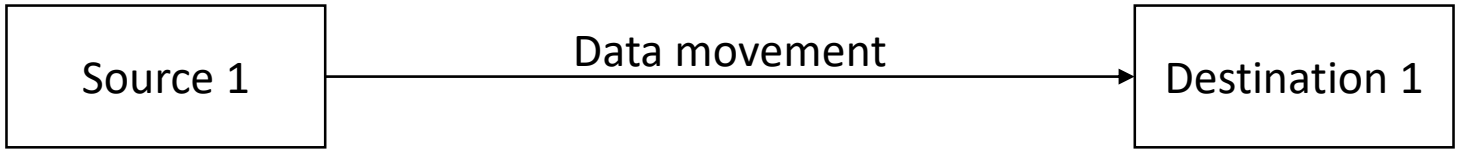
A central component in modern data architecture

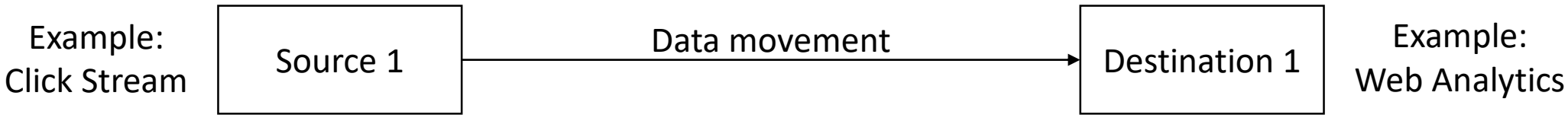
Data Integration

Data integration starts something like this



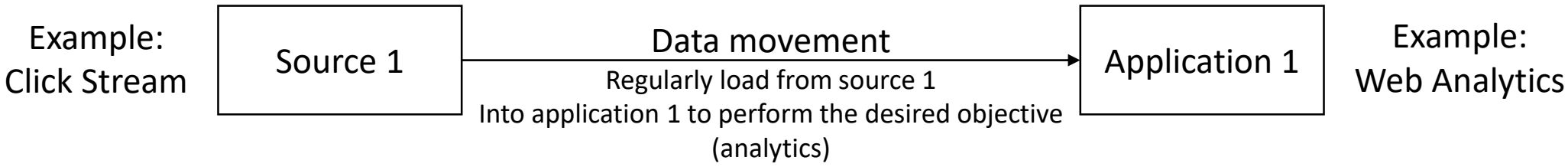
Data Integration

Data integration starts something like this



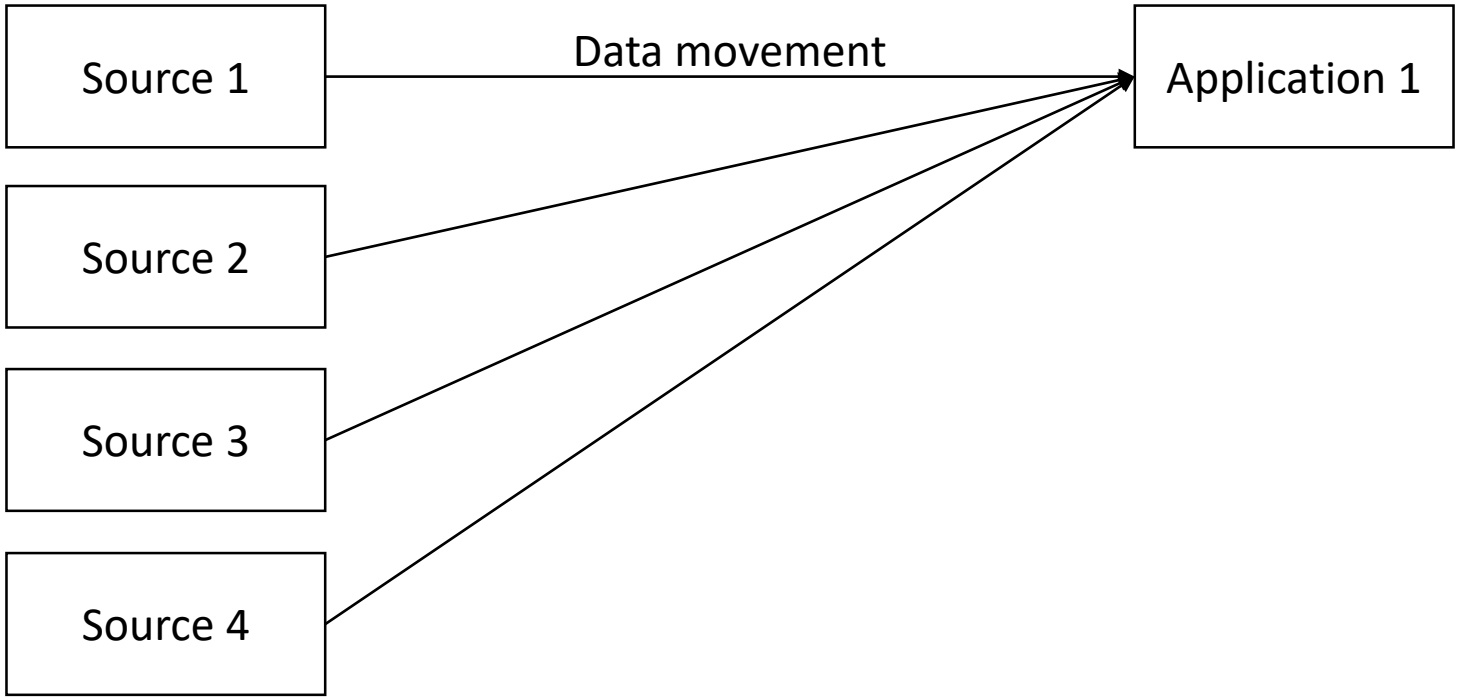
Data Integration

Data integration starts something like this



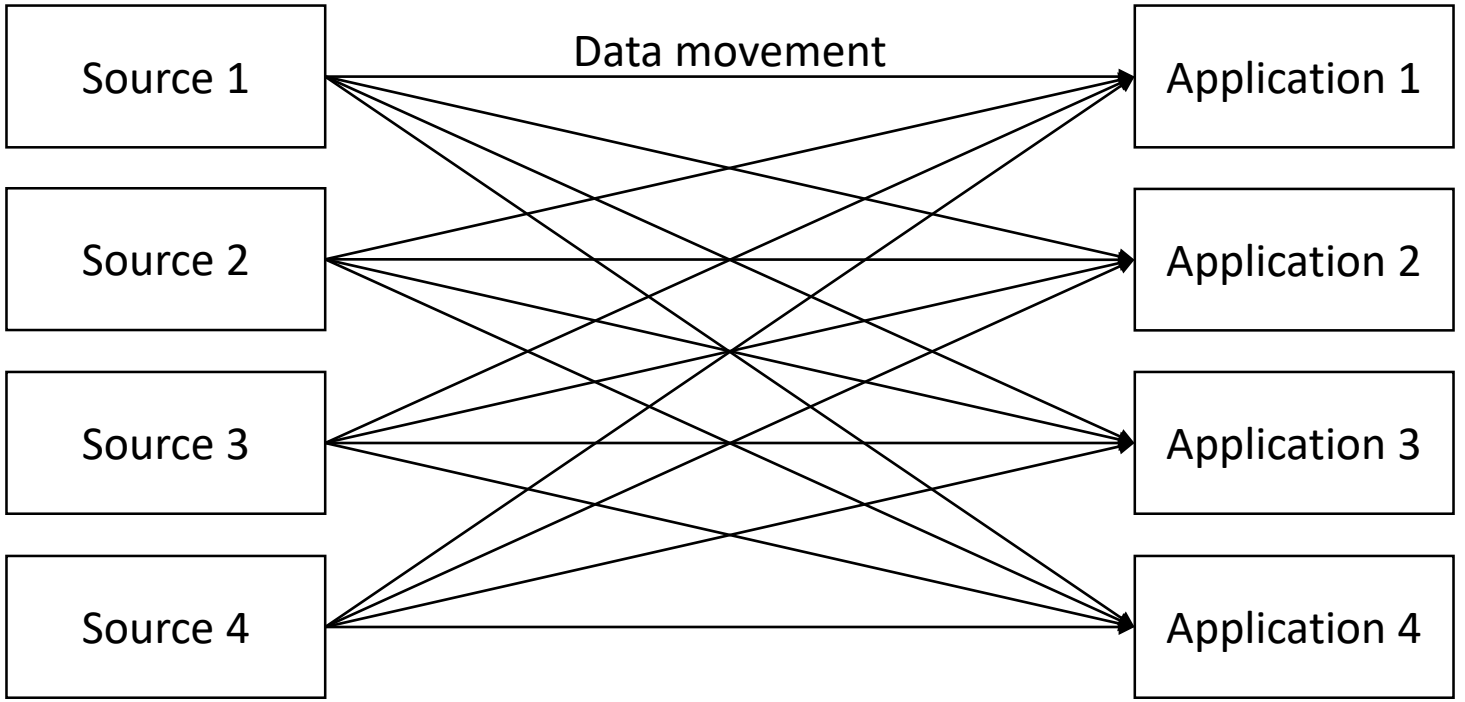
Data Integration

Then we reuse our integration for different objectives



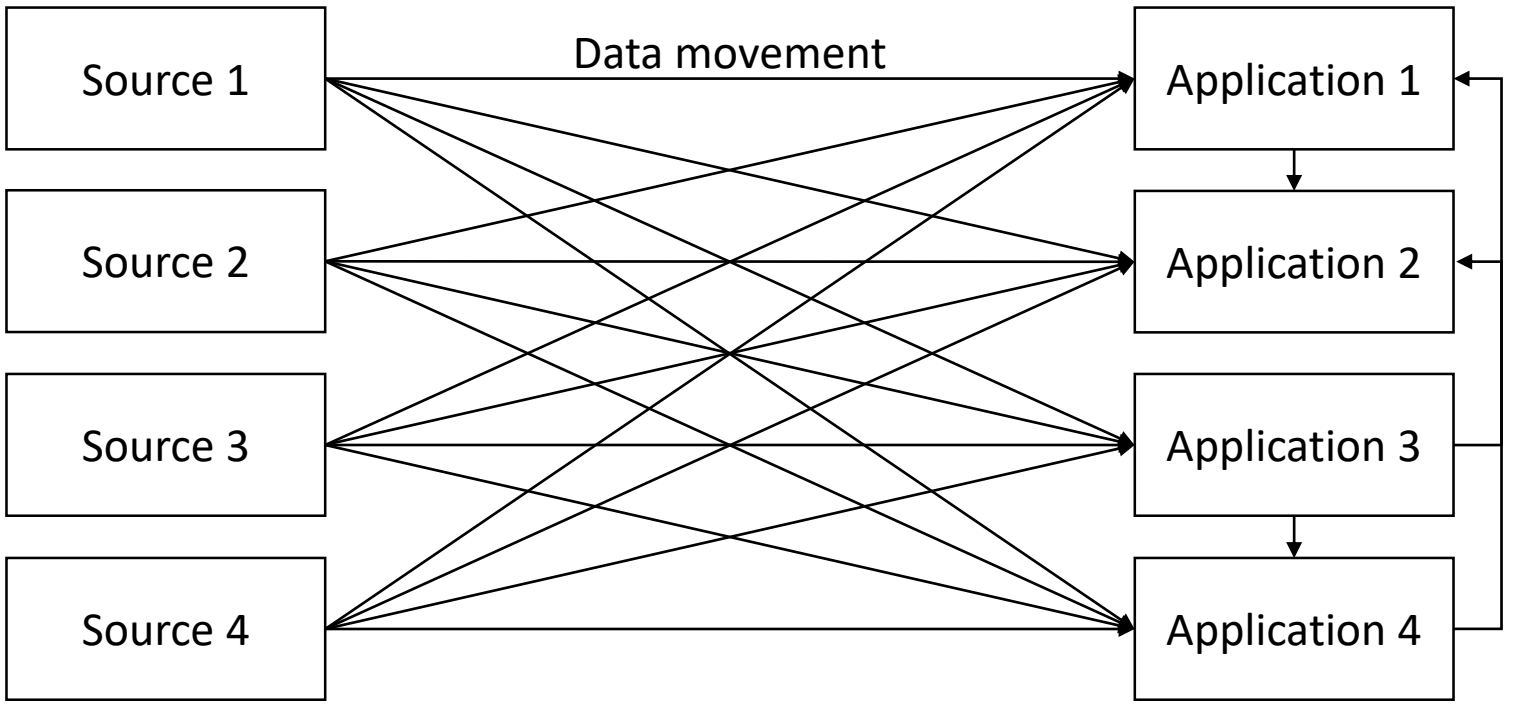
Data Integration

The reality is much complex: multiple sources and destinations



Data Integration

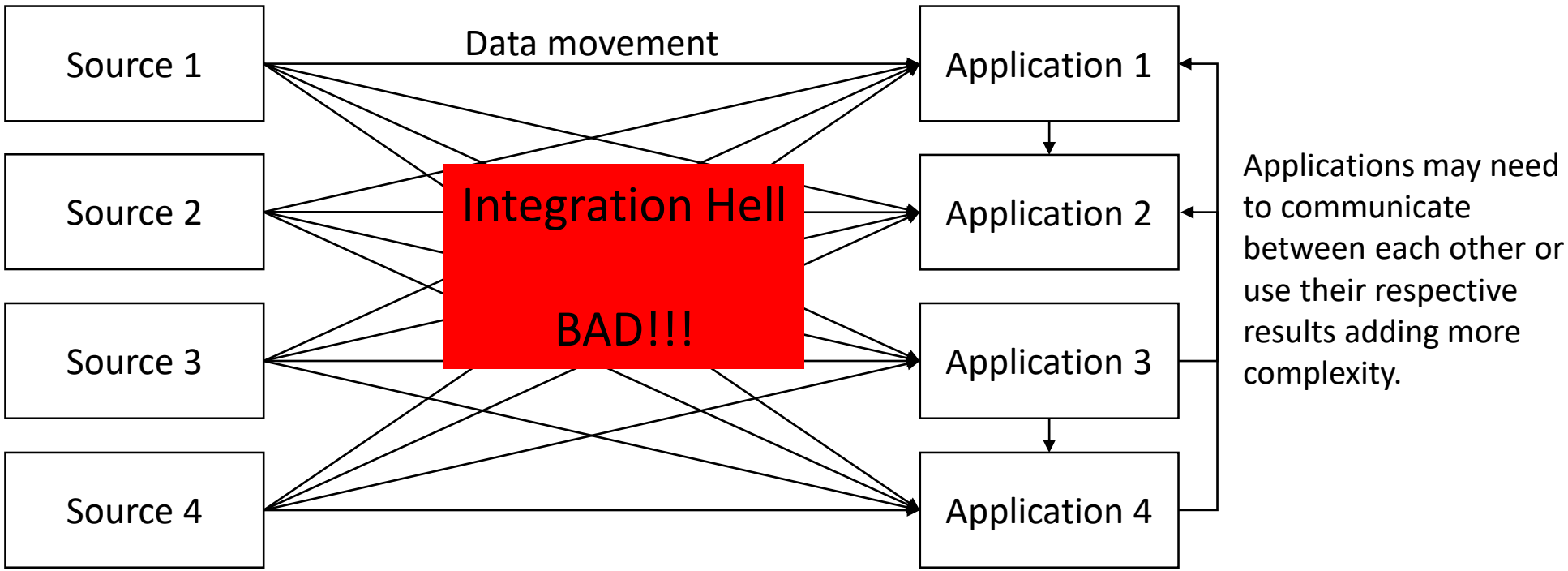
The reality is much complex: multiple sources and destinations



Applications may need to communicate between each other or use their respective results ➔ adding more complexity.

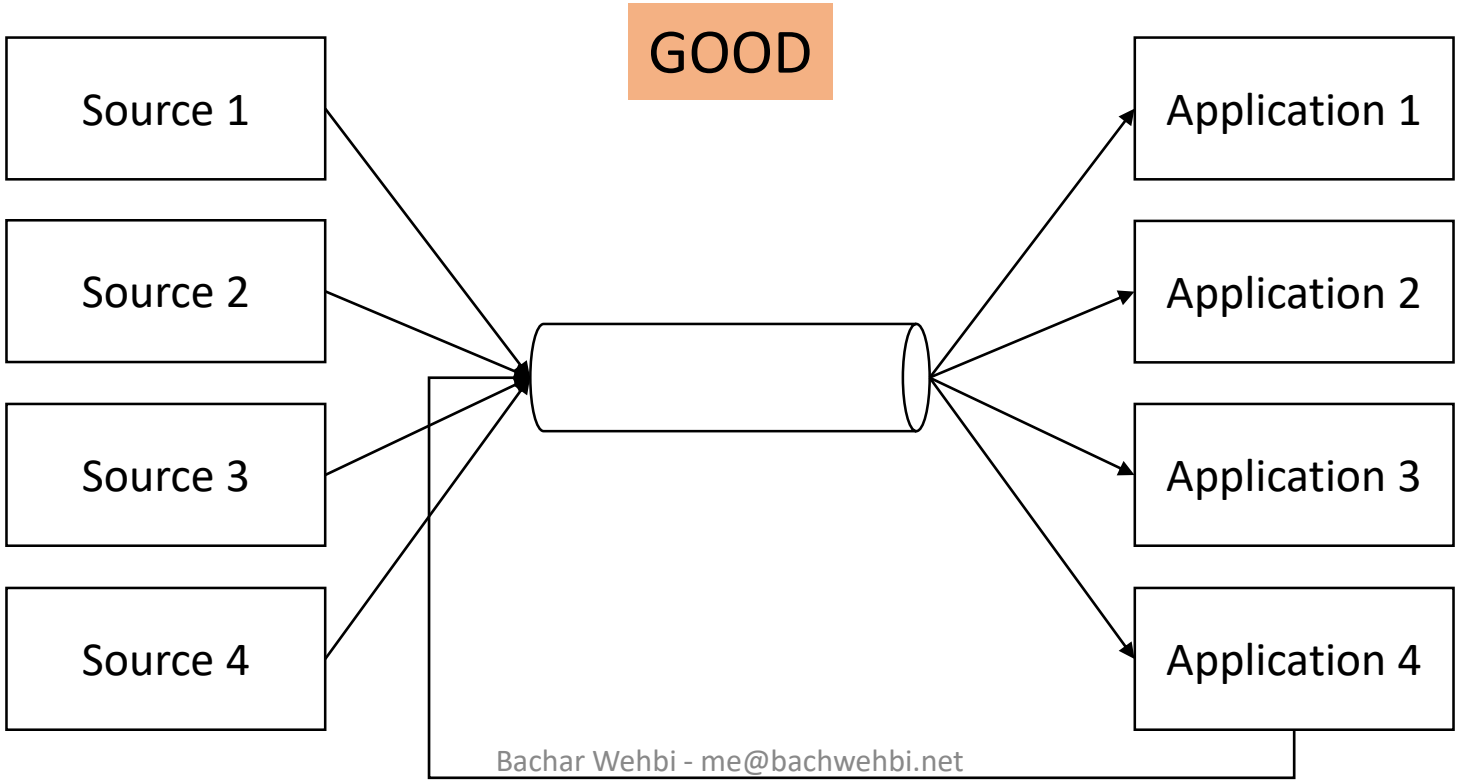
Data Integration

The reality is much complex: multiple sources and destinations



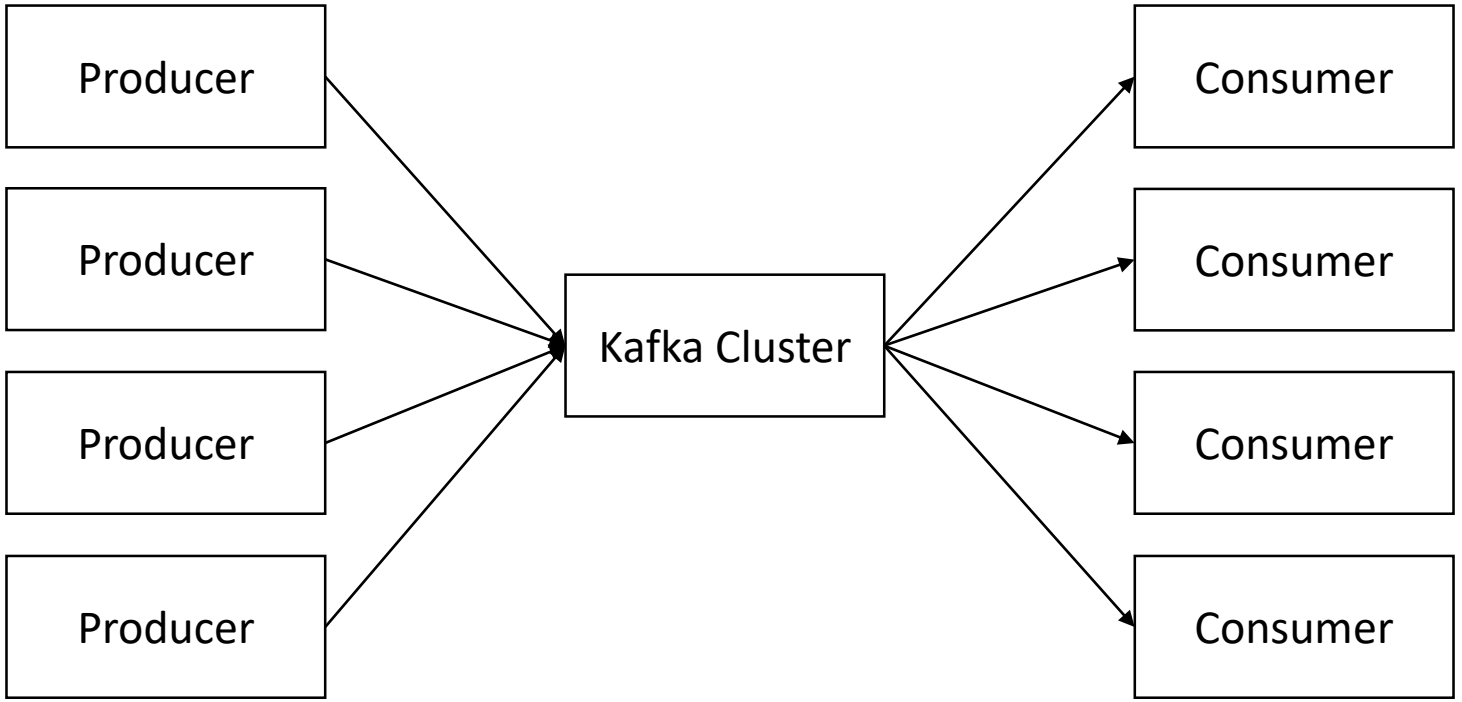
Data Integration

Simplify using a “messaging system”



Apache Kafka

Decouples data producers from data consumers



Apache Kafka: History

- Created at LinkedIn
- Donated to the Apache Foundation in 2011
- Graduated from Apache Incubator in 2012
  - Top level Apache project
- Creators of Kafka founded Confluent in 2014
- Supported by all Hadoop distributions
- V-1.0.0 released in November 2017

Apache Kafka: Capabilities

- Streaming data integration pipeline
  - Publish Subscribe architecture
  - Enterprise messaging system, similar to message queues

- Stores (retains) data messages in a **write-ahead log**
  - Fault-tolerant durable way

- Stream processing:
  - Process streams of records as they occur

Let’s get to know Logs

Called also: Write-ahead logs, Commit logs or Transaction logs

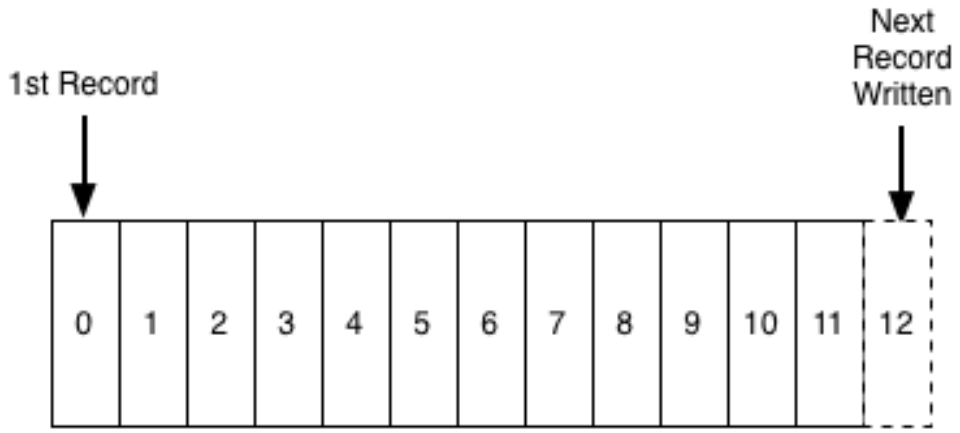
A data structure at the heart of many data stores

What is a log

```
::ffff:54.243.49.1 - - [01/Mar/2018:23:07:31 +0000] "POST /v1/data/write/Ras_Beirut/statusRas_Beirut4 HTTP/1.1" 200 4 "-" "beebotte node.js SDK v1.2.0"
::ffff:54.243.49.1 - - [01/Mar/2018:23:07:31 +0000] "POST /v1/data/write/Ras_Beirut/statusRas_Beirut1 HTTP/1.1" 200 4 "-" "beebotte node.js SDK v1.2.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:31 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:32 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:33 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:34 +0000] "POST /v1/data/write/bbt_raspi/cpu HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:34 +0000] "POST /v1/data/write/bbt_raspi/memory HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:34 +0000] "POST /v1/data/write/bbt_raspi/disk HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:34 +0000] "POST /v1/data/write/bbt_raspi/eth0 HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:34 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:35 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:36 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.243.49.1 - - [01/Mar/2018:23:07:36 +0000] "POST /v1/data/write/Ras_Beirut/statusRas_Beirut5 HTTP/1.1" 200 4 "-" "beebotte node.js SDK v1.2.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:37 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:38 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:39 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:40 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:40 +0000] "POST /v1/data/write/ISS/pos HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:41 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:42 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:43 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:44 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.243.49.1 - - [01/Mar/2018:23:07:44 +0000] "POST /v1/data/write/Hamra/statusHamra6 HTTP/1.1" 200 4 "-" "beebotte node.js SDK v1.2.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:45 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:46 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
::ffff:54.221.205.80 - - [01/Mar/2018:23:07:47 +0000] "POST /v1/data/publish/ISS/position HTTP/1.1" 200 4 "-" "python-requests/2.13.0"
```

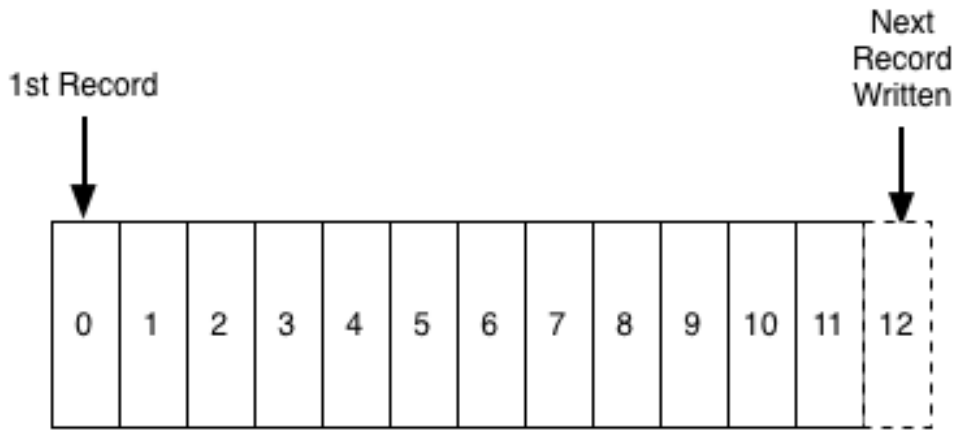
What is a log

- Probably the simplest storage abstraction
- Append only
- Totally ordered records by time
- Immutable



Why it is important

Q. What is the purpose of a log?

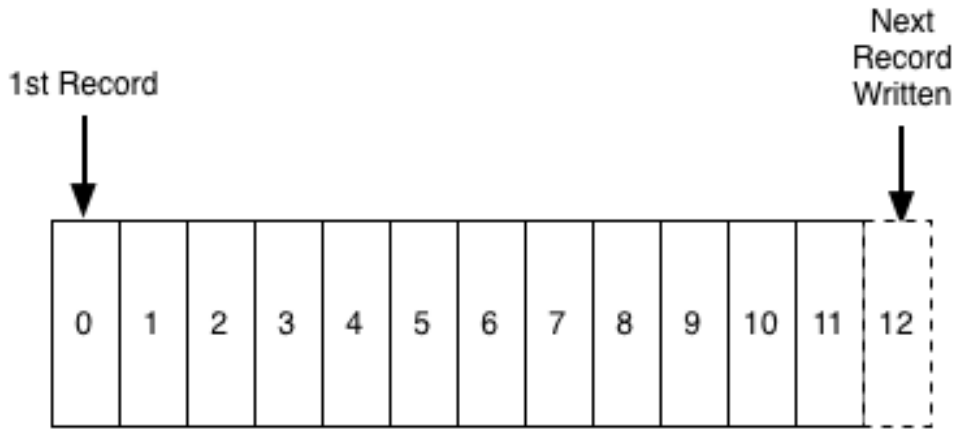


Why it is important

Q. What is the purpose of a log?

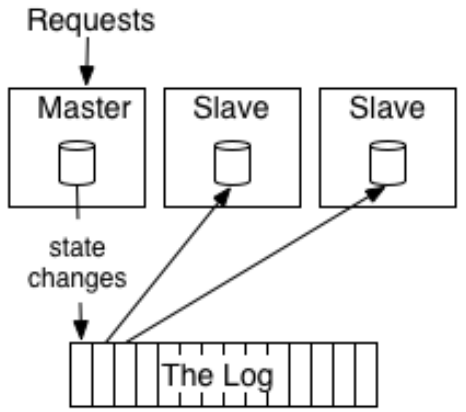
A. Record what happened and when.

For distributed data systems this is the very heart of the problem.



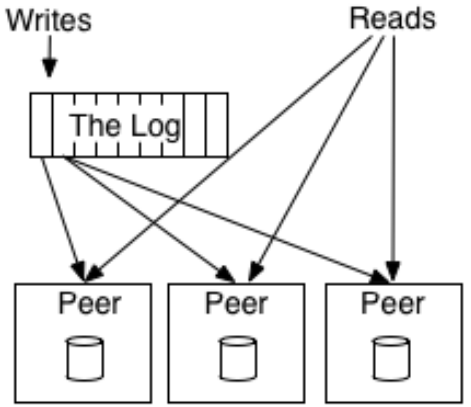
Logs: Example usages

- Primary backup in master slave mode: master writes to Log, slaves read from log and update their state



Primary-Backup

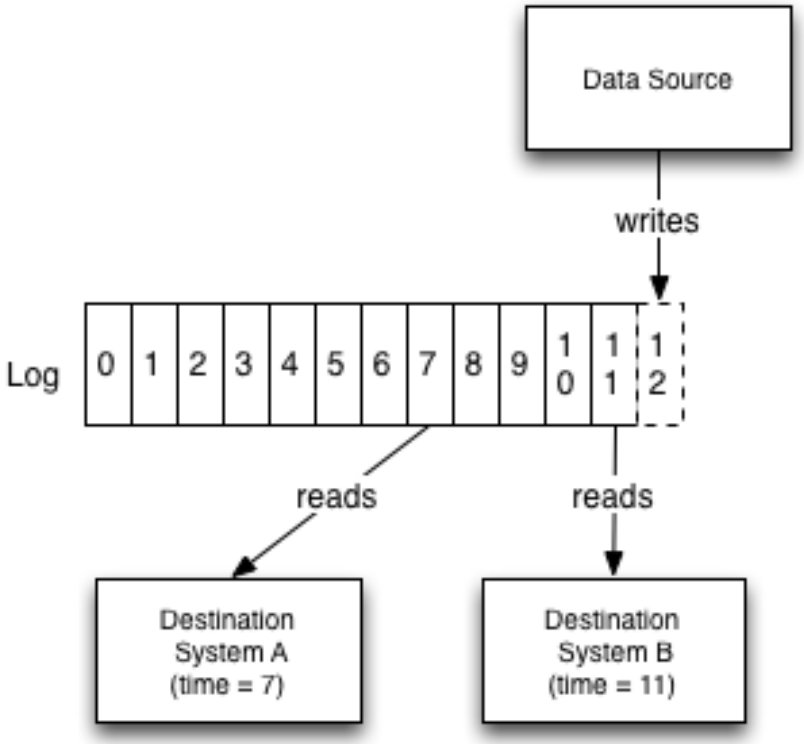
- State machine replication: keep a log of incoming events and each replica processes independently these events



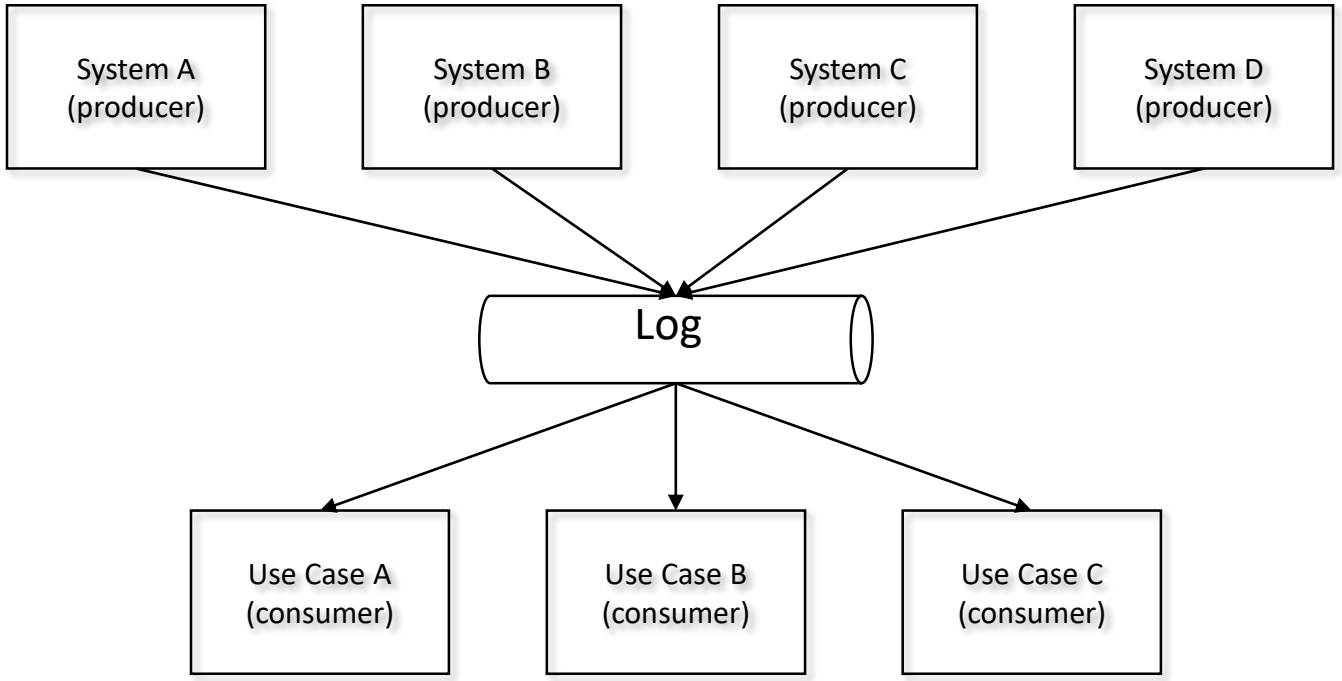
State-Machine Replication

Logs: seperation of concerns

- Data producers (writers) do not need to know about data consumers (readers)
- Multiple consumers can coexist
- Reprocessing history is possible by reading from the start of the log



Log as data integration infrastructure



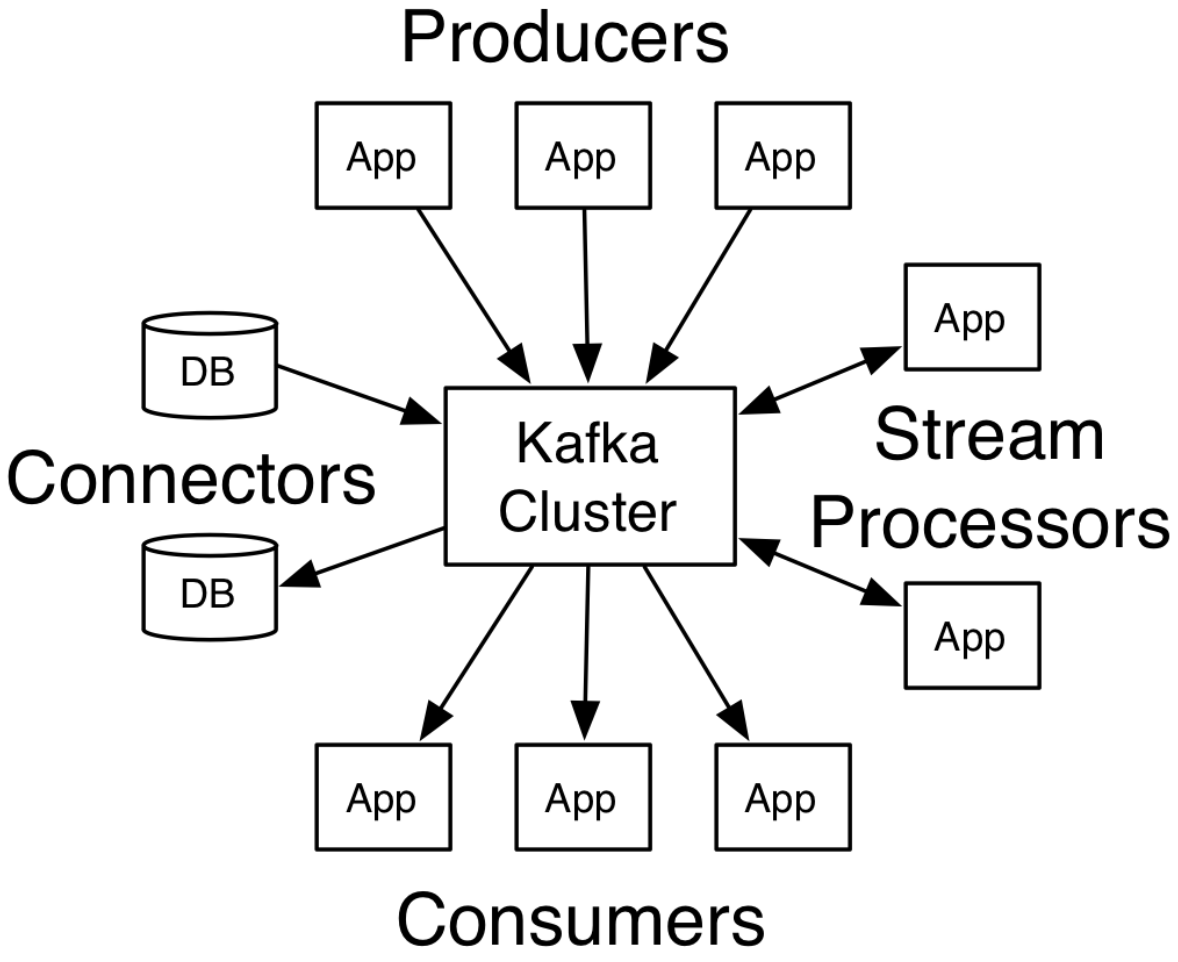
Apache Kafka: Characteristics

- High Scalability
  - Cluster instead of servers: supports multiple nodes
  - High throughput per node: hundreds of MB/sec/node
  - High volume of retained messages per node: multiple TB
- High Guarantees
  - Messages are strictly ordered: a consumer will receive messages in the same order they were produced
  - Data durability: messages can be replicated to multiple nodes to tolerate failures
- Highly Distributed
  - Through **replication** and **partitioning**
  - Avoid needless data copies

Apache Kafka: Use Cases

- Messaging
- Website activity tracking
- Metrics
- Log Aggregation
- Stream Processing
- Event Sourcing
- Commit Log

Apache Kafka: High Level Architecture



Apache Kafka: Terminology

- Message
  - A message in Kafka is a data record published to the broker.
- Topic
  - A named stream of records (messages).
- Producer
  - A process or program that publishes messages to Kafka. Messages are always published to topics.
- Consumer
  - A process or program that subscribes to a topic to process the stream of messages.
- Broker
  - Kafka runs in a cluster comprised of one or multiple brokers. A Broker is a Kafka instance (server or process).
- Kafka Connector
  - A reusable producer or consumer that connects Kafka topics to existing applications or systems
- Kafka Streams
  - A stream processing API that consumes messages from input topics, transform them and publishes new messages to output topics.

Apache Kafka: Messages

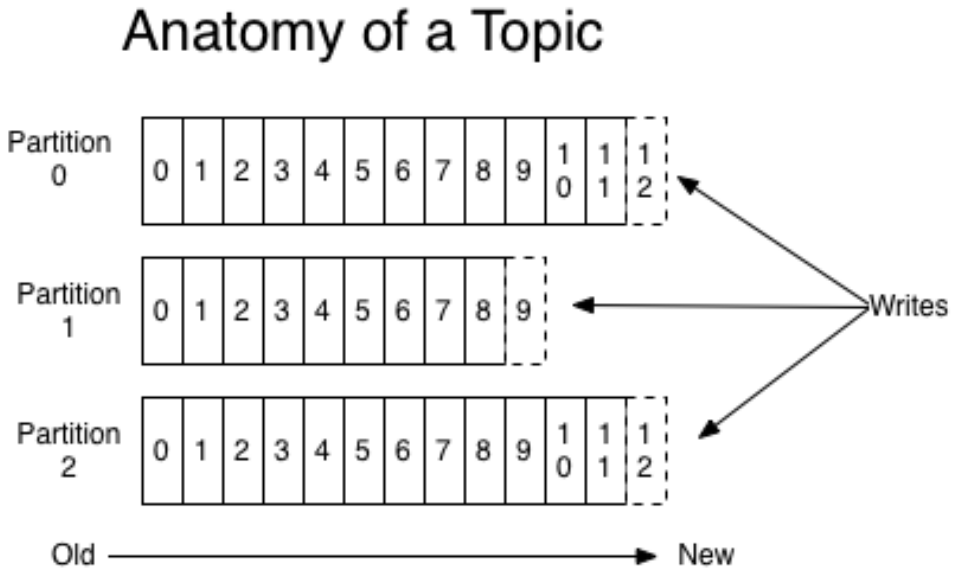
- Kafka messages are byte arrays of variable length
  - The content of the message is arbitrary & completely opaque
    - Design decision not to make a serialization choice
  - The message content format is application specific
    - It can be raw text, JSON, Avro, ProtoBuffer, etc.

- Kafka does not enforce limits on message size
  - It performs better with relatively small messages (few KB)

- Kafka messages are always written in batches named '*record batch*'
  - a record batch contains one or more records.
  - Allows Kafka to achieve high throughput
    - less network overhead
    - Produce and consume batches instead of individual messages

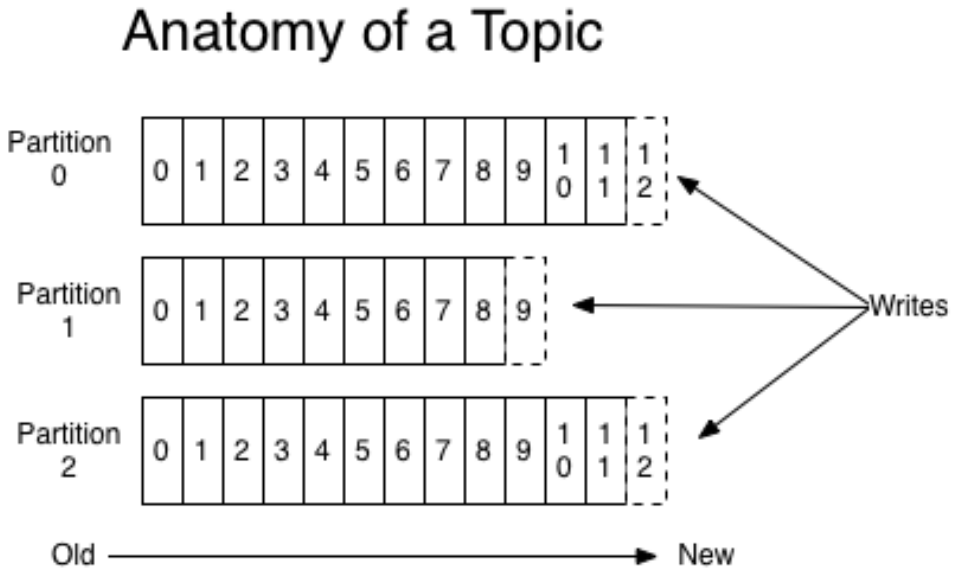
Apache Kafka: Topics

- A topic is a category or feed name to which records are published.
- Multi subscriber: A topic can have 0, 1 or multiple subscribers
- Topics are divided into partitions
  - Partitions help scale out topics on multiple nodes
  - Message order is guaranteed within a partition
- A partition is an ordered, immutable sequence of messages appended to a commit log
- Each message in a partition is assigned a sequential id number called offset that uniquely identifies the message within the partition



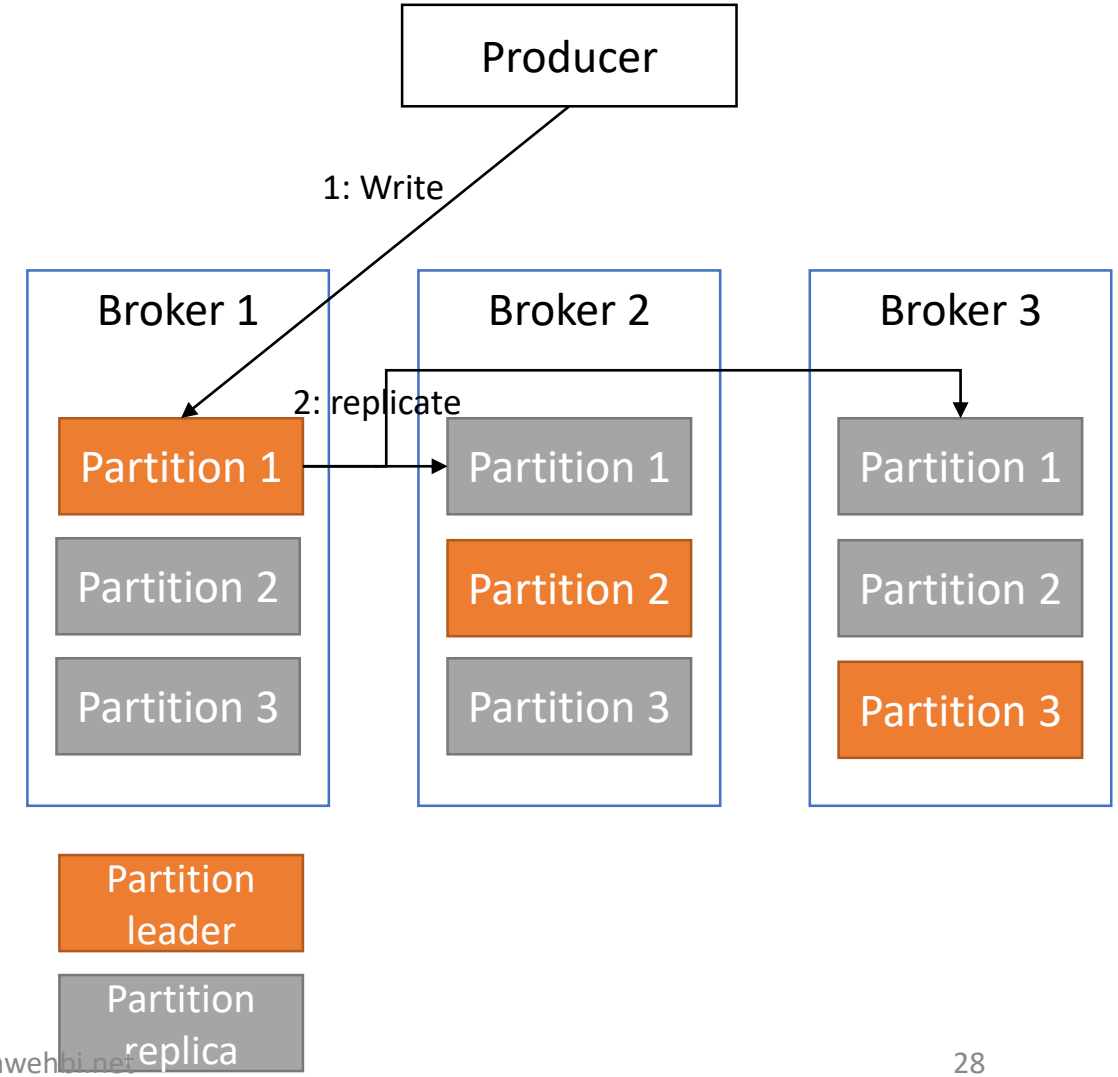
Apache Kafka: Topics

- Kafka is known to perform better with a relatively small number of large topics compared to a large number of small topics.
- Kafka can be configured to create topics implicitly: when message is first published to a new topic
  - Nice to have in dev mode, not in production (accidental creating a large number of topics)
- Data retention on Kafka topics can be configured:
  - For a specific period of time (example: last 30 days)
  - Based on data volume on the topic (example: 64GB)
- Kafka avoids needless copying of data



Apache Kafka: Topic Replication

- Topics can be replicated to ensure durability in case of failure
- Replication is applied to topic partition level
- A replicated partition has one leader (master) and one or multiple replicas
  - Replicas will try always to be in-sync with the leader
  - In-sync replicas are called ISR
- The replication factor is set on the topic level. The replication factor can only be changed to improve durability (higher), it can never be lowered.



Apache Kafka: Producer

- Kafka producers publish messages to topics. Producers never communicate with consumers.
  - Messages are persisted to disk upon reception
- Publish load is load balanced by partitions
  - In round robin fashion or based on message keys (optional)
  - All messages with the same key will be published to the same partition
- Message acknowledgments can have different configuration
  - No Acks (fire and forget): The producer will not wait a confirmation the message was received
  - Ack by the leader: The leader successfully received the message
  - Ack by in-sync replicas: the leader waits the replication of the message before confirming its reception to the consumer
- These different configuration settings have an impact on the data durability

Apache Kafka: Consumer

- Kafka consumers read data from topics

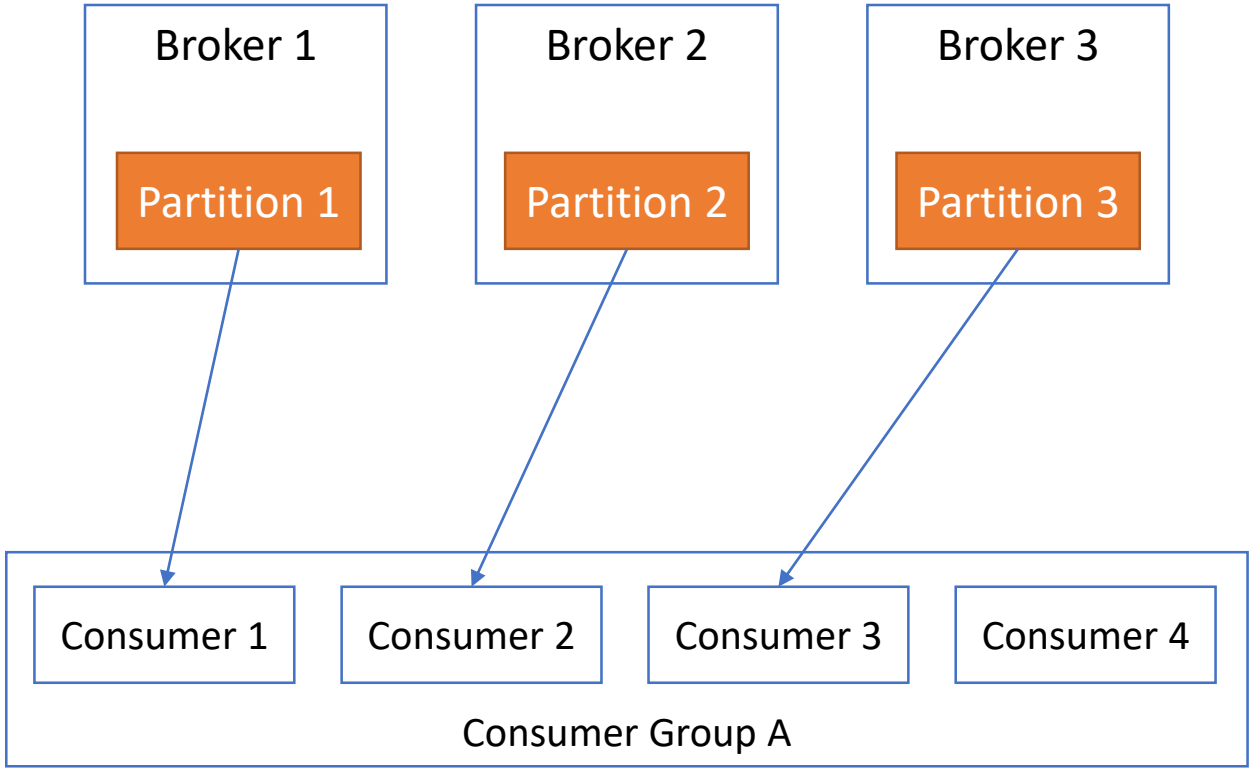
- Multiple consumers can simultaneously read from the same topic
  - Every consumer maintains its status: the offset it read last
  - Kafka will keep messages after they are read: data will always be there for new consumers (as the retention strategy permits).

- A consumer might stop reading (planned or accidental)
  - When resuming reading from a topic it can specify the offset from which to receive data

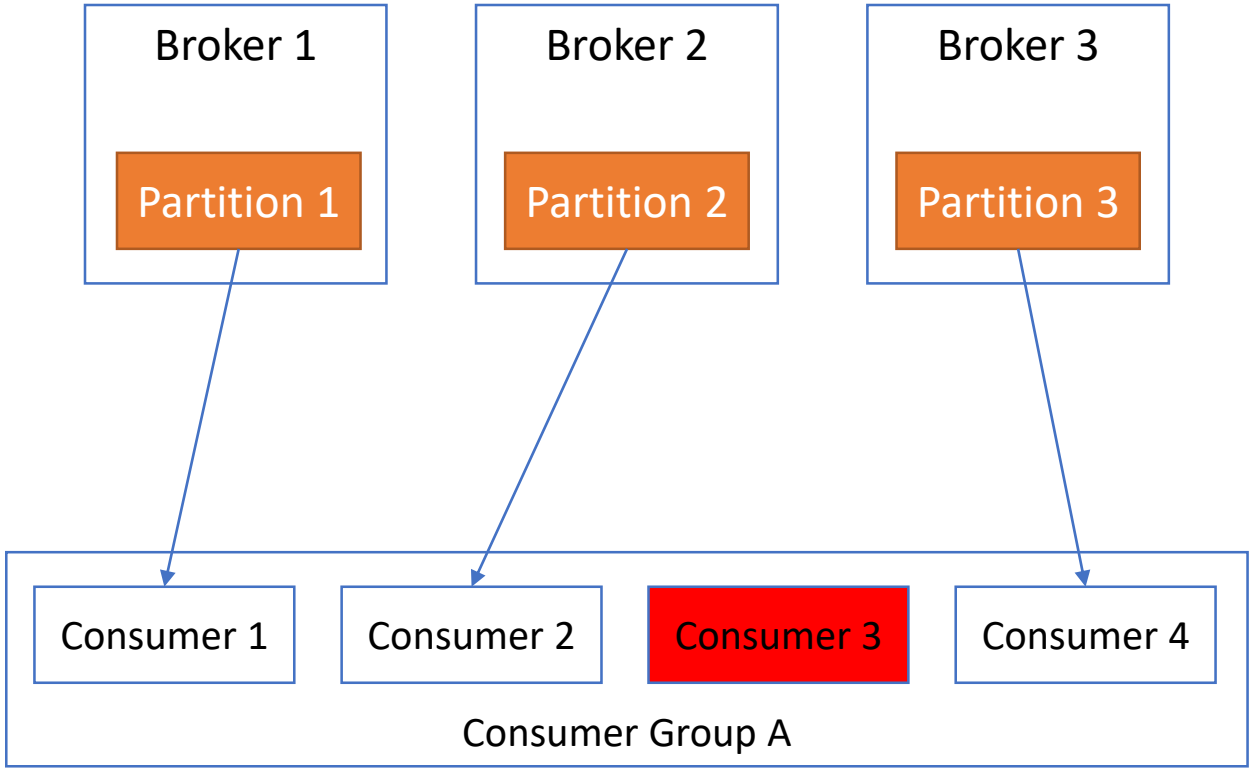
Apache Kafka: Consumer Groups

- Kafka consumers can be grouped into Consumer Group
- All consumers in a group subscribe to the same topic
  - They all share the same group id
- A message is delivered to only one consumer in a group
- A topic partition will only be delivered to one consumer in a group
  - If we have more consumers than topics: some consumers will be idle (failover)
  - If we have more topics than consumers: some will receive multiple topics
  - Kafka will always try to balance the load in a consumer group

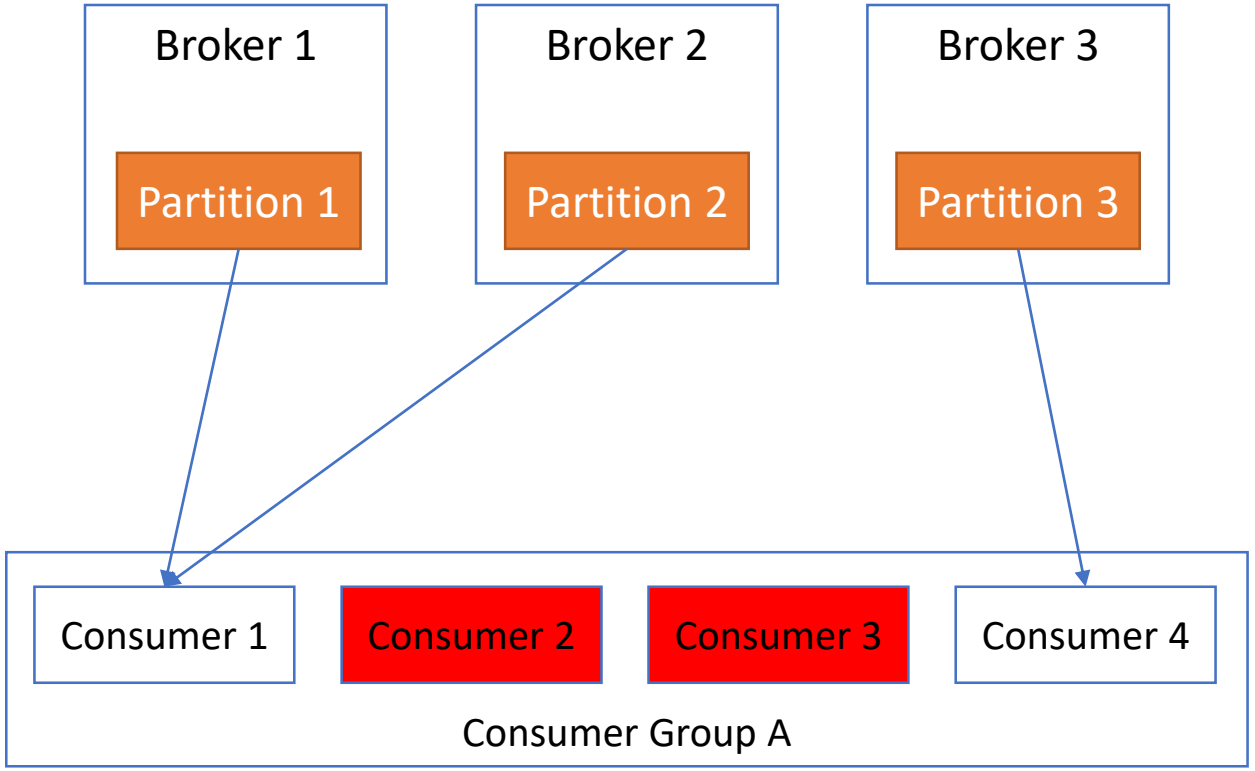
Apache Kafka: Consumer Groups



Apache Kafka: Consumer Groups



Apache Kafka: Consumer Groups



Apache Kafka: Message Delivery Semantics

- Apache Kafka provides different delivery semantics
- At most once delivery
  - This is also called fire and forget. The producer in this case does not request an acknowledgment of published messages.
  - The broker might or might not have received it ➔ at most once
  - The message can be lost and never delivered
- At least once delivery
  - This is the default configuration
  - If the producer failed to receive the acknowledgment, it will retry sending the message ➔ multiple deliveries
  - Mechanisms exist to identify and remove duplicates: using the message sequence number
- Exactly once delivery
  - This is the most complex mechanism as it requires the collaboration of both systems (sender and receiver)
  - Exactly once delivery is now possible with Kafka Streams and transactions support