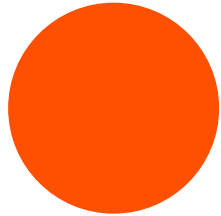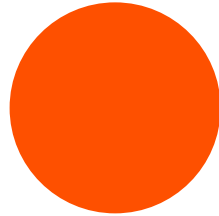# Distributed Messaging Systems

Ilyas Toumlilt
i.toumlilt@criteo.com
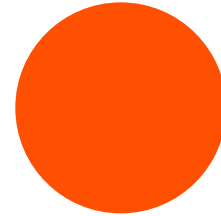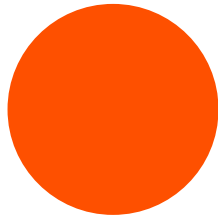
CRITEO

Session 2 – 08/03/2023

Session 1 Recap

Messaging Systems Overview

Apache Kafka

Lab: Getting started into Kafka

CRITEO

Session 1 Recap

CRITEO

Data-driven decision making

Information Theory: Data is a frozen information

Data drives decisions not every day, every second...

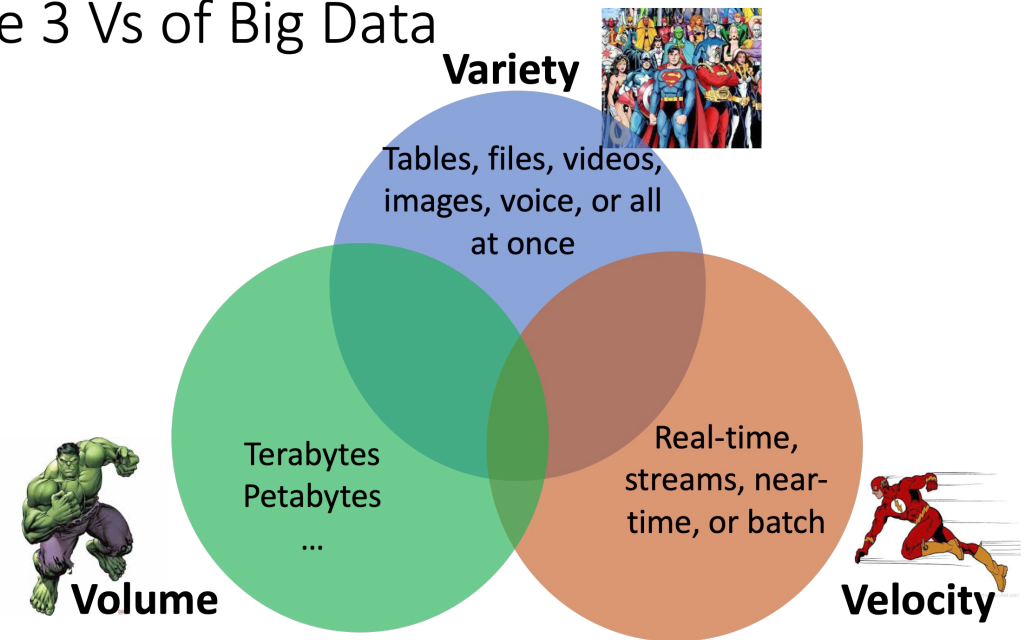Decisions could be made by humans or by software

CRITEO

Problems of Data-Intensive Distributed Computing

Reliability and Availability constraints

Scalability

Data locality

The programming model

The 3 Vs of Big Data

Variety

Tables, files, videos, images, voice, or all at once

Terabytes Petabytes ...

Volume

Real-time, streams, near-time, or batch

Velocity

CRITEO

Streaming 101

Streaming vs Batch Processing

Problems arise in streaming: *time, windows, state...*

Common distributed system problems: *Dumb ways to die*

CRITEO

Real classic problem

You are the owner of an application (imagine any app, remember flappy bird?) in production

It's tested, up and running with a workload of 100 users

Big buzz came and you are getting +100.000 active users

Your clients say the response time is increasing... 😕

Where to start looking?

CRITEO

Real classic problem

You are the owner of an application (imagine any app, remember flappy bird?) in production
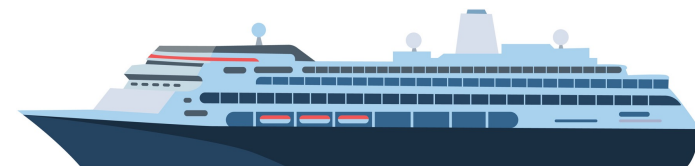
It's tested, up and running with a workload of 100 users

Big buzz came and you are getting +100.000 active users

Your clients say the response time is increasing... 😕

Where to start looking?

Resource Usage

CRITEO

Vertical scaling

The simplest approach to scale to higher load is to buy more powerful machines

- vertical scaling or scaling up

Many CPUs, many RAM chips, and many disks can be joined together under one operating system, and a fast interconnect allows any CPU to access any part of the memory or disk.

- In this kind of shared-memory architecture, all the components together can be treated as a single machine.

*Photo: Designed by pch.vector / Freepik*
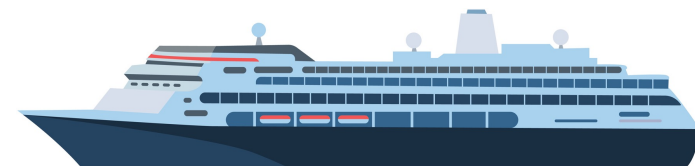
Vertical scaling

The problem with a shared-memory approach is that the cost grows faster than linearly

- A machine with twice as many CPUs, RAM, and disk capacity costs significantly more than twice the price.

- Due to bottlenecks, a machine twice the size cannot necessarily handle twice the load.

A shared-memory architecture may offer limited fault tolerance.

CRITEO

Horizontal scaling
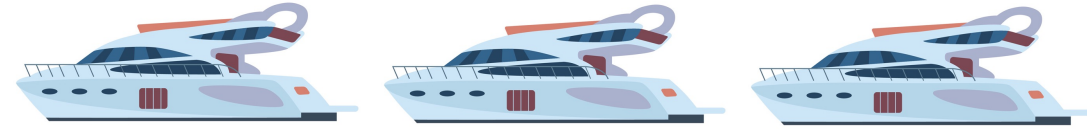
In a shared-nothing architecture multiple machines are used.

- horizontal scaling or scaling out

Each node uses its CPUs, RAM, and disks independently.

No special hardware is required.

Any coordination between nodes is done at the software level, through the network.
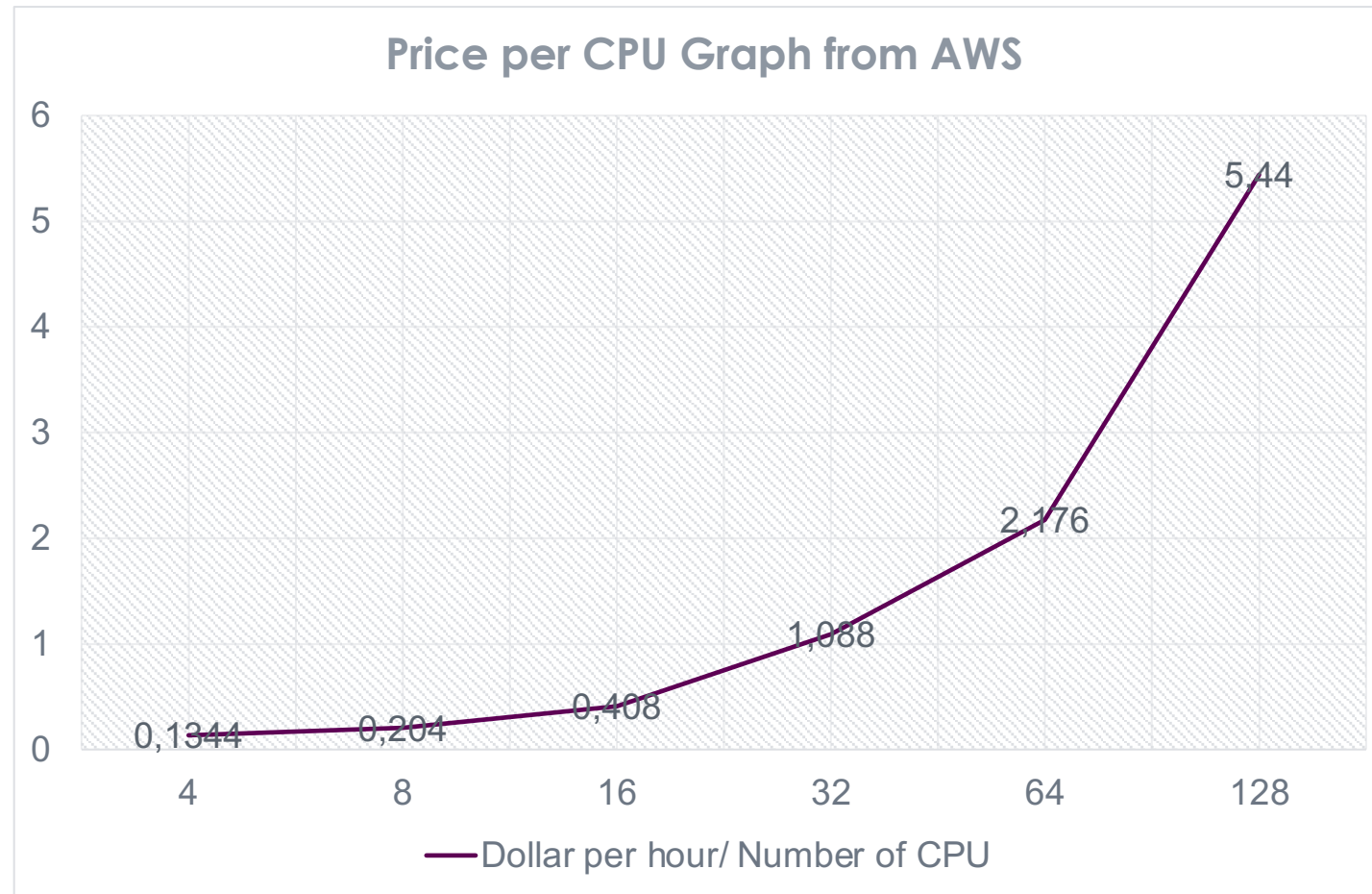
While a distributed shared-nothing architecture has many advantages, it adds more complexity.

More machines may lead to more failures too.

CRITEO

Distributed Systems

- There are several reasons why you might want to distribute your data across multiple machines:
  - Scalability
  - Fault tolerance/high availability
  - Latency
  - Price

CRITEO

What about pricing

Price per CPU Graph from AWS

0,1344 — 0,204 — 0,408 — 1,088 — 2,176 — 5,44

4   8   16   32   64   128

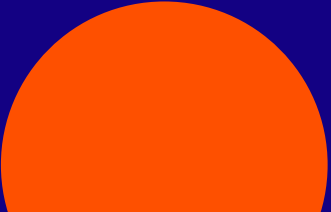— Dollar per hour/ Number of CPU

CRITEO

Replication Versus Partitioning

There are two common ways data is distributed across multiple nodes:

- Replication:
    - Keeping a copy of the same data on several different nodes, potentially in different locations.
    - Replication provides redundancy: if some nodes are unavailable, the data can still be served from the remaining nodes.
    - Replication can also help to improve performance(in general).
- Partitioning (*sharding*) e.g. partition by Country
    - Splitting a big data into smaller subsets called partitions so that different partitions can be assigned to different nodes.
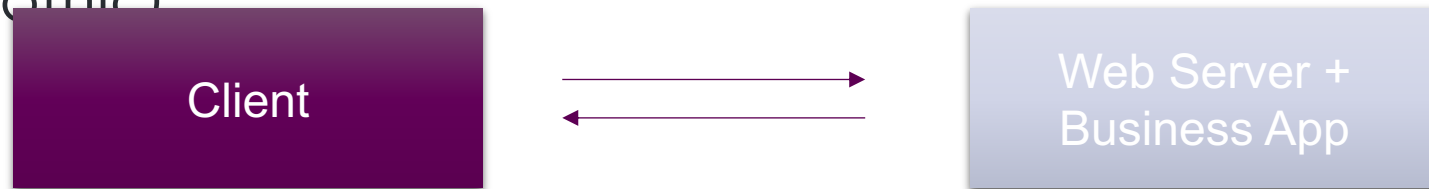
CRITEO

How to pass/deliver data in highly distributed environments ?

CRITEO

Replication Versus Partitioning

- Let's assume you are a new engineer in a start-up company, and the company provide a face detection software to their clients.

- Very very naive solution: one component does everything ? (Monolothic)

Client	→ ←	Web Server + Business App

What is wrong with it ?

CRITEO

Replication Versus Partitioning

- To reduce coupling, we must split core business logic from web server

Client		Web Server		Business App

2000 qps

1000 qps
Image
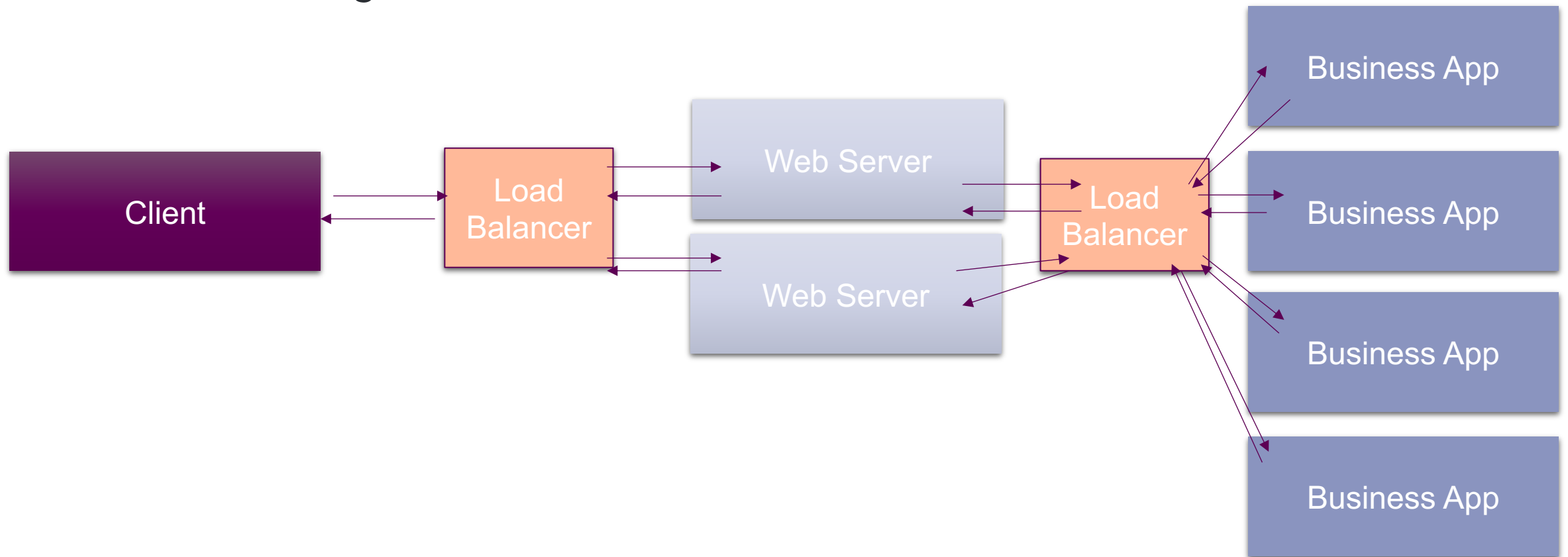processing
app
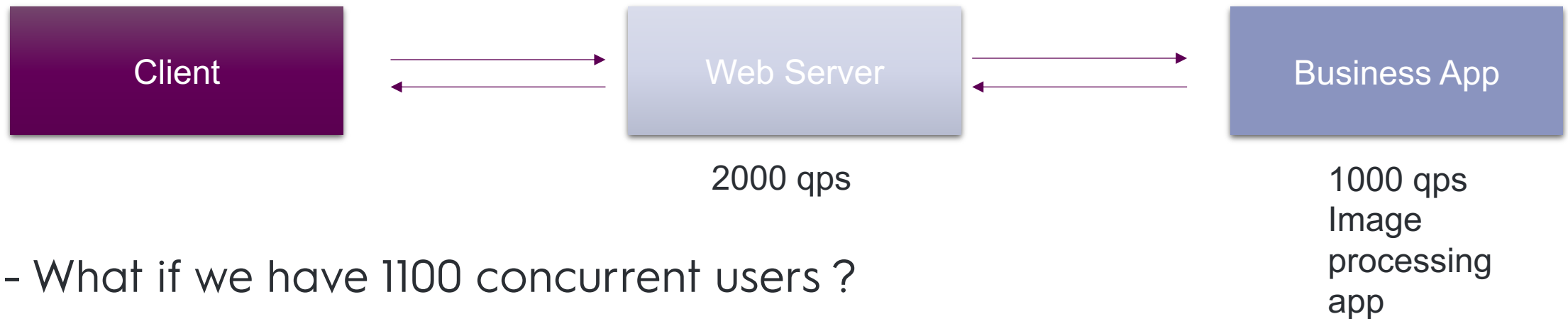
– What if we have 1100 concurrent users ?

Take home note: low coupling, high cohesion

CRITEO

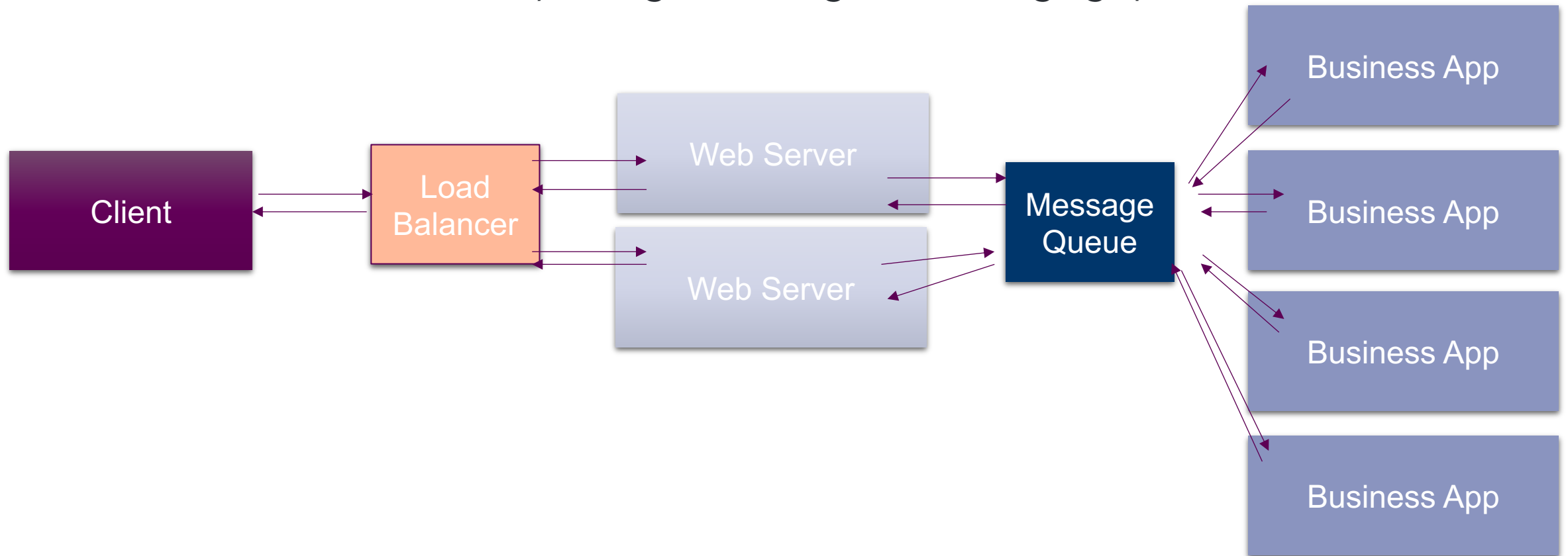# Replication Versus Partitioning

CRITEO
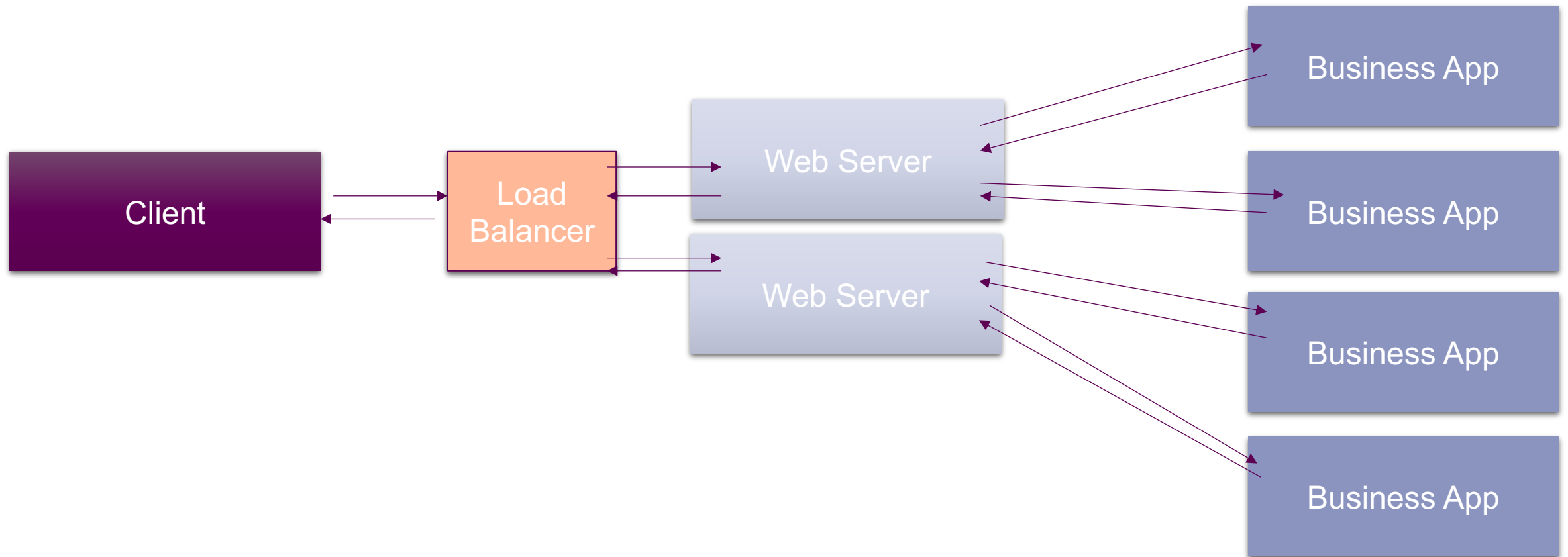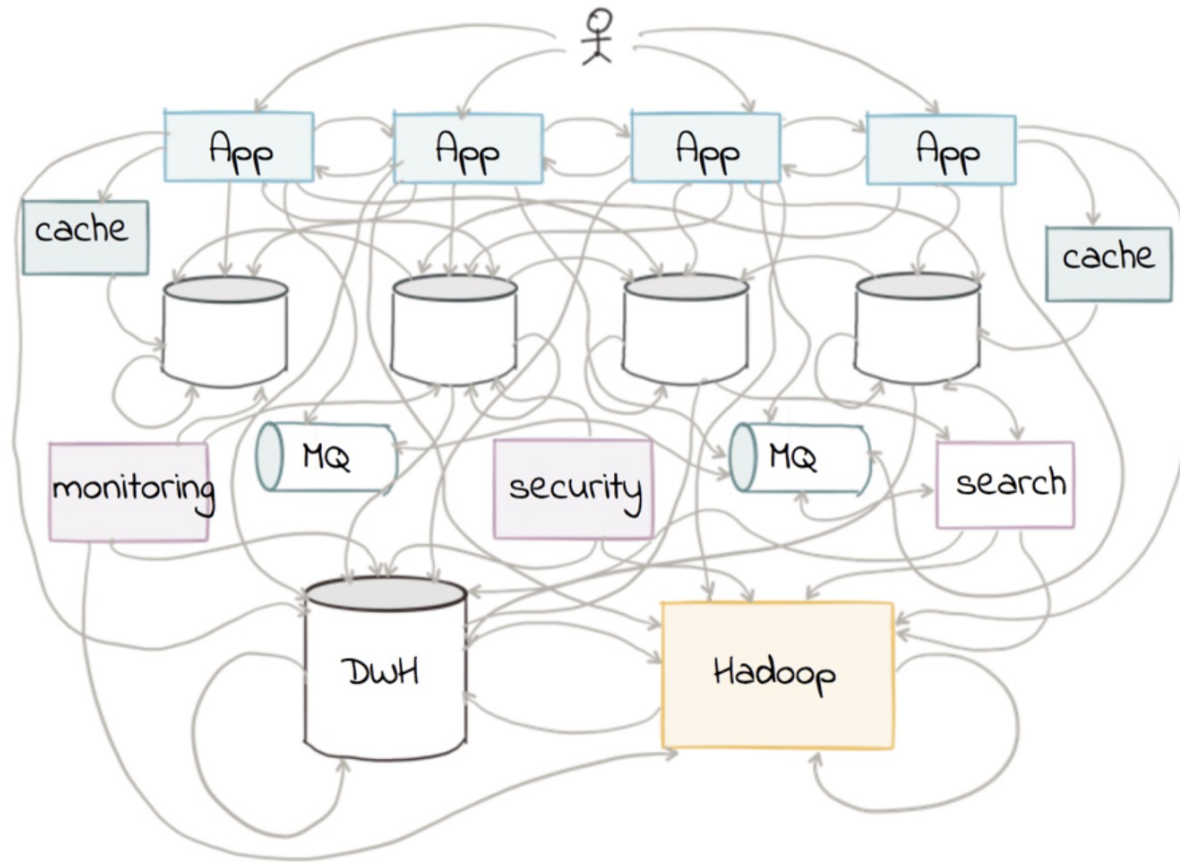
Replication Versus Partitioning

What is wrong ?

CRITEO

Replication Versus Partitioning

What can we achieve by taking advantage of messaging queue ?

CRITEO

Pub/Sub Systems

CRITEO

Pub/Sub Systems
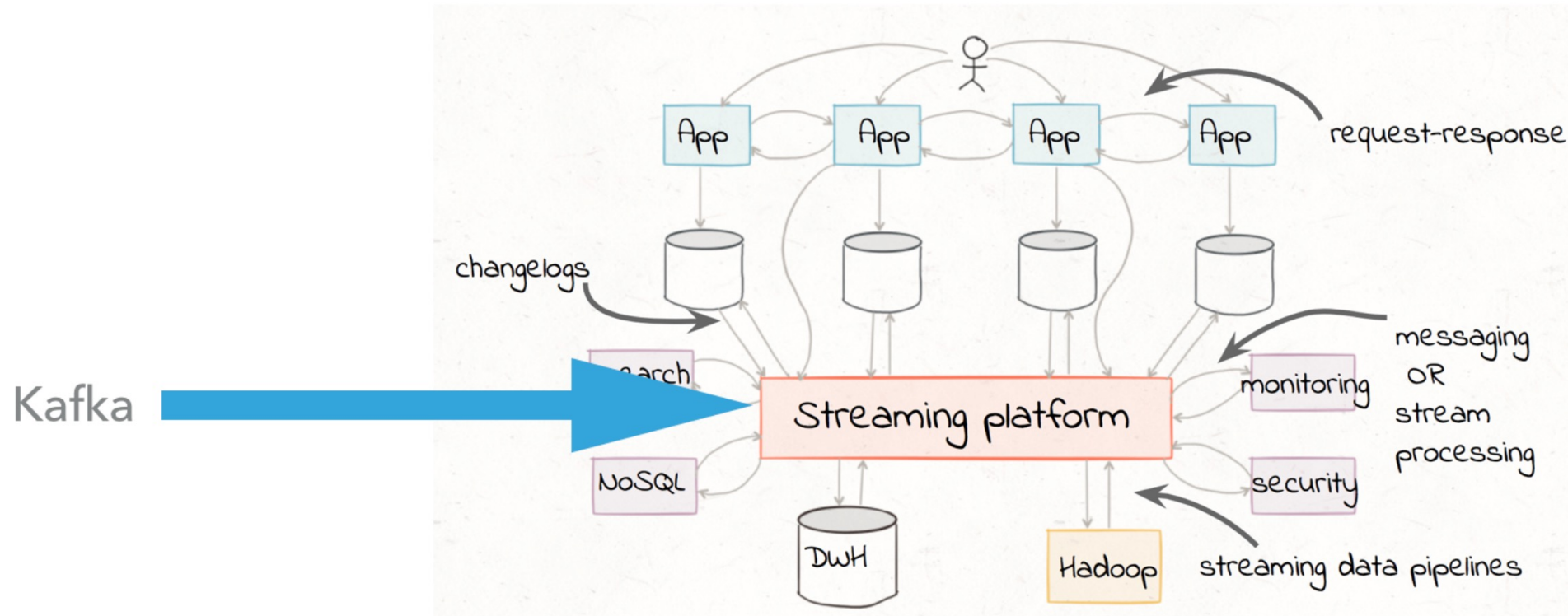
Messaging(Queue) Systems

- Publisher sends a piece of data (message) not specifically directing it to a receiver.
- The publisher classifies the message somehow, and that receiver (subscriber) subscribes to receive certain classes of messages.
- Pub/sub systems often have a broker, a central point where messages are published.

CRITEO
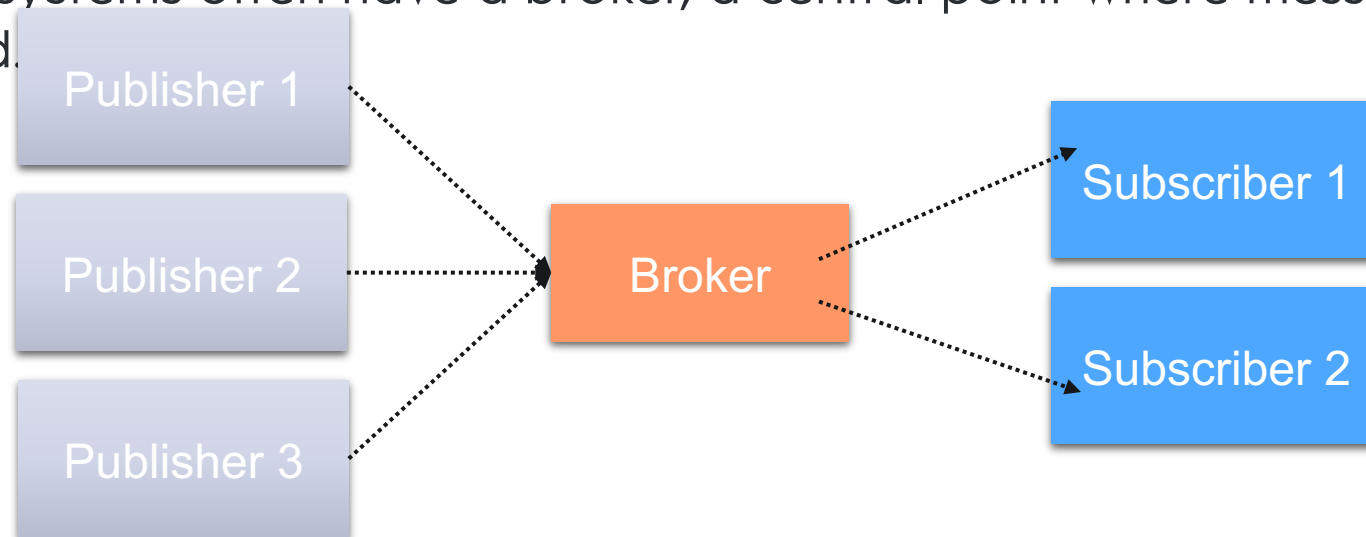
Messaging queue system (Pull vs Push based)

Push based

Pull based

Data flow

Request flow

CRITEO
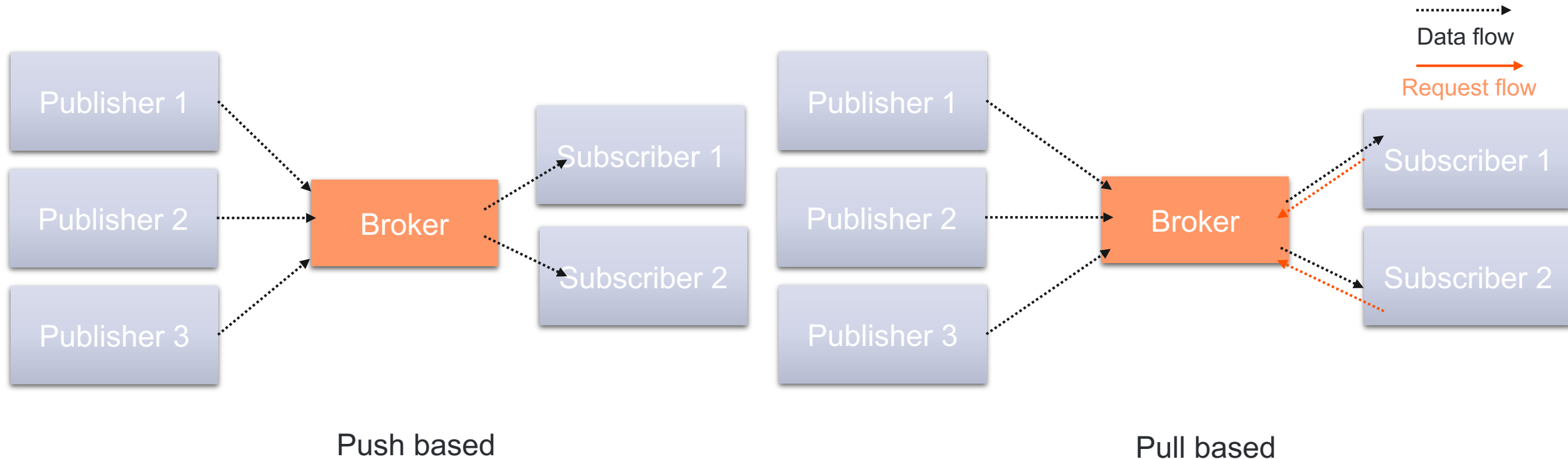
Apache Kafka

What is Kafka ?

Apache Kafka is a publish/subscribe distributed messaging system.

Also known as " distributed commit log" "distributing streaming platform".

Data stored in Kafka is durable, in order, and can be read deterministically.

Open-sourced by LinkedIn in 2011.

- High-throughput
- Highly distributed
- Fault-tolerant
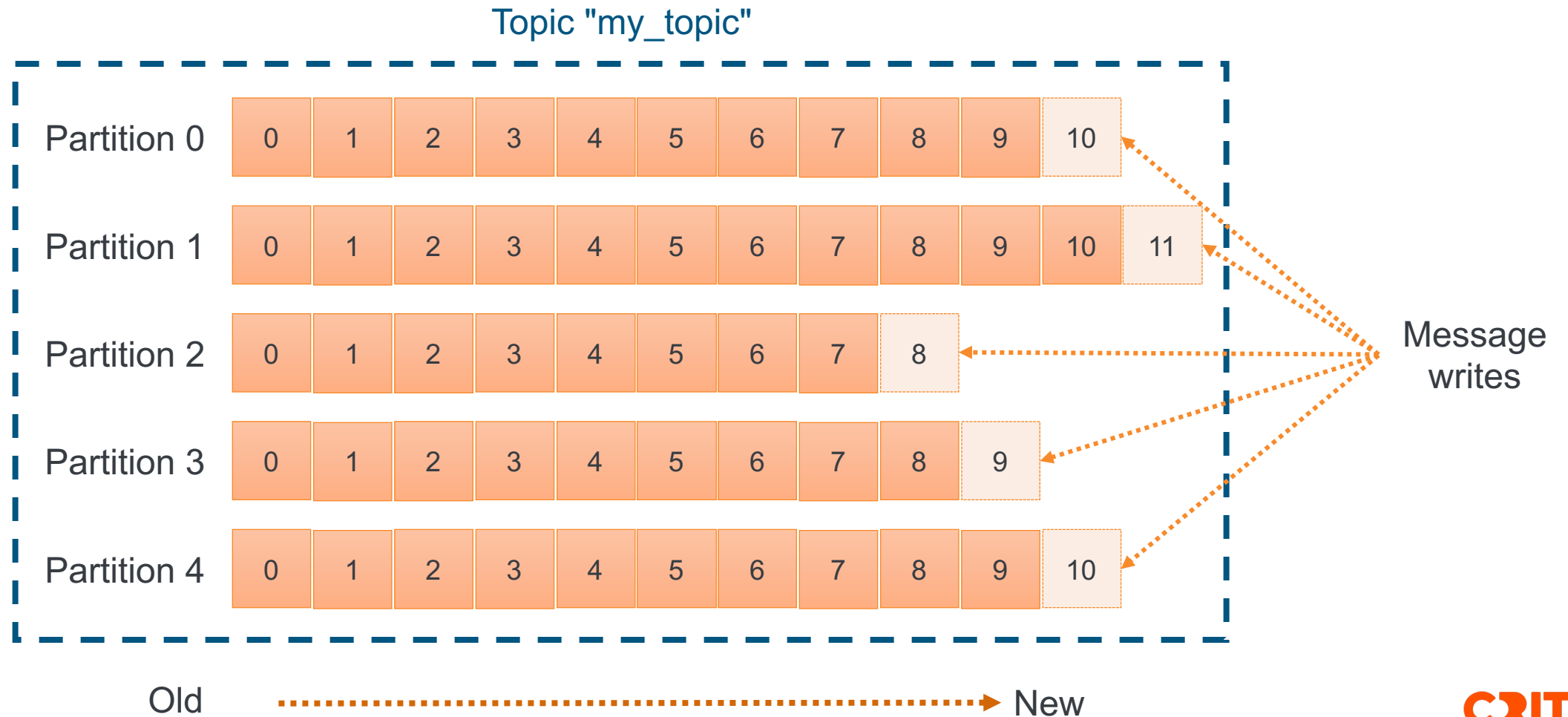- Low-latency

CRITEO

Anatomy of Kafka

- Message is the unit of data.

- Producer creates a new message

- Consumer (i.e., Subscribers/Readers)

- Messages in Kafka are categorized into *topics*.

- Topics are broken down into a *partitions*.

- Messages are persistant,

- and written in an append-only fashion.

CRITEO

Anatomy of Kafka

Topic "my_topic"

Message

- The unit of data within Kafka is a *message.*

- A message is simply an array of bytes.

- A message can have an optional *key* (also an array of bytes)*.*

- Messages contain a timestamp and optional headers.

- Messages are stored in order within a topic-partition.

- Once in Kafka topic, a message has an offset.

- For efficiency, messages are written into Kafka in batches(collection of messages).

offset	optional key/value headers	timestamp	key	value

CRITEO
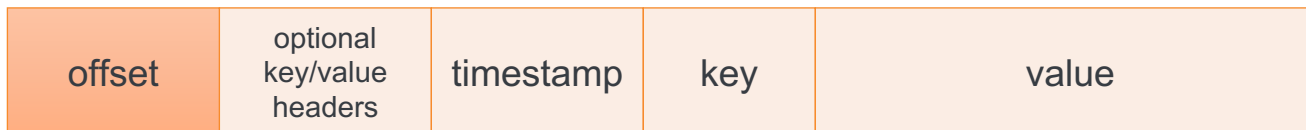
Topics and Partitions

- Messages sent by a producer to a particular topic partition will be append in the order they sent.

- However, there is no guarantee of message time-ordering across the entire topic.

- Partitions are the way that Kafka provides redundancy and scalability.

- Each partition can be hosted on a different server.

  - It means that a single topic can be scaled horizontally across multiple servers.

  - Partitions also define the level of paralelism a topic can support.

- Each partition has one leader and zero or more server follower servers.

CRITEO

Topics and Partitions

- Partitions are replicated across the cluster
- Each partition has one leader and zero or more server follower servers

```
bin/kafka-topics.sh --describe --bootstrap-server=localhost:9092 --topic xxx_yyy_zzz
```

Topic : xxx_yyy_zzz	Partition: 0	Leader: 19	Replicas: 19,20,17	ISR: 19
Topic : xxx_yyy_zzz	Partition: 1	Leader: 20	Replicas: 20,17,18	ISR: 18, 17, 20
Topic : xxx_yyy_zzz	Partition: 2	Leader: 17	Replicas: 17,18,16	ISR: 18, 16, 17
Topic : xxx_yyy_zzz	Partition: 3	Leader: 18	Replicas: 18, 16, 19	ISR: 18, 16, 19
Topic : xxx_yyy_zzz	Partition: 4	Leader: 16	Replicas: 16,19, 20	ISR: 16,19, 20

CRITEO

Kafka Broker

- A single Kafka server is called a broker.

- When interacting with producers:

  - The broker receives messages from producers;

  - Assigns offsets to them;

  - Replicates messages across other brokers;

  - Commits the messages to storage on disk.

- When serving consumers

  - It responds to fetch requests for partitions with the messages that have been committed.

CRITEO

Kafka Cluster

- Kafka brokers are designed to operate as part of a cluster.

- A partition is owned by a single broker in the cluster (*leader*).

- A partition can be replicated to other brokers in the cluster (*replicas*)

CRITEO
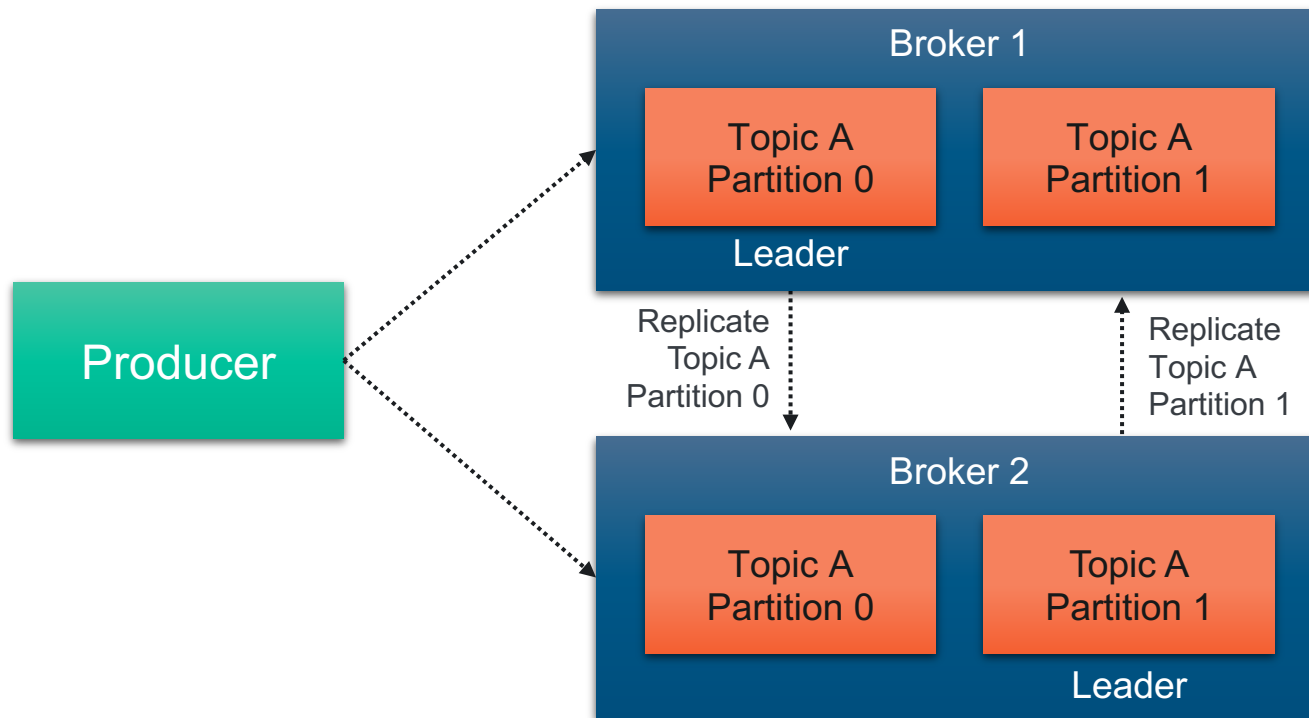
Kafka Producer

Producers decide to which partitions to send messages to.

Producers wait for the acknowledgment from the broker:

- ack=0
  - Doesn't wait for confirmation from the partition leader broker.
- ack=1
  - Only waits for the acknowledgment of the leader broker.
- ack=all
  - Waits for the acknowledgments of the leader broker and all replicas.

CRITEO
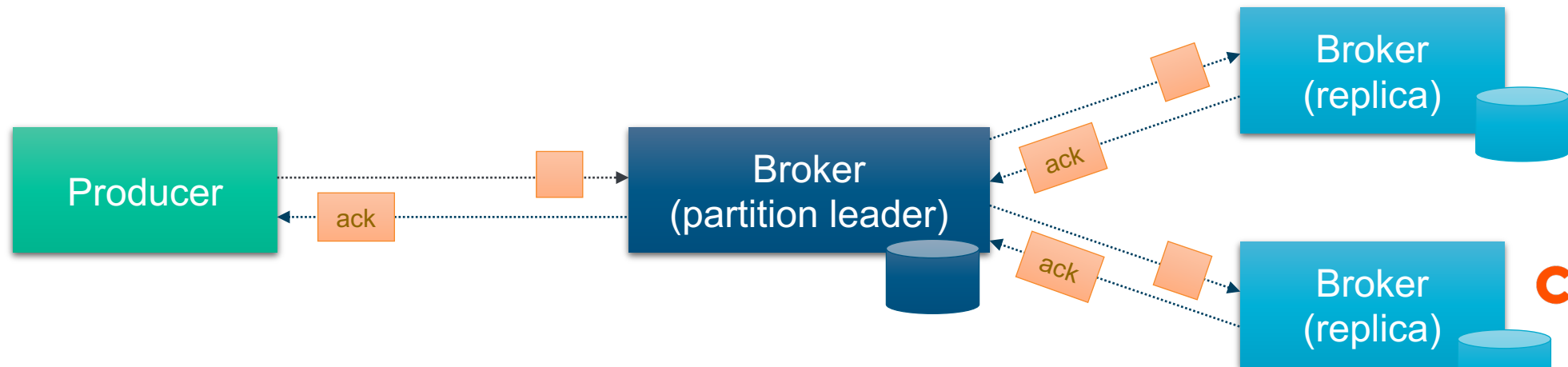
Kafka Consumer

- A client that consumes records from a Kafka cluster.

- Kafka consumers transparently handles the failure of Kafka brokers, and transparently adapts as topic partitions it fetches migrate within the cluster.

CRITEO
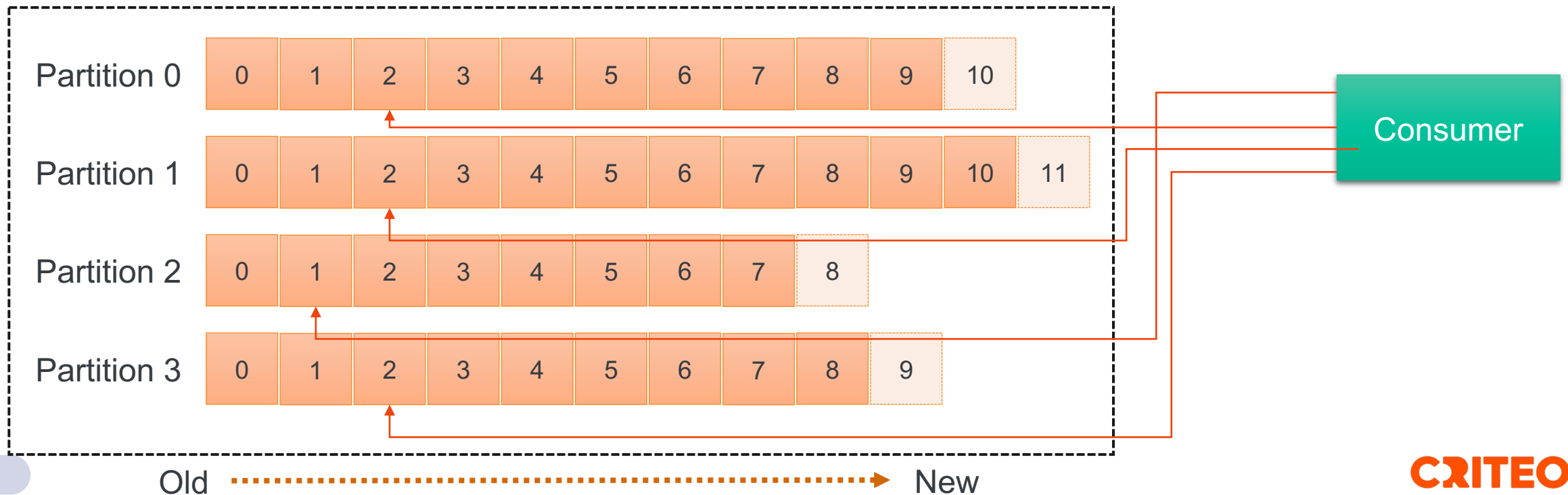
Consumer Group

- If all the consumer instances have the same consumer group, then this works as a queue balancer load over the consumers.

- If all the consumer instances have different consumer groups, then this works like publish-subscribe and all messages are broadcast to all consumers.
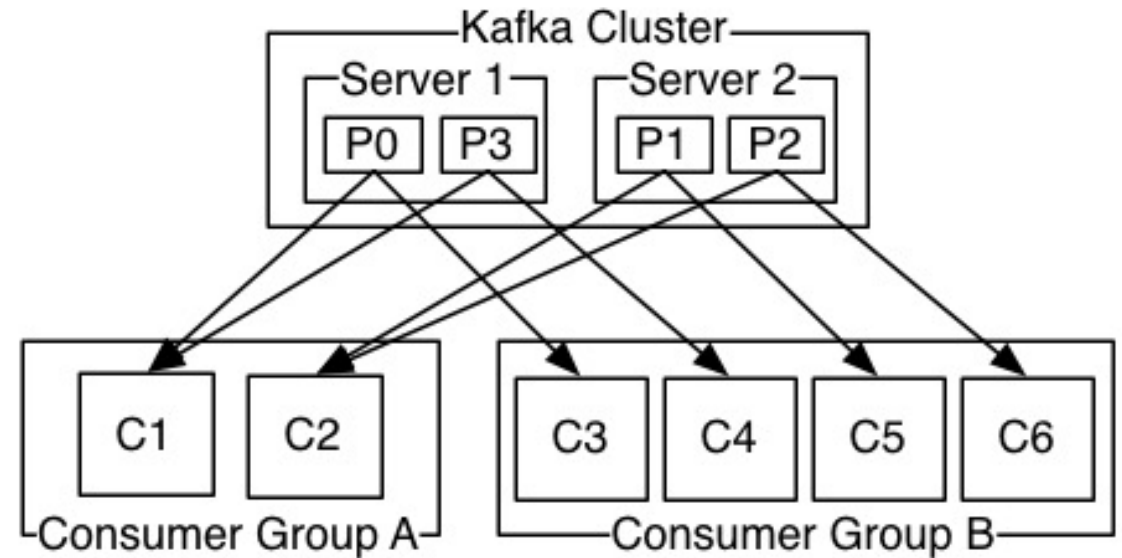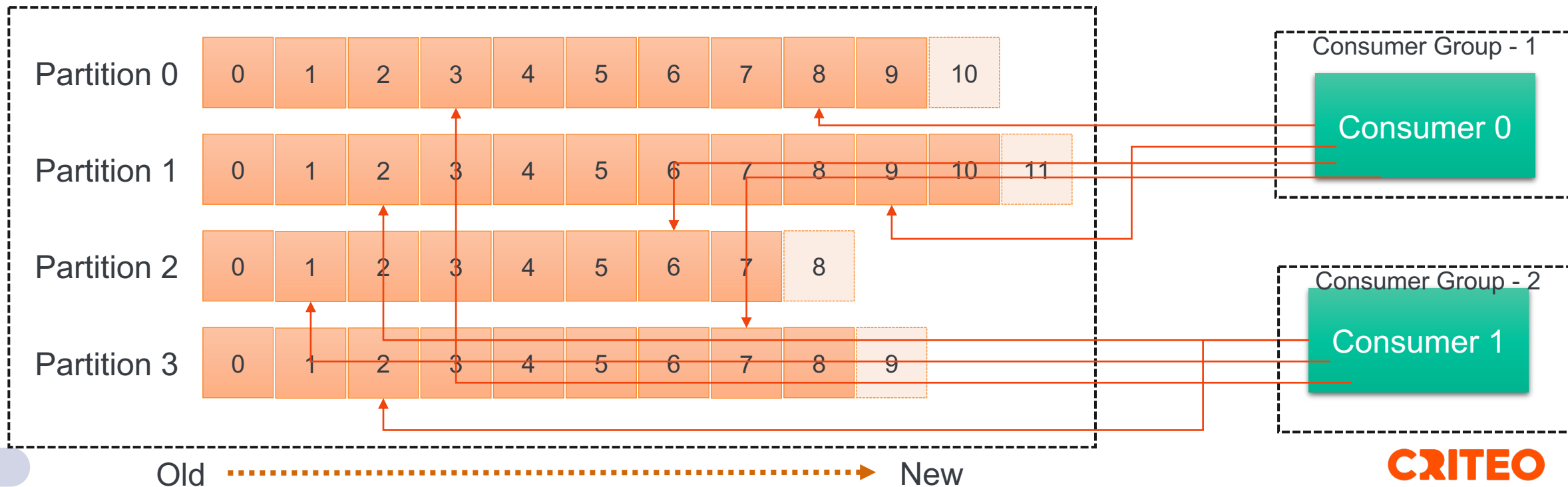
Kafka Consumer Group

- Each partition can be consumed by only one consumer within a consumer group.

- The consumers organize themselves to balance the load.
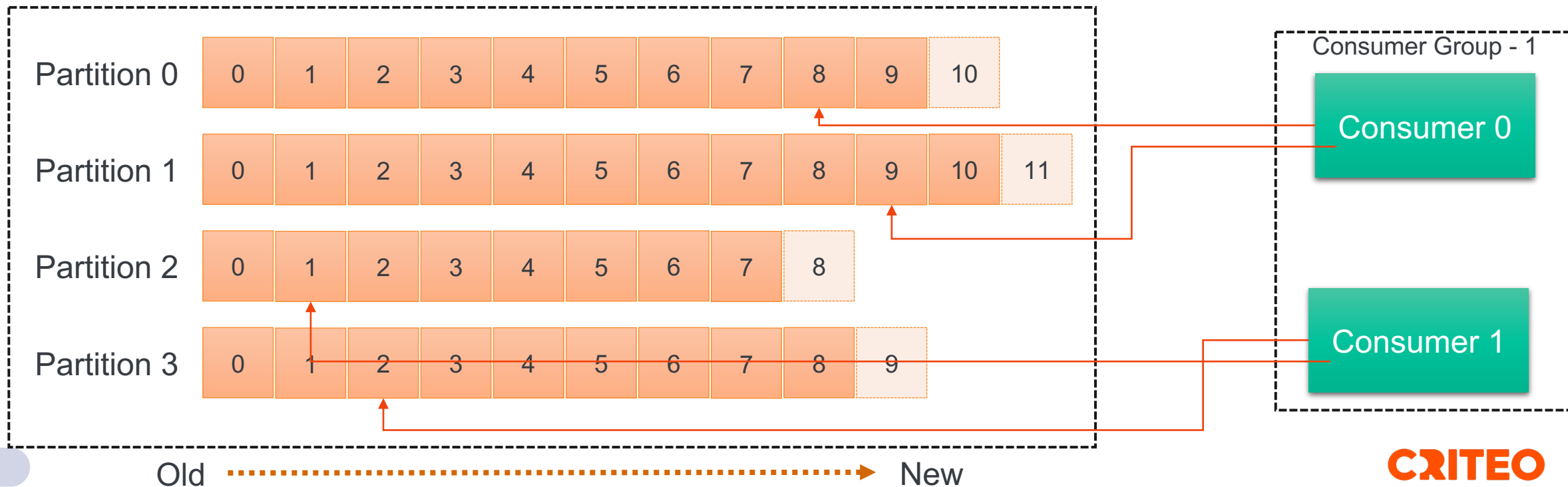
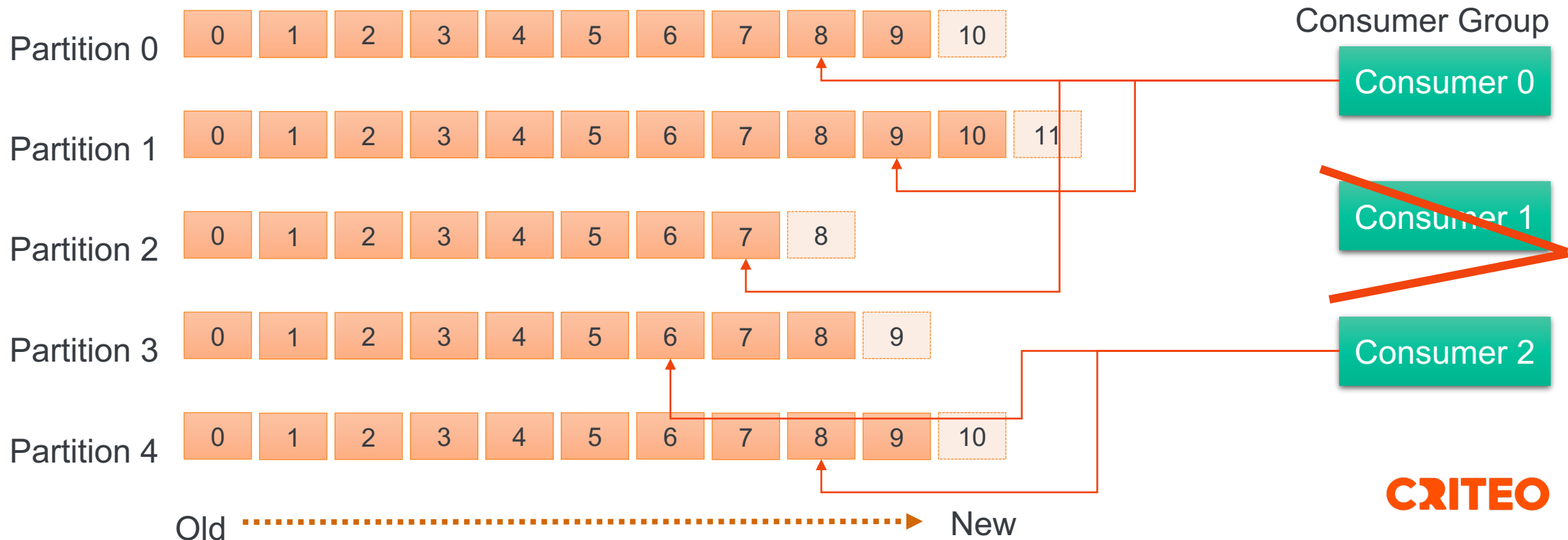# Kafka Consumer Group

- If each consumer in different group then they don't share workload starting from scratch.

CRITEO

Consumer Rebalance

- If a consumer is removed from the consumer group, the partitions from that consumer will be assigned to the remaining consumers.

- If a new consumer is added to the consumer group, it will receive partitions from other consumers, keeping a good balance.

Commiting Offsets

- If a consumer fails within a consumer group, another consumer should take place, starting from the same offsets where the failing consumer stopped.

- In order to save this state somewhere, consumers may send their offsets back to Kafka in a specific topic.

- In case of failure:

  - Commiting offsets after processing it may generates duplicates.

  - Commiting offsets before processing it, may end up in messages not processed.

CRITEO
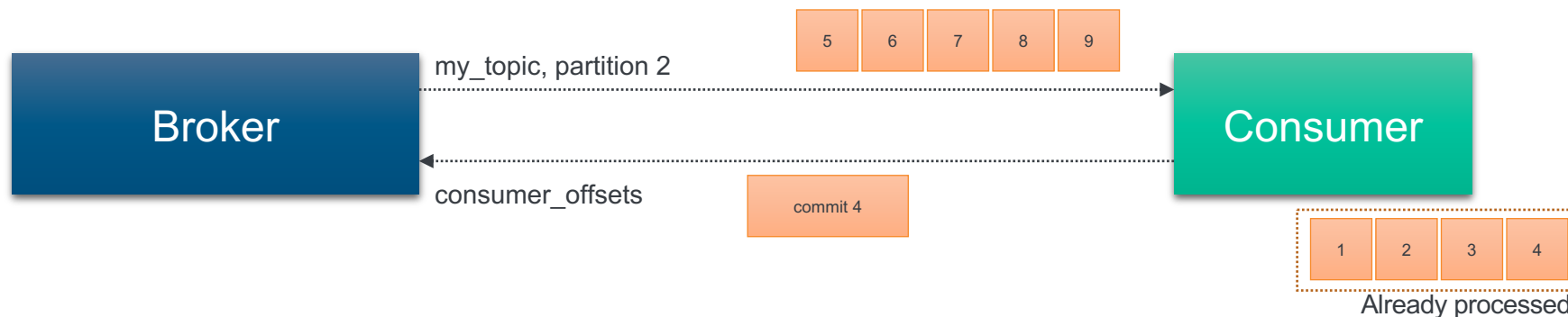
Kafka Guarantees

- Messages sent by a producer to a particular topic partition will be appended in the order they are sent.
  - That is, if a record M1 is sent by the same producer as a record M2, and M1 is sent first, then M1 will have a lower offset than M2 and appear earlier in the log.
- A consumer instance sees records in the order they are stored in the log.
- For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any records committed to the log.

CRITEO

Kafka Semantics

- ## At most once
  - When producers don't retry when ack times out or
  - an error is returned.
  - When consumers fail, they may skip some messages.

- ## At least once
  - When producers retry, they may send messages twice.
  - When consumers fail, they may reprocess messages twice.

- ## Exactly once
  - Kafka can deduplicate messages within the same partition (idempotent producer).
  - If consumers use a transactional sink, they may commit partitions within the transaction.

Hacker News
new | comments | show | ask | jobs | submit

▲ You Cannot Have Exactly-Once Delivery (bravenewgeek.com)
113 points by tylertreat 794 days ago | hide | past | web | 55 comments | favorite

Mathias Verraes
@mathiasverraes
👤+ Follow

There are only two hard problems in distributed systems: 2. Exactly-once delivery 1. Guaranteed order of messages 2. Exactly-once delivery

CRITEO
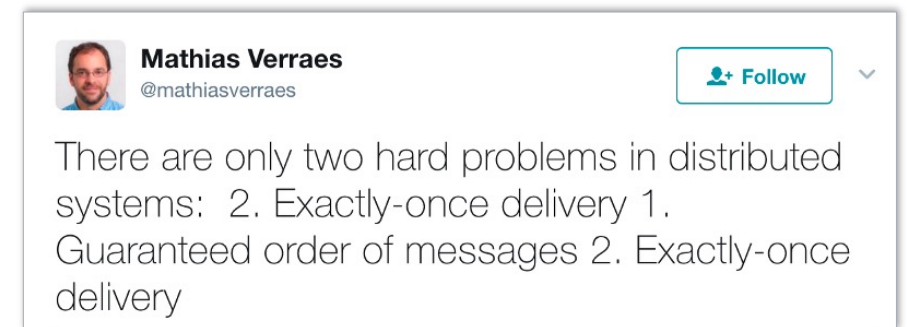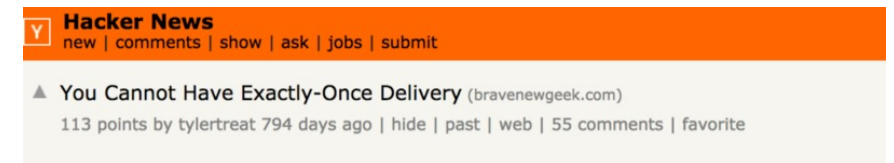
Retention

- Kafka organizes topic-partition data based on segments.

- It stores the segments for some period of time or until it reaches a certain size.

- Messages are discarded based on retention policy if they were consumed or not.
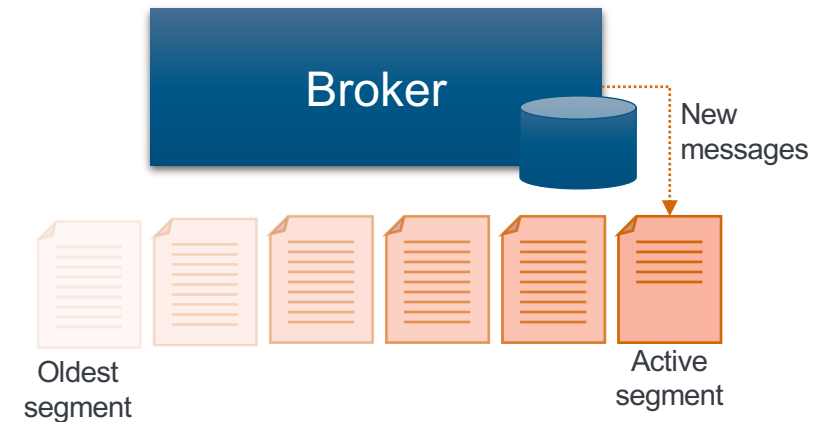
•**Property**:
log.retention.hours (Broker-wise)
retention.ms(Topic-wise)

log.retention.bytes (Broker-wise)
retention.bytes (Max size of a partition)

Broker

New messages

Oldest segment

Active segment

CRITEO

Zookeeper

- Apache Zookeper is a distributed coordination service for distributed applications.
- Zookeeper provides some guarantees:
  - Sequential Consistency
  - Atomicity
  - Single System Image
  - Reliability
  - Timeliness
- Kafka uses Apache Zookeeper to keep the cluster state:
  - List of currently members of a cluster;
  - List of topics, partitions leaders and replicas;
  - Current controller; etc.
- **Deprecated since Kafka 3.0.0, not required anymore**

CRITEO

Cap Theorem in context of Kafka

CAP Theorem(published in PODC '02) states that any distributed system can provide at most two out of the three guarantees: Consistency, Availability and Partition tolerance.
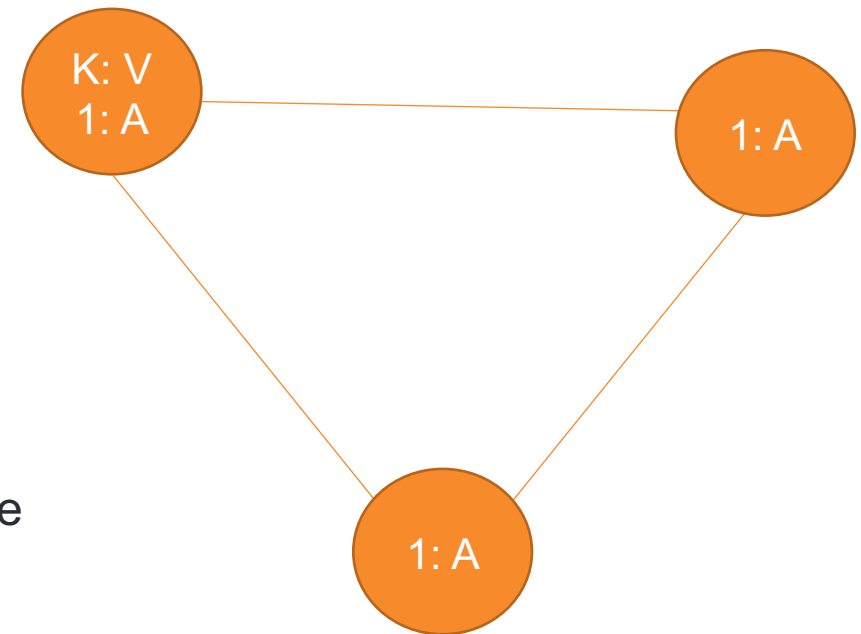
Partition tolerance**(*necessity)**
- The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Consistency
- Every read receives the most recent write or an error

Availability
- Every request receives a (non-error) response, without the guarantee that it contains the most recent write
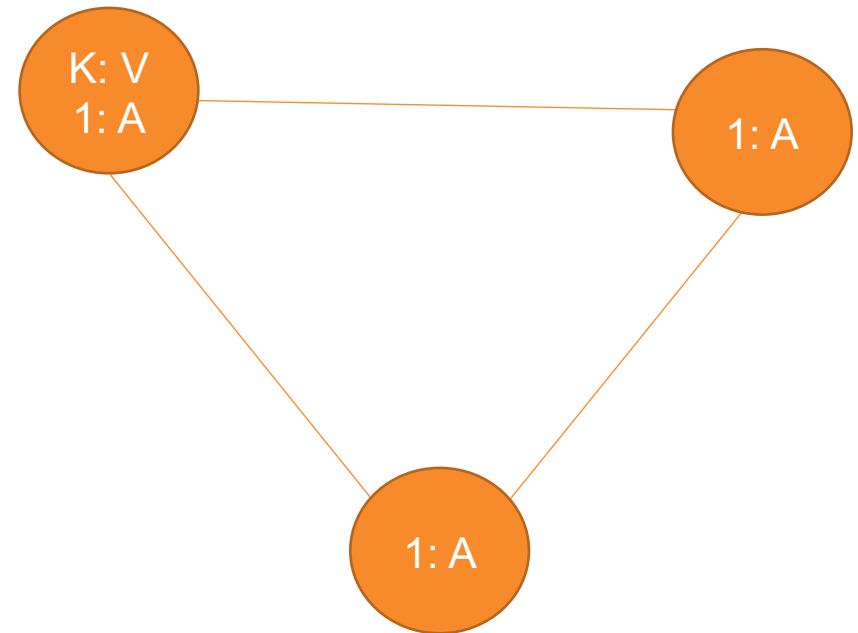
CRITEO

Cap Theorem on Kafka Brokers

For Consistency:

- Producers acks=all
- Replicator factor >= 3
- Disable unclean leader election
- Min insync replicas=replicatorFactor -1

For Availability:

- Enable unclean leader election
- Min insync replicas=1

What about a break? ☕

CRITEO

Practice time