# CE 528 Cloud Computing

## Lecture 12: Data Processing
## Spring 2026

**Prof. Yigong Hu**


Slides courtesy of Ata Turk

# Administrivia

## Demo 2 is on this Wednesday

| Criteria | Expectation |
| --- | --- |
| Progress | A prototype that can address the problem proposed in Demo 1 |
| Code Quality | Working code with no bugs or issues |
| Design Doc + Video | TA can independently run your code following the doc and demo video |
| Slides | Clearly communicates the design and architecture of your prototype |
| Novelty | Demonstrates the technical difficulty and ambition of the project |

# Administrivia

## Demo 2 logistics

| Deadline | Deliverable | Penalty for missing |
|---|---|---|
| Before 12:00 PM | Updated slides + short demo video uploaded to your repo | 50% deduction on slide or 50% on design doc score |
| Before 3:00 PM | One quiz question (with answer) sent to instructor's email | 50% deduction on demo quiz score |
| Before 11:59 PM | Upload all the code into your main branch | 50% deduction on Progress and code |

## Presentation

- 8–10 minutes to present
- No Q&A session

# 500 Internal Server error

## Look at a log: demo

## What is a log?

- Recorded set of events and activities occurring within a computer system or software application

# How OS handles I/O?

## What lives on a disk?

- Files don't live on disk
- Bytes live on disk
- Files are an interface

## `echo "foo" > bar.txt` takes microseconds to write

1. Operating system copies “foo” into the page cache
2. A few microseconds later the OS tells you the write succeeded
3. Asynchronously, up to 30 seconds later the OS actually writes “foo” to disk

# A Few Scenarios

**What happens when lot of processes are trying to write to disk?**

**What happens when we try to read a file while it's still being written?**

**What happens when a packet is sent over the network?**

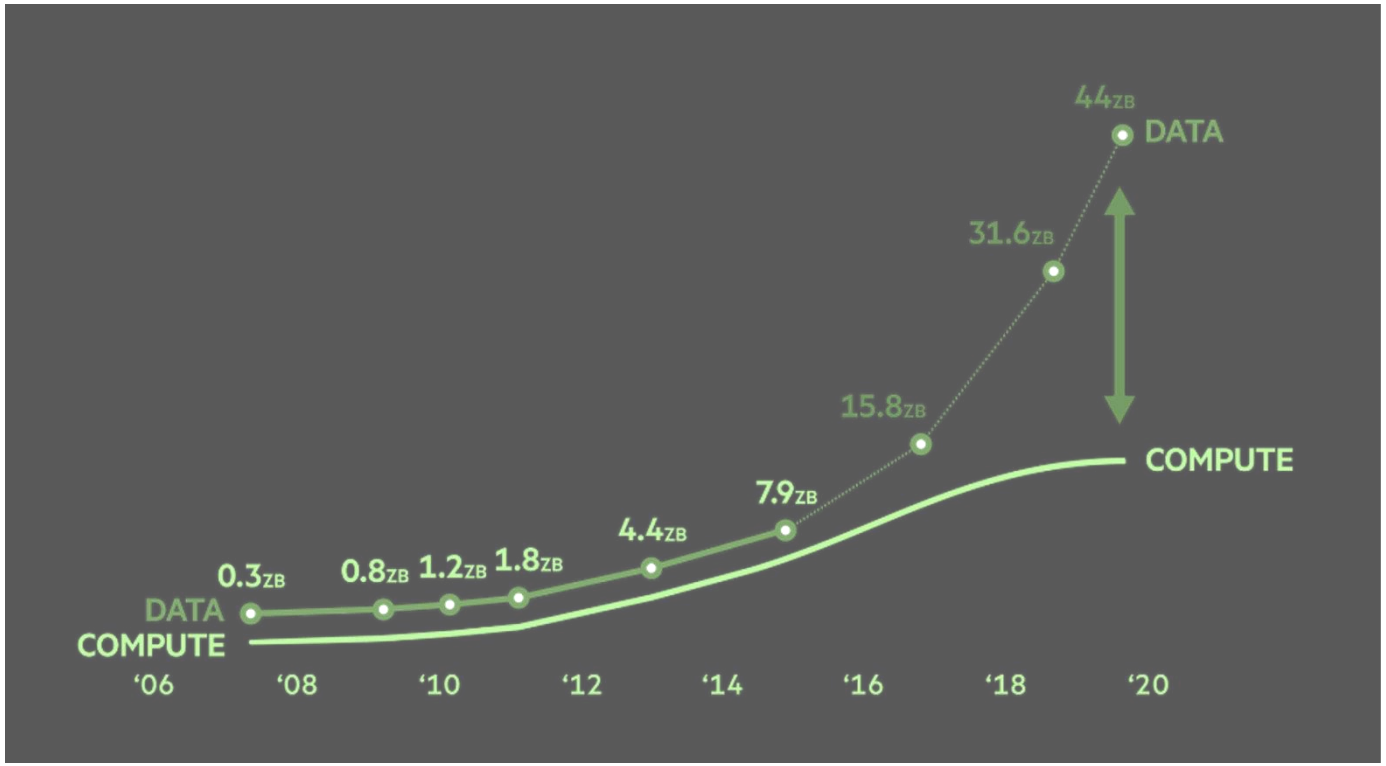
**What is the impact of sending small files over the network?**

# Kafka: a Distributed Messaging System for Log Processing

**Jay Kreps, Neha Narkhede and Jun Rao**

Slides adapted from: Ken Krebs, Cagri Yoruk, Brian Mahabir, Tingyi Zhang and Xiaoyu An

# Motivation




- **Every 2 years, we create 2x more data than what we have created in all of human history**
- **Efficiency of computer systems needs to catch up**

[1] Kirk Bresniker, World Economic Forum, 2018

# Motivation

## New breed of real-time log data usage

- User activity events in logs:
  - logins, pageviews, clicks, comments, …
- Analyzing logs, we can improve search relevance, recommendations, user status updates, …

## Messaging system must make trade-off between

- Scale (data is huge and growing fast)
- High throughput
- Low latency
- Reliability

# Traditional Messaging System

**Strong delivery guarantees**

**Not optimized for throughput.**

**Limited scaling**

**Performance degrades as messages began to accumulate.**

# Differences: Kafka and Messaging Systems

| | Kafka | Messaging system |
|---|---|---|
| Delivery guarantees | none* | Lots |
| Near Real time analysis | Yes | No |
| Latency | low | low |
| Throughput | high | low |
| Consumer model | pull | push or pull |

# Kafka

**Distributed, high throughput, pub-sub messaging system**

- Durable, highly available, scalable
- Producers push data
- Consumers pull data
- Kafka partitions and stores messages on multiple machines (brokers)

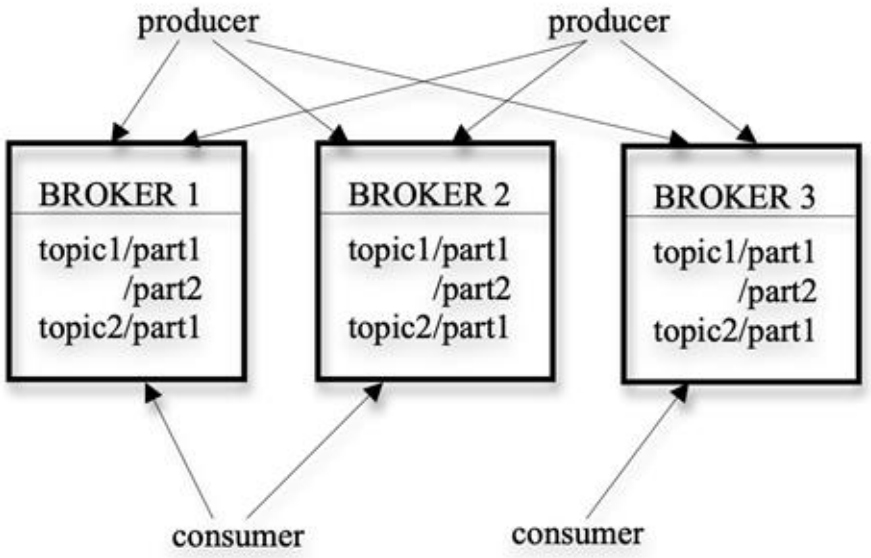



**Figure 1**. Kafka Architecture

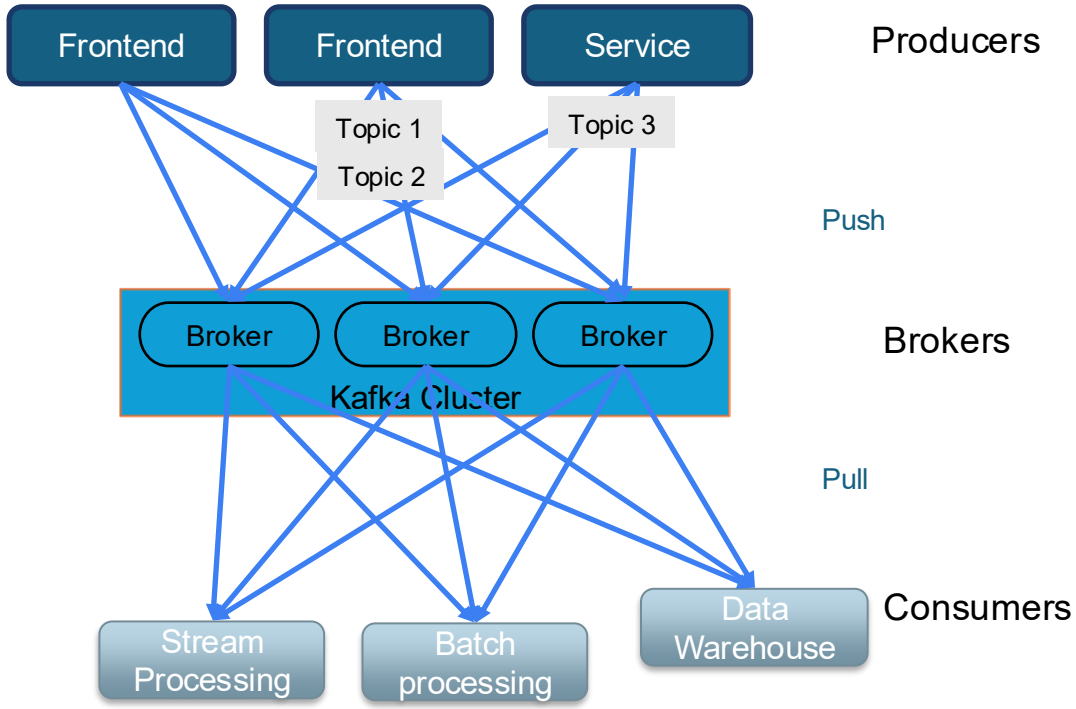
# Kafka Terminology

Topics: categories in which message feed is maintained

Producer: Processes publishing msgs to Kafka

Consumers: Processes subscribing to topics & processing the feed of published messages

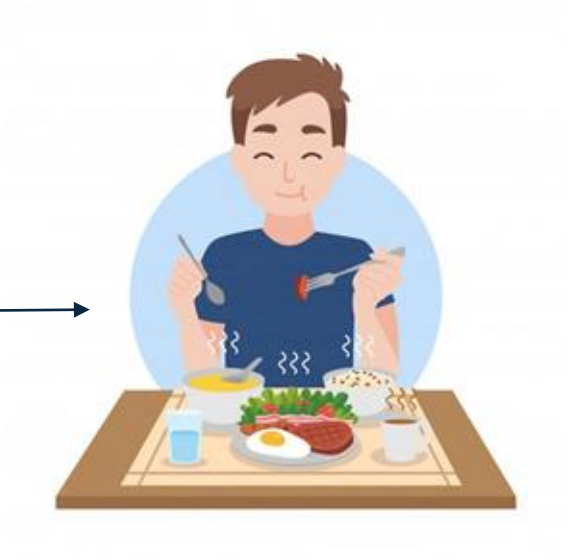
Brokers: Servers forming the Kafka cluster, they act as data transport channel bw producers and consumers



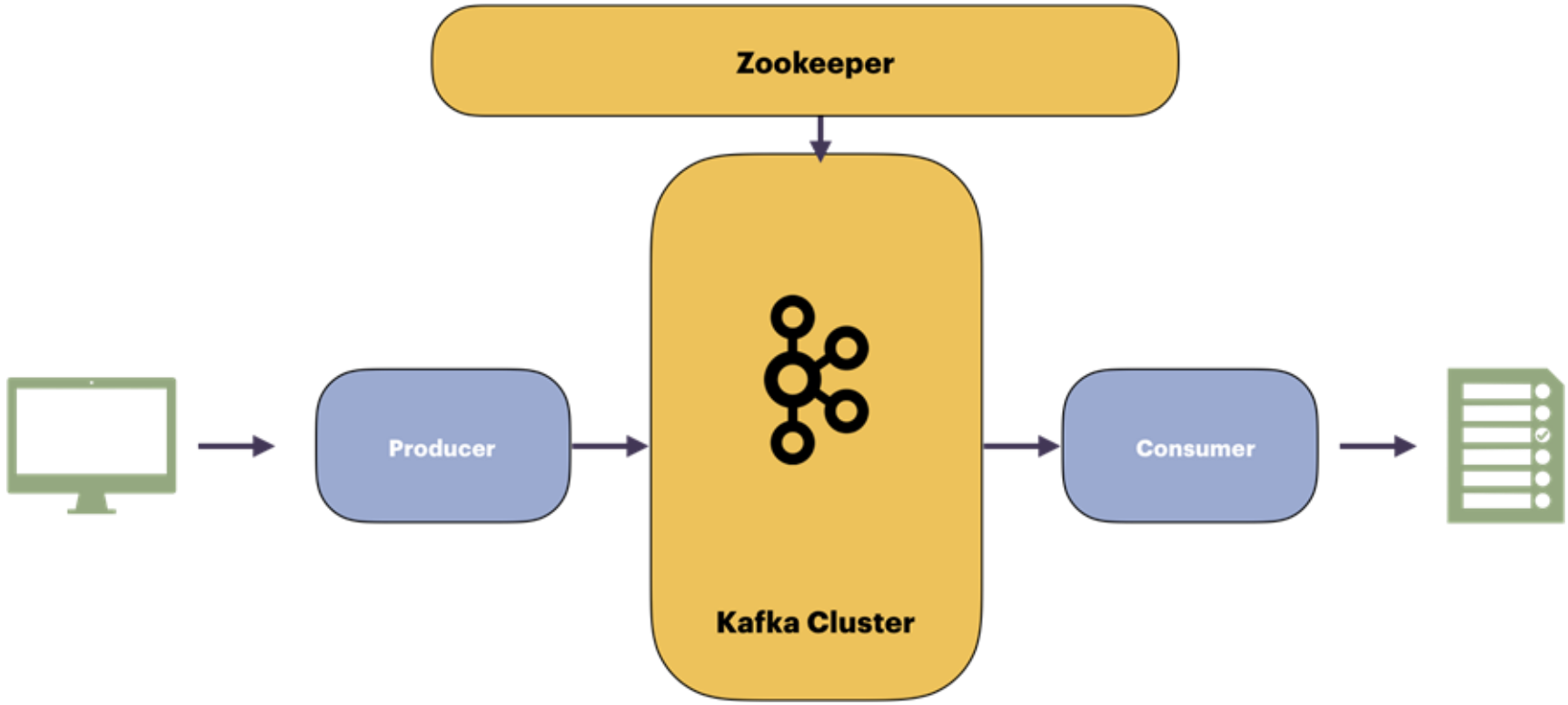
# What is a producer and consumer?


# What Is a Messaging System?





# Kafka Interface

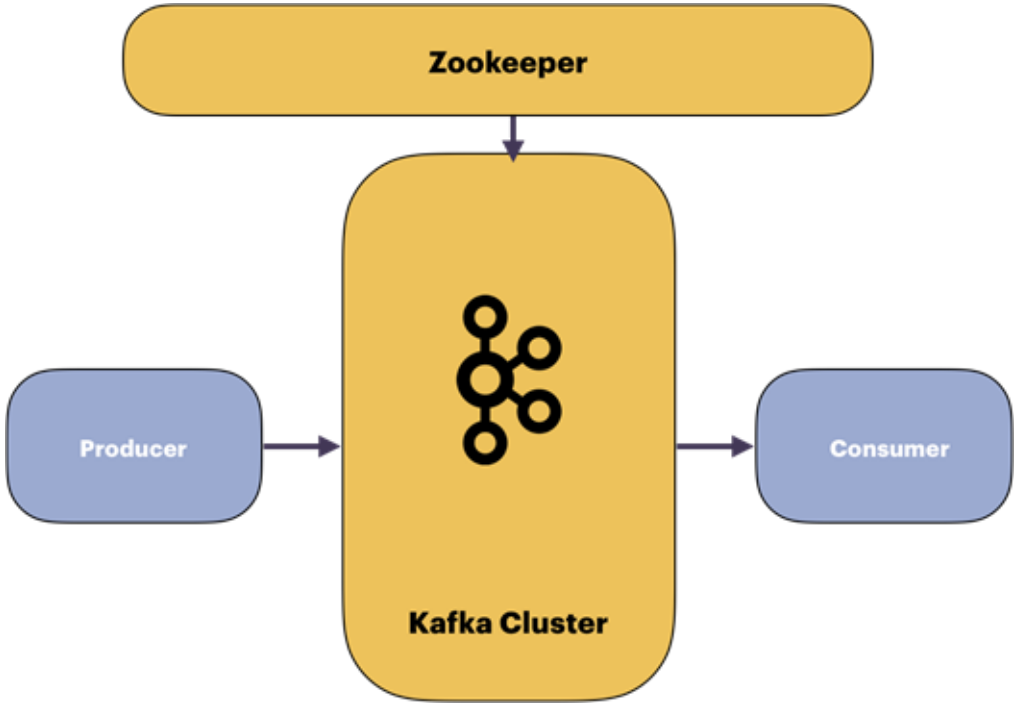


# Producer and Consumer Behavior

**Multiple producers and consumers can publish and retrieve messages at the same time.**

**Kafka is a multi subscriber system so a single message may be consumed multiple times by different consumer applications.**

**Producer and Consumer systems are decoupled**



# Kafka Interface

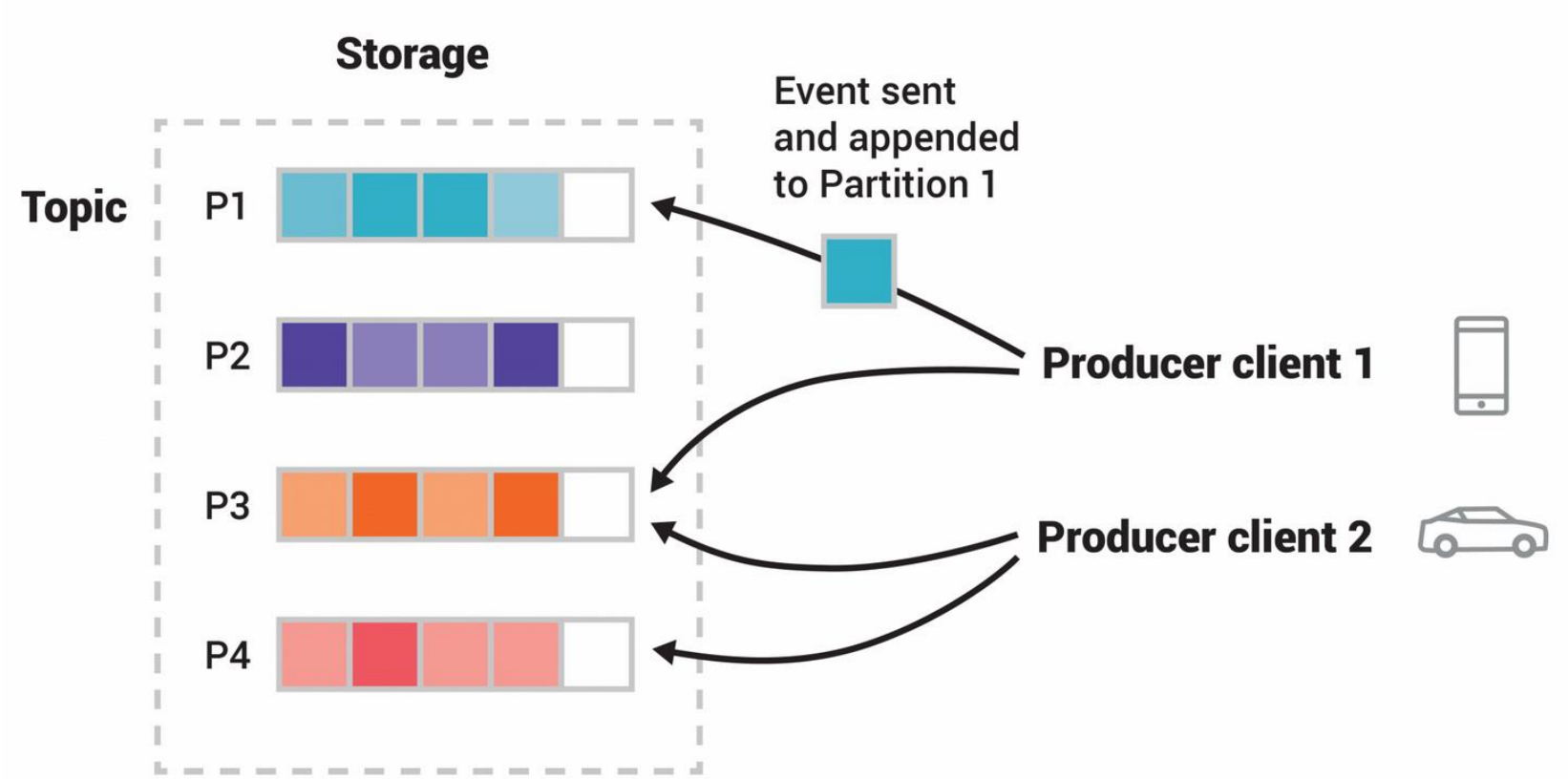
**Producer** - An application that sends messages (data) into a Kafka cluster.

```
producer = new Producer(…);
message = new Message(“test message str”.getBytes());
set = new MessageSet(message);
producer.send(“topic1”, set);
```

**Consumer** - An application that reads out messages (data) from a Kafka cluster.

```
streams[] = Consumer.createMessageStreams(“topic1”, 1)
for (message : streams[0]) {
    bytes = message.payload();
    // do something with the bytes
}
```

# Kafka Topics and Partitions




**Partitions: A an ordered, append-only log of events that divide a topic.**

**Consumers can consume from multiple partitions in parallel**

# More Details about Kafka Partitions

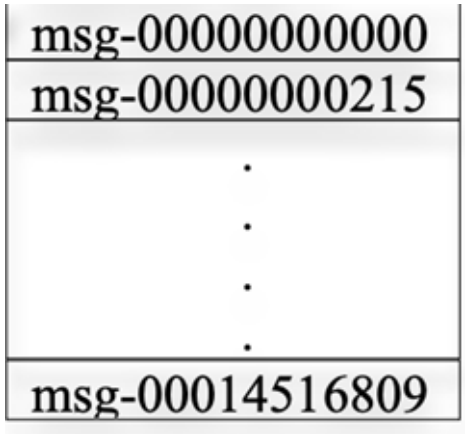
**Msgs are continuously appended to partitions**

**Each msg is assigned an id called offset**

**A msg can be accessed using offset**

**Msg ids are increasing but not consecutive.**

**To compute id of the next msg, add the length of the current msg to its id**



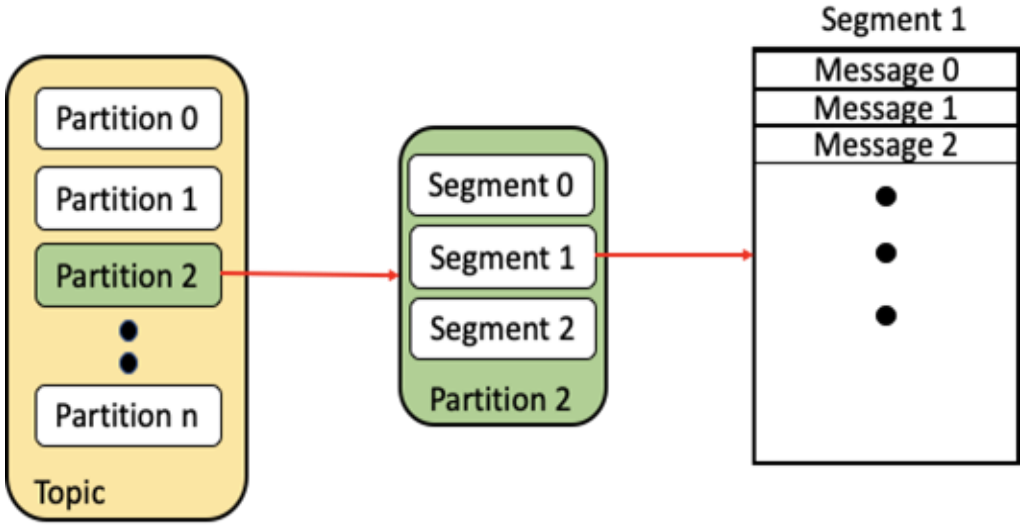
# Kafka Storage

**Each partition is stored as a set of segment files of uniform size**

**Each partition has one leader and N-1 followers brokers**

**When producer publishes a msg, broker appends msg to last segment file**

**Flushing segment files are batched**



# More on Storage

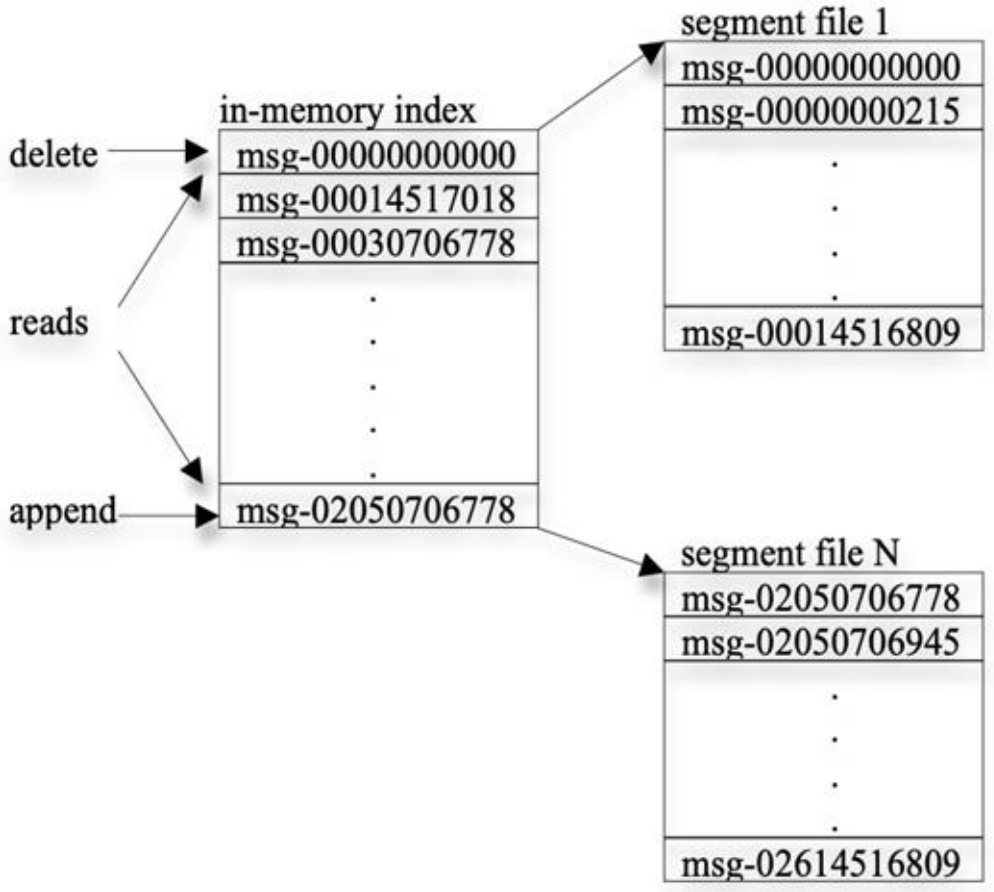
**Kafka Index: growing at the bottom, shrinking at the top.**

**Why segment files instead of one big file?**

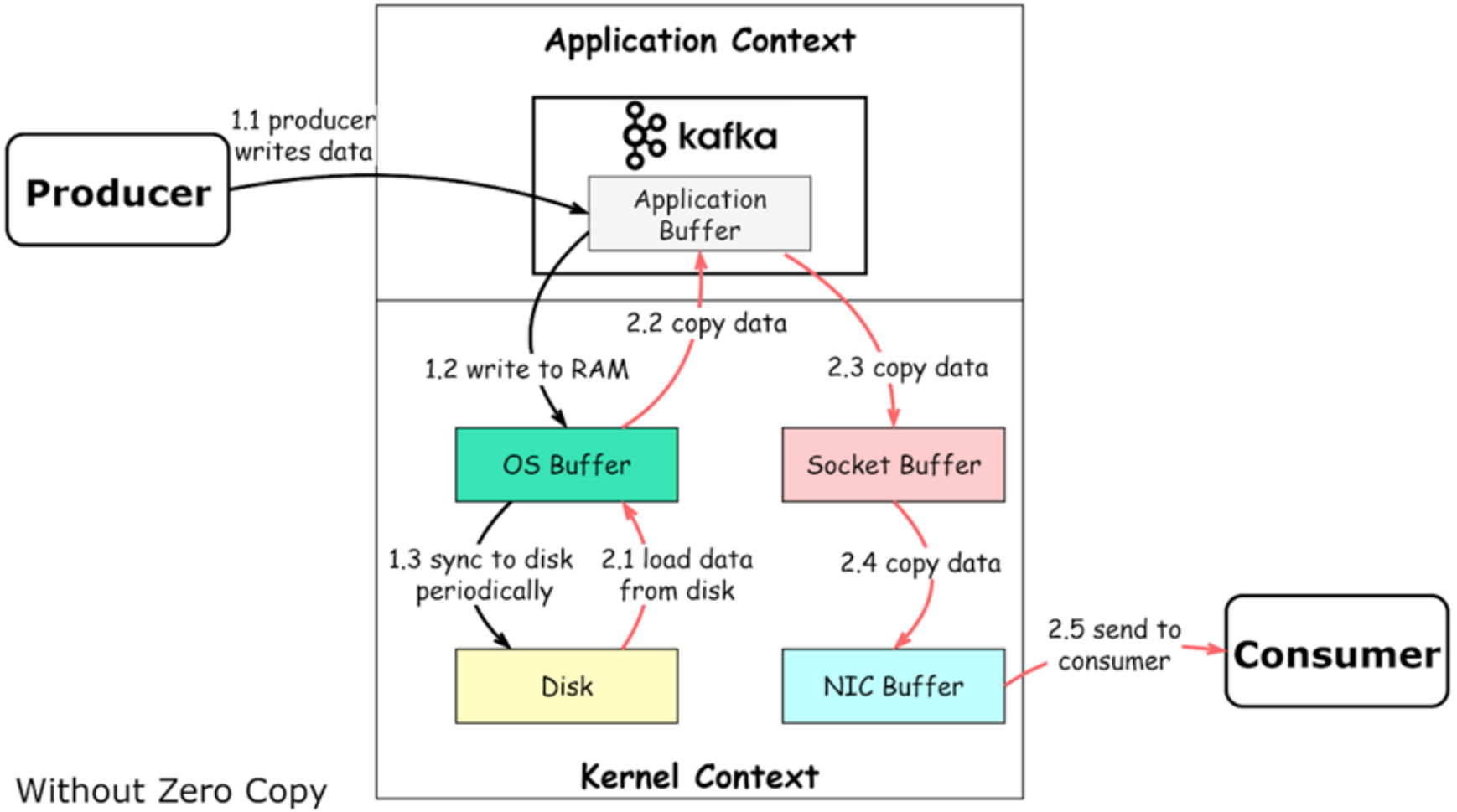
- Deletion is cheap
- Search is fast

**Consumer consumes messages from a partition sequentially**

- Asynchronous batched read
- Offset + acceptable number of bytes
- Recompute offset after each batch



# Normal Produce and Consume Operation

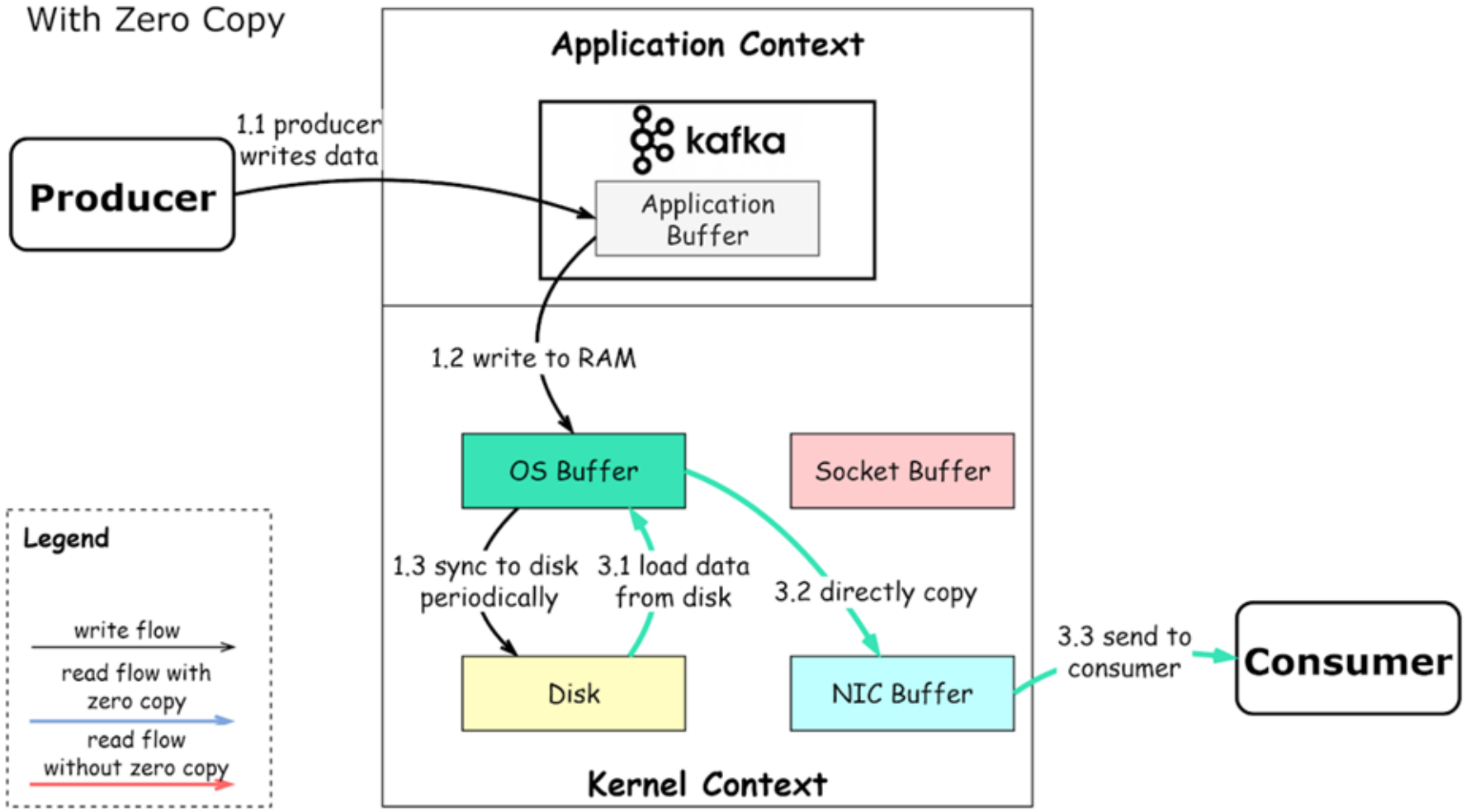



Without Zero Copy

# Latency Numbers

| Operation name | Time |
|---|---|
| L1 cache reference | 0.5 ns |
| Branch mispredict | 5 ns |
| L2 cache reference | 7 ns |
| Mutex lock/unlock | 100 ns |
| Main memory reference | 100 ns |
| Compress 1K bytes with Zippy | 10,000 ns = 10 µs |
| Send 2K bytes over 1 Gbps network | 20,000 ns = 20 µs |
| Read 1 MB sequentially from memory | 250,000 ns = 250 µs |
| Round trip within the same datacenter | 500,000 ns = 500 µs |
| Disk seek | 10,000,000 ns = 10 ms |
| Read 1 MB sequentially from the network | 10,000,000 ns = 10 ms |
| Read 1 MB sequentially from disk | 30,000,000 ns = 30 ms |
| Send packet CA (California) ->Netherlands->CA | 150,000,000 ns = 150 ms |

# Kafka Workflow



# Kafka Stateless Brokers

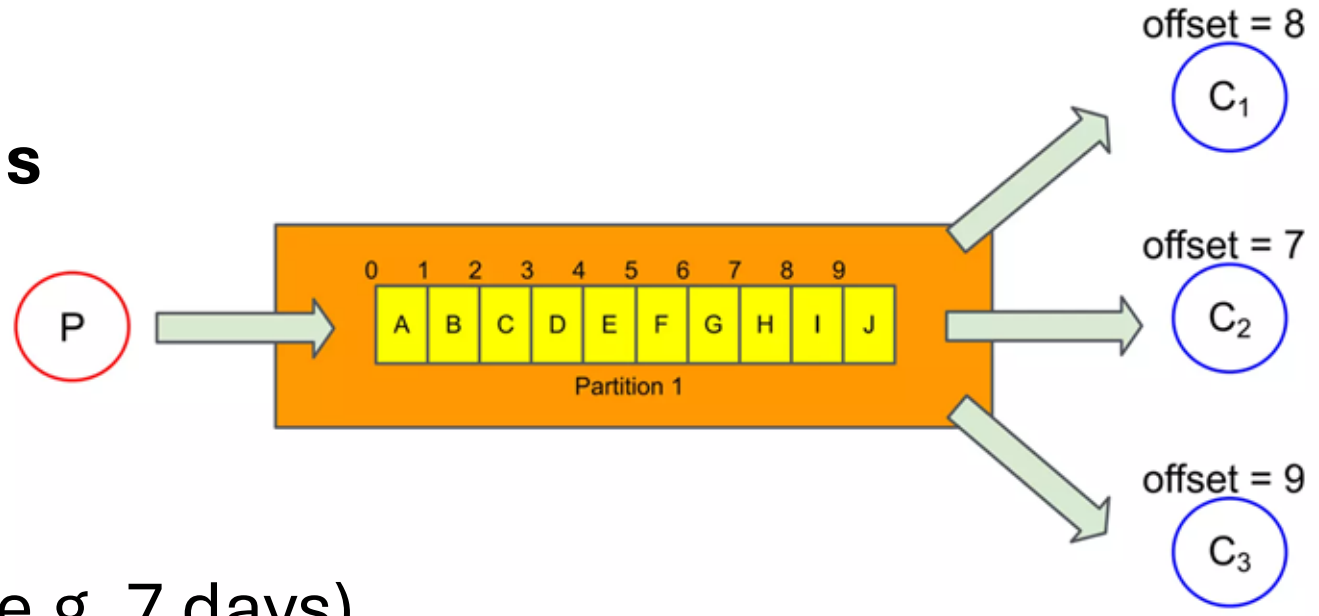
**Information of how much each consumer consumed is not maintained by Kafka brokers.**

**Consumer maintains its own status**

- Scalable
- Ability to rewind back

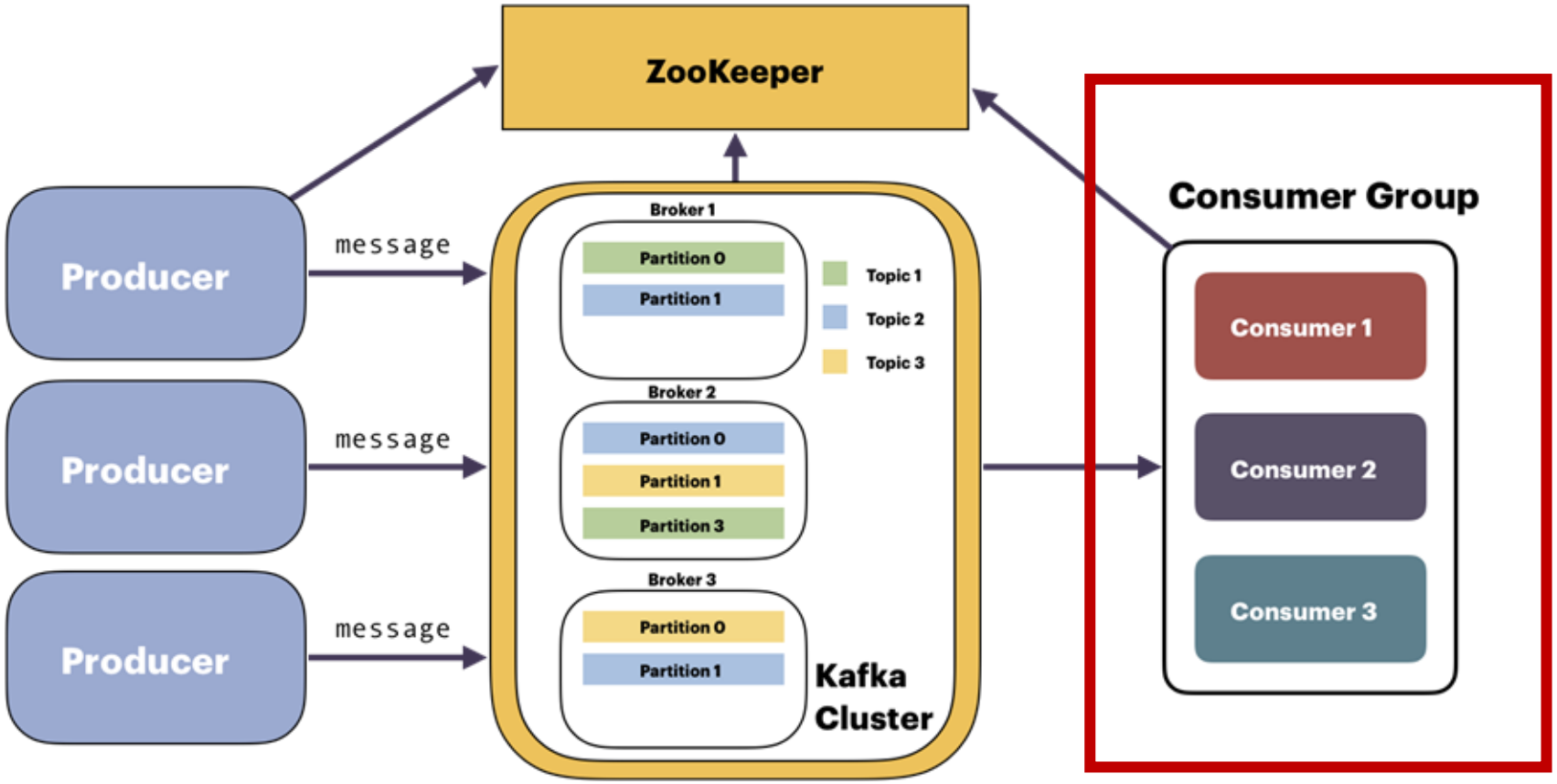
**How to delete msgs then?**

- Delete messages periodically (e.g. 7 days)




*Each Consumer keep its own offset.*

# Kafka Architecture



# Consensus Algorithms

## Load balancing problem

- How to spread work evenly across consumers without a central coordinator?
- Example: 10 consumers and 1 million message
- Coordination: "consumer 3 is done, give it more work”
  - Slow and complexity

## Kafka’s Insight:

- Bake the load balancing into the partition structure itself

# Consensus Algorithms

## Consumer group

- Consumers that share the same group ID form a consumer group
- One partition are consumed by on one consumer within each group
- "which consumer gets which message." → "which consumer owns which partition"

## How to assign partitions to consumers with the group?

- One consumer is elected **group leader**
- The group leader runs the partition assignment algorithm locally
- It sends the result back to the broker
- The broker distributes assignments to all consumers

# Consensus Algorithms

**Zookeeper (kv Store, zab protocol)**

**RAFT (Algorithm / Protocol)**

- KRAFT

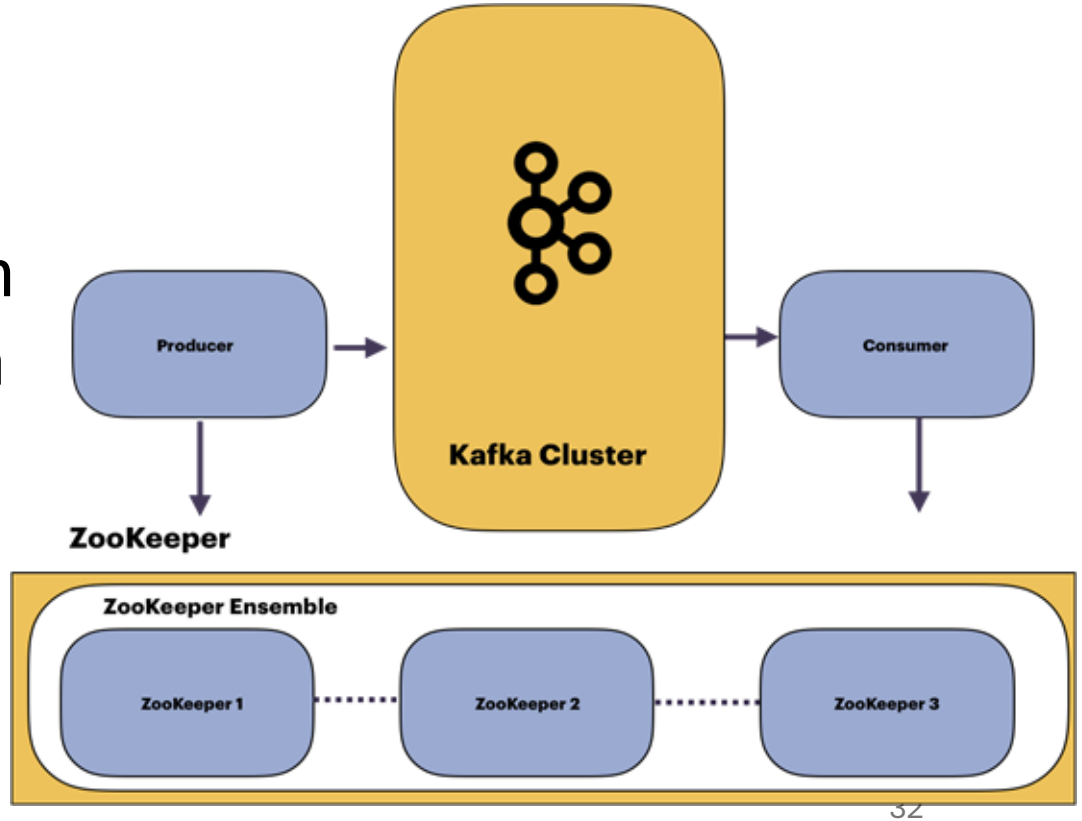
**PAXOS (Protocol)**

- Paxos Made Simple

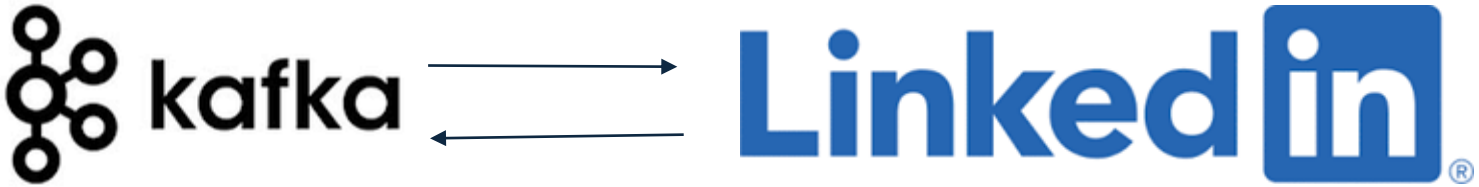
# Zookeeper

**Distributed consensus service, used for**

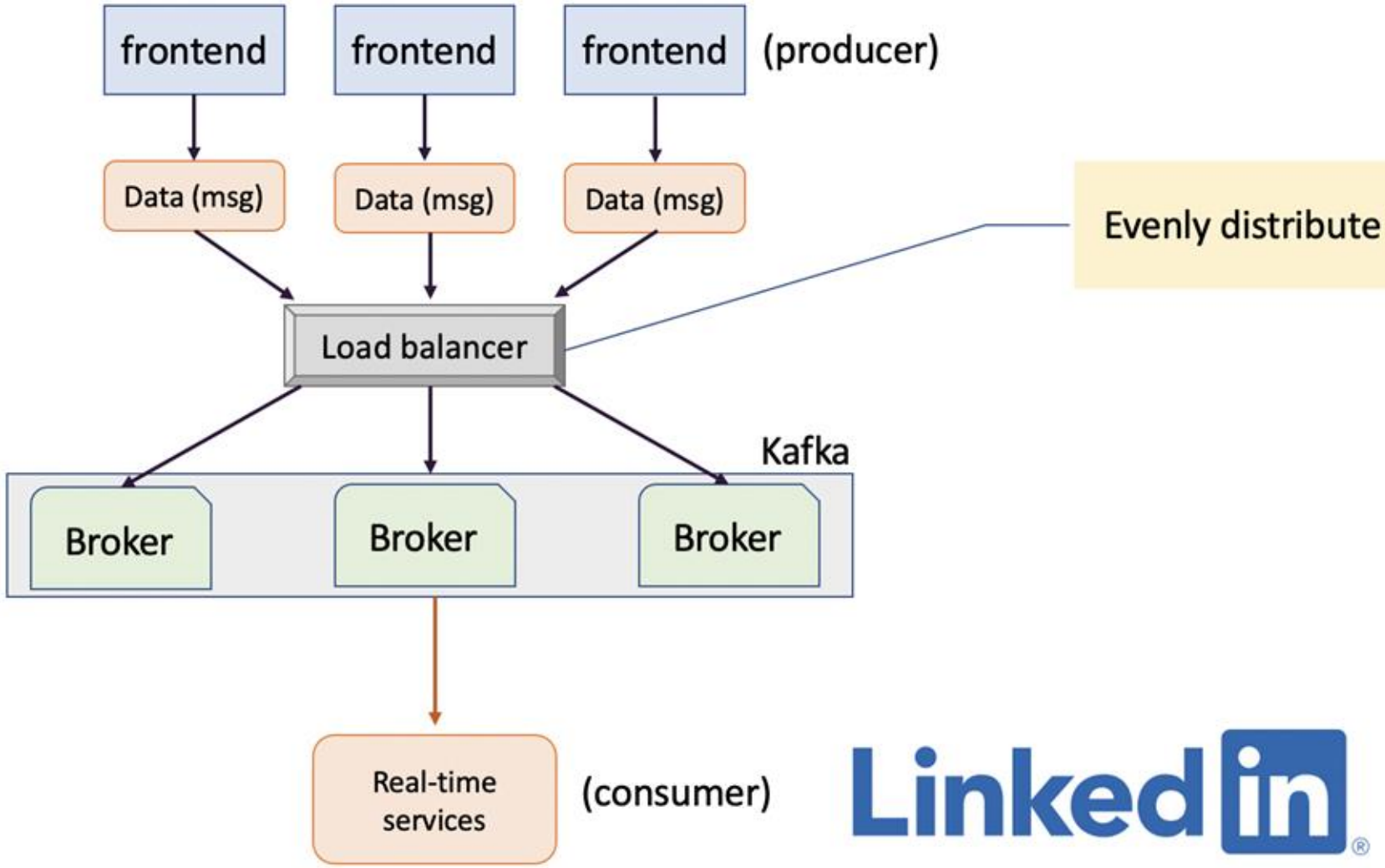
- Broker and consumer registry
  - Detect addition and removal of brokers
- Leader election for partition
- Consumer group coordination
- Topic and partition metadata
  - consumed offset for each partition
  - which broker is the leader for each partition.
  - ….



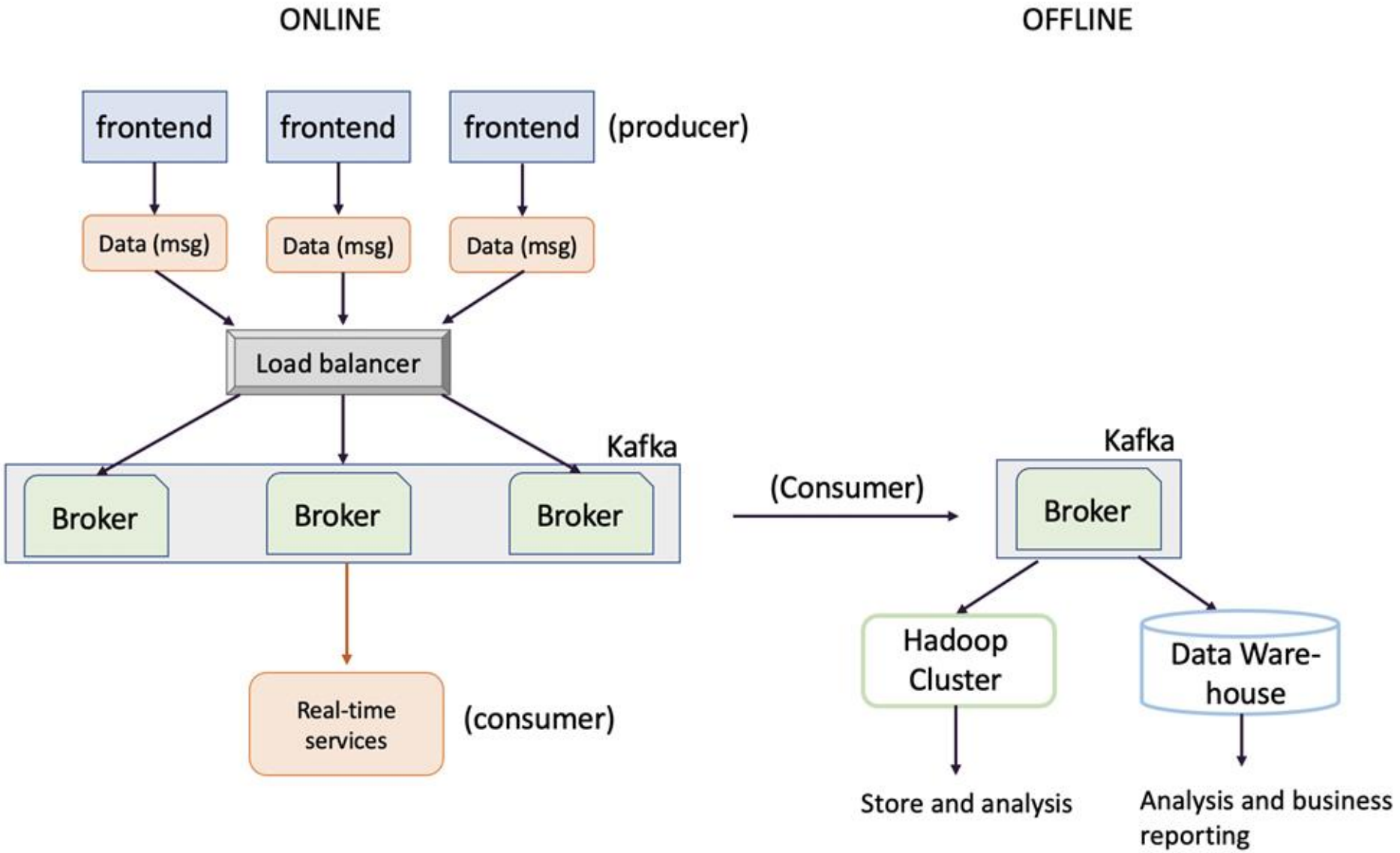
# Usage at LinkedIn



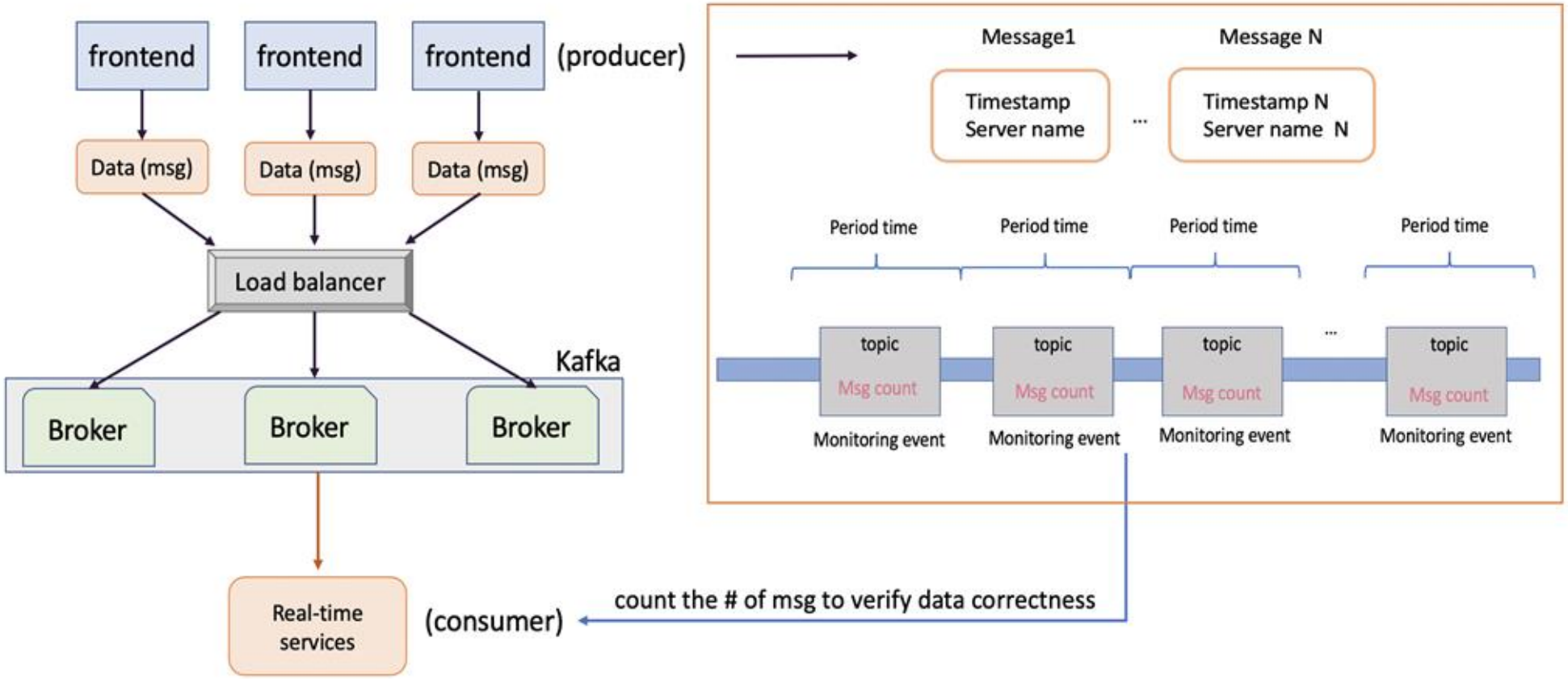
# Online Kafka



# Online Kafka



# Online Kafka



# Conclusion

**A novel system for processing huge volume of log data streams featuring:**

- High Throughput
- Scalability
- Permanent Store
- High Availability


**After paper, implemented**

- Replication of partitions for durability
- Delivery guarantees