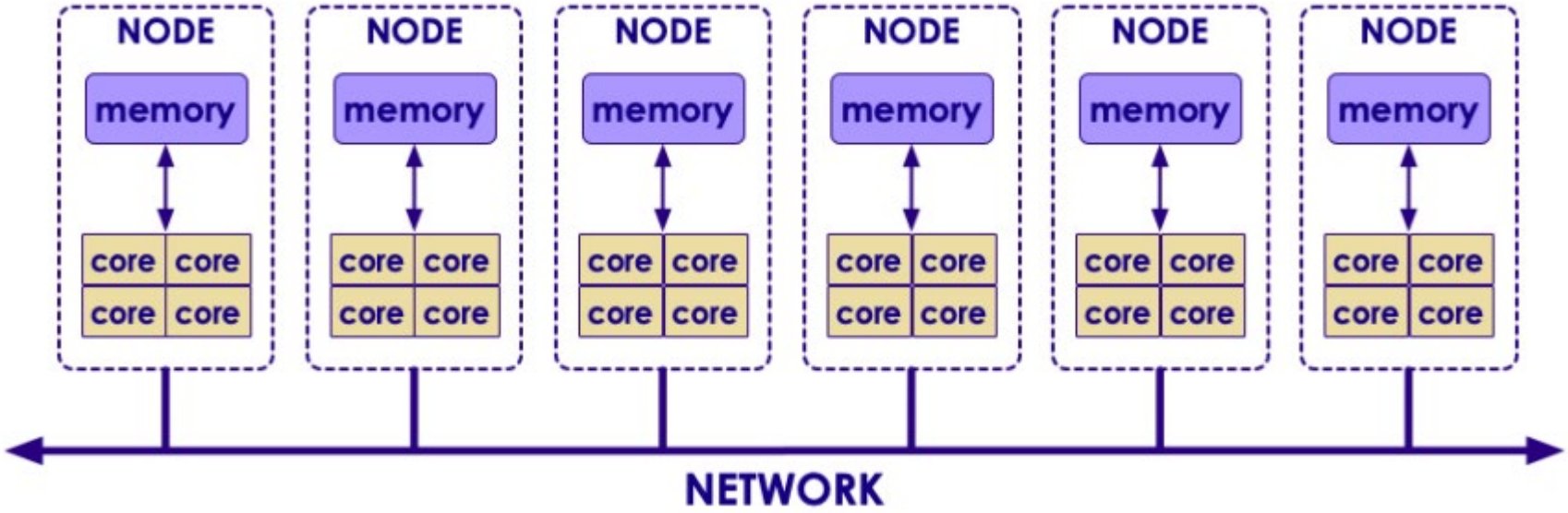
Presents

Kafka Architecture

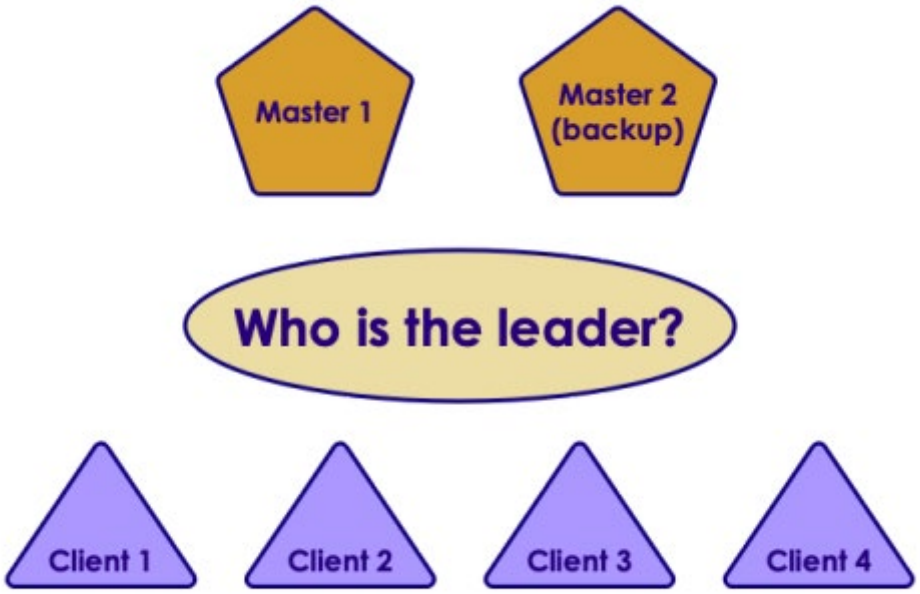
The Distributed Problem

- Distributed systems with
  - ✓ Multiple nodes
  - ✓ Each with multiple cores
  - ✓ How do we co-ordinate them all?

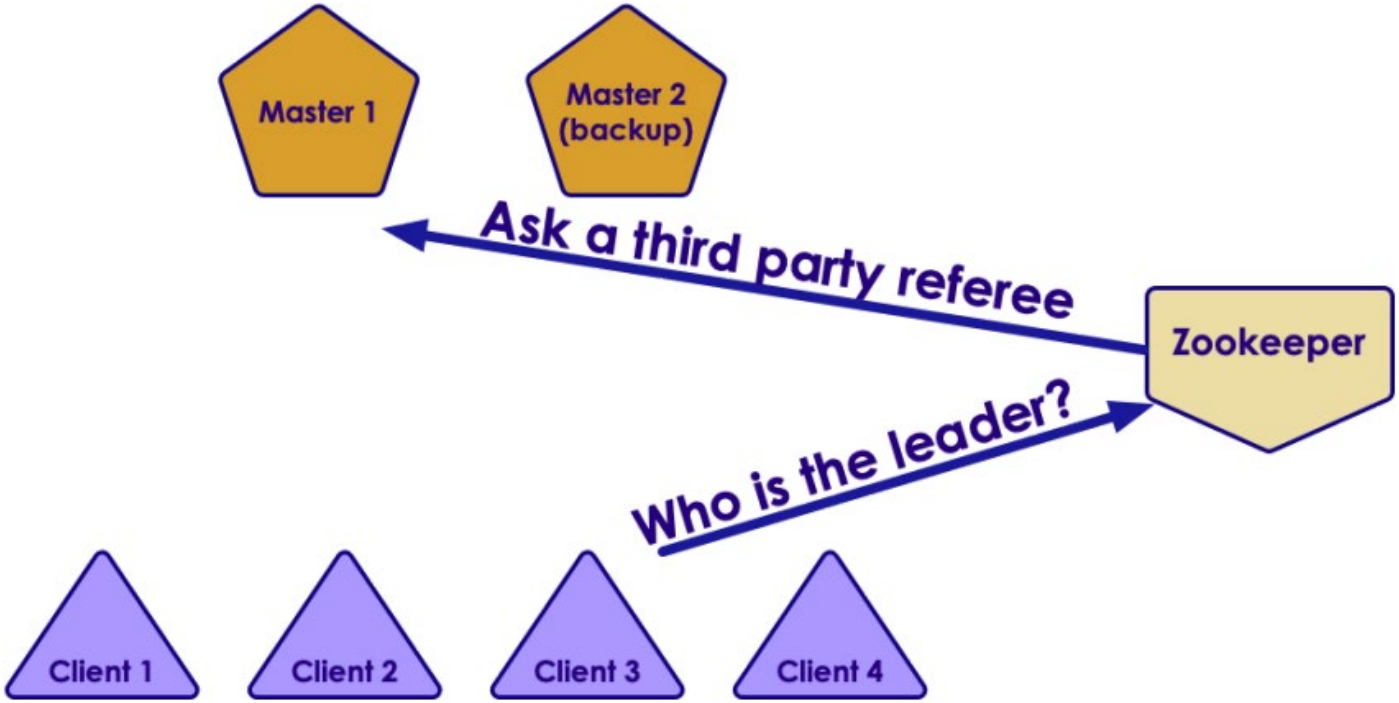


LearnQuest

Leader Election

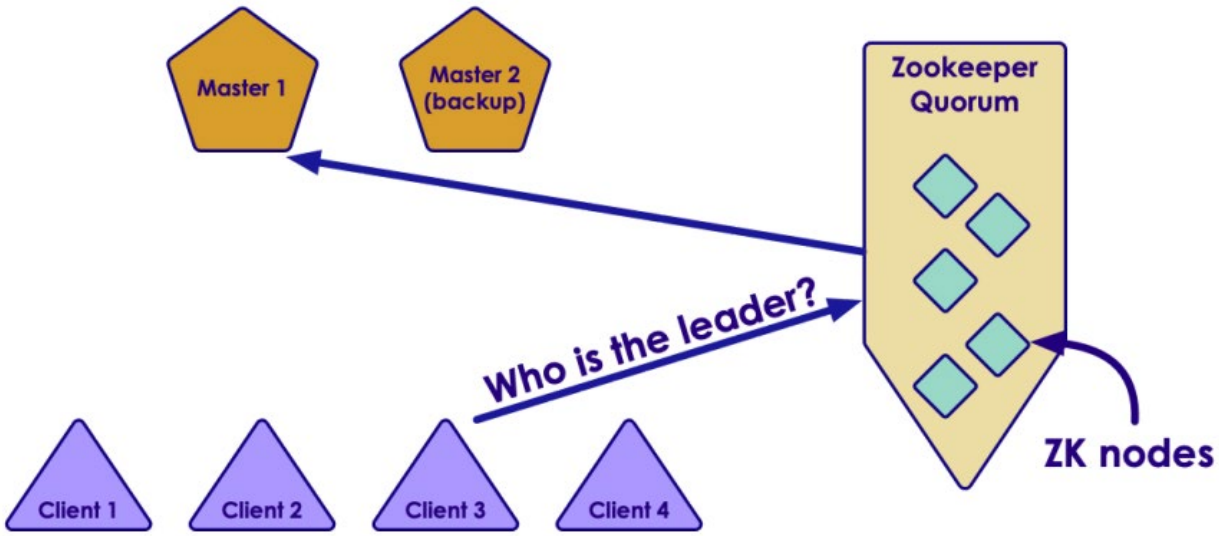


Leader Election With Zookeeper



Zookeeper Cluster / Quorum

- What if ZK goes down?
  - ✓ Run ZK as a cluster - quorum
  - ✓ No single point of failure

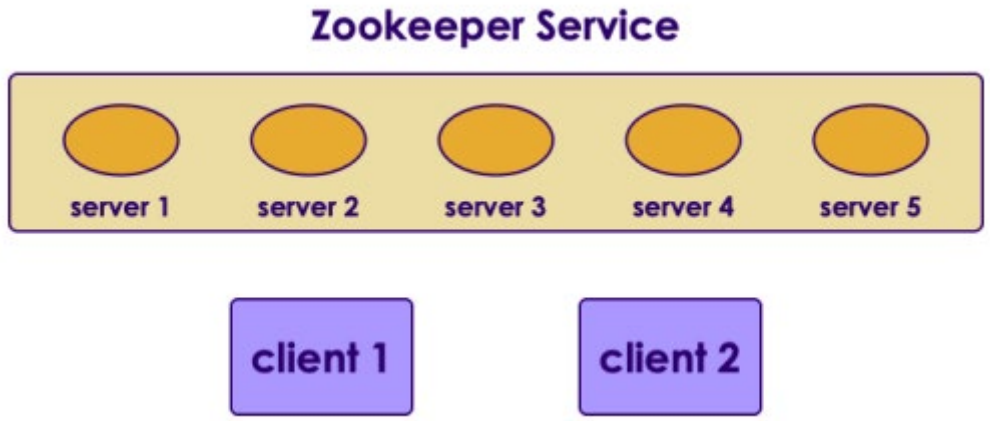


Zookeeper

- Distributed service that provides
  - ✓ Configuration
  - ✓ Synchronization
  - ✓ Name registry
  - ✓ Consensus
  - ✓ Leader election
- Open source
  - ✓ Apache open source project
  - ✓ Battle tested with very large distributed projects
    - Hadoop, HBase, Kafka

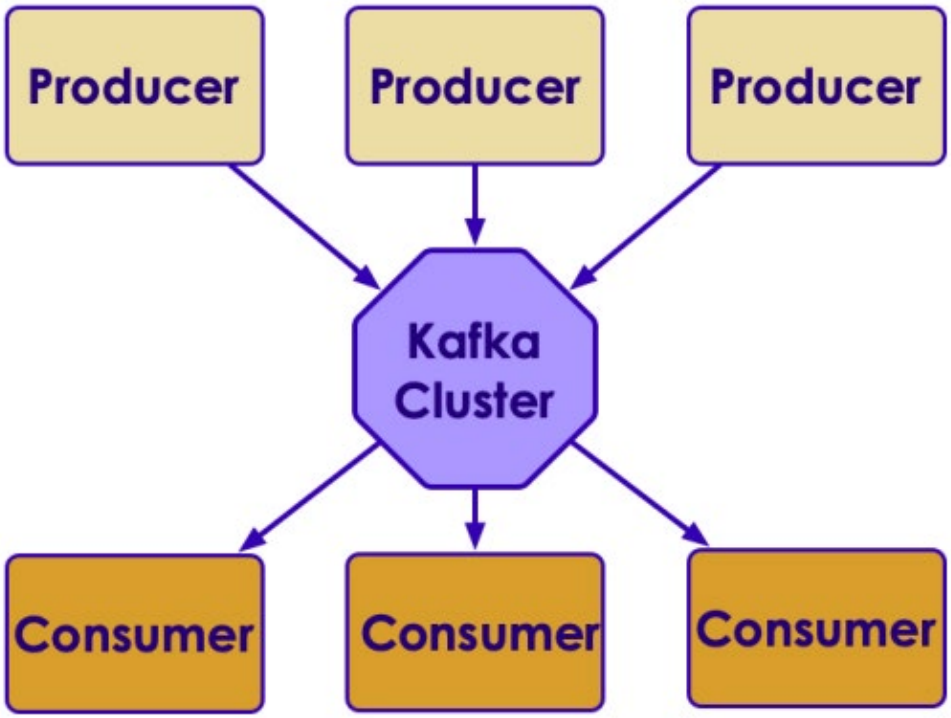
Zookeeper

- Runs as a quorum (multiple nodes)
  - ✓ No single point of failure
- Odd number of nodes (3, 5, 7 ...etc)
  - ✓ Odd number to break tie when voting
  - ✓ Minimum 3 nodes
- Small number of nodes can support thousands of clients



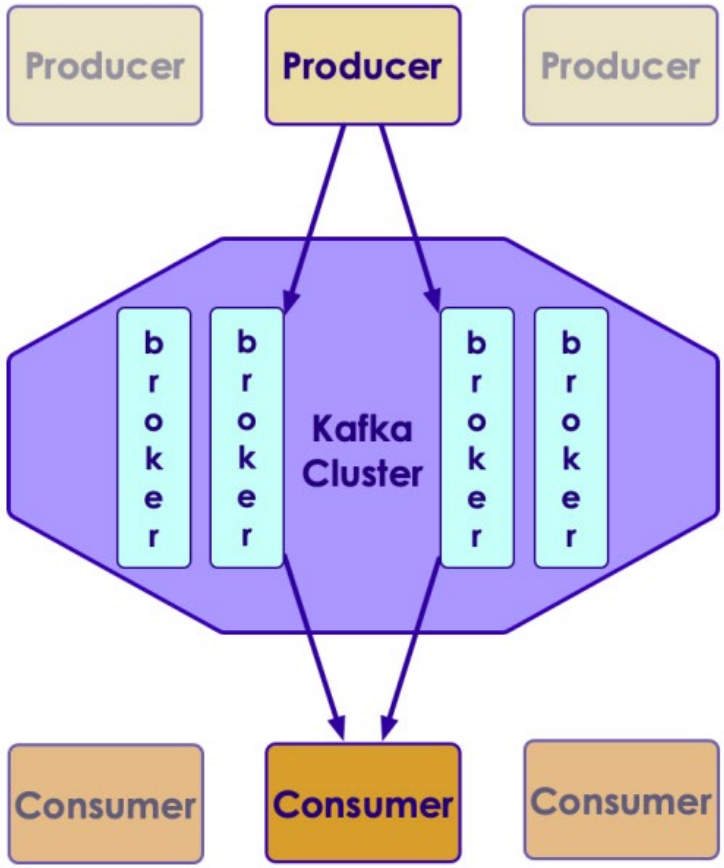
Kafka Architecture

- Kafka is designed as a Pub-Sub messaging system
  - ✓ Producers publish messages
  - ✓ Consumers consume messages



Kafka Architecture

- Kafka is designed to run on many nodes as a cluster
  - ✓ Kafka machines are called 'brokers'
  - ✓ Kafka automatically backs up data on at least another machine (broker)



Kafka Terminology

- Roles
  - ✓ Producers: write data to Kafka
  - ✓ Consumers: read data from Kafka
  - ✓ Brokers: Kafka nodes
  - ✓ Zookeeper: Keep track of brokers
- Data
  - ✓ Message: 'basic unit' of data in Kafka
  - ✓ Topics: Messages are organized as topics
  - ✓ Partitions: Topics are split into partitions
  - ✓ Commit Log: How data is organized
  - ✓ Offset: message's position within a partition

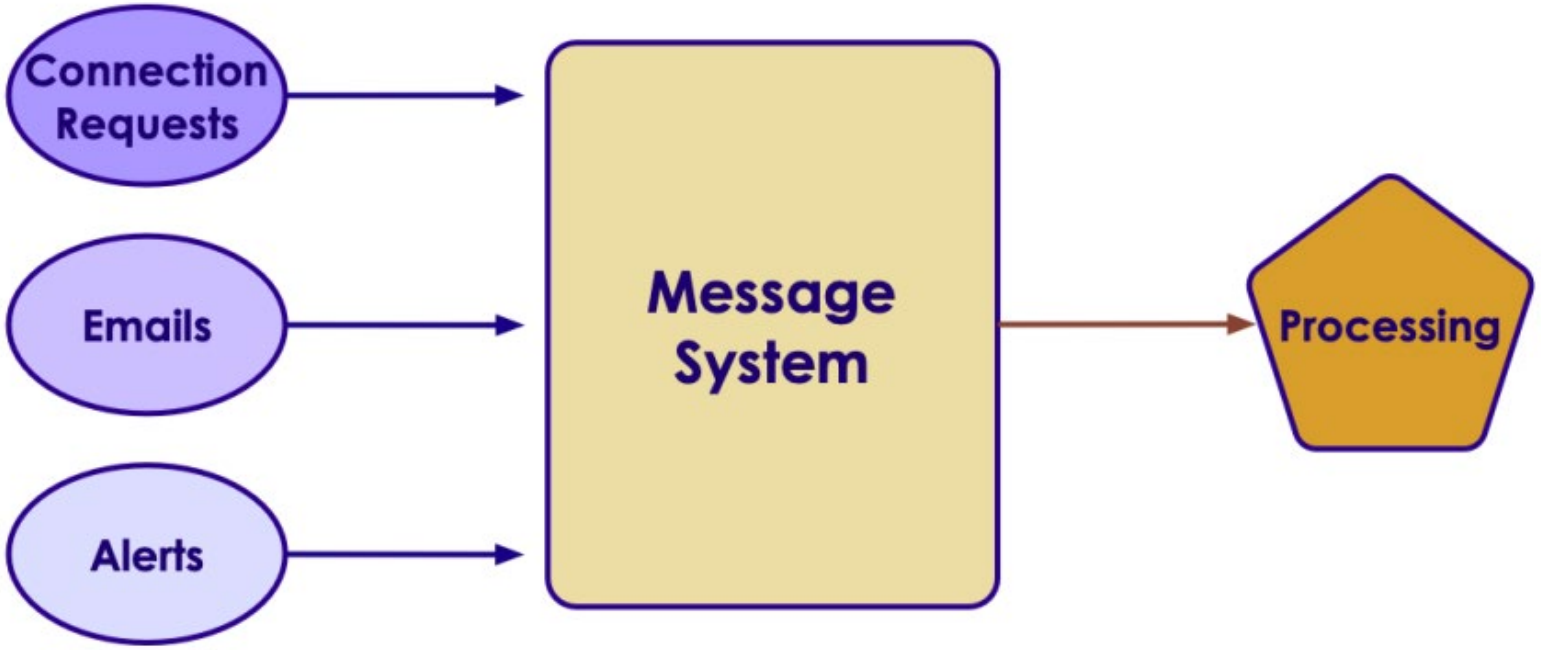
A Kafka Use Case: 'My Connect'

- Features
  - ✓ Users can connect with each other
  - ✓ Users can send messages to each other
  - ✓ Analyze user's usage pattern to customize home page
  - ✓ System metrics and diagnostics
- Design
  - ✓ We will use a message queue instead of database
  - ✓ We are going to send messages for each event
  - ✓ Each user email is sent as a message
  - ✓ System metrics are sent as events

A Kafka Use Case: 'My Connect'

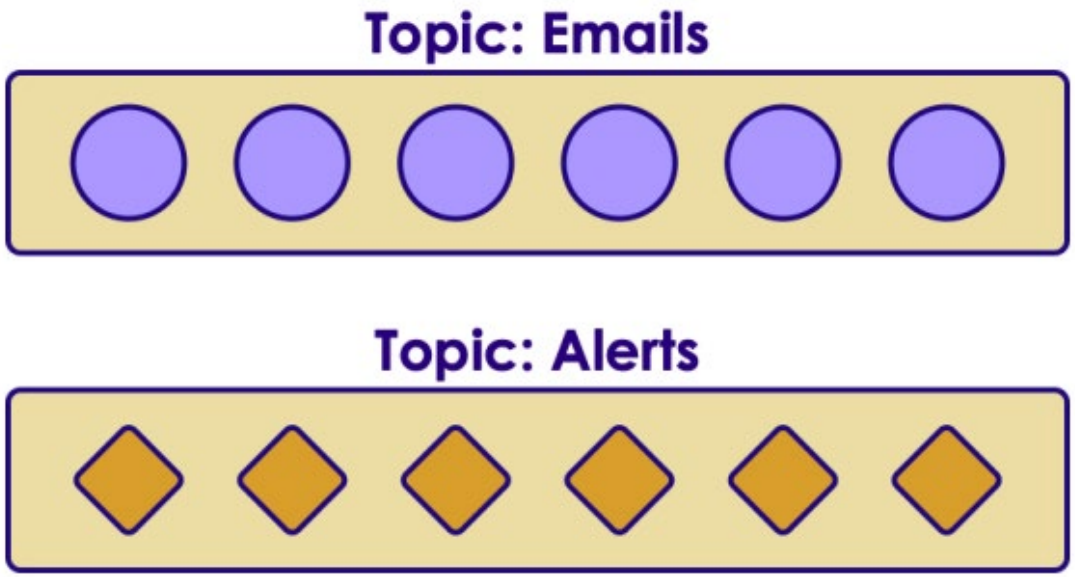
- Features
  - ✓ Users can connect with each other
  - ✓ Users can send messages to each other
  - ✓ Analyze user's usage pattern to customize home page
  - ✓ System metrics and diagnostics
- Design
  - ✓ We will use a message queue instead of database
  - ✓ We are going to send messages for each event
  - ✓ Each user email is sent as a message
  - ✓ System metrics are sent as events

'My Connect' Design



Kafka Concepts

- In Kafka a basic unit of data is a 'message'
  - ✓ Message can be email / connection request / alert event
- Messages are stored in 'topics'
  - ✓ Topics are like 'queues'
  - ✓ Sample topics could be: emails / alerts

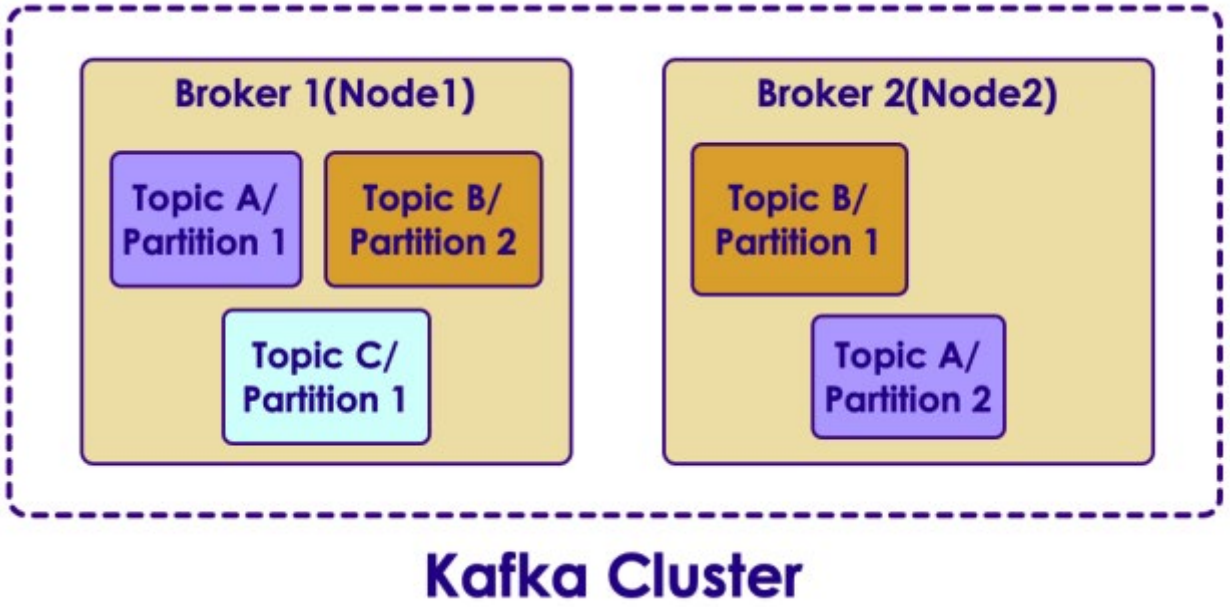


Topics

- Analogous to a 'queue' in a queuing system
  - ✓ Logical / virtual entity
  - ✓ We can set expiration-times & replication settings per topic
  - ✓ Topics are broken into smaller units called partitions

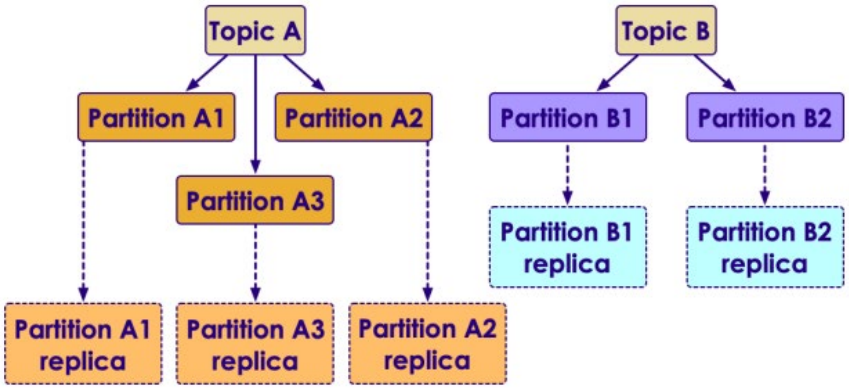
Partitions

- Partition is a physical entity
  - ✓ This is where data lives
  - ✓ One partition resides on ONE machine (1 to 1)
  - ✓ One machine will host many partitions (N <-> M)
    - Possibly from many topics



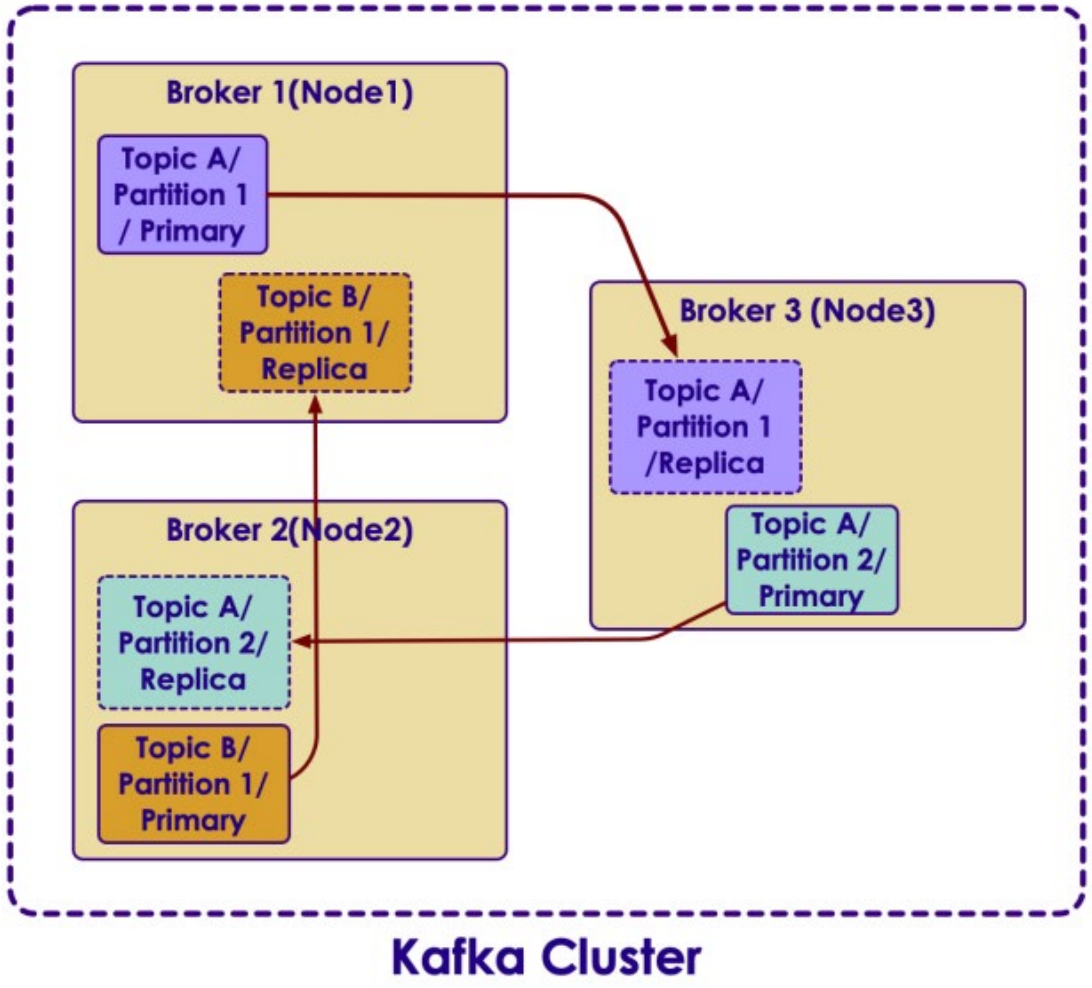
Partitions / Replicas

- One partition is stored in one machine (broker)
  - ✓ Partitions are replicated to prevent data loss, in case a machine crashes
  - ✓ Default setup is 2 copies (one primary, one replica)
  - ✓ One broker is the 'owner' for a partition
  - ✓ Replicas are purely there to prevent data loss
  - ✓ Replicas are never written to, nor read from so increasing number of replicas does not increase throughput



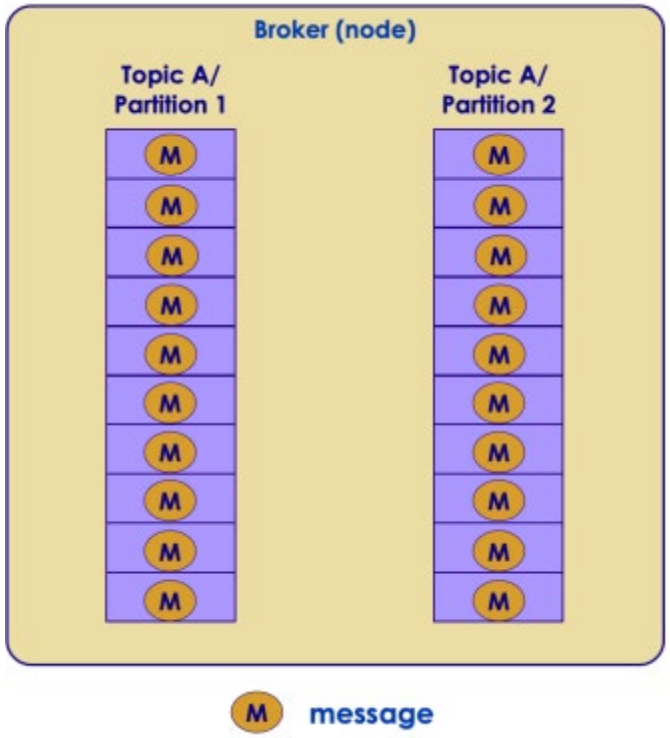
LearnQuest

Topics + Partitions + Replicas



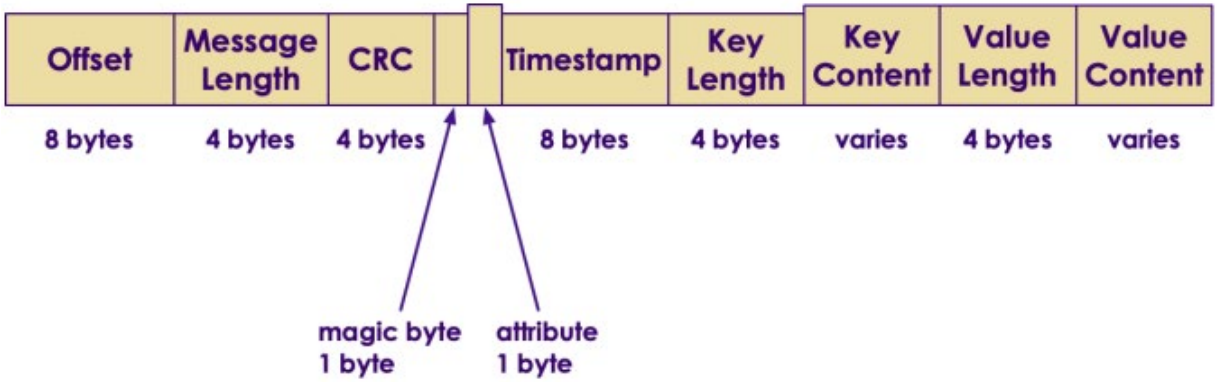
Commit Log

- Commit Log is simple file on disk that stores message bytes
  - ✓ Messages are always appended (to the end) of commit log
  - ✓ Commit log can not be modified in the middle (immutable)
  - ✓ Can read messages in order
  - ✓ Provides high concurrency & high throughput with no locking
  - ✓ Each Partition has it's own commit log



Kafka Message

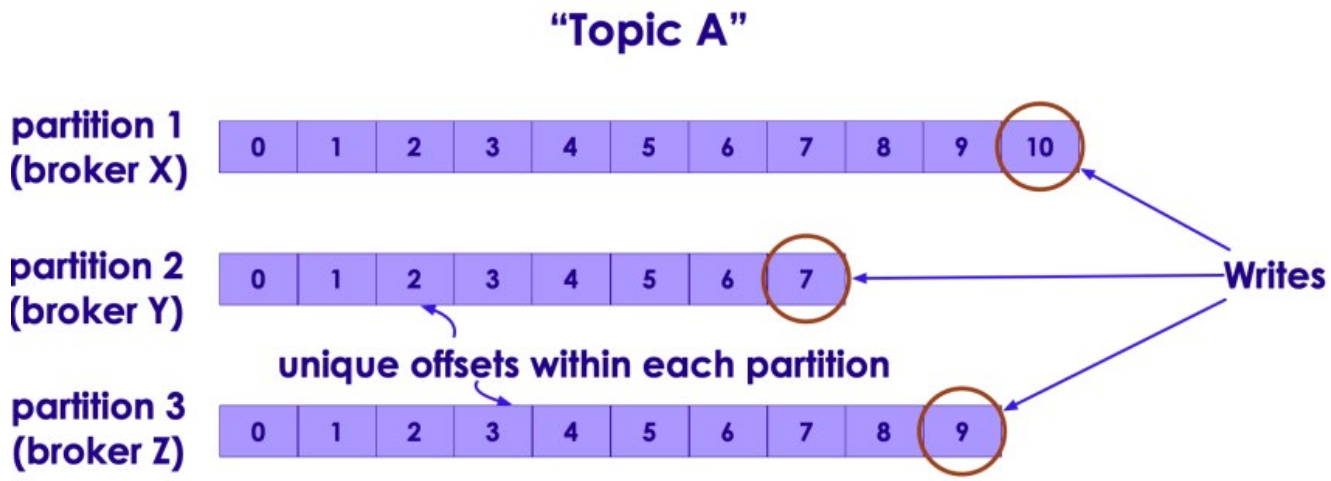
- In Kafka basic 'data unit' is a message
  - ✓ Kafka treats messages as 'bunch of bytes'
  - ✓ Doesn't really care what the message payload is
- Optionally messages can have metadata, like keys
  - ✓ Keys are bytes too
  - ✓ Keys are used to determine which partition to write to
  - ✓ Think 'hashing', Same key always go to same
- Messages can have optional schema



LearnQuest

Partitions / Messages

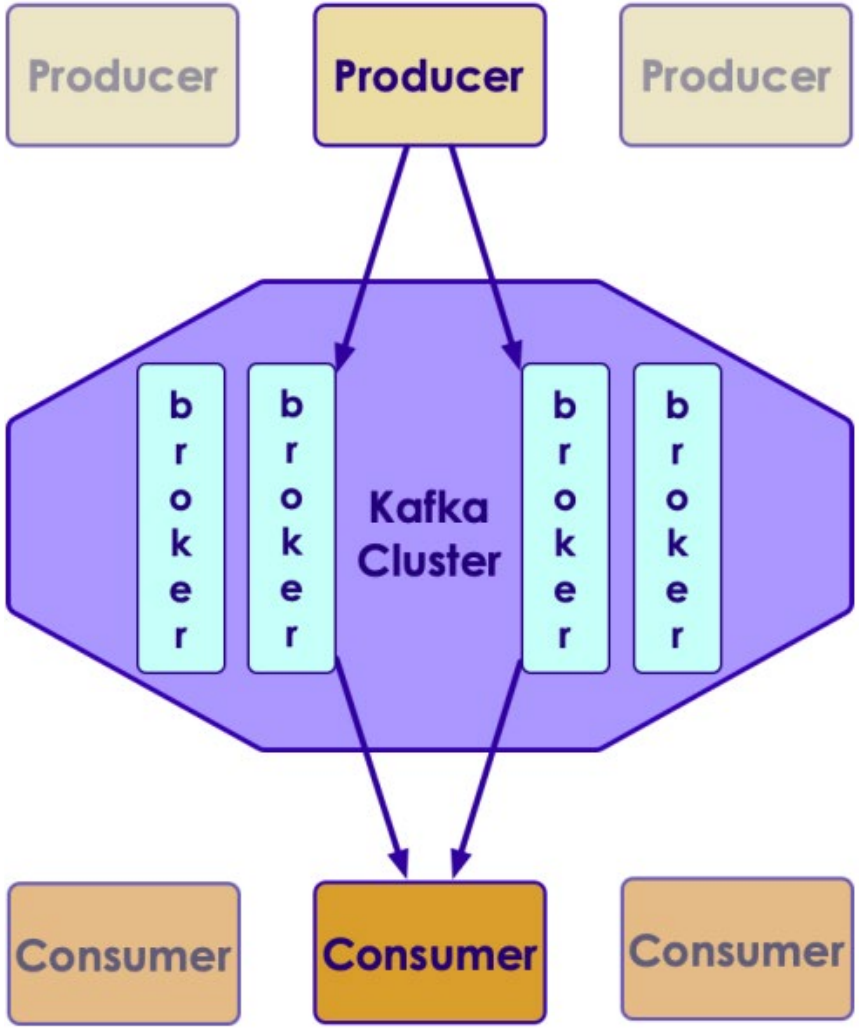
- Messages are written in order on each partition
  - ✓ Partitions are ordered and immutable
  - ✓ No order maintained across partitions
  - ✓ Producers write at the end of partition (append)
  - ✓ Sequential writes -> higher throughput



Brokers

- A Kafka broker is a Java process that runs on a node (machine / host)
  - ✓ Runs as a daemon (background process)
  - ✓ One broker daemon per node
- Brokers are designed to run as cluster
  - ✓ Usually bare metal preferred for performance as opposed to virtualized machines
- A single broker can handle thousands of partitions and millions of messages

Brokers

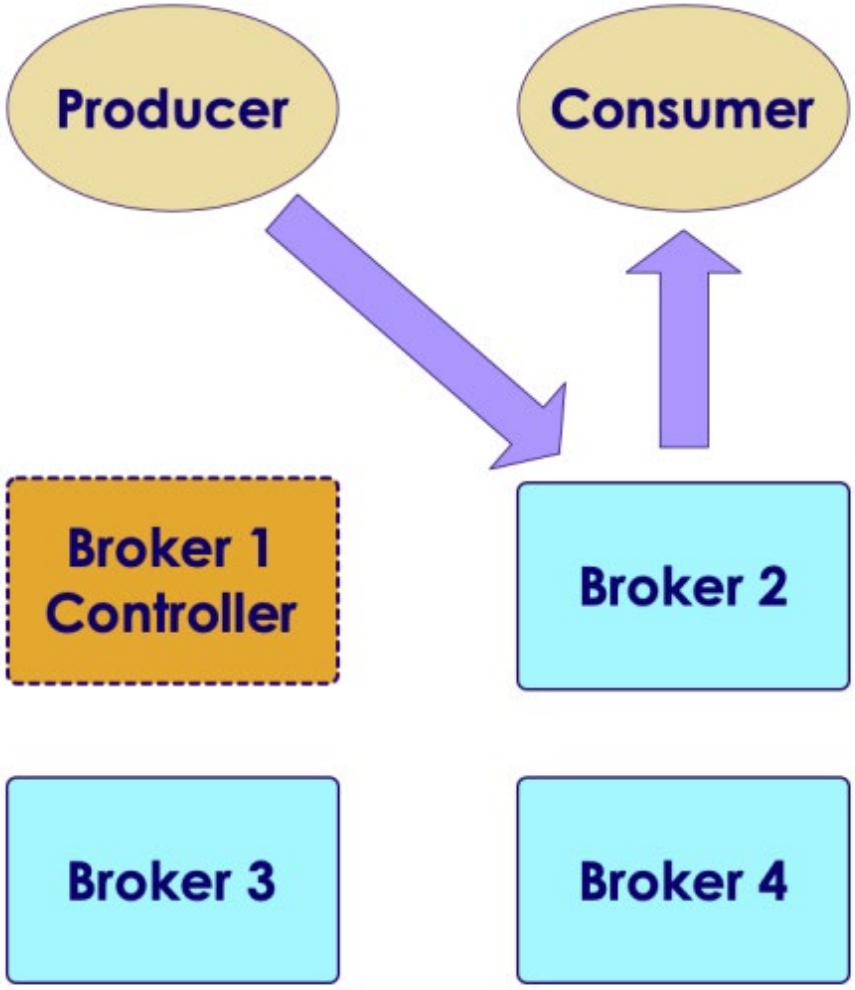


LearnQuest

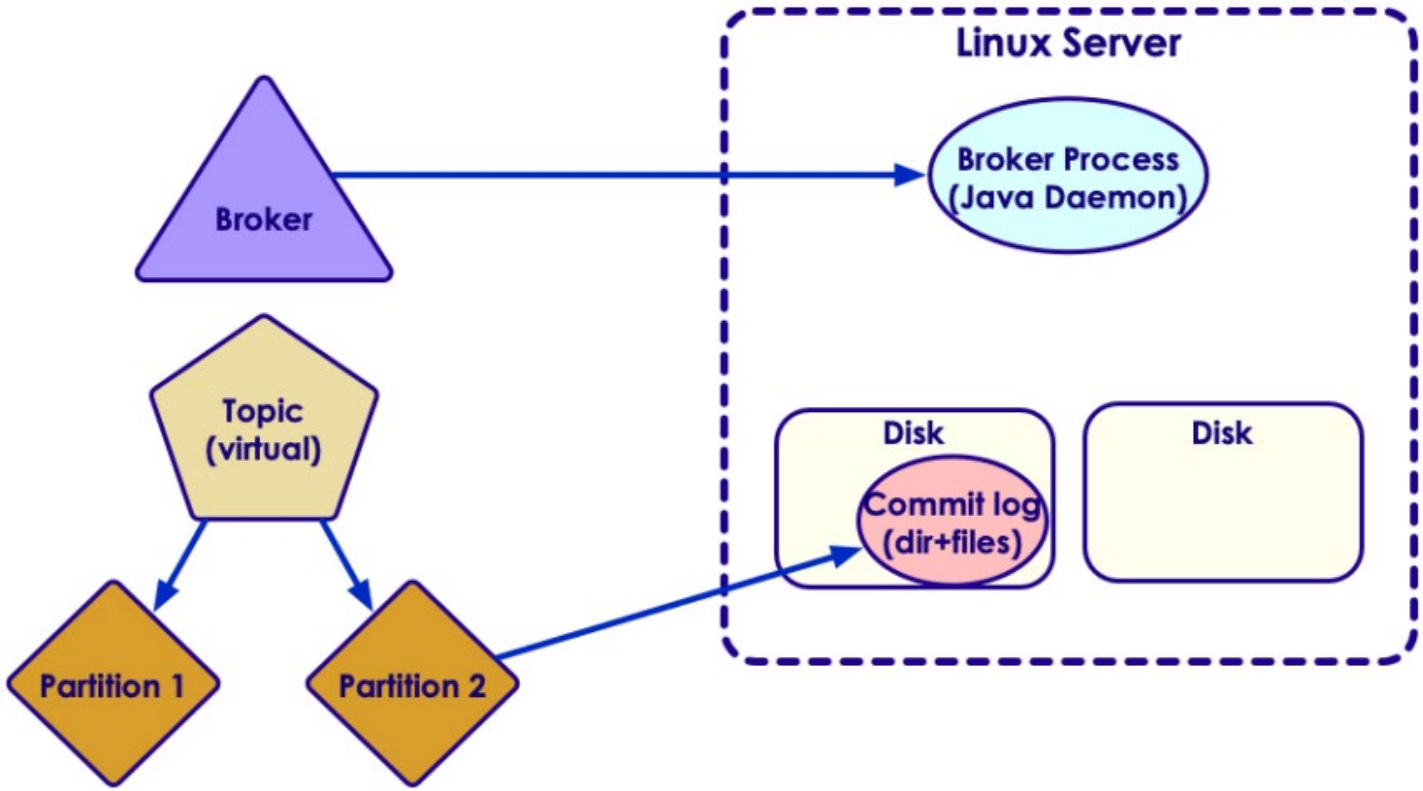
Broker Services

- Cluster
  - ✓ One broker is designated as controller / administrator of cluster
  - ✓ Selected automatically from all brokers
  - ✓ Monitors other brokers and handles failures
  - ✓ Assigns partition ownership
- Services to Producer
  - ✓ Accepts messages from Producers
  - ✓ Assigns a unique offsets (incrementing) to messages
  - ✓ Commits the messages to commitlog
- Services to Consumer
  - ✓ Serve message requests
  - ✓ Assign partitions to consumers in consumer groups

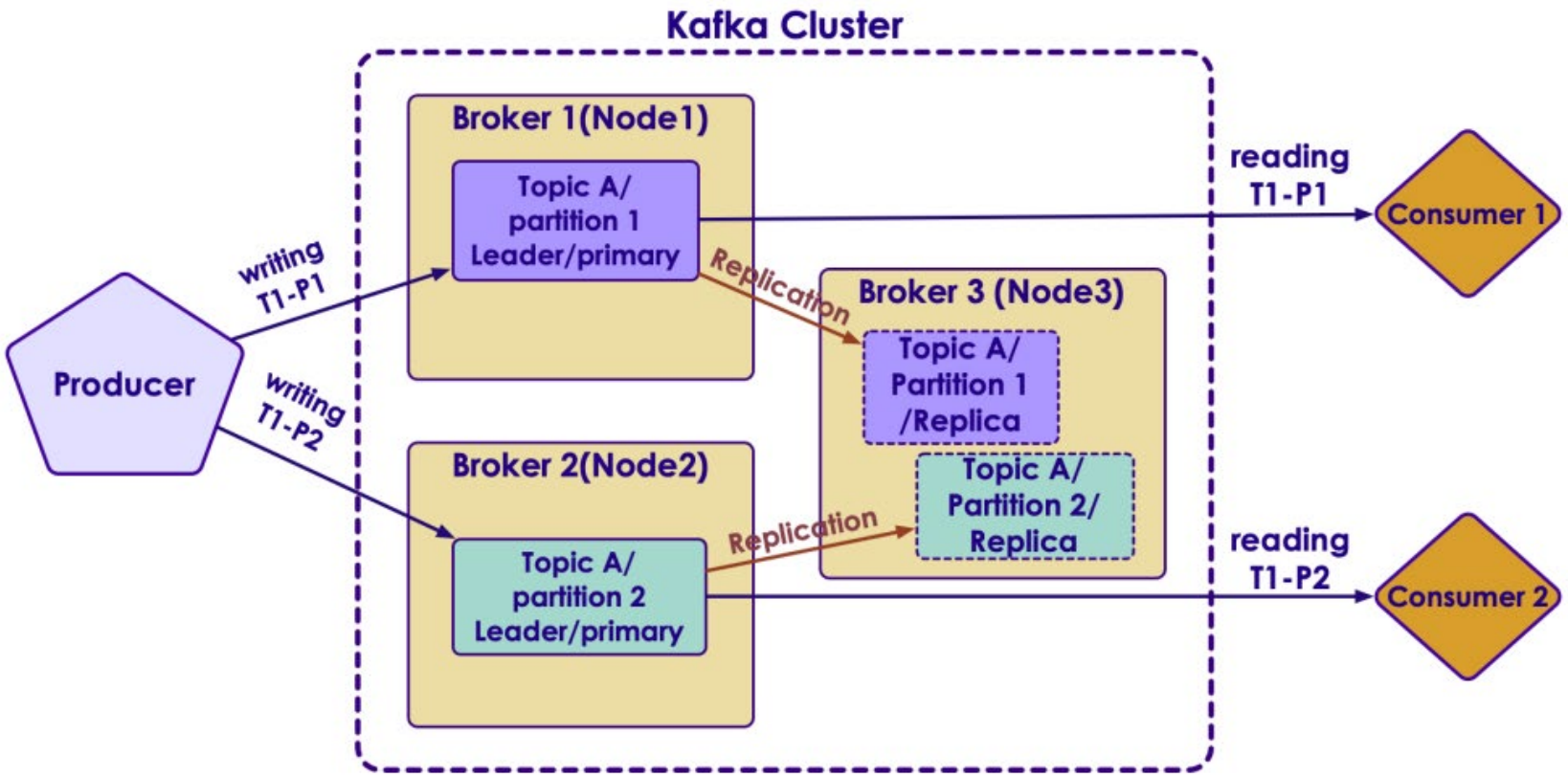
Broker Services



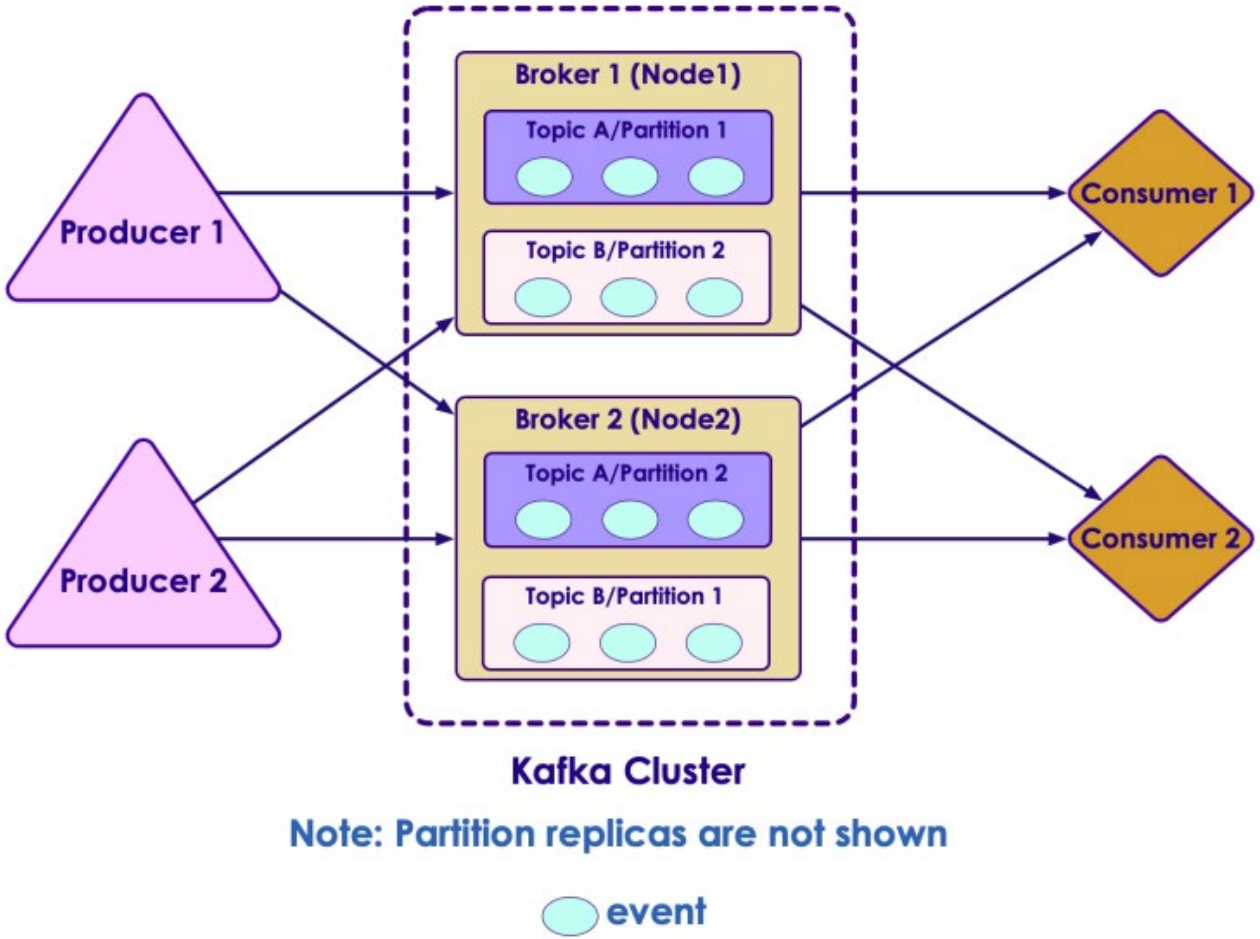
Kafka: Physical and Logical



Brokers / Leaders / Partitions / Replicas



Producers / Consumers / Topics / Partitions



Questions