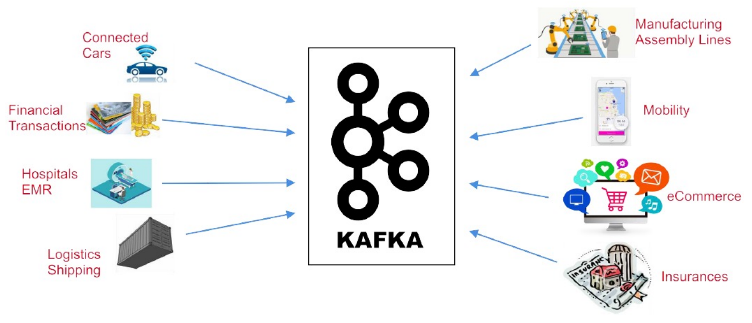
KAFKA

76

76

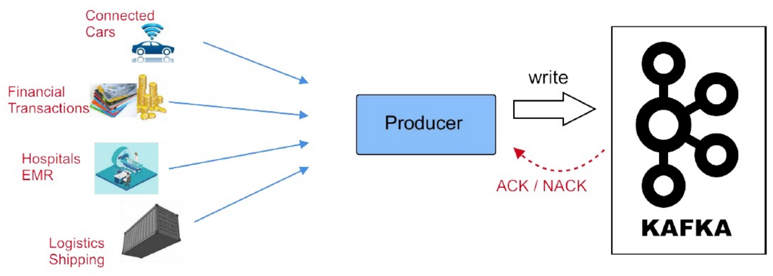
Kafka



77

77

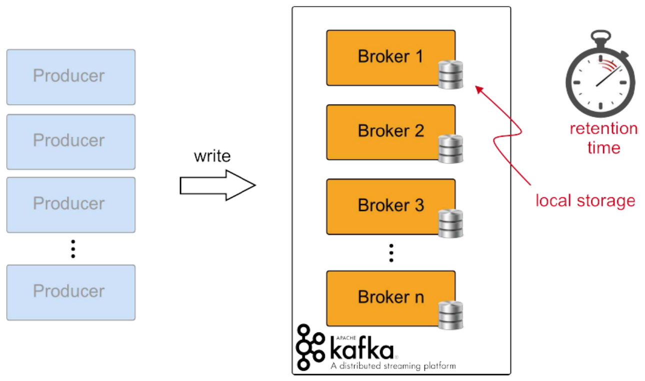
Producers



78

78

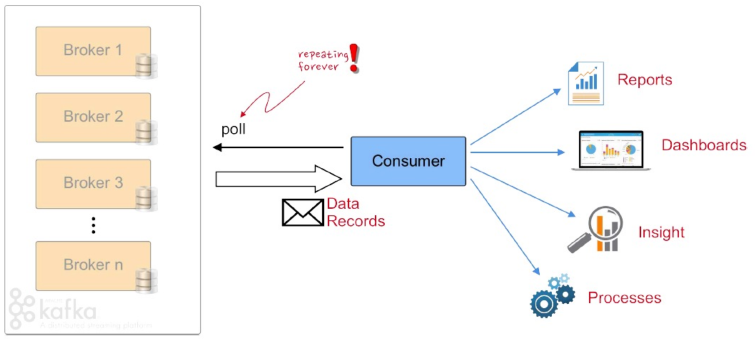
Brokers



79

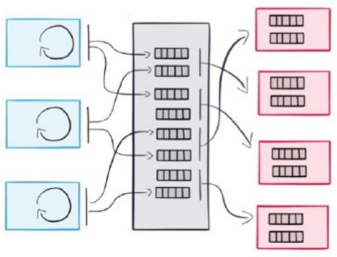
79

Consumers



Producers and Consumers

- Producers and Consumers are decoupled
- Slow Consumers do not affect Producers
- Add Consumers without affecting Producers
- Failure of Consumer does not affect System



What is Kafka?

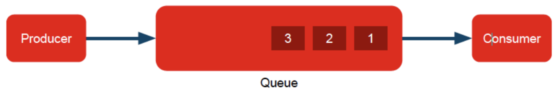
- Publish-subscribe messaging system
- Data streaming platform
- Distributed, horizontally-scalable, fault tolerant commit log

82

82

Traditional Messaging

- Reference count based message retention
- "Smart broker, dumb client"
- Per consumer filtering

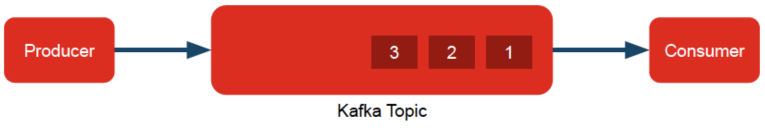


83

83

Kafka Messaging

- Time/size based message retention
  - also "compacted topic"
- "Dumb broker, smart client"
- Message stream replay



84

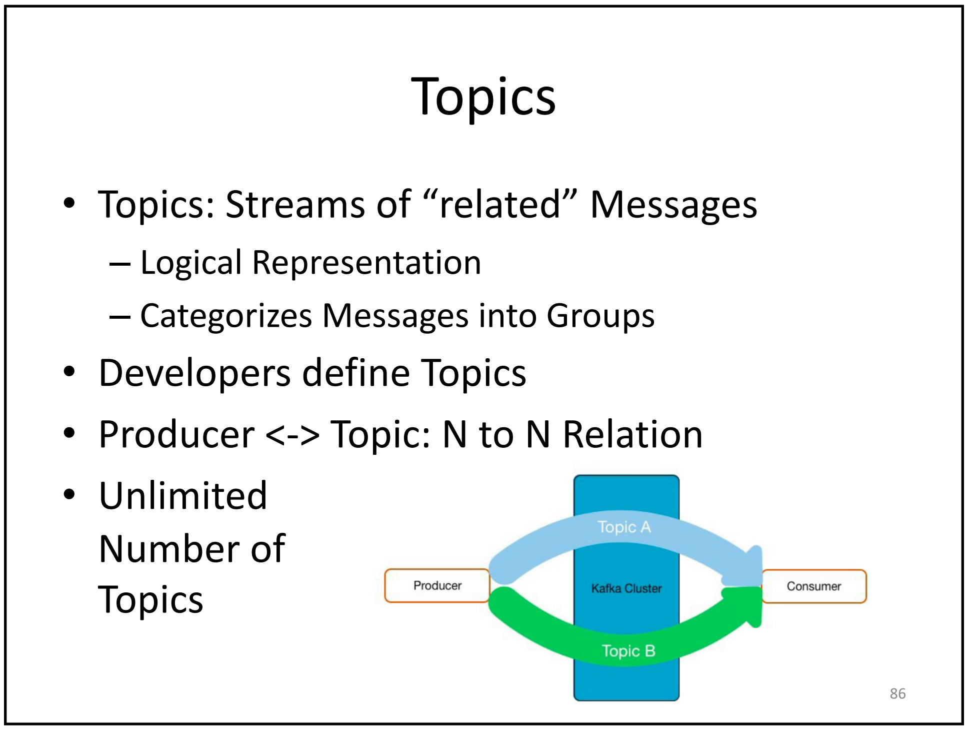
84

MOM vs Streams (Kafka)

	MOM	Kafka
Model	"Smart broker, dumb clients"	"Dumb broker, smart clients"
Durability	Volatile or durable storage	Durable storage
Storage duration	Temporary	Potentially long term
Message retention	Until consumed	Until expired or compacted
Consumer state	Broker managed	Client managed*
Filtering	Yes, per consumer	No
Stream replay	No	Yes
High availability	Replication	Replication
Protocols	AMQP, MQTT, OpenWire, Core, STOMP	Kafka
Delivery guarantee	Best effort or guaranteed	Best effort or guaranteed

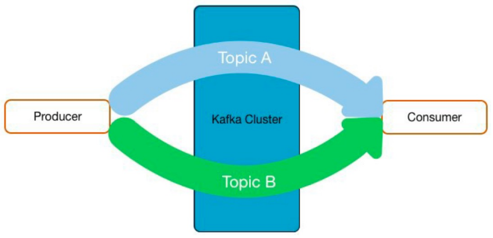
85

85



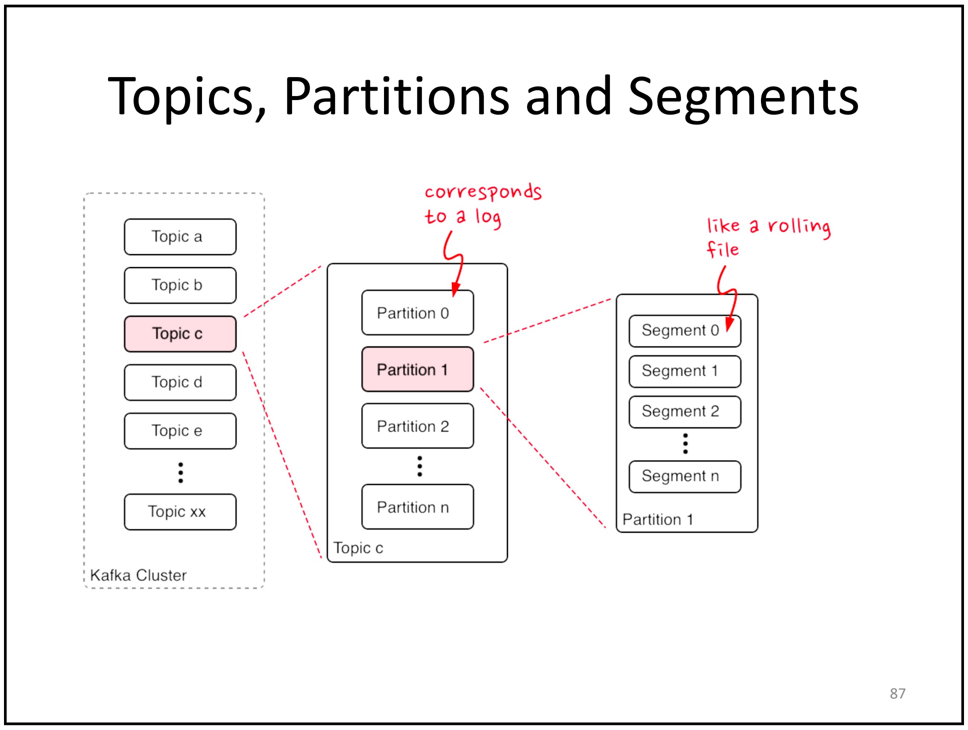
Topics

- Topics: Streams of “related” Messages
  - Logical Representation
  - Categorizes Messages into Groups
- Developers define Topics
- Producer <-> Topic: N to N Relation
- Unlimited Number of Topics

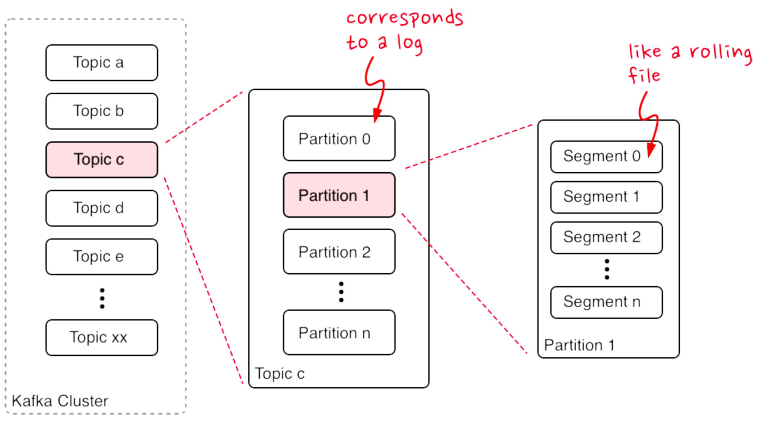


86

86



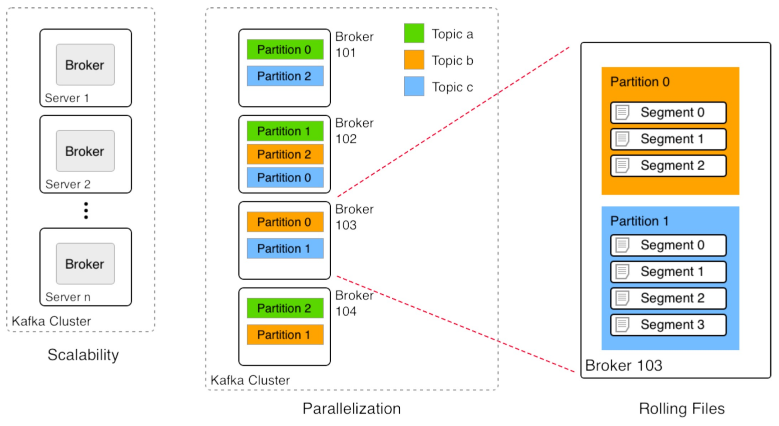
Topics, Partitions and Segments



87

87

Key Insight: Horizontal Scaling



88

88

Use Streams For…

- Scalability and performance
  - Horizontal scaling
- Message ordering
  - Within partition
- Message rewind/replay
  - Reconstruct application state by replay

89

89

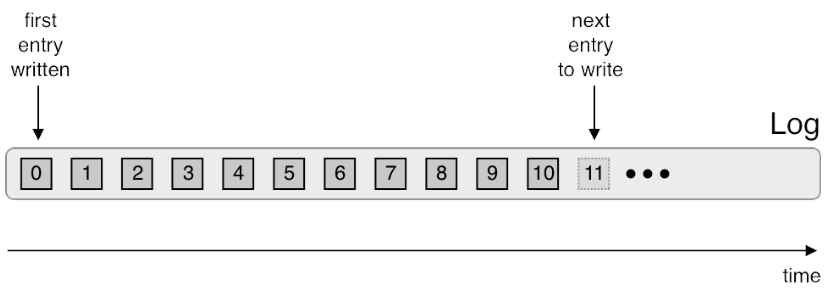
Caveats

- Kafka protocol cannot be proxied
  - Clients access brokers
  - Potential large number of TCP connections
  - Proxy via REST?
- Dumb broker, smart client
  - Decide "right" number of partitions per topic
  - Adding partition may change destination
  - Cannot remove

90

90

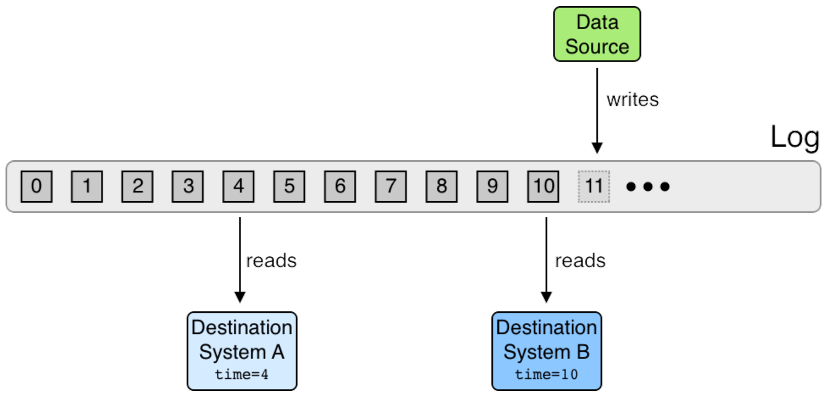
The Log



91

91

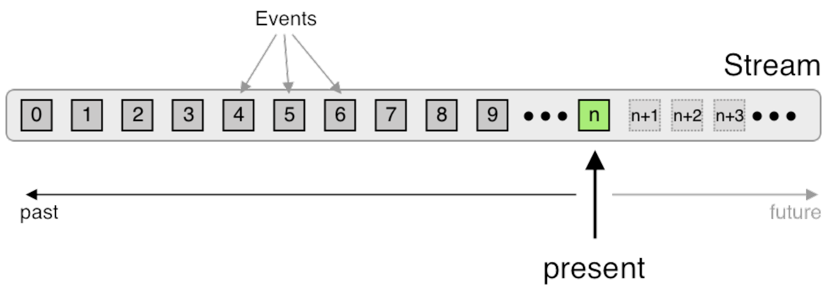
Log-Structured Data Flow



92

92

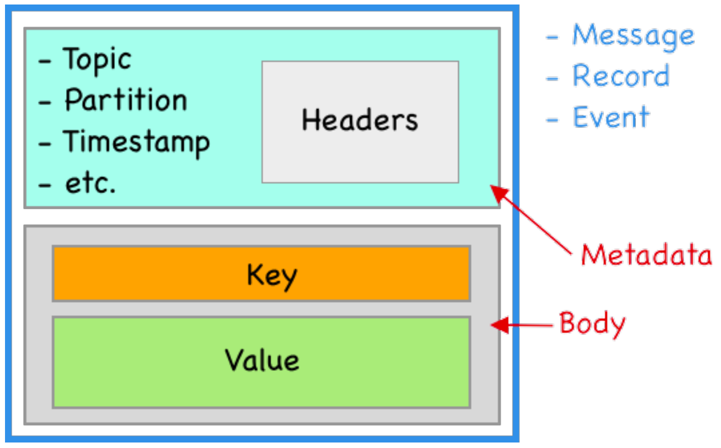
The Stream



93

93

Data Elements



94

94

Brokers Manage Partitions

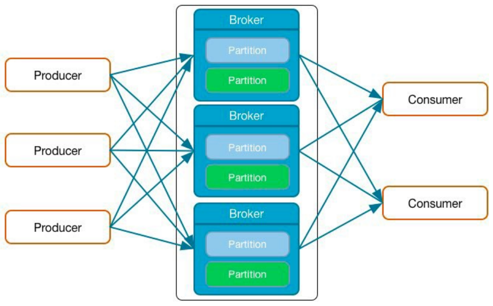
- Messages of Topic spread across Partitions
- Partitions spread across Brokers
- Each Broker handles many Partitions
- Each Partition stored on Broker’s disk
- Partition: 1..n log files
- Each message in Log identified by Offset
- Configurable Retention Policy

95

95

Broker Basics

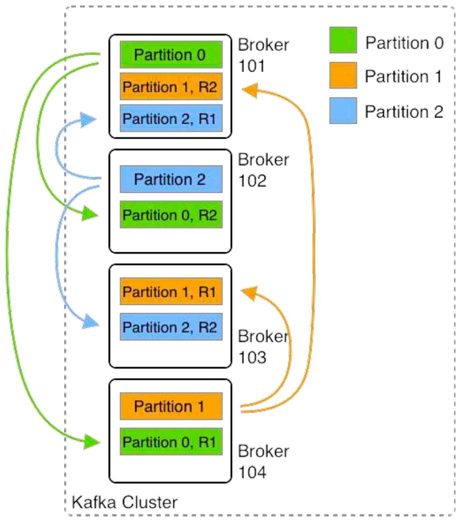
- Producer sends Messages to Brokers
- Brokers receive and store Messages
- A Kafka Cluster can have many Brokers
- Each Broker manages multiple Partitions



96

96

Broker Replication



Replication (Factor 3)

97

97

Producer Basics

- Producers write Data as Messages
- Can be written in any language
  - Native: Java, C/C++, Python, Go,, .NET, JMS
  - More Languages by Community
  - REST Server for any unsupported Language
- Command Line Producer Tool

98

98

Load Balancing and Semantic Partitioning

- Producers use a Partitioning Strategy to assign each message to a Partition
- Two Purposes:
  - Load Balancing
  - Semantic Partitioning
- Partitioning Strategy specified by Producer
  - Default Strategy: hash(key) % number_of_partitions
  - No Key → Round-Robin
- Custom Partitioner possible

99

99

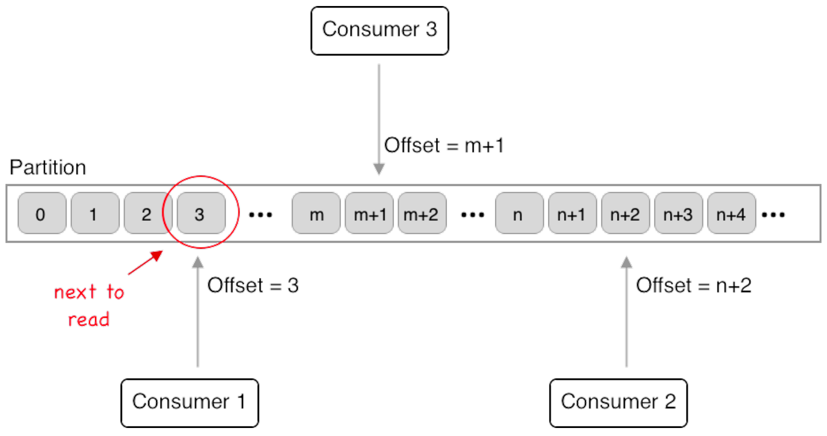
Consumer Basics

- Consumers **pull** messages from 1..n topics
- New inflowing messages are automatically retrieved
- Consumer offset
  - Keeps track of the last message read
  - Is stored in special topic
- CLI tools exist to read from cluster

100

100

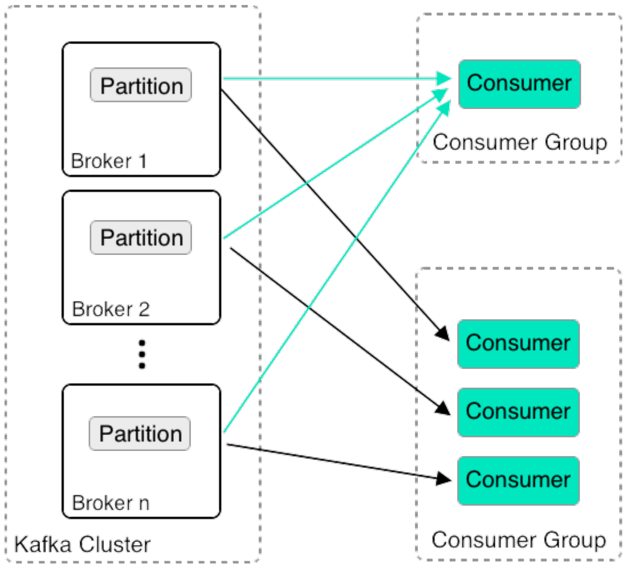
Consumer Offset



101

101

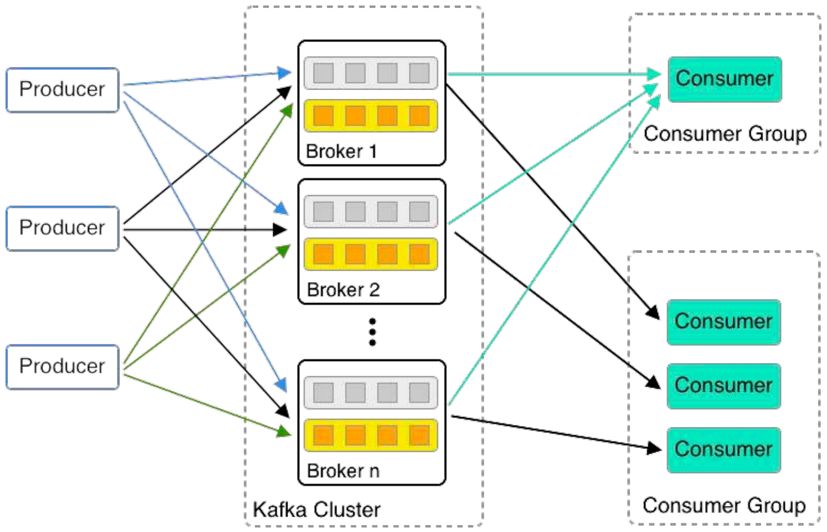
Distributed Consumption



102

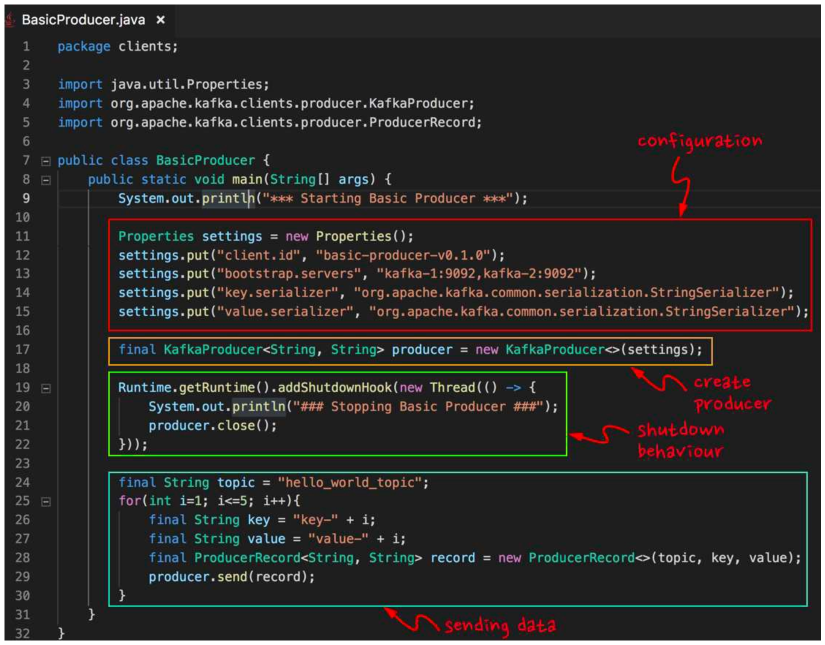
102

Scalable Data Pipeline



103

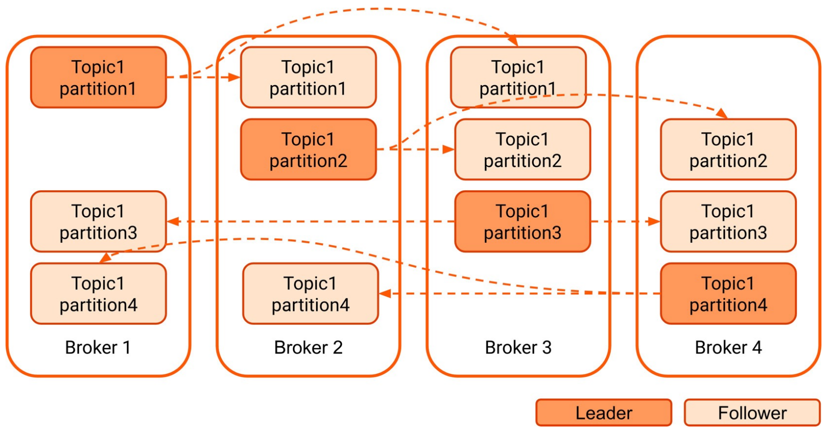
103



```java
package clients;

import java.util.Properties;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class BasicProducer {
    public static void main(String[] args) {
        System.out.println("*** Starting Basic Producer ***");

        Properties settings = new Properties();
        settings.put("client.id", "basic-producer-v0.1.0");
        settings.put("bootstrap.servers", "kafka-1:9092,kafka-2:9092");
        settings.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        settings.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        final KafkaProducer<String, String> producer = new KafkaProducer<>(settings);

        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            System.out.println("### Stopping Basic Producer ###");
            producer.close();
        }));

        final String topic = "hello_world_topic";
        for(int i=1; i<=5; i++){
            final String key = "key-" + i;
            final String value = "value-" + i;
            final ProducerRecord<String, String> record = new ProducerRecord<>(topic, key, value);
            producer.send(record);
        }
    }
}
```

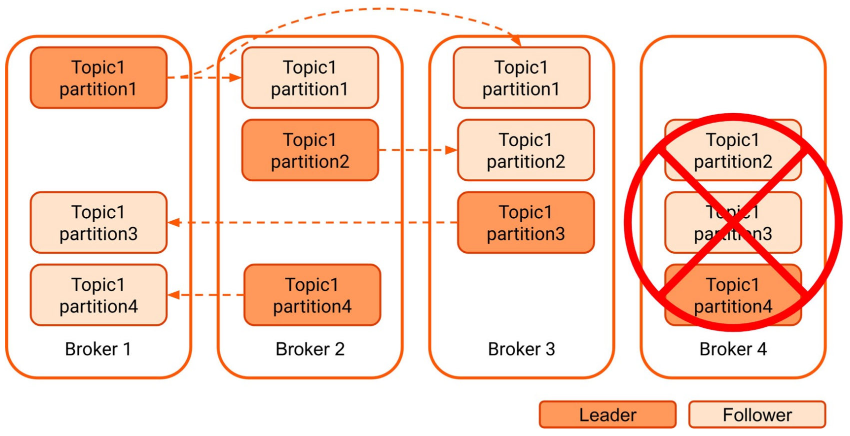
104

Partition Leadership and Replication



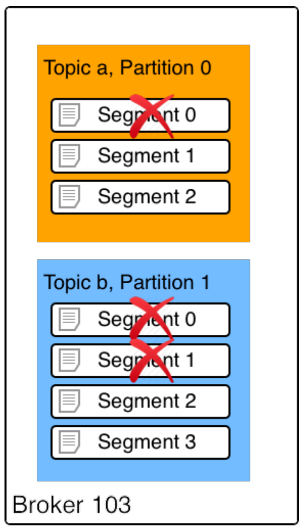
105

Partition Leadership and Replication



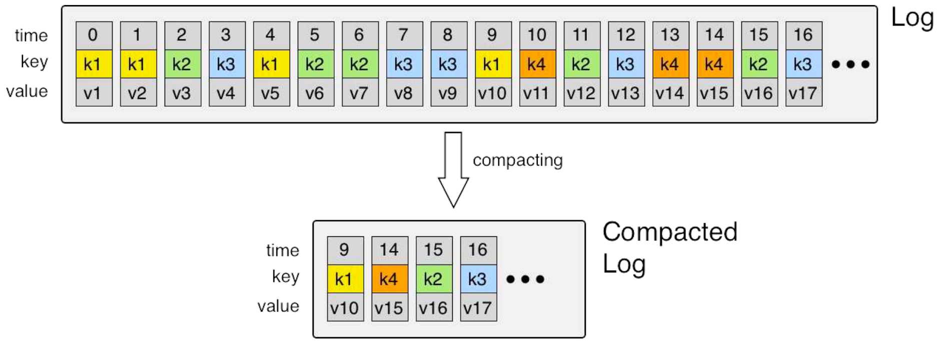
Data Retention Policy

- How long (default: one week)
- Set **globally** or **per topic**
- Business decision
- Cost factor
- Compliance factor (e.g., GDPR)



Retention Policy

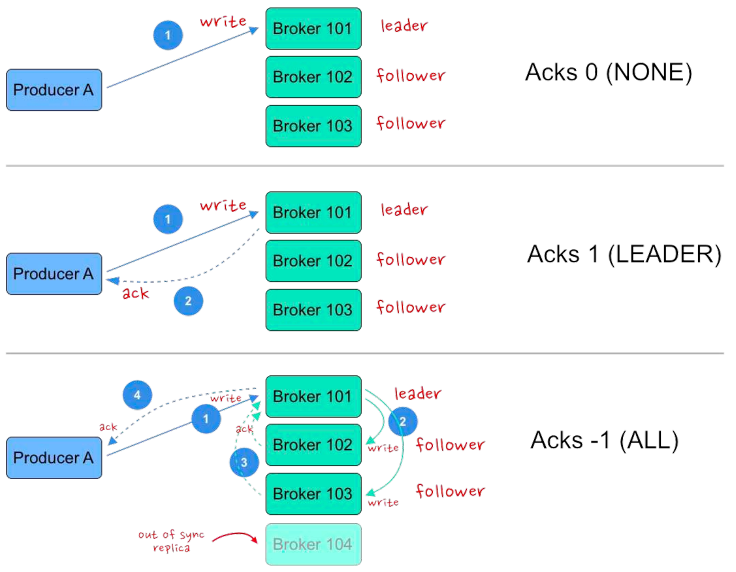
Compacted Topics



108

108

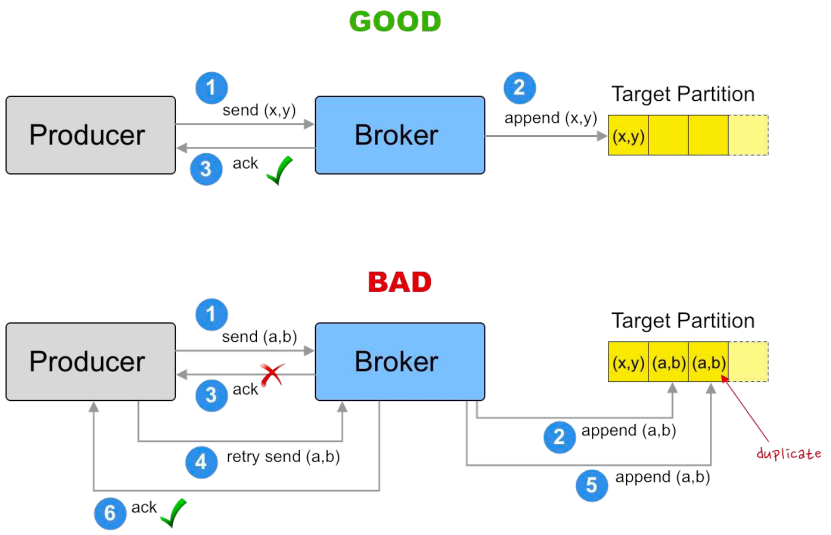
Producer Guarantees



109

109

Challenge: Duplicate Messages



110

110

Delivery Guarantees

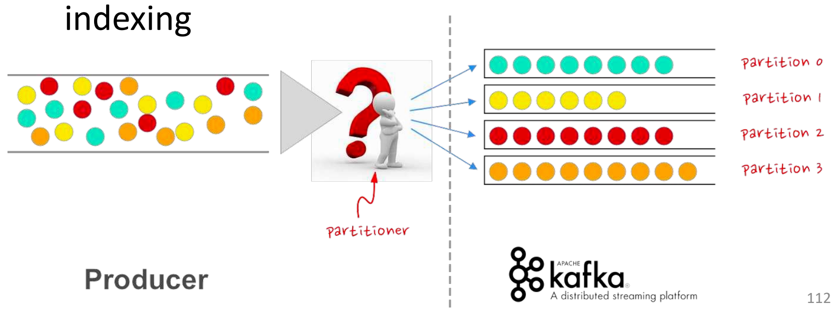
- Exactly-once semantics
  - Sequence numbers for duplicates
- Transactional guarantees
  - Batch writes across partitions
  - Commit consumer offset while processing data

111

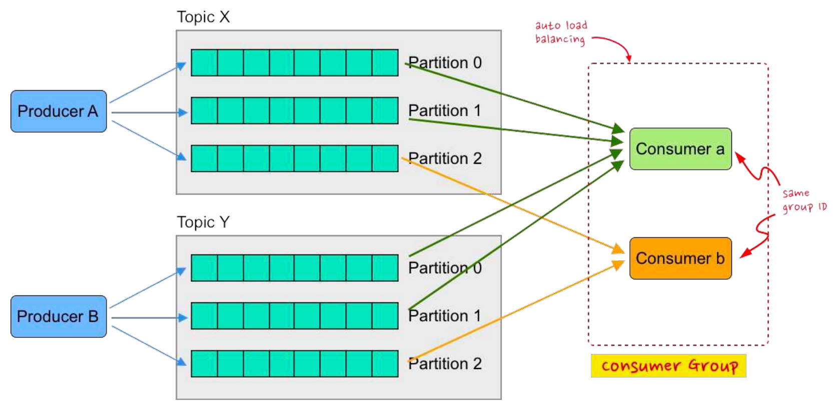
111

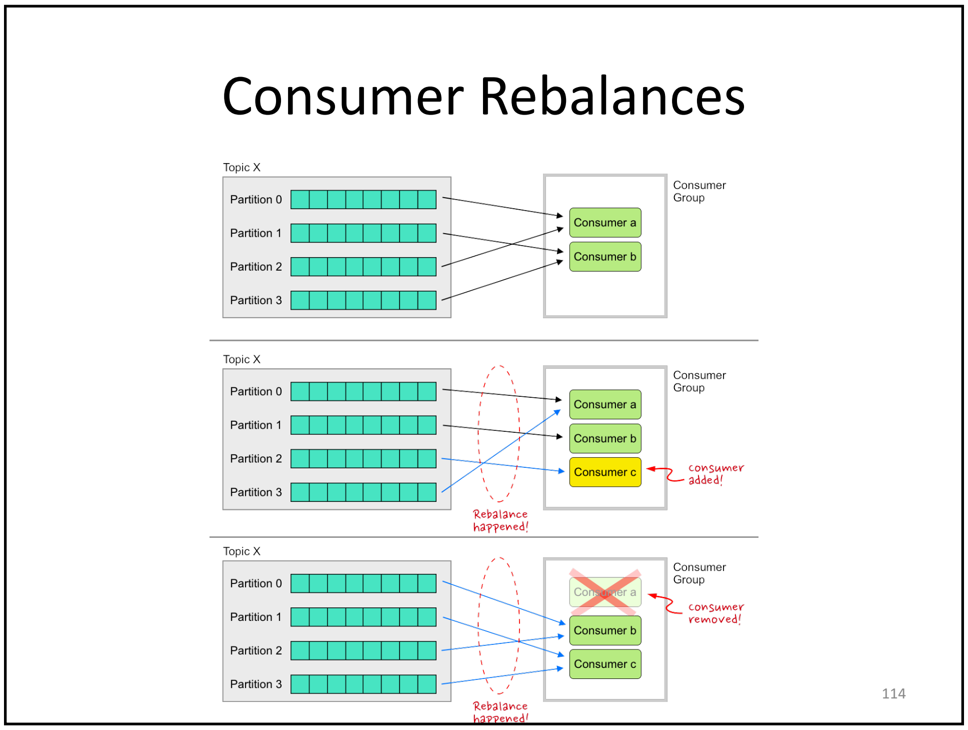
Partitioning Strategies

- Why partitioning?
  - Consumers need to aggregate or join by some key
  - Consumers need ordering guarantee
  - Concentrate data for storage efficiency and/or indexing

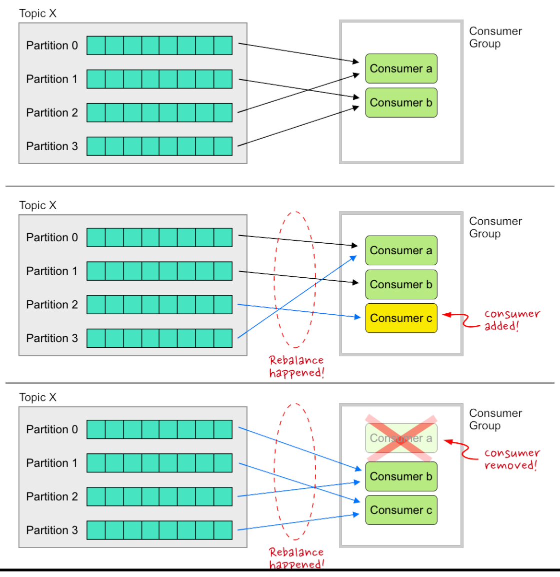


Consumer Groups





Consumer Rebalances



114

114