
AWS
re:Invent
NOV. 27 – DEC. 1, 2023 | LAS VEGAS, NV

API309

Advanced integration patterns & trade-offs for loosely coupled systems

Dirk Fröhner
Principal Solutions Architect
Amazon Web Services

Gregor Hohpe
Senior Principal Evangelist, Serverless
Amazon Web Services

What does an architect see?

Photo by Dirk Fröhner

Legacy modernization!

Photo by Dirk Fröhner

Breaking down the monolith

Photo by Dirk Fröhner

Coupling and cohesion

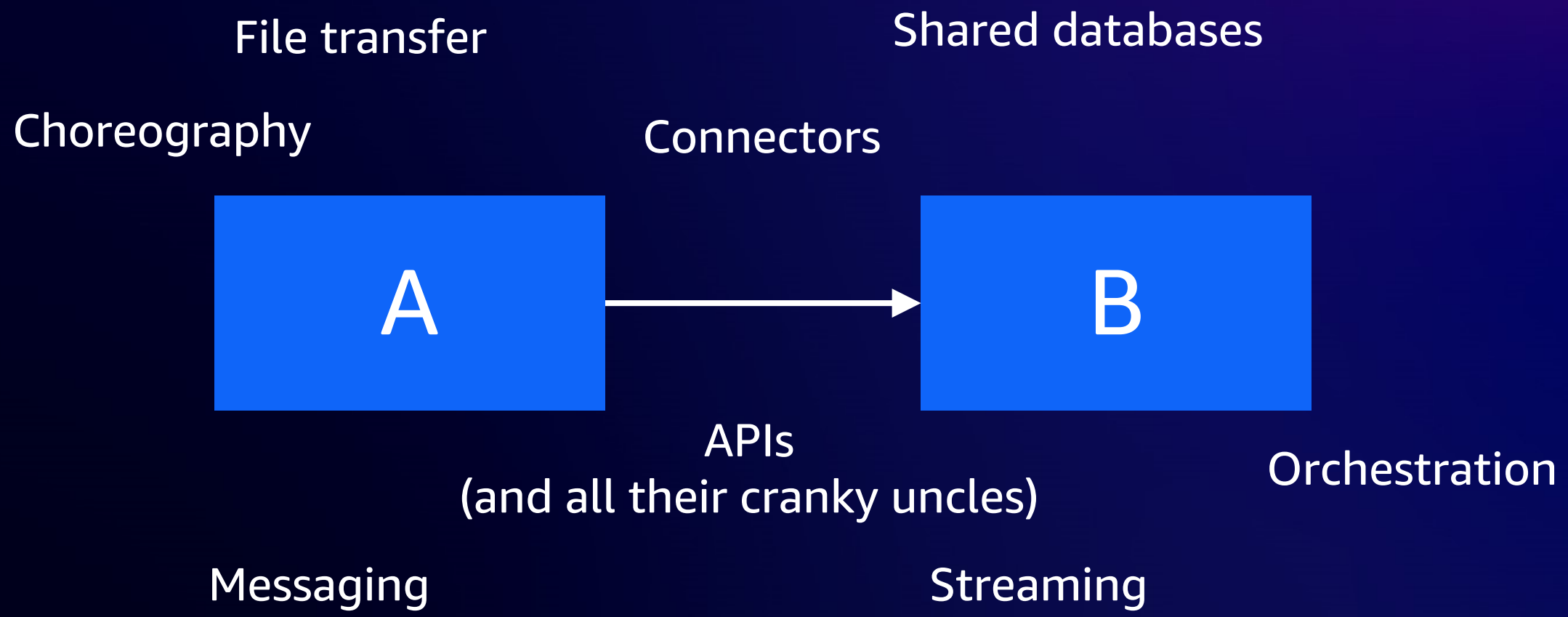
Photo by Dirk Fröhner

In modern cloud applications, integration isn't an afterthought. It's an integral part of the application architecture and the software delivery lifecycle.

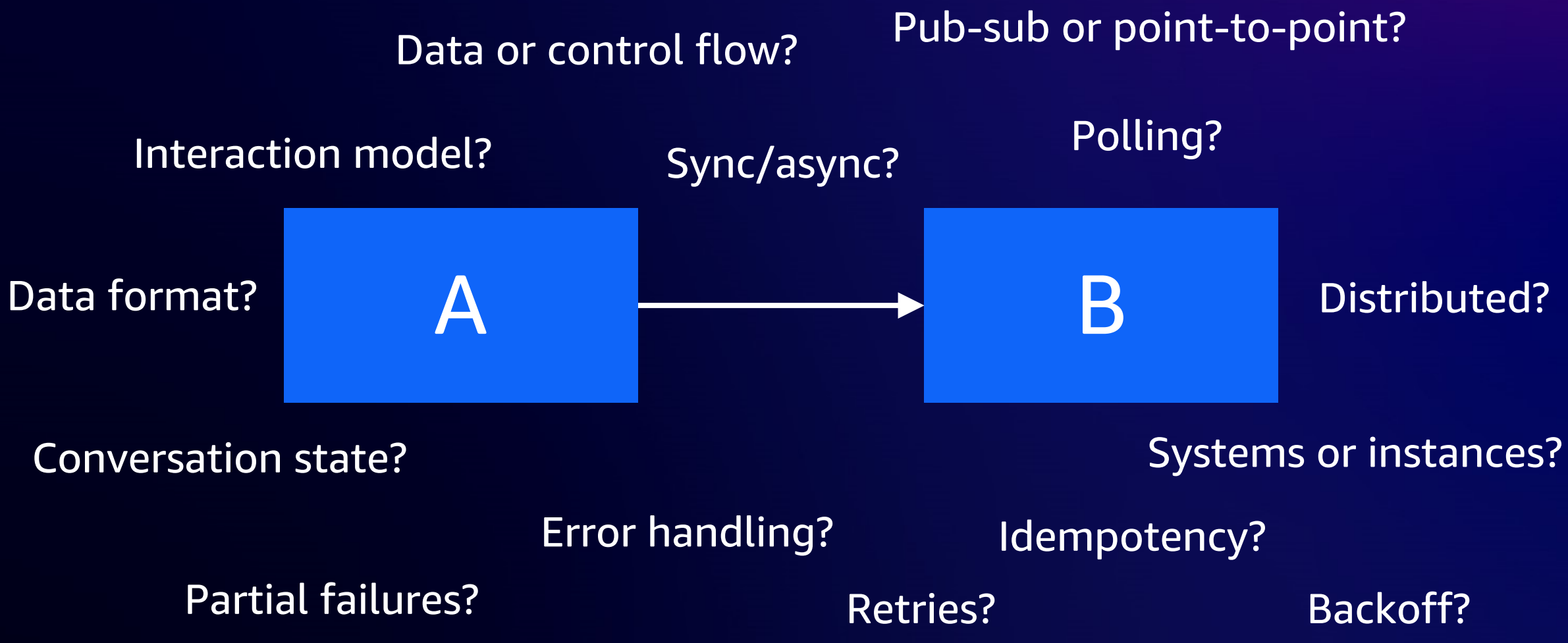
Gregor

Appligration Architect

Many ways to skin an integration scenario



Connecting two systems – How hard can it be?



No architecture decision you take comes **without trade-offs**. **Every** architecture decision brings some **pain**.

Your job as software architect is to identify the least painful option on the table.

Dirk
Trying to manage expectations

Integration versus distributed systems

Integration? Distributed system?

Spanning teams, time, and control

Approach	Level of control	Delivery lifecycle	Team	Tool (indicative)
Migration	Low	One time	One off	
Data synchronization/ traditional integration	Low	Long	Dedicated	
Enterprise service bus	Some	Slower than component development	Likely dedicated	
Distributed cloud applications	High	Same as component development	Embedded	

Spanning teams, time, and control

Approach	Level of control	Delivery lifecycle	Team	Tool (indicative)
Migration	Low	One time	One off	Amazon AppFlow
Data synchronization/ traditional integration	Low	Long	Dedicated	Amazon AppFlow
Enterprise service bus	Some	Slower than component development	Likely dedicated	Amazon MQ
Distributed cloud applications	High	Same as component development	Embedded	Amazon EventBridge, AWS Lambda Destinations

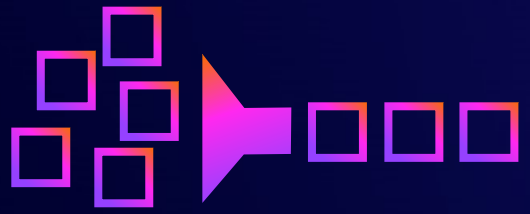
Integration differs from building distributed systems not by technology but by lifecycle, team structure, and level of control.

Gregor

Co-author, *Enterprise Integration Patterns*

Integration patterns and distributed system fundamentals

Distributed system fundamentals



Coupling

Dependencies and change propagation between components

Control flow and flow control

Pushing versus pulling, load shaping

Message order and delivery semantics

FIFO, Pub-Sub, Competing Consumers, At-least-once delivery

Error handling and replays

Archiving, idempotency, deduplication

Distributed system fundamentals

Coupling

Coupling – Integration's magic word

Coupling is a measure of independent variability between connected systems

Decoupling has a cost, both at design and runtime

Coupling isn't binary

Coupling – Many dimensions

- Technology dependency: Java versus C++
- Location dependency: IP addresses, DNS
- Data format dependency: Binary, XML, JSON, ProtoBuf, Avro
- Data type dependency: int16, int32, string, UTF-8, null, empty
- Semantic dependency: Name, Middlename, ZIP
- Temporal dependency: sync, async
- Interaction style dependency: messaging, RPC, query-style (GraphQL)
- Conversation dependency: pagination, caching, retries

The appropriate level of (design-time) coupling depends on the level of control you have over the endpoints.

Gregor

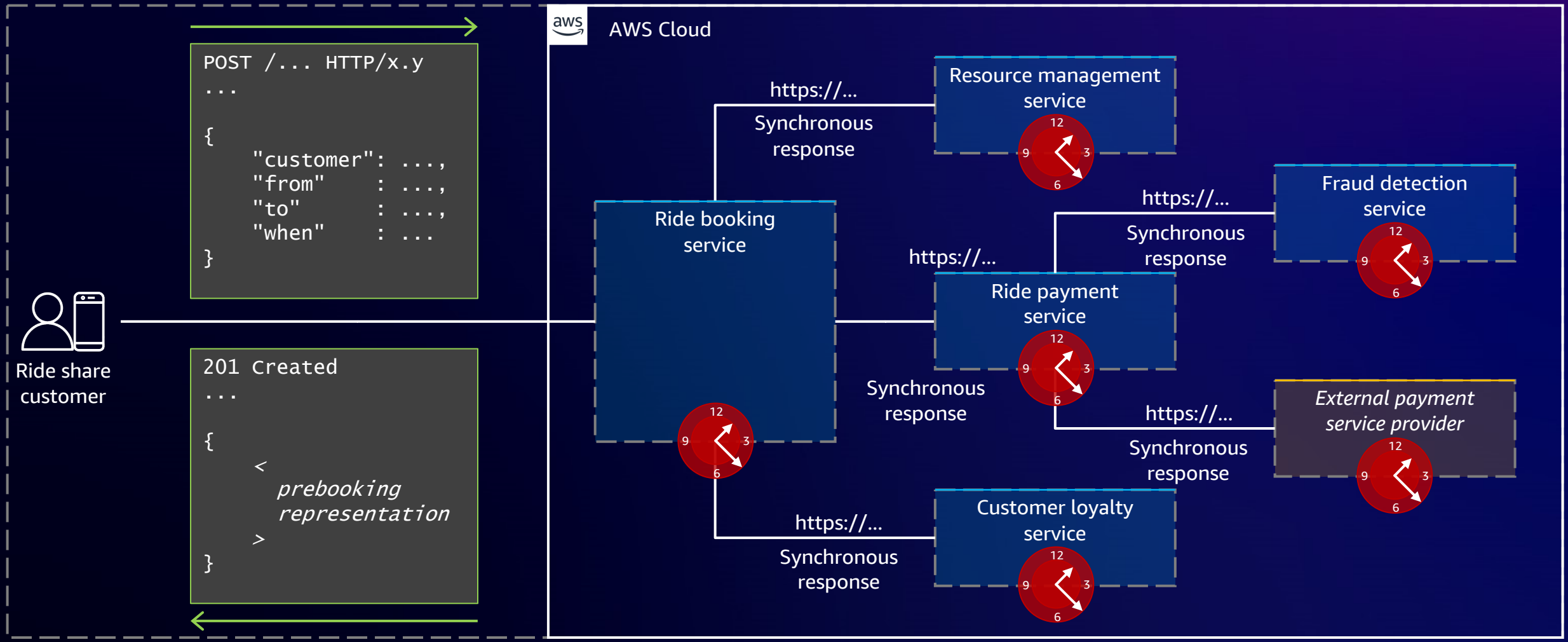
After two decades of struggling with it

Message queues decouple

- Logical channels decouple location
- Queuing decouples timing dependency
- Message passing decouples interaction style and data format dependencies

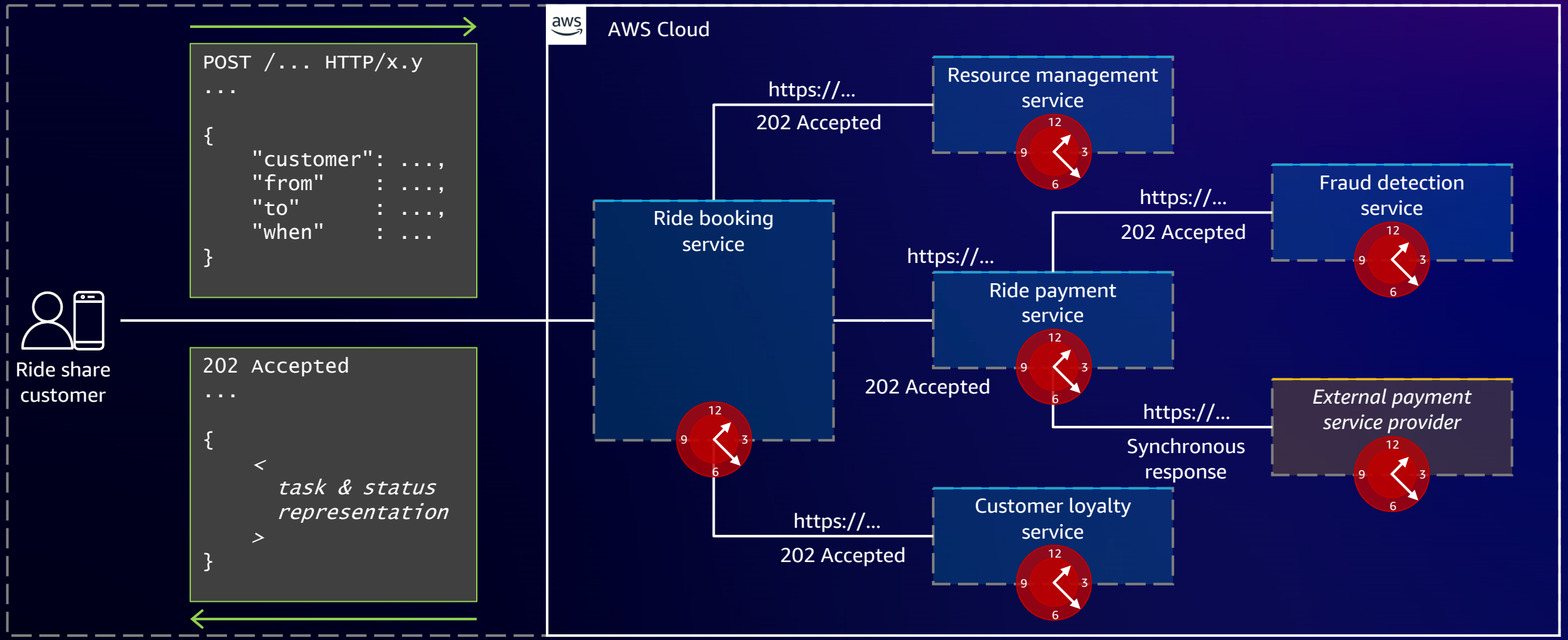
Synchronous communication (APIs)

RIDE SHARE – PREBOOKING CAMPAIGN



Asynchronous communication (APIs)

RIDE SHARE – PREBOOKING CAMPAIGN



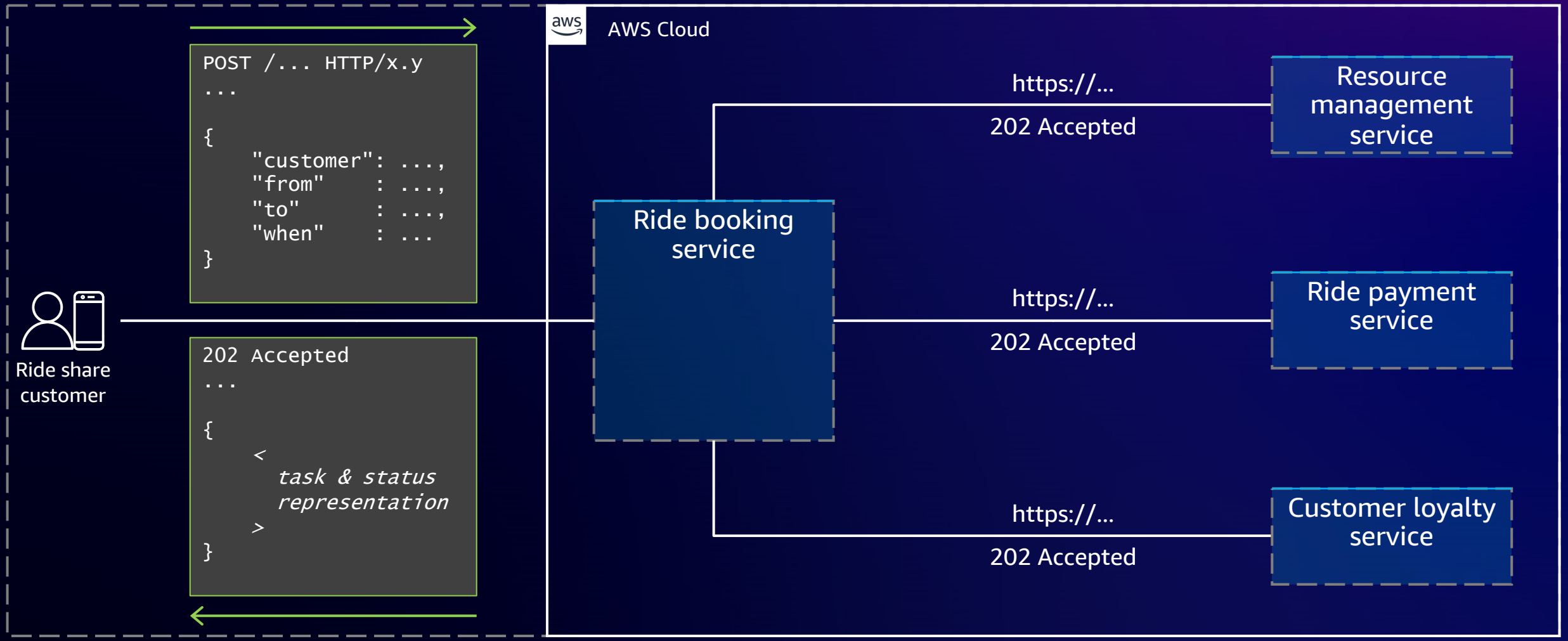
Friends don’t let friends rely on synchronous integration.

Dirk

Being friendly

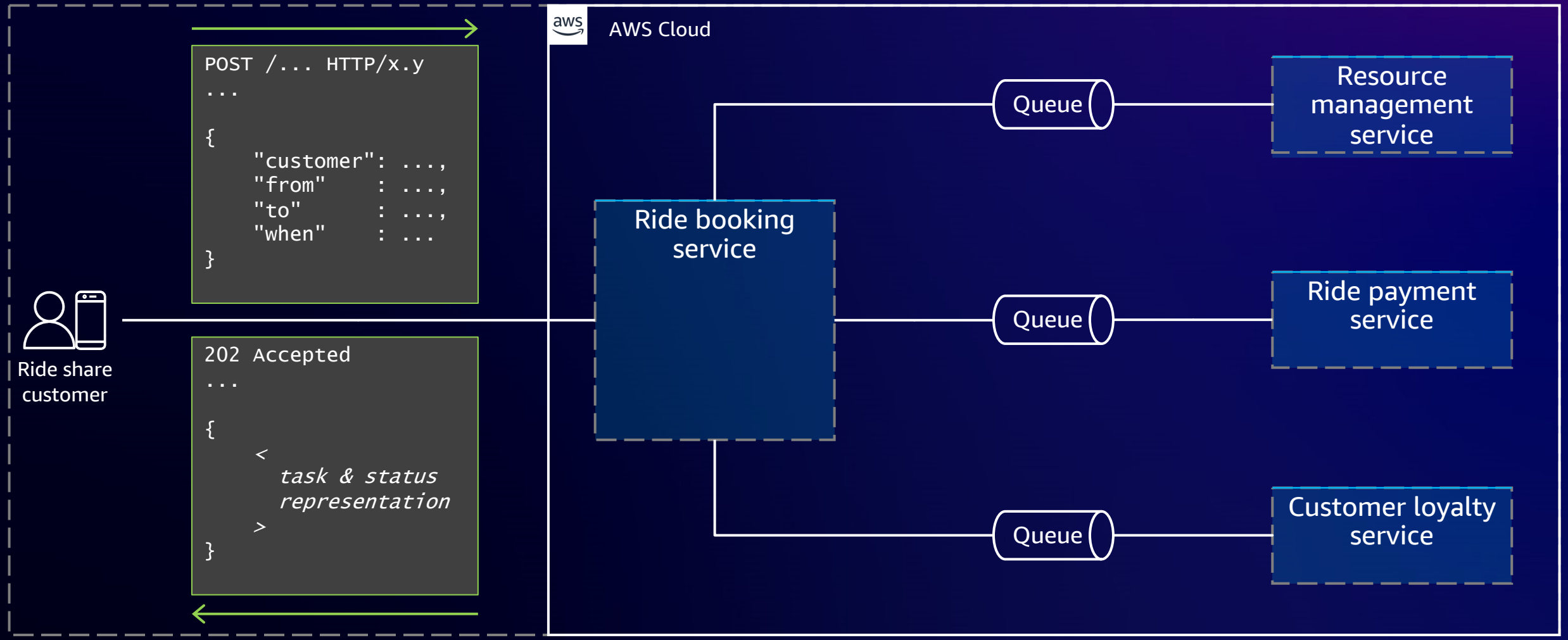
Asynchronous communication (APIs)

RIDE SHARE – PREBOOKING CAMPAIGN



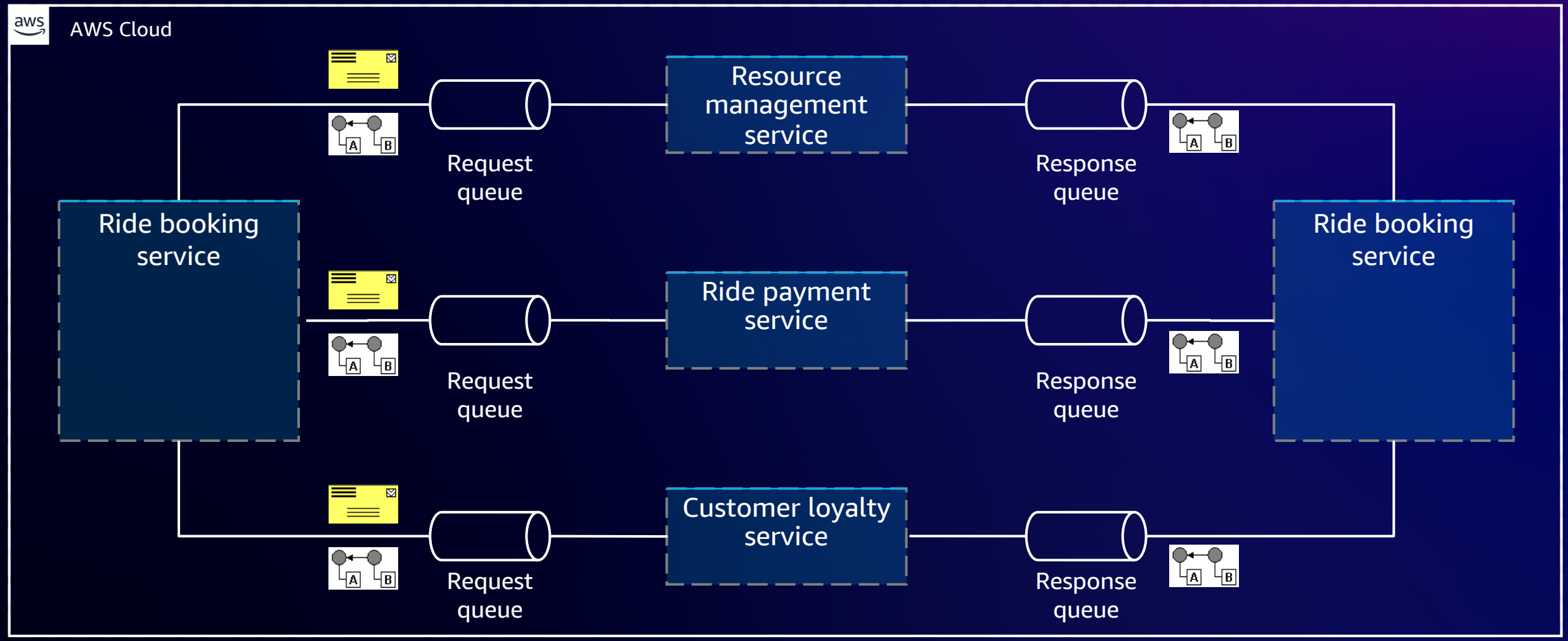
Asynchronous communication (messaging)

RIDE SHARE – PREBOOKING CAMPAIGN



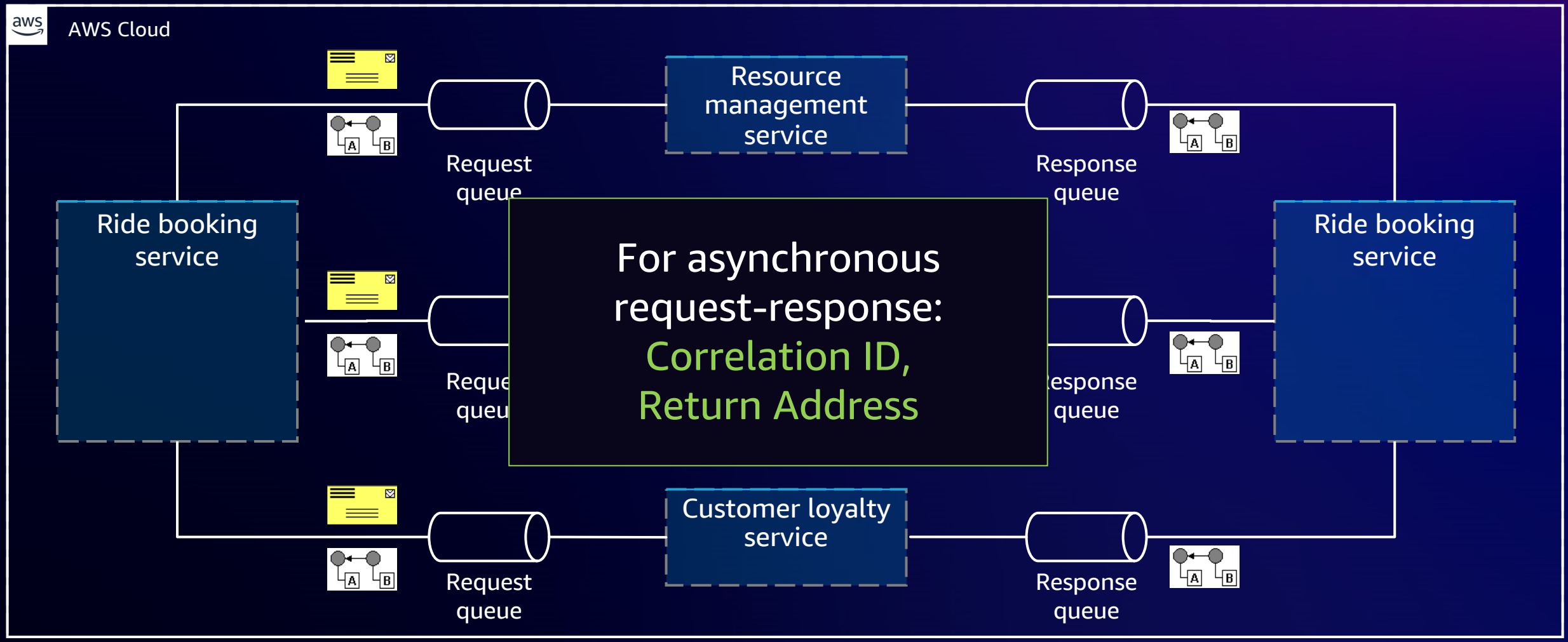
Asynchronous request-response (messaging)

RIDE SHARE – PREBOOKING CAMPAIGN



Asynchronous request-response (messaging)

RIDE SHARE – PREBOOKING CAMPAIGN



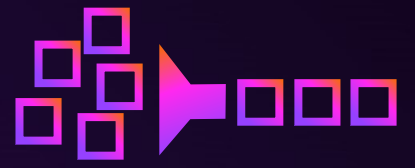
Loose coupling is ~~always~~ mostly better than lousy coupling.

Dirk

Being an armchair philosopher

Distributed system fundamentals

Control flow and flow control

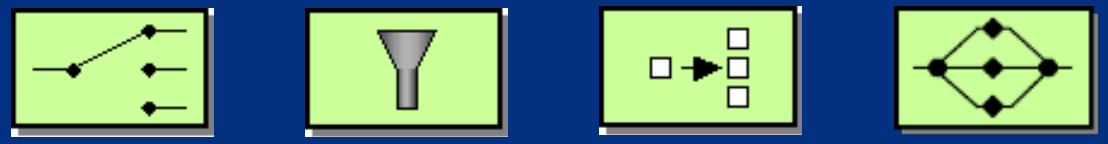


Control flow versus data flow

Enterprise Integration Patterns

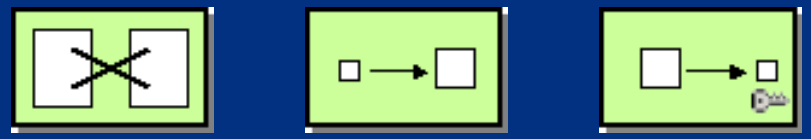
Data (message) flow

Message routing



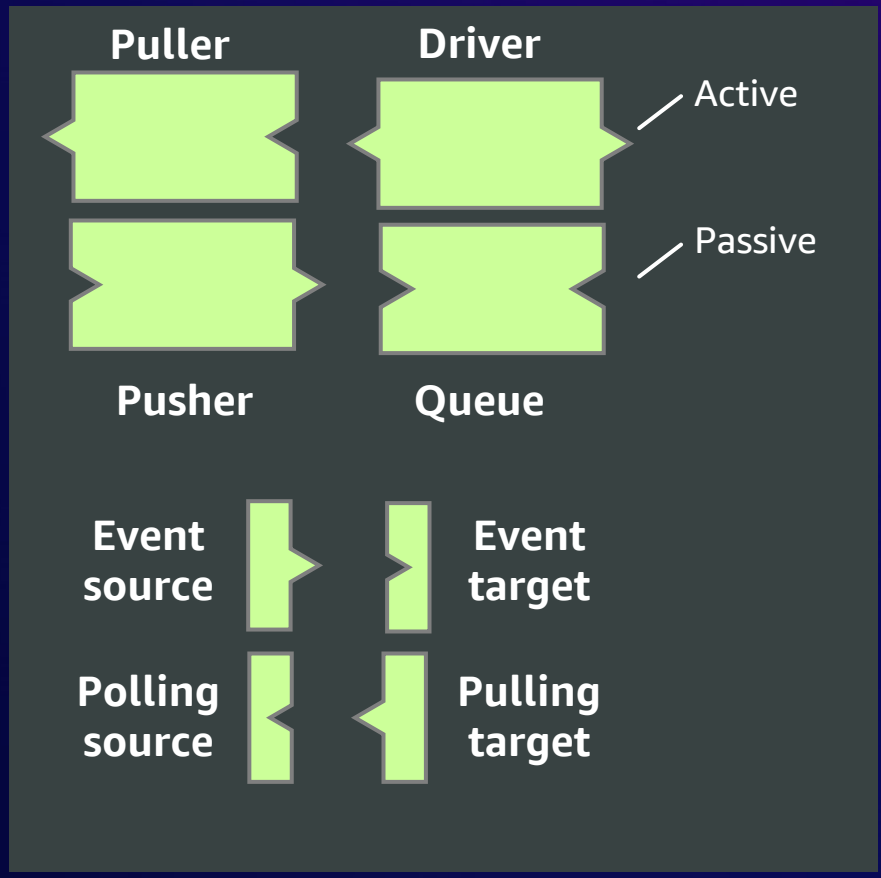
Content-Based Router | Message Filter | Splitter | Scatter Gather

Message transformation



Message Translator | Content Enricher | Claim Check

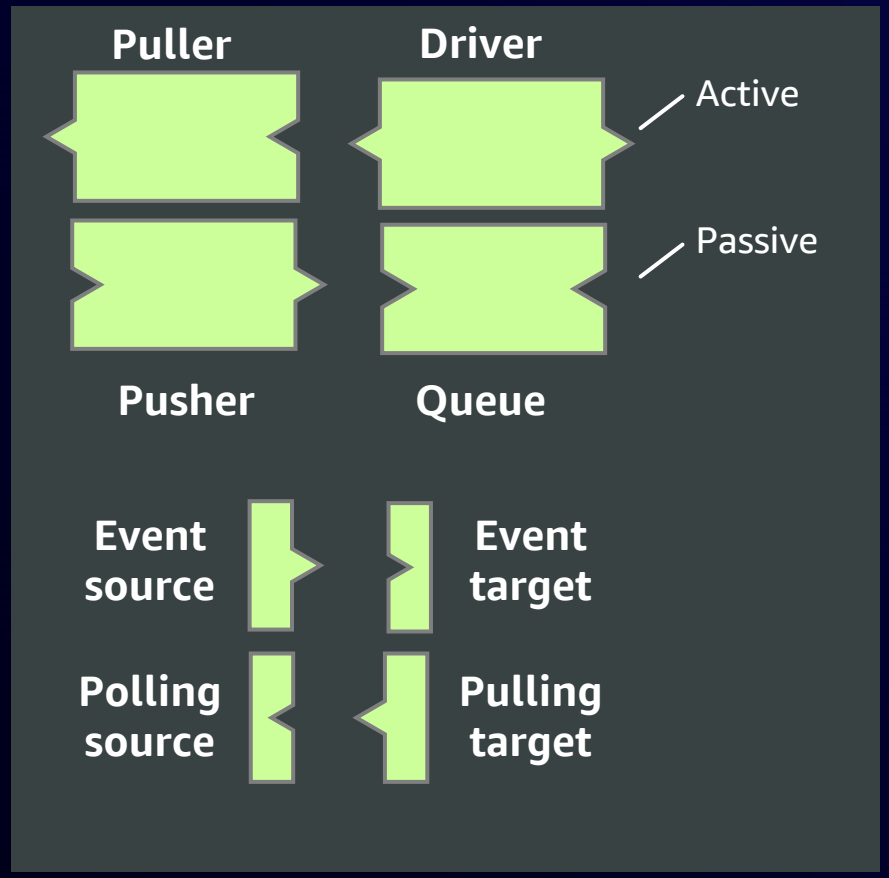
Control flow



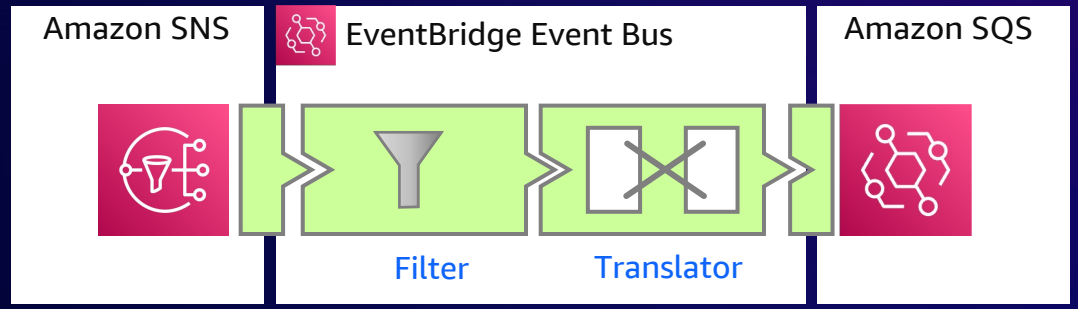
Source: Enterprise Integration Patterns, https://www.enterpriseintegrationpatterns.com/

Visualizing control flow: Amazon EventBridge Event Bus

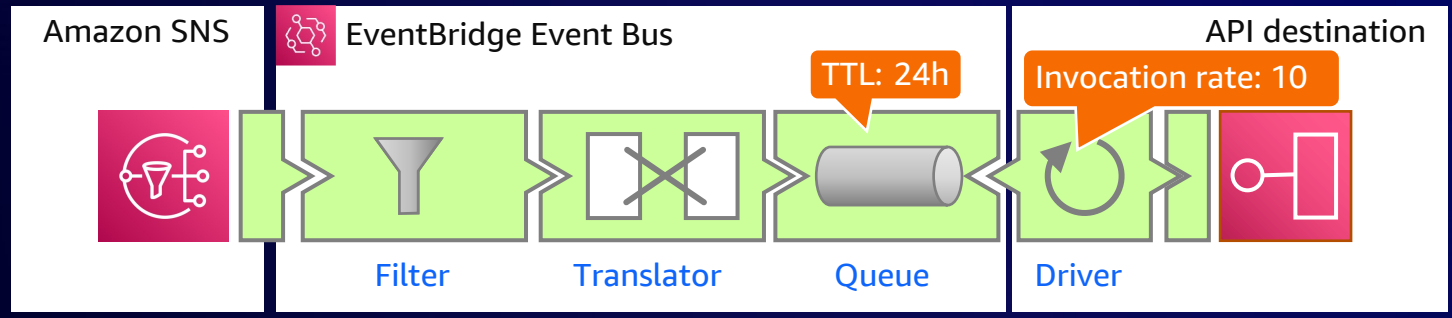
Control flow



EventBridge with event target



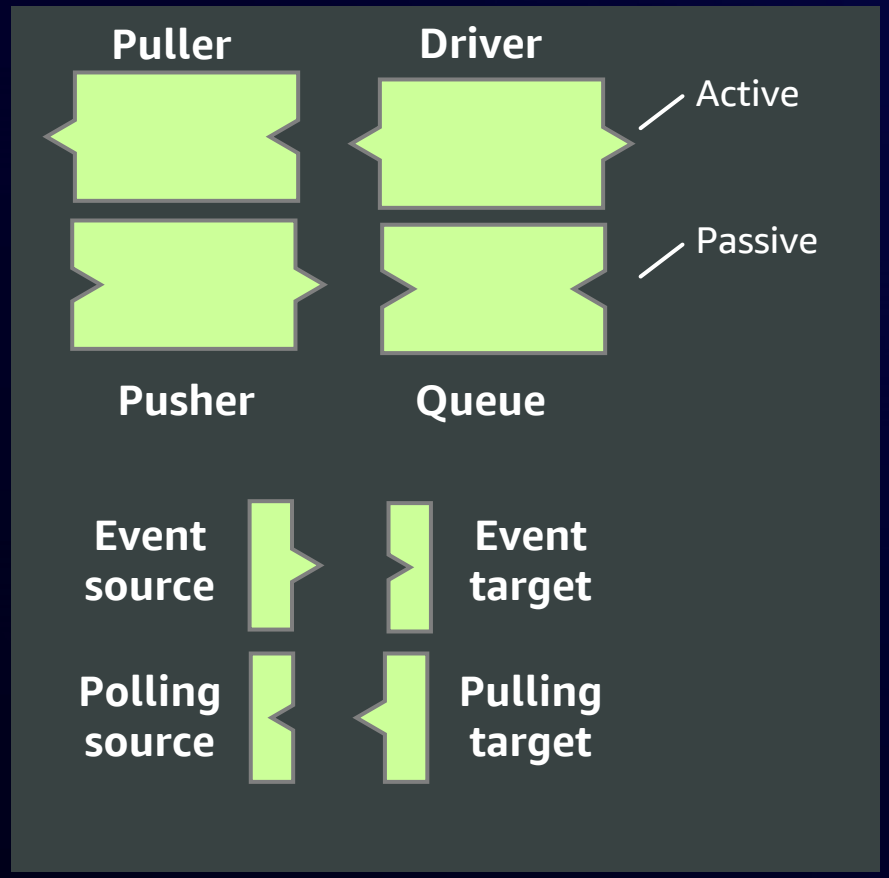
EventBridge with rate-limited target

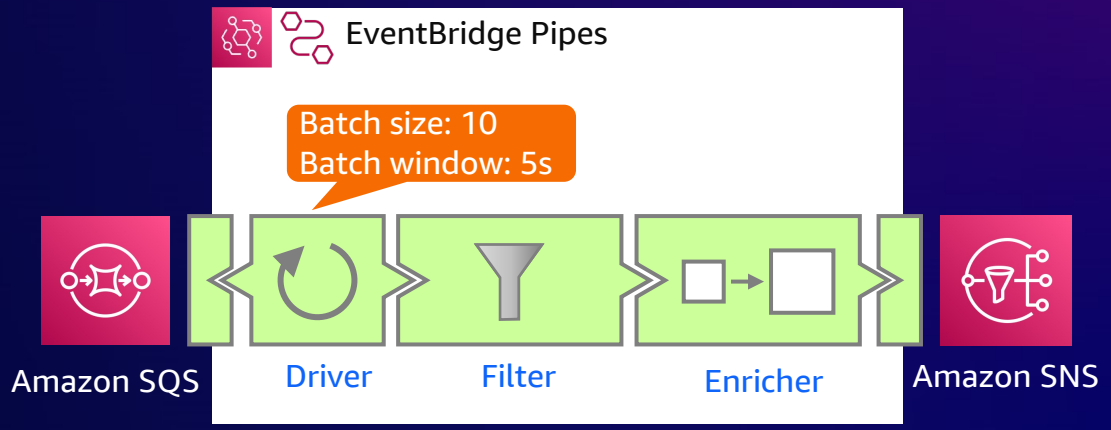


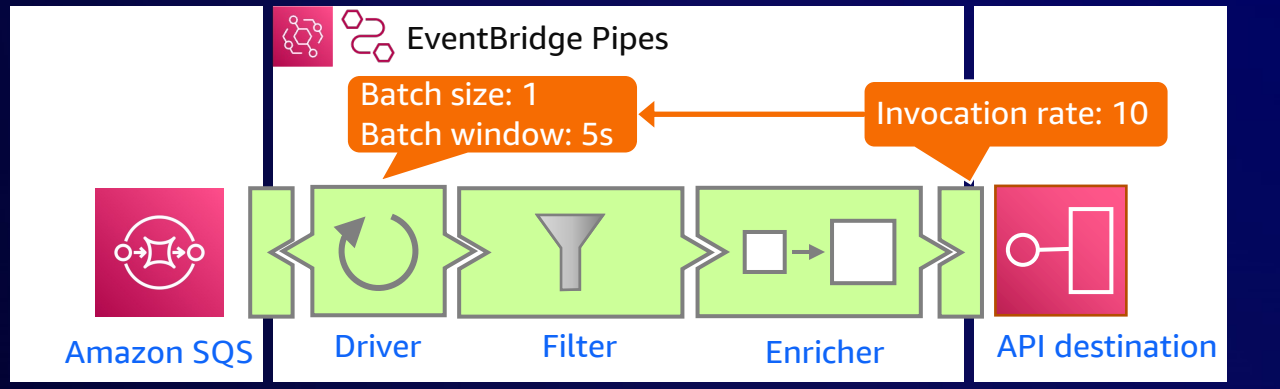
Source: Enterprise Integration Patterns, https://www.enterpriseintegrationpatterns.com/

Visualizing control flow: Amazon EventBridge Pipes

Control flow



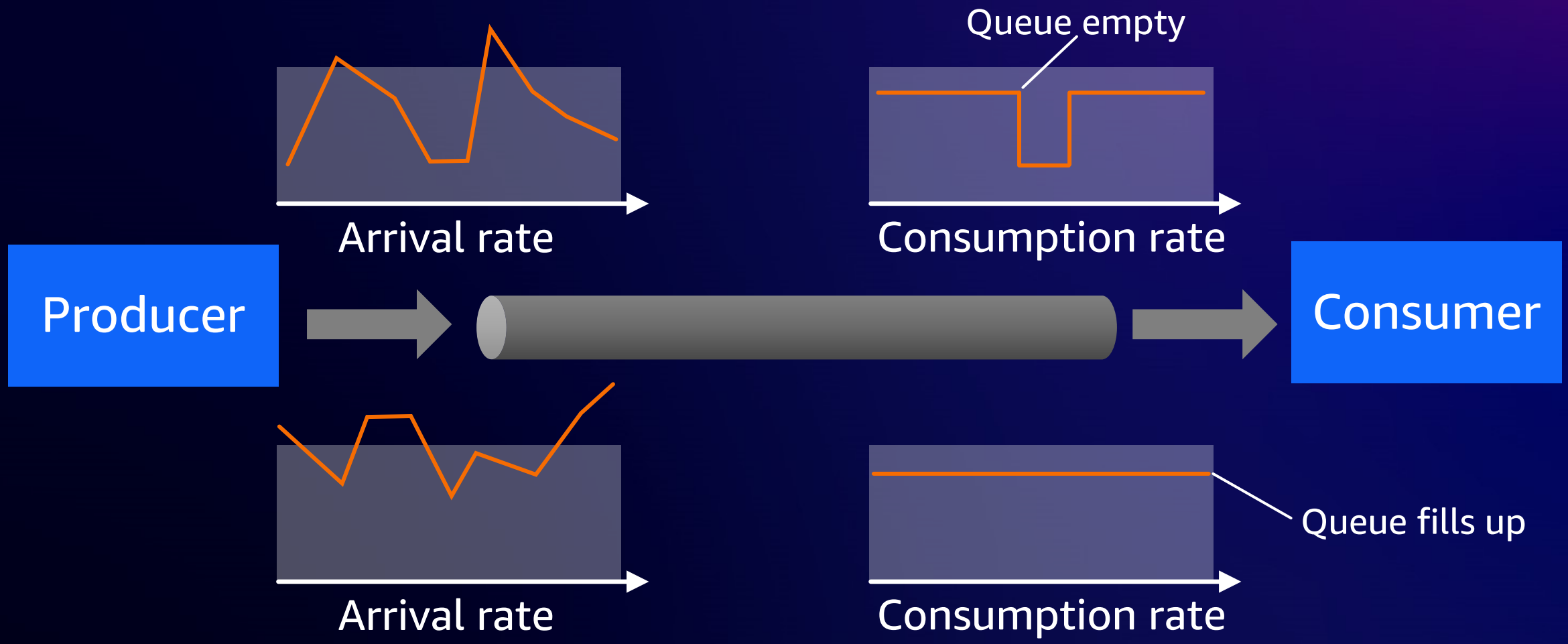




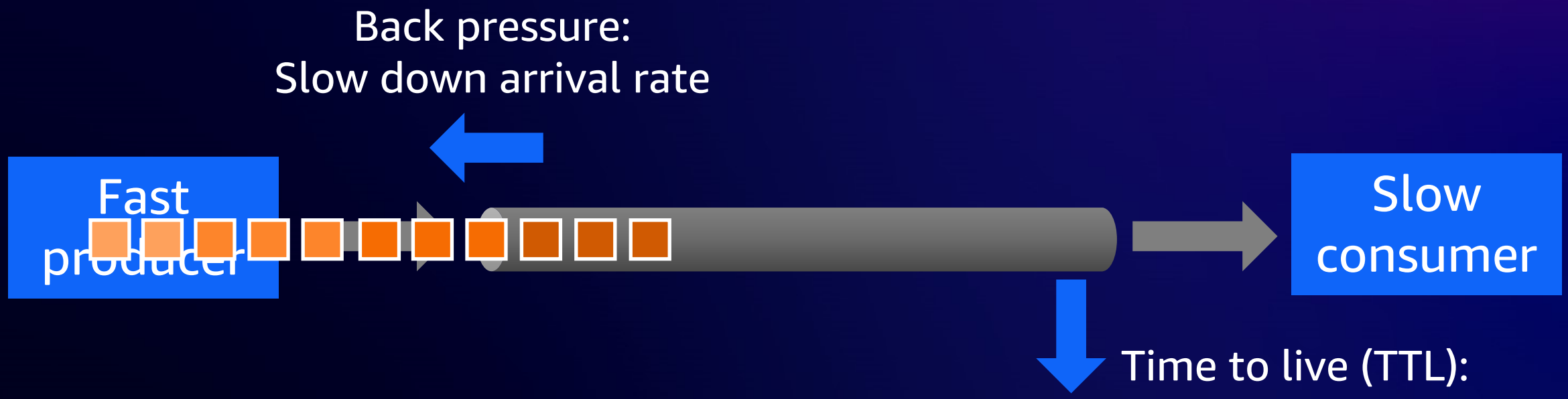
Source: Enterprise Integration Patterns, https://www.enterpriseintegrationpatterns.com/

Control flow is essential to understanding dynamic system behavior such as latency, scaling, batching, and time-outs

Flow control: Queues are traffic shapers



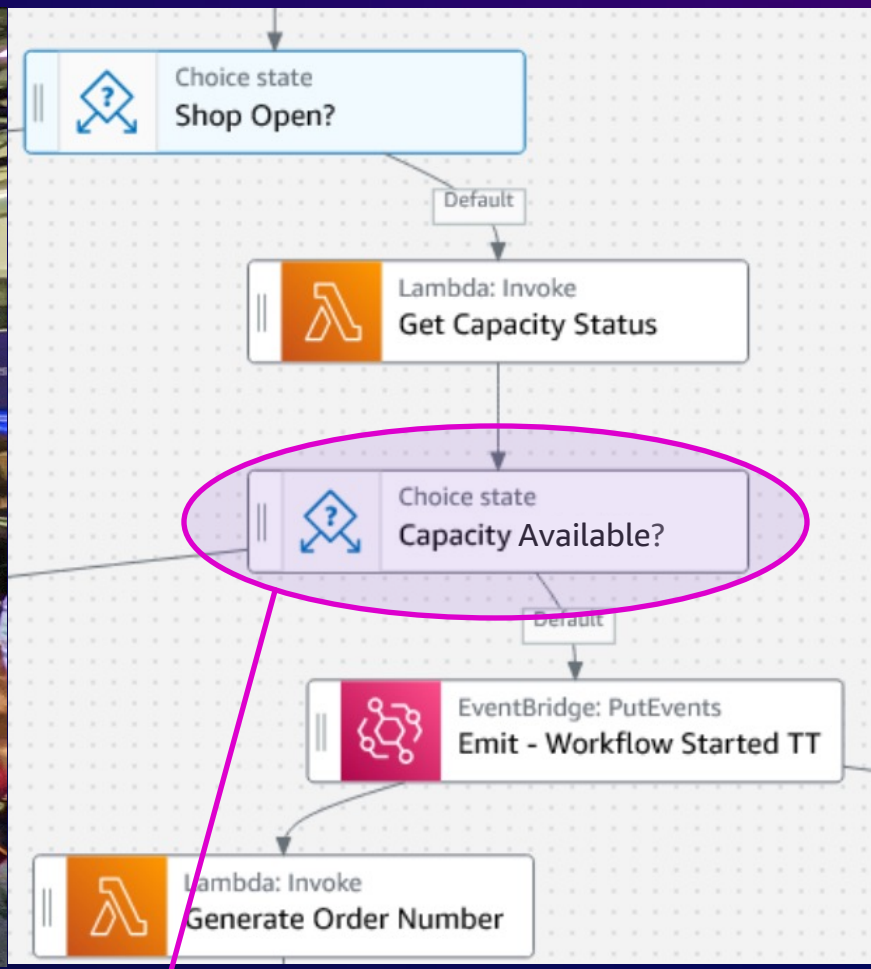
Queues require flow control



TTL is an Amazon SQS setting; back pressure has to be built

Even valuable messages have a meaningful time to live

Yes, you can learn from a coffee shop



https://serverlessland.com/reinvent2022/svs312

Back pressure

Queues decouple control flow but require flow control.

Hard-earned experience from an integration architect

Distributed system fundamentals

Order and delivery semantics

Distributed system fundamentals

Order and delivery semantics

FIFO channels

Message order is frequently requested – from time to time even with good reason

Message channels

FIFO queues

Message channels

FIFO queues

Strict order for message delivery

Message channels

FIFO queues

Producer

Consumer

Strict order for message delivery

Message channels

FIFO queues

Producer

Consumer

Strict order for message delivery

All magic comes with a price, deary

Message channels

FIFO queues

What happens with multiple consumers?

Message channels

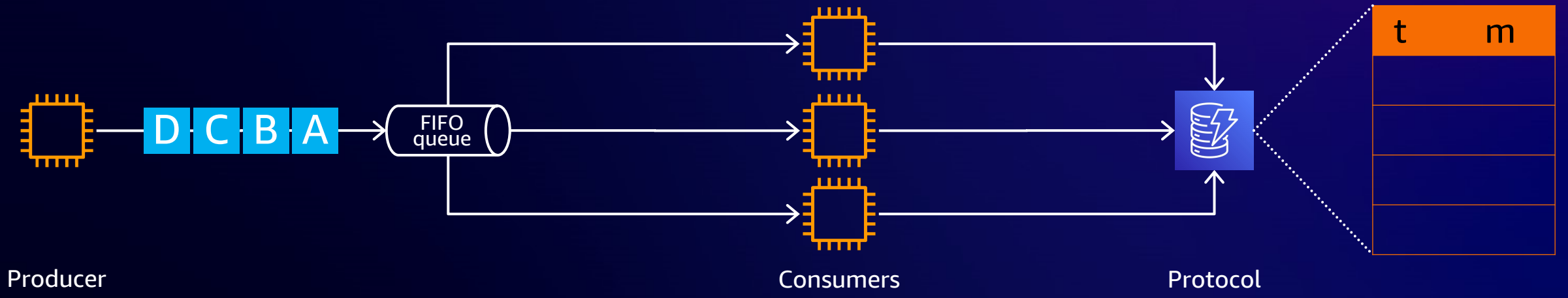
FIFO queues

What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

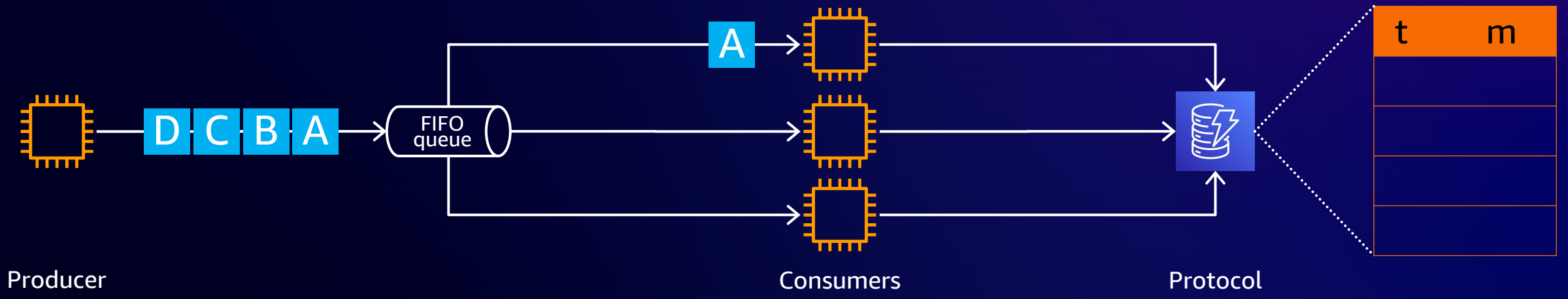


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

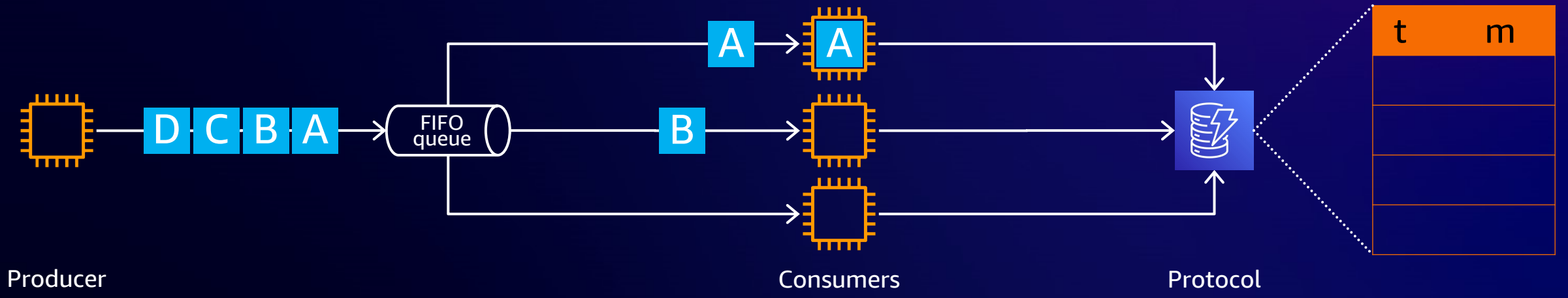


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

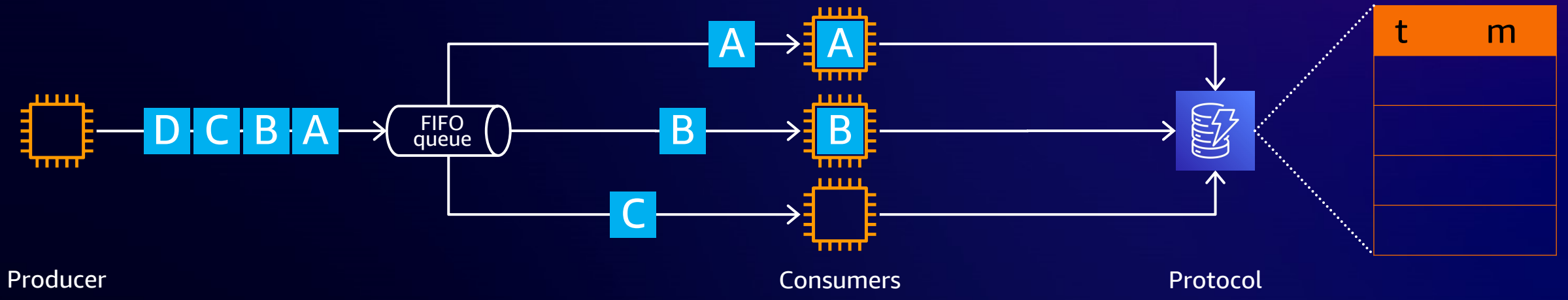


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

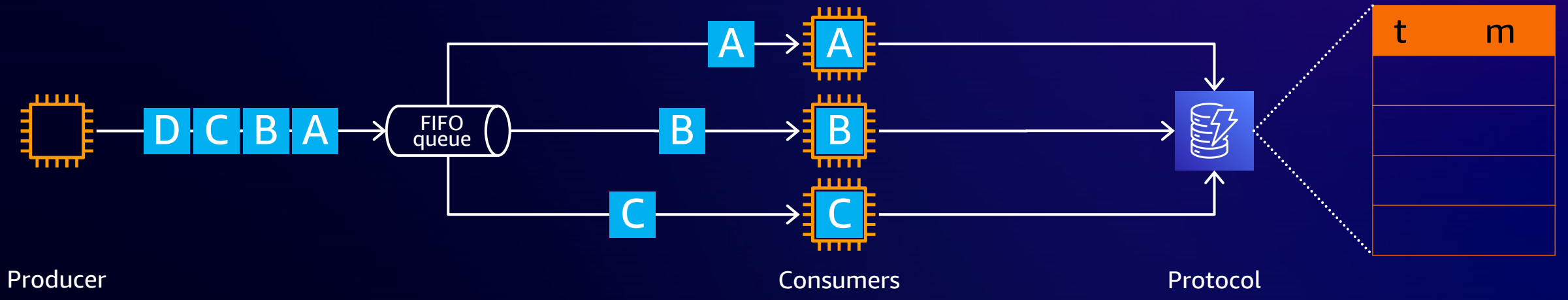


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

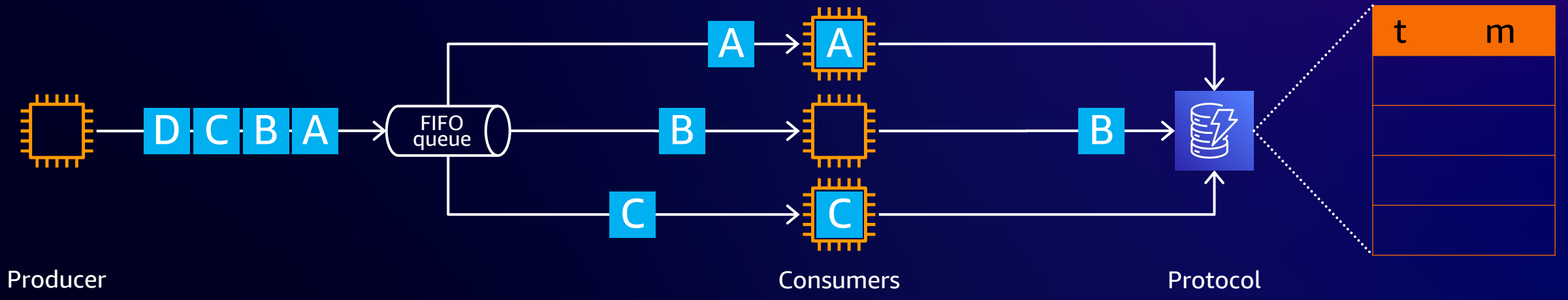


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

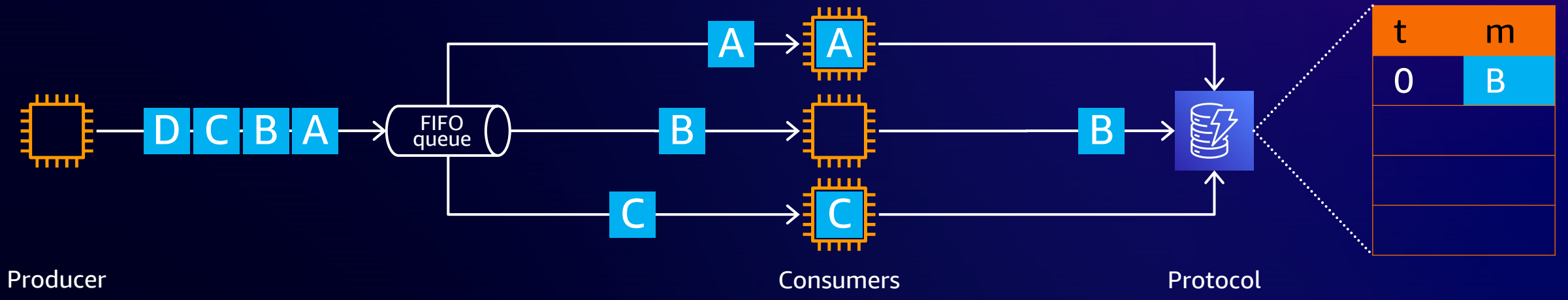


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

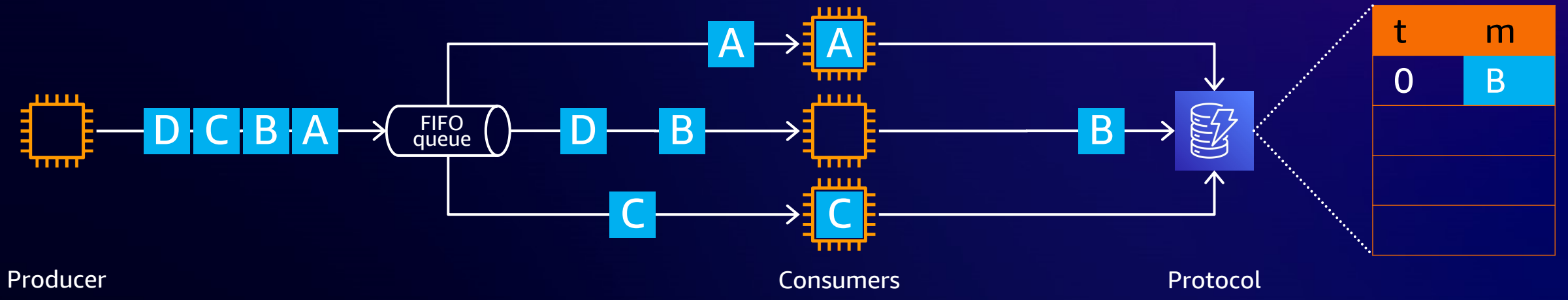


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

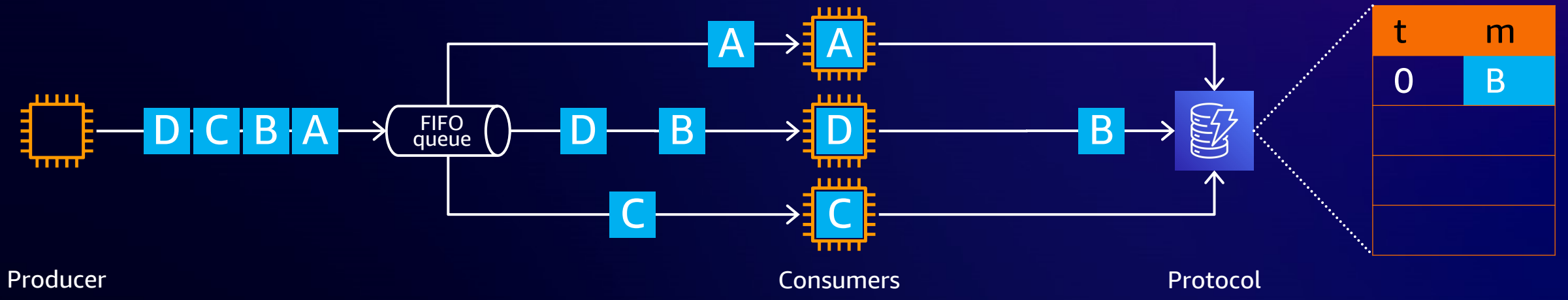


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

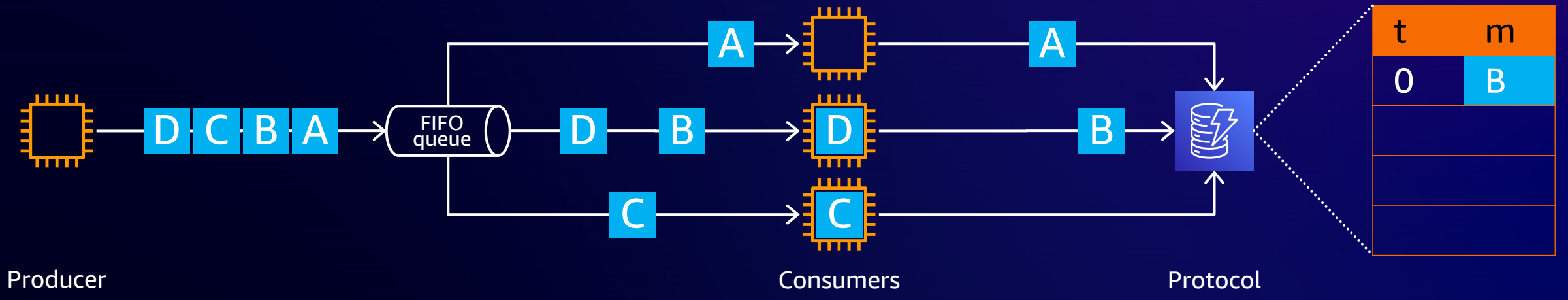


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

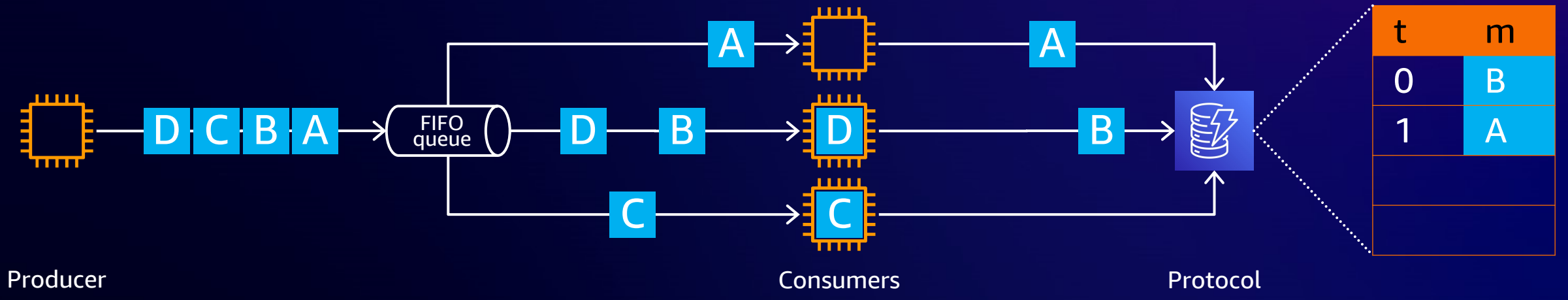


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

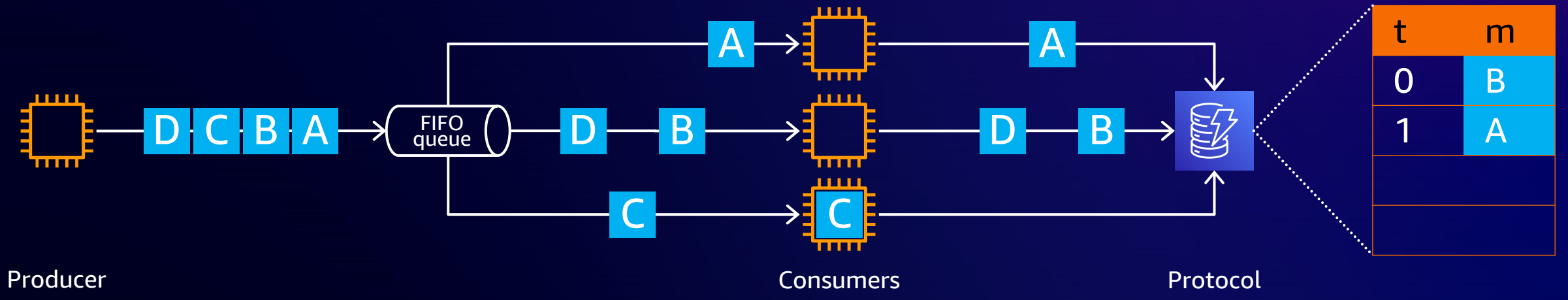


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

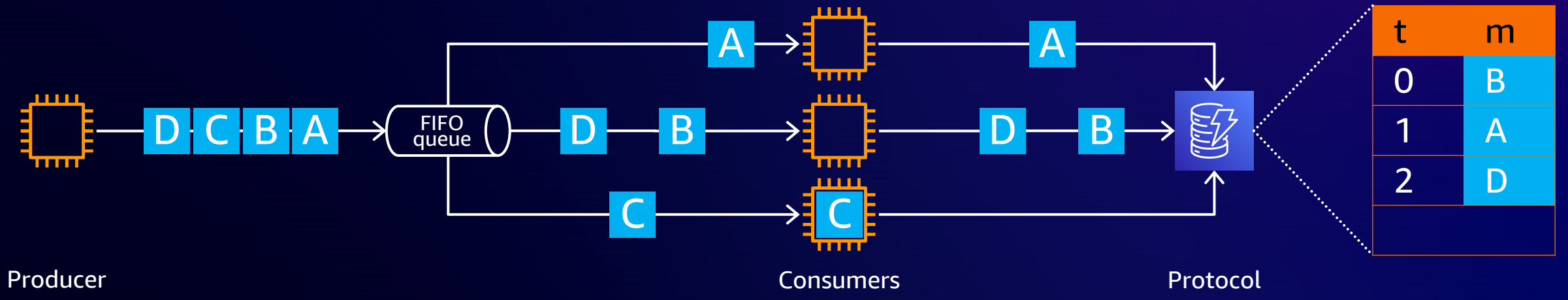


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

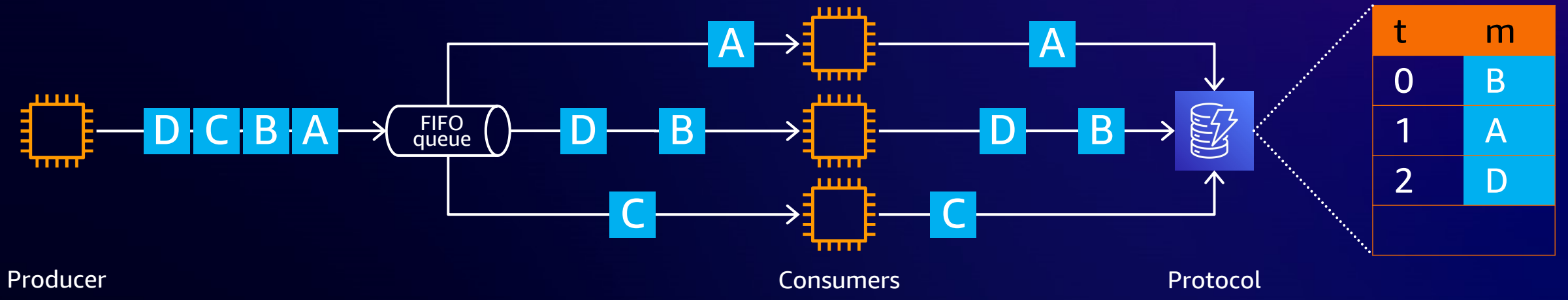


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

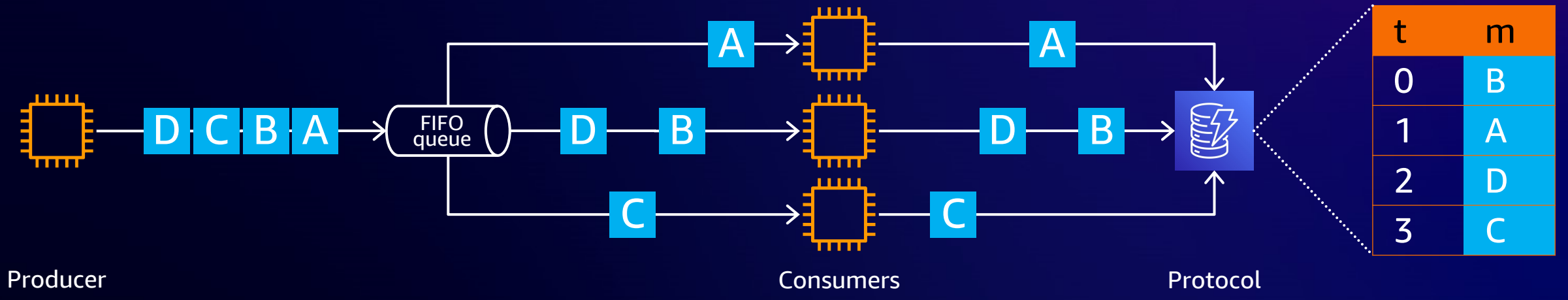


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)

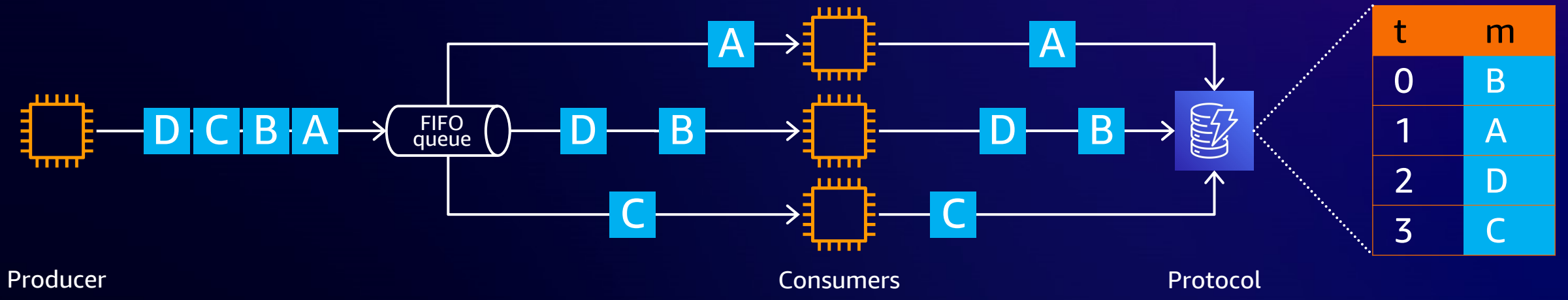


What happens with multiple consumers?

Concurrent message consumption!

Message channels

FIFO queues (naïve approach)



What happens with multiple consumers?

Concurrent message consumption!

Message processing time can vary and lead to order violation in consumer processes

Message channels

FIFO queues + message groups

Message channels

FIFO queues + message groups

Messages grouped by discriminator attribute

Message channels

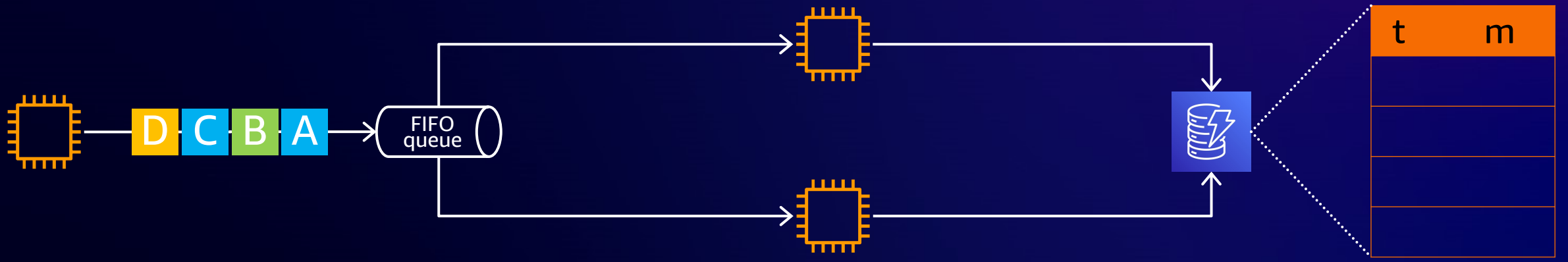
FIFO queues + message groups

Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

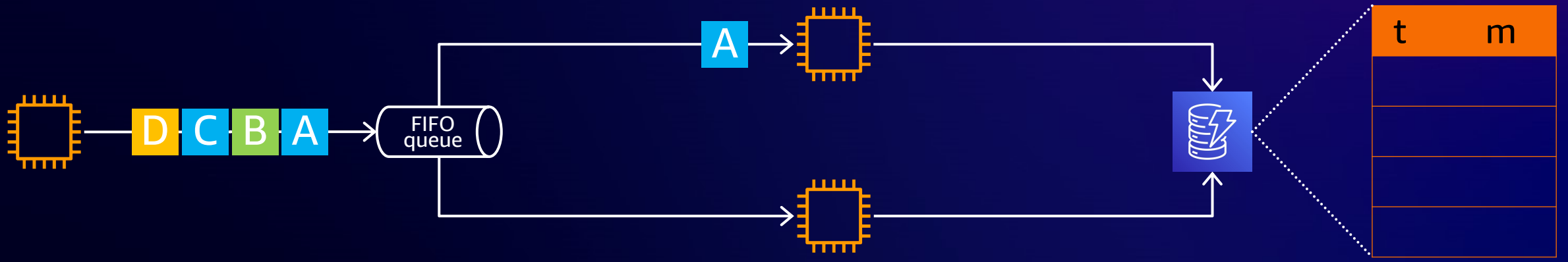


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

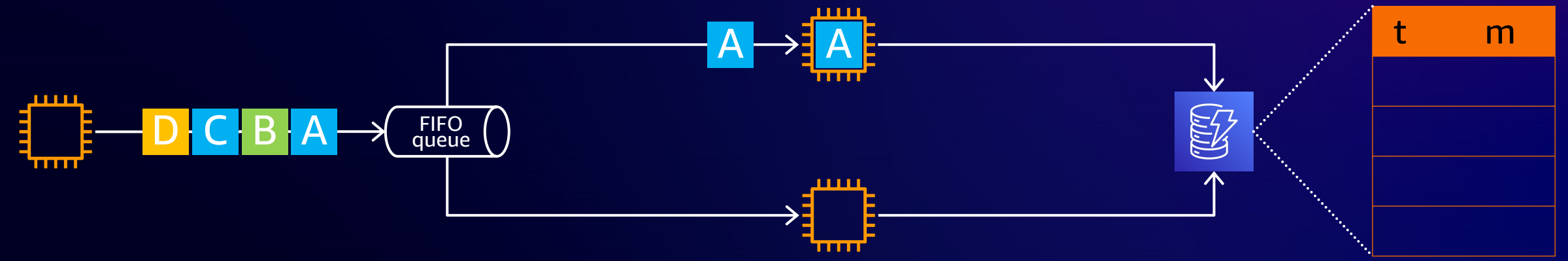


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

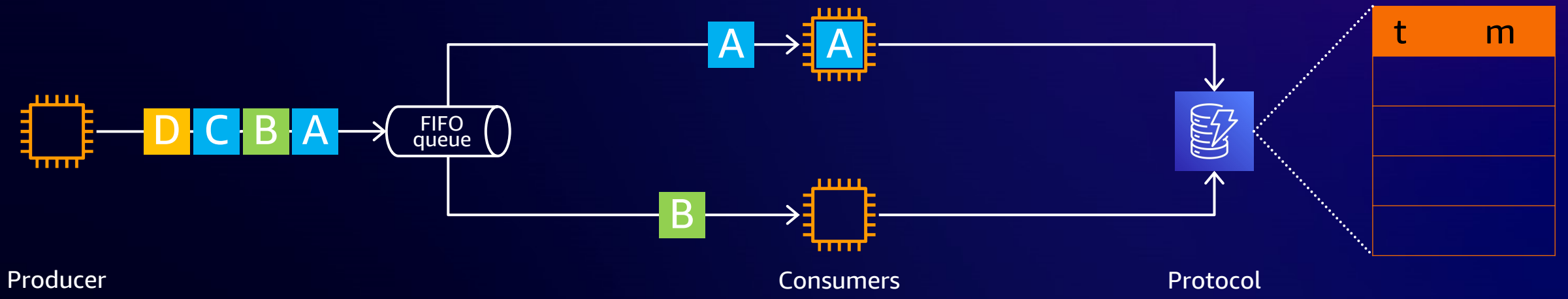


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

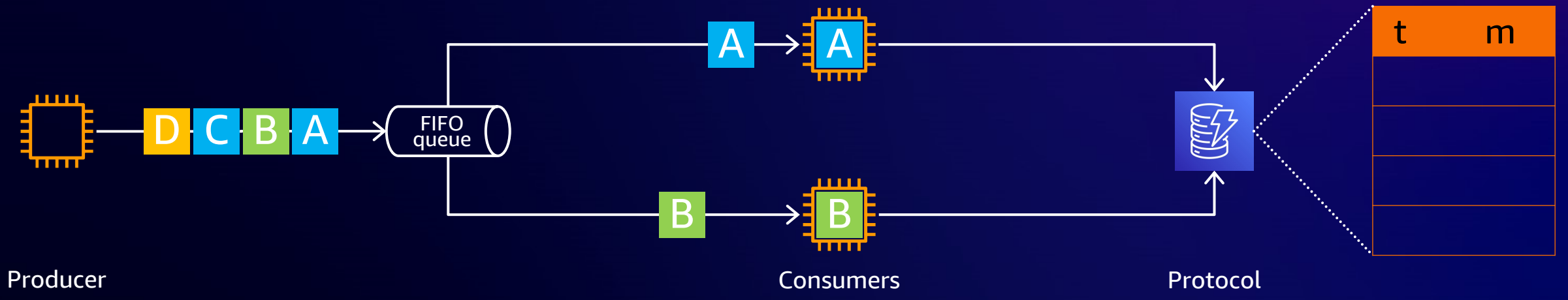


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

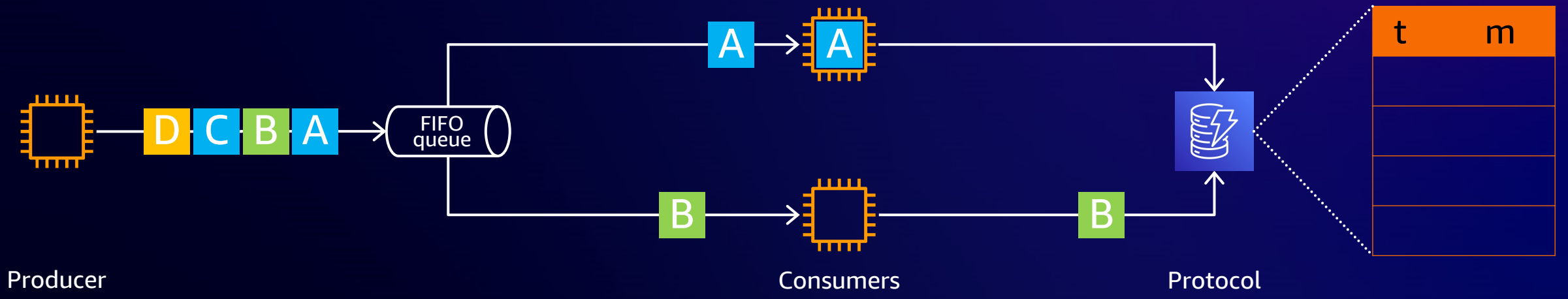


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

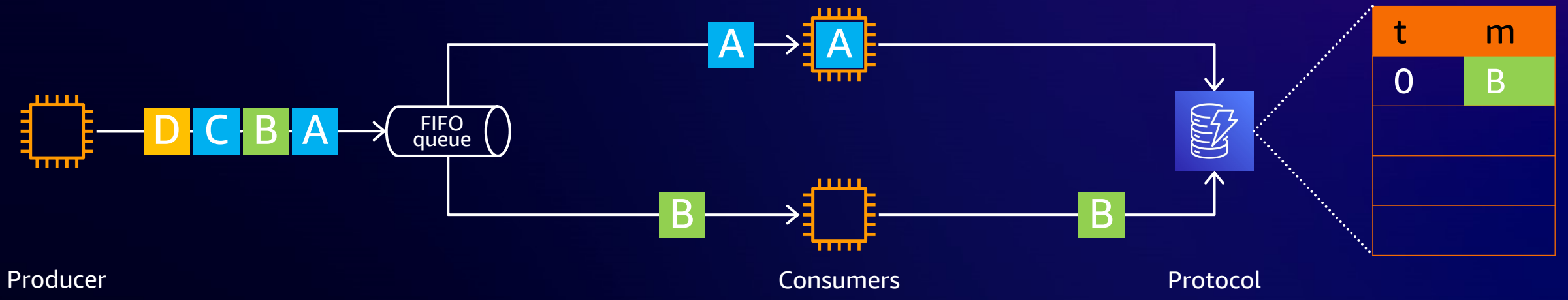


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

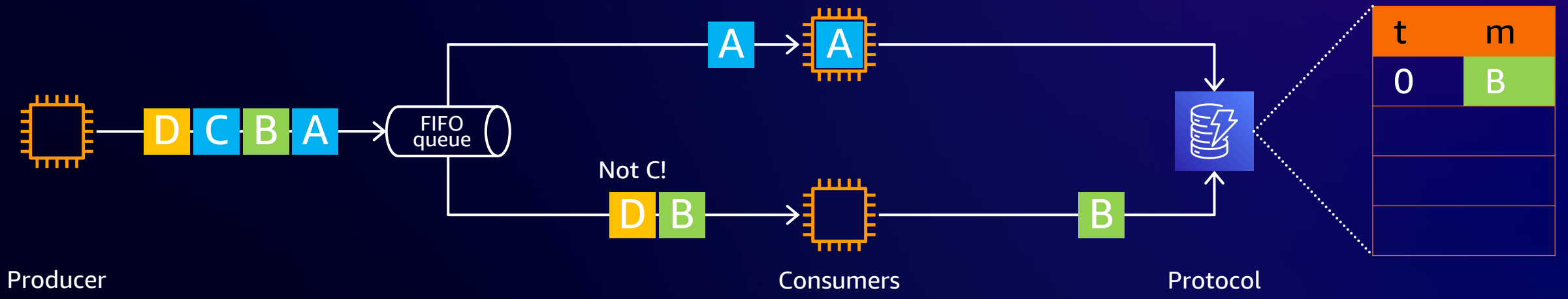


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

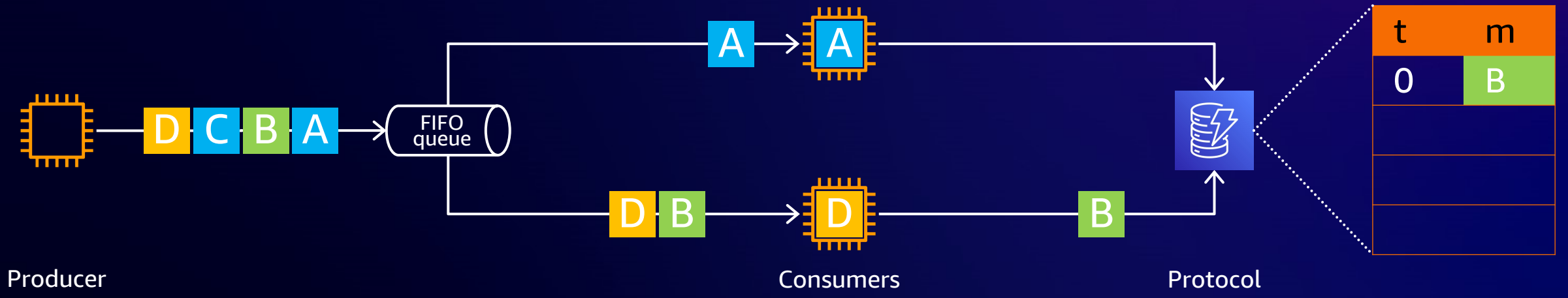


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

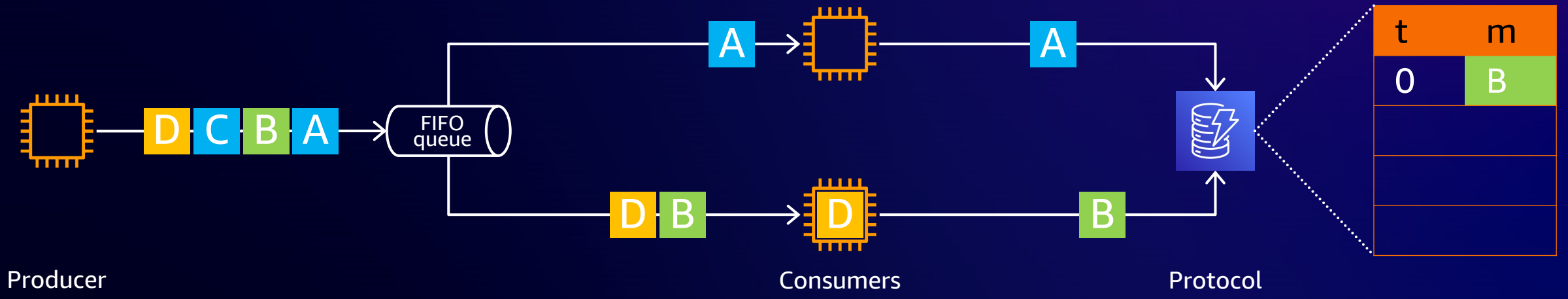


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

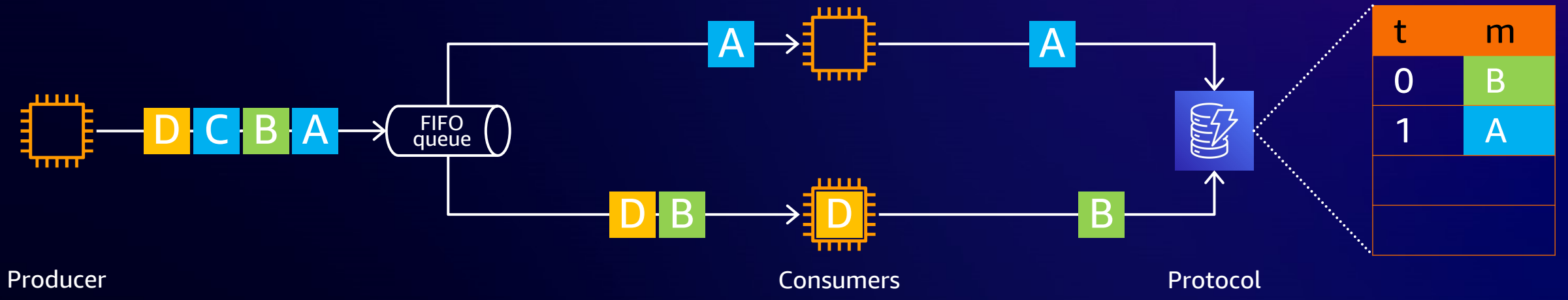


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

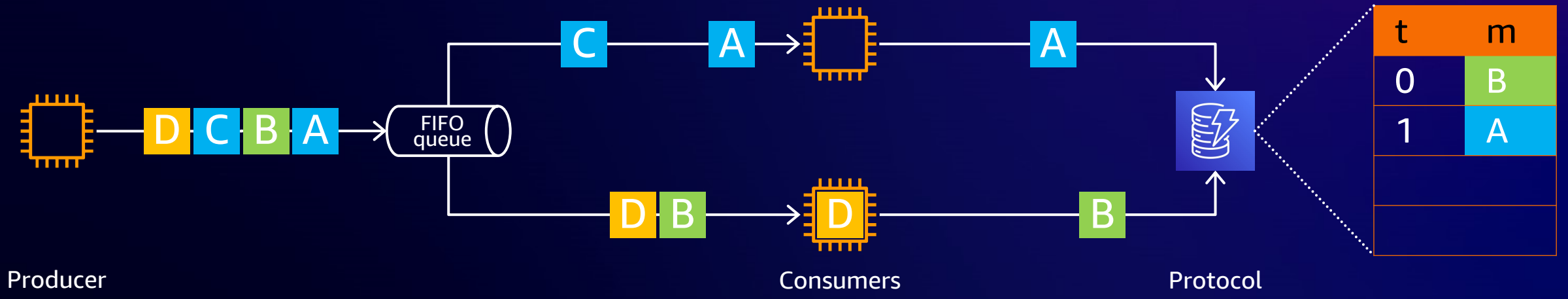


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

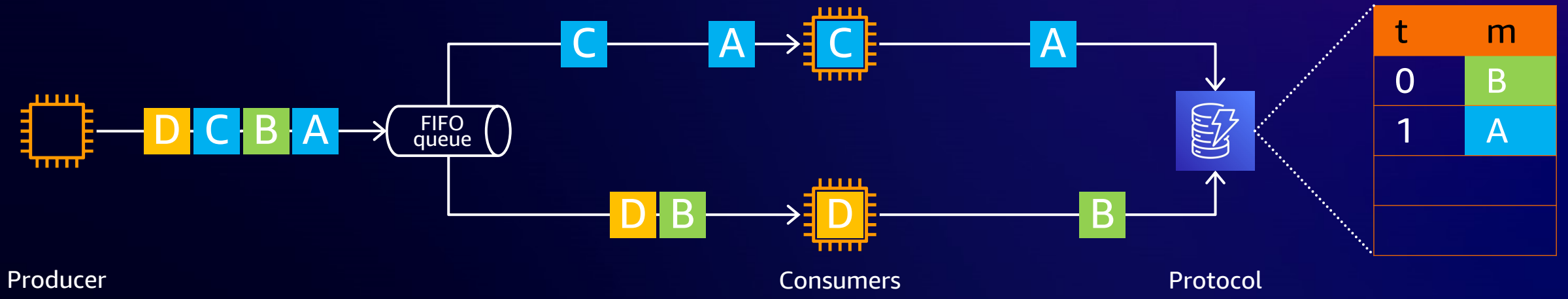


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

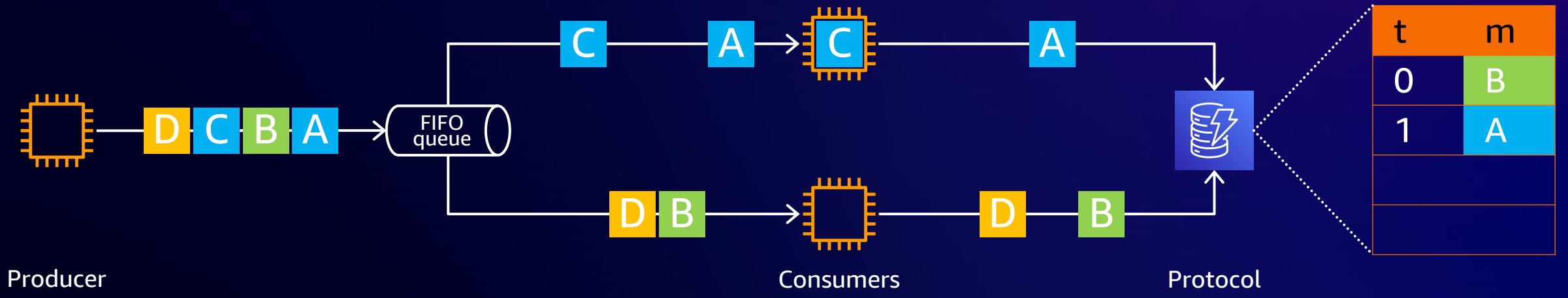


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

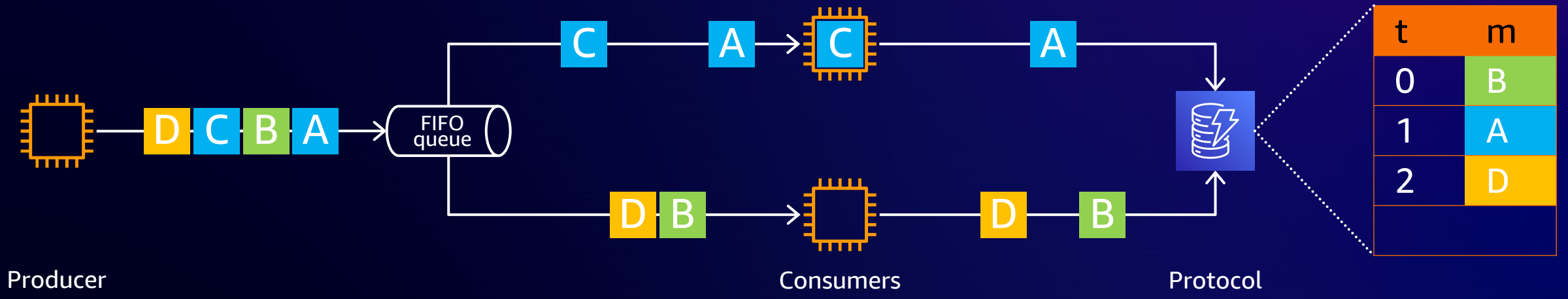


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

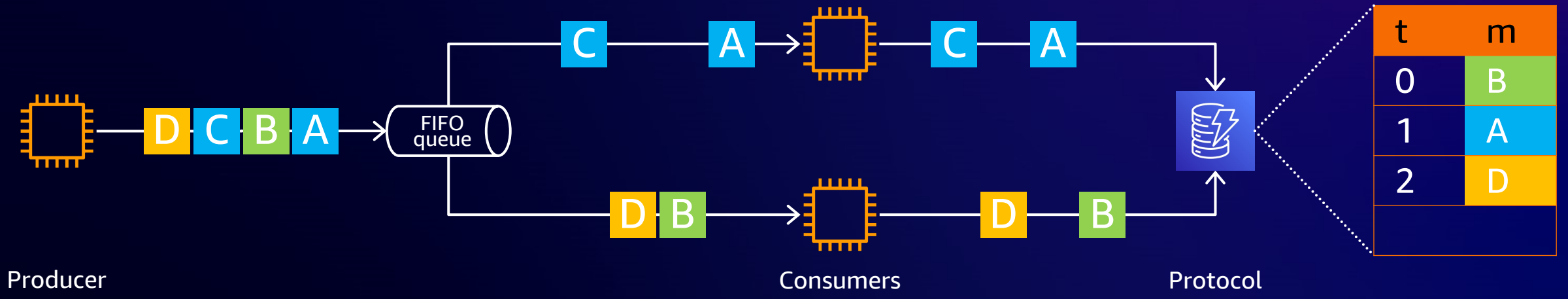


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

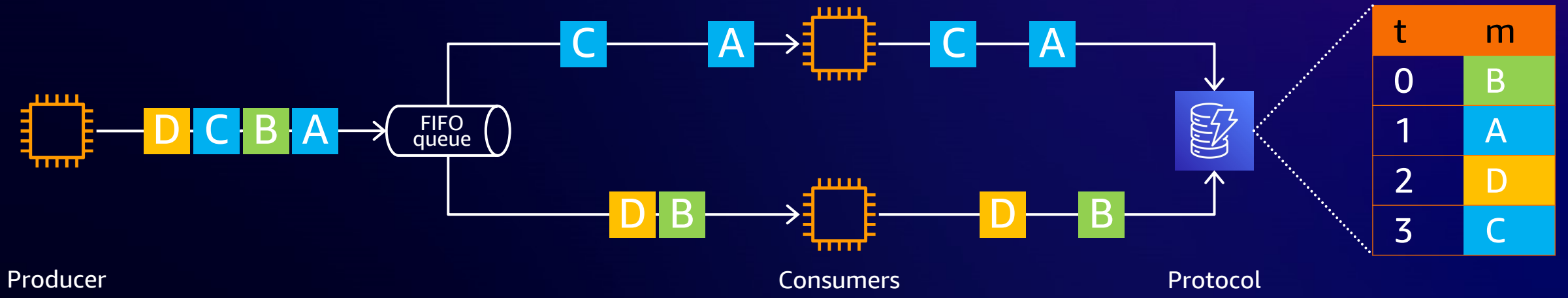


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups

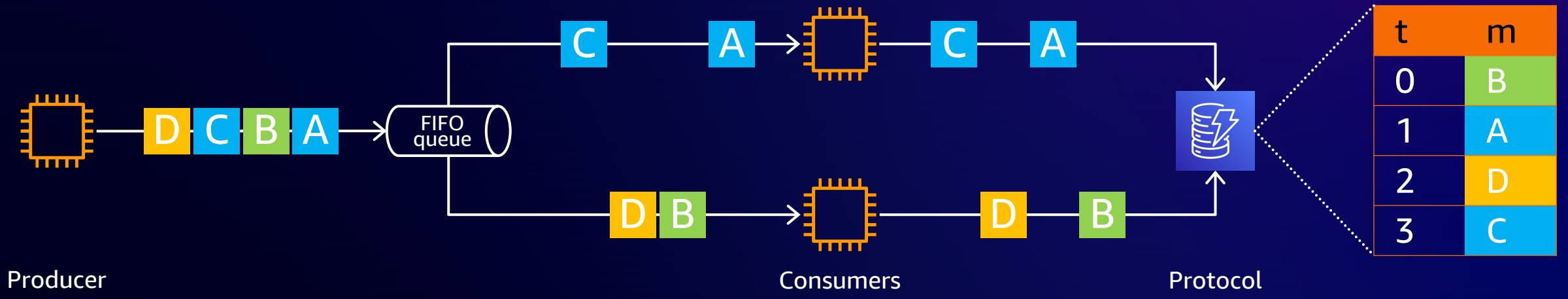


Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Message channels

FIFO queues + message groups



Messages grouped by discriminator attribute

Example: Amazon SQS message group IDs

Order retained within each message group

Consumer scale-out on # of message group IDs

Order (of messages in distributed systems) is relative to a defined scope

Message channels

FIFO queues + message groups

Local versus global message order?

t	m
0	B
1	A
2	D
3	C

Producer

Messages groupe

age group IDs

Order retained within each message group

Consumer scale-out on # of message group IDs

Message channels

FIFO queues + message groups

Producer

t	m
0	B
1	A
2	D
3	C

What if a global order is needed?

Messages groupe

age group IDs

Order retained within each message group

Consumer scale-out on # of message group IDs

Message channels

FIFO queues + message groups

Producer

t	m
0	B
1	A
2	D
3	C

What if a global order is needed?

One message group for all messages, but global is still relative

Messages groupe

age group IDs

Order retained within each message group

Consumer scale-out on # of message group IDs

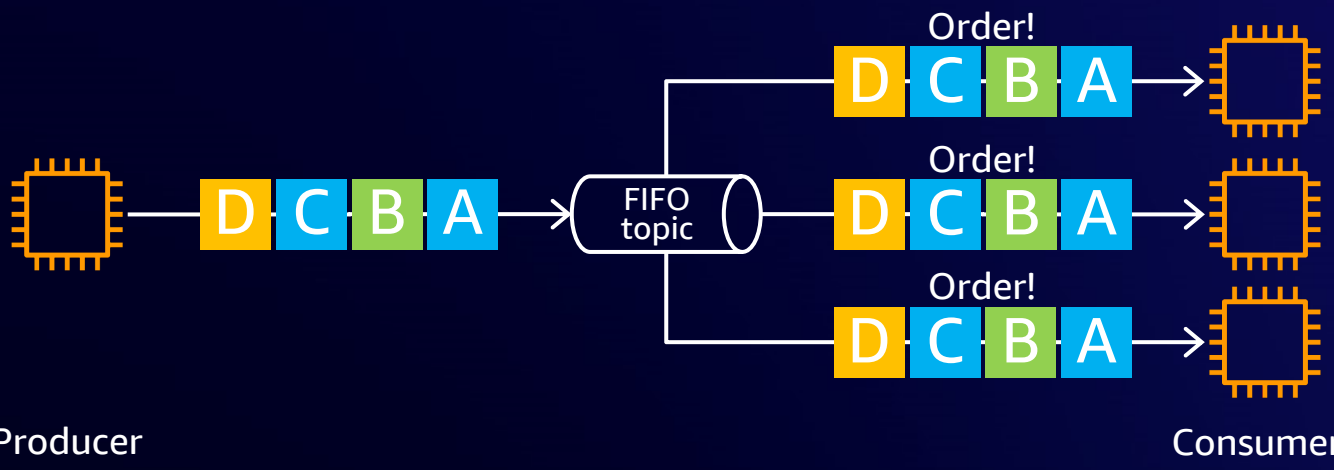
If you require a global message order, you end up back at sequential database writes

Message channels

FIFO topics

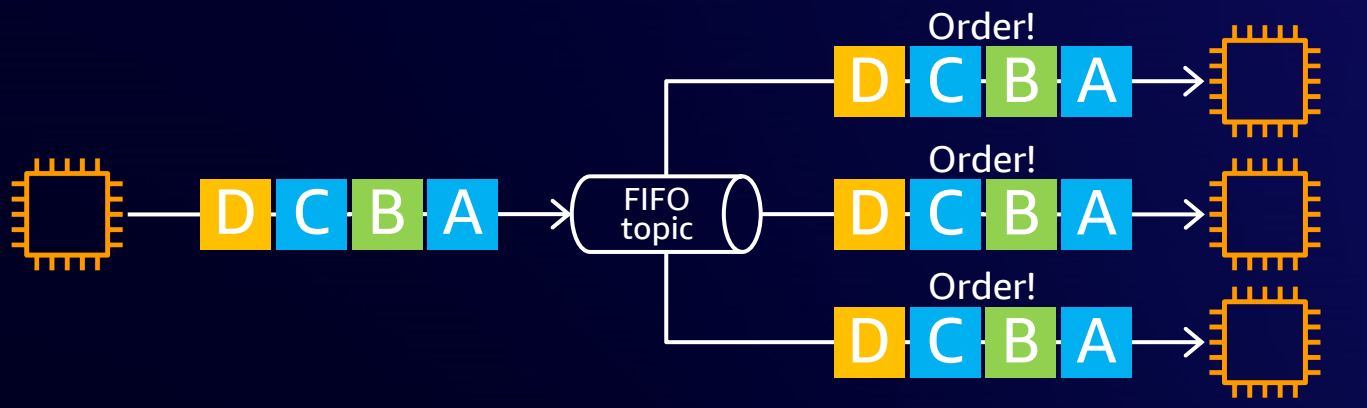
Message channels

FIFO topics



Message channels

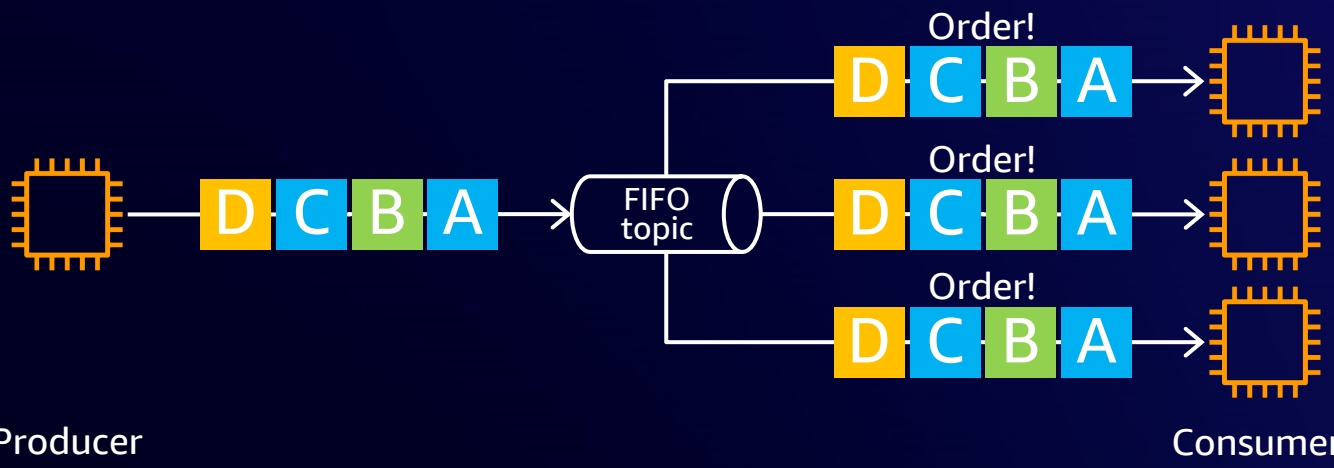
FIFO topics



No concurrent consumers

Message channels

FIFO topics

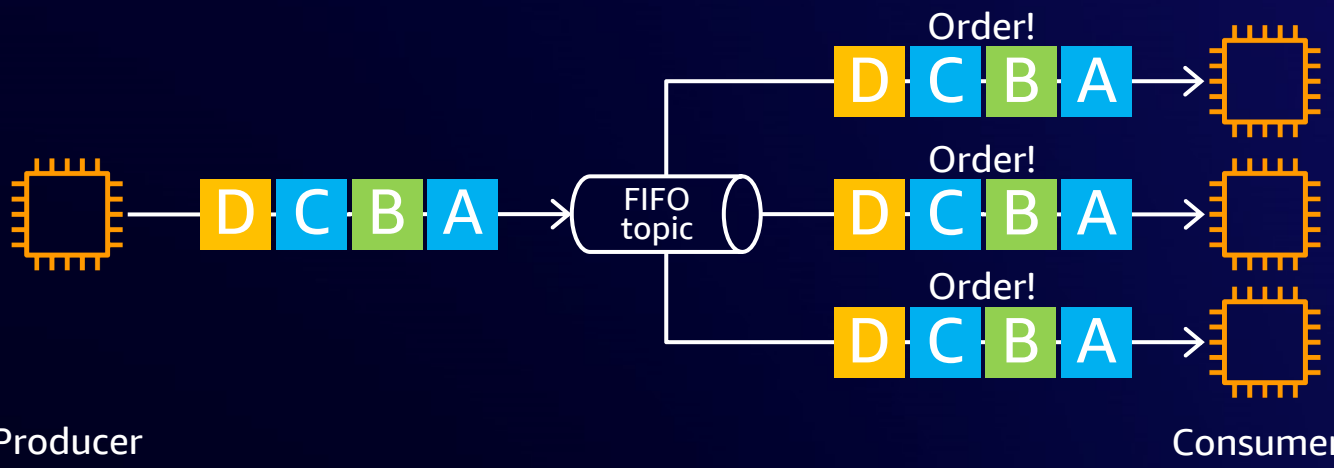


No concurrent consumers

What happens for topic-queue-chaining?

Message channels

FIFO topics



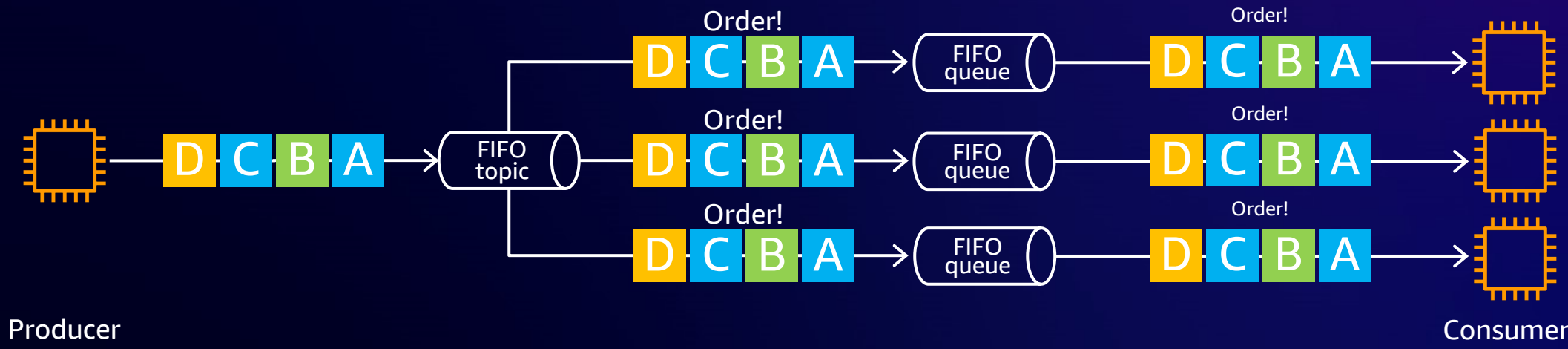
No concurrent consumers

What happens for topic-queue-chaining?

SNS FIFO topics also use message groups (MG)

Message channels

FIFO topics



No concurrent consumers

What happens for topic-queue-chaining?

SNS FIFO topics also use message groups (MG)

MG IDs to be forwarded into queue messages

Distributed system fundamentals

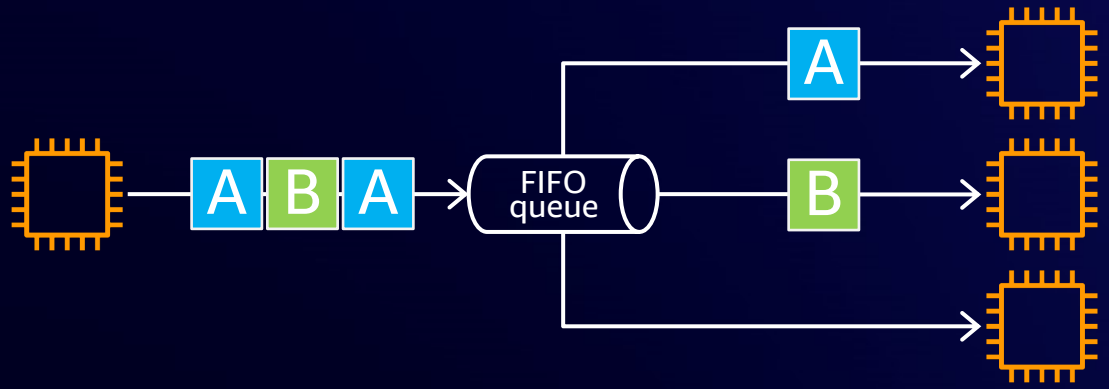
Order and delivery semantics

Deduplication and how-often delivery

Message deduplication is frequently requested – from time to time even with good reason

Deduplication

Amazon SQS FIFO

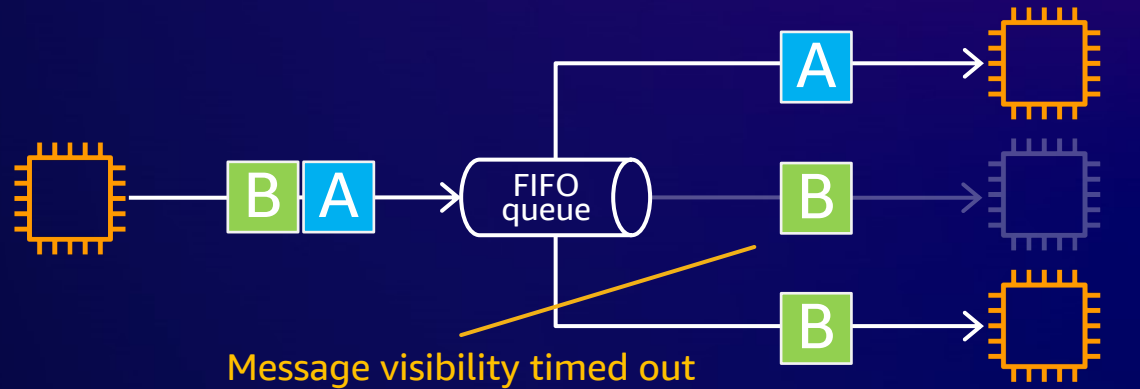


Deduplication in production

Message deduplication ID

!! 5-minute interval !!

Amazon SQS FIFO



Deduplication in consumption

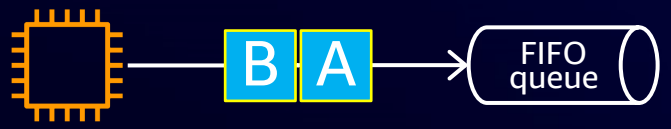
Receive request attempt ID

!! Message visibility timeout !!

Deduplication (of messages in distributed systems)
is **relative** to a defined **scope**

How-often delivery

Looking for exactly-once dingus?

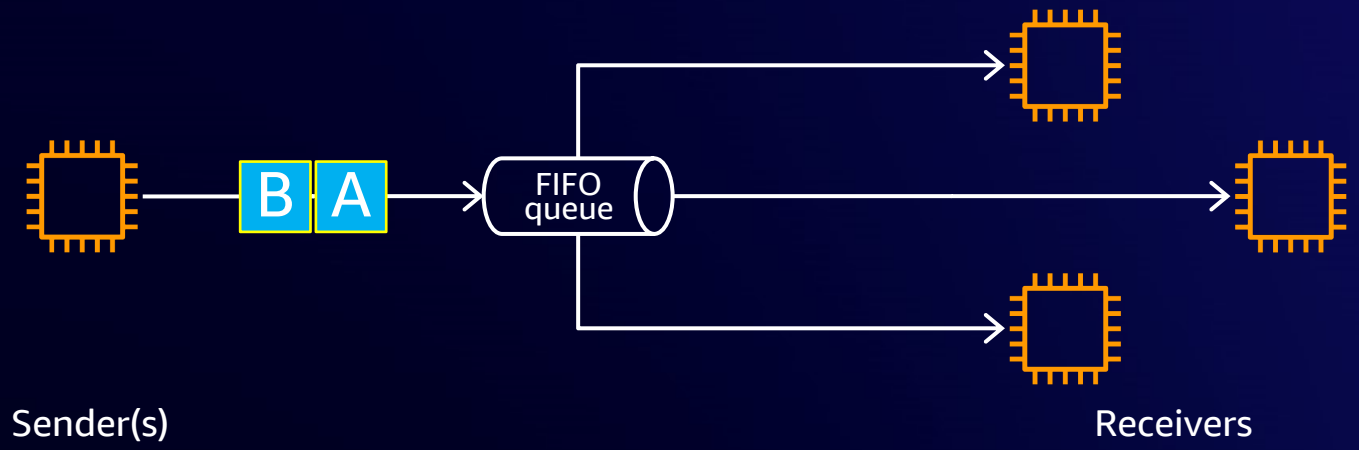


Sender(s)

Use all deduplication options for producers and consumers

How-often delivery

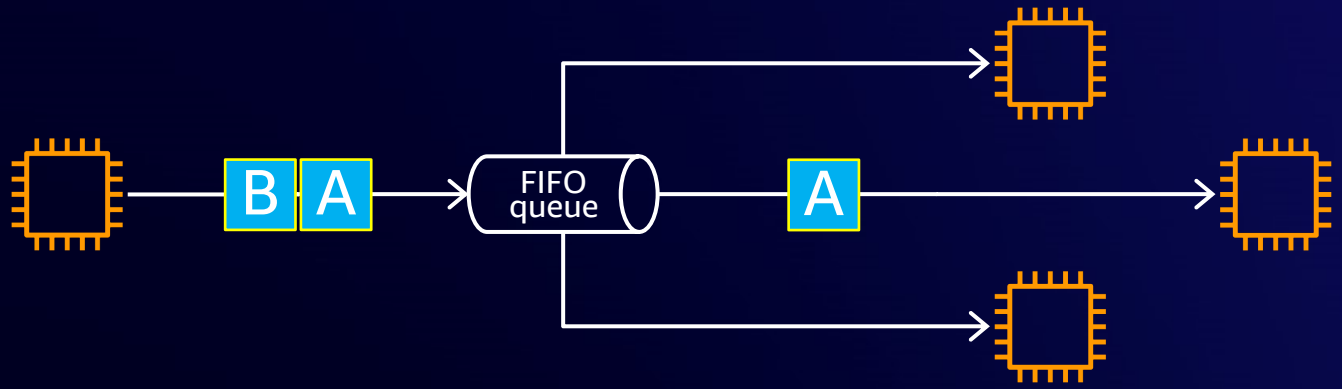
Looking for exactly-once dingus?



Use all deduplication options for producers and consumers

How-often delivery

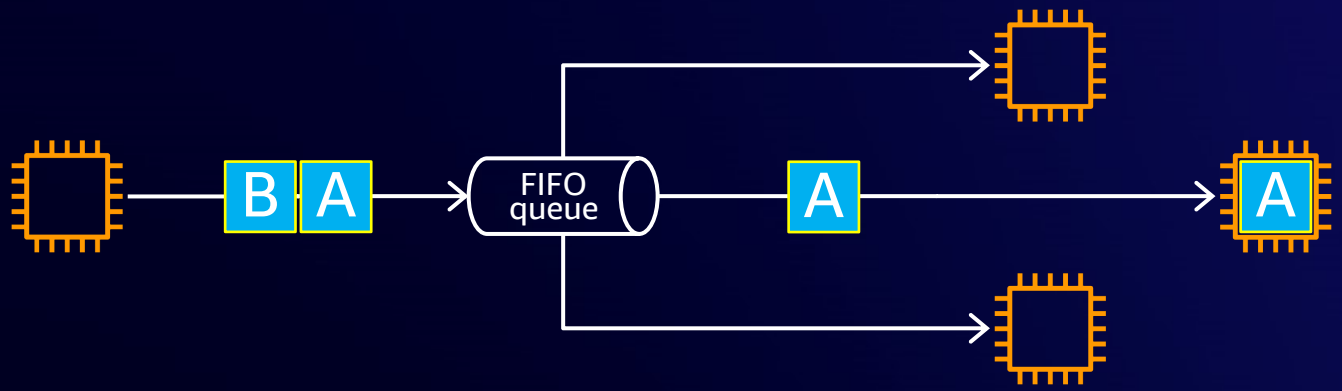
Looking for exactly-once dingus?



Use all deduplication options for producers and consumers

How-often delivery

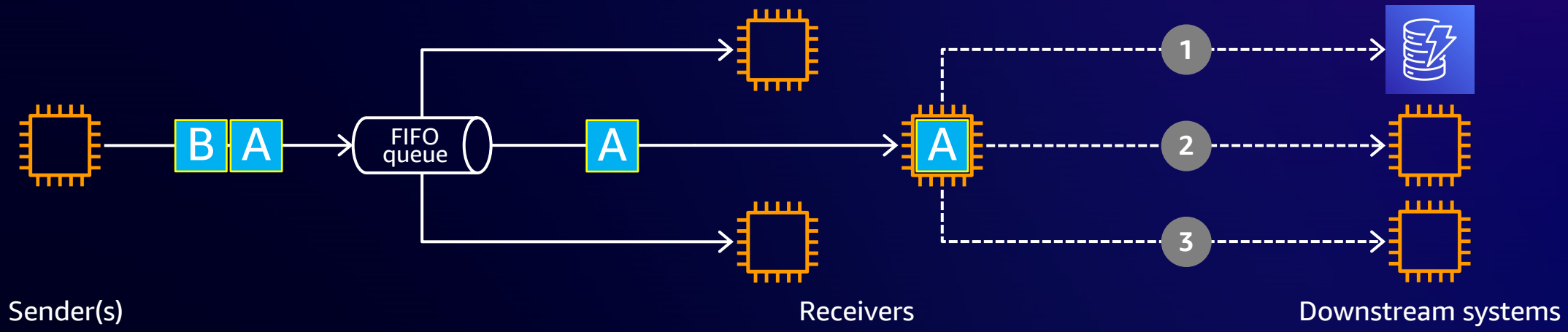
Looking for exactly-once dingus?



Use all deduplication options for producers and consumers

How-often delivery

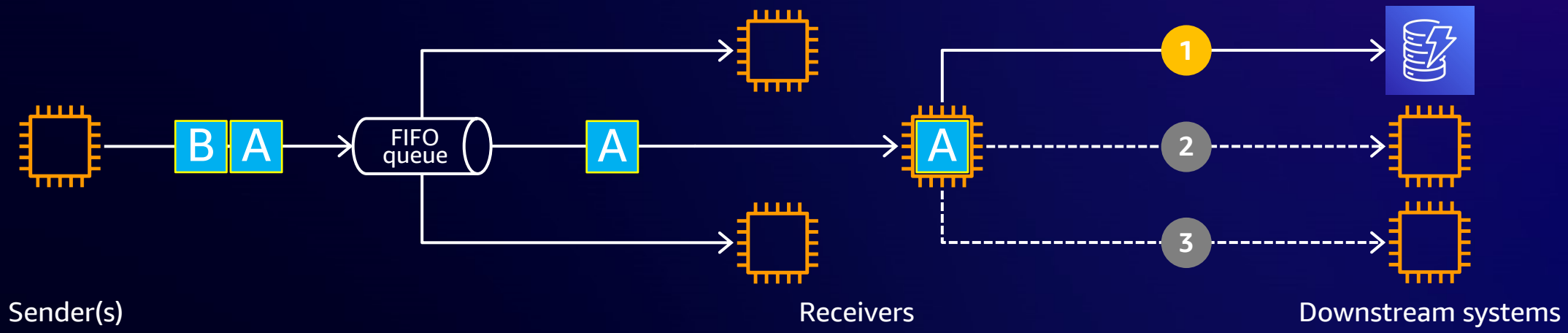
Looking for exactly-once dingus?



Use all deduplication options for producers and consumers

How-often delivery

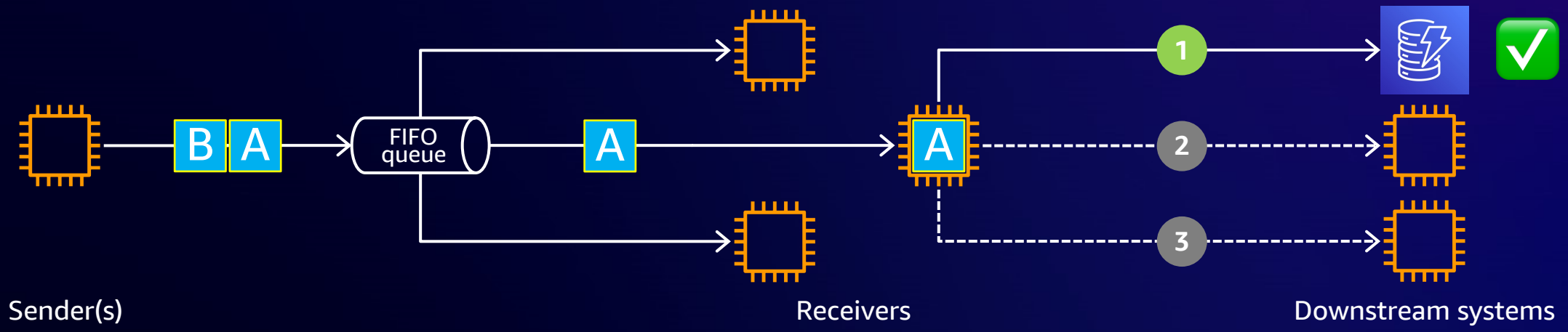
Looking for exactly-once dingus?



Use all deduplication options for producers and consumers

How-often delivery

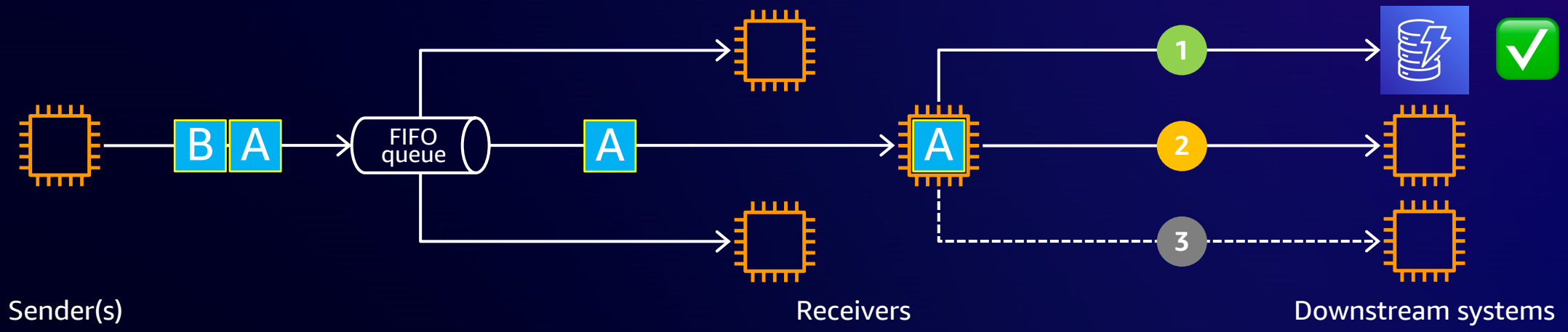
Looking for exactly-once dingus?



Use all deduplication options for producers and consumers

How-often delivery

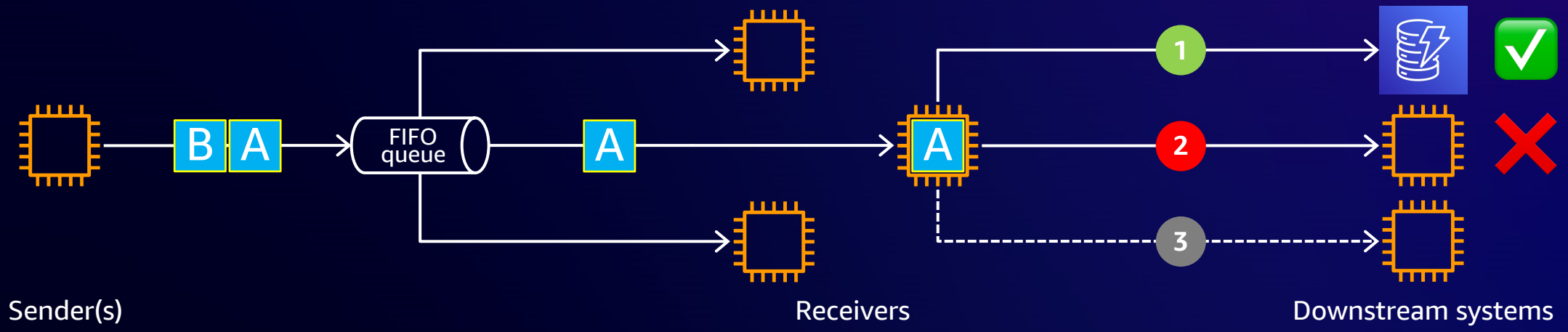
Looking for exactly-once dingus?



Use all deduplication options for producers and consumers

How-often delivery

Looking for exactly-once dingus?

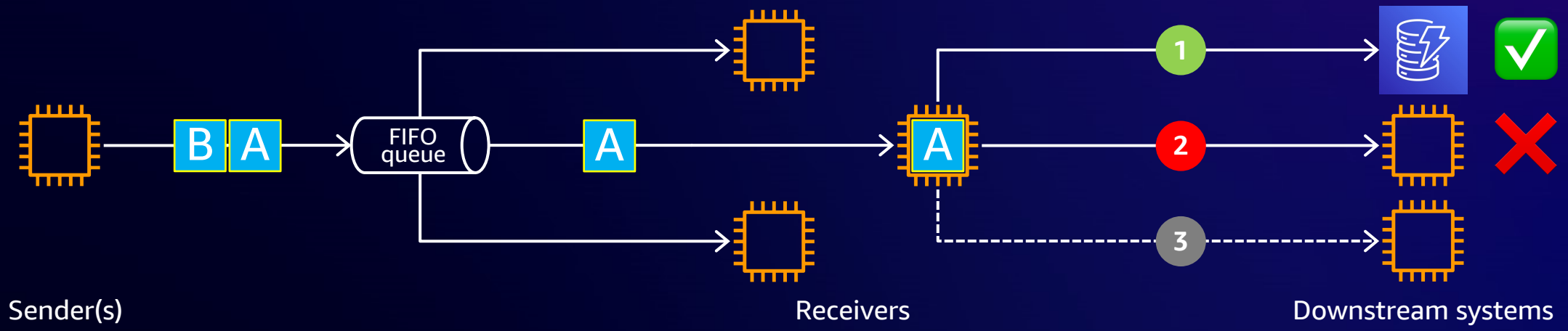


Use all deduplication options for producers and consumers

Exception in step 2

How-often delivery

Looking for exactly-once dingus?



Use all deduplication options for producers and consumers

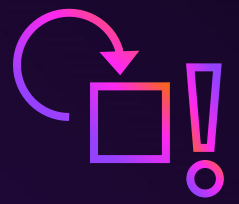
Exception in step 2

What behavior do you want now?

If you require exactly-once delivery, you end up back at synchronous integration or shared database

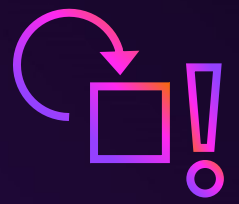
Distributed system fundamentals

Error handling and replay



Distributed system fundamentals

Error handling and replay



Poison pills and dead-letter channels

Everything fails all the time.

Dr. Werner Vogels

VP and CTO, Amazon.com

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

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

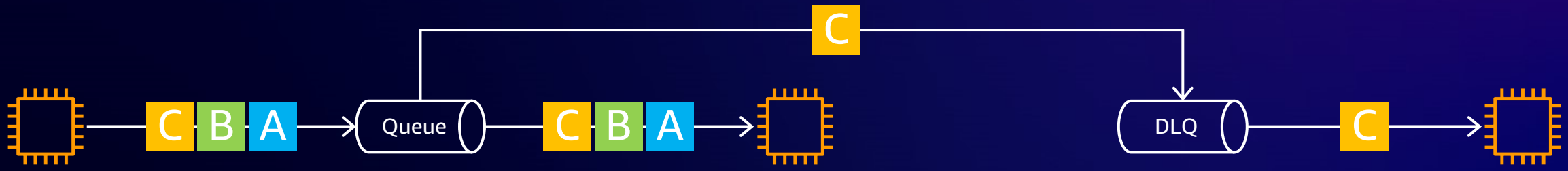
Consumers

Transient failure mitigation

Poison-pill handling

Message channels

Dead-letter queue (DLQ)



Producer

Consumers

Application ops

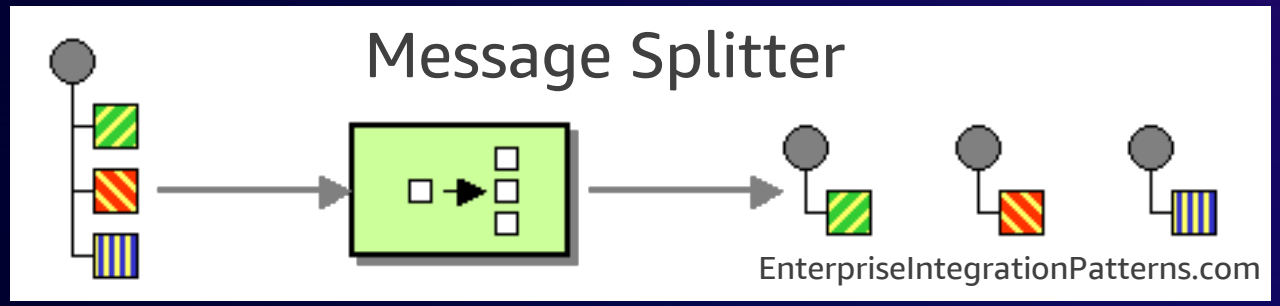
Transient failure mitigation

Poison-pill handling

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

Failure is inevitable.
Embrace failure.
And let the tools help you.

Retry logic or infinite loop?



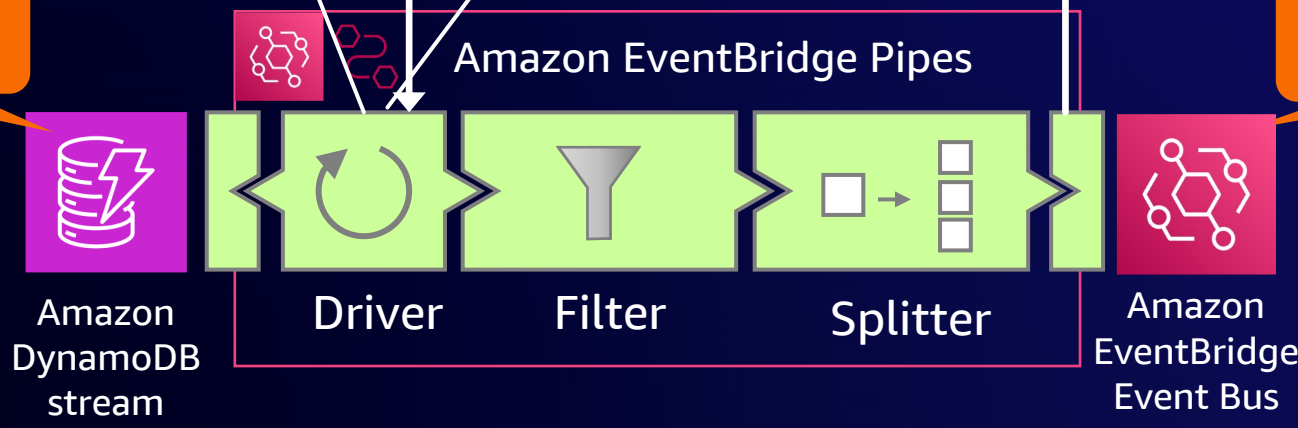
`MaximumRetryAttempts -> (integer)`
Discard records after the specified number of retries. The default value is -1, which sets the maximum number of retries to infinite.

Backs off to 1 retry per minute. Phew!

Retry!

Message with 11 elements

Batch size limit 10



No mechanism brought more distributed systems down than retry logic.

Gregor

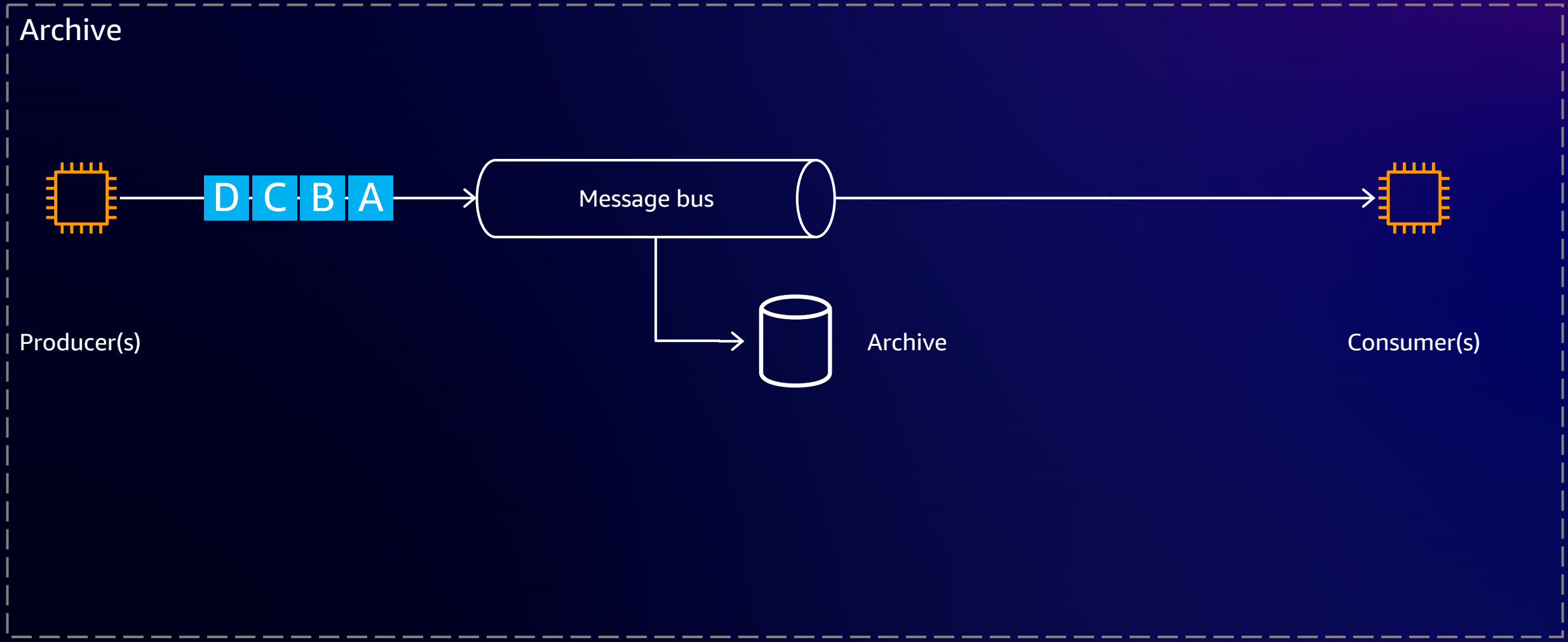
Member of the infinite loop club

Distributed system fundamentals

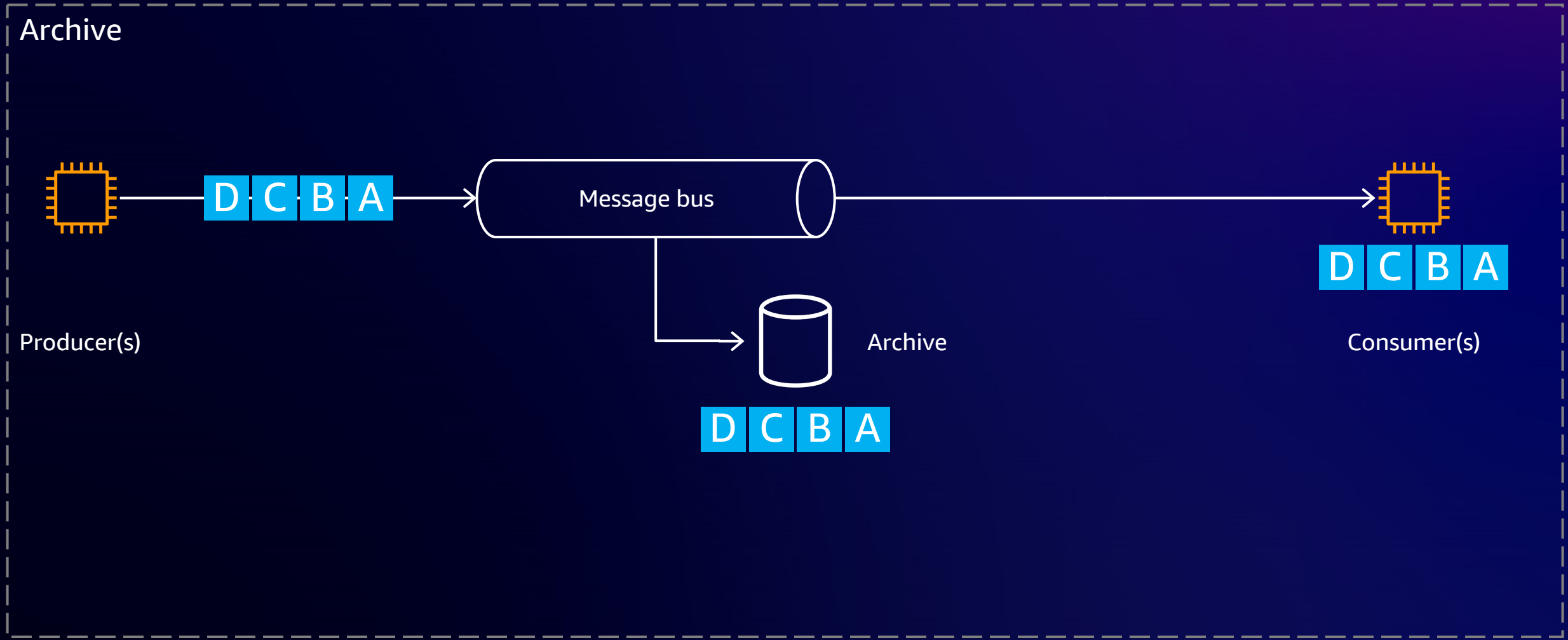
Error handling and replay

Archive and replay

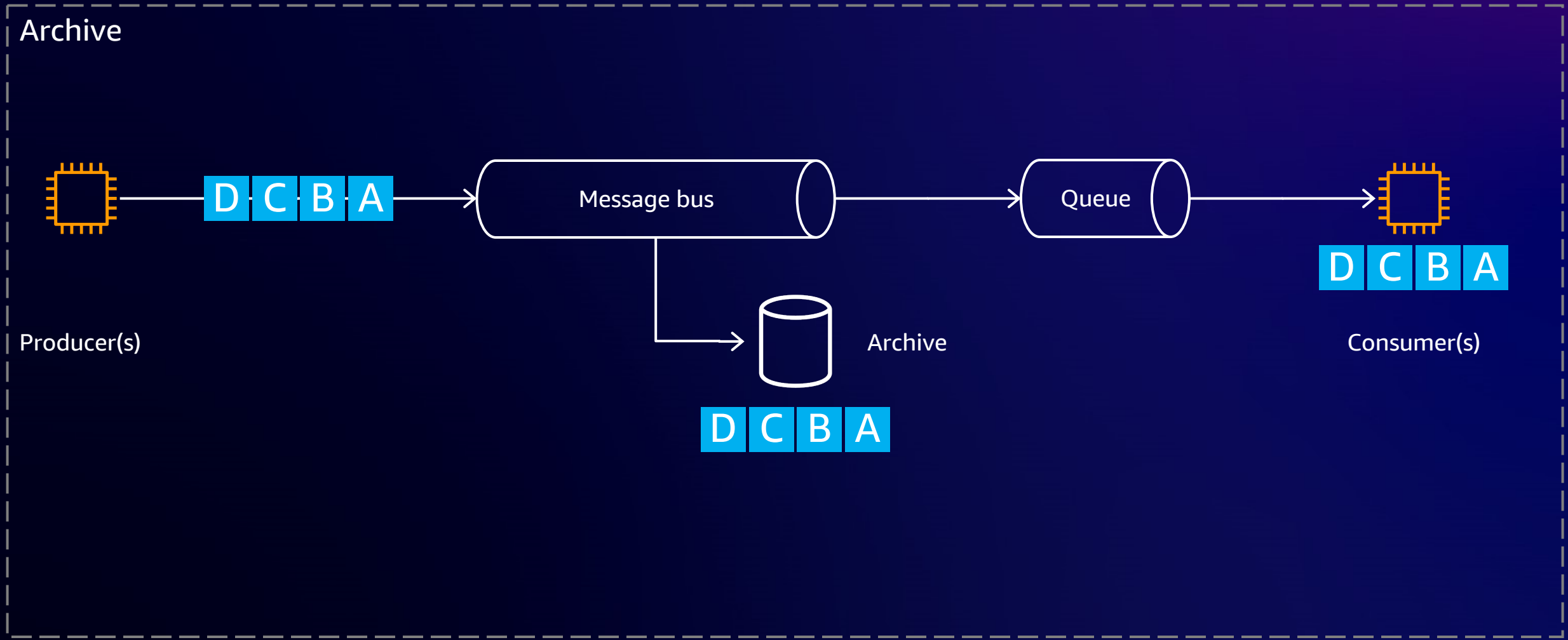
Archive and replay



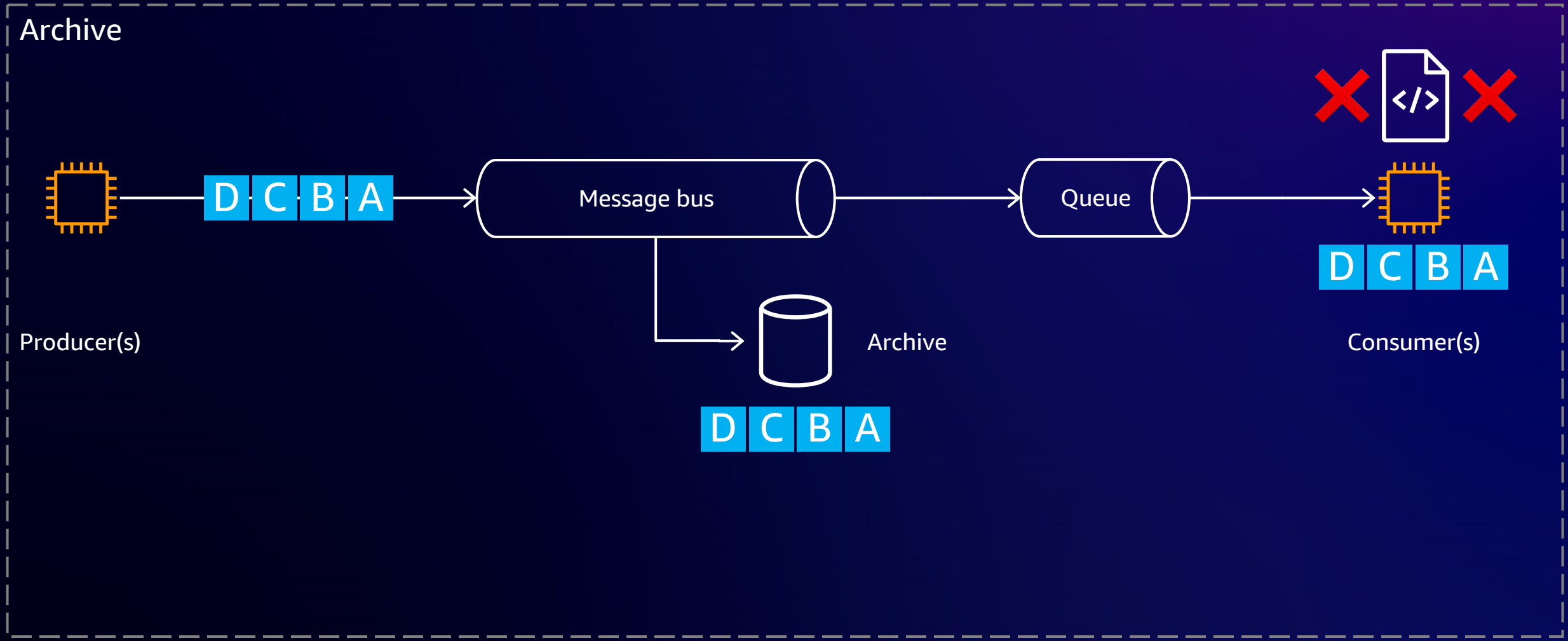
Archive and replay



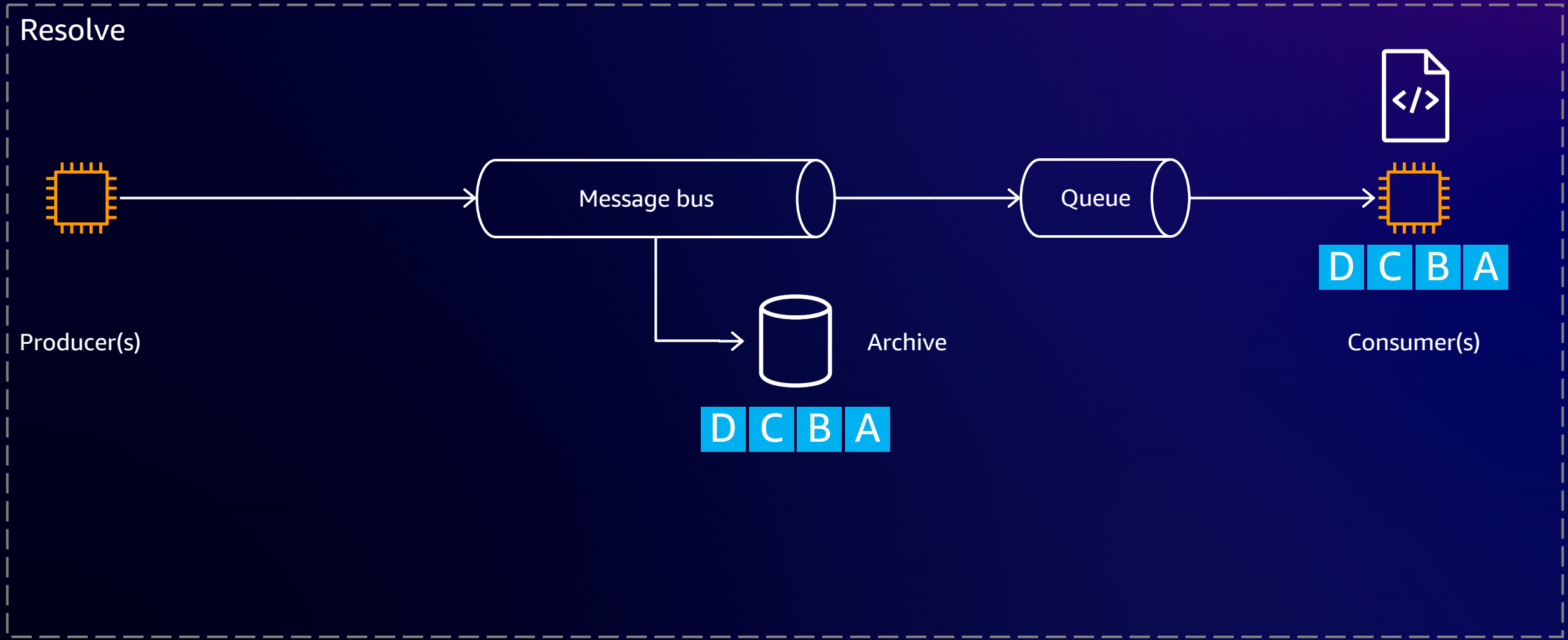
Archive and replay



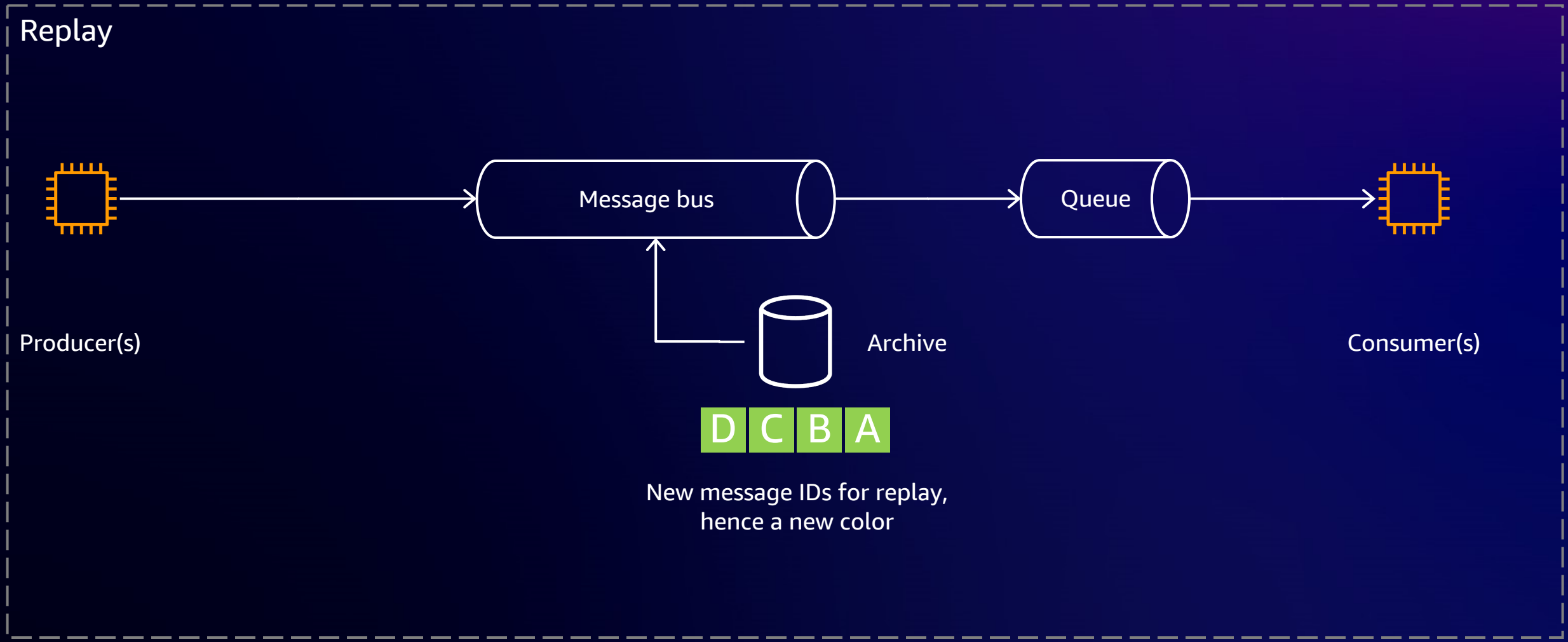
Archive and replay



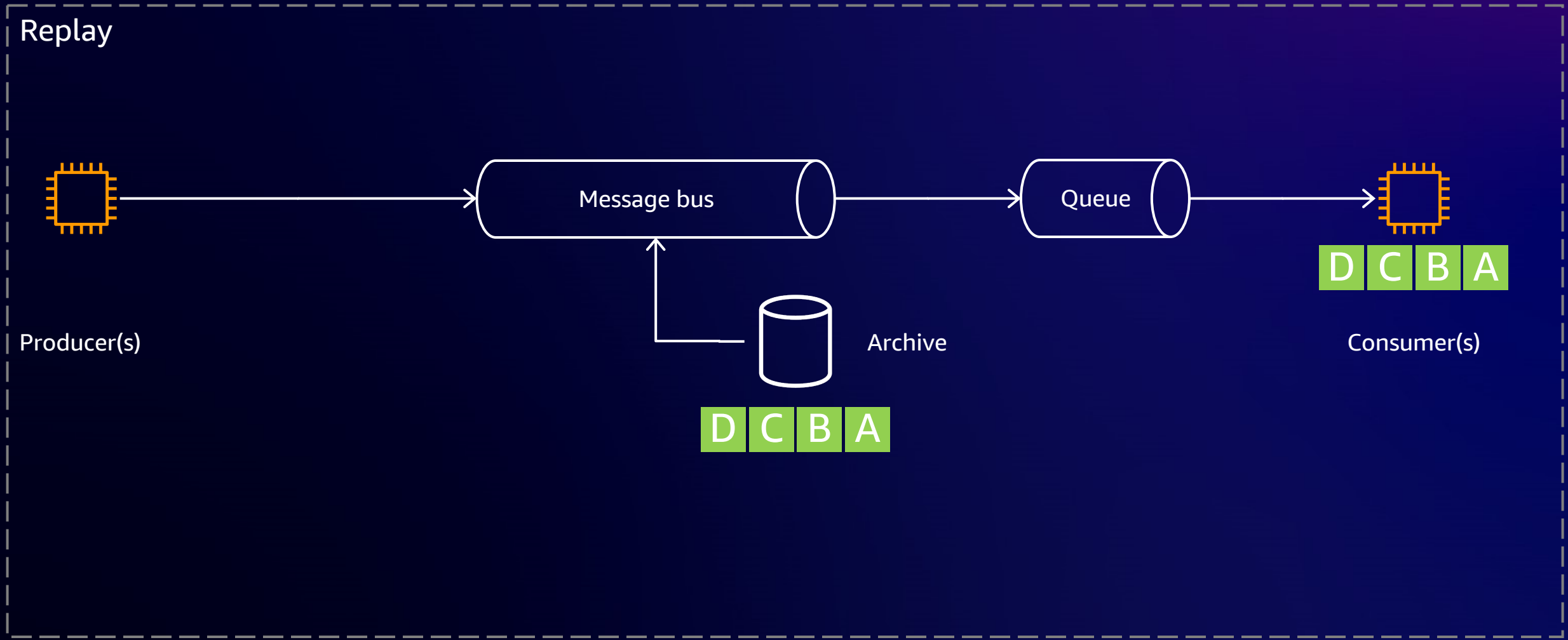
Archive and replay



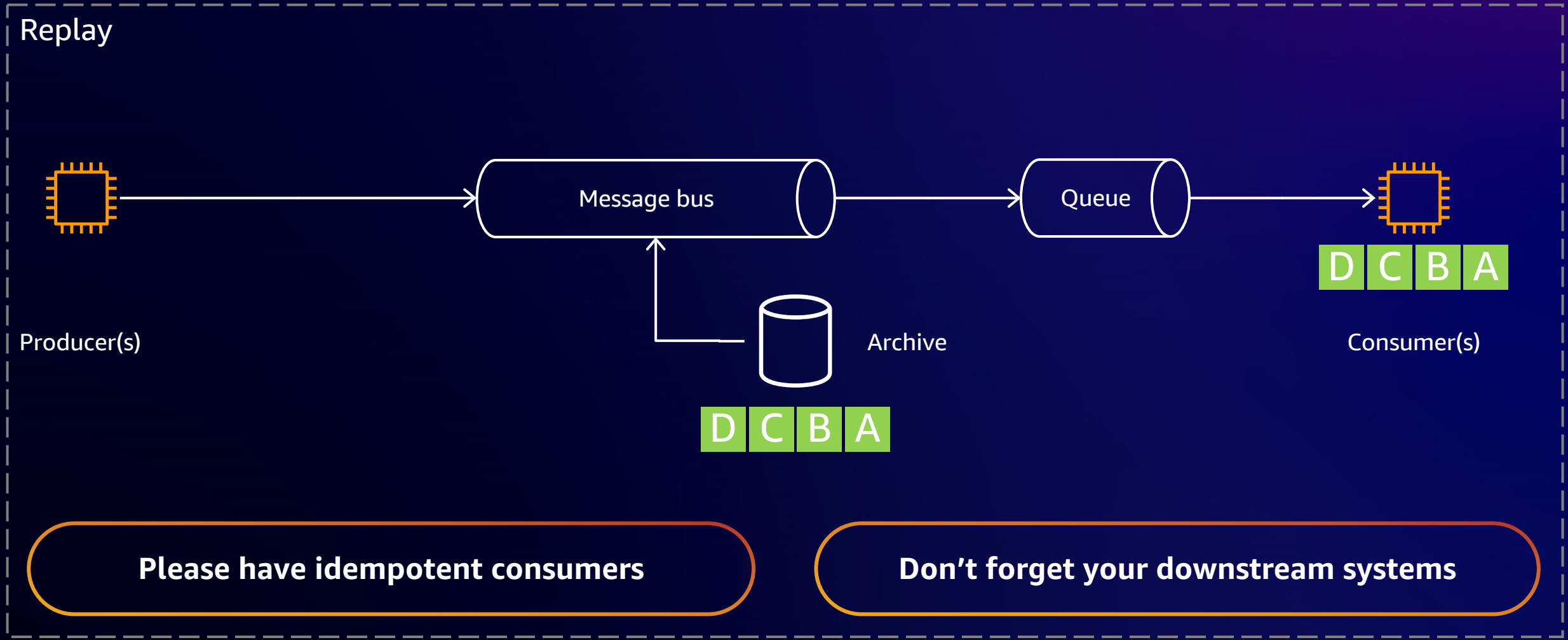
Archive and replay



Archive and replay



Archive and replay



No architecture decision you take comes **without trade-offs**. **Every** architecture decision brings some **pain**.

Your job as software architect is to identify the least painful option on the table.

Dirk
Trying to manage expectations

Distributed system design, integration patterns, and cloud automation

Cloud
– Automation
= Just another data center

An opinionated architect

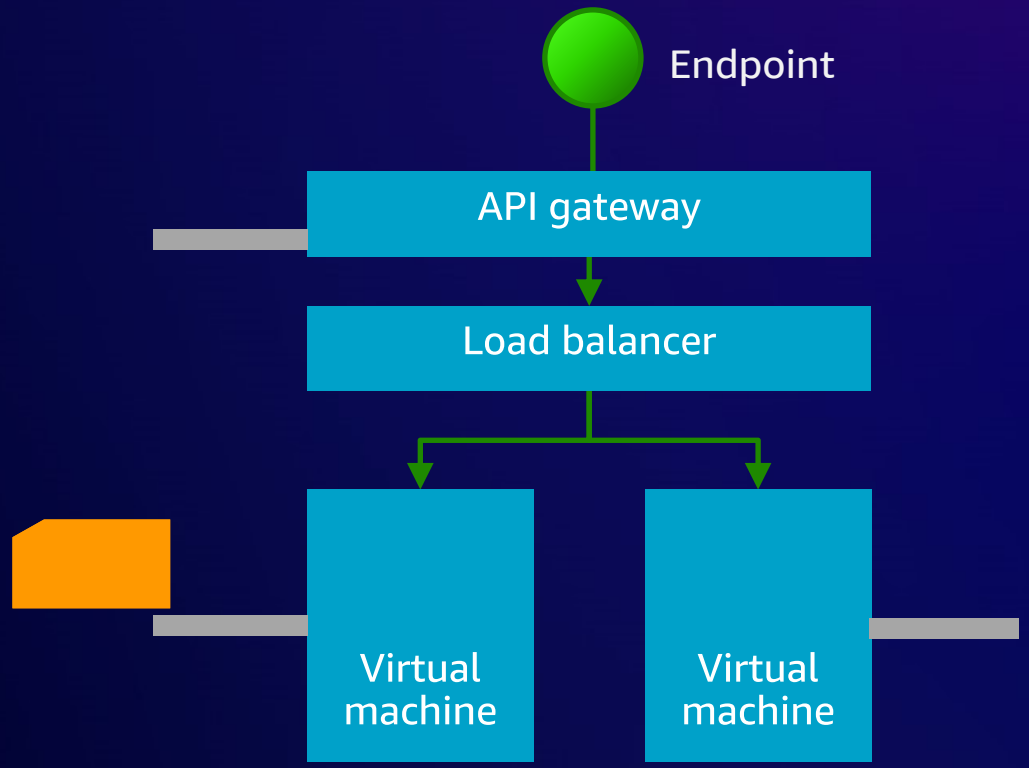
Modern automation is more than provisioning

Config

Compose

Deploy

Provision



Serverless automation:

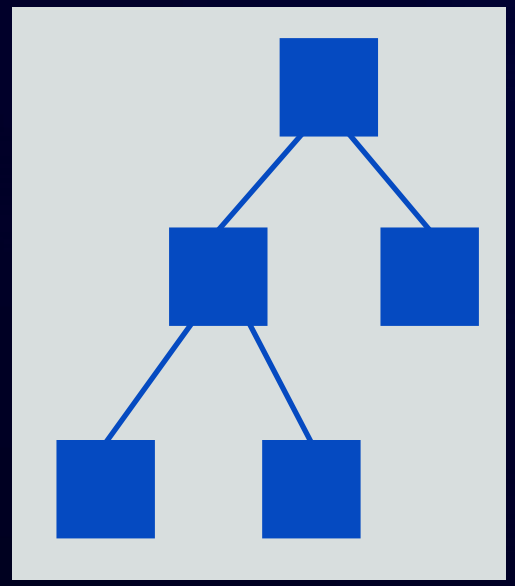
Actual*

~~Infrastructure~~ **as code**

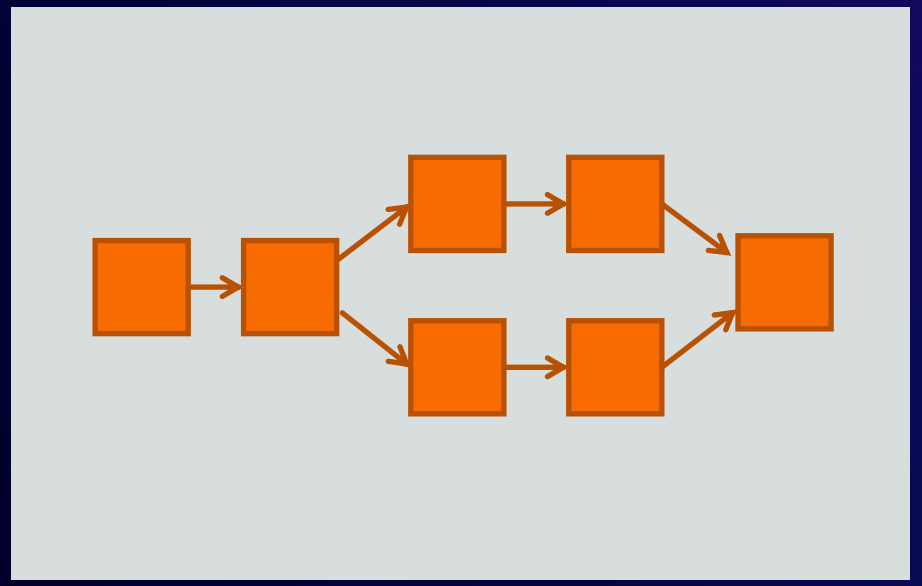
Application architecture

* Thanks, AWS CDK!

From IaC to AaC: Architecture as code



Classic IaC
(including AWS CDK):
Resource hierarchy



Serverless Architecture:
Data and control flow

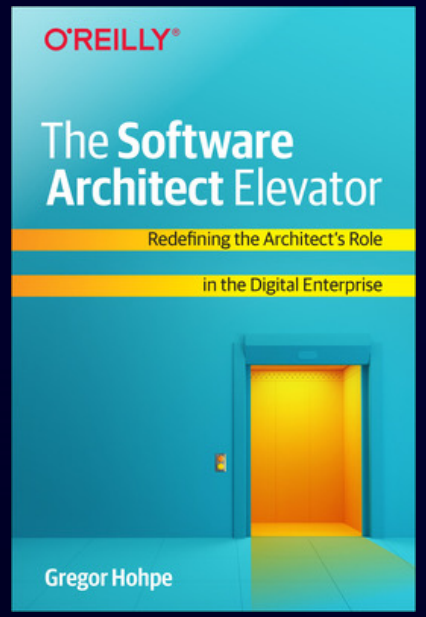
> “Serverless automation:
> 1 string, 2 integers, and 30 pages of documentation.”

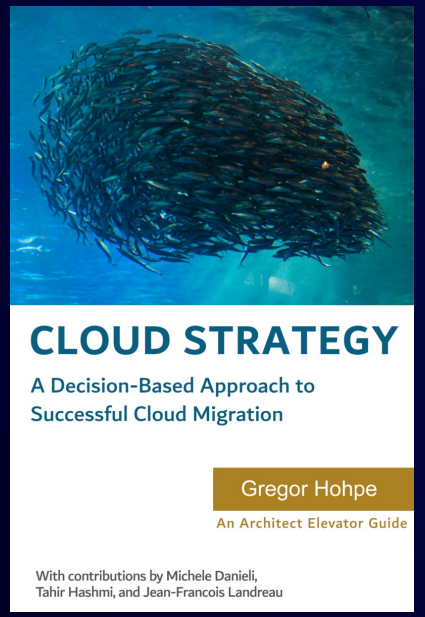
A slightly cynical cloud architect

> “From stringly typed (ARNs) to a technical domain model represented in a type system.”

A hopeful cloud architect

https://architectelevator.com/cloud/iac-ifc-trends/





ArchitectElevator.com

Redefining the role of an architect as connecting strategy and IT engine room

Hands-on workshop

Decoupled microservices/ asynchronous messaging

Material for this session (videos, articles): https://s12d.com/api309-2023

Check out these sessions

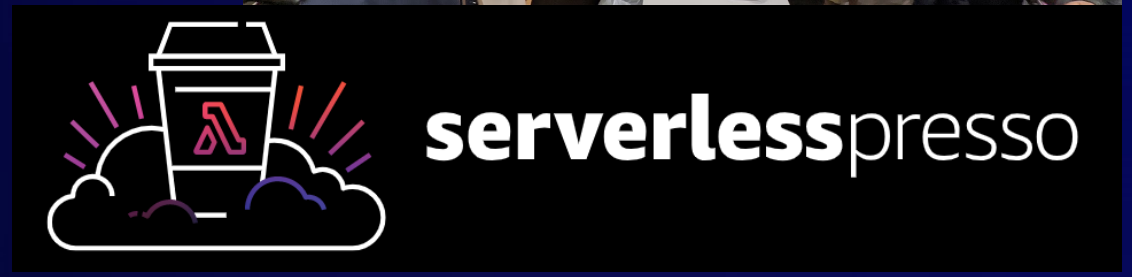
API311:
Combine messaging services for workload resilience and ordering

November 30 | 2:00 PM (PST) – Caesars Forum | Forum 104

ARC307:
Do modern cloud apps lock you in?

November 30 | 11:00 AM (PST) – The Venetian | Summit Showroom

AWS Village, AWS Modern Application and Open Source Zone



Thank you!

Please complete the session survey in the mobile app

Dirk Fröhner

@dirk_f5r

linkedin.com/in/dirk-froehner/

Gregor Hohpe

@ghohpe

linkedin.com/in/ghohpe/