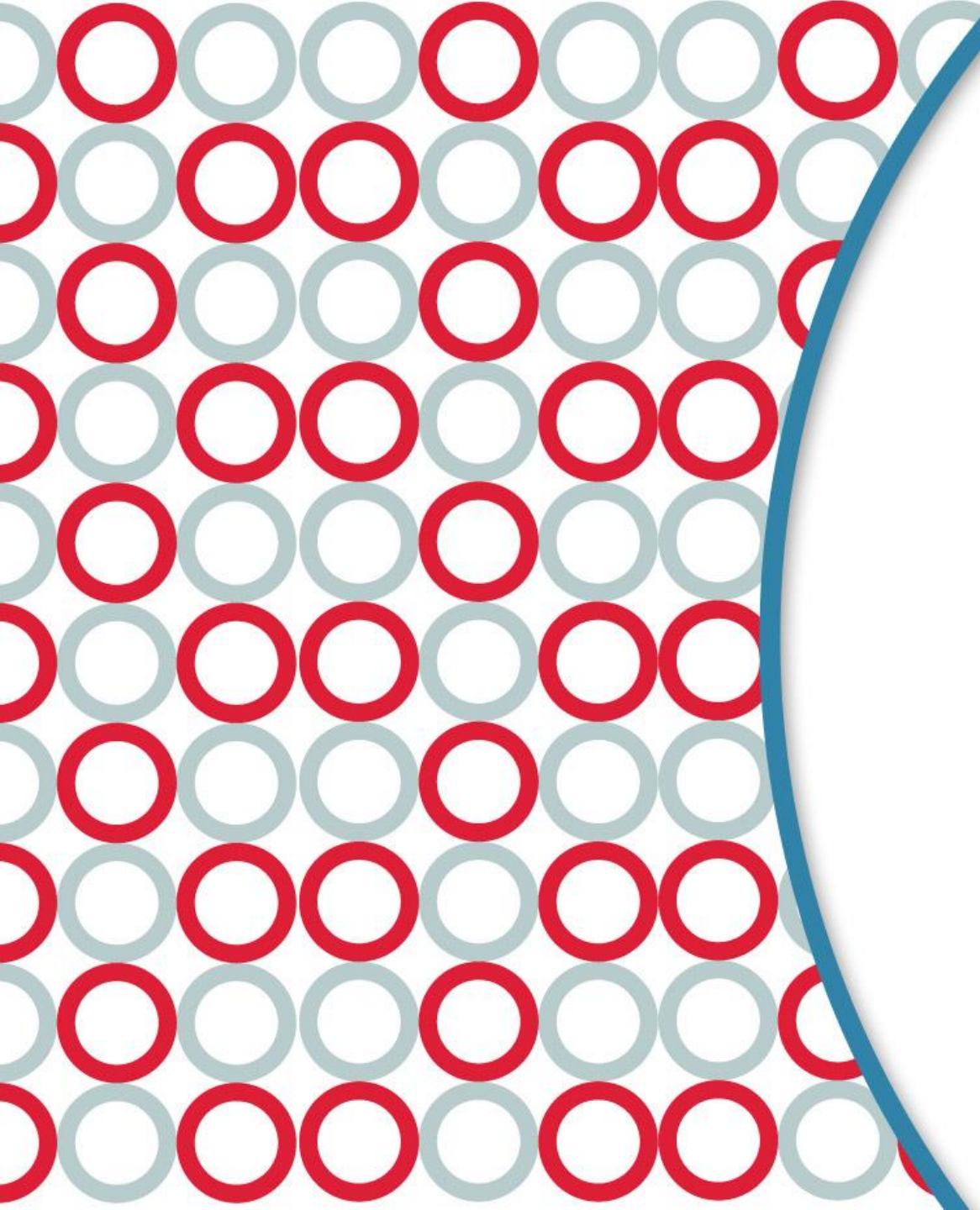


May 2019, IPT Course
Java Web Debelopment

Apache Kafka

Trayan Iliev

tiliev@iproduct.org

http://iproduct.org

About me

Trayan Iliev

– CEO of IPT – Intellectual Products & Technologies

– Oracle certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6/7, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer

– Voxxed Days, jPrime, jProfessionals, BGOUG, BGJUG, DEV.BG speaker, Java2Days

– Organizer RoboLearn hackathons and IoT enthusiast (http://robolearn.org)

Where to Find the Code?

Java Web Development projects and examples are available @ GitHub:

https://github.com/iproduct/course-apache-kafka

Agenda for This Session

- Introduction to Kafka - types of applications and use-cases
- Stream processing architectures
- Kafka Main Concepts and Core APIs
- Topics and Logs
- Consumers Offset and Data Retention
- Kafka Partitions and Distribution
- Brokers and data replication
- Kafka Producers and Consumers
- Kafka Consumer Groups
- Messages (Records)
- Log compaction
- Kafka Streams and Connect
- Zookeeper

Apache Kafka
Distributed Streaming Platform

- Kafka achieves high-throughput, low-latency, durability, and near-limitless scalability by maintaining a distributed system based on commit logs, delegating key responsibility to clients, optimizing for batches and allowing for multiple concurrent consumers per message.
- Publish and subscribe (Pub/Sub) to streams of records – similar to a message queue or enterprise messaging system
- Store streams of records – in a fault-tolerant and durable way
- Process streams of records – as they occur (in real-time)

Two Types of Applications for Kafka

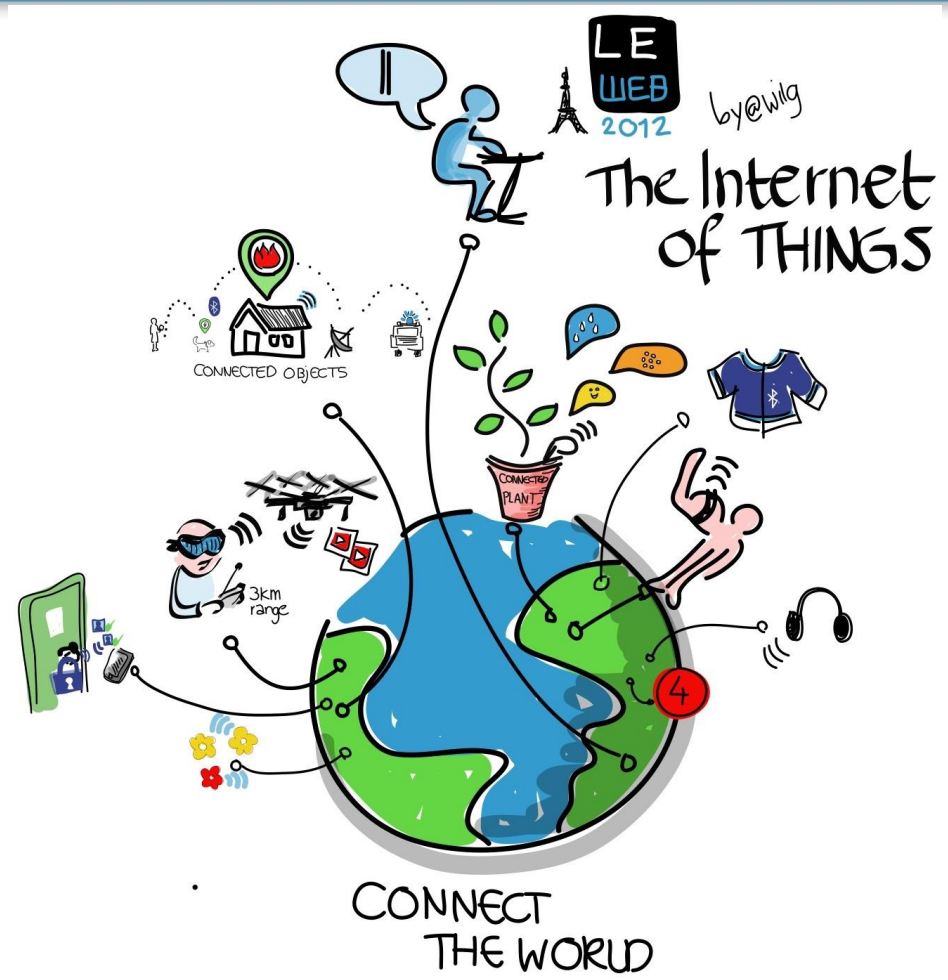
- Building real-time streaming data pipelines that reliably get data between systems or applications

- Building real-time streaming applications that transform or react to the streams of data

Apache Kafka Typical Use-Cases

- IoT, telemetry, and sensor networks
- Positional data / Logistics - supply chain and transportation alerts
- Service/process monitoring - aggregating metrics and logs from distributed servers and applications (Event-driven SOA)
- Real-time analytics, fraud detection – processing of business/customer events in real time
- Click stream analytics, real-time predictive analytics
- Stock-trading analysis

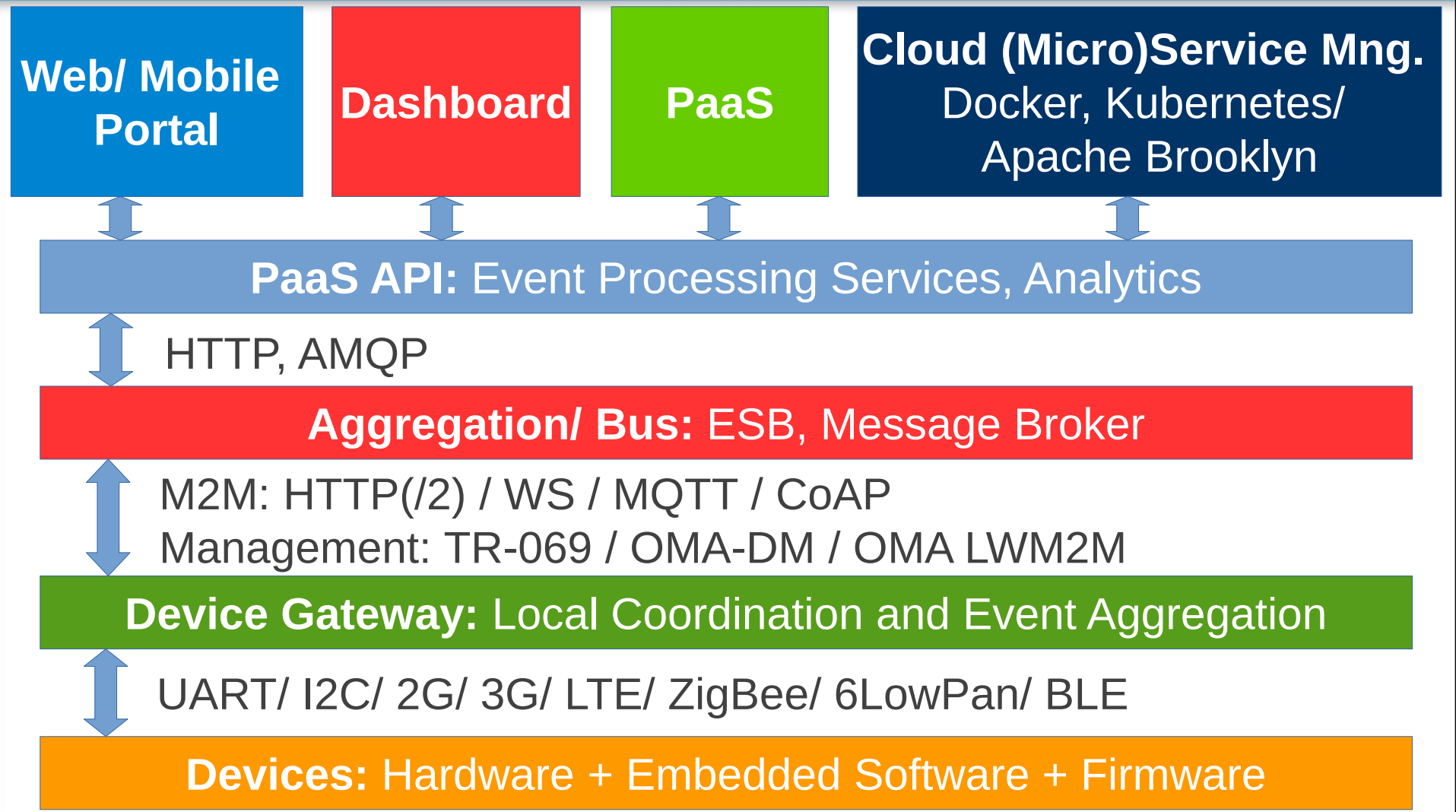
Example: Internet of Things (IoT)



CC BY 2.0, Source: https://www.flickr.com/photos/wilgengebroed/8249565455/

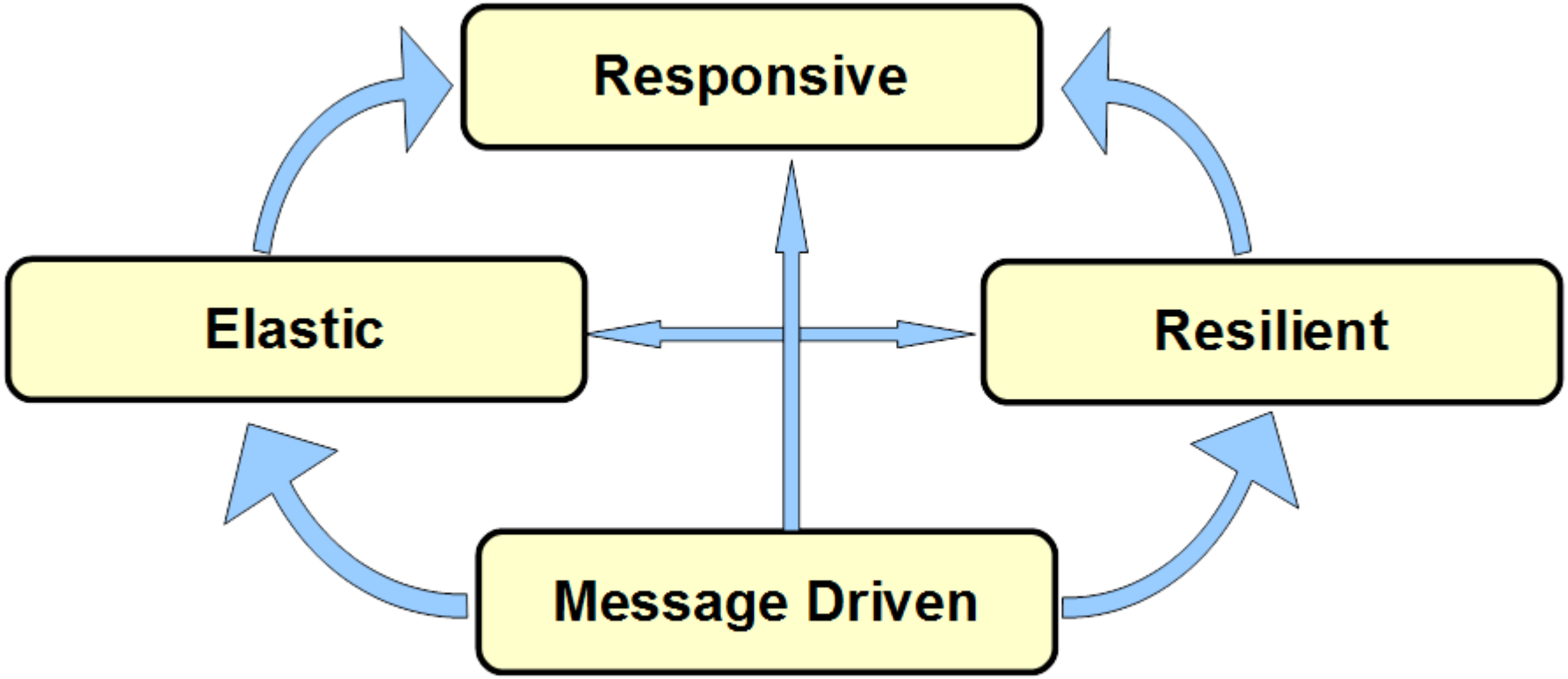
Radar, GPS, lidar for navigation and obstacle avoidance (2007 DARPA Urban Challenge)

IoT Services Architecture



Reactive Manifesto

[http://www.reactivemanifesto.org]



Scalable, Massively Concurrent

- **Message Driven** – asynchronous message-passing allows to establish a boundary between components that ensures loose coupling, isolation, location transparency, and provides the means to delegate errors as messages [Reactive Manifesto].
- The main idea is to separate concurrent producer and consumer workers by using **message queues.**
- **Message queues** can be **unbounded** or **bounded** (limited max number of messages)
- **Unbounded** message queues can present memory allocation problem in case the producers outrun the consumers for a long period → **OutOfMemoryError**

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

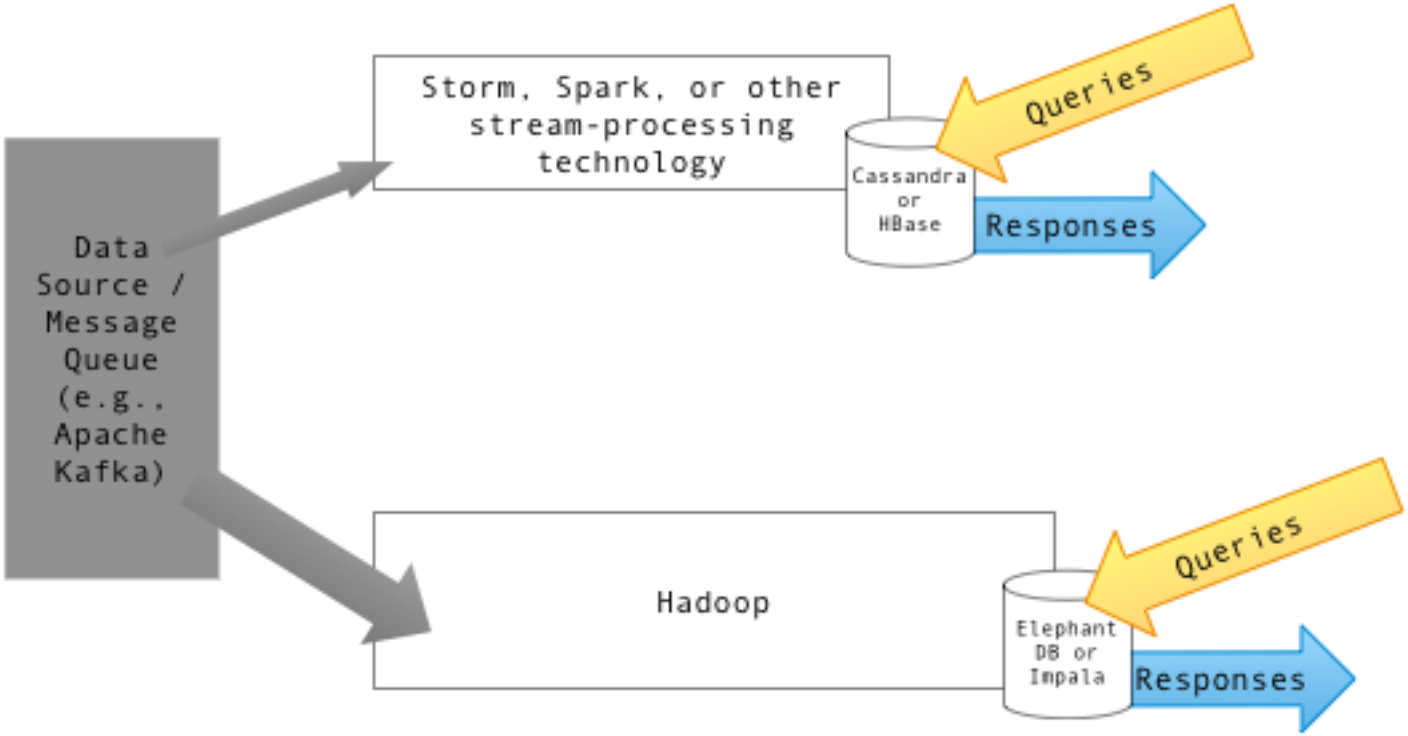
The idea of abstracting logic from execution is hardly new -- it was the dream of SOA. And the recent emergence of microservices and containers shows that the dream still lives on.
For developers, the question is whether they want to learn yet one more layer of abstraction to their coding. On one hand, 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 competition for squashing the Lambda Architecture?

Event Sourcing

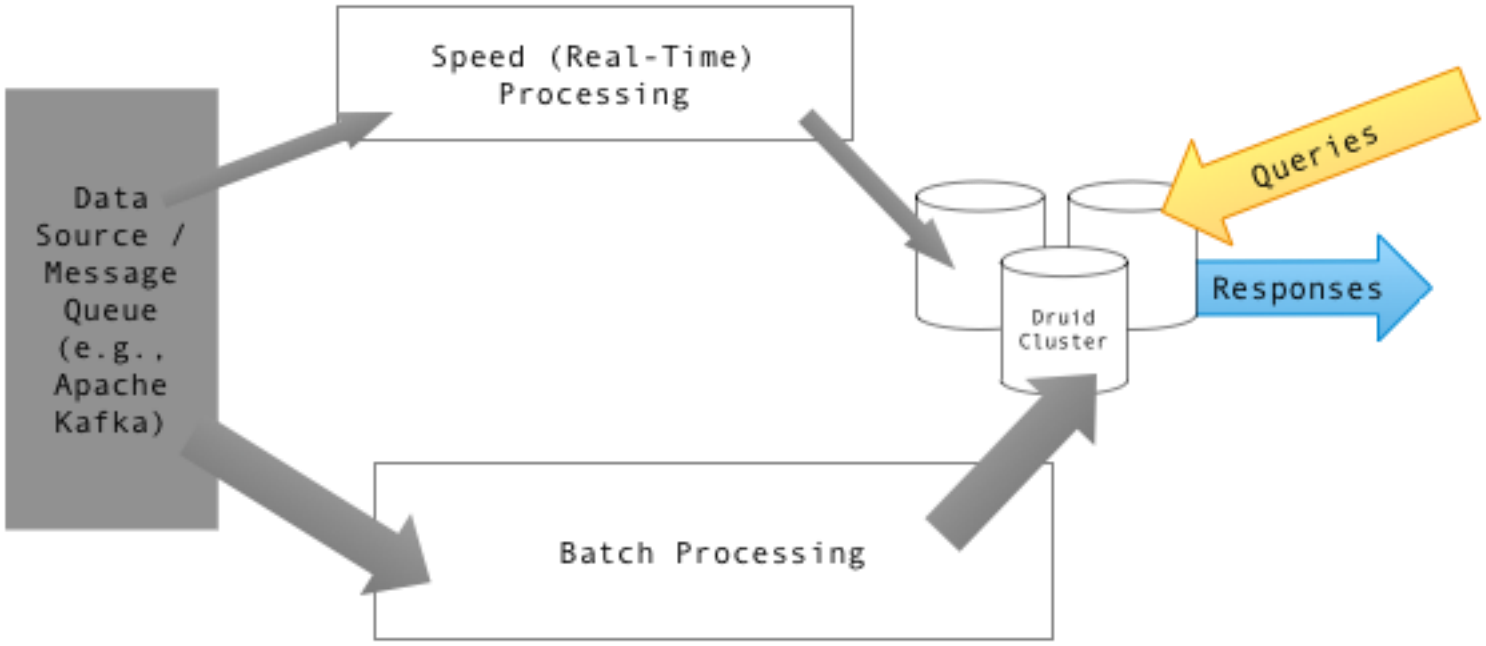
Event sourcing is a concept of using the events to make prediction as well as storing the changes in a system on the real time basis a change of state of a system, an update in the databases or an event can be understood as a change.

Lambda Architecture - I



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

Lambda Architecture - II



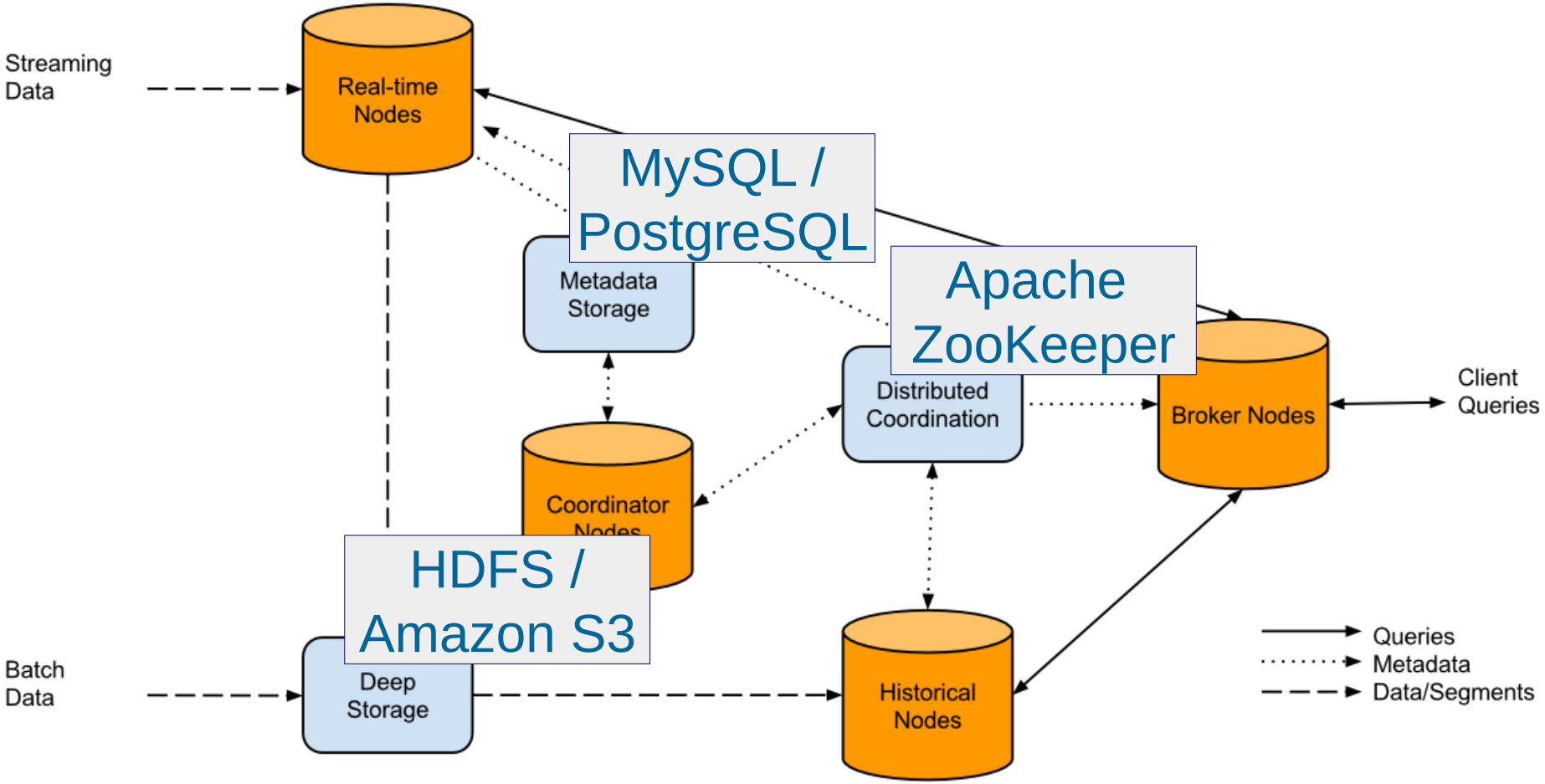
- *Query = λ (Complete data)*
  *= λ (live streaming data) * λ (Stored data)*

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

Lambda Architecture - III

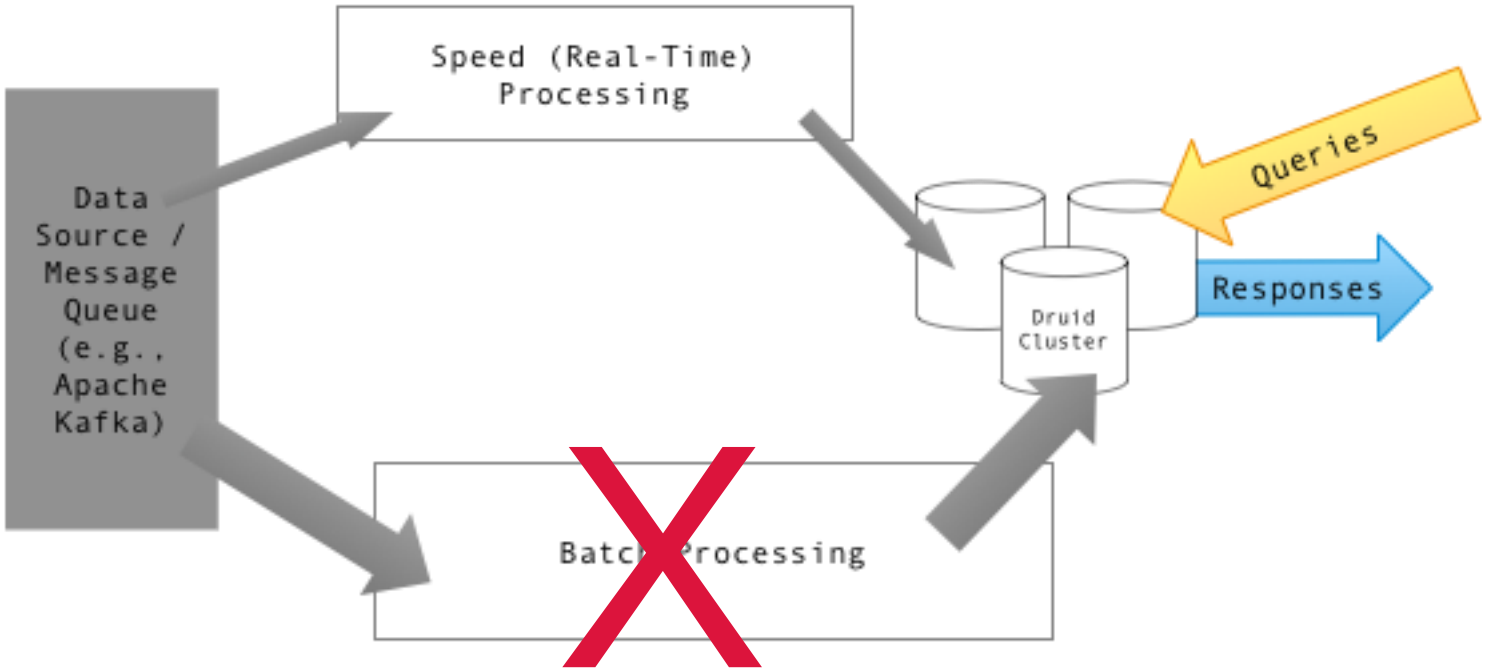
- Data-processing architecture designed to handle massive quantities of data by using both *batch-* and *stream-processing* methods
- Balances *latency, throughput, fault-tolerance, big data, real-time analytics*, mitigates the latencies of map-reduce
- Data model with an append-only, *immutable data* 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.

Data Processing Example



https://commons.wikimedia.org/w/index.php?curid=33899448 By Fangjin Yang, GFDL

Kappa Architecture



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

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

Stream Processing - 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 Samza is a distributed real-time stream processing framework.

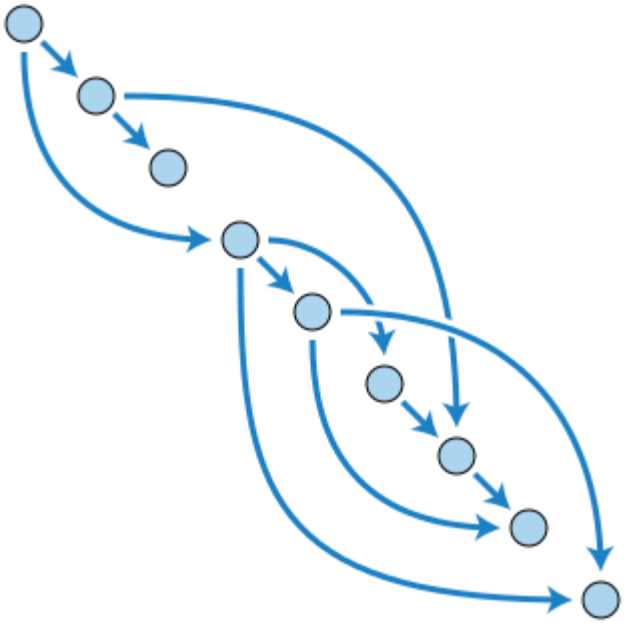
Stream Processing - Projects - II

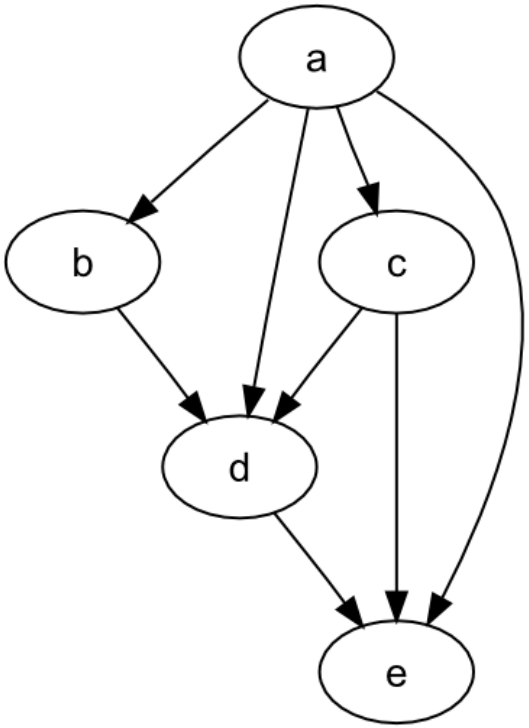
- Apache Flink - open source stream processing framework – Java, Scala

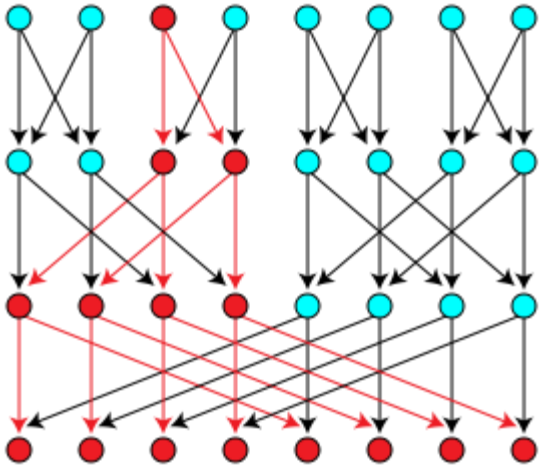
- Apache Kafka - open-source stream processing (Kafka Streams), real-time, low-latency, high-throughput, massively scalable pub/sub

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

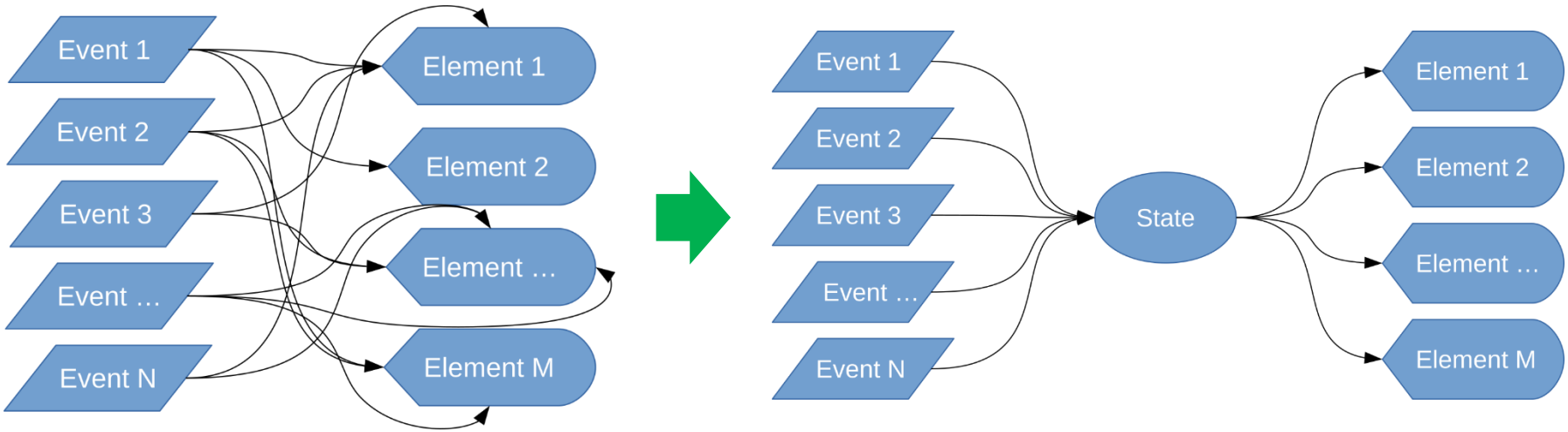
Direct Acyclic Graphs - DAG







Event Sourcing – Events vs. Sate (Snapshots)



What can FP offer to distributed computing ?

- No side-effects and immutable variables – FP facilitates code distribution over several CPU and eases concurrent programming
- Functions are better building components than objects:
  - Functions can be combined, sent remotely
  - Functions can be applied locally on distributed data sets (e.g. parallel stream, using Fork-Join pool underneath).
- In order to do the splitting of the work between multiple threads (forking) the Java Streams use:

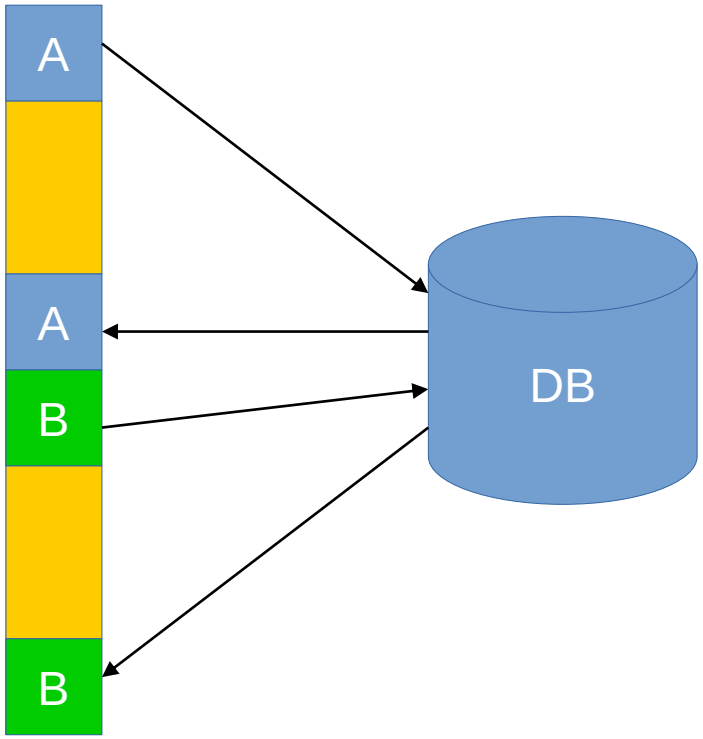
  spliterator = split iterator

- The results can be joined after that in a single result (e.g. reduce)

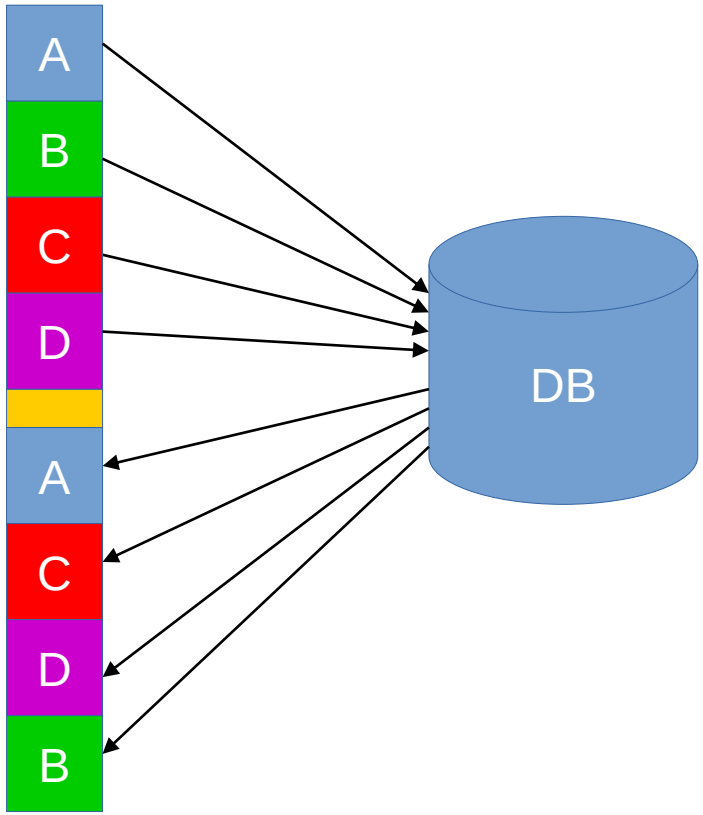
Example: Map – Reduce big data architecture (Google, Hadoop)

Synchronous vs. Asynchronous IO

Synchronous



Asynchronous



What's High Performance?

- **Performance** is about 2 things (Martin Thompson – http://www.infoq.com/articles/low-latency-vp):
  - **Throughput** – units per second, and
  - **Latency** – response time
- **Real-time** – time constraint from input to response regardless of system load.
- **Hard real-time system** if this constraint is not honored then a total system failure can occur.
- **Soft real-time system** – low latency response with little deviation in response time
- **100 nano-seconds** to **100 milli-seconds**. [Peter Lawrey]

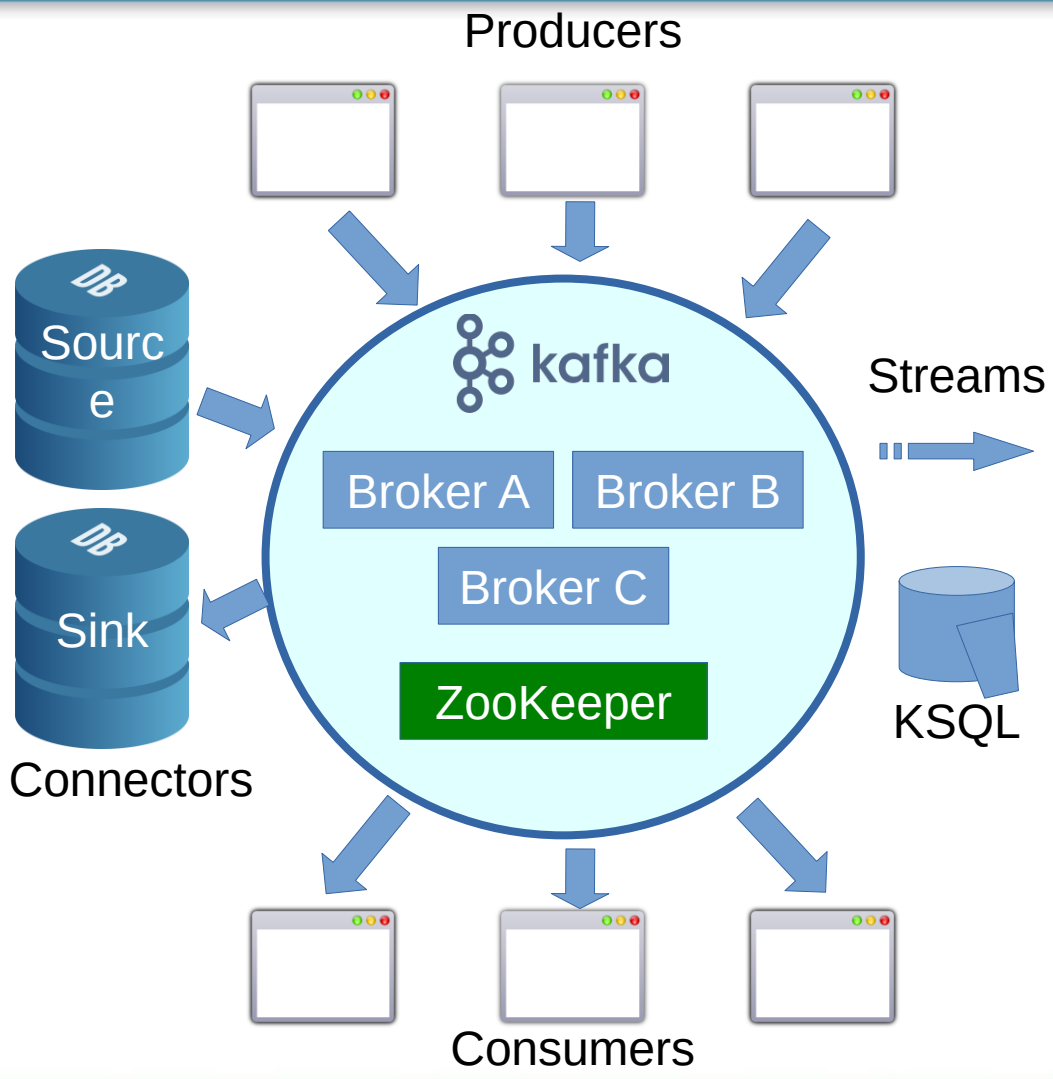
How to Do Async Programming?

- **Callbacks** – asynchronous methods do not have a return value but take an extra callback parameter (a lambda or anonymous class) that gets called when the result is available – **EventListener** pattern
- **Futures, Promises** – asynchronous methods return a **Future<T>** immediately. The value is not immediately available, and the object can be polled Ex.: **Callable<T>** task
- **Streams**:
  - **Composability** and readability
  - **Data as a flow** manipulated with a rich vocabulary of operators
  - **Lazy evaluation** – nothing happens until you subscribe ()
  - **Backpressure** – consumer can signal to producer when rate is high
  - **High level but high value abstraction** that is concurrency-agnostic

Source: https://projectreactor.io/docs/core/release/reference

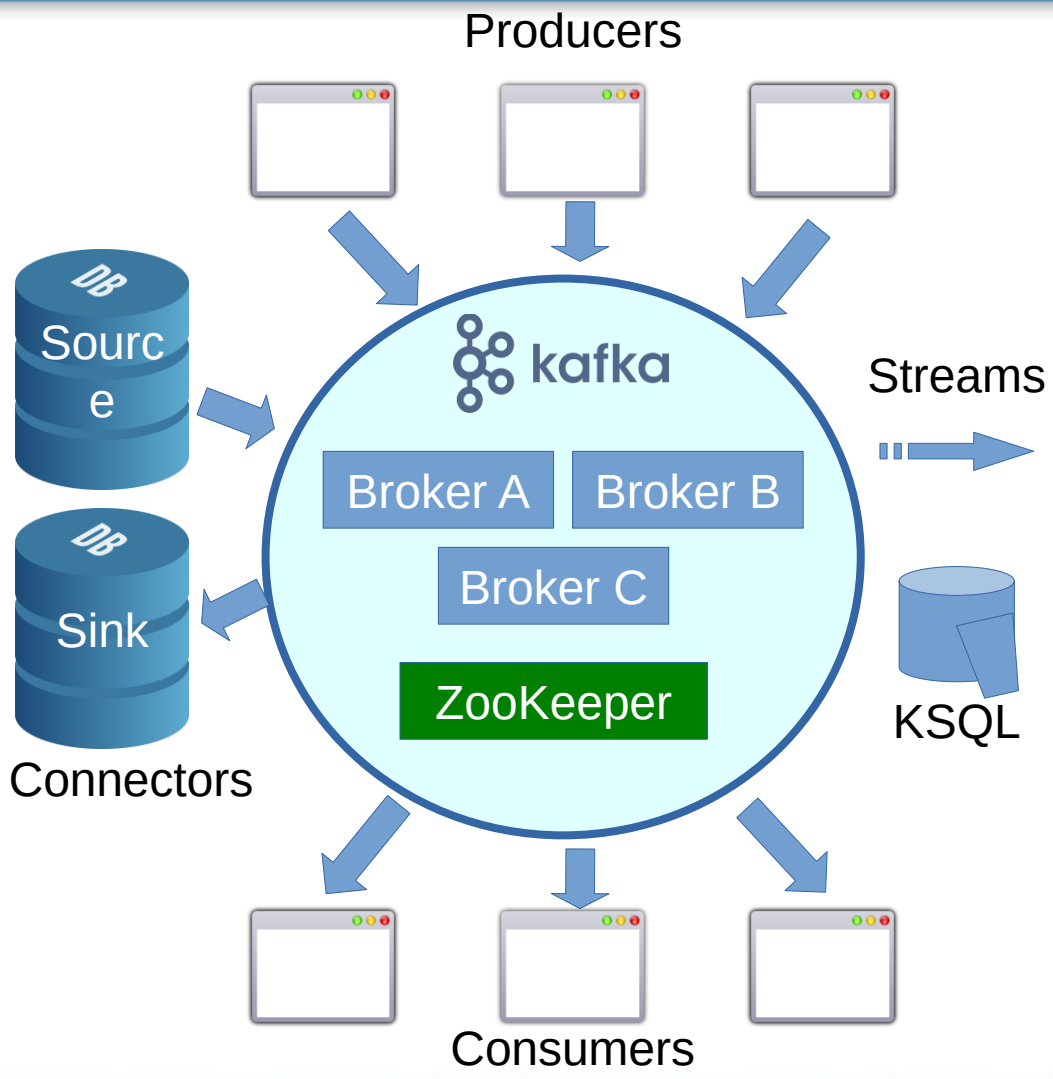
Kafka Main Concepts

- 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.



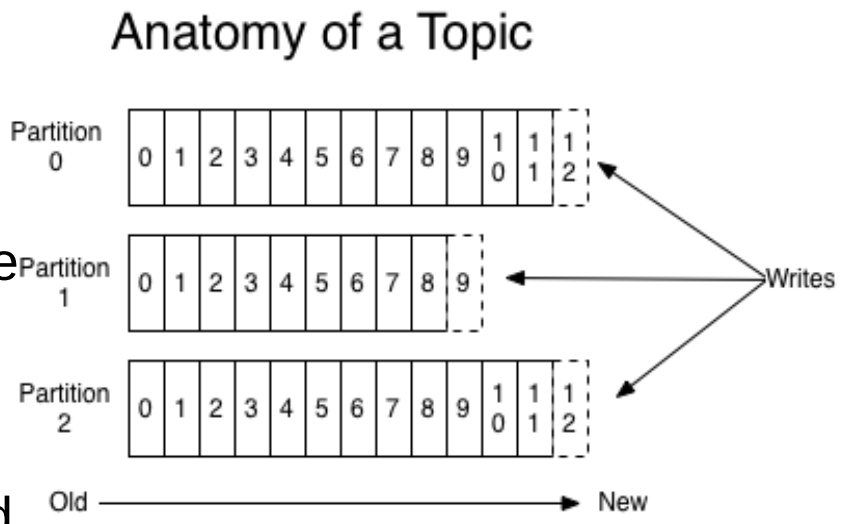
Kafka Core APIs

- 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

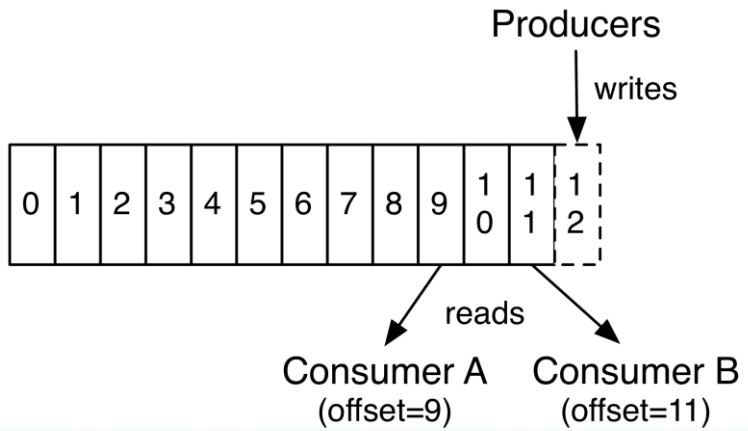
- Topic = stream of records
- A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data
- For each topic, the Kafka cluster maintains a partitioned log --->
- Each partition 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

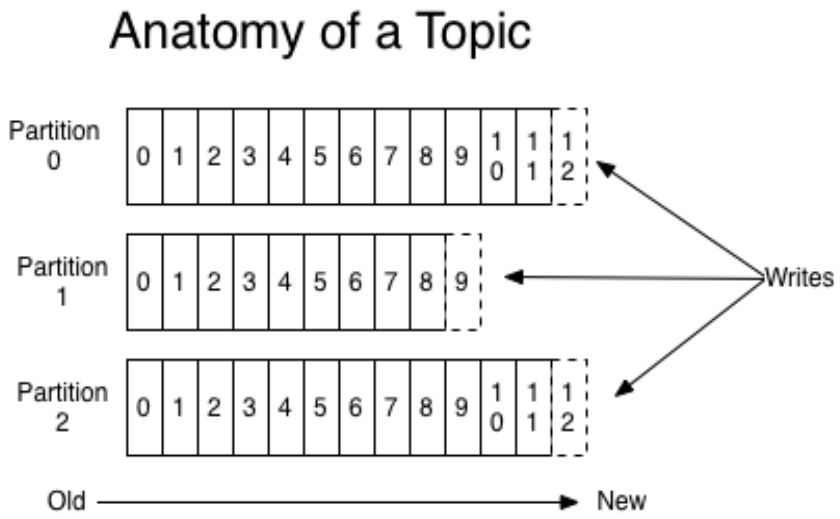
- Kafka cluster durably persists all published records - whether or not they have been consumed, using configurable retention period
- Kafka performance is effectively constant with respect to data size
- Offset or position of that consumer in the log – controlled by the consumer: normally a consumer 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.

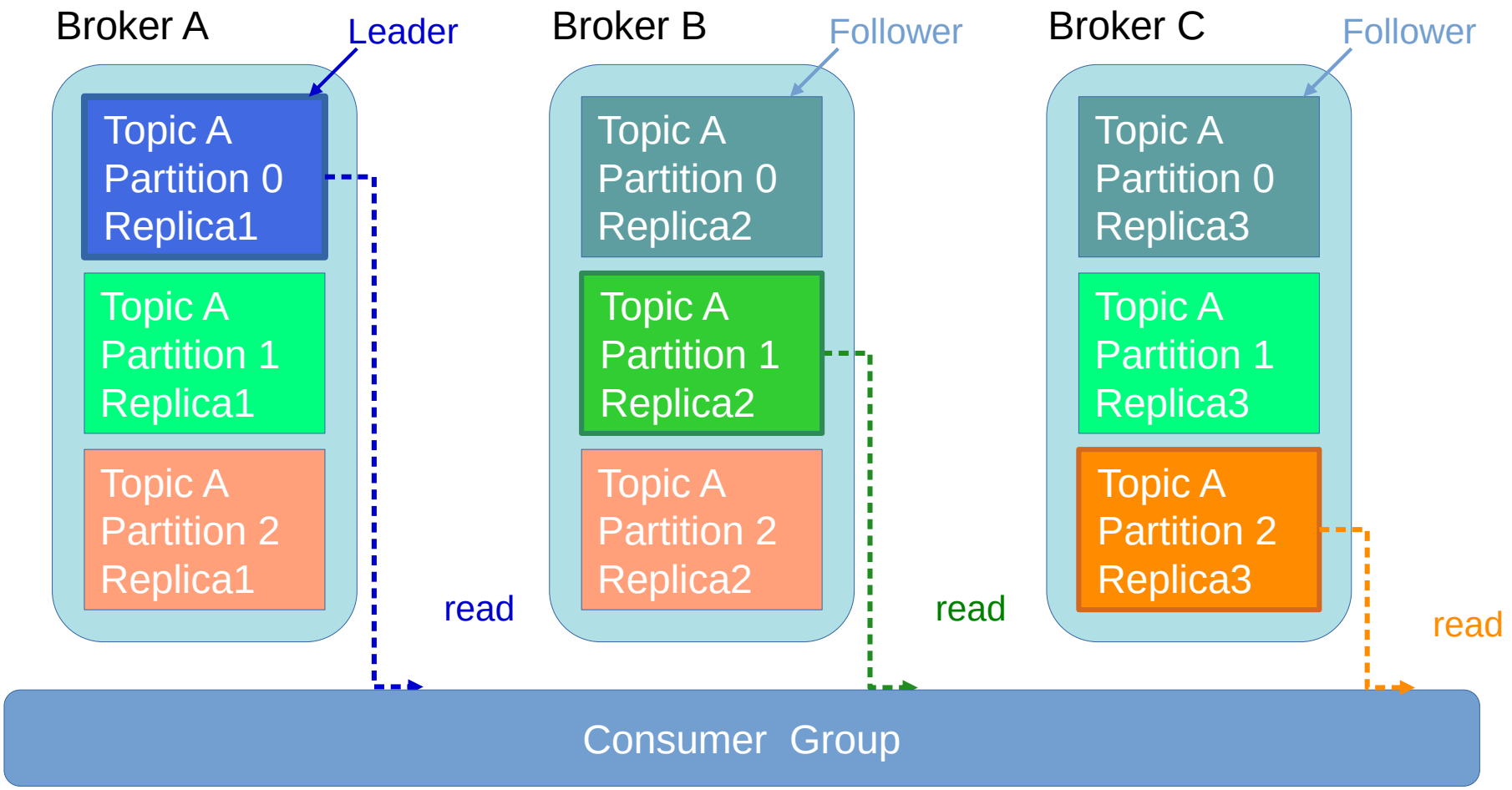


Kafka Partitions and Distribution

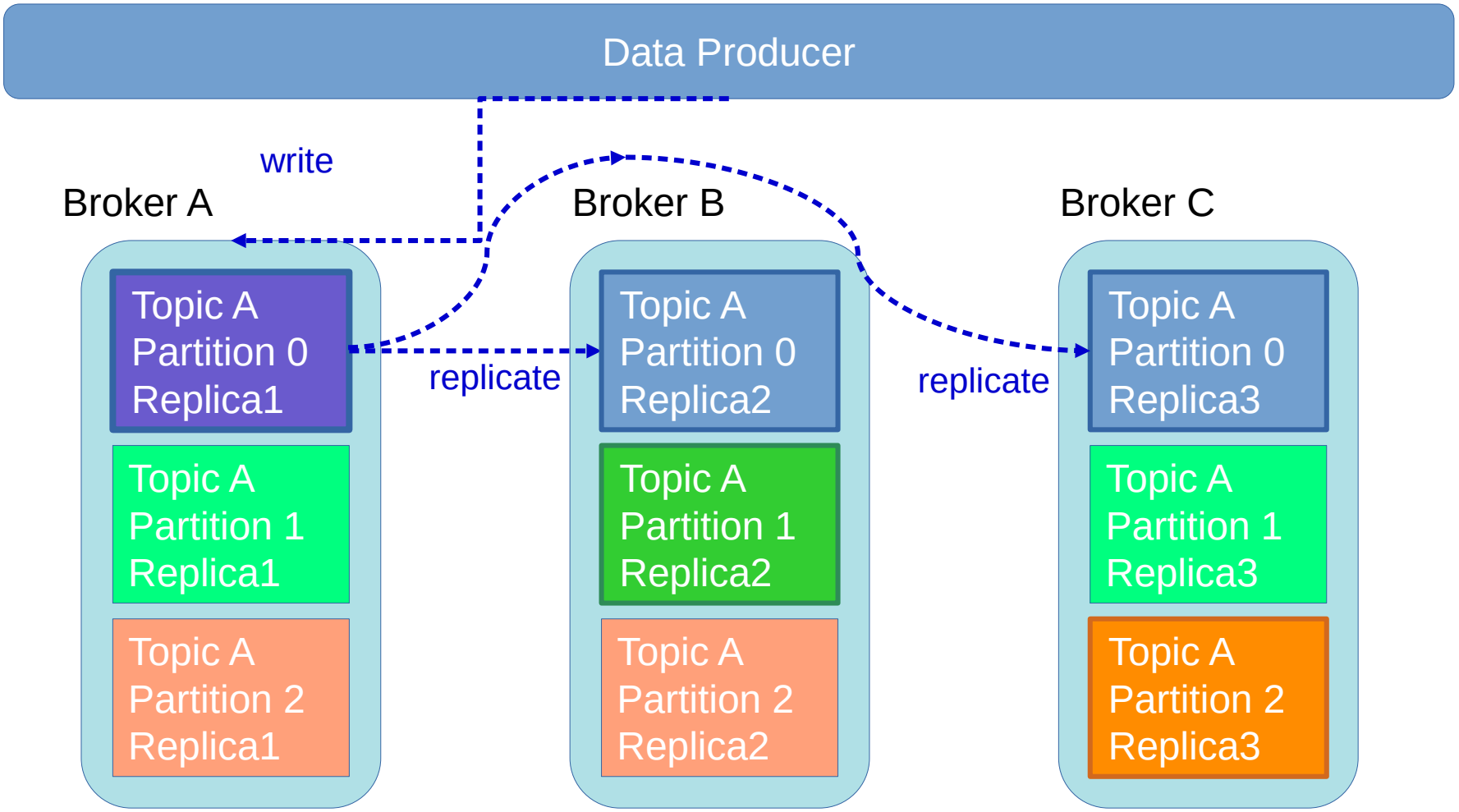
- The partitions in the log serve several purposes:
  1. Allow the log to scale beyond a size that will fit on a single server. Each individual partition must fit on the servers that host it, but a topic may have many partitions so it can handle an arbitrary amount of data.
  2. Act as the unit of parallelism—more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



Distribution of Partitions and Read Balancing

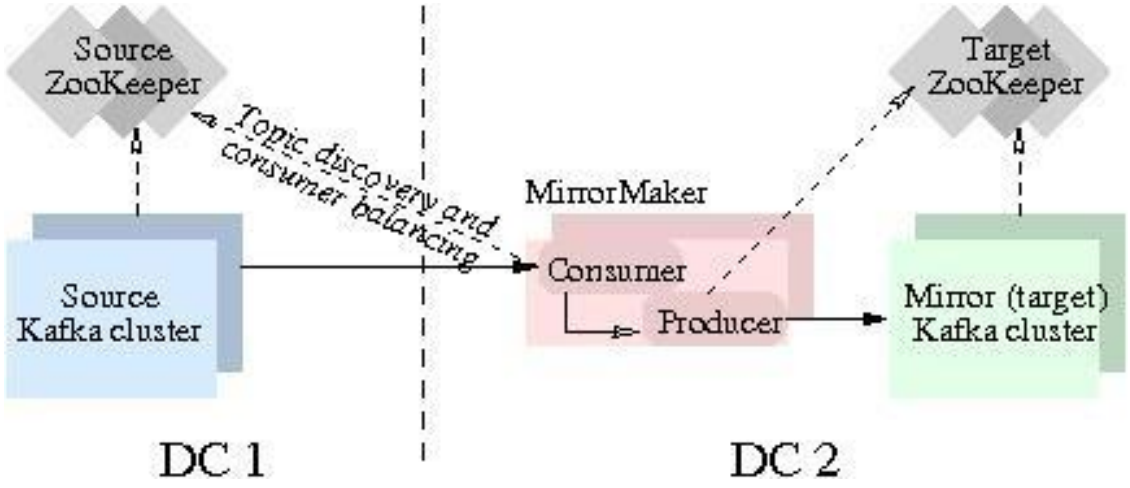


Data Replication



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.

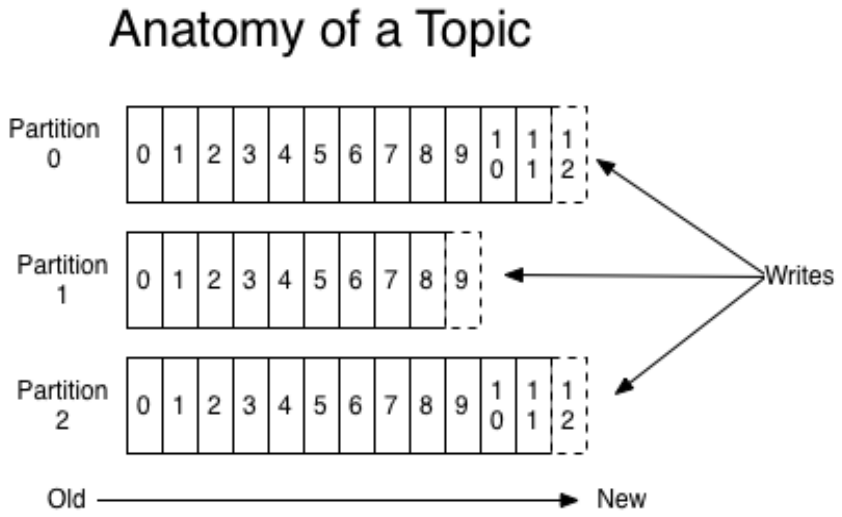


Kafka Brokers

- 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.

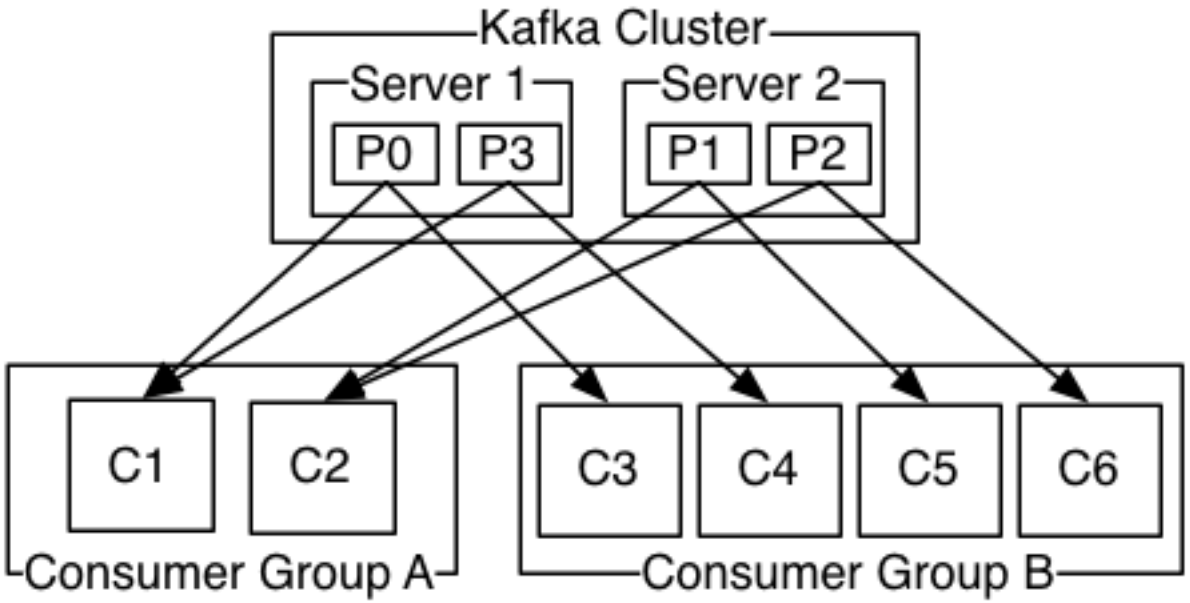
Kafka Producers

- 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)

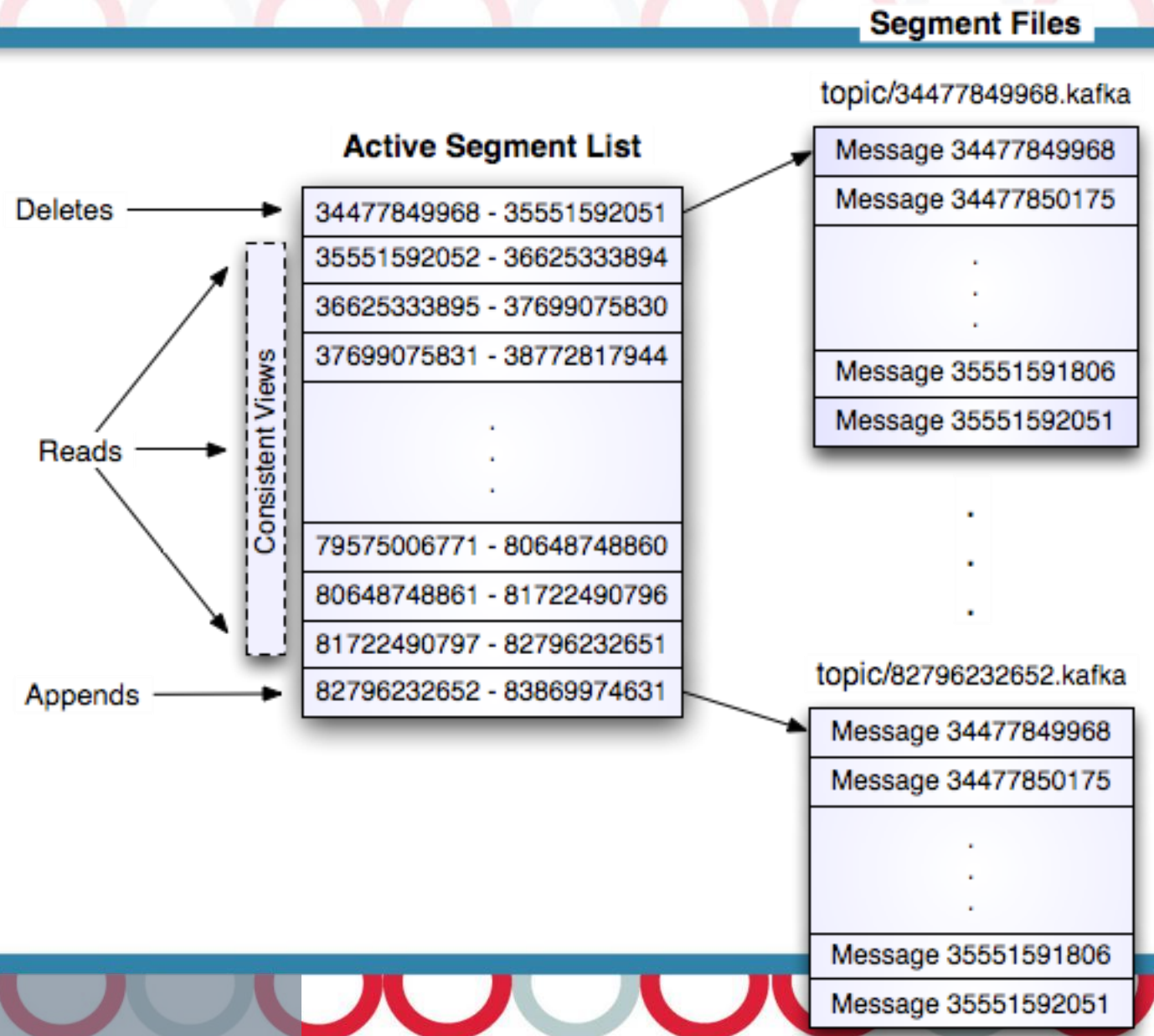
- Records are messages that contain a key/value pair along with metadata such as a timestamp, message key, and headers. Headers may store application-specific metadata:
  - length: varint
  - attributes: int8 (bit 0~7: unused)
  - timestampDelta: varint
  - offsetDelta: varint
  - keyLength: varint
  - 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

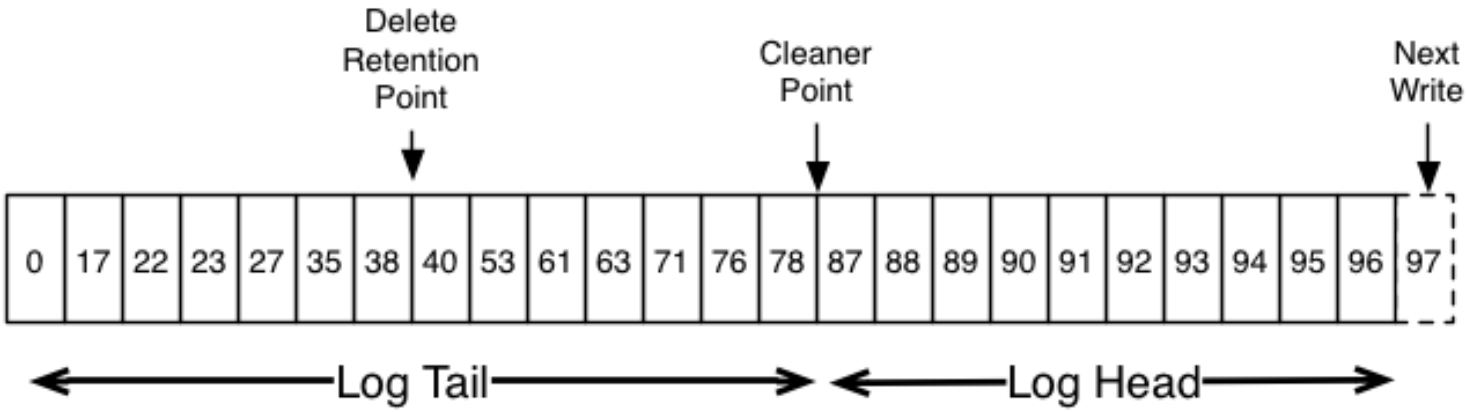
Kafka Log Implementation



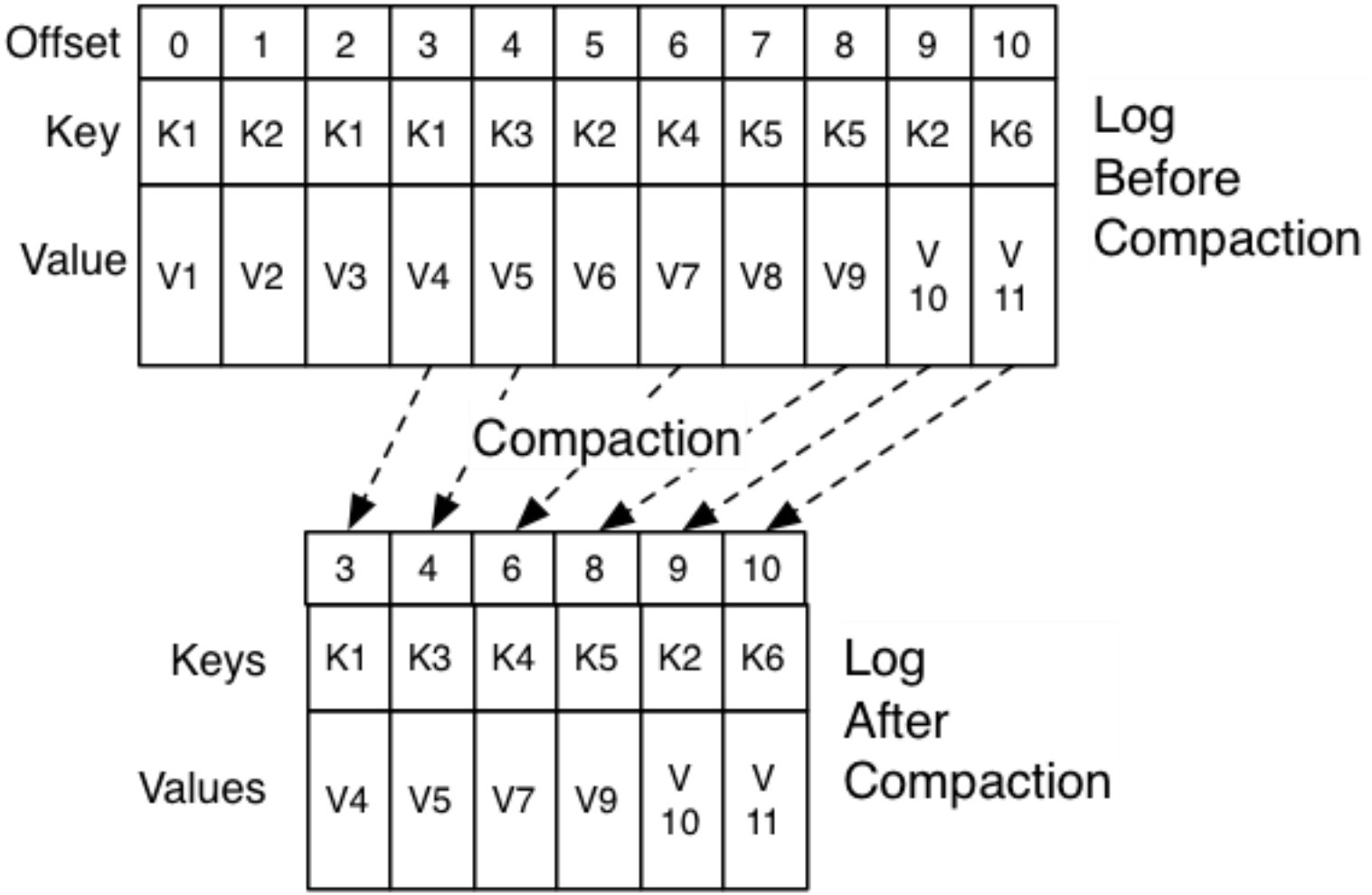
Log Compaction - I

- 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



Compaction Settings

1. cleanup.policy=compact – enables topic compaction

2. [log.cleaner.]min.compaction.lag.ms - the minimum time a message will remain uncompacted in the log. Defaults to 0.

3. [log.cleaner.]max.compaction.lag.ms - the maximum time a message will remain ineligible for compaction in the log. Defaults to MAXLONG.

4. delete.retention.ms – the amount of time to retain delete tombstone markers (nulled keys) for log compacted topics. This setting also gives a bound on the time in which a consumer must complete a read if they begin from offset 0 to ensure that they get a valid snapshot of the final stage (otherwise delete tombstones may be collected before they complete their scan).

Compaction Settings

1. log.cleaner.backoff.ms - the amount of time to sleep when there are no logs to clean

2. min.cleanable.dirty.ratio - this configuration controls how frequently the log compactor will attempt to clean the log (assuming log compaction is enabled). By default we will avoid cleaning a log where more than 50% of the log has been compacted. This ratio bounds the maximum space wasted in the log by duplicates (at 50% at most 50% of the log could be duplicates).

3. log.cleaner.threads - the number of background threads to use for log cleaning. Defaults to 1.

Compaction Guarantees

1. Any consumer that stays caught-up to within the head of the log will see every message that is written; these messages will have sequential offsets. The topic's min.compaction.lag.ms can be used to guarantee the minimum length of time must pass after a message is written before it could be compacted. I.e. it provides a lower bound on how long each message will remain in the (uncompacted) head. The topic's max.compaction.lag.ms can be used to guarantee the maximum delay between the time a message is written and the time the message becomes eligible for compaction.

2. Ordering of messages is always maintained. Compaction will never re-order messages, just remove some.

3. The offset for a message never changes. It is the permanent identifier for a position in the log.

4. Any consumer progressing from the start of the log will see at least the final state of all records in the order they were written. Additionally, all delete markers for deleted records will be seen, provided the consumer reaches the head of the log in a time period less than the topic's delete.retention.ms setting (the default is 24 hours). In other words: since the removal of delete markers happens concurrently with reads, it is possible for a consumer to miss delete markers if it lags by more than delete.retention.ms.

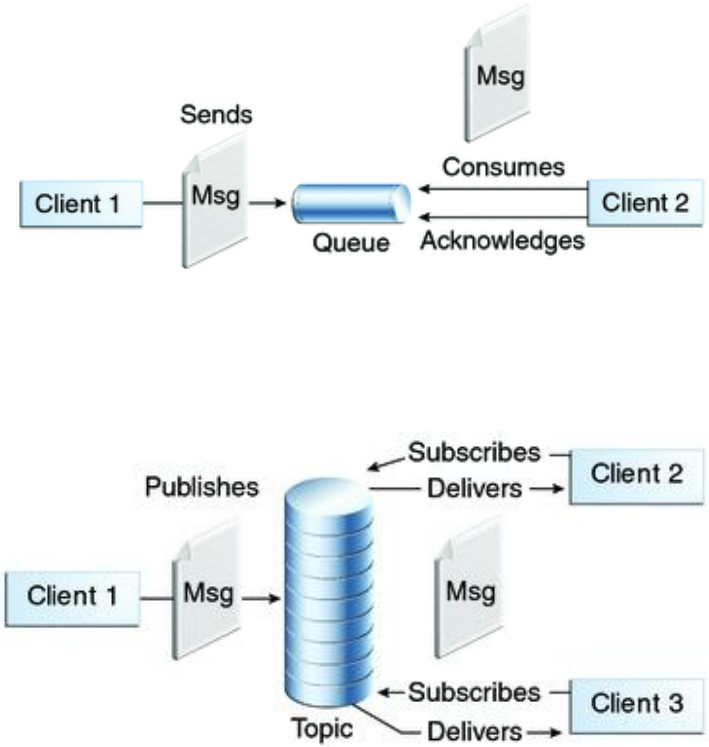
Kafka Guarantees

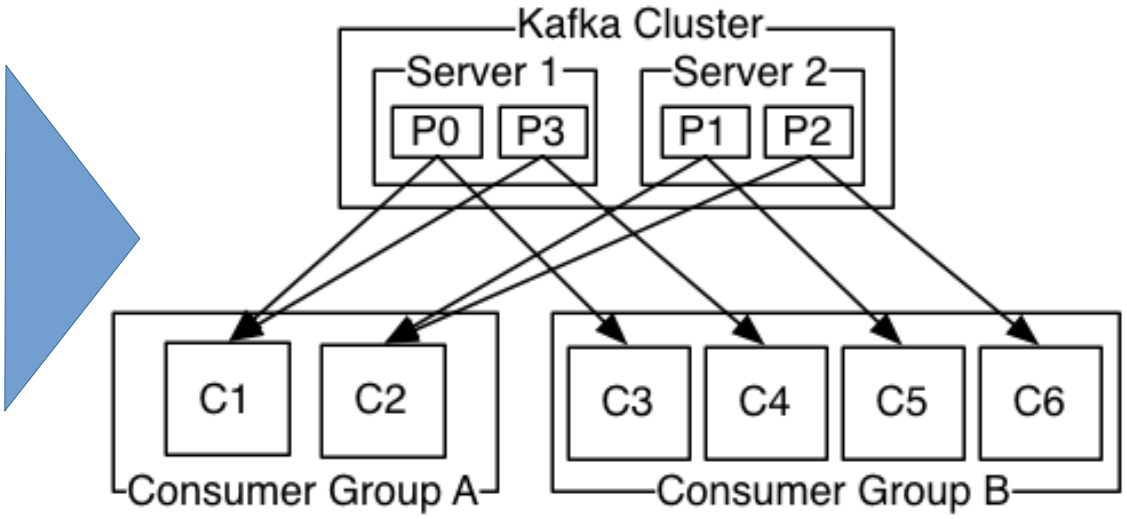
1. Messages sent by a producer to a particular topic partition will be appended in the order they are sent. That is, if a record M1 is sent by the same producer as a record M2, and M1 is sent first, then M1 will have a lower offset than M2 and appear earlier in the log.

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.

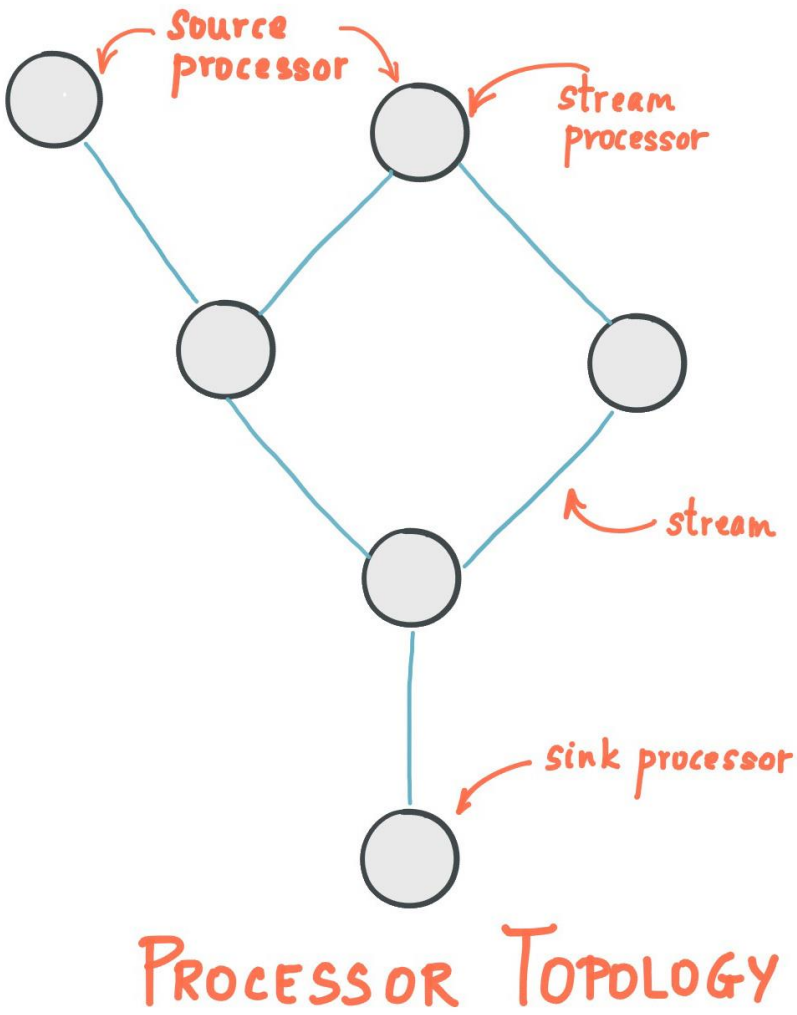
Kafka Stream Processing - I

- 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.

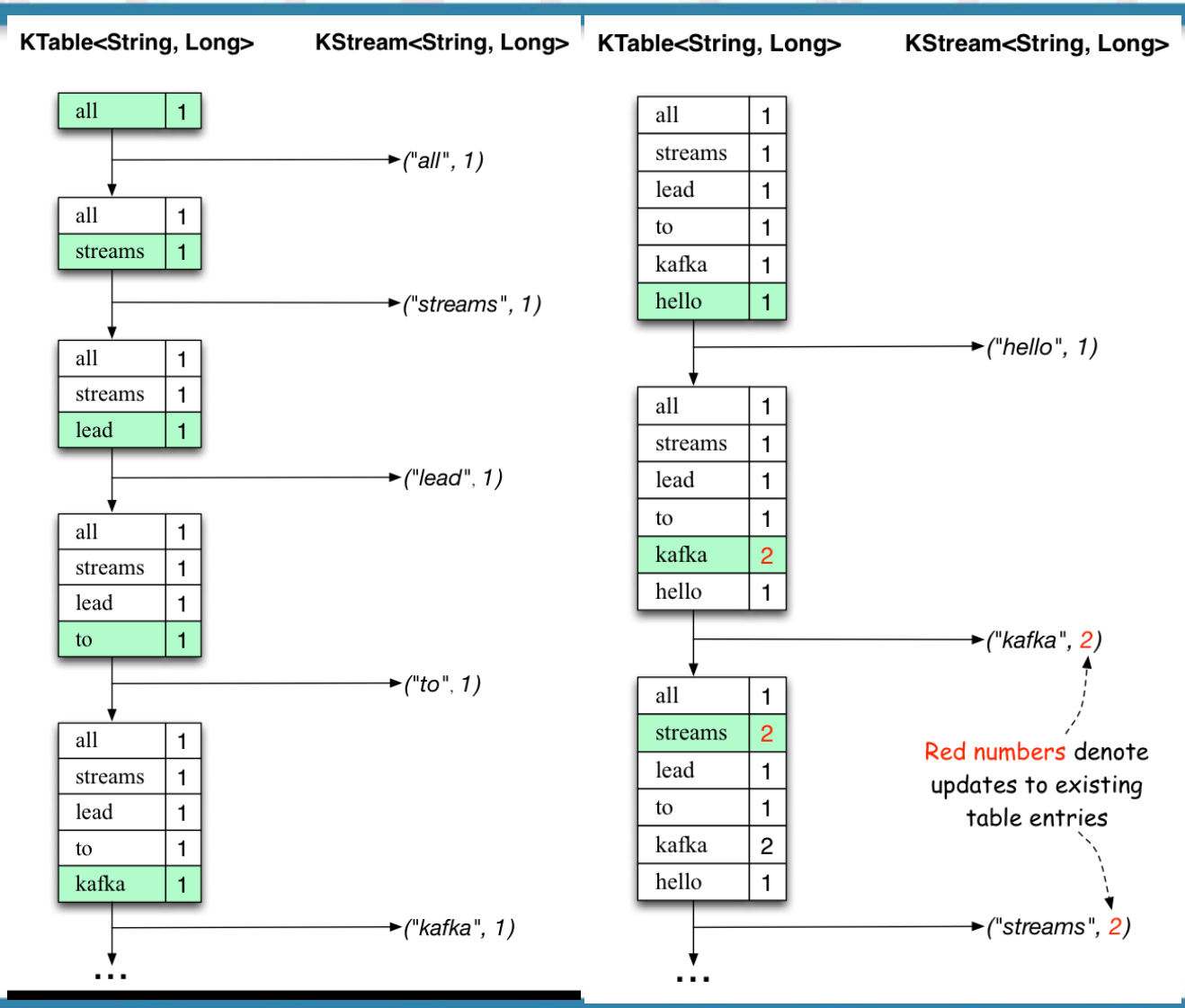
Kafka Stream Processing - II

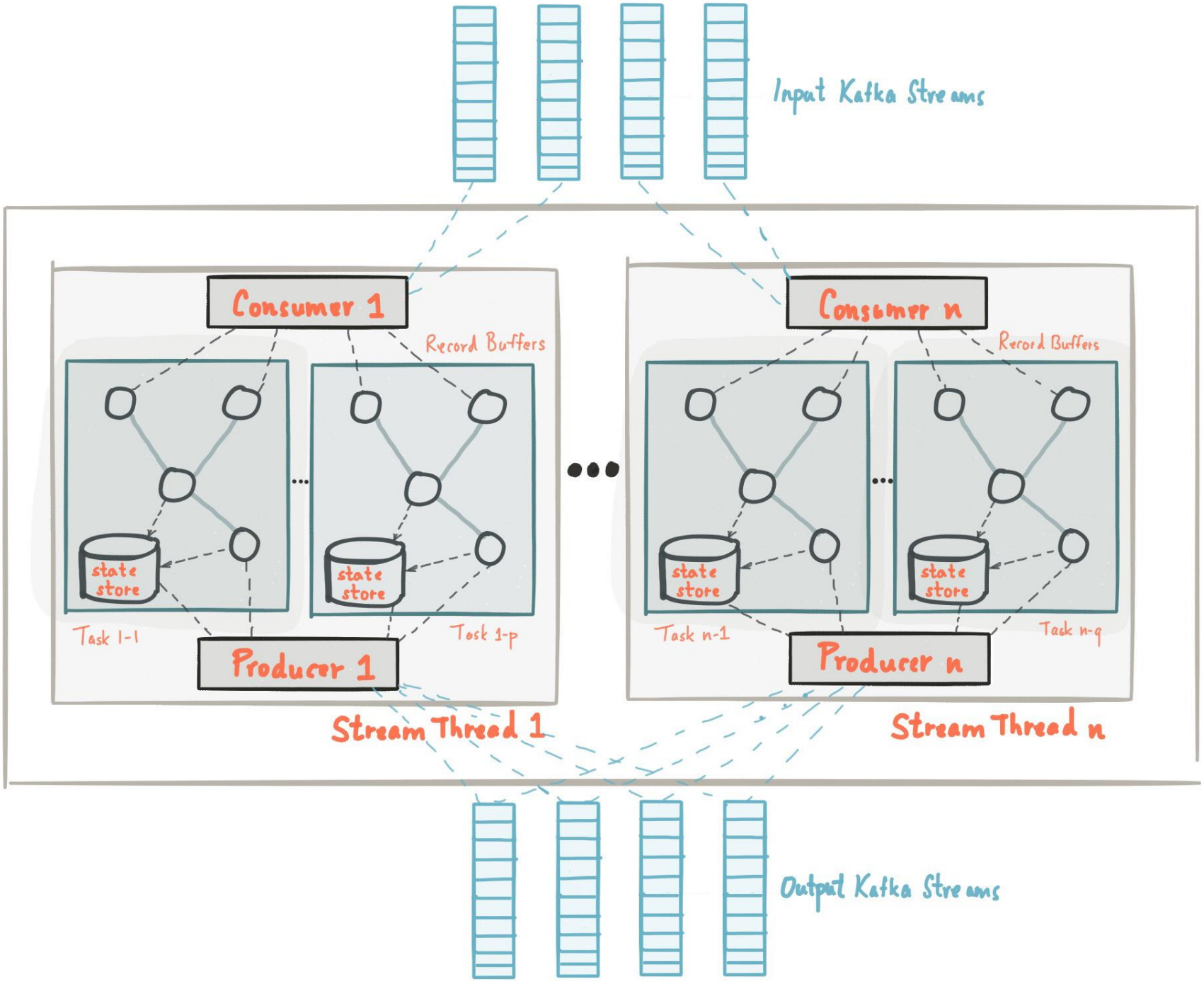
- Kafka streams read data from a topic, running some form of analysis or data transformation, and finally writing the data back to another topic or shipping it to an external source.
- Streams we can achieve real-time stream processing rather than batch processing. For the majority of cases, it's recommended 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.

Moved to Brokers

KRAFT - Kafka's new quorum controller

- Enables Kafka clusters to scale to millions of partitions through improved control plane performance with the new metadata management
- Improves stability, simplifies the software, and makes it easier to monitor, administer, and support Kafka.
- Allows Kafka to have a single security model for the whole system
- Provides a lightweight, single process way to get started with Kafka
- Makes controller failover near-instantaneous

wsl --distribution docker-desktop-data

docker-compose -f examples\docker-compose.yml run kafka

Source: https://developer.confluent.io/learn/kraft/

Raft Algorithm

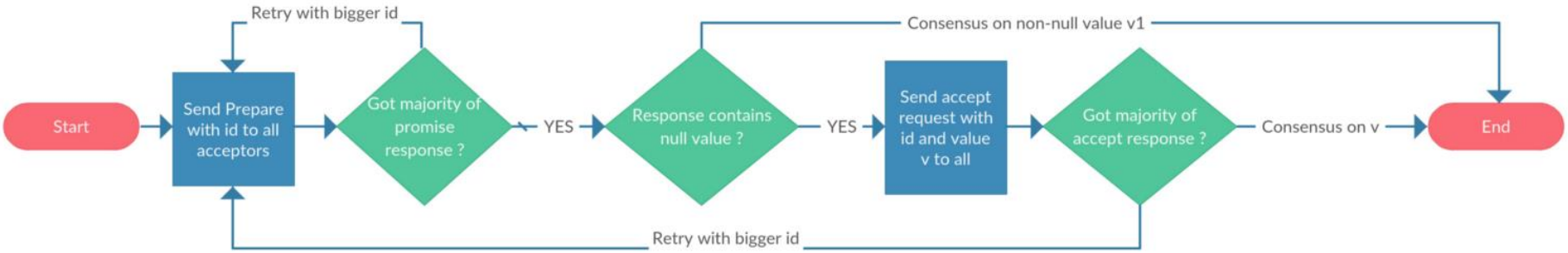
- **Raft** is a consensus algorithm designed as an alternative to the Paxos family of algorithms.
- It is named after Reliable, Replicated, Redundant, And Fault-Tolerant.
- It was meant to be more understandable than Paxos by means of separation of logic, but it is also formally proven safe and offers some additional features
- **Raft** offers a generic way to distribute a state machine across a cluster of computing systems, ensuring that each node in the cluster agrees upon the same series of state transitions.
- It has a number of open-source reference implementations, with full-specification implementations in Go, C++, Java, and Scala.

Safety rules in Raft

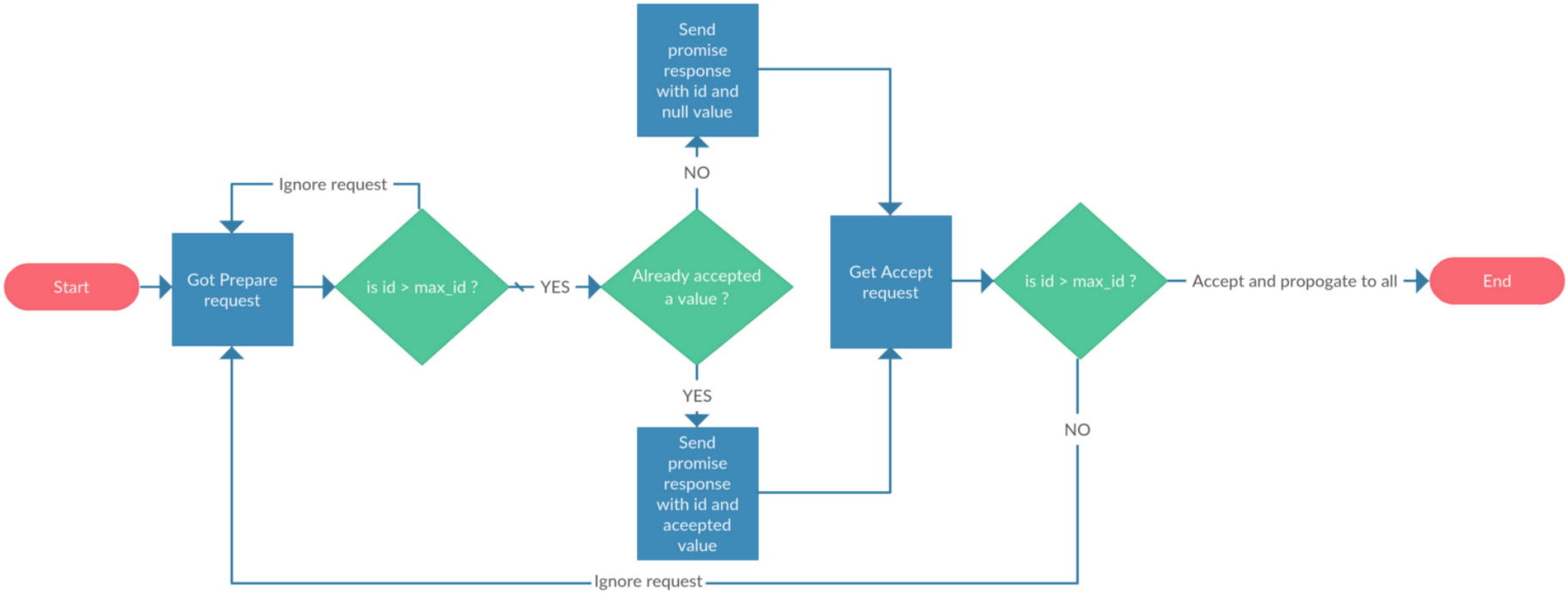
- Raft guarantees each of these safety properties :
  - Election safety: at most one leader can be elected in a given term.
  - Leader append-only: a leader can only append new entries to its logs (it can neither overwrite nor delete entries).
  - Log matching: if two logs contain an entry with the same index and term, then the logs are identical in all entries up through the given index.
  - Leader completeness: if a log entry is committed in a given term then it will be present in the logs of the leaders since this term
  - State machine safety: if a server has applied a particular log entry to its state machine, then no other server may apply a different command for the same log.
- The first four rules are guaranteed by the details of the algorithm described in the previous section. The State Machine Safety is guaranteed by a restriction on the election process.

Raft Algorithm – Proposers

http://thesecretlivesofdata.com/raft/



Raft Algorithm - Acceptors



Raft Algorithm – Simulation and Visualization

http://thesecretlivesofdata.com/raft/

https://raft.github.io/

Producer Configuration

Source: https://developer.confluent.io/learn/kraft/

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

Thank's for Your Attention!

Trayan Iliev

CEO of IPT – Intellectual Products

& Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

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

https://plus.google.com/+IproductOrg