Lambda & Kappa Architectures

Apache Kafka

Machine Learning + Big Data in Real Time + Cloud Technologies

=> The Future of Intelligent Systems

Where to Find The Code and Materials?

https://github.com/iproduct/course-ml

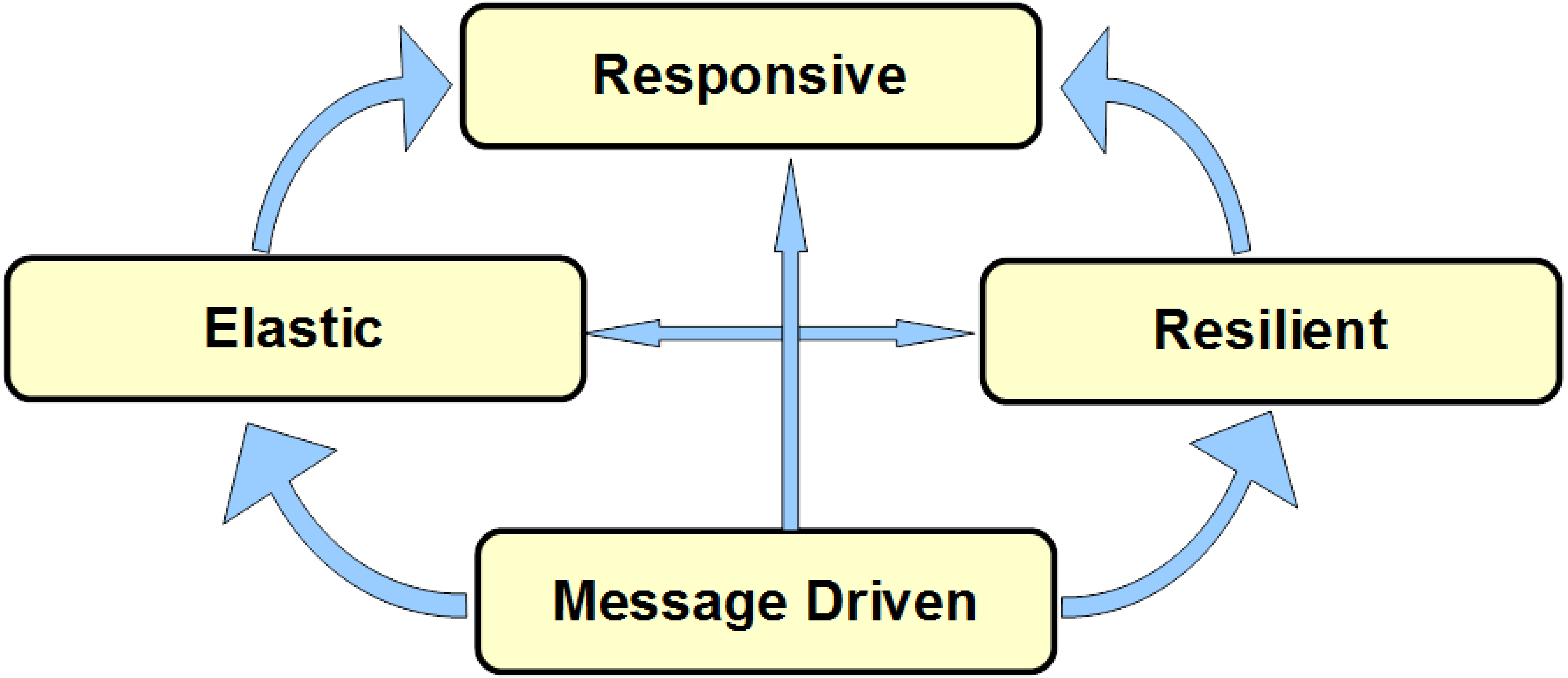
Agenda for This Lesson - I

- Introduction to distributed reactive systems (Reactive Manifesto)
- State and behavior of distributed systems – event sourcing.
- Distributed state consistency guarantees – eventual consistency.
- CAP theorem.
- Technologies and architectures for Big Data processing in real time – Lambda and Cappa architectures.
- Web scale systems, data lakes and Zeta architecture.

Lambda Architecture

Reactive Manifesto

[http://www.reactivemanifesto.org]



Data / Event / Message Streams

“Conceptually, a stream is a (potentially never-ending) flow of data records, and a transformation is an operation that takes one or more streams as input, and produces one or more output streams as a result.”

Apache Flink: Dataflow Programming Model

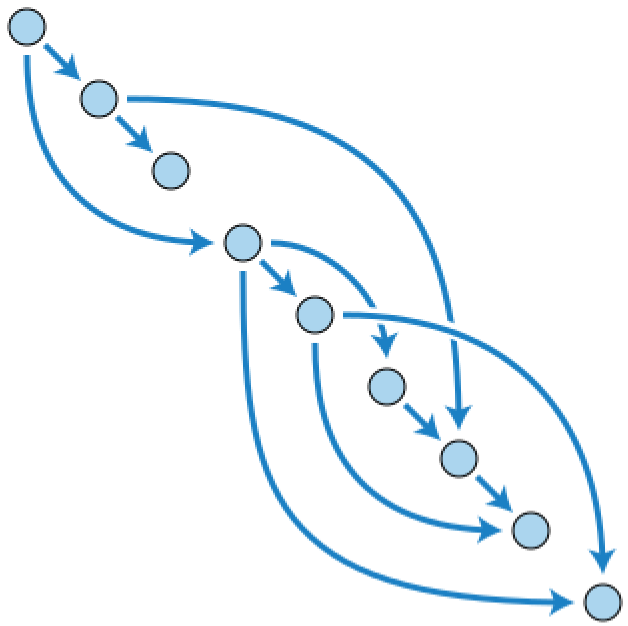
Data Stream Programming

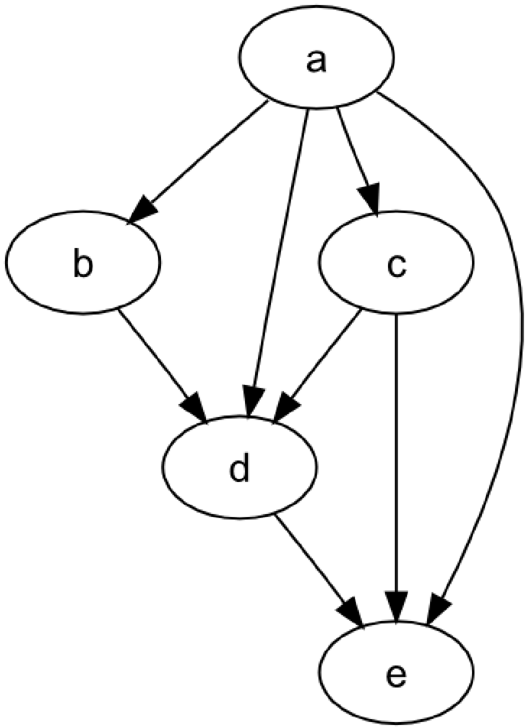
The idea of abstracting logic from execution is hardly new -- it was the dream of SOA. And the recent emergence of microservices and containers shows that the dream still lives on.

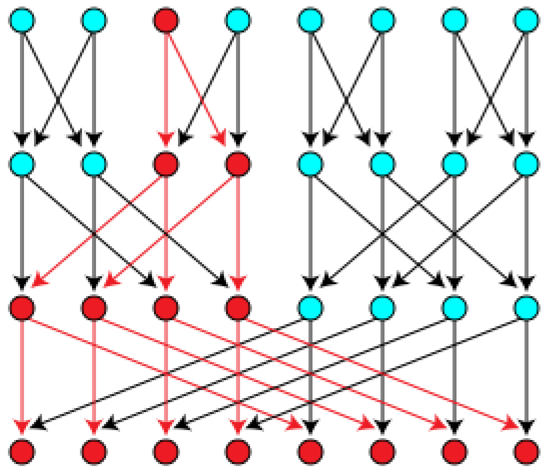
For developers, the question is whether they want to learn yet one more layer of abstraction to their coding. On one hand, there's the elusive promise of a common API to streaming engines that in theory should let you mix and match, or swap in and swap out.

Tony Baer (Ovum) @ ZDNet - Apache Beam and Spark: New coopetition for squashing the Lambda Architecture?

Direct Acyclic Graphs - DAG

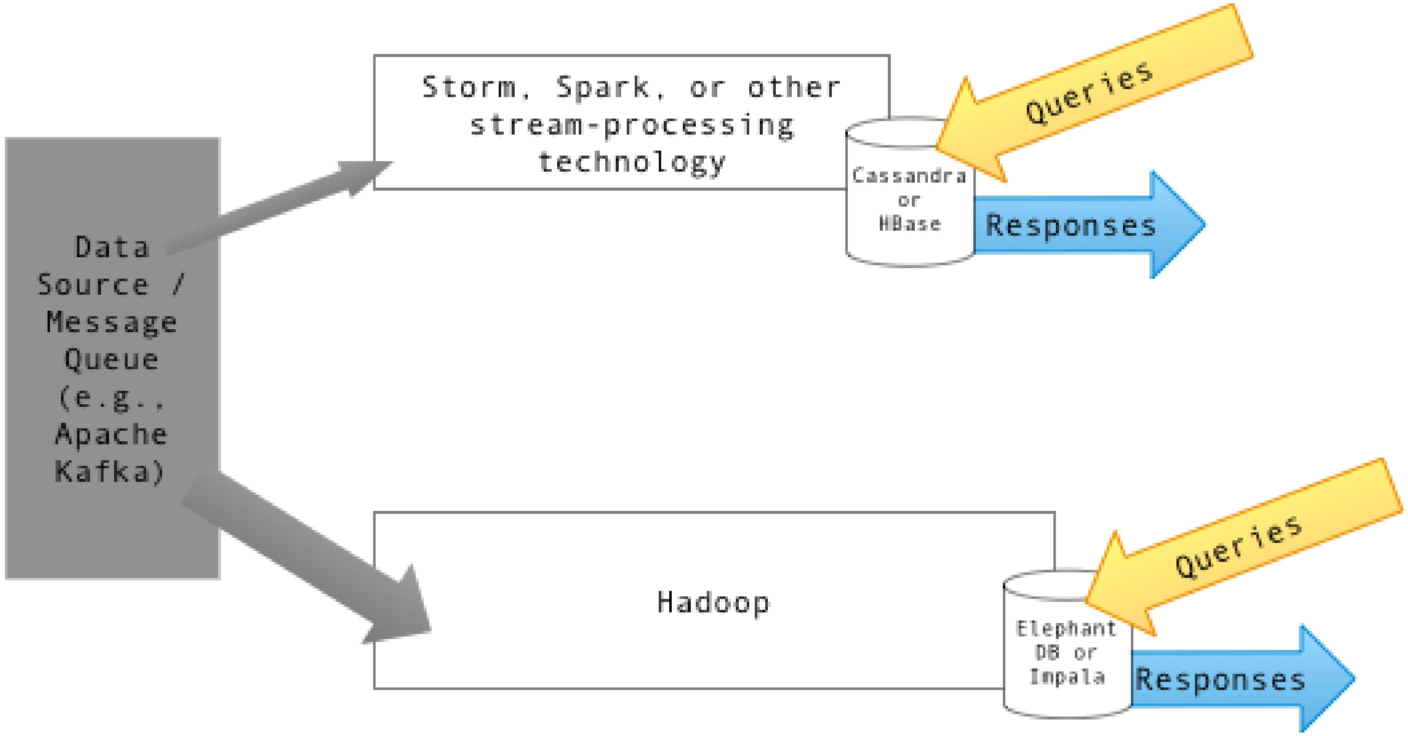






Lambda Architecture - I

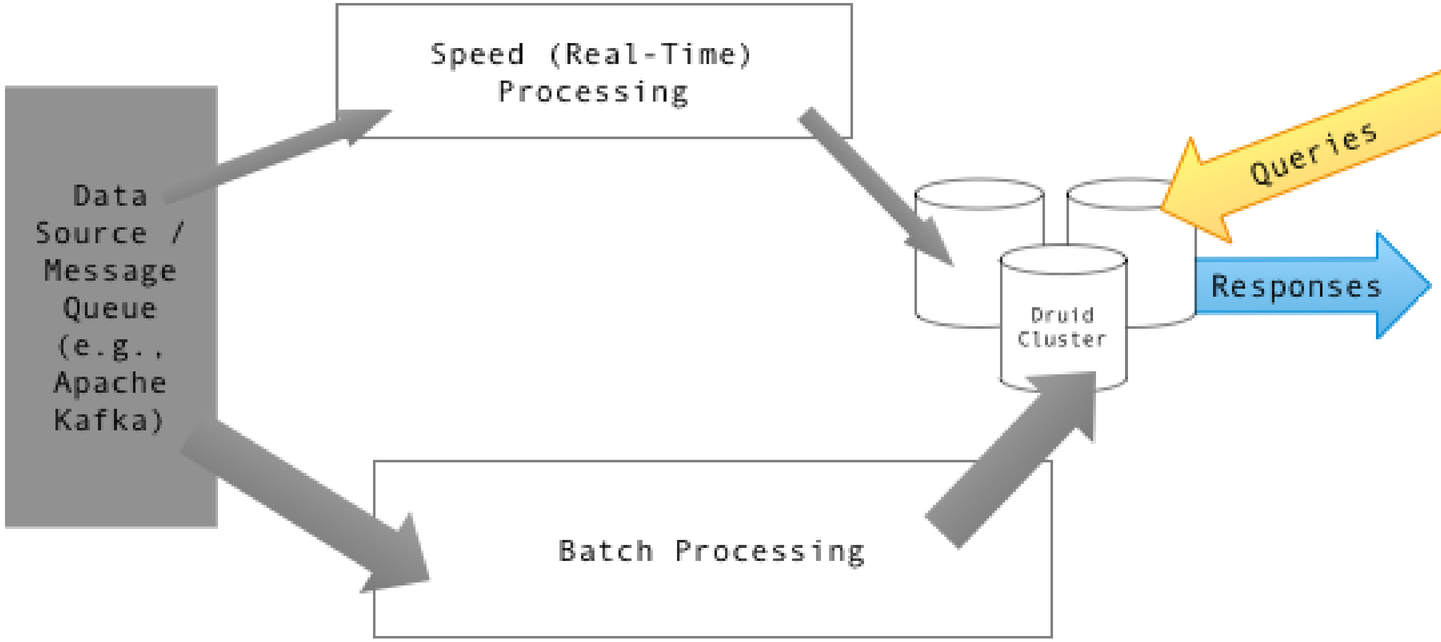
Query = λ (Complete data) = λ (live streaming data) * λ (Stored data)



https://commons.wikimedia.org/w/index.php?curid=34963986, By Textractor - Own work, CC BY-SA 4

Lambda Architecture - II

Query = λ (Complete data) = λ (live streaming data) * λ (Stored data)

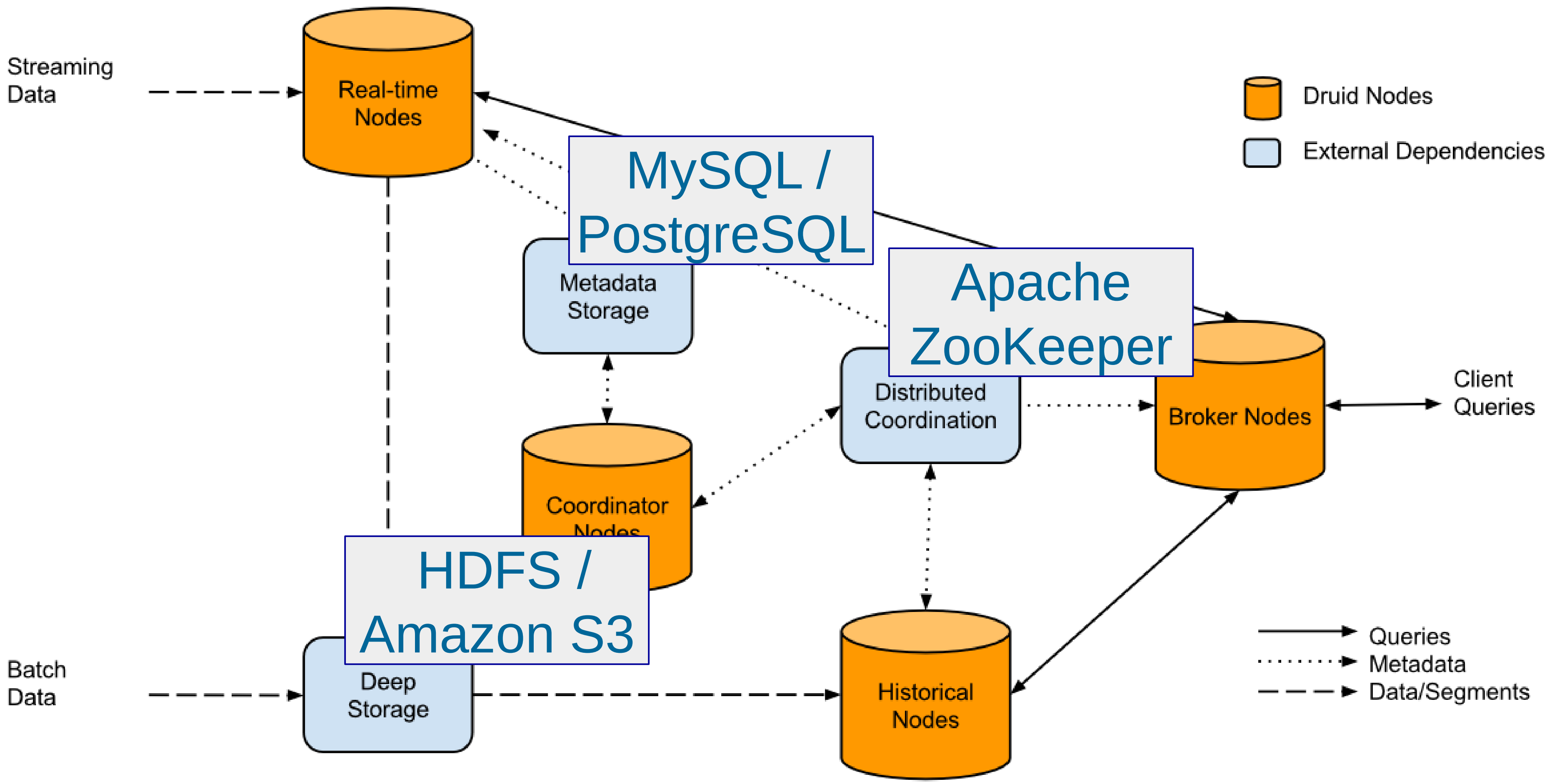


https://commons.wikimedia.org/w/index.php?curid=34963987, By Textractor - Own work, CC BY-SA 4

Lambda Architecture - III

- Data-processing architecture designed to handle massive quantities of data by using both *batch-* and *stream-processing* methods
- Balances *latency, throughput, fault-tolerance, big data, real-time analytics*, mitigates the latencies of map-reduce
- Data model with an append-only, *immutable data* source that serves as a system of record
- Ingesting and processing *timestamped events* that are appended to existing events. State is determined from the *natural time-based ordering* of the data.
- Can use relational SQL DBs or a key-value store like Cassandra

Druid Distributed Data Store (Java)



https://commons.wikimedia.org/w/index.php?curid=33899448 By Fangjin Yang - sent to me personally, GFDL

Lambda Architecture: Projects - I

❖Apache Spark is an open-source cluster-computing framework. Spark Streaming, Spark Mllib

❖Apache Storm is a distributed stream processing – streams DAG

❖Apache Apex™ unified stream and batch processing engine.

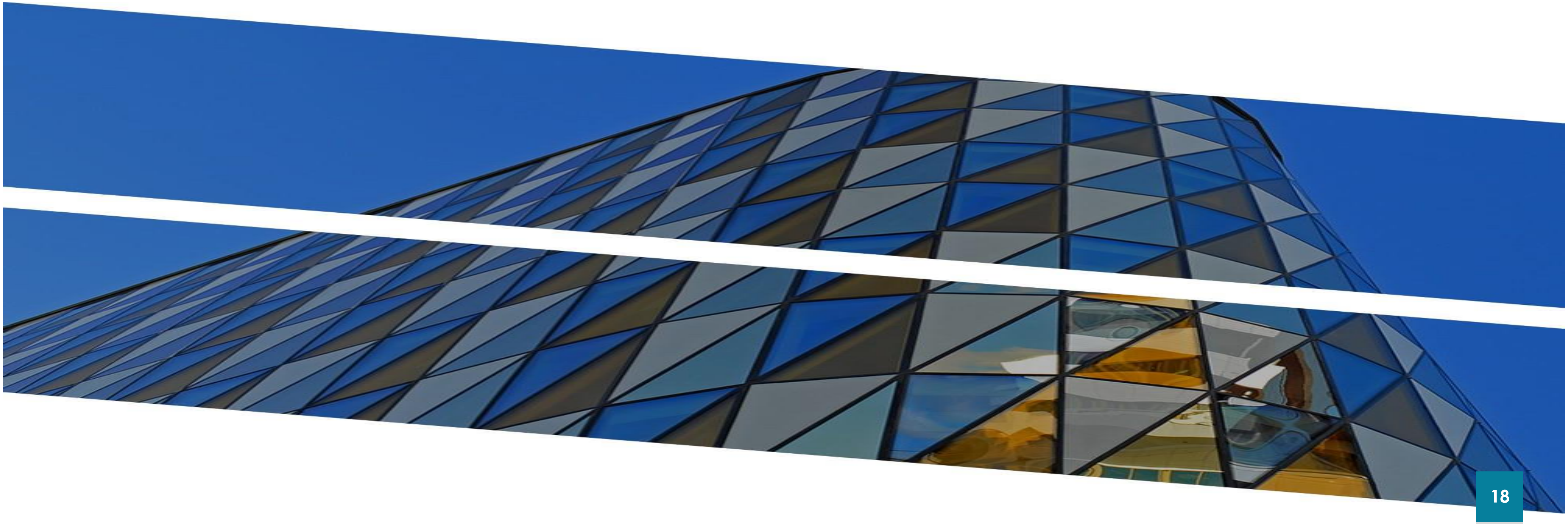
Lambda Architecture: Projects - II

- ❖Apache Flink - open source stream processing framework – Java, Scala

- ❖Apache Kafka - open-source stream processing (Kafka Streams), real-time, low-latency, high-throughput, massively scalable pub/sub

- ❖Apache Beam – unified batch and streaming, portable, extensible

Kappa Architecture



Kappa Architecture

Query = K (New Data) = K (Live streaming data)

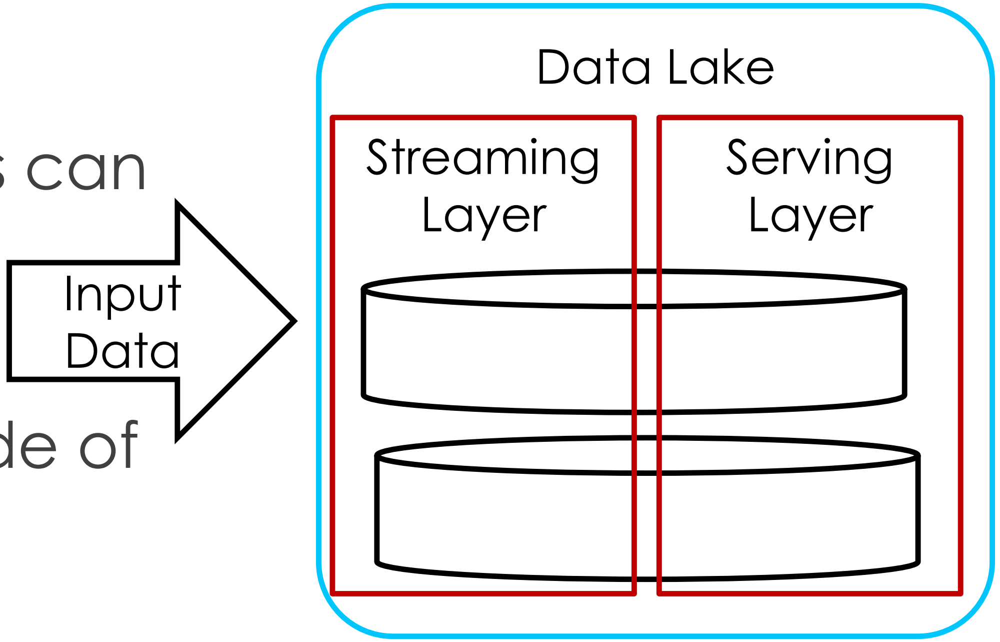
λ **vs** $\varkappa$

- Proposed by Jay Kreps in 2014
- Real-time processing of distinct events
- Drawbacks of Lambda architecture:
  - It can result in coding overhead due to involvement of comprehensive processing.
  - Re-processes every batch cycle which is not beneficial in certain scenarios.
  - A data modeled with Lambda architecture is difficult to migrate or reorganize.
- Canonical data store in a Kappa Architecture system is an ***append-only immutable log***

Kappa Architecture II

Query = K (New Data) = K (Live streaming data)

- Multiple data events or queries are logged in a queue to be catered against a distributed file system storage or history.
- The order of the events and queries is not predetermined. Stream processing platforms can interact with database at any time.
- It is resilient and highly available as handling Terabytes of storage is required for each node of the system to support replication.



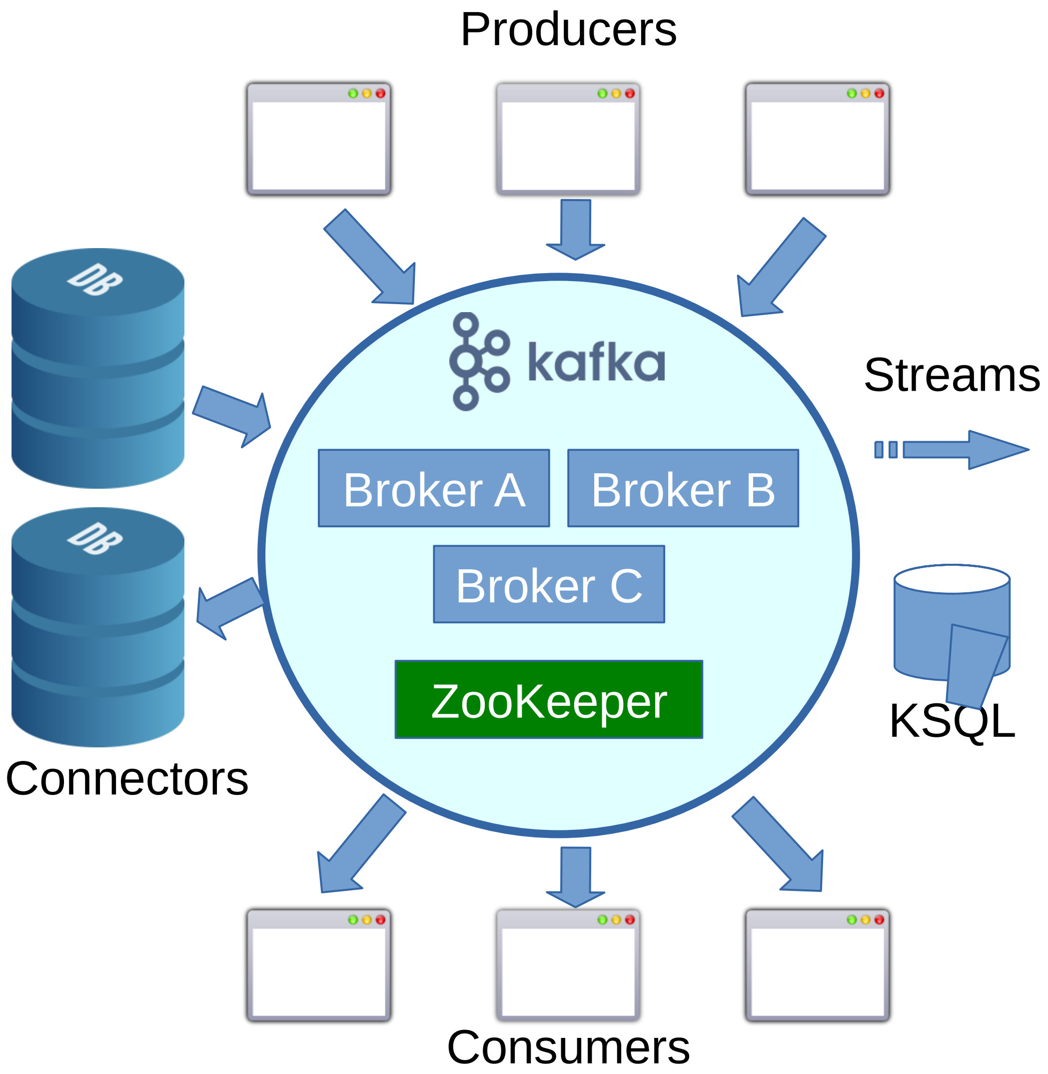
Pros and Cons of Kappa architecture

- **Pros**
  - Kappa architecture can be used to develop data systems that are online learners and therefore don't need the batch layer.
  - Re-processing is required only when the code changes.
  - It can be deployed with fixed memory.
  - It can be used for horizontally scalable systems.
  - Fewer resources are required as the machine learning is being done on the real time basis.
- **Cons**
  - Absence of batch layer might result in errors during data processing or while updating the database that requires having an exception manager to reprocess the data or reconciliation.

Apache Kafka

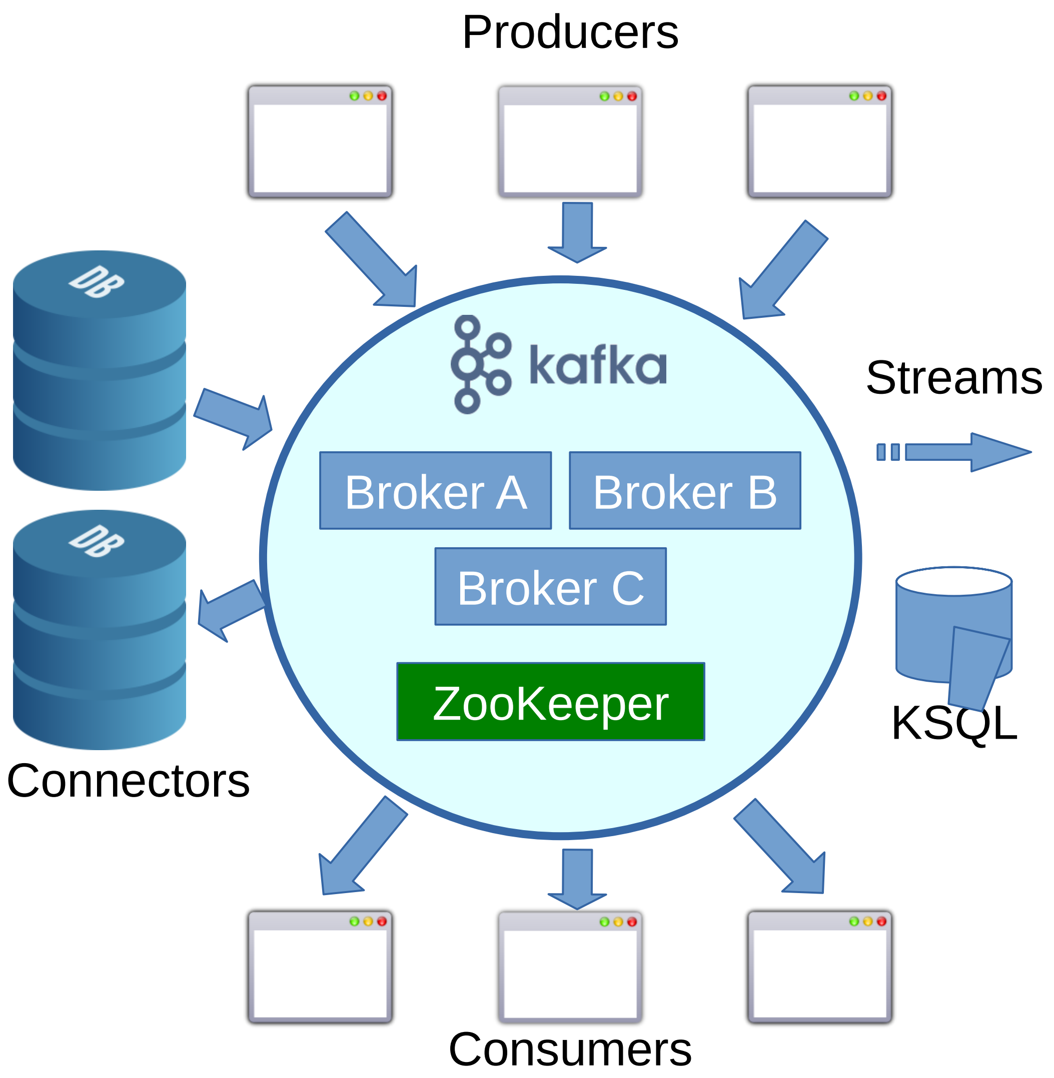
Kafka Main Concepts

- Kafka is run as a cluster on one or more servers (brokers) that can span multiple datacenters.
- The Kafka cluster stores streams of records in categories called topics.
- Each record consists of a key, value, and timestamp.



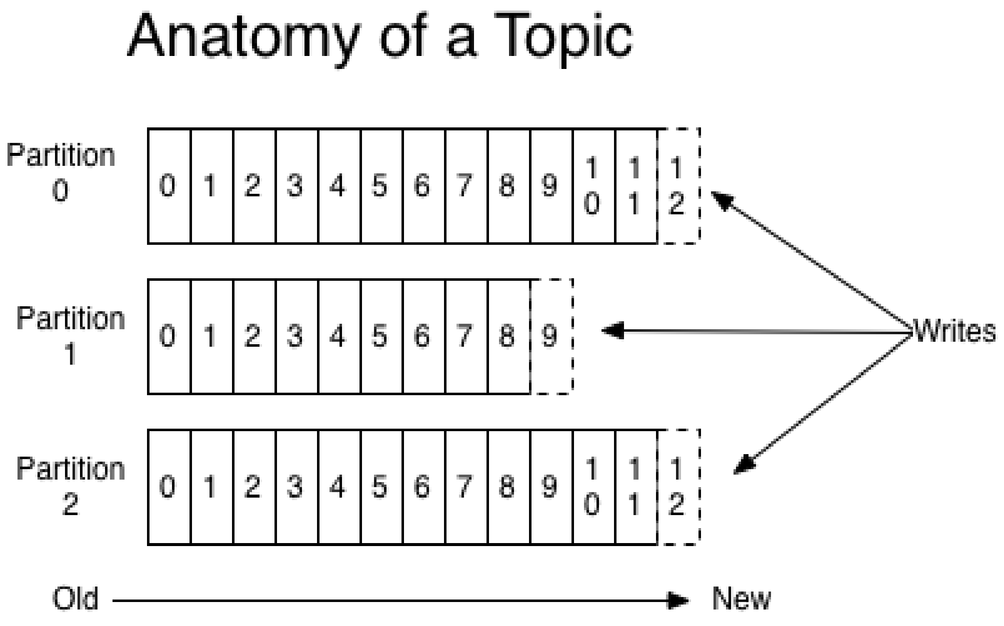
Kafka Core APIs

- The Producer API - publish a stream of records to one or more Kafka topics.
- The Consumer API - subscribe to one or more topics and process the stream of records produced to them.
- The Streams API - a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems – e.g. connector to a DB might capture every change in a table



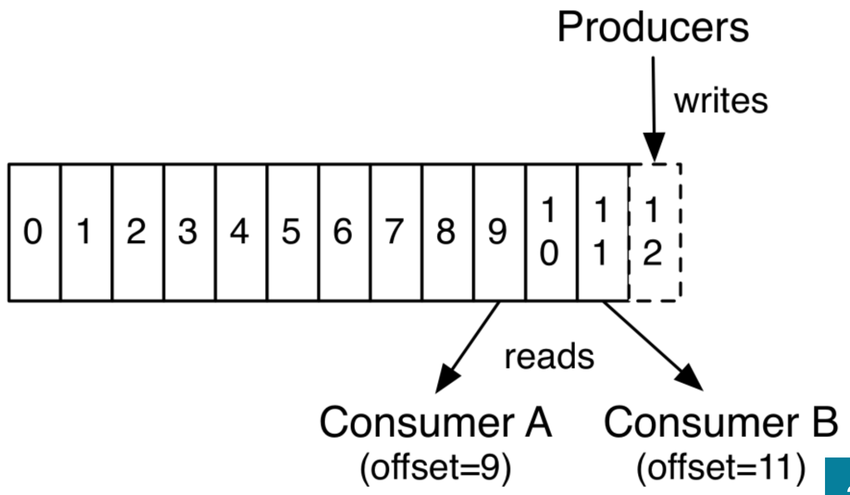
Topics and Logs

- Topic = stream of records
- A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data
- For each topic, the Kafka cluster maintains a partitioned log --->
- Each partition is an ordered, immutable sequence of records that is continually appended to - a structured commit log
- The records in the partitions are each assigned a sequential id number called the offset that uniquely identifies each record within the partition.



Consumers Offset and Data Retention

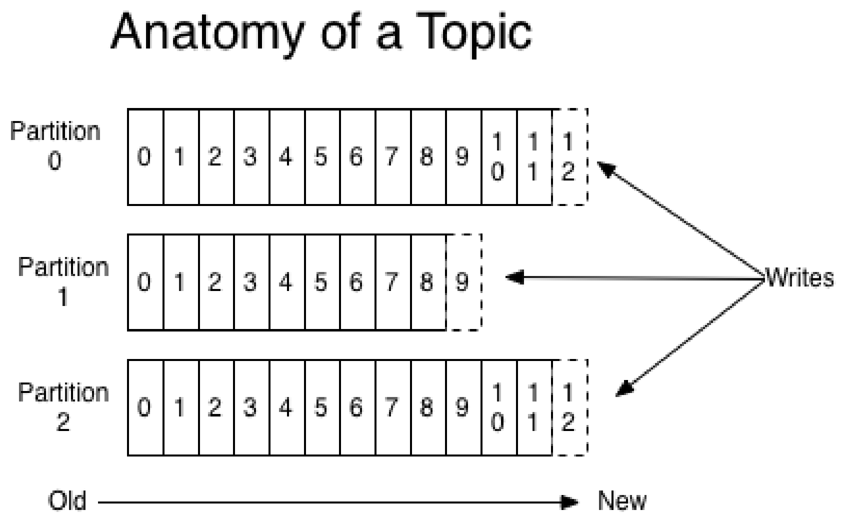
- Kafka cluster durably persists all published records - whether or not they have been consumed, using configurable retention period
- Kafka performance is effectively constant with respect to data size
- Offset or position of that consumer in the log – controlled by the consumer: normally a consumer will advance its offset linearly as it reads records, but, since the position is controlled by the consumer it can consume records in any order. E.g. a consumer can reset to an older offset to reprocess data from the past or skip ahead to the most recent record and start consuming from "now".
- Kafka consumers are very cheap - they can come and go without much impact on the cluster or on other consumers.



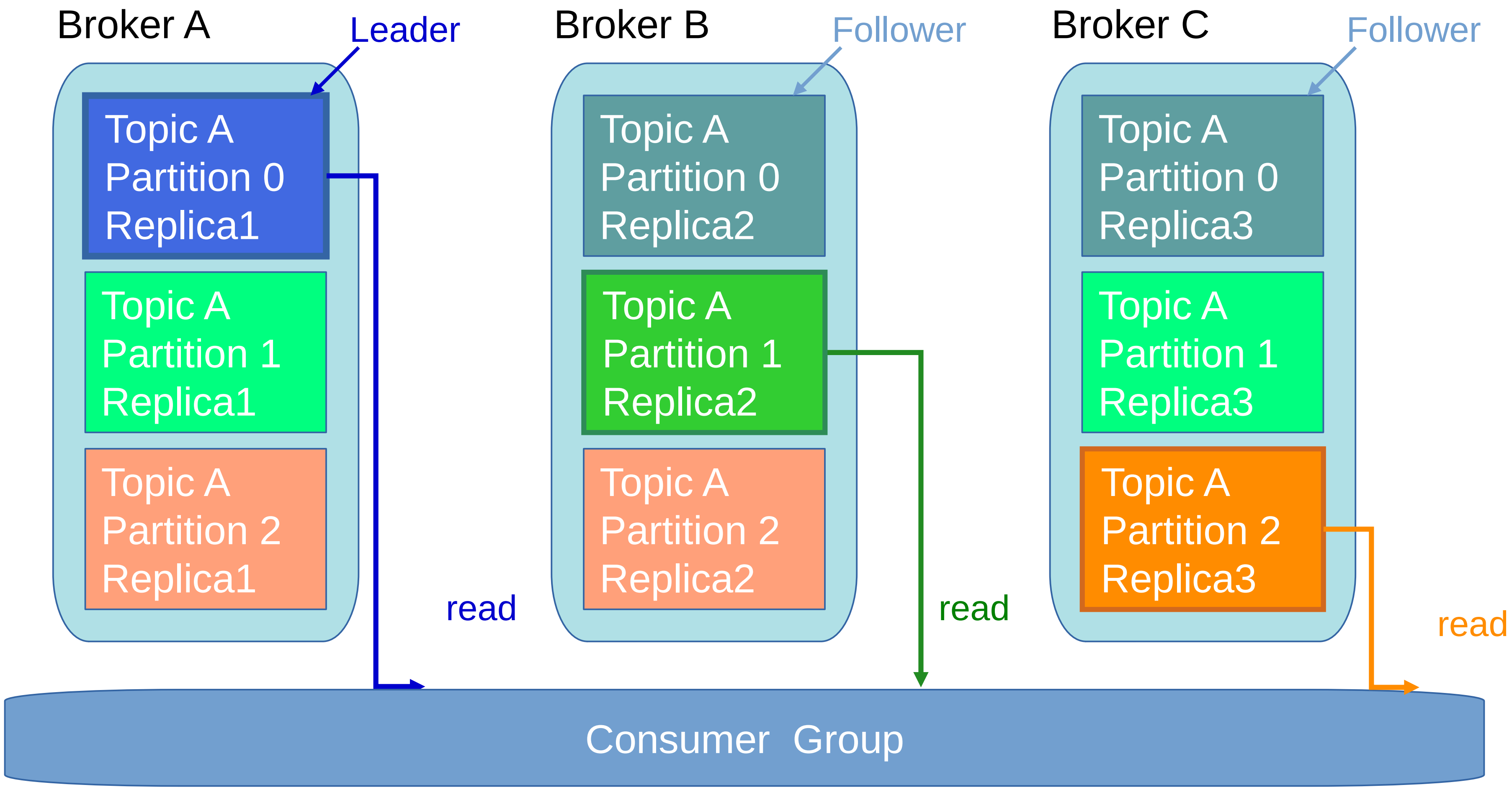
Kafka Partitions and Distribution

❖The partitions in the log serve several purposes:

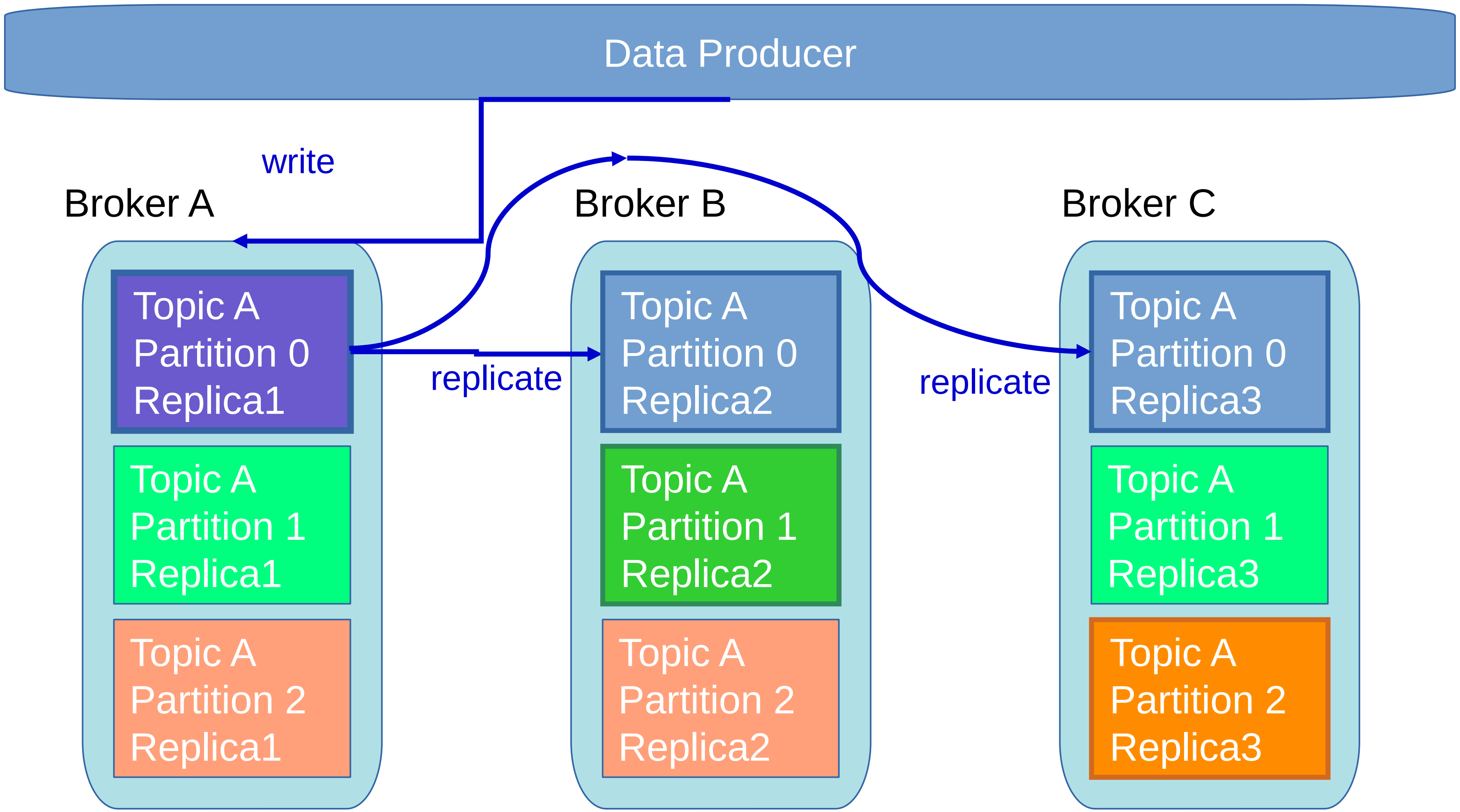
1. Allow the log to scale beyond a size that will fit on a single server. Each individual partition must fit on the servers that host it, but a topic may have many partitions so it can handle an arbitrary amount of data.
2. Act as the unit of parallelism—more on that in next slide



Distribution of Partitions and Read Balancing

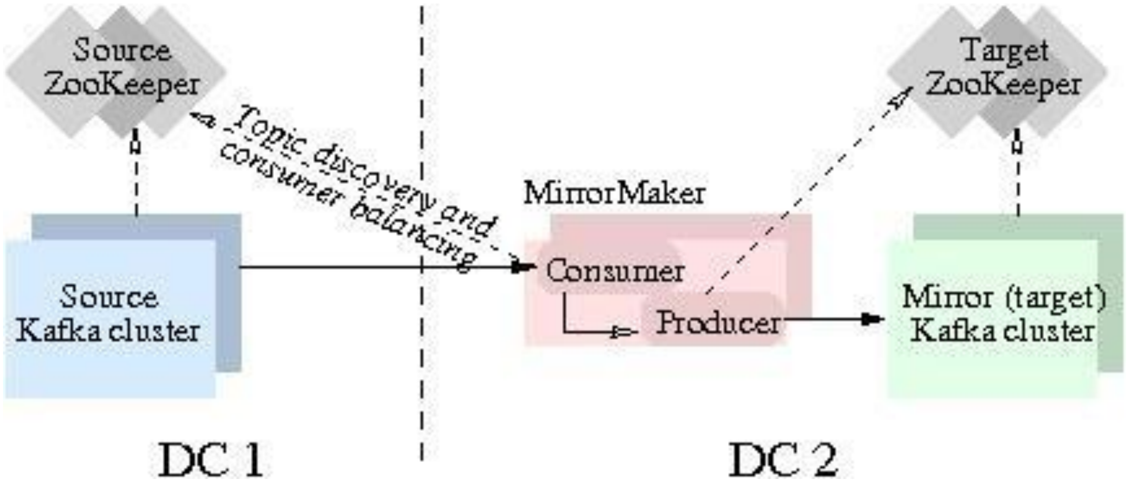


Data Replication



Kafka Geo-Replication

❖Kafka MirrorMaker provides geo-replication support for your clusters. With MirrorMaker, messages are replicated across multiple datacenters or cloud regions. You can use this in active/passive scenarios for backup and recovery; or in active/active scenarios to place data closer to your users, or support data locality requirements.



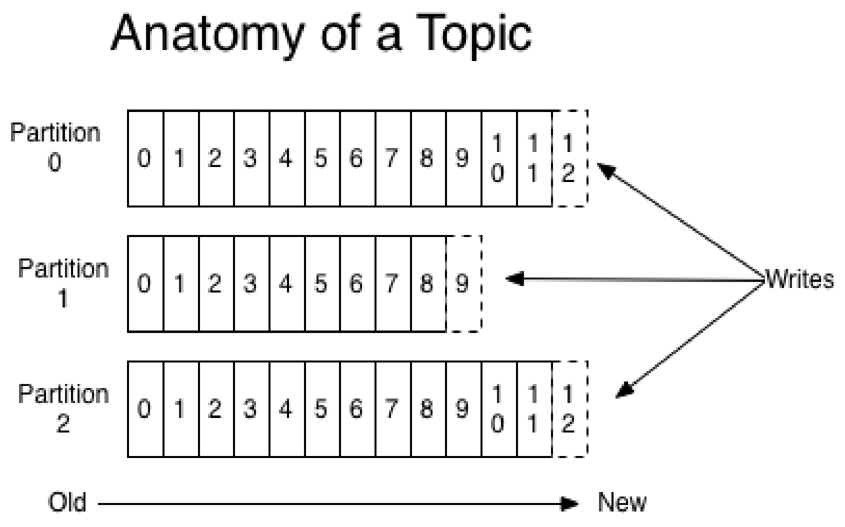
Kafka Brokers

❖Kafka is maintained as clusters where each node within a cluster is called a Broker. Multiple brokers allow us to evenly distribute data across multiple servers and partitions. This load should be monitored continuously and brokers and topics should be reassigned when necessary.

❖Each Kafka cluster will designate one of the brokers as the Controller which is responsible for managing and maintaining the overall health of a cluster, in addition to the basic broker responsibilities. Controllers are responsible for creating/deleting topics and partitions, taking action to rebalance partitions, assign partition leaders, and handle situations when nodes fail or get added. Controllers subscribe to receive notifications from ZooKeeper which tracks the state of all nodes, partitions, and replicas.

Kafka Producers

❖ Producers publish data to the topics of their choice. The producer is responsible for choosing which record to assign to which partition within the topic. This can be done in a round-robin fashion simply to balance load or it can be done according to some semantic partition function (e.g. based key's hash value)

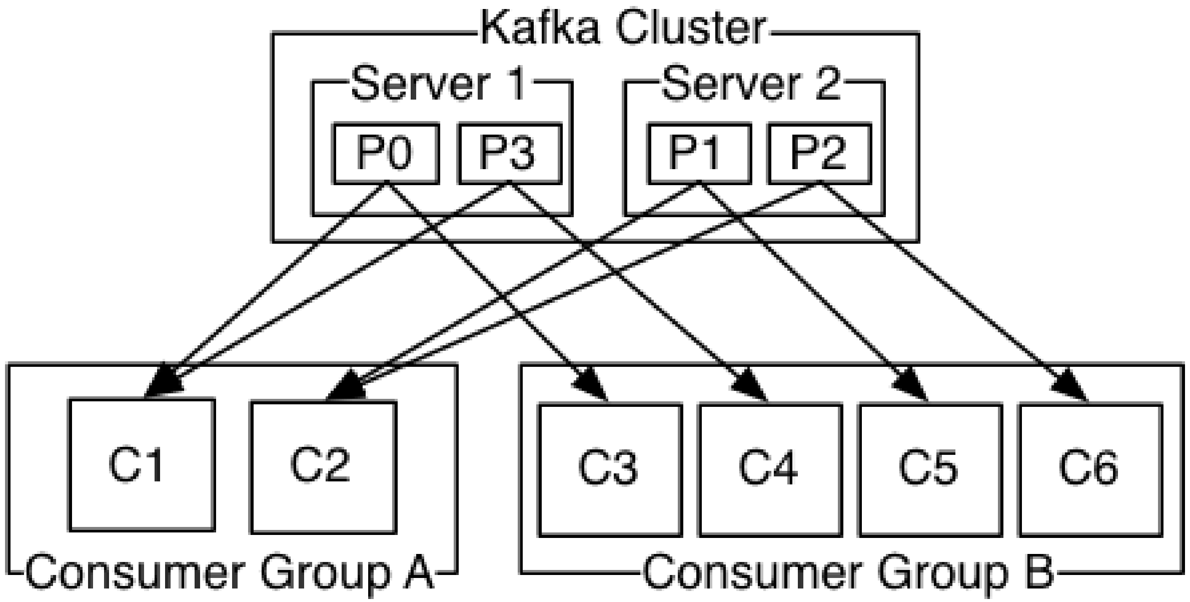


Kafka Consumers

❖ Consumers label themselves with a consumer group, and each record published to a topic is delivered to one consumer instance within each consumer group. Can be separate processes/machines

❖ Same consumer group --> records will effectively be load balanced

❖ Different consumer groups, --> broadcast to all the consumers



Kafka Consumer Groups

- Consumer group = "logical subscriber" -> many consumer instances for scalability and fault tolerance = publish-subscribe semantics where the subscriber is a cluster of consumers instead of a single process.
- The way consumption is implemented in Kafka is by dividing up the partitions in the log over the consumer instances so that each instance is the exclusive consumer of a "share" of partitions at any point in time.
- Group membership is handled by the Kafka protocol dynamically. If new instances join the group they will take some partitions from other group members; if an instance dies, its partitions will be distributed.
- Kafka only provides a total order over records within a partition, not between different partitions in a topic. Per-partition ordering combined with the ability to partition data by key is sufficient for most apps.
- If you require a total order over records --> use a topic that has only one partition = only one consumer process per consumer group.

Kafka Messages (Records)

❖ Records are messages that contain a key/value pair along with metadata such as a timestamp, message key, and headers. Headers may store application-specific metadata:

- length: varint
- attributes: int8 (bit 0~7: unused)
- timestampDelta: varint
- offsetDelta: varint
- keyLength: varint
- key: byte[]
- valueLen: varint
- value: byte[]
- Headers => [Header]

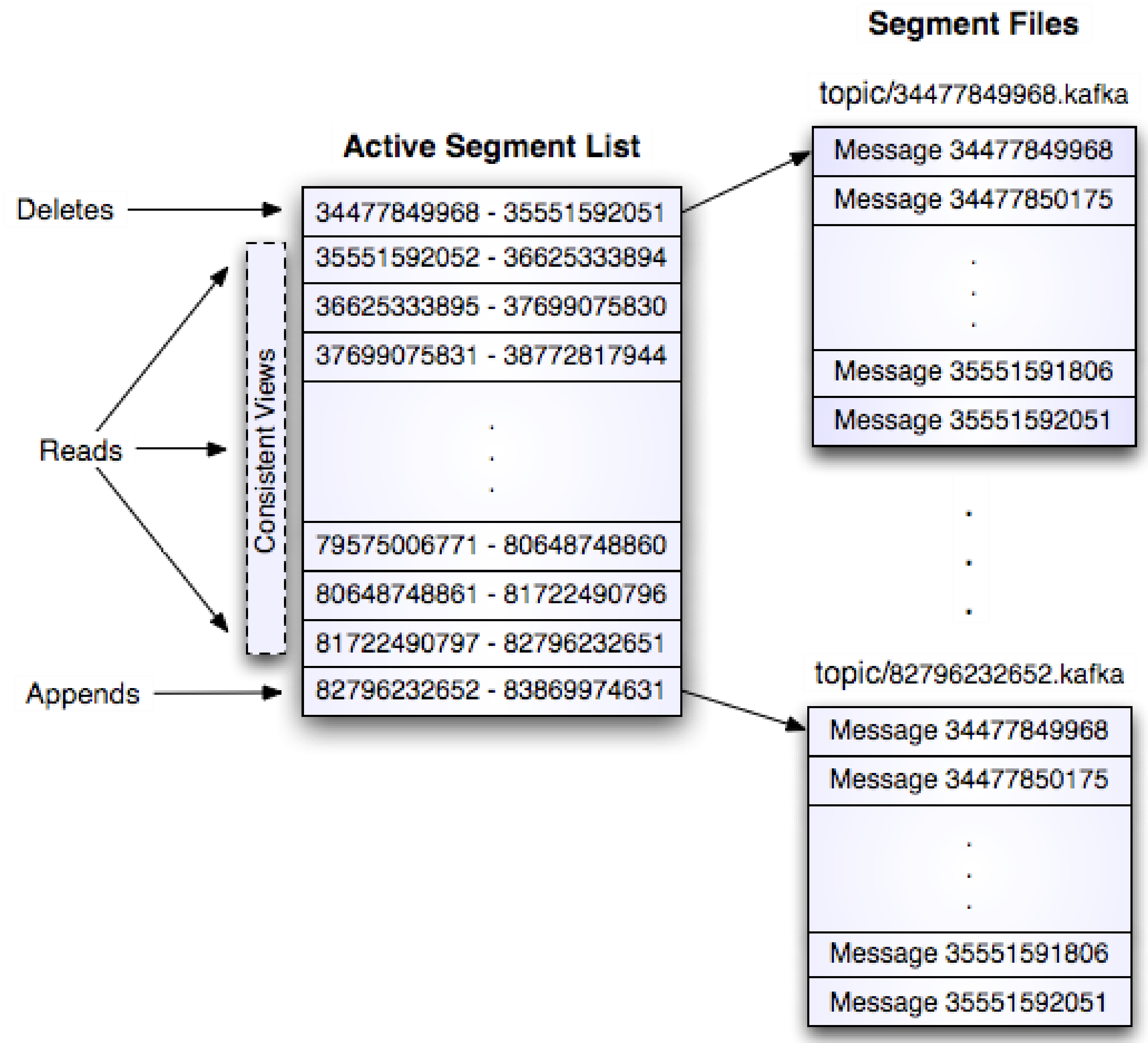
❖ Record Header - in the context of the header, keys are strings and values are byte arrays:

- headerKeyLength: varint
- headerKey: String
- headerValueLength: varint
- Value: byte[]

Kafka Log Format

- Messages are stored inside topics within a log structured format, where the data gets written sequentially.
- A message can have a maximum size of 1MB by default, and while this is configurable, Kafka was not designed to process large size records. It is recommended to split large payloads into smaller messages, using identical key values so they all get saved in the same partition as well as assigning part numbers to each split message in order to reconstruct it on the consumer.
- Messages (aka Records) are always written in record batches, Record batches and records have their own headers. The detailed format of each is described in: http://kafka.apache.org/documentation/#recordbatch

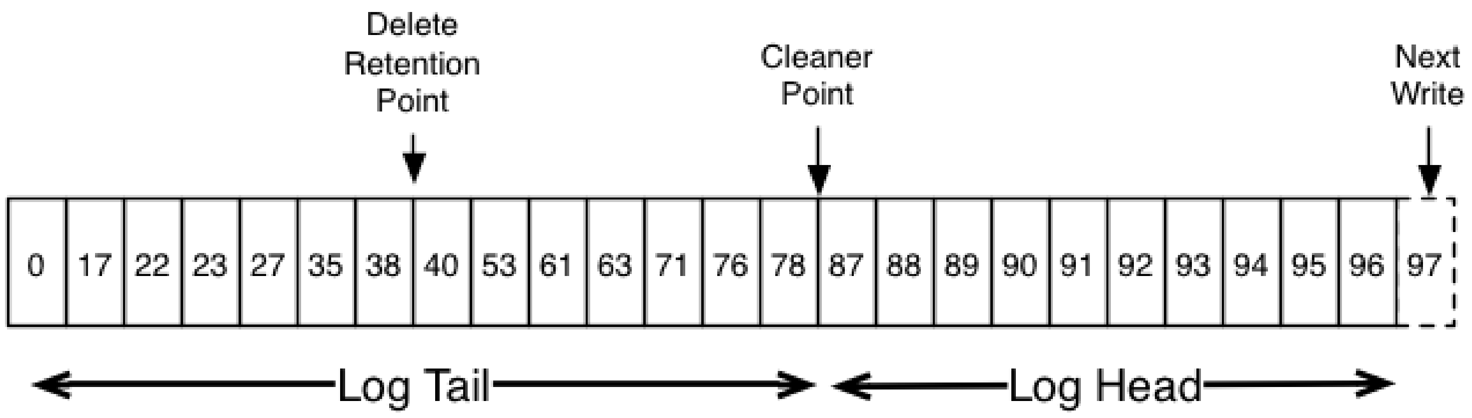
Kafka Log Implementation



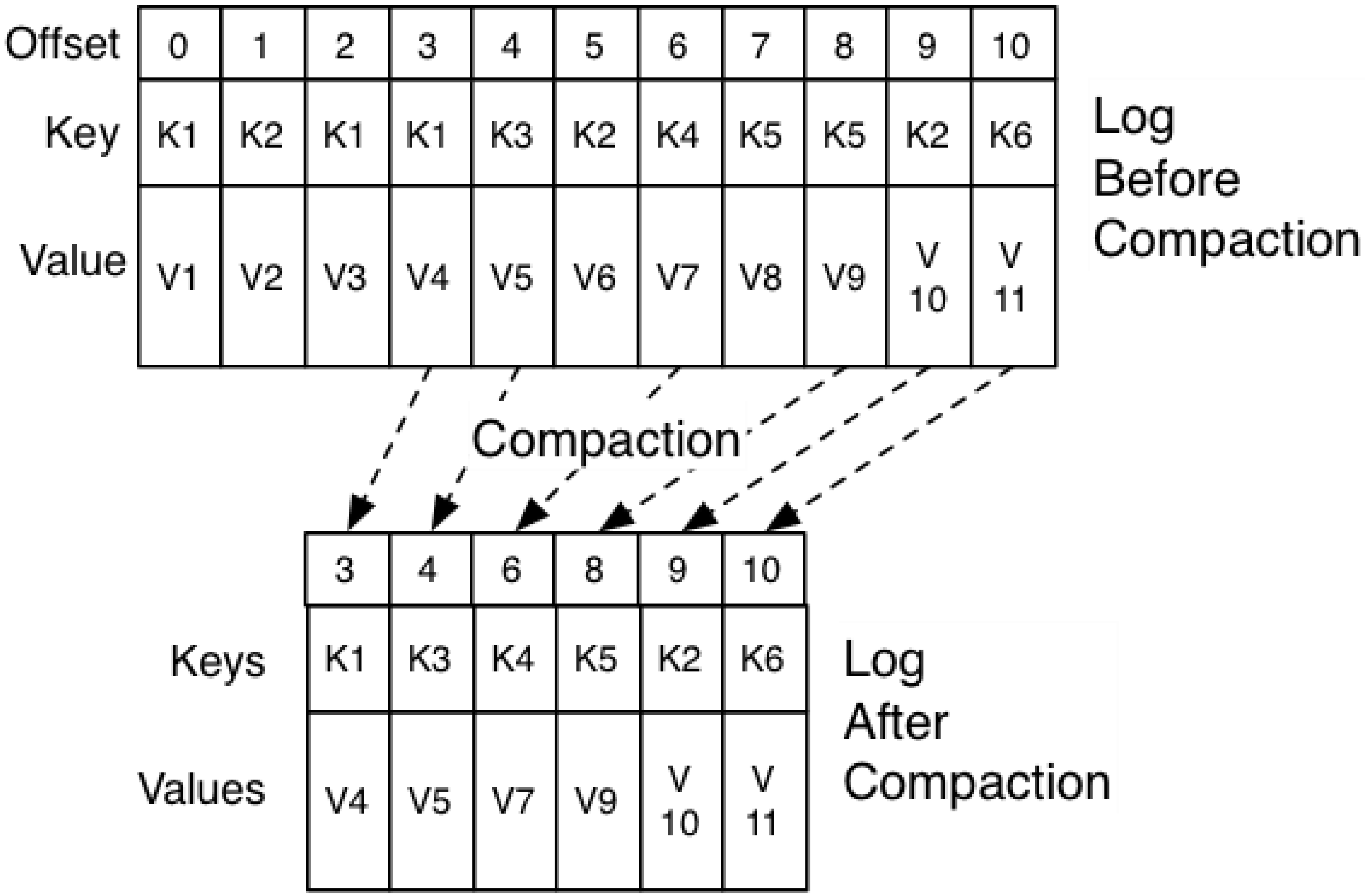
Log Compaction - I

- Log compaction ensures that Kafka will always retain at least the last known value for each message key within the log of data for a single topic partition.
- It addresses use cases and scenarios such as restoring state after application crashes or system failure, or reloading caches after application restarts during operational maintenance – e.g. database change subscription, event sourcing, journaling for high-availability.
- An important class of data streams are the log of changes to keyed, mutable data (for example, the changes to a DB table).
- Example: 123 => bill@microsoft.com; ...
- 123 => bill@gatesfoundation.org …
- 123 => bill@gmail.com ...

Log Compaction - II



Log Compaction - III



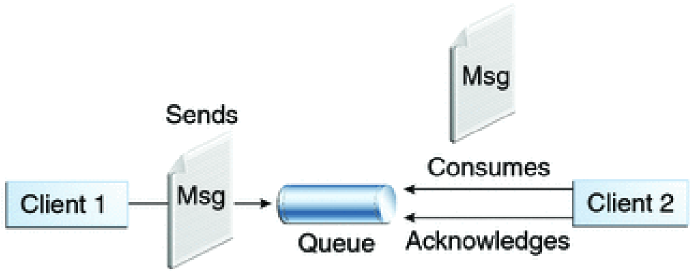
Kafka Guarantees

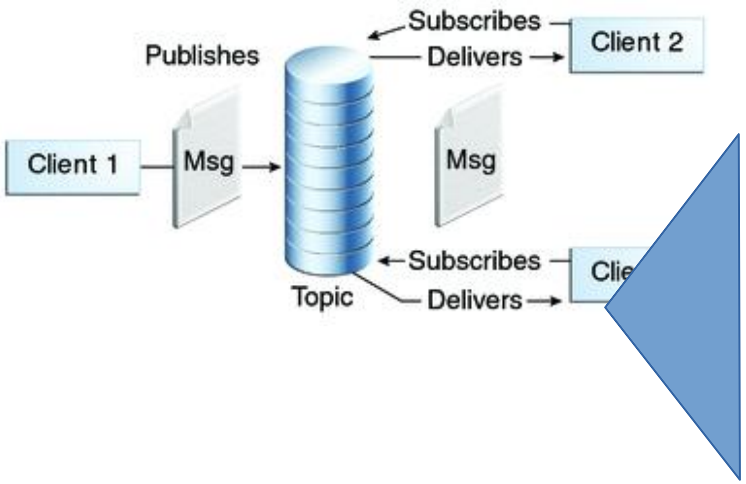
1. Messages sent by a producer to a particular topic partition will be appended in the order they are sent. That is, if a record M1 is sent by the same producer as a record M2, and M1 is sent first, then M1 will have a lower offset than M2 and appear earlier in the log.

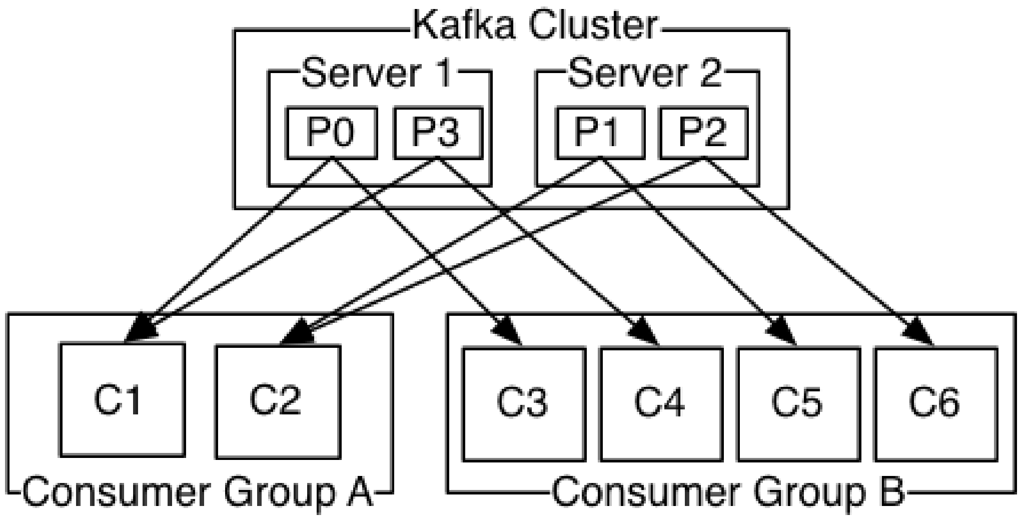
2. A consumer instance sees records in the order they are stored in the log.

3. For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any records committed to the log.

Kafka as a Messaging System







Kafka as a Storage System

- ❖ Any message queue that allows publishing messages decoupled from consuming them is effectively acting as a storage system for the in-flight messages. Kafka is a very good storage system.
- ❖ Data written to Kafka is written to disk and replicated for fault-tolerance. Kafka allows producers to wait on acknowledgement so that a write isn't considered complete until it is fully replicated and guaranteed to persist even if the server written to fails.
- ❖ The disk structures Kafka uses scale well – Kafka will perform the same whether you have 50 KB or 50 TB of persistent data.
- ❖ Efficient storage + allowing the clients to control their read position => Kafka becomes a special purpose distributed filesystem dedicated to high-performance, low-latency commit log storage, replication, and propagation.

Kafka Connect

- ❖Kafka Connect is a framework for importing data into Kafka from external data sources or exporting data to external sources like databases and applications.
- ❖Connectors allow you to move large amounts of data in and out of Kafka to many common external data sources, and also provides a framework for creating your own custom connectors.
- ❖Kafka connect comes with the standard Kafka download, although it requires separate setup.

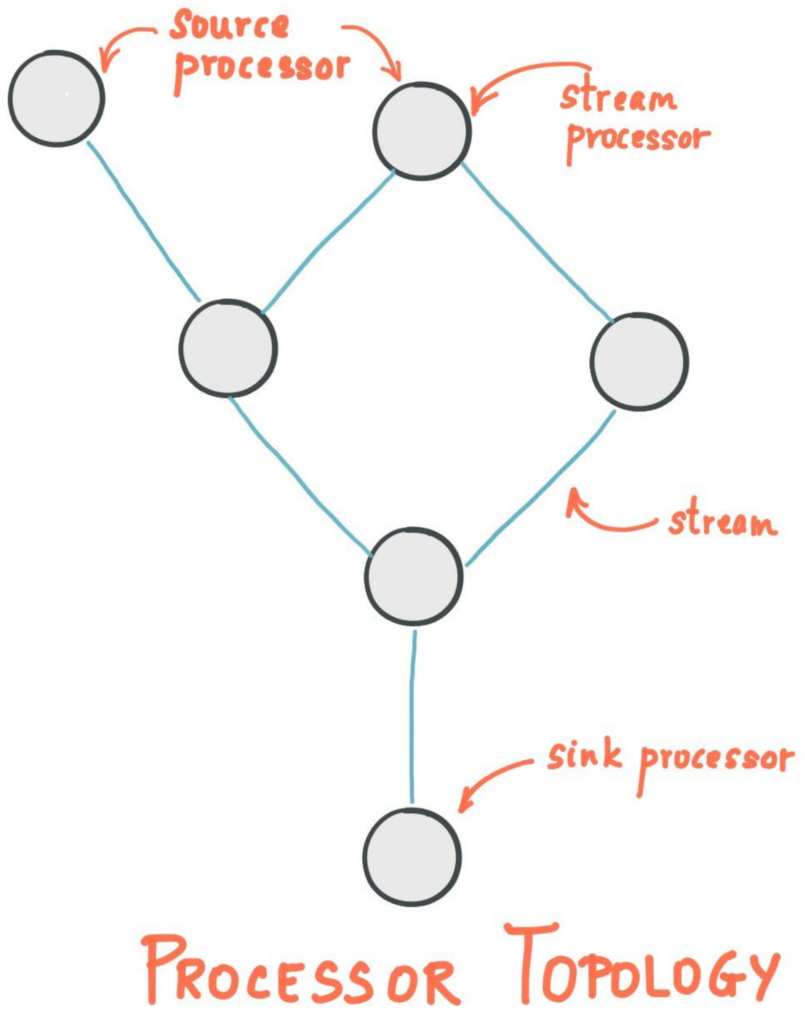
Kafka Stream Processing - I

- By combining storage and low-latency subscriptions, streaming applications can treat both past and future data the same way. That is a single application can process historical, stored data but rather than ending when it reaches the last record it can keep processing as future data arrives. This is a generalized notion of stream processing that subsumes batch processing as well as message-driven applications ==> Kappa architecture
- Likewise for streaming data pipelines the combination of subscription to real-time events make it possible to use Kafka for very low-latency pipelines; but the ability to store data reliably make it possible to use it for critical data where the delivery of data must be guaranteed or for integration with offline systems that load data only periodically or may go down for extended periods of time for maintenance. The stream processing facilities make it possible to transform data as it arrives.

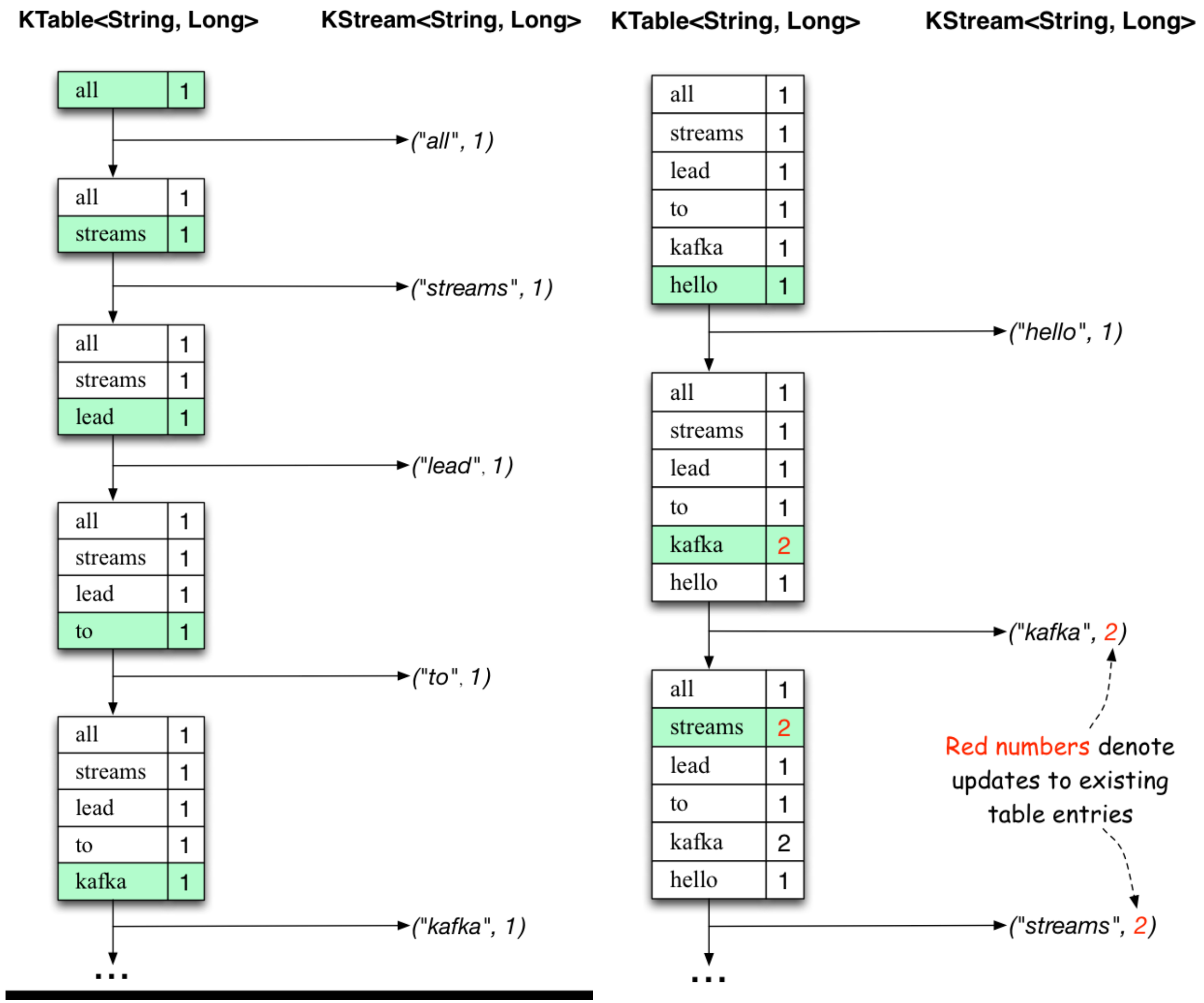
Kafka Stream Processing - II

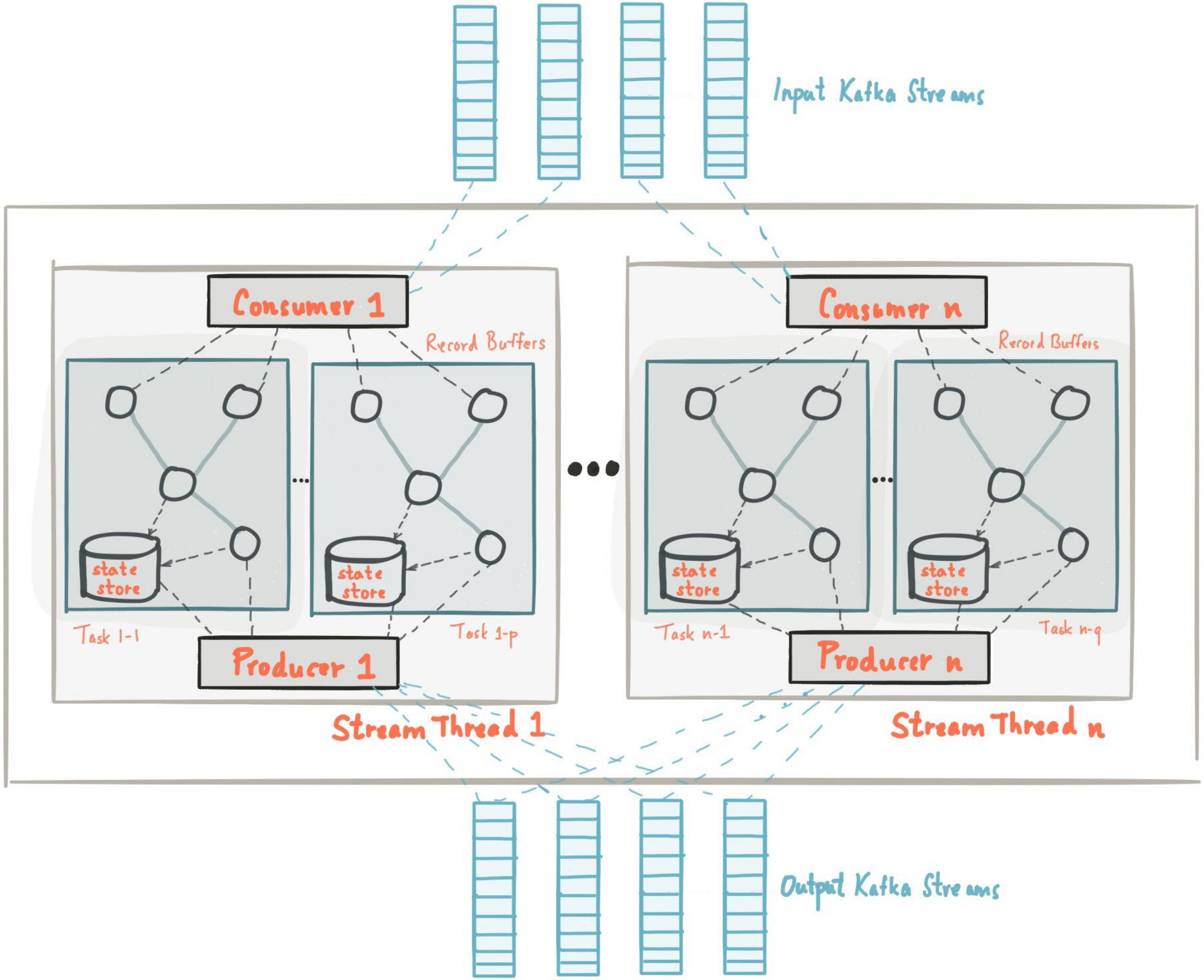
- Kafka streams read data from a topic, running some form of analysis or data transformation, and finally writing the data back to another topic or shipping it to an external source.
- Streams we can achieve real-time stream processing rather than batch processing. For the majority of cases, it's recommended to use Kafka Streams Domain Specific Language (DSL) to perform data transformations (e.g. map, filter, join, aggregations).
- Stream processors are independent of Kafka Producers, Consumers, and Connectors.
- Kafka offers a streaming SQL engine called KSQL for working with Kafka Streams in a SQL-like manner without having to write code like Java. KSQL allows you to transform data within Kafka streams such as preparing the data for processing, running analytics and monitoring, and detecting anomalies in real-time.

Kafka Stream Processing - III



Kafka Stream Processing Example




Input Kafka Streams
Consumer 1
Record Buffers
state store
state store
Task 1-1
Task 1-p
Producer 1
Stream Thread 1
Consumer n
Record Buffers
state store
state store
Task n-1
Task n-q
Producer n
Stream Thread n
Output Kafka Streams

ZooKeeper

- ZooKeeper maintains metadata for all brokers, topics, partitions, and replicas. Since the metadata changes frequently, sustaining ZooKeeper's performance and connection to brokers is critical to the overall Kafka ecosystem.
- Since Kafka Brokers are stateless, they rely on ZooKeeper to maintain and coordinate Brokers, such as notifying consumers and producers of the existence of a new Broker or when a Broker has failed, as well as routing all requests to partition Leaders. ZooKeeper can read, write, and observe updates to data as a distributed coordination service.
- Zookeeper maintains the last offset position of each consumer so that a consumer can quickly recover from the last position in case of a failure. ZooKeeper stores the current offset value of each consumer as it acknowledges each message as received so that the consumer can receive the next offset in the partition's sequence.

Using Kafka Command Line Tools

- bin\windows\zookeeper-server-start.bat config\zookeeper.properties
- bin\windows\kafka-server-start.bat config\server.properties
- kafka-topics.bat --list --zookeeper localhost:1281
- kafka-topics.bat --list --bootstrap-server localhost:9092
- kafka-topics.bat --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic my-new-topic
- kafka-topics.bat --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic events
- kafka-topics.bat --list --bootstrap-server localhost:9092
- kafka-topics.bat --describe --bootstrap-server localhost:9092 --topic events
- kafka-console-producer.bat --broker-list localhost:9092 --topic events
- kafka-console-producer.bat --broker-list localhost:9092 --topic events –sync
- kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic events --from-beginning

Replicated Topic Using Kafka

copy config\server.properties config\server-1.properties

copy config\server.properties config\server-2.properties

config/server-1.properties:
  broker.id=1
  listeners=PLAINTEXT://:9093
  log.dirs=D:\\CourseKafka\\kafka_2.12-2.2.1\\kafka-logs-1

config/server-2.properties:
  broker.id=2
  listeners=PLAINTEXT://:9094
  log.dirs=D:\\CourseKafka\\kafka_2.12-2.2.1\\kafka-logs-2

bin\windows\kafka-server-start config\server-1.properties

bin\windows\kafka-server-start config\server-2.properties

kafka-topics --describe --bootstrap-server localhost:9092 --topic my-replicated-topic

wmic process where "caption = 'java.exe' and commandline like '%server-1.properties%'" get processid

taskkill /F /PID pid_number

Kafka Connect Example

- connect-standalone config\connect-standalone.properties config\connect-file-source.properties config\connect-file-sink.properties

- connect-standalone .\src\main\resources\connect-standalone.properties .\src\main\resources\connect-file-source.properties .\src\main \resources\connect-file-sink.properties

Consumers & Consumer Groups

[https://kafka.apache.org/documentation/#basic_ops_consumer_group]

- kafka-consumer-groups --bootstrap-server localhost:9092 –list
- kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group event-consumer

Kafka SSL Security – Prepare Keys

[https://kafka.apache.org/documentation/#security]

- [https://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html]
- keytool -keystore server.keystore.jks -alias localhost -validity 180 -genkey -keyalg RSA
- keytool -list -v -keystore server.keystore.jks
- openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
- keytool -keystore client.truststore.jks -alias CARoot -import -file ca-cert
- keytool -keystore server.truststore.jks -alias CARoot -import -file ca-cert
- keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-file
- openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 180 -CAcreateserial -passin pass:changeit
- keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert
- keytool -keystore server.keystore.jks -alias localhost -import -file cert-signed

Kafka SSL Security - Brokers

[https://kafka.apache.org/documentation/#security]

- ############################ SSL Config ####################################
- listeners=PLAINTEXT://:9092,SSL://:8092

- ssl.endpoint.identification.algorithm=
- ssl.keystore.location=server.keystore.jks
- ssl.keystore.password=changeit
- ssl.key.password=changeit
- ssl.truststore.location=server.truststore.jks
- ssl.truststore.password=changeit
- ssl.client.auth=none
- ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
- ssl.keystore.type=JKS
- ssl.truststore.type=JKS

Kafka SSL Security - Clients

[https://kafka.apache.org/documentation/#security]

- ################################# SSL Config ################################
- bootstrap.servers=localhost:8092

- ssl.endpoint.identification.algorithm=
- security.protocol=SSL
- ssl.truststore.location=client.truststore.jks
- ssl.truststore.password=changeit
- ssl.truststore.type=JKS
- ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1

- openssl s_client -debug -connect localhost:8092 -tls1
- kafka-console-producer --broker-list localhost:8092 --topic test --producer.config config/producer-ssl.properties
- kafka-console-consumer.bat --bootstrap-server localhost:8092 --topic test --from-beginning --consumer.config config/consumer-ssl.properties

Kafka SSL Security - Azure

- https://docs.microsoft.com/en-us/azure/hdinsight/kafka/apache-kafka-ssl-encryption-authentication

Zeta Architecture

Zeta Architecture

- Main characteristics of Zeta architecture:
  - file system (HDFS, S3, GoogleFS),
  - realtime data storage (HBase, Spanner, BigTable),
  - modular processing model and platform (MapReduce, Spark, Drill, BigQuery),
  - containerization and deployment (cgroups, Docker, Kubernetes),
  - Software solution architecture (serverless computing),
- Recommender systems and machine learning
- Business applications and dynamic global resource management (Mesos + Myriad, YARN, Diego, Borg).
- Lab – introduction to Docker, Docker-Compose and Kubernetes.

Thank’s for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD