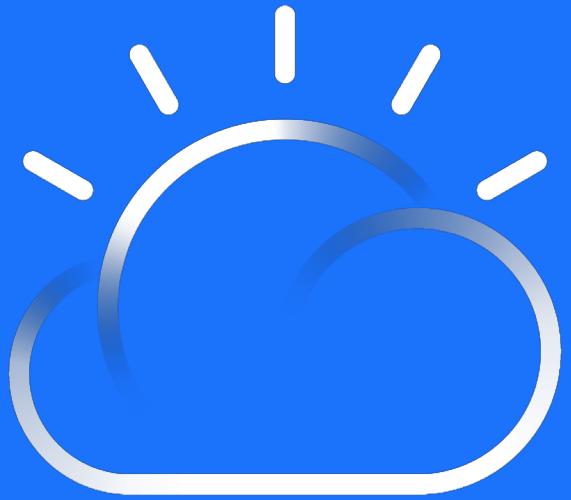
Integration Technical Conference 2019

M01
Making sense of queues and event streams

Andrew Schofield
Senior Technical Staff Member
IBM Messaging



IBM **Cloud**

IBM

Please note

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice and at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Two styles of messaging

MESSAGE QUEUING

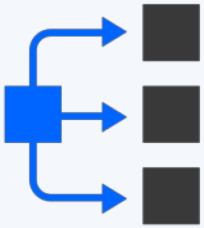
Transient data persistence

Conversational messaging

Targeted reliable delivery

EVENT STREAMING

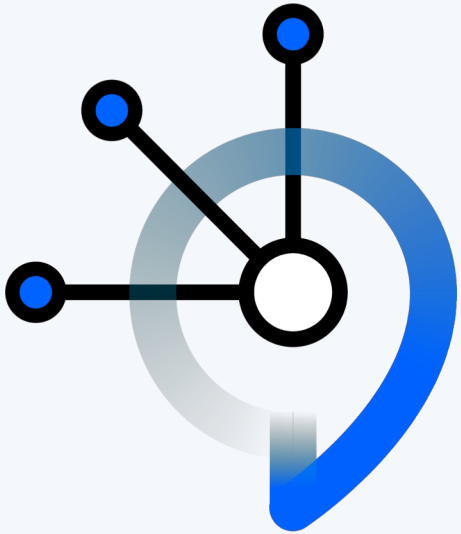
Stream history



Scalable consumption

Immutable data

IBM MQ



IBM Event Streams

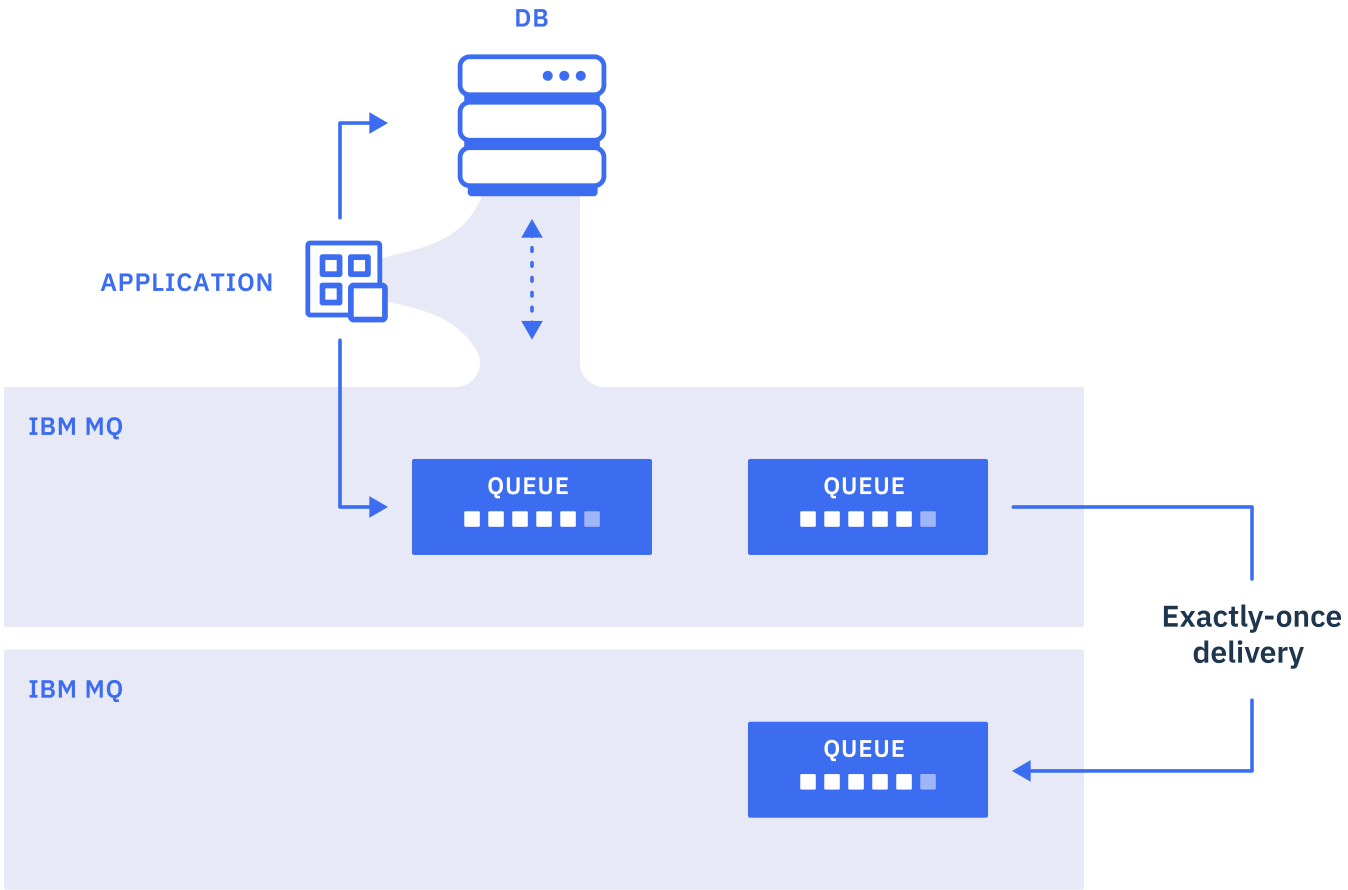
Apache Kafka® is the de-facto standard for event-driven applications, delivering:

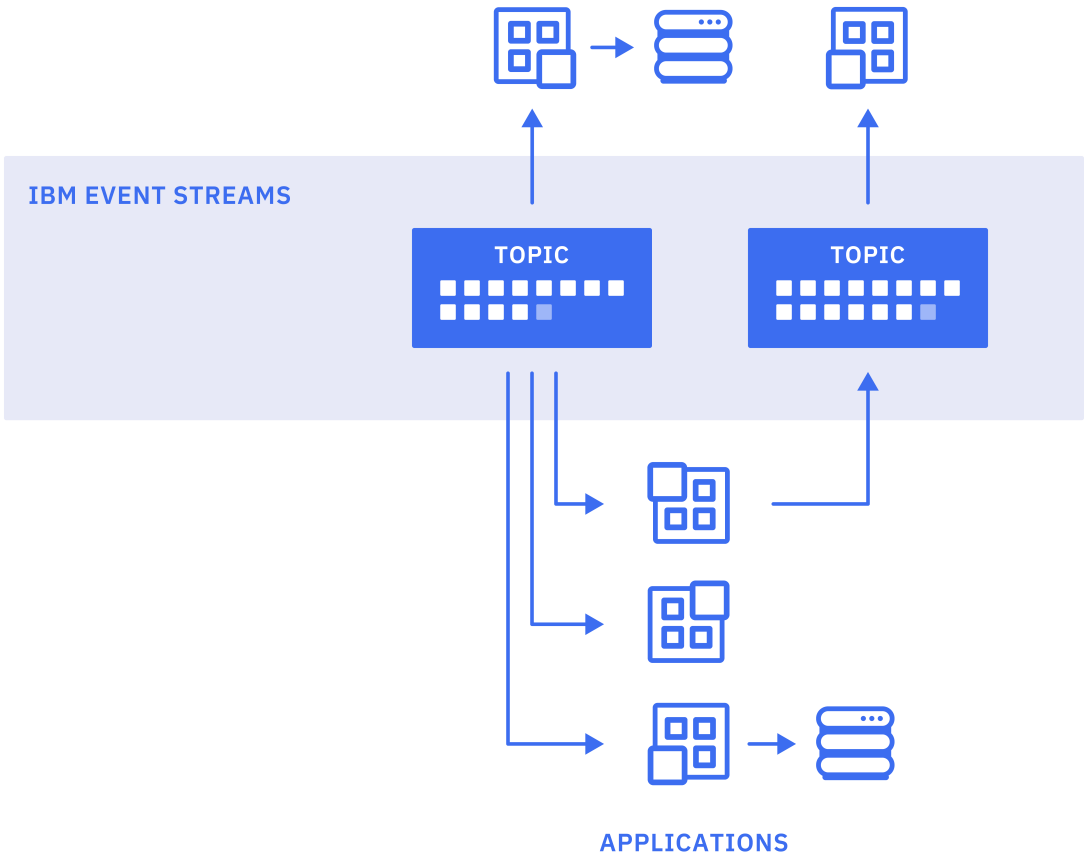
- Event Distribution – to get real-time data to where it can be acted upon
- Event Store – for retaining event stream history to power big data analytics
- Real-time Processing – for insights that can be acted upon in the moment

IBM Event Streams is fully supported Apache Kafka® with value-add capabilities

APACHE kafka®
A distributed streaming platform

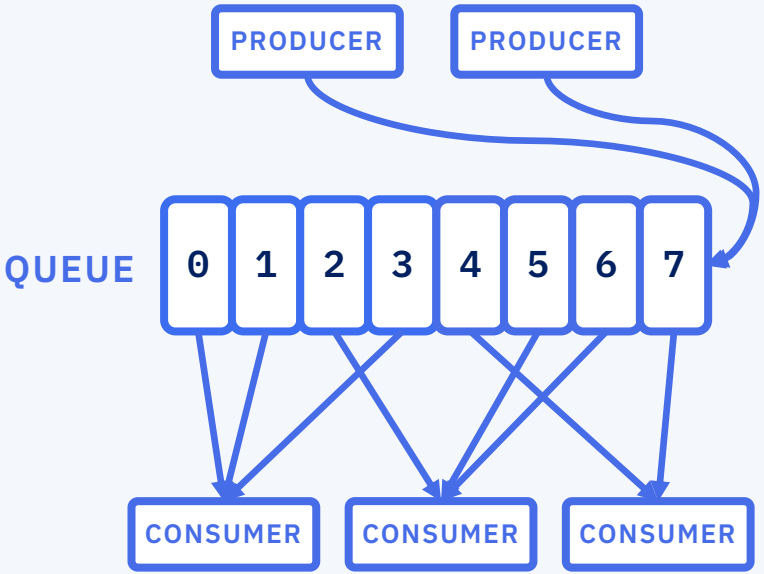
Application styles are different

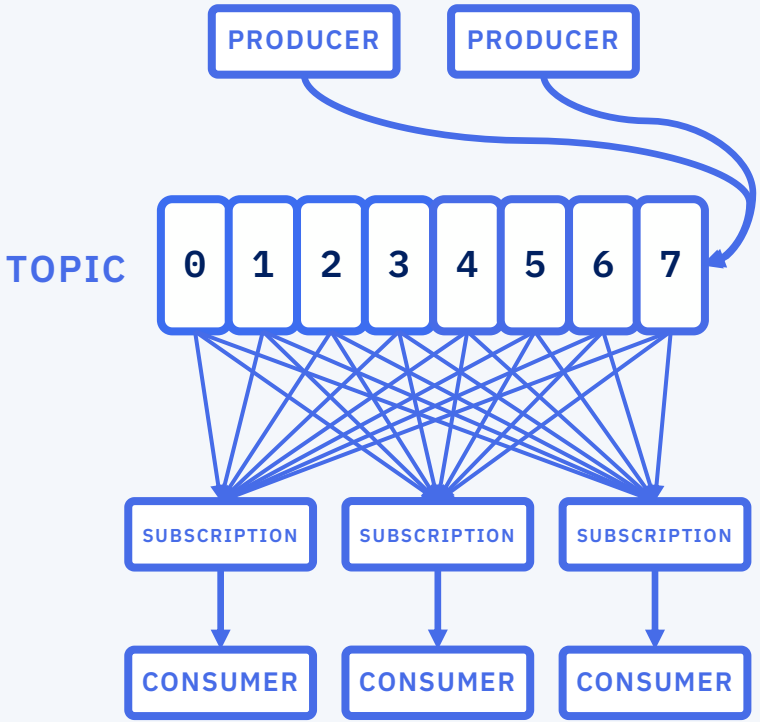




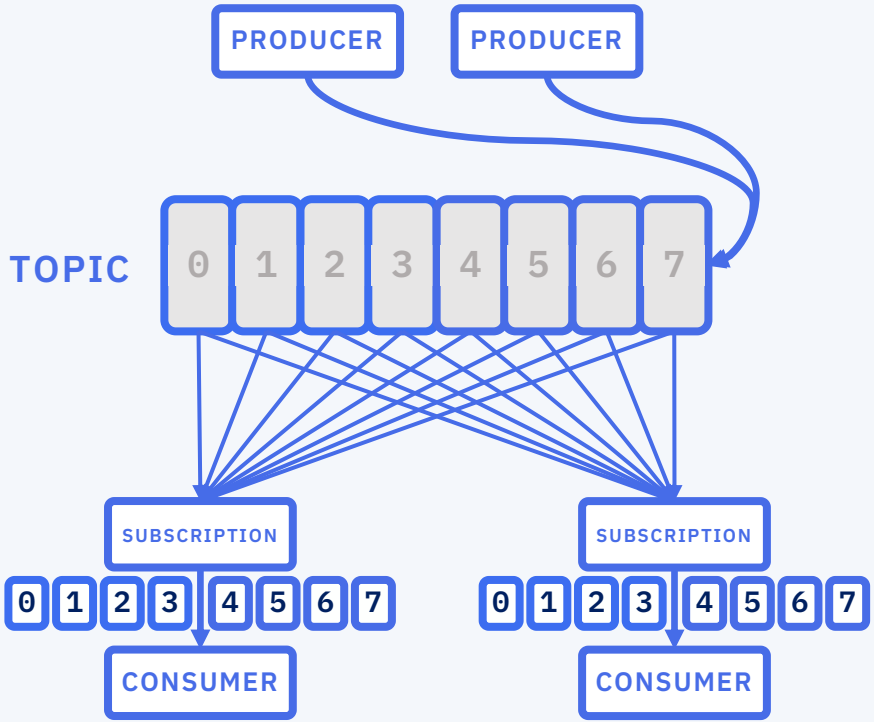
Queues and topics

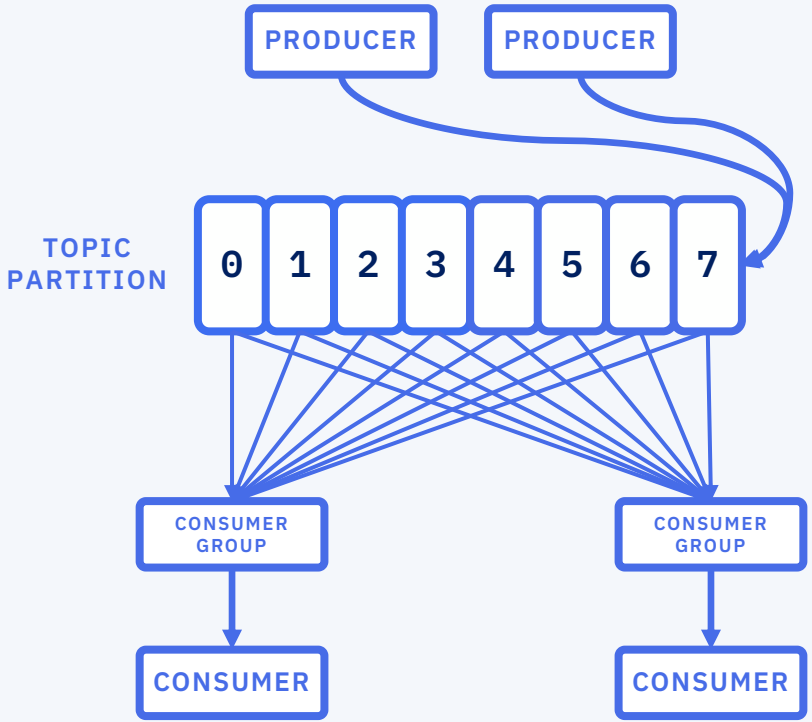
Comparing queues and topics



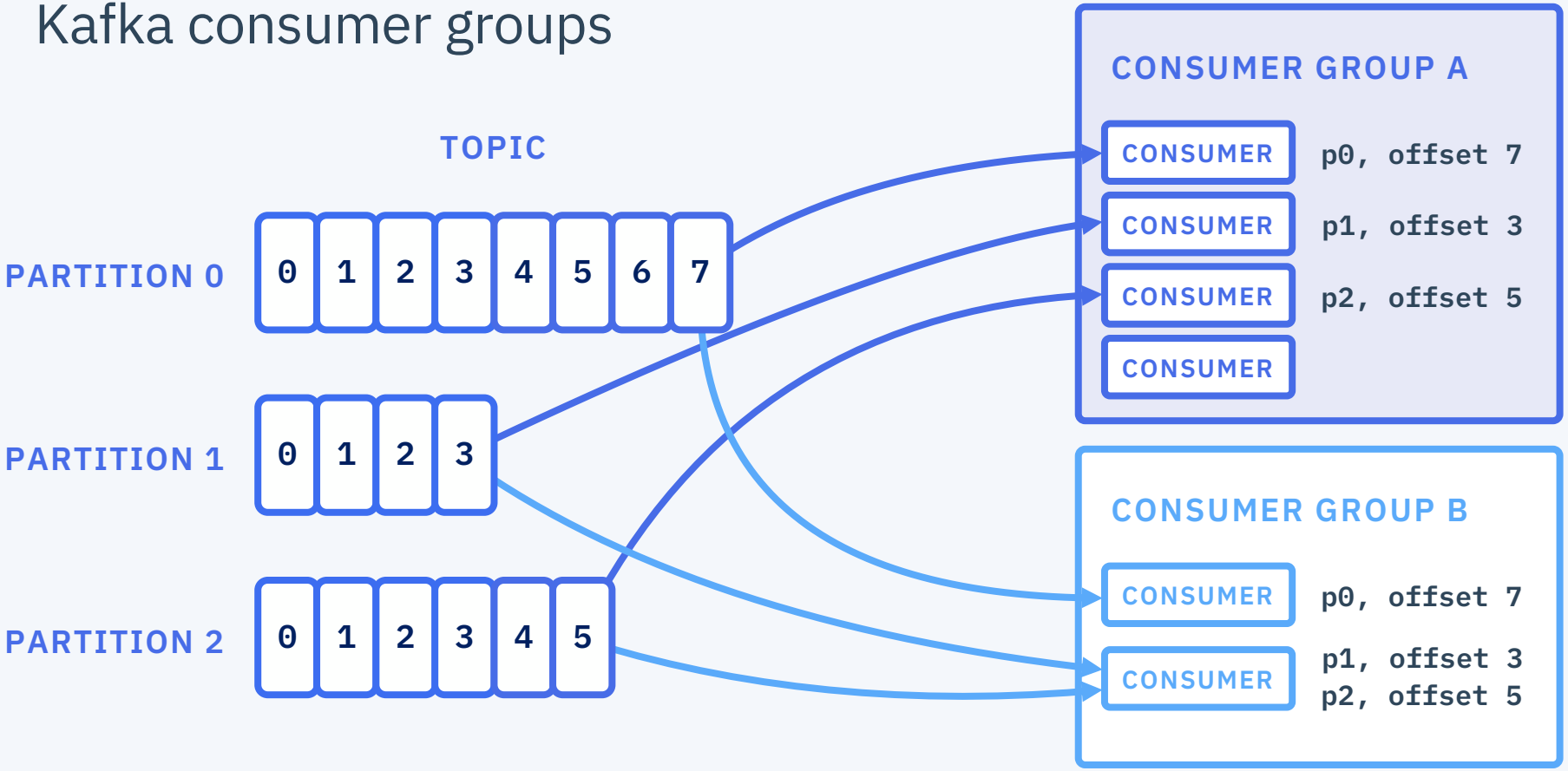


Comparing MQ and Kafka topics



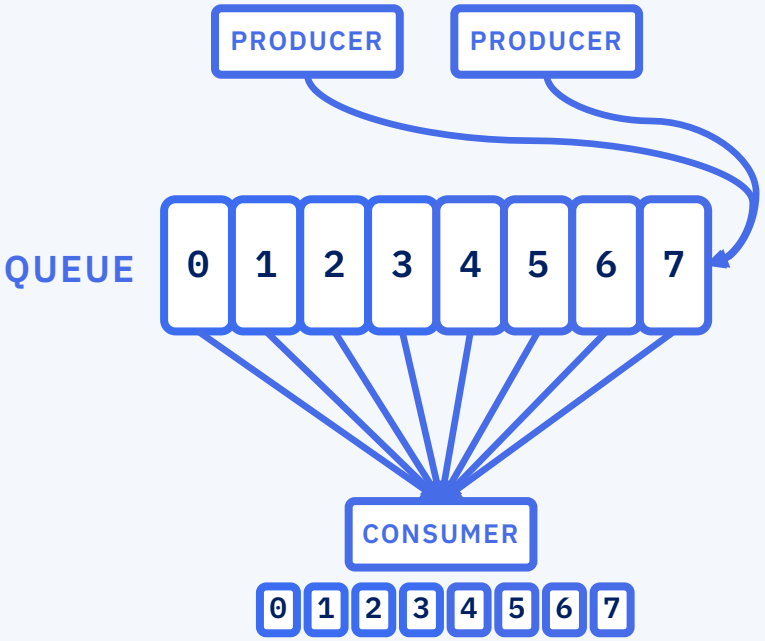


Kafka consumer groups



Message consumption

Queues – destructive consumption, exclusive

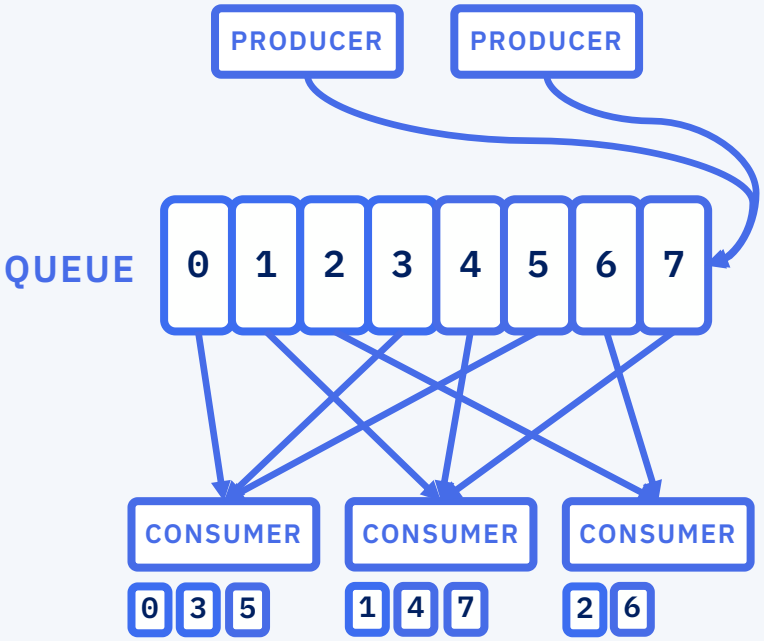


Single consumer

In-order delivery

Relatively tight coupling

Queues – destructive consumption, shared

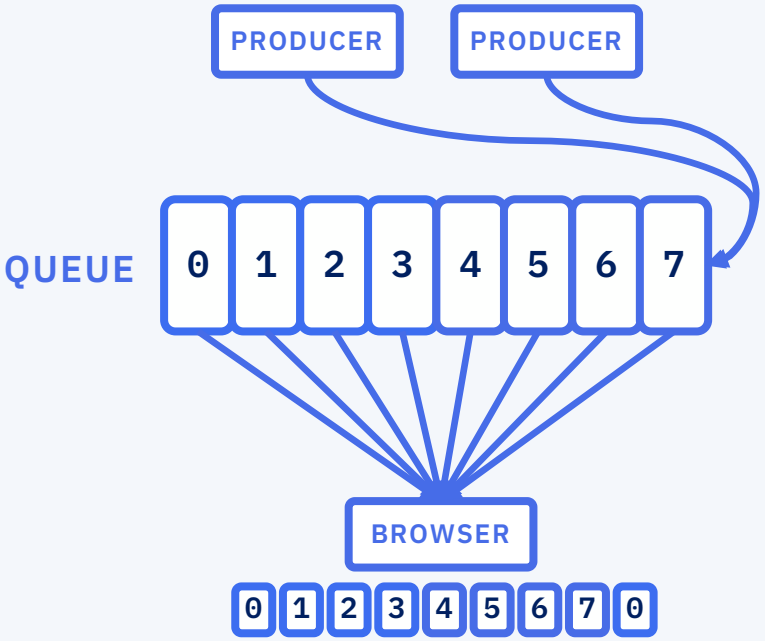


Multiple consumers

Out-of-order delivery

Easy workload distribution

Queues – non-destructive consumption



In-order delivery

Can reset position

Message consumption in Kafka

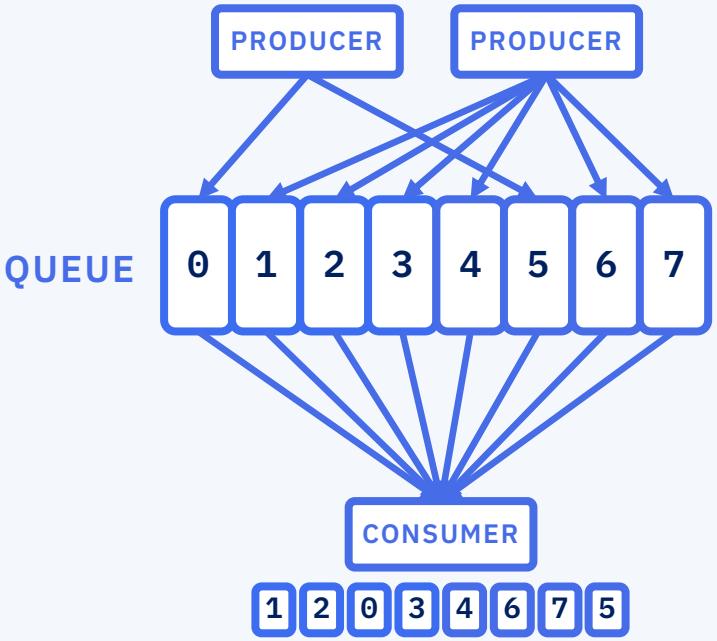
A queue is not a log

Most similar to browsing in MQ

- Non-destructive
- Consumer controls the position and can move backwards and forwards
- Kafka remembers consumer offset persistently on an internal topic

Transactions

Transactional message visibility in MQ

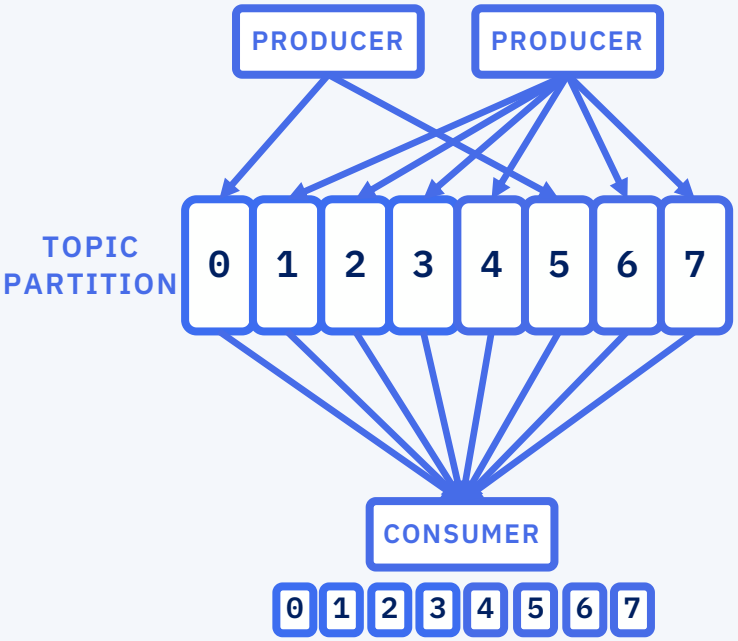


Ordered when produced

Uncommitted are invisible

Uncommitted are skipped

Transactional message visibility in Kafka



Ordered when produced

Consumer gets same order

Two isolation levels
- Read uncommitted
- Read committed

Kafka transactions

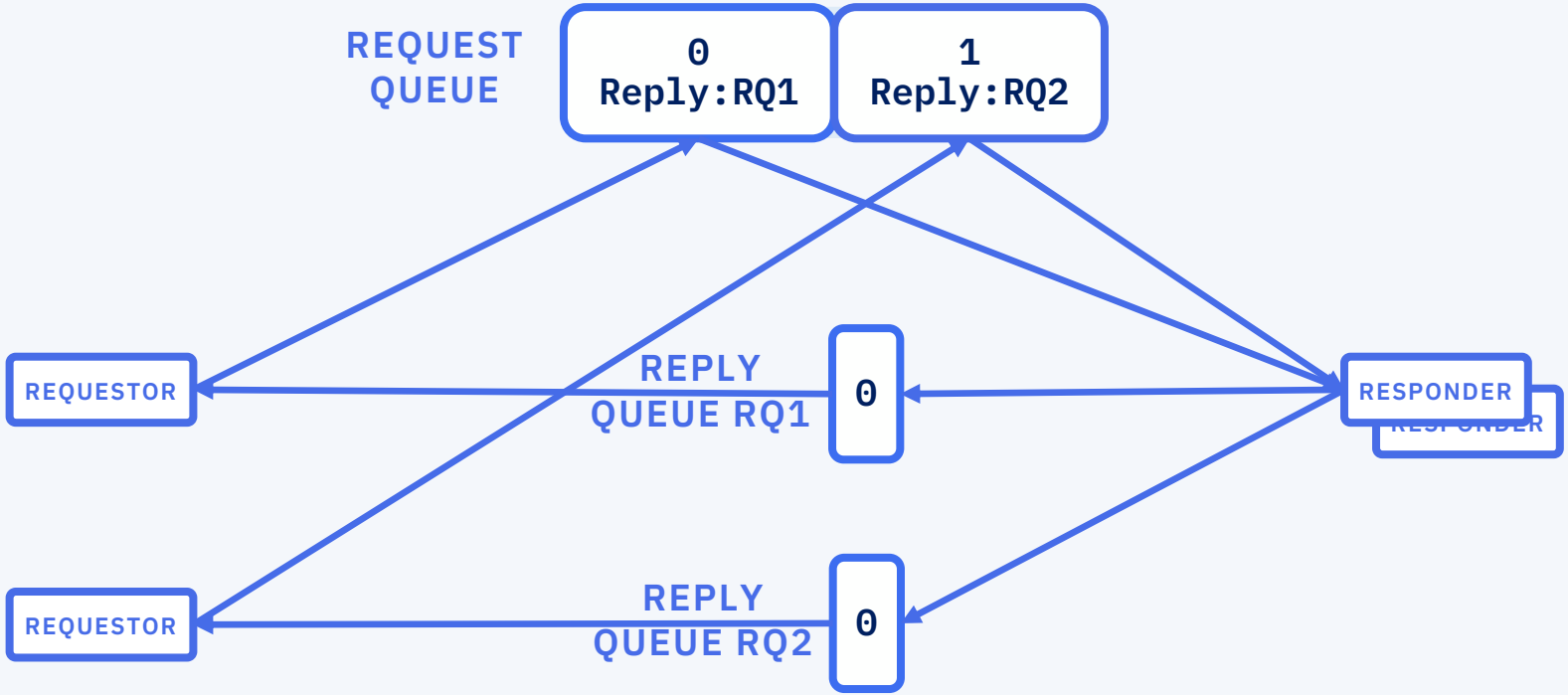
Aimed primarily at exactly-once stream processing applications

Cannot be coordinated with an external system

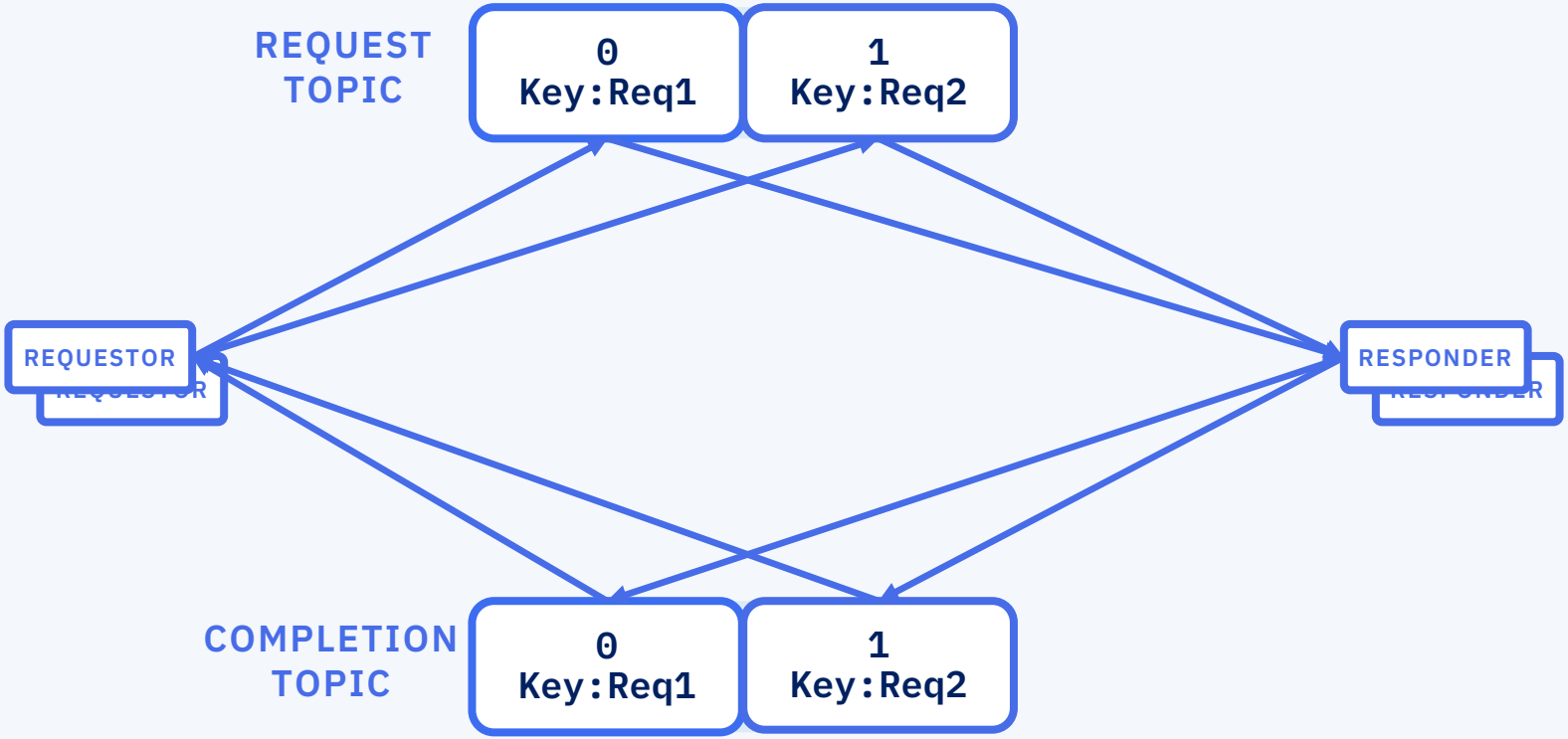
Strong message ordering, but committed messages can be delayed

Request/reply

What about request/reply?

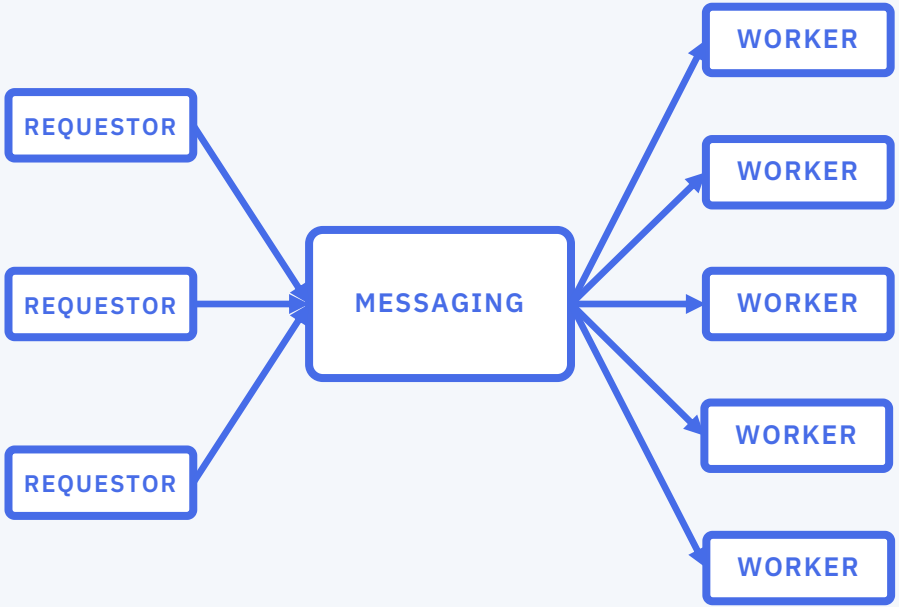


With topics, it’s more like request/completion



Use cases

Job scheduling



A set of requestors send messages to request a job to be performed

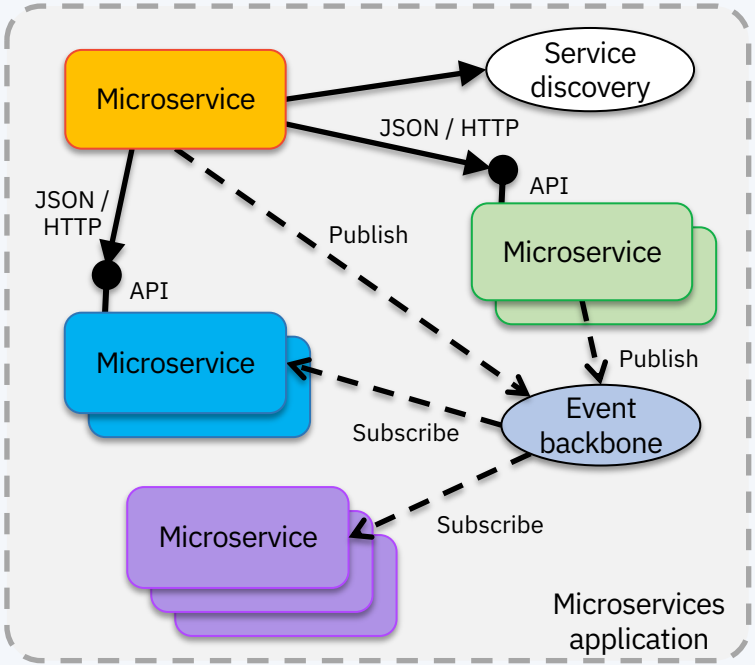
A set of workers pick off the requests and perform the work

The work needs to be shared but not in order

Best handled using a **queue**

- Easy workload distribution

Event-driven microservices



Microservices communicate primarily using events, with APIs where required

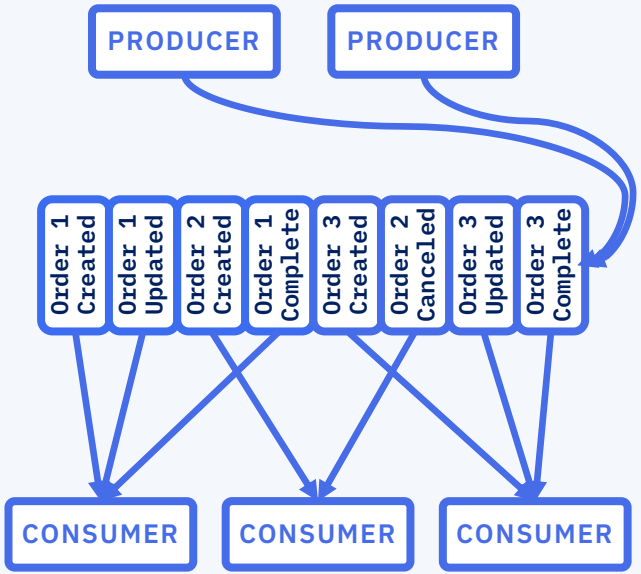
Microservices can produce and consume events

Data is eventually consistent

Best handled using **publish/subscribe**

- More loosely coupled than a queue

Event sourcing



Every state change to a business object is captured as an event

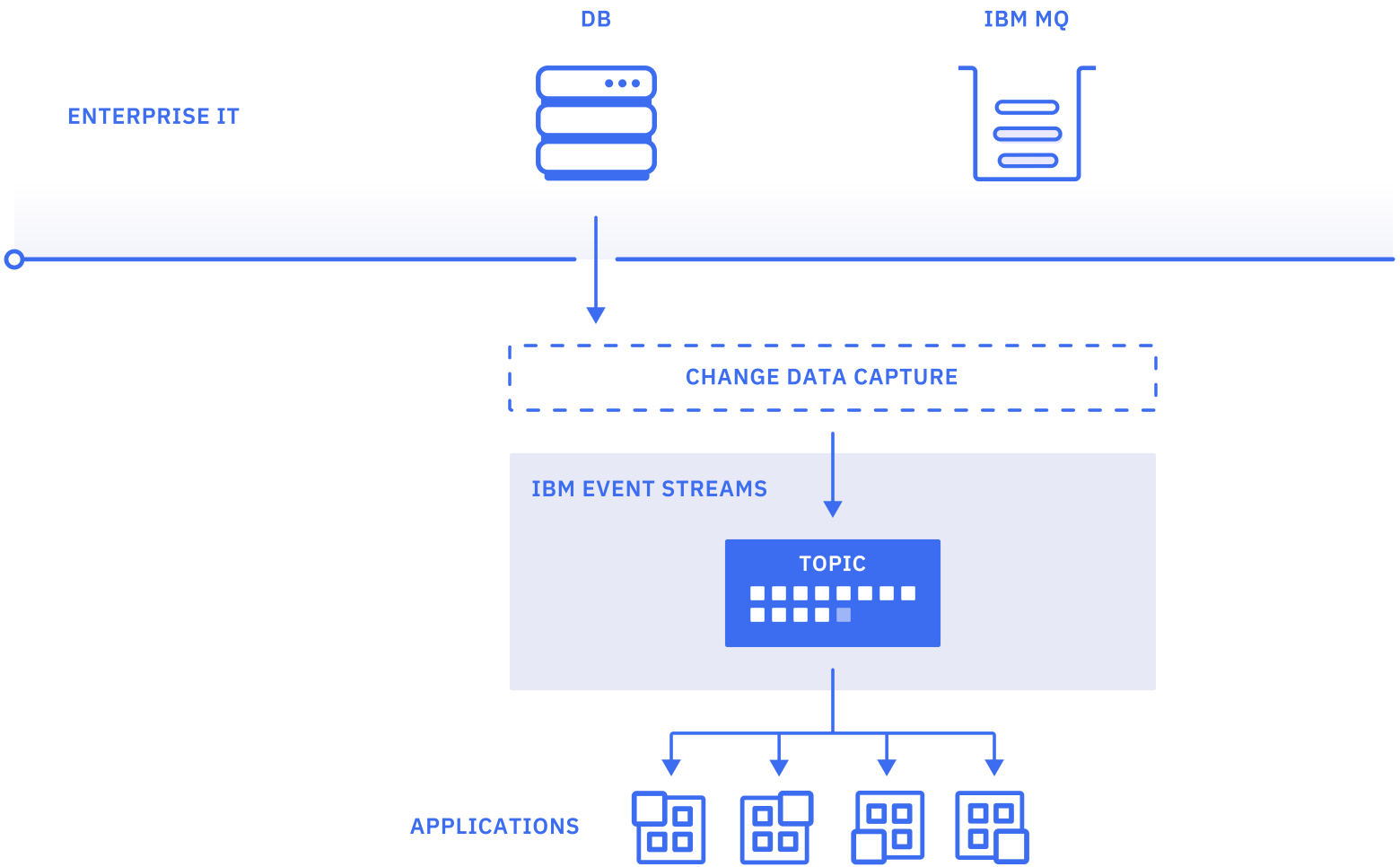
Replaying the events rebuilds the business object

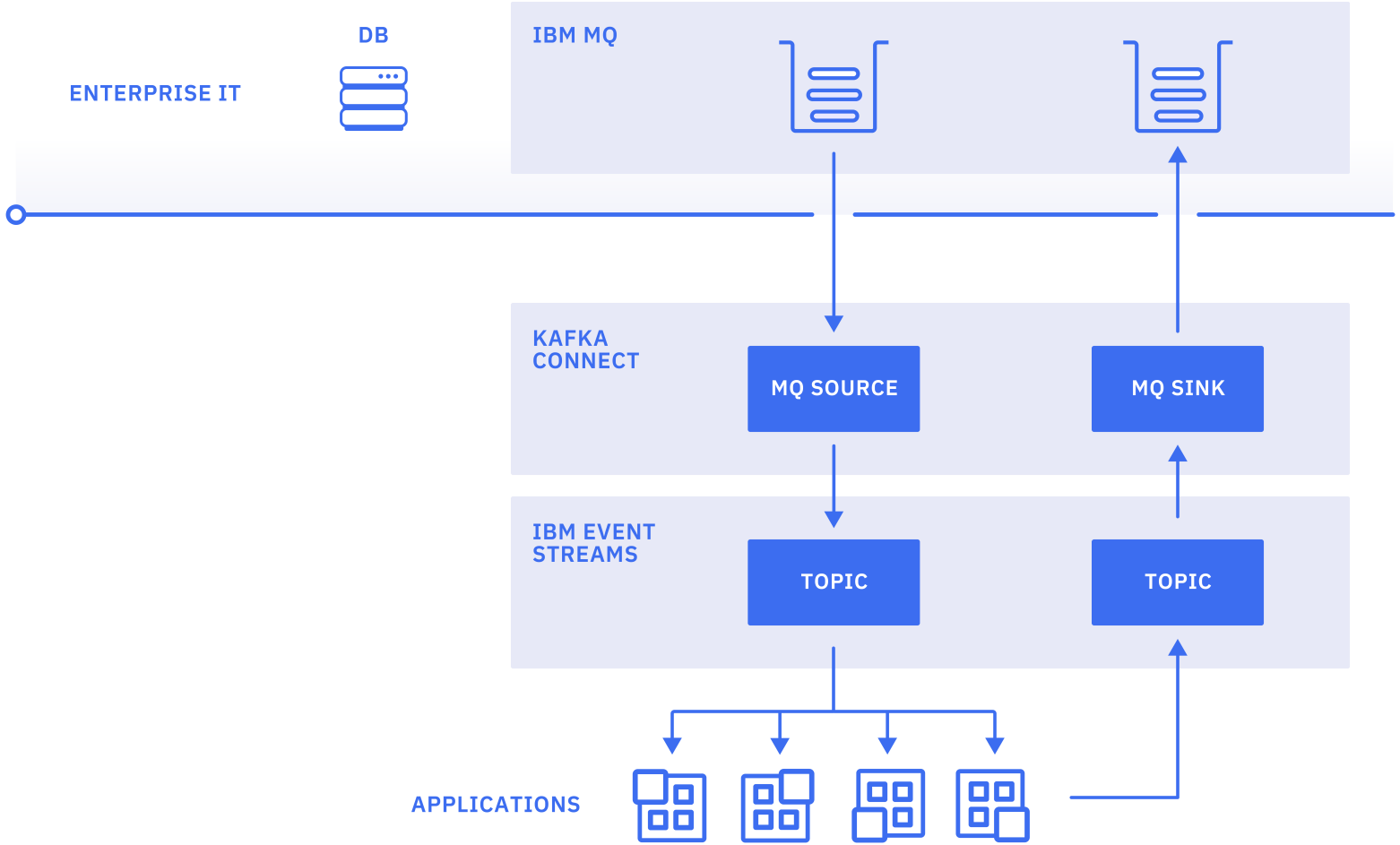
Subscribing to the events takes an evolving copy of the business object

Best handled using **event streaming**

- Stream history is vital

Unlocking events from existing systems





It’s easy to connect IBM MQ to Apache Kafka

IBM has created a pair of connectors, available as source code or as part of IBM Event Streams

Source connector

From MQ queue to Kafka topic

https://github.com/ibm-messaging/kafka-connect-mq-source

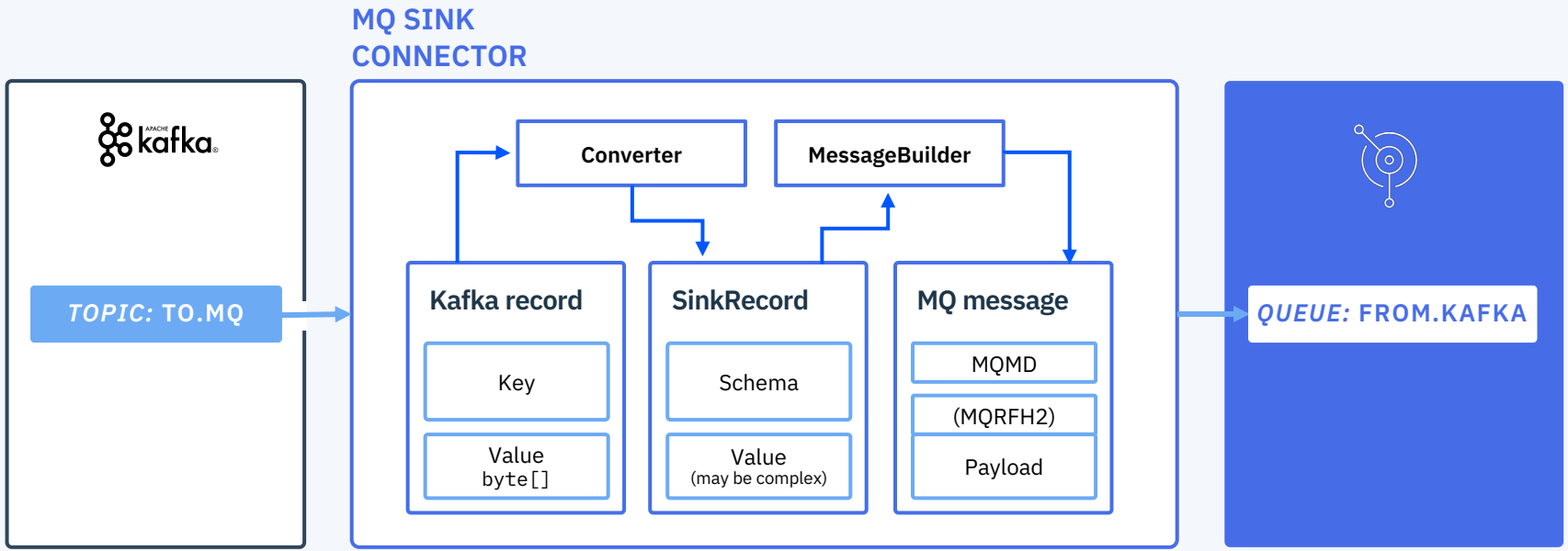
Sink connector

From Kafka topic to MQ queue

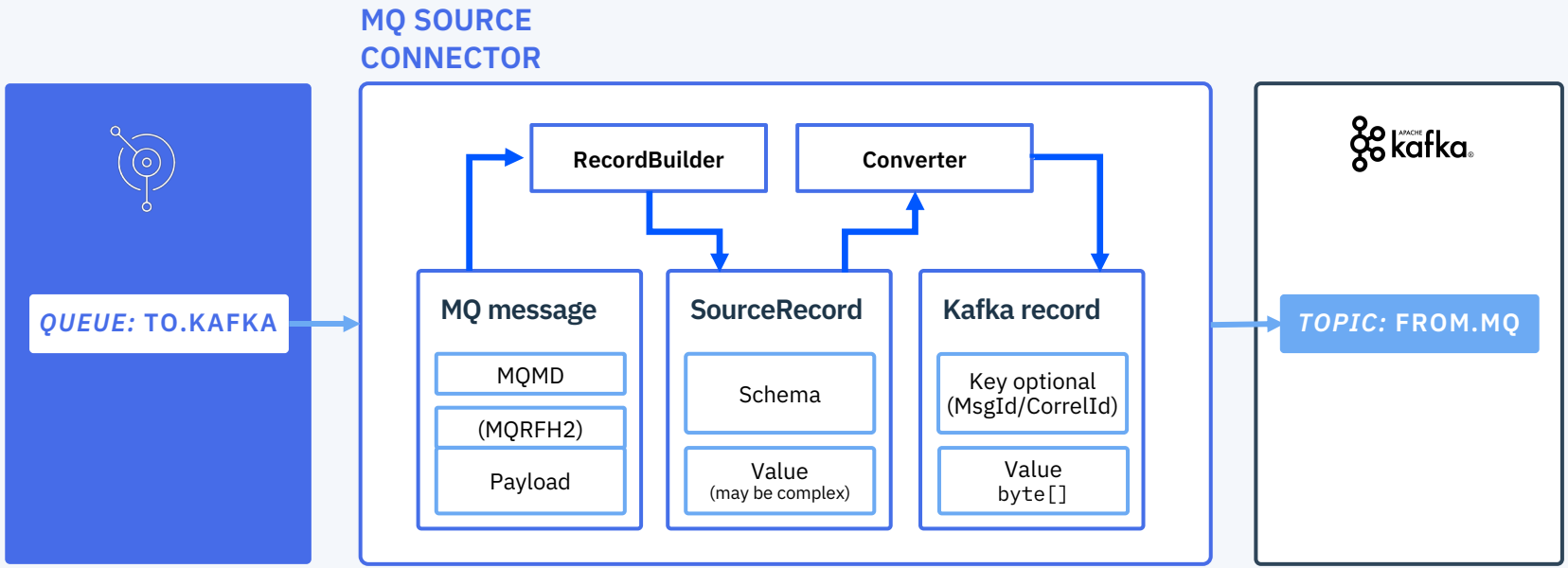
https://github.com/ibm-messaging/kafka-connect-mq-sink

Fully supported by IBM for customers with support entitlement for IBM Event Streams

MQ sink connector



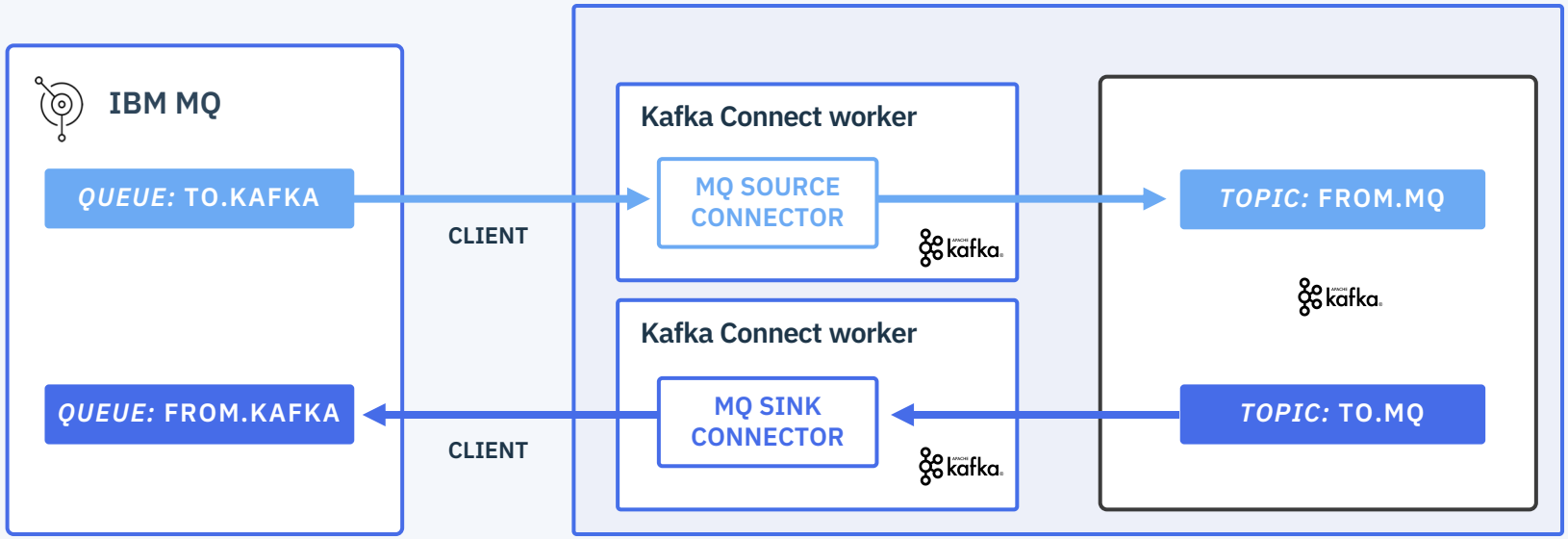
MQ source connector



Connecting IBM MQ to Apache Kafka

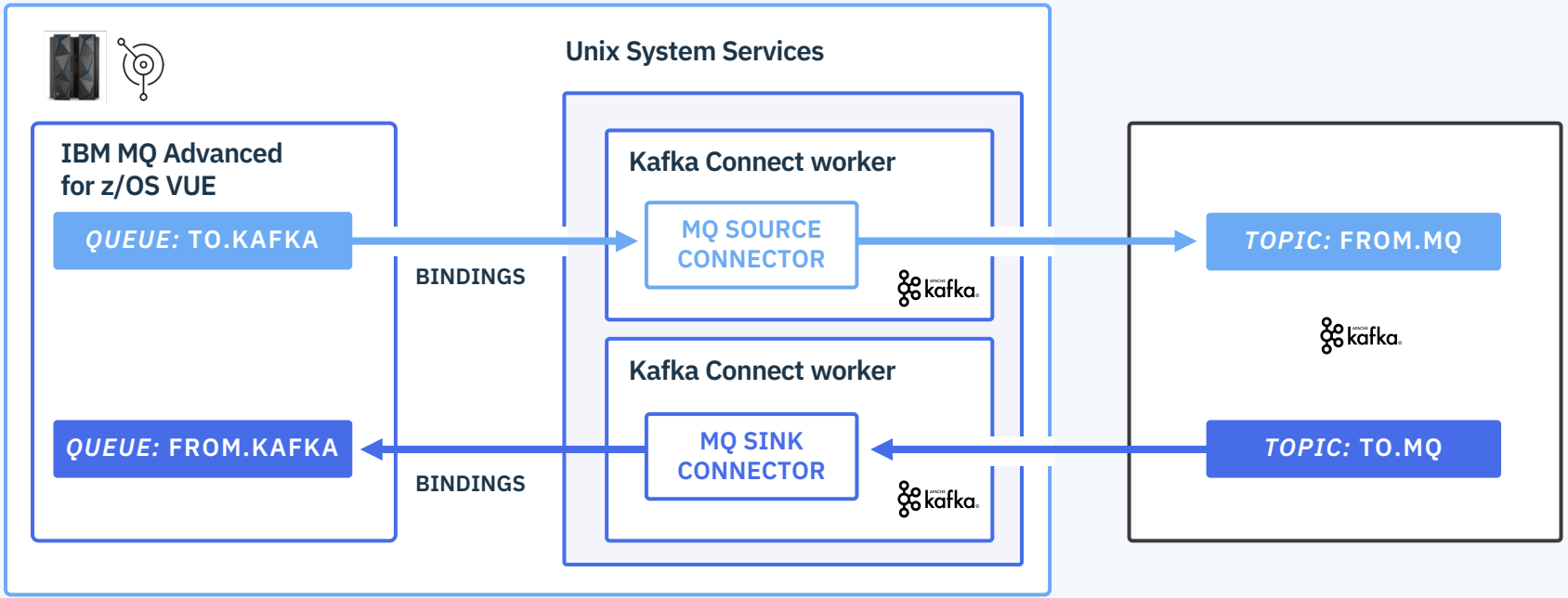
The connectors are deployed into a Kafka Connect runtime

This runs between IBM MQ and Apache Kafka



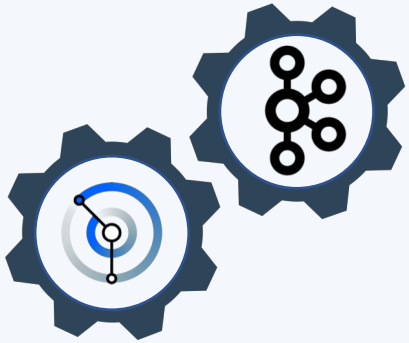
Running Kafka Connect on a mainframe

IBM MQ Advanced for z/OS VUE provides support for the Kafka Connect workers to be deployed onto z/OS Unix System Services using bindings connections to MQ



Summary

How do I make sense of this?



Queues are good for:

- Easy workload distribution with multiple consumers
- Out of order message acknowledgement

Event streams are good for:

- Stream history because consumption is always non-destructive
- Ordering with workload distribution, but the consumers must keep up

You might well want to use them together

- Queues for commands, event streaming for events

Thank You