UNIFYING MESSAGING, QUEUING, STREAMING & COMPUTE WITH APACHE PULSAR

KARTHIK RAMASAMY

CO-FOUNDER AND CEO

streamlio

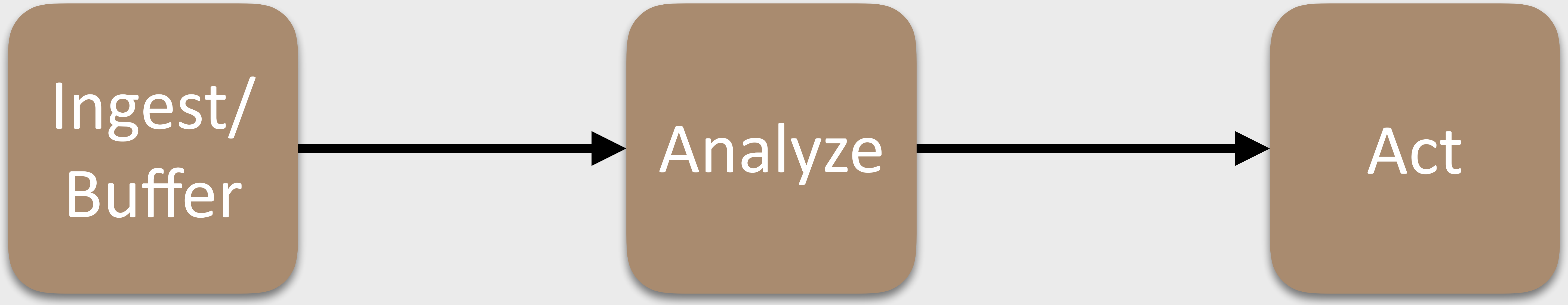
streamlio

Connected World

Ubiquity of Real-Time Data Streams & Events

EVENT/STREAM DATA PROCESSING

- Events are analyzed and processed as they arrive
- Decisions are timely, contextual and based on fresh data
- Decision latency is eliminated
- Data in motion



streamlio

EVENT/STREAM PROCESSING PATTERNS

MONITORING

MICROSERVICES

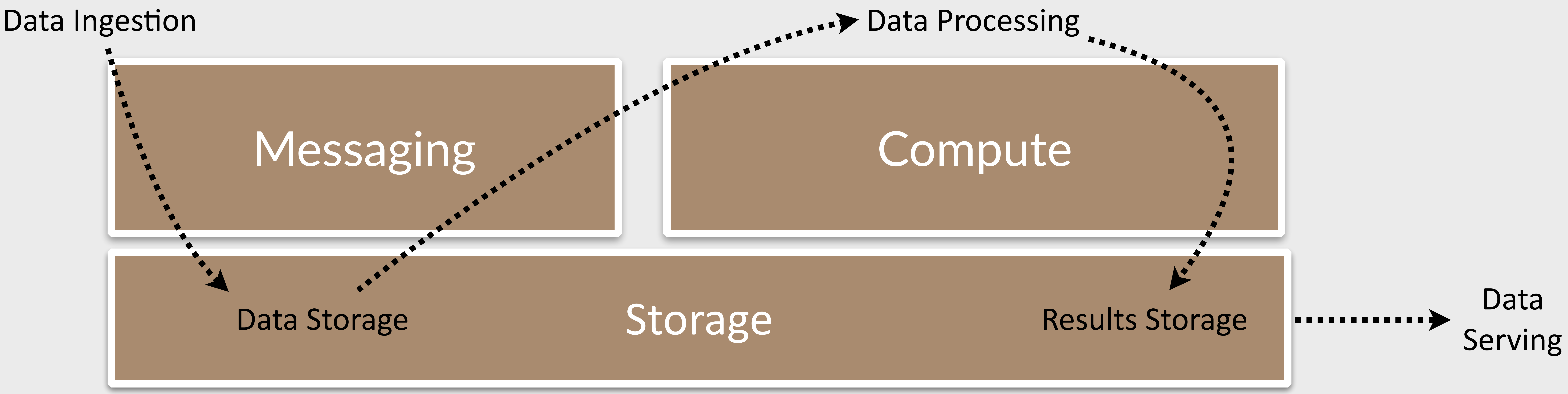
WORKFLOWS

MODEL INFERENCE

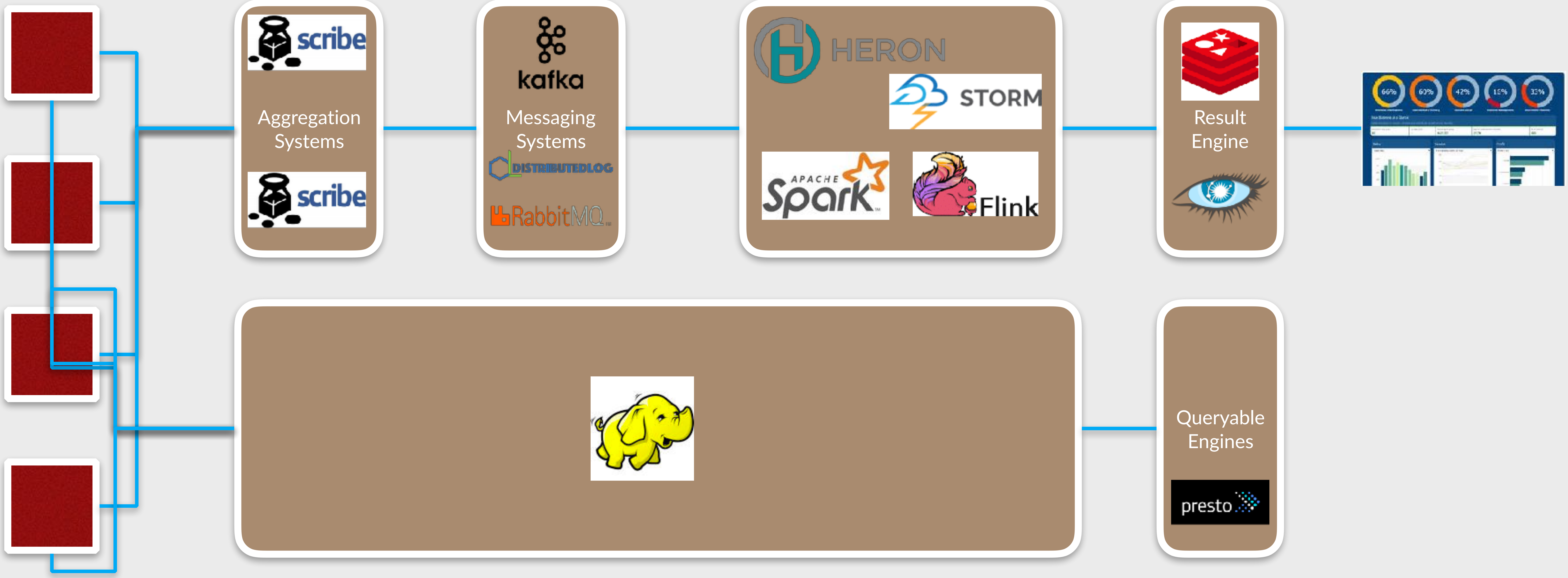
ANALYTICS

streamlio

STREAM PROCESSING PATTERN



ELEMENTS OF EVENT/STREAM PROCESSING



APACHE PULSAR

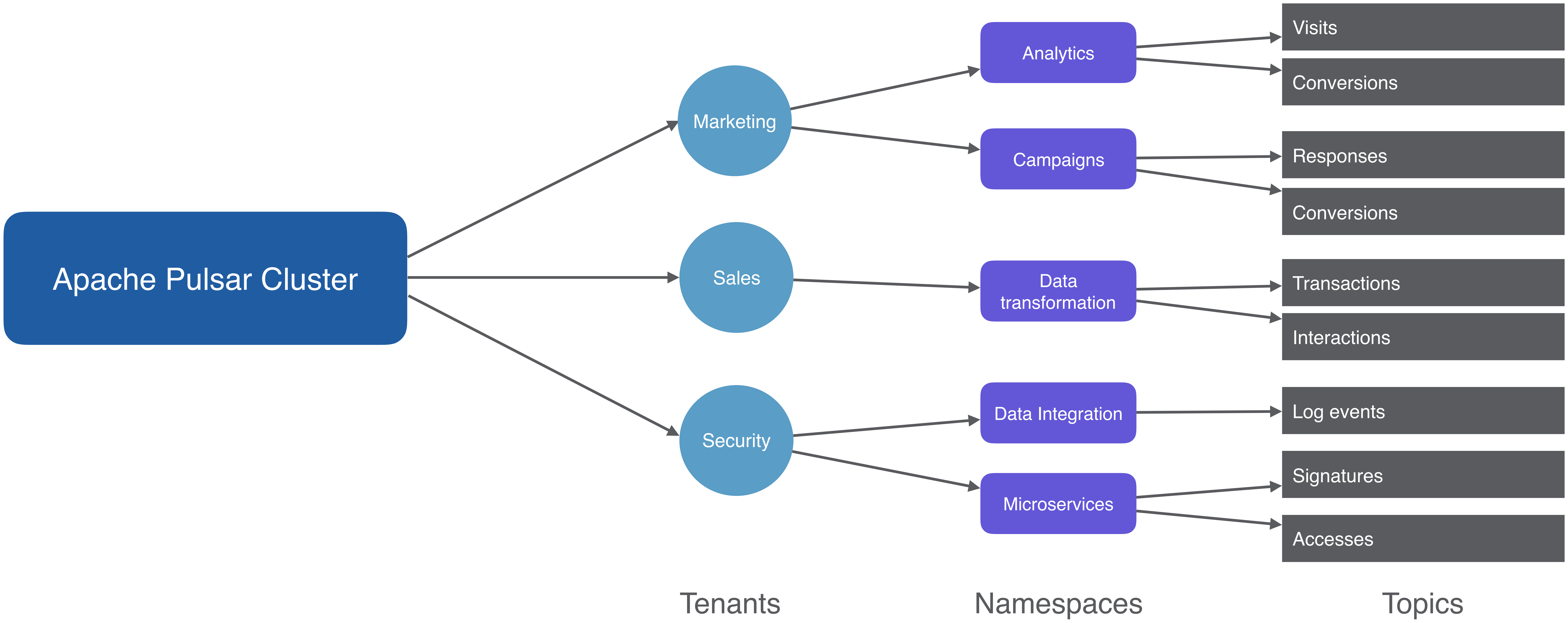
Flexible Messaging + Streaming System

backed by a durable log storage

Key Concepts

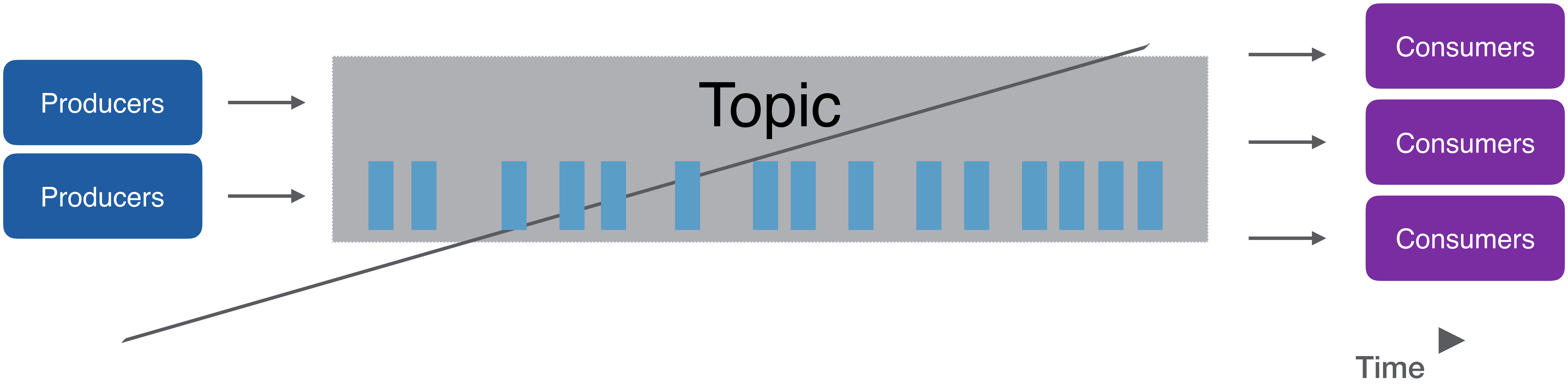
streamlio

Core concepts: Tenants, namespaces, topics



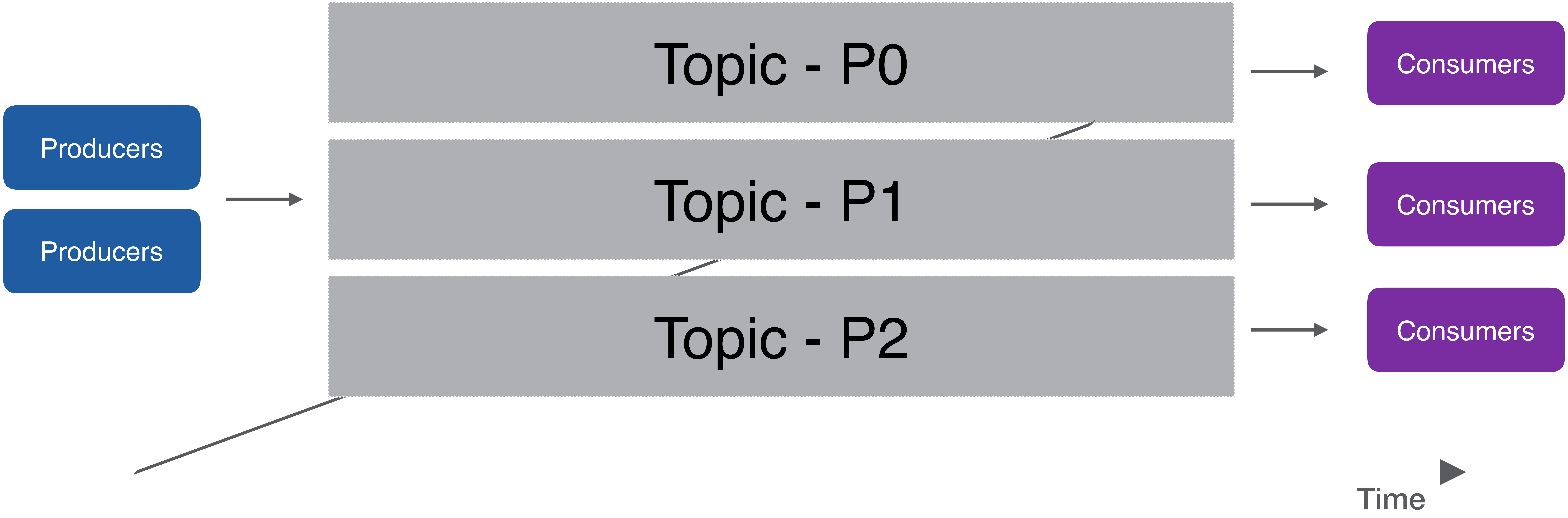
streamlio

Topics

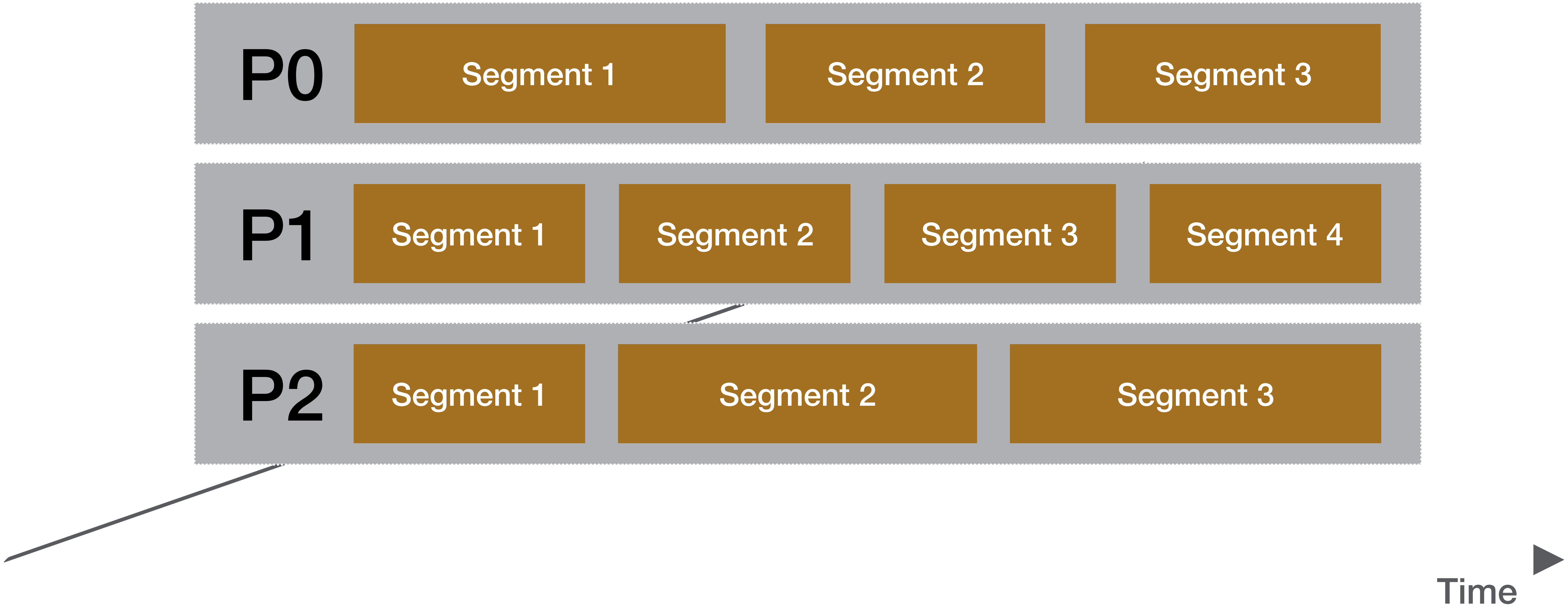


Producers
Producers
Topic
Consumers
Consumers
Consumers
Time

Topic partitions



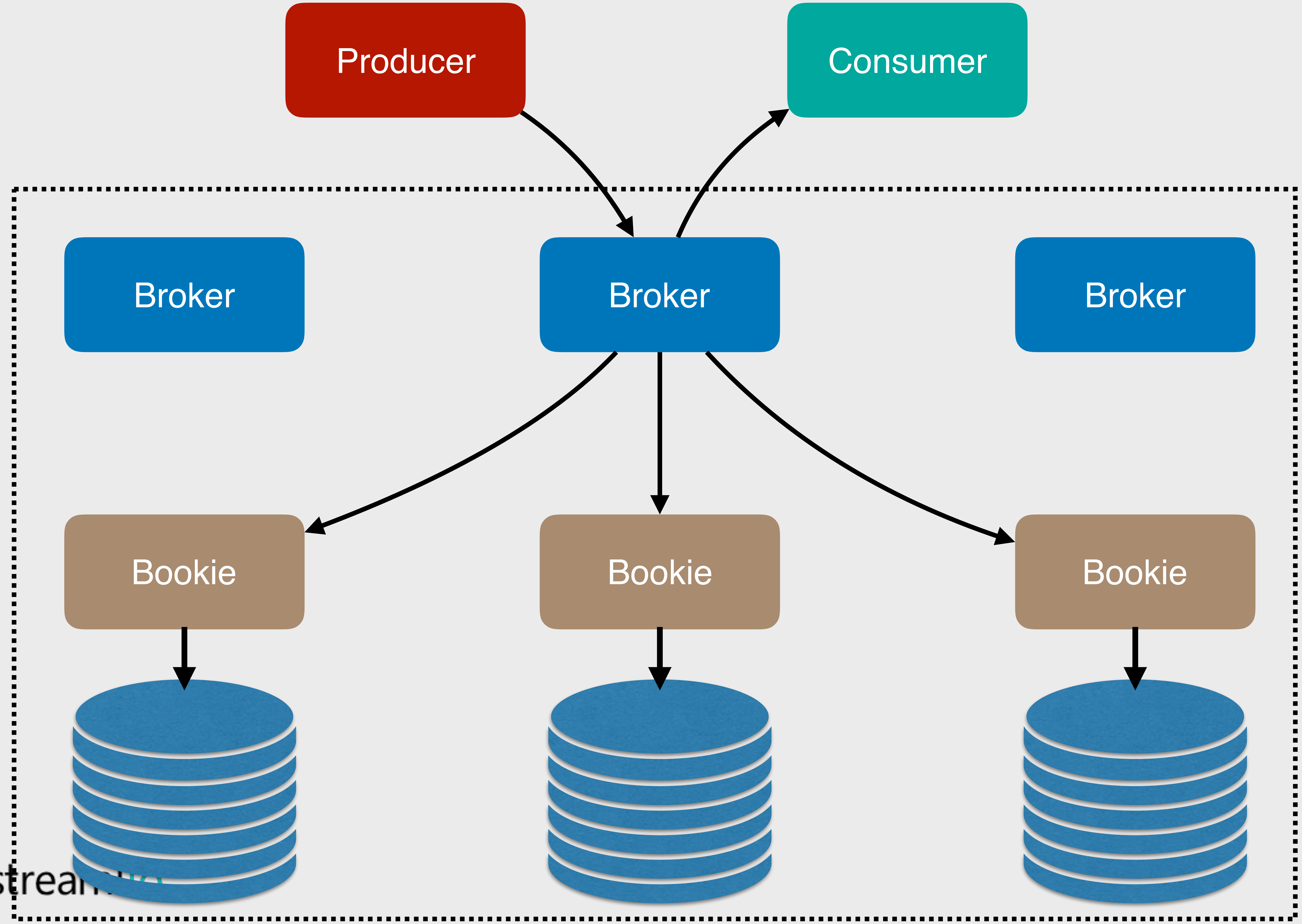
Segments



streamlio

Architecture

APACHE PULSAR



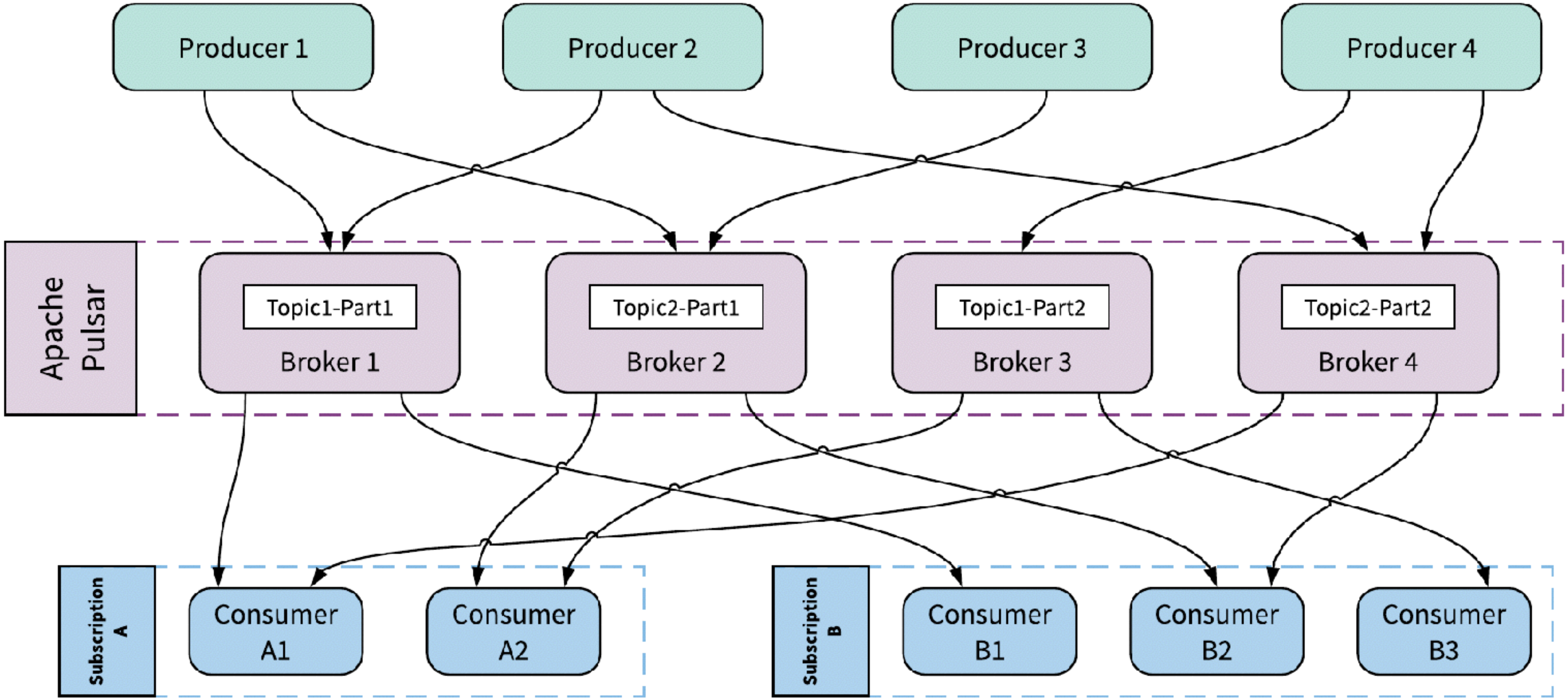
SERVING
Brokers can be added independently
Traffic can be shifted quickly across brokers

STORAGE
Bookies can be added independently
New bookies will ramp up traffic quickly

APACHE PULSAR - BROKER

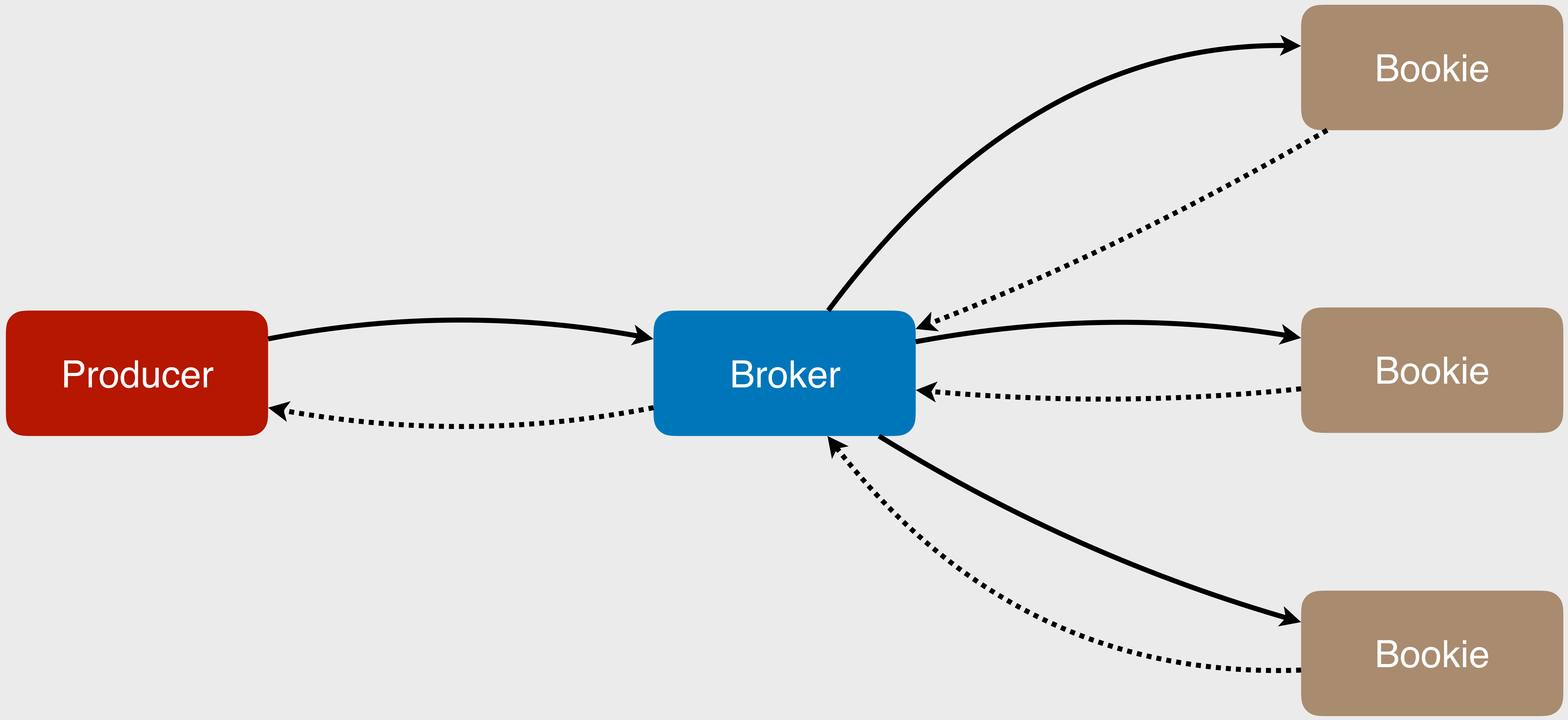
- ✦ Broker is the only point of interaction for clients (producers and consumers)
- ✦ Brokers acquire ownership of group of topics and “serve” them
- ✦ Broker has no durable state
- ✦ Provides service discovery mechanism for client to connect to right broker

APACHE PULSAR - BROKER



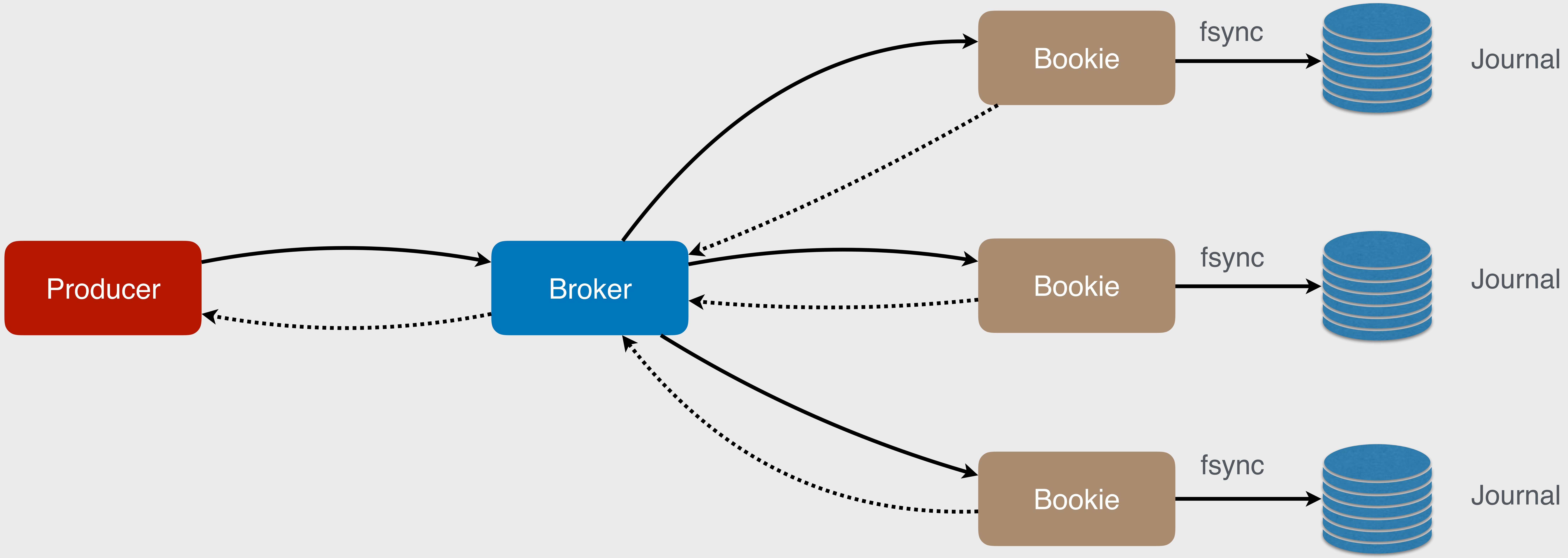
streamlio

APACHE PULSAR - CONSISTENCY



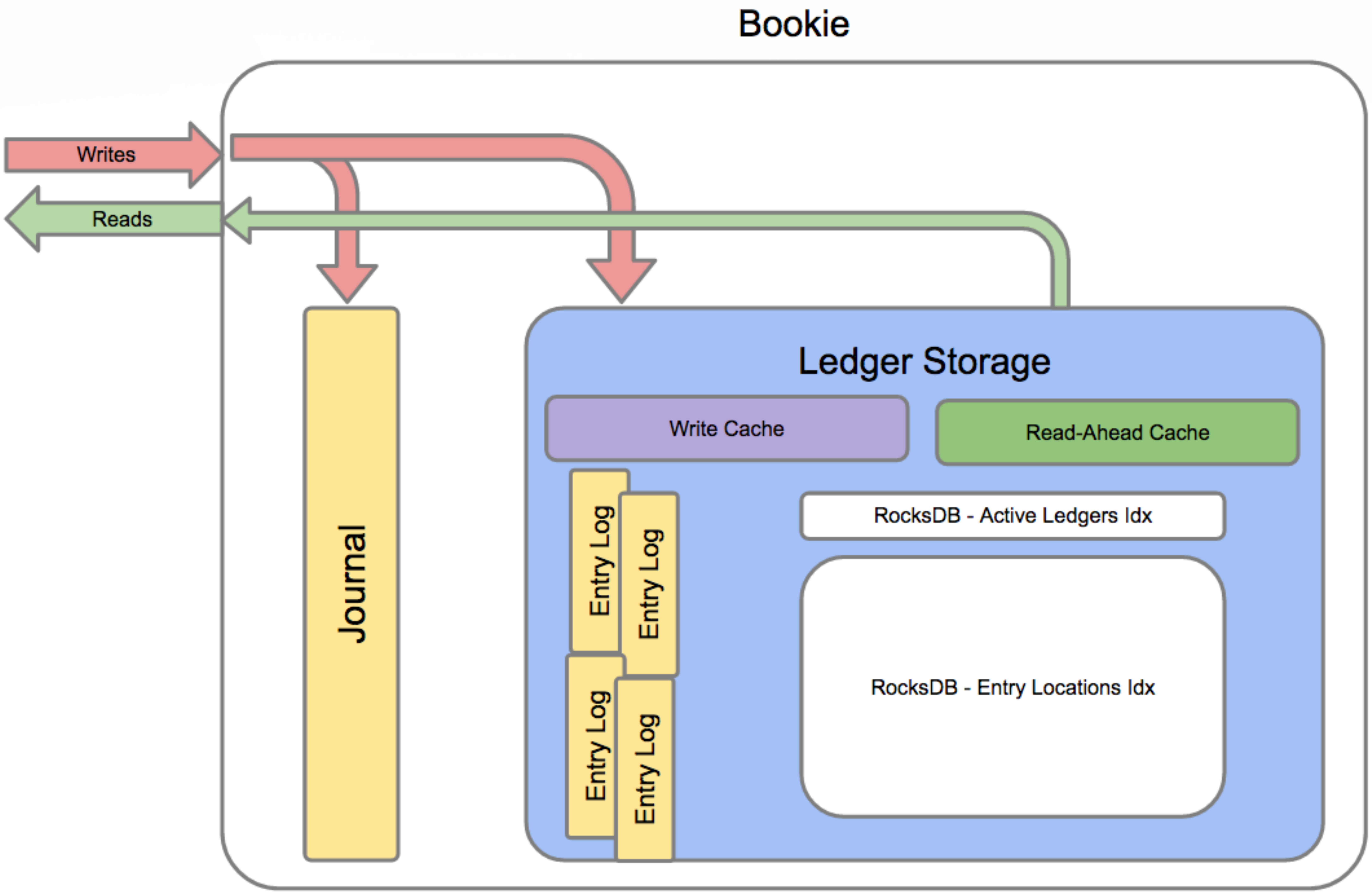
streamlio

APACHE PULSAR - DURABILITY (NO DATA LOSS)



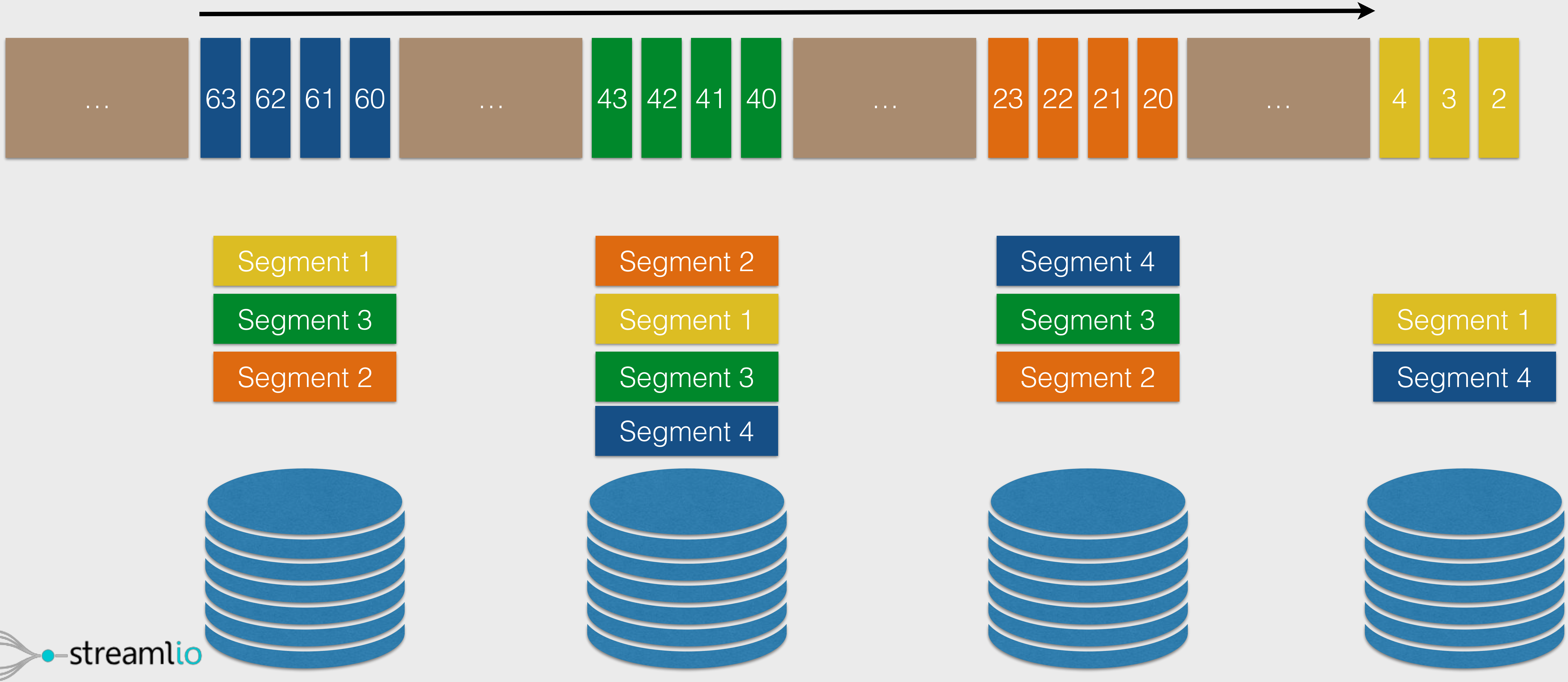
streamlio

APACHE PULSAR - ISOLATION



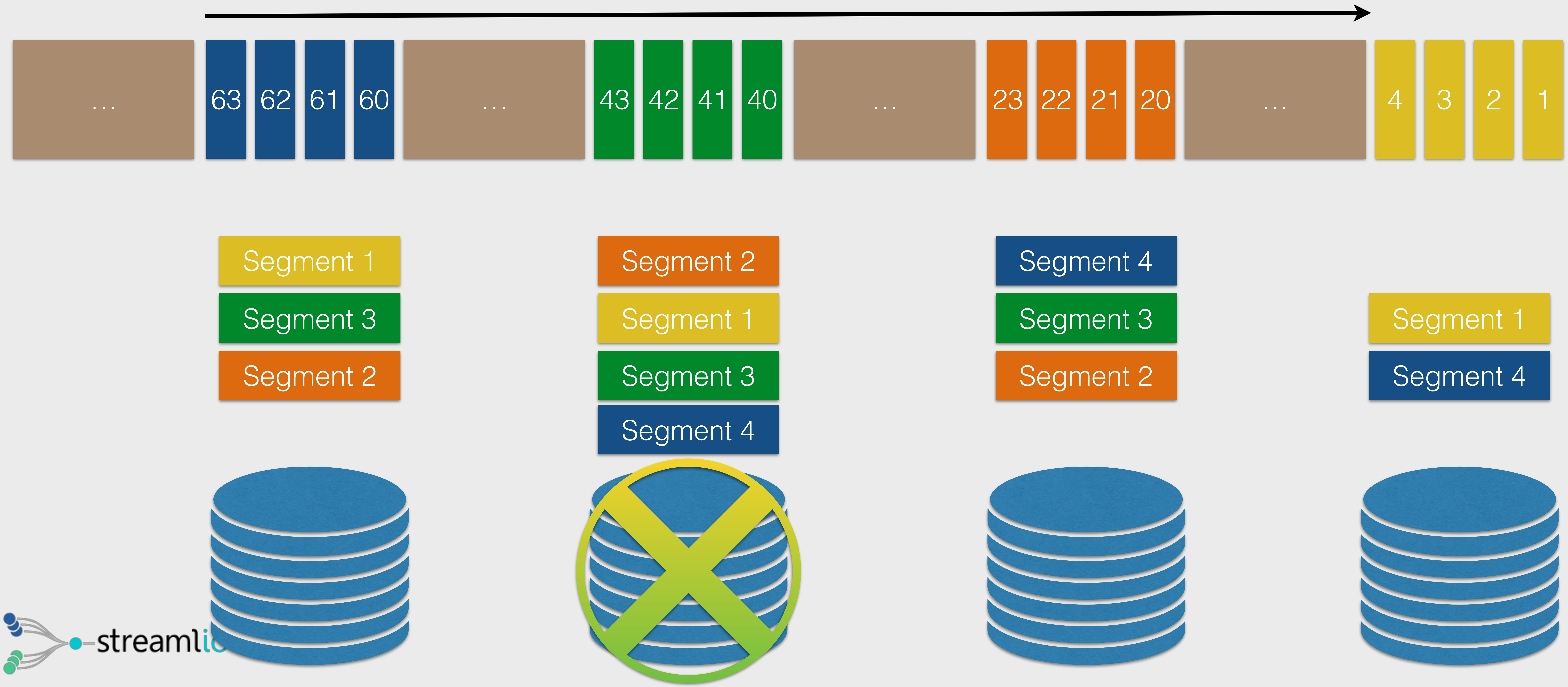
streamlio

APACHE PULSAR - SEGMENT STORAGE

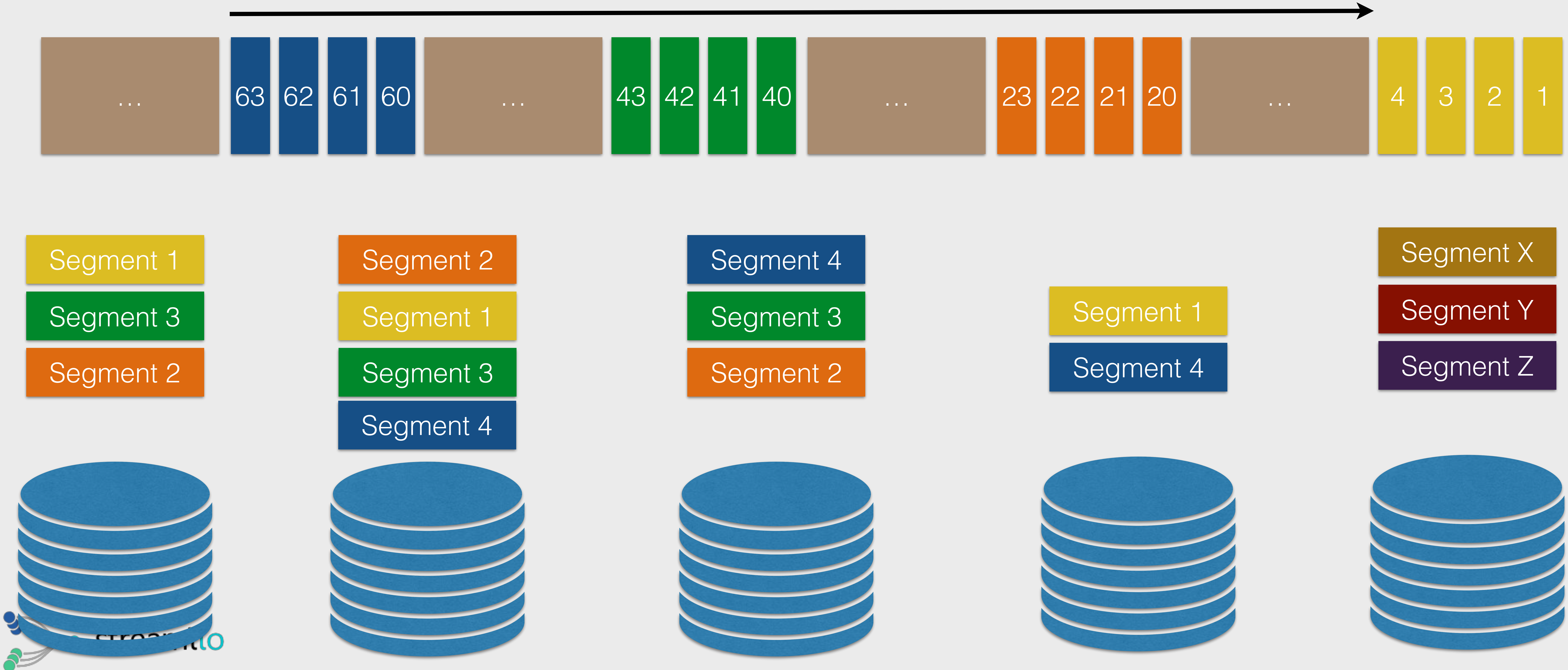


streamlio

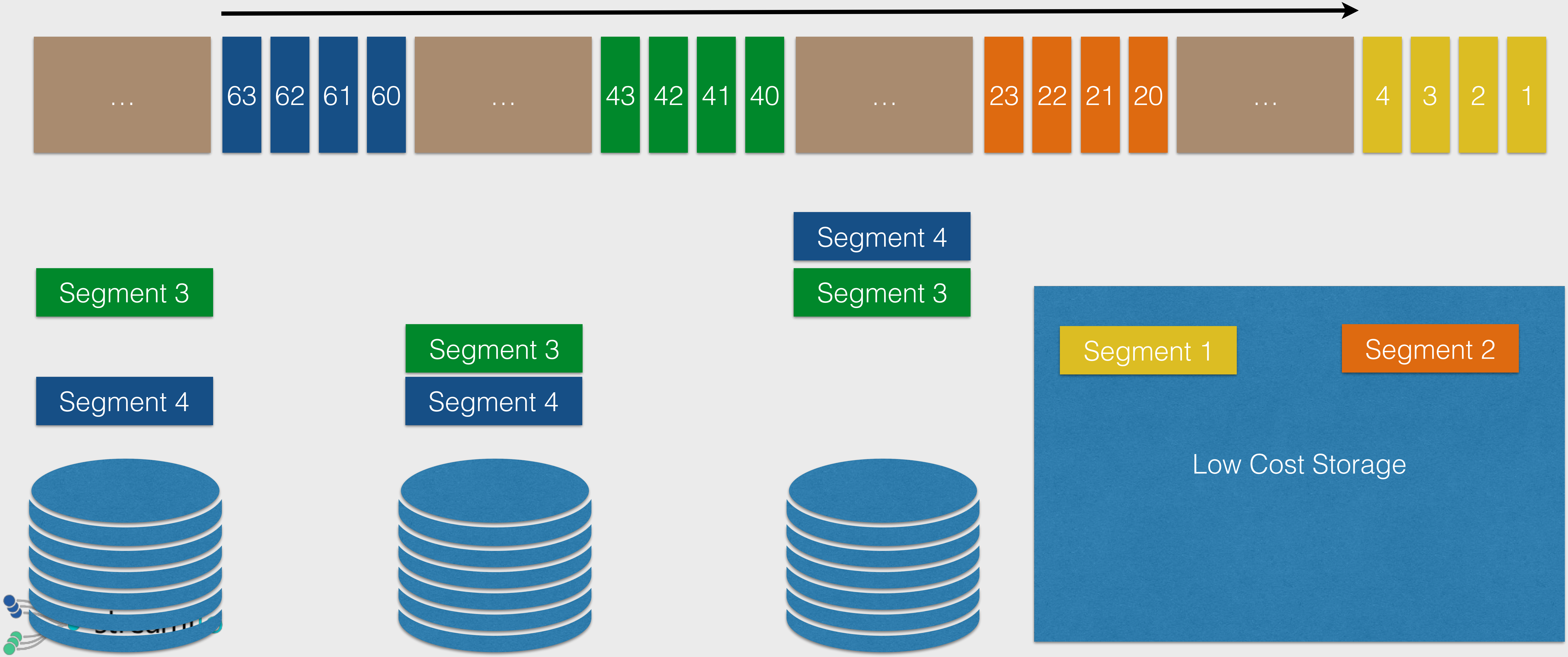
APACHE PULSAR - RESILIENCY



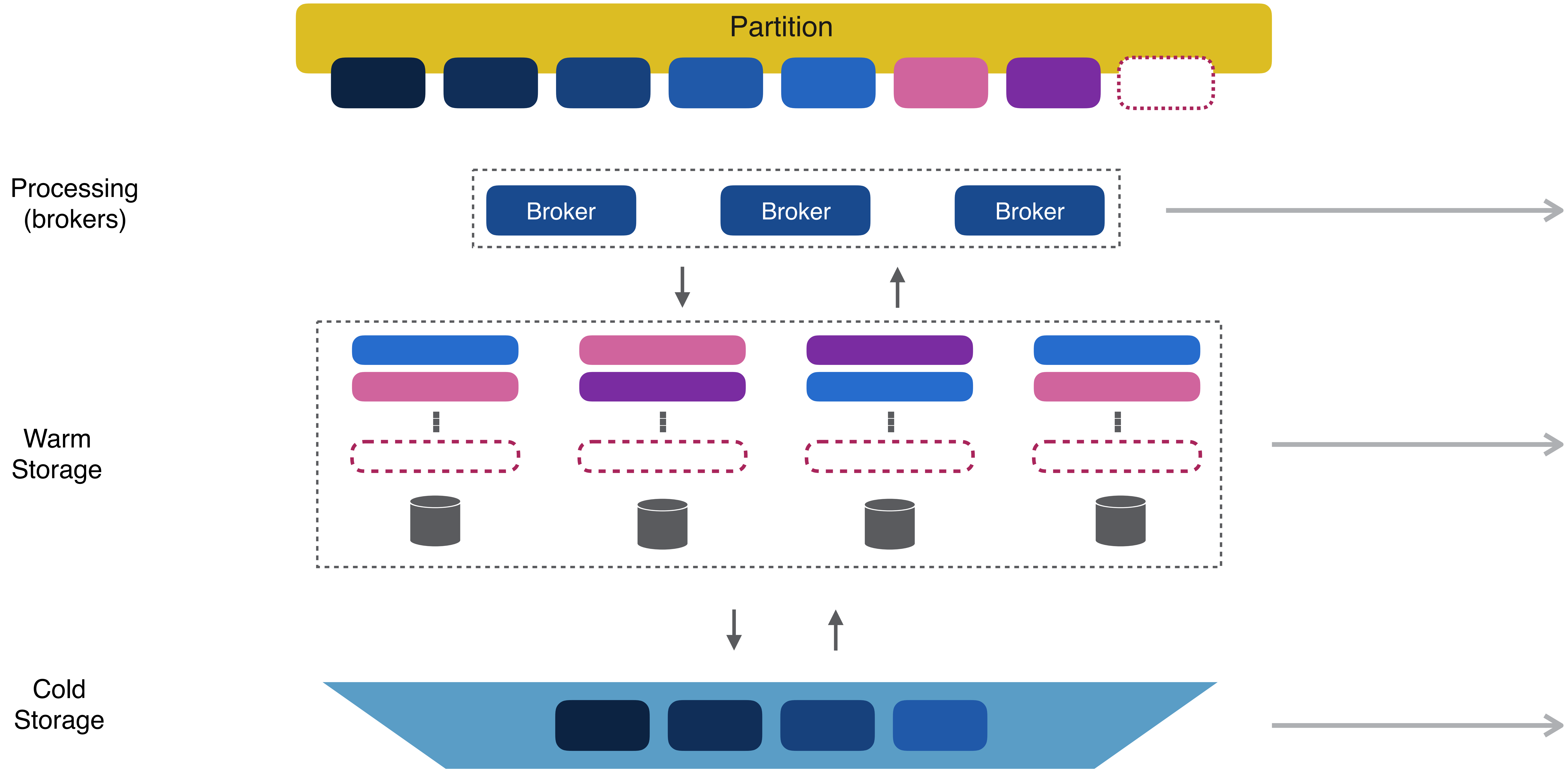
APACHE PULSAR - SEAMLESS CLUSTER EXPANSION



APACHE PULSAR - TIERED STORAGE



Multi-tiered storage and serving



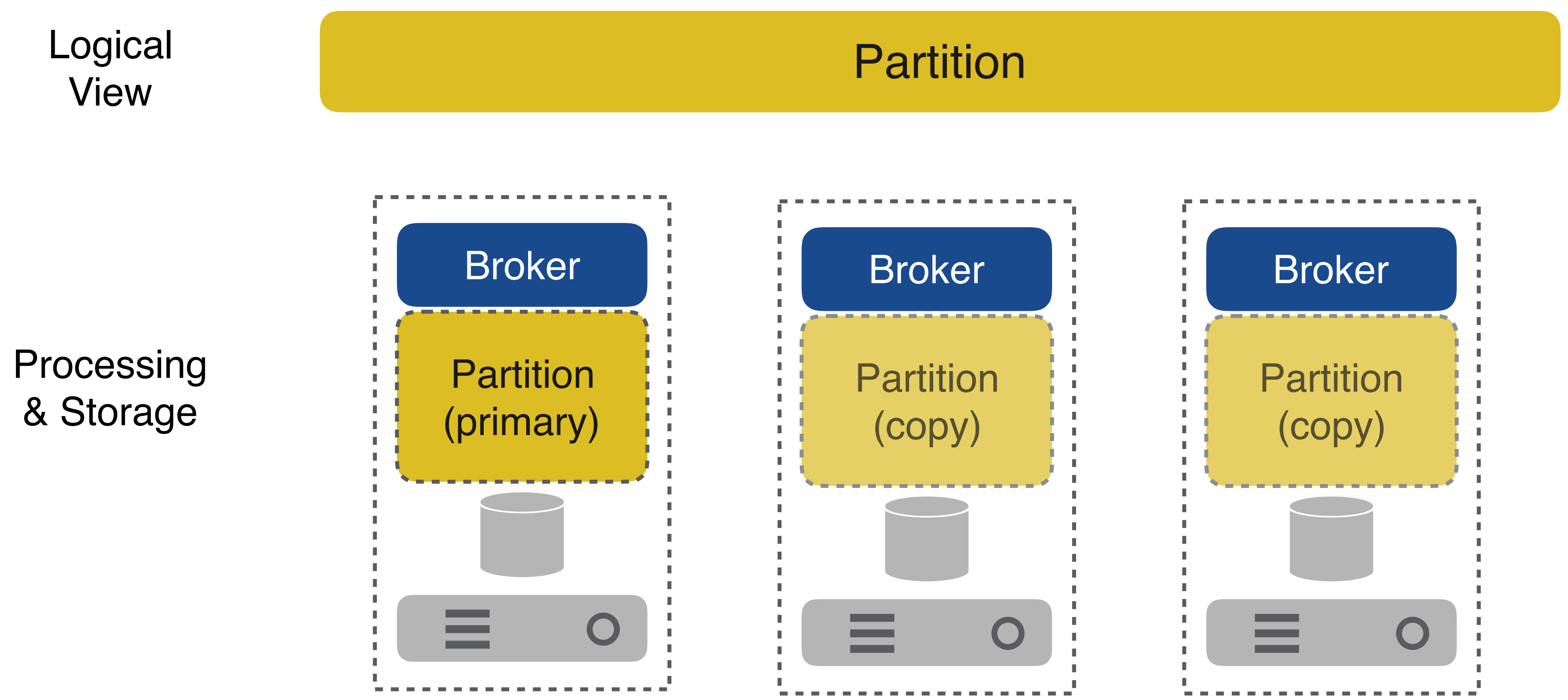
Tailing reads: served from in-memory cache

Catch-up reads: served from persistent storage layer

Historical reads: served from cold storage

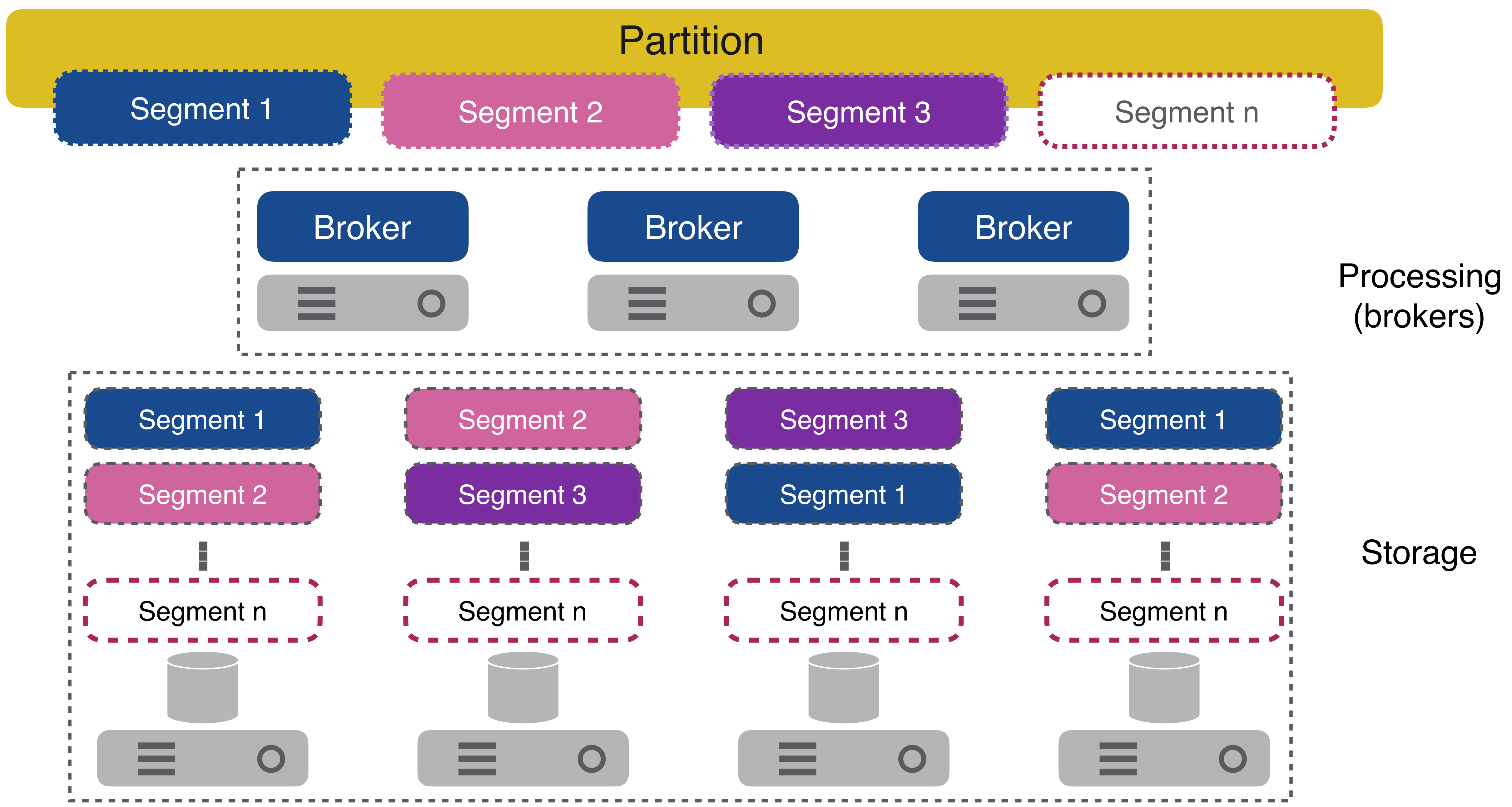
streamlio

PARTITIONS VS SEGMENTS - WHY SHOULD YOU CARE?



Legacy Architectures

- Storage co-resident with processing
- Partition-centric
- Cumbersome to scale--data redistribution, performance impact



Apache Pulsar

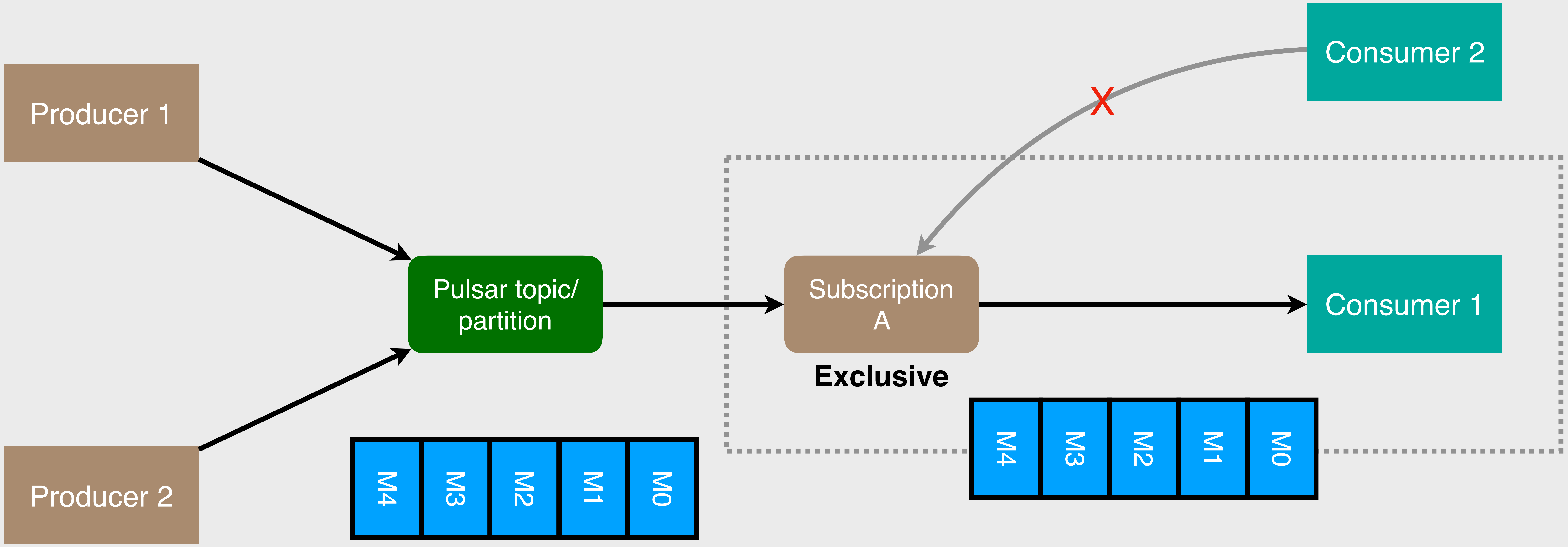
- Storage decoupled from processing
- Partitions stored as segments
- Flexible, easy scalability

streamlio

PARTITIONS VS SEGMENTS - WHY SHOULD YOU CARE?

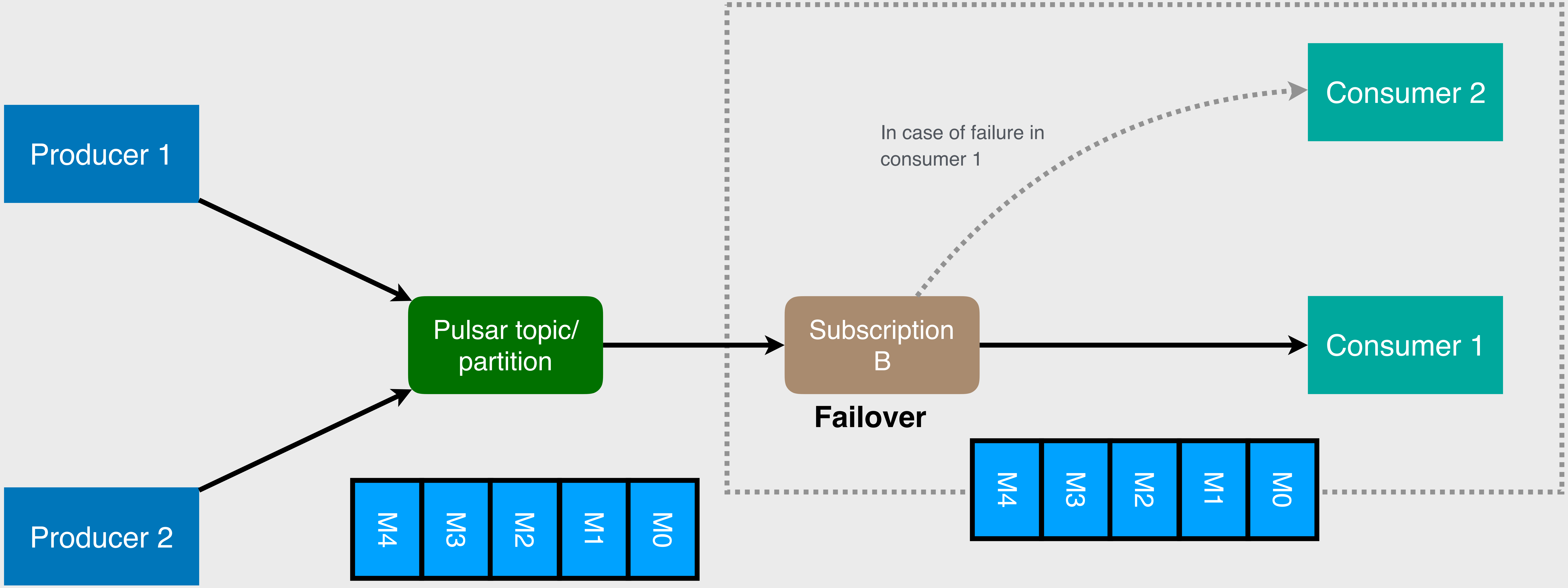
- In Kafka, partitions are assigned to brokers “permanently”
- A single partition is stored entirely in a single node
- Retention is limited by a single node storage capacity
- Failure recovery and capacity expansion require expensive “rebalancing”
- Rebalancing has a big impact over the system, affecting regular traffic

UNIFIED MESSAGING MODEL - STREAMING



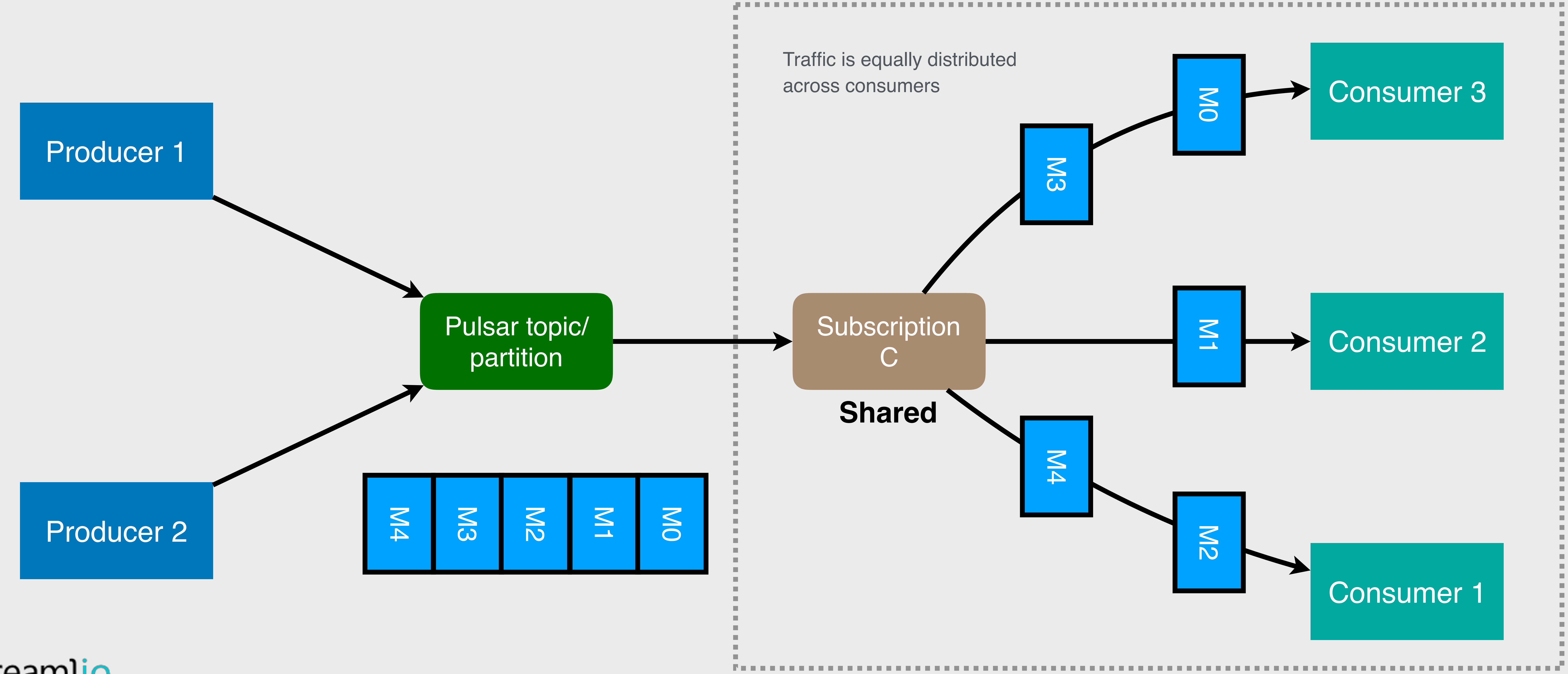
streamlio

UNIFIED MESSAGING MODEL - STREAMING



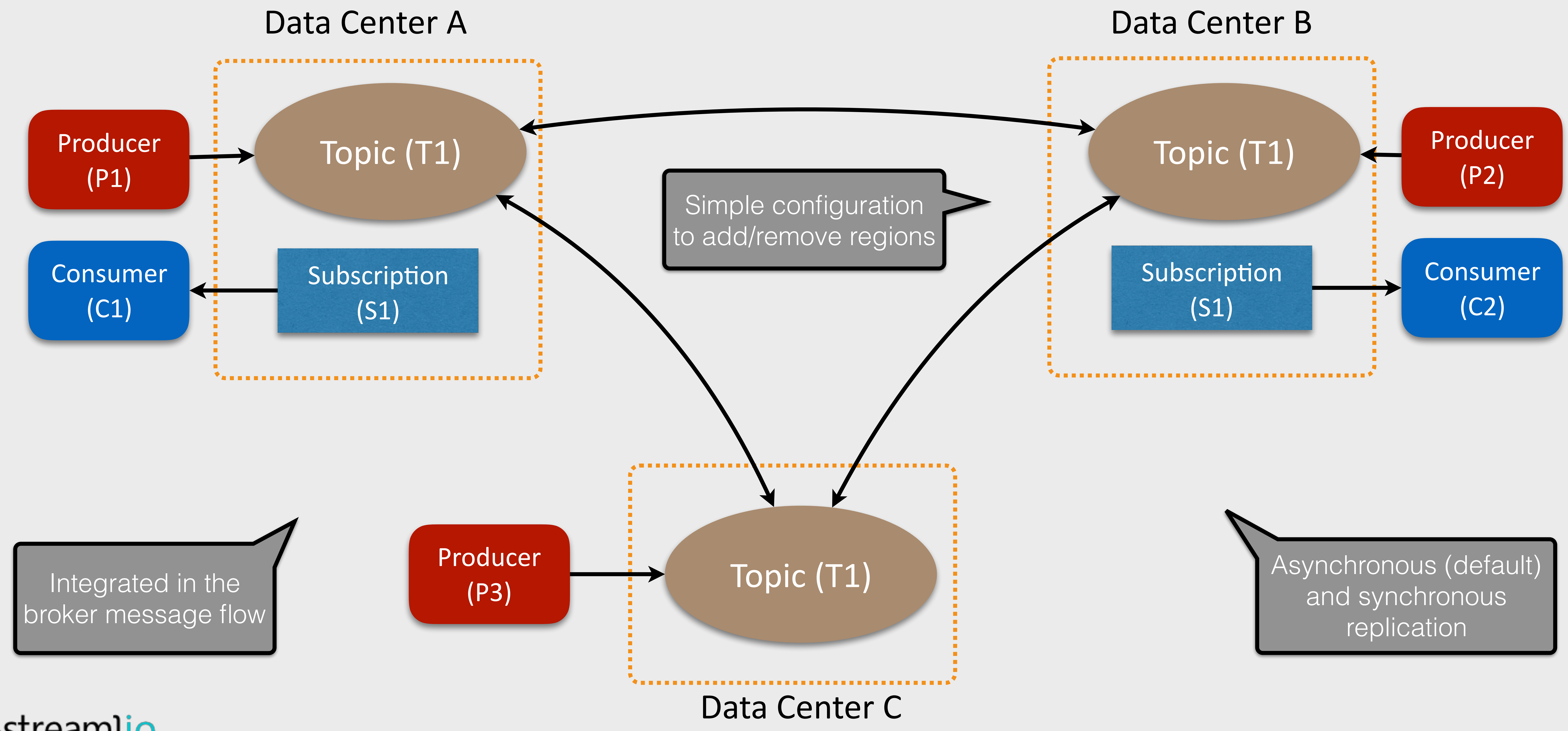
streamlio

UNIFIED MESSAGING MODEL - QUEUING



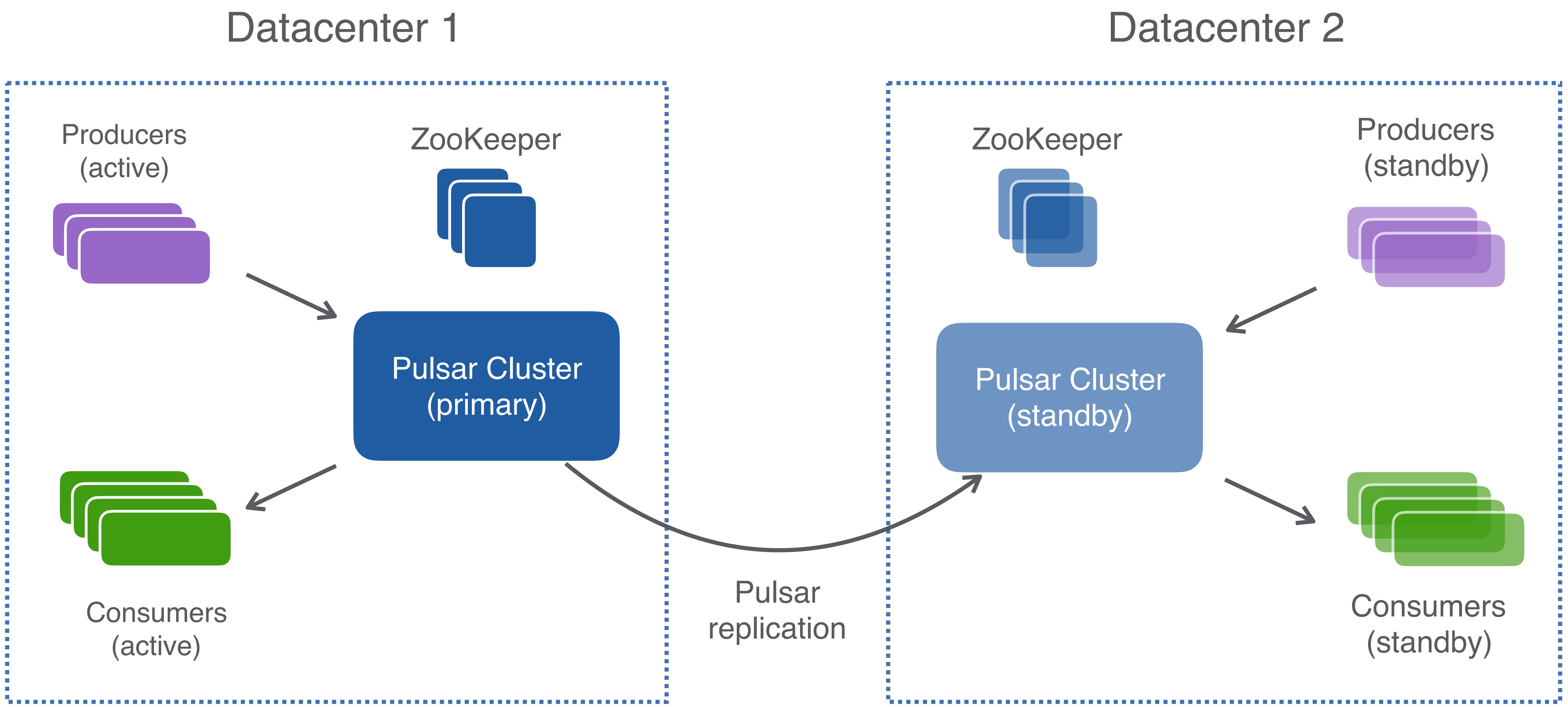
streamlio

DISASTER RECOVERY



streamlio

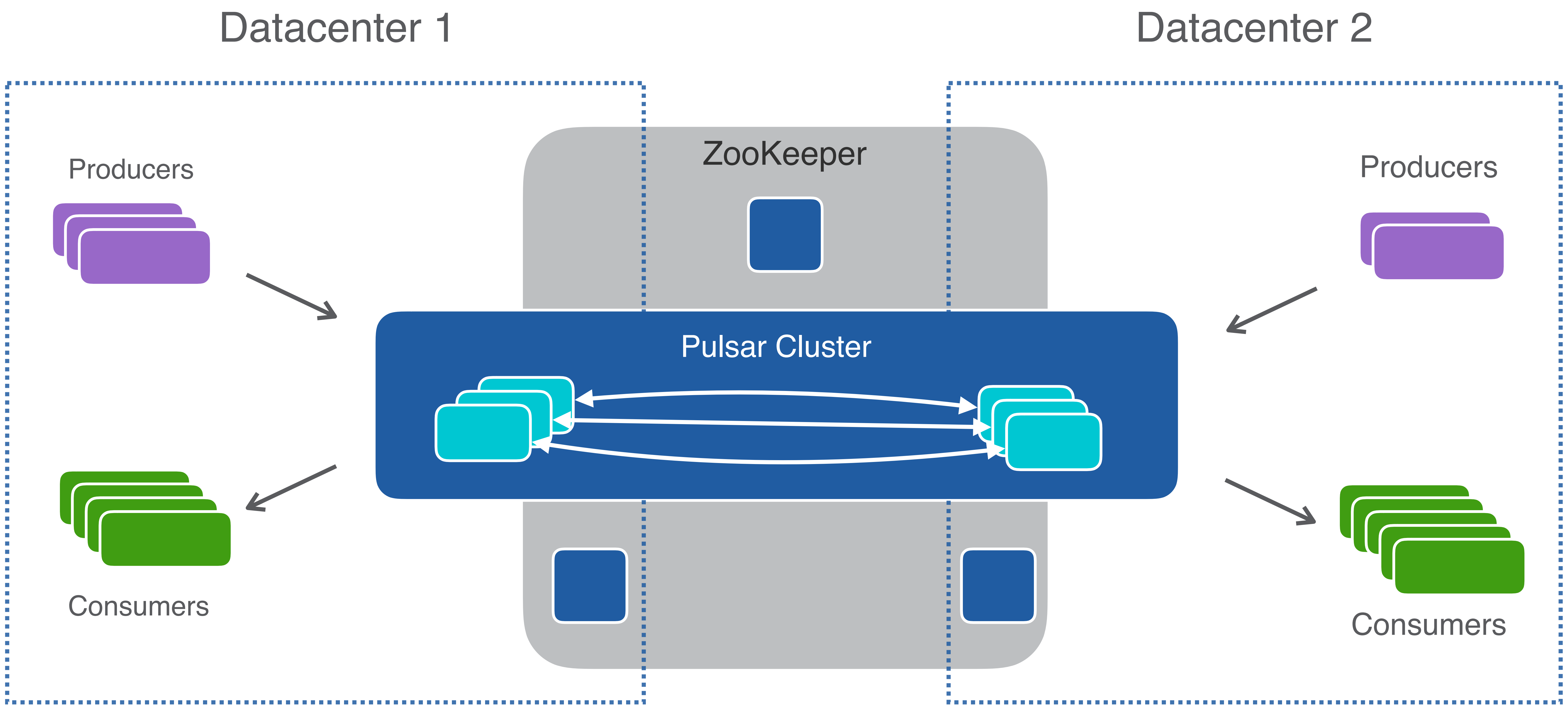
Asynchronous replication example



- Two independent clusters, primary and standby
- Configured tenants and namespaces replicate to standby
- Data published to primary is asynchronously replicated to standby
- Producers and consumers restarted in second datacenter upon primary failure

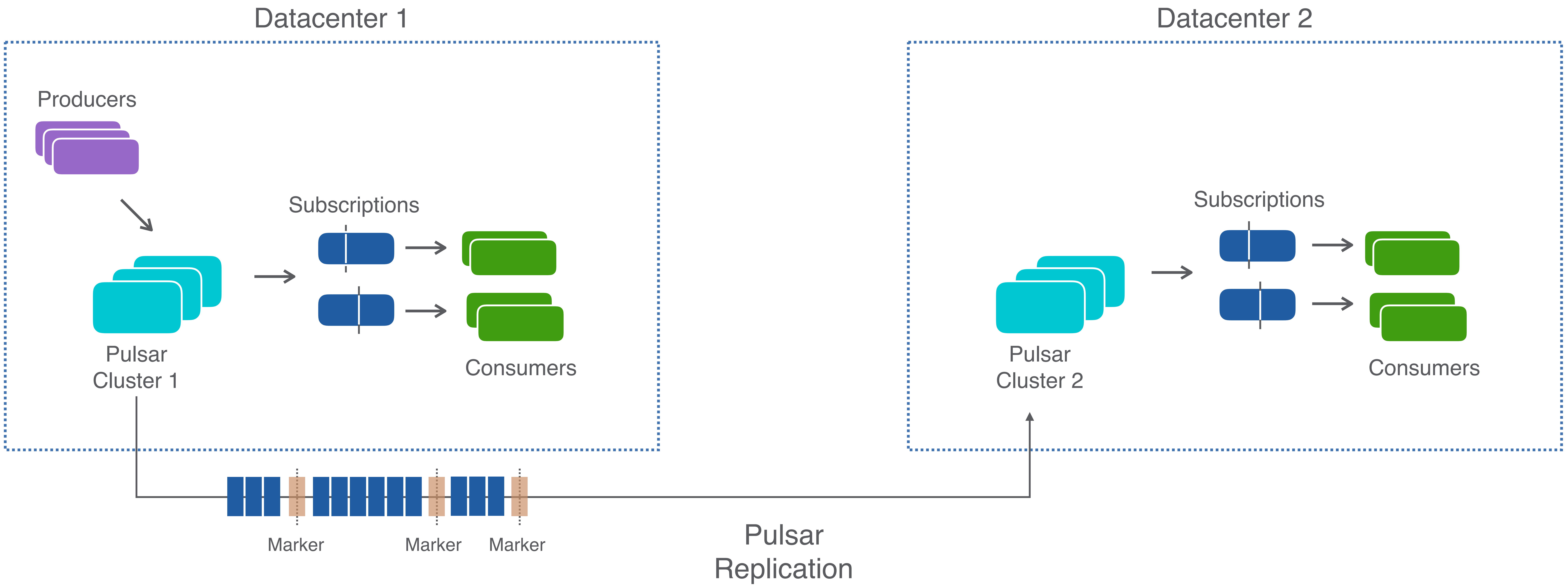
streamlio

Synchronous replication example

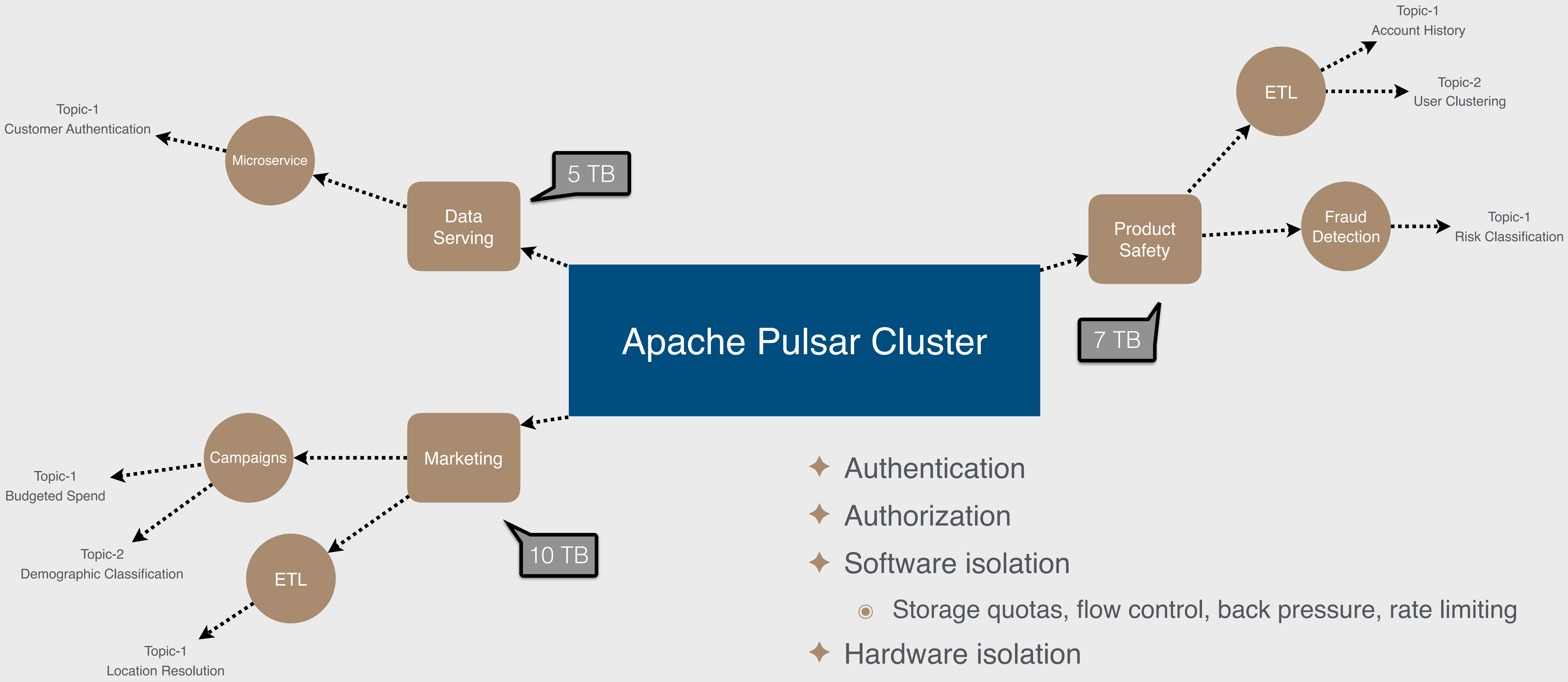


- Each topic owned by one broker at a time, i.e. in one datacenter
- ZooKeeper cluster spread across multiple locations
- Broker commits writes to bookies in both datacenters
- In event of datacenter failure, broker in surviving datacenter assumes ownership of topic

Replicated subscriptions

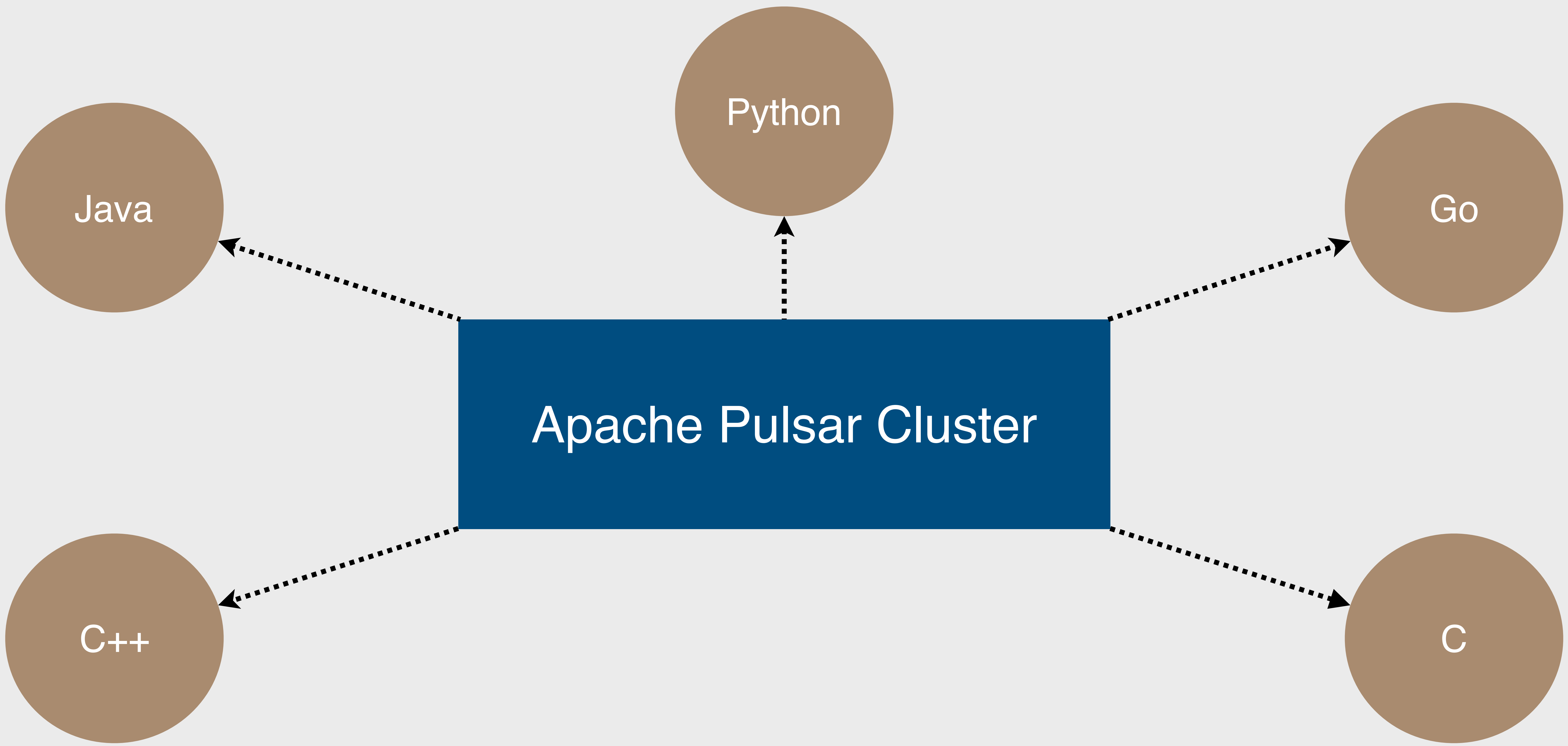


MULTITENANCY - CLOUD NATIVE



- Authentication
- Authorization
- Software isolation
  - Storage quotas, flow control, back pressure, rate limiting
- Hardware isolation
  - Constrain some tenants on a subset of brokers/bookies

PULSAR CLIENTS



streamlio

PULSAR PRODUCER

```
PulsarClient client = PulsarClient.create(
                                     "http://broker.usw.example.com:8080");

Producer producer = client.createProducer(
         "persistent://my-property/us-west/my-namespace/my-topic");

// handles retries in case of failure
producer.send("my-message".getBytes());

// Async version:
producer.sendAsync("my-message".getBytes()).thenRun(() -> {
     // Message was persisted
});
```

PULSAR CONSUMER

```java
PulsarClient client = PulsarClient.create(
                                    "http://broker.usw.example.com:8080");

Consumer consumer = client.subscribe(
      "persistent://my-property/us-west/my-namespace/my-topic",
      "my-subscription-name");

while (true) {
  // Wait for a message
  Message msg = consumer.receive();

  System.out.println("Received message: " + msg.getData());

  // Acknowledge the message so that it can be deleted by broker
  consumer.acknowledge(msg);
}
```

SCHEMA REGISTRY

- Provides type safety to applications built on top of Pulsar
- Two approaches
  - Client side - type safety enforcement up to the application
  - Server side - system enforces type safety and ensures that producers and consumers remain synced
- Schema registry enables clients to upload data schemas on a topic basis.
- Schemas dictate which data types are recognized as valid for that topic

PULSAR SCHEMAS - HOW DO THEY WORK?

- Enforced at the topic level
- Pulsar schemas consists of
  - Name - Name refers to the topic to which the schema is applied
  - Payload - Binary representation of the schema
  - Schema type - JSON, Protobuf and Avro
  - User defined properties - Map of strings to strings (application specific - e.g git hash of the schema)

SCHEMA VERSIONING

```
PulsarClient client = PulsarClient.builder()
     .serviceUrl("http://broker.usw.example.com:6650")
     .build()

Producer<SensorReading> producer = client.newProducer(JSONSchema.of(SensorReading.class))
     .topic("sensor-data")
     .sendTimeout(3, TimeUnit.SECONDS)
     .create()
```

Scenario	What happens
No schema exists for the topic	Producer is created using the given schema
Schema already exists; producer connects using the same schema that's already stored	Schema is transmitted to the broker, determines that it is already stored
Schema already exists; producer connects using a new schema that is compatible	Schema is transmitted, compatibility determined and stored as new schema

streamlio

Processing framework

HOW TO PROCESS DATA MODELED AS STREAMS

- ✦ Consume data as it is produced (pub/sub)
- ✦ Light weight compute - transform and react to data as it arrives
- ✦ Heavy weight compute - continuous data processing
- ✦ Interactive query of stored streams

LIGHT WEIGHT COMPUTE

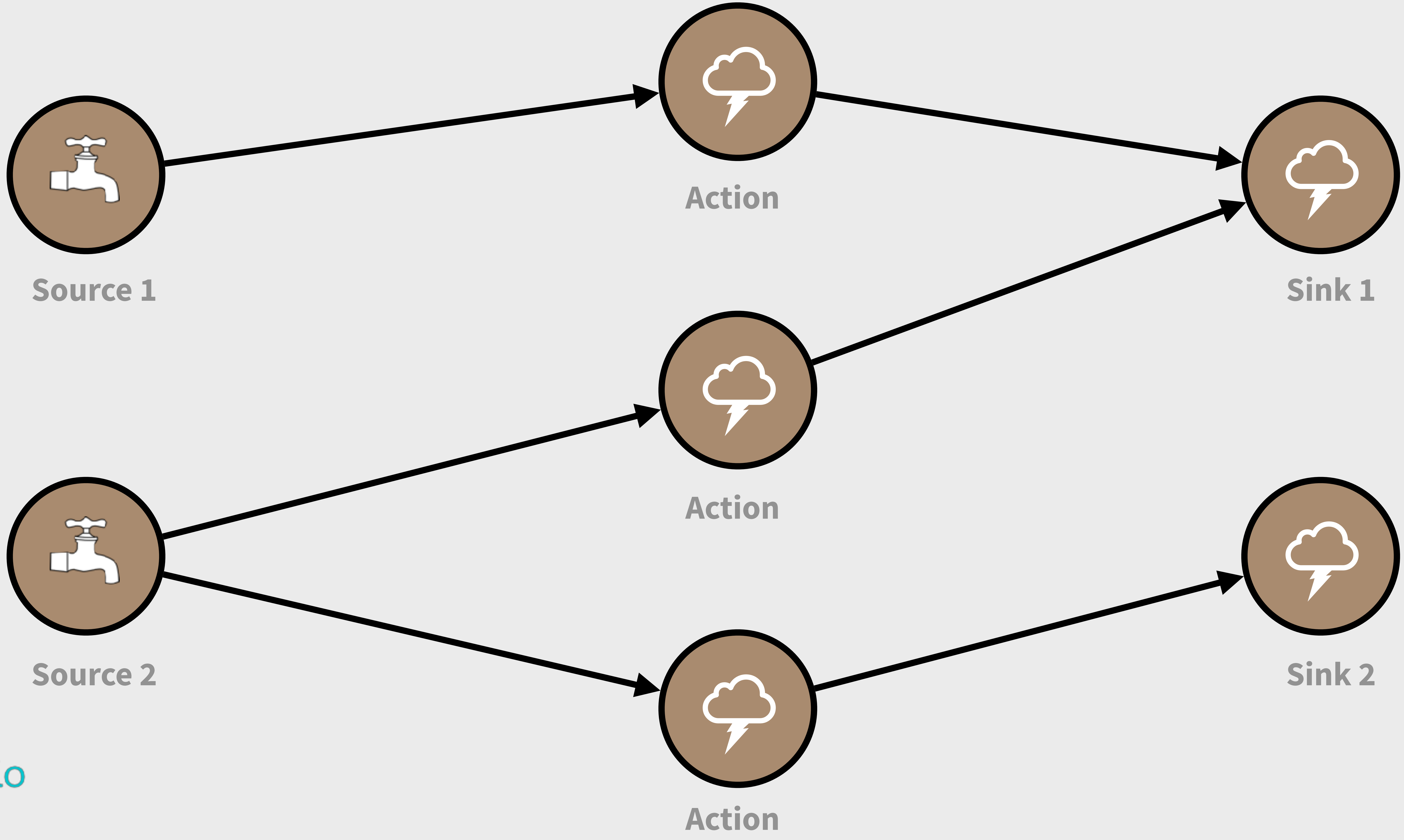
ABSTRACT VIEW OF COMPUTE REPRESENTATION

Incoming Messages

Output Messages

TRADITIONAL COMPUTE REPRESENTATION

DAG



REALIZING COMPUTATION - EXPLICIT CODE

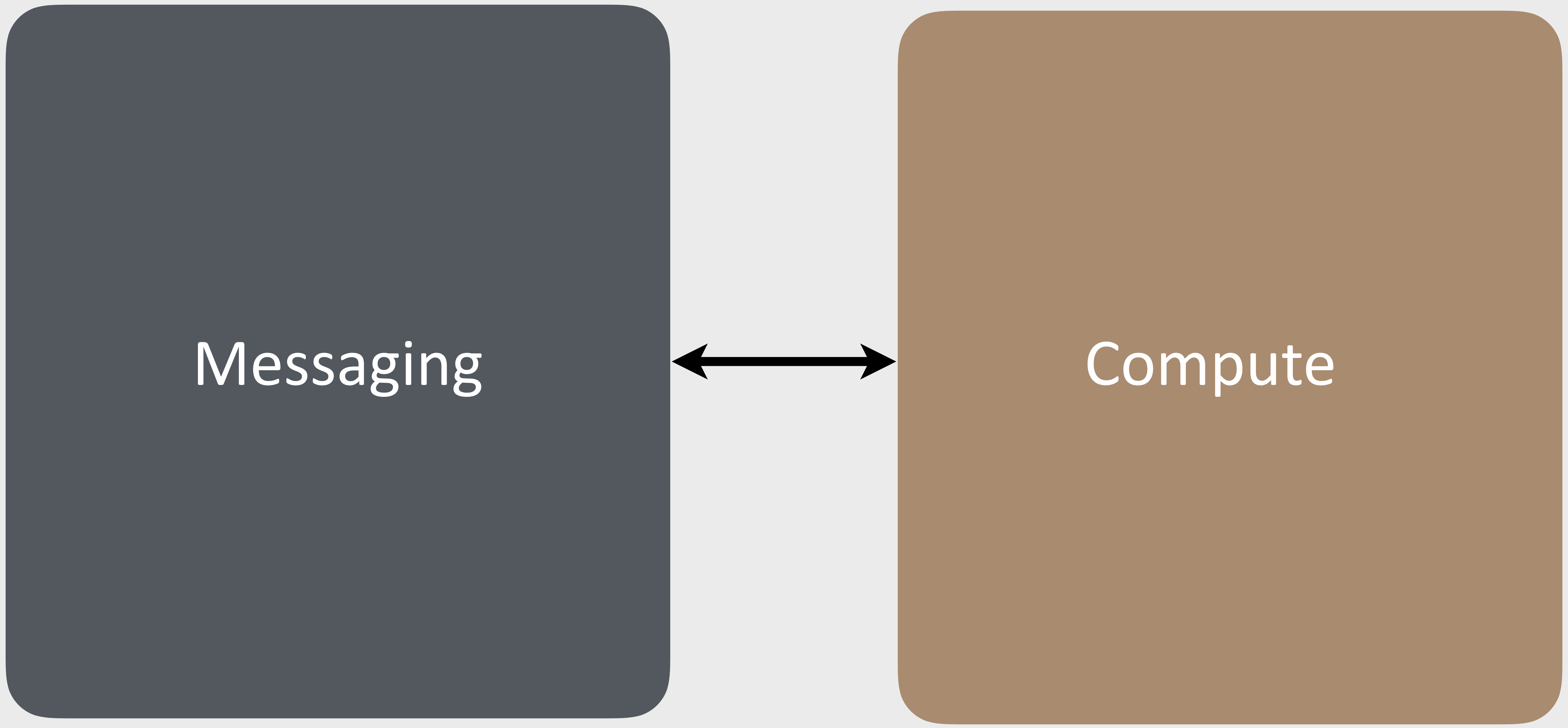
STITCHED BY PROGRAMMERS

```
public static class SplitSentence extends BaseBasicBolt {
    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
      declarer.declare(new Fields("word"));
    }

    @Override
    public Map<String, Object> getComponentConfiguration() {
      return null;
    }

    public void execute(Tuple tuple, BasicOutputCollector 
basicOutputCollector) {
      String sentence = tuple.getStringByField("sentence");
      String words[] = sentence.split(" ");
      for (String w : words) {
        basicOutputCollector.emit(new Values(w));
      }
    }
}
```

REALIZING COMPUTATION - FUNCTIONAL

```
Builder.newBuilder()
     .newSource(() -> StreamletUtils.randomFromList(SENTENCES))
     .flatMap(sentence -> Arrays.asList(sentence.toLowerCase().split("\\s+")))
     .reduceByKeyAndWindow(word -> word, word -> 1,
                           WindowConfig.TumblingCountWindow(50),
                           (x, y) -> x + y);
```

TRADITIONAL REAL TIME - SEPARATE SYSTEMS



streamlio

TRADITIONAL REAL TIME SYSTEMS

DEVELOPER EXPERIENCE

- Powerful API but complicated
  - Does everyone really need to learn functional programming?
- Configurable and scalable but management overhead
- Edge systems have resource and management constraints

TRADITIONAL REAL TIME SYSTEMS

OPERATIONAL EXPERIENCE

- Multiple systems to operate
  - IoT deployments routinely have thousands of edge systems
- Semantic differences
  - Mismatch and duplication between systems
  - Creates developer and operator friction

LESSONS LEARNT - USE CASES

- Data transformations
- Data classification
- Data enrichment
- Data routing
- Data extraction and loading
- Real time aggregation
- Microservices

streamlio

Significant set of processing tasks are exceedingly simple

EMERGENCE OF CLOUD - SERVERLESS

- Simple function API
- Functions are submitted to the system
- Runs per events
- Composition APIs to do complex things
- Wildly popular

SERVERLESS VS STREAMING

- Both are event driven architectures
- Both can be used for analytics and data serving
- Both have composition APIs
  - Configuration based for serverless
  - DSL based for streaming
- Serverless typically does not guarantee ordering
- Serverless is pay per action

STREAM NATIVE COMPUTE USING FUNCTIONS

APPLYING INSIGHT GAINED FROM SERVERLESS

- Simplest possible API -function or a procedure
- Support for multi language
- Use of native API for each language
- Scale developers
- Use of message bus native concepts - input and output as topics
- Flexible runtime - simple standalone applications vs managed system applications

PULSAR FUNCTIONS

SDK LESS API

```
import java.util.function.Function;
public class ExclamationFunction implements Function<String, String> {
    @Override
    public String apply(String input) {
        return input + "!";
    }
}
```

PULSAR FUNCTIONS

SDK API

```java
import org.apache.pulsar.functions.api.PulsarFunction;
import org.apache.pulsar.functions.api.Context;
public class ExclamationFunction implements PulsarFunction<String, String> {
    @Override
    public String process(String input, Context context) {
        return input + "!";
    }
}
```

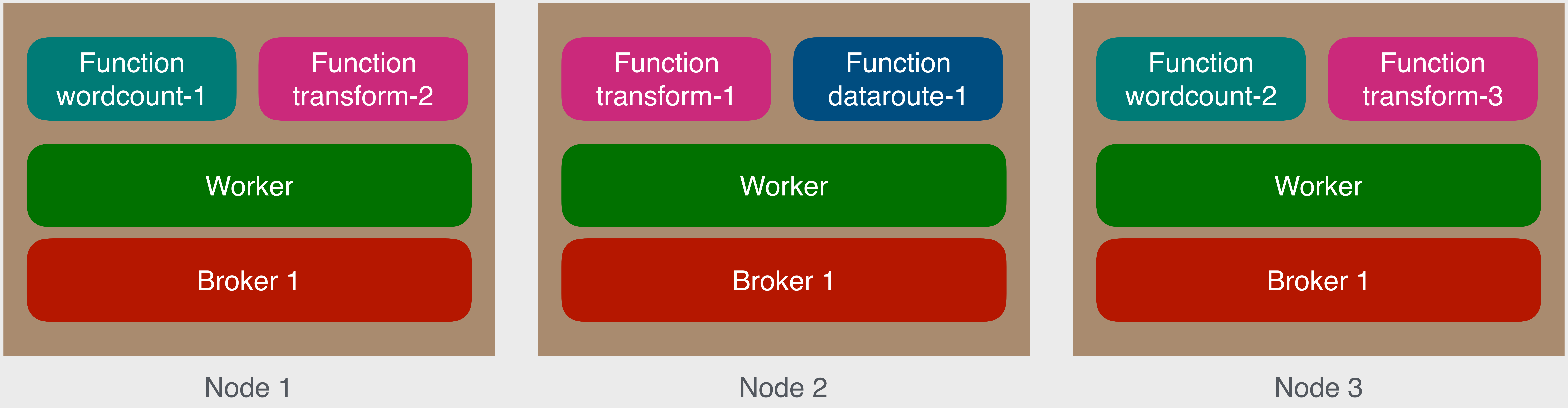
PULSAR FUNCTIONS

- Function executed for every message of input topic
- Support for multiple topics as inputs
- Function output goes into output topic - can be void topic as well
- SerDe takes care of serialization/deserialization of messages
    - Custom SerDe can be provided by the users
    - Integration with schema registry

PROCESSING GUARANTEES

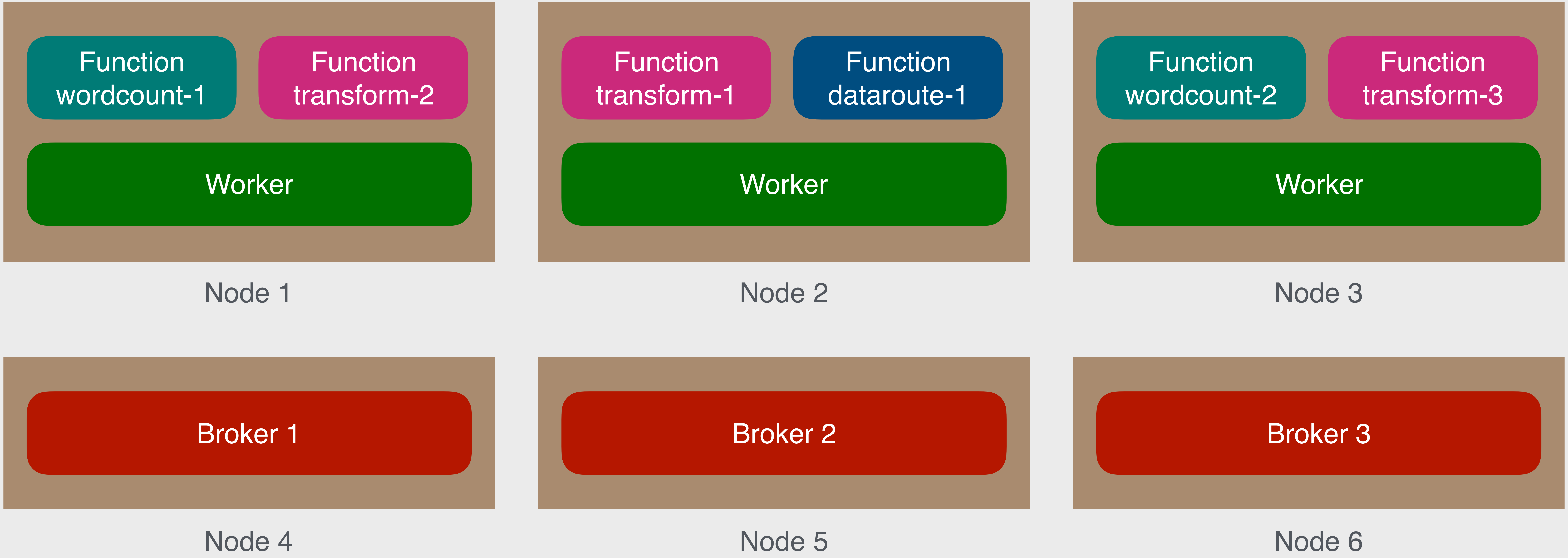
- ATMOST_ONCE
  - Message acked to Pulsar as soon as we receive it
- ATLEAST_ONCE
  - Message acked to Pulsar after the function completes
  - Default behavior - don’t want people to loose data
- EFFECTIVELY_ONCE
  - Uses Pulsar’s inbuilt effectively once semantics
- Controlled at runtime by user

DEPLOYING FUNCTIONS - BROKER



streamlio

DEPLOYING FUNCTIONS - WORKER NODES



streamlio

DEPLOYING FUNCTIONS - KUBERNETES

Function
wordcount-1

Pod 1

Function
transform-1

Pod 2

Function
transform-3

Pod 3

Function
dataroute-1

Pod 4

Function
wordcount-2

Pod 5

Function
transform-2

Pod 6

Broker 1

Pod 7

Broker 2

Pod 8

Broker 3

Pod 9

streamlio

BUILT-IN STATE MANAGEMENT IN FUNCTIONS

- Functions can store state in inbuilt storage
  - Framework provides a simple library to store and retrieve state
- Support server side operations like counters
- Simplified application development
  - No need to standup an extra system

DISTRIBUTED STATE IN FUNCTIONS

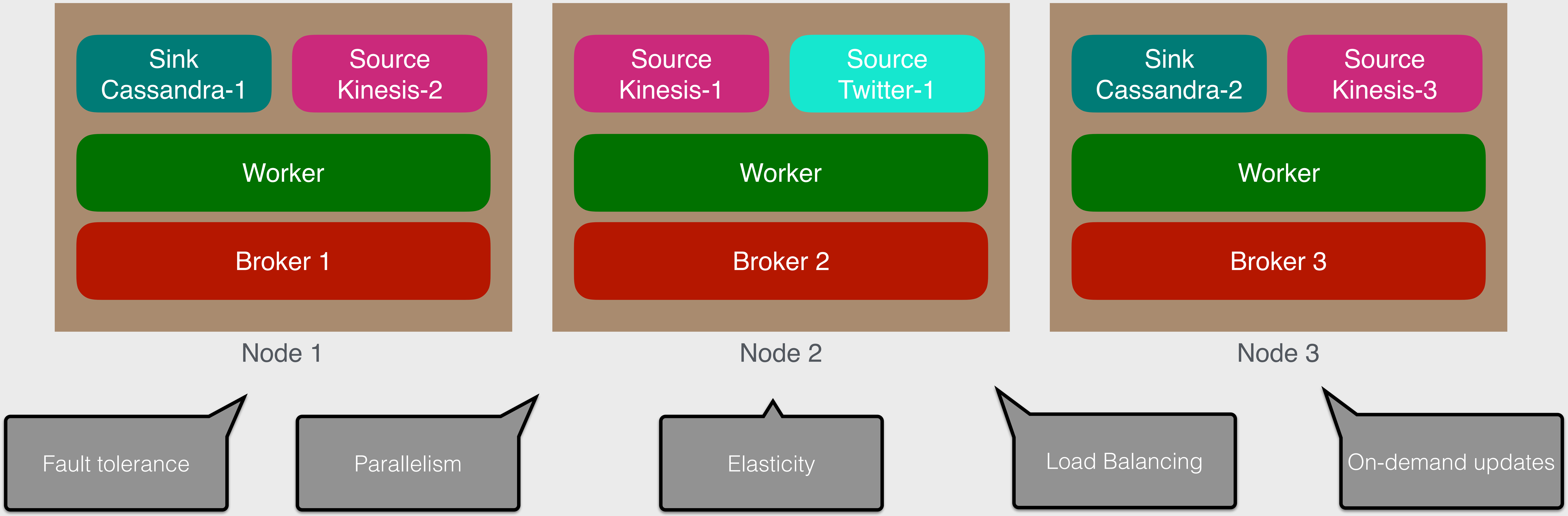
```
import org.apache.pulsar.functions.api.Context;
import org.apache.pulsar.functions.api.PulsarFunction;

public class CounterFunction implements PulsarFunction<String, Void> {
    @Override
    public Void process(String input, Context context) throws Exception {
        for (String word : input.split("\\.")) {
            context.incrCounter(word, 1);
        }
        return null;
    }
}
```

PULSAR - DATA IN AND OUT

- Users can write custom code using Pulsar producer and consumer API
- Challenges
  - Where should the application to publish data or consume data from Pulsar?
  - How should the application to publish data or consume data from Pulsar?
- Current systems have no organized and fault tolerant way to run applications that ingress and egress data from and to external systems

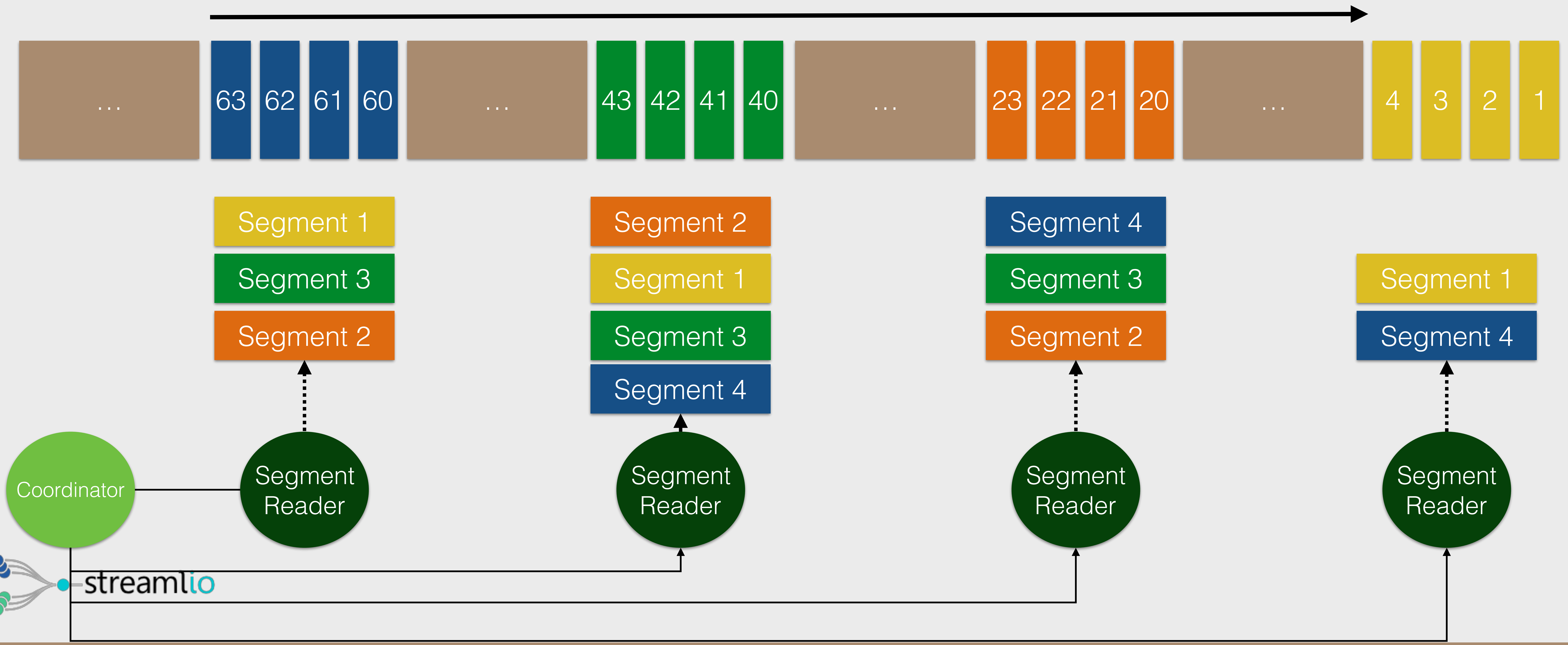
PULSAR IO TO THE RESCUE

PULSAR IO - EXECUTION



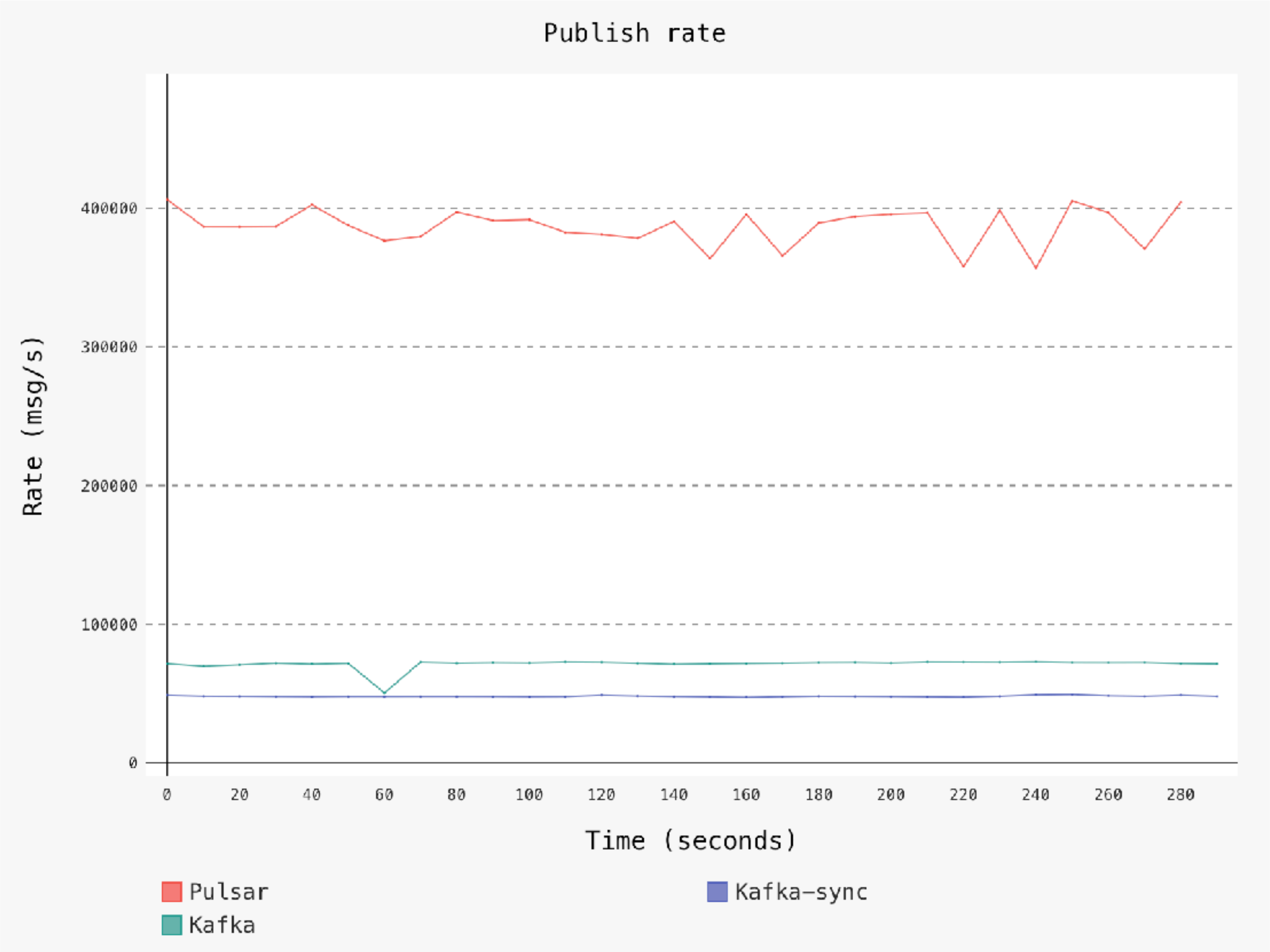
streamlio

INTERACTIVE QUERYING OF STREAMS - PULSAR SQL

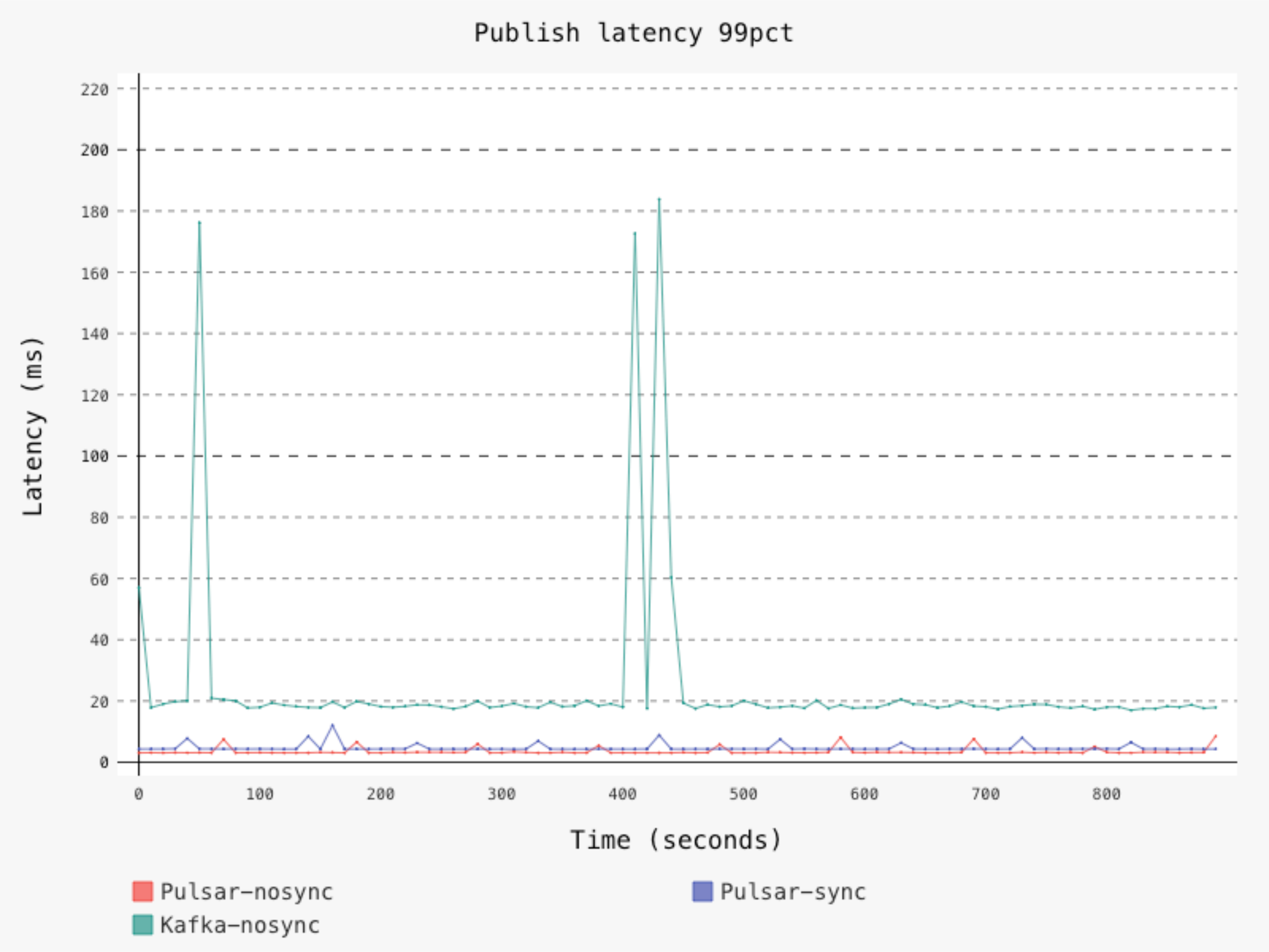


streamlio

PULSAR PERFORMANCE



PULSAR PERFORMANCE - LATENCY



streamlio

APACHE PULSAR vs. APACHE KAFKA

Durability
Data replicated and synced to disk

Multi-tenancy
A single cluster can support many tenants and use cases

Tiered Storage
Hot/warm data for real time access and cold event data in cheaper storage

Geo-replication
Out of box support for geographically distributed applications

Seamless Cluster Expansion
Expand the cluster without any down time

Pulsar Functions
Flexible light weight compute

Unified messaging model
Support both Topic & Queue semantic in a single model

High throughput & Low Latency
Can reach 1.8 M messages/s in a single partition and publish latency of 5ms at 99pct

Highly scalable
Can support millions of topics, makes data modeling easier

Examples of companies using Apache Pulsar

Streamlio outreach

Growing funnel of validation and leads from outbound, inbound and open source

Open source adopters

Open source evaluators

streamlio

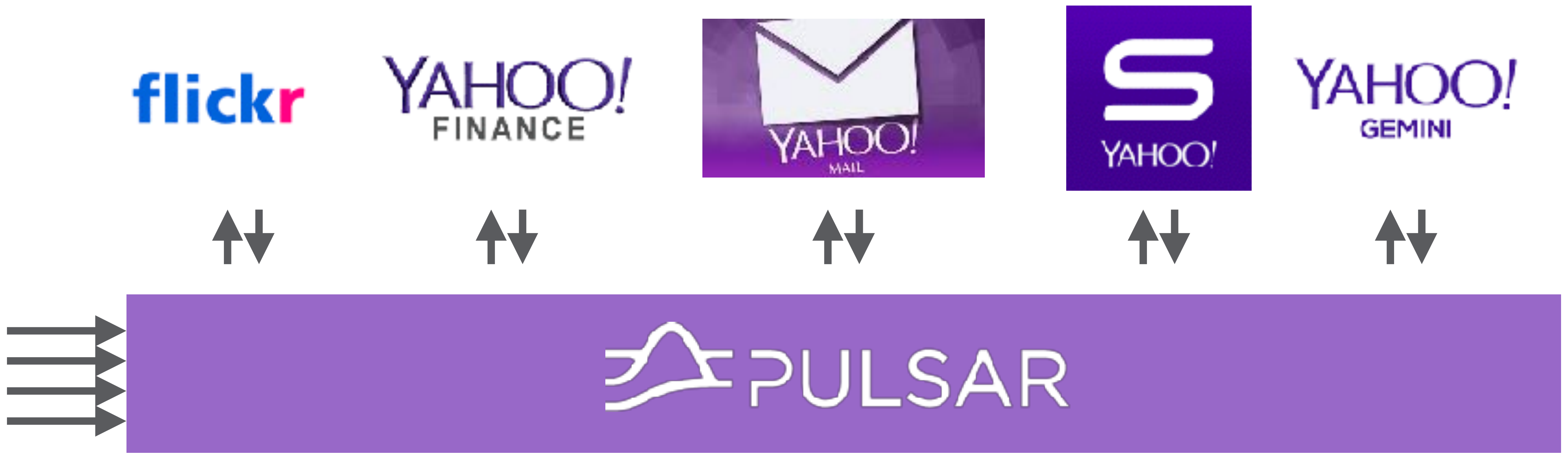
Yahoo!

Scenario

Need to collect and distribute user and data events to distributed global applications at Internet scale

Challenges

- Multiple technologies to handle messaging needs
- Multiple, siloed messaging clusters
- Hard to meet scale and performance
- Complex, fragile environment



Solution

- Central event data bus using Apache Pulsar
- Consolidated multiple technologies and clusters into a single solution
- Fully-replicated across 8 global datacenter
- Processing >100B messages / day, 2.3M topics

streamlio

APACHE PULSAR IN PRODUCTION @SCALE

- 4+ years
- Serves 2.3 million topics
- 700 billion messages/day
- 500+ bookie nodes
- 200+ broker nodes
- Average latency < 5 ms
- 99.9% 15 ms (strong durability guarantees)
- Zero data loss
- 150+ applications
- Self served provisioning
- Full-mesh cross-datacenter replication - 8+ data centers

Growing ecosystem

AEROSPIKE
JDBC
Netty
ORACLE
Microsoft SQL Server
MySQL
cassandra
kafka
PostgreSQL
mongoDB
elasticsearch
Spark Streaming
STORM
amazon KINESIS
StreamSets
hadoop HDFS
Amazon S3
RabbitMQ
debezium
Google Cloud Storage
redis
Microsoft Azure Blob Storage
Solr
influxdb

Use Cases

Example use cases

Real-time monitoring and notifications

Interactive applications

Log processing and analytics

IoT analytics

Streaming data transformation

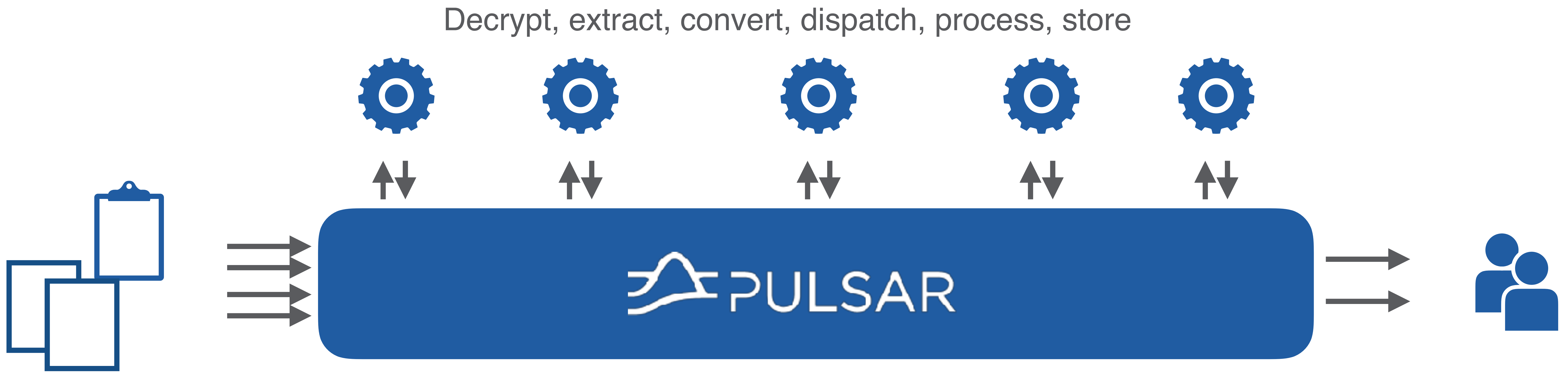
Real-time analytics

Data distribution

Event-driven workflows

streamlio

Data-driven workflows



Scenario

Application processes incoming events and documents that generate processing workflows

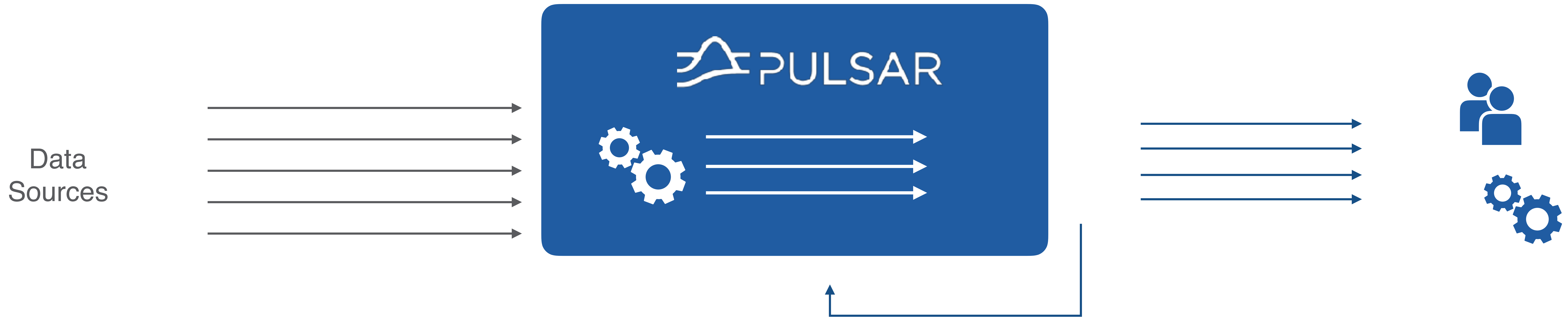
Challenges

Operational burdens and scalability challenges of existing technologies growing as data grows

Solution

Process incoming events and data and create work queues in same system

Data distribution



Data collected from multiple sources

Normalized, enriched transformed and put into topics

Delivered to applications and users as data streams

Distribution and usage logged for auditing

streamlio

Simplifying the data pipeline

Scenario

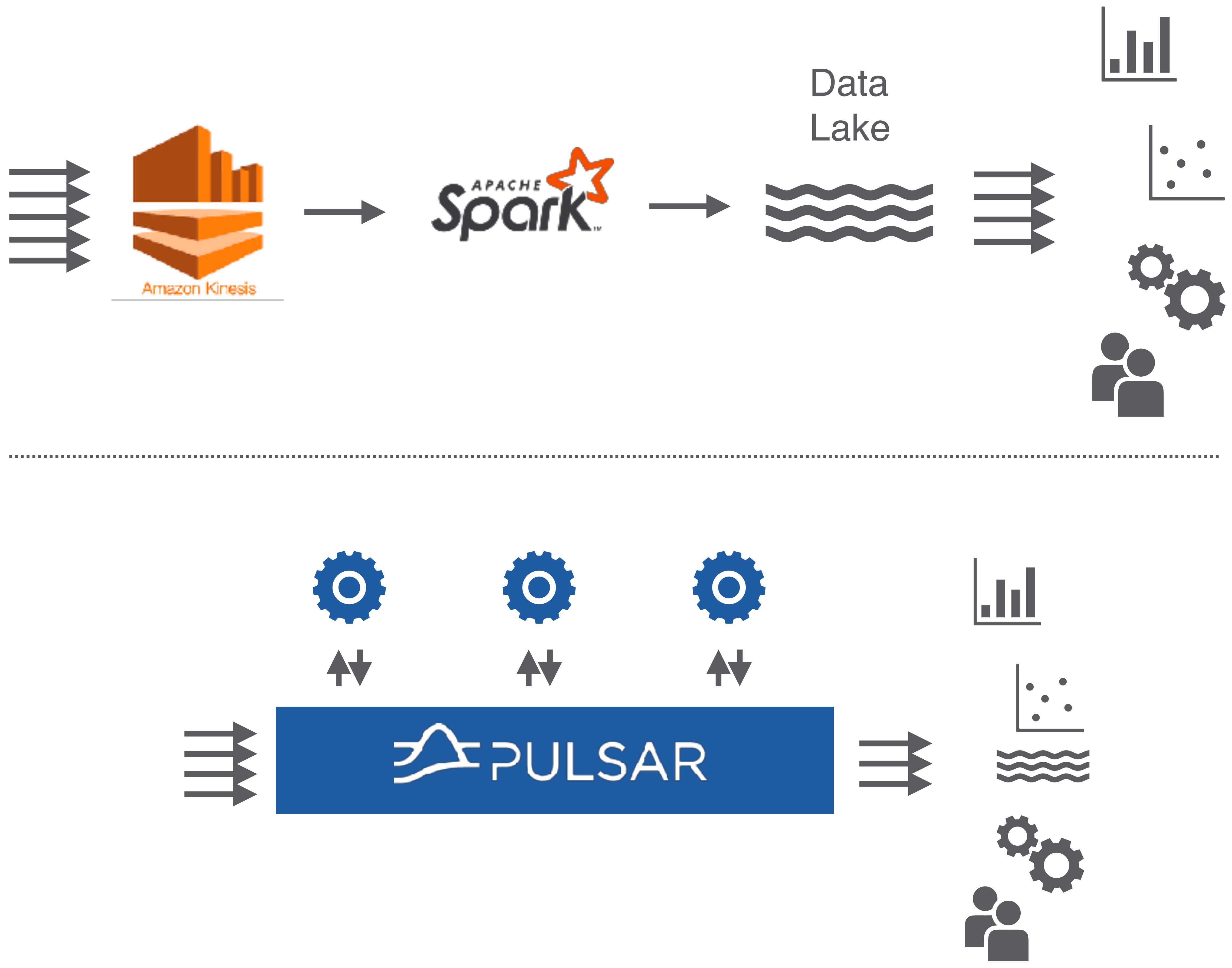
Retail analytics software provider brings together operational and market research data for insights.

Challenges

Existing Kinesis + Spark + data lake infrastructure was unnecessarily complex and burdensome to operate and maintain.

Solution

- Replaced Kinesis + Spark with Apache Pulsar
- Simplified data transformation pipeline
- Reduced operations burdens



streamlio

Event sourcing

Problem

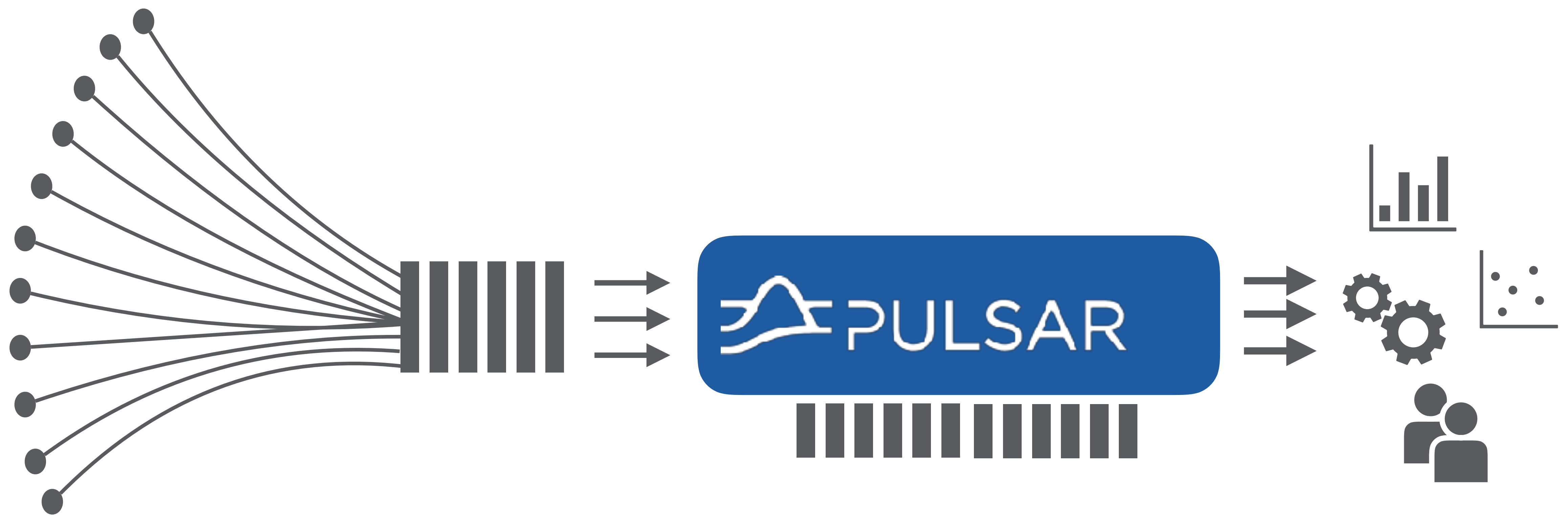
Event-driven applications require long-term retention of data streams, but current technologies are cumbersome and expensive to use for data retention and cannot efficiently replay data.

Solution

Deploy Apache Pulsar for long-term retention and scalable processing and distribution of event data.

Why Streamlio

- Architected for scalable and efficient long-term storage
- High performance, scalable processing and distribution of data due to unique architecture

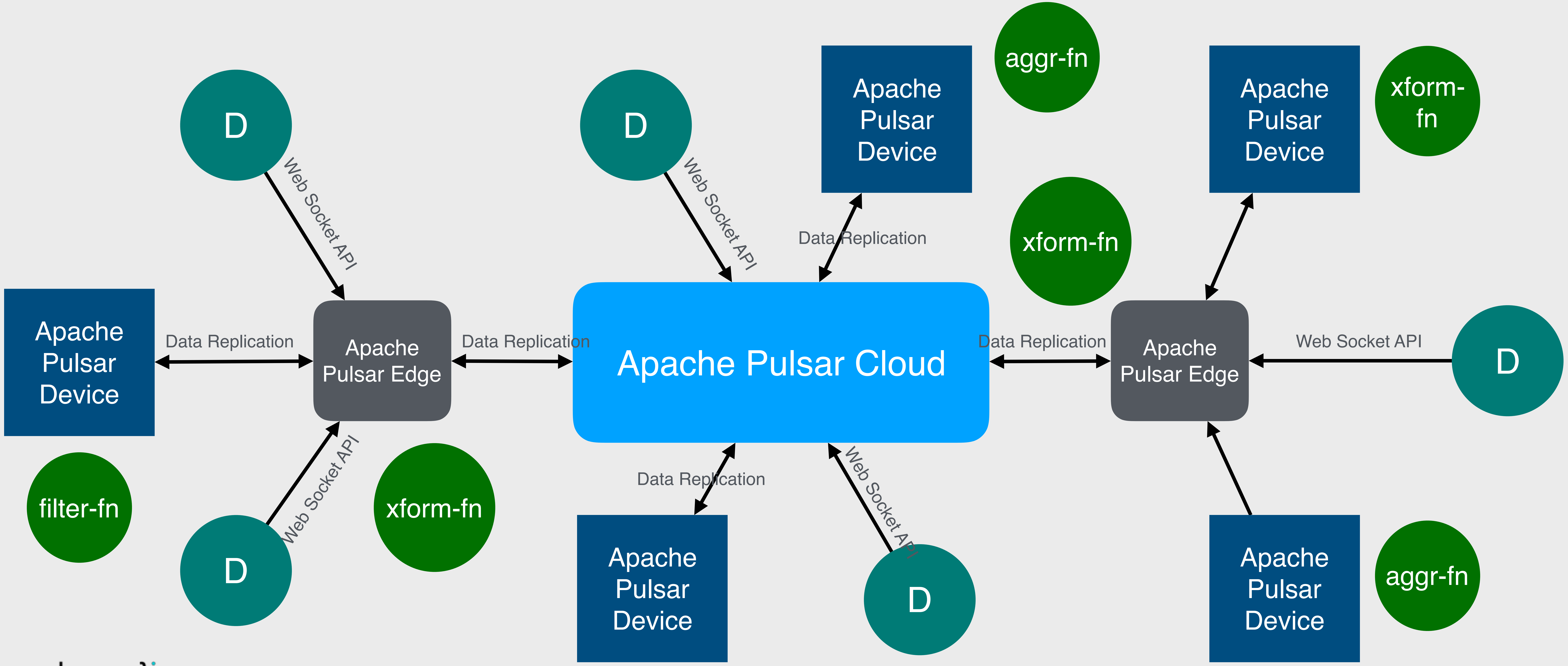


IOT ENVIRONMENT

Light Device	Smart Device	Edge Node	Cloud
D	Smart D	Edge Aggregator	Cloud
✦ Typically sensors	✦ Typically ARM based	✦ Multicore based	✦ Multiple machines
✦ Only one functionality	✦ Multiple functionality	✦ Versatile functionality	✦ Versatile functionality
✦ Simple to configure	✦ Basic but generic computational logic, limited storage	✦ Complex and generic computational logic, decent amount of storage	✦ Complex and generic computational logic
✦ Light weight protocols to communicate	✦ Light weight and propriety protocols to communicate	✦ Light weight and propriety protocols to communicate	✦ Lots of storage

streamlio

IOT DATA FABRIC WITH APACHE PULSAR



streamlio

Large Car Manufacturer: Connected vehicle

Scenario

Continuously-arriving data generated by connected cars needs to be quickly collected, processed and distributed to applications and partners

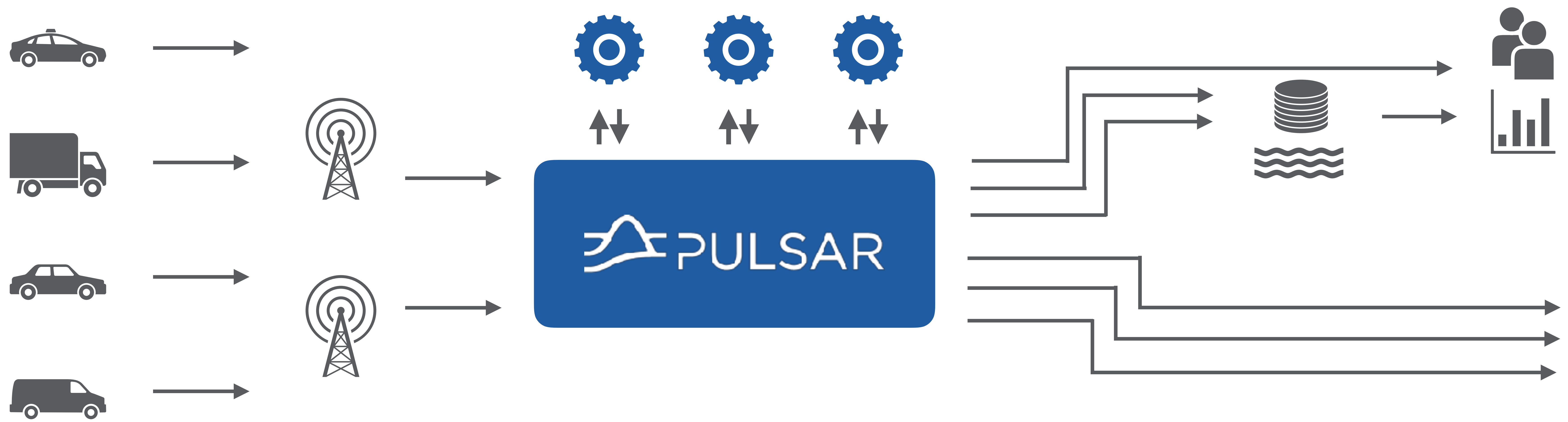
Challenges

Require scalability to handle growing data sources and volumes without complex mix of technologies

Solution

Leverage Streamlio solution to provide data backbone that can receive, transform, and distribute data at scale

Large Car Manufacturer: Connected vehicle



Data made available to internal teams for analysis and reports

Data feeds supplied to partners and partner applications

Telemetry data from connected vehicles transmitted and published to Pulsar

Data cleansing, enrichment and refinement processed inside Pulsar

Large Car Manufacturer: Big Data Logging System

Scenario

Continuously ingest logs from big data system for distributed to appropriate teams with appropriate log transformations and enrichment

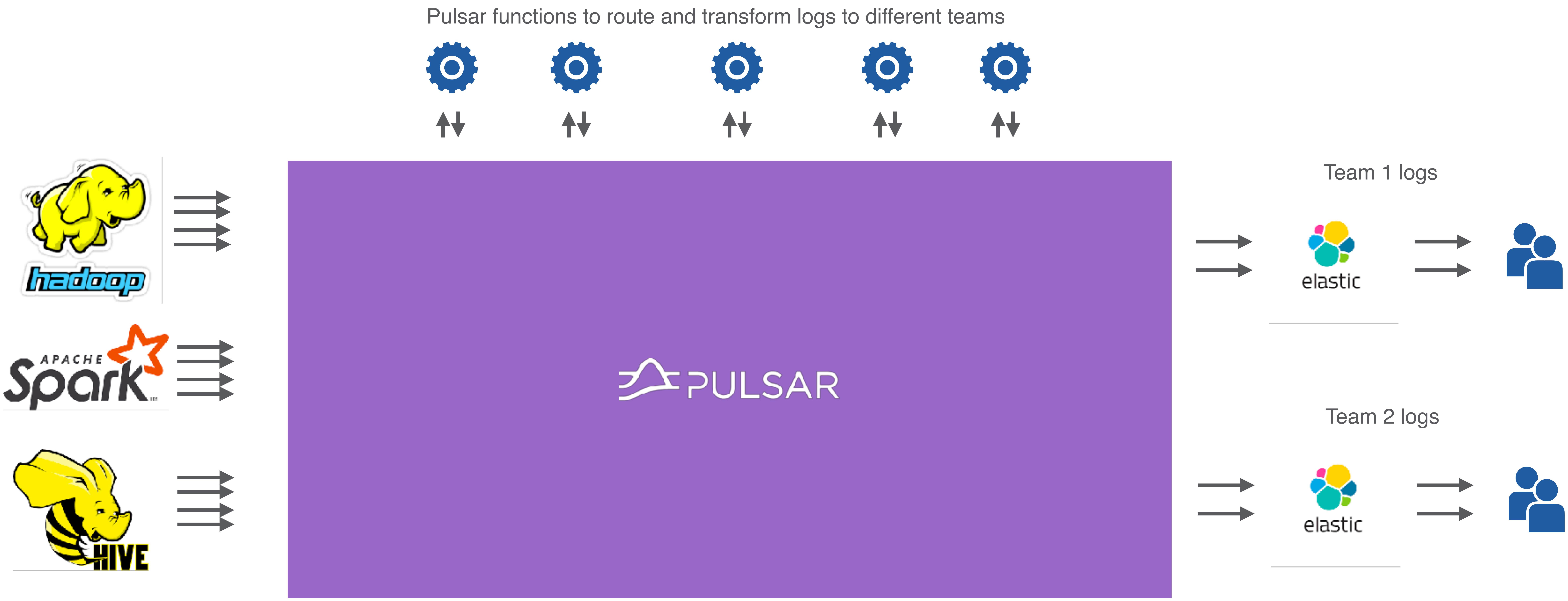
Challenges

Require scalability to handle growing set of big data systems and larger log volumes

Solution

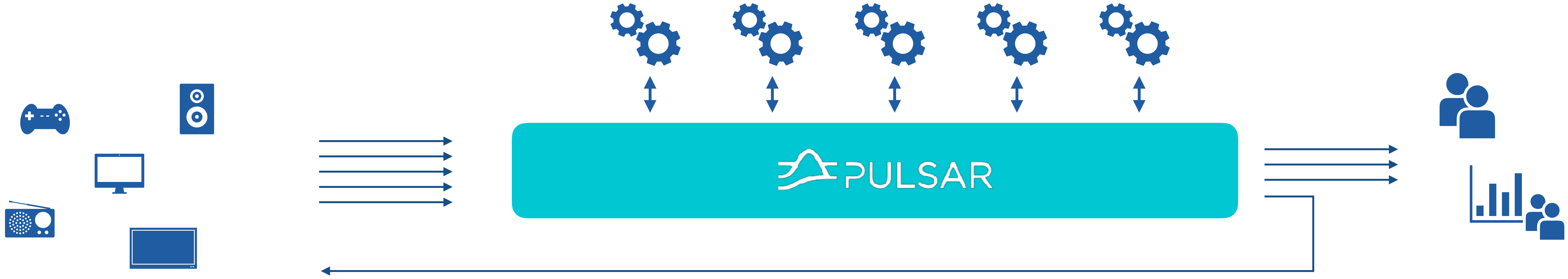
Leverage Streamlio Pulsar solution to provide logging backbone that can ingest, transform, and distribute logs at scale

Large Car Manufacturer: Big Data Logging System



streamlio

Connected consumer



Connected consumer electronic devices

Emit event data that is collected and processed in Pulsar

Generating notifications and work requests

Distributed to microservices for processing

Supporting connected services and applications

streamlio

MORE READINGS

✓ **Understanding How Pulsar Works**

https://jack-vanlightly.com/blog/2018/10/2/understanding-how-apache-pulsar-works

✓ **How To (Not) Lose Messages on Apache Pulsar Cluster**

https://jack-vanlightly.com/blog/2018/10/21/how-to-not-lose-messages-on-an-apache-pulsar-cluster

streamlio

MORE READINGS

- ✓ **Unified queuing and streaming**

  https://streaml.io/blog/pulsar-streaming-queuing

- ✓ **Segment centric storage**

  https://streaml.io/blog/pulsar-segment-based-architecture

- ✓ **Messaging, Storage or Both**

  https://streaml.io/blog/messaging-storage-or-both

- ✓ **Access patterns and tiered storage**

  https://streaml.io/blog/access-patterns-and-tiered-storage-in-apache-pulsar

- ✓ **Tiered Storage in Apache Pulsar**

  https://streaml.io/blog/tiered-storage-in-apache-pulsar

streamlio

QUESTIONS

?

?

?

STAY IN TOUCH

@

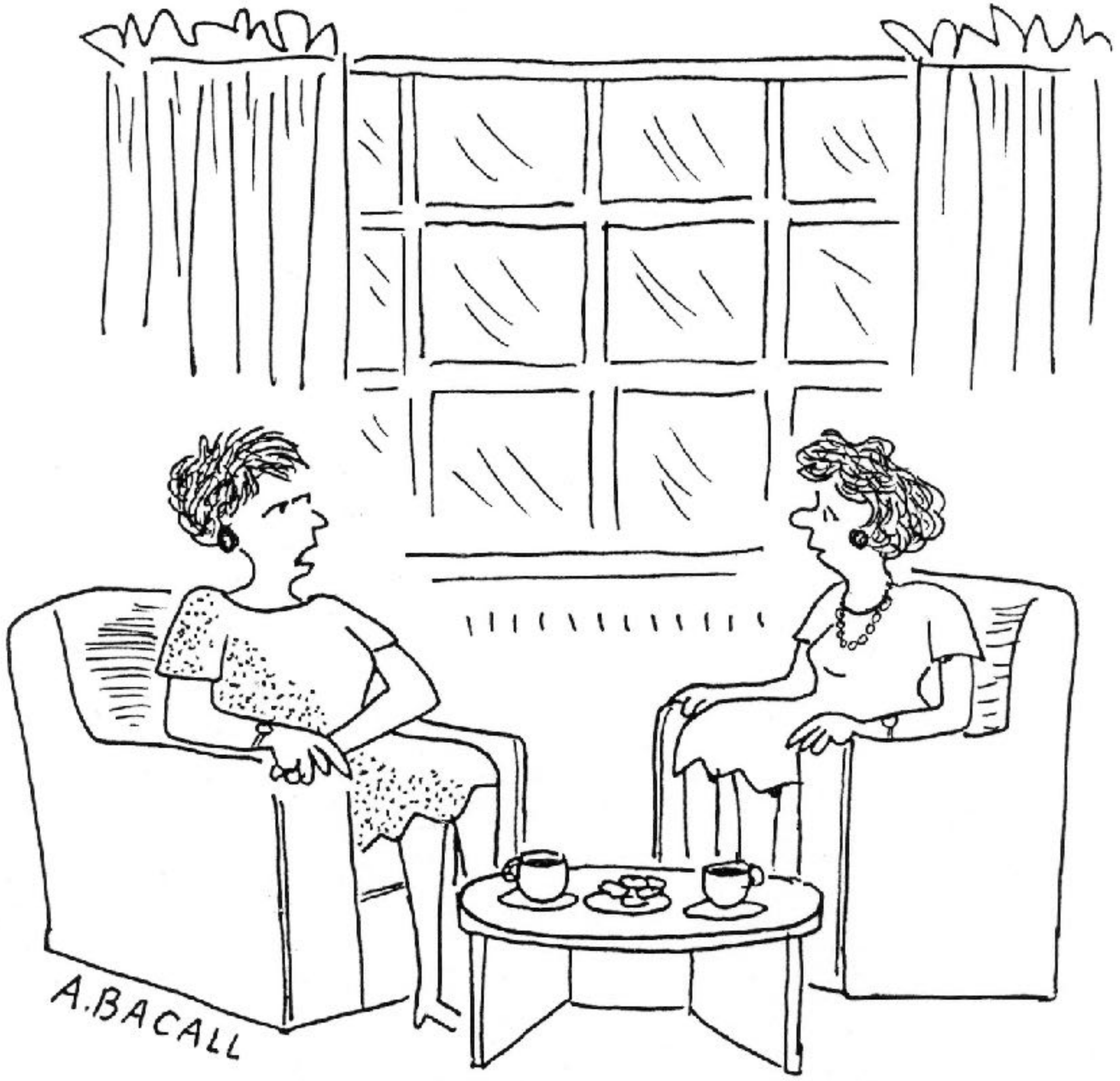
EMAIL

karthik@streaml.io

TWITTER

@karthikz

streamlio



" My son is a corporate communications director.
He never calls and he never writes."

@karthikz