Real Time Data Foundations

Kafka

BY TED MALASKA

High Level Overview

- What is Kafka
- Architecture of Kafka
- Example: Setting up Kafka locally with Docker
- Example: Commend Line
- Configurations of a Producer
- Example: Java/Scala/Python Producer Example
- Configurations of a Consumer
- Example: Java/Scala/Python Consumer Example
- Partition & Topic Optimizations
- Impact of many Producers and/or Consumers
- Optimizing for Throughput and/or Latency
- Acknowledgement Patterns

High Level Overview

- Custom Partition Considerations
- Scaling Topics and Clusters
- Global Relocation Strategy
- Failure Considerations
- Compaction Queues
- How long to store data in Kafka
- Data Modeling for Kafka
- Routing Definition Management
- Architecture: Data Pipeline
- Architecture: Application Back bone
- Architecture: RPC
- Architecture: Datastore

What is Kafka

What is Kafka

- Message Bus
- Logging System
- Message Queue
- Storage System
- Streaming Platform

MQ Systems

- Types
  - RabbitMQ
  - MQSeries
  - ActiveMQ
- Advanced Message Queuing Protocol (AMQP)

Advanced Message Queuing Protocol (AMQP)

Feature	AMQP 0-9-1	RabbitMQ < 2.6.0	RabbitMQ 2.6.0-2.7.1	RabbitMQ >= 2.8.0
transactional `basic.publish`	yes	yes	yes	yes
transactional `basic.ack`	yes	yes	yes	yes
transactional `basic.reject`	no	no	no	yes
transactional exchange/queue/binding creation/deletion	no	no	no	no
transactional consuming/getting of messages	no	no	no	no
atomicity in single queue	yes	no	no	no
atomicity across multiple queues	no	no	no	no
error detection (e.g. invalid exchange)	undefined	immediate	immediate	immediate
sending of 'no_route' `basic.return`	undefined	immediate	on commit	on commit
effect visibility / responsibility transfer / durability	undefined	on commit	on commit	on commit

How does Kafka compare to AMQP

- Guarantees Extremally limited
- Enables 10x speed and throughput

Distributed Log

- Pub-sub vs distributed log
- Think about log4J
- Many readers one or many

What about Kafka as a Storage System

- Pros
  - Topic can store data like Table or Logs
  - Replication
  - Full TTL support
  - Compaction Support
- Cons
  - Non-deterministic* landing pattern of the data
  - Lack of Indexing support
  - Schema not strongly enforced
  - Row level storage vs columnar storage
  - Not optimized for randomize access reads and multi user access
  - Lacking basic bucketing and partition structures

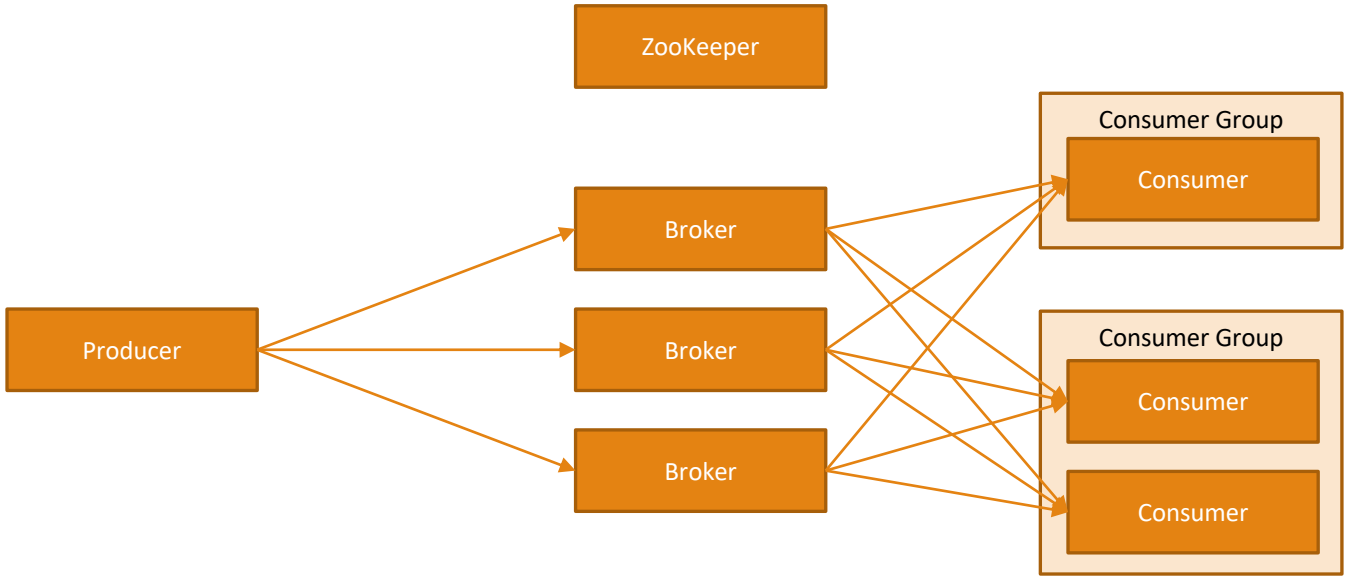
What about Kafka as a Streaming Platform

- Many types of Streaming Platforms
  - Spark Streaming
  - Flink
  - Micro Services
  - Kafka Streams

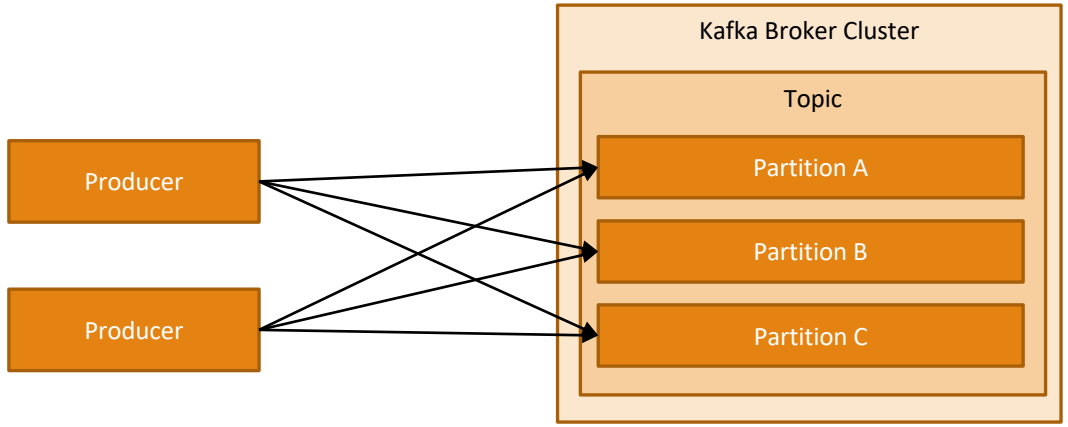
Kafka Architecture

- Producers
- Brokers
- Zookeeper
- Consumer Groups
- Consumers
- Topics
- Partitions
- Listeners
- Failure and Rebalancing

Basic Parts


ZooKeeper
Consumer Group
Consumer
Broker
Producer
Broker
Consumer Group
Consumer
Broker
Consumer

Logical View from the Producer



What is a Topic and Partition

- A Topic is a Topic
  - Is it like a Database Table?
- A Partition is like a log file
  - Once written order is maintained
  - Many log files

What is a Producer Record

- Key Value Structure
- Key and Value are byte arrays
- Can apply Serializer

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("acks", "all"); props.put("retries", 0);
props.put("batch.size", 16384); props.put("linger.ms", 1);
props.put("buffer.memory", 33554432);
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

Producer<String, String> producer = new KafkaProducer<>(props); for (int i = 0; i < 100; i++) {
        producer.send(new ProducerRecord<String, String>("my-topic", Integer.toString(i), Integer.toString(i)));
}

producer.close();
```

How does a Producer Deliver

- Steps
  - Add record into send function
  - Record goes into a buffer depending on it’s topic, partition, and broker
  - Buffer is triggered to publish
  - Sends some data to a broker
  - Asks for N number of acknowledgements
  - Receives acknowledgements
  - Fires of listener’s Futures
- Considerations
  - Round trip takes time
  - Multithreaded Options (Possible out of order sends)

What is a Listener's Future

- It is a Object that has a method that will get fired on record delivery or failure

```
class TrainingCallback(producerRecord: ProducerRecord[String, String]) extends Callback {
  override def onCompletion(recordMetadata: RecordMetadata, e: Exception): Unit = {
    if (recordMetadata != null) {
      val partitionNum = recordMetadata.partition()
      val keySize = recordMetadata.serializedKeySize()
      val valueSize = recordMetadata.serializedValueSize()
      val offset = recordMetadata.offset()

      println("Callback:{\"partitionNum\"=" + partitionNum +
        ",\"keySize\"=" + keySize +
        ",\"valueSize\"=" + valueSize +
        ",\"offset\"=" + offset +
        ",\"key\"=" + producerRecord.key +
        ",\"value\"=" + producerRecord.value + "}")
    }
  }
}
```

How does a Producer Buffering

- Buffer per Broker
- In side one buffer could be
  - Many Partitions
  - Many Topics
- Buffers are Flushed
  - On command
  - On number of records
  - On linger time

Producer Buffering

- One Buffer per Broker
  - Inside one buffer could be
    - Many Partitions
    - Many Topics
- Many Buffers per Broker
  - Depending on Thread configs
  - Next Buffer fills as one is sent
- Buffers are Flushed
  - On command
  - On number of records
  - On linger time

How does a Producer Partition

- Types of Partitioning
  - Null Key = Round Robin
  - Non-Null Key = Hash Mod

```
Int DistPartition = Math.abs(Key.hachCode() % number of Partitions)
```

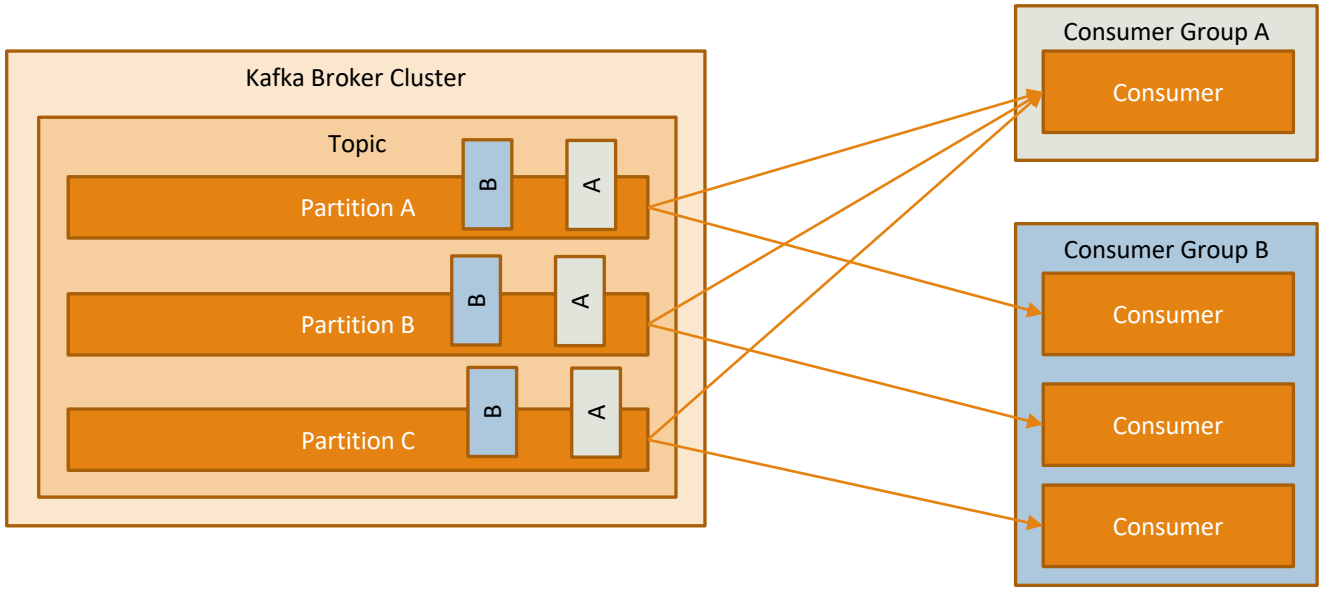
  - Custom

```
class TrainingCustomerPartitioner extends Partitioner {
 override def close(): Unit = {  }

 override def partition(topic: String, key: Any, keyBytes: Array[Byte], value: Any, valueBytes: Array[Byte], cluster: Cluster): Int = {
  val partitions = cluster.partitionsForTopic(topic);
  val key = new String(keyBytes)

  Math.abs(key.hashCode % partitions.size())
 }

 override def configure(map: util.Map[String, _]): Unit = {}
}
```

Producer considerations for # of Partitions

- Starting Theory
  - More Partitions = more throughput
- Reality
  - At least one Partition Per Broker
  - More threads to disk can go faster then one thread
  - Partitions equal more seeks on disk
  - No impact number on buffers
  - Considerations as Cluster grows (real limits to cluster size)
  - Repartitioning may effect Partition Logic’s destination selection

View from the Consumer



What is a Consumer

- Reads N number of Partitions
- Partitions can not be shared
- Consumer reads from Offsets
- Listens to Partition Assignment Changes
- Acknowledges when done with Records
- Consumers belong to a consumer group

What is a Consumer Group

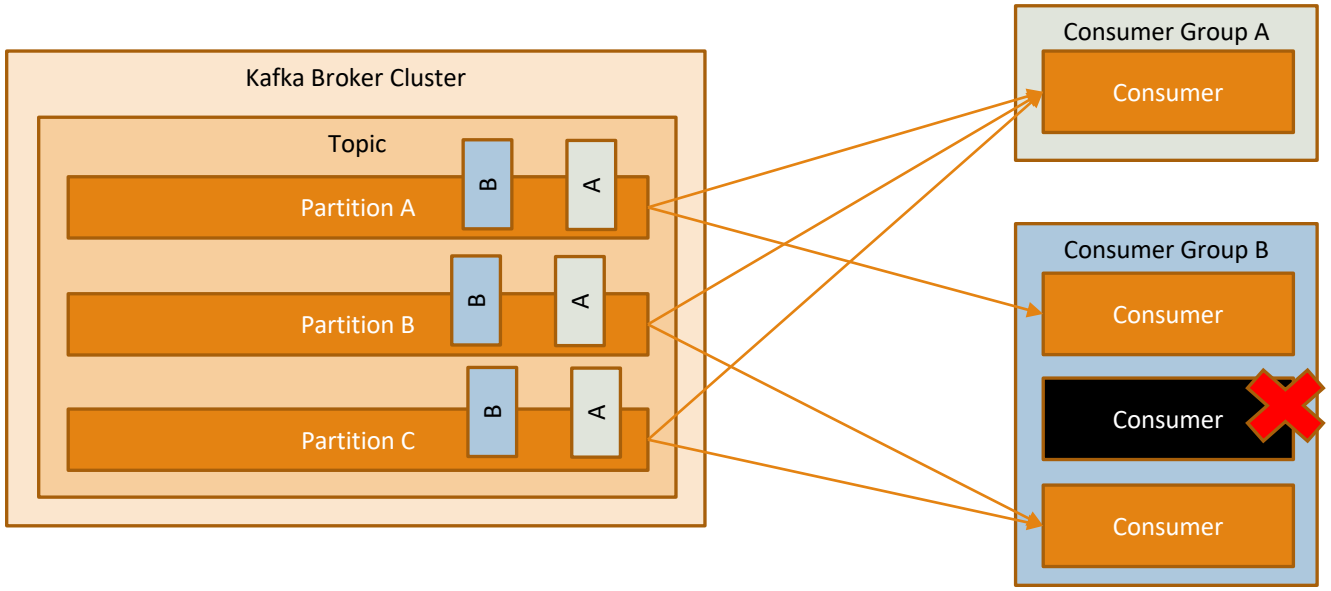
- It is a logic grouping of consumers
- With the goal that partitions are divided up to different consumers
- Logic is configurable
- ???

Partition Switching

- Listener

```
class RebalanceListener extends ConsumerRebalanceListener {
 override def onPartitionsAssigned(collection: util.Collection[TopicPartition]): Unit = {
  print("Assigned Partitions:")
  val it = collection.iterator()
  while (it.hasNext) {
   print(it.next().partition() + ",")
  }
  println
 }

 override def onPartitionsRevoked(collection: util.Collection[TopicPartition]): Unit = {
  print("Revoked Partitions:")
  val it = collection.iterator()
  while (it.hasNext) {
   print(it.next().partition() + ",")
  }
  println
 }
}
```

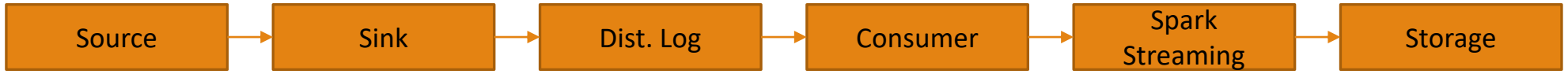
View from the Consumer



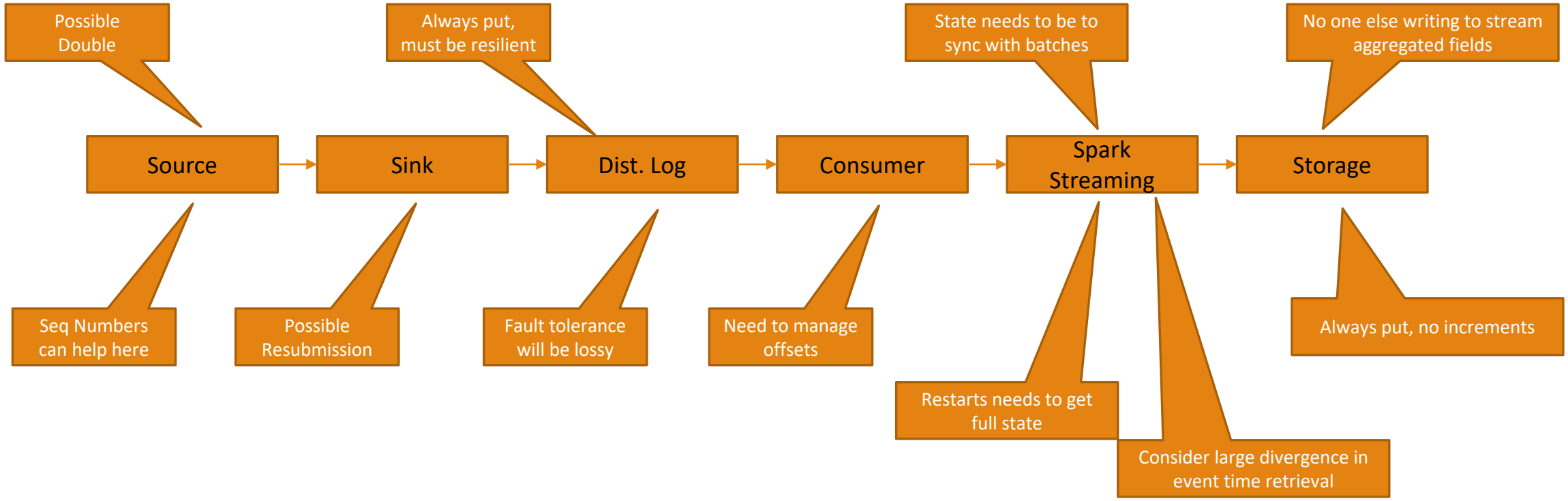
Partition Switching – Double Consume

- Consumer A reads 1000 records from partition 1
- Consumer A does stuff on 999 records
- Consumer A fails before acknowledgement
- Consumer B now gets partition 1
- Consumer B will read the same 1000 records from partition 1
- Consumer B acknowledges

Things can go wrong in many places



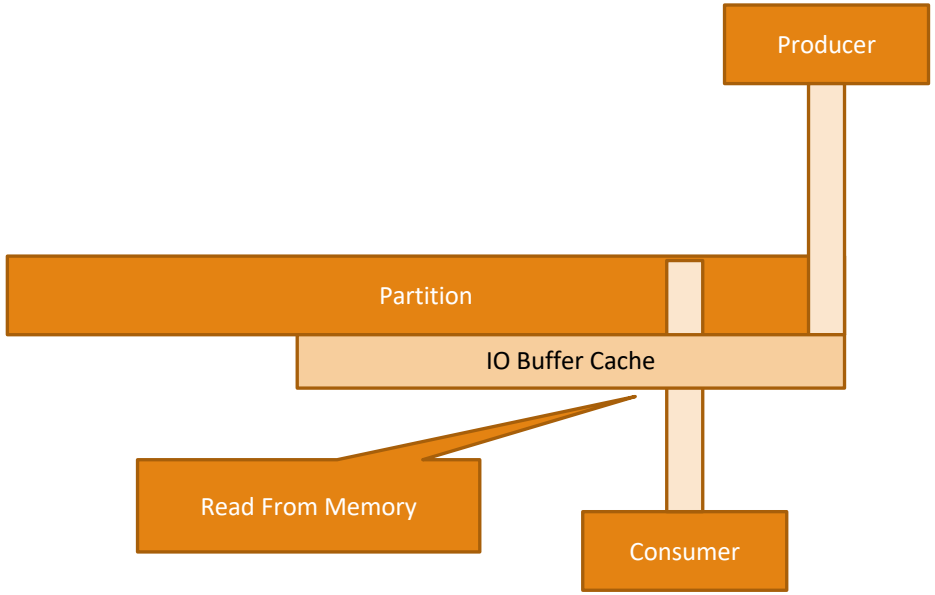
Things can go wrong in many places



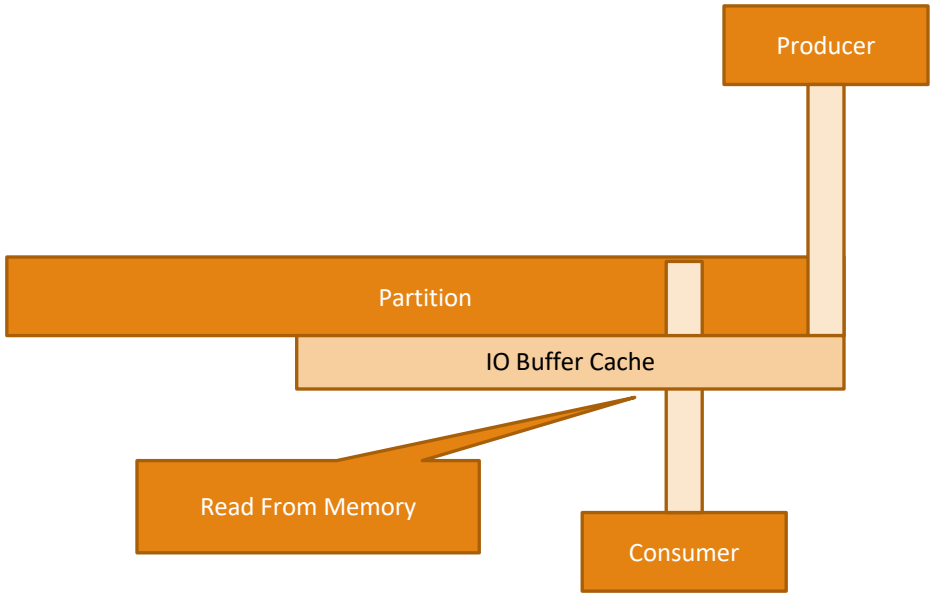
Considerations for Double Send

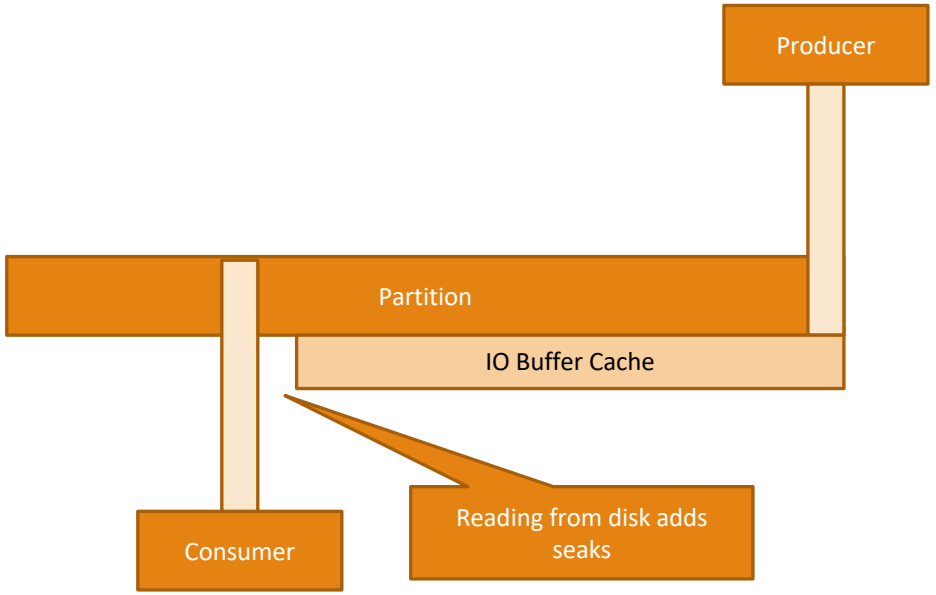
- Idempotent is great
  - Order is respected
- Locking
  - ZooKeeper or Console
  - On Key or Partition
- Using Offset numbers
  - Think about NoSQL

Reading Performance Kafka



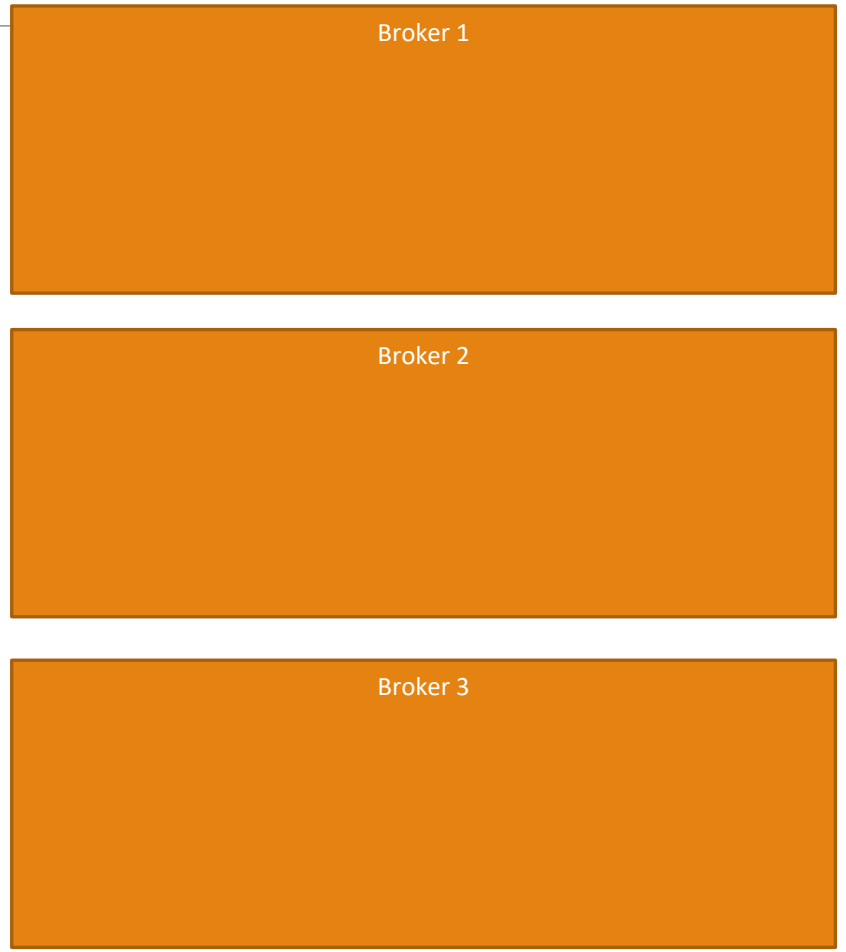
Reading Performance Kafka – Not Always





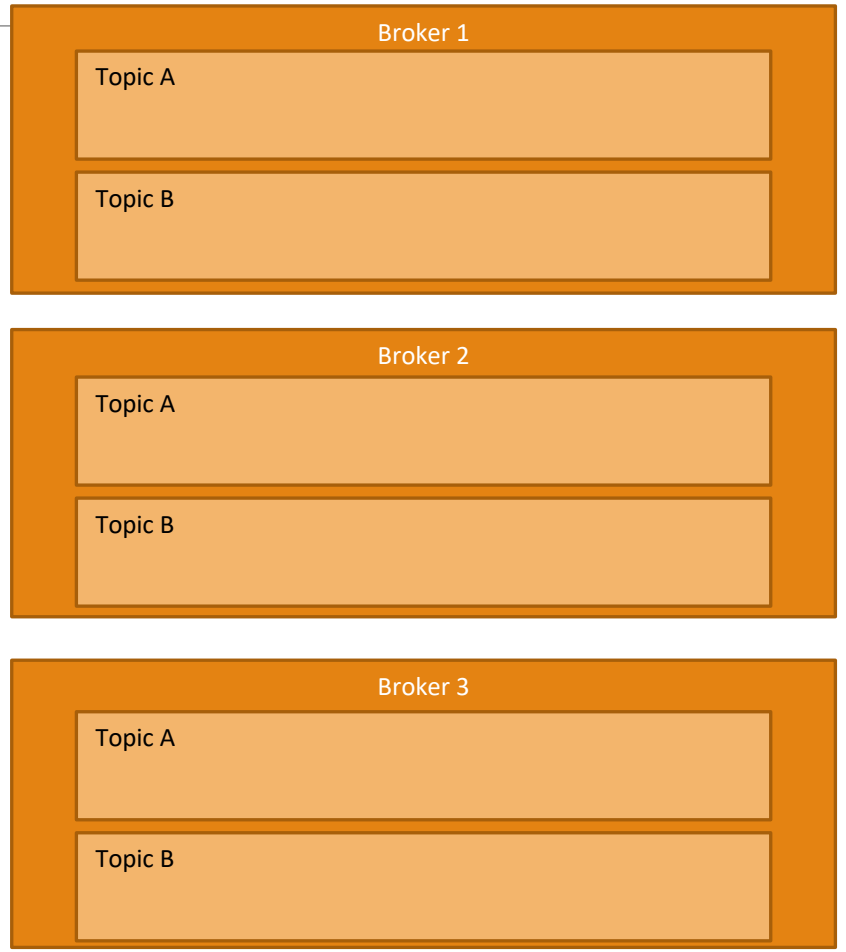
The Kafka Cluster

- A Collection of Brokers
- One will be assigned the leader



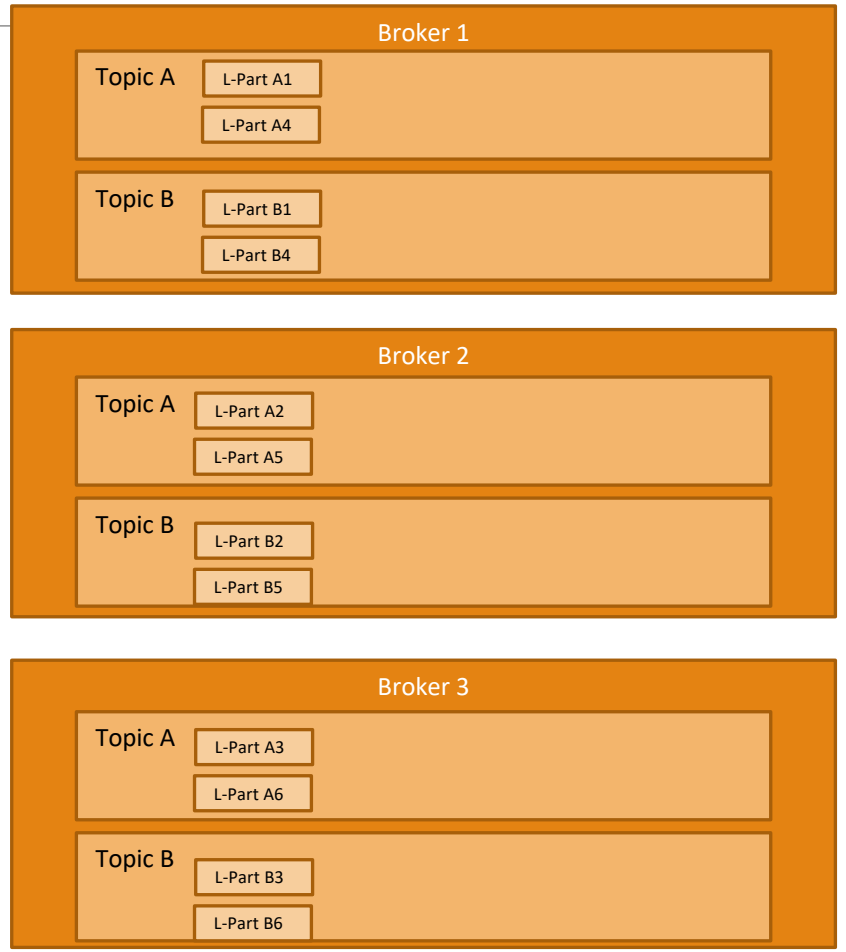
The Kafka Cluster

- Topics will exist on All Brokers*



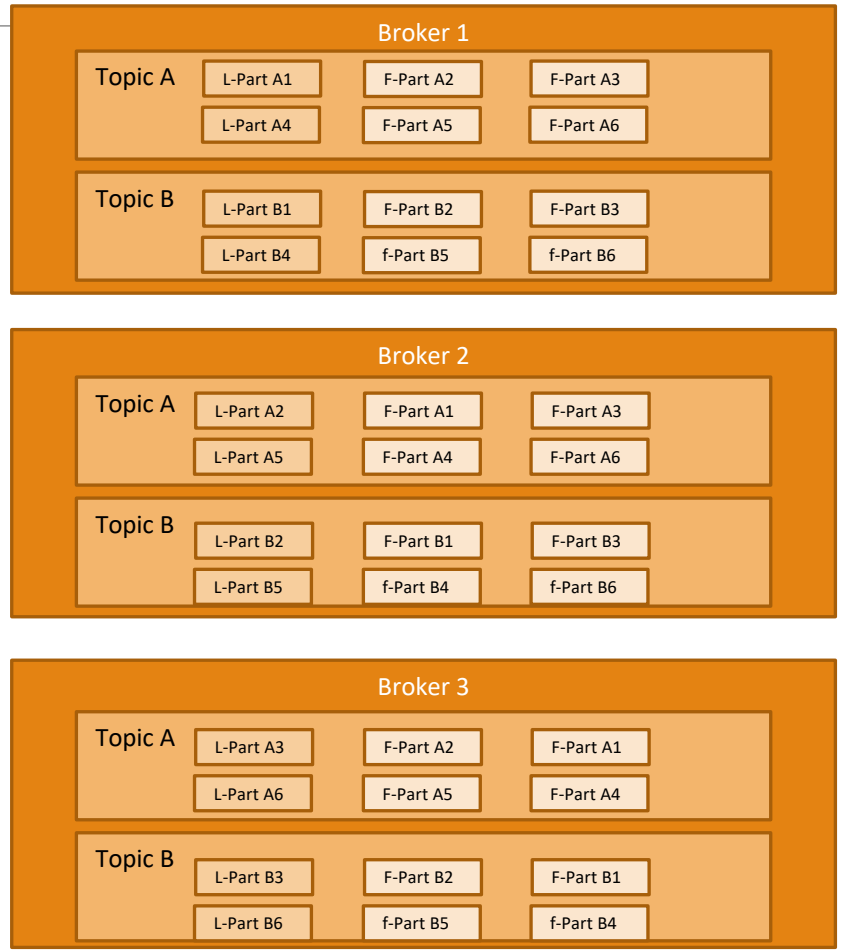
The Kafka Cluster

- Leader Partitions will be spread across Brokers

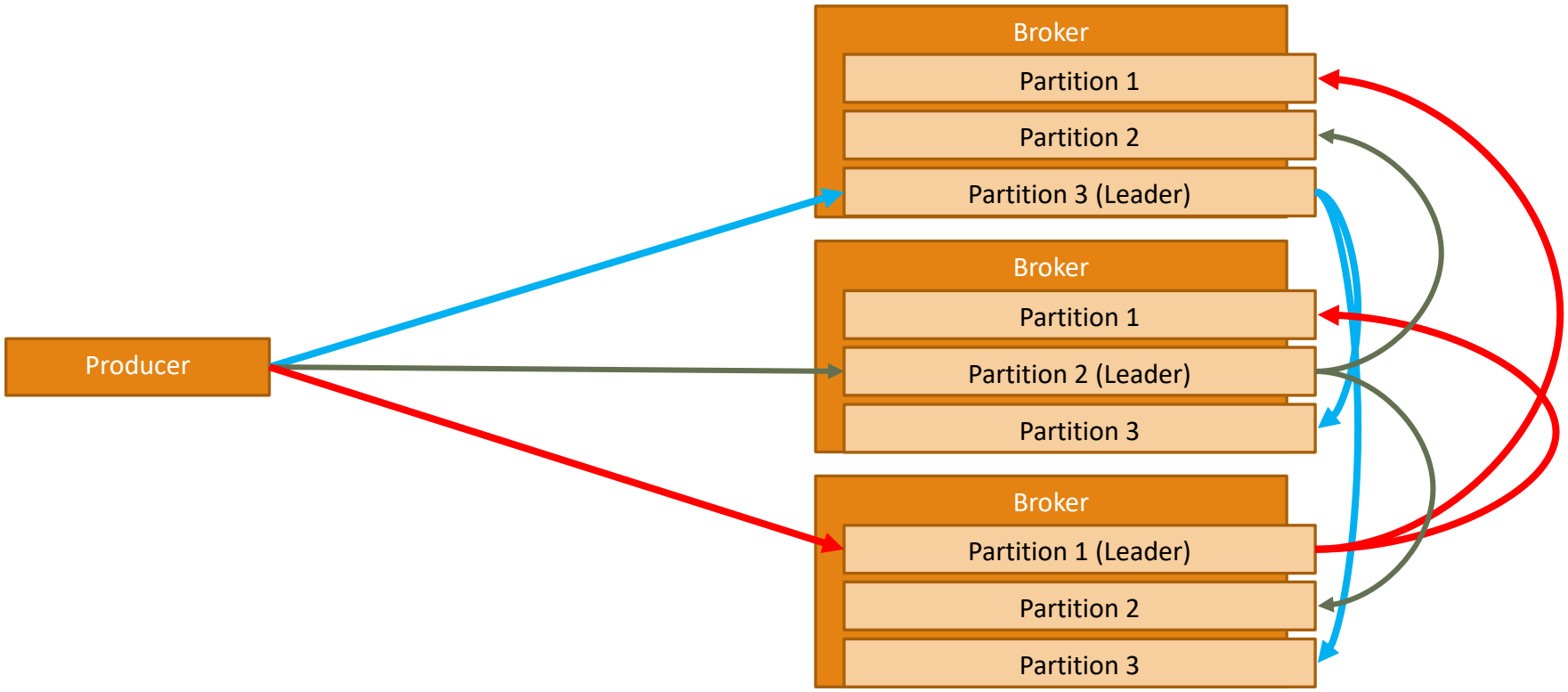


The Kafka Cluster

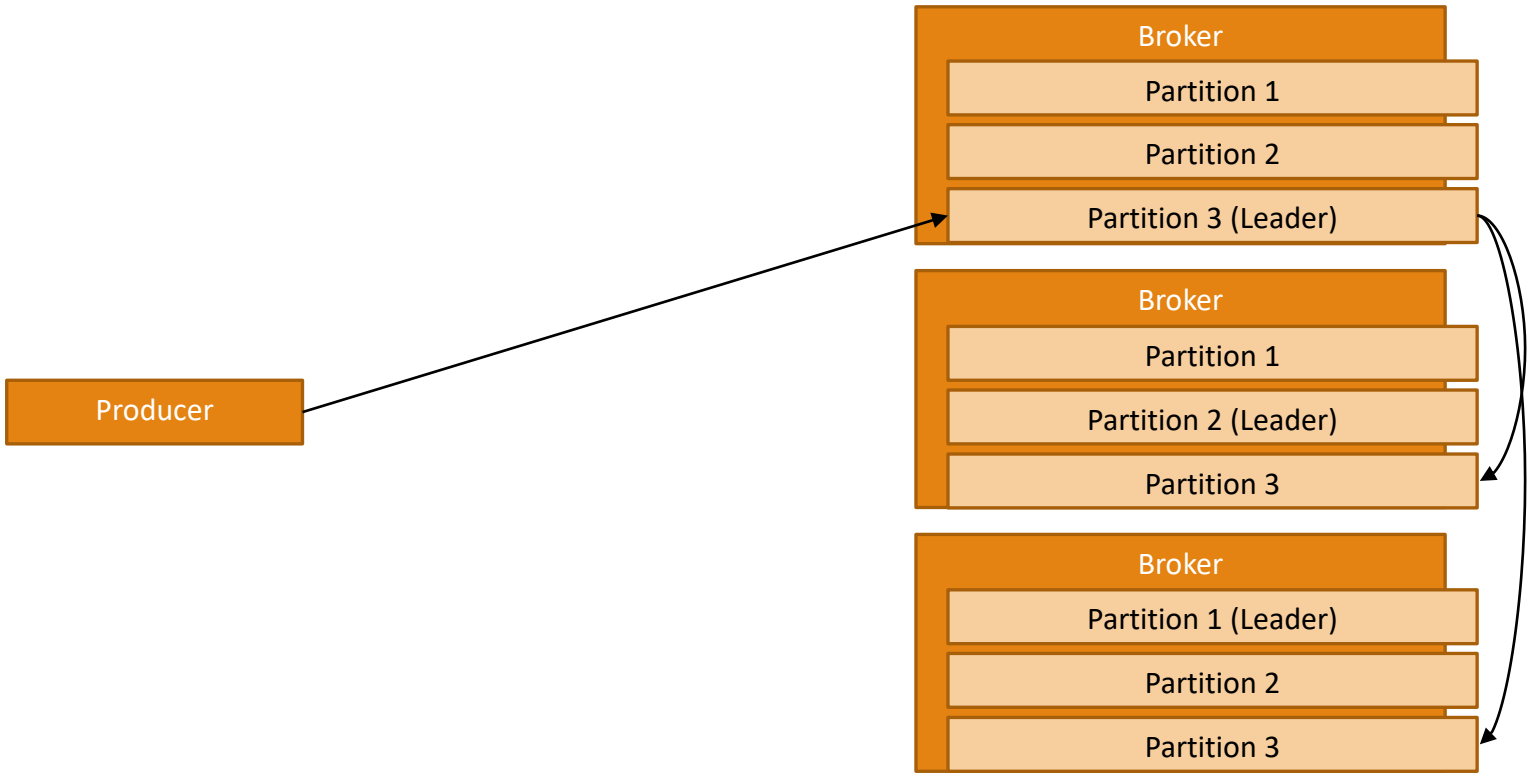
- Replicated Partitions are Spread Out



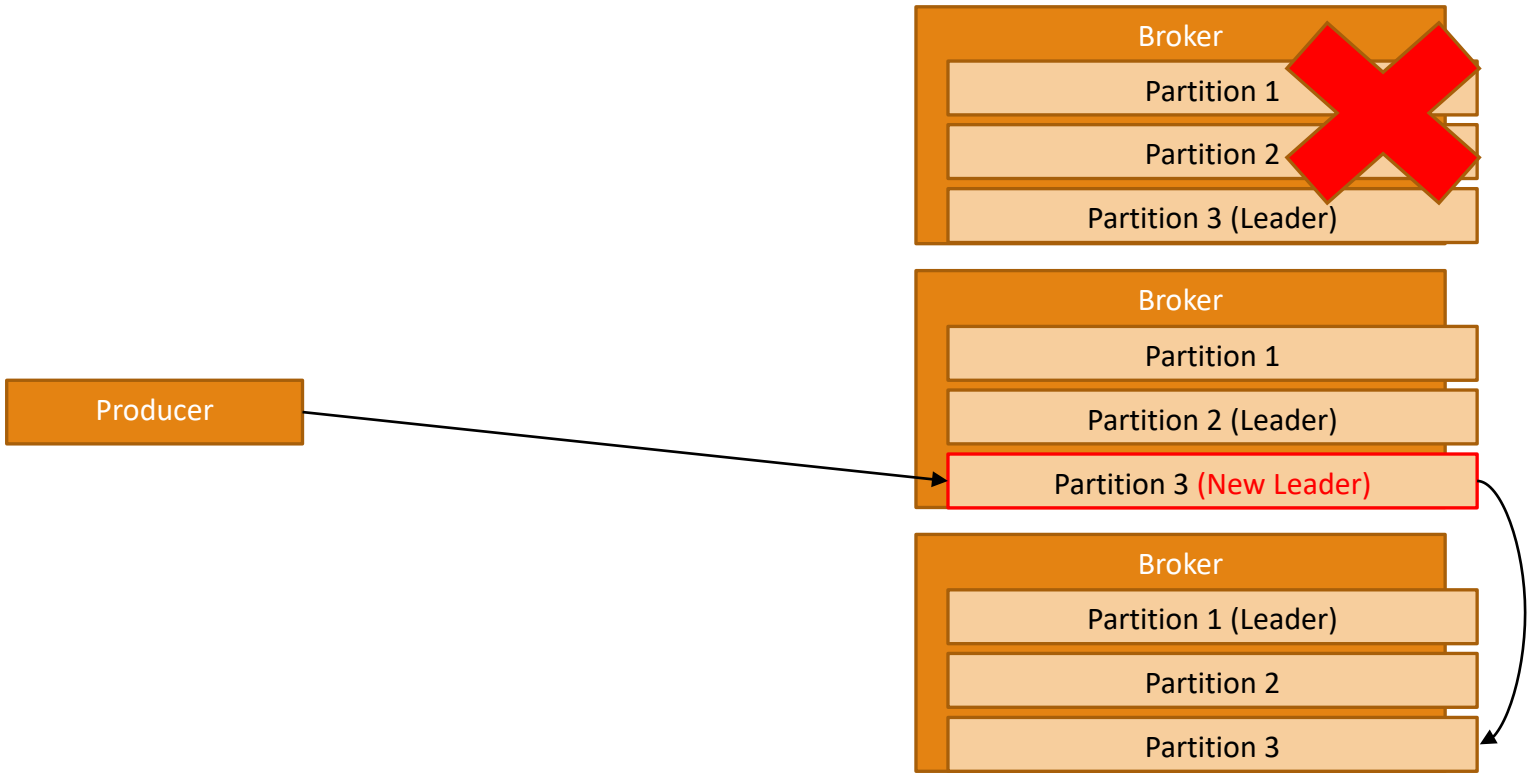
Kafka Write Path



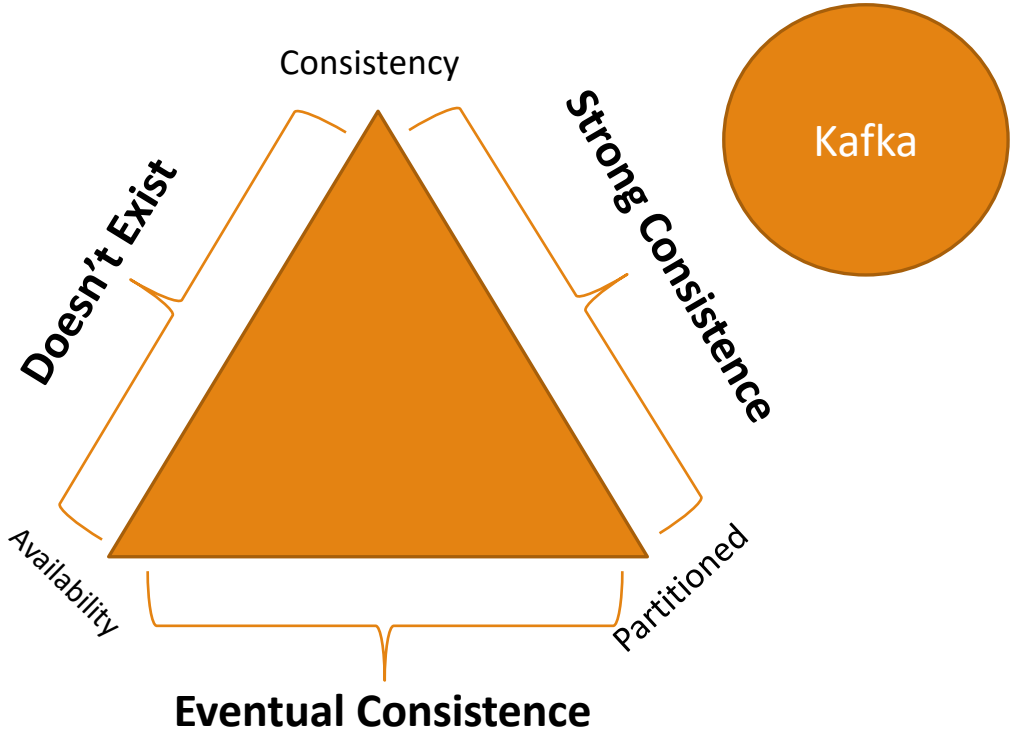
Kafka Write Path



Kafka Write Path



What do they share (CAP theorem)



What happens on Failure

- You need to define a new Leader
  - This will take time
    - Too fast and you will switch all the time
    - Too long and you have unneeded down time
- Need to make sure you have agreement on the state
- Need to replicate to get back to replication N
  - Remember the OS Cache

Benchmarking Throughput and Latency

- Max throughput
- Throughput on failure
- Throughput when off OS Cache
- Need extra to cache up

Kafka Cluster Sizing

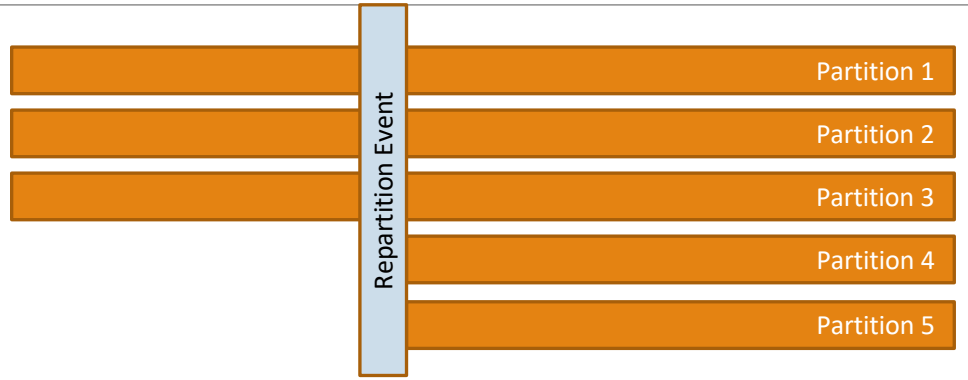
- More nodes more problems
  - Broker Broadcast
  - Failure over concerns
- Ideal 5 to 20 nodes
- Personally like smaller clusters and more clusters

Replication Between Clusters

- Implementations
  - Mirror Maker
  - Kafka Streams
  - Custom
- Buffering is Key
  - Network outages
  - Cluster bouncing
- Consistency
  - Eventual Consistence
  - Data loss chances high if you lose a cluster

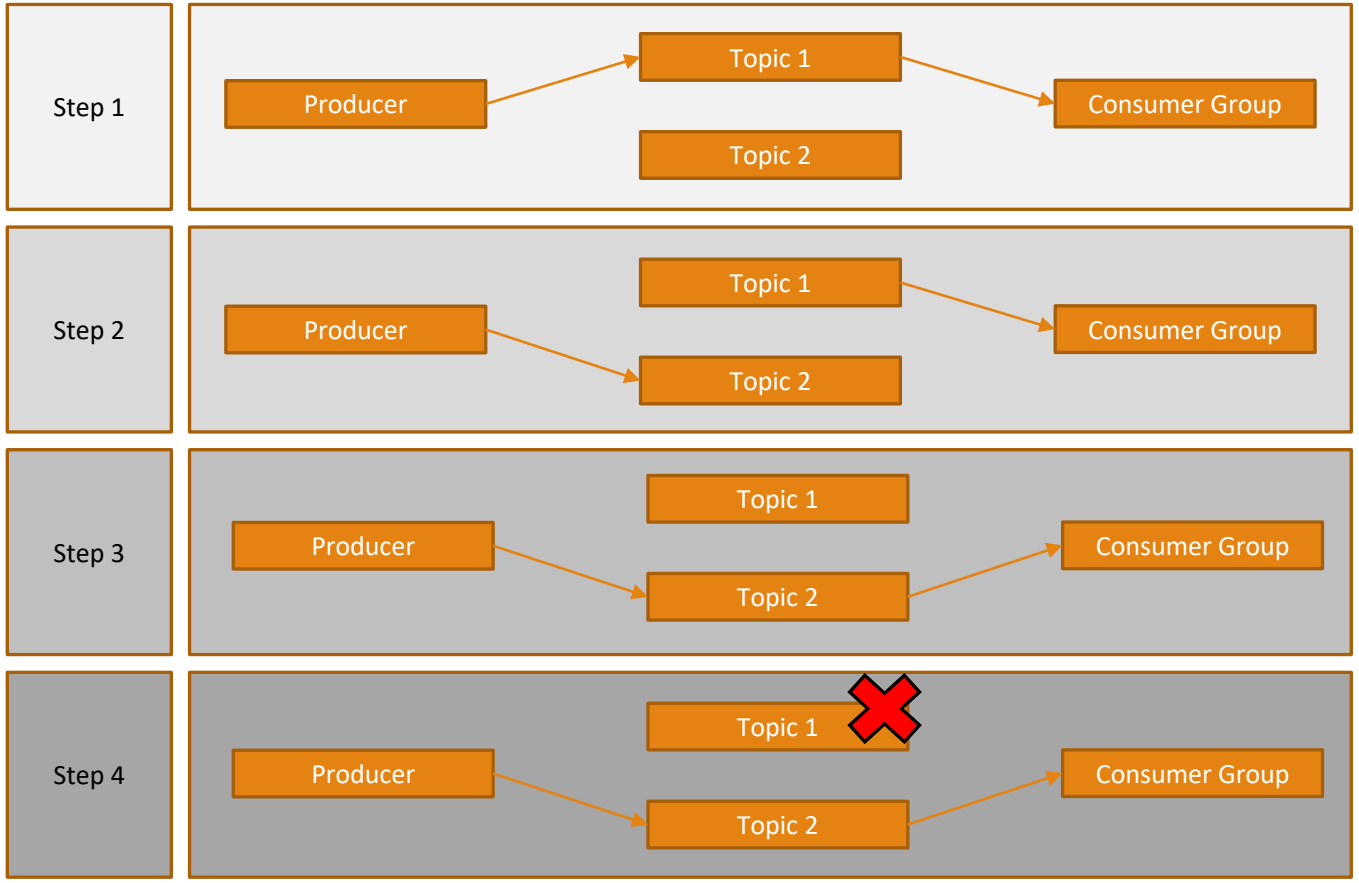
Repartitioning Topics

- Repartition in place
  - Uneven Partitions
  - Might impact Partitioning Logic



- Other option changing Topics or Clusters
  - Requires switching lock
  - Required if you live in the real world

Switching Clusters



Summary & Q/A

Why Kafka

- Throughput
- Partitions
- Order
- Replay ability
- TTL

Throughput

- Files per Partition
- Append only
- IO Buffer Stuff
- No required for the consumer guarantees

Partitions

- RePartition solves a boat load of processing problems
  - Joins
  - Distributed cache
- Custom Partitioning
- Listeners
- Custom Assignment

Order

- Determinist behavior is the key to failure recover
- Partition and Order can be used to simulate eventually consistent transactions

Replay Ability

- Order mixed with replay allows for failure recovery
- Testing
- Buffering
- Multi consumers

TTL

- Data will age out
- Be careful not to fill your drives
- Allows for a dumb simple way to remove data
- Satisfies most desires for consumer guarantees

Summary & Q/A

Setting up Kafka

- Set up Kafka
- Create some topics
- Push some data to Kafka
- Read some data to Kafka
- Play around with Partitions a little

Setting up Kafka Locally

- We are going to use Docker Containers
- docker pull wurstmeister/kafka
- https://hub.docker.com/r/wurstmeister/kafka/
- spotify/docker-kafka
- https://hub.docker.com/r/spotify/kafka/

Create a Topic

- **bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test**
- **bin/kafka-topics.sh --list --zookeeper localhost:2181**

Test out out topic with the cmd line

- **bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test**
- **bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning**

Setting Up Java Env

1. **Intellij**
2. **Maven**
3. **Pom File**

Java based producer

Lets look at the code

Java based Custom Partitioner

Let’s look at the code

Java based Consumer

Let’s look at the code

Java based Consumer Listeners

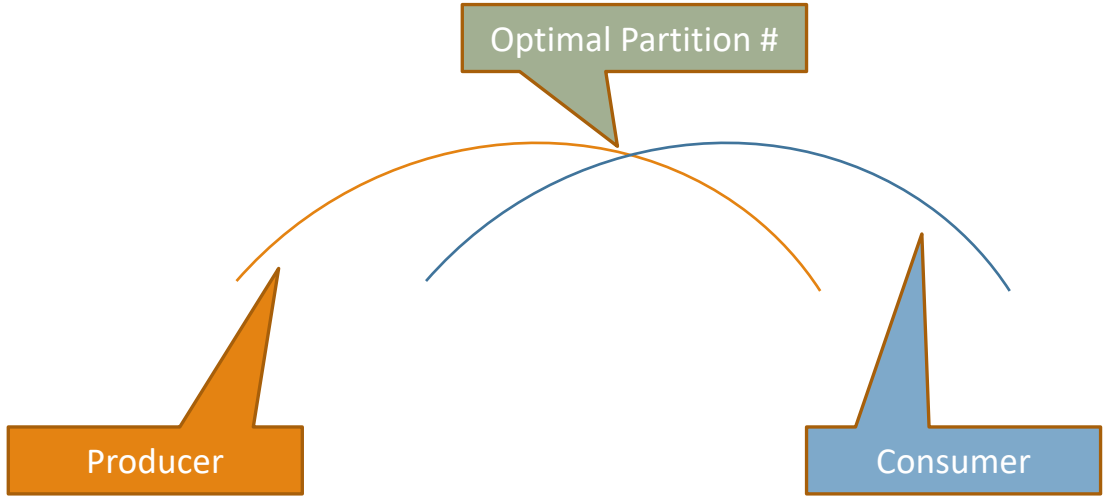
Let’s look at the code

Configuring Things

1. Number of Partitions
2. Linger Times
3. Batch sizes
4. Message sizes

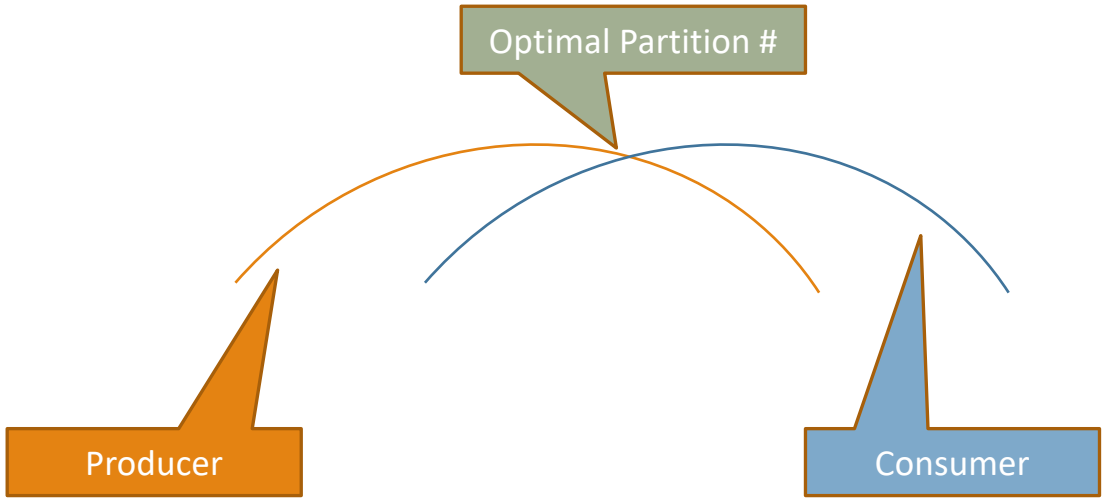
Configuring Things

1. Number of Partitions
    1. Benchmark for Producers
    2. Benchmark for Consumers
    3. Benchmark during failure
    4. Identify recovery time
    5. Look for the double hump



Configuring Things

1. Look for the double hump



Configuring Things

1. Linger Time
    1. Start with 0 and then try 1 millisecond

Configuring Things

1. Batch Sizes
    1. Start with a 1000 and look for the best option for you
    2. There will be a deminising rate of return as you increase
2. Consider message size

Configuring Things

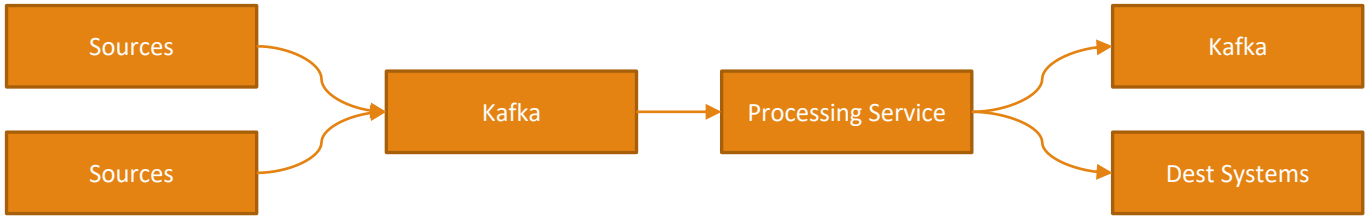
1. Message Size
    1. Try to stay under a 1KB
    2. If pre-compressed consider that also

Summary & Q/A

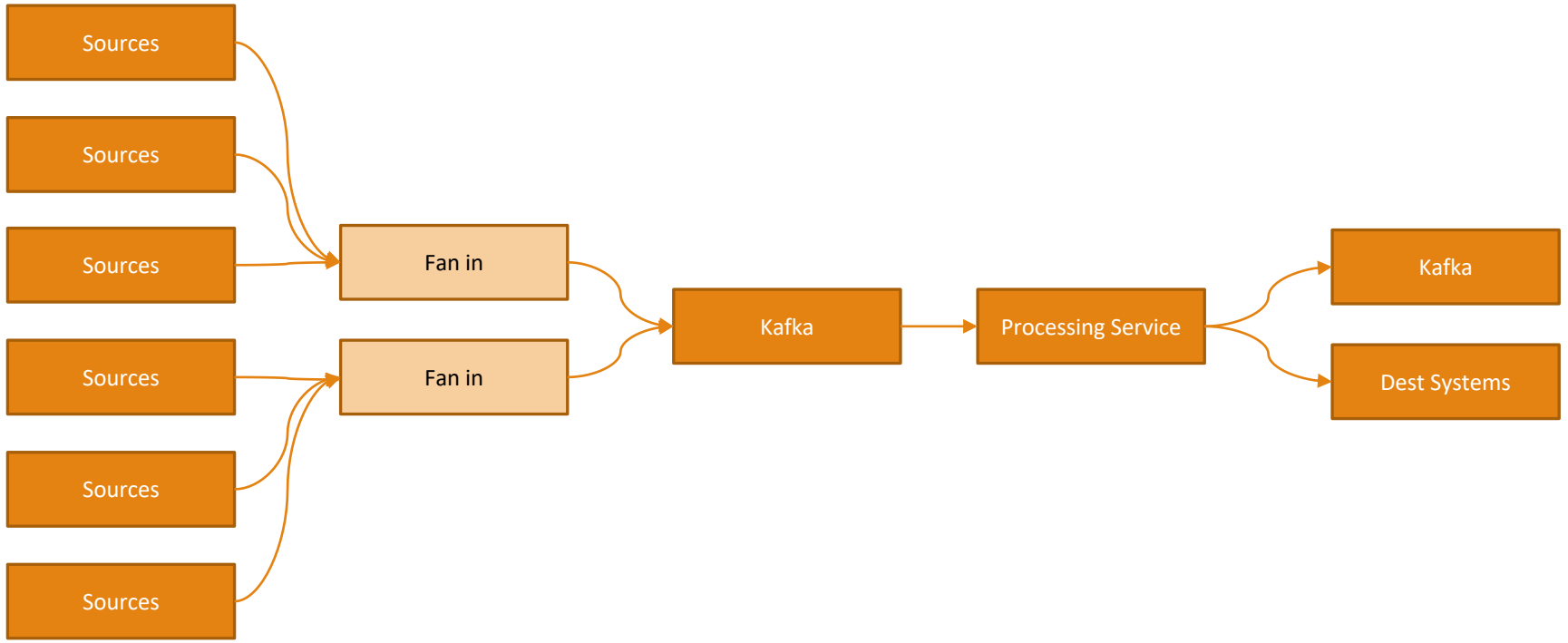
Components of a Streaming Architecture

- Source
- Pipes
- Processing
- Ingestion

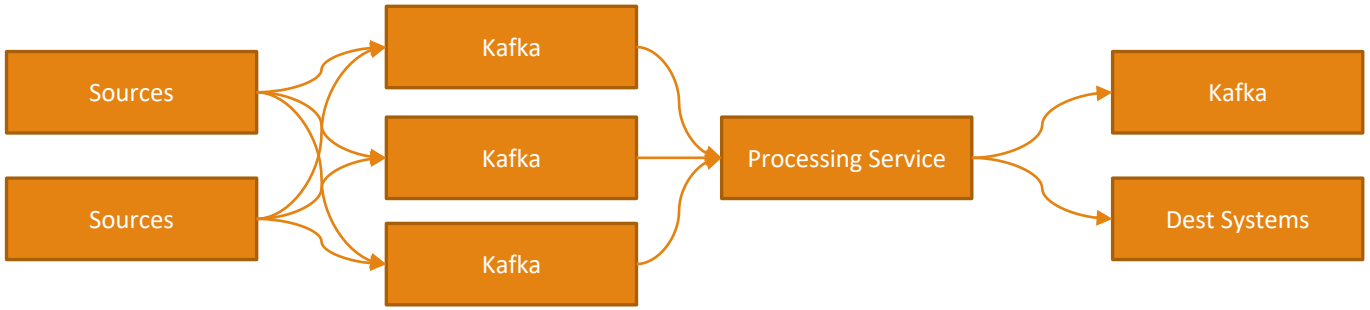
Basic Parts


Sources
Sources
Kafka
Processing Service
Kafka
Dest Systems

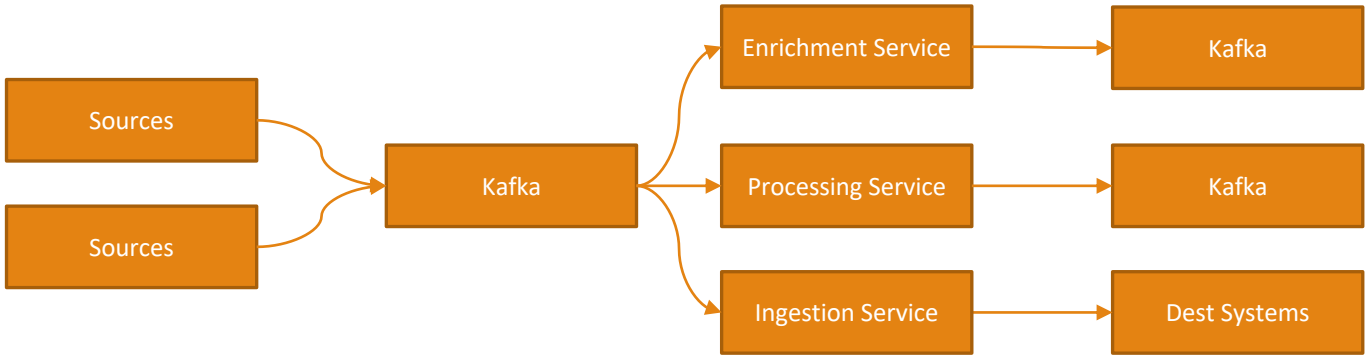
Advanced Parts (Fan In)



Advanced Parts (Multiple Kafkas)



Advanced Parts (Multiple Consumer Groups)



Advanced Parts (RPC)