Kafka Fundamentals

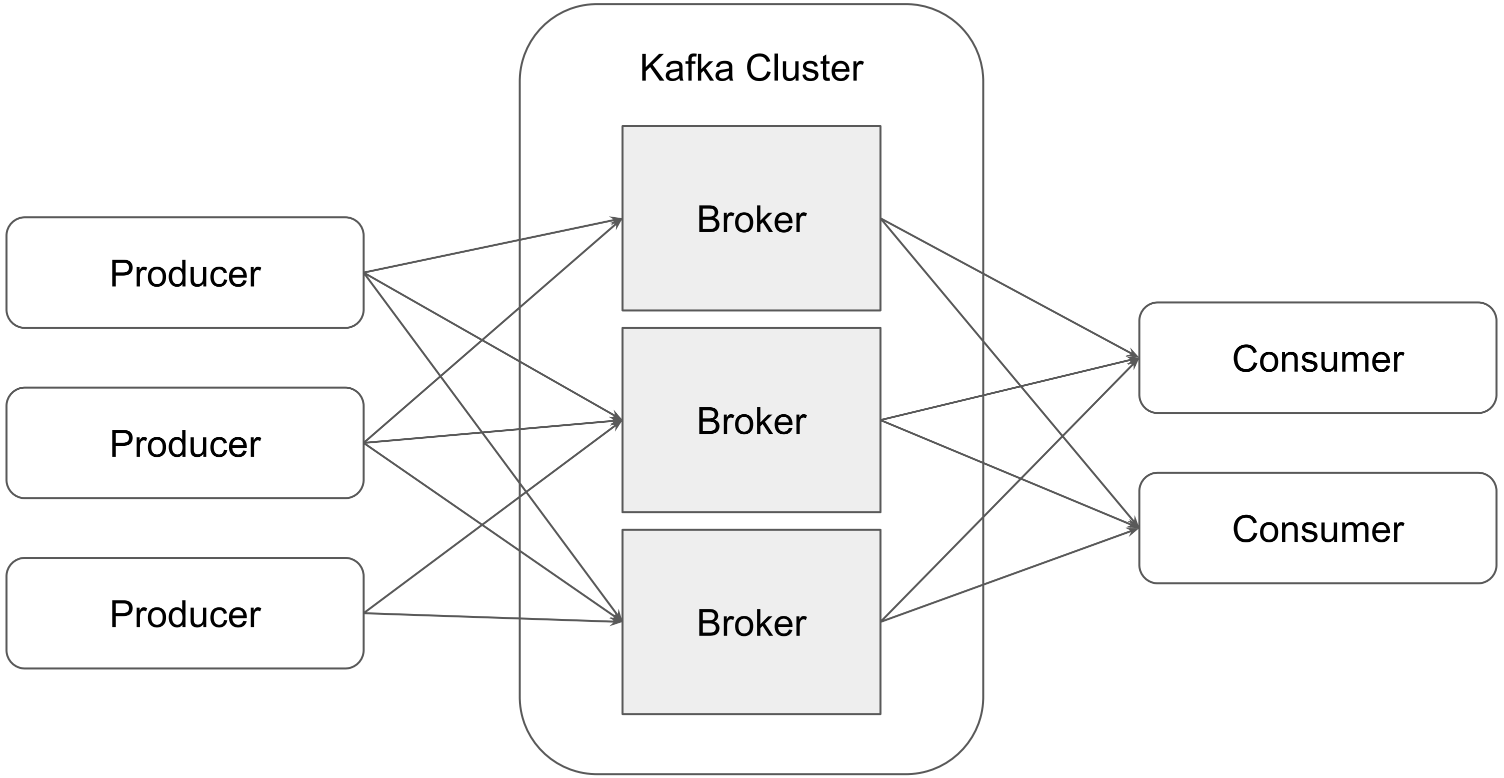
Workshop: Apache Kafka Operations

Agenda

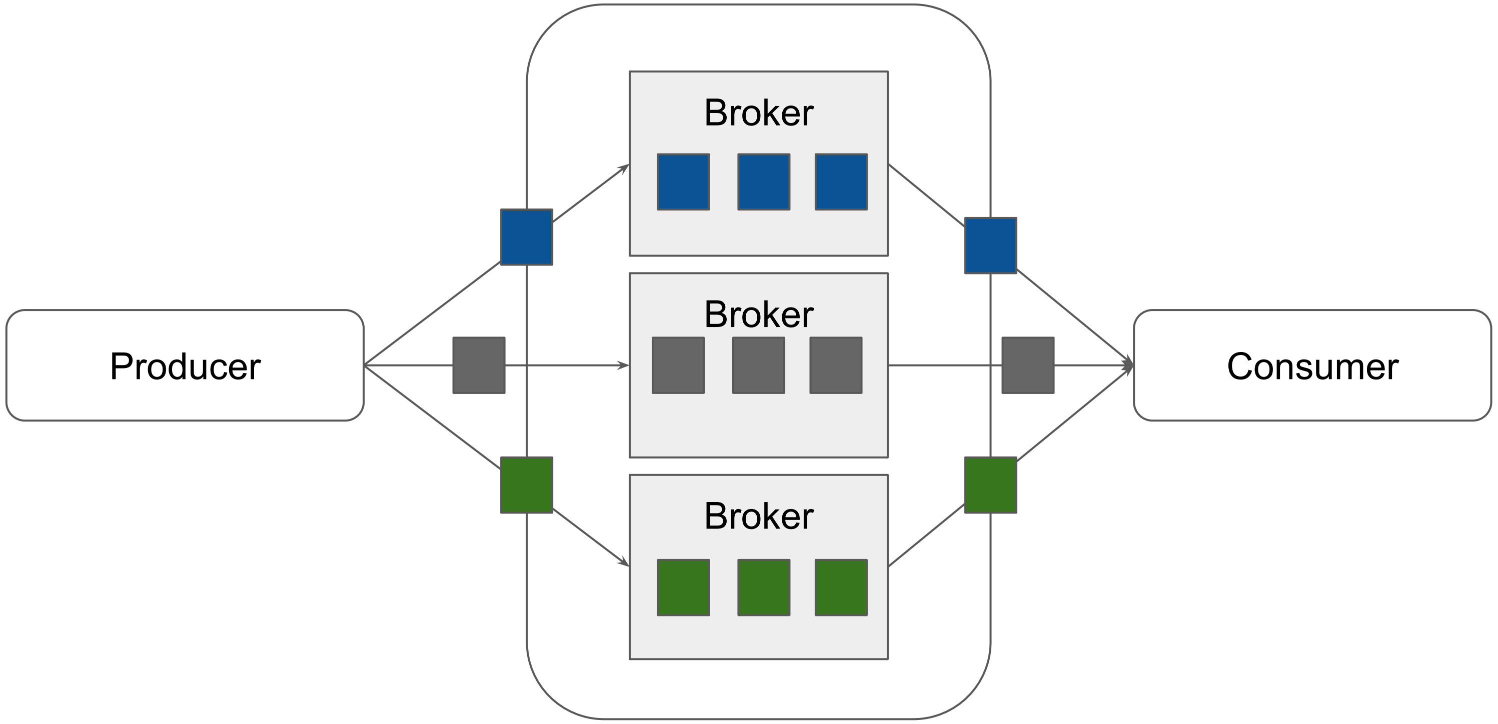
- An Overview of Kafka
- Kafka Producers
- Kafka Brokers
- Kafka Consumers
- Zookeeper
- Hands-on lab: Kafka Installation

An Overview of Kafka

High level view of Apache Kafka



Kafka Messages (a.k.a. Records)



Key-Value pairs

A message is a **Key-Value** pair

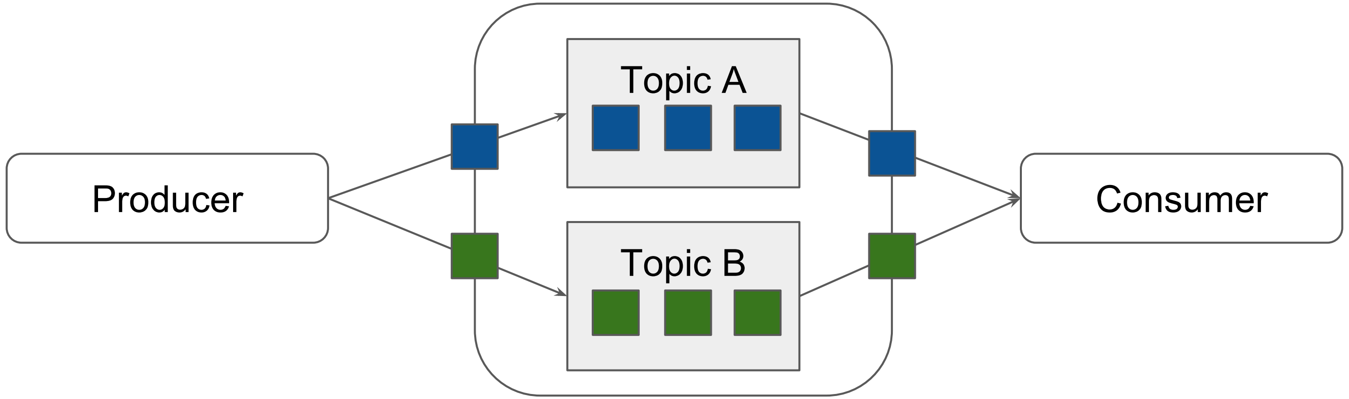
- ***All*** data is stored in Kafka as **byte arrays**
- Producer provides **serializers** to **convert** the key and value to **byte arrays**
- Key and value can be ***any* data type**

Topics

Stream of messages are categorized in a **logical representation** called ***Topics***.

`1..* Producers can write to 1..* Topics`

No limit to the number of **Topics** that can be used.

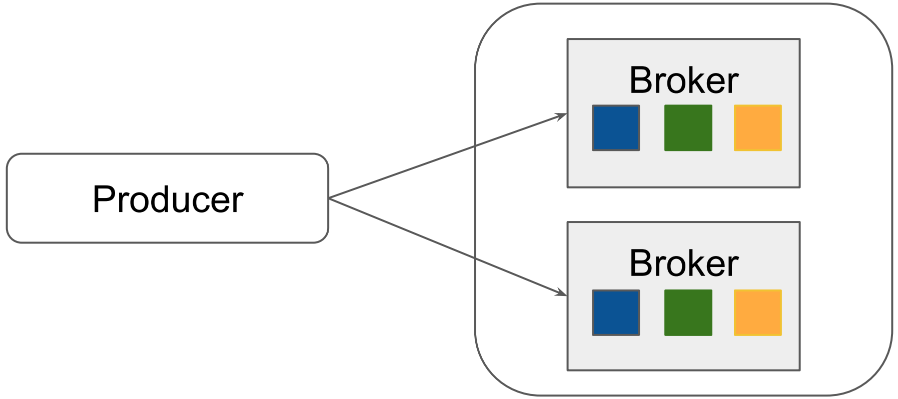


Partitioned Data Ingestion

Producers ***shard*** data over a set of Topic **Partitions**.

- Each **Partition** contains a **subset** of Topic's messages.
- Each **Partition** is an ***ordered*, immutable log** of messages.

Partitions are distributed across the Brokers.



Load Balancing and Semantic Partitioning

Producers use a **partitioning strategy** to assign each message to a Partition:

- **Load balancing:** shares the load across the Brokers
- **Semantic partitioning:** user-specified key allows locality-sensitive message processing

The partitioning strategy is ***specified by the Producer***

- Default strategy is a hash of the message key
  - `hash(key) % number_of_partitions`
- If a key is *not specified*, messages are sent to Partitions on a *round-robin basis*

Kafka Producer

Producer basics

Producers **write data** in the form of messages to the Kafka cluster

Producers can be written in **any language**

- **Native Java/Scala** come with the open-source project
- Clients for many other languages exist
- C/C++, Python, Go, .NET, JMS clients are **supported by Confluent**, plus a **REST Proxy** for other languages.

A command-line Producer tool exists to send messages to the cluster

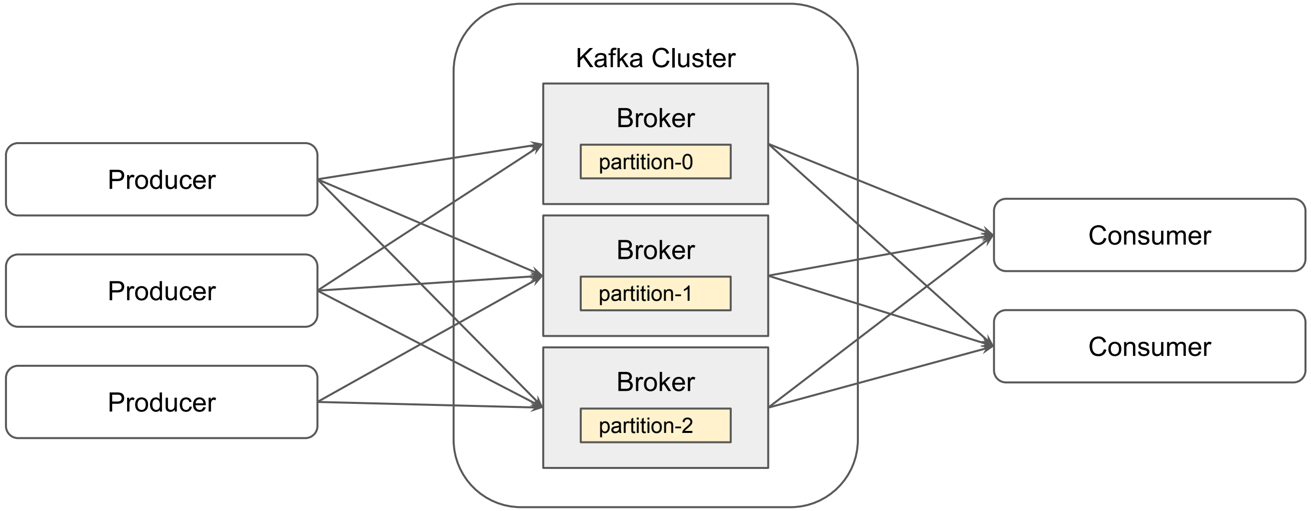
Kafka Brokers

Brokers basics

Brokers **receive and store messages** when they are sent by the Producers

A ***production*** Kafka cluster will have **three or more Brokers**

- Each can handle hundreds of thousands, or millions, of messages per second



Brokers and Partitions

Messages in a **Topic** are spread **across Partitions** in **different Brokers**

Typically, **a Broker** manages **multiple Partitions**

Each Partition is **stored** on the Broker’s disk **as one or more log files**

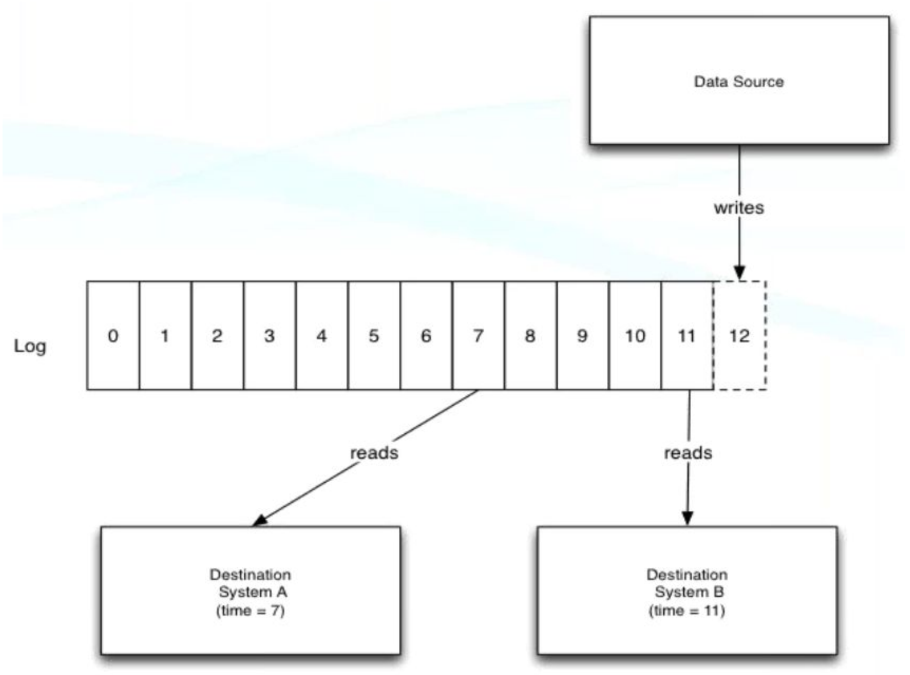
Each message in the log is **identified by its offset** number

- A monotonically increasing value

Kafka provides a **configurable retention policy** for messages to manage log file growth

- Retention policies can be configured per Topic

Messages are stored in a Persistent Log


Data Source
writes
Log
0
1
2
3
4
5
6
7
8
9
10
11
12
reads
reads
Destination System A (time = 7)
Destination System B (time = 11)

Kafka Message Structure

A Kafka Message contains data and metadata

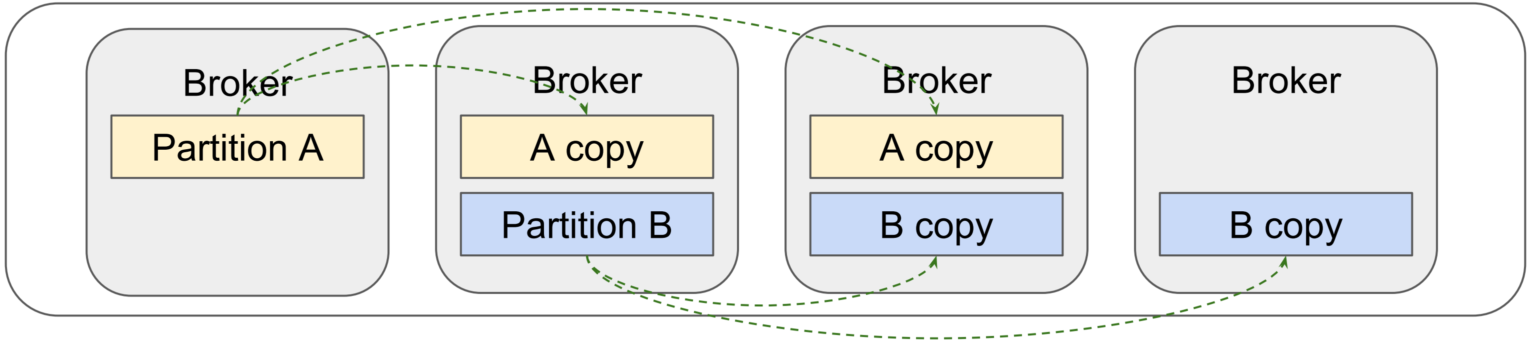
- Key/Value pair
- Offset
- Timestamp
- Compression type
- Magic byte
- Optional message headers
- Additional fields to support batching, exactly once semantics, etc

Docs: http://kafka.apache.org/documentation.html#messageformat

Fault tolerance via Replicated Log

Partitions can be **replicated across** multiple **Brokers**

Replication provides **fault tolerance** in case a Broker goes down (automatically)



Kafka Consumers

Consumer Basics

Consumers **pull messages from** one or more Topics in the cluster

The **Consumer Offset** keeps track of the latest message read

- If necessary, the Consumer Offset can be changed/shifted

The **Consumer Offset** is stored in a special Kafka Topic (by default)

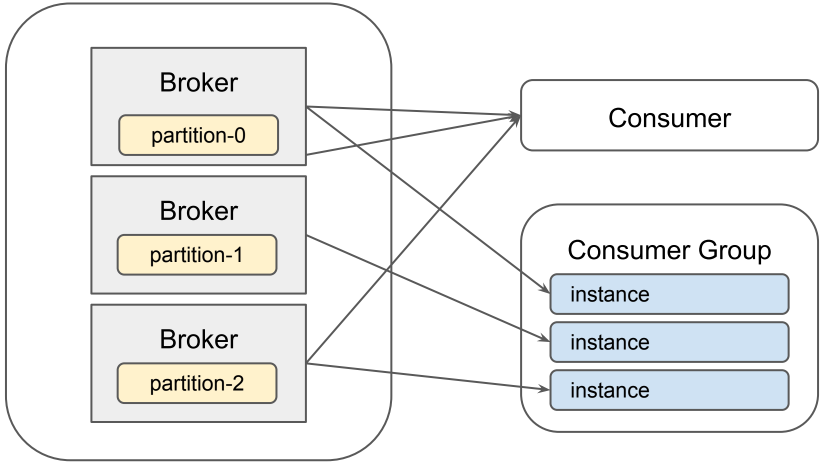
A command-line Consumer tool exists to read messages from the cluster

Distributed Consumption

Different Consumers can read data from the same Topic

Multiple Consumers can be combined into a Consumer Group

- Consumer Groups provide scaling capabilities
- Each Consumer is assigned a subset of Partitions for consumption



Zookeeper

What is Zookeeper?

ZooKeeper is a centralized service that can be used by distributed applications

- Open source Apache project
- Enables highly reliable distributed coordination
- Maintains configuration information
- Provides distributed synchronization

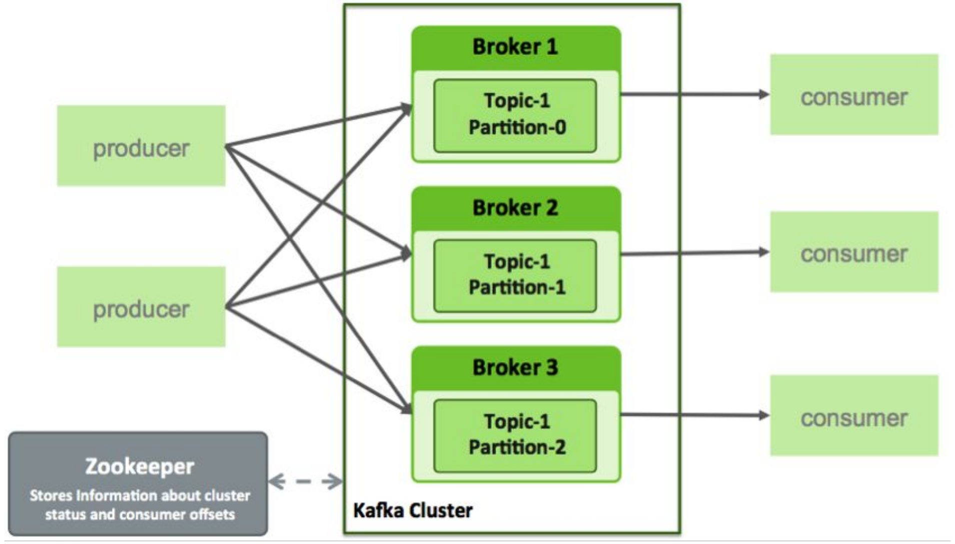
Used by many projects (Including Kafka and Hadoop)

Typically consists of ***three*** or ***five*** servers in a ***ensemble***

How Kafka uses Zookeeper

Kafka Brokers use ZooKeeper for a number of important internal features

- Cluster management
- Failure detection and recovery
- Access Control List (ACL) storage



Additional features

The Page Cache for High Performance

Unlike some systems, Kafka itself **does not** require **a lot of RAM**

Logs are held **on disk** and read when required

Kafka makes use of the operating system's **page cache** to hold recently-used data

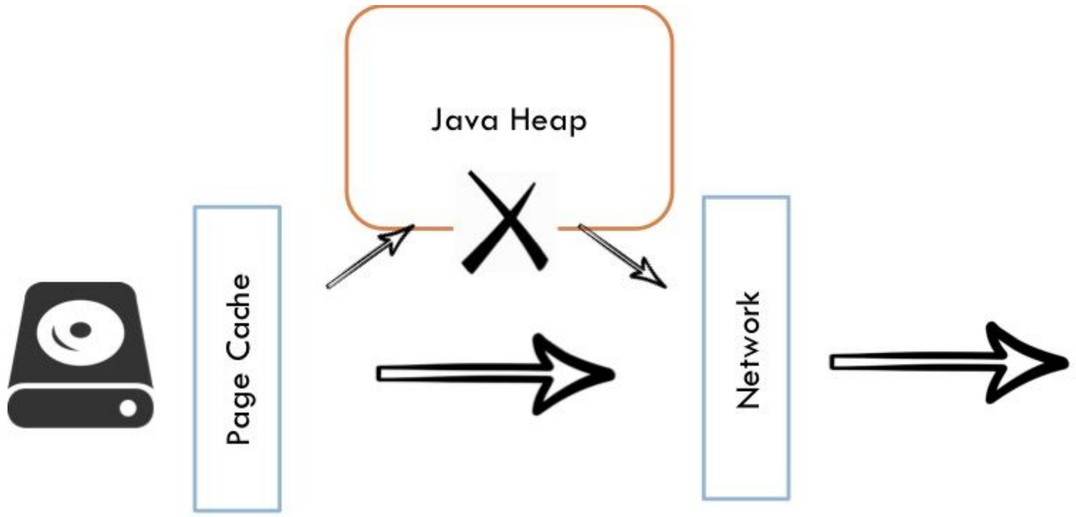
- Typically, recently-Produced data is the data which Consumers are requesting

A Kafka Broker running on a system with a reasonable amount of RAM for the OS to use as cache will typically be able to swamp its network connection

- In other words the network, not Kafka itself, will be the limiting factor on the speed of the system

Speeding Up Data Transfer

Kafka uses **zero-copy data transfer** (Broker → Consumer)



Kafka metrics

Kafka has metrics that can be exposed in JMX and inspected through JMX clients

Types of metrics:

- **Gauge**: instantaneous measurement of one value
- **Meter**: measurement of ticks in a time range, *e.g. OneMinuteRate, FiveMinuteRate.*
- **Histogram**: measurement of a value variants, *e.g. 50thPercentile, 99thPercentile.*
- **Timer**: measurement of timings Meter + Histogram

Hands-on lab: Kafka Installation