Building Serverless Stream-based Event Driven Architectures with Kafka and Lambda

Veda Raman

SA, Serverless

AWS

Joseph Morais

AWS Evangelist | Staff Cloud Partner Solutions Architect

Confluent

Agenda

Streaming data

Streaming data usecases

Confluent Kafka

Serverless stream processing

Lambda consumers

Best Practices

Demo

What is streaming data?

Typical characteristics

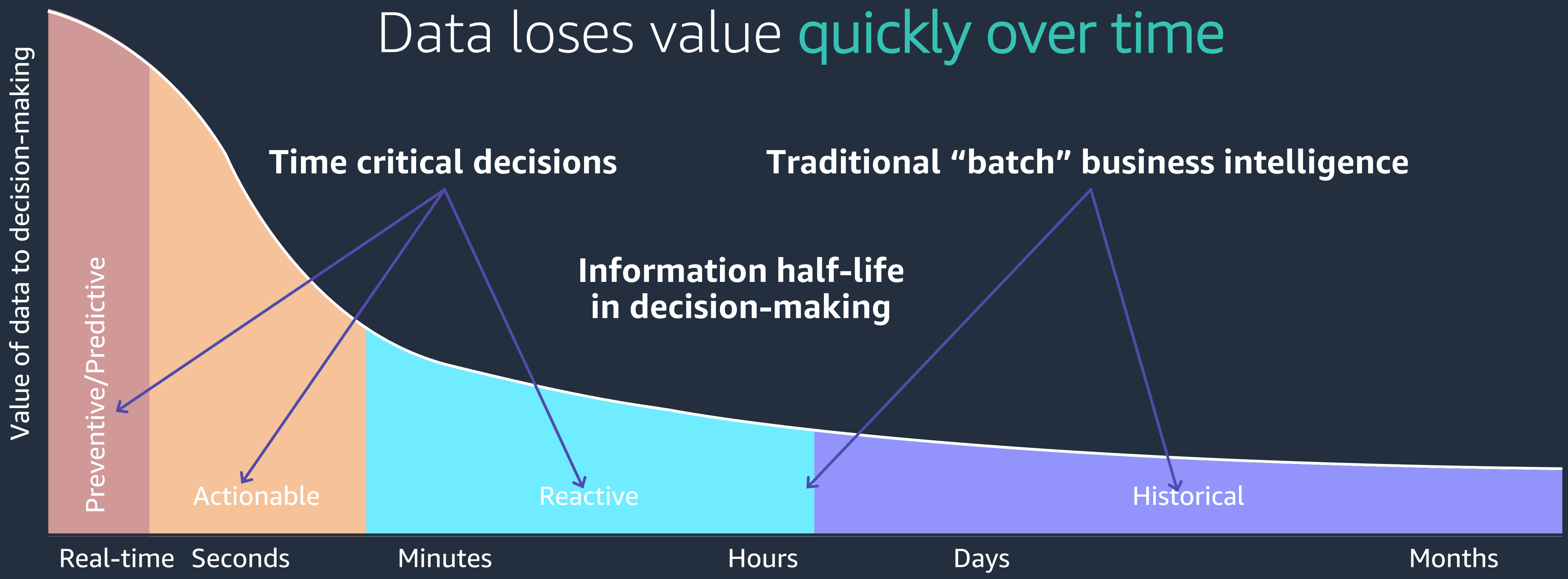
High volume

Continuous

Ordered, incremental

Low-latency

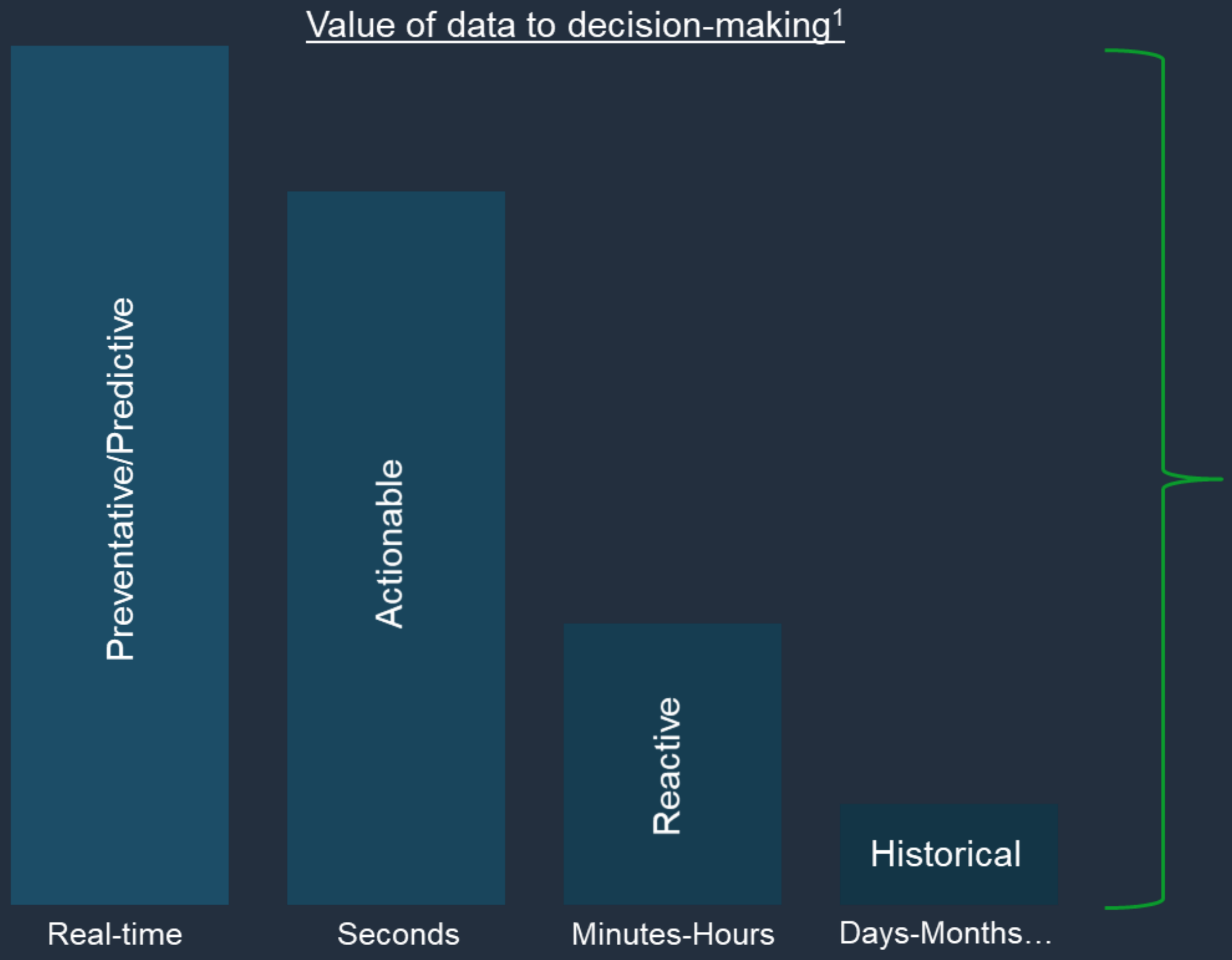
Why streaming data?



Source: Perishable insights, Mike Gualtieri, Forrester

The value of data diminishes over time

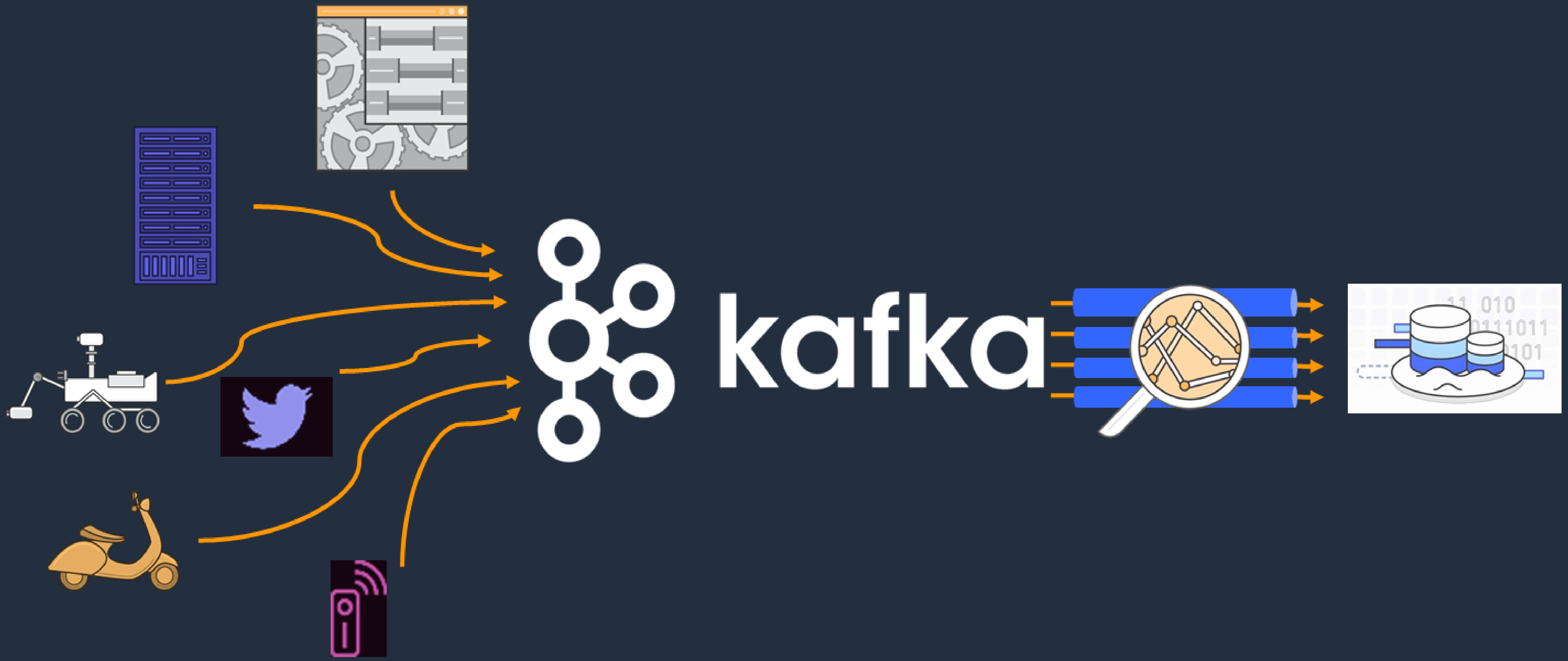
Data have a short shelf life of actionability[1]. AWS lets you act on that data as fast as the market dictates.



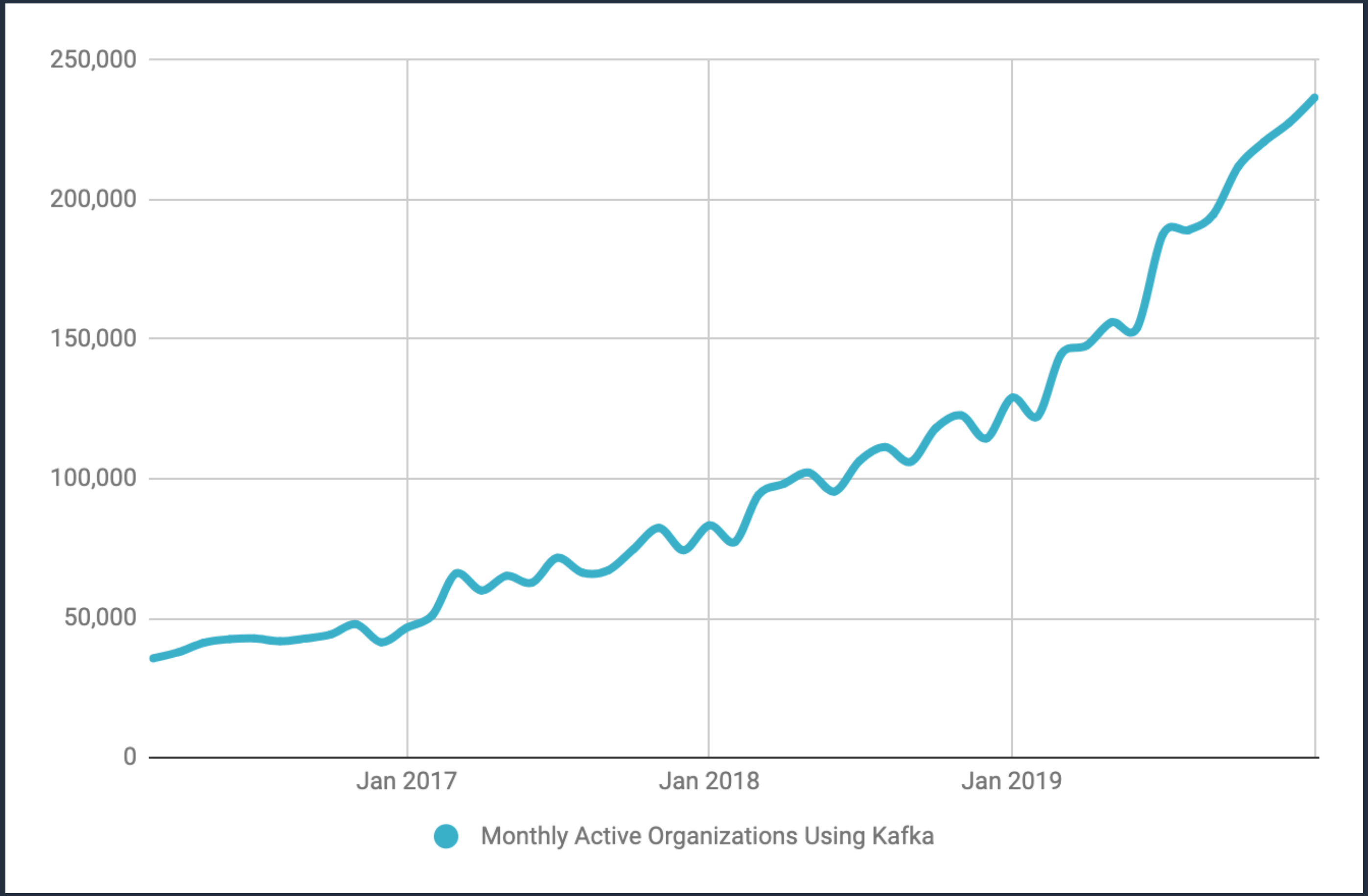
58% Percentage of those surveyed who say their organization saw **significant increases in customer retention and loyalty** as a result of using real-time customer analytics[2]

75% Percentage of enterprise organizations surveyed that believe **untimely data inhibited business opportunities**[3]

Apache Kafka



Kafka helps companies achieve their real-time goals



Kafka is in Production at

200,000+

Companies

65+%

of the Fortune 500 uses Kafka

Apache Kafka – Use Cases

- Real-time web and log analytics
- Transaction and Event Sourcing
- Messaging
- Decoupled Microservices
- Streaming ETL
- Metrics and Log Aggregation
- Streaming ML

Kafka can be run in several different ways..

- Apache Kafka on-prem(self managed)
- Confluent Platform on-prem
- Confluent Platform in the cloud
- Confluent Cloud (SaaS)
- MSK (AWS Managed service for Kafka)

Confluent Snapshot | Founded September 2014

Founded by
the Original Creators of
Apache Kafka®

Kafka is in Production at
200,000+
Companies

65+%
Fortune 500
already using Kafka

SaaS Rev-Rec partner (SRRP)
RedShift Ready Partner
ISV Accelerate Co-Sell

awsmarketplace

BENCHMARK Index Ventures SEQUOIA

Confluent runs everywhere

SELF-MANAGED SOFTWARE

Confluent Platform

The Enterprise Distribution of Apache Kafka

In the datacenter

FULLY-MANAGED SOFTWARE

Confluent Cloud

Apache Kafka Re-Engineered for the Cloud

In the cloud

Start streaming with Kafka within minutes

	Basic Get started with scale to $0 pricing	**Standard** Production-ready for most applications	**Dedicated** Customizable for any application
Sizing	No sizing required Stream up to 100MBps Store up to 5TB	No sizing required Stream up to 100MBps Store up to 5TB	Limits based on provisioned capacity
Replication options	Single AZ	Single & Multi AZ	Single & Multi AZ
Uptime SLAs	99.5%	99.95%	99.95%
Private networking options	-	-	VPC/VNet Peering AWS Transit Gateway AWS Private Link
	Prototyping, early development, and early production use cases	Production use cases streaming below 100MBps	Mission-critical applications at any scale

Mix and match any cluster type across your organization

Serverless app integration

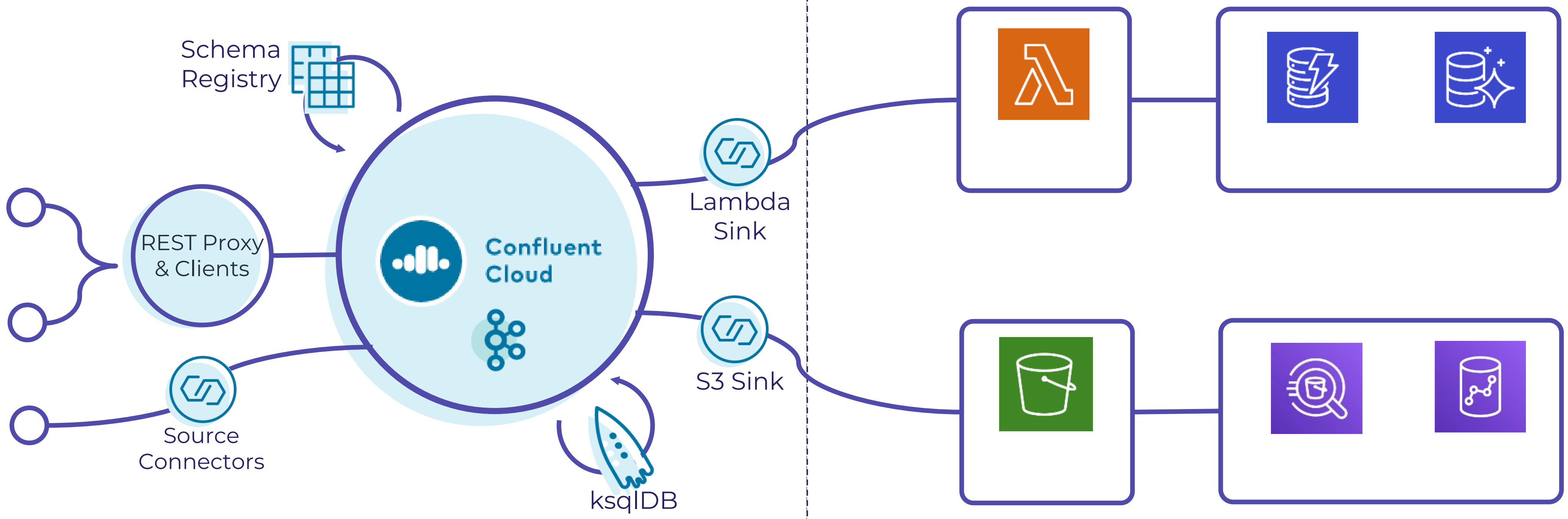
Increase developer agility & speed of innovation

Serverless integration

Connect existing and apps & data stores in a repeatable way without having to manage- **Apache Kafka**, **Schema Registry** to maintain app compatibility, **ksqlDB** to develop real-time apps with SQL syntax and **Connect** for effortless integrations with Lambda & data stores

AWS serverless platform

Stop provisioning, maintaining or administering servers for backend components such as compute, databases and storage so that you can focus on increasing agility and innovation for your developer teams



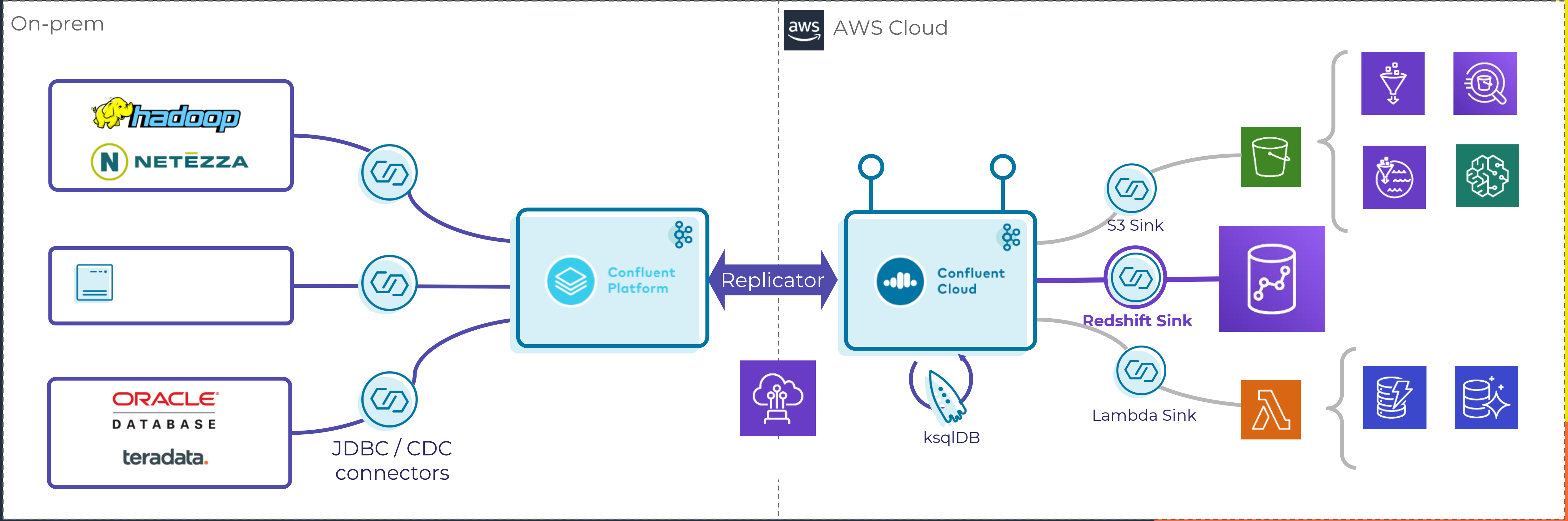
On-prem to AWS modernization

Accelerate modernization from on-prem to AWS

Connect
Leverage **+100 Confluent pre-built connectors** to continuously bring valuable data from existing services on-prem including enterprise data warehouse, databases and mainframes

Bridge
Hybrid cloud streaming with consistent, event-driven architecture for modern apps

Modernize
Increase agility in getting applications to market and **reduce TCO** when freeing up resources to focus on value generating activities and not in managing servers



Serverless Stream Processing

What is serverless?

No provisioning,
no management

Automatic
scaling

Pay for value

Highly available
and secure

Benefits of Serverless stream processing

- No servers to manage
- Only pay for stream consumption when processing messages
- Automatically scales consumers
- Write less code

Serverless processing

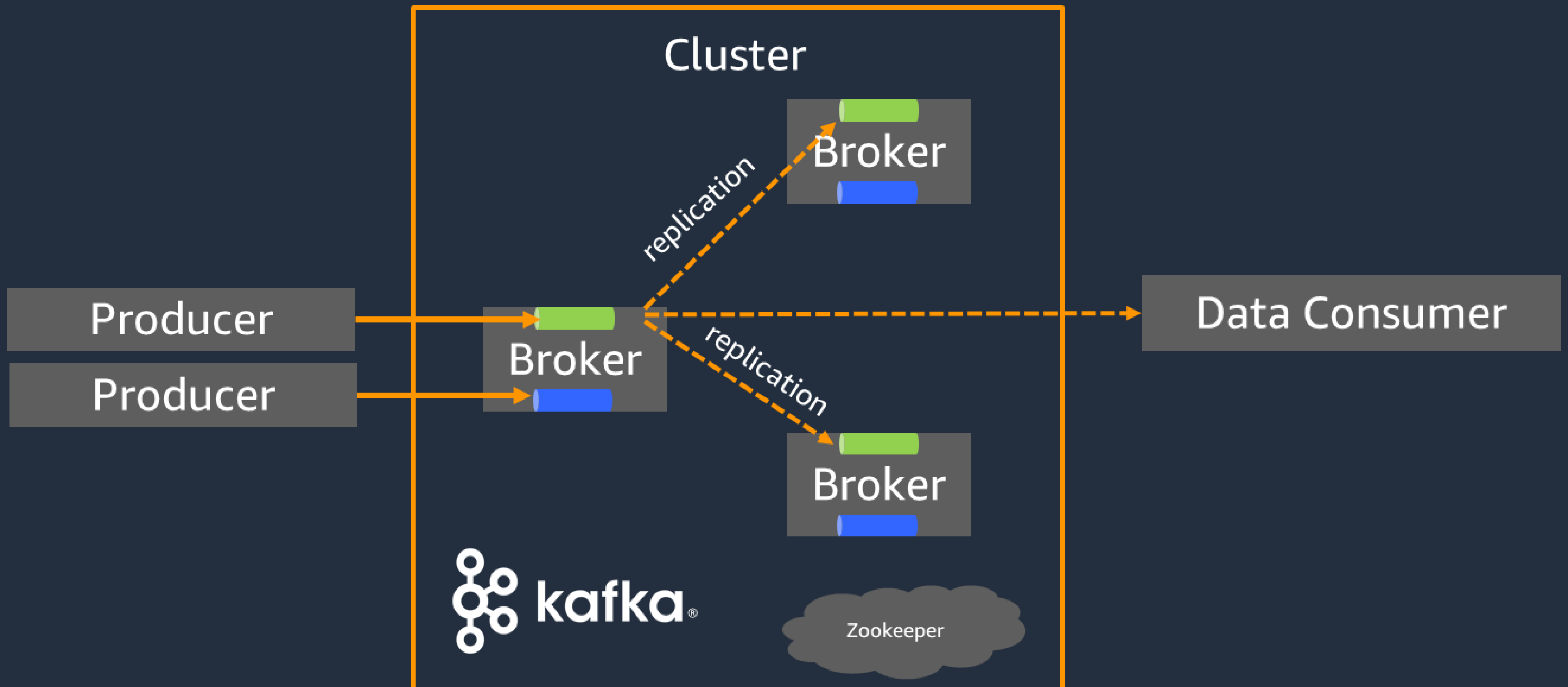
- ✓ Stream polling logic is separate from application logic
- ✓ Event driven processing
- ✓ Scaling is handled automatically
- Poller: Lambda ESM

  Confluent Lambda Sink Connector

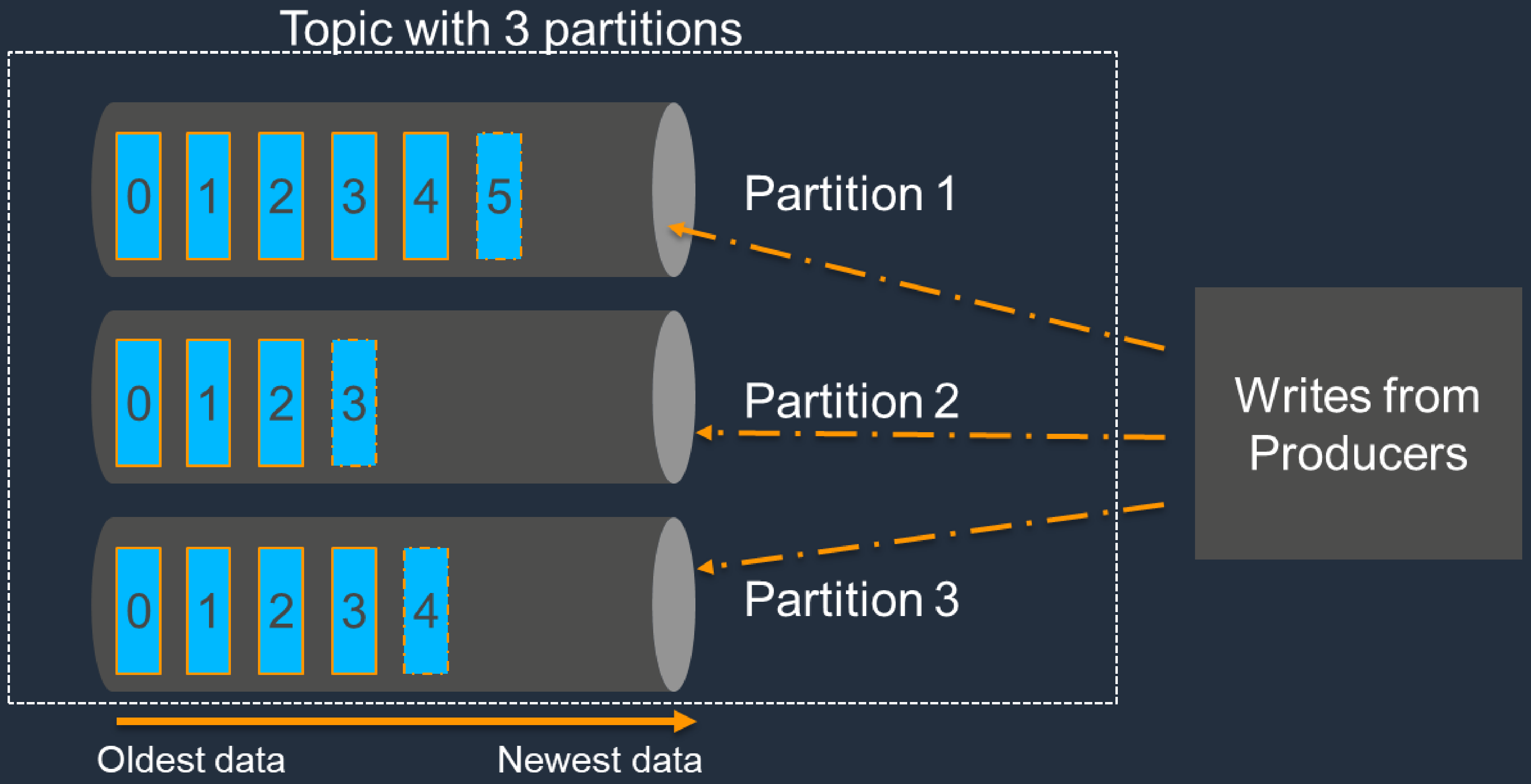
Server-based processing

- ✓ Stream polling logic is baked into your application code
- ✓ Consumer must be running to poll the kafka clusters
- ✓ Scaling is done using consumer groups.
- ✓ Poller: Open source APIs/libraries(KafkaStreams javalibrary, kafka-python)

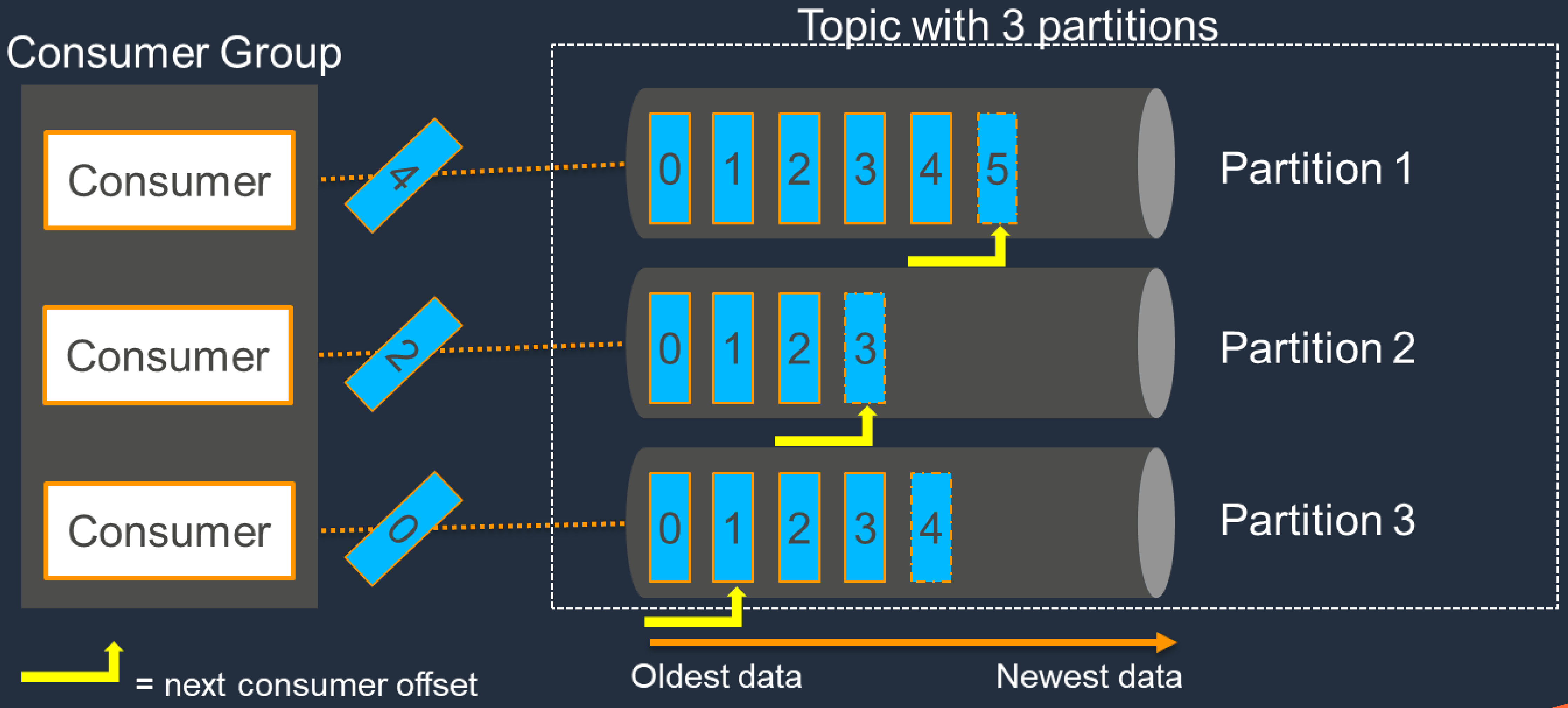
Apache Kafka anatomy 101



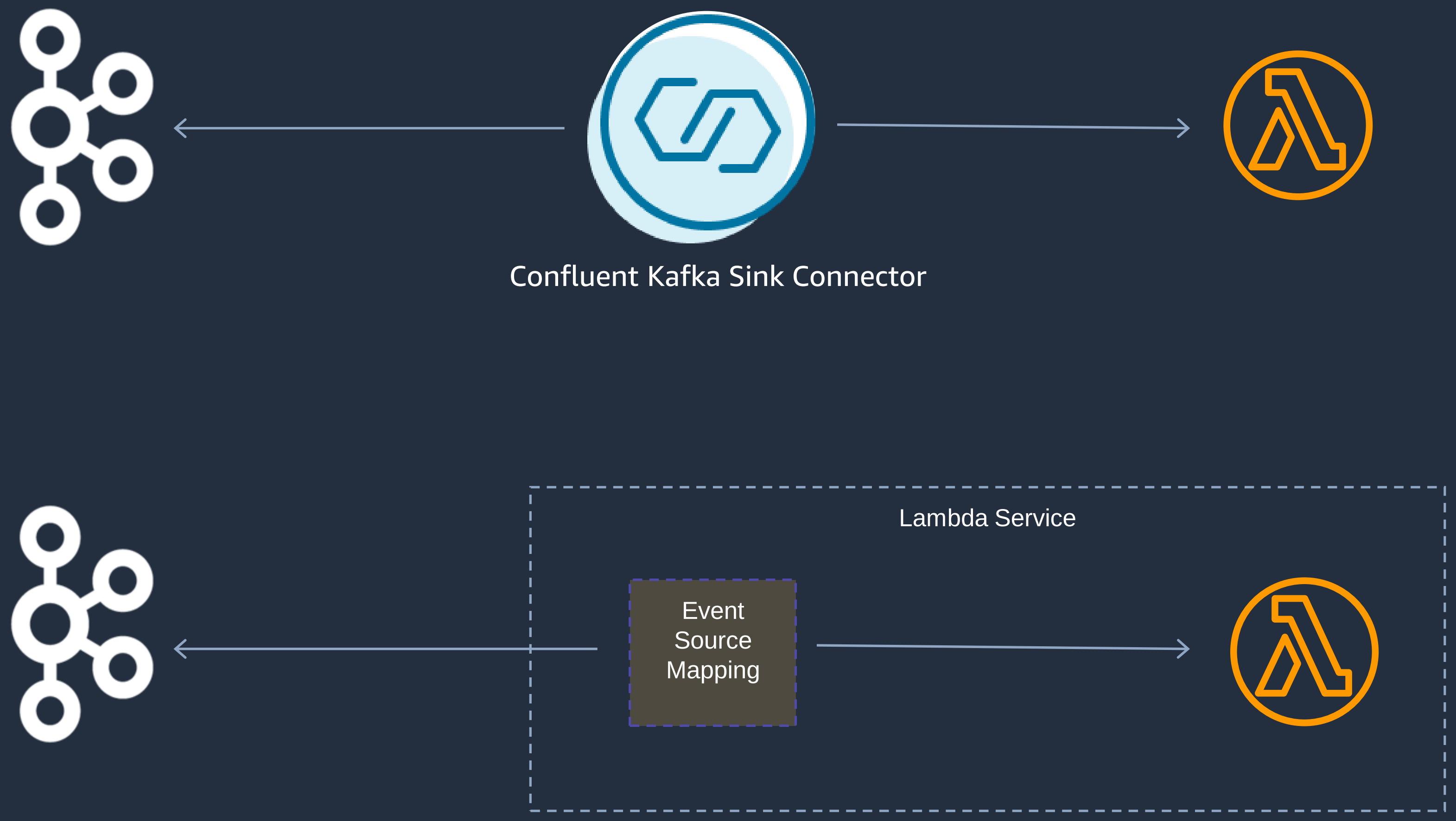
Apache Kafka – Writes to partitions



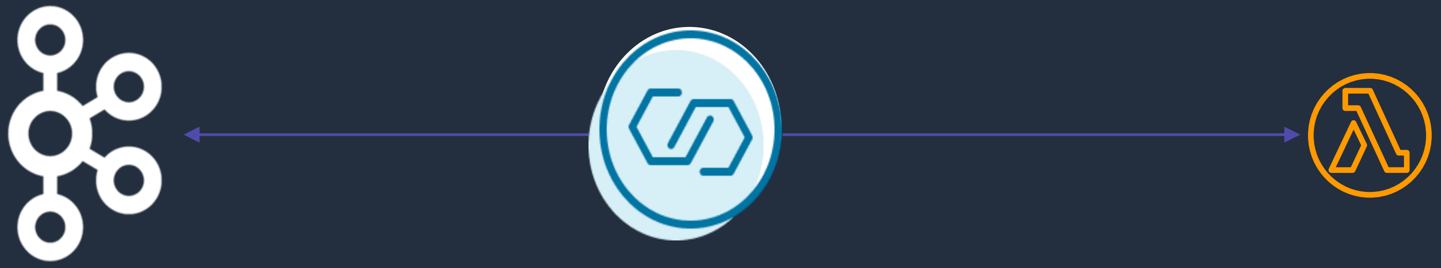
Apache Kafka – Reads from partitions



Lambda consumer options

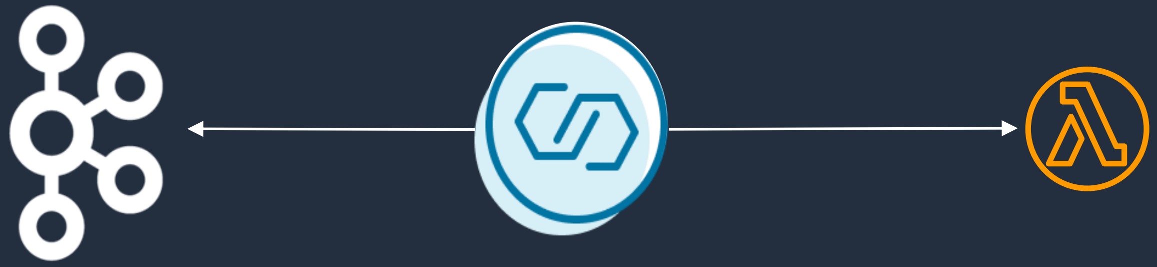


Confluent Lambda Sink connector



- Sink connector polls Kafka partitions and calls your function
- Lambda can be called synchronously or asynchronously.
- Atleast once semantics
- Provides a dead letter queue (DLQ) for any failed invocations

Confluent Lambda Sink connector – Scaling and Error Handling

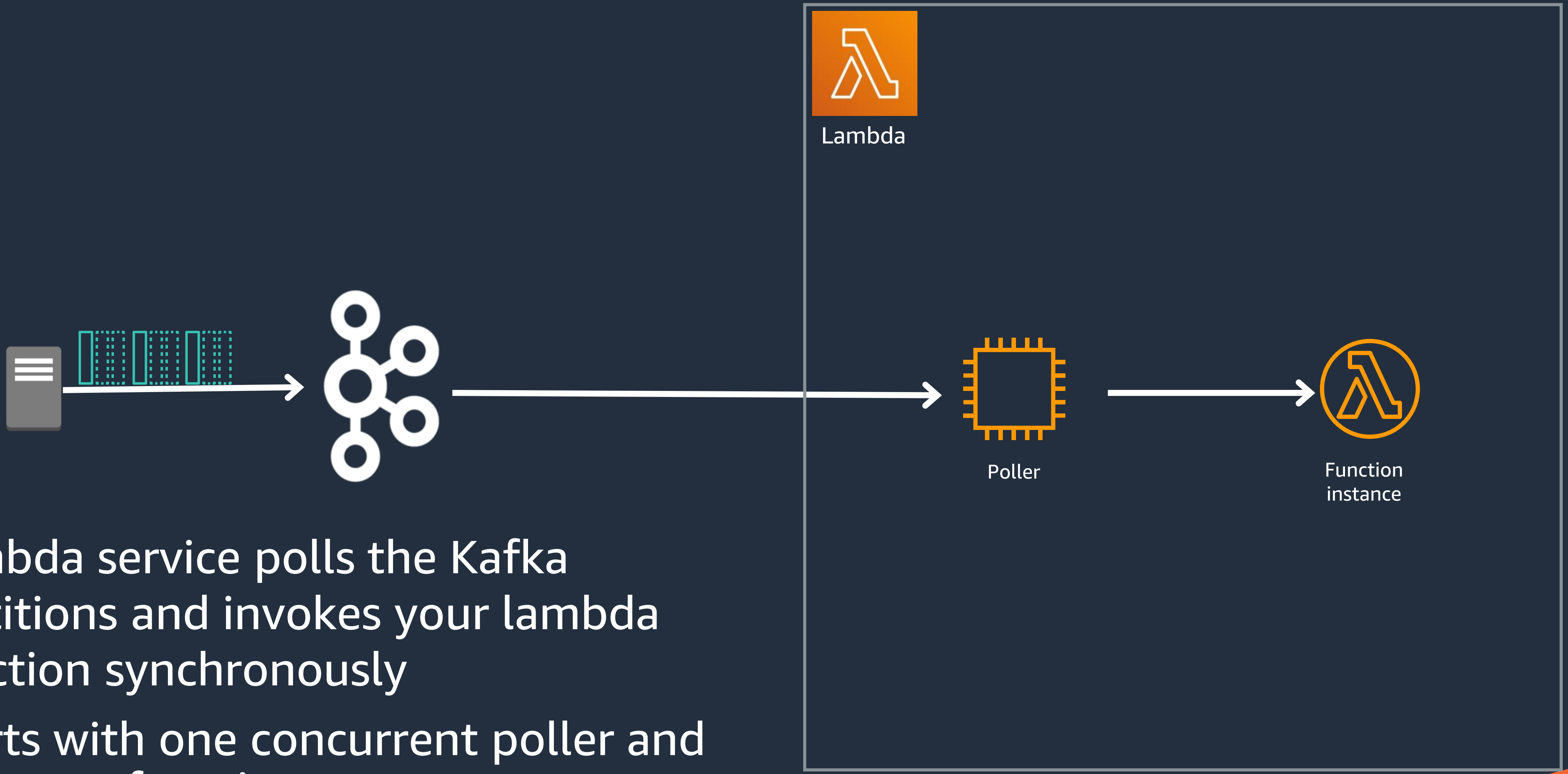


- Sink connector scales upto a soft maximum of 10 connectors.

- Error handling semantics similar to sync and async lambda invocations.
  - Async: Lambda service retries twice (three total attempts)
  - Sync: By default, fails and stop processing for that partition.
    Option to log to another kafka topic and continue processing

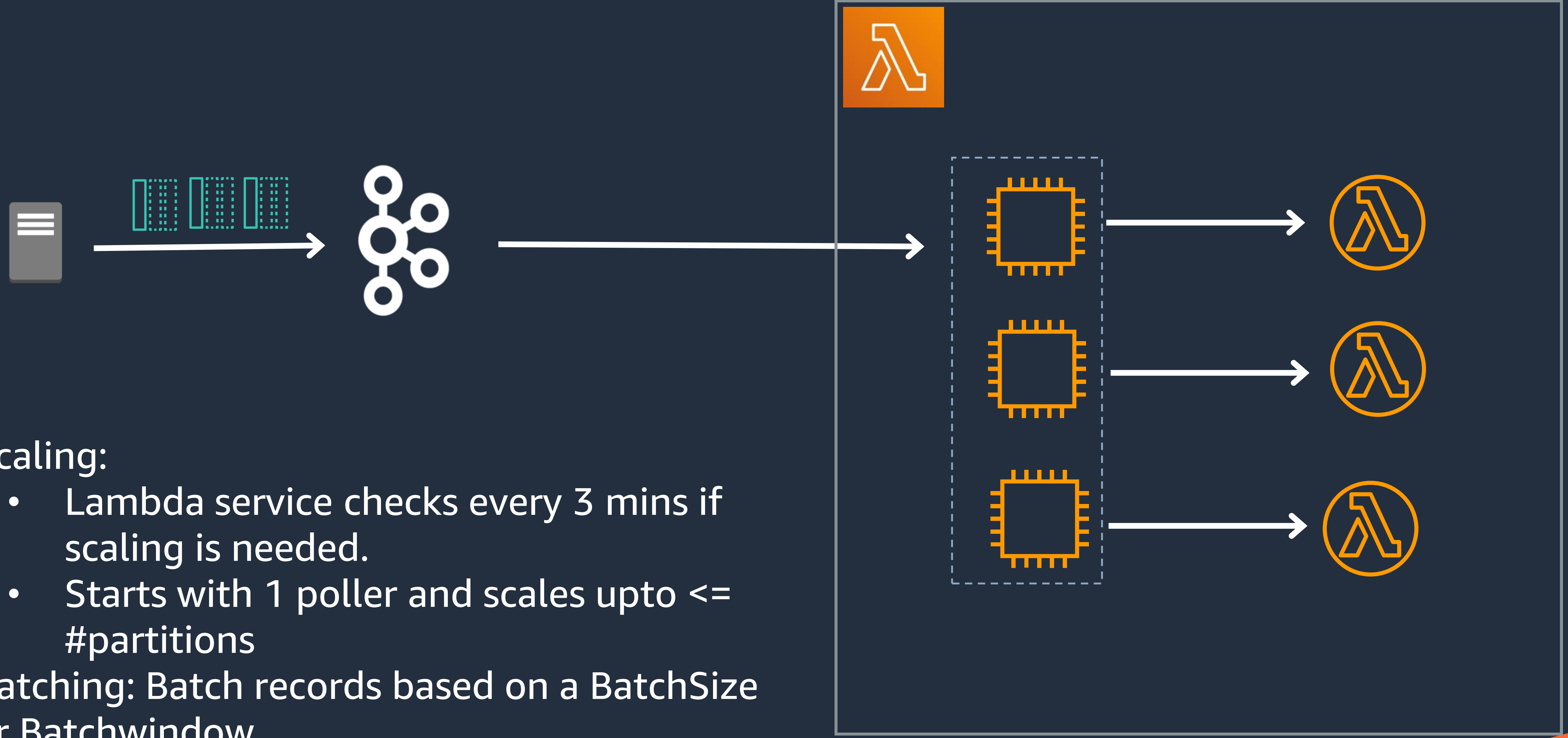
- Option to batch records. Configured through aws.lambda.batch.size

Lambda ESM consumer for Kafka



- Lambda service polls the Kafka partitions and invokes your lambda function synchronously
- Starts with one concurrent poller and customer function

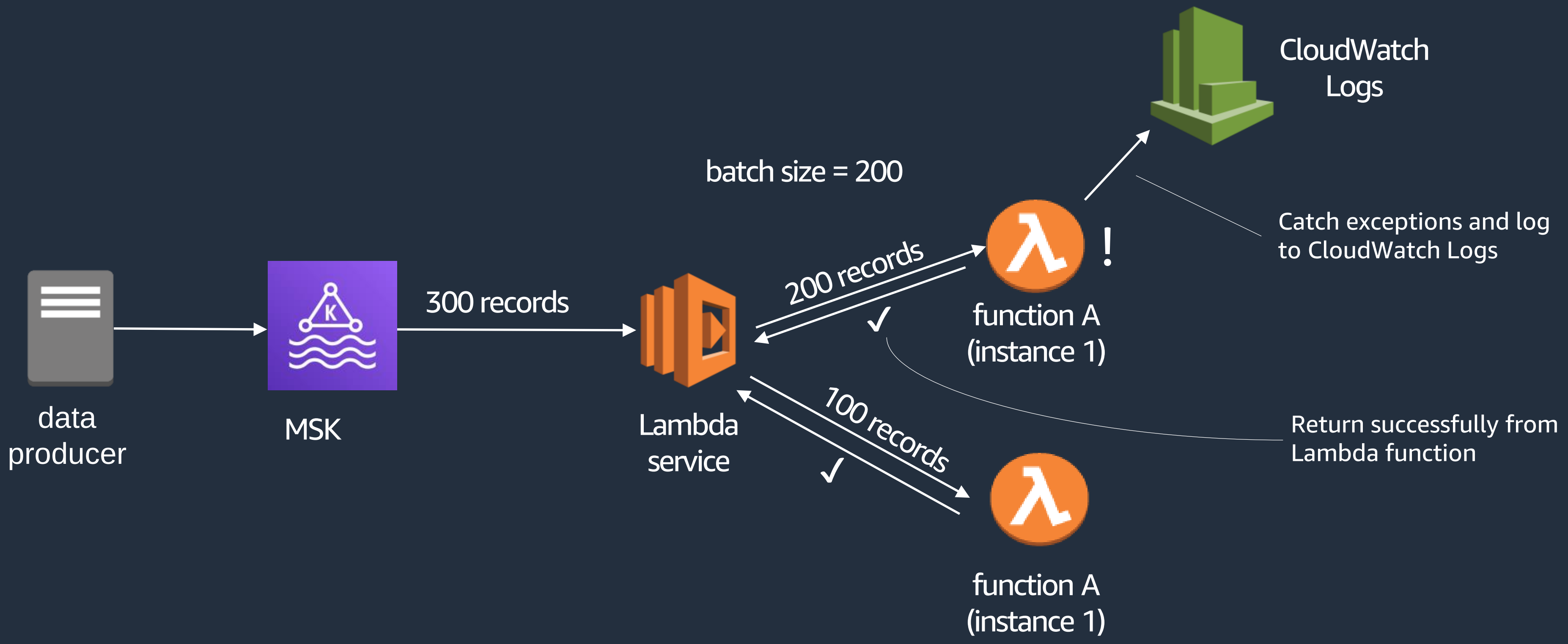
Lambda ESM consumer for Kafka – Scaling and Batching



- Scaling:
  - Lambda service checks every 3 mins if scaling is needed.
  - Starts with 1 poller and scales upto <= #partitions
- Batching: Batch records based on a BatchSize or Batchwindow.

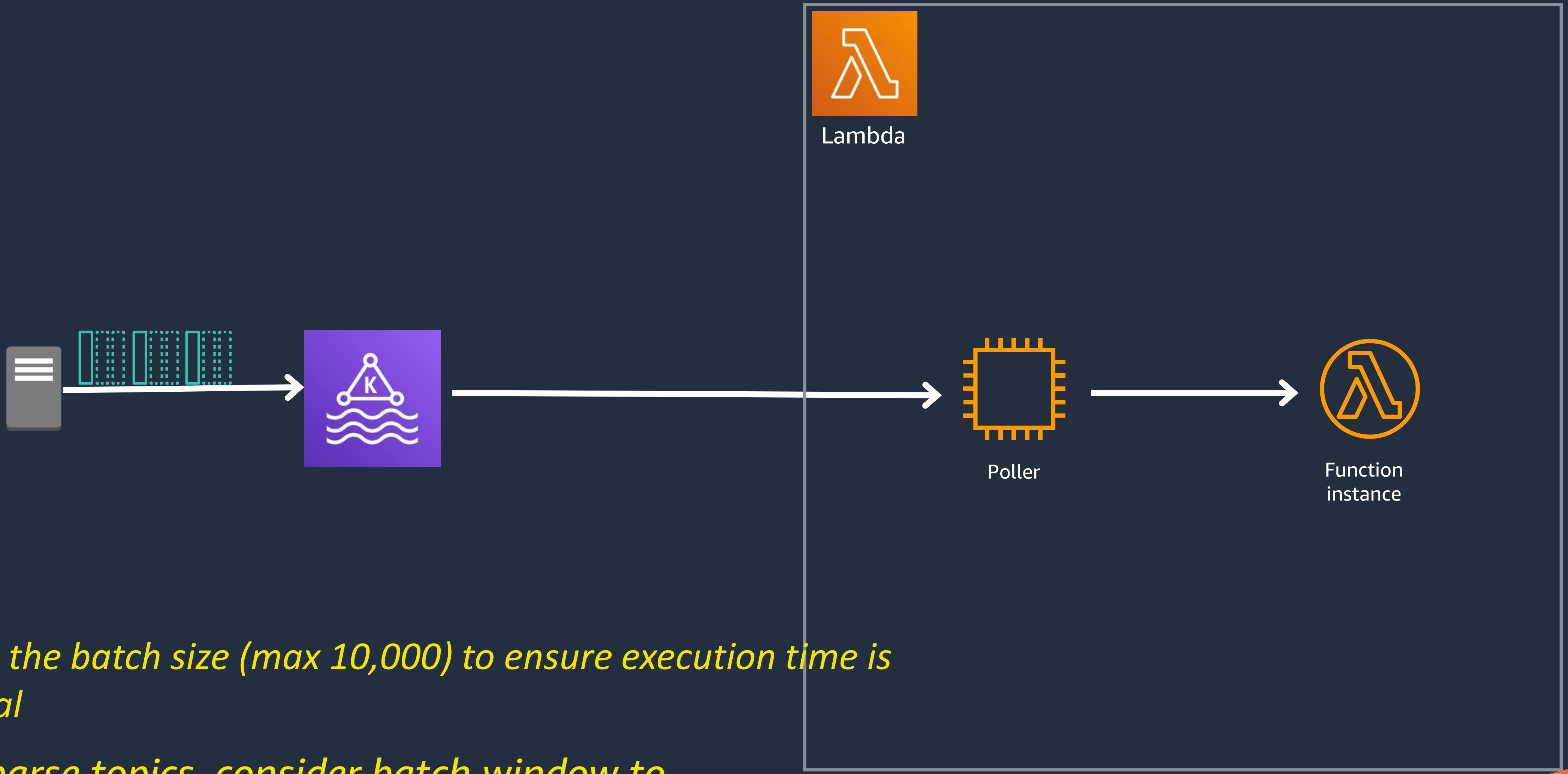
Best Practices

Capture and log exceptions



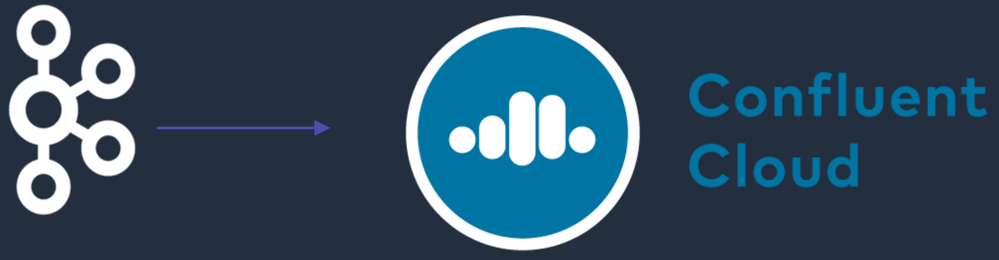
- *Ensure processing moves forward by catching exceptions and returning successfully*

Optimize batch-size/batch-window to lower cost

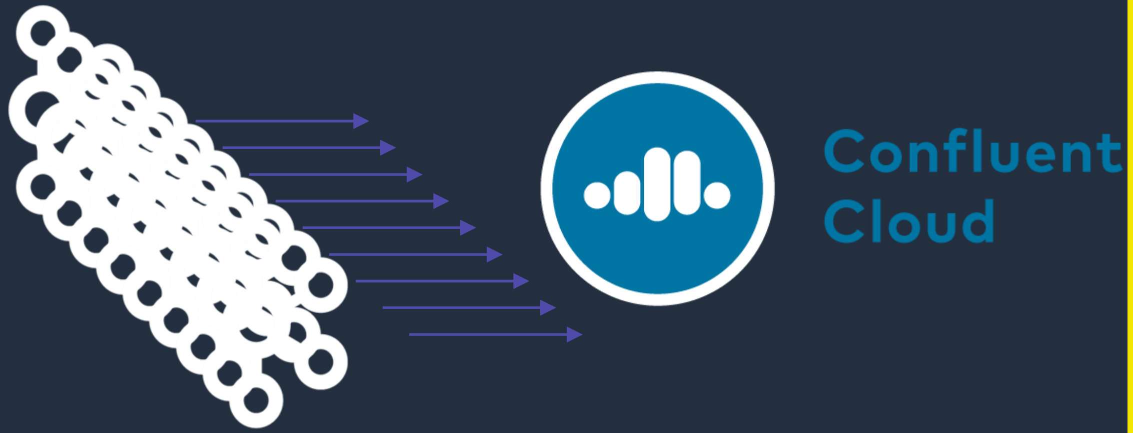


- *Adjust the batch size (max 10,000) to ensure execution time is optimal*
- *For sparse topics, consider batch window to aggregate over a time period*

Kafka Producer in Lambda (create once, use many)



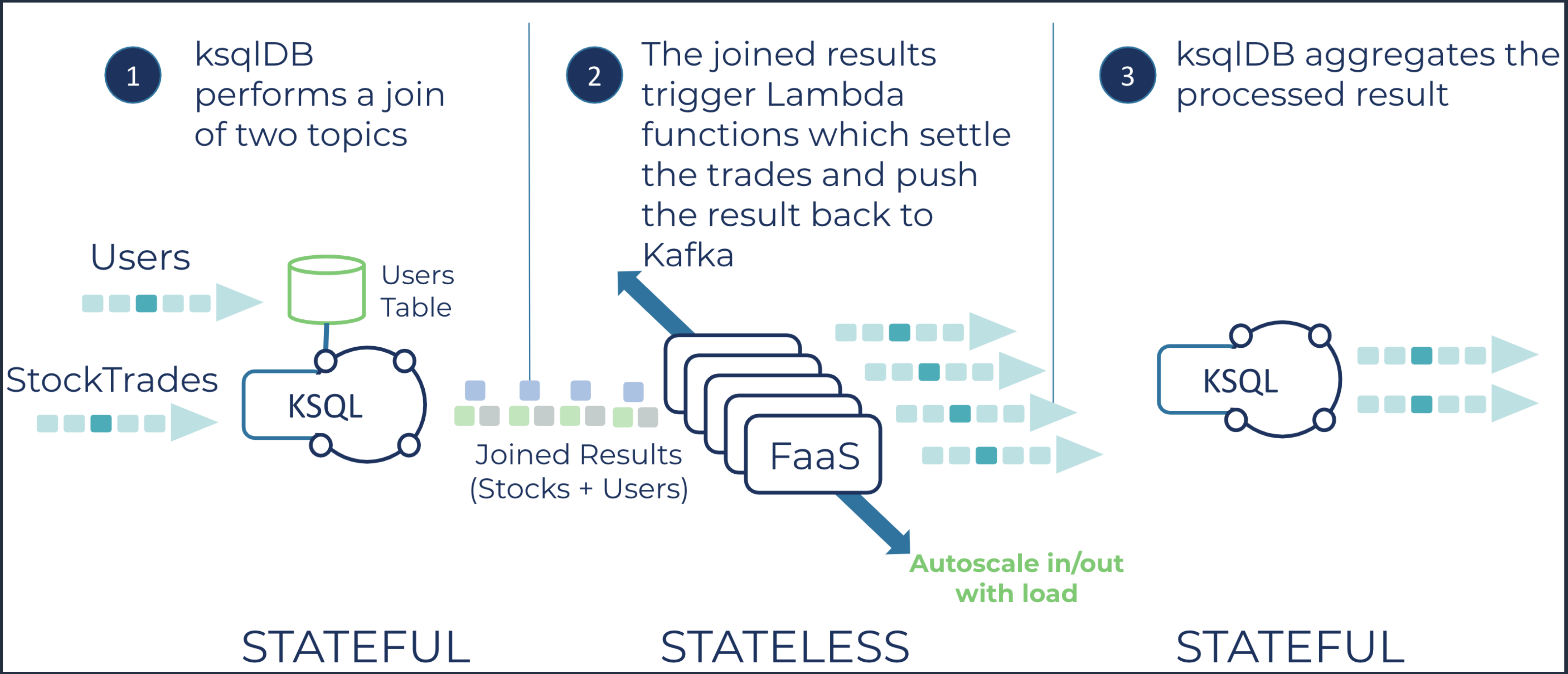
This



Not this

- *Producer will be re-used across executions for the life of the Lambda instance*
- *Reduce strain on brokers by minimizing connections and producer clients*

Consider using ksqlDB for state


1
ksqlDB performs a join of two topics
2
The joined results trigger Lambda functions which settle the trades and push the result back to Kafka
3
ksqlDB aggregates the processed result
Users
Users Table
StockTrades
KSQL
Joined Results (Stocks + Users)
FaaS
Autoscale in/out with load
KSQL
STATEFUL
STATELESS
STATEFUL

- *A powerful combination of ksqlDB and Lambda provides a stateful -> stateless -> stateful pattern*

DEMO

Thank you!