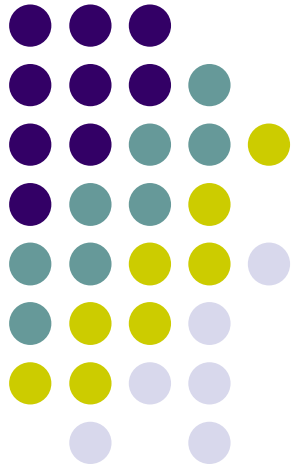
SIC

Serviços e Infraestruturas de Comunicação

Apache Kafka

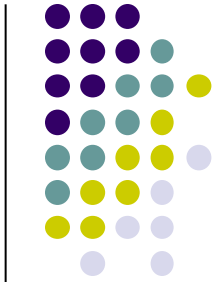


Credits

Several slides and figures in this presentation are based on the following materials:

- The Kafka materials, available at https://kafka.apache.org
- The Adam Shook's UMBC materials available at https://redirect.cs.umbc.edu/~shadam1/491s16/lectures
- The Sylvester Daniel tutorials provided at https://www.linkedin.com/pulse/kafka-consumer-overview-sylvester-daniel

Outlines

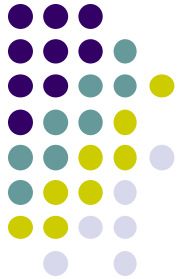


Apache Kafka is a publish-subscribe messaging platform, “rethought as a distributed commit log”

- Fast
- Scalable
- Durable
- Distributed

Event Streaming

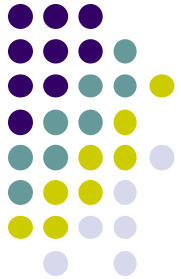
The digital equivalent of the human body's central nervous system



- *“Event streaming is the practice of capturing data in real-time from event sources like databases, sensors, mobile devices, cloud services, and software applications in the form of streams of events; storing these event streams durably for later retrieval; manipulating, processing, and reacting to the event streams in real-time as well as retrospectively; and routing the event streams to different destination technologies as needed.”* (from: https://kafka.apache.org/intro)

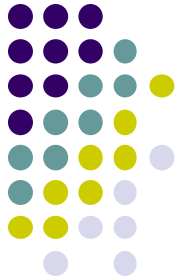
- Event streaming ensures a continuous flow and interpretation of data, so that the right information is at the right place, at the right time.

Applications of event streaming



- Processing of payments and financial transactions in real-time.
- Tracking and monitoring cars, trucks, fleets and shipments in real-time, (e.g., logistics and automotive industry).
- Continuous capture and analysis of sensor data from IoT devices or other equipment, such as in factories and wind parks.
- Collection and immediate reaction to customer interactions and orders (e.g., retail, hotel and travel industry, mobile applications).
- To monitor patients in hospital care and timely predict changes in condition.
- To connect, store, and make available data produced by different divisions of a company.
- To serve as the foundation for data platforms, event-driven architectures, and microservices.

What is the Kafka event streaming platform?



Kafka combines three key capabilities:

- To **publish (to *write*) and subscribe (to *read*) streams** of events, including continuous import/export of data from other systems.
- To **store streams** of events durably and reliably, for as long as deemed necessary.
- To **process streams** of events as they occur or retrospectively.

All this functionality is provided in a distributed, highly scalable, elastic, fault-tolerant, and secure manner.

Kafka can be deployed on bare-metal hardware, virtual machines, and containers, and on-premises as well as in the cloud.

What is the Kafka event streaming platform?

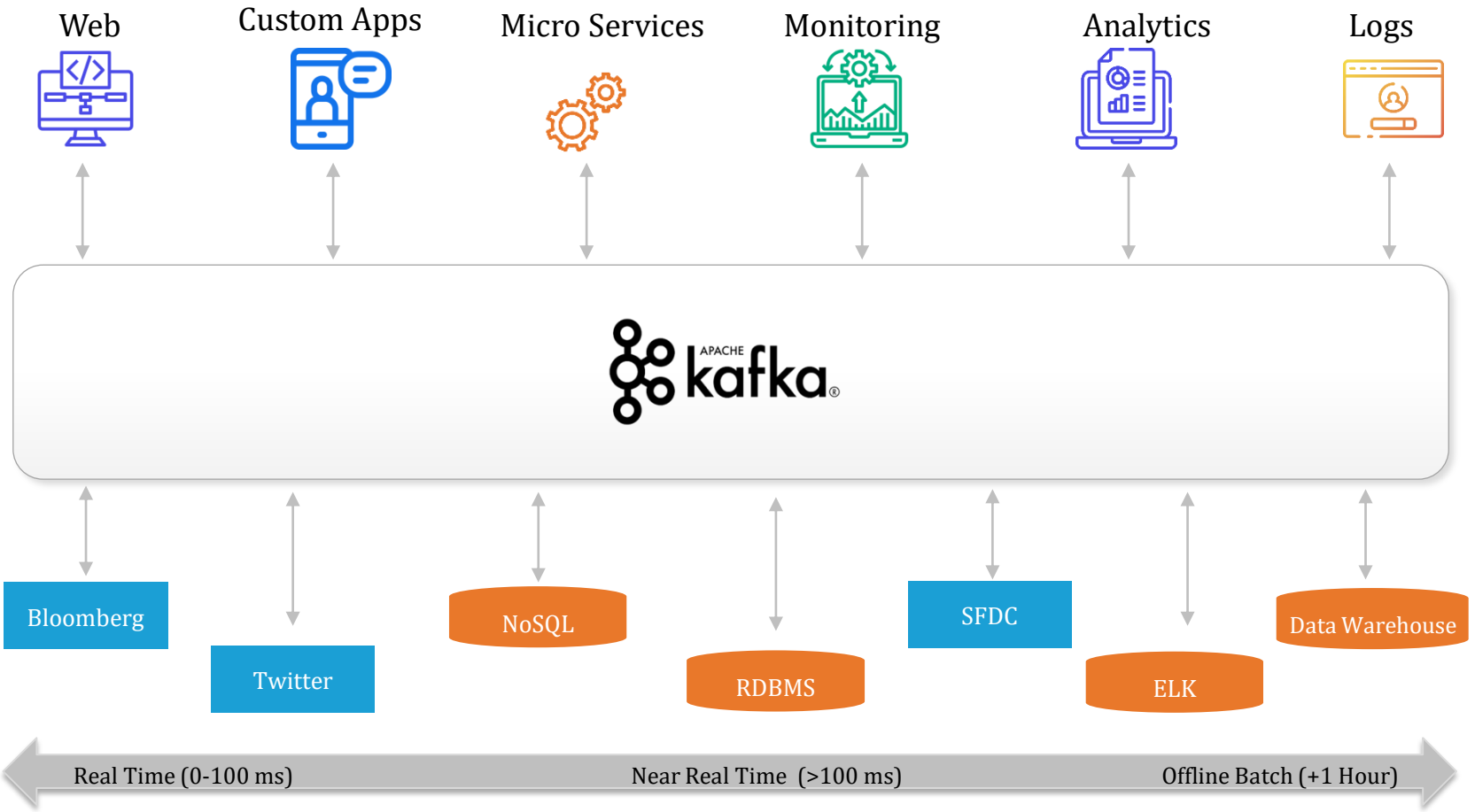


Image source: https://www.cignex.com/file/24702/download?token=00dlmy09

Kafka adoption and use cases

- **LinkedIn:** 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
- Airbnb, Cisco, Gnip, InfoChimps, Ooyala, Square, Uber, …

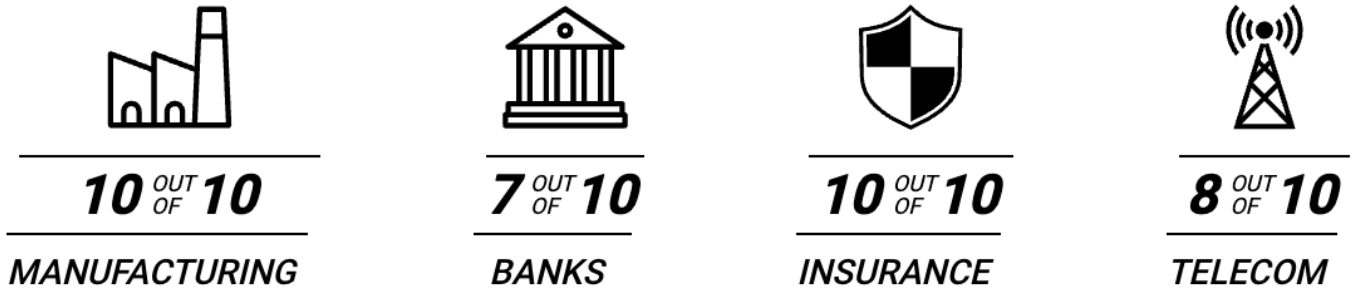


Image source: https://kafka.apache.org/

How does Kafka work?
Servers and Clients

Servers:

- Kafka is run as a cluster of one or more servers that may span multiple datacenters or cloud regions.
- Two types of servers:
  - *Broker servers* form the storage layer.
  - *Connect servers* continuously import and export data as event streams, to integrate Kafka with existing systems such as relational databases.
- Kafka clusters are highly scalable and fault-tolerant → If a server fails, the others will take over their work to ensure continuous operations with no data loss.

How does Kafka work?
Servers and Clients

Clients:

- Allow to write distributed applications and microservices that read, write, and process streams of events in parallel, at scale, and in a fault-tolerant manner.
- Kafka ships with some such clients included, which are augmented by dozens of clients provided by the Kafka community: there are clients for Java and Scala including the higher-level Kafka Streams library, for Go, Python, C/C++, and many other programming languages as well as REST APIs.

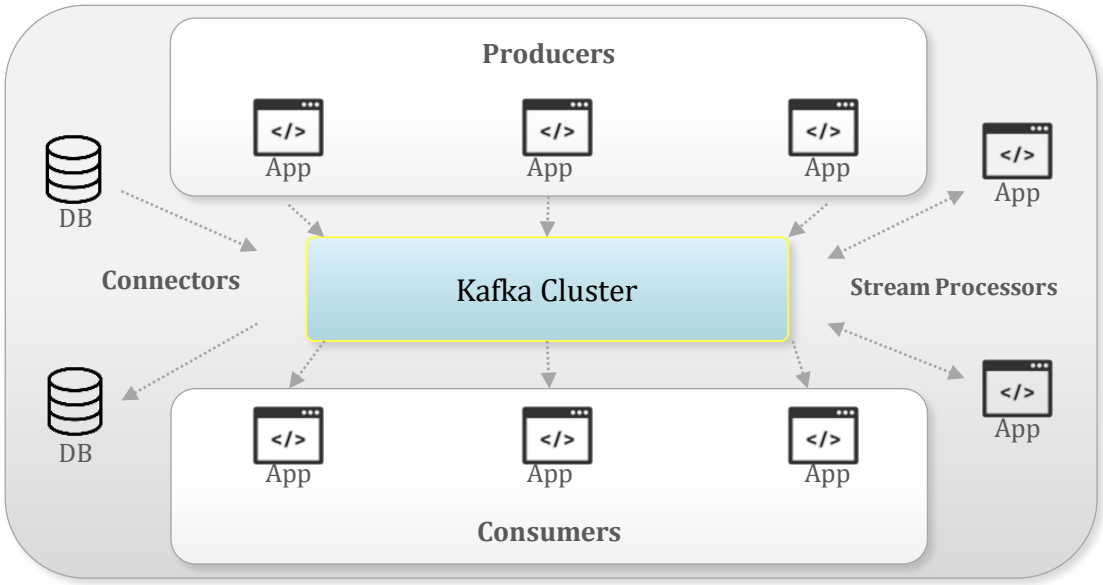


Image source: https://www.cignex.com/file/24702/download?token=00dImy09

Kafka main concepts
Events, Producers, Consumers

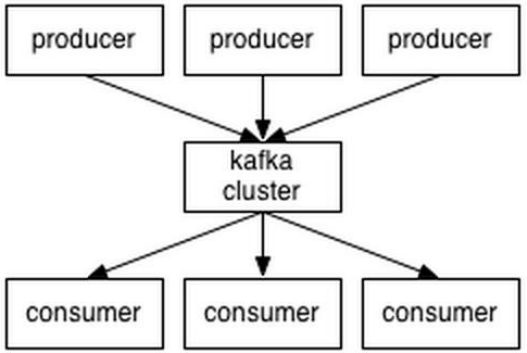
Events record the fact that "something happened". Data is read or written to Kafka in the form of events. Conceptually, an event has a key, value, timestamp, and optional metadata headers. For instance:

- Event key: "Alice"
- Event value: "Made a payment of $200 to Bob"
- Event timestamp: "Jun. 25, 2020 at 2:06 p.m."

Producers are client applications that publish (write) events to Kafka.

Consumers are clients that subscribe to (read and process) events.

- In Kafka, producers and consumers are fully decoupled and agnostic of each other, which is a key design element to achieve the high scalability that Kafka is known for.
  - Slow consumers do not affect fast producers
  - Adding more consumers does not affect producers
  - Failure of consumers does not affect producers



Kafka main concepts
The workflow of a producer

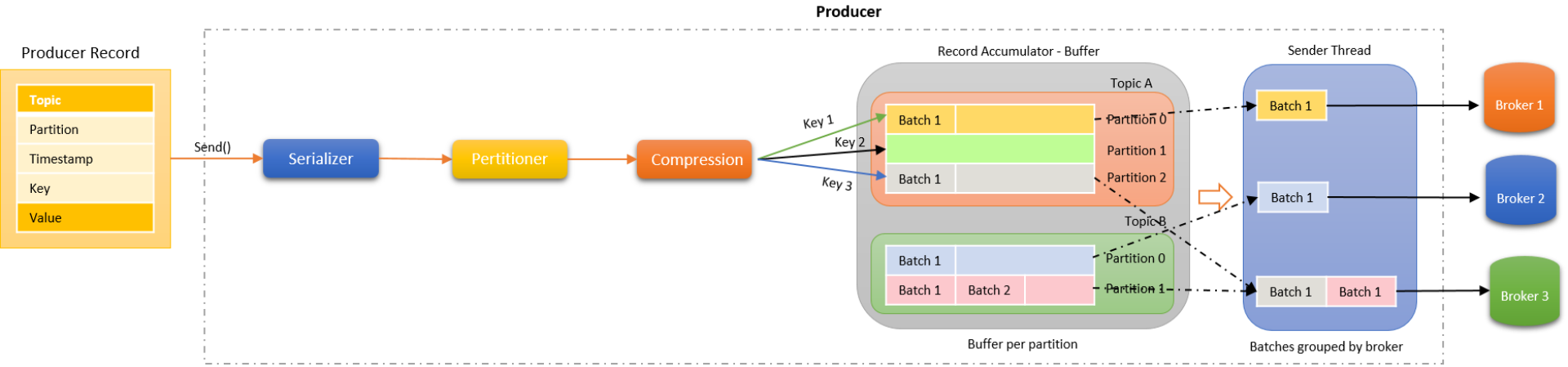


Image source: https://www.linkedin.com/pulse/kafka-producer-overview-sylvester-daniel

Kafka main concepts
Consumer overview

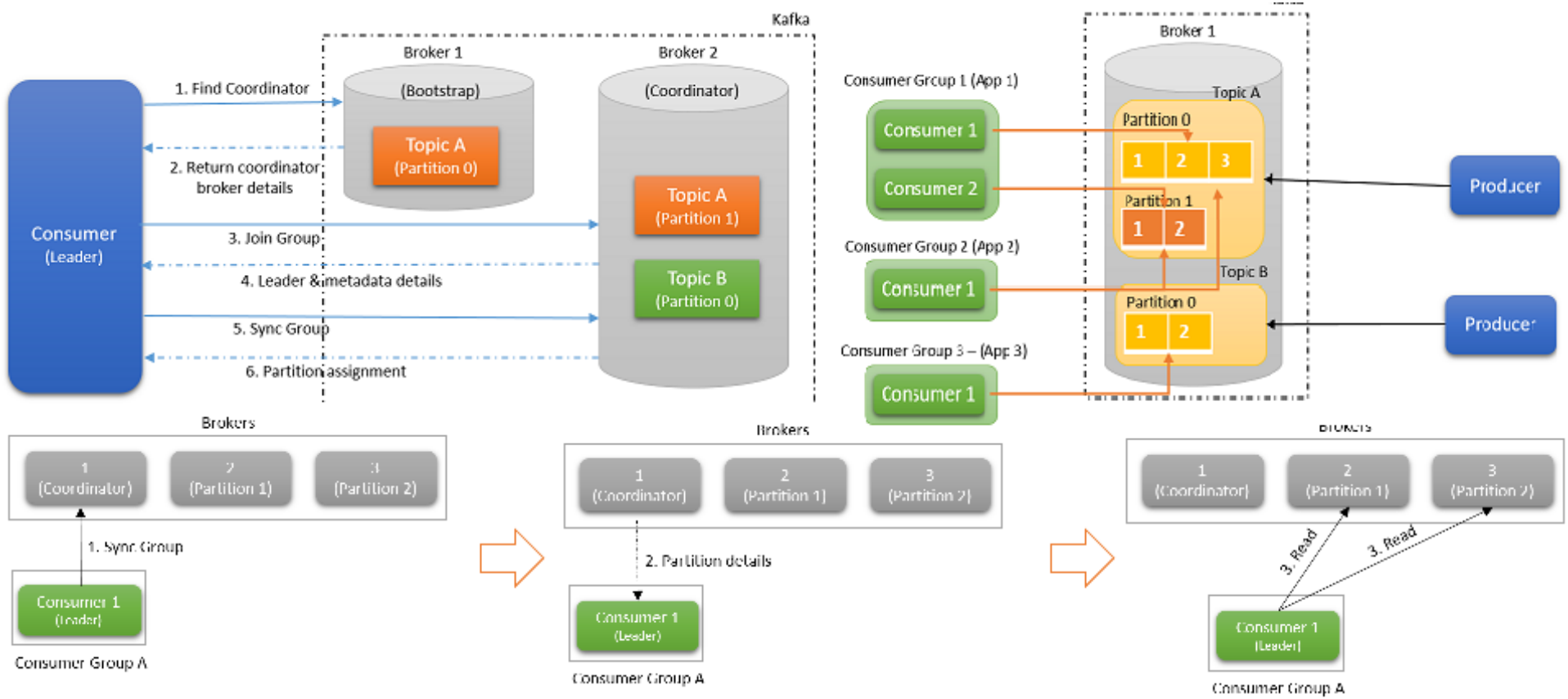


Image source: https://www.linkedin.com/pulse/kafka-producer-overview-sylvester-daniel

Kafka main concepts
Consumer API / batch processing

Allows applications to process messages from topics, supporting:

- Separation of responsibility between consumers and producers
- Single processing
- Support for **batch processing**
- Only stateless support (the client does not keep the previous state and evaluates each record in the stream individually)
- Writing an application requires a lot of code
- No use of threading or parallelism
- It is possible to write in several Kafka clusters.

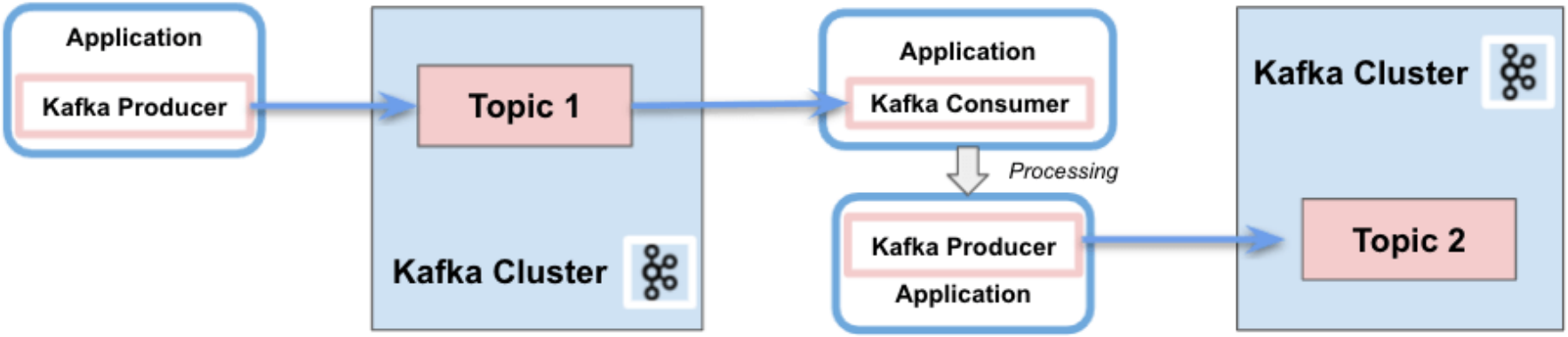


Image source: https://www.baeldung.com/java-kafka-streams-vs-kafka-consumer

Kafka main concepts
Streams API / stream processing

Simplifies the stream processing from topics, provides data parallelism, distributed coordination, fault tolerance, and scalability.

It deals with messages as an unbounded, continuous, and real-time flow of records, with the following characteristics:

- Single Kafka Stream to consume and produce
- Support for (complex) **stream processing**
- No specific support for batch processing
- Support stateless and stateful operations
- Stream partitions and tasks as logical units for storing and transporting messages

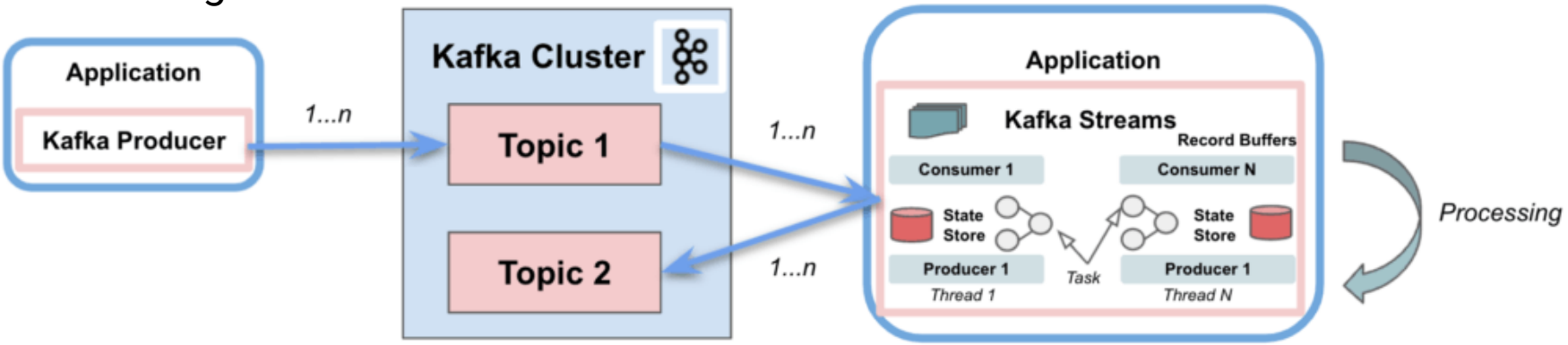
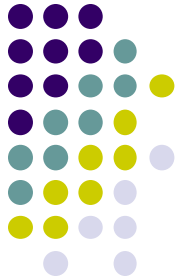


Image source: https://www.baeldung.com/java-kafka-streams-vs-kafka-consumer

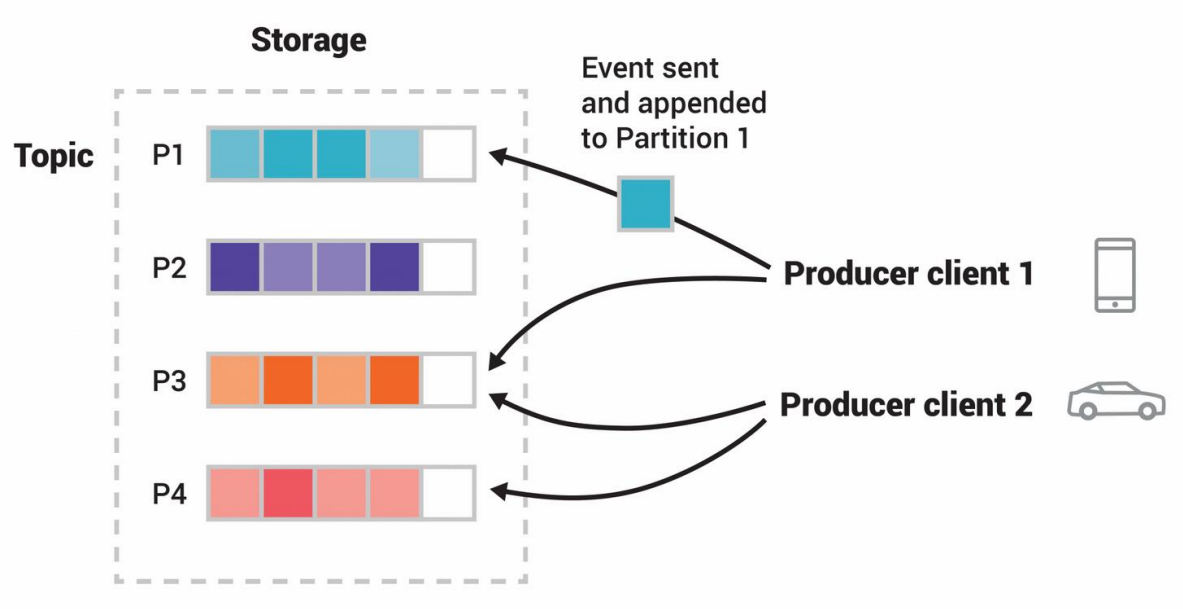
Kafka main concepts
Topics and partitions



Events are organized in **topics**.

- Topics in Kafka are always multi-producer and multi-subscriber: a topic can have zero, one, or many producers that write events to it, as well as zero, one, or many consumers that subscribe to these events.
- Events in a topic can be read as often as needed – unlike traditional messaging systems such as MQTT, events are not deleted after consumption. Instead, it is possible to define for how long Kafka should retain events through a per-topic configuration setting, after which old events will be discarded.
- Topics are **partitioned**, meaning a topic is spread over a number of "buckets" located on different Kafka brokers. This distributed placement of data is very important for scalability, because it allows client applications to both read and write the data from/to many brokers at the same time.
- When a new event is published to a topic, it is appended to one of the topic's partitions. Events with the same event key (e.g., a customer or vehicle ID) are written to the same partition, and Kafka ensures that any consumer of a given topic-partition will always read that partition's events in the same order as they were written.

Kafka main concepts
Topics



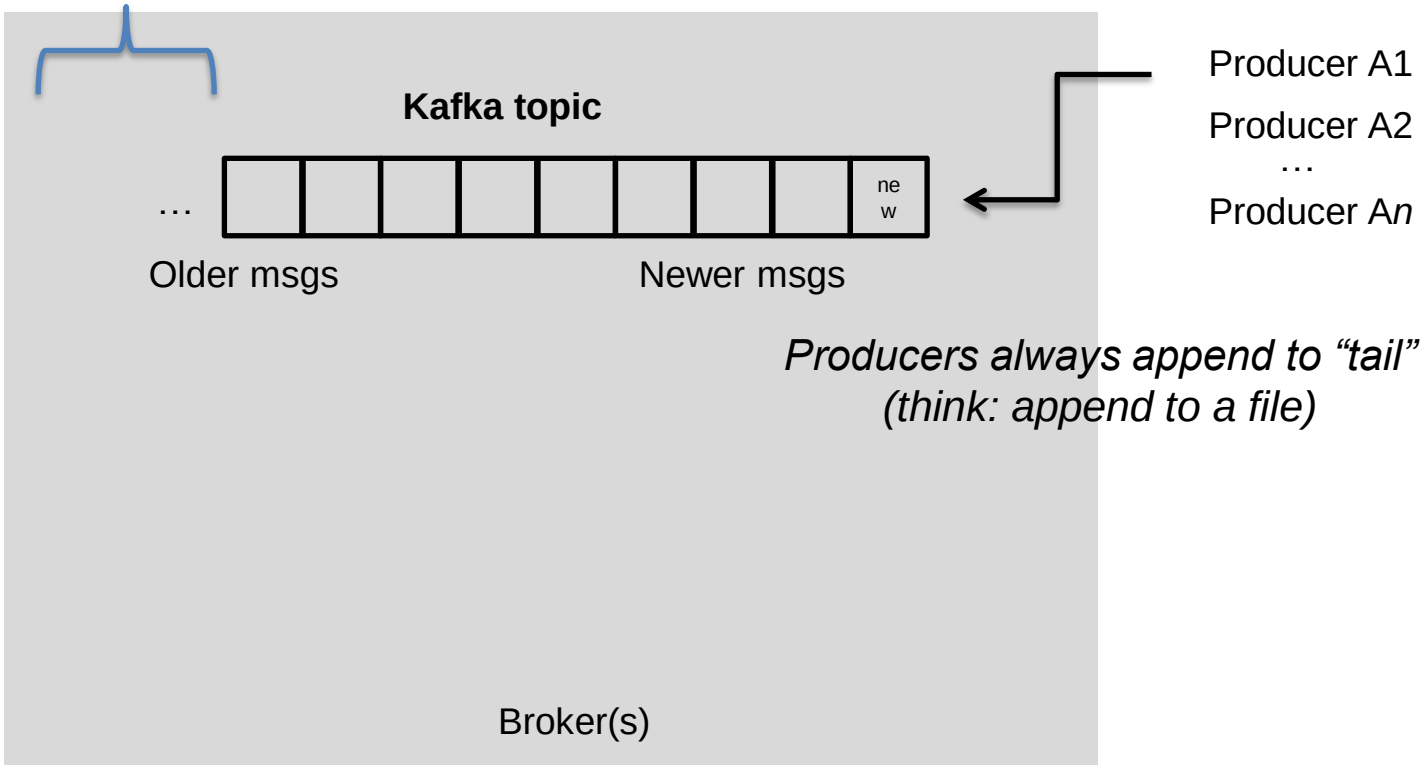
- This example topic has four partitions (P1–P4).
- Two different producer clients are publishing new events to the topic, independently from each other, by writing events over the network to the topic's partitions.
- Events with the same key (denoted by their color in the figure) are always written to the same partition. Both producers can write to the same partition if appropriate.

Image source: https://kafka.apache.org/

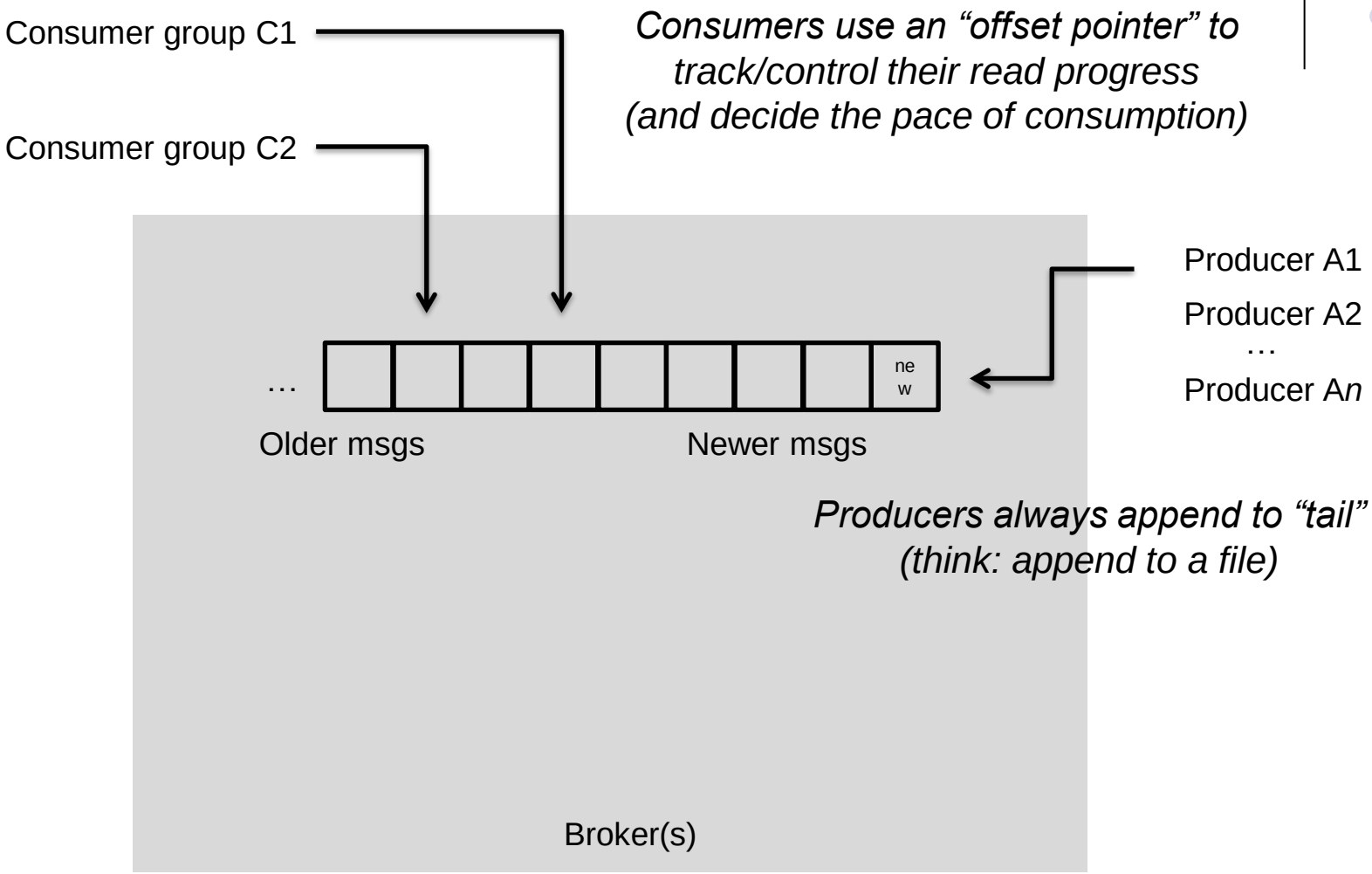
Topics – writing and pruning

how is data retention handled?

Kafka prunes topic "head" based on age or max size or "key"

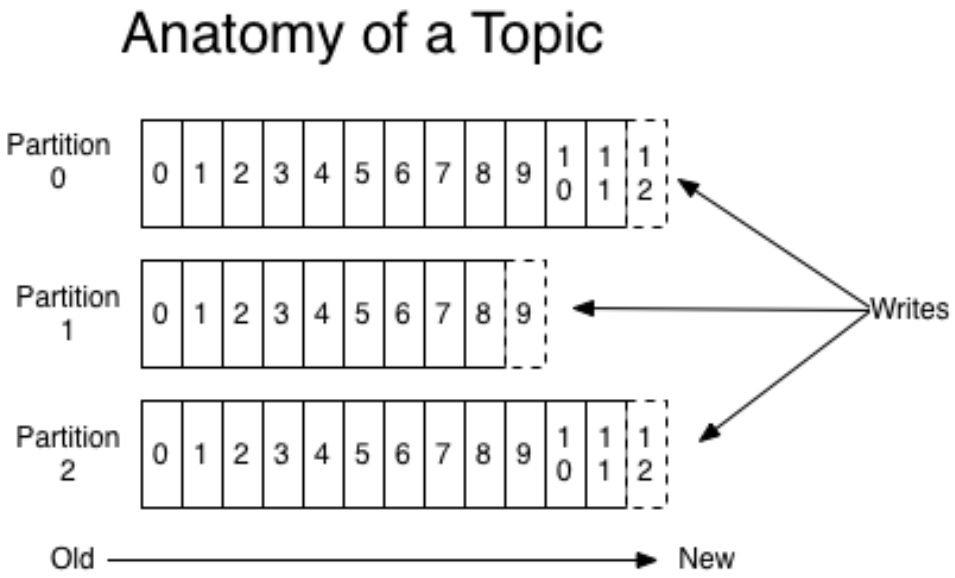


Topics – offset pointers



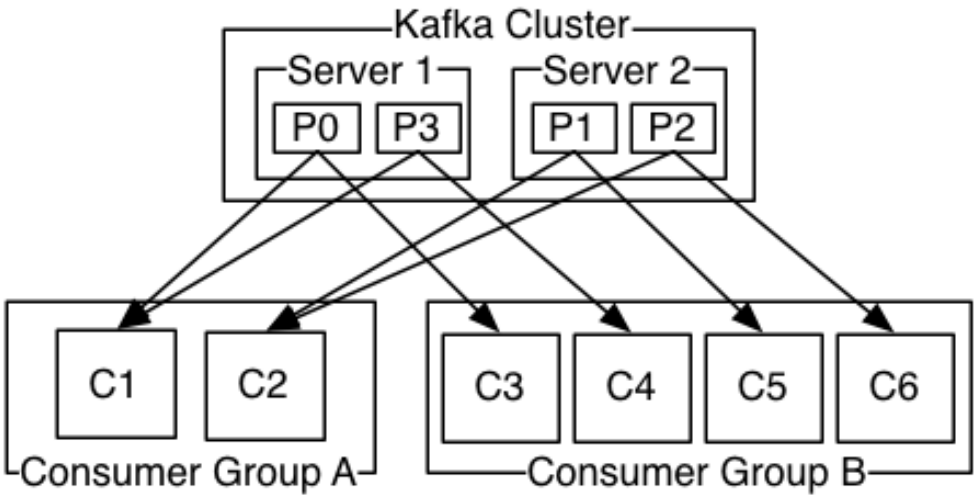
Partitions

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



Partitions

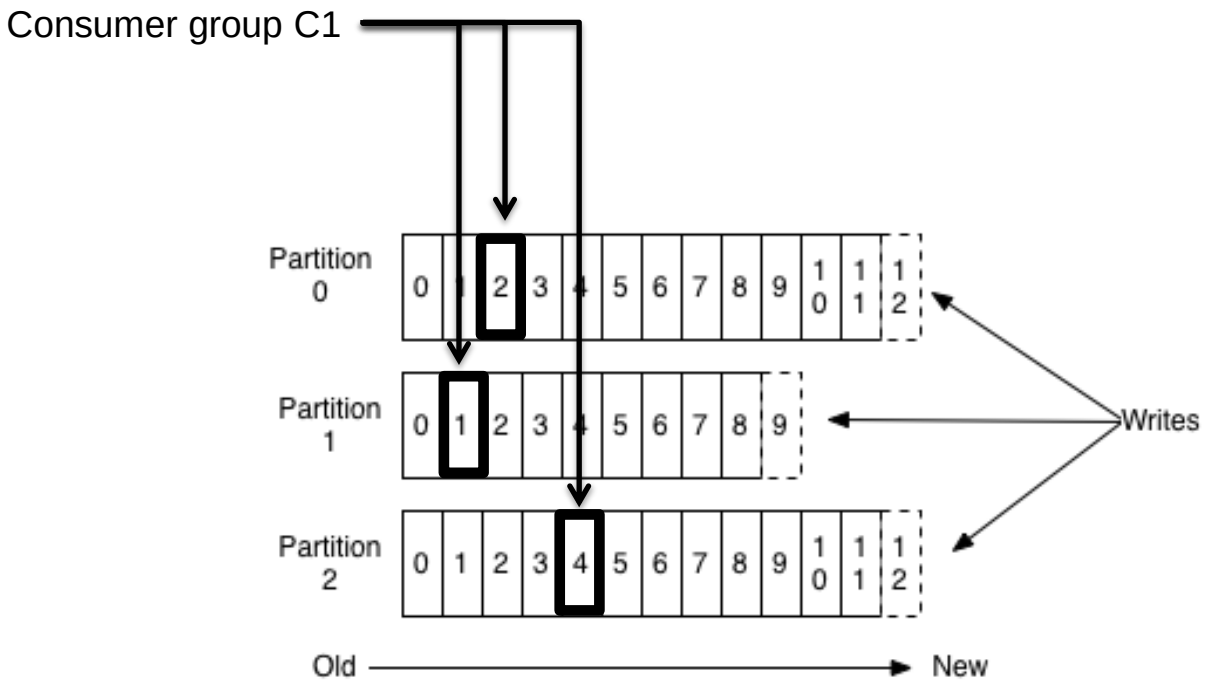
- Number of partitions of a topic is configurable
- Number of partitions indirectly determines the maximum size of each “consumer group”



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

Partition offsets

- Each message in the partition is assigned a unique (per partition) and sequential ID, called the *offset*
  - Consumers track their pointers via *(offset, partition, topic)* tuples



Replicas of a partition

- **Replicas** are “backups” of a partition
  - They exist solely to prevent data loss
  - Replicas are never read from or written to
    - They do NOT increase producer or consumer parallelism!
  - Kafka tolerates *(numReplicas - 1)* dead brokers with no data loss (e.g., if *numReplicas* is 3, up to 2 brokers may die before losing data).

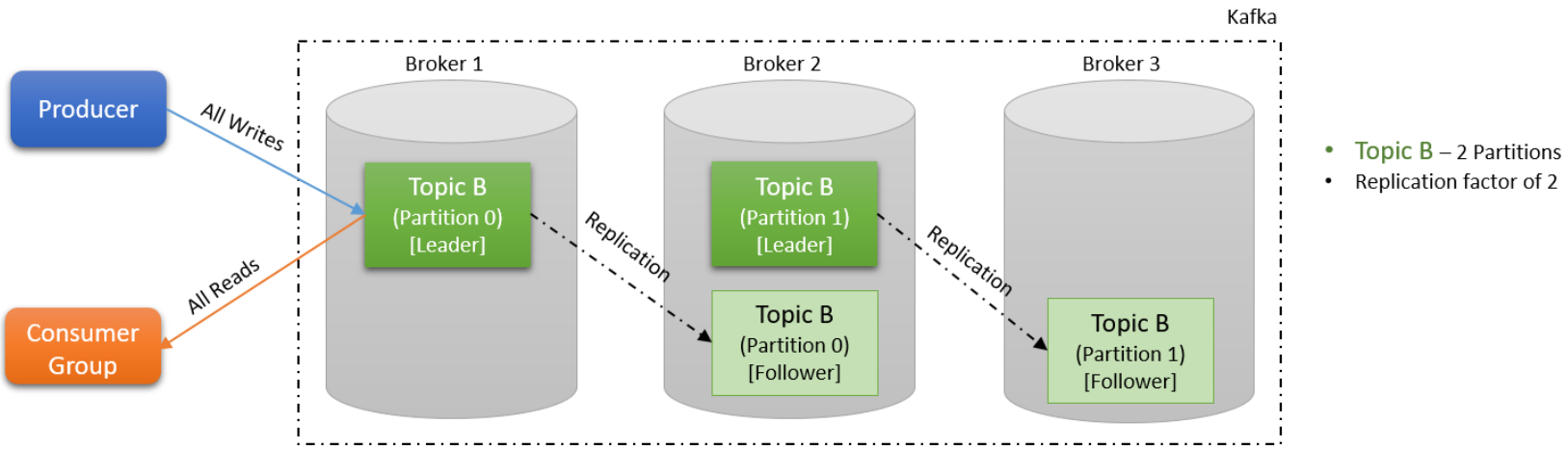


Image source: https://www.linkedin.com/pulse/kafka-producer-overview-sylvester-daniel/

Replicas of a partition

- Both producer and consumers are severed only by the leader.
- If a leader fails, the partition from another broker is elected as leader and it starts serving the producers and consumer groups.
- Replica partitions that are in sync with the leader are flagged as ISR (In Sync Replica).

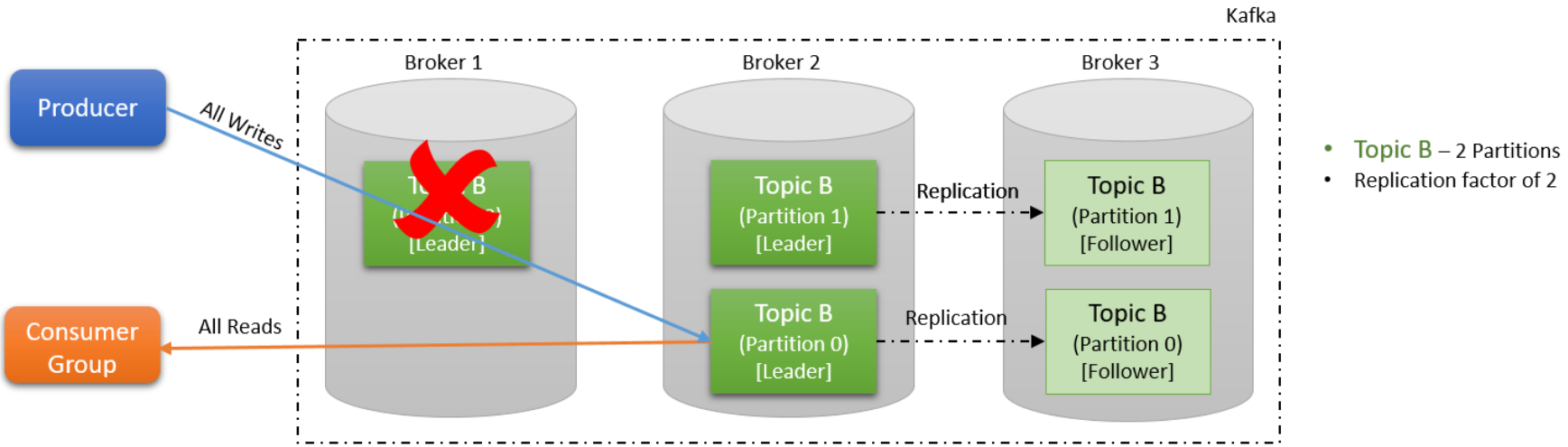
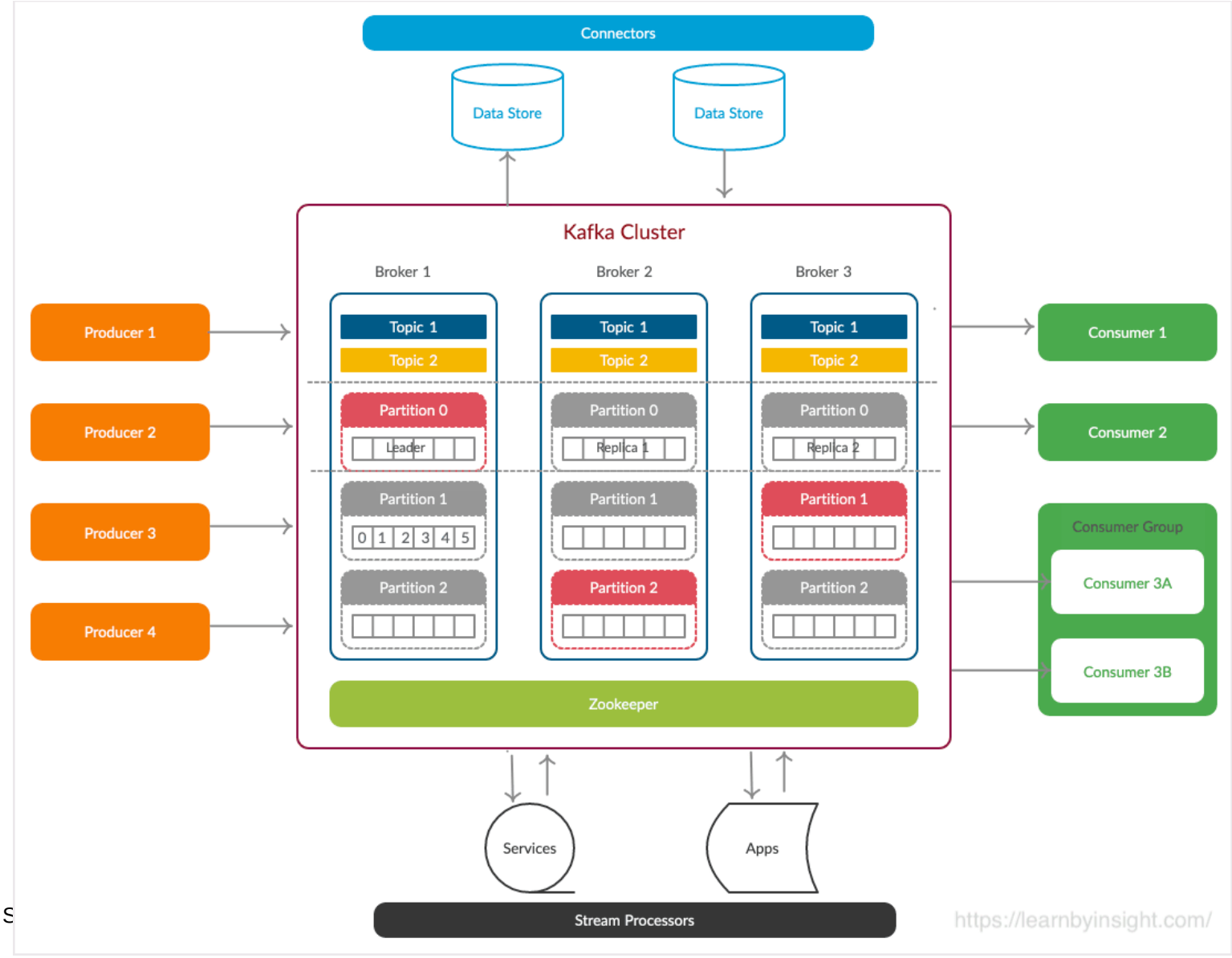
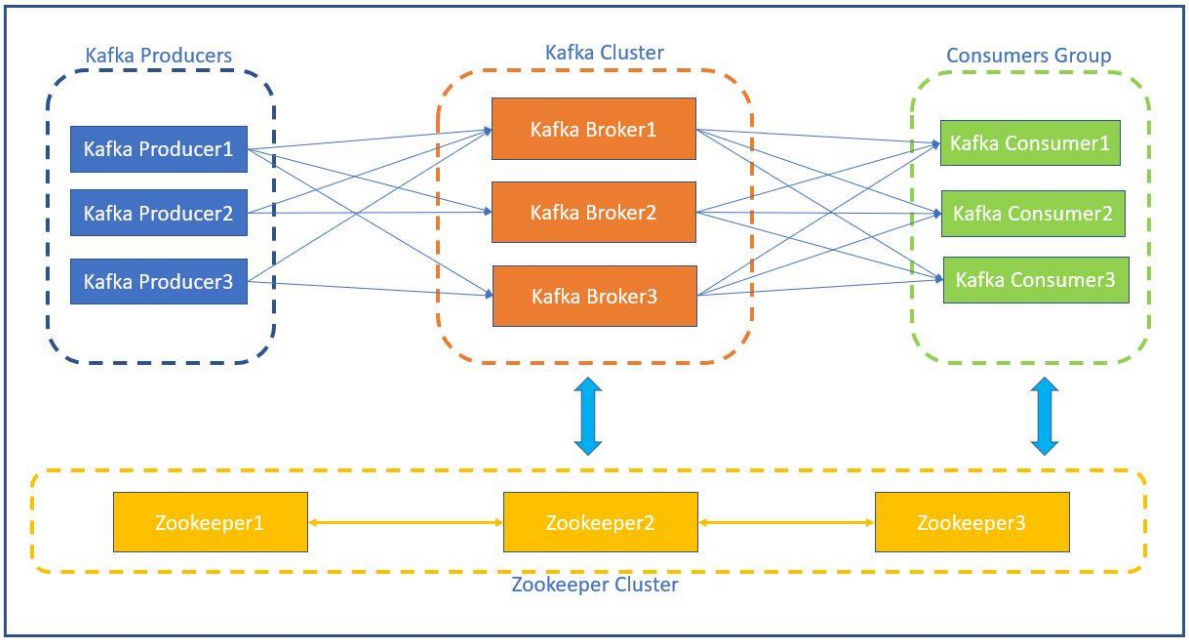


Image source: https://www.linkedin.com/pulse/kafka-producer-overview-sylvester-daniel

Yet another perspective



The role of Zookeeper

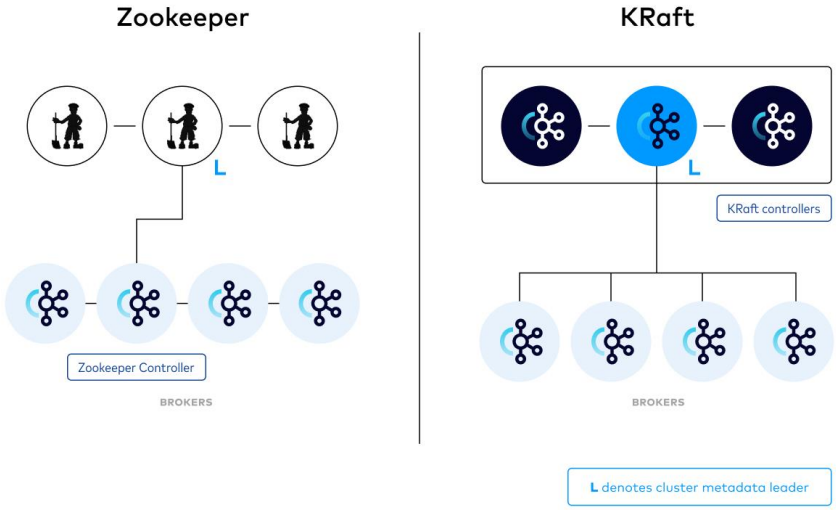


Zookeeper is in charge of managing and maintaining the Brokers, Topics, and Partitions of the Kafka Clusters. It keeps track of the Brokers of the Kafka Clusters. It determines which Brokers have crashed and which Brokers have just been added to the Kafka Clusters, as well as their lifetime. Then, it notifies the Producer or Consumers of Kafka queues about the state of Kafka Clusters. This facilitates the coordination of work with active Brokers for both Producers and Consumers.

Zookeeper also keeps track of which Broker is the subject Partition's Leader and gives that information to the Producer or Consumer so they may read and write messages.

Image source: http://www.liferaysavvy.com/2021/07/setup-kafka-cluster.html

KRaft



Apache Kafka Raft (KRaft) is the consensus protocol that was introduced in KIP-500 to remove Apache Kafka's dependency on ZooKeeper for metadata management. This greatly simplifies Kafka's architecture by consolidating responsibility for metadata into Kafka itself, rather than splitting it between two different systems: ZooKeeper and Kafka. KRaft mode makes use of a new quorum controller service in Kafka which replaces the previous controller and makes use of an event-based variant of the Raft consensus protocol.

Image source: http://www.liferaysavvy.com/2021/07/setup-kafka-cluster.html

How fast is Kafka?

- **“Up to 2 million writes/sec on 3 cheap machines”**
  - Using 3 producers on 3 different machines (3x asynchronous replication)
    - Only 1 producer/machine because NIC already saturated
- **Sustained throughput as stored data grows**
  - Slightly different test config than 2M writes/sec above.

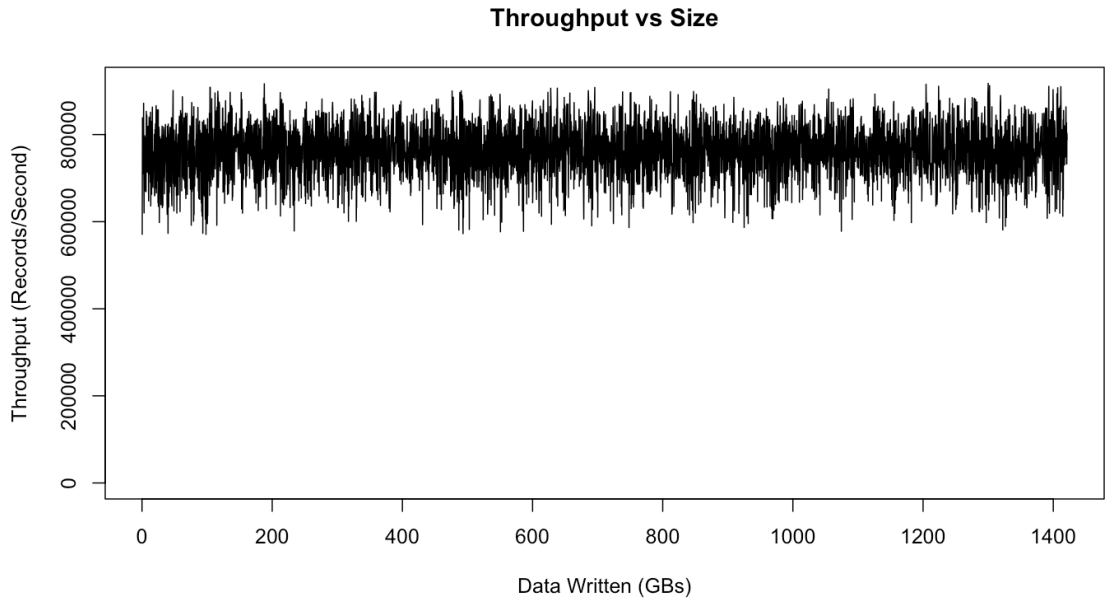
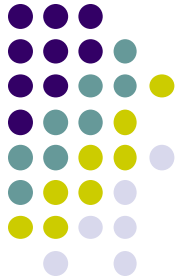


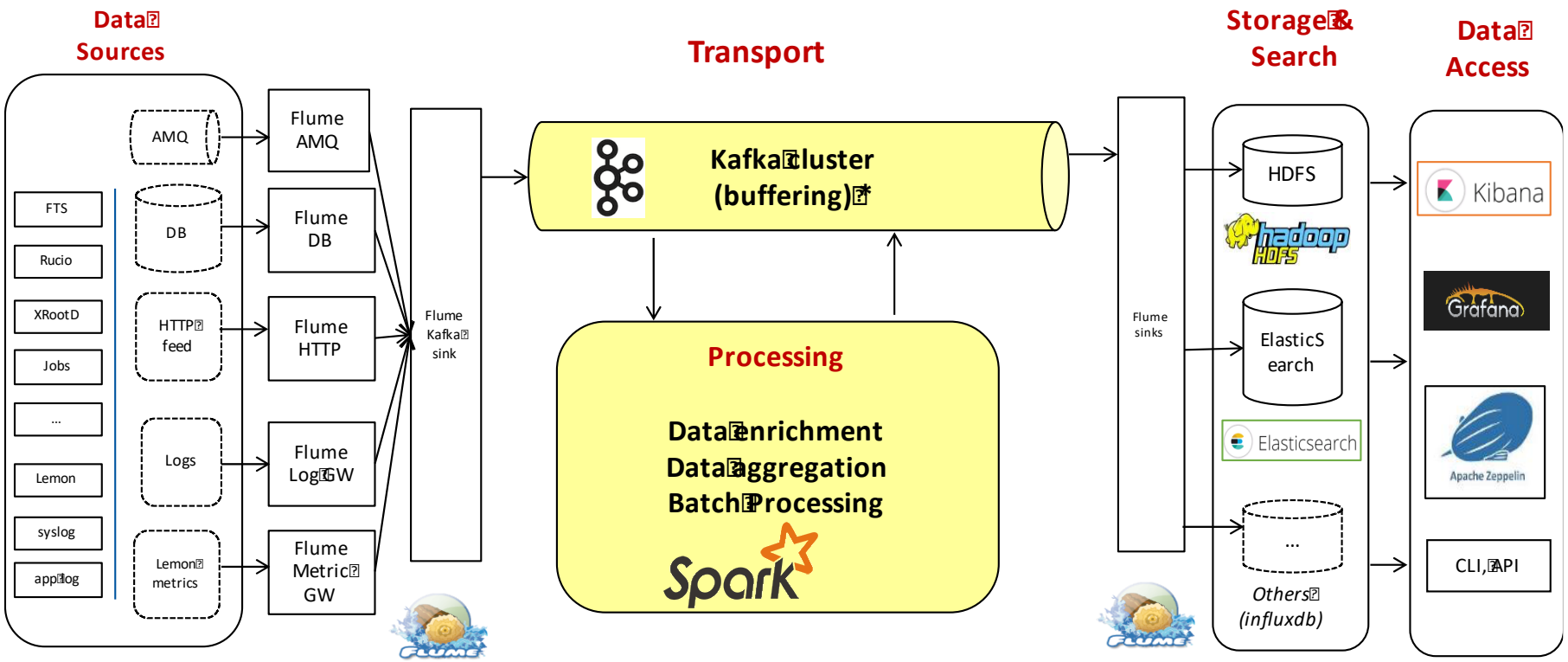
Image source: https://www.csee.umbc.edu/~shadam1/491s16/lectures/04-Kafka.pptx

Why is Kafka so fast?



- Fast **writes**:
  - While Kafka persists all data to disk, essentially all writes go to the **page cache** of the Operating System, i.e., to RAM memory
- Fast **reads**:
  - Very efficient to transfer data from RAM (page cache) to a network socket
- Combination of the two = fast Kafka!
  - Example (Operations): On a Kafka cluster where the consumers are mostly caught up you will see no read activity on the disks as they will be serving data entirely from the cache

The CERN Use Case
IT monitoring



Source: https://indico.cern.ch/event/578621/contributions/2344074/attachments/1388632/2114416/HUF-Kafka.pptx