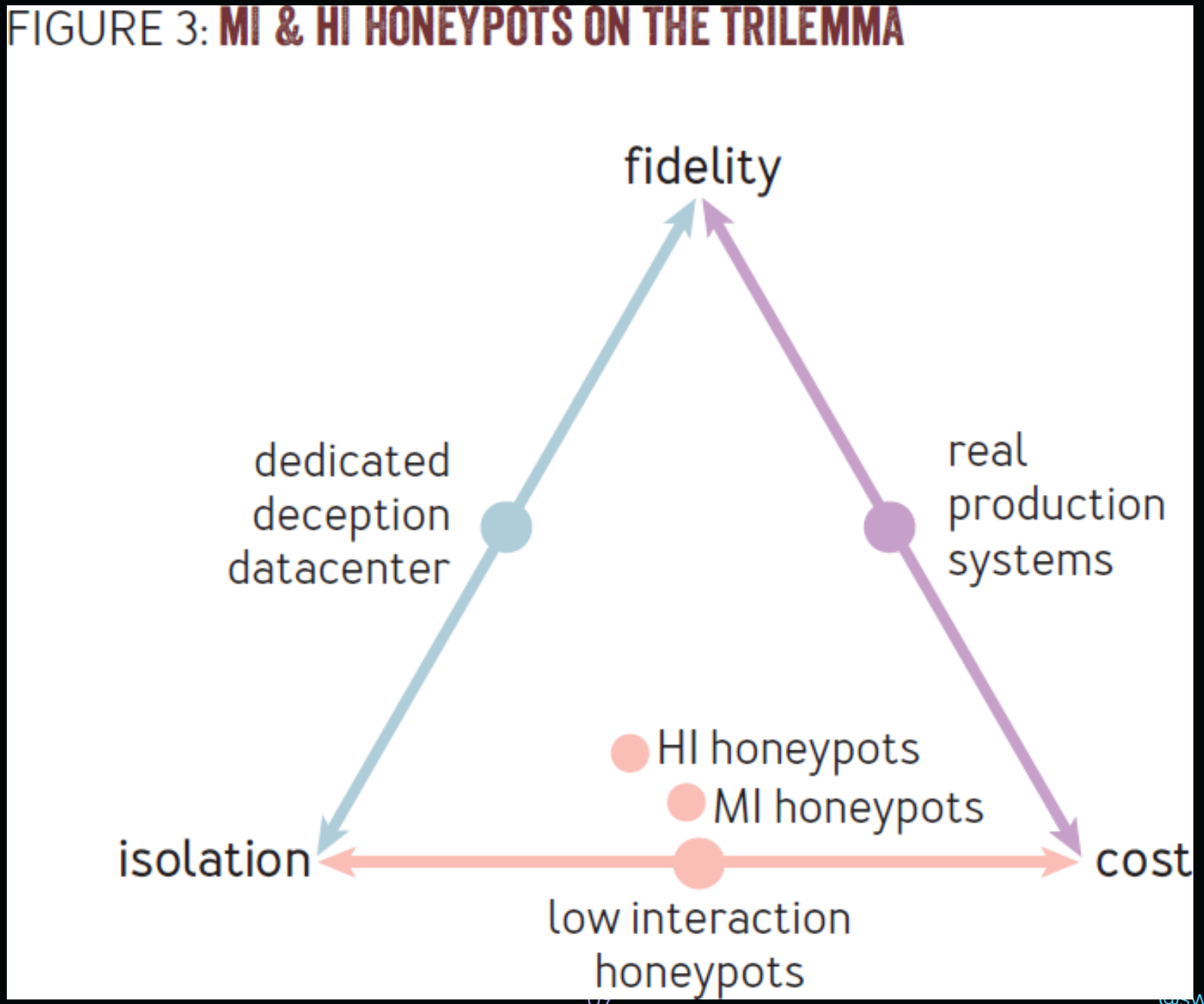
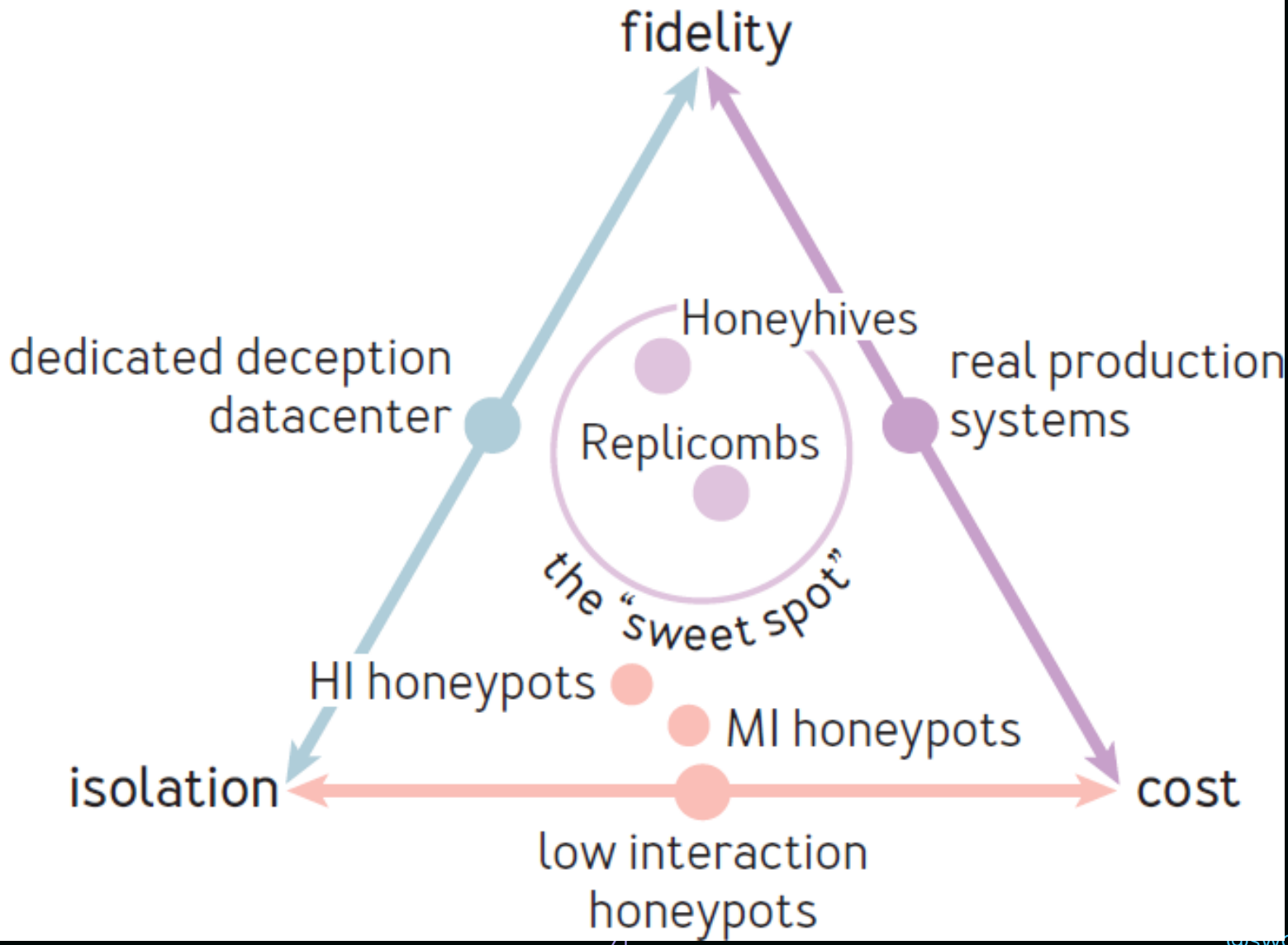


FIGURE 3: MI & HI HONEYPOTS ON THE TRILEMMA



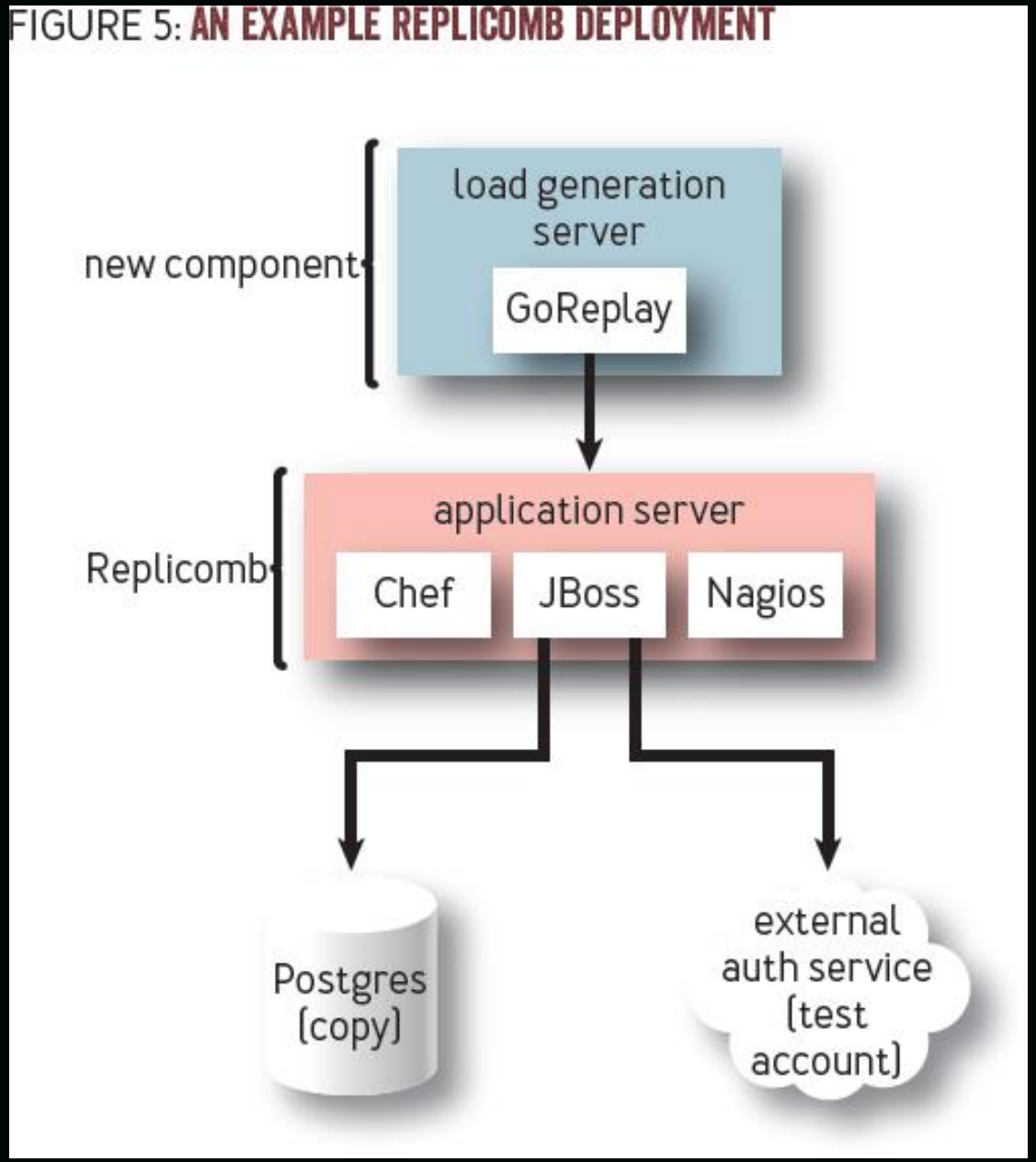
But there is a previously unexplored “sweet spot” for deception mechanisms…

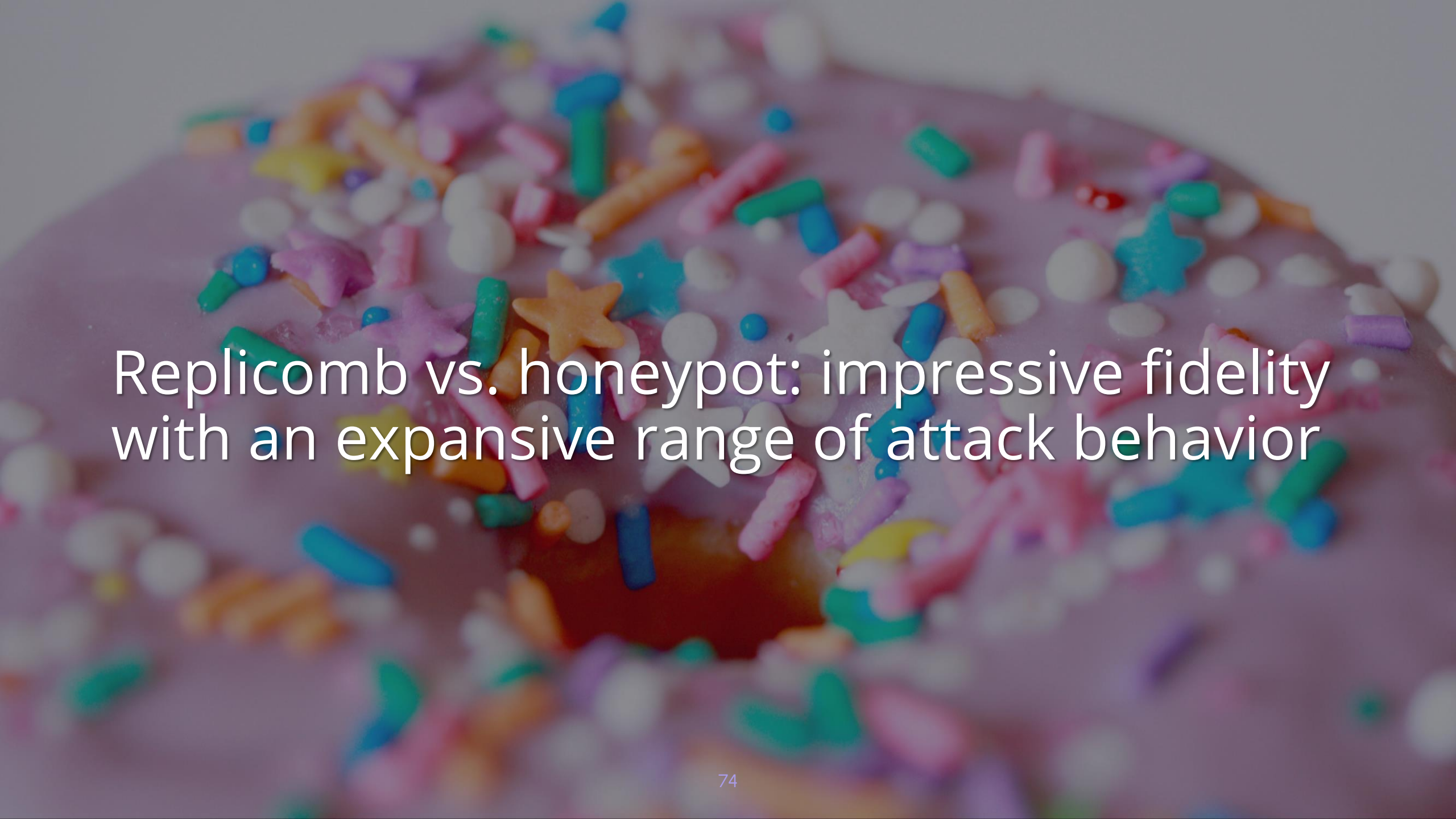
FIGURE 4: THE FIC "SWEET SPOT": HONEYHIVES AND REPLICOMBS



Replicombs: downgraded replicas of prod hosts with the same services seen in prod

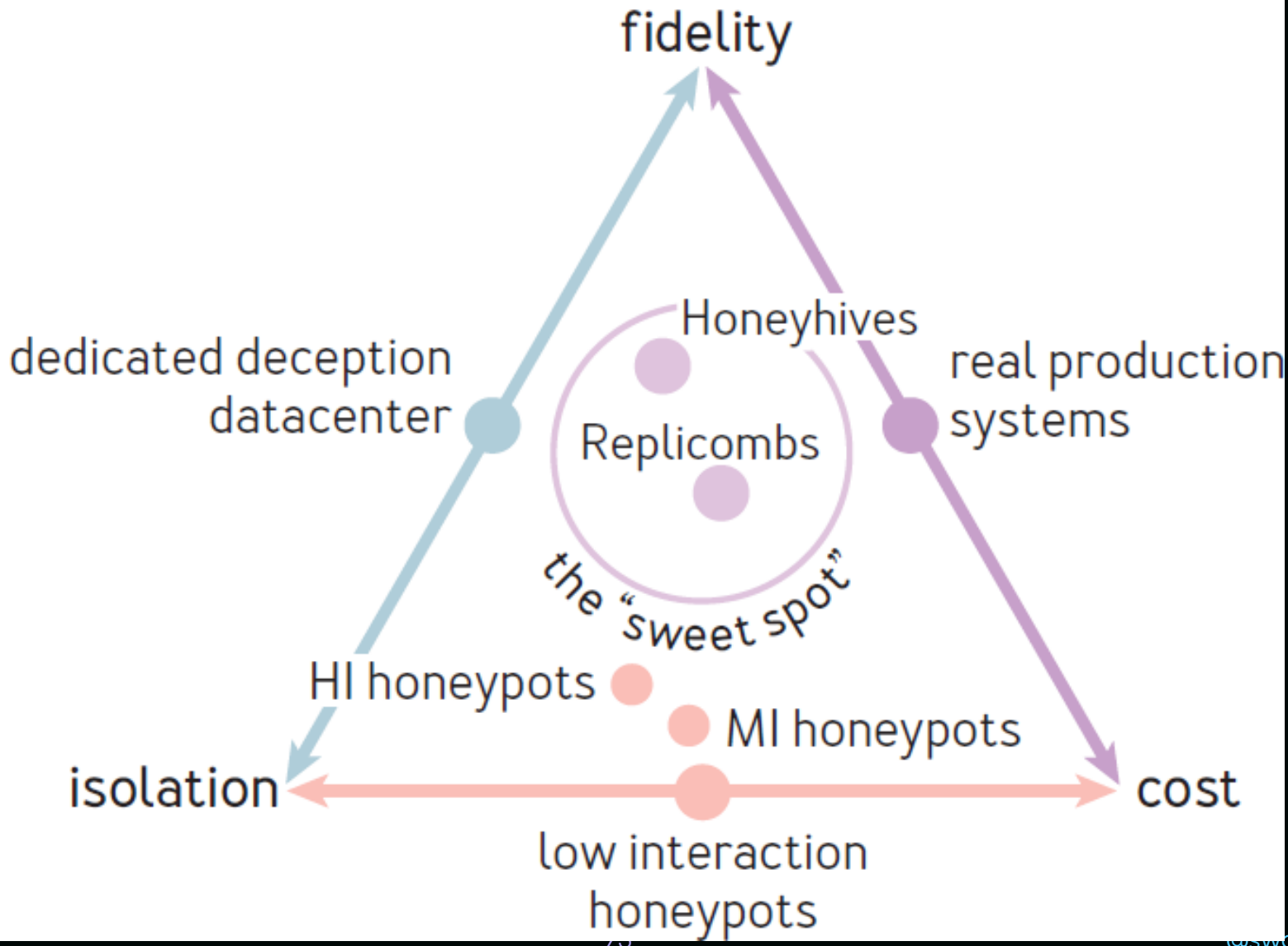
FIGURE 5: AN EXAMPLE REPLICOMB DEPLOYMENT





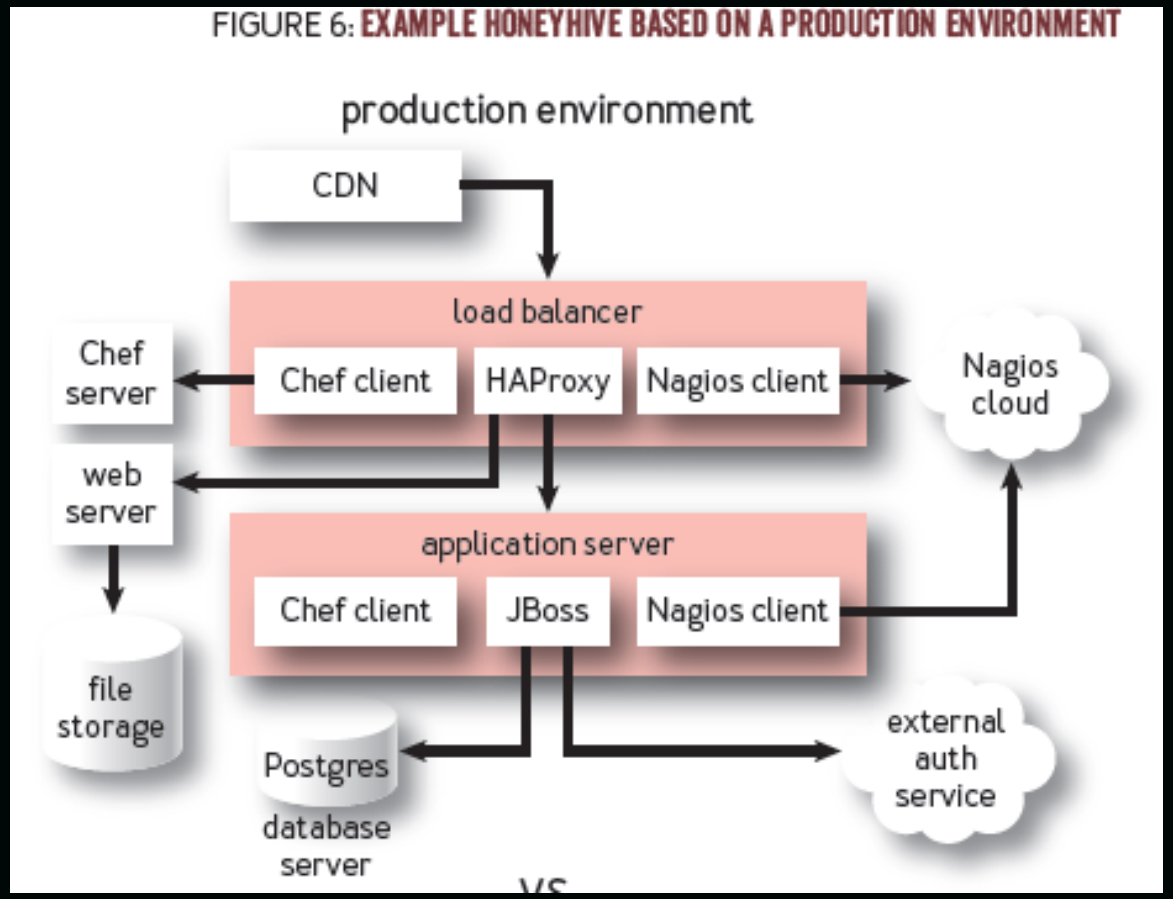
Replicomb vs. honeypot: impressive fidelity with an expansive range of attack behavior

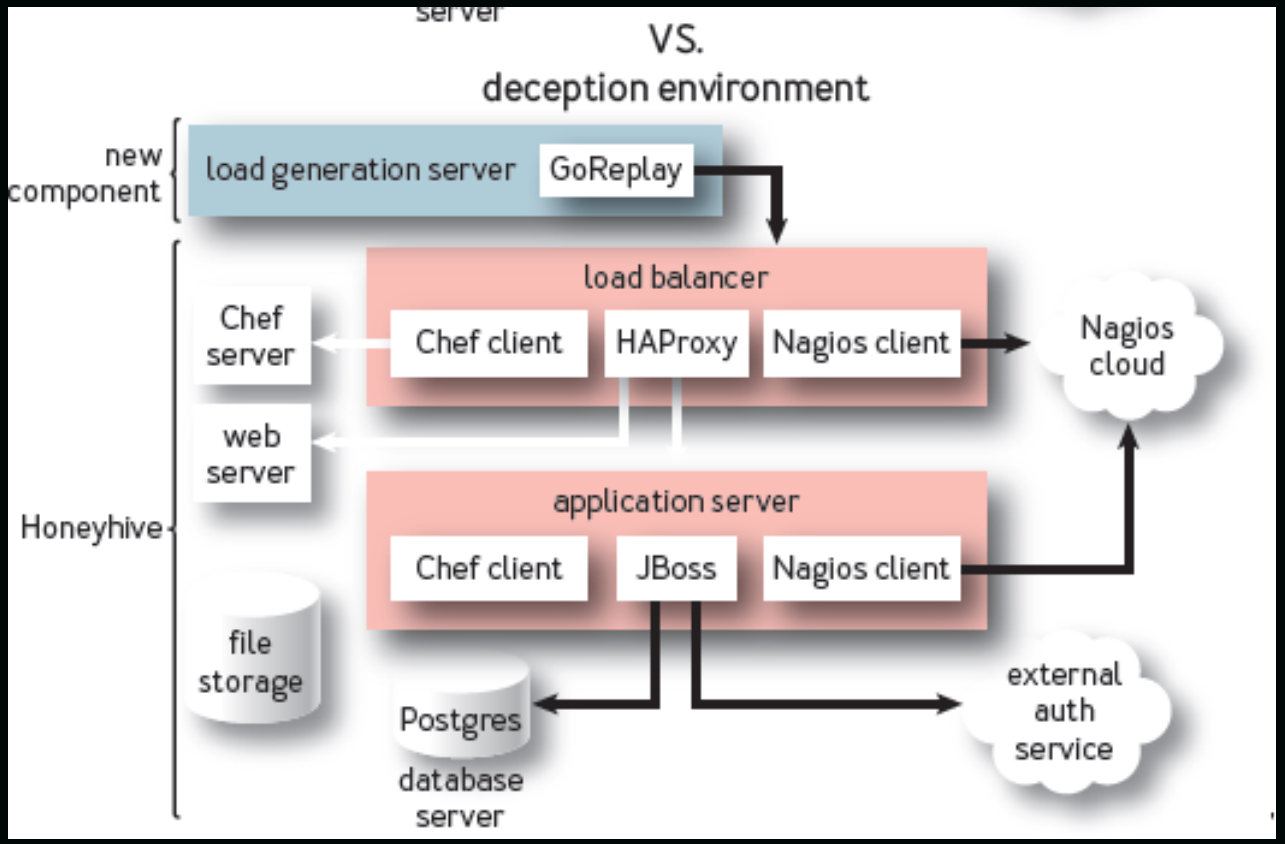
FIGURE 4: THE FIC "SWEET SPOT": HONEYHIVES AND REPLICOMBS



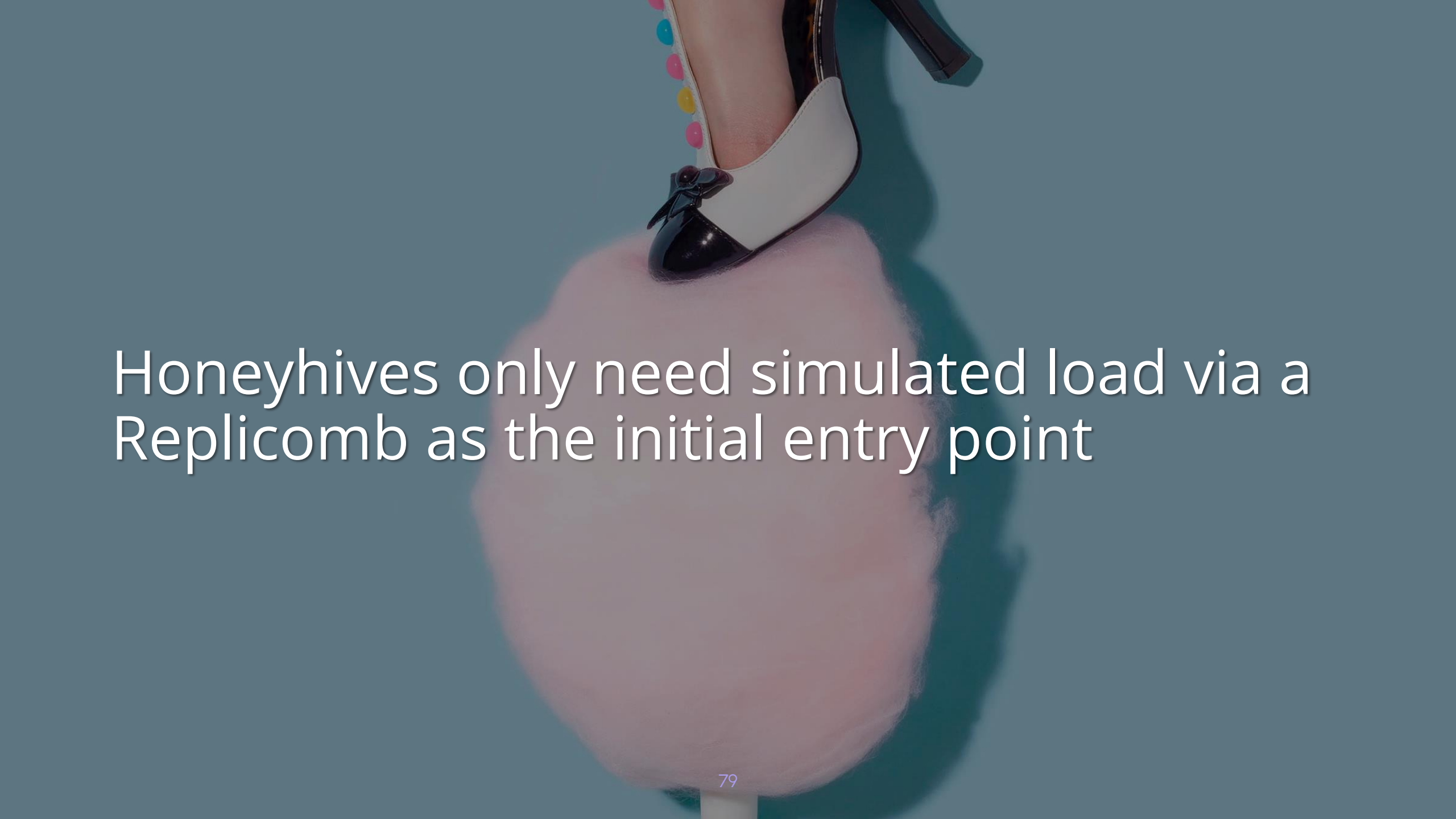
Honeyhives: full network of like-prod hosts to observe attacker movement x-system

FIGURE 6: EXAMPLE HONEYHIVE BASED ON A PRODUCTION ENVIRONMENT





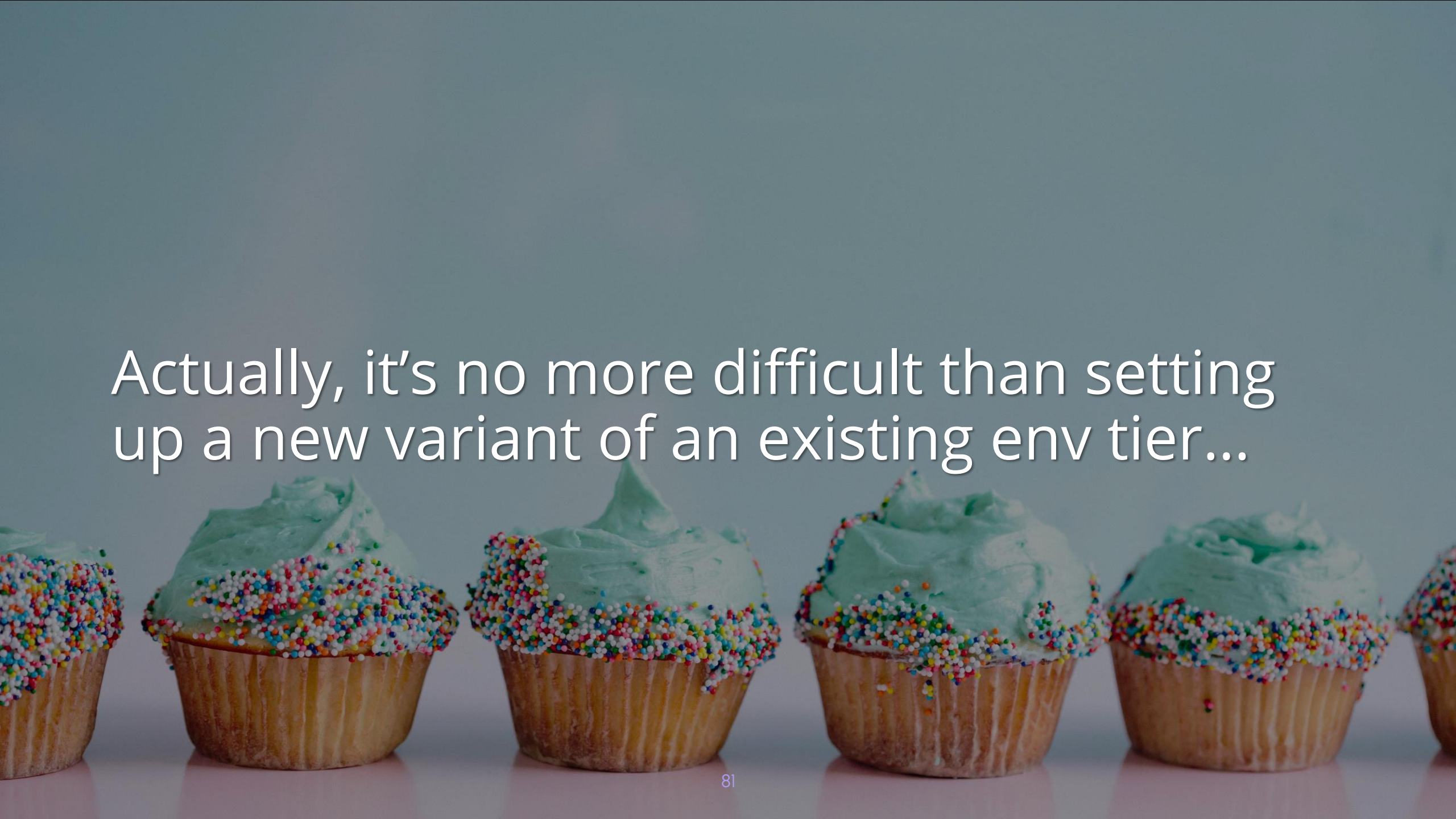
Modern IaC practices + inexpensive full isolation via cloud computing are key



Honeyhives only need simulated load via a Replicomb as the initial entry point

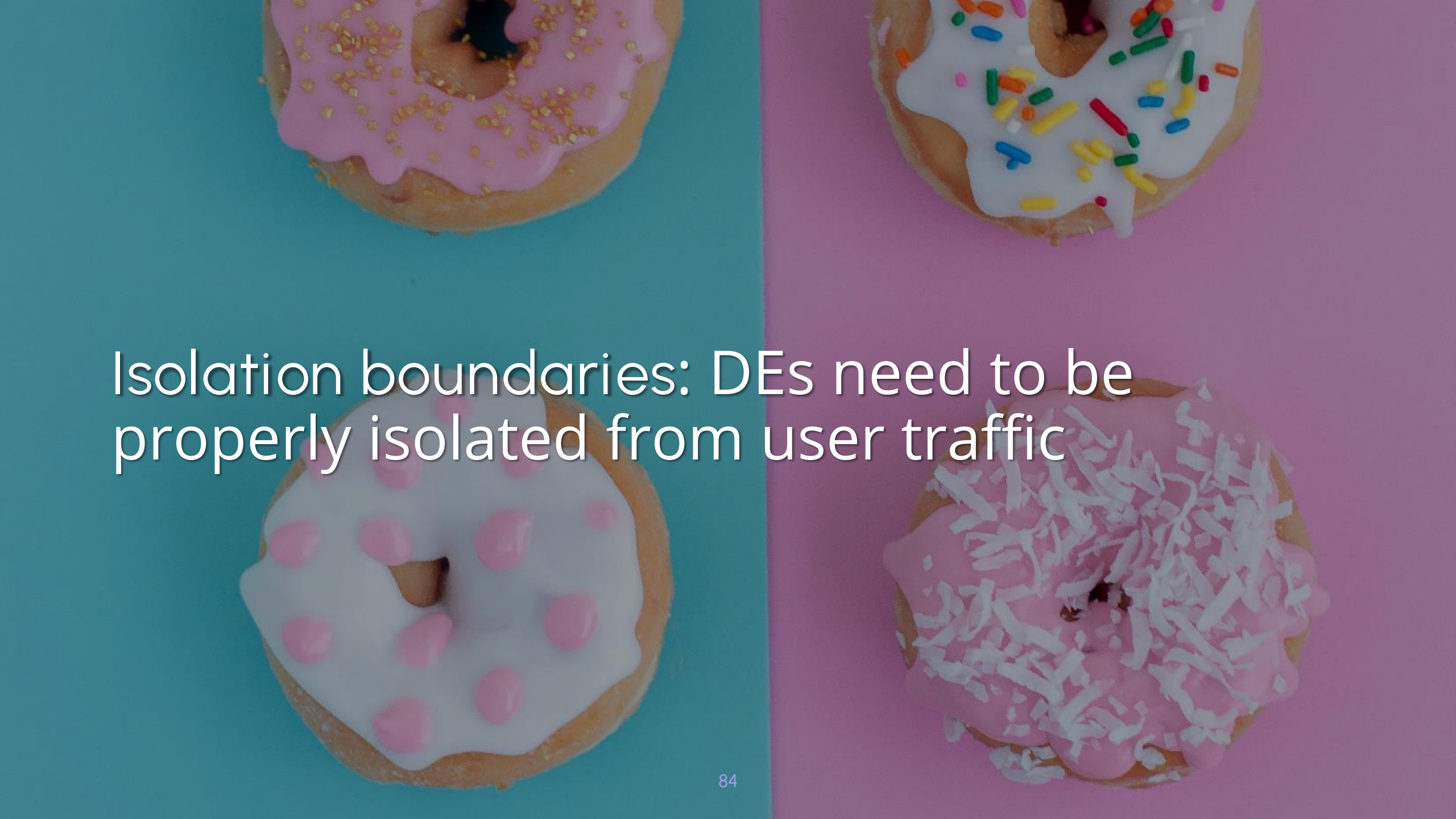
Okay, but how tf do you implement this in the real world of messy software eng?

Actually, it’s no more difficult than setting up a new variant of an existing env tier…



Replicomb is similar to a canary release.
Honeyhive is like a soak or load test env.

But there are details to consider when implementing this in your org…



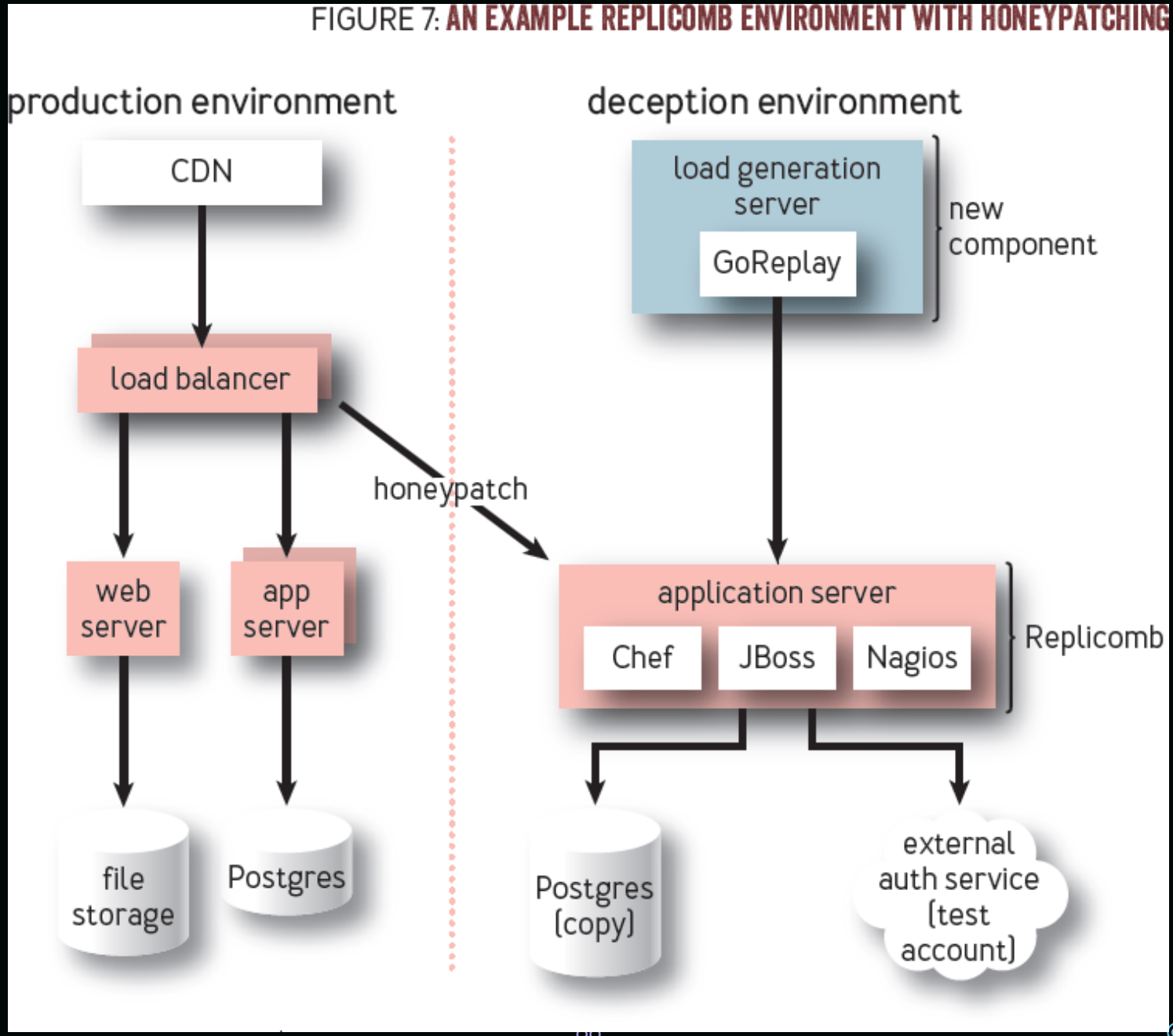
Isolation boundaries: DEs need to be properly isolated from user traffic

Virtualization, SDNs, cloud computing can help create fully isolated networks for DEs

Discoverability: attackers need to find the DE for you to collect real data on their ops

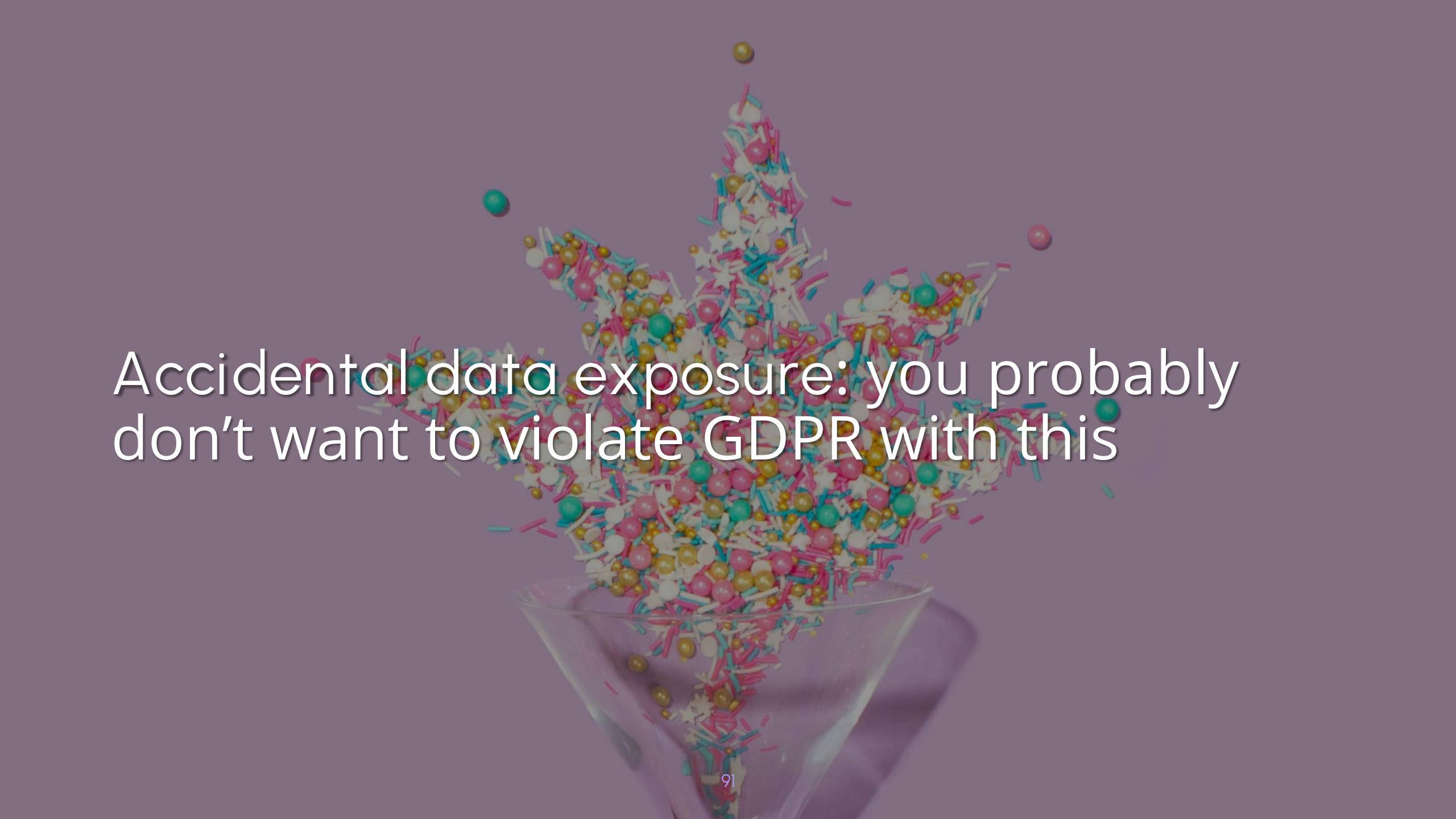
Honeypatching can support discoverability

FIGURE 7: AN EXAMPLE REPLICOMB ENVIRONMENT WITH HONEYPATCHING



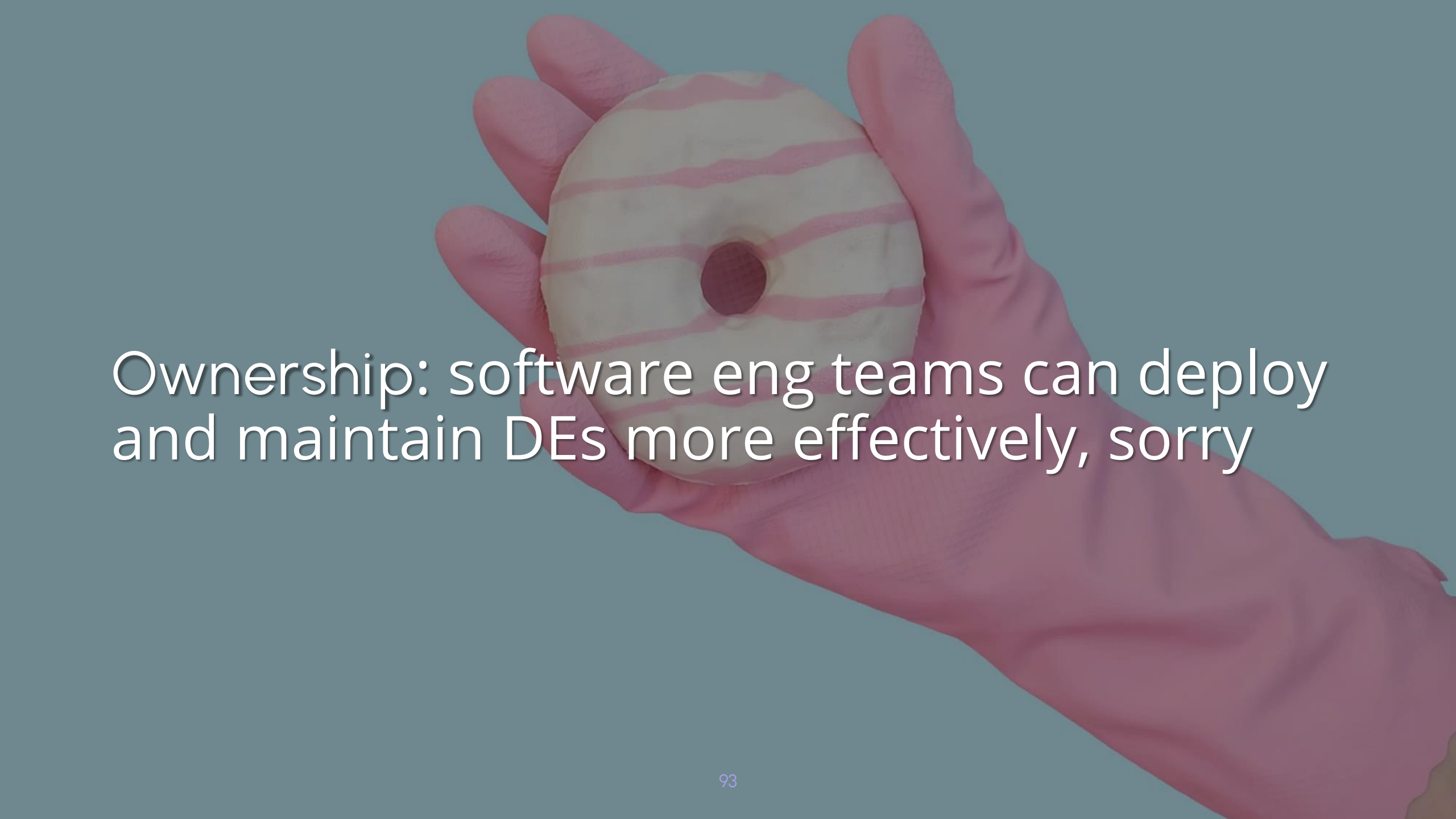
Tamper-free observation: tracing should be invisible to attackers + resistant to tamper

Traffic archiving, memory & disk snapshots, process launch events, file activity…



Accidental data exposure: you probably don’t want to violate GDPR with this

Mitigation: anonymize or scramble traffic or generate synthetic data sets to replay



Ownership: software eng teams can deploy and maintain DEs more effectively, sorry

SWEs can treat attackers as a kindred engineer with the exact opposite goals

V. Harvesting potential

1. Resilient system design
2. Attacker tracing
3. Experimentation platform

Resilient system design

DEs let you explore how attacks impact systems to inform design improvements

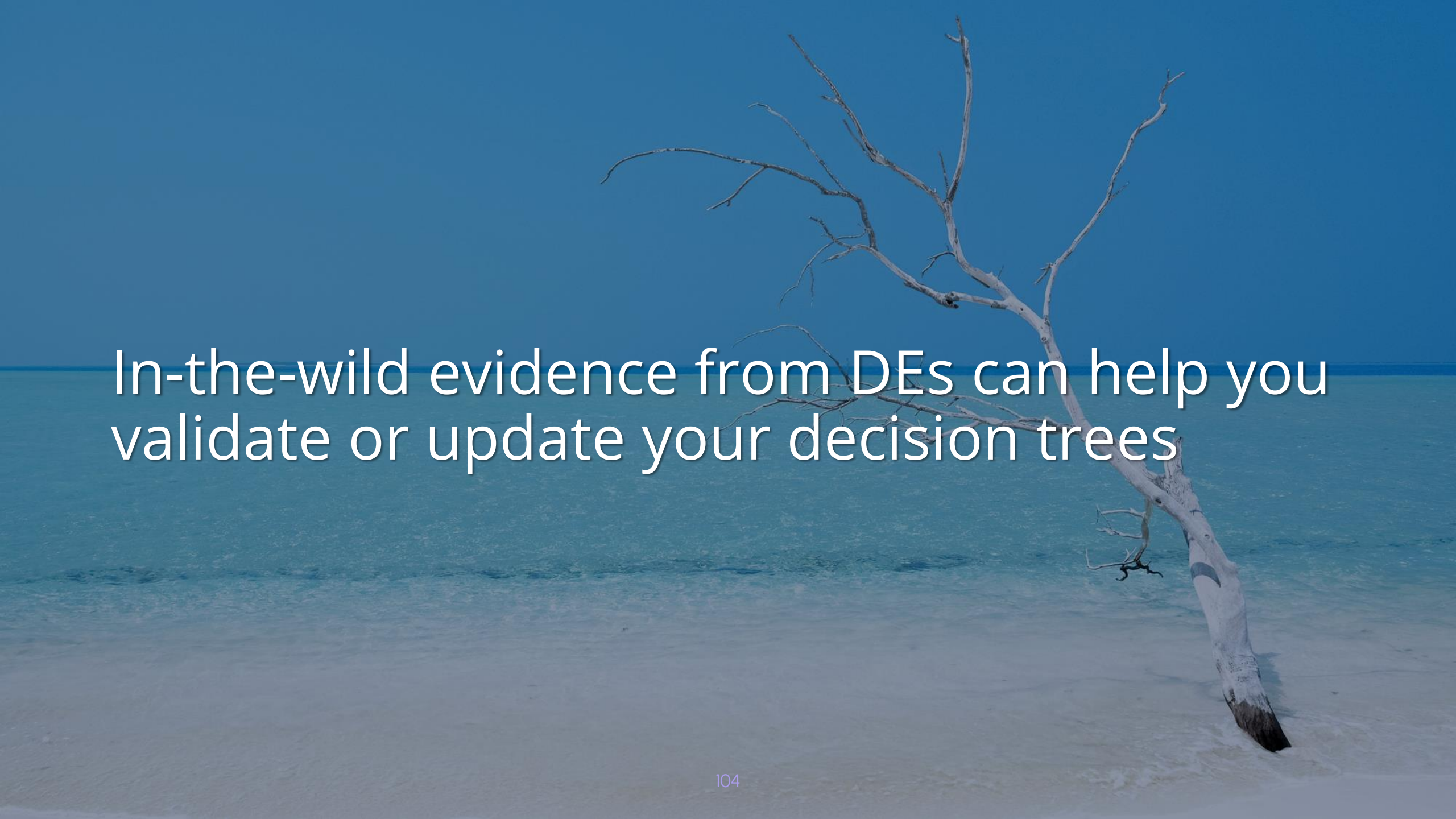
Attackers interact with monitoring, logging, alerting, failover, and service components in ways that stress their overall reliability

DEs expose opportunities for architectural improvements in operability & simplicity

Eng teams can leverage a feedback loop fueled by real-world evidence from DEs


Attacker tracing

Attack observability enables pragmatic threat modeling during design & planning



In-the-wild evidence from DEs can help you validate or update your decision trees

Text Editor

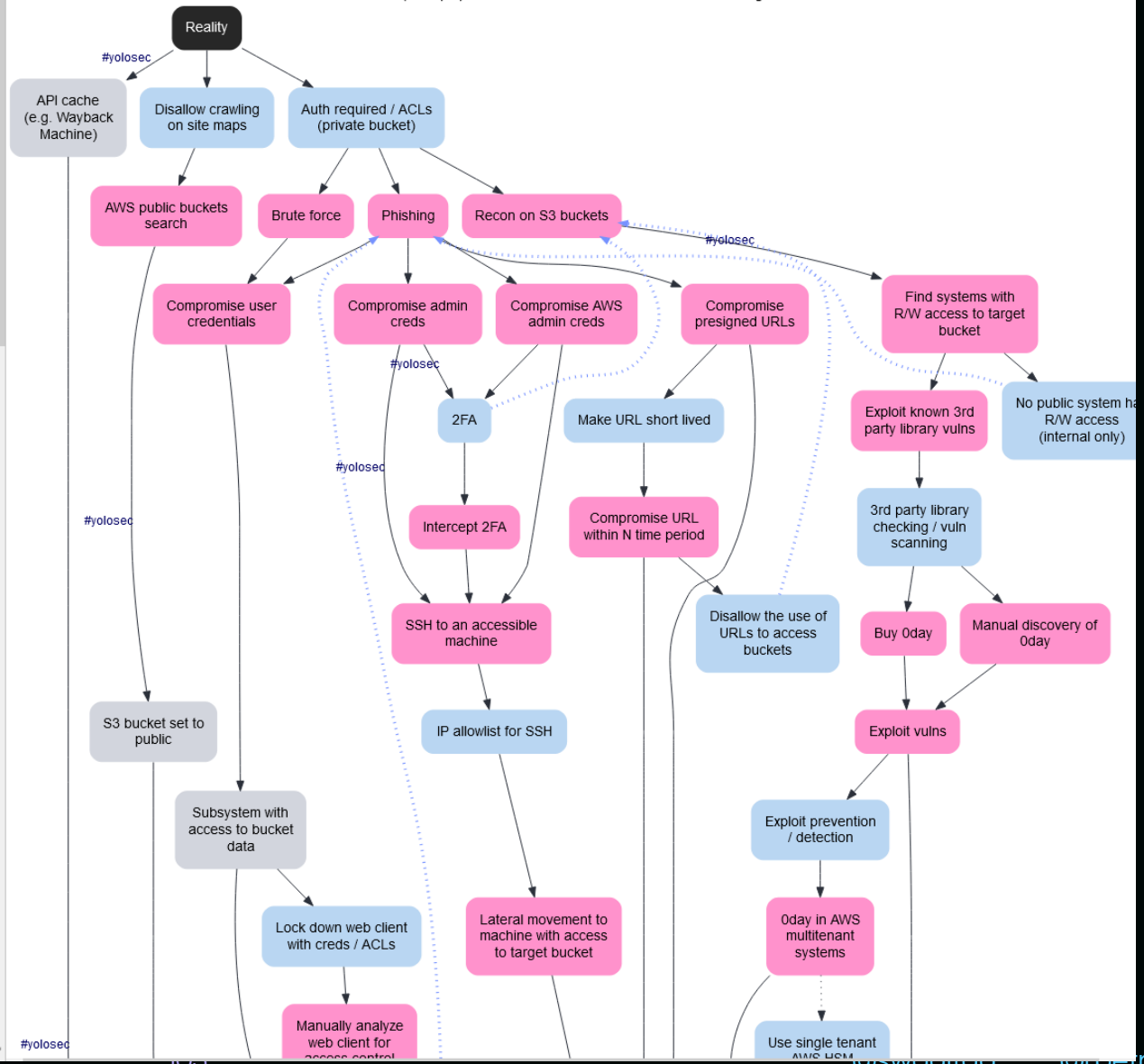
```
title: (Example) Attack Tree for S3 Bucket with Video Recordings

facts:
- wayback: API cache (e.g. Wayback Machine)
  from:
  - reality: '#yolosec'
- public_bucket: S3 bucket set to public
  from:
  - bucket_search: '#yolosec'
- subsystem_with_access: Subsystem with access to bucket data
  from:
  - compromise_user_creds

attacks:
- bucket_search: AWS public buckets search
  from:
  - disallow_crawling
- brute_force:
  from:
  - private_bucket
- phishing:
  from:
  - private_bucket
  - internal_only_bucket:
    backwards: true
  - access_control_server_side:
    backwards: true
- compromise_user_creds: Compromise user credentials
  from:
  - brute_force
  - phishing
- analyze_web_client: Manually analyze web client for access control misconfig
  from:
  - lock_down_acls
- compromise_admin_creds: Compromise admin creds
  from:
  - phishing
- compromise_aws_creds: Compromise AWS admin creds
  from:
  - phishing
- intercept_2fa: Intercept 2FA
  from:
  - 2fa
- ssh_to_public_machine: SSH to an accessible machine
  from:
  - compromise_admin_creds: '#yolosec'
  - compromise_aws_creds:
  - intercept_2fa
- lateral_movement_to_machine_with_access: Lateral movement to machine with access t
  from:
  - ip_allowlist_for_ssh
- compromise_presigned: Compromise presigned URLs
  from:
  - phishing
- compromise_quickly: Compromise URL within N time period
  from:
```

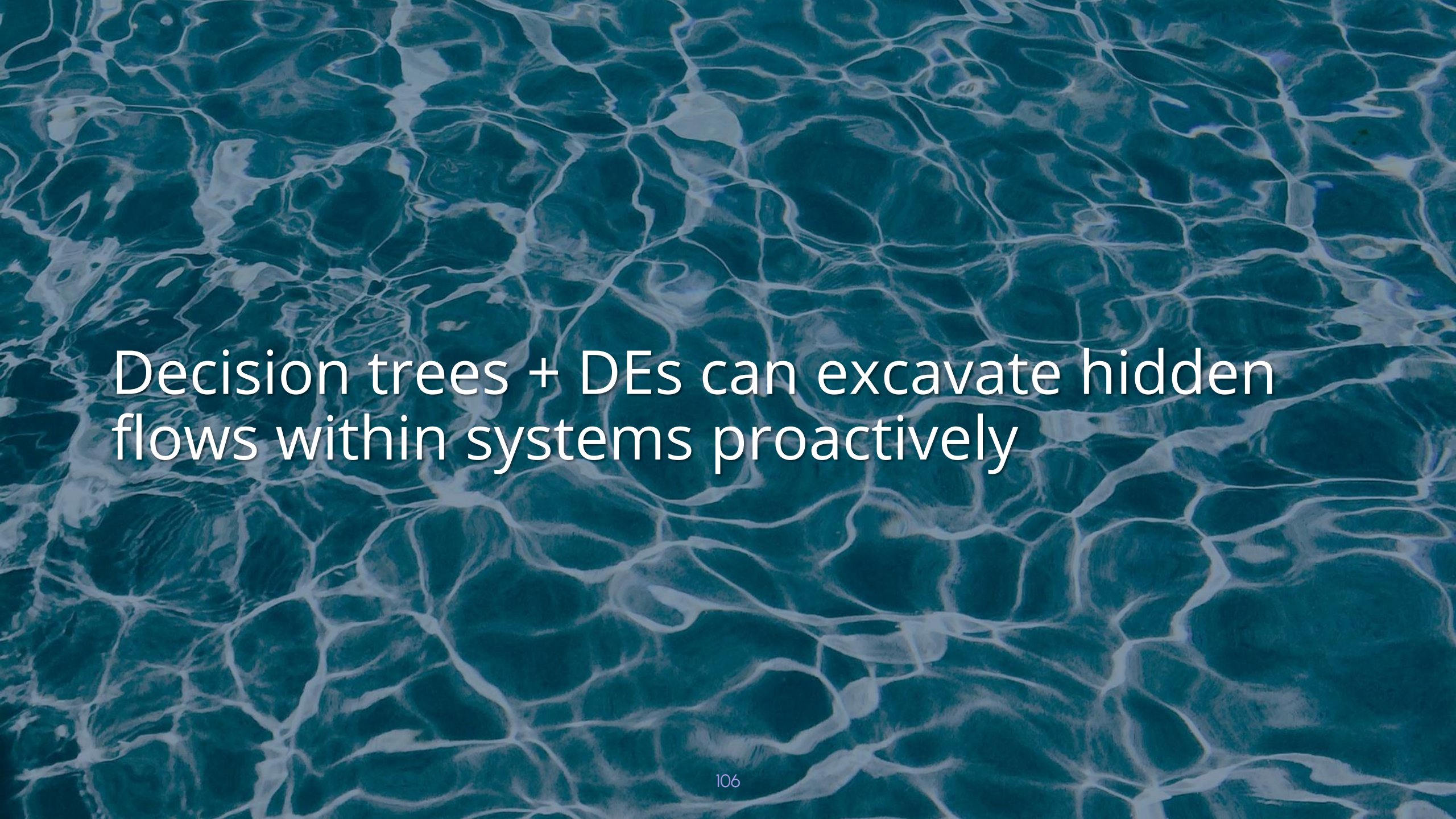
DECIDUOUS

Inspired by and with example taken from Kelly Shortridge's Creating Security Decision Trees With Graphviz

Import GitHub Gist Download .svg Download .dot

(Example) Attack Tree for S3 Bucket with Video Recordings




Decision trees + DEs can excavate hidden flows within systems proactively

Attacker tracing also fuels experimentation: each branch is a chain of hypotheses

Experimentation platform

Experimentation can test the efficacy of monitoring or resilience measures

Deception Environments become a tool in the Security Chaos Engineering arsenal

Fidelity degradation experiments divulge how attackers react to different envs

Swap standard components for substitutes to disrupt attack plans in prod (sow F.U.D.)

Tune the difficulty of accessing the DE to study different types of attackers

Augment honeyhives with honeytokens for flavor (like Thinkst's AWS key canarytoken)

VI. Future opportunities


Just-in-time terraforming

JIT creation of isolated deception VMs via copy-on-write or page deduplication


Systems terraforming: reify an entire constellation of hosts upon connection

Potential network & hypervisor tricks: unfreeze assets & fast-forward execution...

Virtualization is one big lie to software—
why not extend this lie a little bit further?

Instance emulation

Full emulation of CSP APIs would facilitate DEs but also other operational benefits...

Honeypatching at scale: redirect attackers towards a DE + deploy via update pipelines

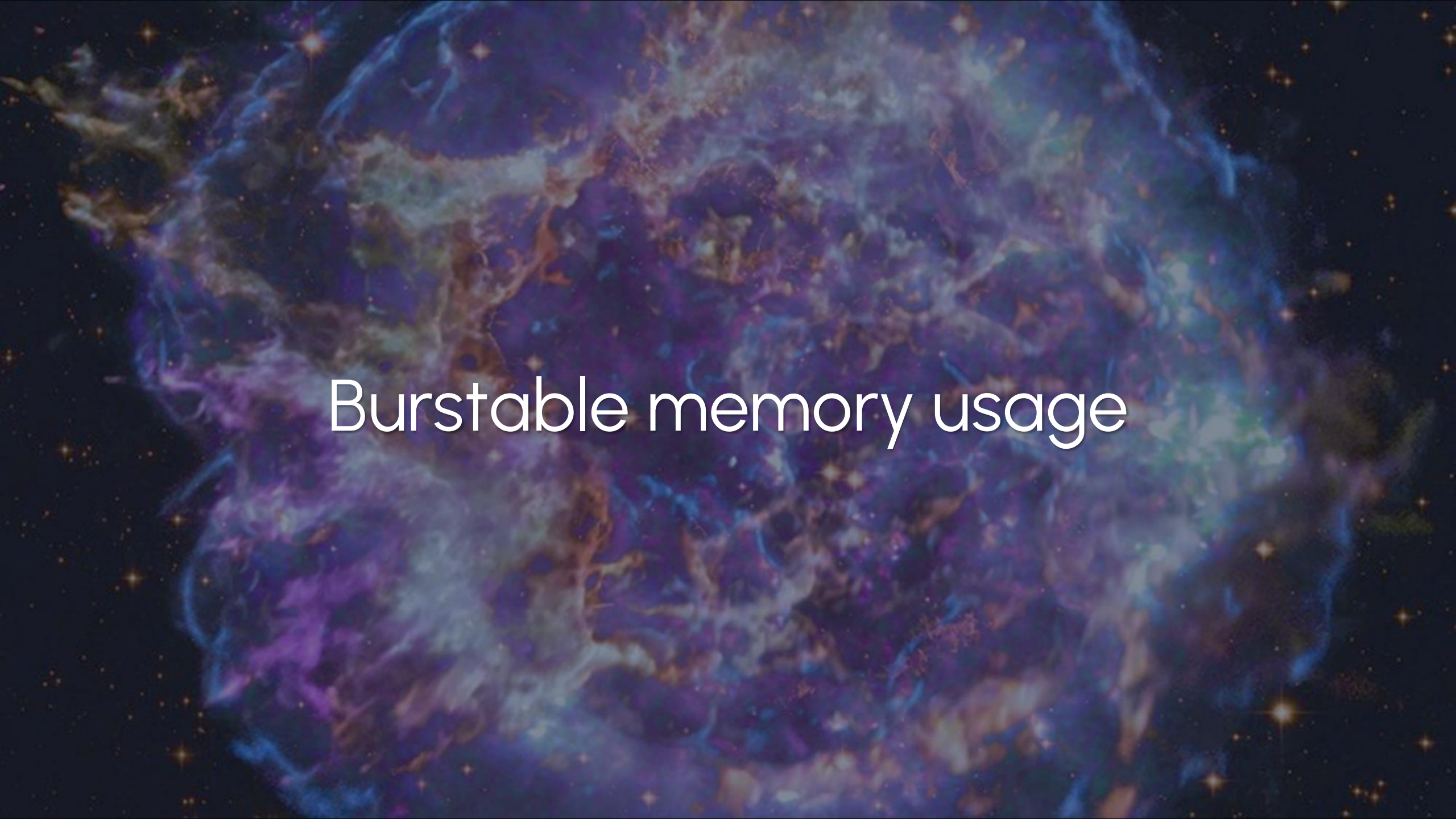
Anonymization via mirroring

Extend traffic-mirroring tech to include data anonymization features (layer 7 ftw)

Hypervisor-based observability

Tracing & observability tools often execute with root privileges & are simple to subvert

OSes could expose core events (process and file ops) over a common protocol...


Burstable memory usage

CSPs could support burstable performance instances via ballooning or swapped mem

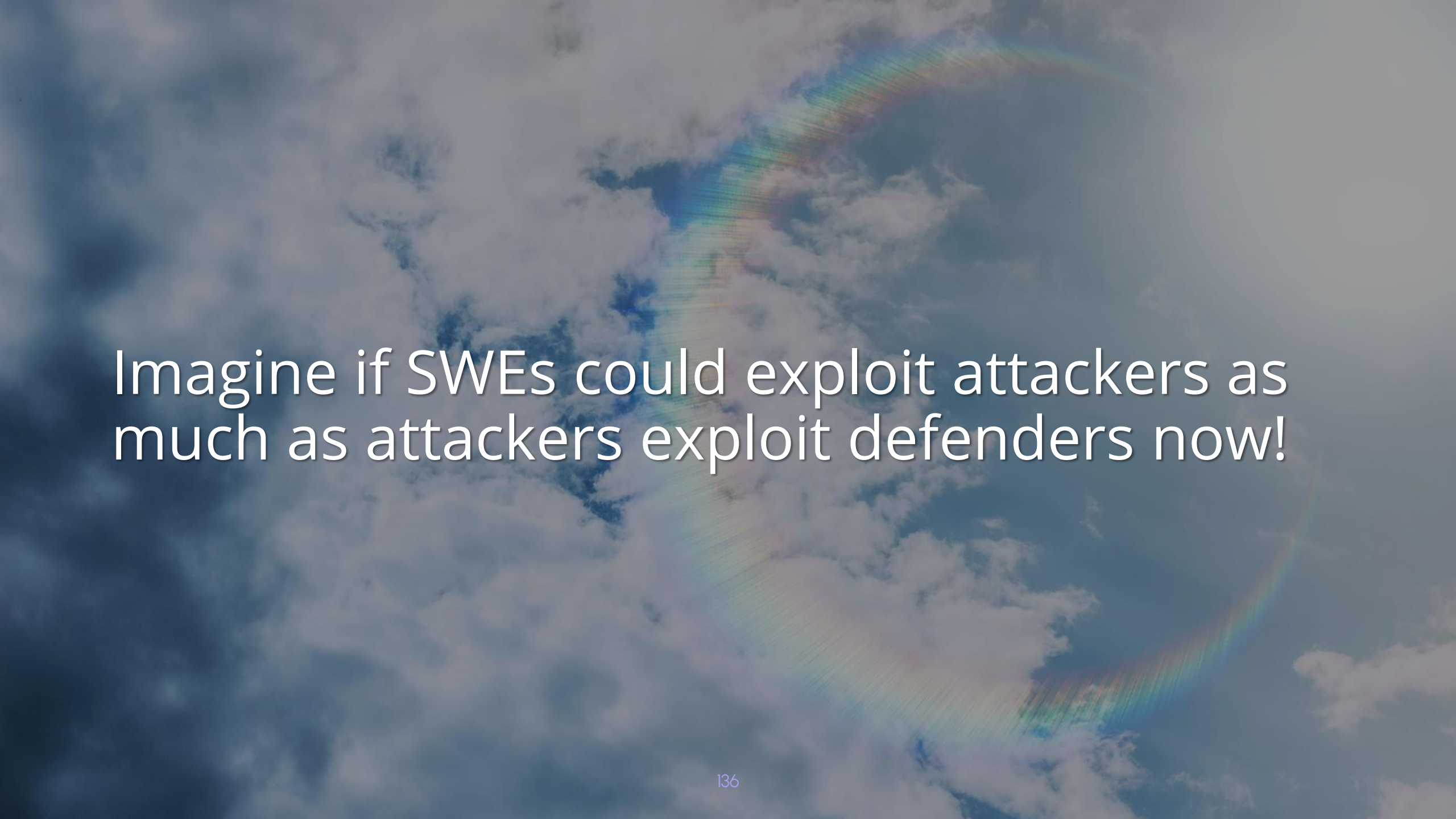
Temporarily migrate VMs across physical instances when their activity bursts...


Per-account billing limits

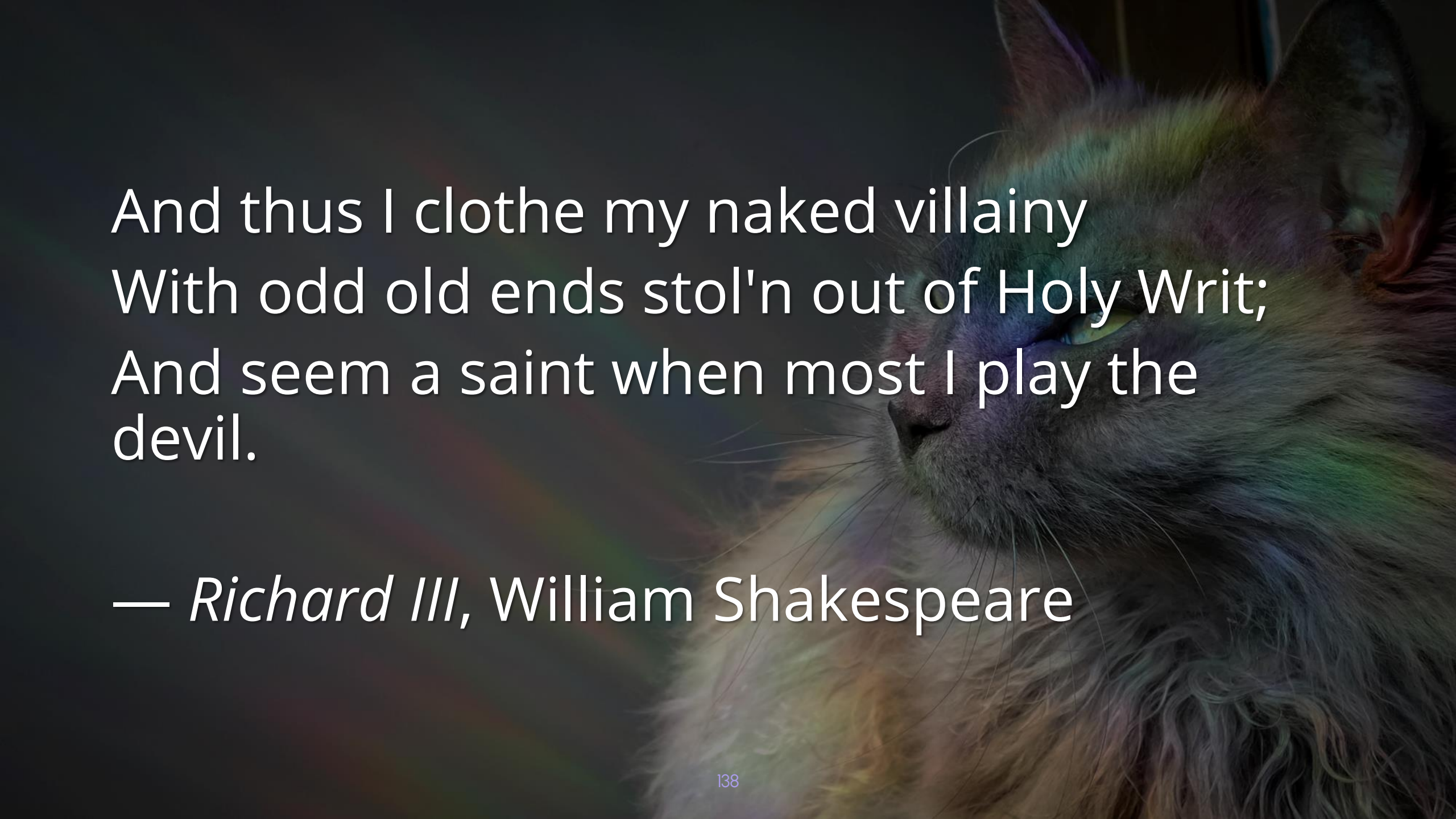
Per-account billing limits can restrict the amount of your $$$ attackers can spend

CSPs have effective tools for isolation every resource except for customers’ wallets

VI. Conclusion


Imagine if SWEs could exploit attackers as much as attackers exploit defenders now!


Deception envs allow you to bamboozle attackers for fun and profit (and resilience)



And thus I clothe my naked villainy
With odd old ends stol'n out of Holy Writ;
And seem a saint when most I play the devil.

— *Richard III*, William Shakespeare

@swagitda_

@rpetrich

/in/kellyshortridge

/in/rpetrich

chat@shortridge.io

rpetrich@gmail.com