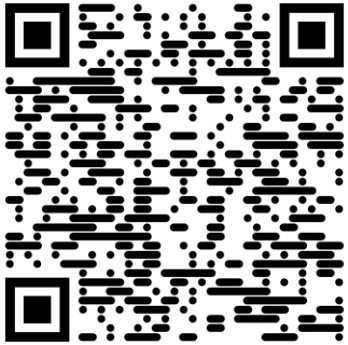
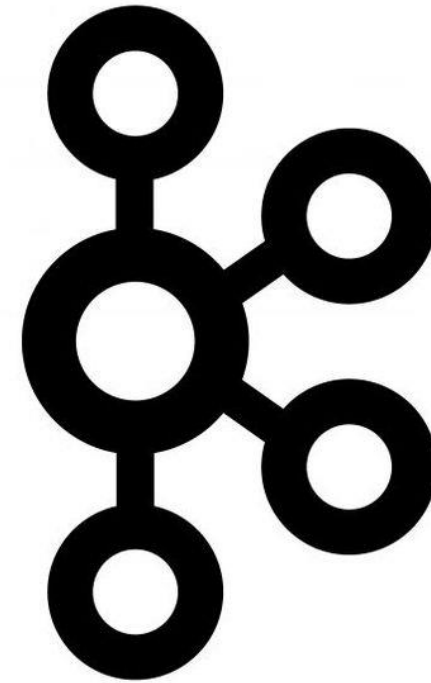


# Streaming Data at Scale

## with Kafka Consumer Groups



Slides and Demos





## Justin Reock

Tech Community Thought Leader  
Field CTO / Chief Evangelist

Justin has over 20 years of experience working in various software roles including JEE work. He is an outspoken free software evangelist, delivering enterprise solutions, technical leadership, various publications and community education on databases, architectures, and integration projects.



[@justinreock](https://www.linkedin.com/company/justinreock)



SCAN ME

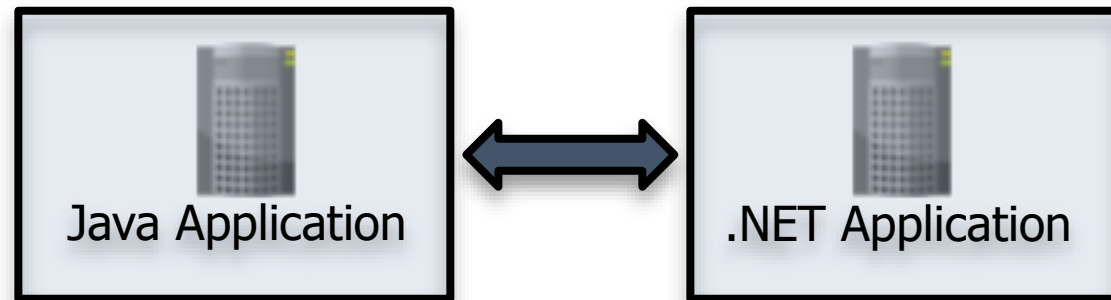


**“Where there is data smoke,  
there is business fire.”**

- Dr. Thomas Redman, President, Data Quality Solutions

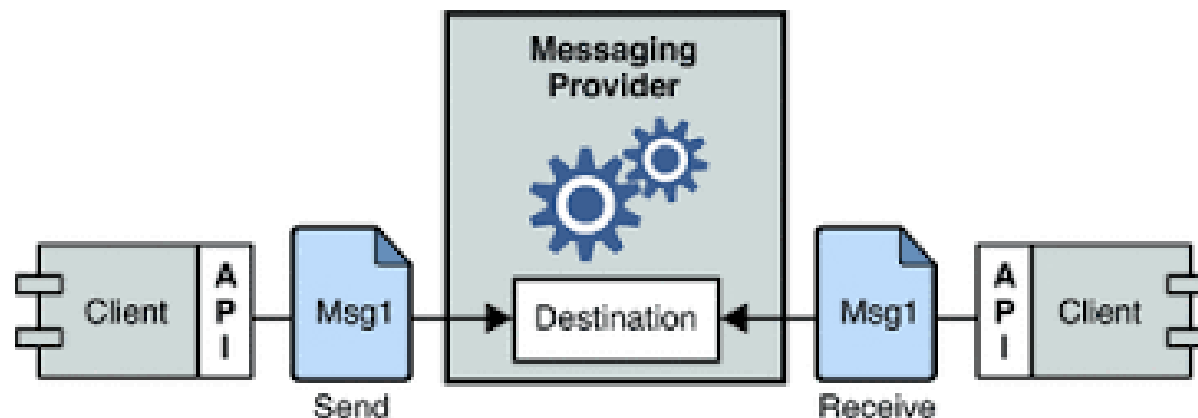
# Sharing Application Data Reliably is Difficult

- Applications often have a need to **send information back and forth** to one another
- It was once **difficult if not impossible** to “federate” data across disparate languages or residing on heterogeneous platforms
- Java Messaging Service (JMS), for example, arose out of a growing **need to share data between very different systems**



# Message Oriented Middleware To The Rescue

- Technically, MOM is any **pattern** that **sends and receives messages** between distributed applications
- This pattern allows for **asynchronous processing**, and **normalization** of data exchanges
- Clients connect to a **messaging provider**, and send and receive messages via that provider



# Message Oriented Middleware Platforms



## Message Broker

- A **Message Broker** is an application responsible for systematically providing data in the form of a **Message** between endpoints

## Producers and Consumers

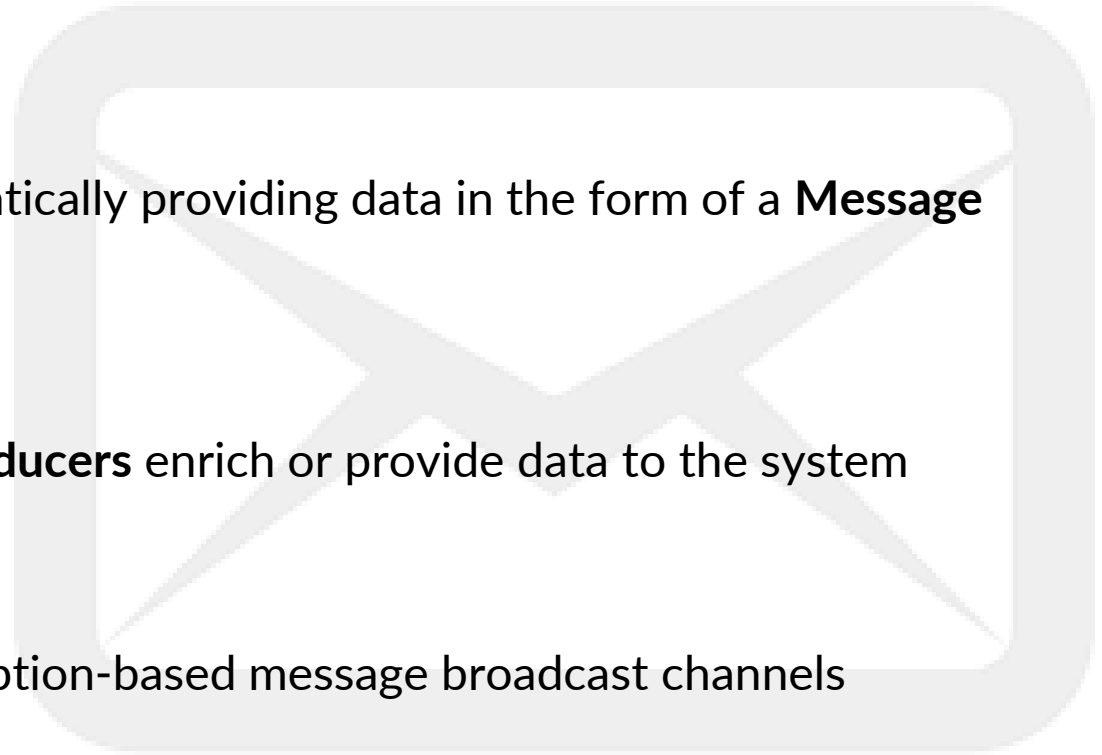
- **Consumers** receive data from a messaging system, and **Producers** enrich or provide data to the system

## Queues and Topics


- **Queues** are FIFO pipelines of messages, **Topics** are subscription-based message broadcast channels (Newsstand vs. Paper Route)

## Persistence and Durability

- **Persistent** messages are written to disk for HA and no loss, **Durable** subscribers have message retained per-subscriber



# Message Oriented Middleware Concepts

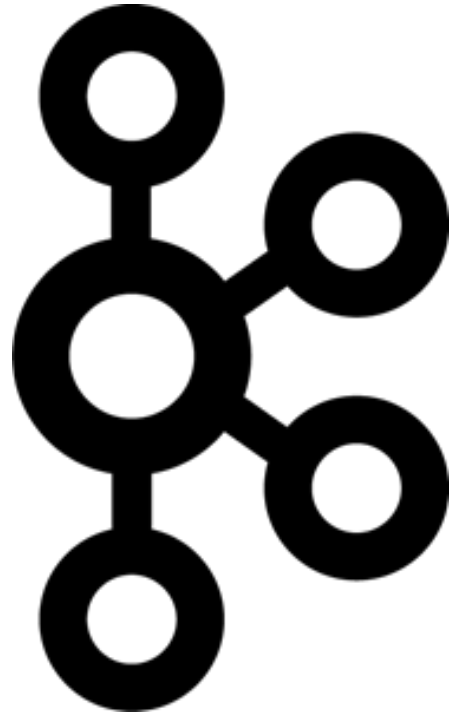
- 
- Producer Backpressure
  - Journal Health
  - Broker Availability
  - Data Security
  - Exceptions
  - Dead Letters

- Distributed Transactions
- Low-Bandwidth Environments
- Stale Data
- Protocol Compatibility
- Geolocational Distance
- Contract Enforcement

## Challenges with Traditional Message Oriented Middleware



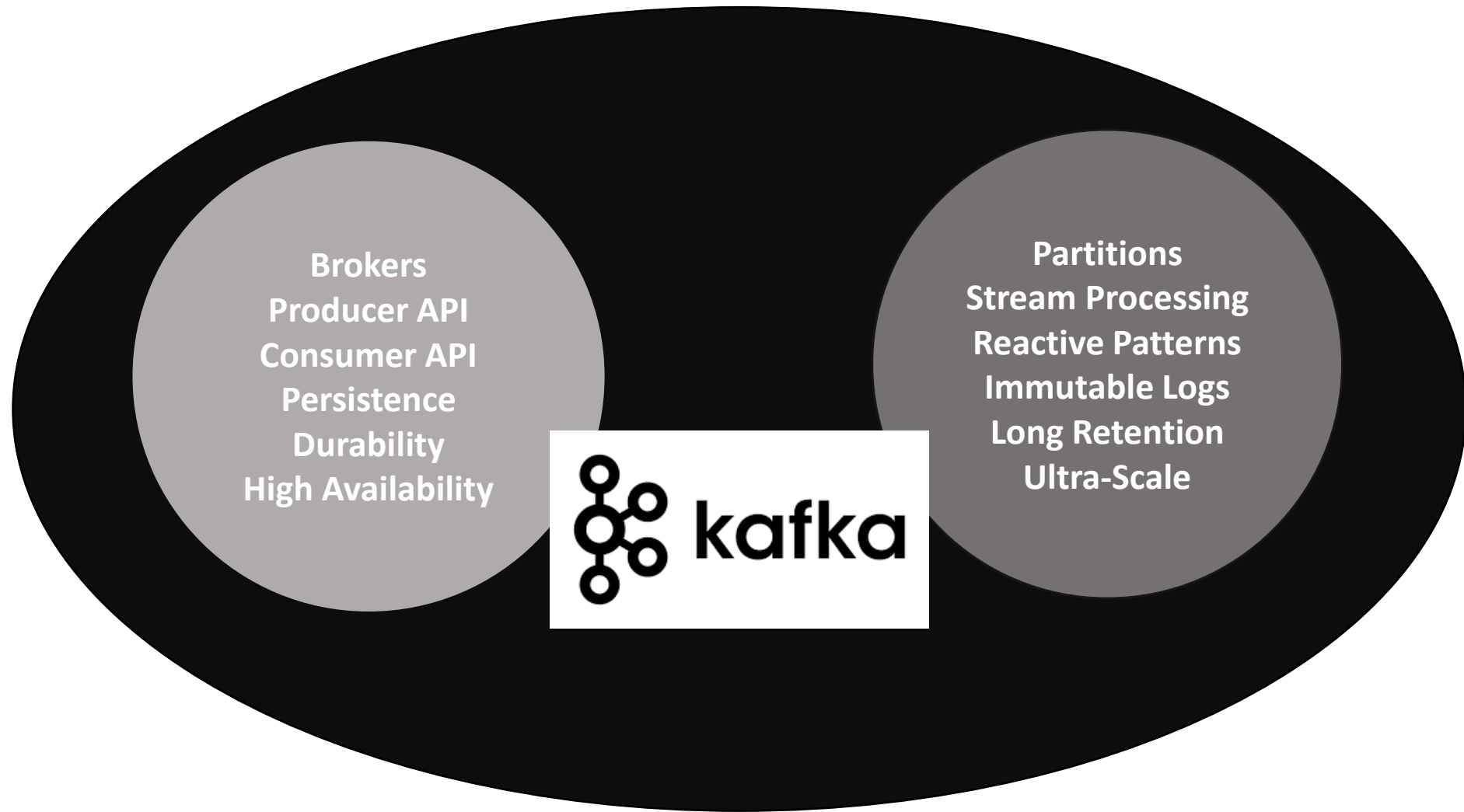
Introducing some really great writers...



# Apache Kafka For High Frequency Messaging

- Kafka is a **high-throughput streaming event system** meant for heavy traffic
- Kafka is great for **Streaming Architecture** and **Event-Driven Architecture**
- Kafka is **free and open source** under the Apache 2.0 license
- Kafka was **conceived at LinkedIn** with LinkedIn scale in mind
- Kafka's architecture and ability to scale **will be more than enough** for most businesses
- Kafka has a **wide library of clients making it easy to integrate** with multiple platforms





**Kafka Extends Traditional Message Oriented Middleware**

# Kafka is Highly Successful

**Astronomically  
Scalable**

**Optimized for High-  
Volume Throughput**

**Full Durability and  
Reliability**



Jul 06, 2023

**"No better solution to support huge data streaming"**

Highly scalable, easy to configure,  
highly available, guaranteed sequential  
processing, no data loss

Trusted by the Fortune 100 and Fortune 500

**NETFLIX**

**ORACLE**

**LinkedIn**

**Square**

**PayPal**

**Spotify**



**Tencent**



## Project Summary

A high-throughput, distributed, publish-subscribe messaging system.

## Tags

[messaging](#)[nosql](#)[pubsub](#)[queuing](#)

## In a Nutshell, Apache Kafka...

... has had 11,892 commits made by 1,250 contributors representing 880,510 lines of code

... is mostly written in Java with a low number of source code comments

... has a well established, mature codebase maintained by a very large development team with increasing Y-O-Y commits

... took an estimated 246 years of effort (COCOMO model) starting with its first commit in August, 2011 ending with its most recent commit 2 days ago

## Quick Reference

Organization: [Apache Software Foundation](#)

Project Links: [Homepage](#) [Community](#) [Documentation](#) [Download](#) [Issue Trackers](#) [Mailing Lists](#)

Code Locations: <https://github.com/apache/kafka>

Similar Projects:  [RDFbus](#)  [redpanda-data](#)

[redpanda-data](#)

 [pylib](#)

 [Mongoosel](#) [M p...](#)

Managers: [jaykrels](#)

OpenHub Metrics: <https://openhub.net/p/apache-kafka>

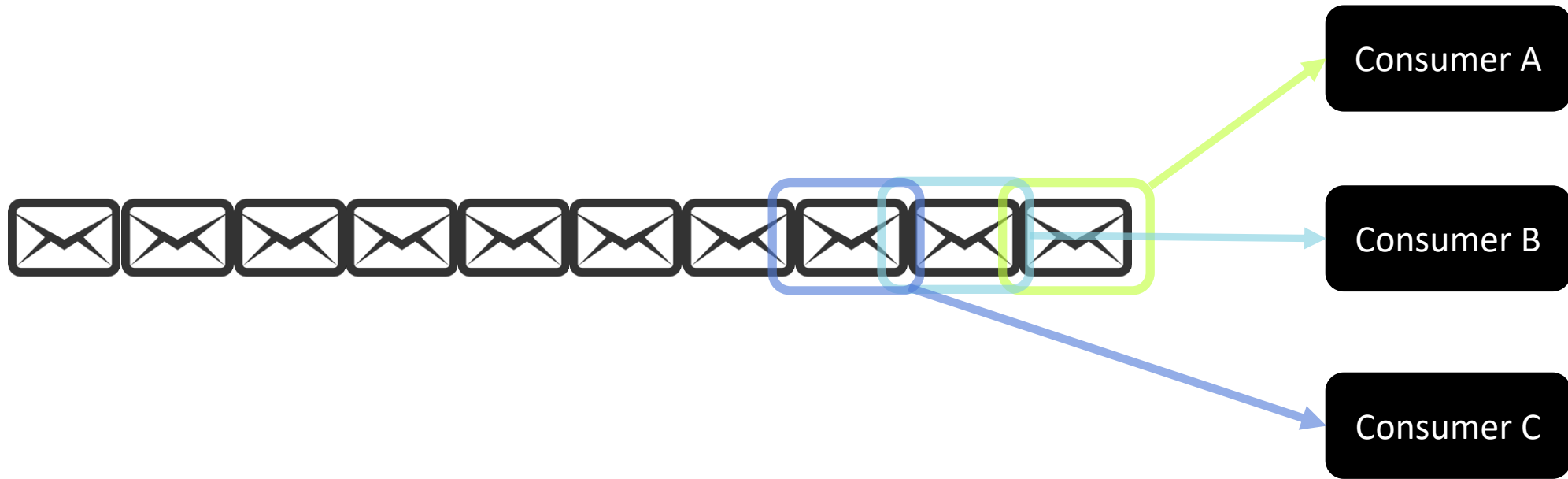
# Kafka Is a Streaming Engine

- Traditional messaging systems **focus on reliable delivery of a single message**
- Kafka can present **large amounts of data** in **historically-tagged chunks**
- Kafka allows queueing, but its **primary use case is a Topic pattern**
- A Topic will have **multiple interested consumers** receiving streams of data
- Kafka scales these Topics even further by **breaking them into Partitions**



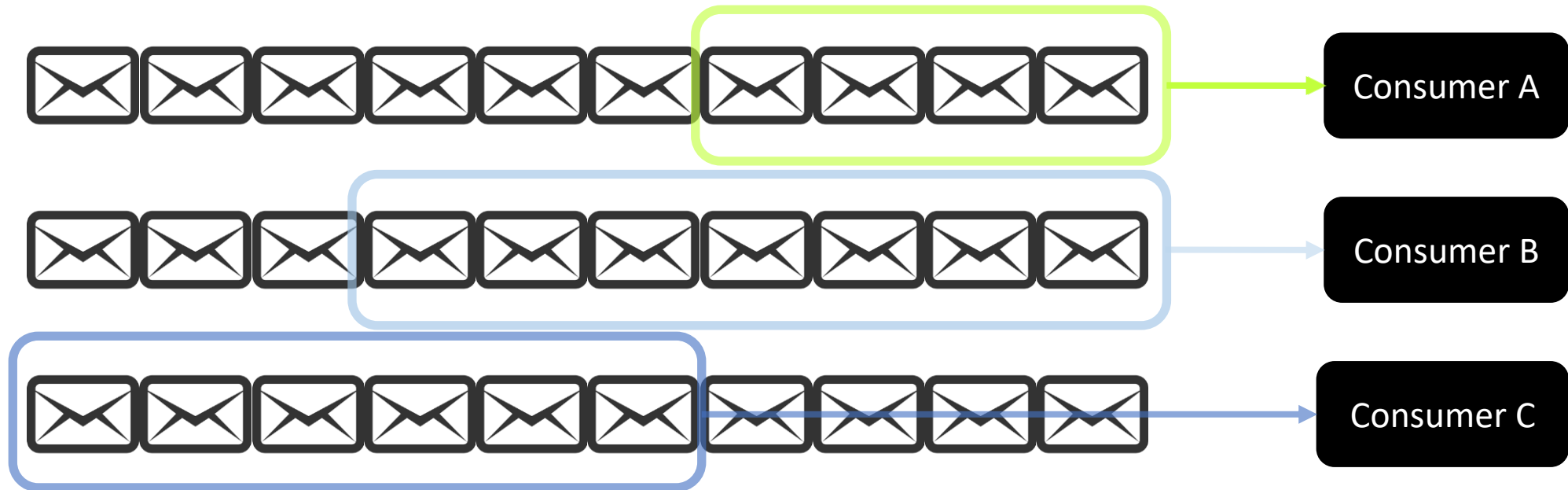
# Kafka Is a Streaming Engine

In traditional message queueing (not topics), a **single message is processed at a time**, even if that payload contains a lot of data



# Kafka Is a Streaming Engine

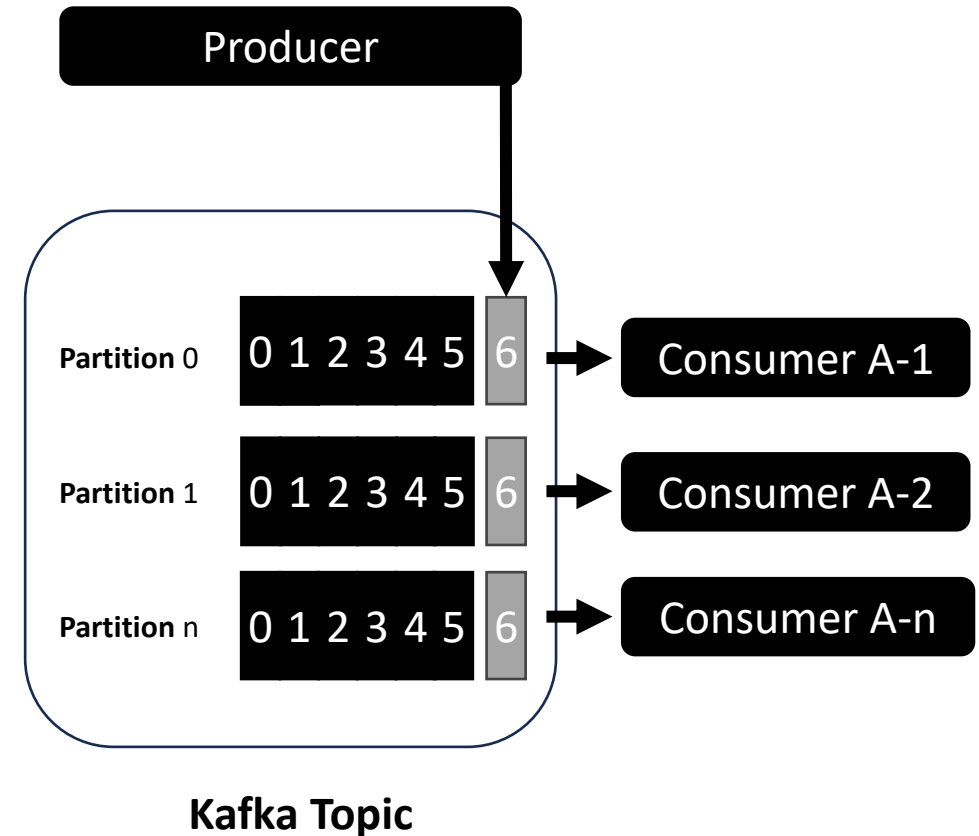
**With Kafka, a series or stream of messages can be processed at a time. Immutable logs allow long-term data storage, so consumers can time slice data they want to receive**



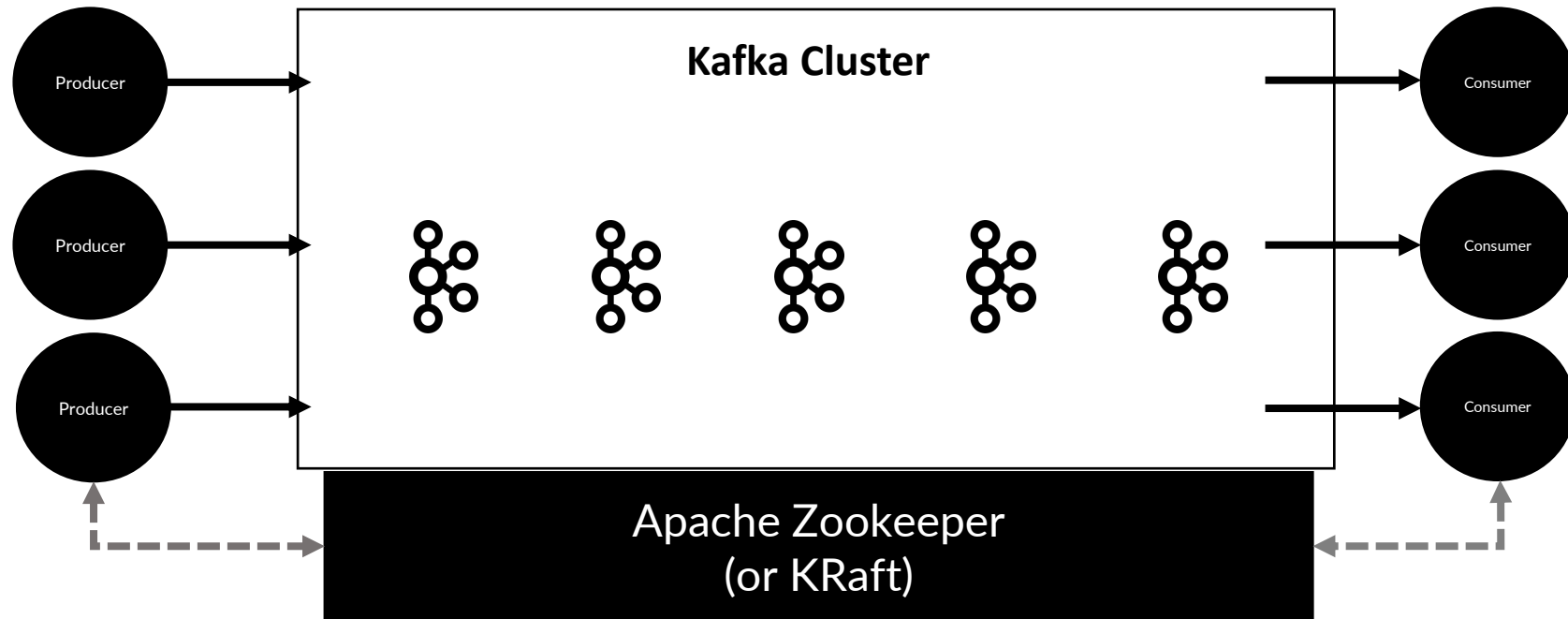


# Partitions For Replication and Scale

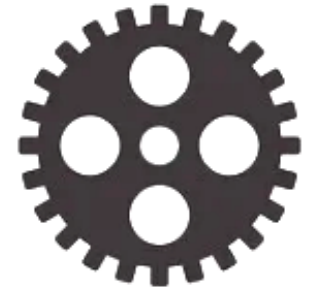
- Kafka **stores streamed data in Partitions**
- One consumer may attach to one partition
- **Superfluous consumers** will be ignored
- Partitions are **replicated across brokers** according to a replication factor
- This allows for **excellent redundancy** in retaining the data and **good load balancing and horizontal scale** across storage



# Kafka High Level Topology



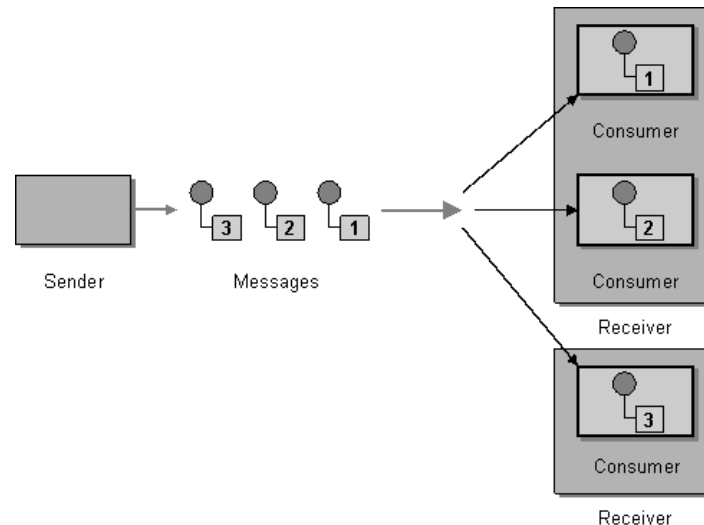
- Launch Kafka Cluster and View in Kafdrop
- Create Topic and Attach Consumers and Producers
- Demonstrate Topic Durability



**Demo – Producers and Consumers**

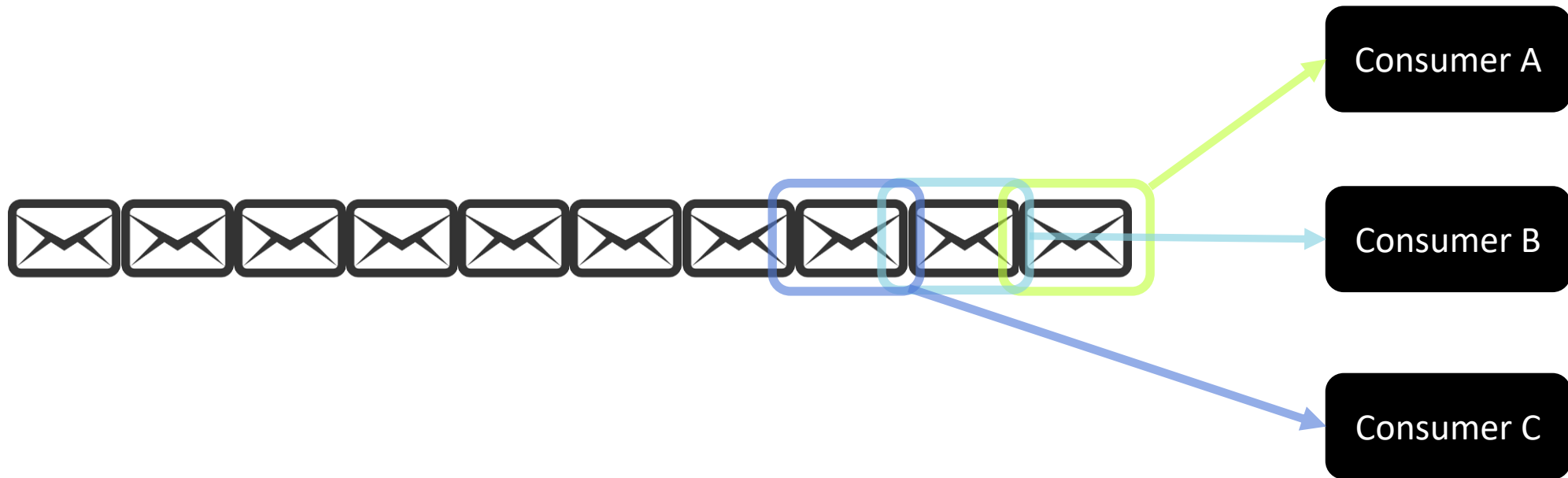
# Traditional Problems with Scale

- In most messaging systems, data is **produced** at a rate faster than it is consumed
- This creates “**backpressure**” on Producers
- Many solutions have been presented, from **Producer Flow Control** to horizontally-scaled **Networks of Brokers**
- Consumers **need to load balance** processing of messages



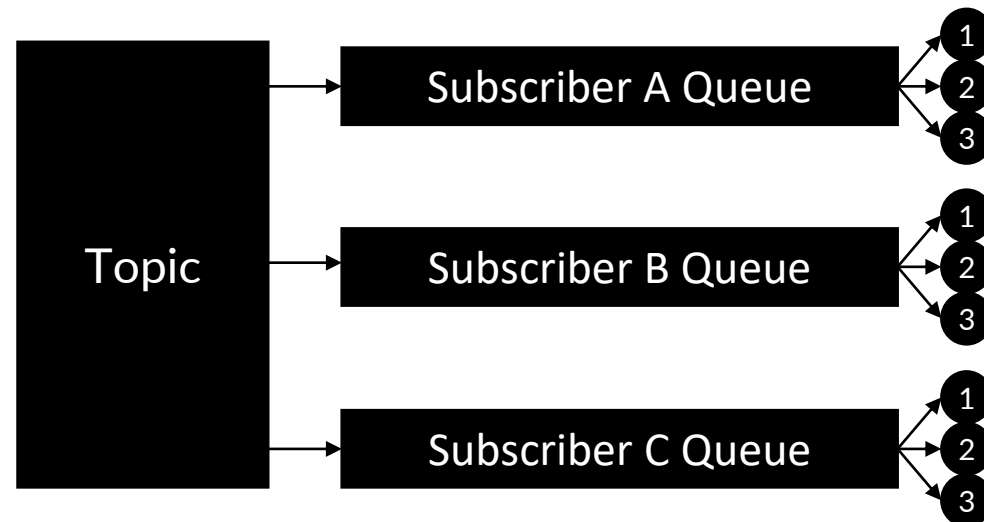
# Pattern: Queue Consumer Load Balancing

Recall that in queuing Scenarios, Consumers load balance messages from Queues



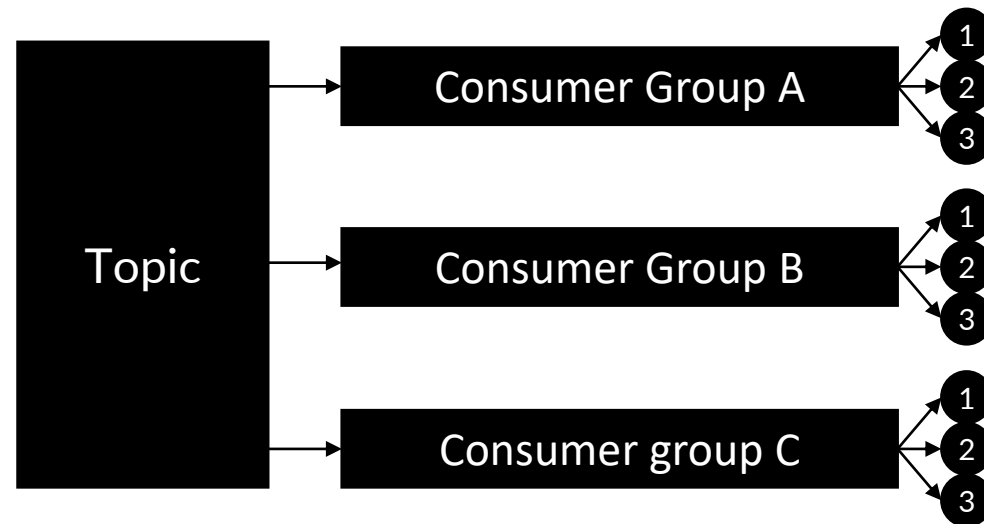
# Pattern: Virtual Topic Subscriptions

- Subscribers in Pub/Sub patterns are **traditionally single-threaded**
- **ActiveMQ introduced Virtual Topics** to load balance across subscription threads
- Hard to guarantee **delivery order** and **idempotency** across a network
- Replication strategies are **limited and brittle**



# Kafka Wins: Consumer Groups

- Kafka Consumer Groups present a pattern that **fulfills all needs** while reliably replicating data across partitions and brokers
  - **Load-balancing** across single pub/sub threads and queuing patterns
  - **Guaranteed order** per-consumer
  - **Idempotency** when “ack” configured properly



# Consumer Group Protocol

Kafka handles mapping of consumers to partitions automatically when multiple Consumers specify the same Topic and Group ID

```
@KafkaListener(topics = "DemoTopic", groupId = "Consumer-Group-A")
public void listenA1(String message) { procMesg("A::1", message); }

@KafkaListener(topics = "DemoTopic", groupId = "Consumer-Group-A")
public void listenA2(String message) { procMesg("A::2", message); }
```

```
2023-11-13T01:00:18.402-05:00 INFO 30408 --- [ntainer#1-0-C-1]
o.s.k.l.KafkaMessageListenerContainer : Consumer-Group-A: partitions assigned:
[DemoTopic-1]
2023-11-13T01:00:18.402-05:00 INFO 30408 --- [ntainer#0-0-C-1]
o.s.k.l.KafkaMessageListenerContainer : Consumer-Group-A: partitions assigned:
[DemoTopic-0]
```



# Consumer Group Protocol (V2)

- V1 uses a “rebalance” process to assign partitions to consumers
- Consumers are reassigned to partitions when topics change or consumers connect or disconnect
- This can pause processing during rebalance
- V2 improves this with “incremental rebalancing,” allowing consumer or partition changes without a full rebalance



- Add more consumers to the existing consumer group
- Note partition initialization and rebalance
- Increase message production rate
- Observe as messages are spread across partitions
- Observe as Consumers load balance distribution



**Demo – Load Balancing Queue**

- Subscribe Additional Consumer Groups to Topic
- Subscribe Multiple Consumers per Consumer Group to Topic
- Significantly increase Production Rate
- Increase consumers as needed



**Demo – Load Balancing Topic**

# Wrapping Up

- Message oriented middleware patterns can help organizations reliably federate data across their systems
- Traditional MOM systems can run into problems at scale
- The Apache Kafka message broker contains bleeding-edge features that address many of these problems
- Kafka also allows for Stream Processing and Event Driven Architecture
- Consumer Groups in Kafka are used to achieve even greater scale
- Data can be load balanced across queues and topic subscriptions
- Consumer Groups are easy to configure and highly automated

Questions?

