Eötvös Loránd University (ELTE)
Faculty of Informatics (IK)
Pázmány Péter sétány 1/c
1117 Budapest, Hungary

Streaming systems: Kafka

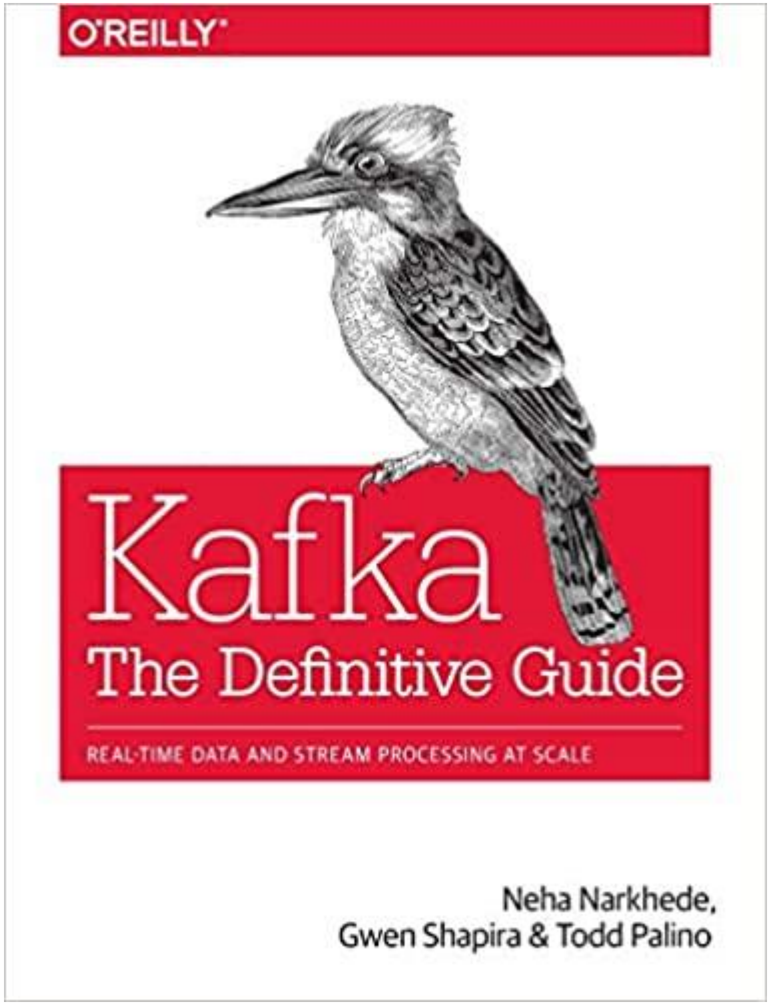
Open-source Technologies for Real-Time Data Analytics

Imre Lendák, PhD, Associate Professor

2020
Budapest, Hungary

Overview

- Introduction
- Motivation
- Architecture
- Processes
    - Brokers
    - Producers
    - Consumers
- Replication & consistency
- Monitoring & control



O'REILLY

Kafka

The Definitive Guide

REAL-TIME DATA AND STREAM PROCESSING AT SCALE

Neha Narkhede,
Gwen Shapira & Todd Palino

Definition & origins

Apache Kafka

A high-throughput distributed messaging system.

- **DEF:** Apache **Kafka** is an open-source data stream processing platform
- **Originally developed by:** LinkedIn
- **Initial release:** Jan 2011
- **Current release:** 2.6.0 in August 2020
- **Written in:** Scala & Java
- **License:** Apache License 2.0
- **Author(s):** 9 core committers, plus ~ 20 contributors
- **Website:** http://kafka.apache.org/

Key features

Guarantees

- Data integrity checks
- At least once delivery
- In order delivery, per partition

Characteristics

- Very high performance
- Elastically scalable
- Low operational overhead
- Durable, highly available

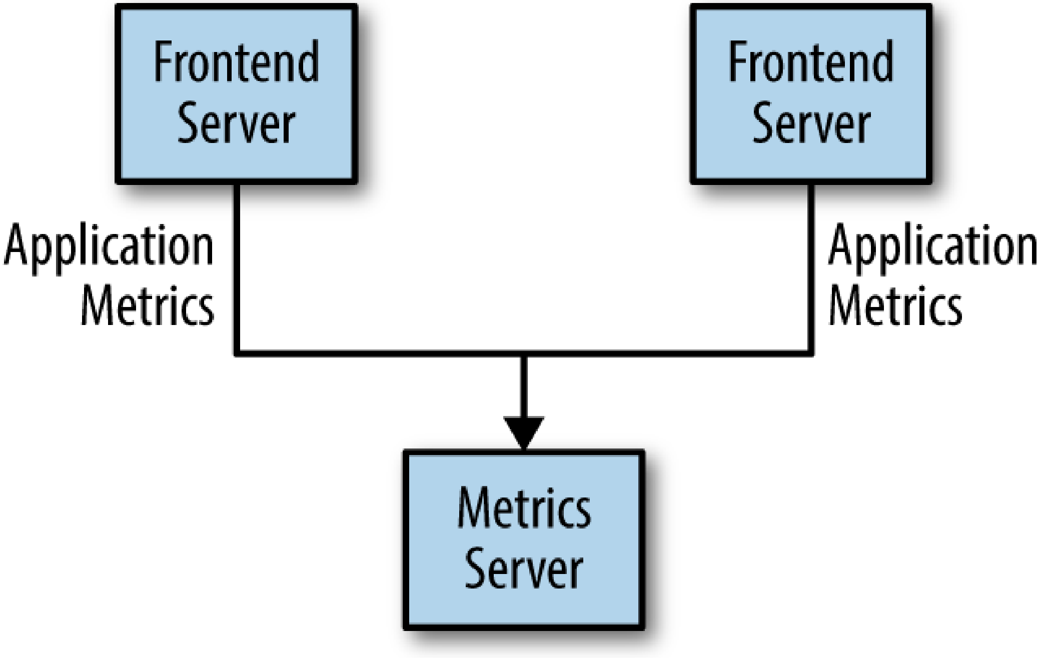
Motivation

Motivation

- LinkedIn’s motivation for Kafka was:
  - “A unified platform for handling all the real-time data feeds a large company might have.”

- Features
  - High throughput to support **high volume event feeds**.
  - Support real-time processing of these feeds to create **new, derived feeds**.
  - Support large data backlogs to handle periodic ingestion from **offline systems**.
  - Guarantee **fault-tolerance** in the presence of machine failures.

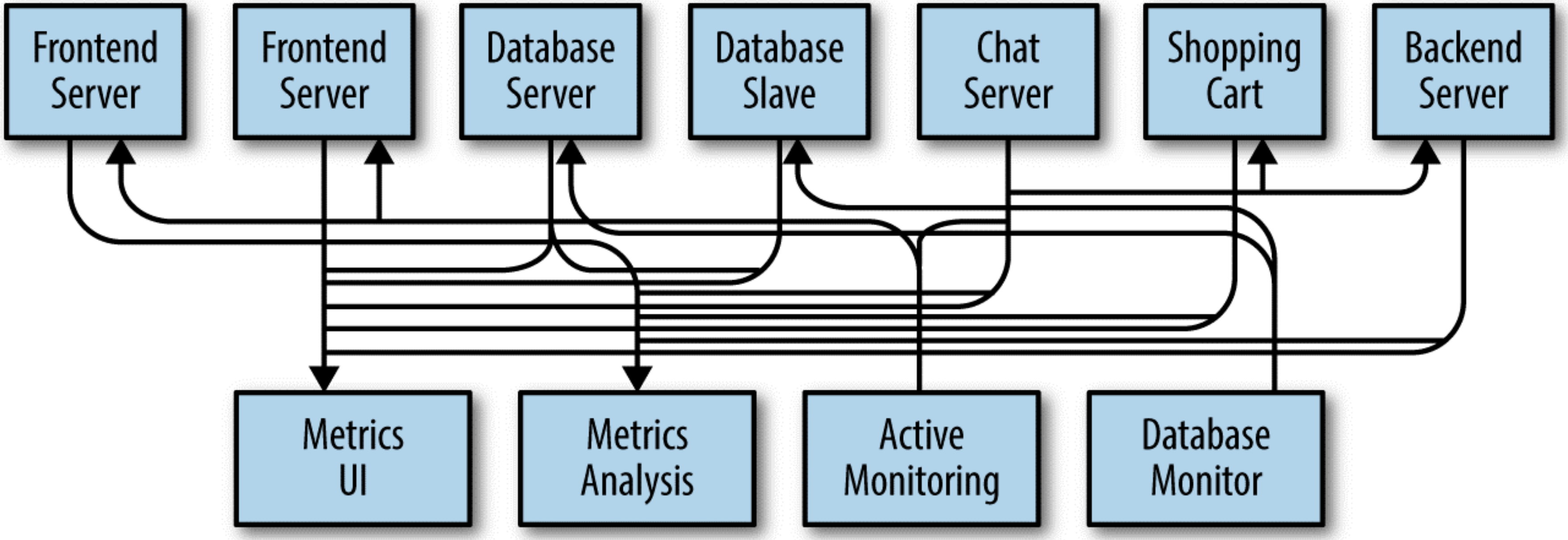
Scalability challenges – 1



- **Challenge:** What if the number of producers (i.e. server) and consumers (not shown here) increases?

Narkhede, N., Shapira, G., & Palino, T. (2017). *Kafka: the definitive guide: real-time data and stream processing at scale*. " O'Reilly Media, Inc.".

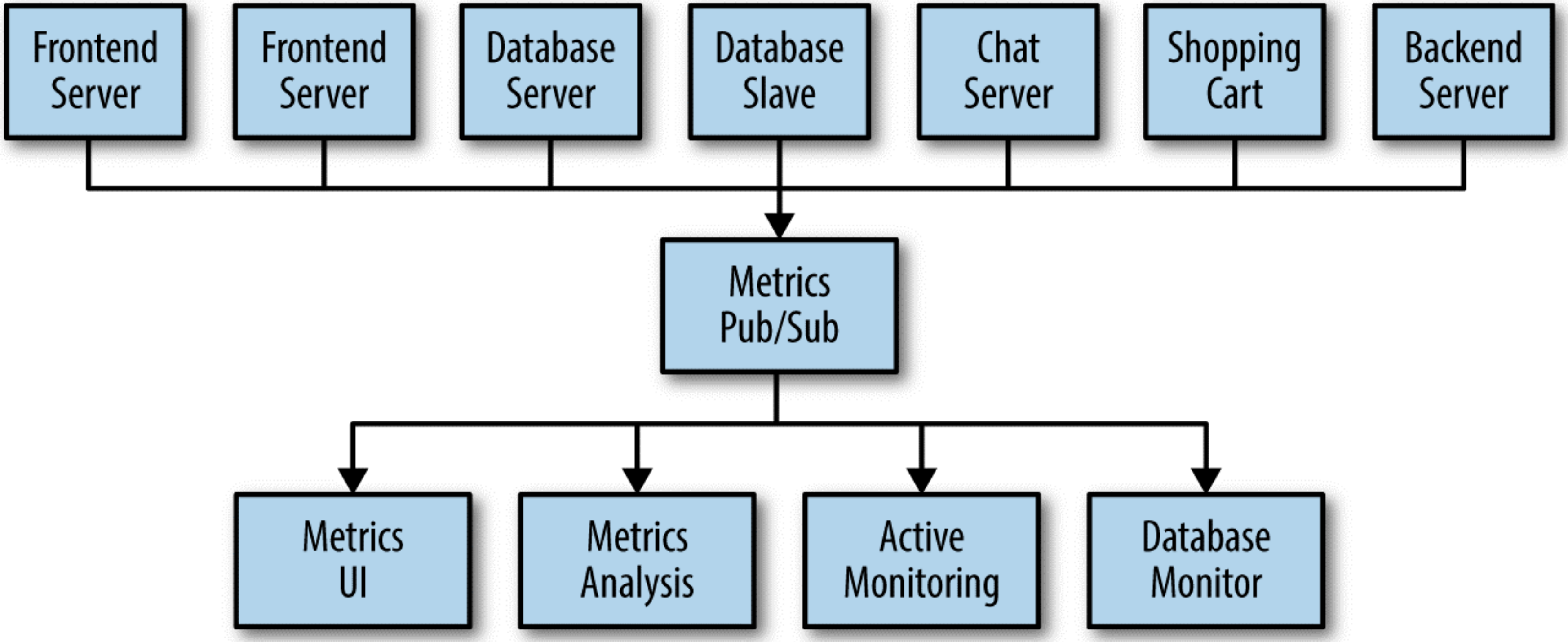
Scalability challenges – 2



- **Challenge:** How to handle the large number of interconnections between the different sources & sinks?

Narkhede, N., Shapira, G., & Palino, T. (2017). *Kafka: the definitive guide: real-time data and stream processing at scale*. " O'Reilly Media, Inc.".

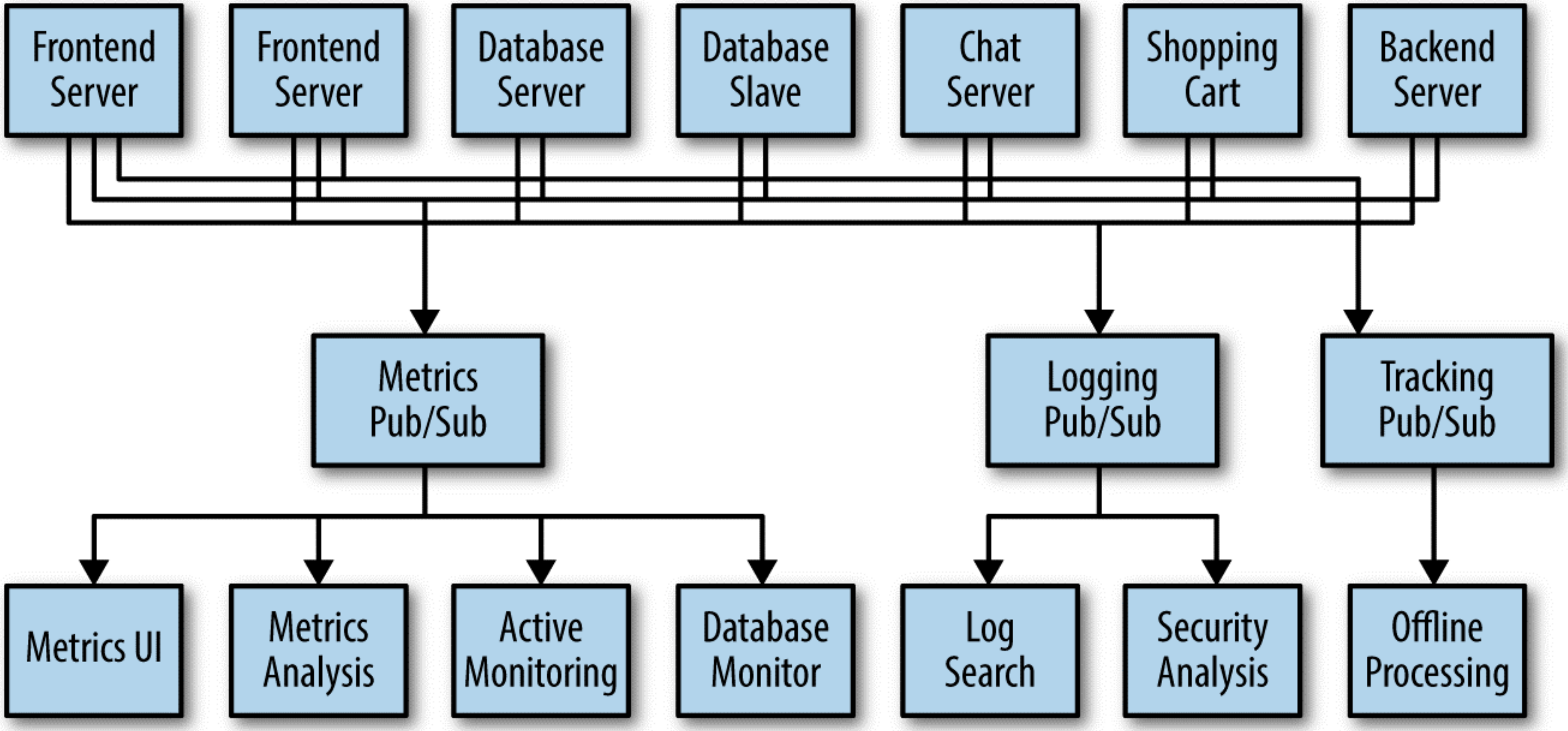
Scalability challenges – 3



- **Challenge:** Metrics data management solved! But what if there are other data types in a large enterprise?

Narkhede, N., Shapira, G., & Palino, T. (2017). *Kafka: the definitive guide: real-time data and stream processing at scale*. " O'Reilly Media, Inc.".

Scalability challenges – 4

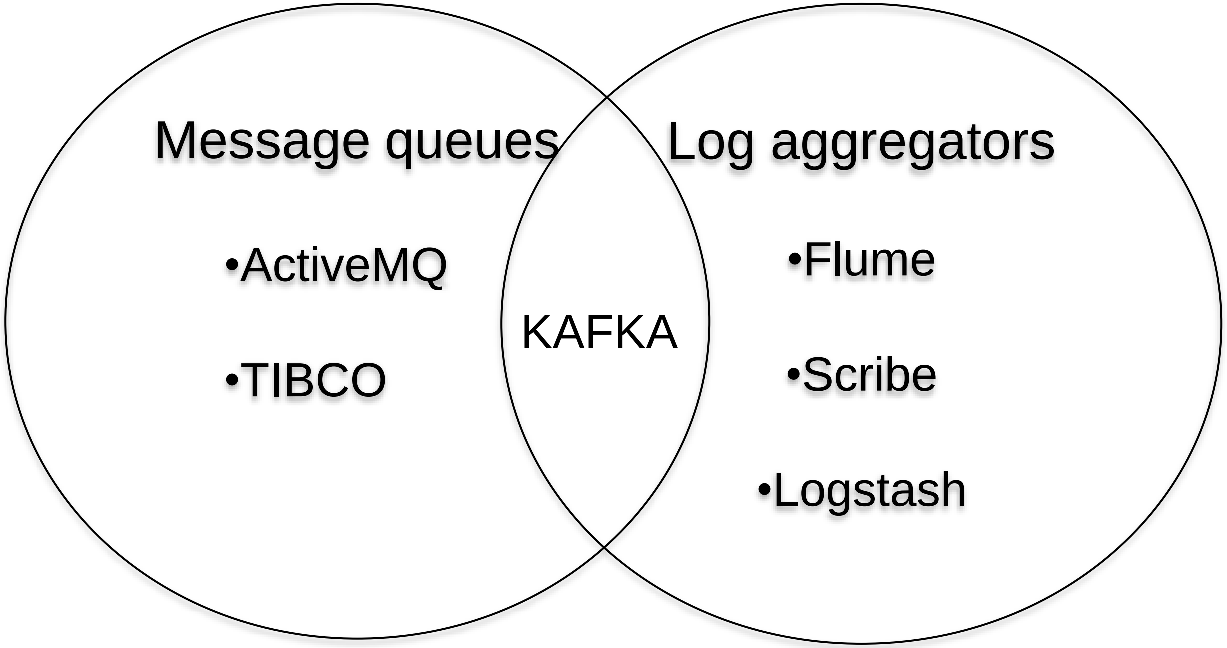


- **Challenge:** How to handle the diverse data types (in the nodes in the middle?

Narkhede, N., Shapira, G., & Palino, T. (2017). *Kafka: the definitive guide: real-time data and stream processing at scale*. " O'Reilly Media, Inc.".

ARCHITECTURE

Queues and aggregators



Message queues

- ActiveMQ
- TIBCO

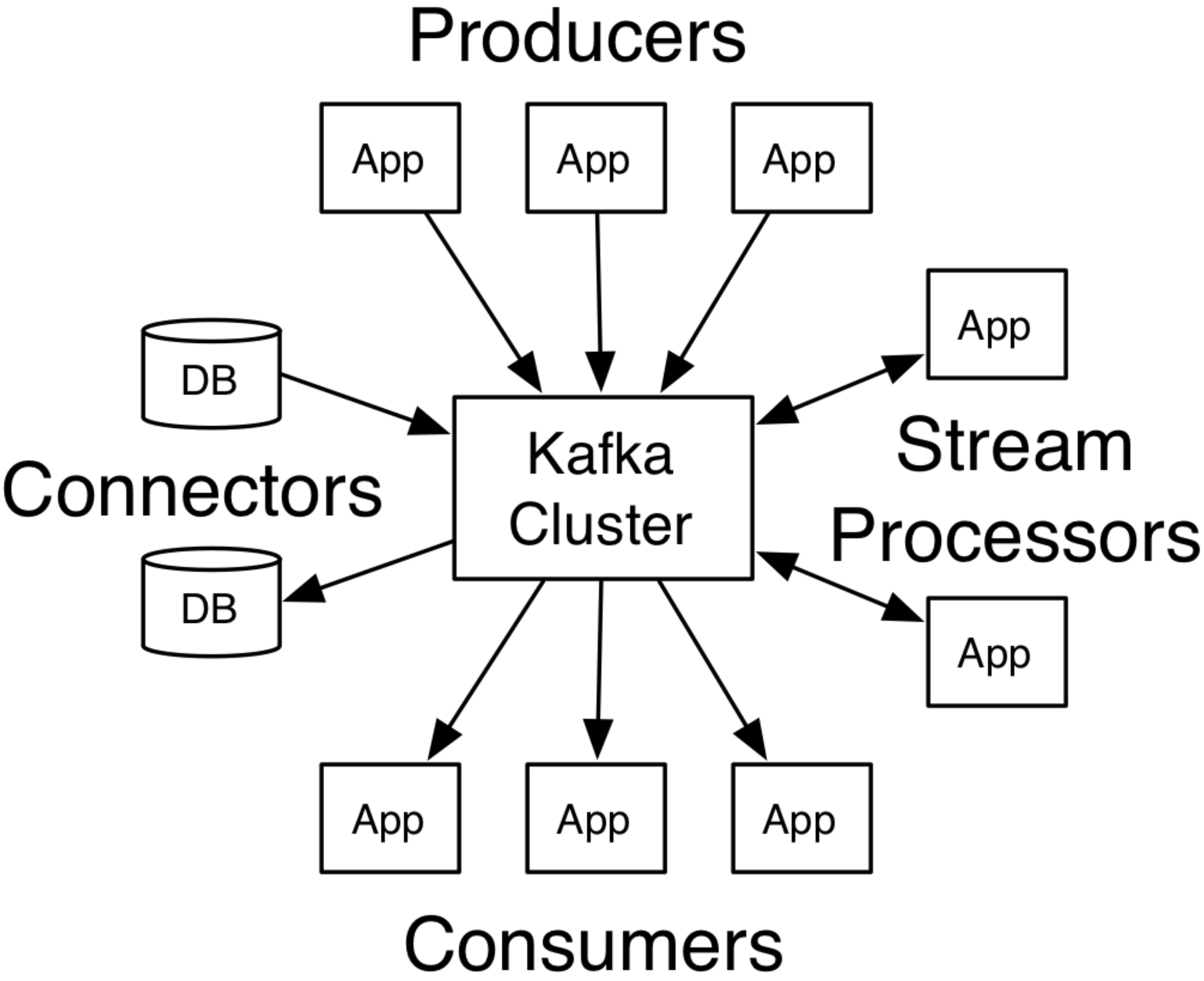
KAFKA

Log aggregators

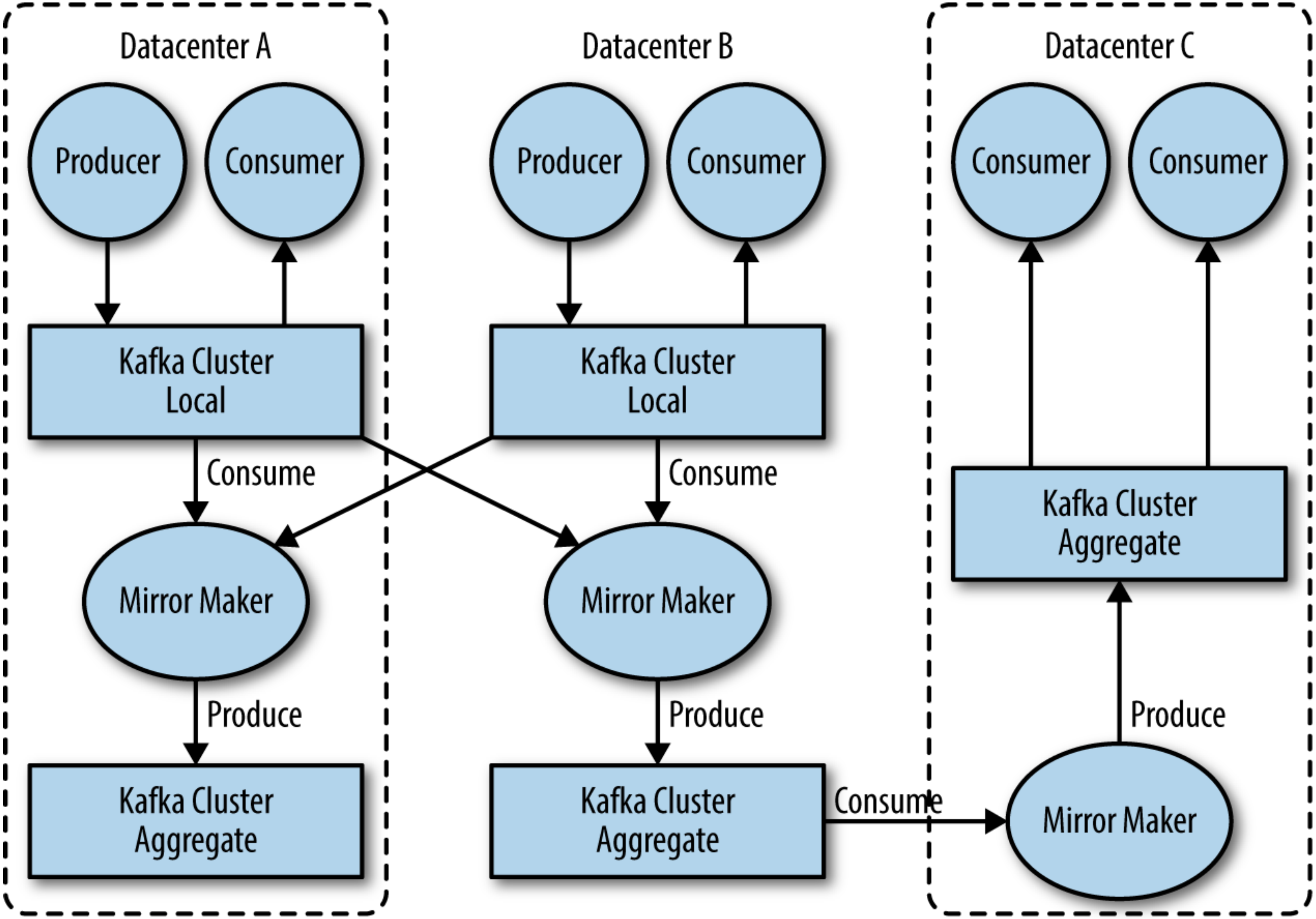
- Flume
- Scribe
- Logstash

- Kafka has a publish-subscribe (often abbreviated as pubsub) architecture → a distributed mix of message queues and log aggregators

Architecture overview

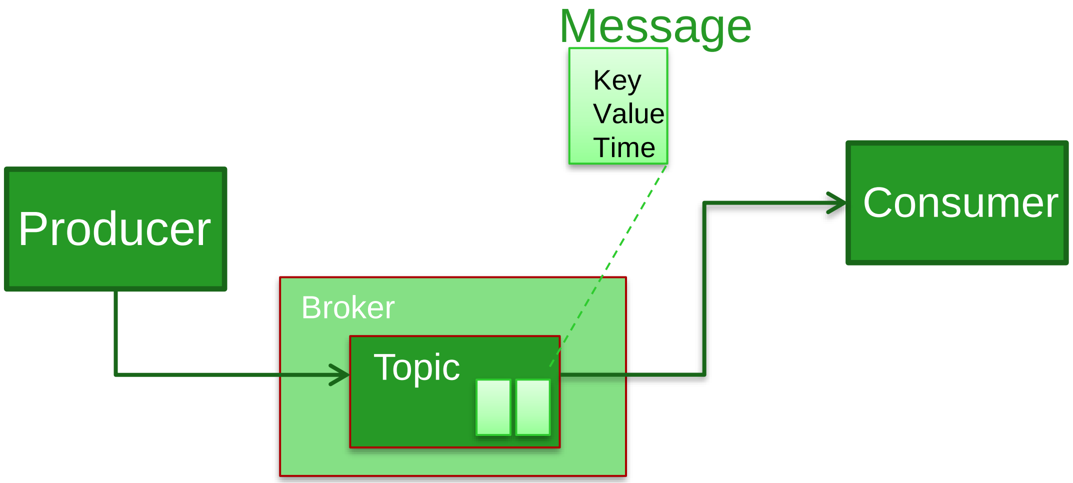


Multi-datacenter architecture



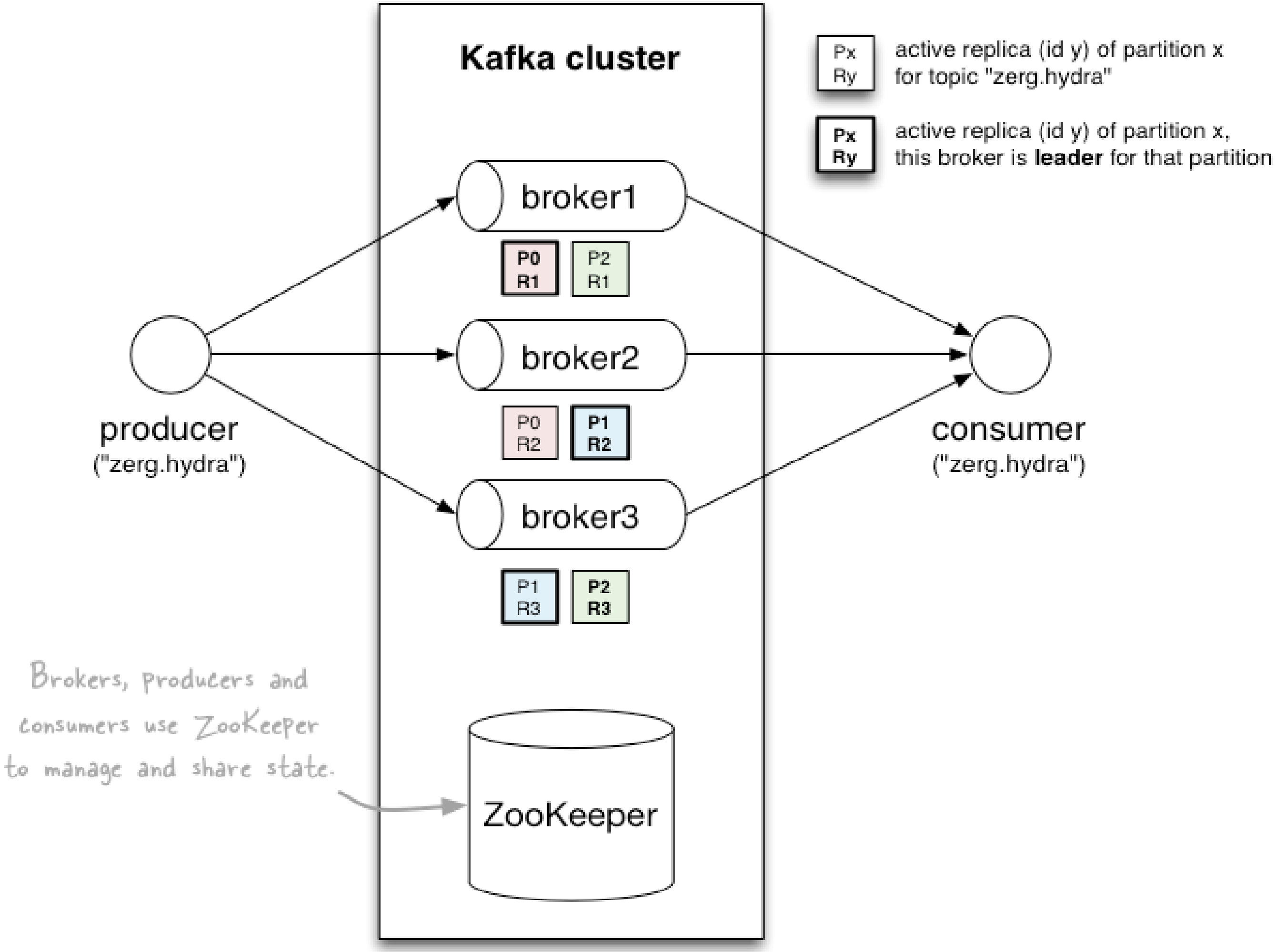
Who is who? (Terminology)

- Topic
  - partition
- Message
  - == ByteArray
- Broker
  - replicated
- Producer
- Consumer
  - Working together in Consumer Groups



kafka

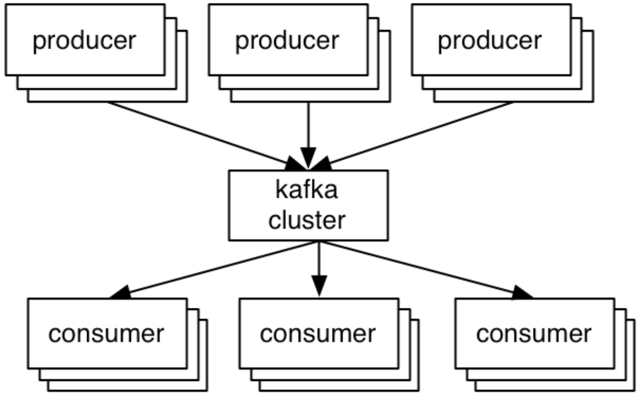
Architecture



Key processes

- The who is who
  - **Producers** write data to **brokers**.
  - **Consumers** read data from **brokers**.

- The data
  - Data is stored in **topics**.
  - **Topics** are split into **partitions**, which are replicated.

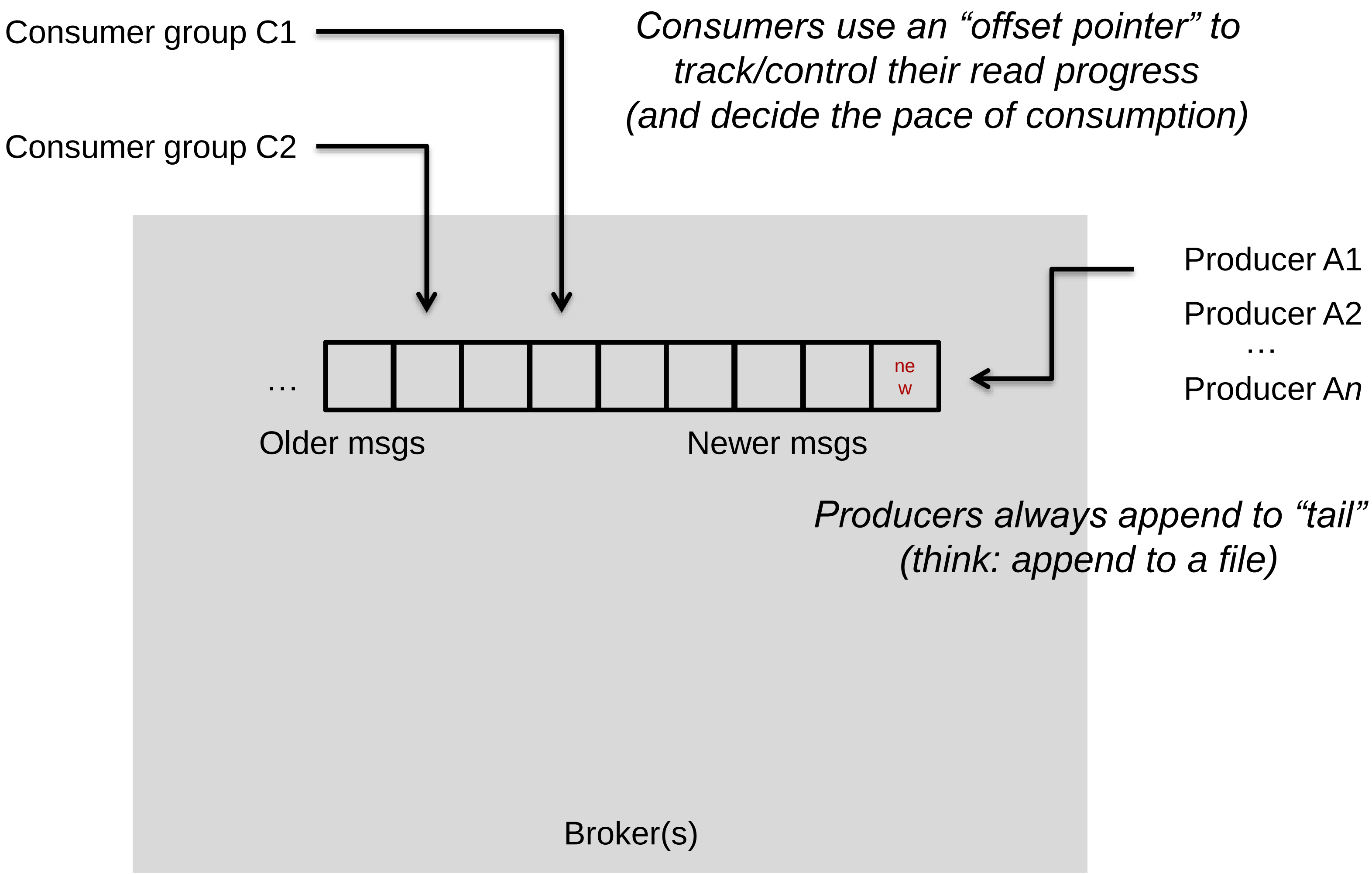


Processes: Brokers

Brokers

- **DEF:** Kafka brokers are the processes tasked to receive messages from producers, consistently store them and respond to requests from Kafka consumers
- A Kafka cluster consists of one or more brokers
- Brokers are usually executed different servers
- One broker can maintain multiple partitions of different Kafka topics
- The brokers maintain special, non-producer-defined topics for administrative purposes, e.g. topics for memorizing message offsets for consumers

Topics



Consumer group C1

Consumer group C2

Consumers use an "offset pointer" to track/control their read progress (and decide the pace of consumption)

…

ne
w

Older msgs

Newer msgs

Producer A1

Producer A2

…

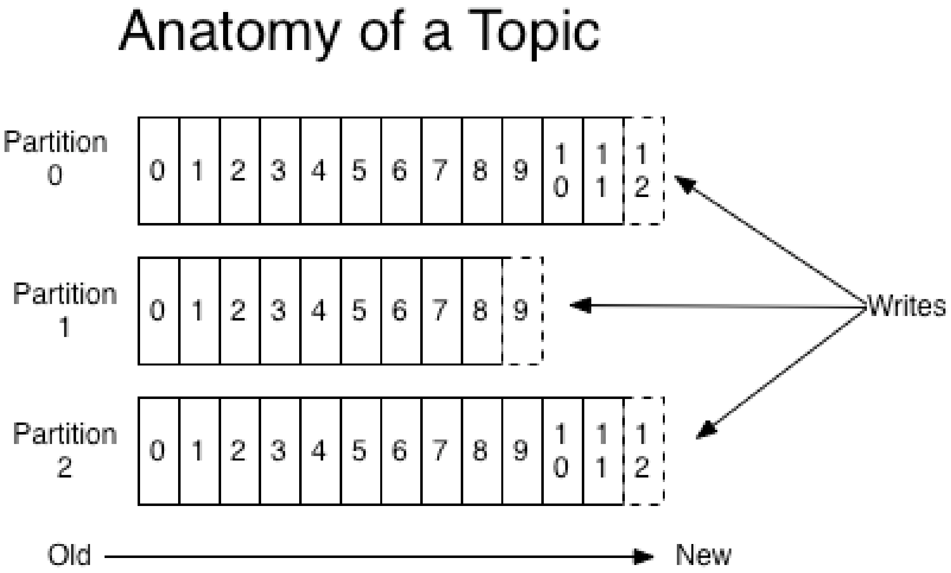
Producer A*n*

Producers always append to "tail" (think: append to a file)

Broker(s)

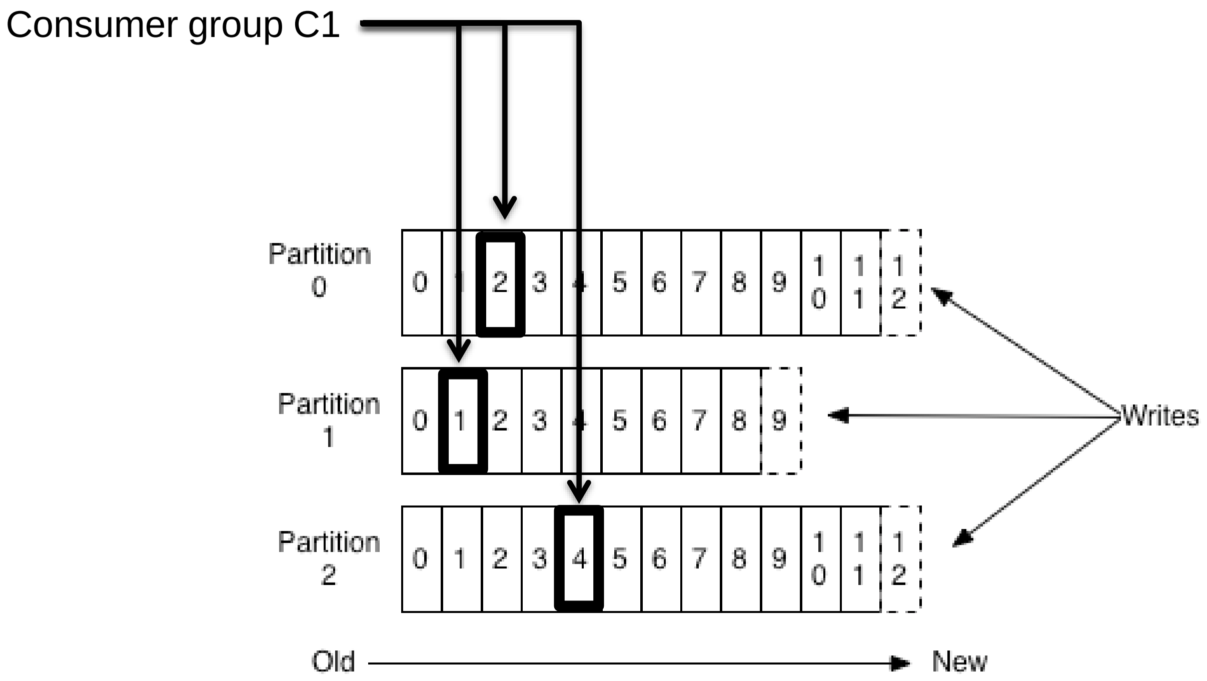
Partitions

- A topic consists of **partitions.**
- Partition: **ordered + immutable** sequence of messages that is continually appended to



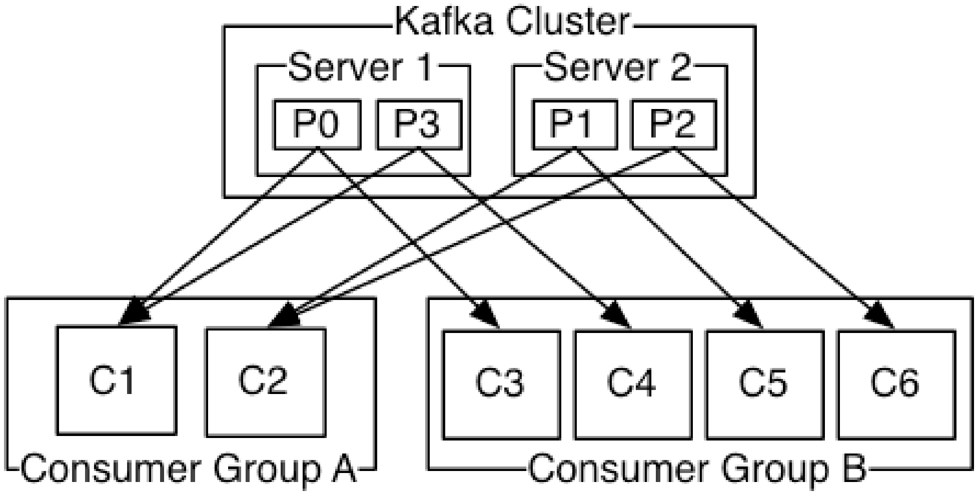
Partition offsets

- **Offset**: messages in the partitions are each assigned a unique (per partition) and sequential id called the *offset*
  - Consumers track their pointers via *(offset, partition, topic)* tuples



Partitions

- #Partitions of a topic is configurable
- #Partitions determines **max** consumer (group) parallelism



- Consumer group A, with 2 consumers, reads from a 4-partition topic
- Consumer group B, with 4 consumers, reads from the same topic

Broker performance

Efficiency

- Each topic has an ever-growing log
  - A log == a list of files
- A message is addressed by a log offset
- Batch send and receive
- No message caching in JVM
- Rely on file system buffering
- 1 file system operation per request

Implementation

- Fast **writes**:
  - While Kafka persists all data to disk, essentially all writes go to the **page cache** of OS, i.e. RAM.
- Fast **reads**:
  - Very efficient to transfer data from page cache to a network **socket**
  - Linux: **`sendfile()`** system call
- Combination of the above two features → highly efficient Kafka

Processes: Producers

Producer intro

- Producer **use cases**:
  - Record user activities for auditing and analysis
  - Record infrastructure metrics, e.g. CPU load, RAM use, bandwidth utilization
  - Store logs
  - Record information from smart devices, e.g. in an electric power system setting
  - Buffer information before writing to a database
- Producers create **producer record** objects which consist of (topic, partition, key, value) tuples
  - The partition and key values are optional
  - When defined, the partition defines the destination partition → if it undefined, the partitioner assigns the record
- Producers rely on different serializers to convert records, e.g. Apache Avro, Java serialization, custom serialization

Producer workflow

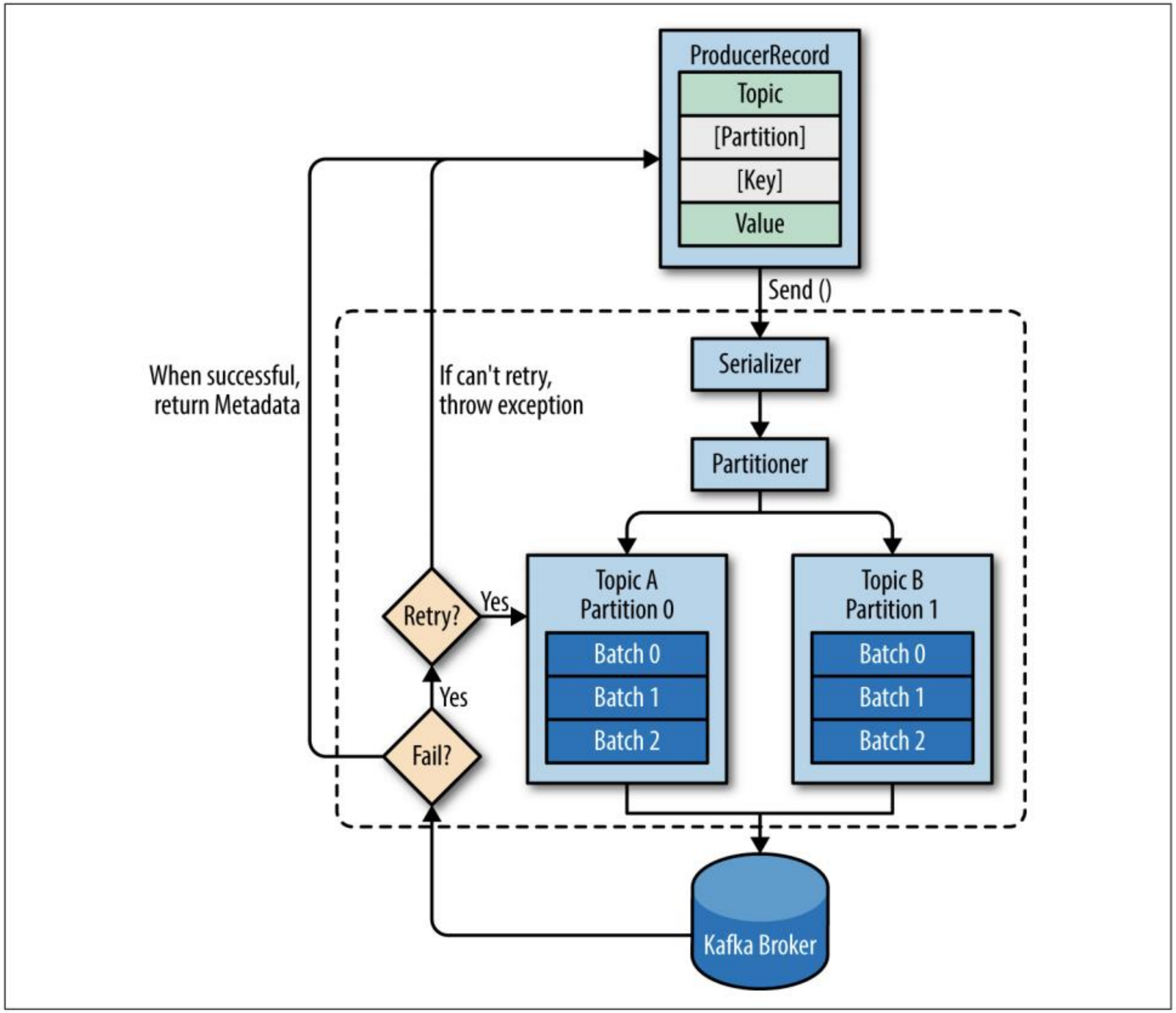


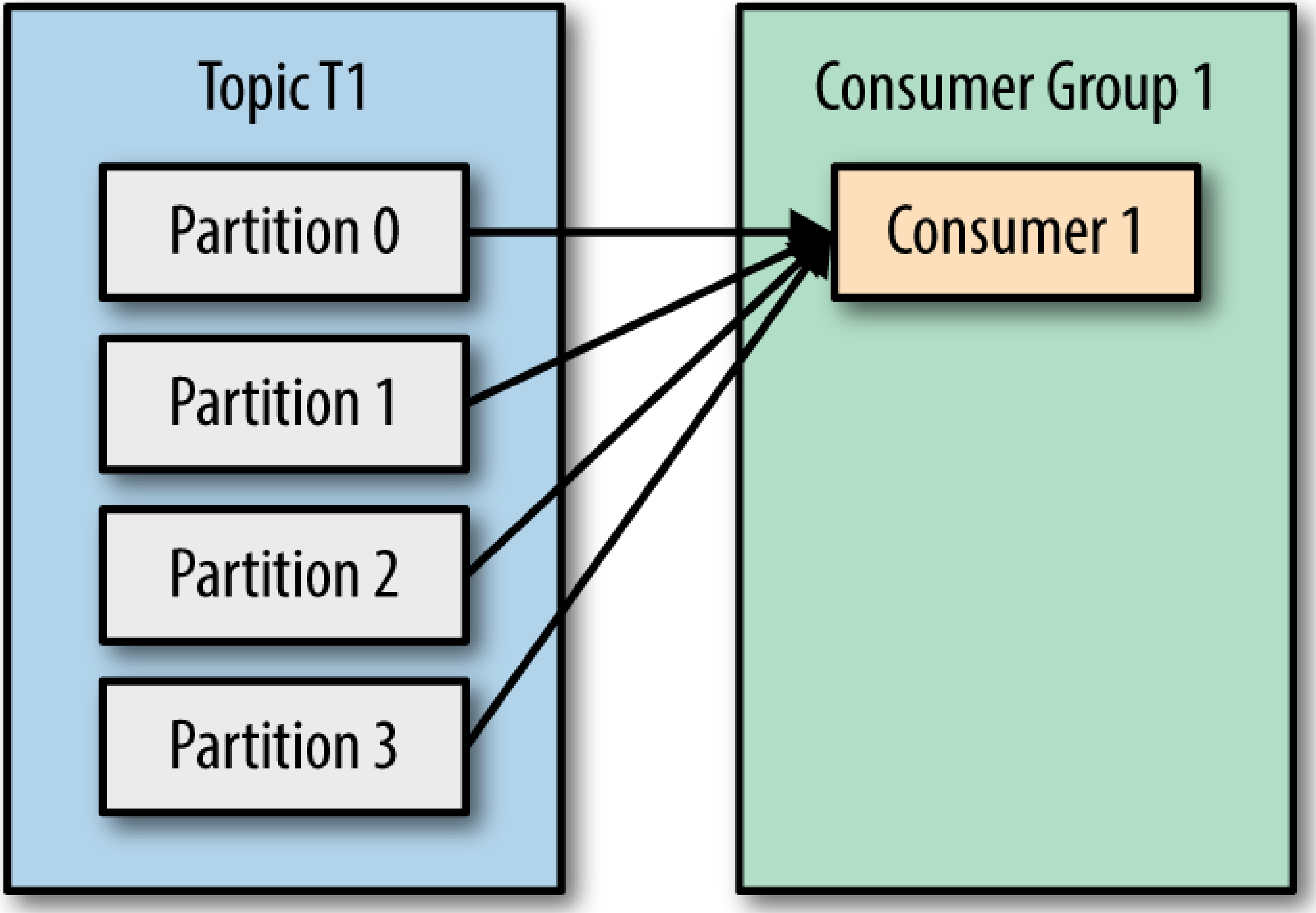
Figure 3-1. High-level overview of Kafka producer components

Processes: Consumers

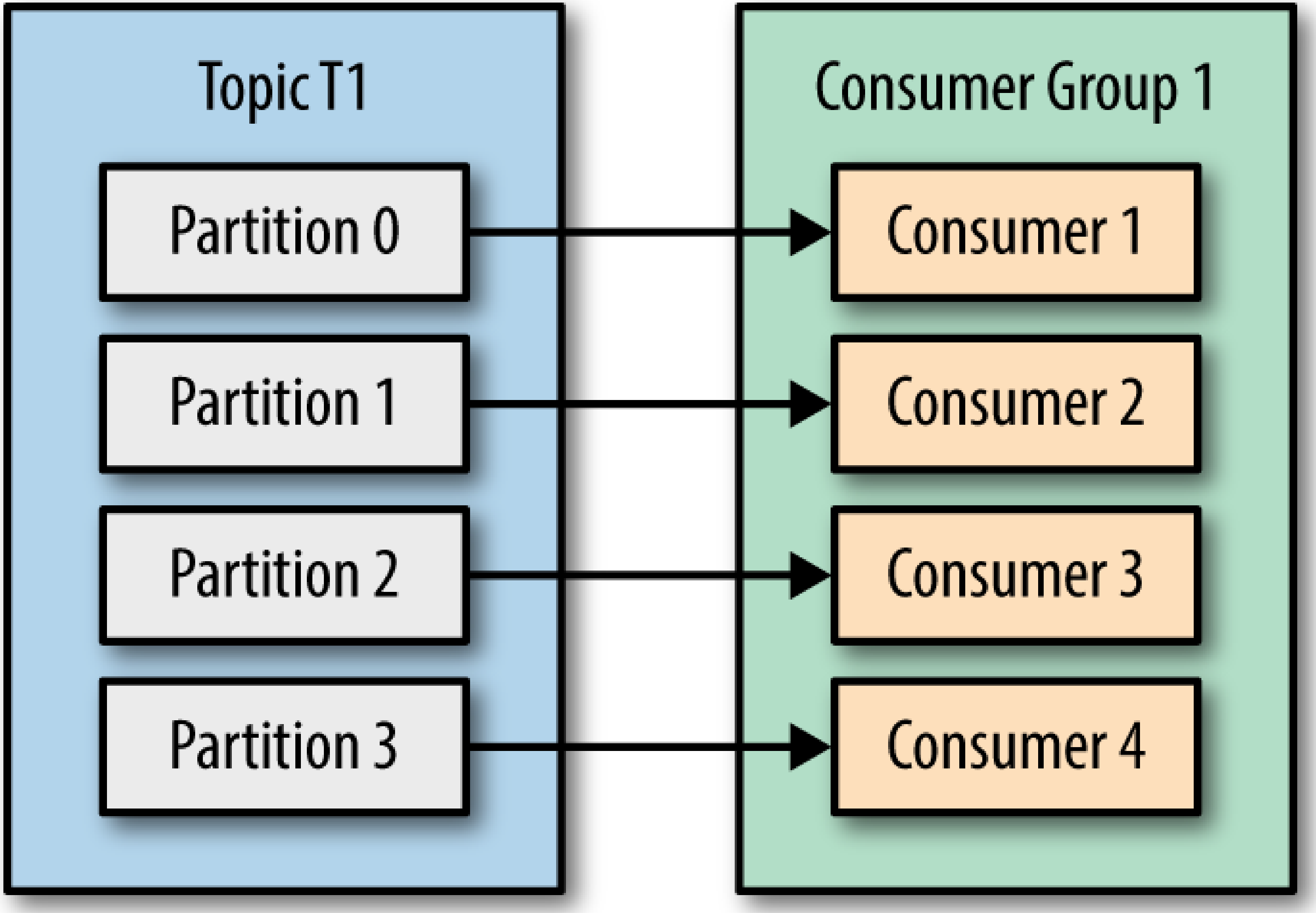
Consumer intro

- **DEF: Kafka consumers** are processes which subscribe to and receive messages from Kafka topics
- **Simple consumer** scenario: a single consumer subscribe to a single or multiple topics and processes the data
  - A single consumer can become a bottleneck if it performs costly data analyses or writes to a database → multi-consumer usage scenario
- **Multi-consumer** scenario: a single consumer process cannot process the tide of incoming, unbounded data flows → consumer groups with multiple consumers
  - The different consumers receive messages from a different subset of topic partitions
- When consumers consume messages, they commit the current partition offsets to a special Kafka topic
  - Earlier versions (prior to 0.10.x) committed offsets to Zookeeper

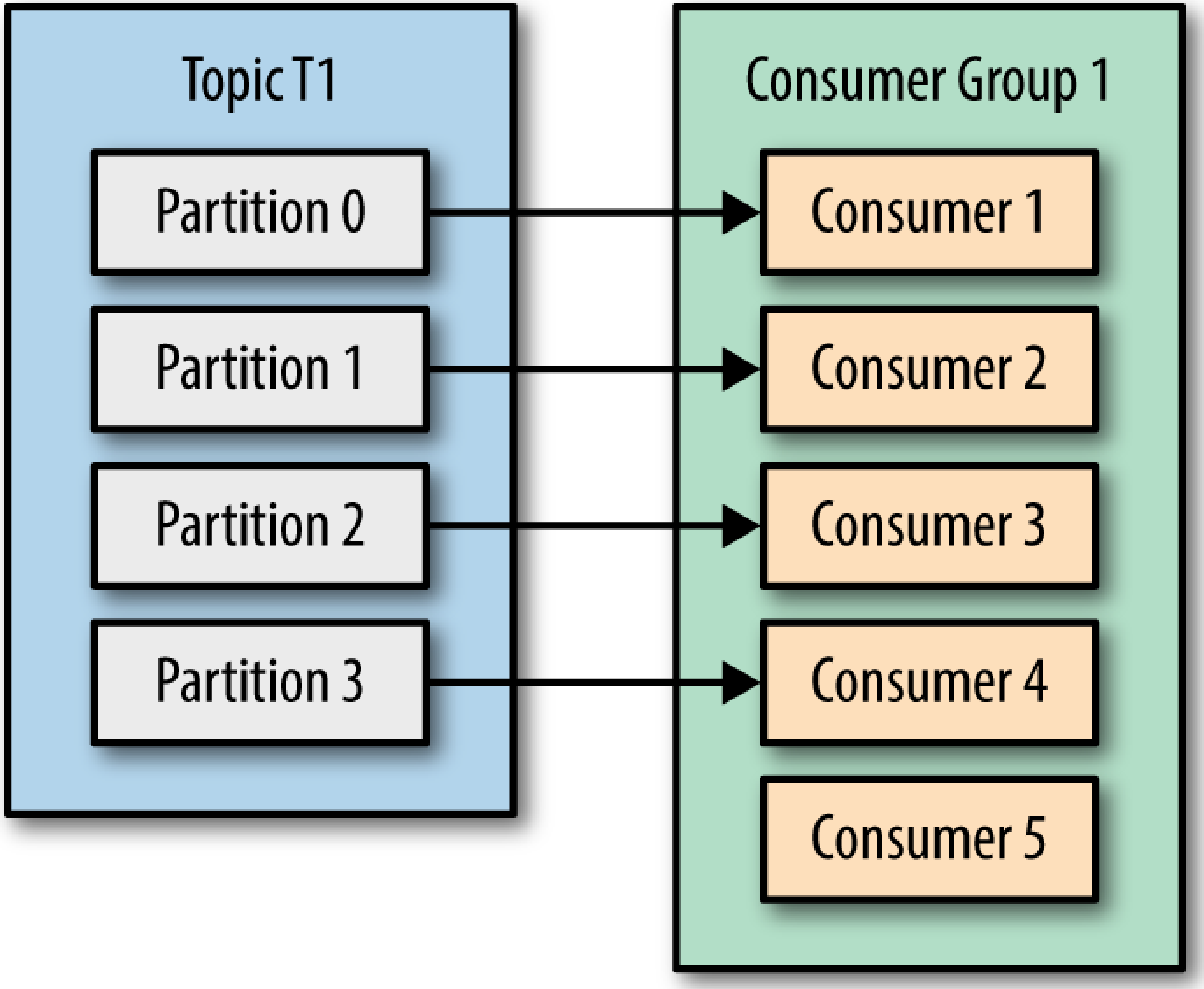
Single group, single consumer



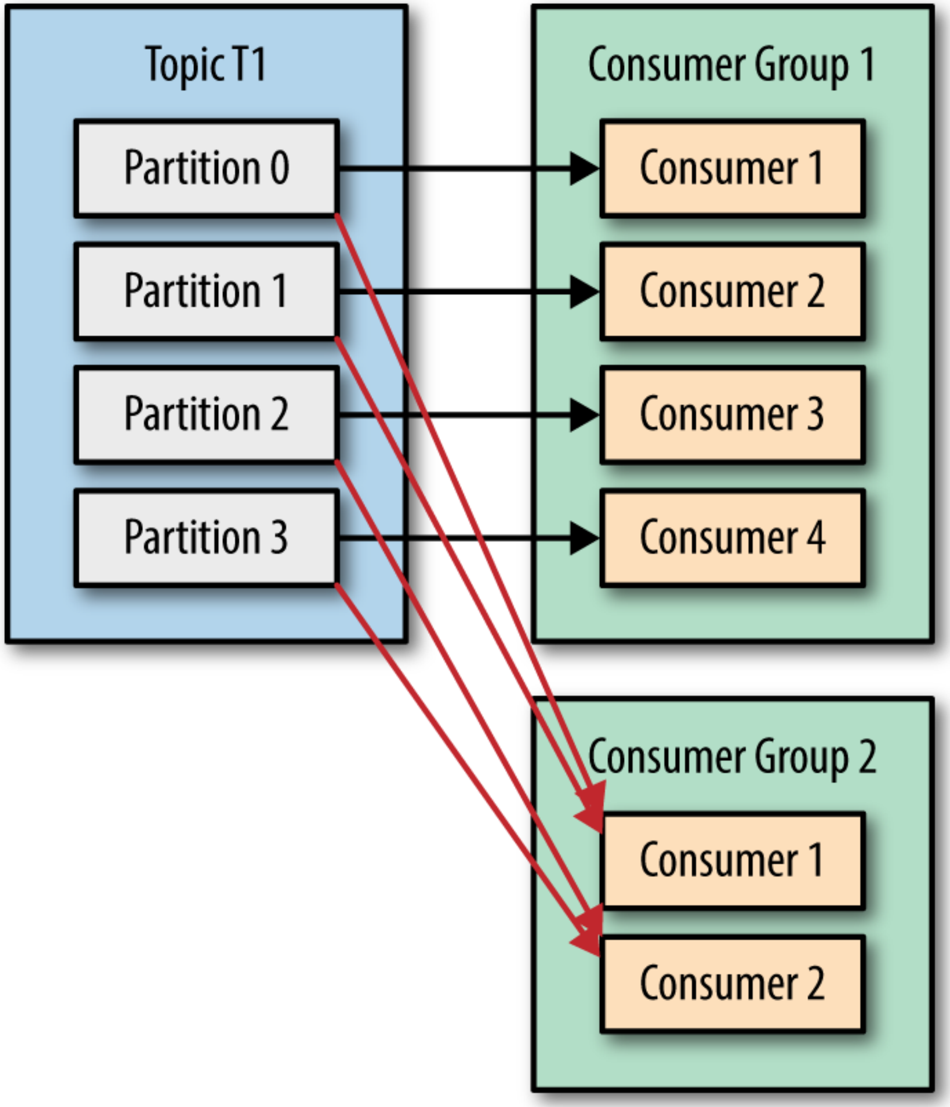
Optimum number of consumers



Too many consumers



Multiple consumer groups



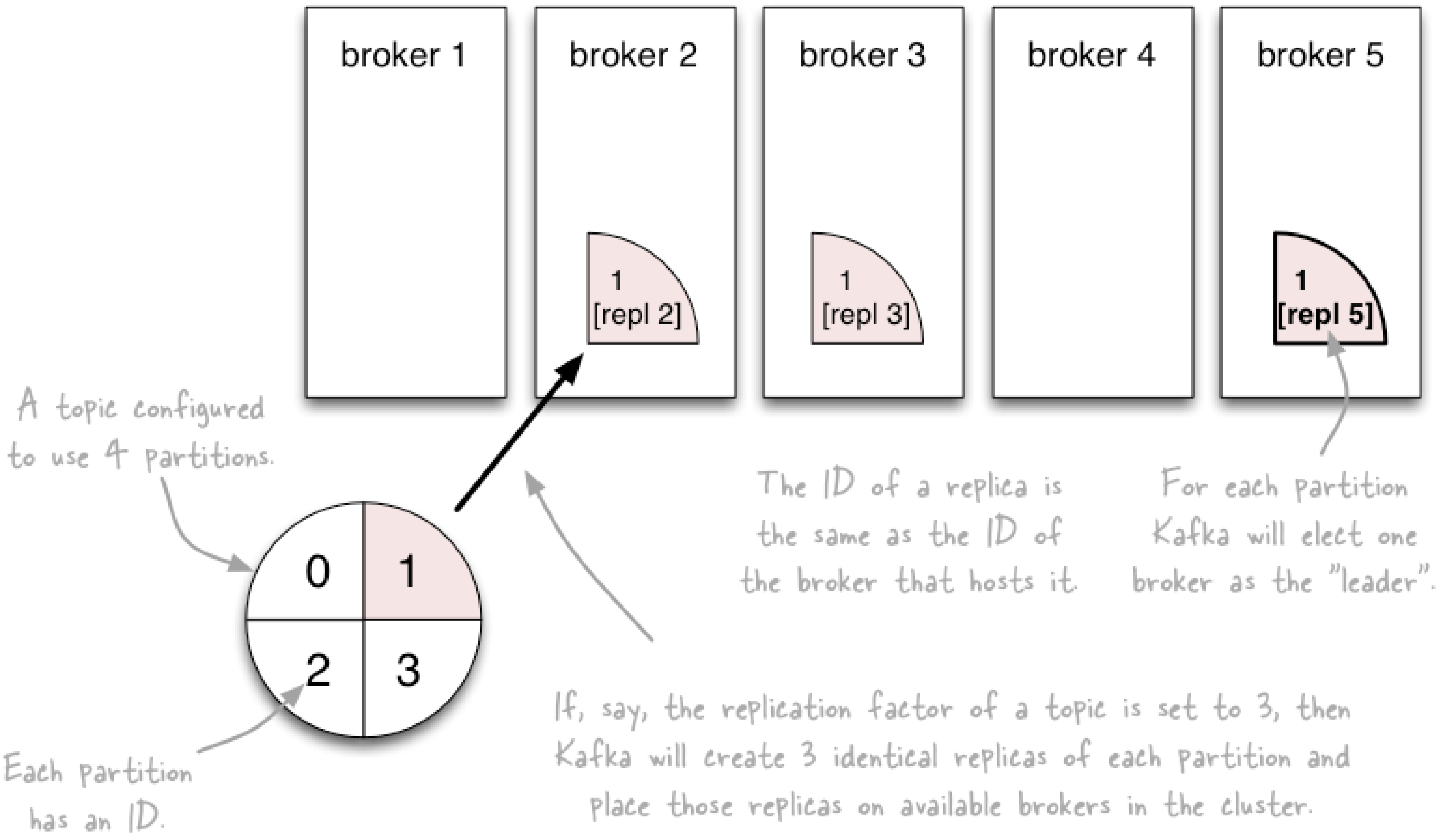
- The different consumer groups can perform different types of data transformations
- Note: adding additional consumers on-the-fly is possible → partition rebalance

Replication and Consistency

Replicas of a partition

- Kafka **replicas** are partition “backups”
  - Kafka tolerates *(numReplicas - 1)* dead brokers before losing data
- Replica types:
  - **Leader replica:** Each partition has a single replica designated as the leader. All produce/consume requests go to the leader.
  - **Follower replica:** All non-leader replicas for each partition are called followers. Followers do not serve produce/consumer requests. They exist to avoid data loss.
- It is the task of the leader to know which follower replicas are up-to-date → in-sync replicas
- Only in-sync replicas can become leaders when the current leader exits the Kafka cluster, e.g. it fails

Topics vs. Partitions vs. Replicas



A topic configured to use 4 partitions.

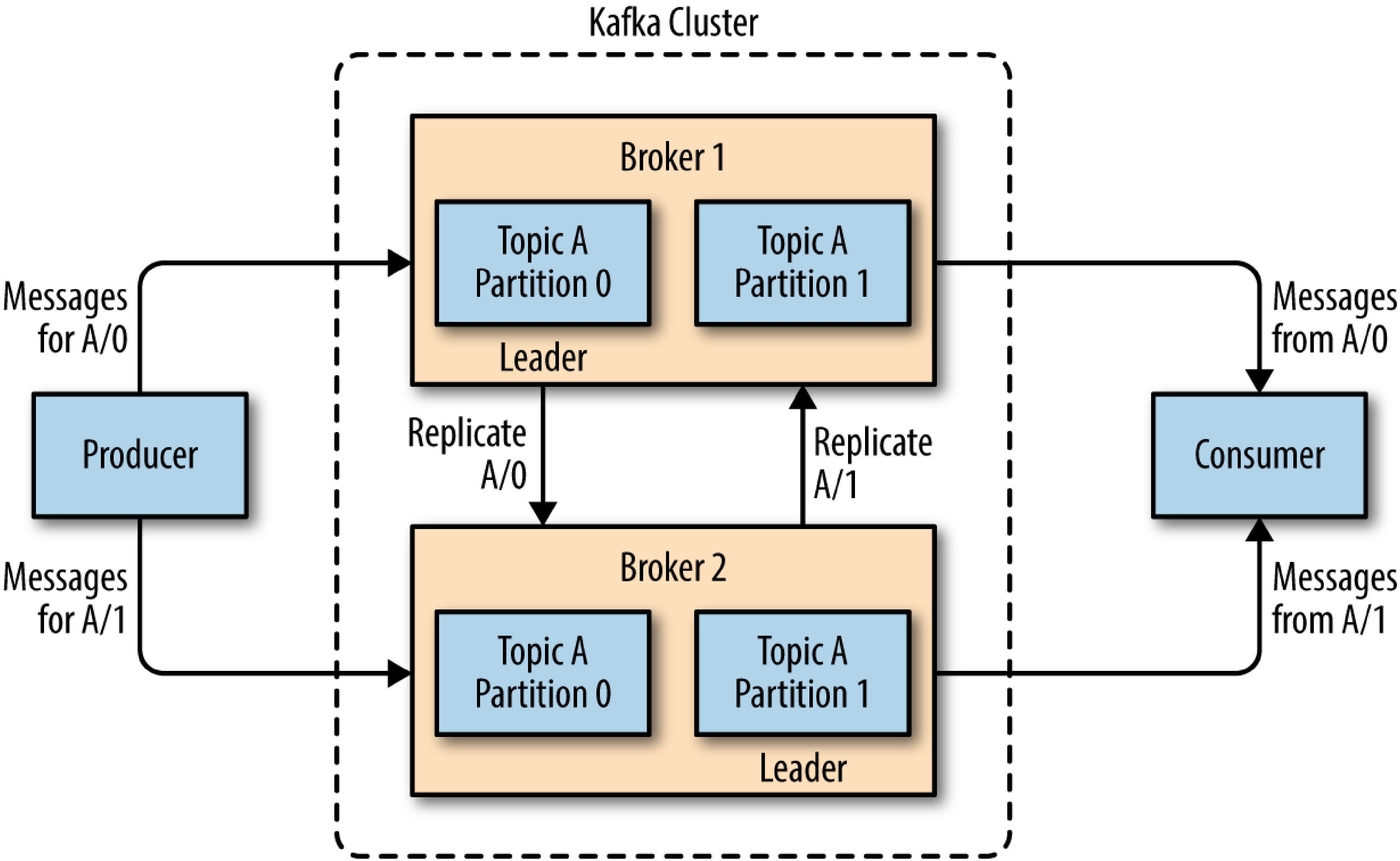
Each partition has an ID.

The ID of a replica is the same as the ID of the broker that hosts it.

For each partition Kafka will elect one broker as the "leader".

If, say, the replication factor of a topic is set to 3, then Kafka will create 3 identical replicas of each partition and place those replicas on available brokers in the cluster.

Partition replication



Replication and visibility

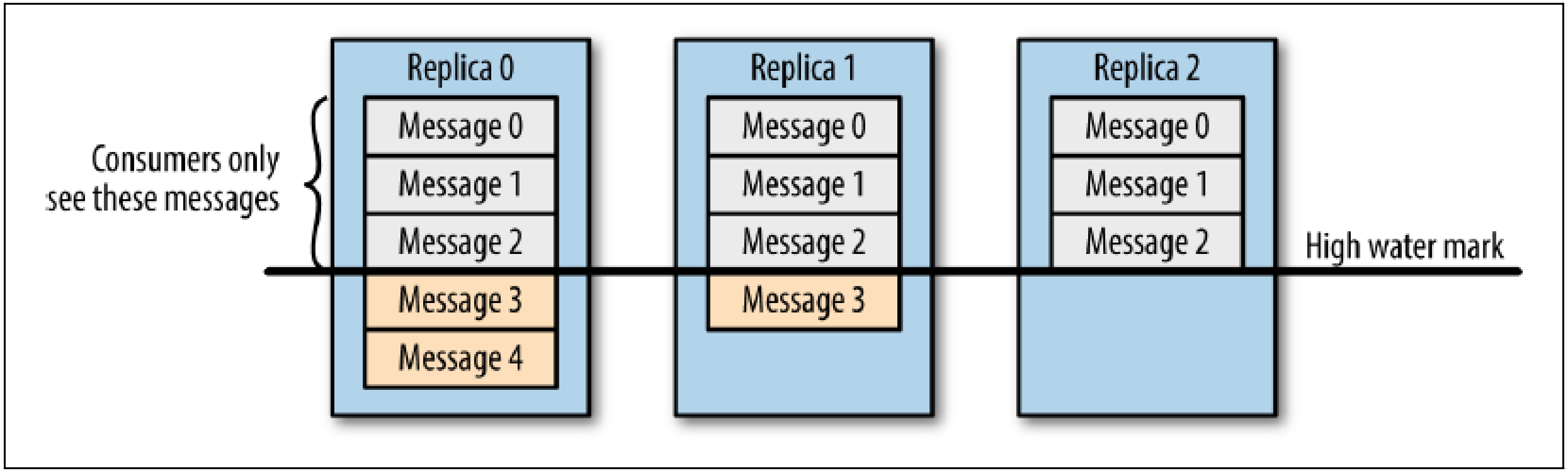
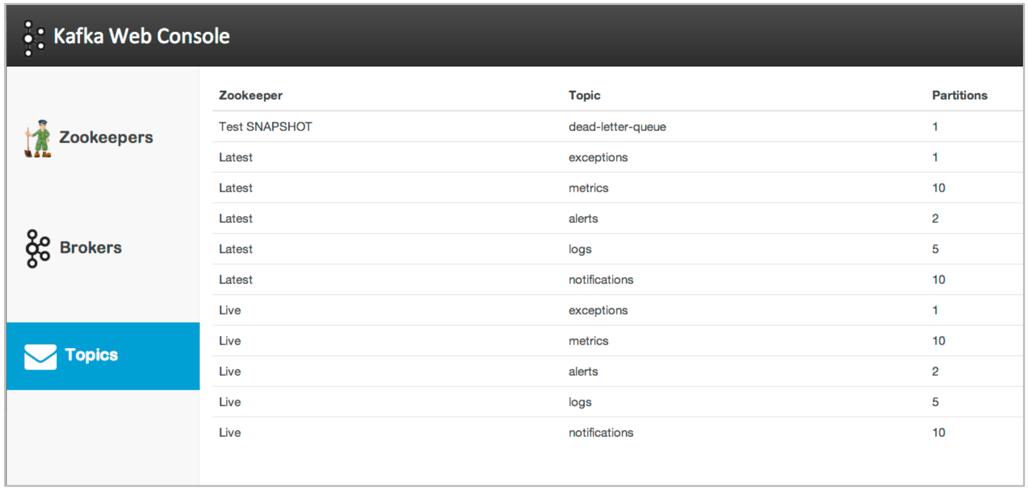


Figure 5-4. Consumers only see messages that were replicated to in-sync replicas

Monitoring and Control

Monitoring Kafka

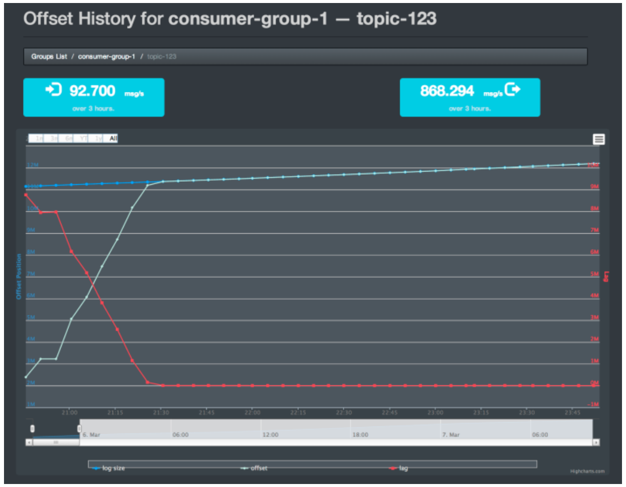
- Nothing fancy built into but see:
  - https://cwiki.apache.org/confluence/display/KAFKA/System+Tools
  - https://cwiki.apache.org/confluence/display/KAFKA/Ecosystem



Kafka Web Console

Zookeeper	Topic	Partitions
Test SNAPSHOT	dead-letter-queue	1
Latest	exceptions	1
Latest	metrics	10
Latest	alerts	2
Latest	logs	5
Latest	notifications	10
Live	exceptions	1
Live	metrics	10
Live	alerts	2
Live	logs	5
Live	notifications	10

Kafka Web Console



Offset History for consumer-group-1 — topic-123

Kafka Offset Monitor

SUMMARY

Who uses Kafka?

- **LinkedIn:** (user) activity streams, operational metrics, data bus
  - 400 nodes, 18k topics, 220B msg/day (peak 3.2M msg/s), May 2014
- **Netflix**: real-time monitoring and event processing
- **Twitter**: as part of their Storm real-time data pipelines
- **Spotify**: log delivery (from 4h down to 10s), Hadoop
- **Loggly**: log collection and processing
- **Mozilla**: telemetry data
- Others: Airbnb, Cisco, Gnip, InfoChimps, Ooyala, Square, Uber, etc.

Summary

- Introduction
- Motivation
- Architecture
- Processes
  - Brokers
  - Producers
  - Consumers
- Replication & consistency
- Monitoring & control

Thank you for your attention!