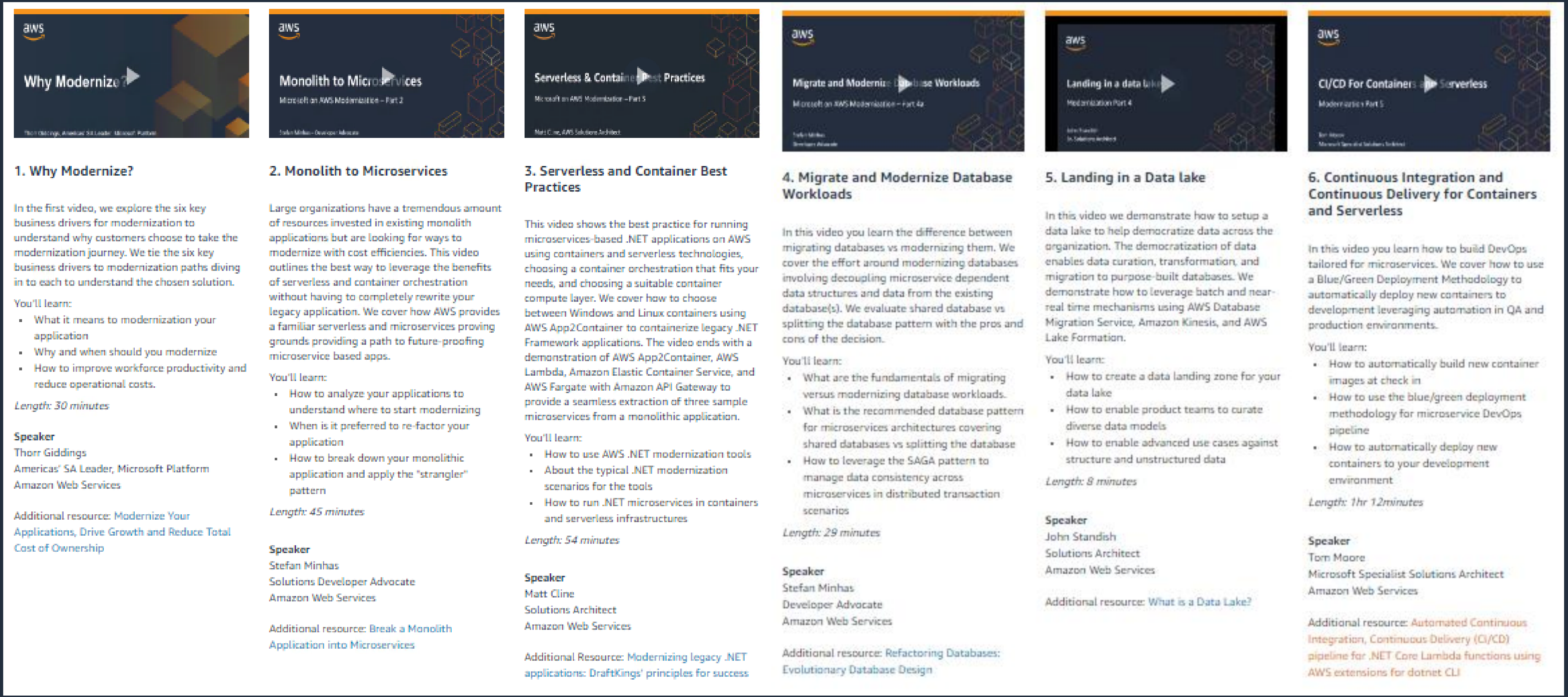
Migrations

Breaking the Monolith

Re-factoring your legacy applications to microservices

Stefan Minhas
Solution Architect

External On-Demand Series:
A Customer's Journey Modernizing Microsoft Workloads on AWS



Why Modernize?
Monolith to Microservices
Microsoft on AWS Modernization – Part 2
Serverless & Container Best Practices
Microsoft on AWS Modernization – Part 3
Matt Cline, AWS Solutions Architect
Migrate and Modernize Database Workloads
Microsoft on AWS Modernization – Part 4a
Landing in a data lake
Modernization Part 4
CI/CD For Containers and Serverless
Modernization Part 5

1. Why Modernize?

In the first video, we explore the six key business drivers for modernization to understand why customers choose to take the modernization journey. We tie the six key business drivers to modernization paths diving in to each to understand the chosen solution.

You'll learn:

- What it means to modernization your application
- Why and when should you modernize
- How to improve workforce productivity and reduce operational costs.

Length: 30 minutes

Speaker
Thorr Giddings
Americas' SA Leader, Microsoft Platform
Amazon Web Services

Additional resource: Modernize Your Applications, Drive Growth and Reduce Total Cost of Ownership

2. Monolith to Microservices

Large organizations have a tremendous amount of resources invested in existing monolith applications but are looking for ways to modernize with cost efficiencies. This video outlines the best way to leverage the benefits of serverless and container orchestration without having to completely rewrite your legacy application. We cover how AWS provides a familiar serverless and microservices proving grounds providing a path to future-proofing microservice based apps.

You'll learn:

- How to analyze your applications to understand where to start modernizing
- When is it preferred to re-factor your application
- How to break down your monolithic application and apply the "strangler" pattern

Length: 45 minutes

Speaker
Stefan Minhas
Solutions Developer Advocate
Amazon Web Services

Additional resource: Break a Monolith Application into Microservices

3. Serverless and Container Best Practices

This video shows the best practice for running microservices-based .NET applications on AWS using containers and serverless technologies, choosing a container orchestration that fits your needs, and choosing a suitable container compute layer. We cover how to choose between Windows and Linux containers using AWS App2Container to containerize legacy .NET Framework applications. The video ends with a demonstration of AWS App2Container, AWS Lambda, Amazon Elastic Container Service, and AWS Fargate with Amazon API Gateway to provide a seamless extraction of three sample microservices from a monolithic application.

You'll learn:

- How to use AWS .NET modernization tools
- About the typical .NET modernization scenarios for the tools
- How to run .NET microservices in containers and serverless infrastructures

Length: 54 minutes

Speaker
Matt Cline
Solutions Architect
Amazon Web Services

Additional Resource: Modernizing legacy .NET applications: DraftKings' principles for success

4. Migrate and Modernize Database Workloads

In this video you learn the difference between migrating databases vs modernizing them. We cover the effort around modernizing databases involving decoupling microservice dependent data structures and data from the existing database(s). We evaluate shared database vs splitting the database pattern with the pros and cons of the decision.

You'll learn:

- What are the fundamentals of migrating versus modernizing database workloads.
- What is the recommended database pattern for microservices architectures covering shared databases vs splitting the database
- How to leverage the SAGA pattern to manage data consistency across microservices in distributed transaction scenarios

Length: 29 minutes

Speaker
Stefan Minhas
Developer Advocate
Amazon Web Services

Additional resource: Refactoring Databases: Evolutionary Database Design

5. Landing in a Data lake

In this video we demonstrate how to setup a data lake to help democratize data across the organization. The democratization of data enables data curation, transformation, and migration to purpose-built databases. We demonstrate how to leverage batch and near-real time mechanisms using AWS Database Migration Service, Amazon Kinesis, and AWS Lake Formation.

You'll learn:

- How to create a data landing zone for your data lake
- How to enable product teams to curate diverse data models
- How to enable advanced use cases against structure and unstructured data

Length: 8 minutes

Speaker
John Standish
Solutions Architect
Amazon Web Services

Additional resource: What is a Data Lake?

6. Continuous Integration and Continuous Delivery for Containers and Serverless

In this video you learn how to build DevOps tailored for microservices. We cover how to use a Blue/Green Deployment Methodology to automatically deploy new containers to development leveraging automation in QA and production environments.

You'll learn:

- How to automatically build new container images at check in
- How to use the blue/green deployment methodology for microservice DevOps pipeline
- How to automatically deploy new containers to your development environment

Length: 1hr 12minutes

Speaker
Tom Moore
Microsoft Specialist Solutions Architect
Amazon Web Services

Additional resource: Automated Continuous Integration, Continuous Delivery (CI/CD) pipeline for .NET Core Lambda functions using AWS extensions for dotnet CLI

https://tinyurl.com/awsmodernization

Windows Modernization Offerings

1 **Application Modernization Lab:**

The Application Modernization Lab is an invitation only no-cost program that provides AWS experts and consulting time to upskill the in-house development teams of Windows customers to self-modernize their Windows hosted applications.

2 **MAP for Windows modernization:**

MAP for Windows modernization offers incentives and guidance to help customers modernize from Windows license-based services to AWS cloud-native services. The pilot utilizes the an EBA-based methodology to kick-start modernization efforts through AWS SA and ProServe resources.

3 **re:Think for Windows modernization:**

re:Think for Windows modernization is designed to kick-start the customer's modernization journey, giving them the guidance they need to identify a workload to transform and execute a proof-of-concept (POC).

4 **Windows Modernization Program (Partner-led):**

The Windows Modernization Program (WMP) is a partner funding offer. Its purpose is to help partners, by offsetting their fees, identify and complete Windows modernization projects.

The Modernization Journey

The Modernization Journey

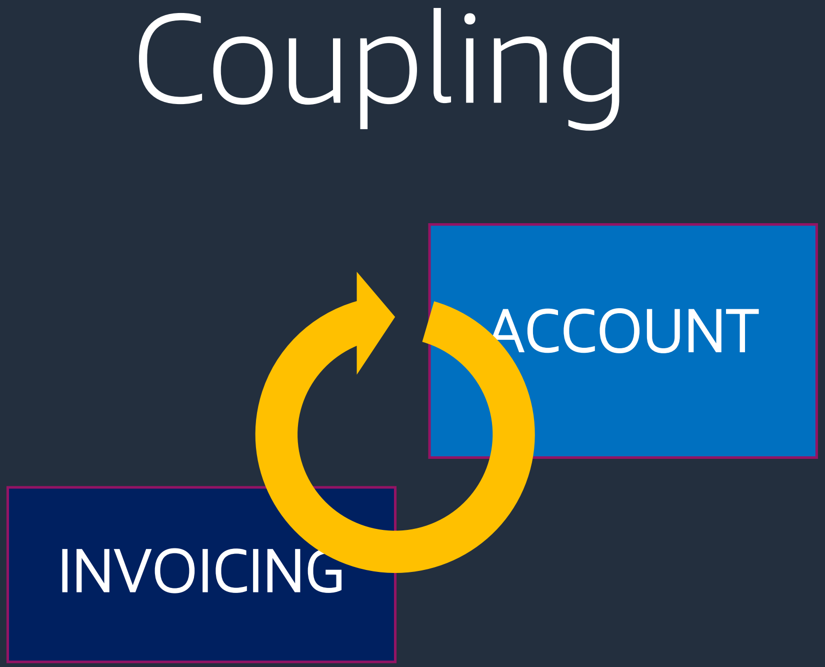
Monolithic	Microservices
Simple deployments	Partial deployments
Binary failure modes	Graceful degradation
Inter-module refactoring	Strong module boundaries
Technology monoculture	Technology diversity
Restricted scaling	Independent scalability

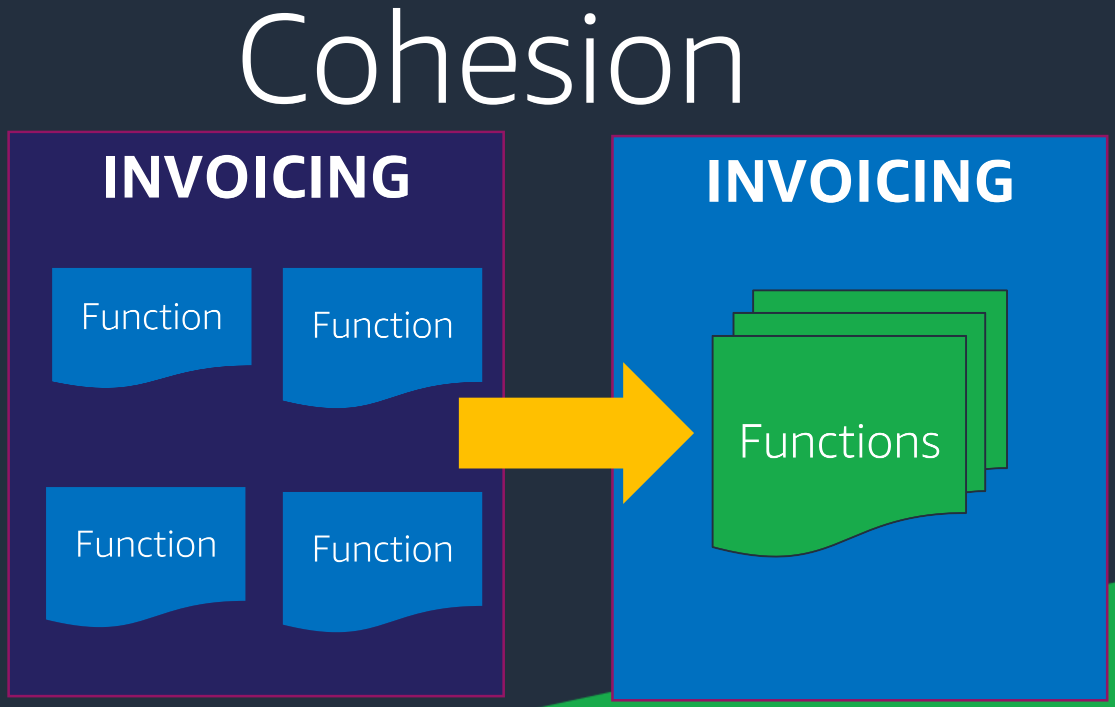
Independent Deployability is Key

- Independent deployability is the idea that we can make a change to a microservice and deploy it into a production environment without having to utilize any other services.
- Get into the habit of releasing the changes for a single microservice into production without having to deploy anything else.
- To guarantee independent deployability, we need to ensure our services are loosely coupled; in other words, we need to be able to change one service without changing anything else.
- The desire for loosely coupled services with stable interfaces guides our thinking about how we find service boundaries in the first place.

Addressing Coupling and Cohesion During Modernization

- **Coupling** refers to the interdependencies between modules, while cohesion describes how related the functions are within a single module.
- Low **cohesion** implies that a given module performs tasks which are not closely related to each other, and hence, can create problems as the module becomes larger.





The Modernization Journey

Coupling

- Reduce coupling
- Tight coupling of unrelated services is bad
- Expensive to change code
- Multiple changes across unrelated services

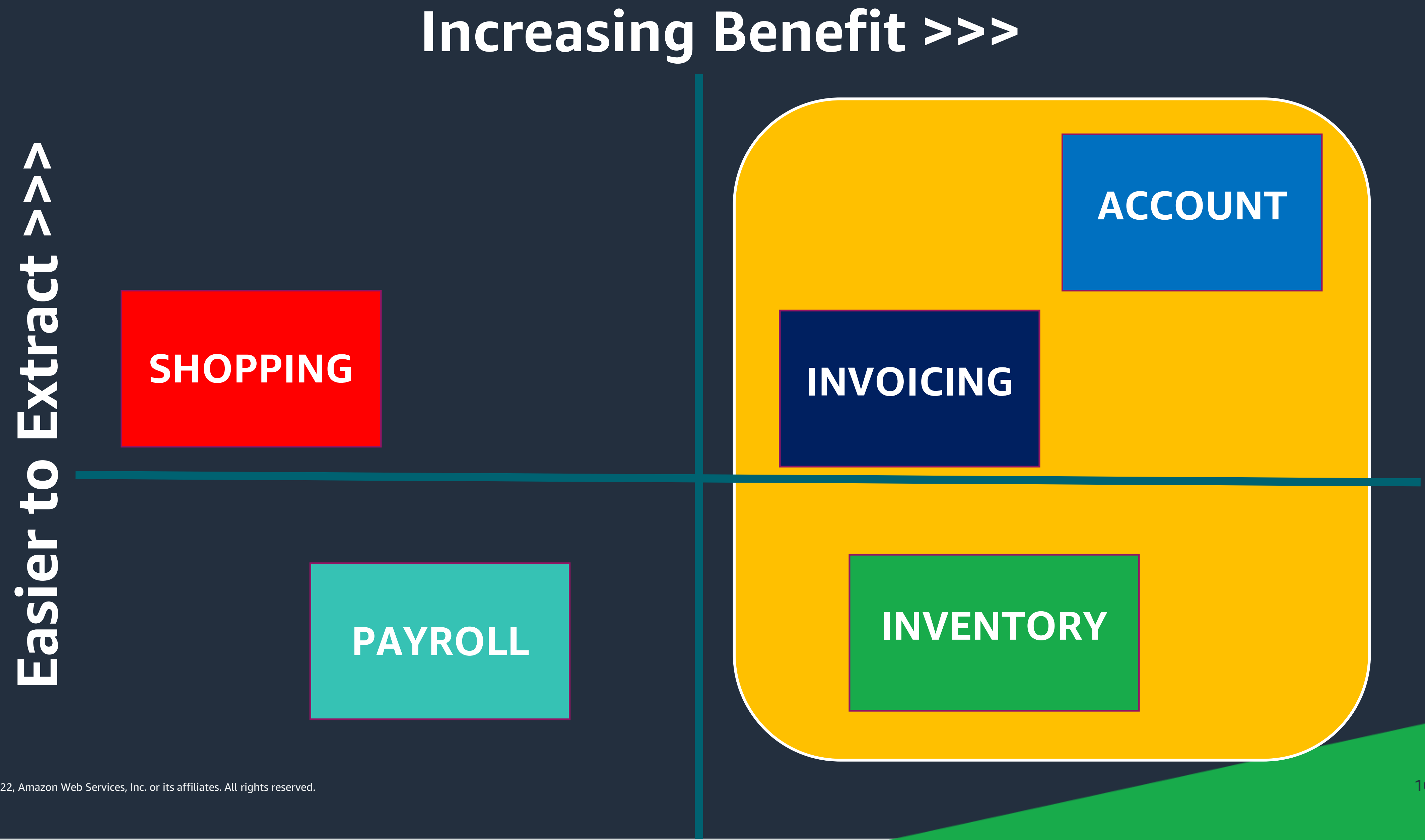
Cohesion

- Increase cohesion
- Code that changes together stays together
- How code is grouped
- Bring together tightly related code

“If you do a big bang rewrite, the only thing you’re certain of is a big bang.

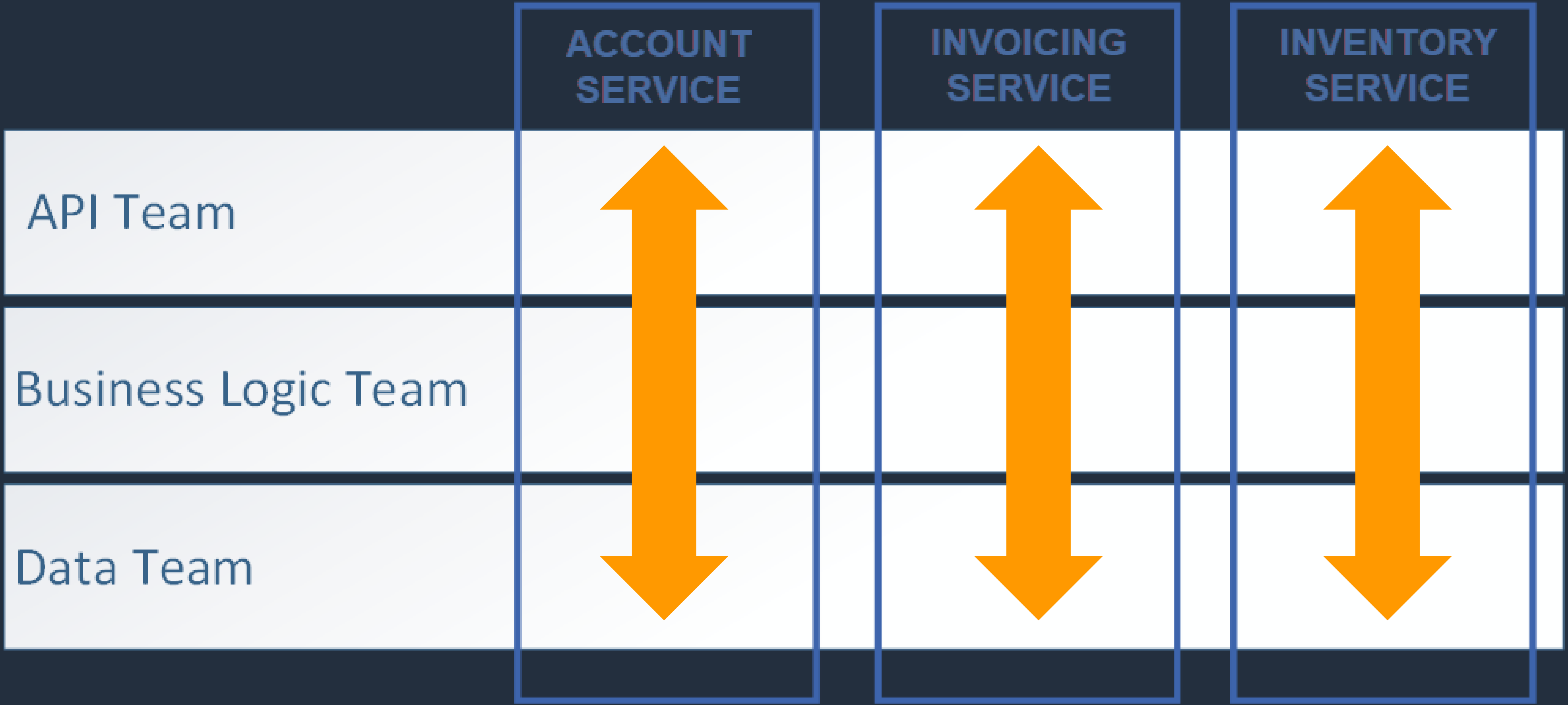
Martin Fowler (paraphrased)

Customers Need to Understand Where to Start Modernizing



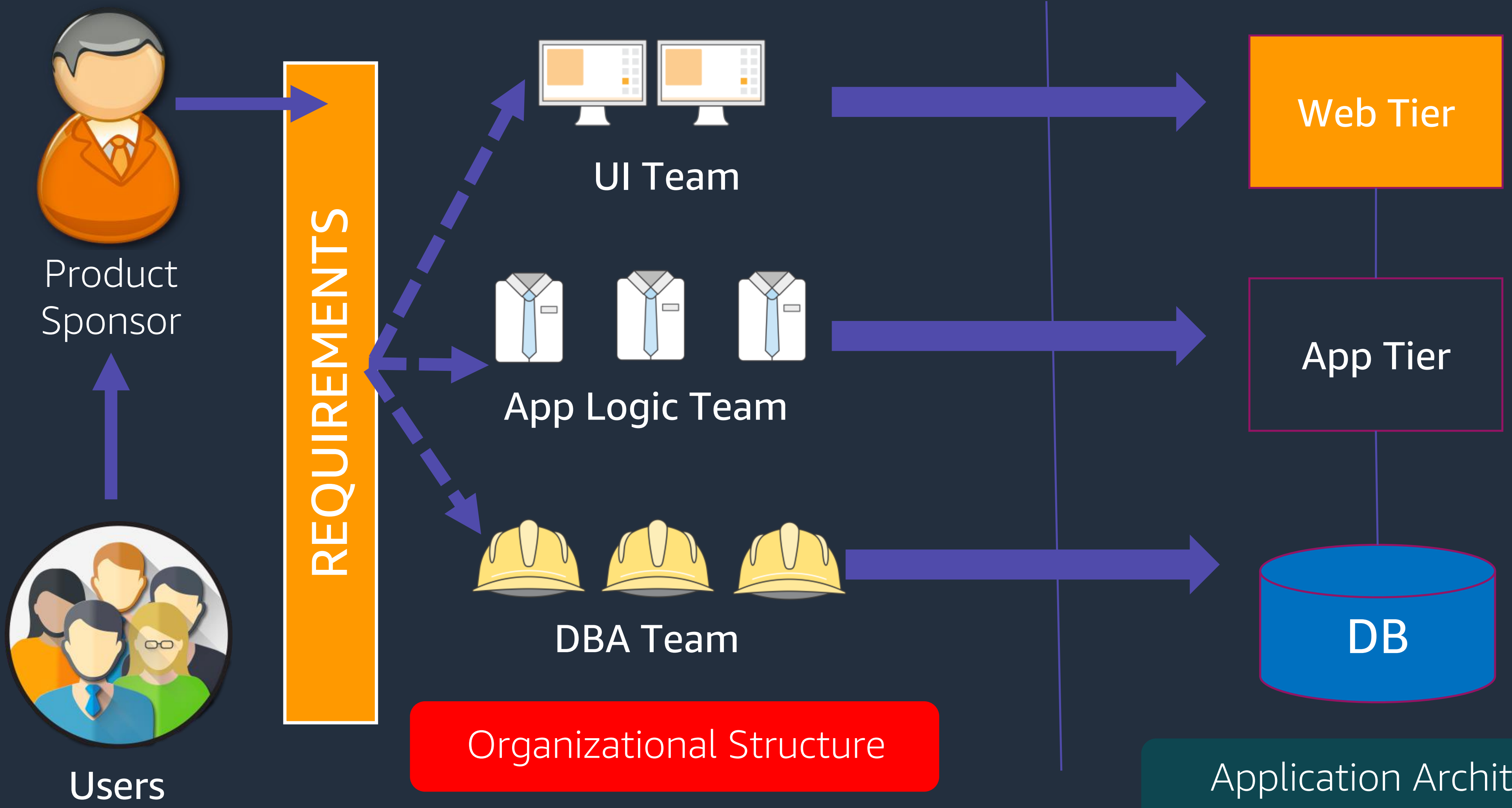
Organizational Change

People Process Culture Change – Poor Team Structure

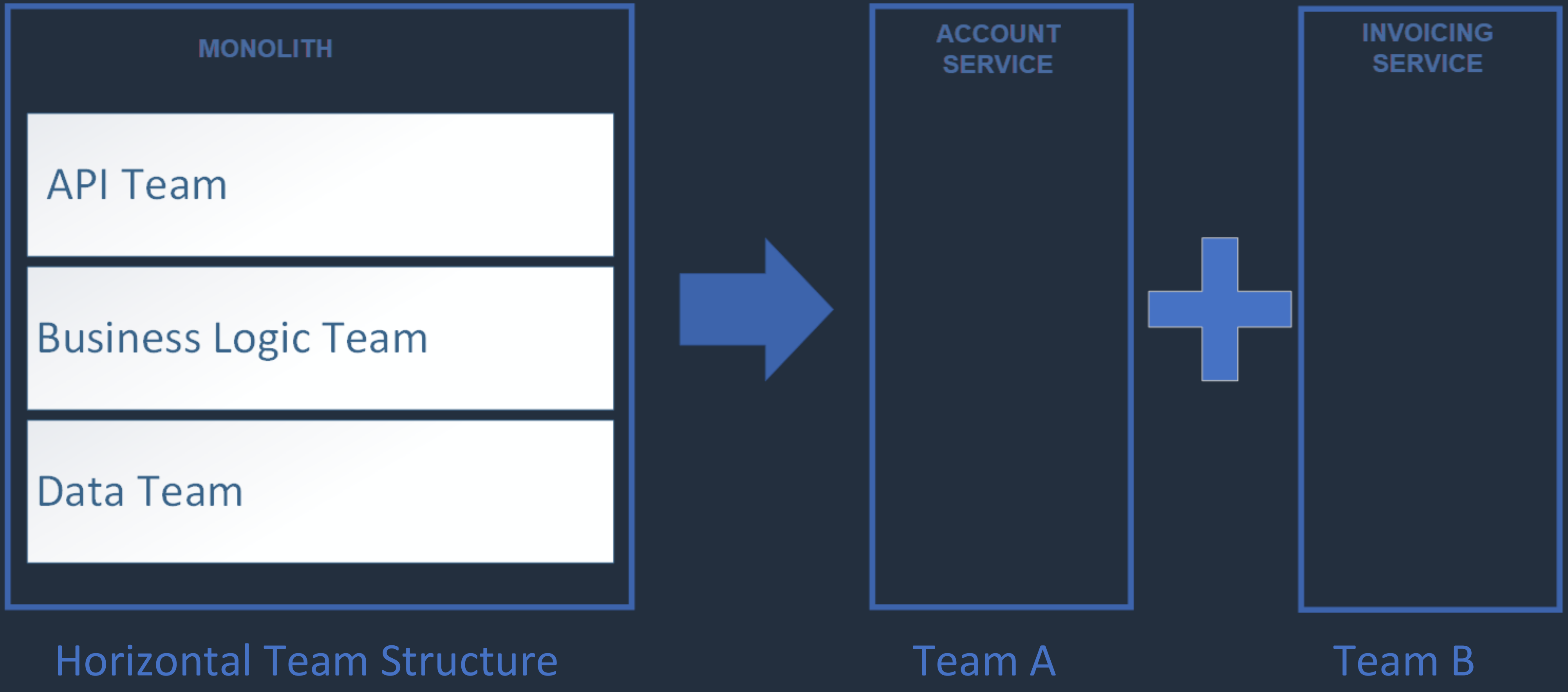


Monolithic Ownership

- Organized on technology capabilities



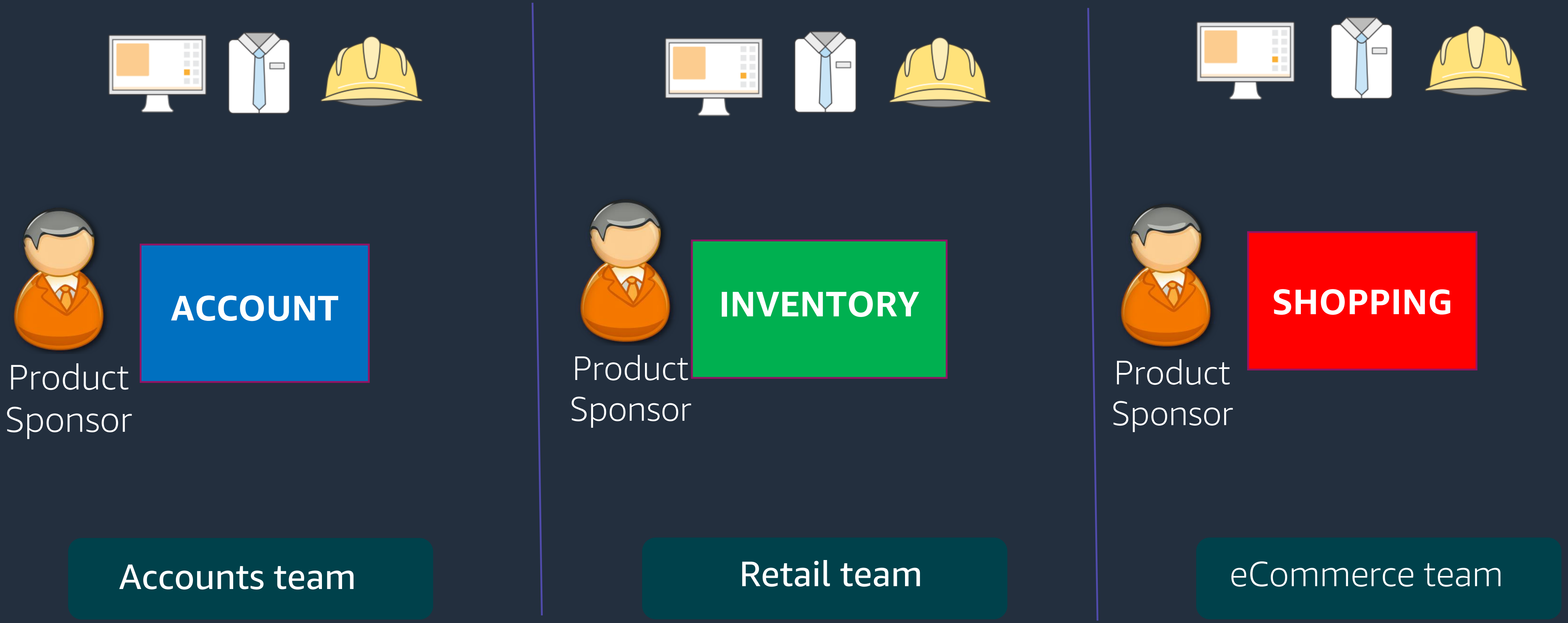
People Process Culture Change – Service Team Structure



Align Teams to Services

Microservice Architectures Ownership

Organized on business responsibilities



Conway's Law applies to modular software systems and states that:

"Any organization that designs a system [defined more broadly here than just information systems] will inevitably produce a design whose structure is a copy of the organization's communication structure".

People Process Culture Change

Modernization Failure Modes

Common Modernization Failure Modes

Re-creating old problems in newer ways	• Re-creating smaller monoliths • Adding to troubleshooting nightmares • Incrementally bloating operational burden

Common Modernization Failure Modes

Re-creating old problems in newer ways	• Re-creating smaller monoliths • Adding to troubleshooting nightmares • Incrementally bloating operational burden
Misaligned organizational and cultural practices	• Large teams without clear ownership • Big bang releases • Siloed product teams, disconnected customer

Common Modernization Failure Modes

Re-creating old problems in newer ways	• Re-creating smaller monoliths • Adding to troubleshooting nightmares • Incrementally bloating operational burden
Misaligned organizational and cultural practices	• Large teams without clear ownership • Big bang releases • Siloed product teams, disconnected customer
One vision for the future	• Siloed innovation teams • Innovation blockers with misaligned business values

Common Modernization Failure Modes

Re-creating old problems in newer ways	• Re-creating smaller monoliths • Adding to troubleshooting nightmares • Incrementally bloating operational burden
Misaligned organizational and cultural practices	• Large teams without clear ownership • Big bang releases • Siloed product teams, disconnected customer
One vision for the future	• Siloed innovation teams • Innovation blockers with misaligned business values
Transformation fatigue and unclear business value	• Piped vision with business value at the tail end • Unclear and delayed business value realization • Multi-year wait • Biting off more than can be chewed

Modernization Strategies

When should you Re-Factor?

- Too much effort to re-use legacy code and port to microservice
- Original product was way off the mark, didn’t achieve goals (e.g. no user adoption)
- Original product does not have traction
- Significant deviation from original intent of product, going after a new market
- Technology holding you back (Mainframe, Visual Basic overly-customized SFDC or AEM)
- You can redefine the business process around the new system

Breaking the Monolith

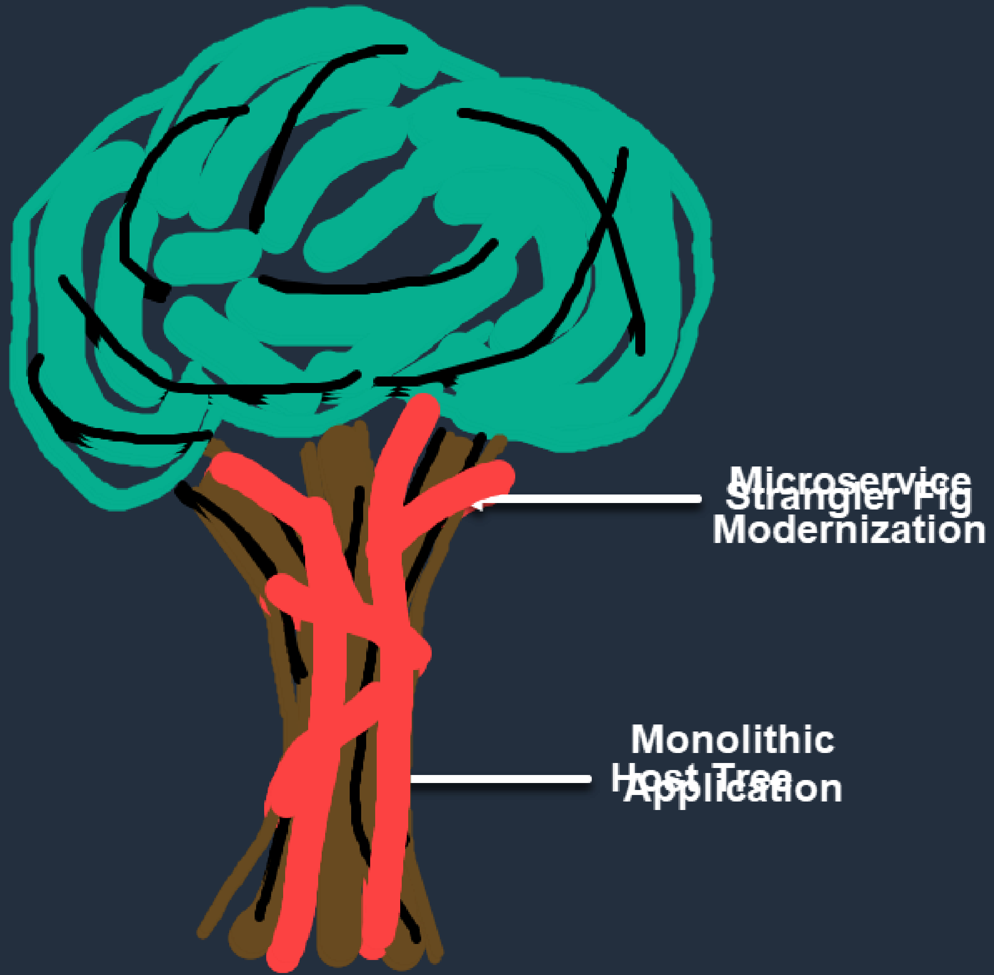
The Strangler Fig

Breaking the Monolith

- You have a tremendous amount of resources invested in existing monolithic applications
- You are looking for a sane way to capture the benefits of microservices without having to completely rewrite
- AWS provides a path forward for microservice-based apps in the future
- AWS provides familiarity and a microservices proving ground

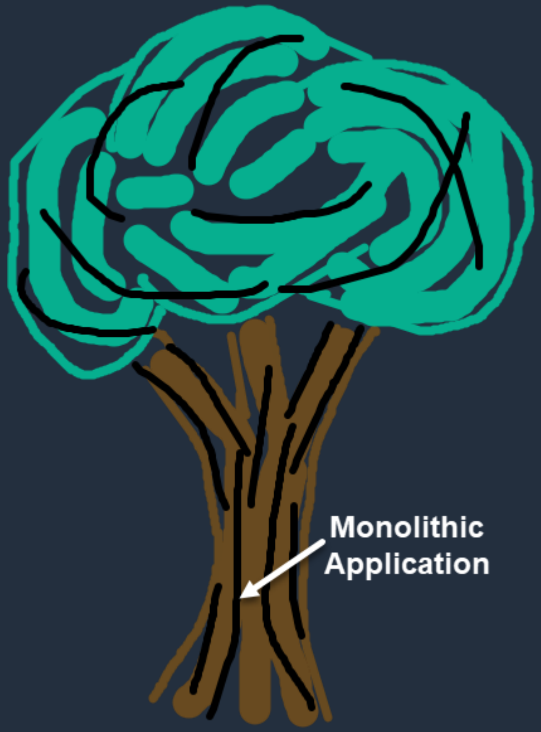
Breaking the Monolith Using the Strangler Fig Pattern

- The Strangler Fig is a form of vine that comes from the natural of growth around the host tree.
- Gradually, the roots wrap around the host tree, widen, and slowly form a latticework that surrounds the host's trunk.
- We can think of the vine as the modernization effort and the host tree as the legacy monolith.
- Eventually, the host tree is replaced, leaving the fig with a hollow trunk



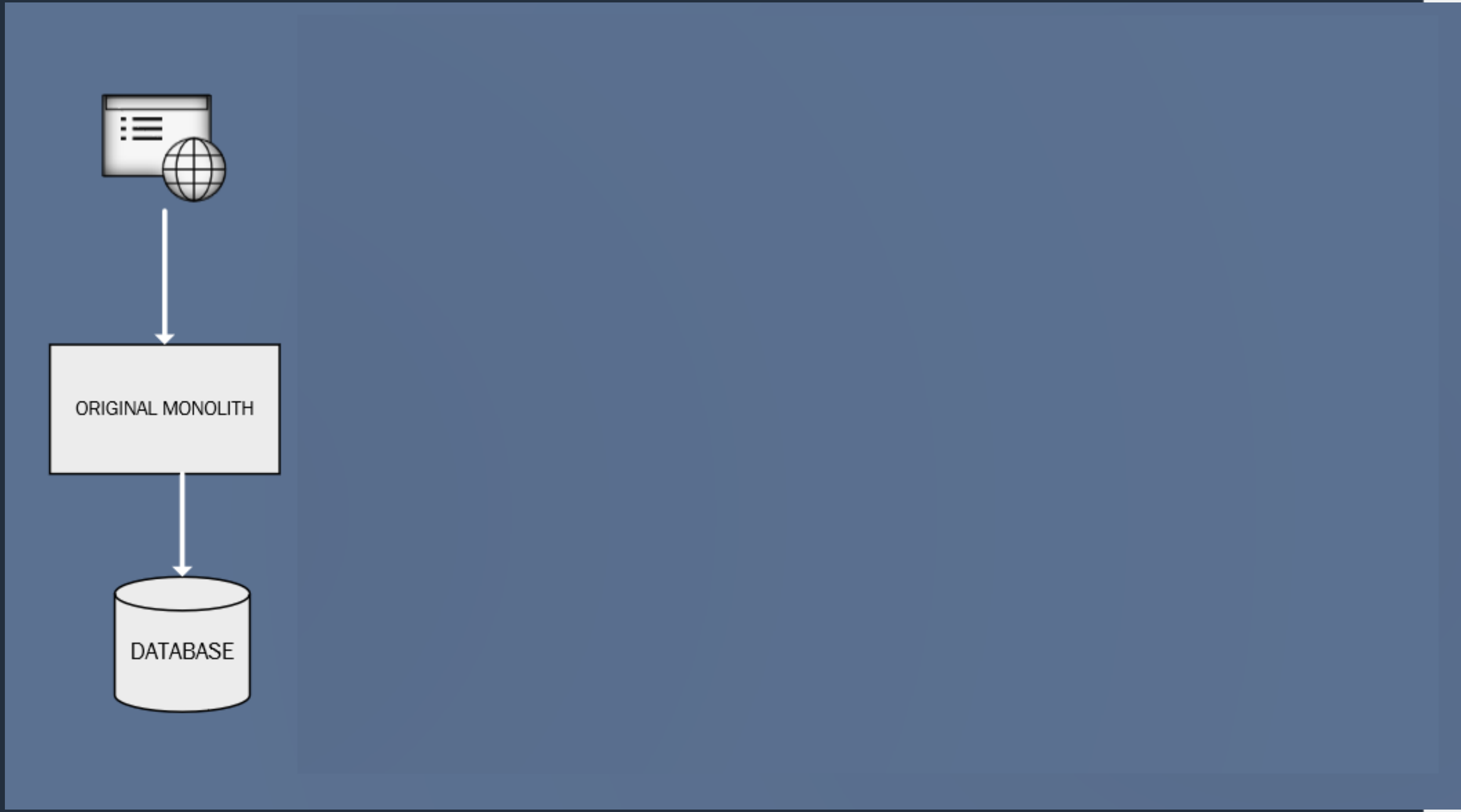
Strangling The Monolith

- This modernization method is based on incrementally replacing existing functionalities of the old legacy system with new applications and services in a phased approach until the new application system eventually replaces all of the old, or key part of legacy system's features.



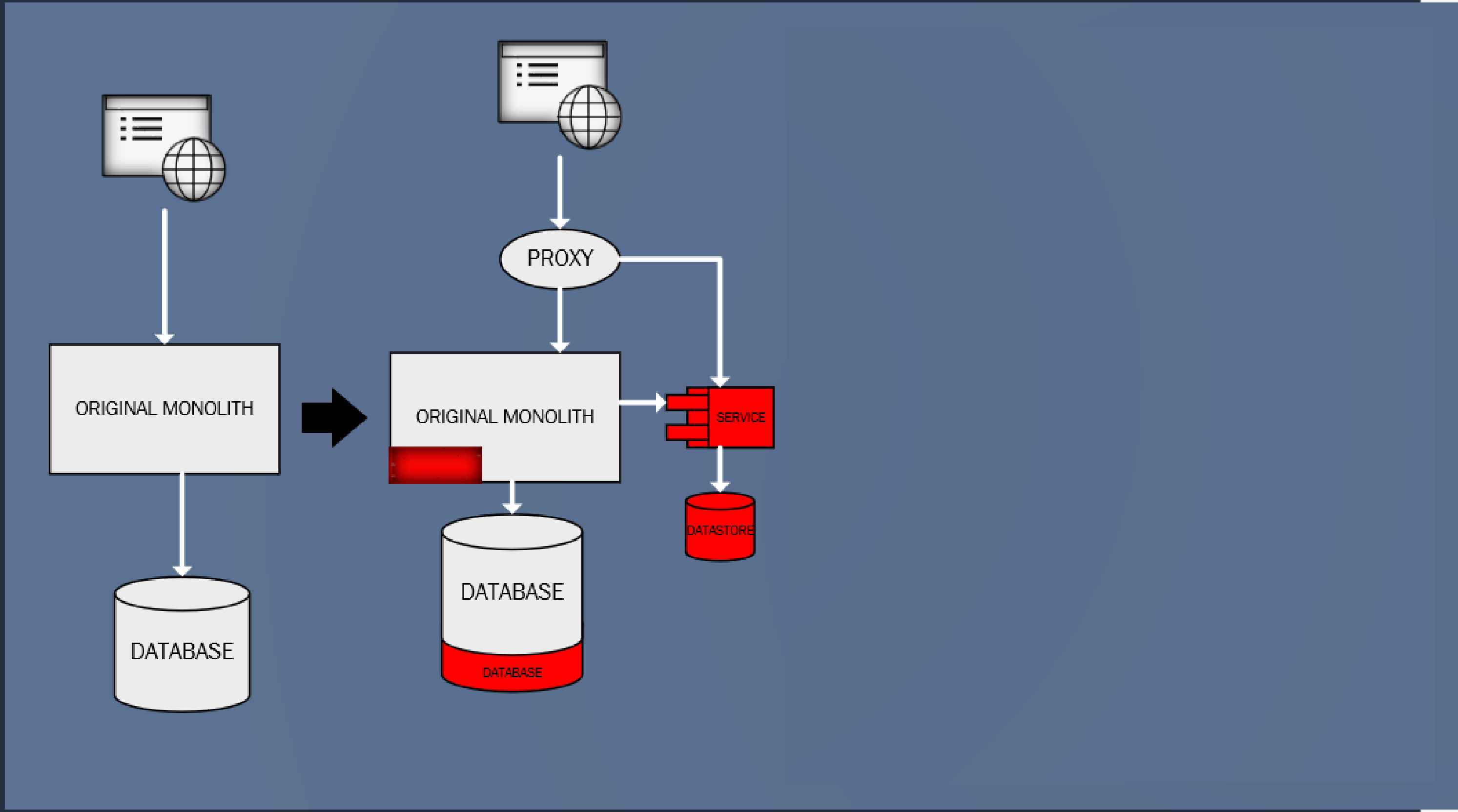
Strangling The Monolith

- Strangling – means incrementally replacing functionality in the app with something better (cheaper, faster, easier to maintain).



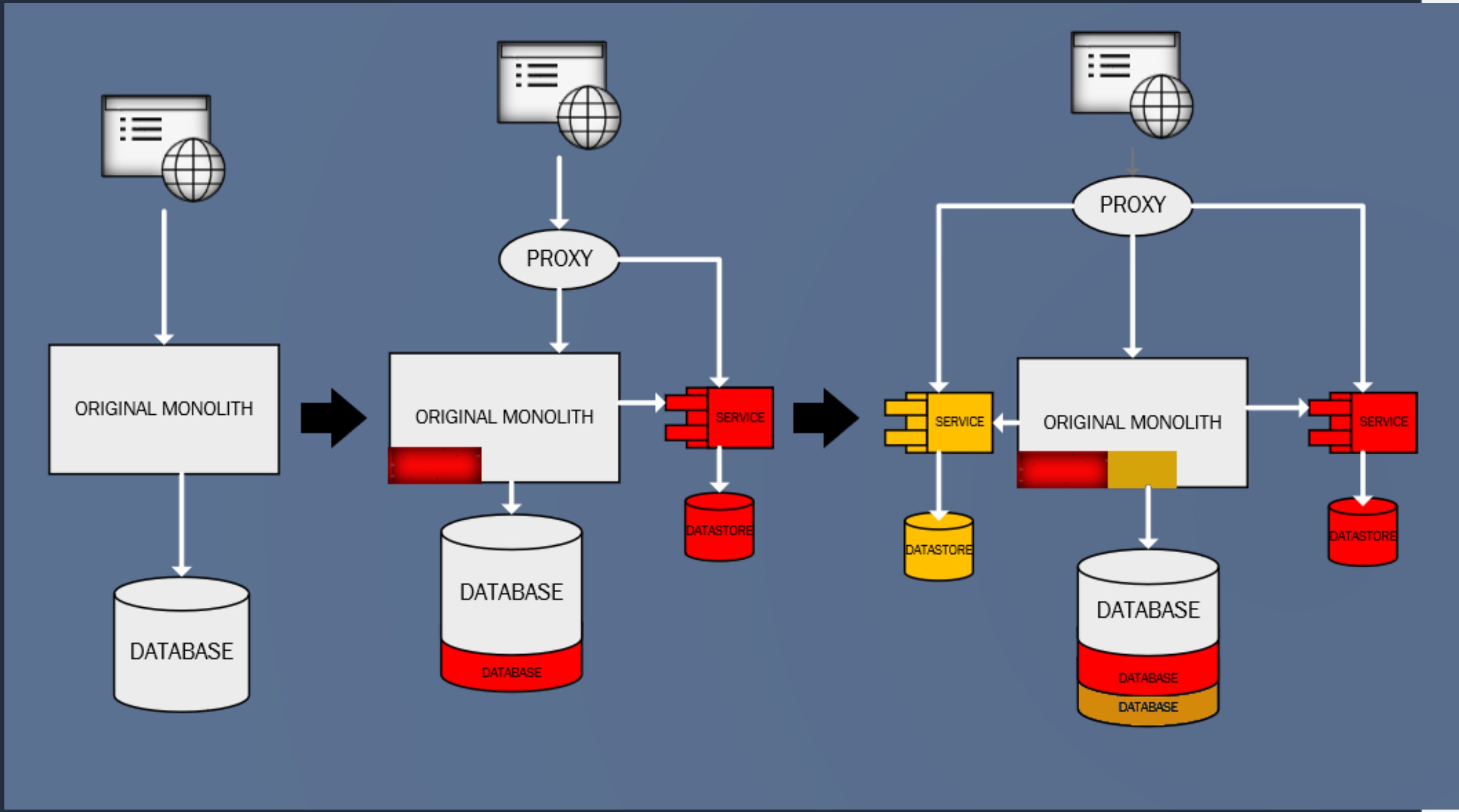
Strangling The Monolith

- Strangling – means incrementally replacing functionality in the app with something better (cheaper, faster, easier to maintain).
- As functionality is replaced, "dead" parts of monolith can be removed/retired.



Strangling The Monolith

- Strangling – means incrementally replacing functionality in the app with something better (cheaper, faster, easier to maintain).
- As functionality is replaced, "dead" parts of monolith can be removed/retired.
- Includes new functionality during strangulation to make it more attractive to business stakeholders.
- The proxy will redirect traffic to the new service as it goes to production.



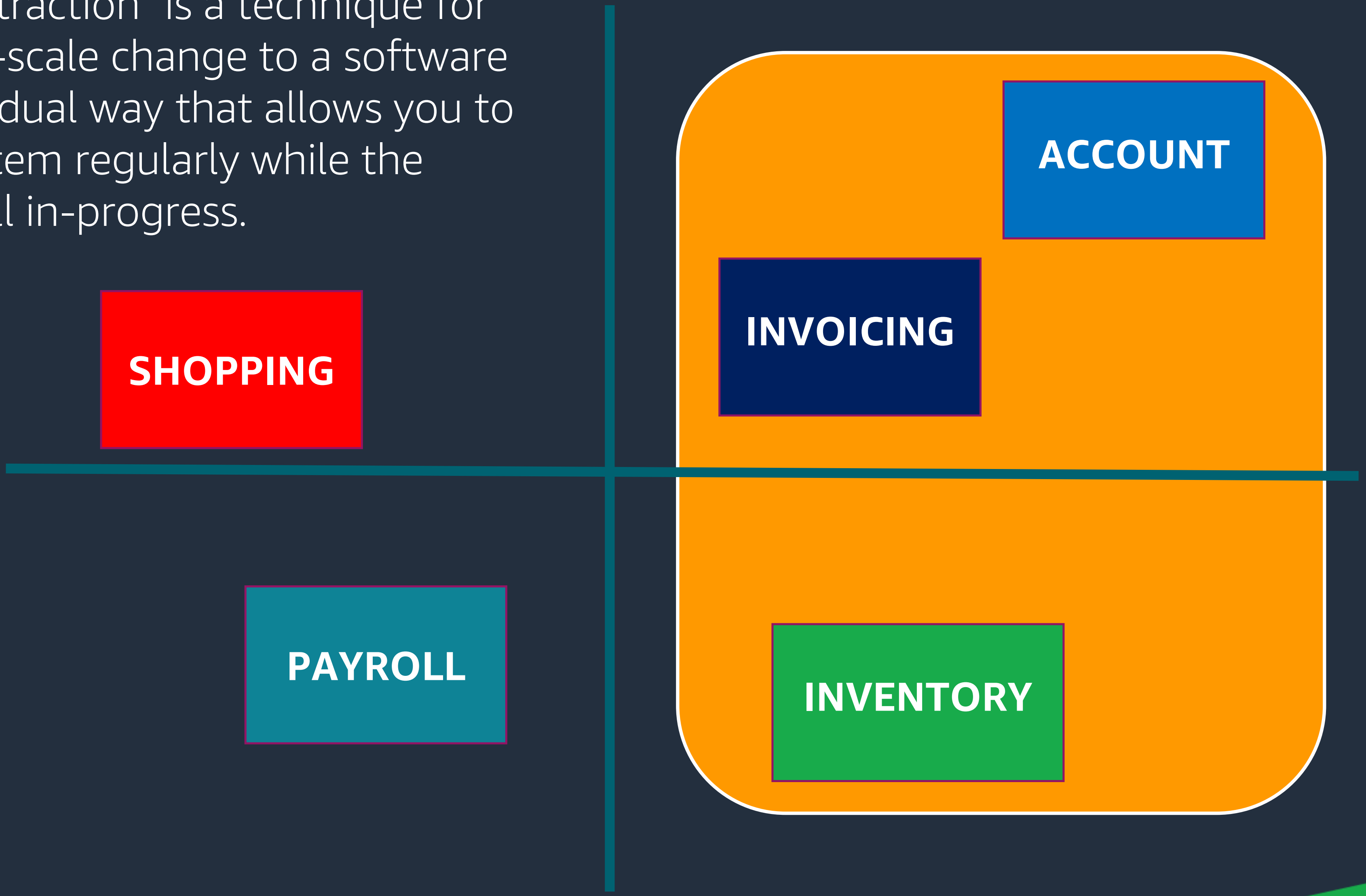
When to Use the Strangler Pattern

The following guidance will ensure that you don't have to redeploy the entire monolithic application regularly, and by breaking services out, it will allow you to independently scale and deploy the application more smoothly.

- Select a component that has good test coverage and less technical debt associated with it. Starting with this component can give teams a lot of confidence during the modernization process.
- Select components that have scalability requirements. Start with one of these components.
- Select a component that has frequent business requirement changes and more frequent deployments.

Branch By Abstraction

"Branch by Abstraction" is a technique for making a large-scale change to a software system in a gradual way that allows you to release the system regularly while the changes are still in-progress.



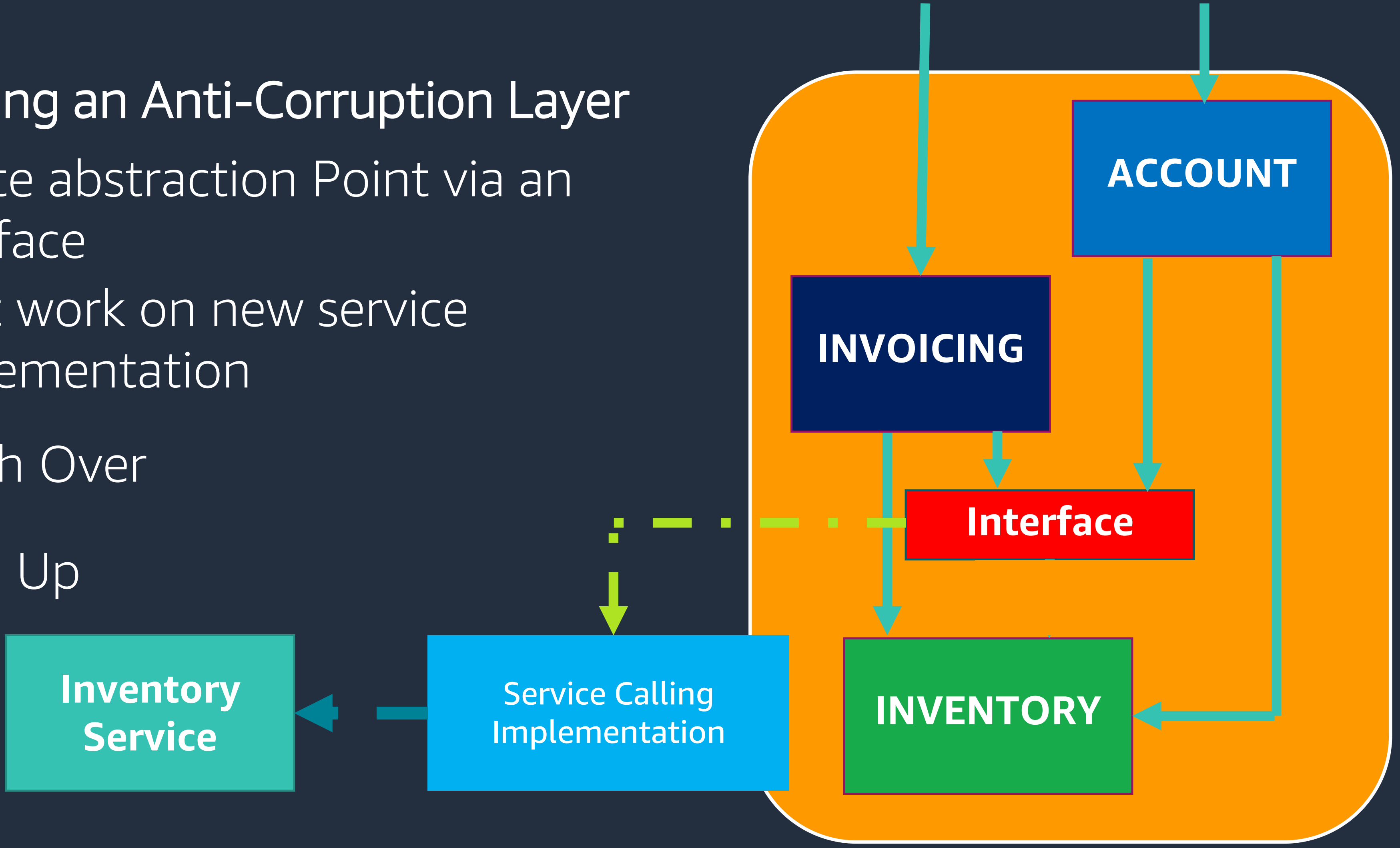
Branch By Abstraction

Introducing an Anti-Corruption Layer

1. Create abstraction Point via an Interface
2. Start work on new service implementation

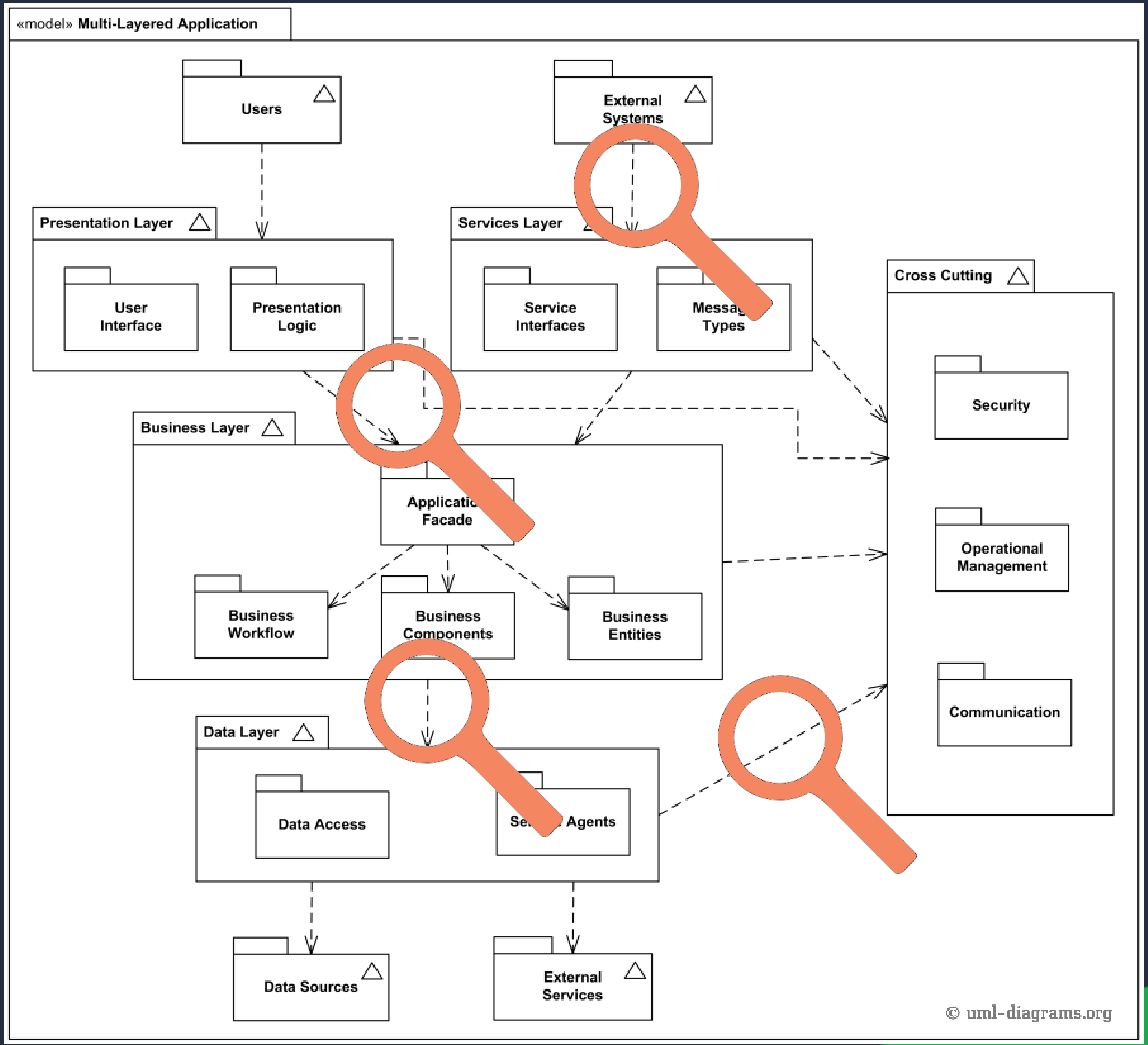
3. Switch Over

4. Clean Up



Where do I start? Follow the Data!

Follow the Data



Follow the Data - Distributed Tracing

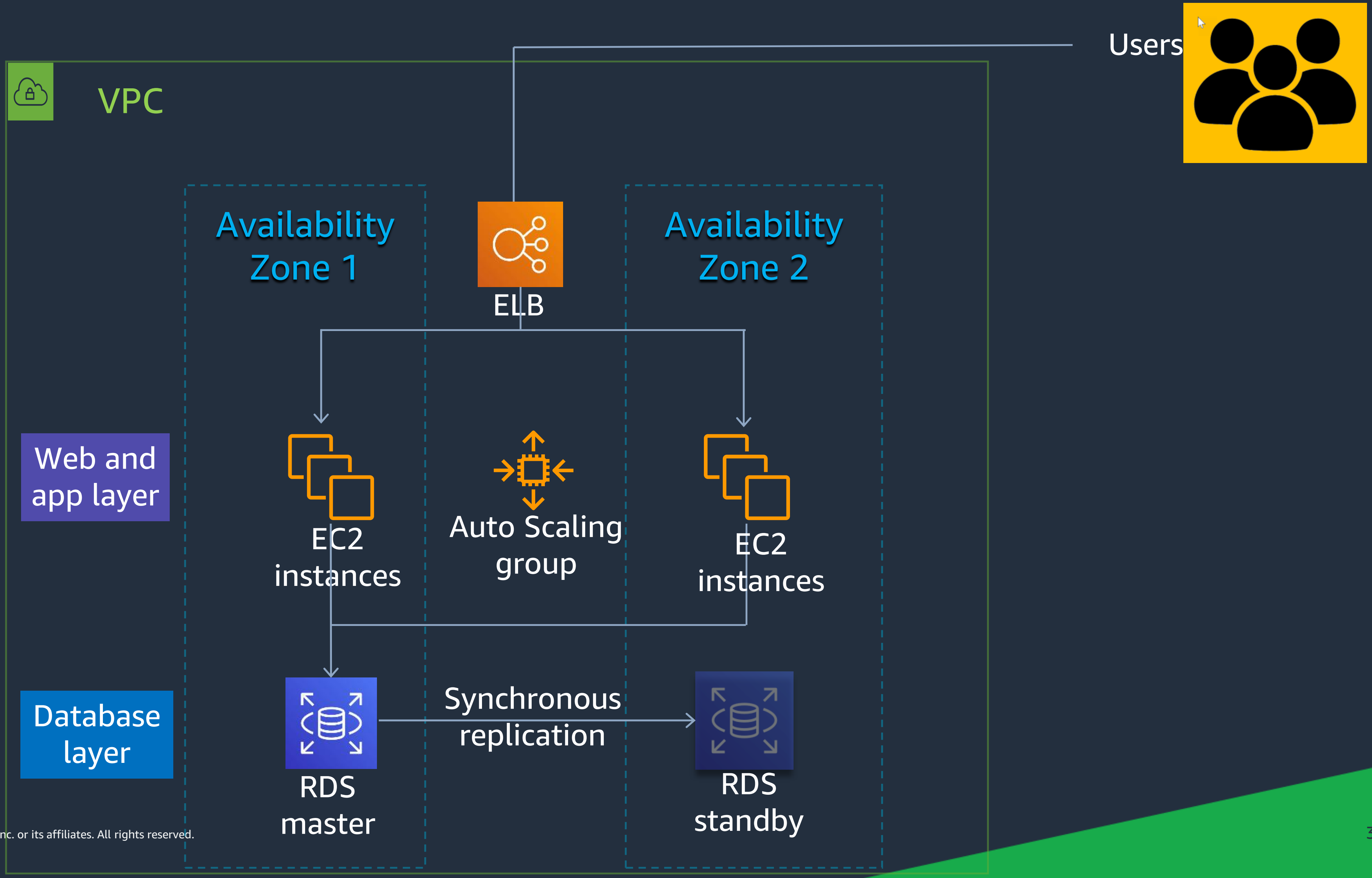
Modern applications and sites increasingly use many interconnected services. An application architecture that relies on many services or microservices is often referred to as a distributed system.

Distributed tracing is the process of tracking the activity resulting from a request to an application.

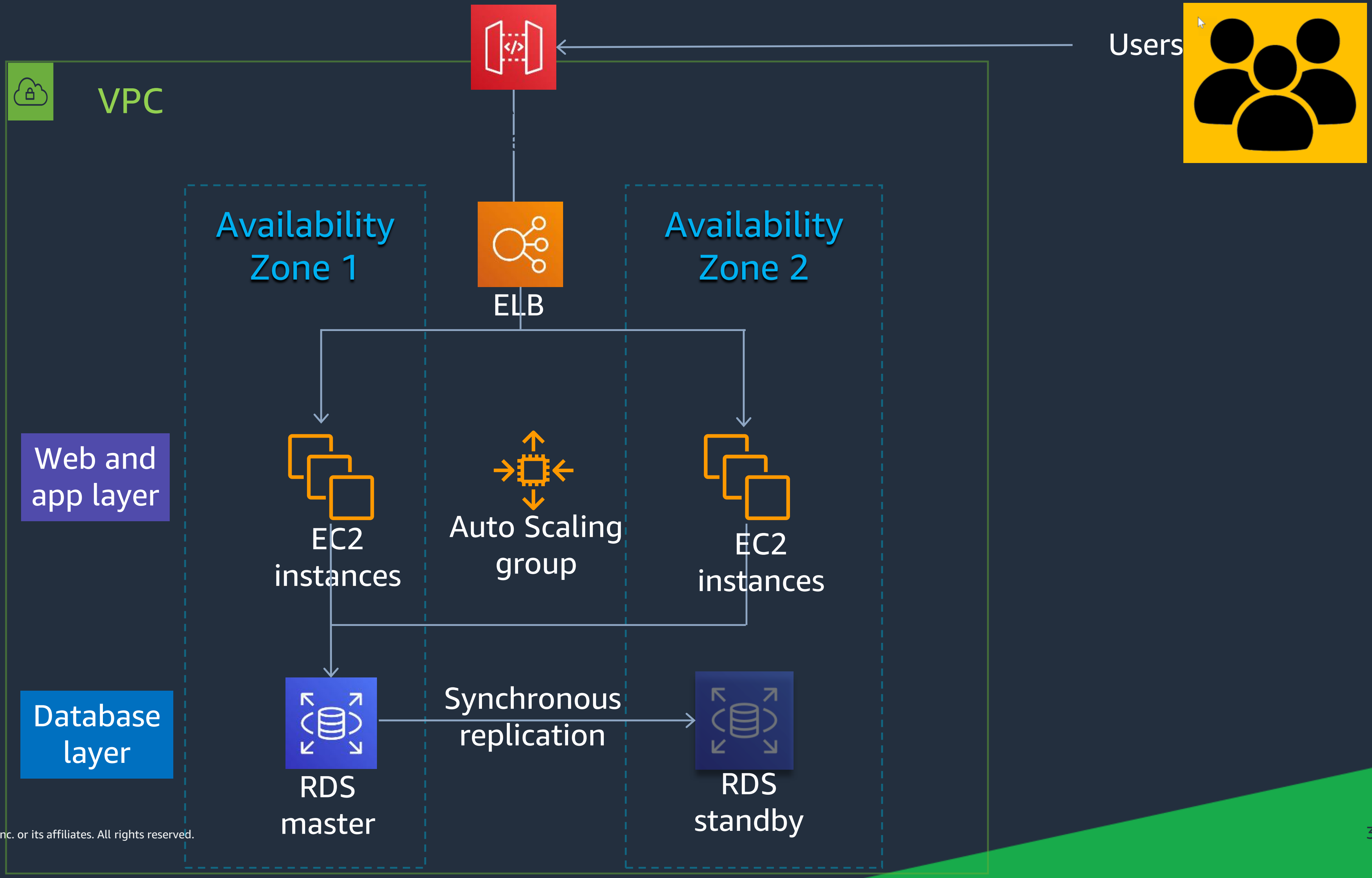
What you need to discover:

- Trace the path of a request as it travels across a complex system
- Discover the latency of the components along that path
- Know which component in the path is creating a bottleneck

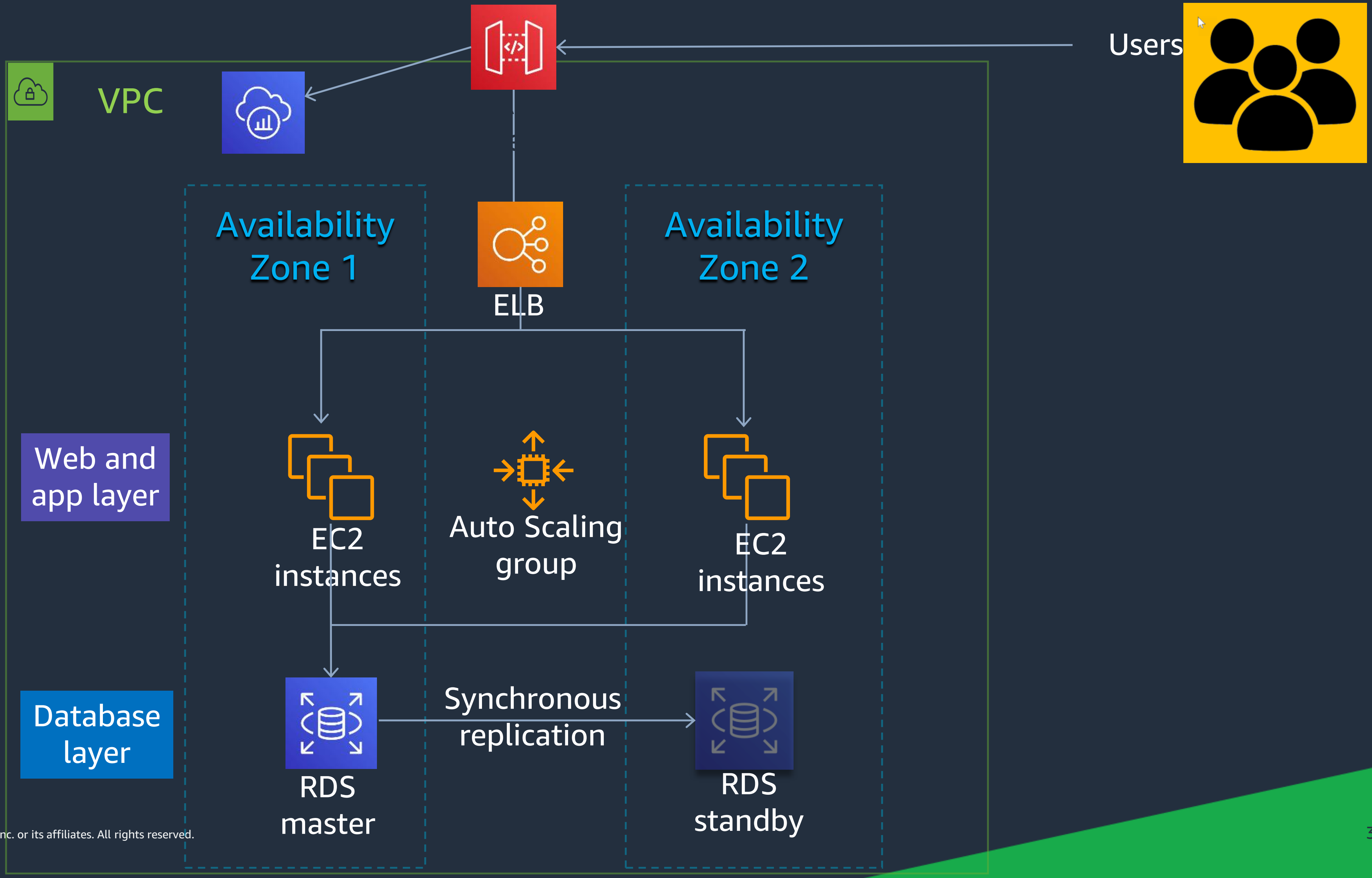
Strangling The Monolith – Follow the Data



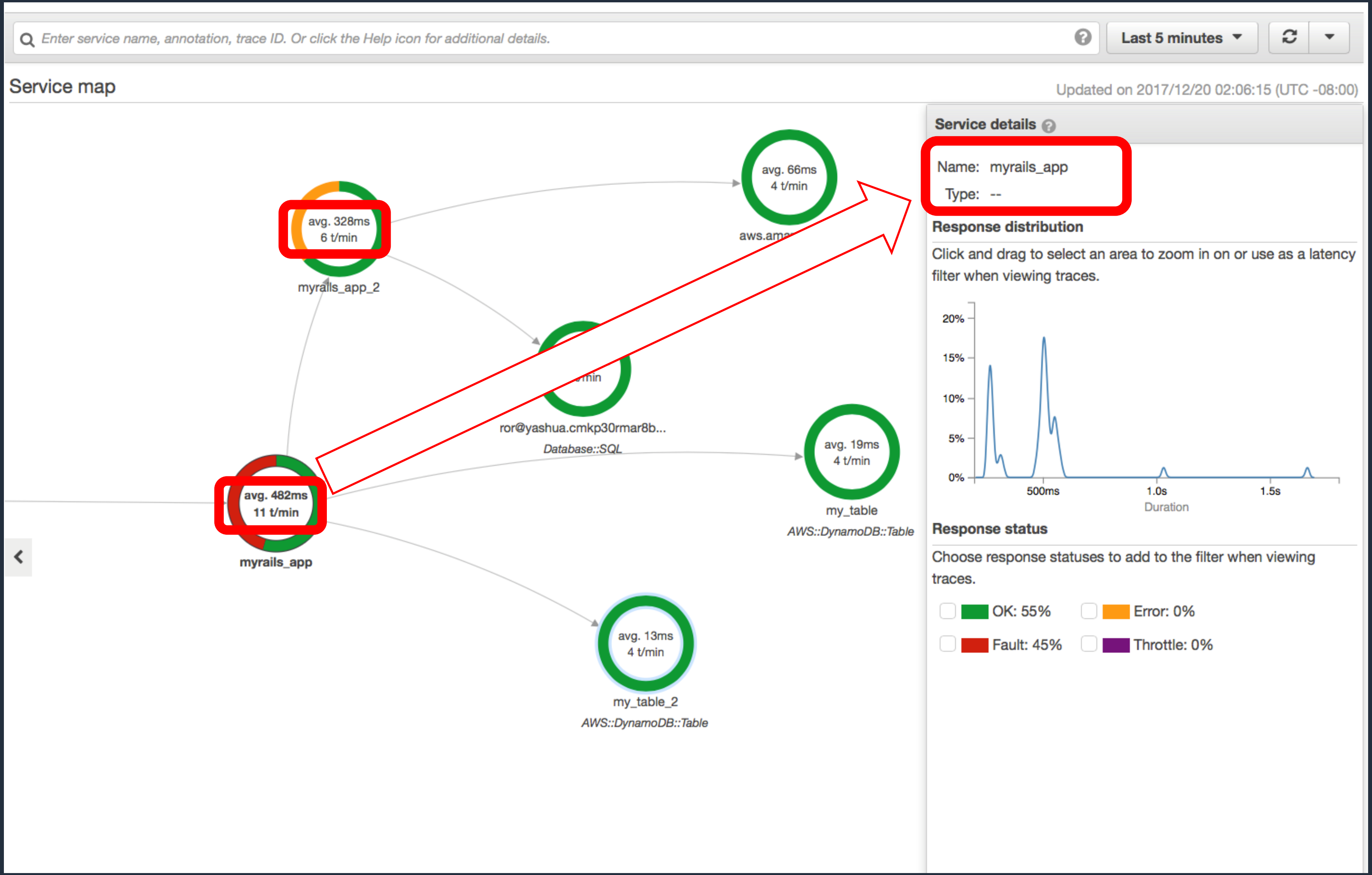
Strangling The Monolith – Facade with Amazon API Gateway



Strangling The Monolith – Follow the Data with AWS X-Ray



Strangling The Monolith – Follow the Data with AWS X-Ray



Service details

Name: myrails_app

Type: --

Response distribution

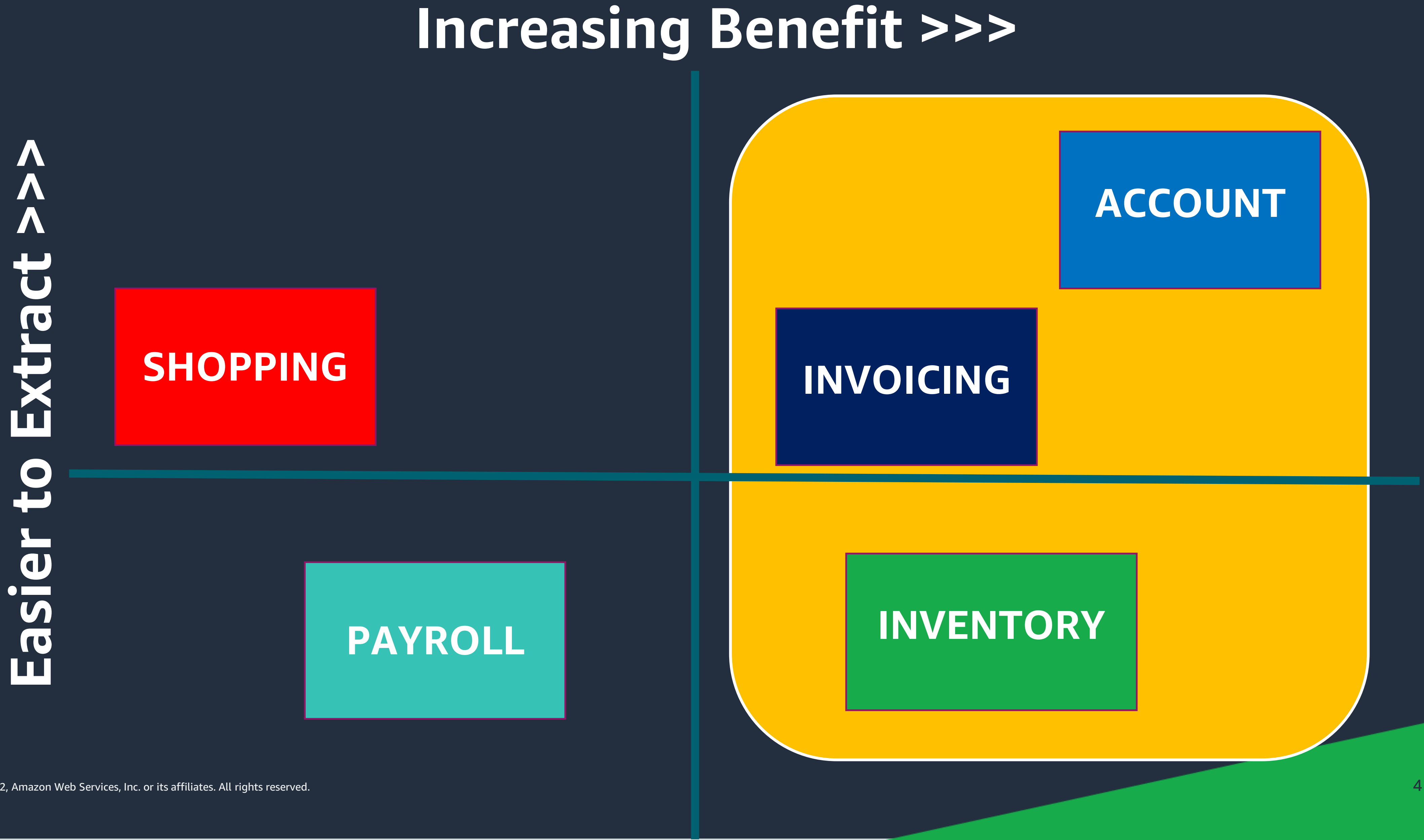
Click and drag to select an area to zoom in on or use as a latency filter when viewing traces.

Response status

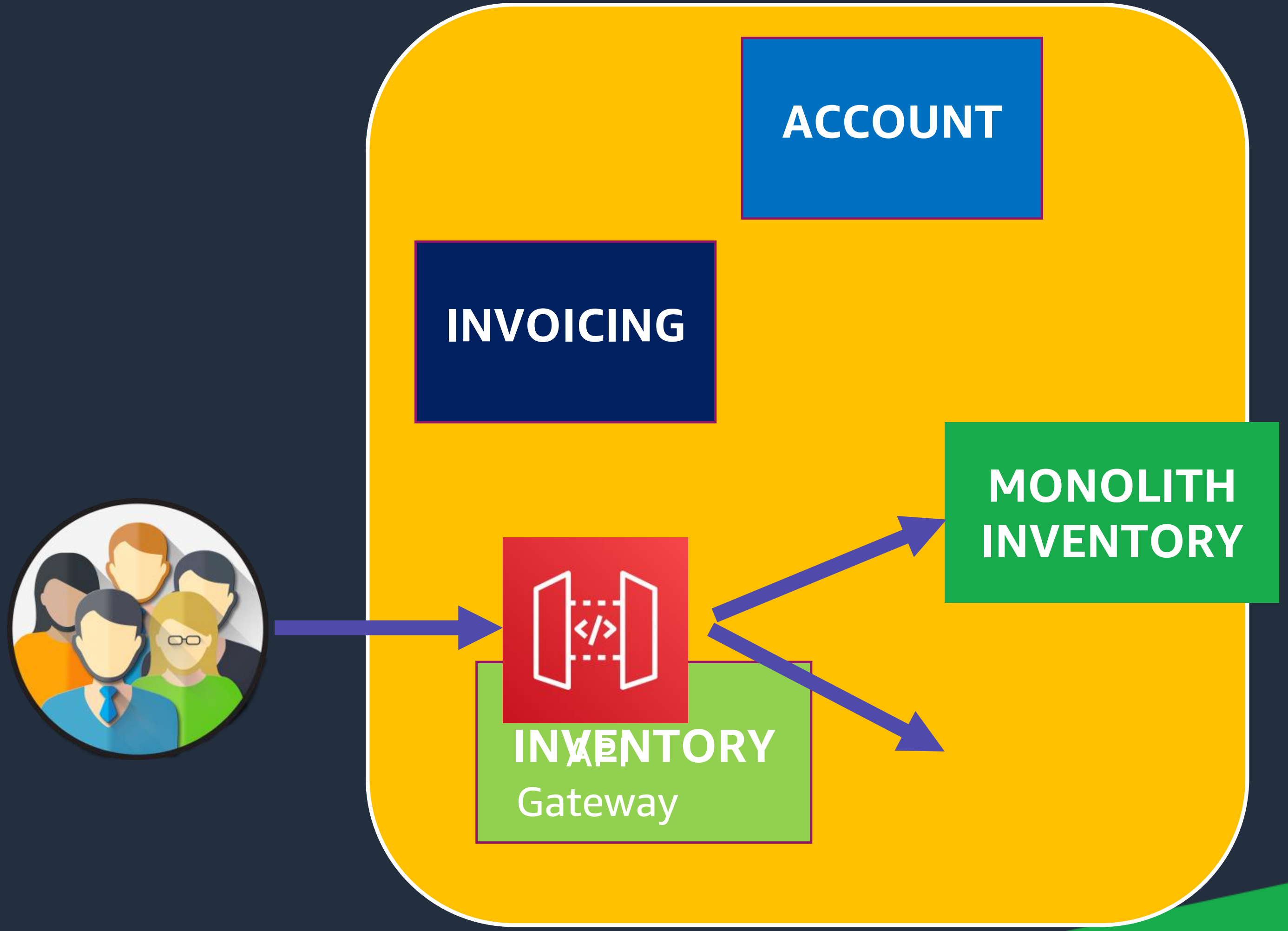
Choose response statuses to add to the filter when viewing traces.

🟩 OK: 55%	🟧 Error: 0%
🟥 Fault: 45%	🟪 Throttle: 0%

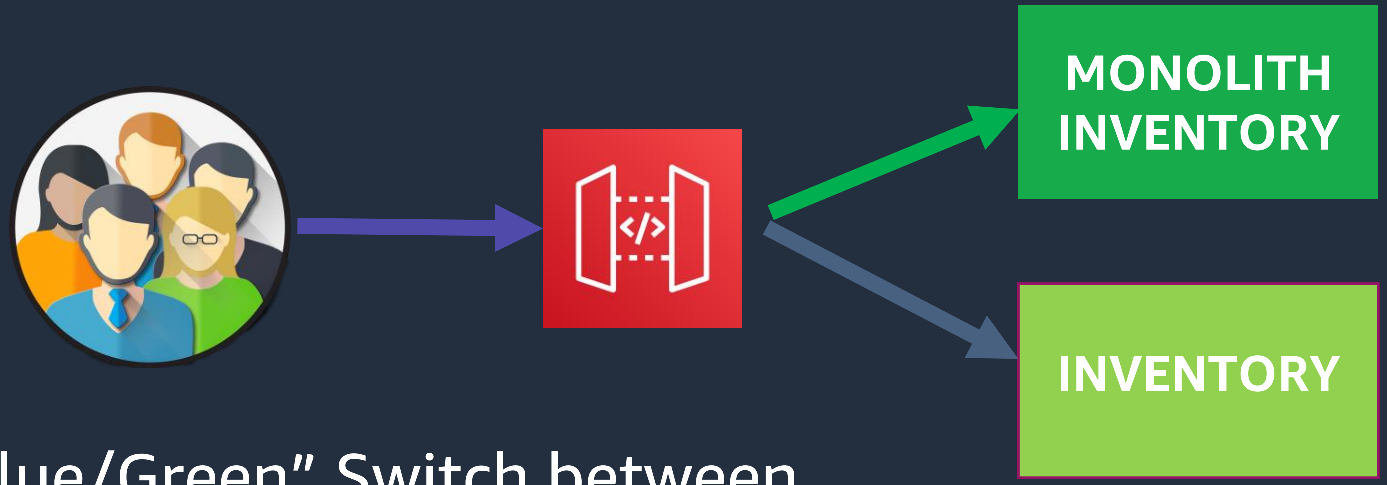
Strangling The Monolith – Hot Spot Services



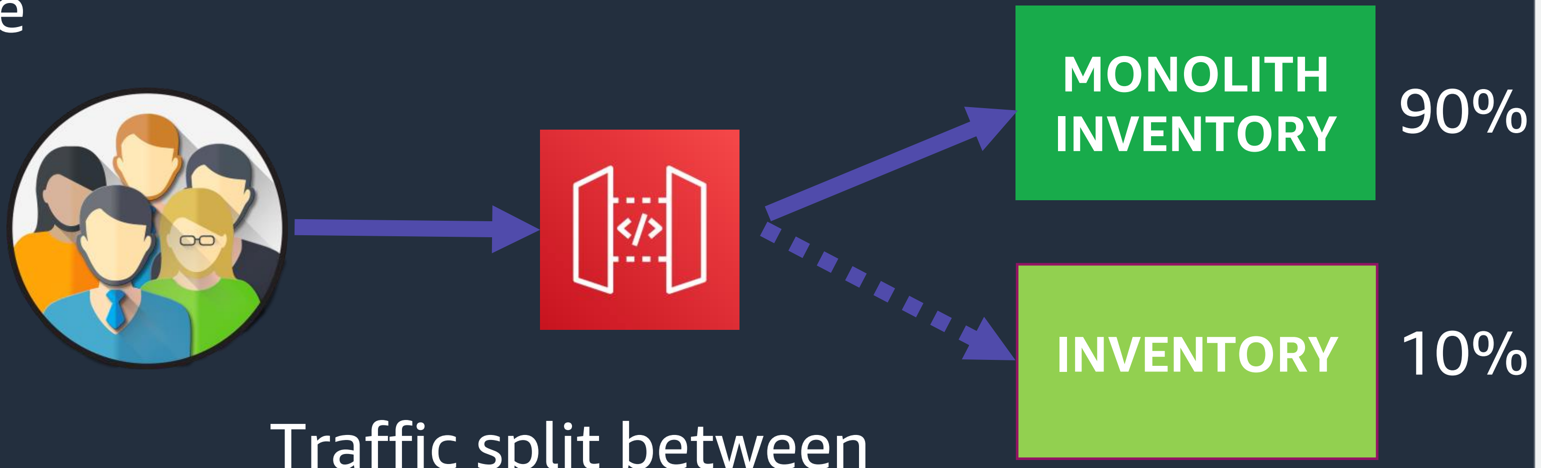
Deployment of New Strangled Service



Blue/Green or a Canary Release?



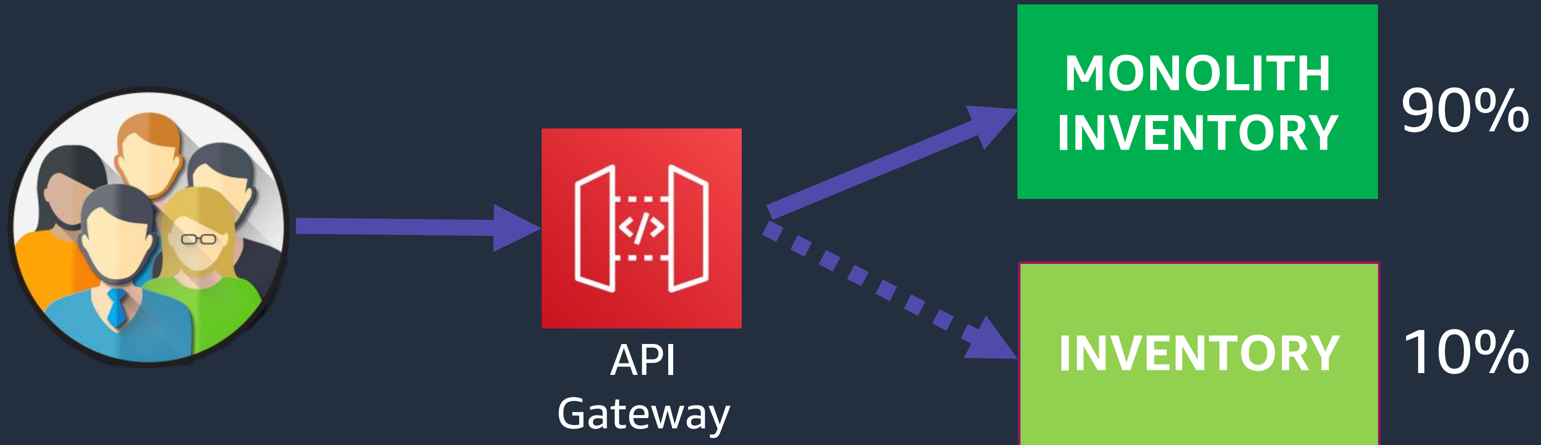
“Blue/Green” Switch between
Old and New Inventory Service



Traffic split between
Old and New Inventory Service

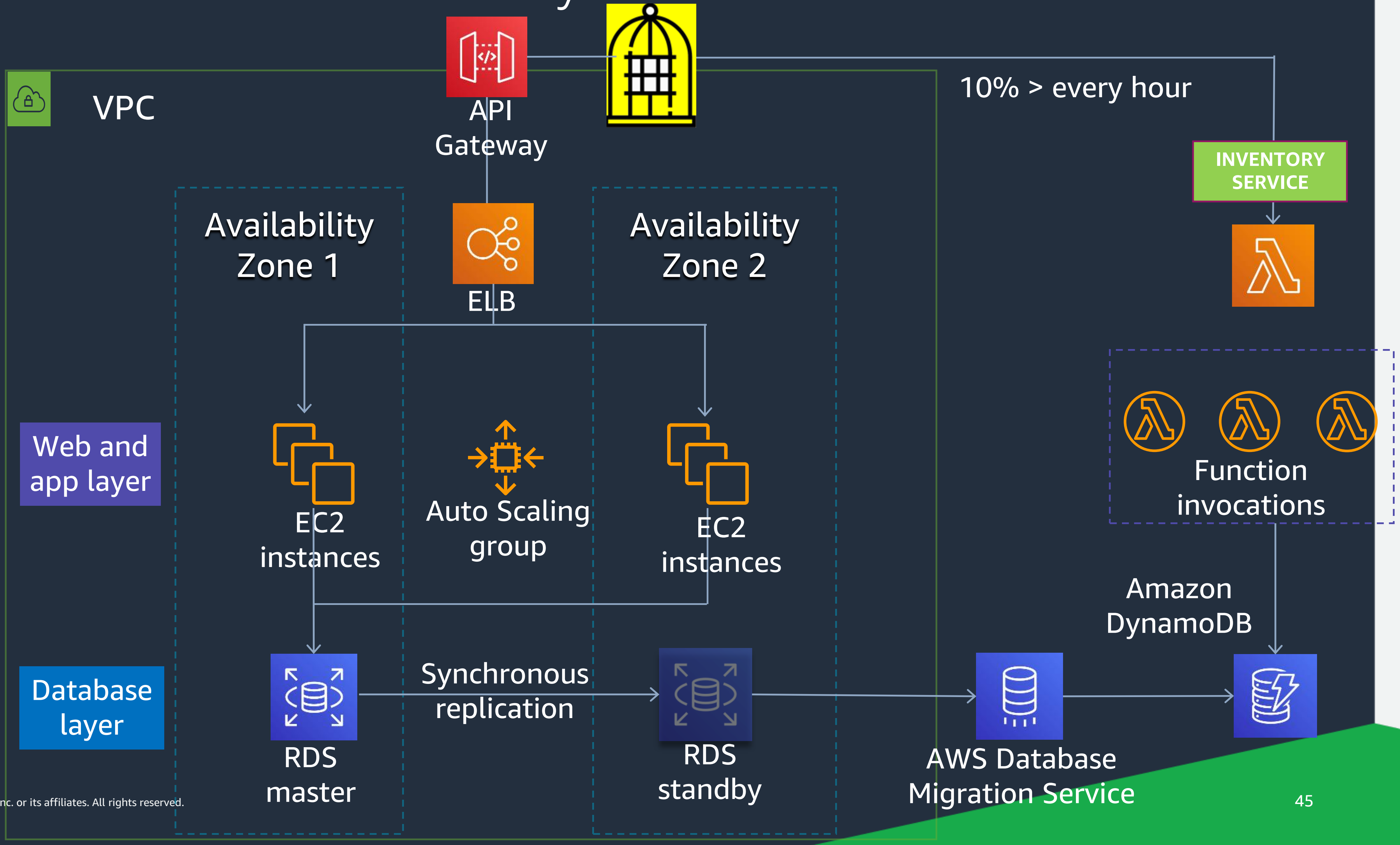
Canary Release

- In a Canary Deployment, you send a small percentage of requests to your new application as a way of testing it without exposing the majority of your users to this new and untested app.
- If the app performs well, you can continue to increase the percentage of requests that it receives until finally, you have promoted it to receive 100% of the requests.



Traffic split between
Old and New Inventory Service

Strangling The Monolith – Canary Release



Thank you!