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

ARC307

Building confidence through chaos engineering on AWS

Laurent Domb (he/him)

Chief Technologist, Federal Financial Services
AWS

Agenda

1. Chaos engineering (CE) introduction
2. Continuous resilience (CR)
3. Building a CR/CE program
4. Chaos engineering resources

Introduction to chaos engineering

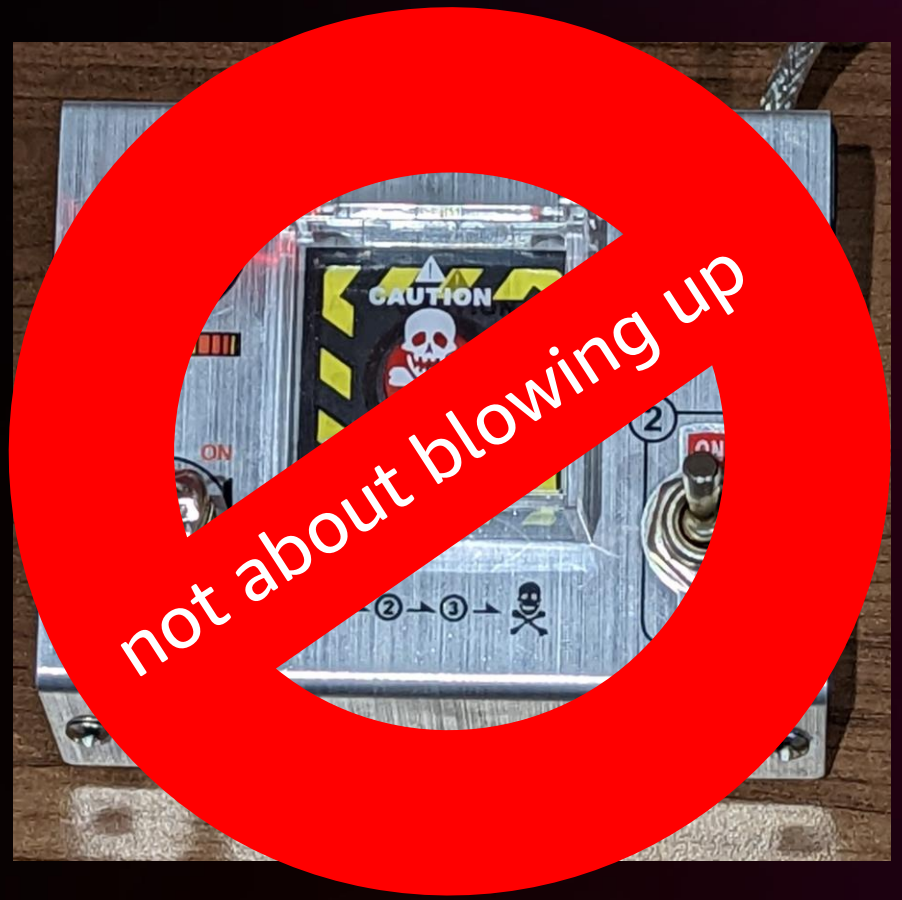
“Chaos engineering is about building a culture of resilience in the presence of unexpected system outcomes.”

Principles of chaos engineering

principlesofchaos.org

Chaos engineering a different perspective

Chaos engineering is

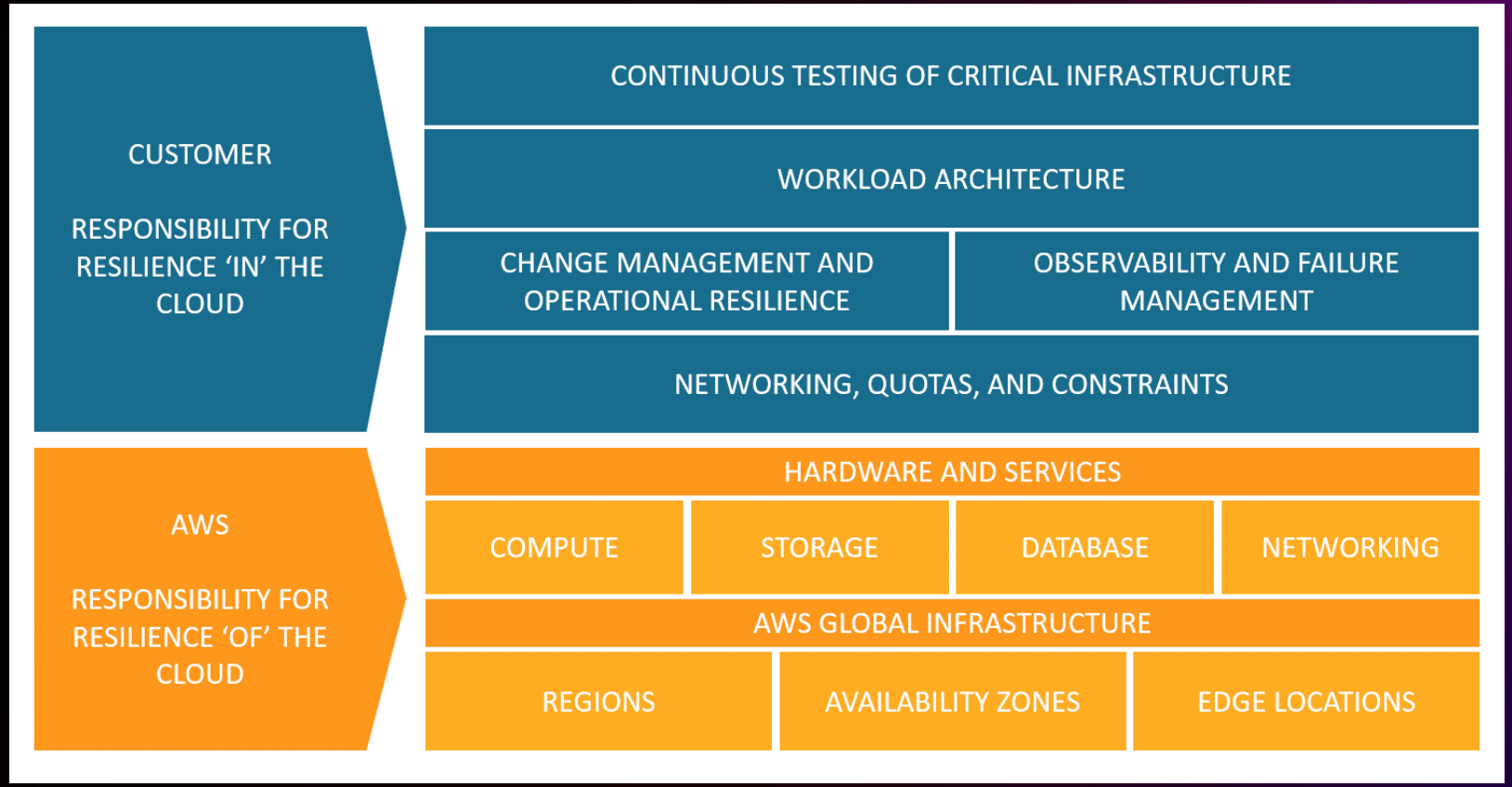


production

Chaos engineering a different perspective

What you have control over

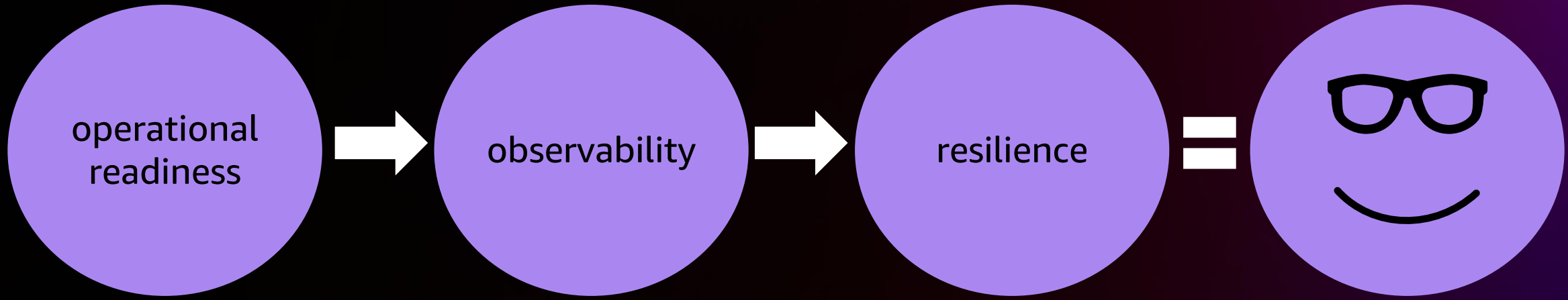
What you have no control over



SHARED RESPONSIBILITY MODEL FOR RESILIENCE

Chaos engineering a different perspective

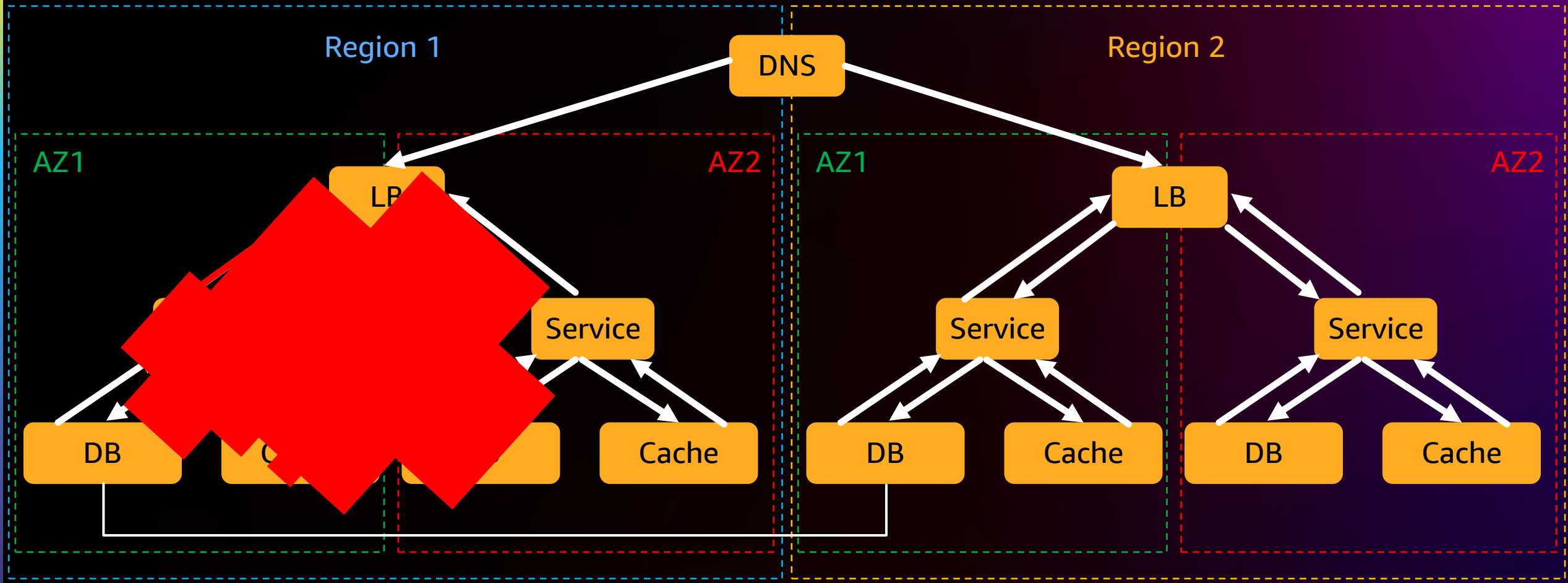
Chaos engineering leads to improved



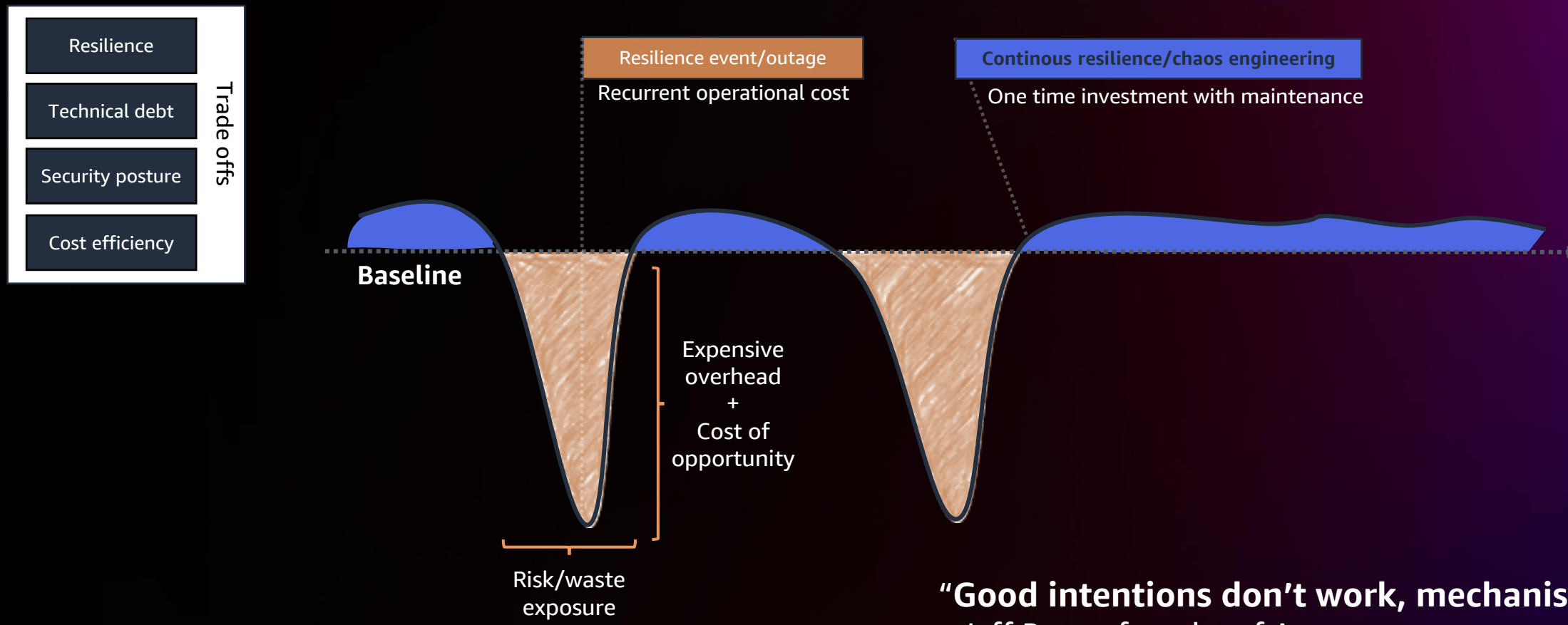
Fail-safe chaos experiments

Perform controlled experiments in which the assumption is that the load or faults that you inject will be tolerated by the system and are fail-safe.

How much confidence do you have in your system?



Why chaos engineering?



“Good intentions don’t work, mechanisms do.”
– Jeff Bezos, founder of Amazon

Chaos engineering stories

Financial Services	Health Care	Insurance
Media & Entertainment	Telco	Retail
Transport/Airlines	Hospitality	Food/Delivery

https://github.com/ldomb/ChaosEngineeringPublicStories

Chaos engineering adoption 2023

40% Will adopt chaos engineering as part of their DevOps initiatives in 2023

Reducing unplanned downtime by 20%

Source: The I&O Leader's Guide to Chaos Engineering by Gartner

Chaos engineering prerequisites

Prerequisite 1:

Basic monitoring/observability

Prerequisite 2:

Organizational awareness

Prerequisite 3:

Real world events/faults

Prerequisite 4:

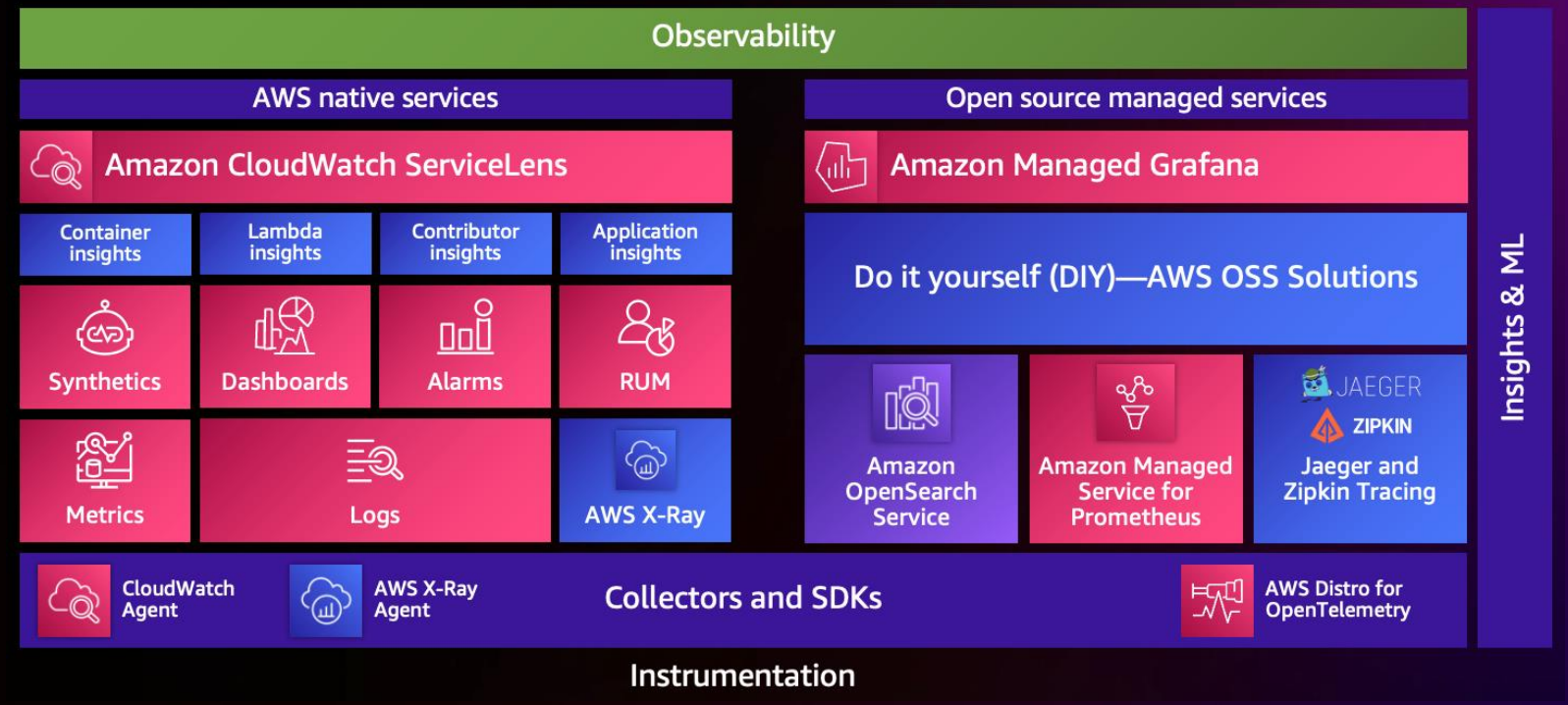
Remediate findings

Prerequisite 1: Basic monitoring/observability

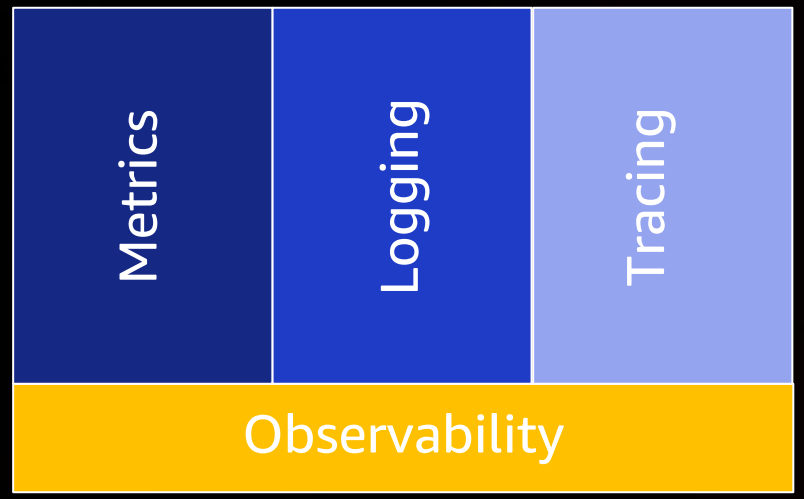
Prerequisite 1: Basic monitoring/observability

Observability

Find the needle in the haystack



The 3 pillars of observability



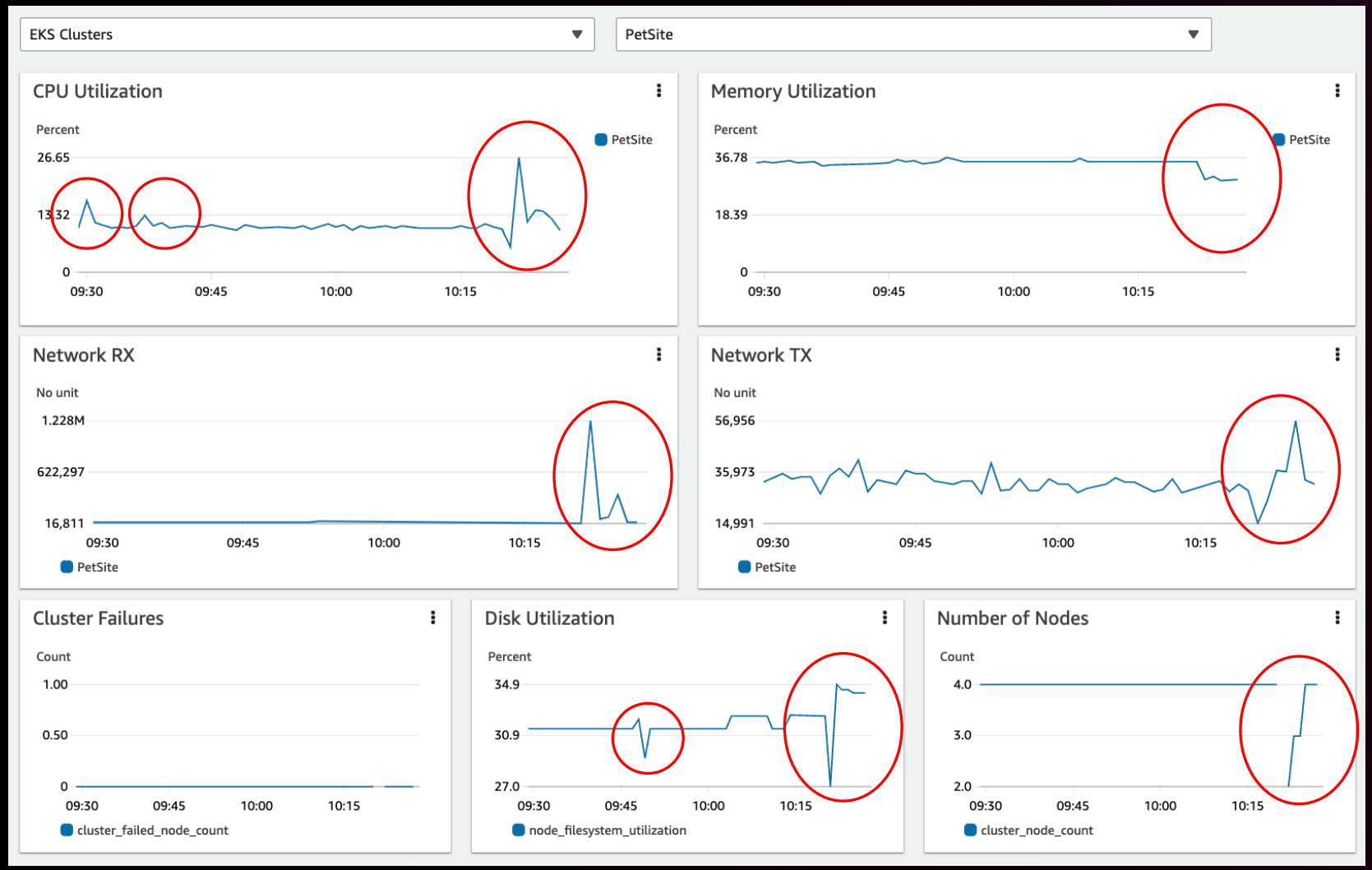
Do you understand the inner workings of your application?

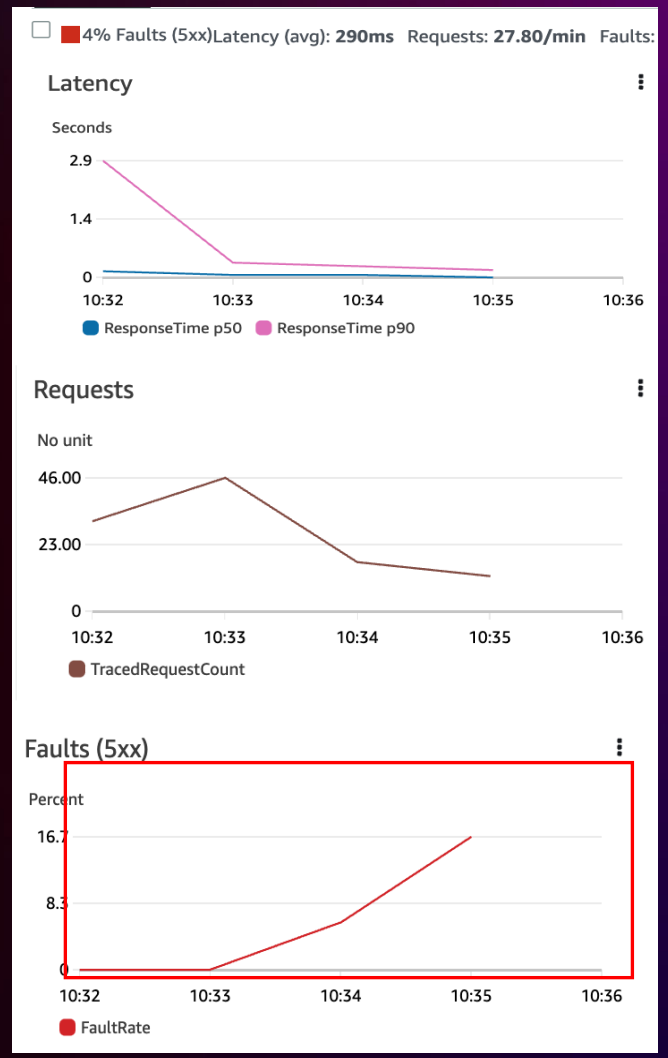
Do you understand any system state your application may have fallen into?

Can you understand the above, just by observing your tools?

Can you understand the state, no matter how unusual the situation is?

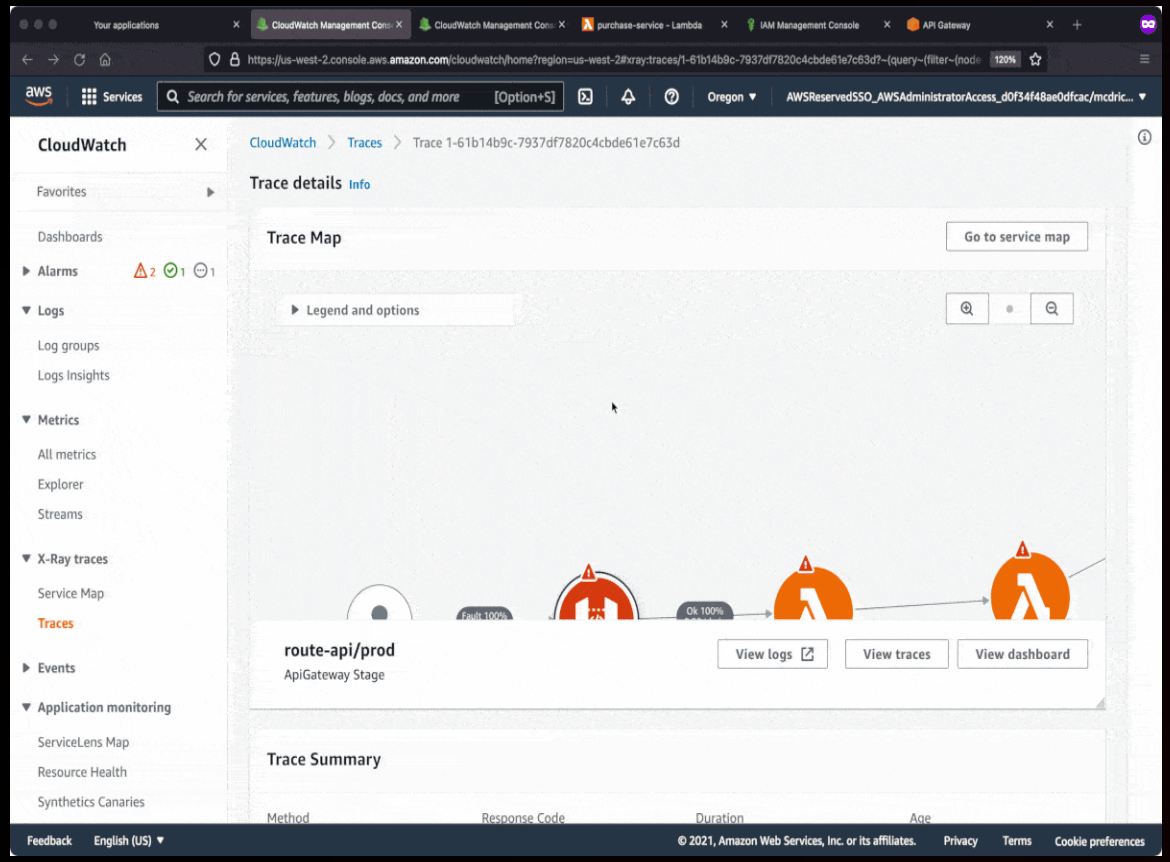
Prerequisite 1: Basic monitoring/observability



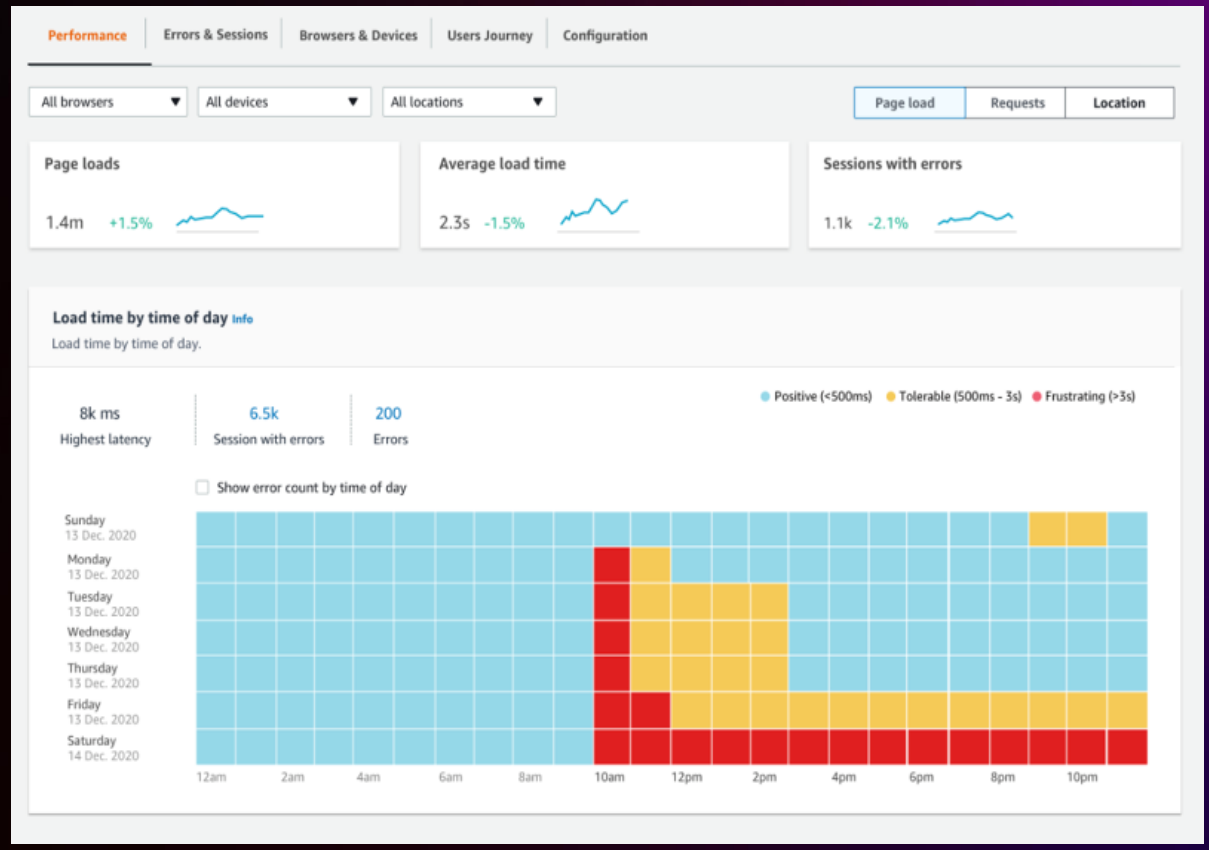


Top level observability

Prerequisite 1: Basic monitoring/observability



Application monitoring

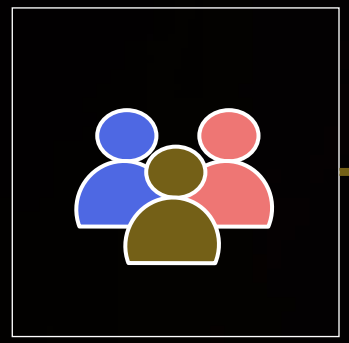


User monitoring

https://aws.amazon.com/blogs/mt/an-observability-journey-with-amazon-cloudwatch-rum-evidently-and-servicelens/

Prerequisite 2: Organizational awareness

Central team that drives chaos engineering



Empowers

Education

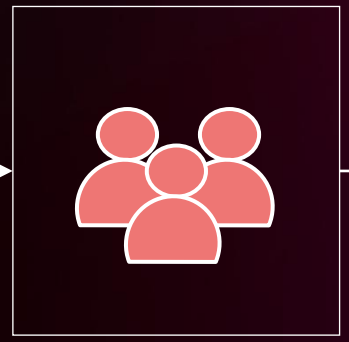
Experiment Catalog

Learnings

Executive sponsor

Engineering team 1

Decentralized execution



- Experiment type
- Environment
- Self-service
- Guardrails
- Game Day
- Automated
- Resilience

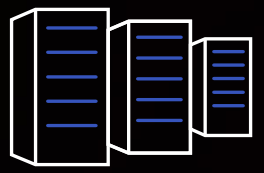
- Experiment type
- Environment
- Self-service
- Guardrails
- Game Day
- Automated
- Resilience

Engineering team 2

Prerequisite 3: Real-world experiments

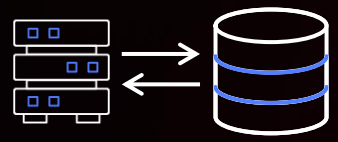
Code & configuration

e.g., bad deployment, cred expiration, host shutdown



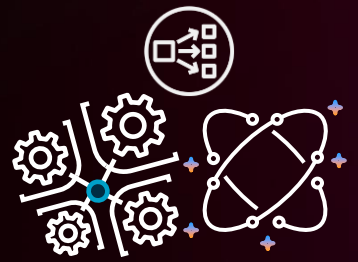
Infrastructure

e.g., datacenter failure, hardware failure



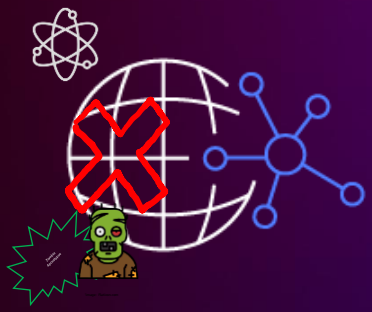
Data and state

e.g., data corruption, overload



Dependencies

e.g., third-party integrations, AWS services



Highly unlikely, but technically feasible

e.g., physical loss of an entire Region, the internet is down

Prerequisite 4: Remediate the findings

- Findings through chaos engineering experiments should be prioritized based on the level of impact they may cause
- Findings that involve the resilience or security of your workload should have priority over new features, as if not addressed timely, can impact your customers
- Find an executive sponsor that can help you address the priority if needed

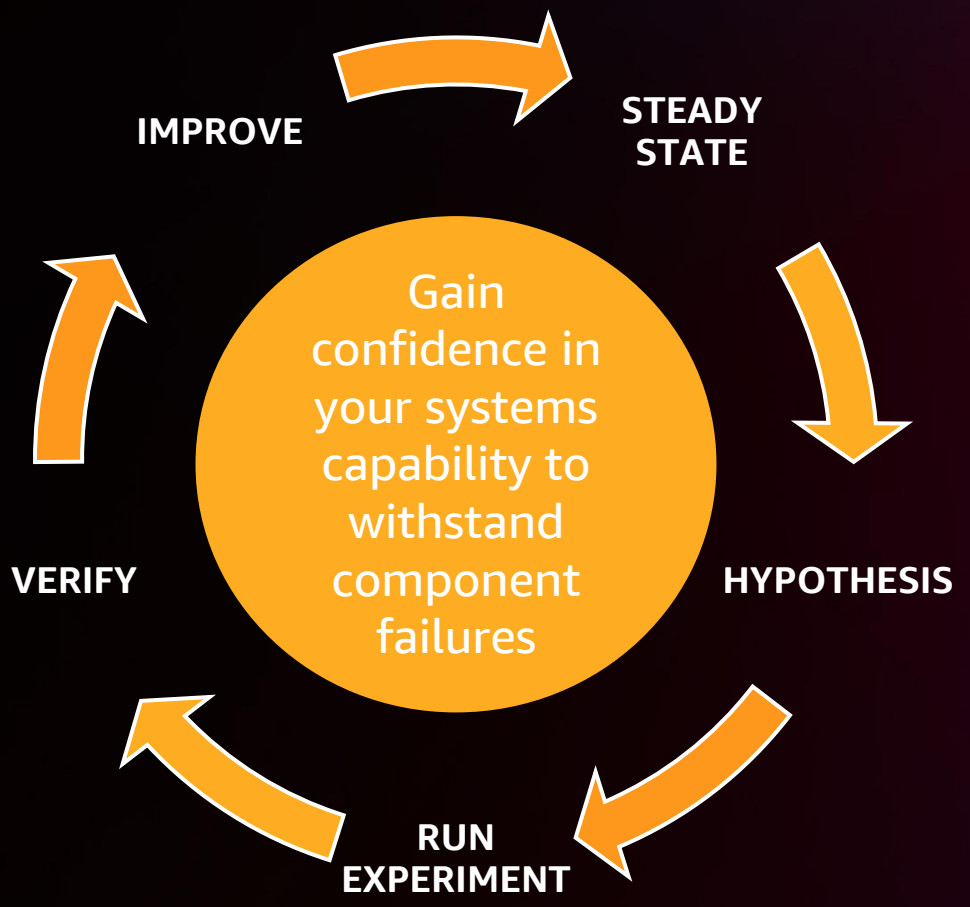
Continuous resilience

Continuous resilience

Code & configuration
e.g. bad deployment, cred expiration

Infrastructure
e.g. datacenter failure, hardware failure

Data and state
e.g. data corruption



Continuous resilience flywheel by Adrian Hornsby

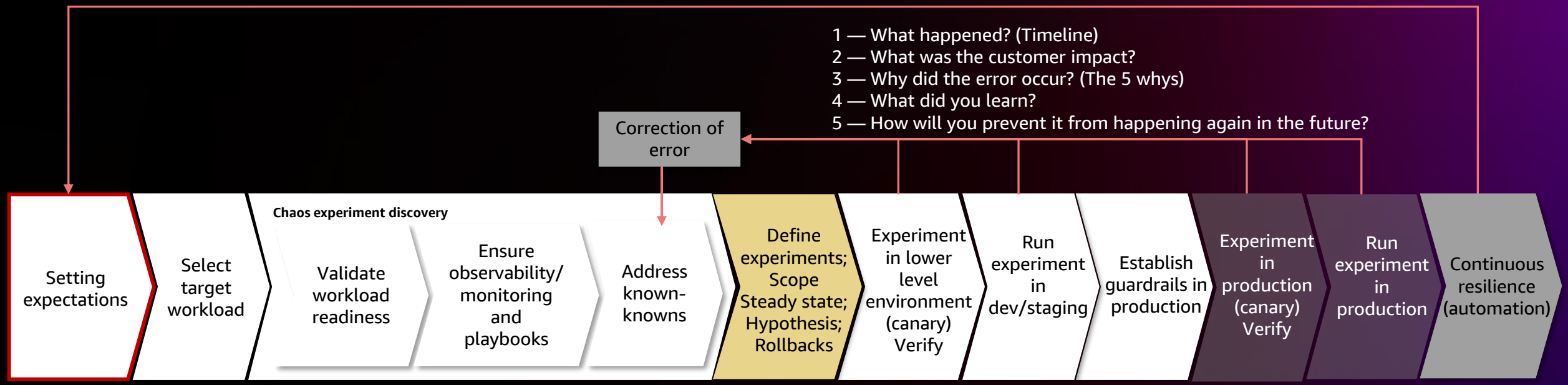
Dependencies
e.g. third-party integrations, AWS services

Highly unlikely, but technically feasible
e.g. physical loss of an entire Region, the internet is down

Building a chaos engineering / continuous resilience program

Setting expectations

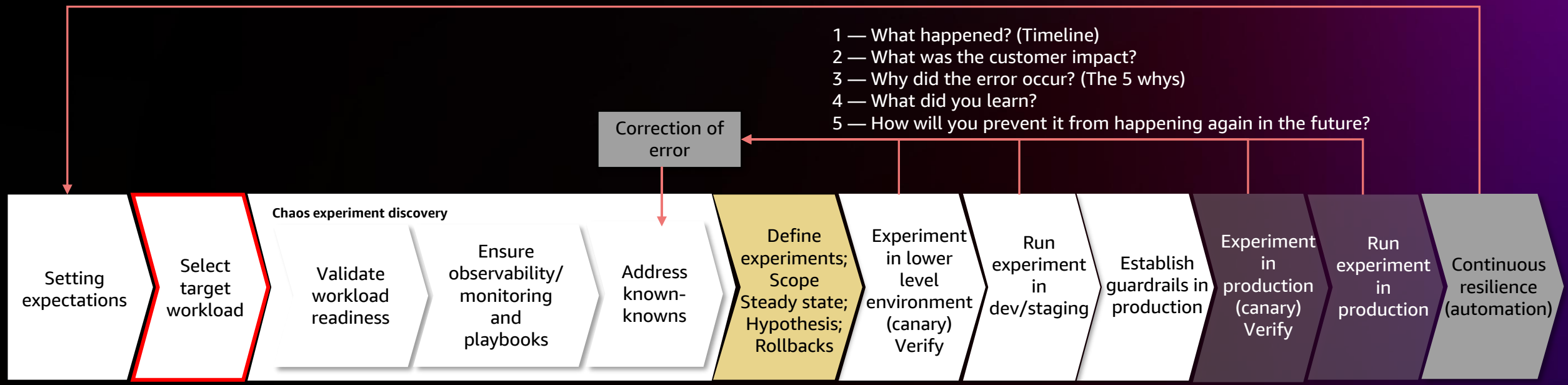
PRIORITIZE FINDINGS, FIX THEM, RE-ITERATE. CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS.



Setting expectations

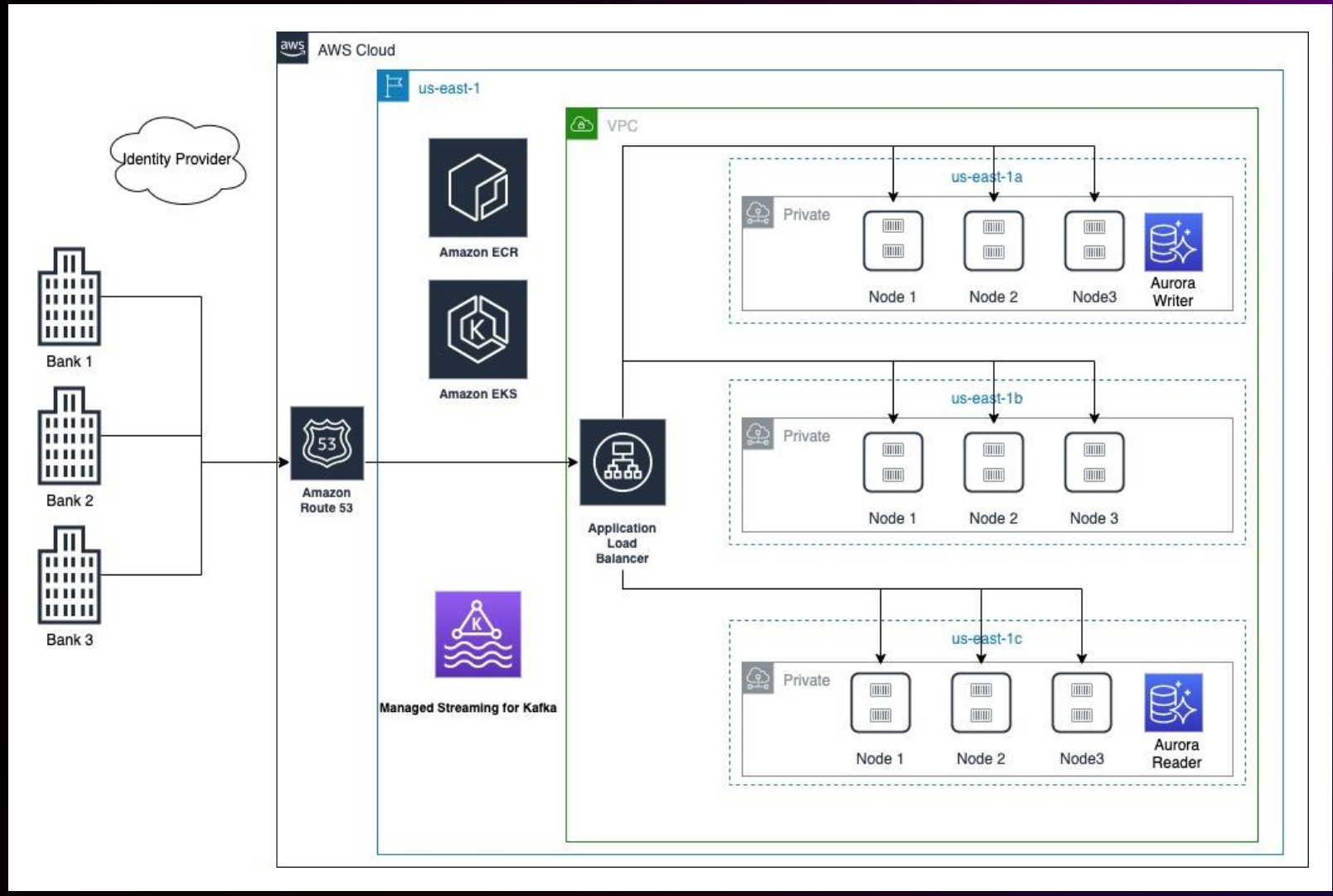
Selecting the target workload

PRIORITIZE FINDINGS, FIX THEM, RE-ITERATE. CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS.



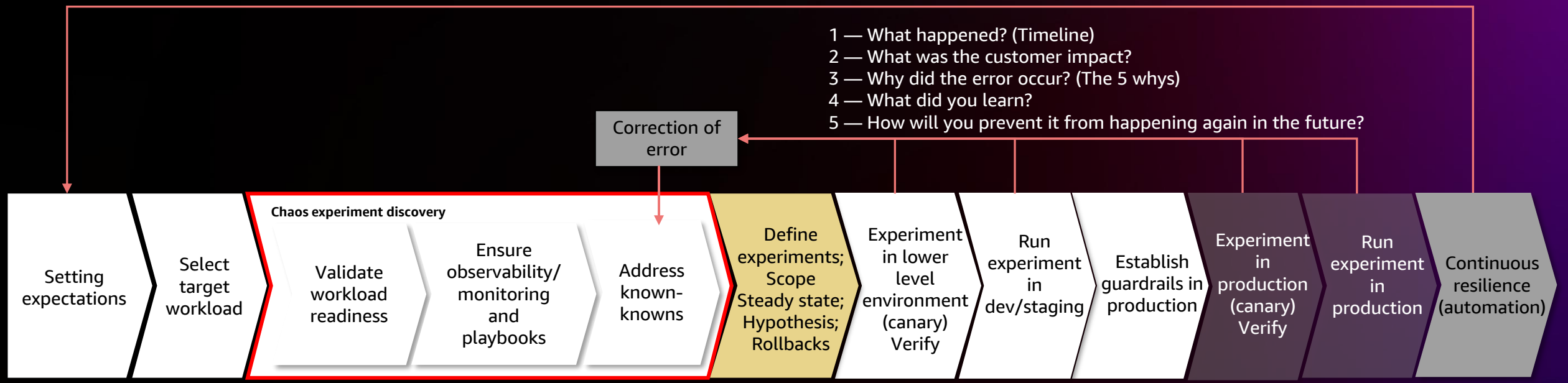
Target workload selections

For your first experiment, choose a workload that if degraded has only minimal to no impact to your internal or external customers.



Chaos experiment discovery

PRIORITIZE FINDINGS, FIX THEM, RE-ITERATE. CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS.



Chaos experiment discovery

Operational readiness review

Failover

Observability

Well-Architected Review

Runbooks

Retries

Incident response

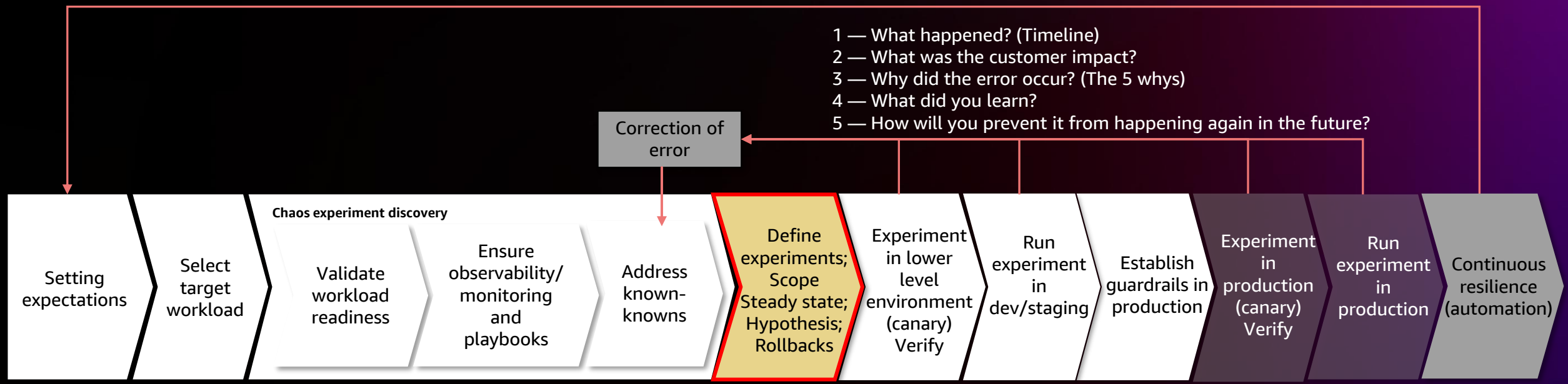
Health checks

Circuit breakers

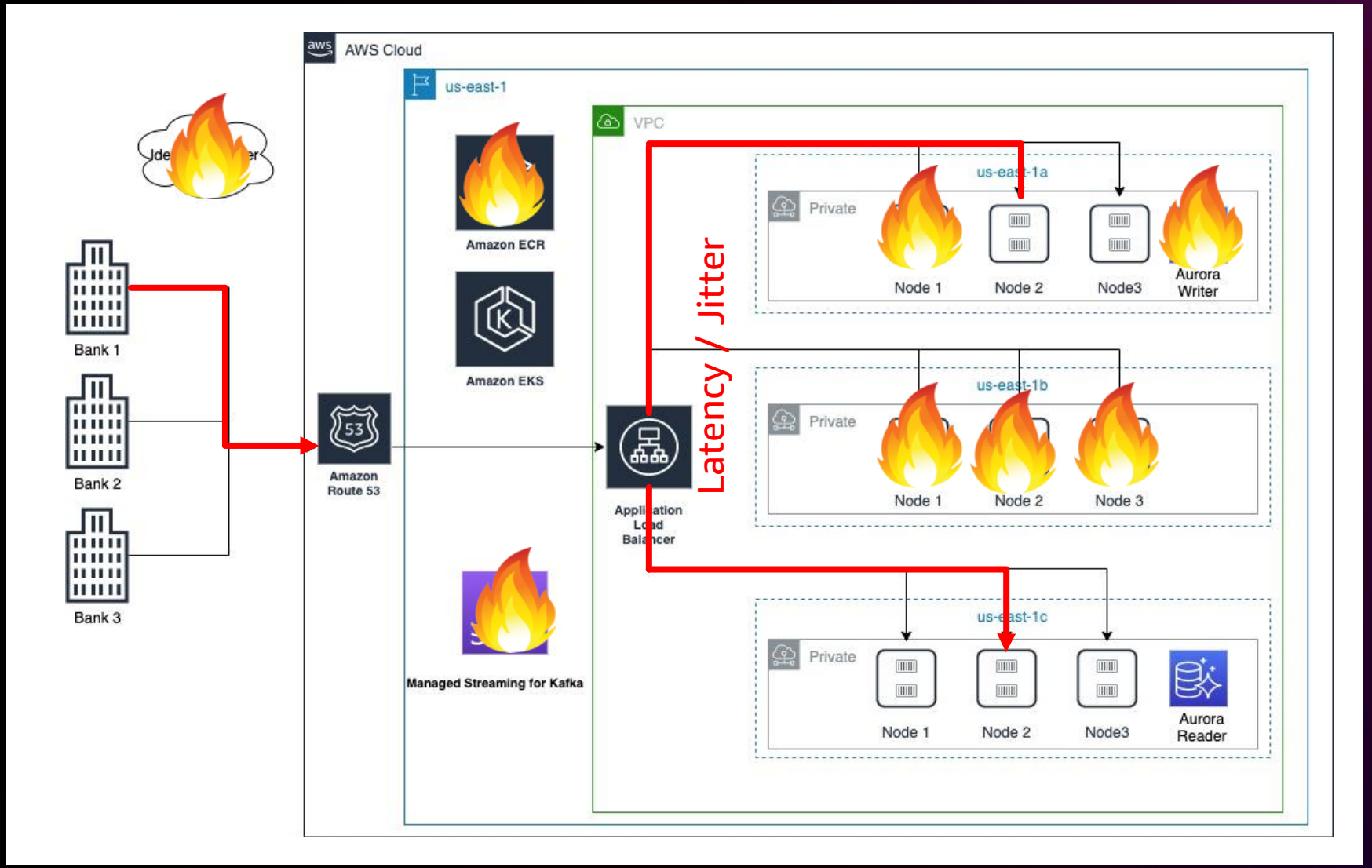
Fix known issues before moving forward with the experiment!

Define experiment

PRIORITIZE FINDINGS, FIX THEM, RE-ITERATE. CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS.

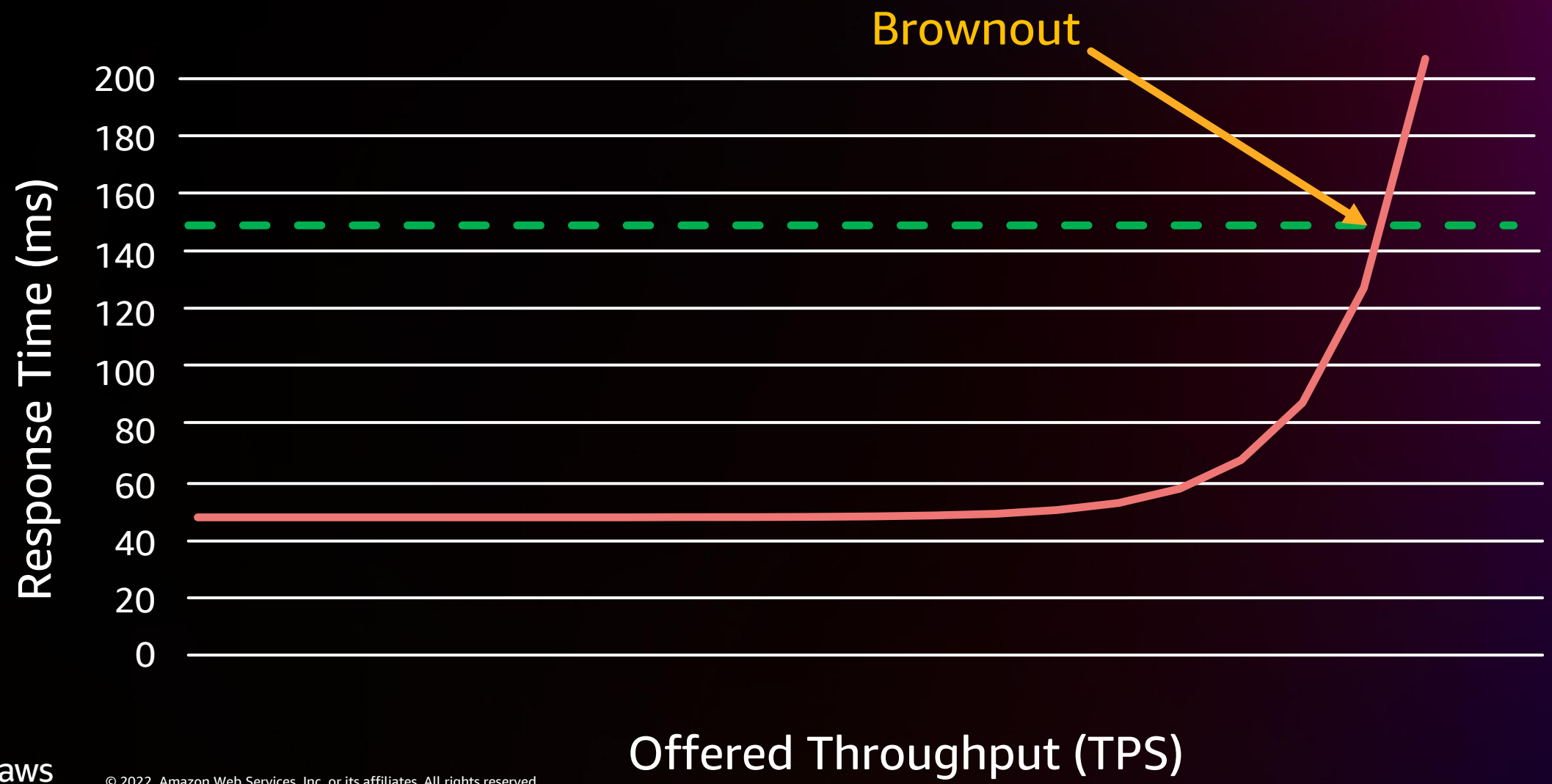


Chaos experiment definition – Failure modes



Chaos experiment definition - Experiment #1

BROWNOUT



Chaos experiment definition - Steady state

Payments – Transactions per second

Retail – Order per second

Streaming – Stream starts per second

Media/Audio – Playback event started

Chaos experiment definition - Hypothesis

At a rate of 300 TPS, if 40% of the nodes in the EKS node-group are terminated, the Transaction Create API continues to serve the 99th percentile of requests in under 100 ms (steady state)

The EKS nodes will recover within 5 minutes, and pods will get scheduled and process traffic within 8 minutes after the initiation of the experiment

Alerts will fire within 3 minutes

Chaos experiment definition – Bring all together

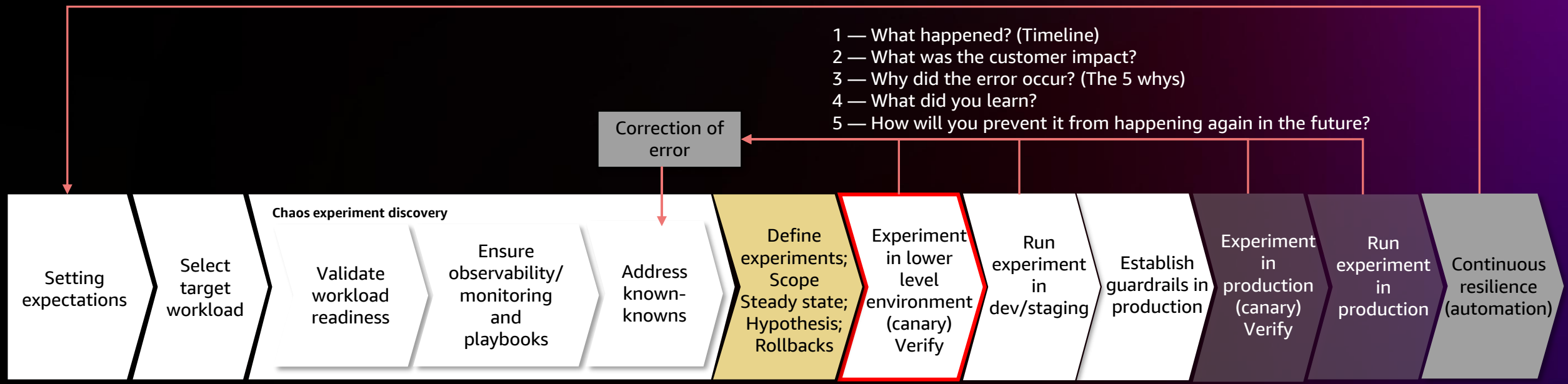
TEMPLATE

Chaos Experiment

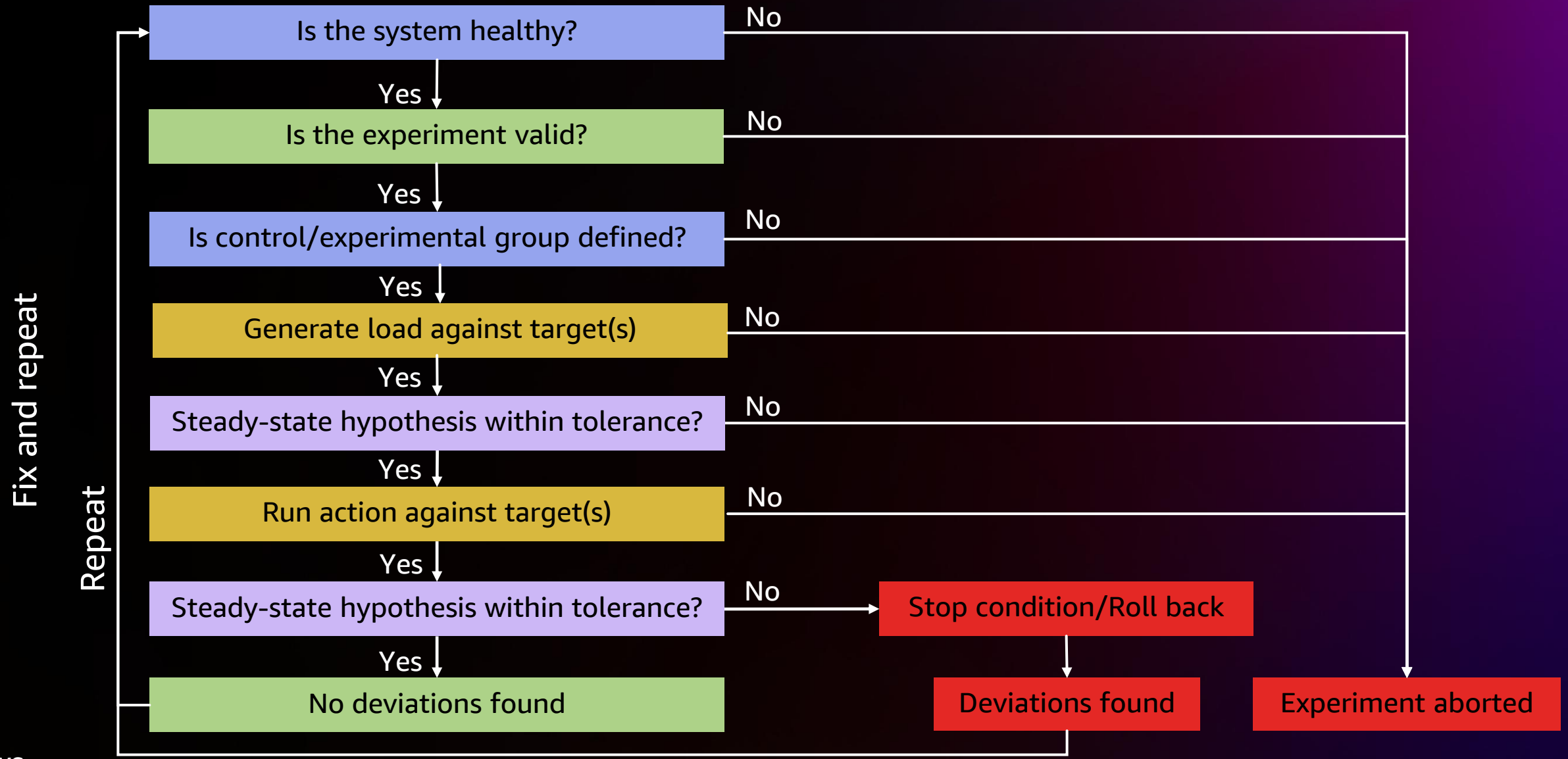
Realtime Payments	Amazon EKS Payments Cluster	OpenSearch/CWL/X-ray
Workload Name	Targets	Observability / Logging
Resilience	Some clients calls will time out	At a rate of 300 TPS, If 40% …
Contribution	Fault isolation boundary	Hypothesis
Brownout - Terminate 40% nodes	CW Alarm when node count < 60%	
Action	Stop Condition	Findings
Staging	CFN template to built nodes	
Environment	Rollback	COE – Actions to mitigate fault
30 minutes	Chaos-ready	
Duration	Resource Tag/ID/Filter	
300 TPS		
Load		

Prime the environment for the experiment

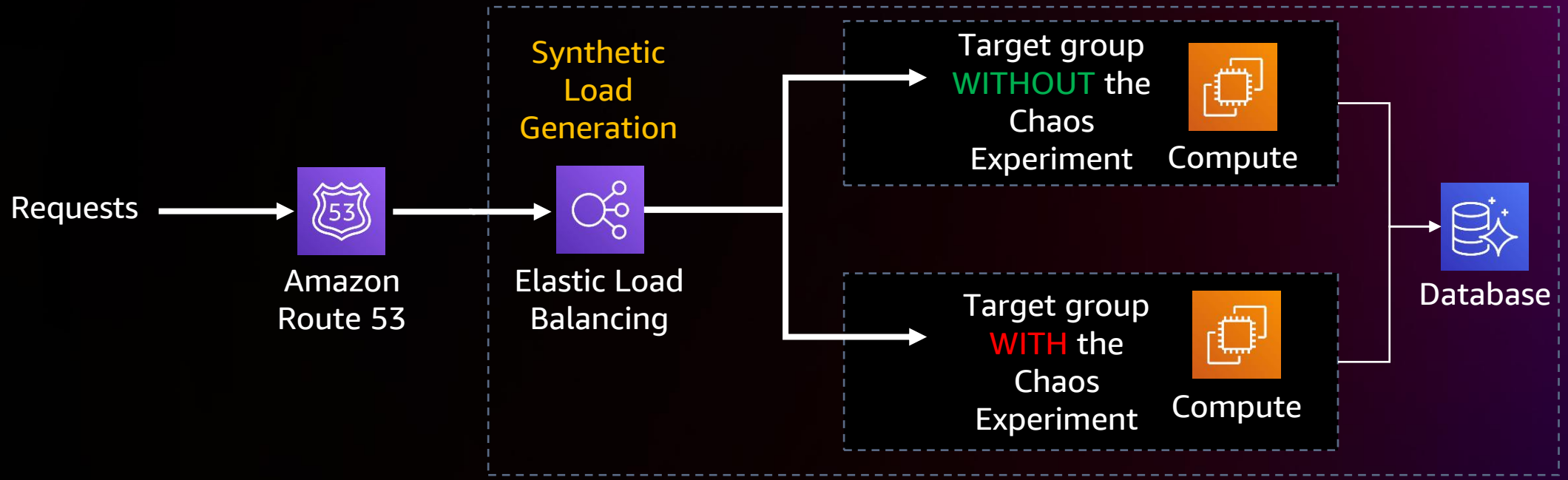
PRIORITIZE FINDINGS, FIX THEM, RE-ITERATE; CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS



Chaos experiment execution flow



Controlled experiments through canary deployments in lower-level environment

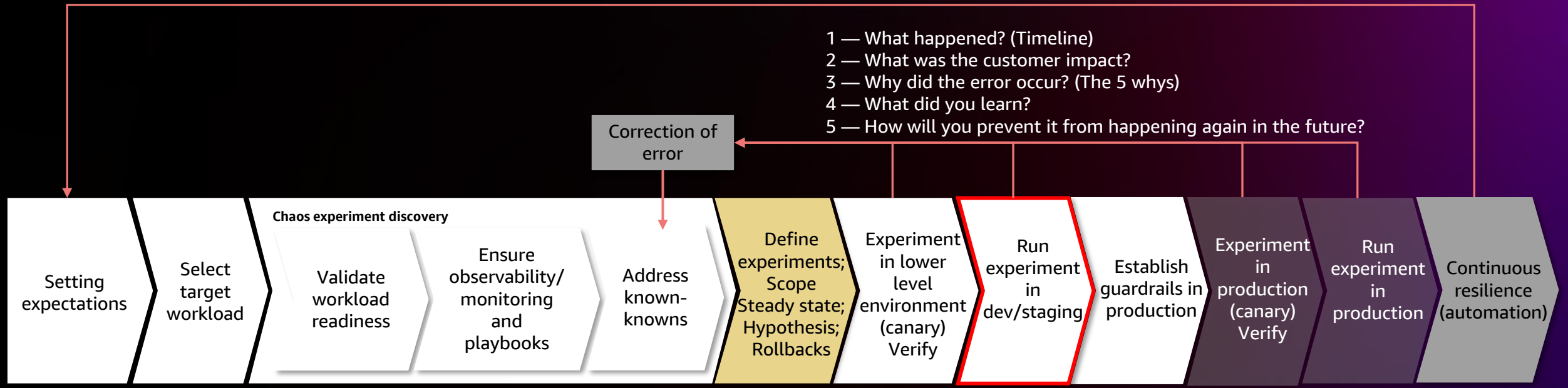


Verify that both groups are healthy before moving forward

Source: https://medium.com/@adhorn/chaos-engineering-q-a-how-to-safely-inject-failure-ced26e11b3db

Run the experiment in development/staging

PRIORITIZE FINDINGS, FIX THEM, RE-ITERATE; CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS



AWS Fault Injection Actions

- Stop, reboot, and terminate instance(s) (Amazon EC2)
- API throttling/internal error/unavailable error (Amazon EC2)
- Increased memory or CPU load (Amazon EC2)
- Kill process (Amazon EC2)
- Latency injection (Amazon EC2)
- Drain container instances (Amazon ECS)
- Terminate task (Amazon ECS)
- Increase memory or CPU consumption per task (Amazon ECS)
- Terminate node group instances (Amazon EKS)
- Litmus Chaos/Chaos Mesh integration (Amazon EKS)
- Network Connectivity Disruption (Amazon EC2)
- Database stop, reboot, and failover (Amazon RDS)

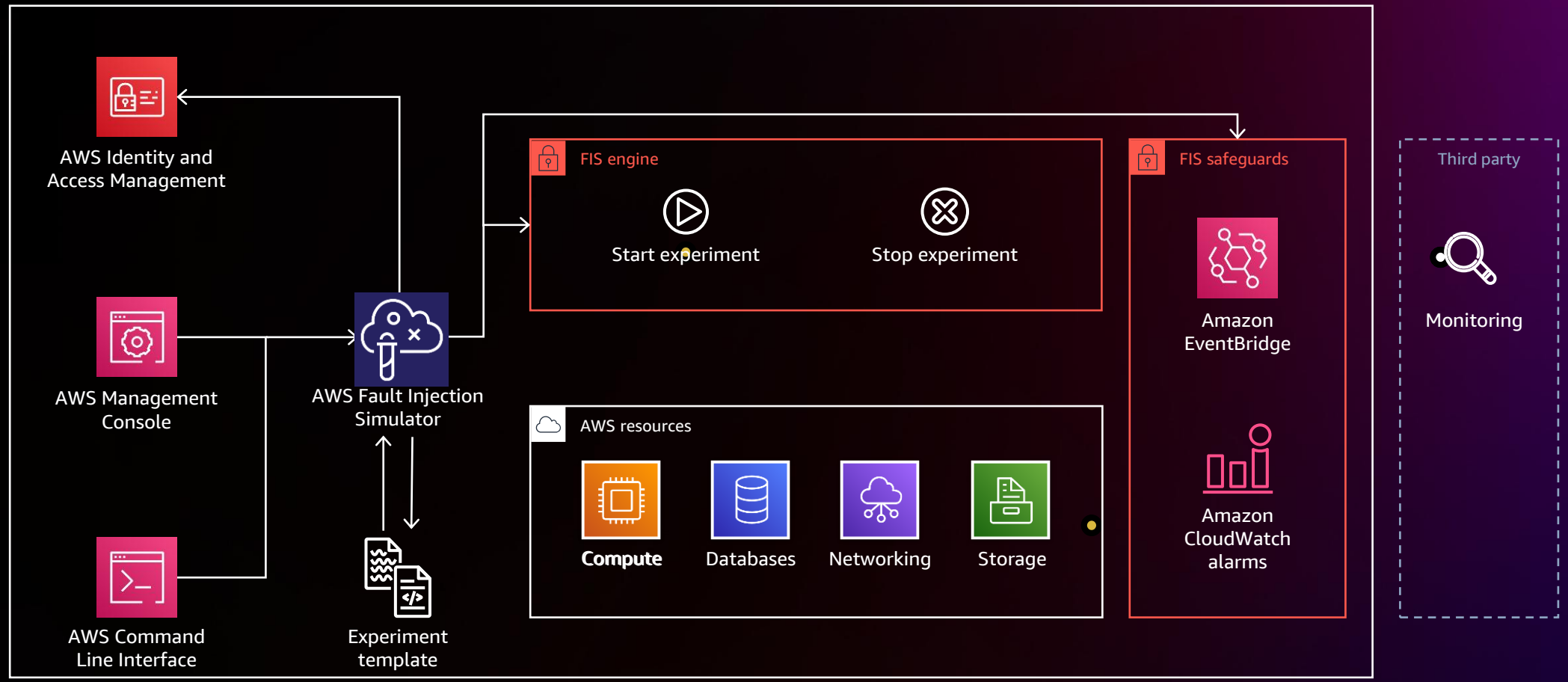
AWS Fault Injection Actions

- Systems manager send command (SSM Agent)
- Increased memory or CPU load (Amazon EC2)
- Kill process (Amazon EC2)
- Latency injection (Amazon EC2)
- Increase memory or CPU consumption per task (Amazon EC2)
- Network port blackhole (Amazon EC2)
- Network latency
- Network latency at target source
- Network packet loss
- Network packet loss at target source

Execute the controlled experiment

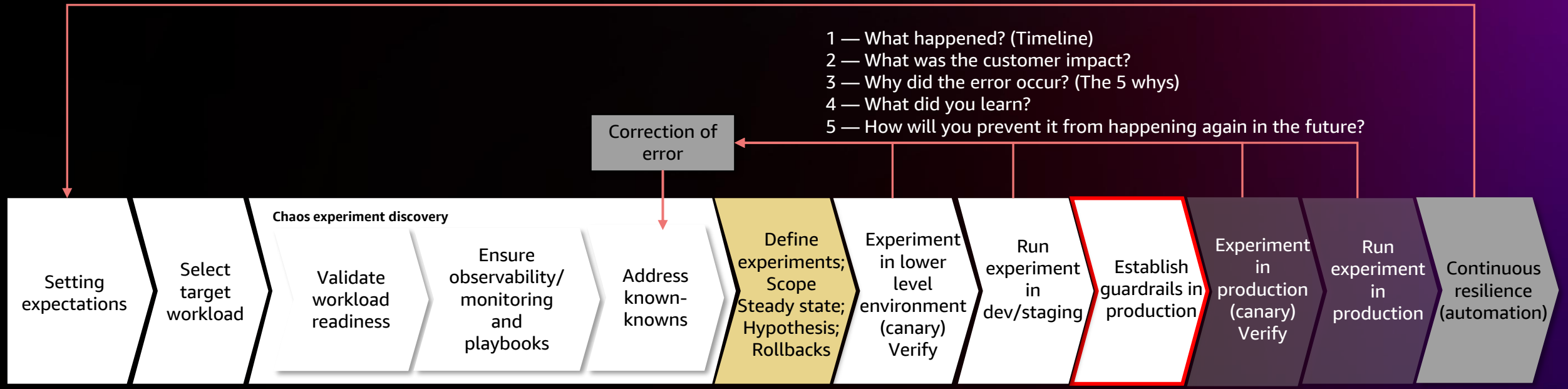
EXECUTE THE CONTROLLED EXPERIMENT

AWS Fault Injection Simulator



Establish guardrails in production

PRIORITIZE FINDINGS, FIX THEM, RE-ITERATE; CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS

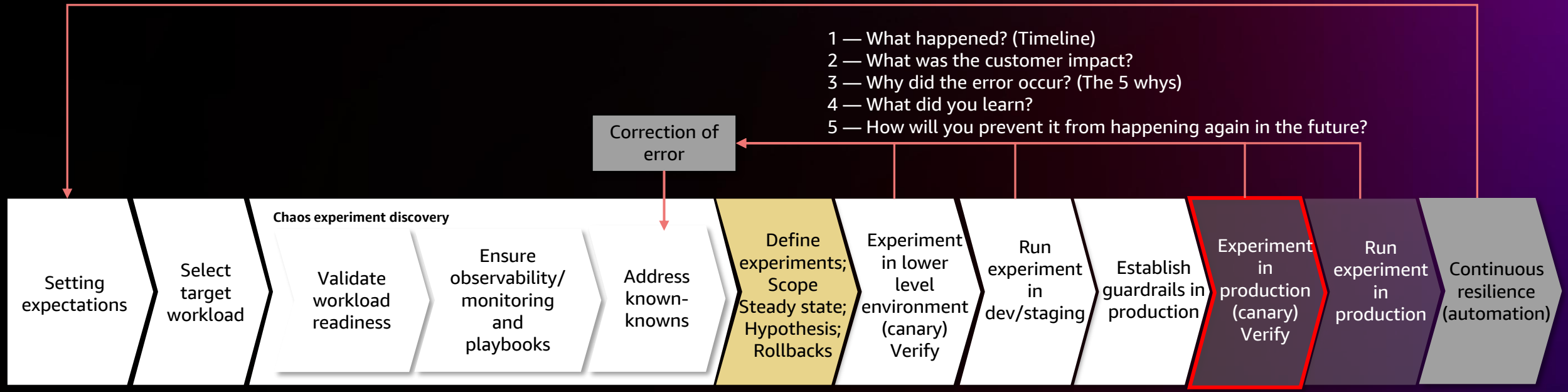


Establish guardrails in production

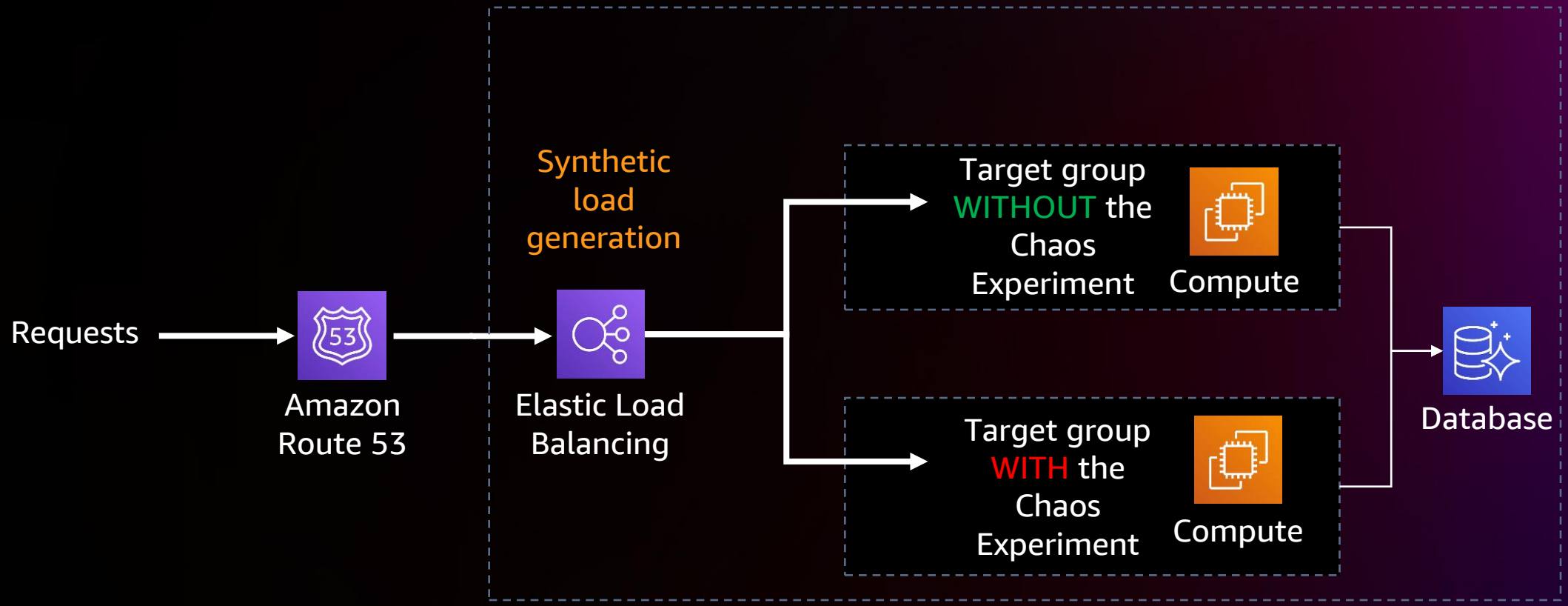
- Run experiments off peak hours
- Validate that your IAM Permissions are sufficient for your experiment in production
- Validate if your fault isolation boundary is the same as in the lower-level environment or changes in the production environment
- Decide if synthetic traffic will be generated, or if you are planning to run the experiment against a subset of your customers
- Validate that observability is in place
- Validate that your runbooks/playbook are up to date in production

Prime the prod environment for the experiment

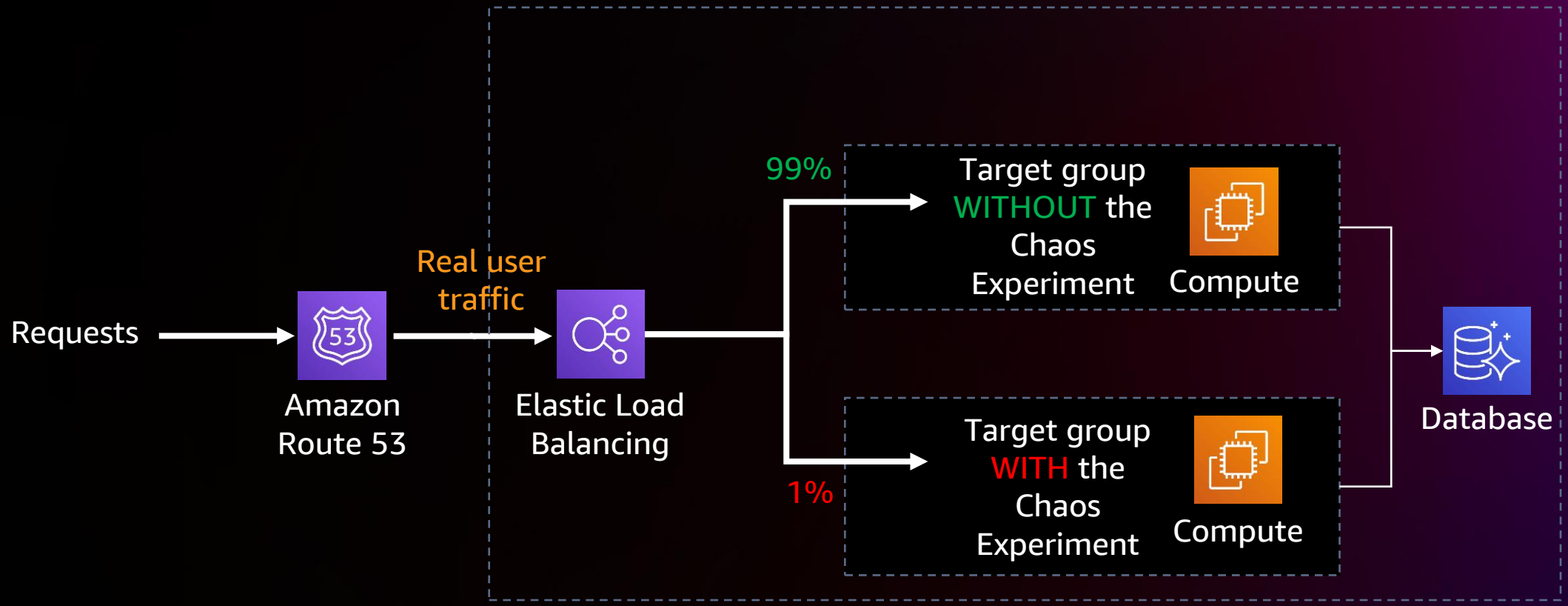
PRIORITIZE FINDINGS, FIX THEM, RE-ITERATE; CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS



Controlled experiments through canary deployments

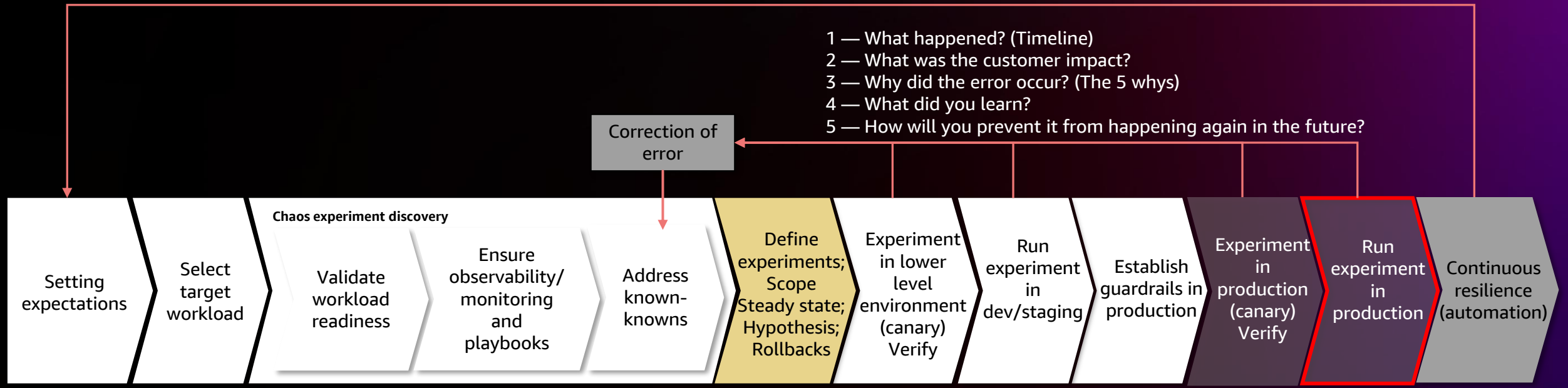


Controlled experiments through canary deployments

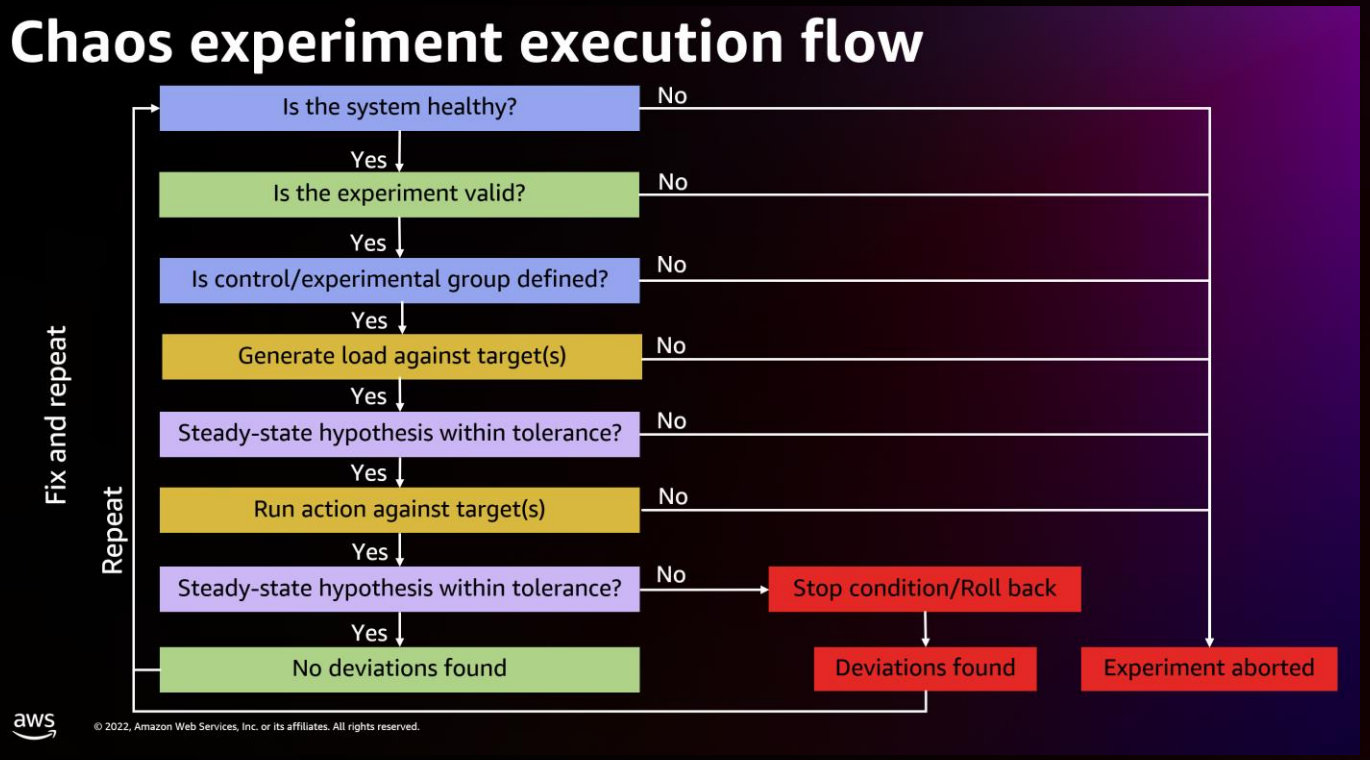


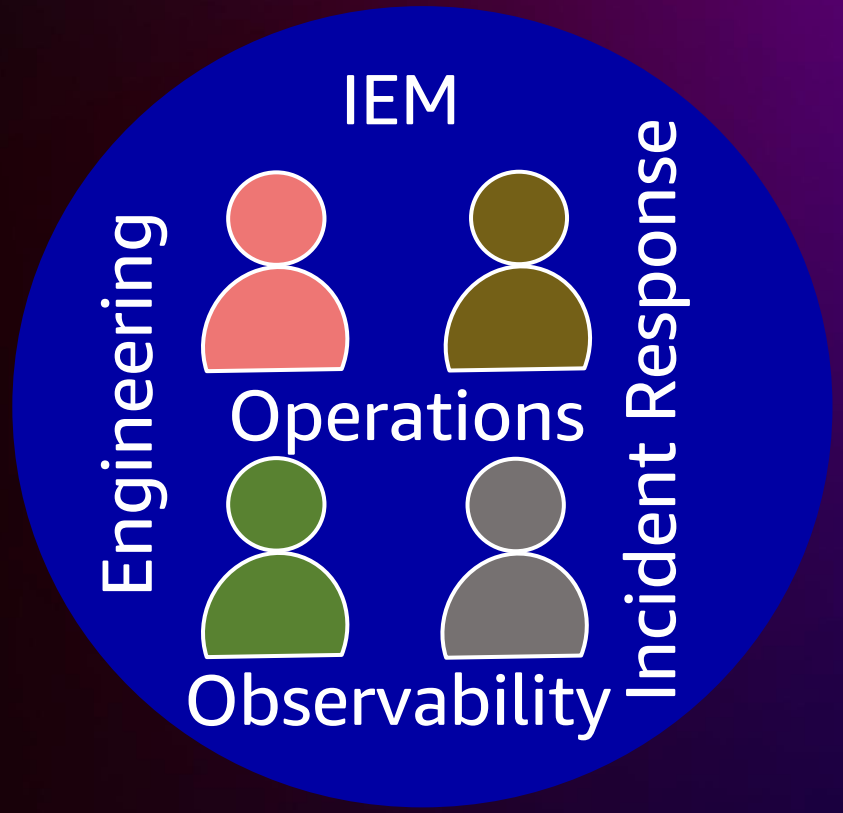
Run the experiment in production

PRIORITIZE FINDINGS, FIX THEM, RE-ITERATE; CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS



Execute the controlled experiment in production

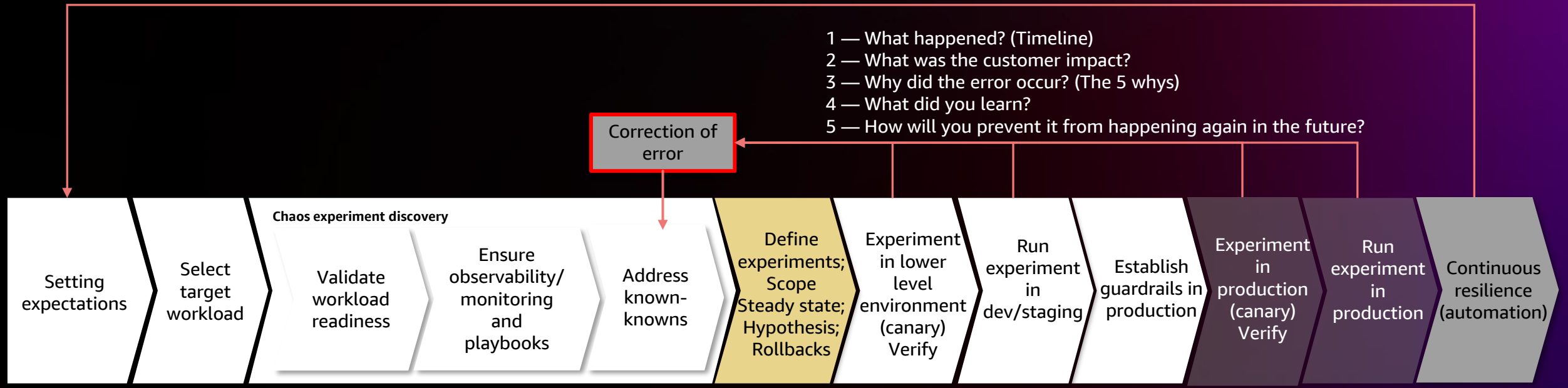




Workload Experts

Run the experiment in production

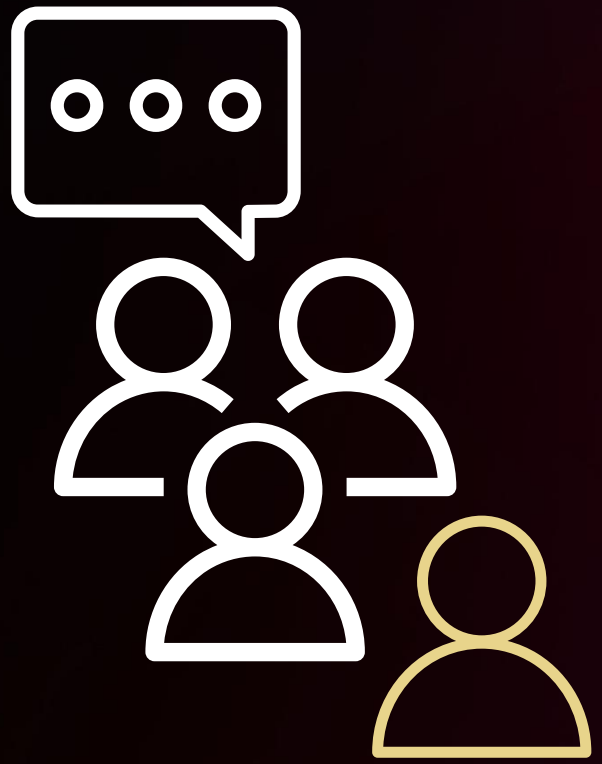
PRIORITIZE FINDINGS, FIX THEM, RE-ITERATE; CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS



Post mortem - Correction of error

What did we learn?

How did we communicate

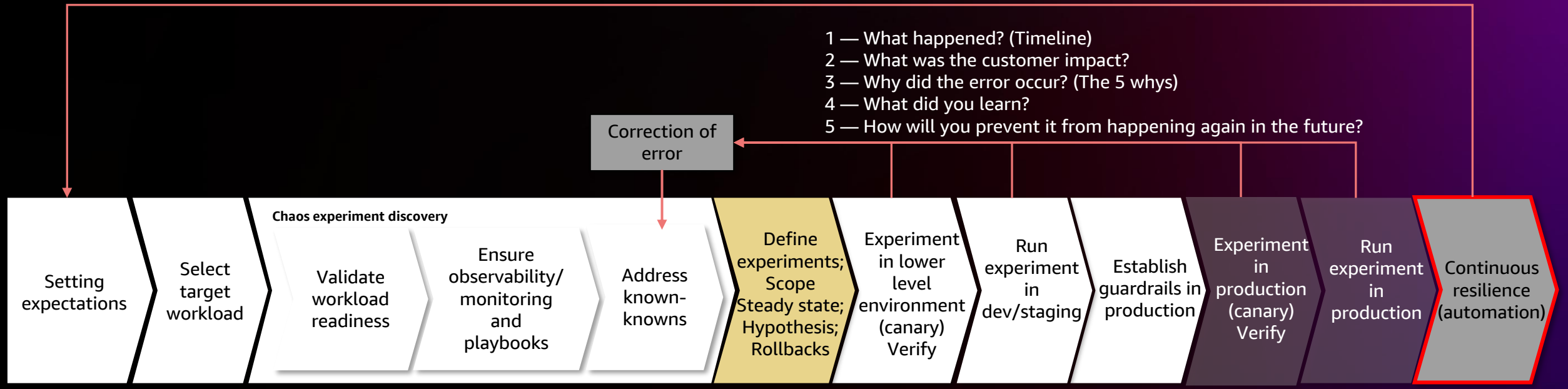


Was everyone needed present?

How do we share what we've learned?

Continuous resilience

PRIORITIZE FINDINGS, FIX THEM, REITERATE; CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS

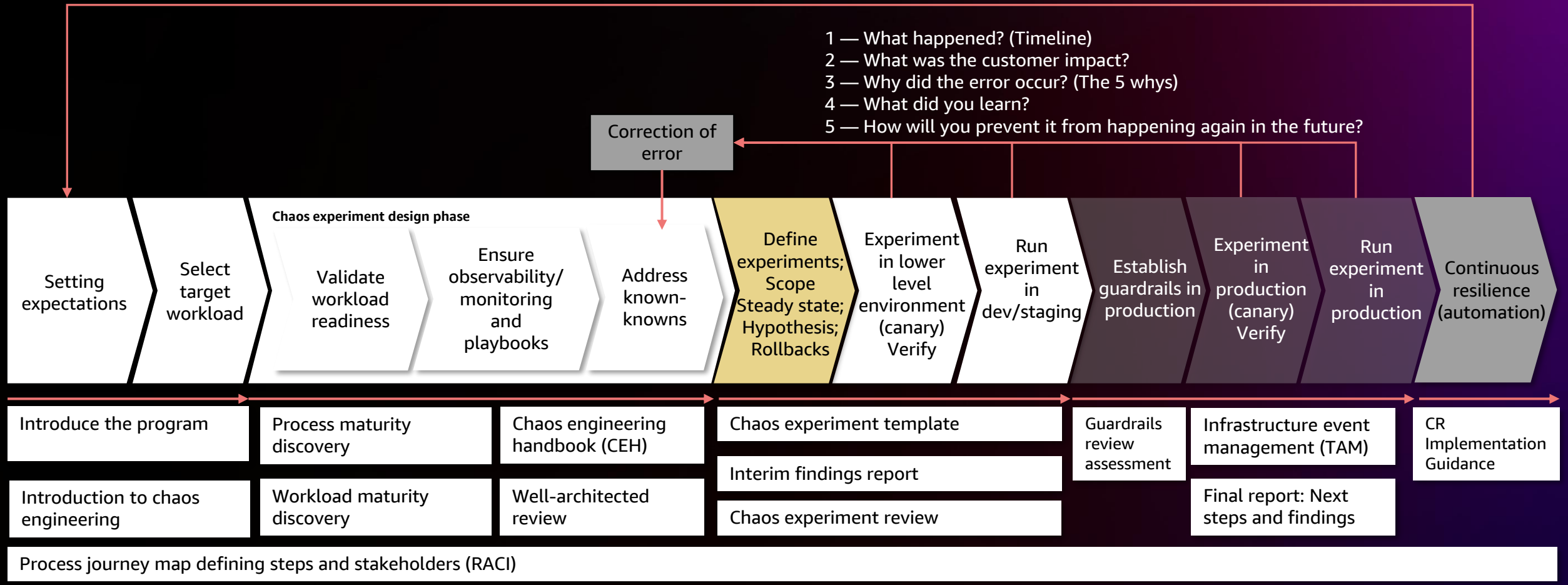


Automate the experiments

AUTOMATE

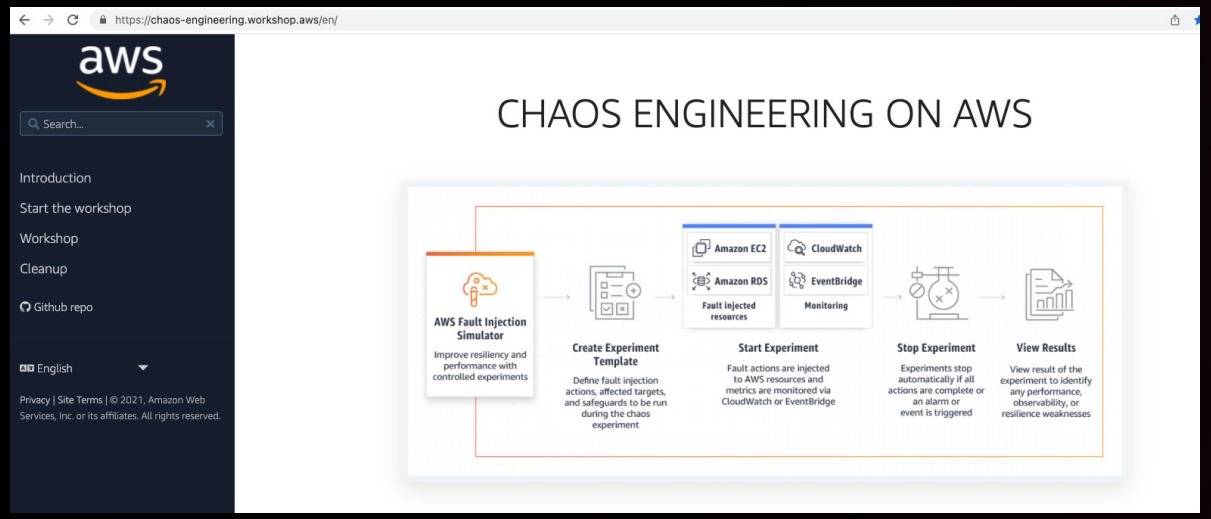
Supporting documents

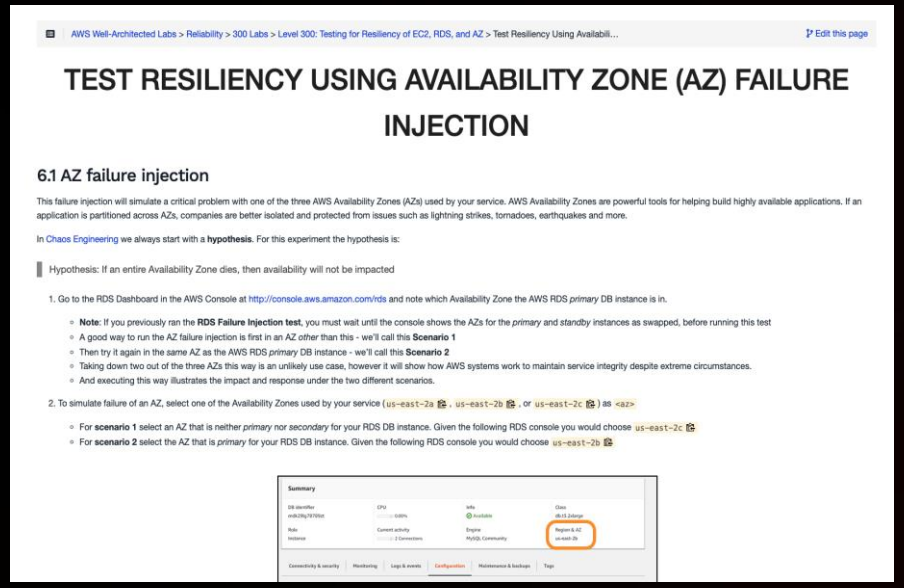
PRIORITIZE FINDINGS, FIX THEM, REITERATE; CONTINUOUSLY RUN EXPERIMENTS AS OFTEN AS WORKLOAD NEEDS; SCALE MECHANISM TO OTHER WORKLOADS

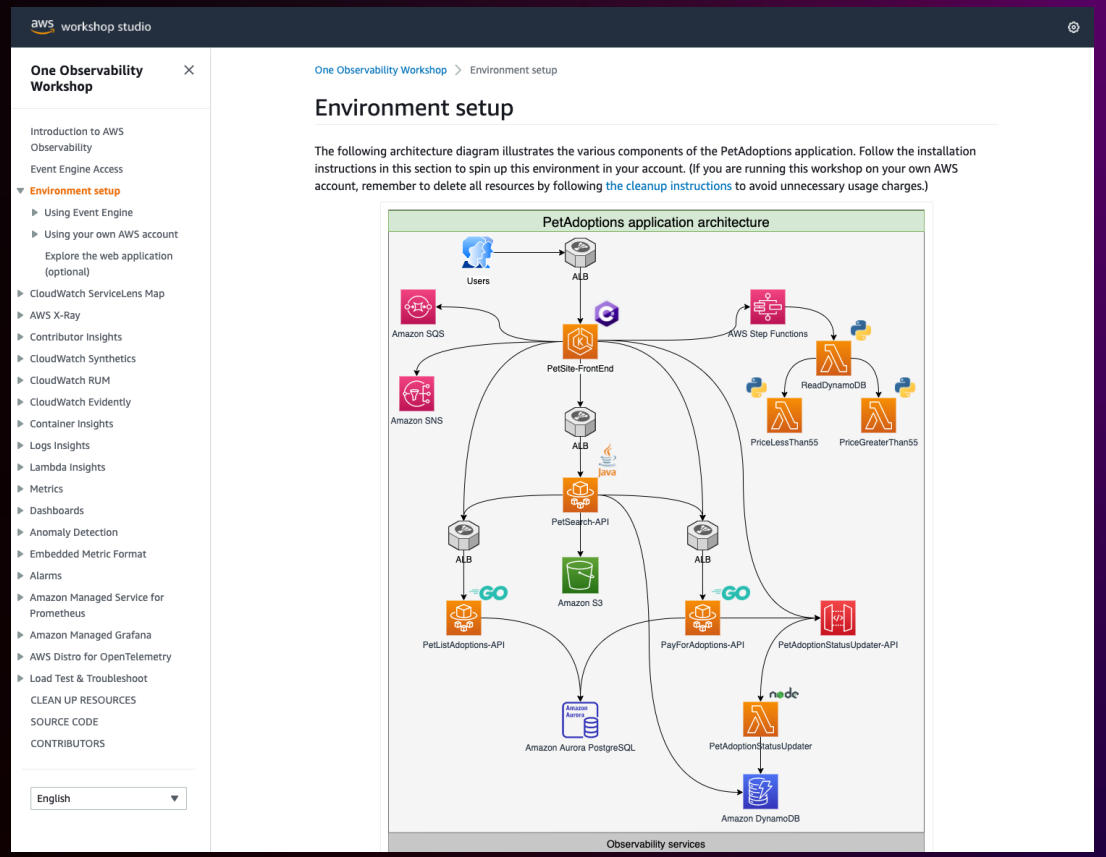


Chaos engineering AWS resources

Chaos engineering/observability workshops







Chaos engineering/observability workshops

Chaos engineering on AWS

Validating security guardrails with chaos engineering

Resilience engineering

Serverless chaos workshop

Observability workshop

AWS fault isolation boundaries

Multi-AZ resilience patterns

Chaos engineering stories

Thank you!

Laurent Domb
ldo@amazon.com

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