Kafka

Viet-Trung Tran

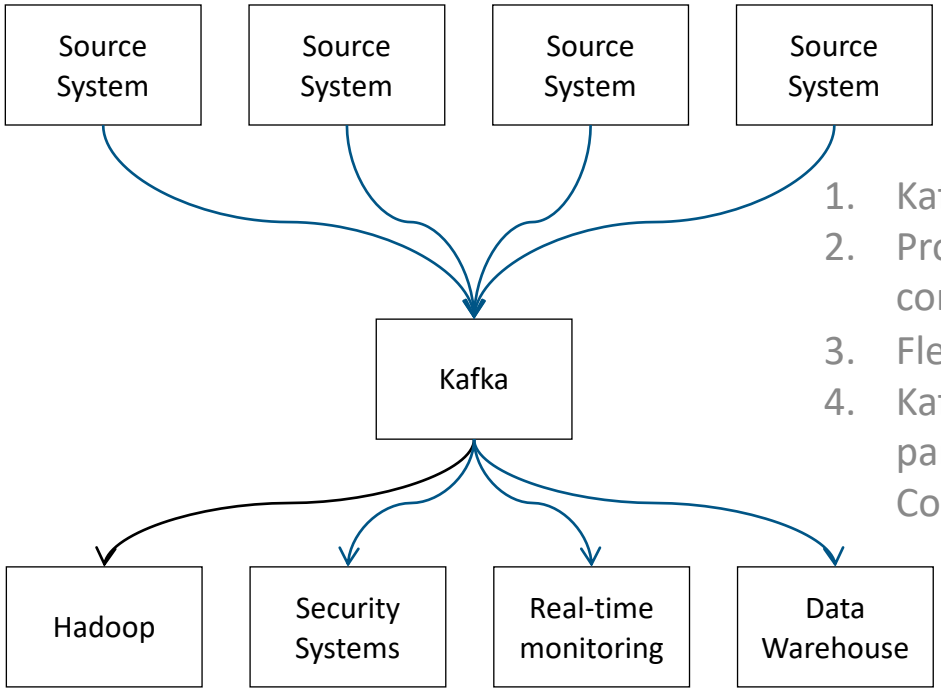
School of Information and Communication Technology

Why Kafka

Producers

Brokers

Consumers



1. Kafka decouple data streams
2. Producers don't know about consumers
3. Flexible message consumption
4. Kafka broker delegates log partition offset (location) to Consumers (clients)

Kafka decouples Data Pipelines

What is Kafka?

- Apache Kafka is a fast, scalable, durable, and fault-tolerant publish-subscribe messaging system
  - Publish and Subscribe to streams of records
  - Fault tolerant storage
    - Replicates Topic Log Partitions to multiple servers
  - Process records as they occur
  - Fast, efficient IO, batching, compression, and more
- Used to decouple data streams
- Kafka is often used instead of JMS, RabbitMQ and AMQP
  - higher throughput, reliability and replication

Kafka possibility

- Build real-time streaming applications that react to streams
  - Feeding data to do real-time analytic systems
  - Transform, react, aggregate, join real-time data flows (eg. Metrics gathering)
  - Feed events to CEP for complex event processing
  - Feeding of high-latency daily or hourly data analysis into Spark, Hadoop, etc.
    - (eg. External commit log for distributed systems. Replicated data between nodes, re-sync for nodes to restore state)
  - Up to date dashboards and summaries
- Build real-time streaming data pipe-lines
  - Enable in-memory microservices (actors, Akka, Vert.x, Qbit, RxJava)

Kafka adoption

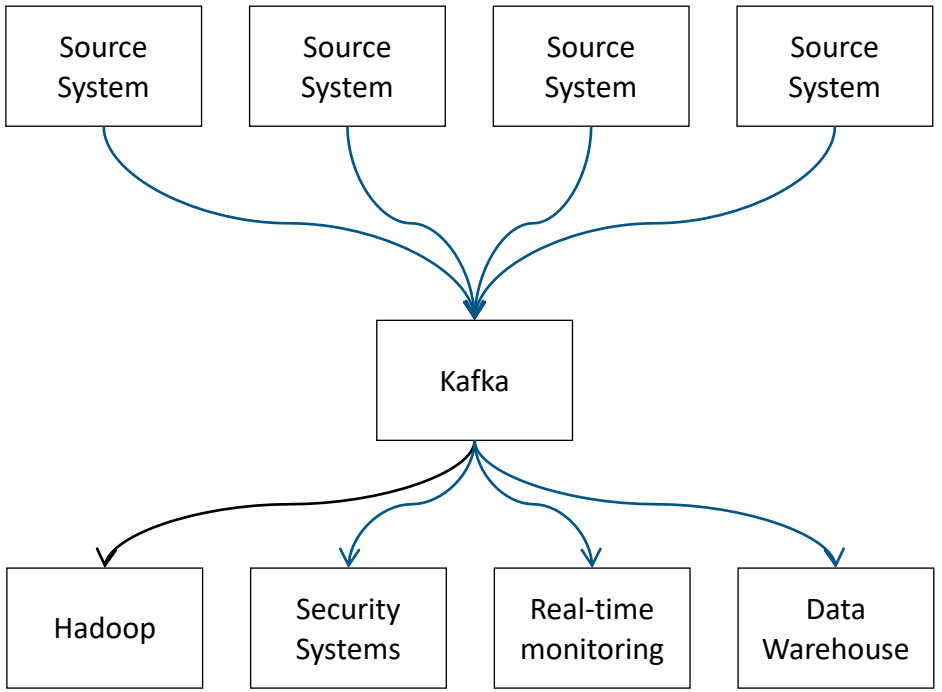
- 1/3 of all Fortune 500 companies
- Top ten travel companies, 7 of ten top banks, 8 of ten top insurance companies, 9 of ten top telecom companies
- LinkedIn, Microsoft and Netflix process 1 billion messages a day with Kafka
- Real-time streams of data, used to collect big data or to do real time analysis (or both)

Why is Kafka popular?

- Great performance
- Operational simplicity, easy to setup and use, easy to reason
- Stable, reliable durability,
- Flexible publish-subscribe/queue (scales with N-number of consumer groups),
- Robust replication,
- Producer tunable consistency guarantees,
- Ordering preserved at shard level (topic partition)
- Works well with systems that have data streams to process, aggregate, transform & load into other stores

Concepts

Basic Kafka Concepts

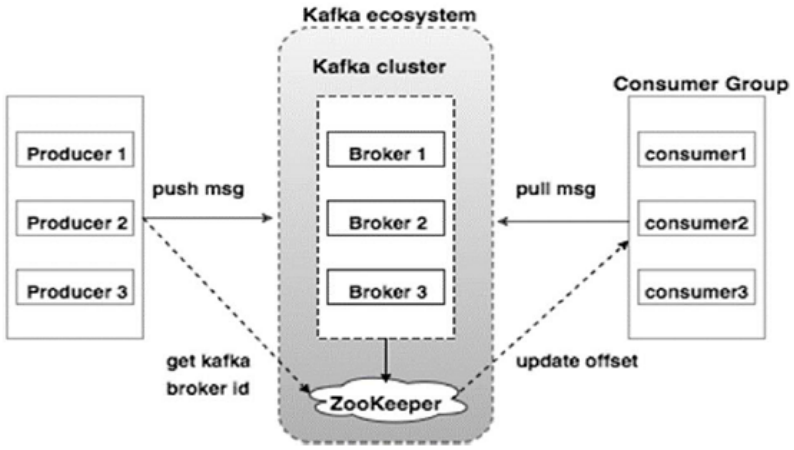


Key terminology

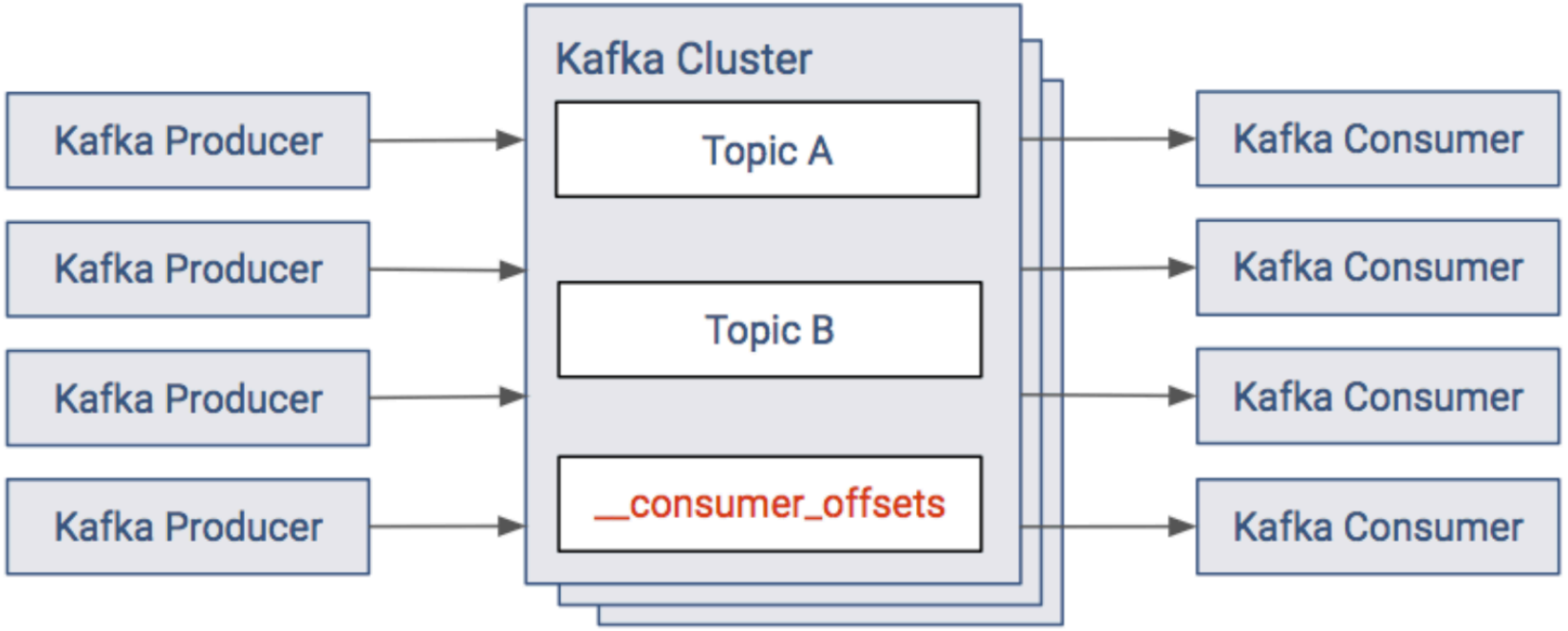
- Kafka maintains feeds of messages in categories called topics.
  - a stream of records (“/orders”, “/user-signups”), feed name
  - Log topic storage on disk
  - Partition / Segments (parts of Topic Log)
- Records have a key (optional), value and timestamp; Immutable
- Processes that publish messages to a Kafka topic are called **producers**.
- Processes that subscribe to topics and process the feed of published messages are called **consumers**.
- Kafka is run as a cluster comprised of one or more servers each of which is called a **broker**.

Kafka architecture

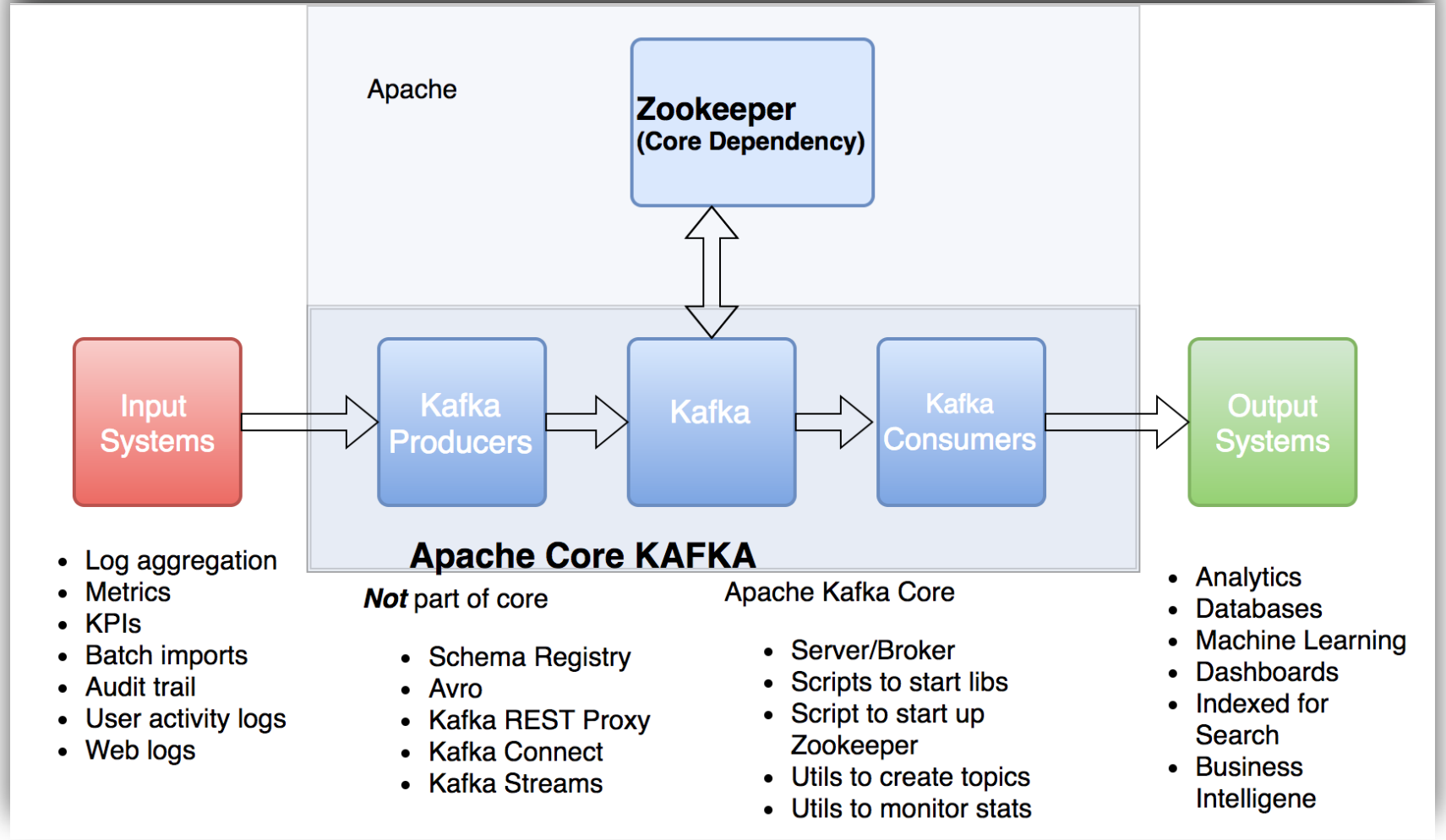
- Kafka cluster consists of mutliple brokers and zookeeper
- Communication between all components is done via a high performance simple binary API over TCP protocol
- Zookeeper provides in-sync view of Kafka Cluster configuration
  - Leadership election of Kafka Broker and Topic Partition pairs
  - manages service discovery for Kafka Brokers that form the cluster
- Zookeeper sends changes to Kafka
  - New Broker join, Broker died, etc.
  - Topic removed, Topic added, etc.



Topics, producers, and consumers



Apache Kafka



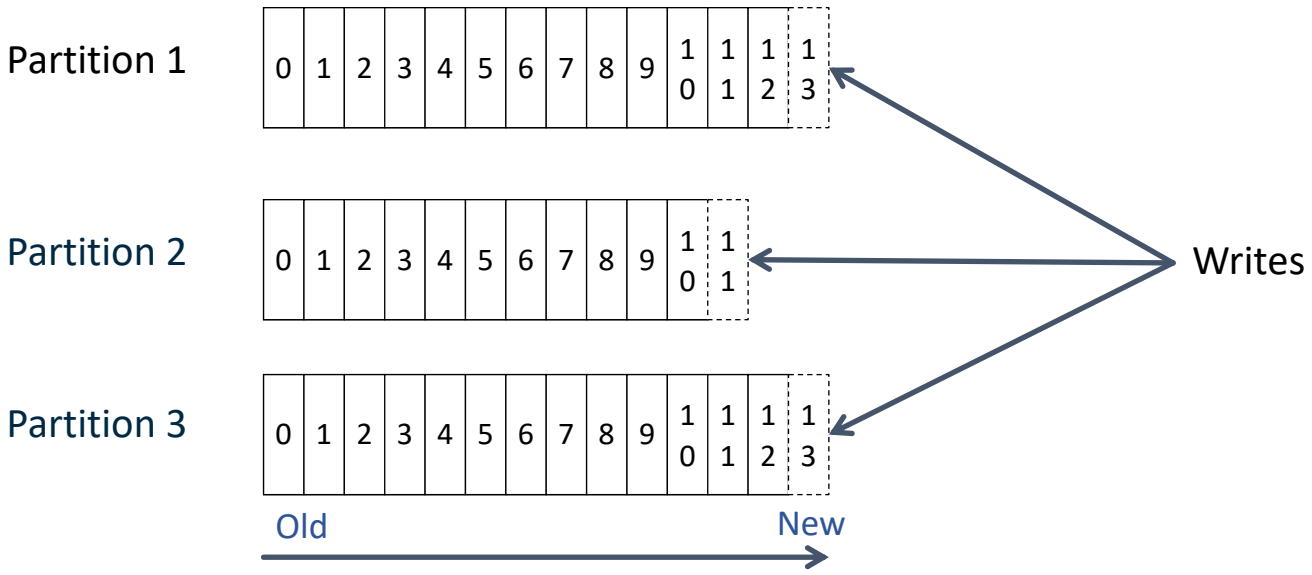
Kafka topics architecture

Kafka topics, logs, partitions

- Kafka topic is a stream of records
- Topics stored in log
- Topic is a category or stream name or feed
- Topics are pub/sub
  - Can have zero or many subscribers - consumer groups

Topic partitions

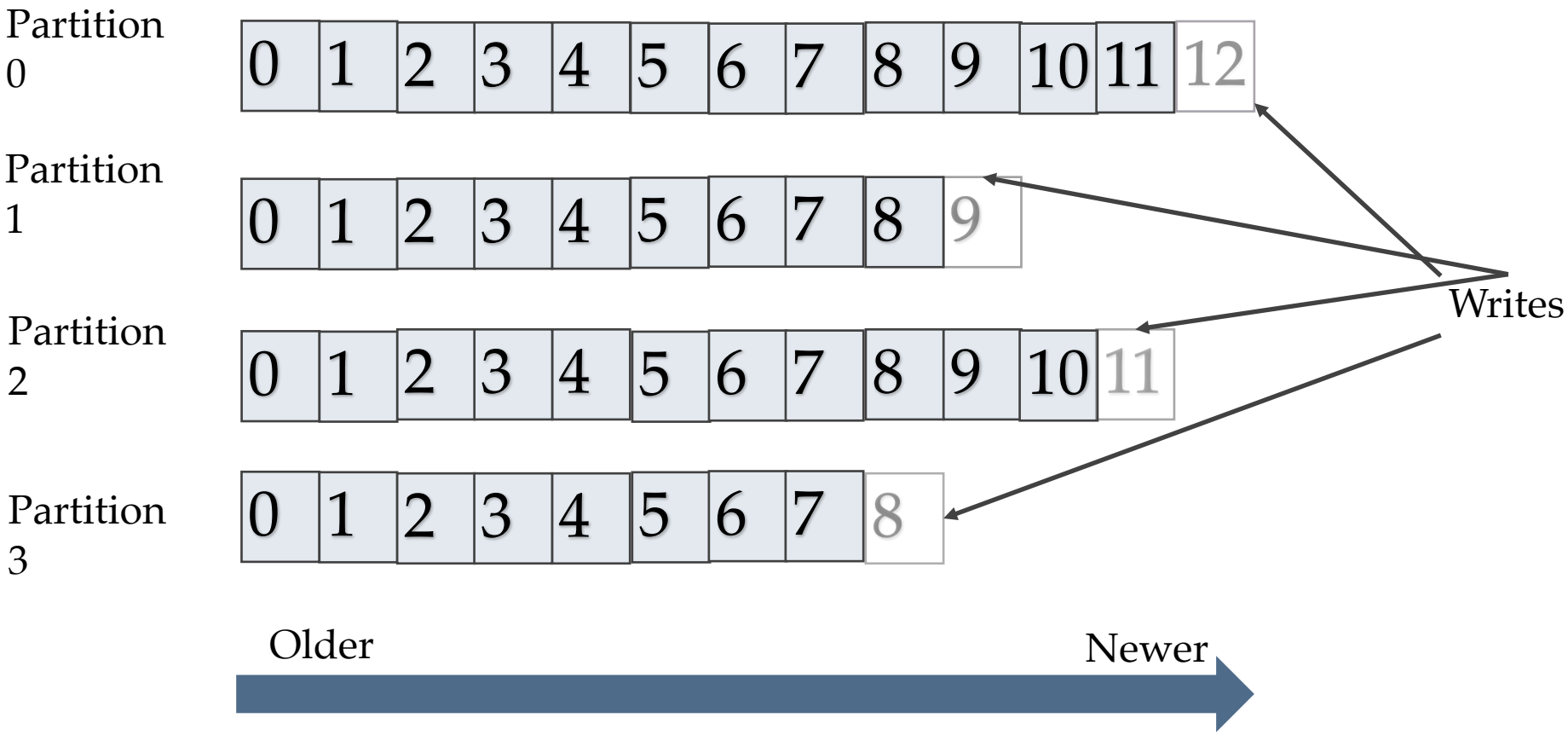
- Topics are broken up into partitions, decided usually by key of record
- Partitions are used to scale Kafka across many servers
  - Record sent to correct partition by key
- Partitions can be replicated to multiple brokers



Topic partition log

- Order is maintained only in a single partition
  - Partition is ordered, immutable sequence of records that is continually appended to—a structured commit log
- Records in partitions are assigned sequential id number called the offset

Kafka topic partitions layout

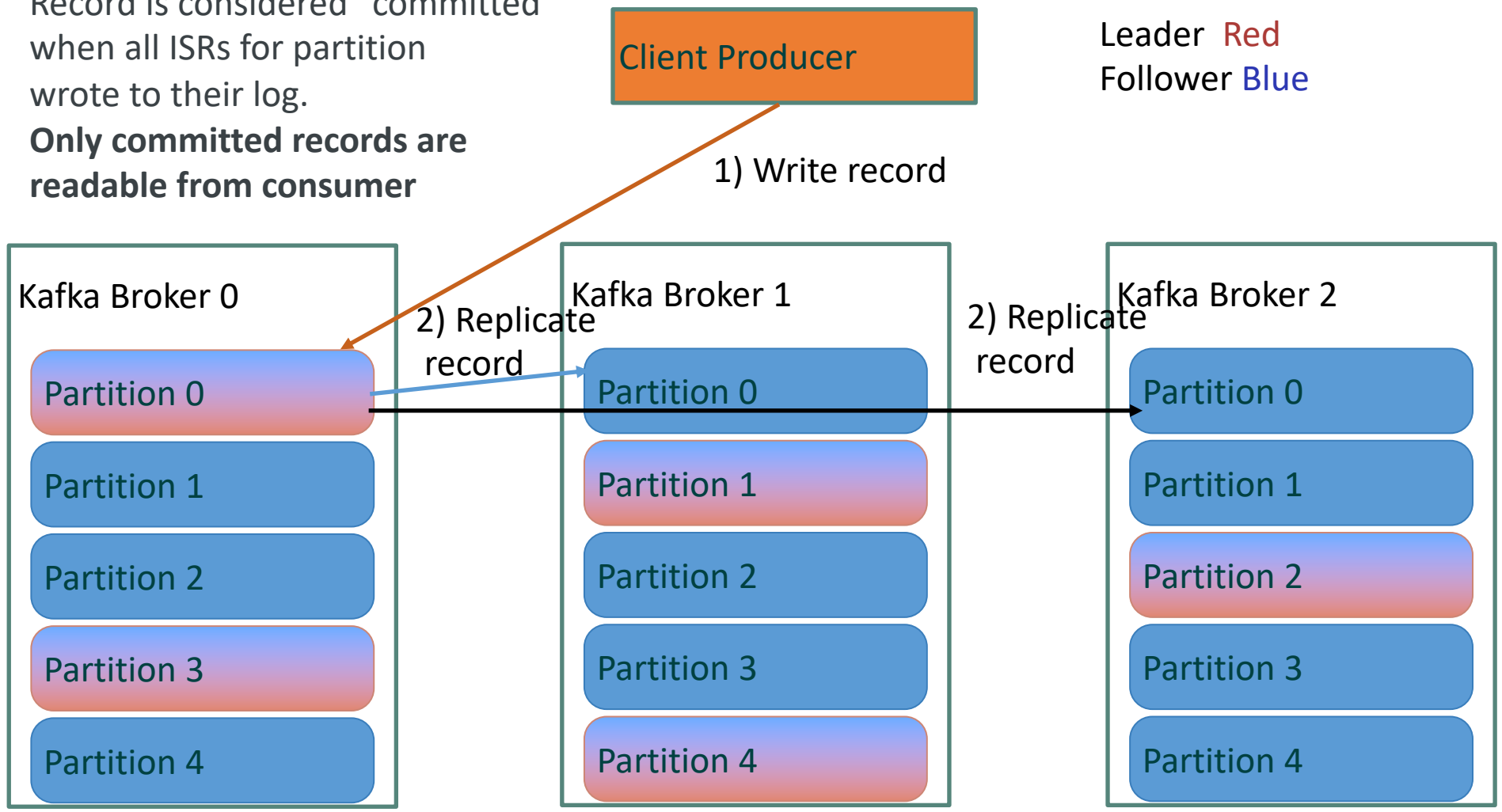


Kafka partition replication

- Each partition has leader server and zero or more follower servers
  - Leader handles all read and write requests for partition
  - Followers replicate leader
  - An follower that is in-sync is called an ISR (in-sync replica)
  - If a partition leader fails, one ISR is chosen as new leader
- Partitions of log are distributed over the servers in the Kafka cluster with each server handling data and requests for a share of partitions
- Each partition can be replicated across a configurable number of Kafka servers
  - Used for fault tolerance

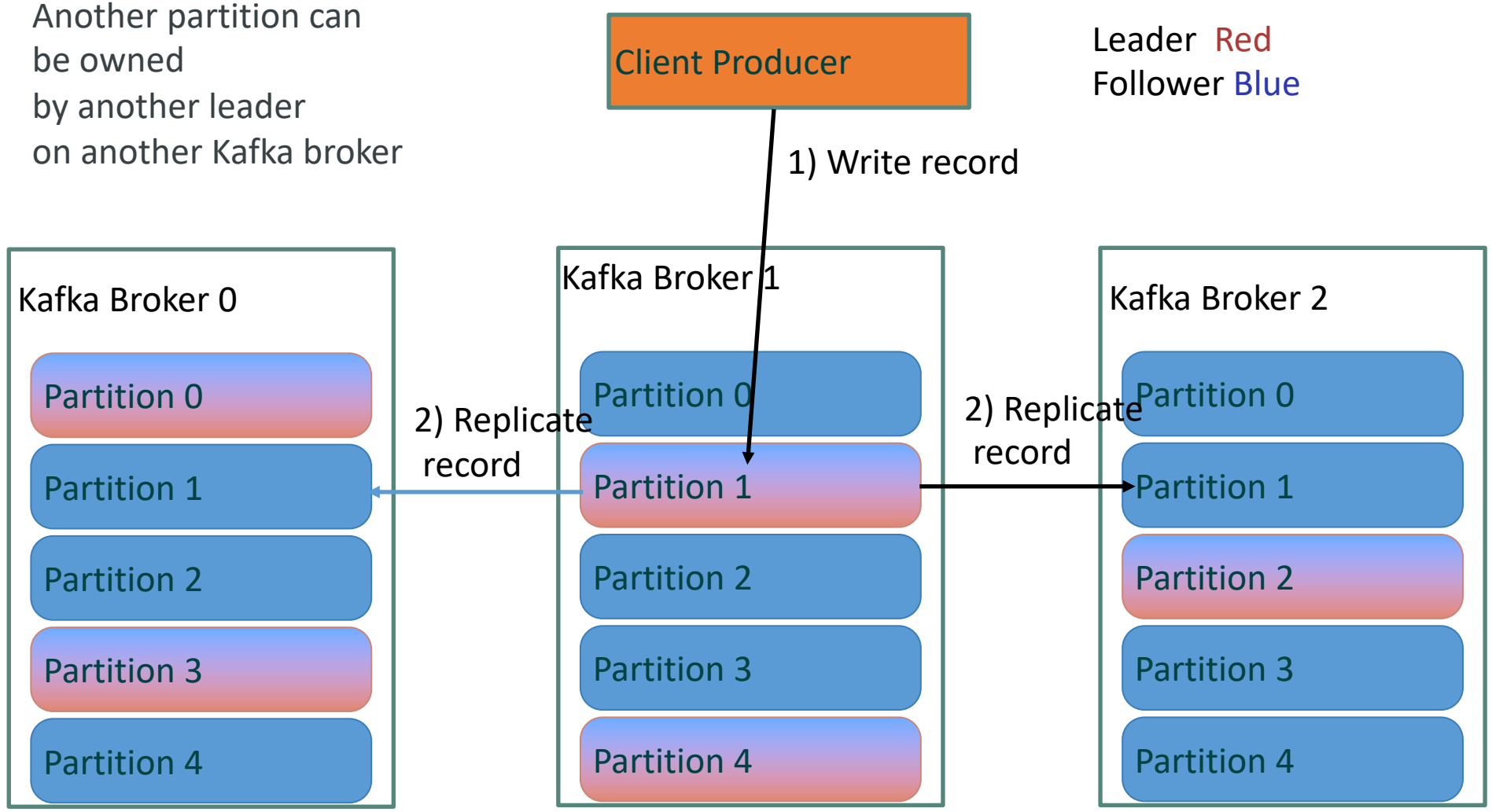
Kafka replication to partition (1)

Record is considered "committed" when all ISRs for partition wrote to their log.
Only committed records are readable from consumer



Kafka replication to partitions (2)

Another partition can
be owned
by another leader
on another Kafka broker

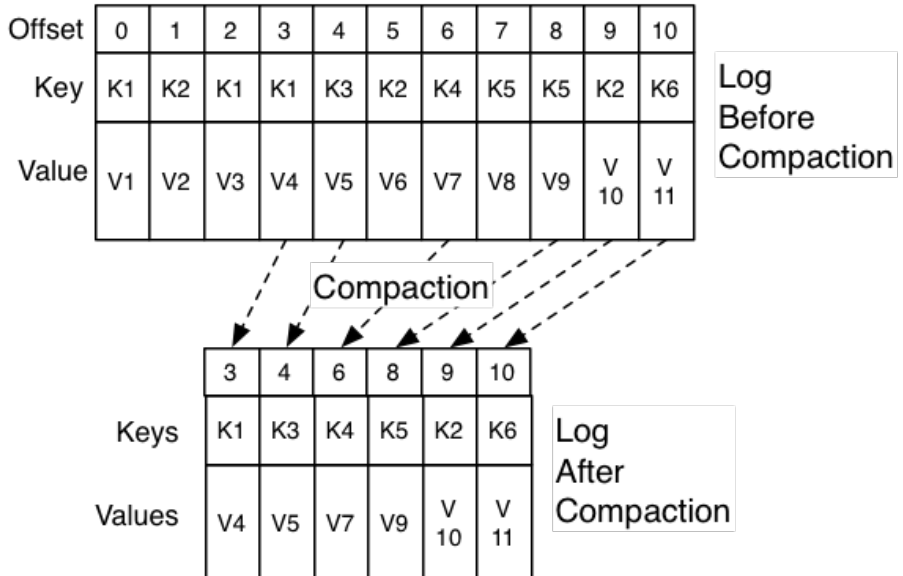


Guarantees

- Messages sent by a producer to a particular topic partition will be appended in the order they are sent
- Minimum available ISR can also be configured such that an error is returned if enough replicas are not available to replicate data
- A consumer instance sees messages in the order they are stored in the log
- For a topic with replication factor N, Kafka can tolerate up to N-1 server failures without “losing” any messages committed to the log

Kafka record retention

- Kafka cluster retains all published records
  - Time based – configurable retention period
  - Size based - configurable based on size
  - Compaction - keeps latest record
- Retention policy of three days or two weeks or a month
- It is available for consumption until discarded by time, size or compaction
- Consumption speed not impacted by size



Durable writes

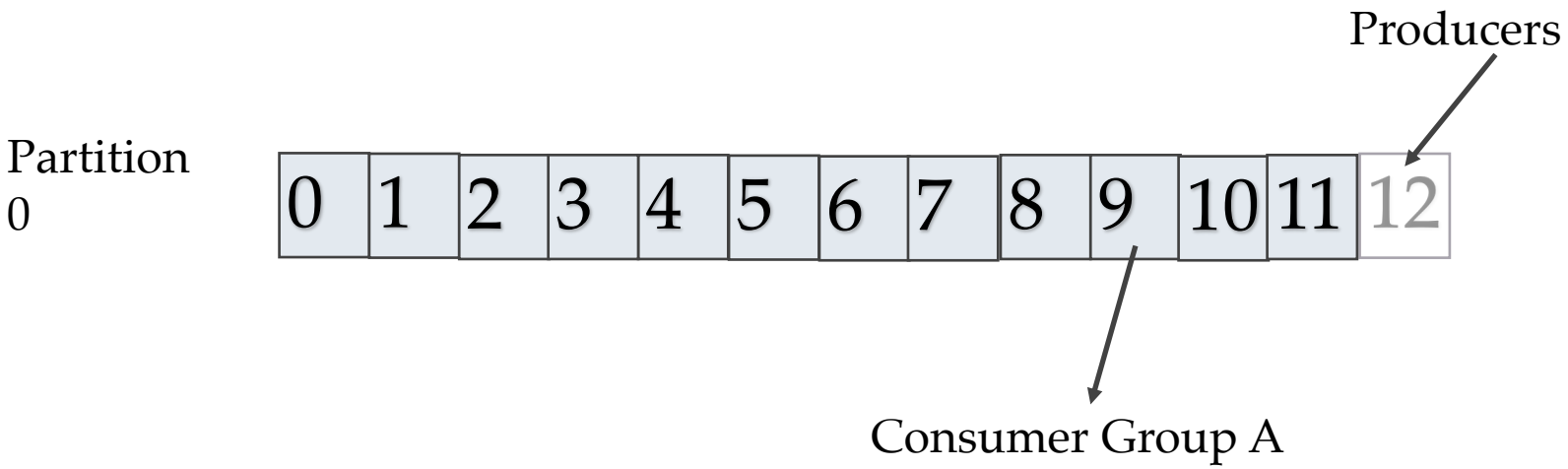
- Producers can choose to trade throughput for durability of writes:
- Note: throughput can also be raised with more brokers…

Durability	Behaviour	Per Event Latency	Required Acknowledgements (request.required.acks)
Highest	ACK all ISRs have received	Highest	-1
Medium	ACK once the leader has received	Medium	1
Lowest	No ACKs required	Lowest	0

Producers

- Producers publish to a topic of their choosing (push)
  - Producer(s) append Records at end of Topic log
- Load can be distributed in number of partitions
  - Typically by “round-robin”
  - Can also do “semantic partitioning” based on a key in the message
    - Example have all the events of a certain 'employeeId' go to same partition
  - Important: Producer picks partition
- All nodes can answer metadata requests about
  - Which servers are alive
  - Where leaders are for the partitions of a topic

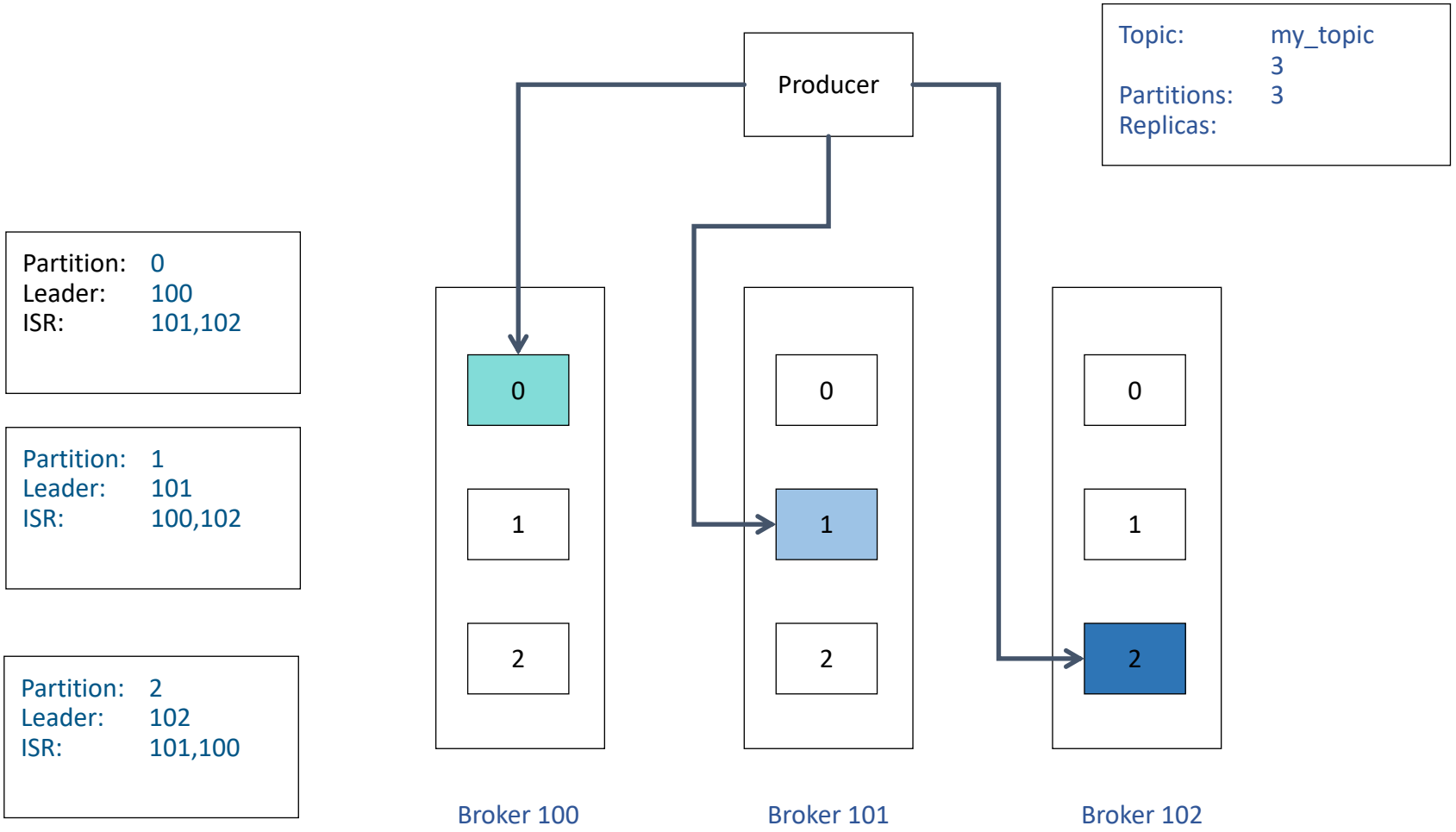
Kafka producers and consumers



Producers are writing at Offset 12

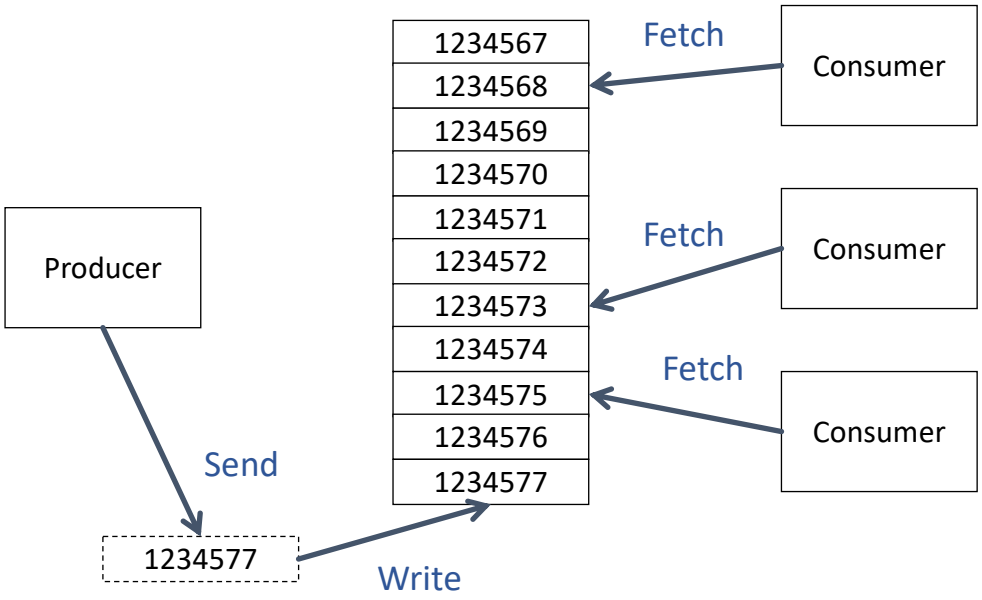
Consumer Group A is Reading from Offset 9.

Producer – Load balancing and ISRs



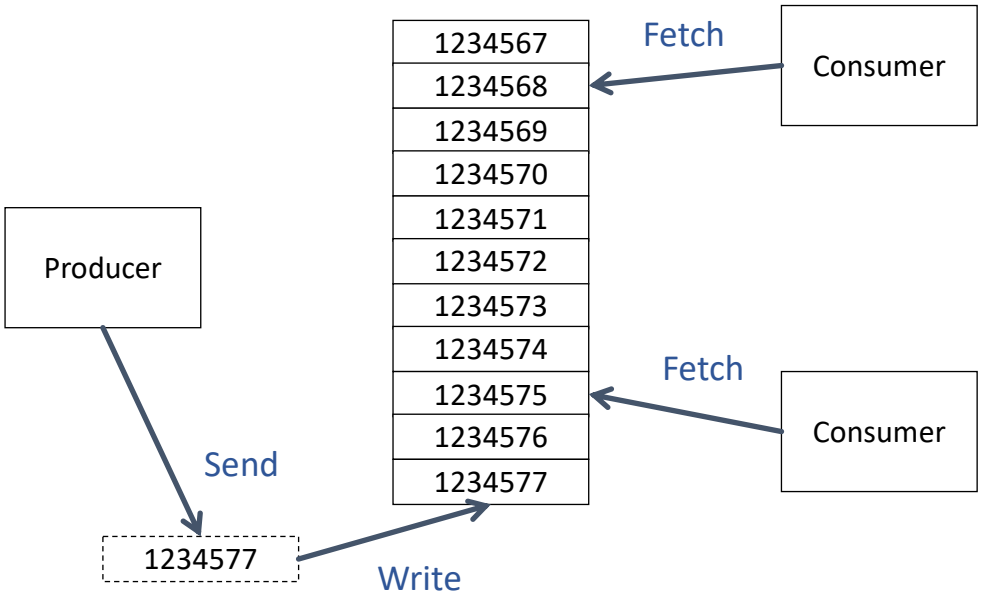
Consumer (1)

- Multiple Consumers can read from the same topic
- Each Consumer is responsible for managing it's own offset
- Messages stay on Kafka…they are not removed after they are consumed



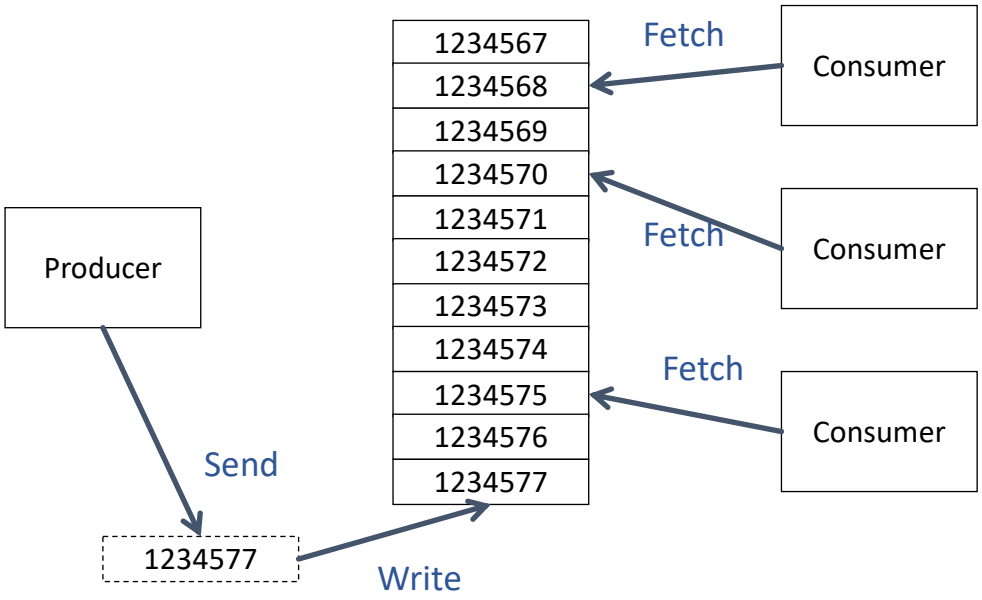
Consumer (2)

- Consumers can go away



Consumer (3)

- And then come back



Consumer Group

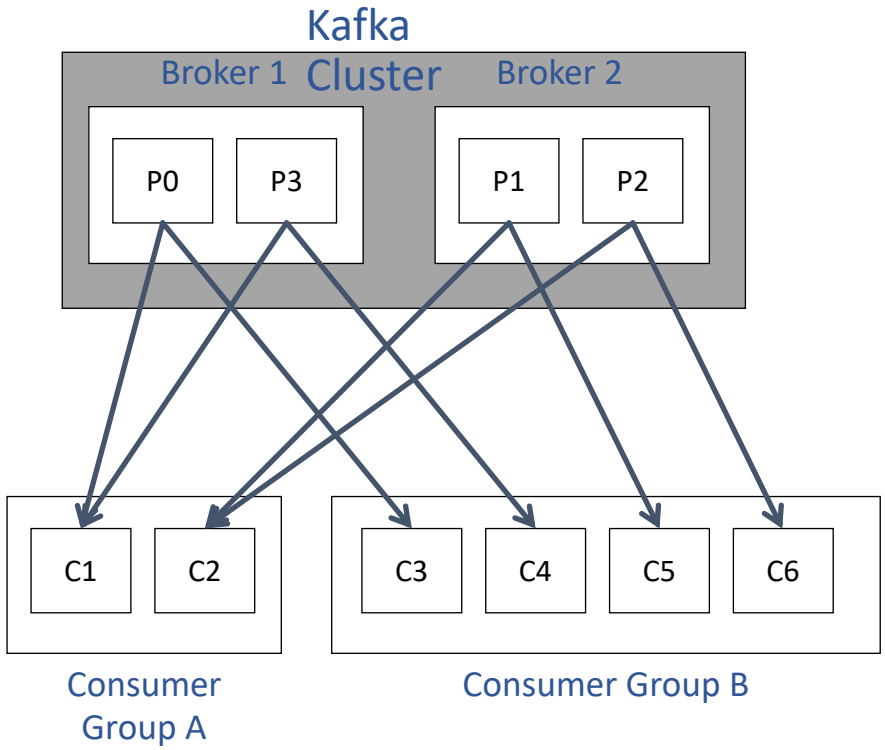
- Consumers are grouped into a Consumer Group
  - Consumer group has a unique id
  - Each consumer group is a subscriber
  - Each consumer group maintains its own offset
  - Multiple subscribers = multiple consumer groups
  - Each has different function: one might delivering records to microservices while another is streaming records to Hadoop
- A record is delivered to one Consumer in a Consumer Group
- Each consumer in consumer groups takes records and only one consumer in group gets same record
- Consumers in Consumer Group load balance record consumption

Common consumer group patterns

- All consumer instances in one group
  - Acts like a traditional queue with load balancing
- All consumer instances in different groups
  - All messages are broadcast to all consumer instances
- “Logical Subscriber” – Many consumer instances in a group
  - Consumers are added for scalability and fault tolerance
  - Each consumer instance reads from one or more partitions for a topic
  - There cannot be more consumer instances than partitions

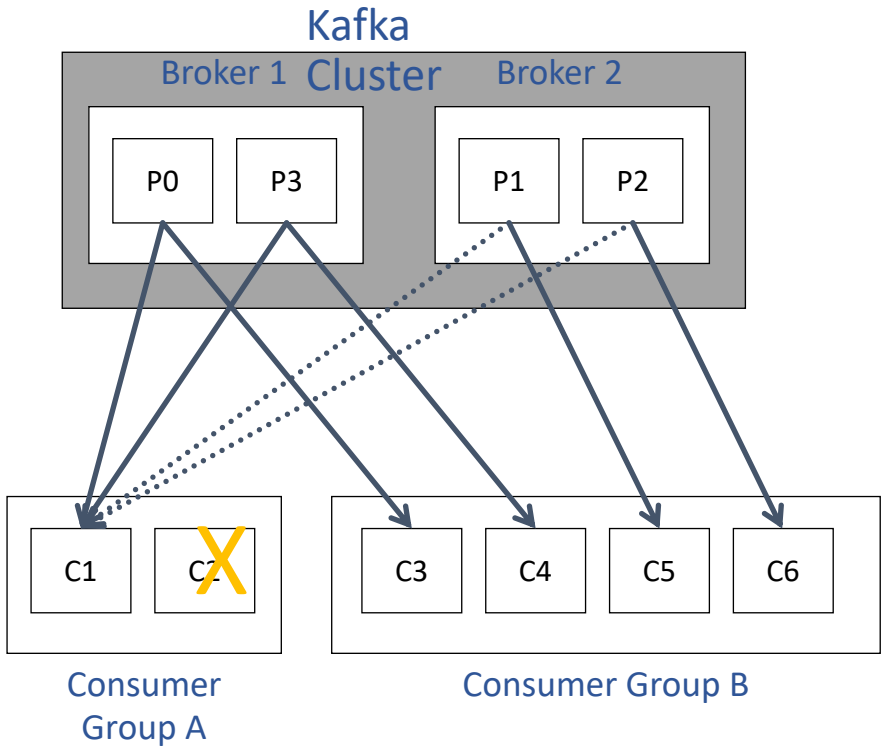
Consumer - Groups

Consumer Groups provide isolation to topics and partitions



Consumer - Groups

Can rebalance themselves



Kafka consumer load share

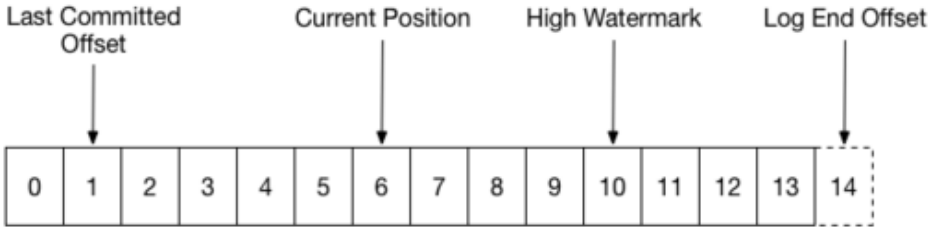
- Consumer membership in Consumer Group is handled by the Kafka protocol dynamically
- If new Consumers join Consumer group, it gets a share of partitions
- If Consumer dies, its partitions are split among remaining live Consumers in Consumer Group

Kafka consumer failover

- Consumers notify broker when it successfully processed a record
  - advances offset (“__consumer_offset”)
- If Consumer fails before sending commit offset to Kafka broker,
  - different Consumer can continue from the last committed offset
  - some Kafka records could be reprocessed
  - at least once behavior
  - messages should be idempotent

What can be consumed

- "Log end offset" is offset of last record written to log partition and where Producers write to next
- "High watermark" is offset of last record successfully replicated to all partitions followers
- Consumer only reads up to “high watermark”. Consumer can’t read un-replicated data



Consumer to partition cardinality

- Only a single Consumer from the same Consumer Group can access a single Partition
- If Consumer Group count exceeds Partition count:
  - Extra Consumers remain idle; can be used for failover
- If more Partitions than Consumer Group instances,
  - Some Consumers will read from more than one partition

Kafka brokers

- Kafka Cluster is made up of multiple Kafka Brokers
- Each Broker has an ID (number)
- Brokers contain topic log partitions
- Connecting to one broker bootstraps client to entire cluster
- Start with at least three brokers, cluster can have, 10, 100, 1000 brokers if needed

Kafka scale and speed

- How can Kafka scale if multiple producers and consumers read/write to same Kafka Topic log?
- Writes fast: Sequential writes to filesystem are fast (700 MB or more a second)
- Scales writes and reads by sharding:
  - Topic logs into Partitions (parts of a Topic log)
  - Topics logs can be split into multiple Partitions different machines/different disks
  - Multiple Producers can write to different Partitions of the same Topic
  - Multiple Consumers Groups can read from different partitions efficiently

Kafka scale and speed (2): high throughput and low latency

- Batching of individual messages to amortize network overhead and append/consume chunks together
  - end to end from Producer to file system to Consumer
  - Provides More efficient data compression. Reduces I/O latency
- Zero copy I/O using sendfile (Java's NIO FileChannel transferTo method).
  - Implements linux sendfile() system call which skips unnecessary copies
  - Heavily relies on Linux PageCache
    - The I/O scheduler will batch together consecutive small writes into bigger physical writes which improves throughput.
    - The I/O scheduler will attempt to re-sequence writes to minimize movement of the disk head which improves throughput.
    - It automatically uses all the free memory on the machine

Delivery semantics

Default

- At least once
  - Messages are never lost but may be redelivered
- At most once
  - Messages are lost but never redelivered
- Exactly once
  - Messages are delivered once and only once

Delivery semantics

- At least once
  - Messages are never lost but may be redelivered
- At most once
  - Messages are lost but never redelivered
- Exactly once
  - Messages are delivered once and only once

Much Harder
(Impossible??)

Getting exactly once semantics

- Must consider two components
  - Durability guarantees when publishing a message
  - Durability guarantees when consuming a message
- Producer
  - What happens when a produce request was sent but a network error returned before an ack?
  - Use a single writer per partition and check the latest committed value after network errors
- Consumer
  - Include a unique ID (e.g. UUID) and de-duplicate.
  - Consider storing offsets with data

https://dzone.com/articles/interpreting-kafkas-exactly-once-semantics

Kafka positioning

- For really large file transfers
  - Probably not, it's designed for "messages" not really for files. If you need to ship large files, consider good-ole-file transfer, or breaking up the files and reading per line to move to Kafka.
- As a replacement for MQ/Rabbit/Tibco
  - Probably. Performance Numbers are drastically superior. Also gives the ability for transient consumers. Handles failures pretty well.
- If security on the broker and across the wire is important?
  - Not right now. We can't really enforce much in the way of security. (KAFKA-1682)
- To do transformations of data
  - Not really by itself