EDA ROADSHOW SÃO PAULO 2023

Thinking Asynchronously: application integration patterns for microservices

Rebekah Kulidzan (she/her)

Go-to-Market Specialist, Integration Services
Amazon Web Services

Quem sou eu?

- Rebekah (Bek) Kulidzan
- GTM Specialist, Integration Services
- Application modernization enthusiast
- Mental Health Advocate
- Former Solutions Architect, Data Engineer, Policy Consultant and Social Media Advisor
- Love travel and speak Spanish, Serbian/Croatian and soon… Portuguese?

Agenda

Common serverless pattern

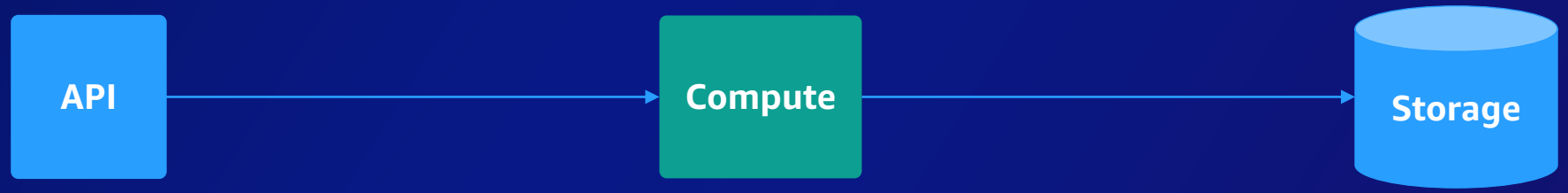
Decoupling your application

Enterprise Integration patterns

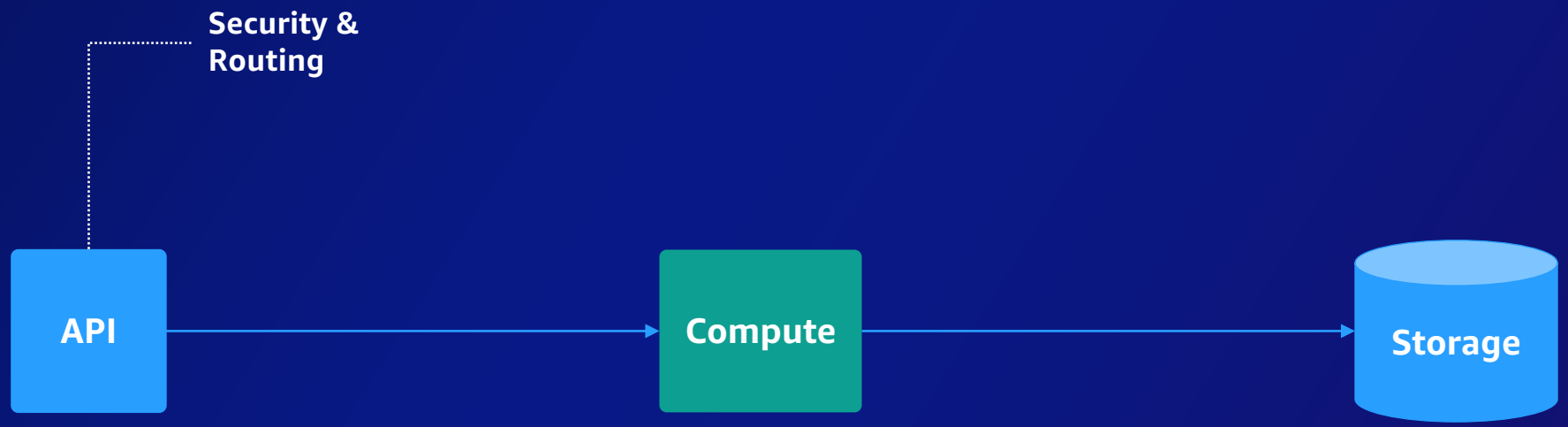
Event-driven architecture

Common serverless pattern

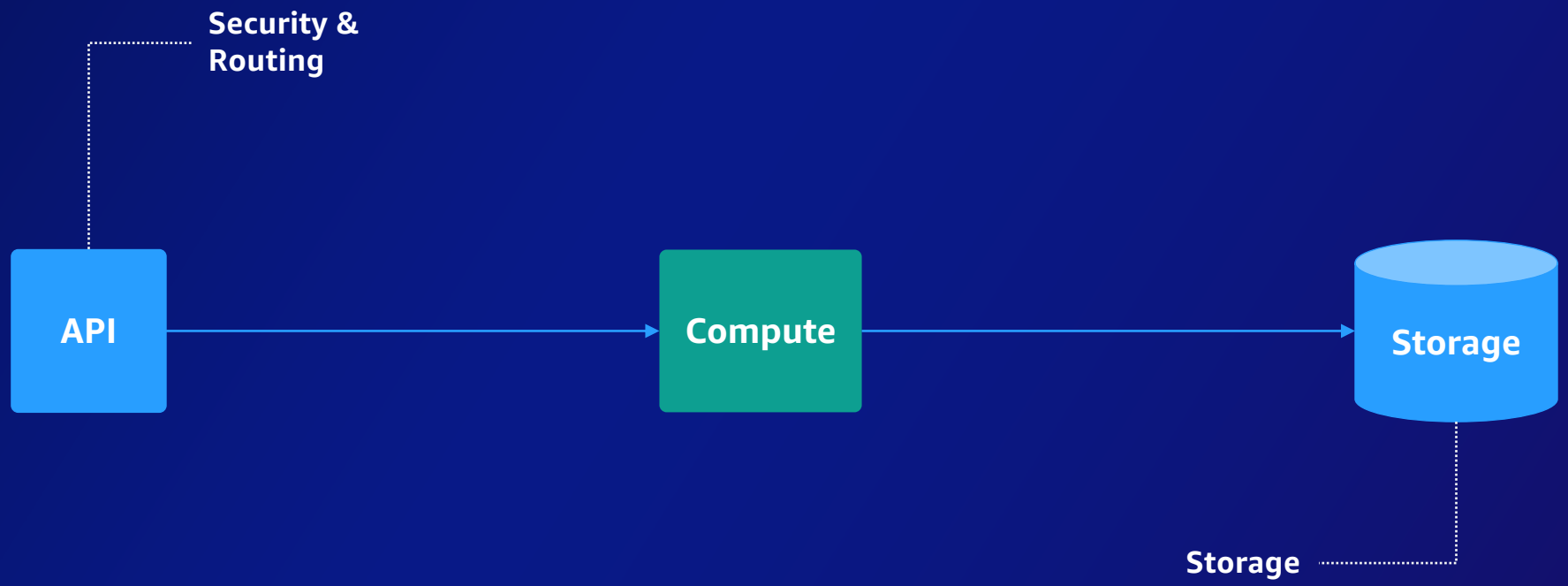
Application elements



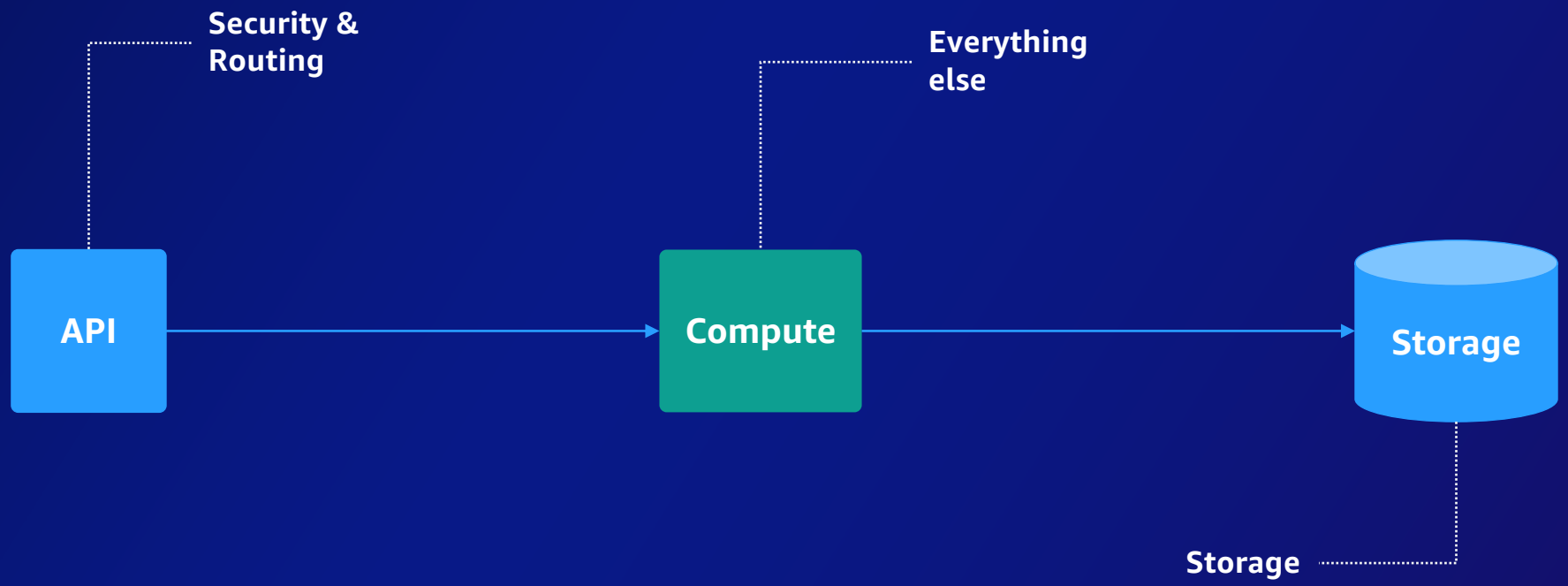
Application elements



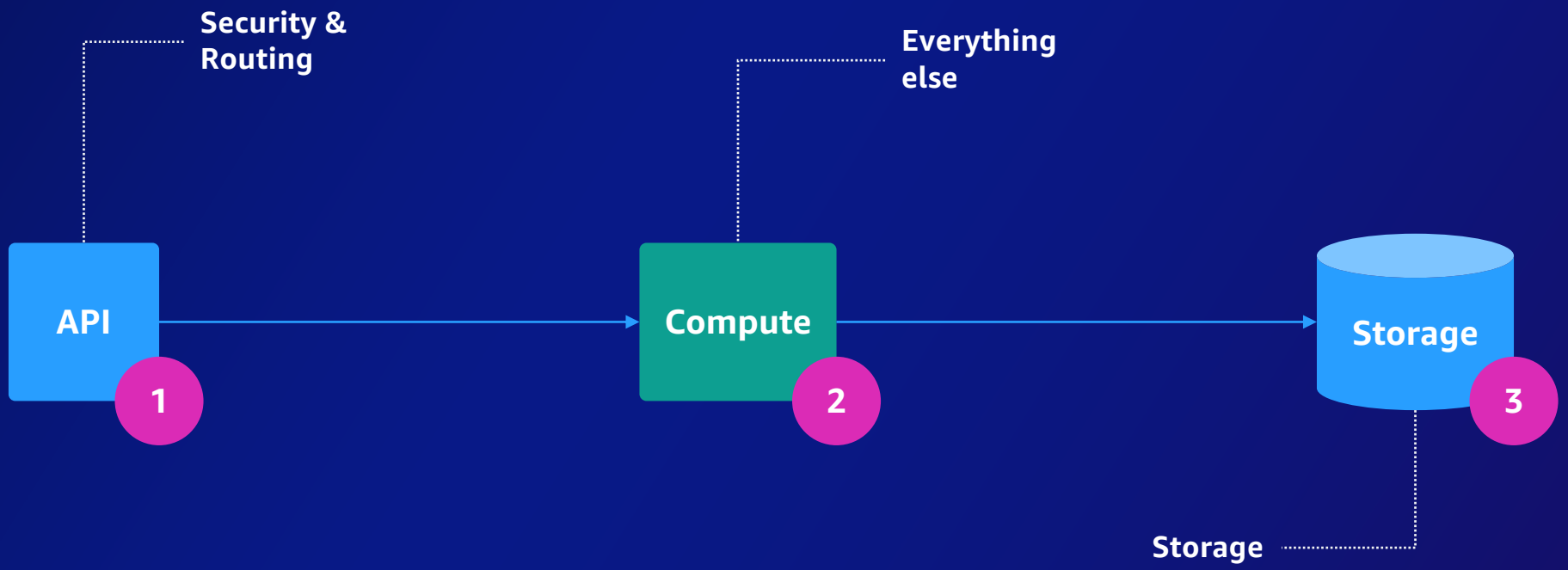
Application elements



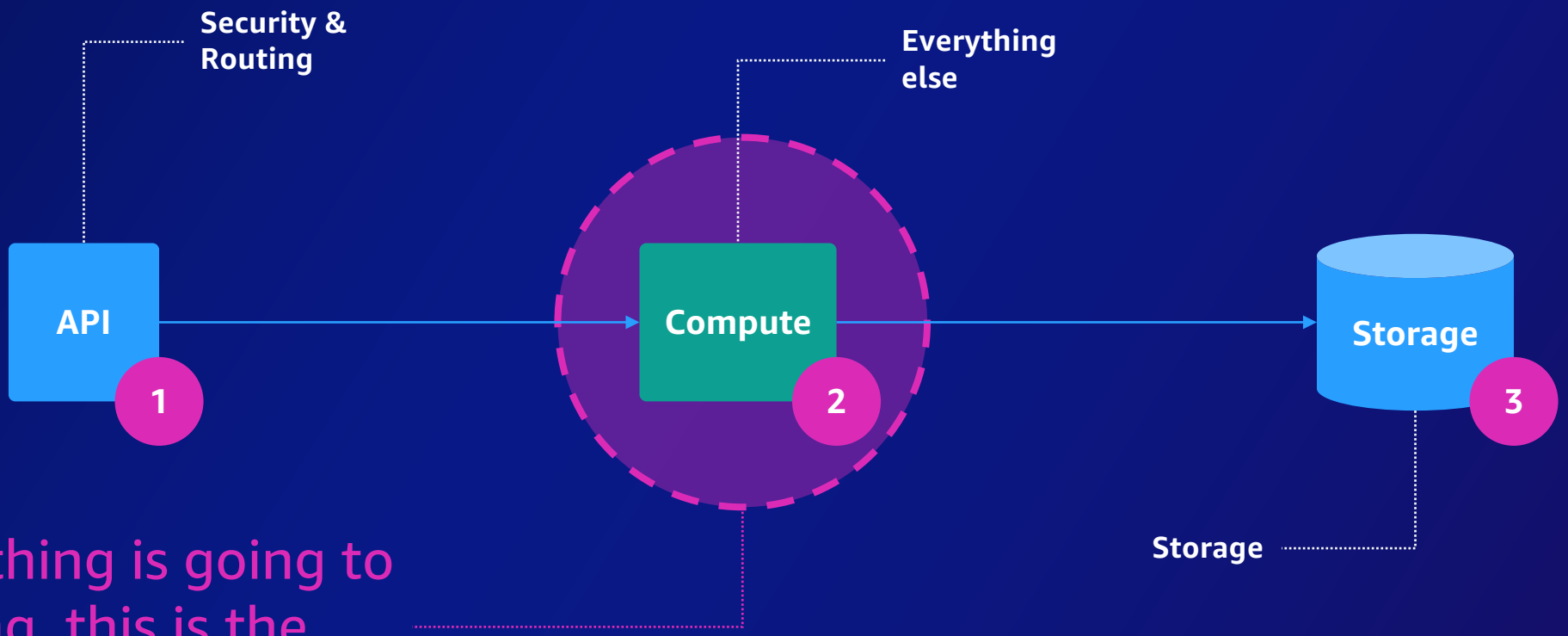
Application elements



Application elements

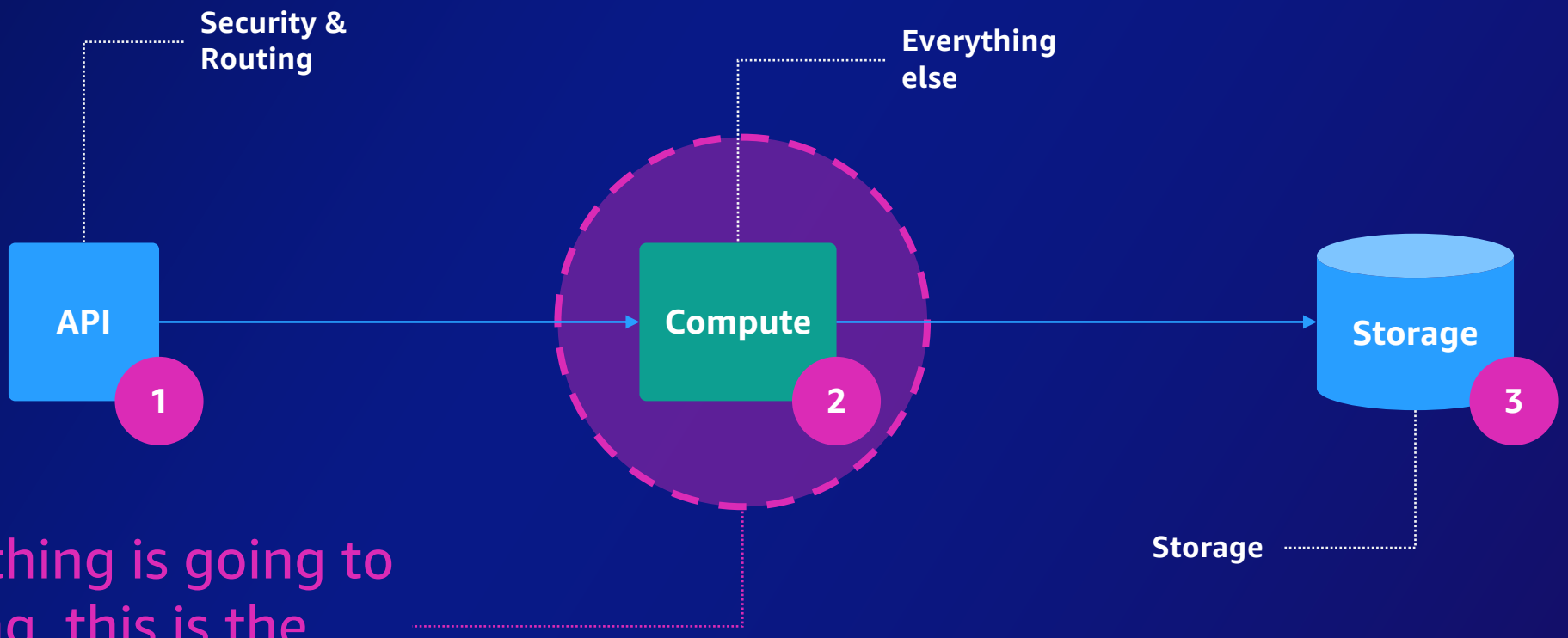


Application elements



If something is going to go wrong, this is the most probable place

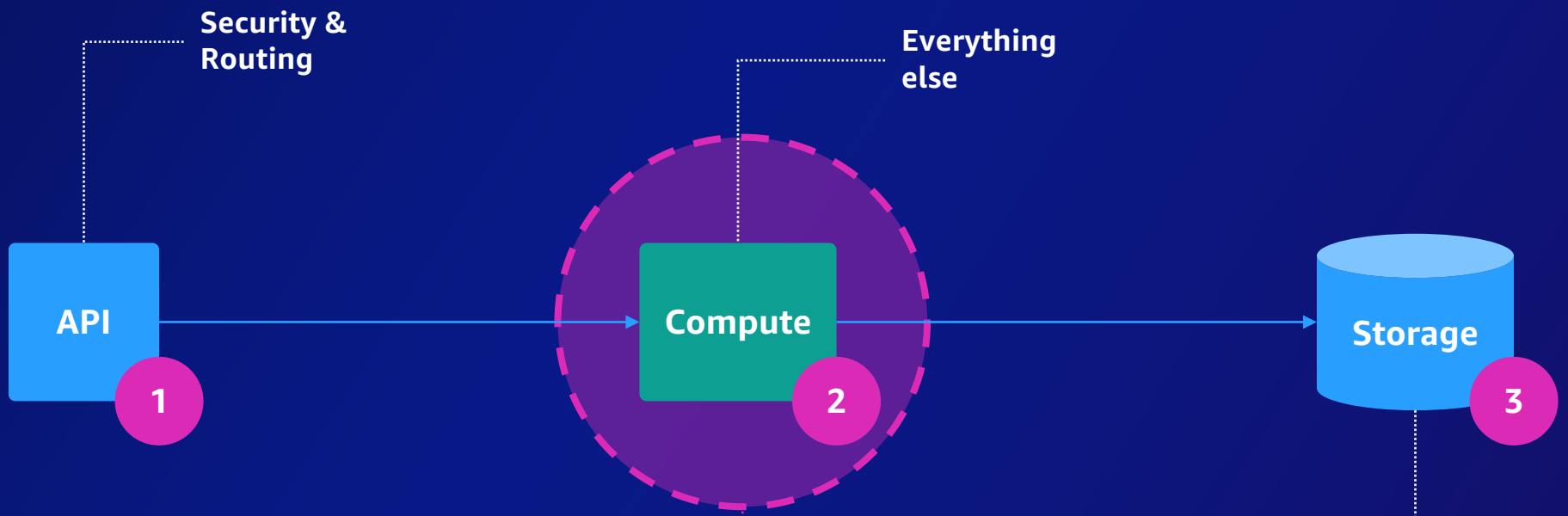
Application elements



If something is going to go wrong, this is the most probable place

Why?

Application elements



If something is going to go wrong, this is the most probable place

Why?

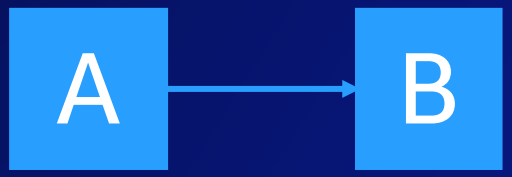
Because it is MY code!

How do we improve this?

Modern application

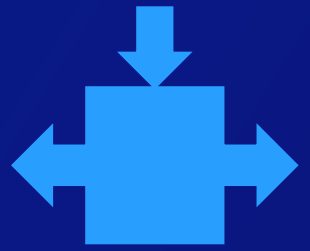
Then you might say the following…

"I want to sequence tasks"

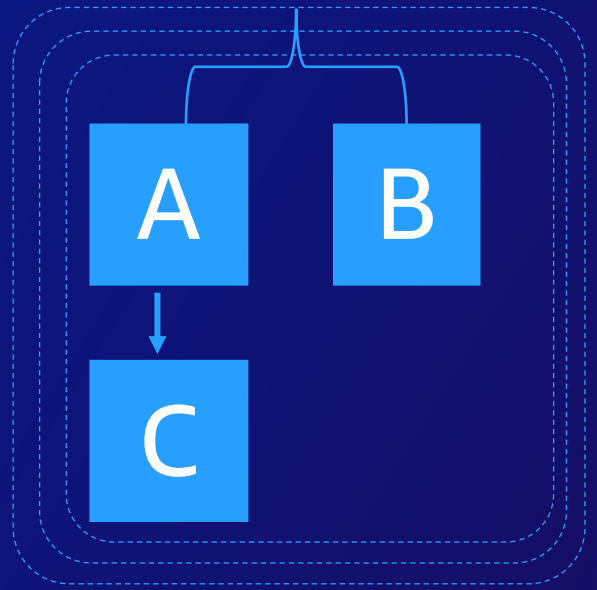


"I want to retry failed tasks"

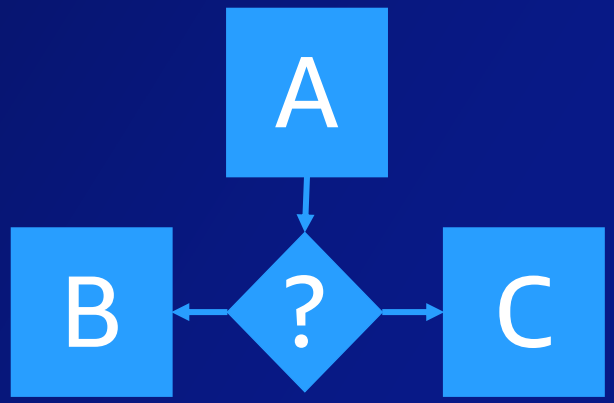
"I want to try / catch"



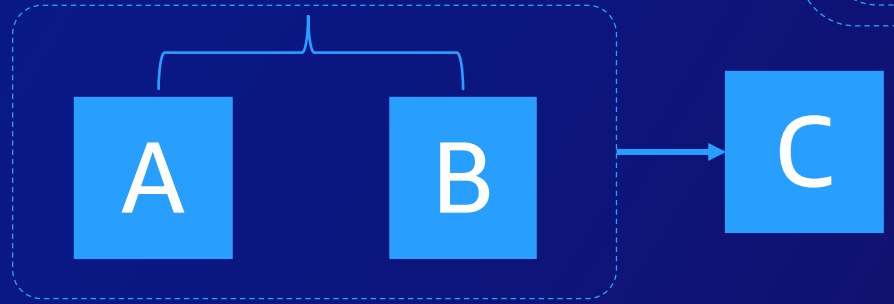
"I want concurrent and iterative tasks"



"I want to select tasks based on data"



"I want to run tasks in parallel"



Decoupling your application

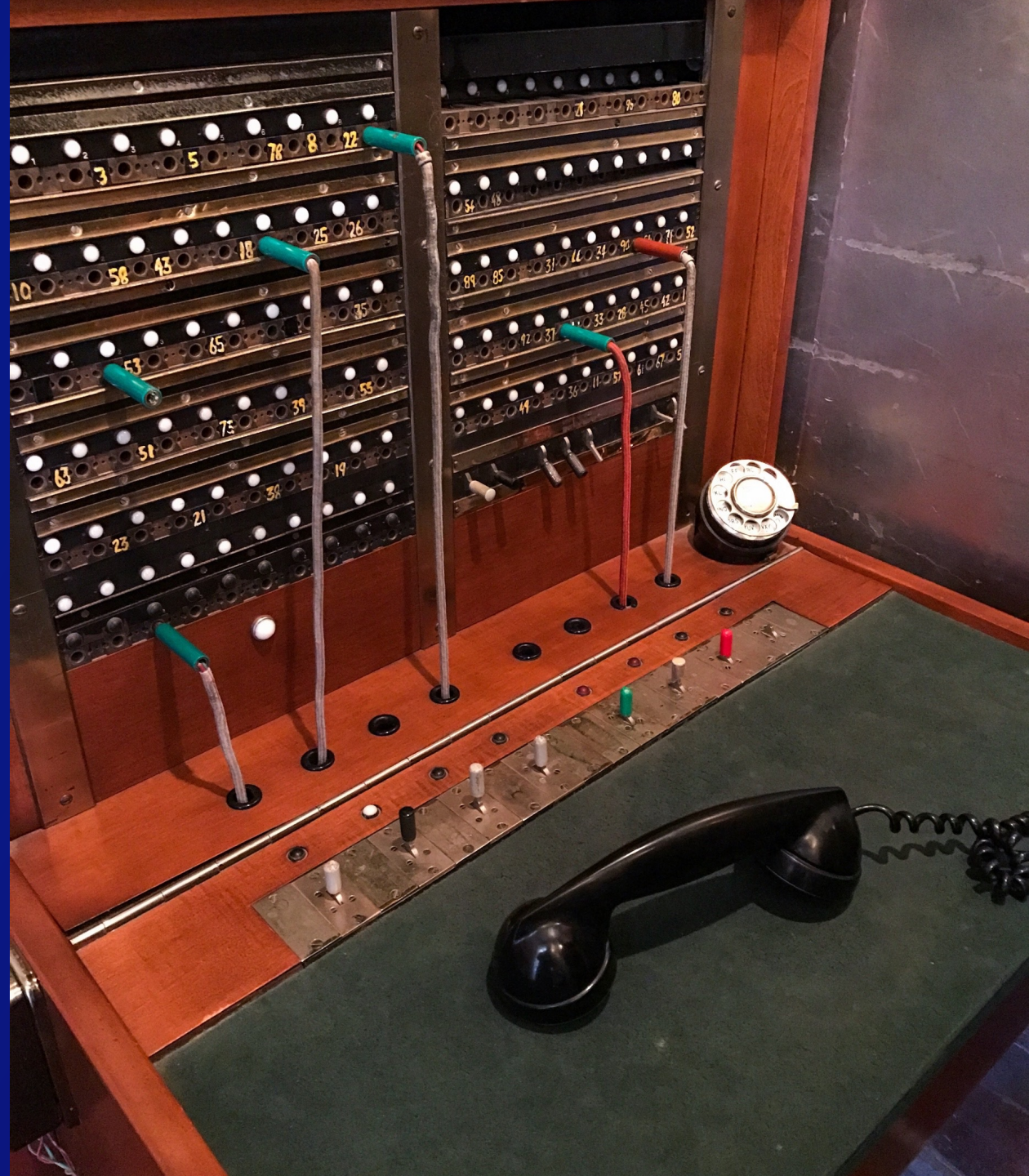
“The appropriate level of coupling depends on the level of control you have over the endpoints.”

Gregor Hohpe

Enterprise Integration Patterns

What does this mean?

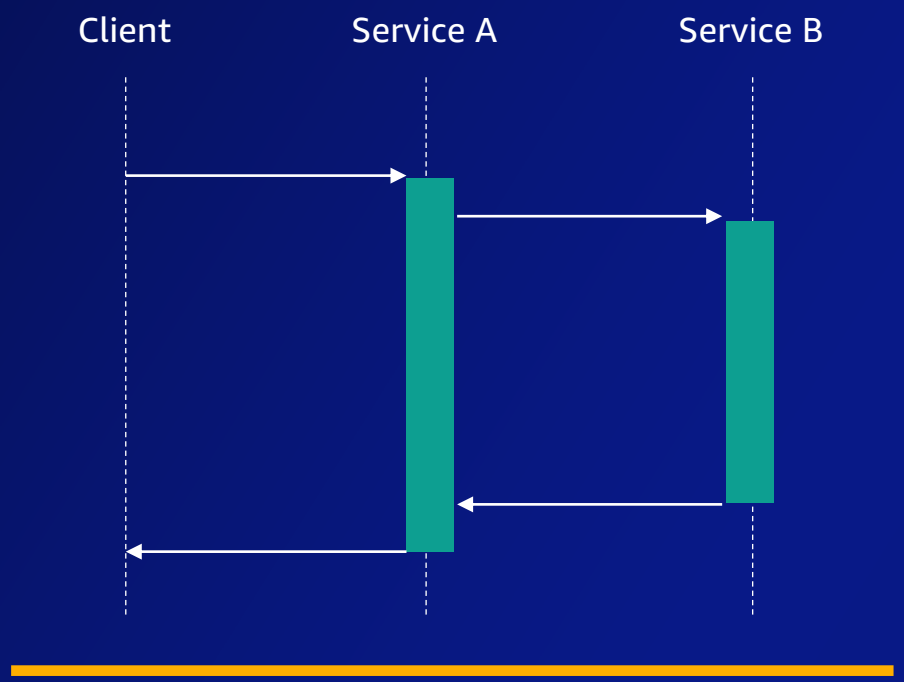
- Coupling is a measure of independent variability between connected systems
- Decoupling has a cost, both at design and run-time
- Coupling isn’t binary
- Coupling isn’t one-dimensional



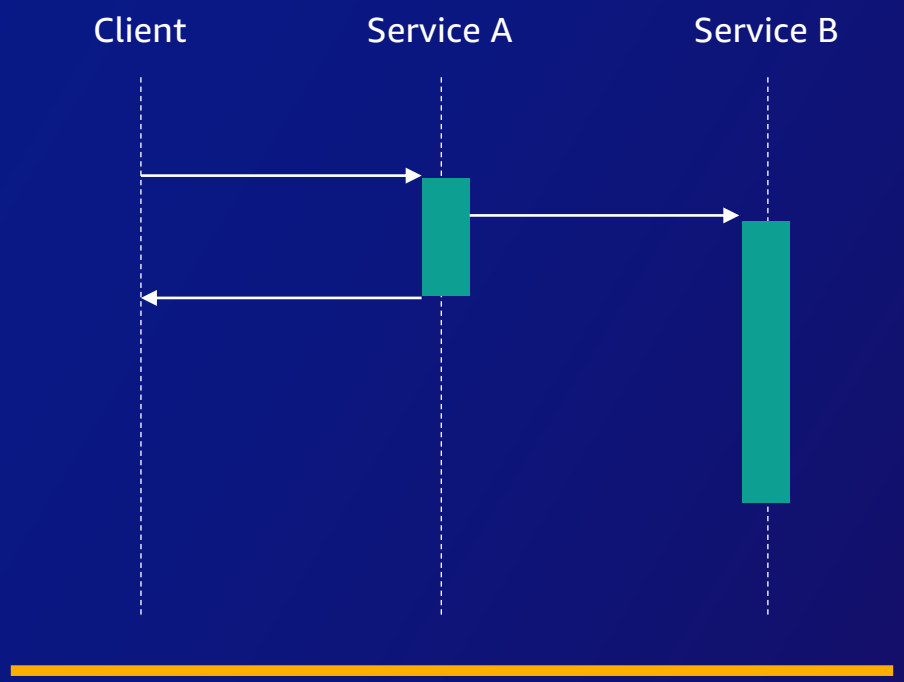
Source: EnterpriseIntegrationPatterns.com

Thinking Asynchronously

Think more asynchronously



Synchronous commands



Asynchronous events

Thinking Asynchronously

Synchronous

- Low latency
- Simple
- Fail fast
- Receiver failure
- Receiver throttled

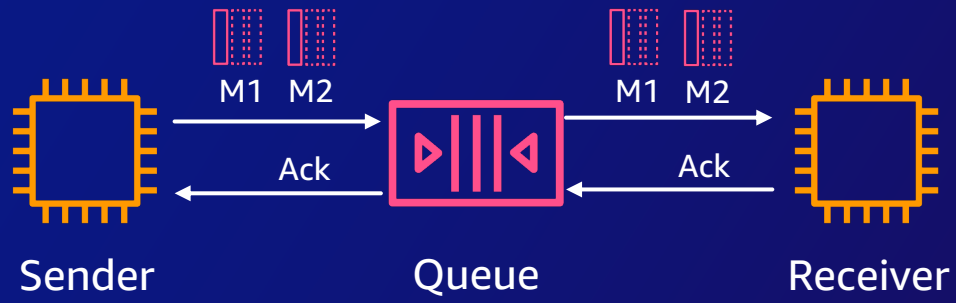
Asynchronous

- Decreases temporal / location coupling
- Resilient to receiver failure
- Receiver controls consumption rate
- Dead-letter queue (DLQ) for errors
- Only one receiver can consume each message
- Response correlation
- Backlog recovery time
- Fairness in multi-tenant systems

If you don’t need a response, execute asynchronously

Synchronous

Asynchronous



“If your application is cloud-native, large-scale, or distributed, and doesn’t include a messaging component, that’s probably a bug.”

Tim Bray

Software developer and environmentalist

Enterprise integration patterns

Message exchange

One-way

Message exchange

One-way

Sender

Receiver

No response expected

Message intent

Message exchange

One-way

Sender

Receiver

No response expected

Message intent

Asynchronous request-response

Message exchange

One-way

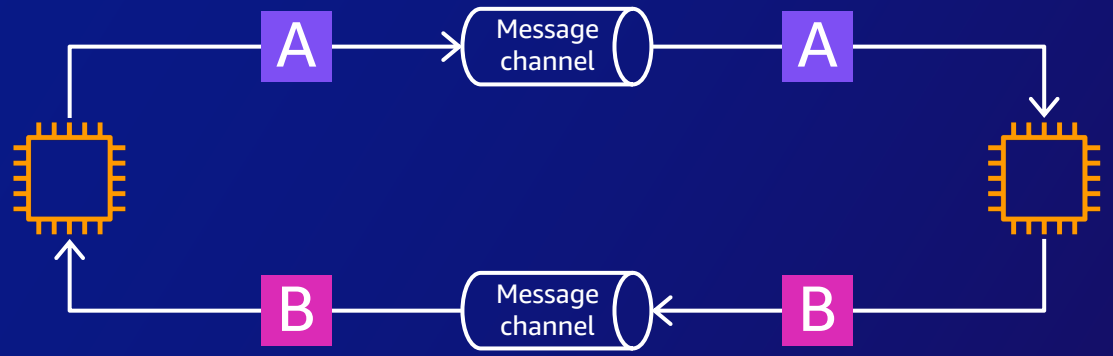
Sender

Receiver

No response expected

Message intent

Asynchronous request-response



Requester

Responder

Response expected

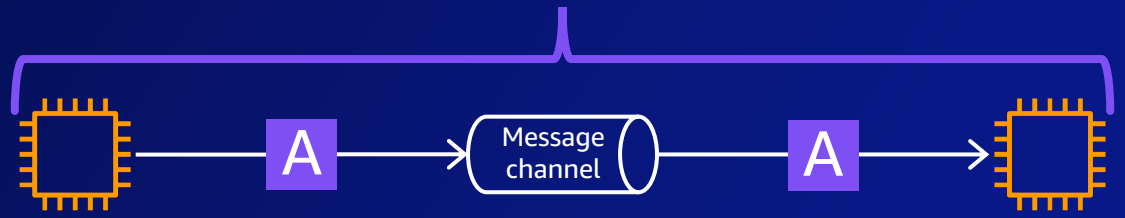
Return address

Correlation ID

Message exchange

One-way

Integration pattern



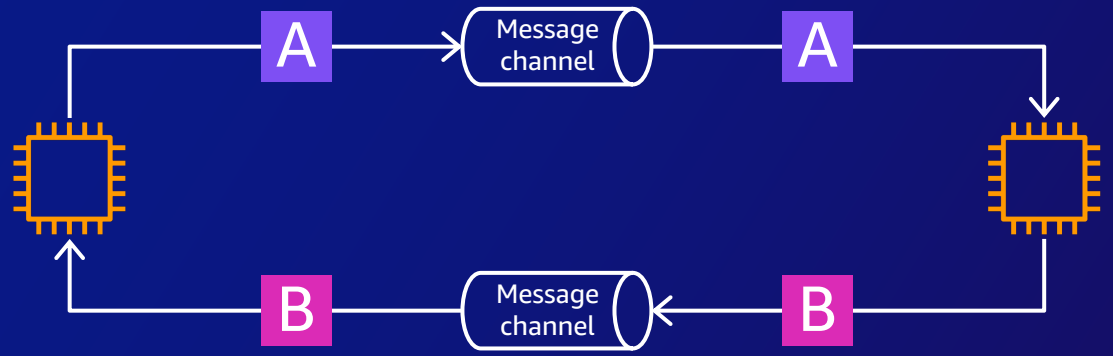
Sender

Receiver

No response expected

Message intent

Asynchronous request-response



Requester

Responder

Response expected

Return address

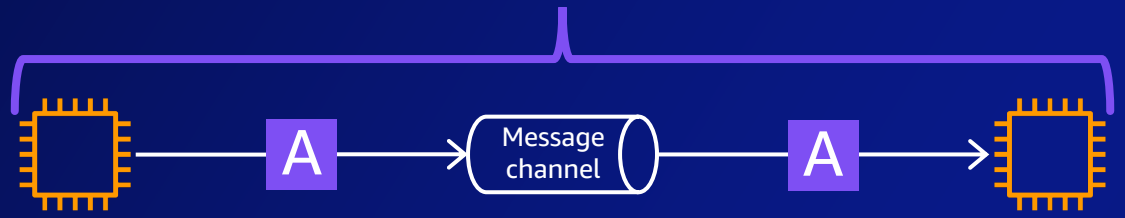
Correlation ID

Message exchange

Conversation pattern

One-way

Integration pattern



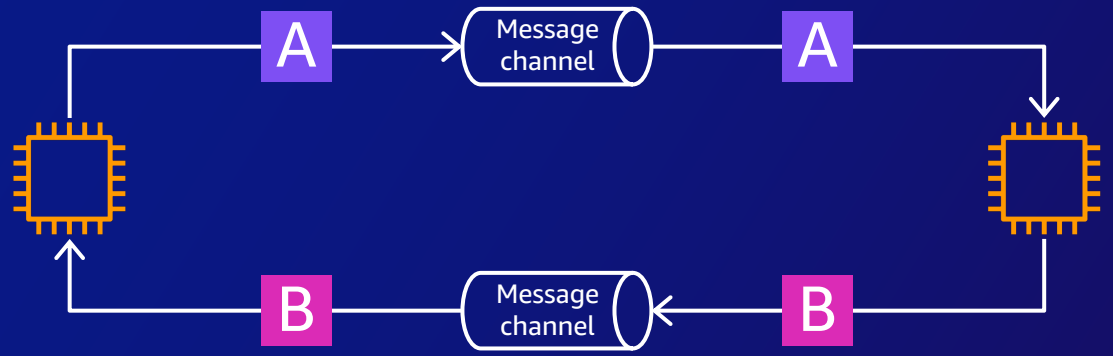
Sender

Receiver

No response expected

Message intent

Asynchronous request-response



Requester

Responder

Response expected

Return address

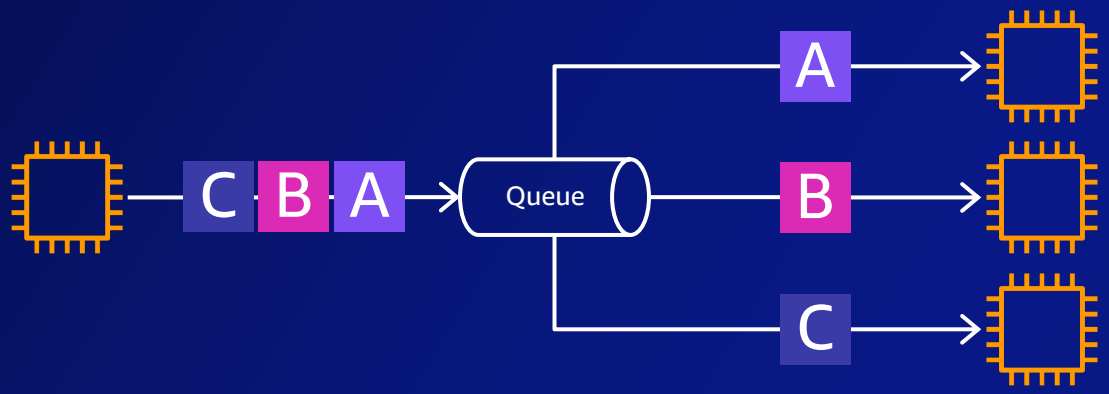
Correlation ID

Message channels

Point-to-point (queue)

Message channels

Point-to-point (queue)



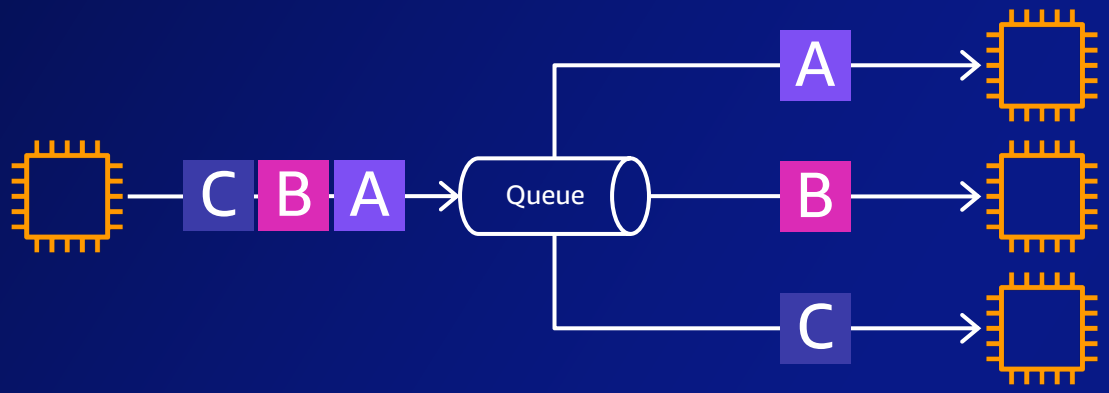
Each message consumed by one receiver

Consumers easy to scale

Buffering can flatten peak loads

Message channels

Point-to-point (queue)



Each message consumed by one receiver

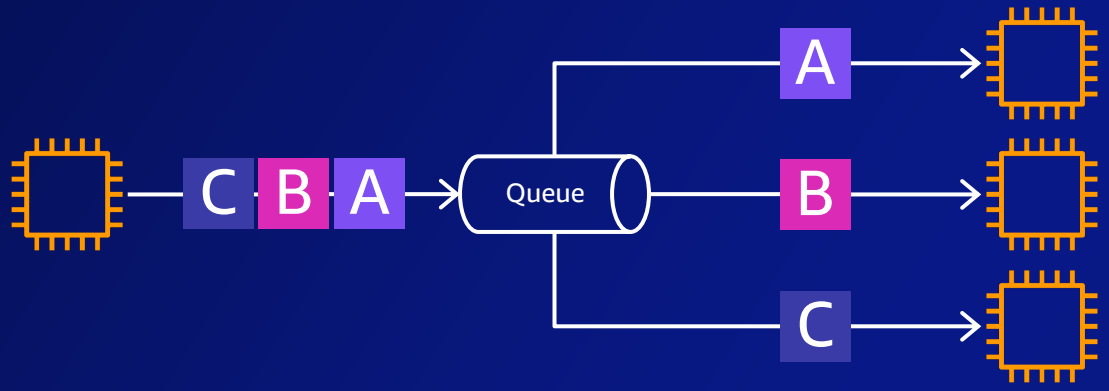
Consumers easy to scale

Buffering can flatten peak loads

Publish-subscribe (topic)

Message channels

Point-to-point (queue)

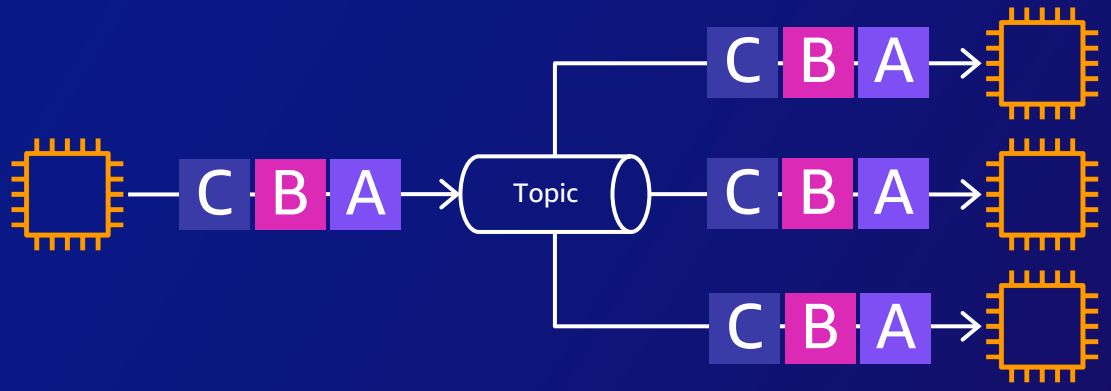


Each message consumed by one receiver

Consumers easy to scale

Buffering can flatten peak loads

Publish-subscribe (topic)



Each message consumed by each subscriber

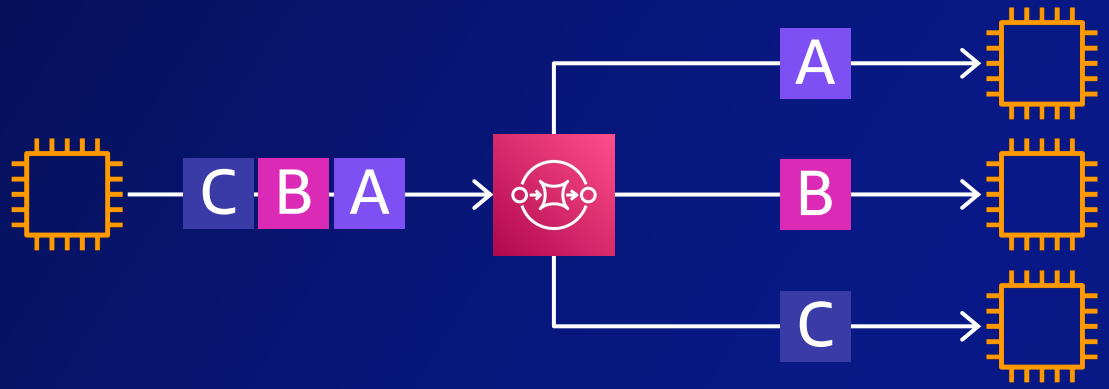
How to scale consumers?

Conceptually no buffering

Message channels

AWS services implementing
message queue and topic functionality

Point-to-point (queue)



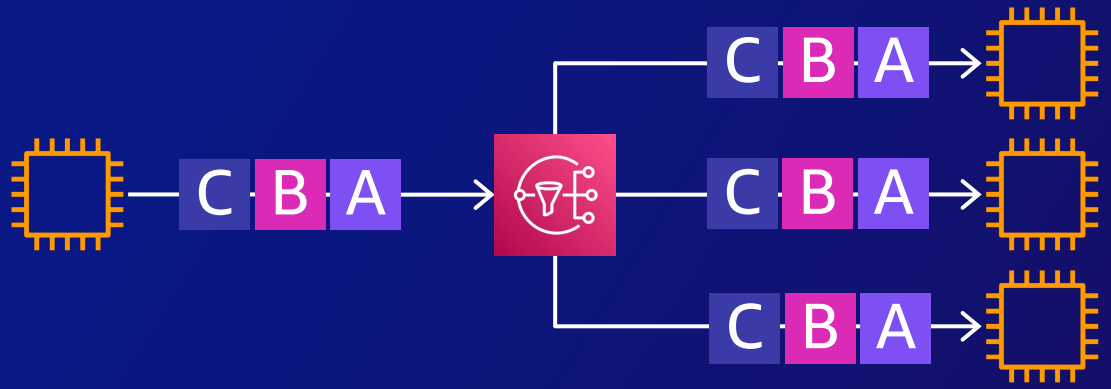
Sender(s)

Receivers

Amazon Simple Queue Service (Amazon SQS)

Cloud-native & serverless

Publish-subscribe (topic)



Publisher(s)

Subscribers

Amazon Simple Notification Service (Amazon SNS)

Cloud-native & serverless

Message channels

Dead-letter queue (DLQ)

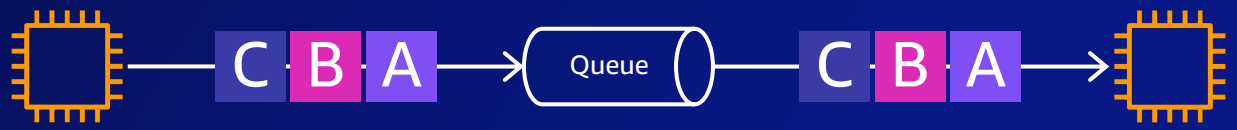
Message channels

Dead-letter queue (DLQ)

Producer

Message channels

Dead-letter queue (DLQ)

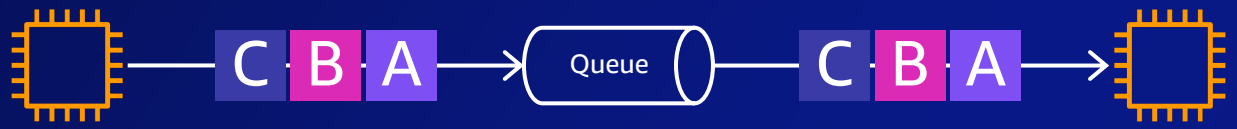


Producer

Consumers

Message channels

Dead-letter queue (DLQ)



Producer

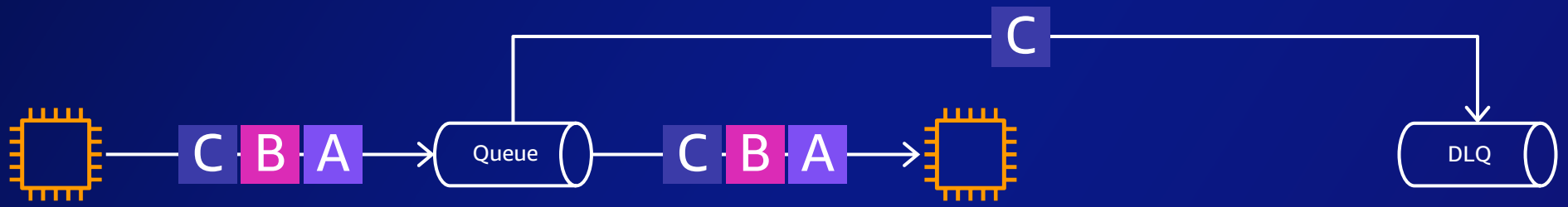
Consumers

Transient failure mitigation

Poison pill handling

Message channels

Dead-letter queue (DLQ)



Producer

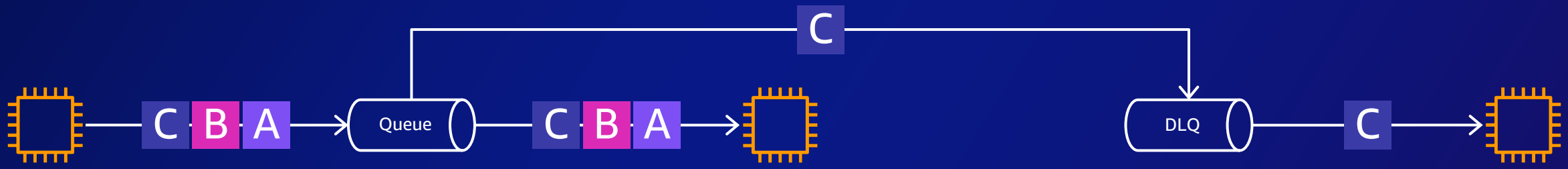
Consumers

Transient failure mitigation

Poison pill handling

Message channels

Dead-letter queue (DLQ)



Producer

Consumers

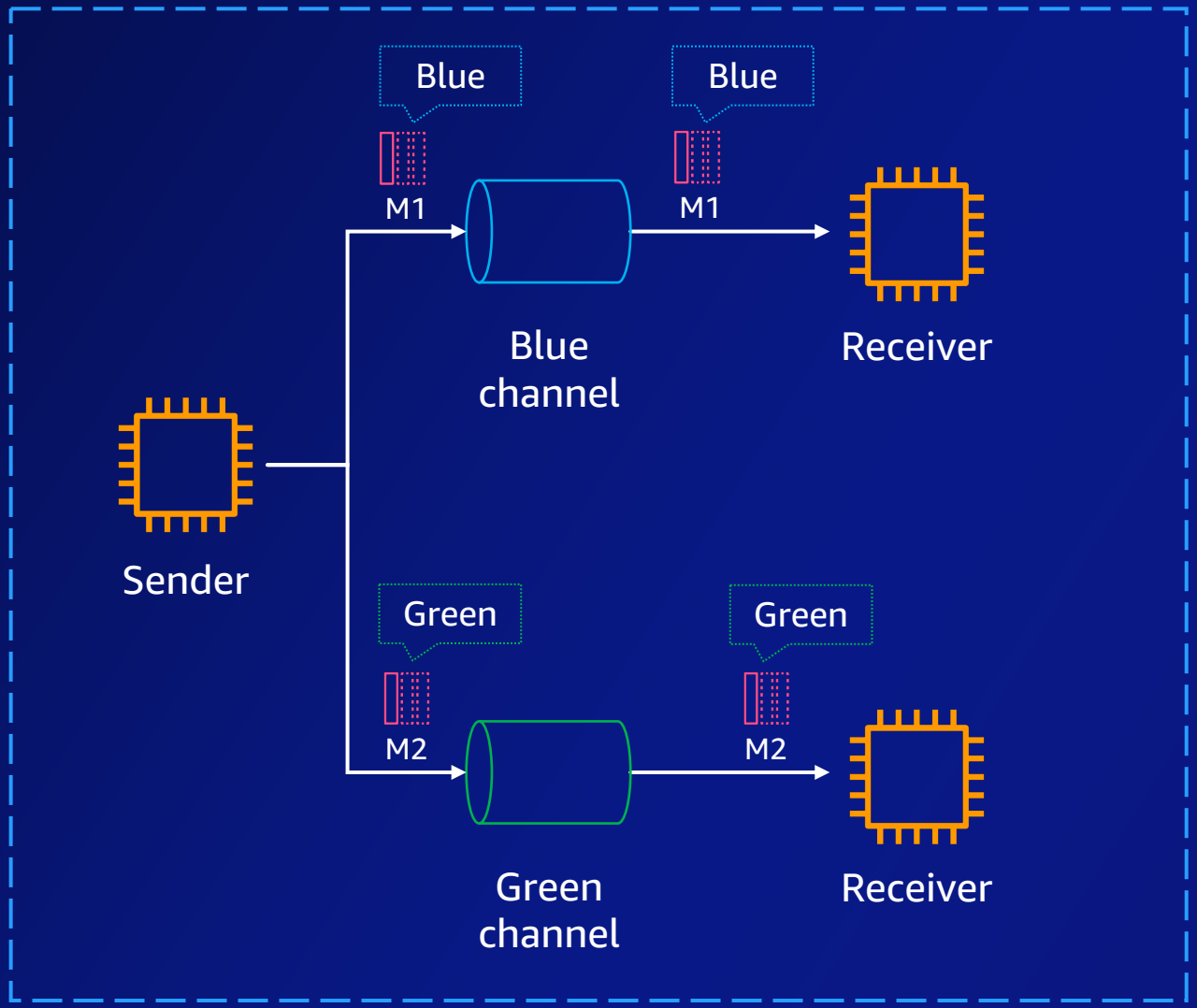
Application ops

Transient failure mitigation

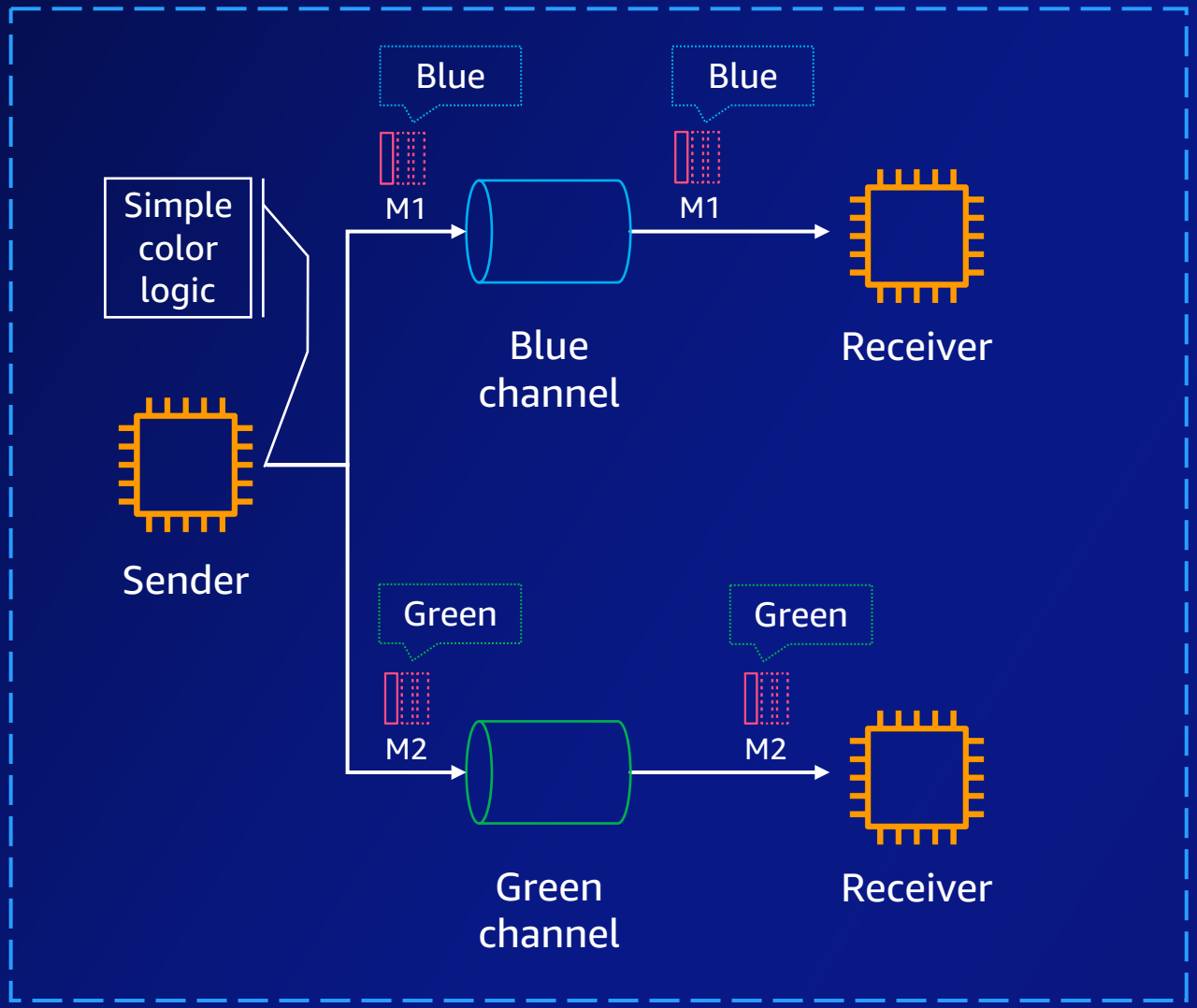
Poison pill handling

No need for open-heart surgery – investigate without the hassle of production

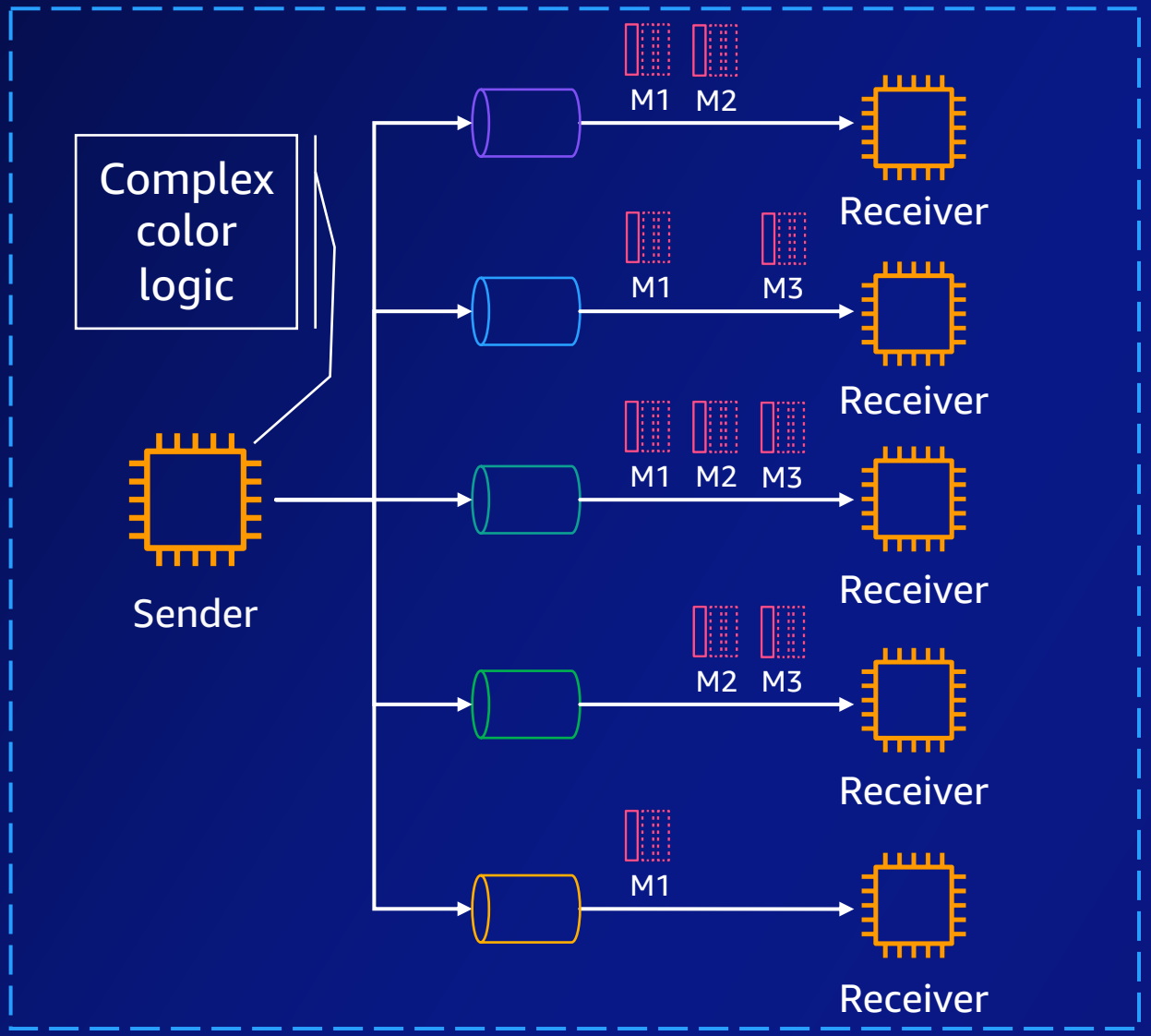
Asynchronous point-to-point model (router)



Asynchronous point-to-point model (router)

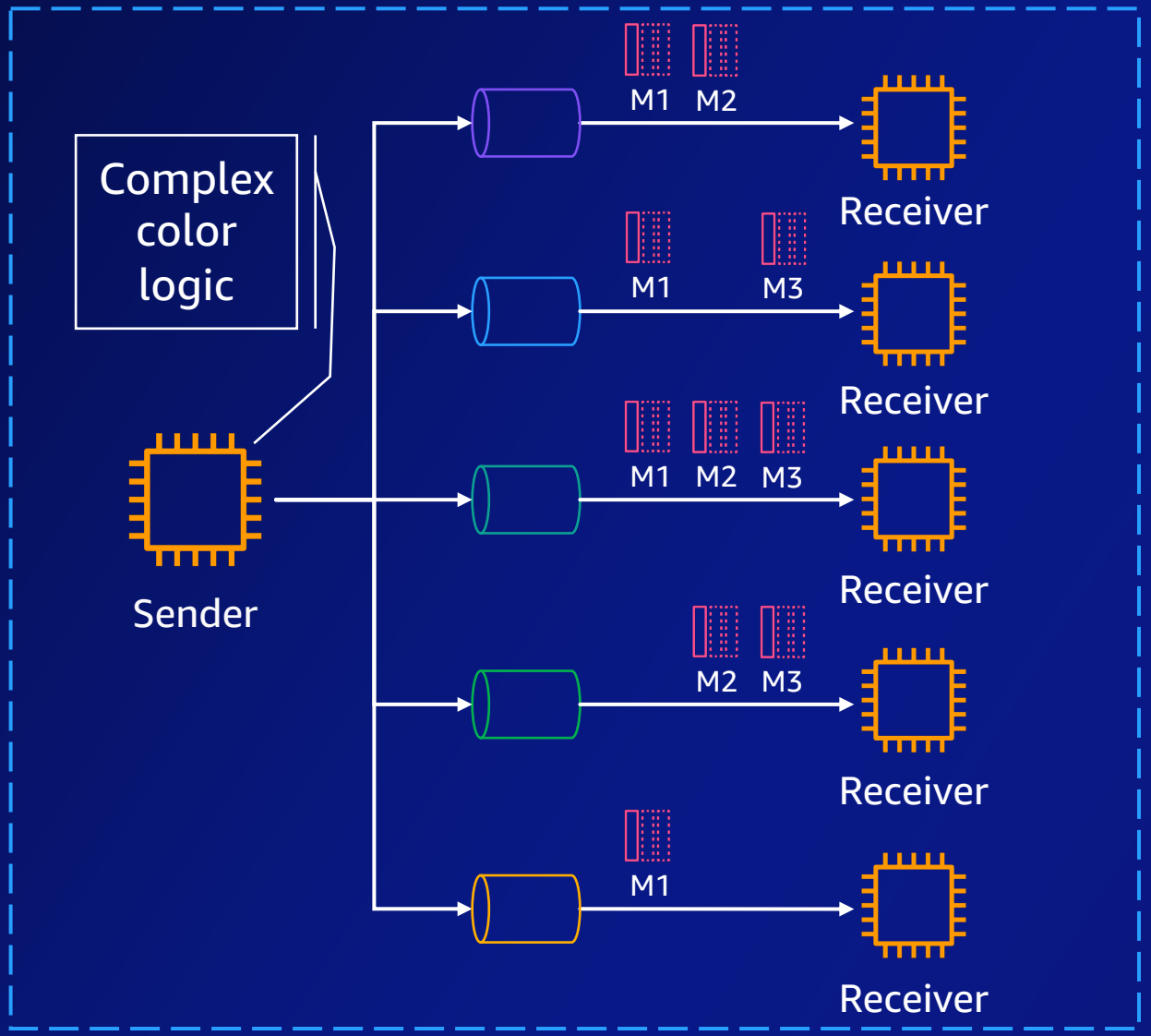


Asynchronous point-to-point model (router)



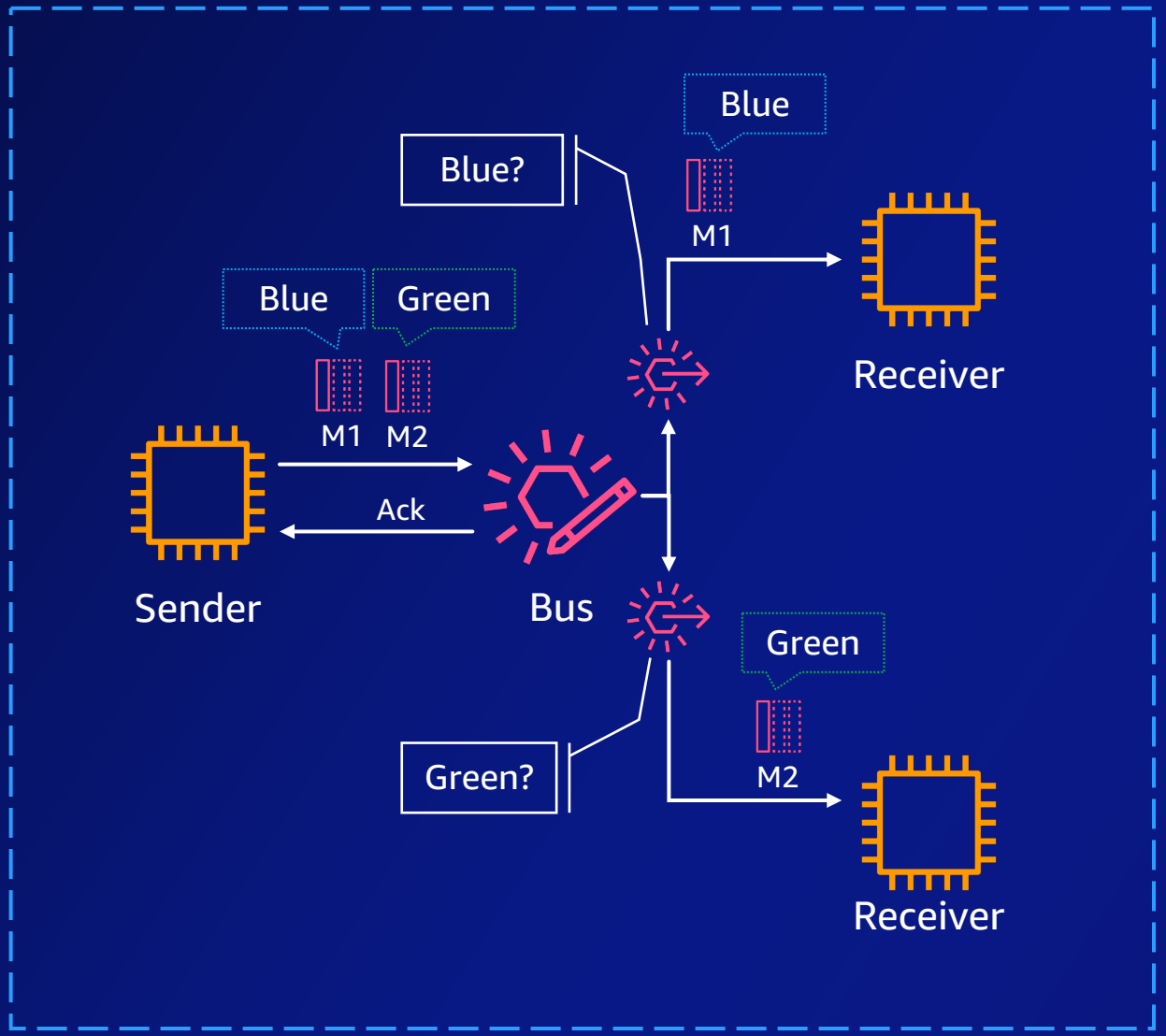
…Increased complexity, logic and tighter coupling

Asynchronous point-to-point model (router)

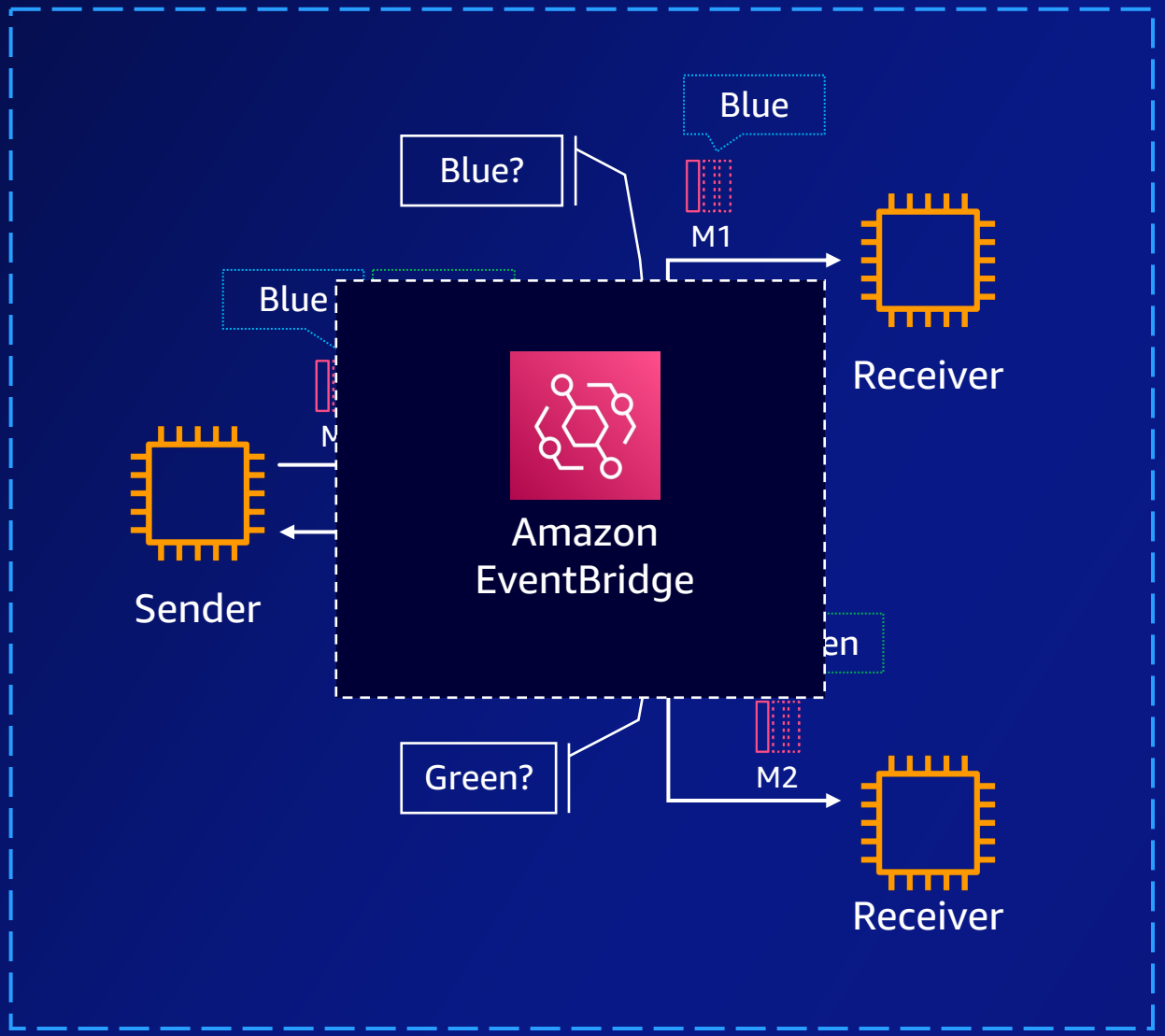


How can we make this easier to manage?

Asynchronous message-router (bus)

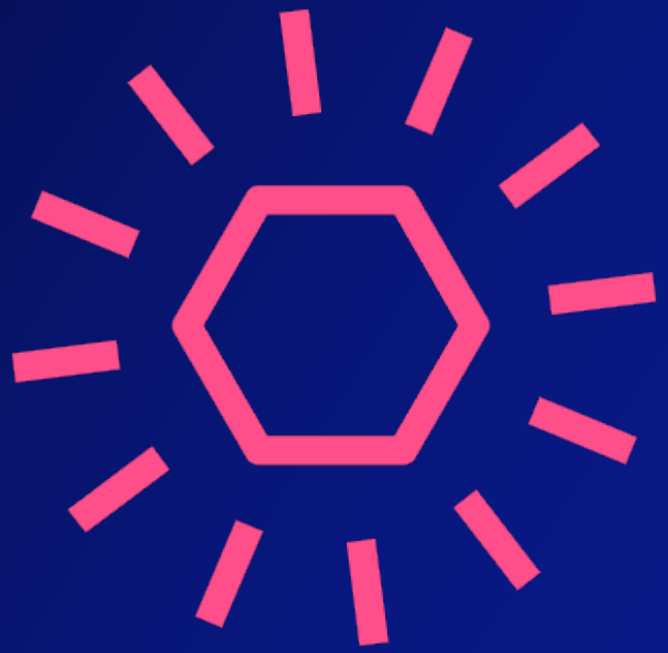


Asynchronous message-router (bus)



Amazon EventBridge is a simple, flexible, fully managed, pay-as-you-go, event bus service that makes it easy to ingest and process data from AWS services, your own applications, and SaaS applications.

Event-driven architecture



event

[i-’vent] **noun**

A signal that a system’s state has changed

Directed vs. observable events

Directed commands

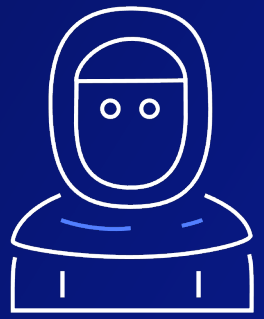
Directed vs. observable events

Directed commands

Observable events

It all starts with business events

OrderCreated



OrderPicked

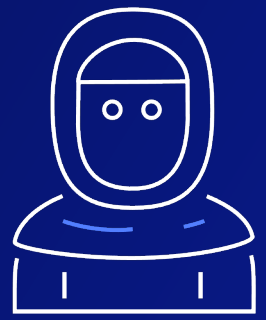
OrderShipped

ReturnRequested

ReturnReceived

Map events to business domains using attributes

OrderCreated



Retail

OrderPicked

OrderShipped

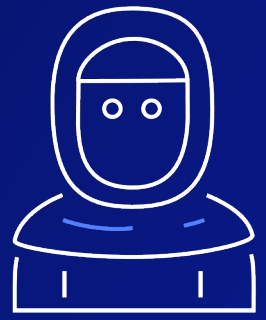
Fulfillment

ReturnRequested

ReturnReceived

Support

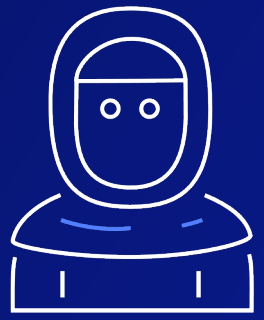
Choreograph events between domains using subscriptions



Retail

Fulfillment

Choreograph events between domains using subscriptions



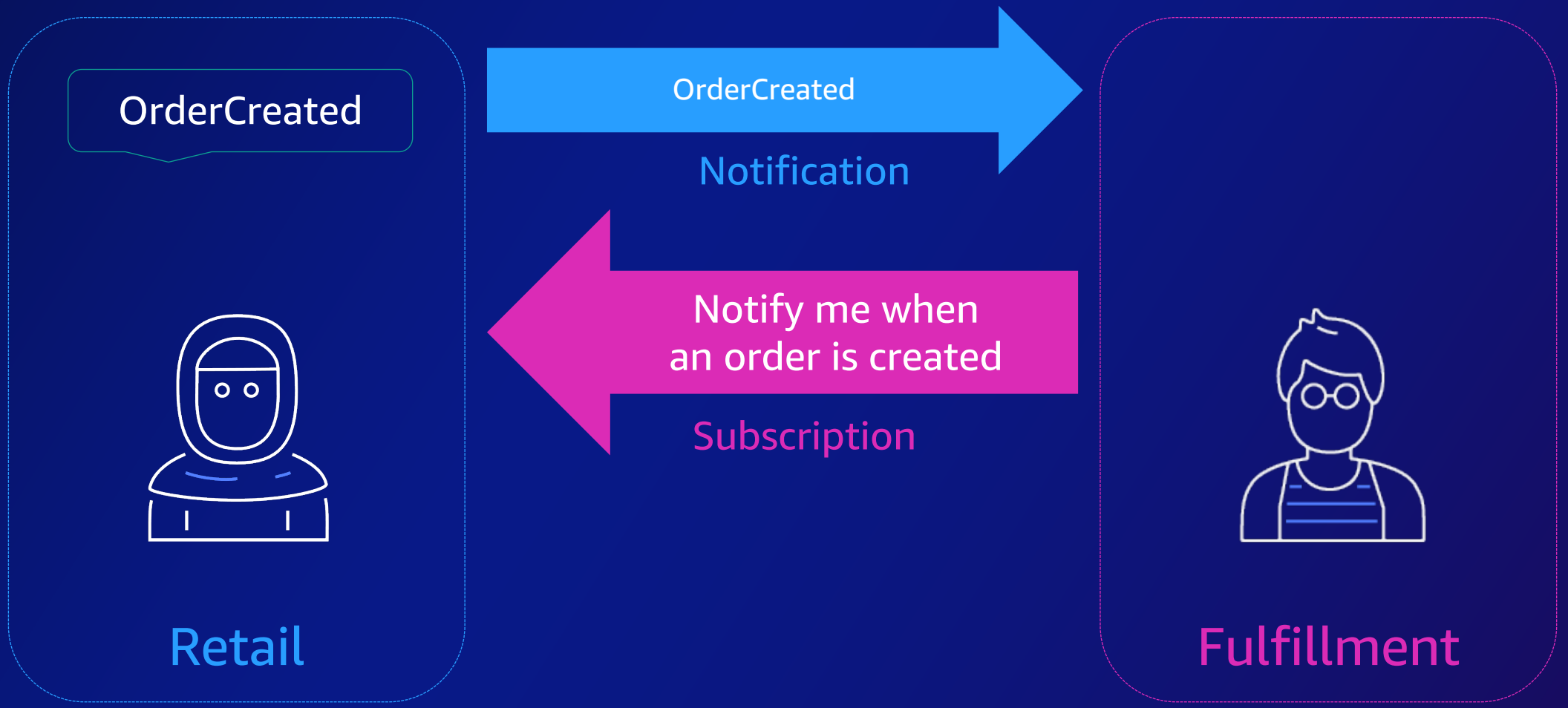
Retail

Notify me when an order is created

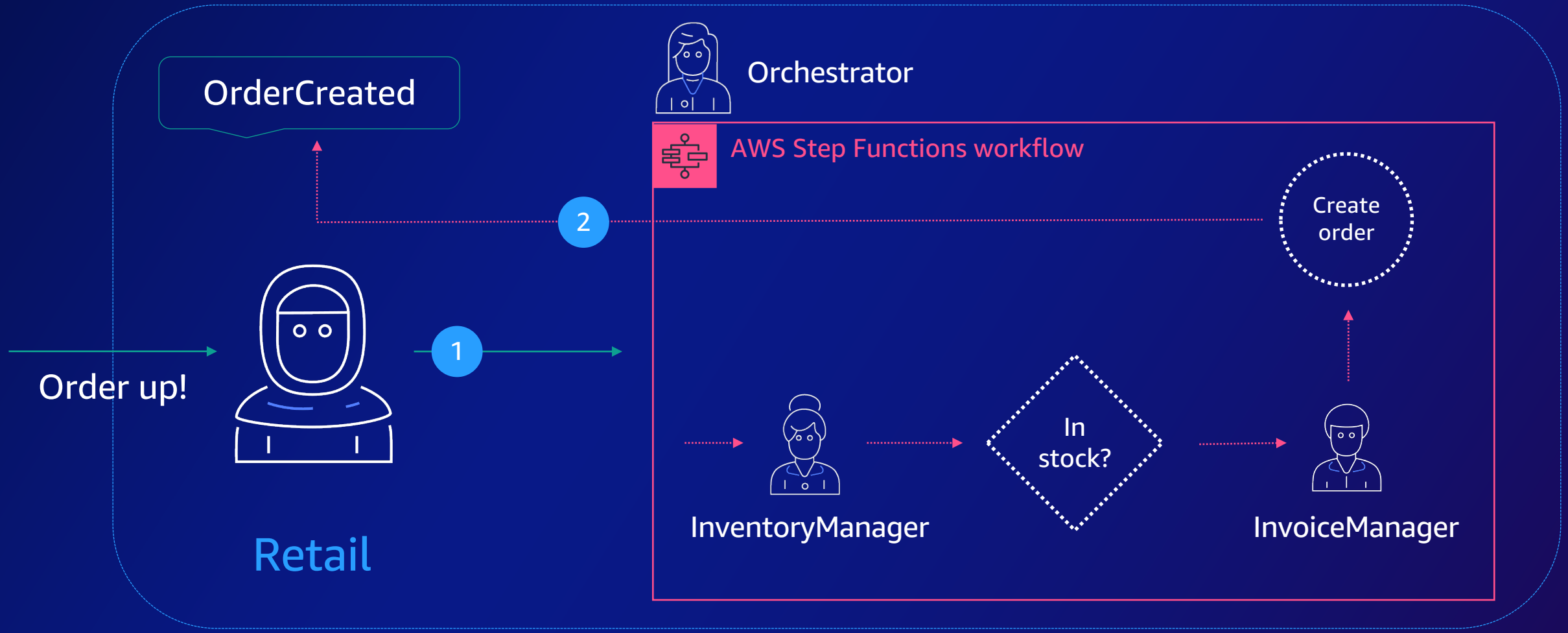
Subscription

Fulfillment

Choreograph events between domains using subscriptions



Orchestrate a business process within a domain, resulting in a published event

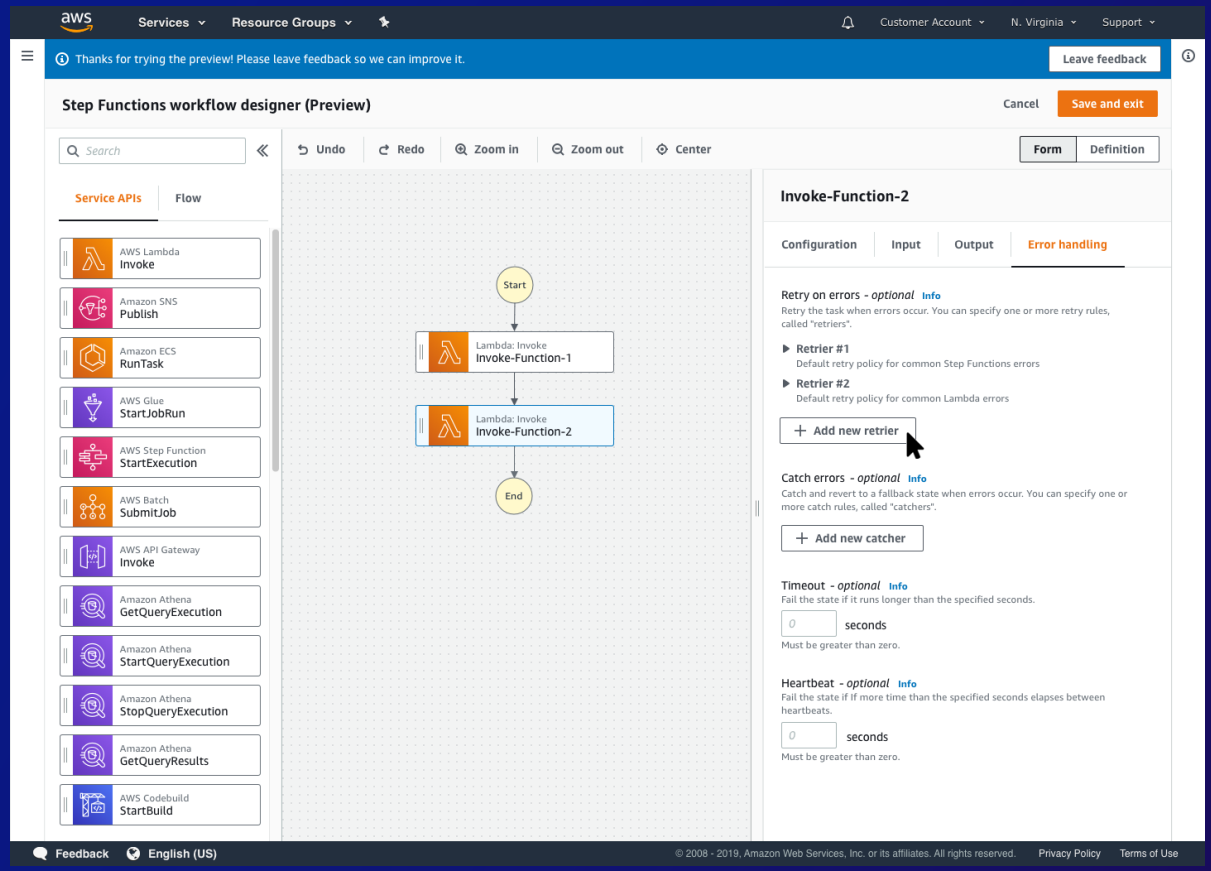


AWS Step Functions

The workflows you build with Step Functions are called state machines, and each step of your workflow is called a state

When you execute your state machine, each move from one state to the next is called a state transition

You can reuse components, easily edit the sequence of steps, or swap out the code called by task states as your needs change



AWS STEP FUNCTIONS WORKFLOW STUDIO

Visual workflows

Define

JSON – Amazon States Language

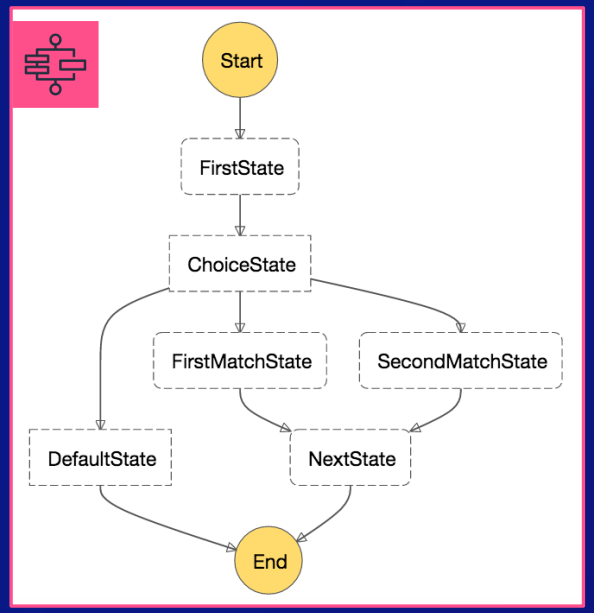
```
Code
{
  "Comment": "An AWL example using a choice state.",
  "StartAt": "FirstState",
  "States": {
    "FirstState": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
      "Next": "ChoiceState"
    },
    "ChoiceState": {
      "Type" : "Choice",
      "Choices": [
        {
```

CDK TypeScript, JavaScript, Python, Java, C#

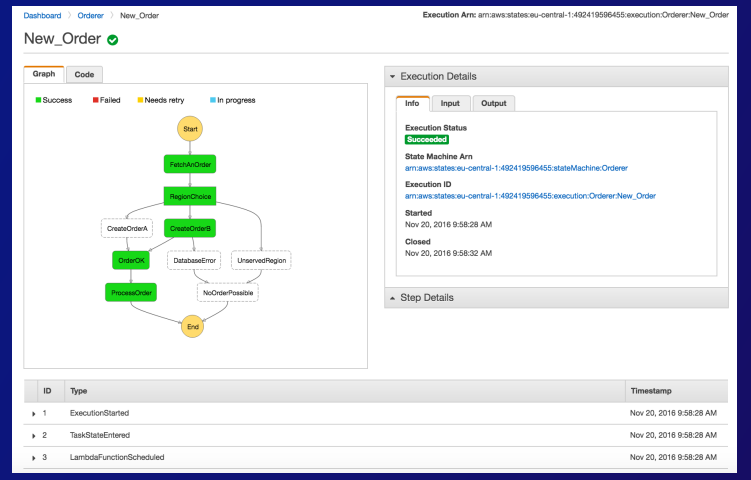
Data Science SDK

Python

Visualize



Execute and monitor



AWS Step Functions integration types

Optimized integrations

Customized to simplify the usage of 17 AWS services

Supported integration patterns:

- Request Response
- Wait for a Callback (.waitForTaskToken)
- Run a Job (.sync)

AWS SDK integrations

Call 220+ AWS services directly (10,000+ API actions)

Supported integration patterns:

- Request Response
- Wait for a Callback (.waitForTaskToken)

Task integration patterns

Request Response	AWS Step Functions will wait for an HTTP response and then progress to the next state. Step Functions will not wait for a job to complete.
Run a Job	Call a service and have Step Functions wait for a job to complete.
Wait for callback	Call a service with a task token and have Step Functions wait until that token is returned with a payload.

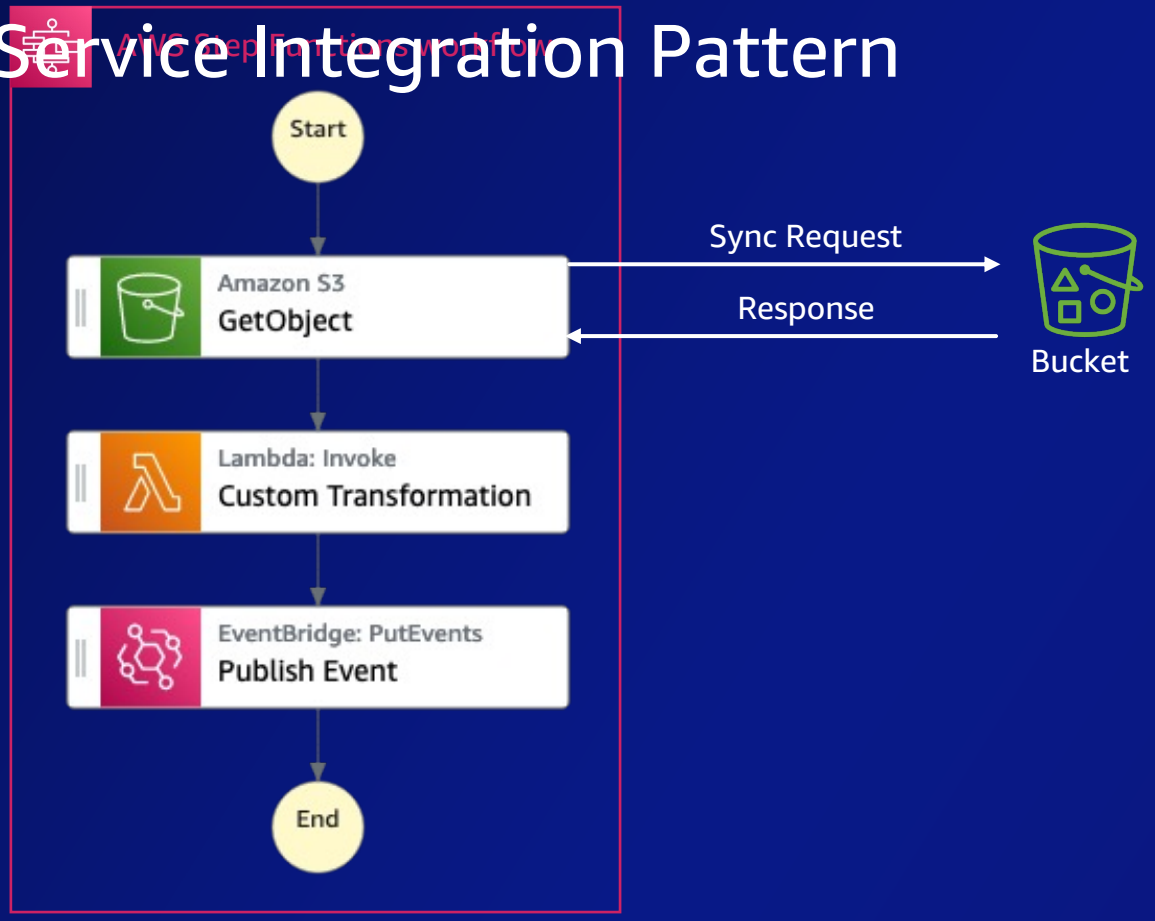
Request Response

SERVICE INTEGRATION PATTERN

- Amazon EMR
- AWS Lambda
- Amazon SQS
- AWS Glue
- Amazon ECS/ AWS Fargate
- Amazon SNS
- Amazon Athena
- Amazon EKS
- AWS Step Functions
- Amazon SageMaker
- AWS Batch
- Amazon EventBridge
- Amazon DynamoDB
- AWS SDK
- Amazon API Gateway

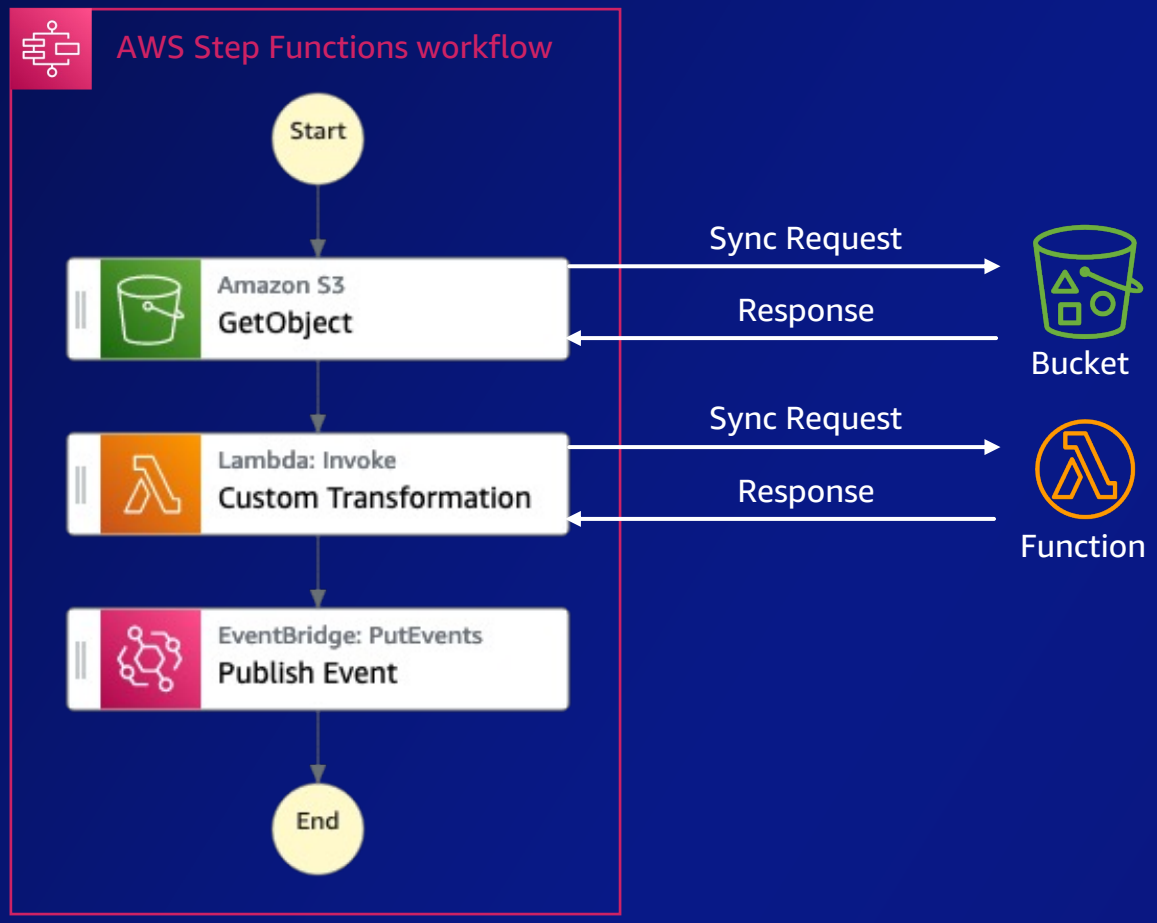
Request Response

Service Integration Pattern



Request Response

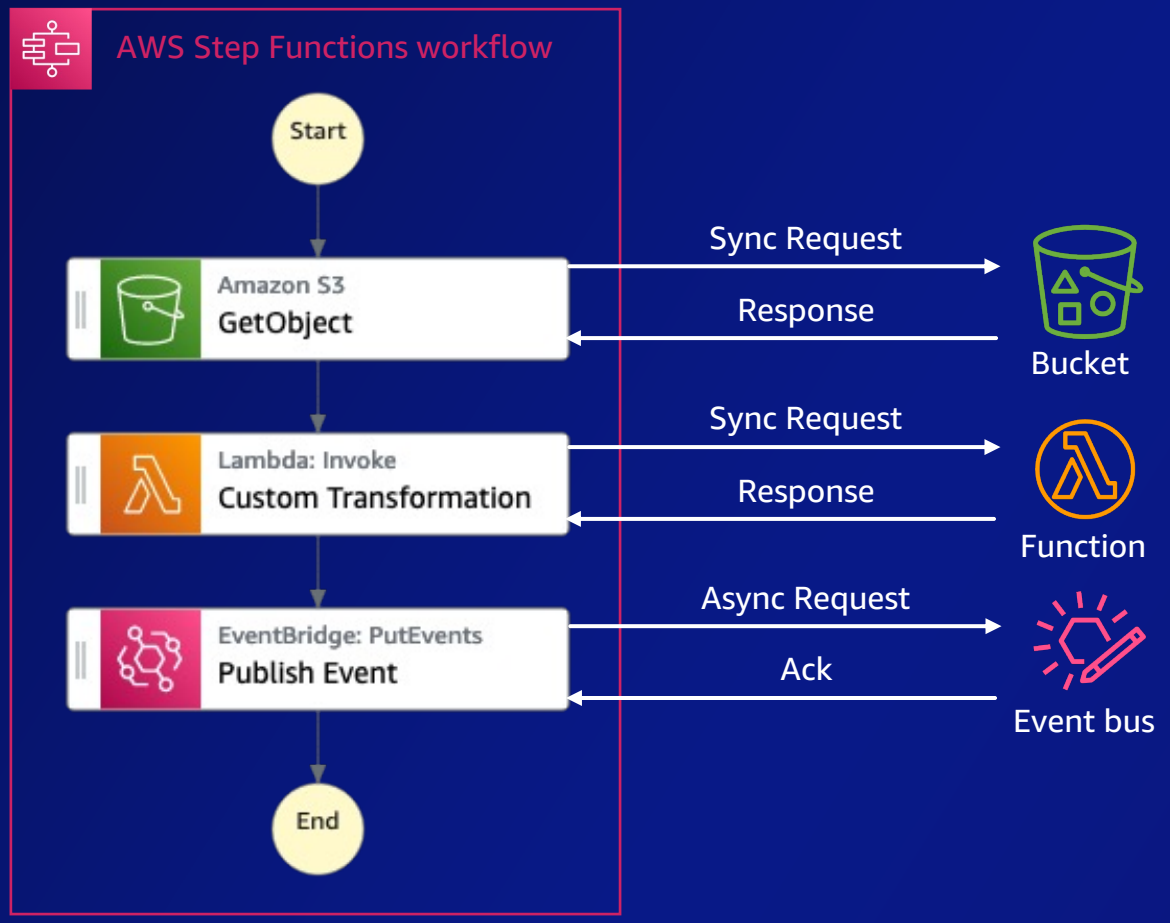
SERVICE INTEGRATION PATTERN



- Amazon EMR
- AWS Lambda
- Amazon SQS
- AWS Glue
- Amazon ECS/ AWS Fargate
- Amazon SNS
- Amazon Athena
- Amazon EKS
- AWS Step Functions
- Amazon SageMaker
- AWS Batch
- Amazon EventBridge
- Amazon DynamoDB
- AWS SDK
- Amazon API Gateway

Request Response

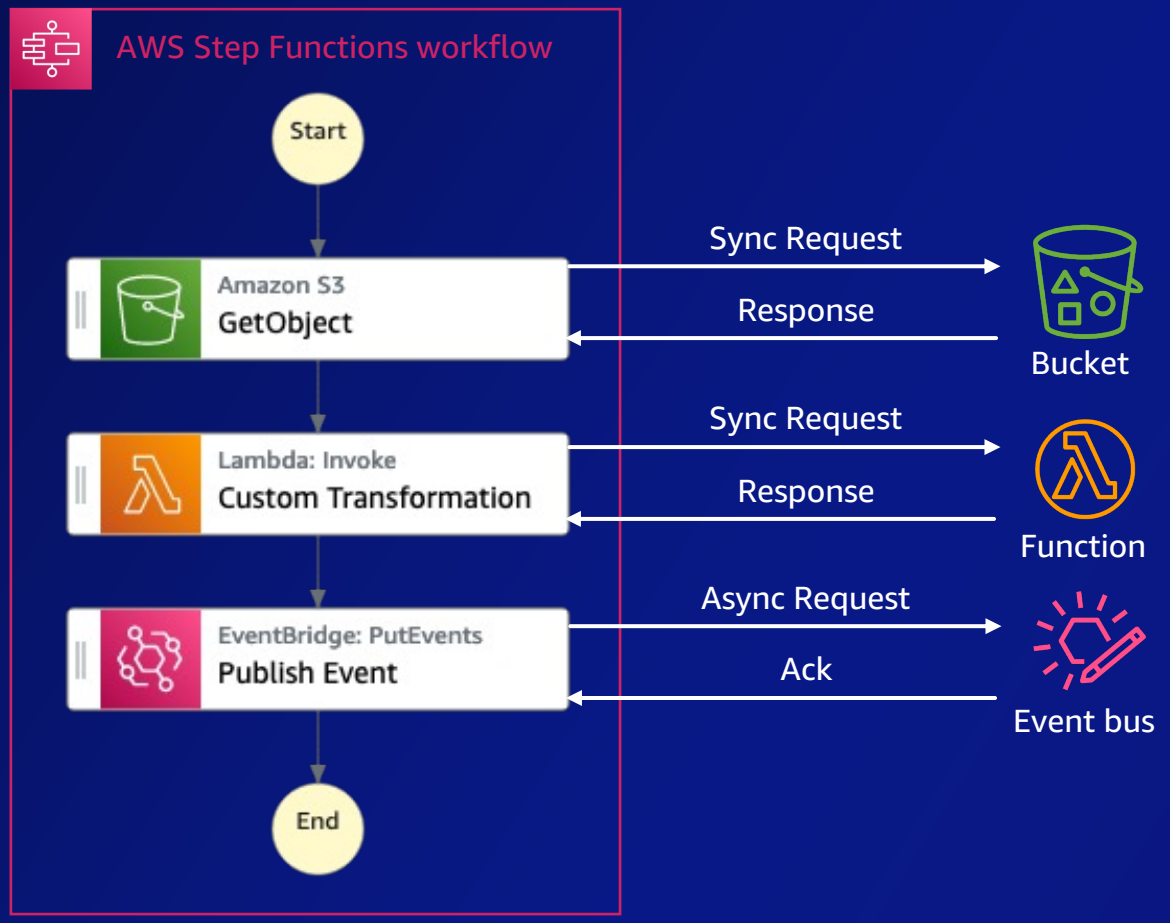
SERVICE INTEGRATION PATTERN



- Amazon EMR
- AWS Lambda
- Amazon SQS
- AWS Glue
- Amazon ECS/ AWS Fargate
- Amazon SNS
- Amazon Athena
- Amazon EKS
- AWS Step Functions
- Amazon SageMaker
- AWS Batch
- Amazon EventBridge
- Amazon DynamoDB
- AWS SDK
- Amazon API Gateway

Request Response

SERVICE INTEGRATION PATTERN



Amazon EMR

AWS Lambda

Amazon SQS

AWS Glue

Amazon ECS/
AWS Fargate

Amazon SNS

Amazon Athena

Amazon EKS

AWS Step Functions

Amazon SageMaker

AWS Batch

Amazon EventBridge

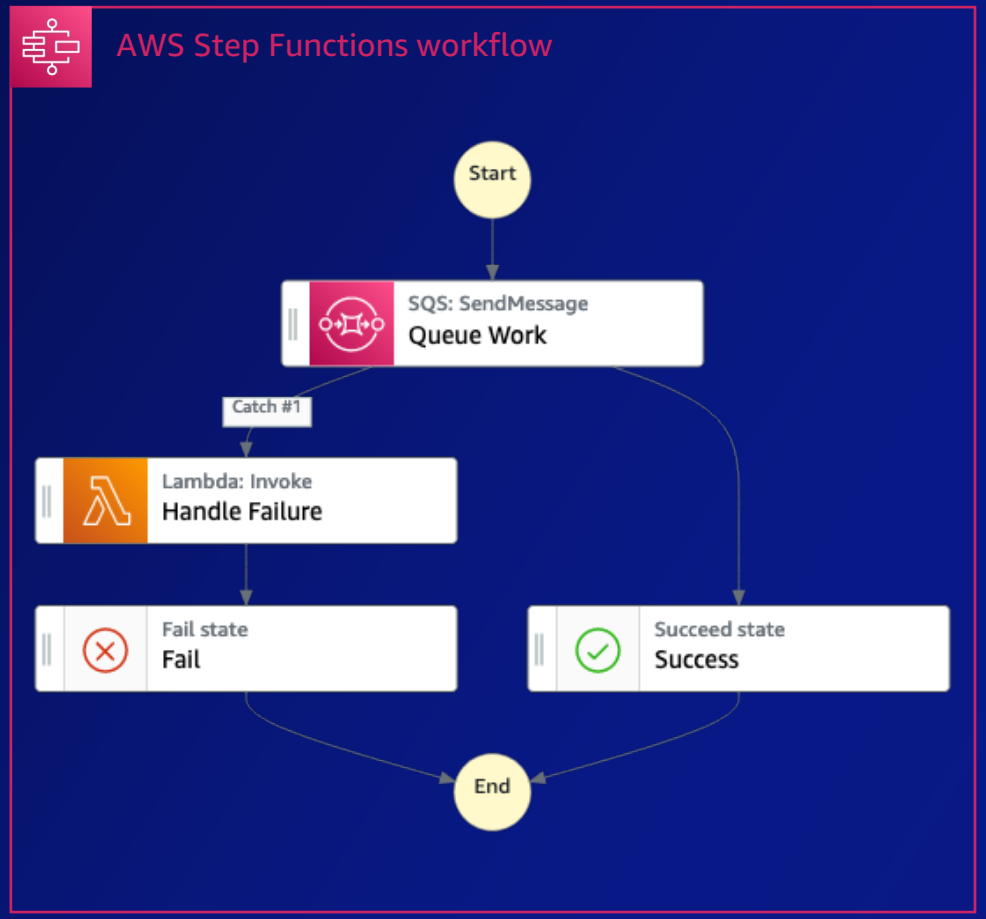
Amazon DynamoDB

AWS SDK

Amazon API Gateway

Wait for a Callback (.waitForTaskToken)

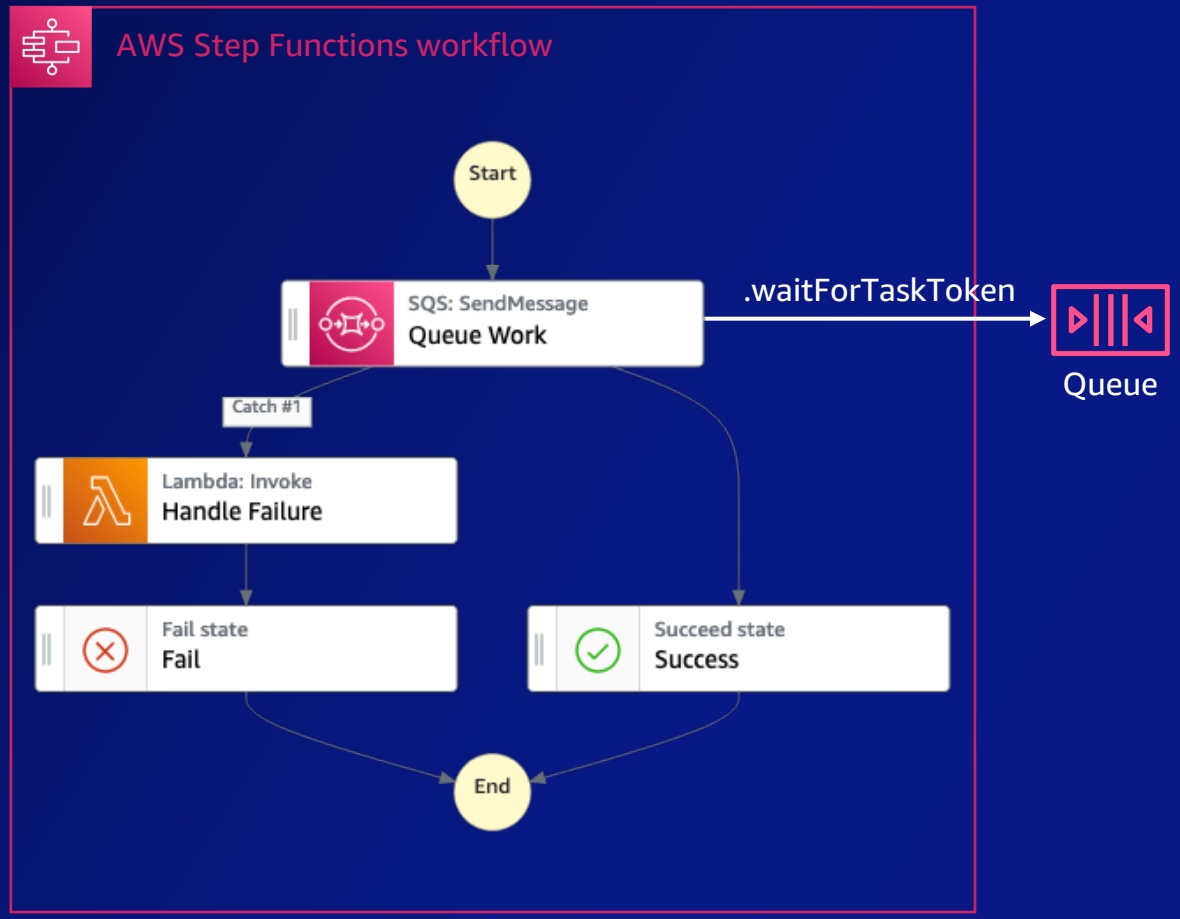
SERVICE INTEGRATION PATTERN



- AWS Lambda
- Amazon SQS
- Amazon ECS/ AWS Fargate
- Amazon SNS
- AWS Step Functions
- Amazon EventBridge
- AWS SDK
- Amazon API Gateway

Wait for a Callback (.waitForTaskToken)

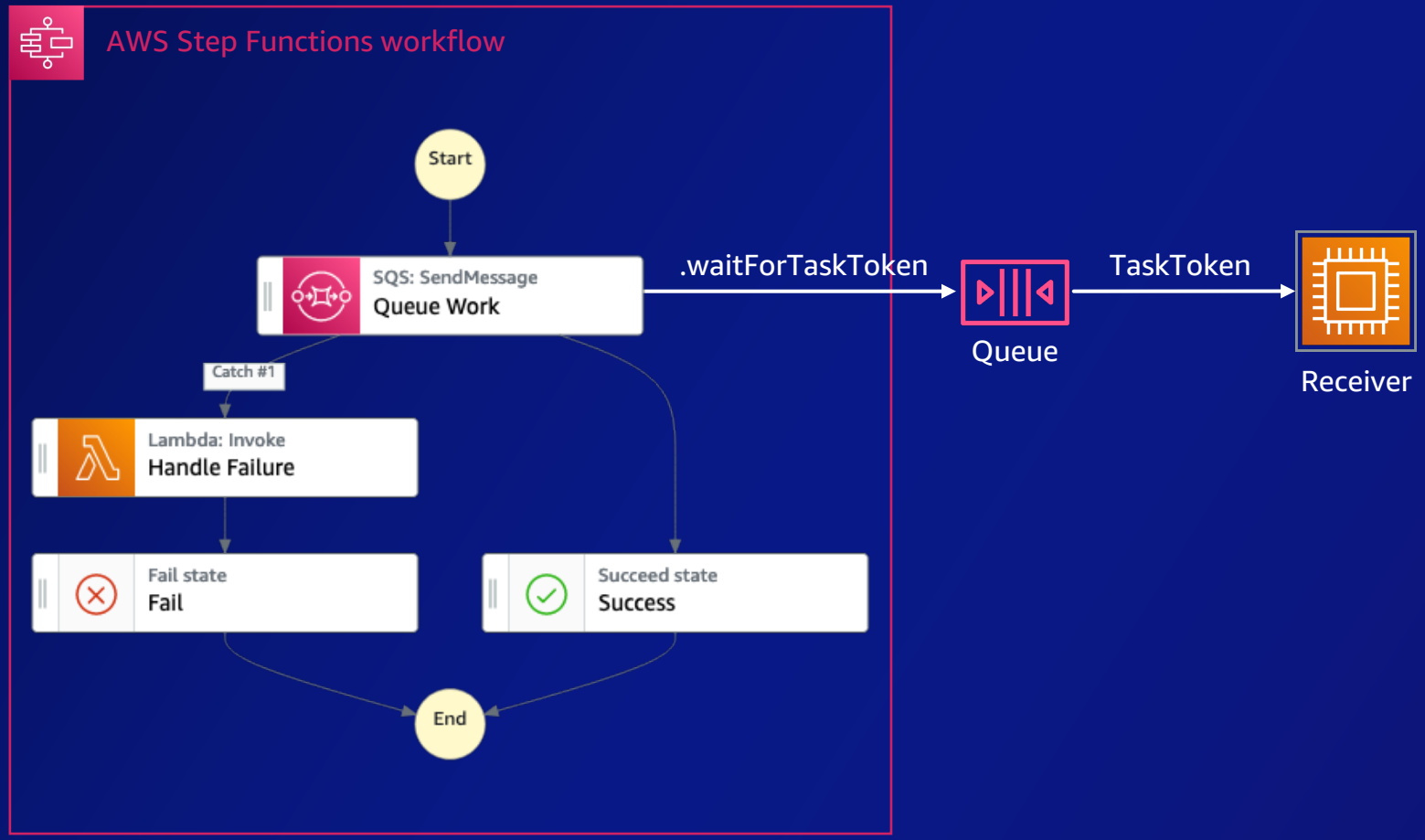
SERVICE INTEGRATION PATTERN



- AWS Lambda
- Amazon SQS
- Amazon ECS/ AWS Fargate
- Amazon SNS
- AWS Step Functions
- Amazon EventBridge
- AWS SDK
- Amazon API Gateway

Wait for a Callback (.waitForTaskToken)

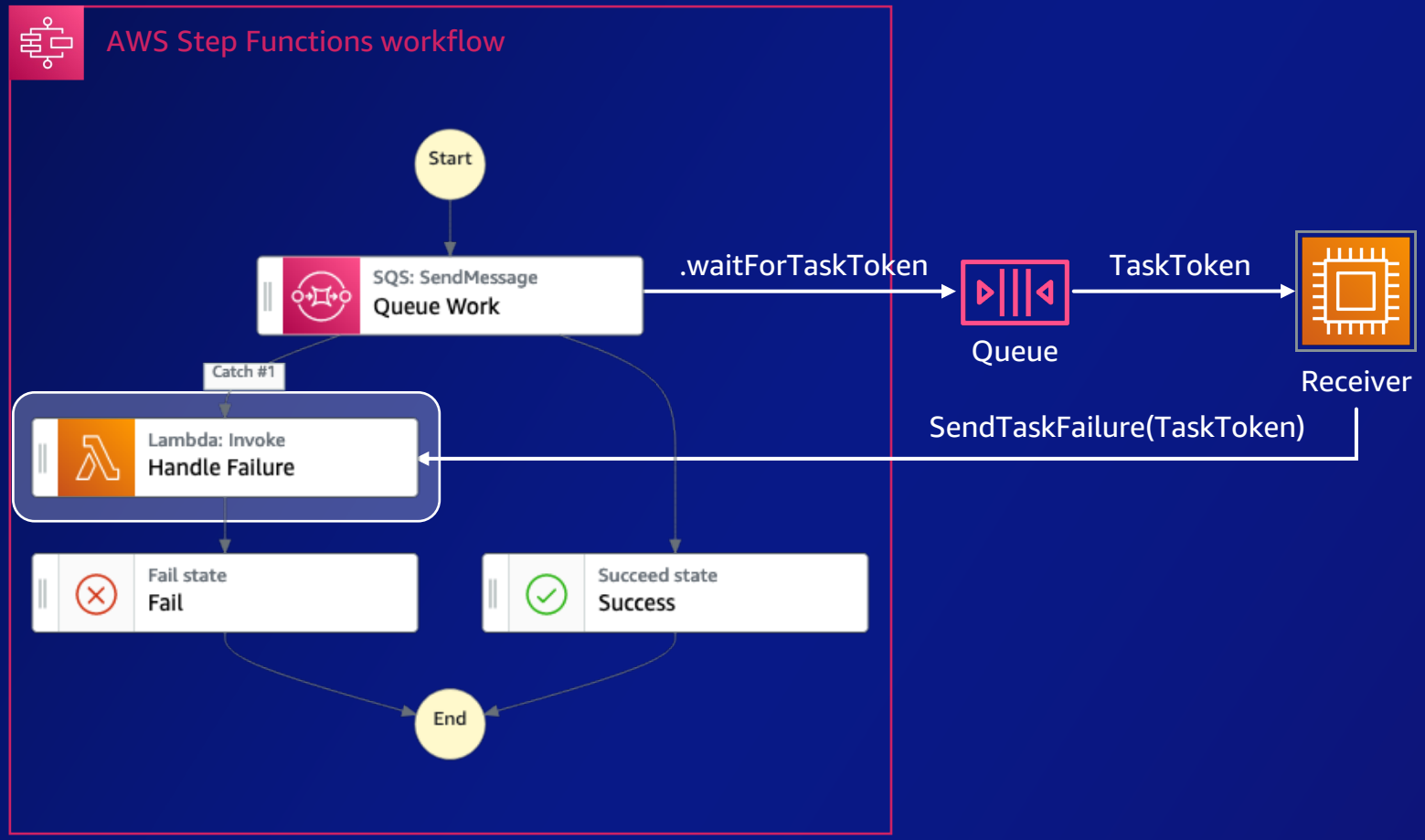
SERVICE INTEGRATION PATTERN



- AWS Lambda
- Amazon SQS
- Amazon ECS/ AWS Fargate
- Amazon SNS
- AWS Step Functions
- Amazon EventBridge
- AWS SDK
- Amazon API Gateway

Wait for a Callback (.waitForTaskToken)

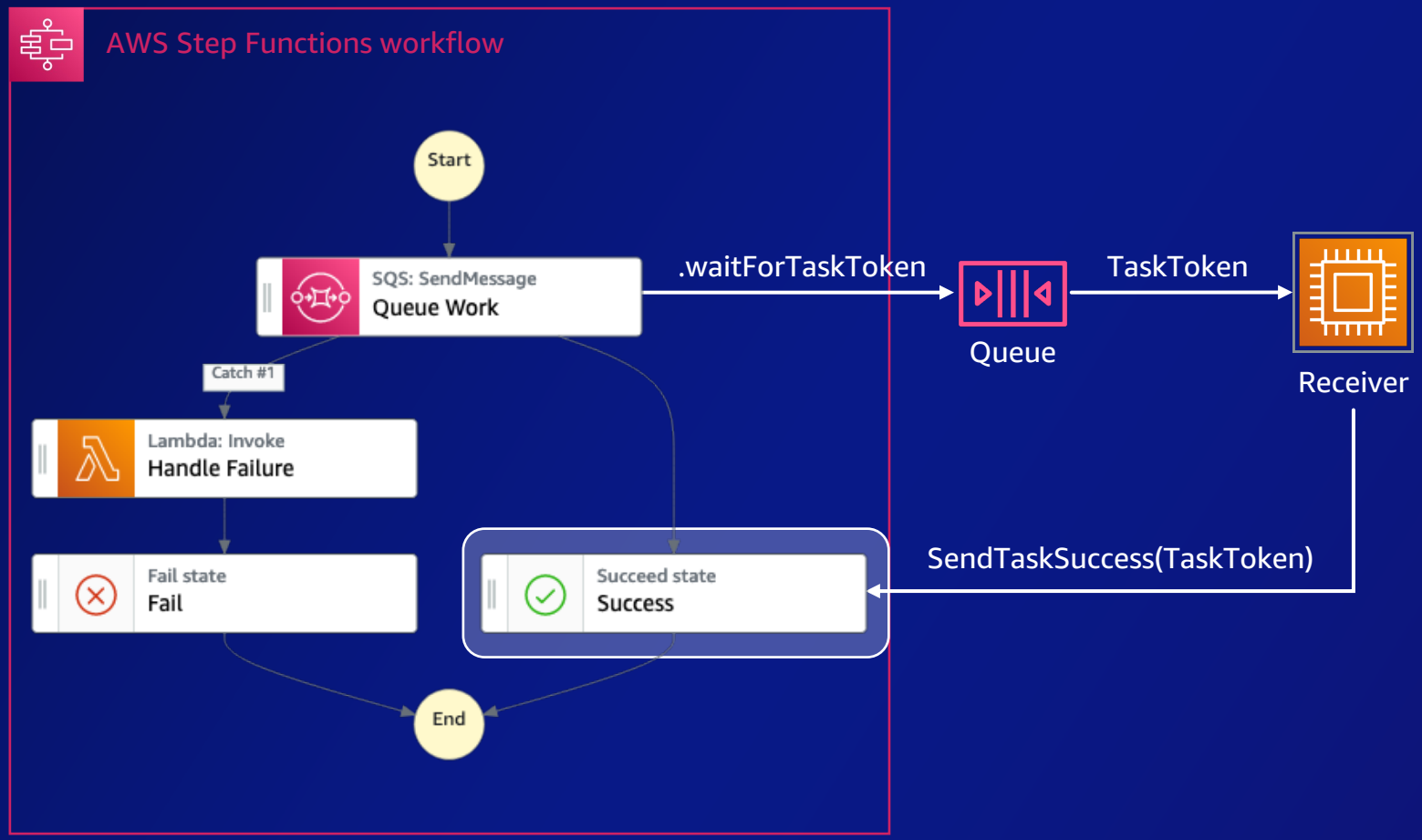
SERVICE INTEGRATION PATTERN



- AWS Lambda
- Amazon SQS
- Amazon ECS/ AWS Fargate
- Amazon SNS
- AWS Step Functions
- Amazon EventBridge
- AWS SDK
- Amazon API Gateway

Wait for a Callback (.waitForTaskToken)

SERVICE INTEGRATION PATTERN




AWS Lambda


Amazon SQS


Amazon ECS/
AWS Fargate


Amazon SNS


AWS Step Functions

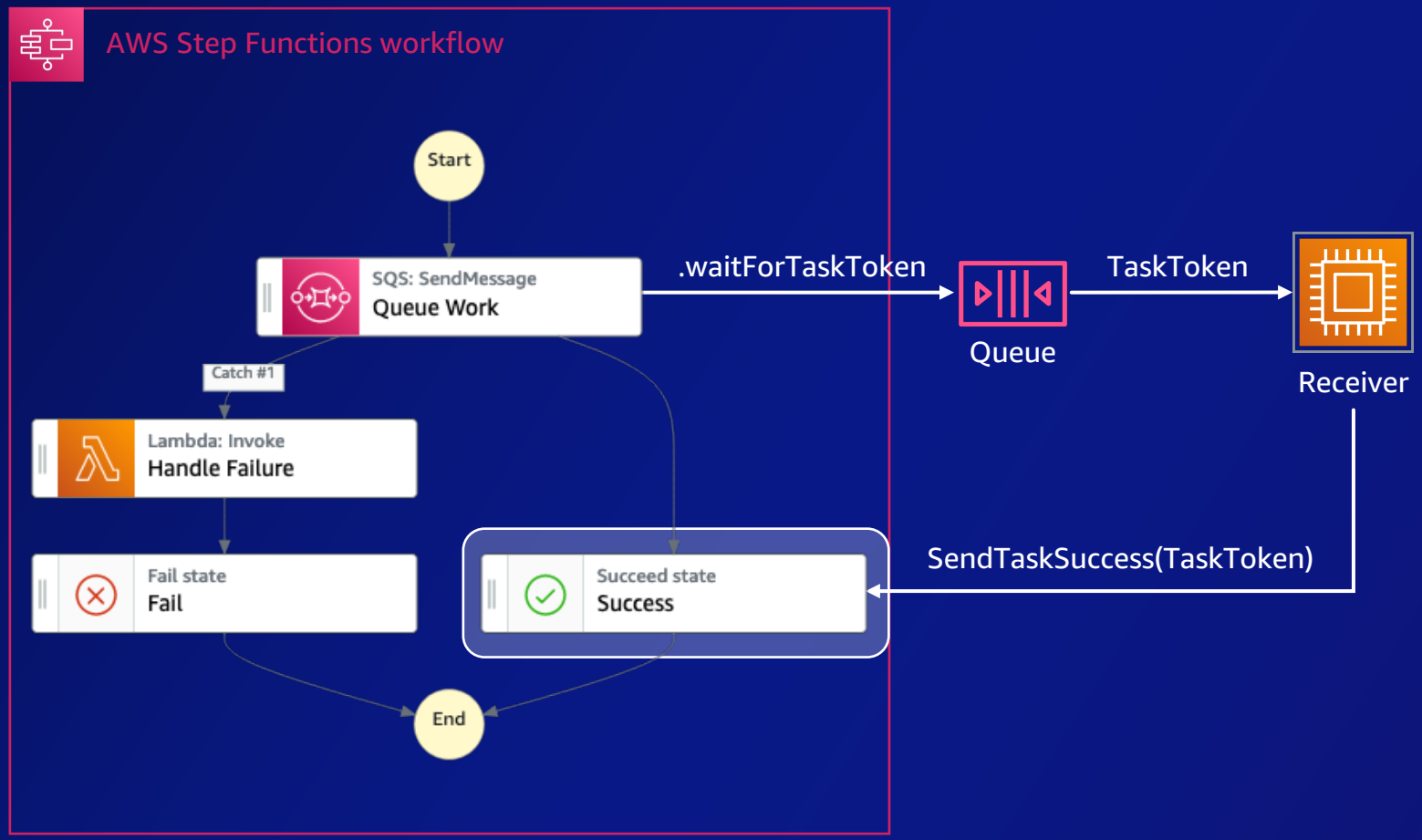

Amazon EventBridge


AWS SDK


Amazon API Gateway

Wait for a Callback (.waitForTaskToken)

SERVICE INTEGRATION PATTERN



AWS Lambda

Amazon SQS

Amazon ECS/
AWS Fargate

Amazon SNS

AWS Step Functions

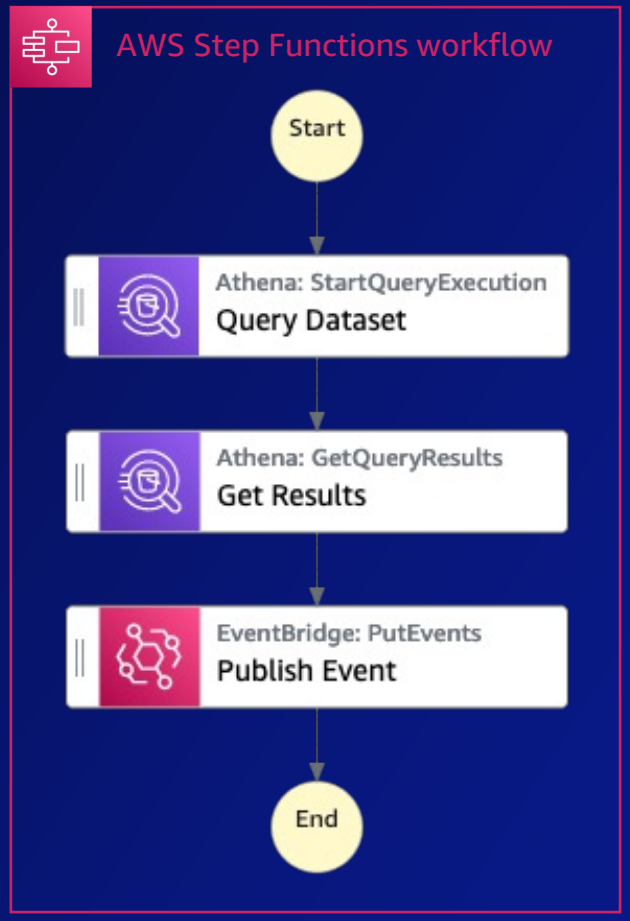
Amazon EventBridge

AWS SDK

Amazon API Gateway

Run a Job (.sync)

SERVICE INTEGRATION PATTERN



Amazon EMR

AWS Glue

Amazon ECS/AWS Fargate

Amazon Athena

Amazon EKS

AWS Step Functions

Amazon SageMaker

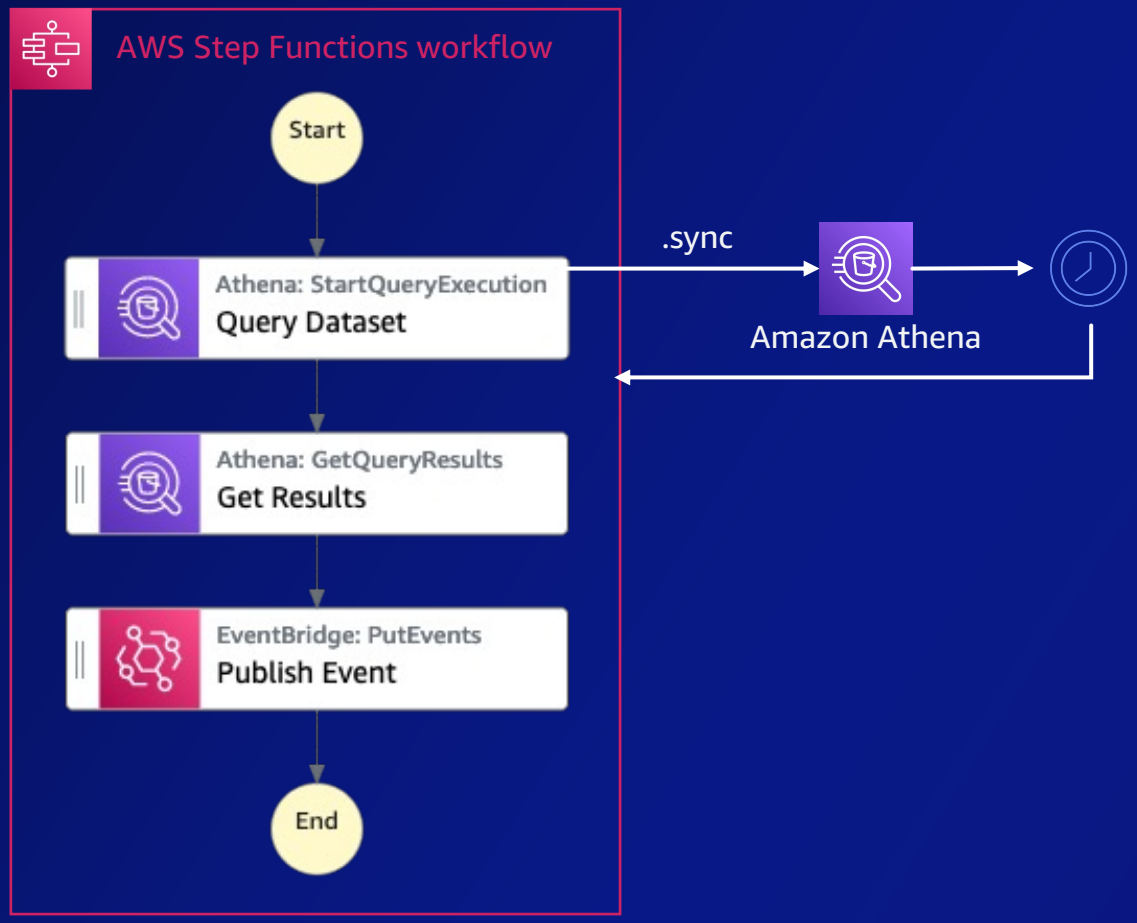
AWS Batch

AWS CodeBuild

Amazon API Gateway

Run a Job (.sync)

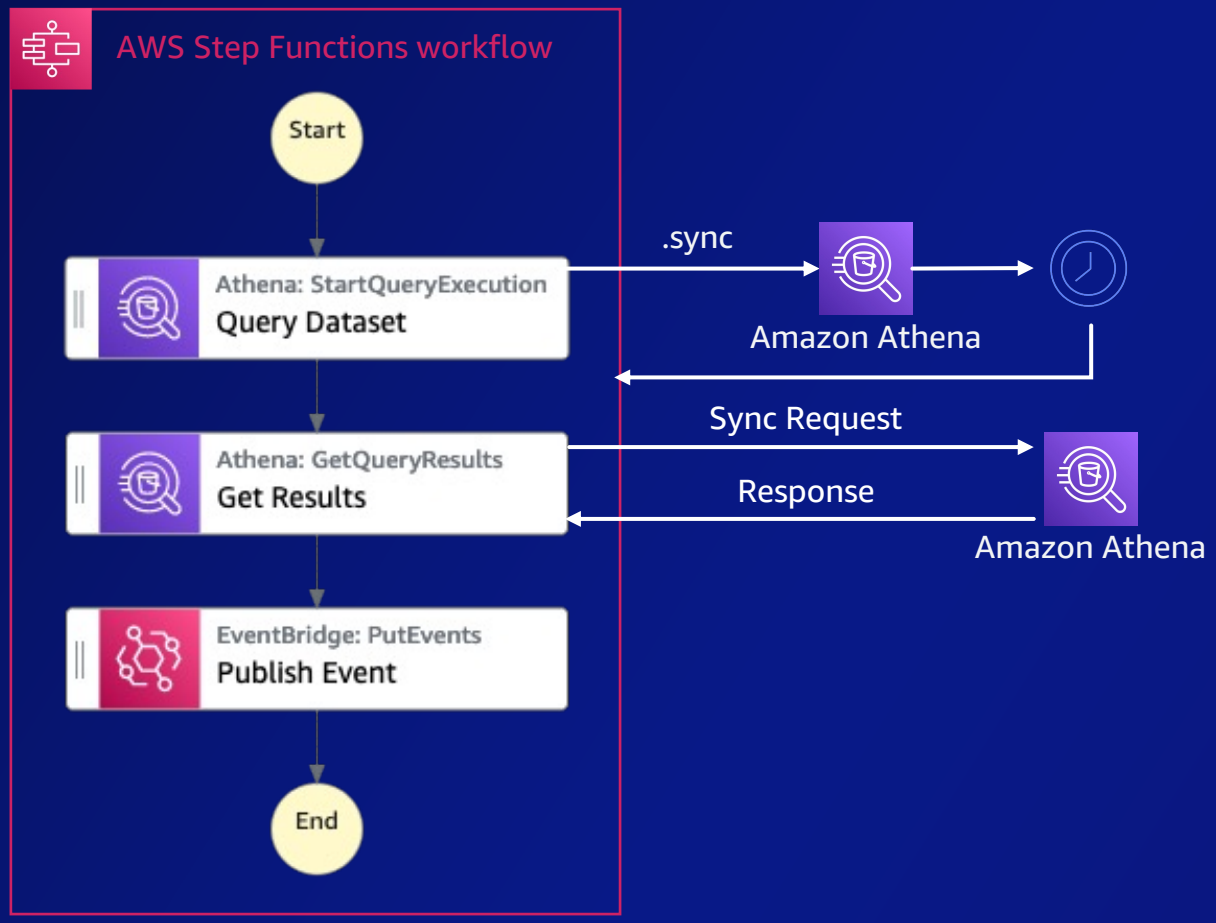
SERVICE INTEGRATION PATTERN



- Amazon EMR
- AWS Glue
- Amazon ECS/AWS Fargate
- Amazon Athena
- Amazon EKS
- AWS Step Functions
- Amazon SageMaker
- AWS Batch
- AWS CodeBuild
- Amazon API Gateway

Run a Job (.sync)

SERVICE INTEGRATION PATTERN



Amazon EMR

AWS Glue

Amazon ECS/AWS Fargate

Amazon Athena

Amazon EKS

AWS Step Functions

Amazon SageMaker

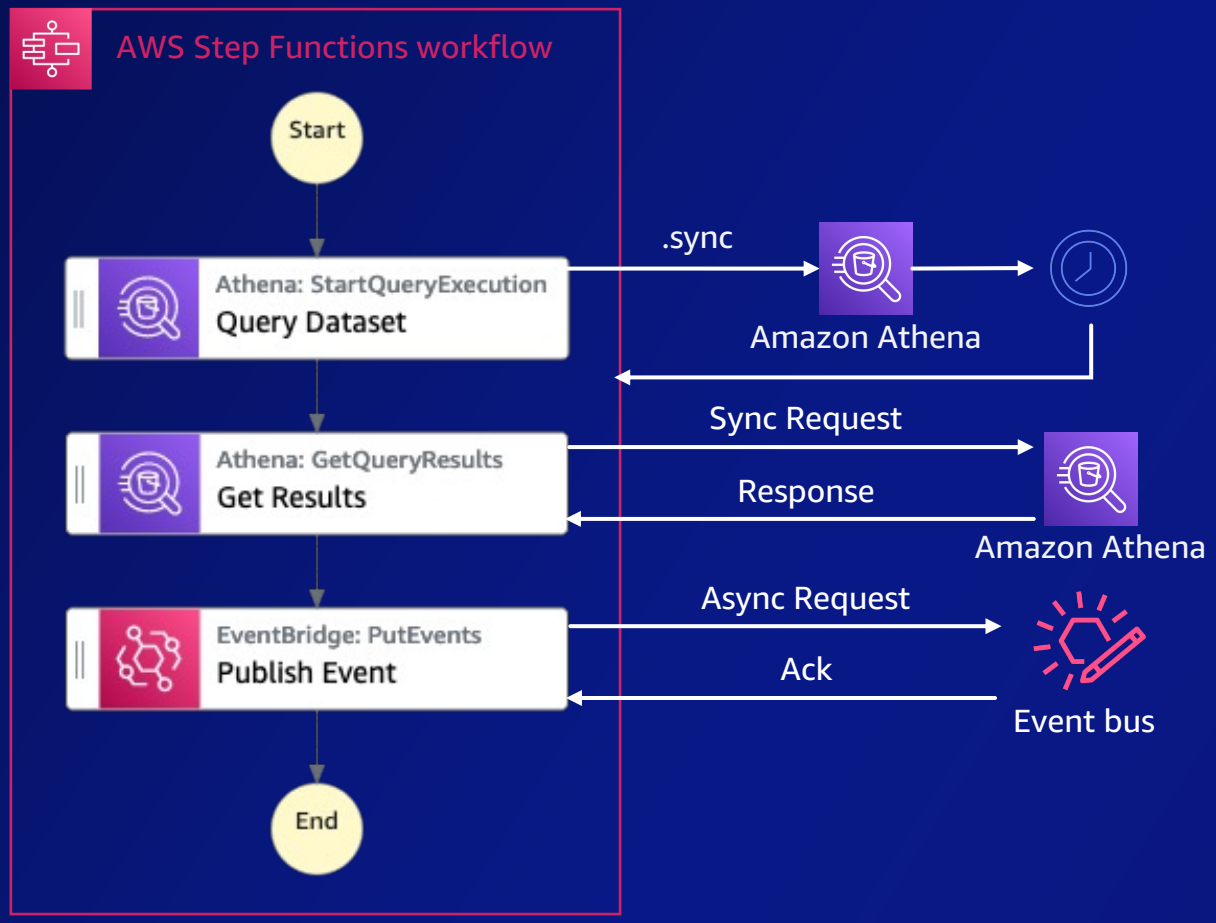
AWS Batch

AWS CodeBuild

Amazon API Gateway

Run a Job (.sync)

SERVICE INTEGRATION PATTERN



Amazon EMR

AWS Glue

Amazon ECS/AWS Fargate

Amazon Athena

Amazon EKS

AWS Step Functions

Amazon SageMaker

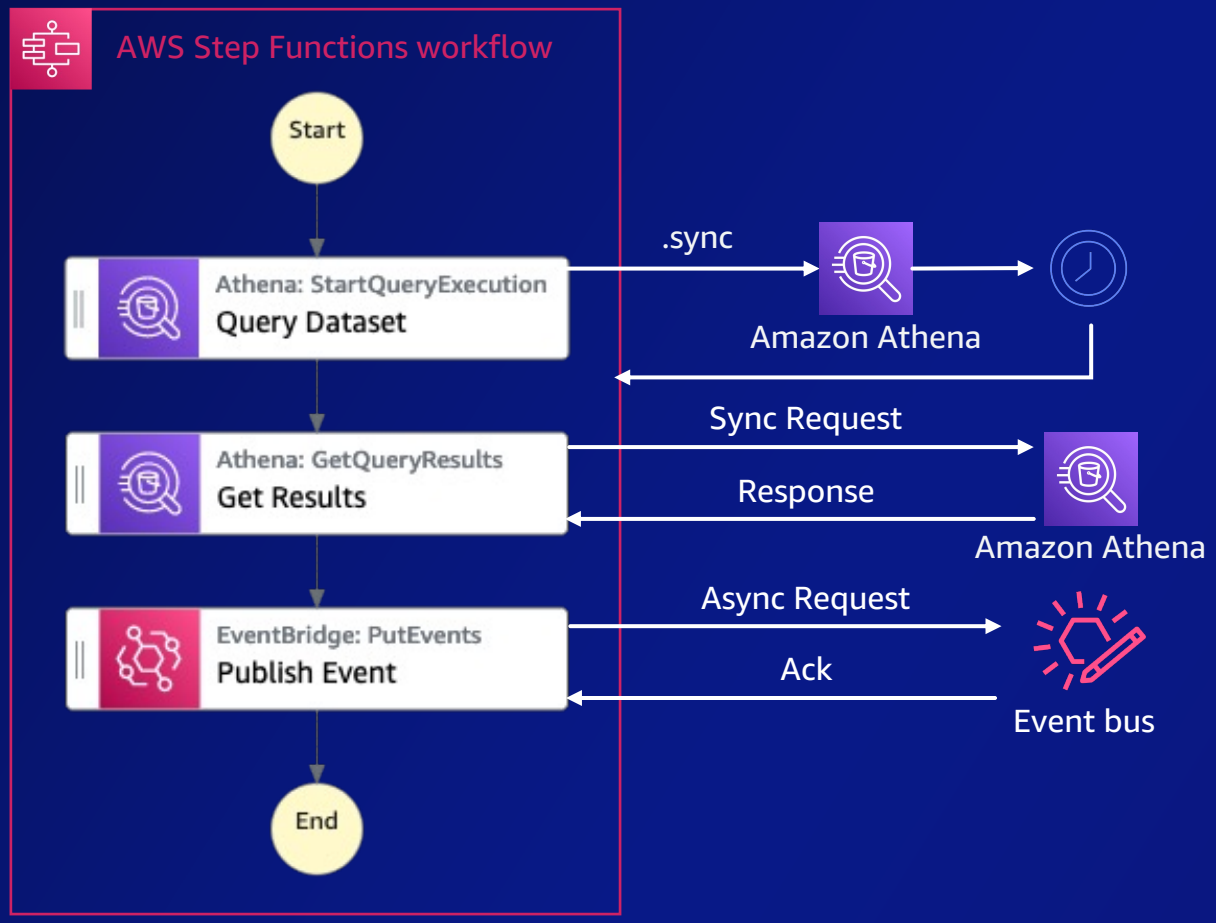
AWS Batch

AWS CodeBuild

Amazon API Gateway

Run a Job (.sync)

SERVICE INTEGRATION PATTERN



Amazon EMR

AWS Glue

Amazon ECS/AWS Fargate

Amazon Athena

Amazon EKS

AWS Step Functions

Amazon SageMaker

AWS Batch

AWS CodeBuild

AWS SDK

Amazon API Gateway

Events enable interaction between services

Managed services provide routing, storage, and distribution of events

Amazon SQS

Messaging

Durable and scalable
Fully managed
Comprehensive security

Amazon EventBridge

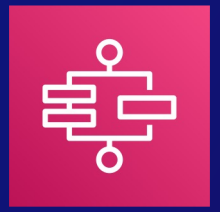
Choreography

Event filtering
Managed and scalable
SaaS integration

Amazon SNS

Eventing

Performance at scale
Fully managed
Enterprise-ready

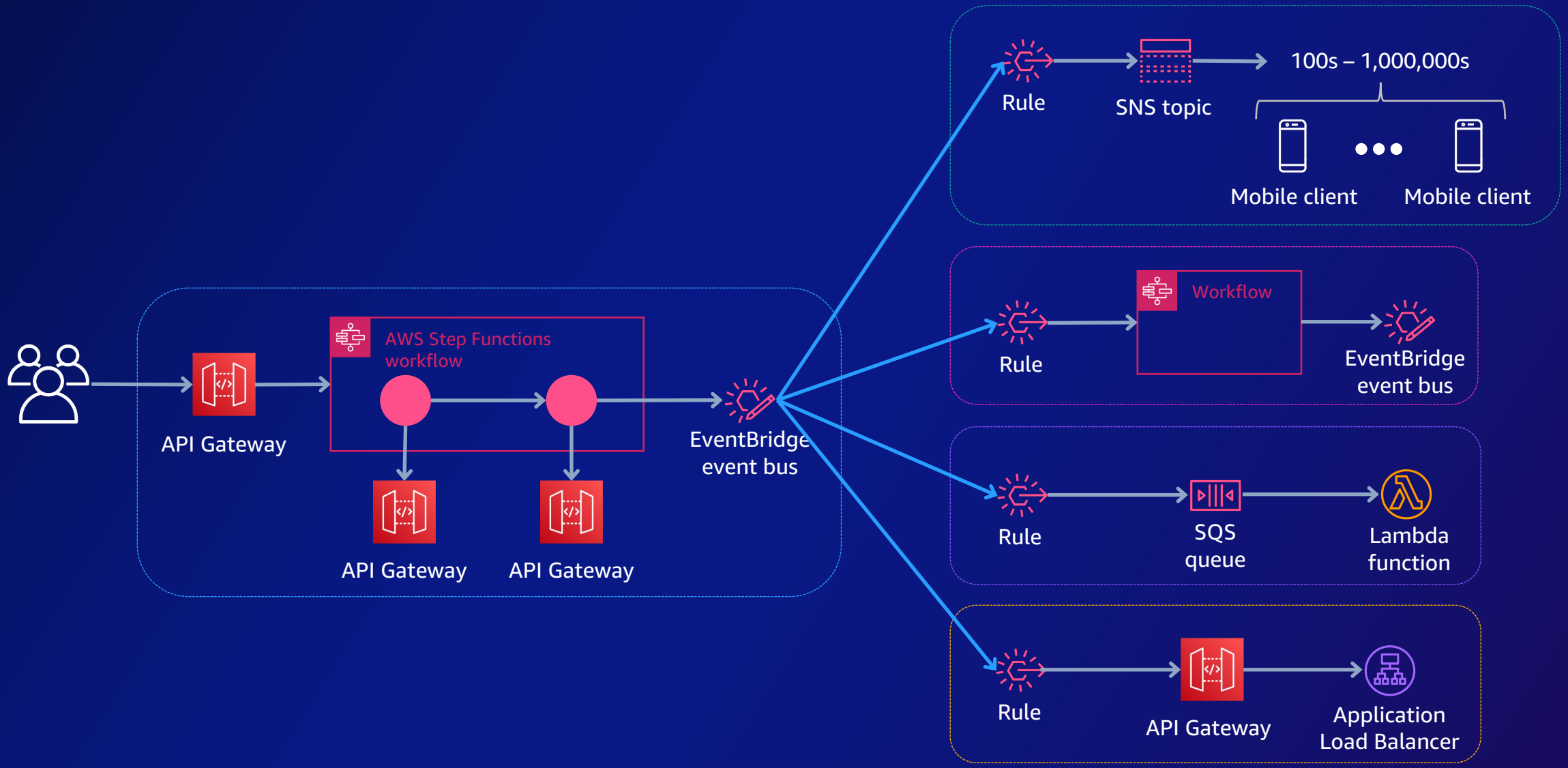


AWS Step Functions

Orchestration

Sequencing
Parallel execution
State management

Better together: Orchestration + choreography



Choose your pattern and go build!

Continue your learning

Serverless Land

AWS Skills Builder

The AWS Step Functions Workshop

Serverlesspresso Workshop

Building an event-driven application with Amazon EventBridge

AWS Step Functions Workflows Collection

Build workflows visually with AWS Step Functions Workflow Studio

Run event-driven workflows with Amazon EKS and AWS Step Functions

Introducing new intrinsic functions for AWS Step Functions

Rebekah Kulidzan

@rkulidzan

linkedin.com/in/rebekahkulidzan/

bek@amazon.com