

# aws INNOVATE

FOR EVERY APPLICATION EDITION

25 August, 2022

# Resiliency and availability design patterns for the cloud

Chandra Munibudha  
Principal Solutions Architect  
AISPL



# Today's agenda

- ❖ Scope of Resiliency

- ❖ Resilient Architectures

  - Timeouts, Retries with jitter*

  - Load Shedding*

  - Constant Work*

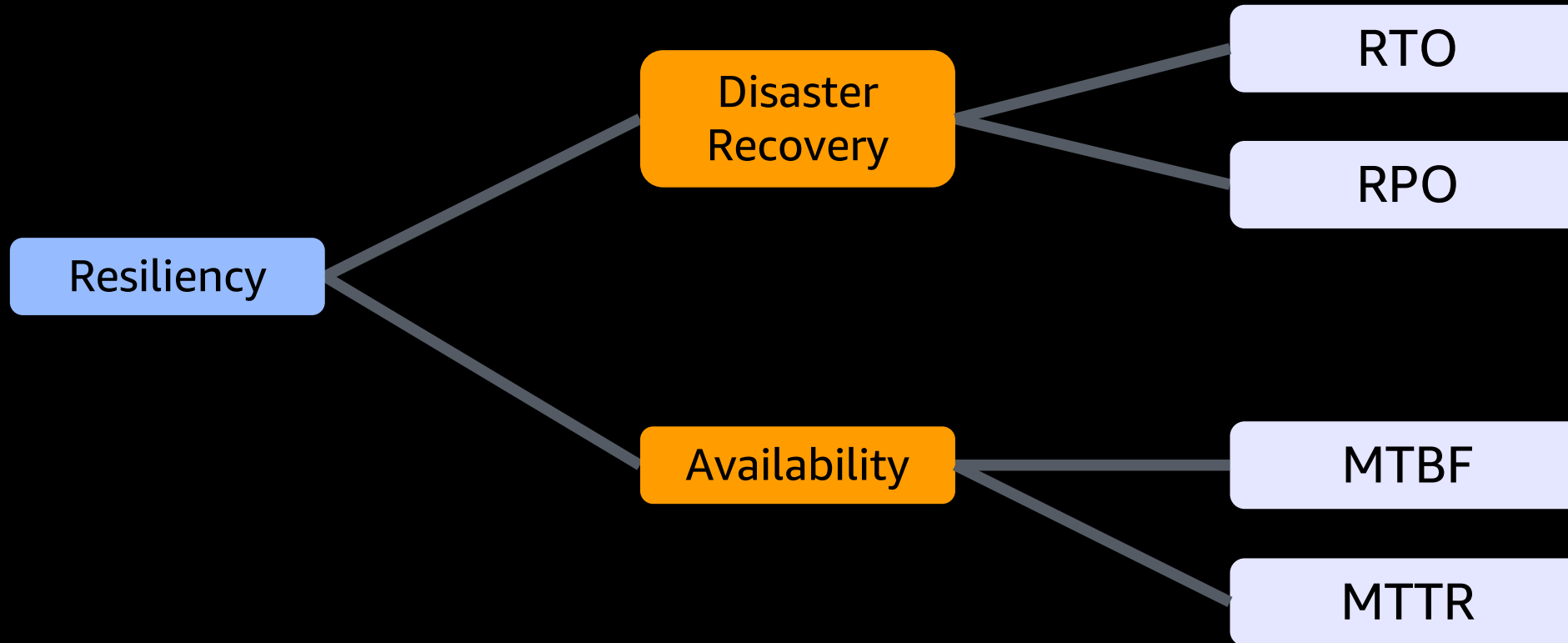
  - Static Stability*

  - Shuffle Sharding*

# What is resiliency?

**“Resilience is the ability of a system to adapt or keep working when challenges occur”**

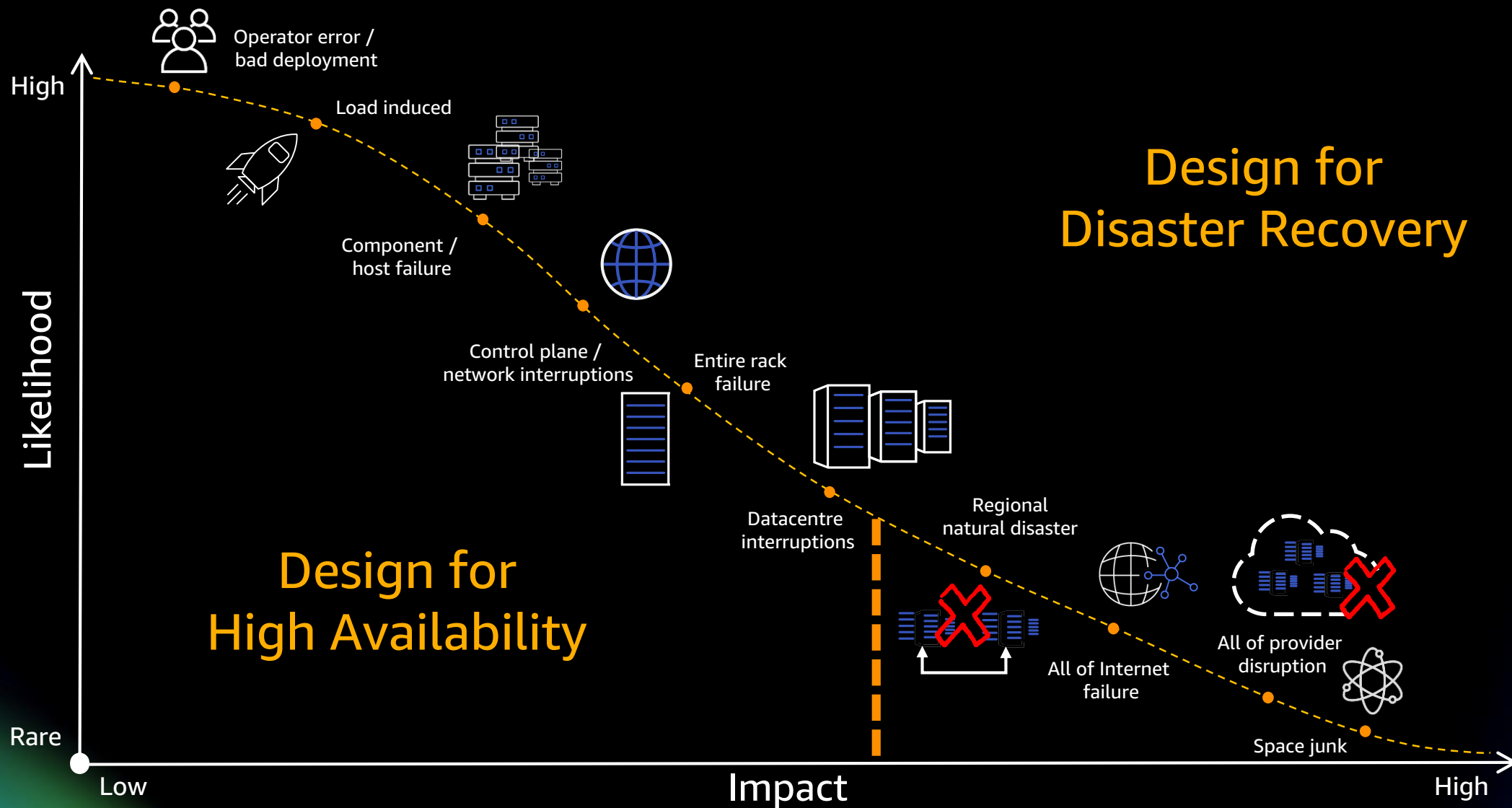
# AWS whitepaper definition



- RTO (Recovery Time)
- RPO (Recovery Point)
- MTBF (Mean Time Between Failures)
- MTTR (Mean Time to Recover)

# Categories of failure

## TYPES OF FAILURE



# Resilient Design Patterns

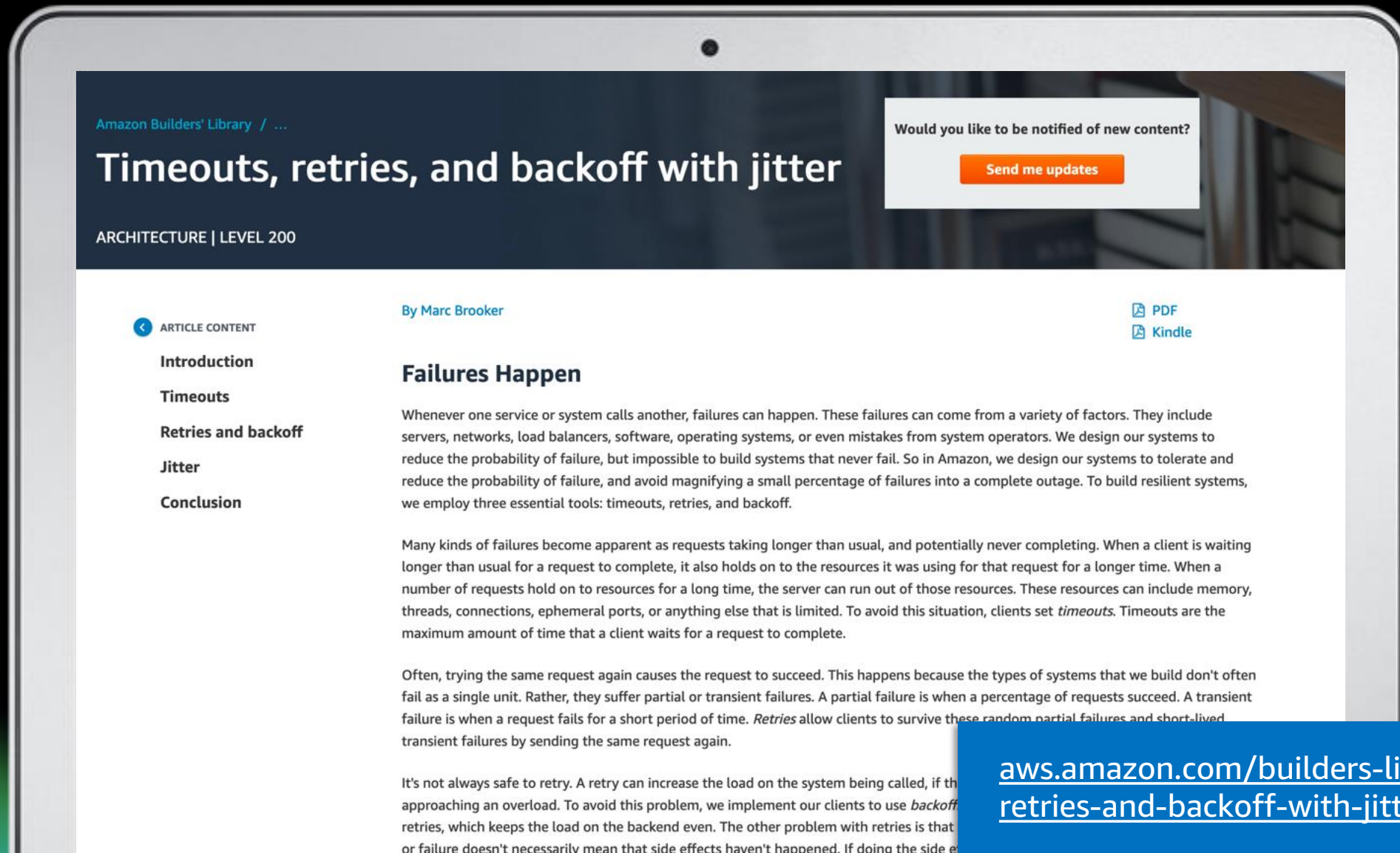
# Client-side patterns

- **Timeouts** - maximum amount of time that a client (or calling dependency) waits for a request to complete.
- **Retries** - survive random partial failures and short-lived transient failures
- Use **backoff** – *Increase* the time between subsequent retries, which keeps the load on the backend even
- **Jitter** - This is a random amount of time before retrying a request to help prevent large simultaneous bursts

AWS Resilience Workshop: <https://tinyurl.com/bhjbsk67>



# Timeouts, retries, and backoff with jitter

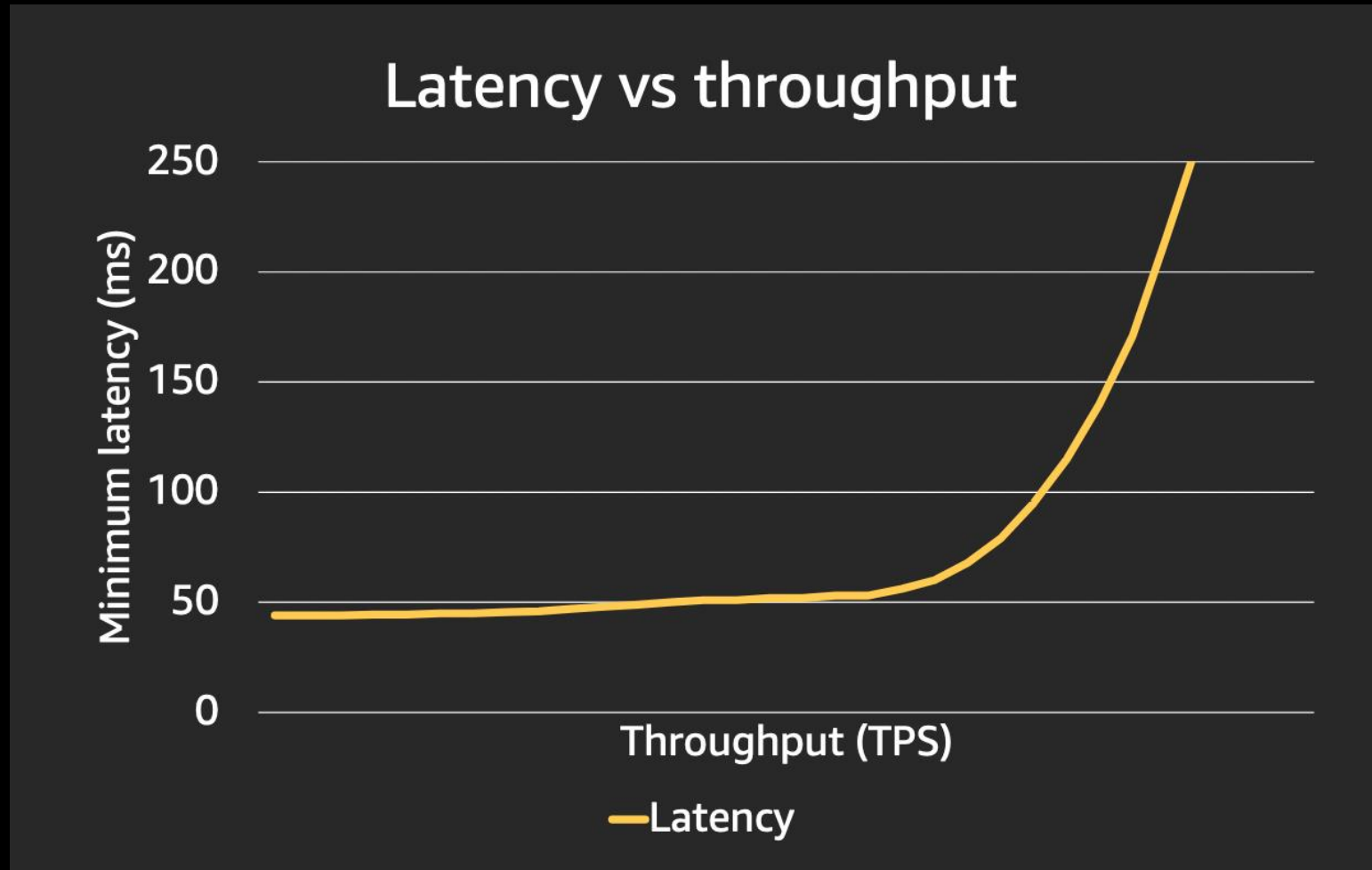


# Server/Backend patterns

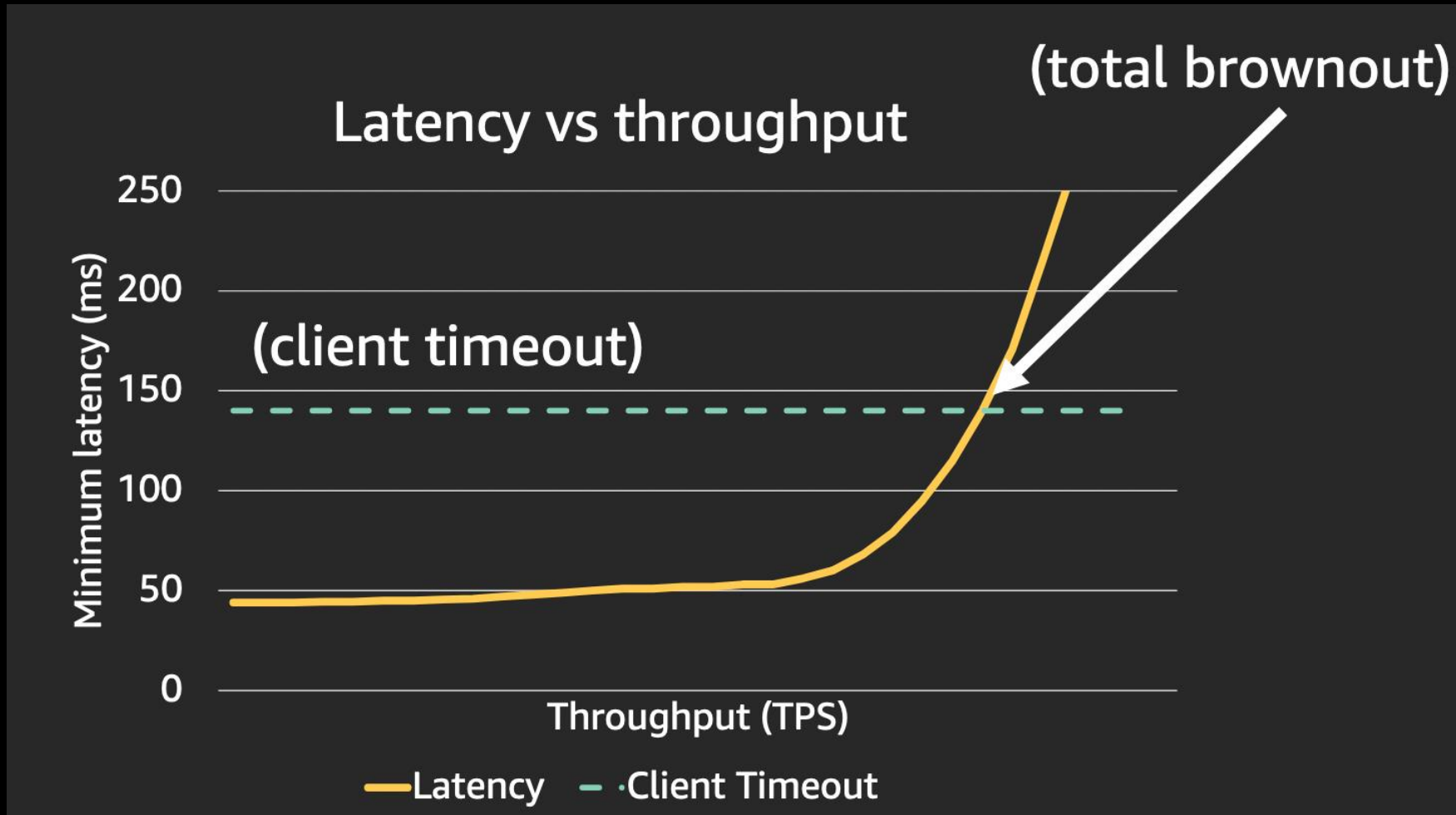
- **Load Shedding** – avoid brownout by rejecting excess load.
- **Constant Work** – survive random partial failures and short-lived transient failures.
- **Static Stability** – the overall system keeps working even when a dependency becomes impaired.
- **Shuffle Sharding** – *Isolating customers/resources to virtual shards and reduce overall impact of bad requests.*

# Load Shedding

# Load Shedding



# Load Shedding





# Load Shedding

Amazon Builders' Library / ...

## Using load shedding to avoid overload

SOFTWARE DELIVERY AND OPERATIONS | LEVEL 400

Would you like to be notified of new content?

Send me updates

ARTICLE CONTENT

By David Yanacek

PDF  
Kindle

Introduction

The anatomy of overload

Testing

Visibility

Load shedding mechanisms

Thinking about overload differently

Further reading

For a few years, I worked on the Service Frameworks team at Amazon. Our team wrote tools that helped the owners of AWS services such as Amazon Route 53 and Elastic Load Balancing build their services more quickly, and service clients call those services more easily. Other Amazon teams provided service owners with functionality such as metering, authentication, monitoring, client library generation, and documentation generation. Instead of each service team having to integrate those features into their services manually, the Service Frameworks team did that integration once and exposed the functionality to each service through configuration.

One challenge we faced was in determining how to provide sensible defaults, especially for features that were performance or availability related. For example, we couldn't set a default client-side timeout easily, because our framework had no idea what the latency characteristics of an API call might be. This wouldn't have been any easier for service owners or clients to figure out themselves, so we kept trying, and gained some useful insights along the way.

One common question we struggled with was determining the default number of connections for service clients at the same time. This setting was designed to prevent a server from taking on too many connections. Specifically, we wanted to configure the maximum connections settings for the server. This was before the days of Elastic Load Balancing, so hardware load balancers were used.

We set out to help Amazon service owners and service clients figure out the ideal value for the maximum connections setting.

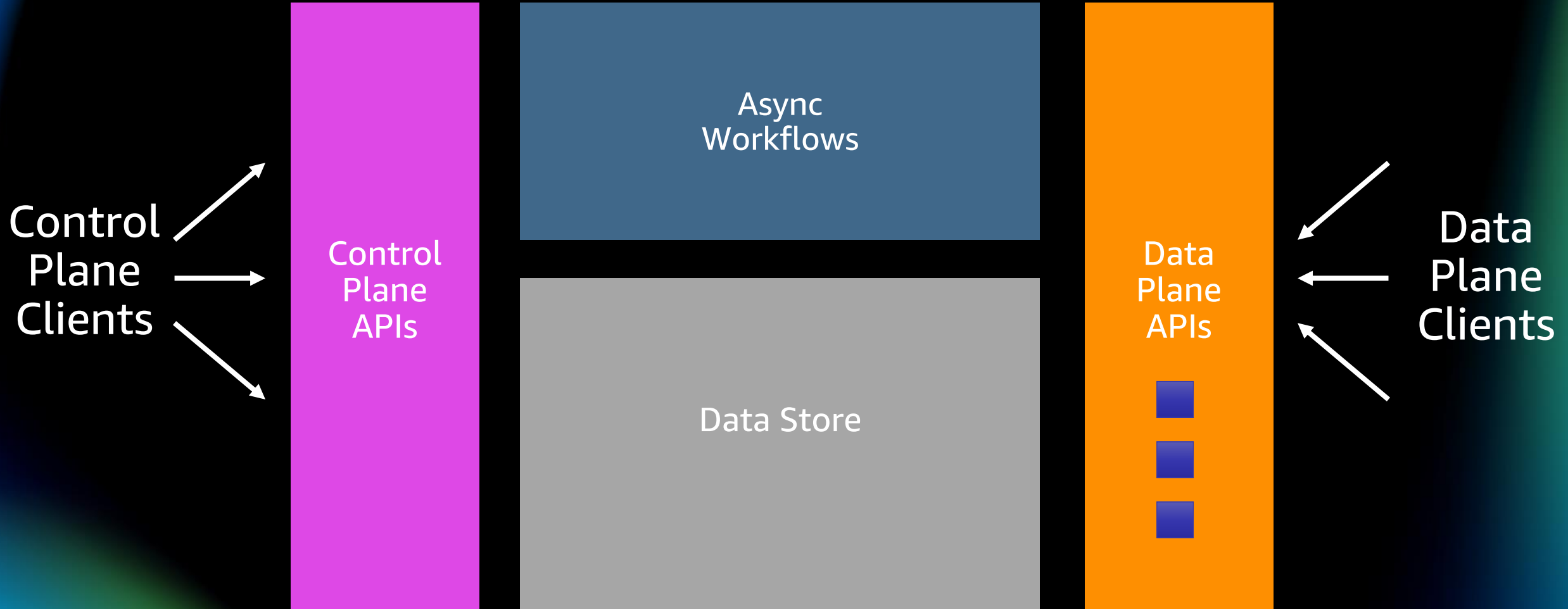
[aws.amazon.com/builders-library/using-load-shedding-to-avoid-overload/](https://aws.amazon.com/builders-library/using-load-shedding-to-avoid-overload/)



# Static stability

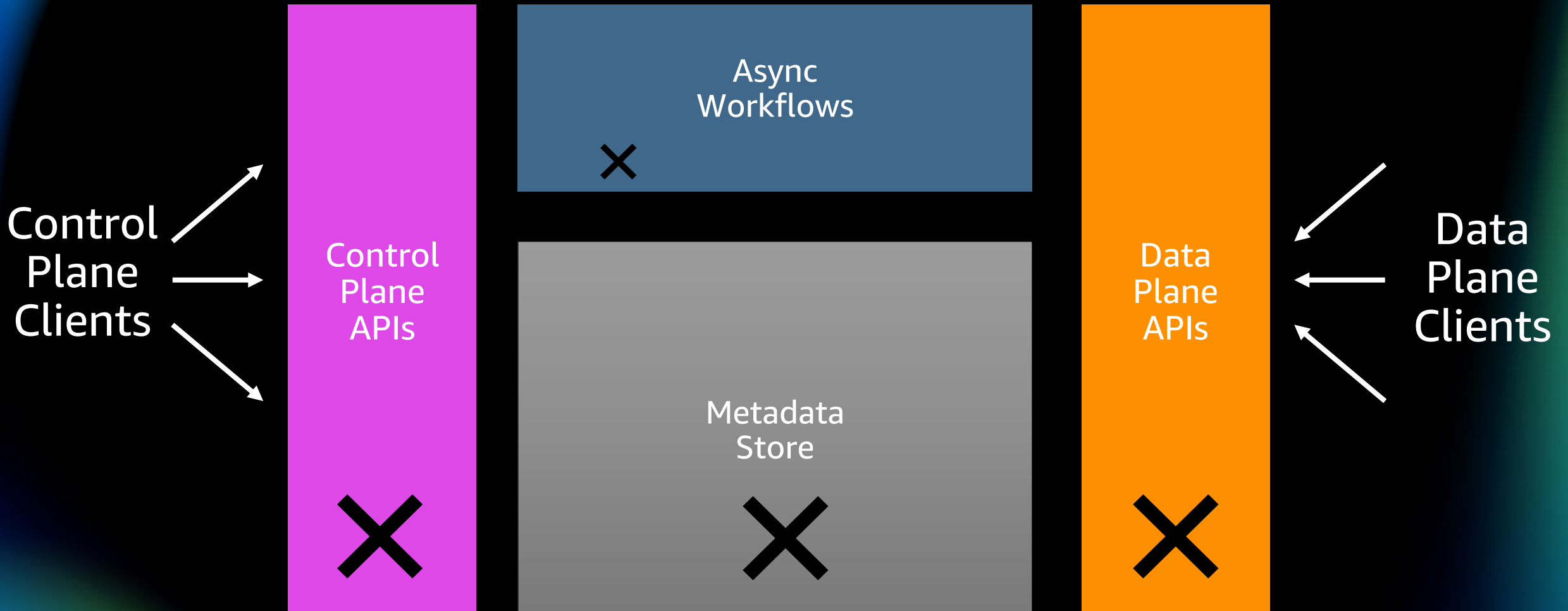
# Control Plane and Data Plane

MODULAR SEPARATION CREATES STATIC STABILITY

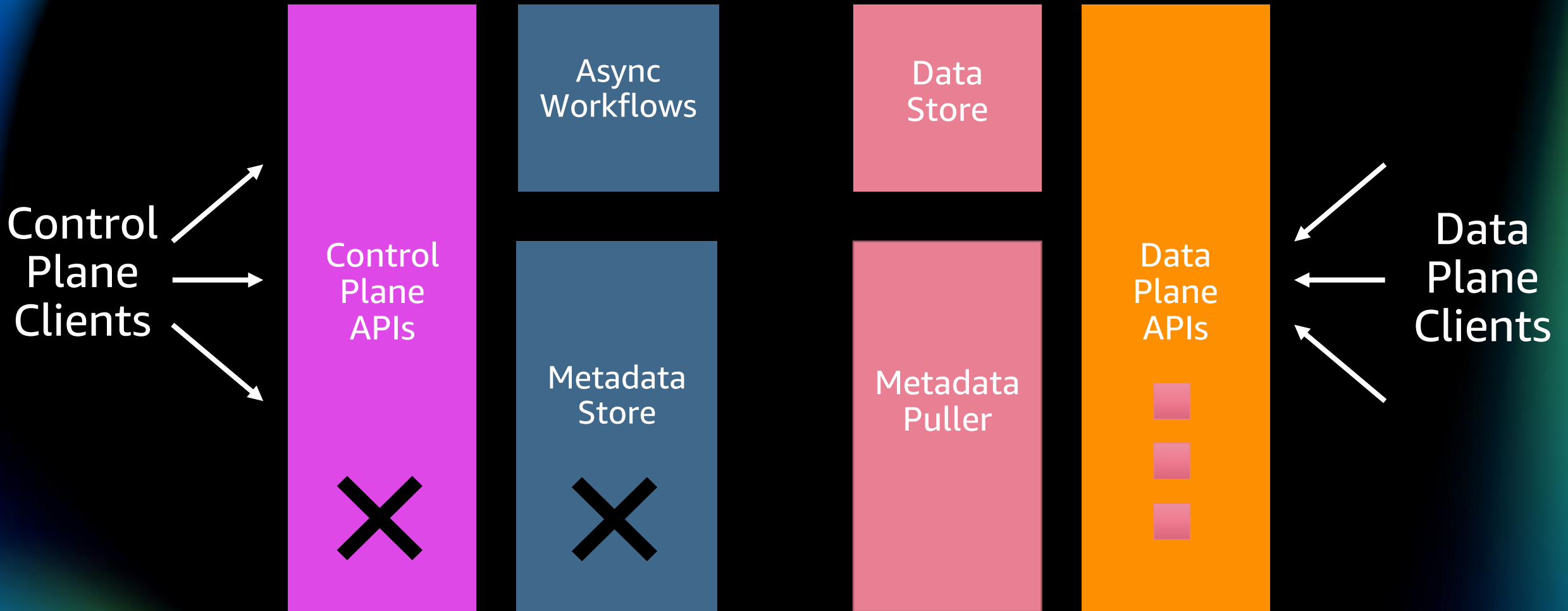




# Static Stability



# Static Stability



# Static Stability

Amazon Builders' Library / ...

## Static stability using Availability Zones

ARCHITECTURE | LEVEL 300

Would you like to be notified of new content?

Send me updates

ARTICLE CONTENT

Introduction

Static stability

Static stability patterns

Under the hood: Static stability inside of Amazon EC2

Conclusion

By [Becky Weiss](#) and [Mike Furr](#)

 PDF  
 Kindle

At Amazon, the services we build must meet extremely high availability targets. This means that we need to think carefully about the dependencies that our systems take. We design our systems to stay resilient even when those dependencies are impaired. In this article, we'll define a pattern that we use called *static stability* to achieve this level of resilience. We'll show you how we apply this concept to Availability Zones, a key infrastructure building block in AWS and therefore a bedrock dependency on which all of our services are built.

In a statically stable design, the overall system keeps working even when a dependency becomes impaired. Perhaps the system doesn't see any updated information (such as new things, deleted things, or modified things) that its dependency was supposed to have delivered. However, everything it was doing before the dependency became impaired continues to work despite the impaired dependency. We'll describe how we built Amazon Elastic Compute Cloud (EC2) to be statically stable. Then we'll provide two statically stable example architectures we've found useful for building highly available regional systems on top of Availability Zones.

Finally, we'll go deeper into some of the design philosophy behind Amazon EC2 including how it's architected to provide Availability Zone independence at the software level. In addition, we'll discuss some of the tradeoffs that come with building a service with this choice of architecture.

The role of Availability Zones

Availability Zones are logically isolated sections of an AWS Region: Each zone operates independently. Availability Zones are physically separated by a

[aws.amazon.com/builders-library/static-stability-using-availability-zones](https://aws.amazon.com/builders-library/static-stability-using-availability-zones)



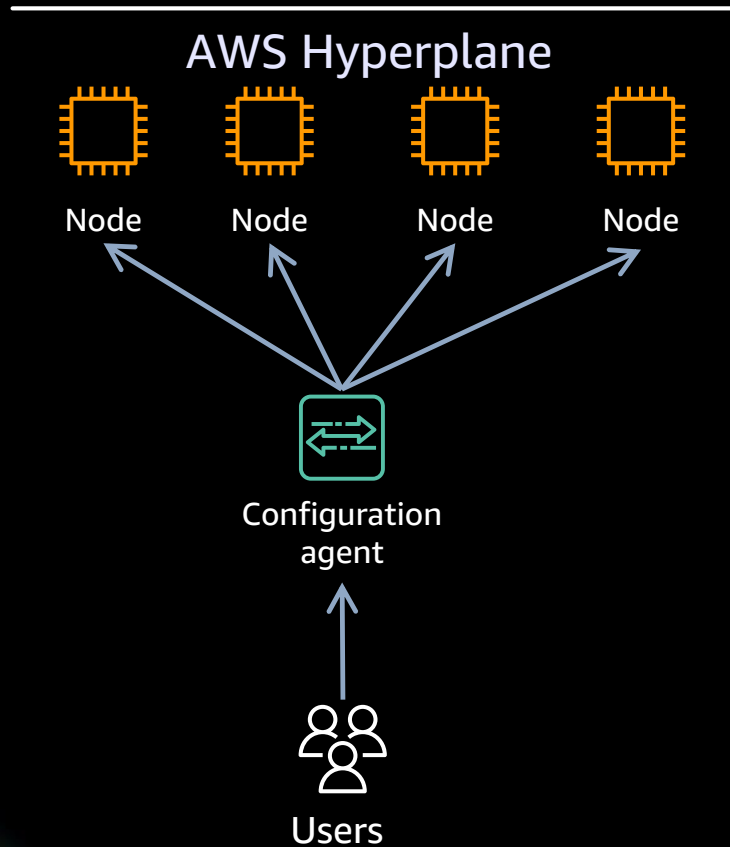
# Simple designs and constant work

# Simple designs and constant work

- Risk is often 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

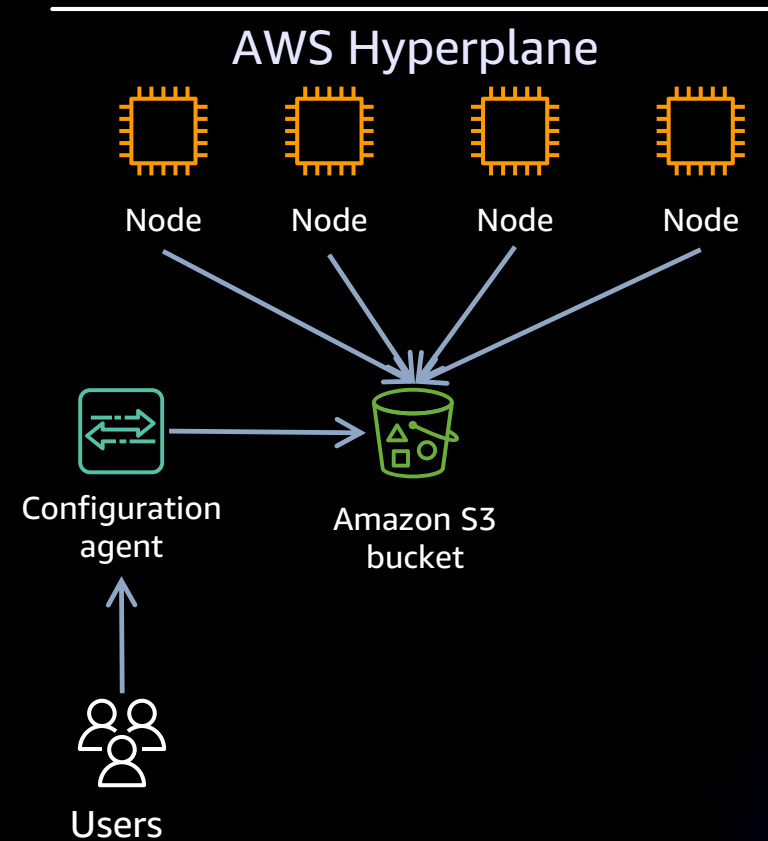
# Simple designs and constant work

## PUSH



VS.

## PULL



# Simple designs and constant work

Amazon Builders' Library / ...

## Reliability, constant work, and a good cup of coffee

ARCHITECTURE | LEVEL 300

Would you like to be notified of new content?

Send me updates

ARTICLE CONTENT

### Introduction

Computers: They do exactly as you tell them

Amazon Route 53 health checks and healthiness

Amazon S3 as a configuration loop

Constant work and self-healing

Design and manageability

The value of a simple design

By Colm MacCárthaigh

PDF

One of my favorite paintings is “[Nighthawks](#)” by Edward Hopper. A few years ago, I was lucky enough to see it in person at the Art Institute of Chicago. The painting’s scene is a well-lit glassed-in city diner, late at night. Three patrons sit with coffee, a man with his back to us at one counter, and a couple at the other. Behind the counter near the single man a white-coated server crouches, as if cleaning a coffee cup. On the right, behind the server loom two coffee urns, each as big as a trash can. Big enough to brew cups of coffee by the hundreds.

Coffee urns like that aren’t unusual. You’ve probably seen some shiny steel ones at many catered events. Conference centers, weddings, movie sets. . . we even have urns like these in our kitchens at Amazon. Have you ever thought about why coffee urns are so big? Because they are always ready to dispense coffee, the large size has to do with constant work.



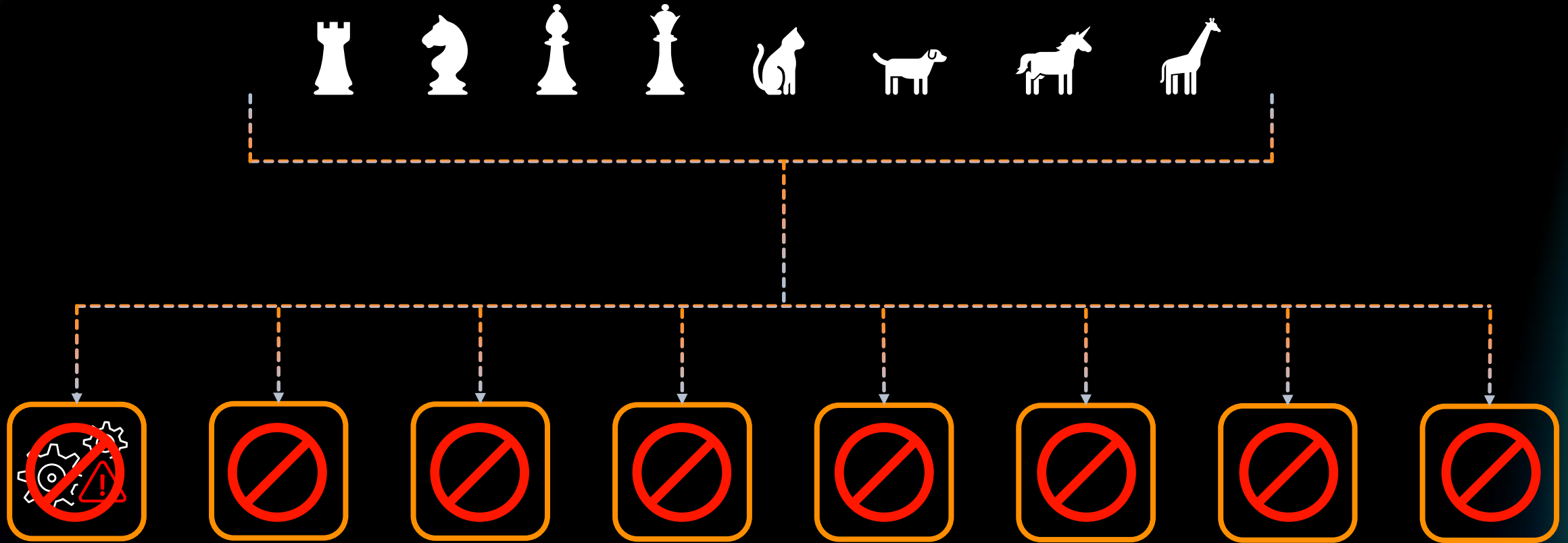
[aws.amazon.com/builders-library/reliability-and-constant-work](https://aws.amazon.com/builders-library/reliability-and-constant-work)



# Shuffle Sharding



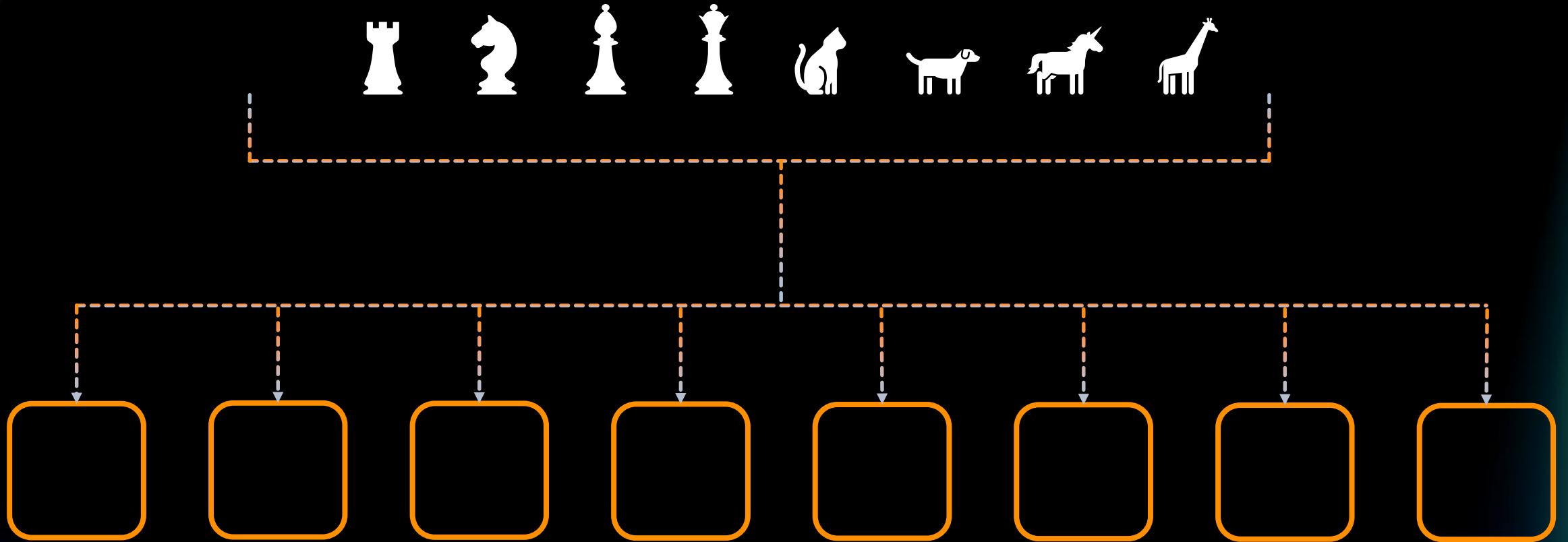
# Traditional architecture



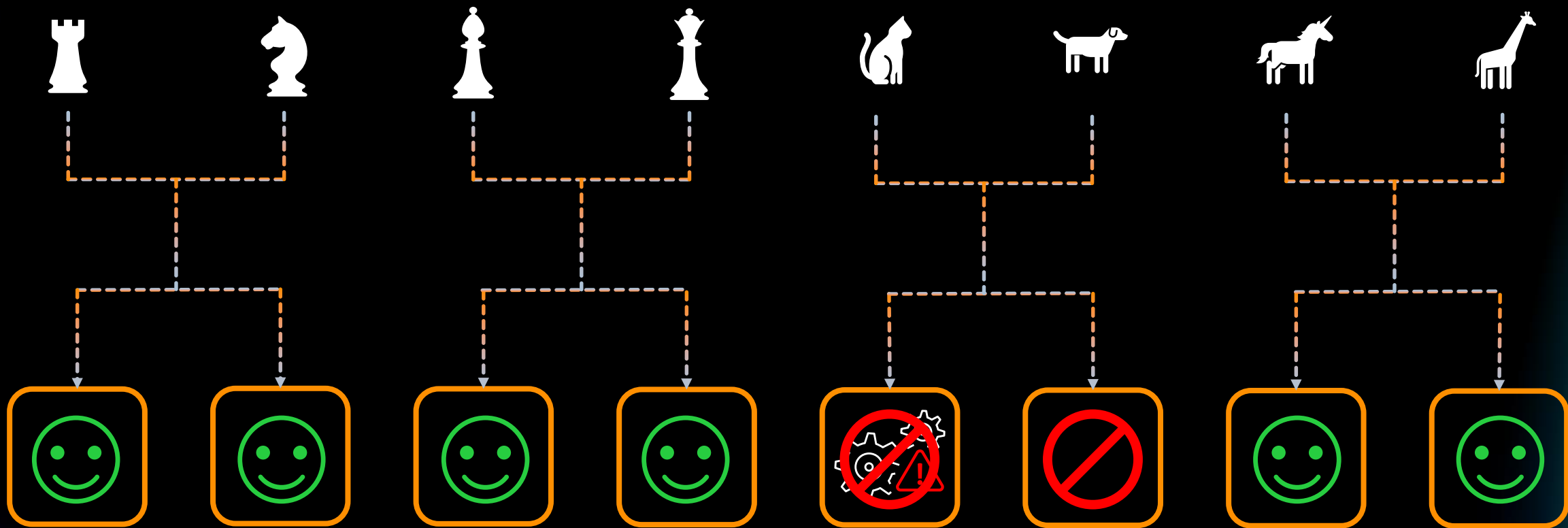
Scope of impact = All customers

# Combining **Partitioning** and **Replication**

# Sharding

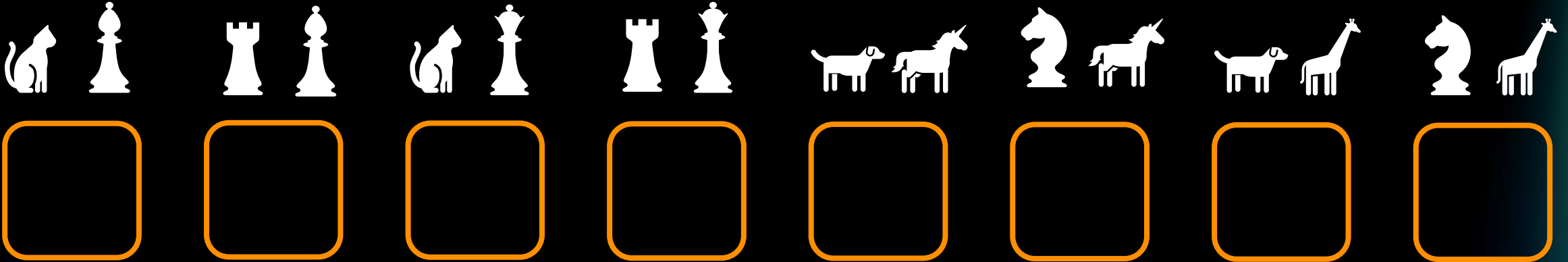
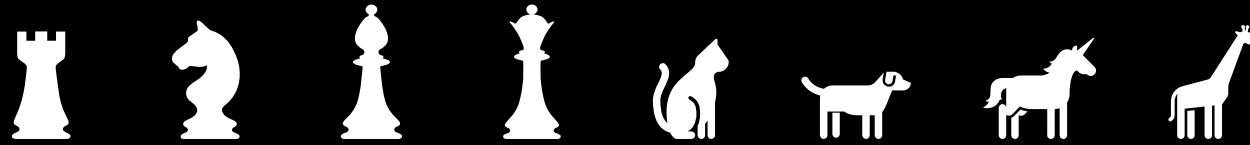


# Sharding



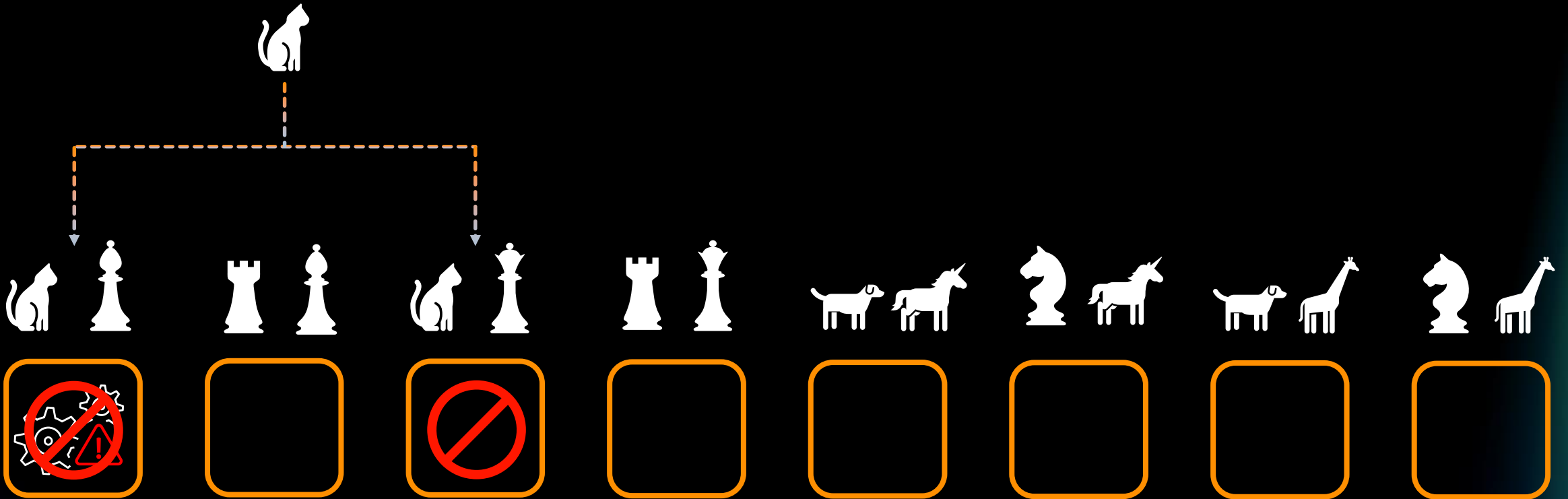
Scope of impact =  $\frac{\text{Customers}}{\text{Shards}}$

# Shuffle sharding

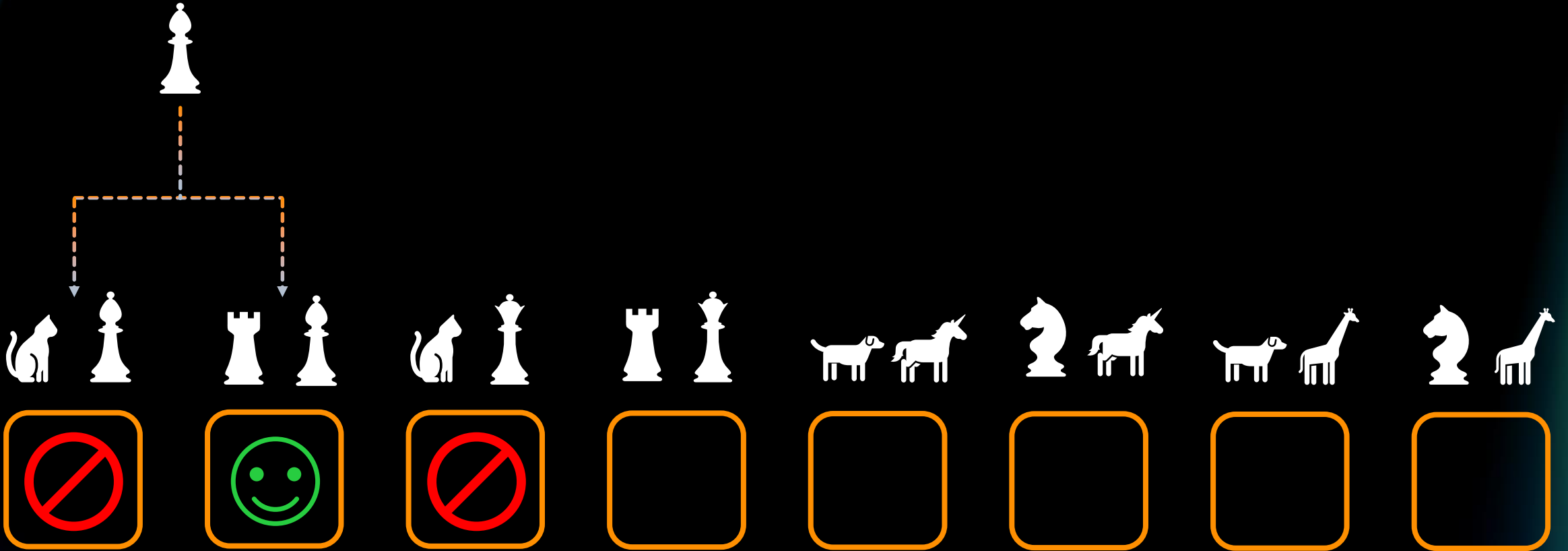


By choosing two instances from eight there are 56 potential shuffle shards, much more than the four simple shards we had before.

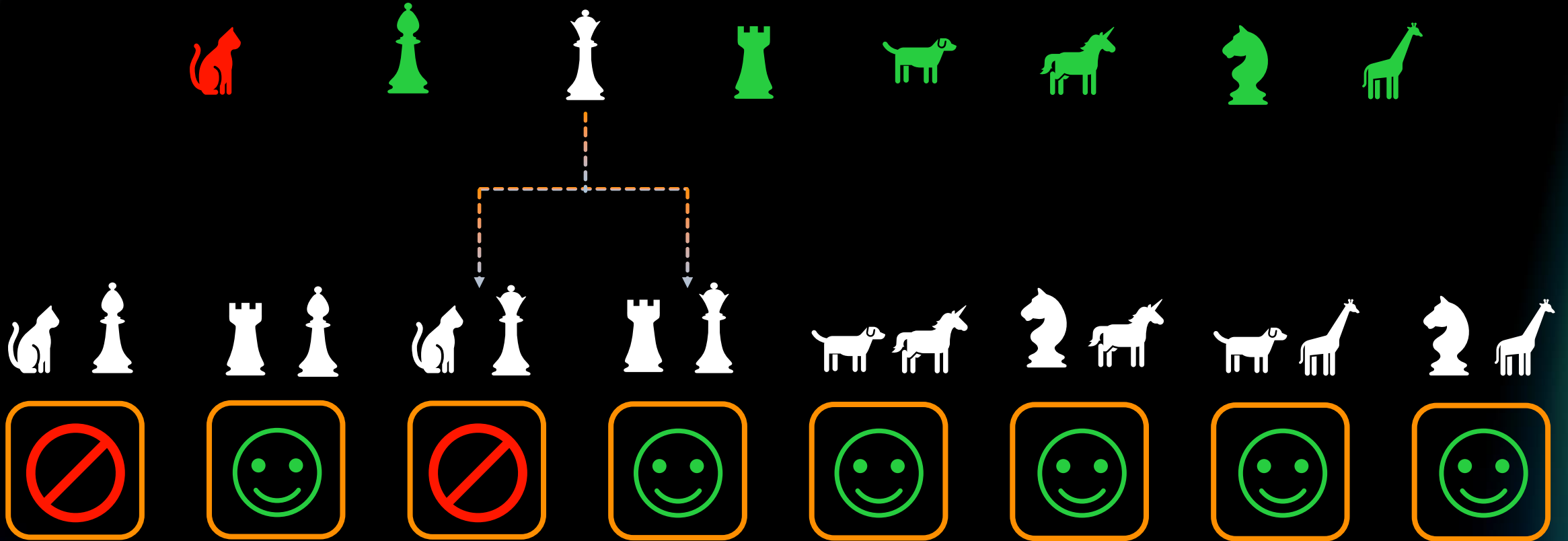
# Shuffle sharding



# Shuffle sharding



# Shuffle sharding



$$\text{Scope of impact} = \frac{\text{Customers}}{\text{Combinations}}$$

\*retry logic in the client to try every endpoint in a Shuffle Shard, until one succeeds



# Shuffle Sharding

- Needs a client that retires or is fault tolerant
- Needs a routing mechanism – per customer DNS name, per resource DNS name of Shuffle sharding aware router
- Sharding strategy - customer id, customer-resource-operation-type, multidimensional sharding

# Shuffle Sharding

Amazon Builders' Library / ...

## Workload isolation using shuffle-sharding

ARCHITECTURE | LEVEL 400

Would you like to be notified of new content?

Send me updates

ARTICLE CONTENT

Introduction

Taking on DNS hosting

Handling DDoS attacks

What is shuffle sharding?

Amazon Route 53 and  
shuffle sharding

Conclusion

Hands-on lab

By Colm MacCárthaigh

PDF  
Kindle

Today, Amazon Route 53 hosts many of the world's biggest businesses and most popular websites, but its beginnings are far more humble.

### Taking on DNS hosting

Not long after AWS began offering services, AWS customers made clear that they wanted to be able to use our Amazon Simple Storage Service (S3), Amazon CloudFront, and Elastic Load Balancing services at the "root" of their domain, that is, for names like "amazon.com" and not just for names like "www.amazon.com".

That may seem very simple. However, due to a design decision in the DNS protocol, made back in the 1980s, it's harder than it seems. DNS has a feature called CNAME that allows the owner of a domain to point a subdomain to another domain. However, CNAME doesn't work at the root or top level of a domain. To serve a customer's domain, we need to return whatever the customer wants at the root of their domain. When we host a customer's domain, we can return whatever the customer wants at the root of their domain. Elastic Load Balancing. These services are constantly expanding. Customers don't want to have to hard-code in their domain configurations either.

[aws.amazon.com/builders-library/workload-isolation-using-shuffle-sharding](https://aws.amazon.com/builders-library/workload-isolation-using-shuffle-sharding)



# Summary

1. **Timeouts, Retries** – Use timeouts on any remote or inter-process calls. Think of idempotency while retrying.
2. **Load Shedding** – Avoid overloading. Understand the cost of rejecting a request. Categorize your incoming request types and decide which ones to shed.
3. **Static Stability** – Build some redundancy into your data plane so that they can continue to work despite control plane failure.
4. **Constant Work** – Work that a system does should be idempotent in nature and not get affected by variations in load or stress.
5. **Shuffle Sharding** – Applying fault isolation to traditional horizontal scaling.

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

# Thank you for attending AWS Innovate – For Every Application Edition

We hope you found it interesting! A kind reminder to **complete the survey**.  
Let us know what you thought of today's event and how we can improve the event  
experience for you in the future.



[aws-apj-marketing@amazon.com](mailto:aws-apj-marketing@amazon.com)



[twitter.com/AWSCloud](https://twitter.com/AWSCloud)



[facebook.com/AmazonWebServices](https://facebook.com/AmazonWebServices)



[youtube.com/user/AmazonWebServices](https://youtube.com/user/AmazonWebServices)



[slideshare.net/AmazonWebServices](https://slideshare.net/AmazonWebServices)



[twitch.tv/aws](https://twitch.tv/aws)

# Thank you!