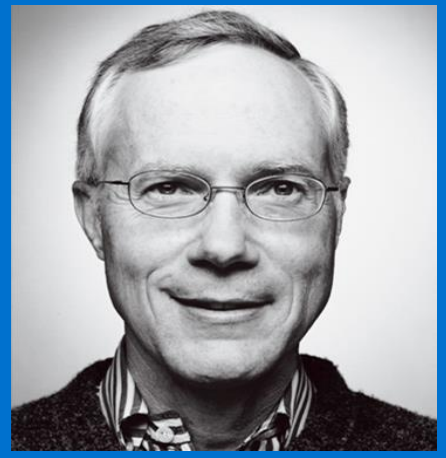
Apache Kafka Workshop

Indian Institute of Technology, Kharagpur - February 1, 2020

Speakers: Vineel Aggidi, Ashish Singh, Abhijith Rao, Naresh Ramesh



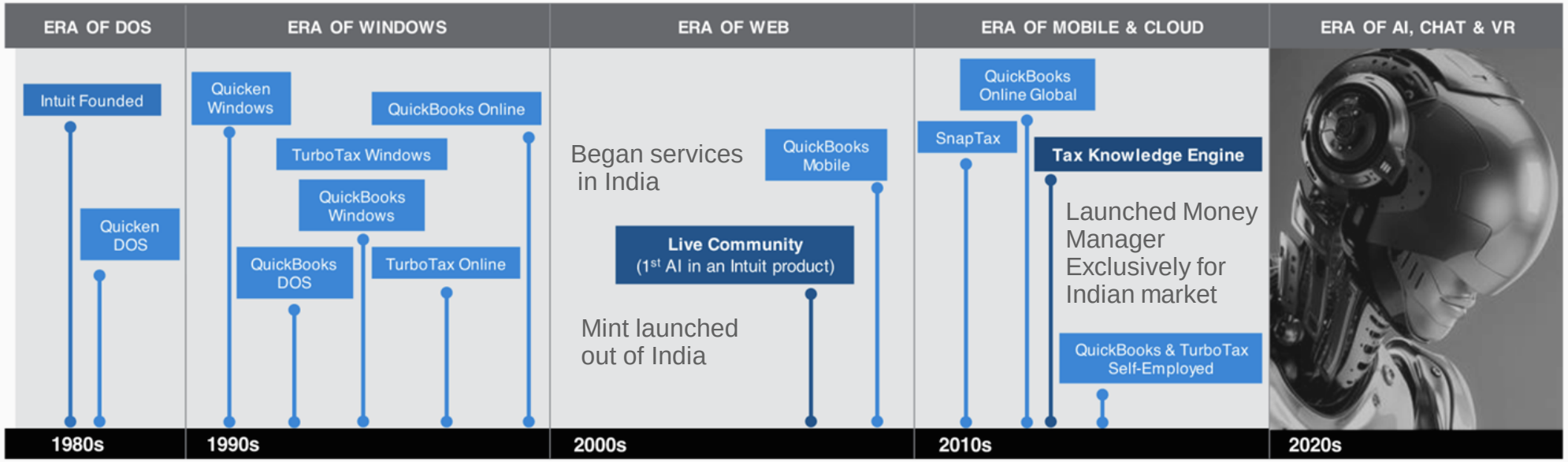
WHO ARE WE?

We are Intuit

A company conceived 35 years ago at our co-founder's kitchen table to help small businesses and individual customers grow, eliminate work and give them complete confidence.

Our Mission

Our Journey So Far



CUSTOMER-OBSESSED • DESIGN-INSPIRED • TECHNOLOGY-POWERED

Products that power prosperity

Our technology has helped us innovate four of our major products that are simplifying work of millions, worth millions.

50M
CUSTOMERS

$6B
COMPANY

Agenda

Fundamentals

Introduction to Kafka

Kafka terminologies

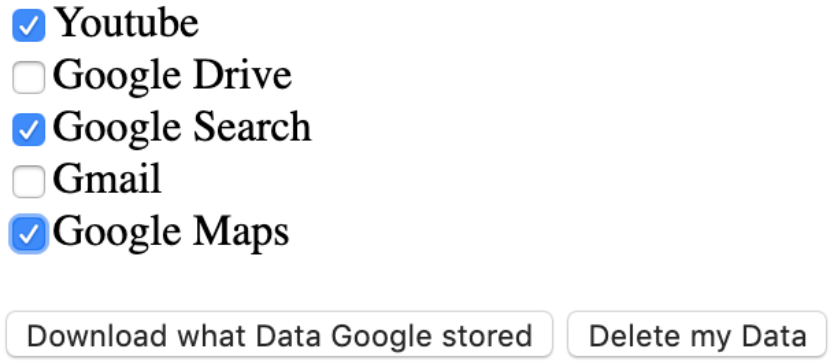
Kafka Architecture

Kafka Internals

Quiz

Use Case

- CCPA: California Consumer Privacy Act
- Ability for consumers to request Read or Delete their data present with the organization
- Complexity increases with multiple products tied to one account



One Possible Solution (Handle Synchronously)

```java
package com.workshop.CCPA;

import com.workshop.YoutubeDataManager;
import com.workshop.GoogleDriveDataManager;
import com.workshop.GoogleMapsDataManager;
import com.workshop.GMailDataManager;
import com.workshop.GoogleSearchDataManager;
import com.workshop.User;
import com.workshop.Utils;

public class DataManager {

    public static Boolean handleDeleteRequest(User deleteRequestUser) {
        Boolean youtubeDeleteReqStatus = YoutubeDataManager.deleteUserData(deleteRequestUser);
        Boolean googleDriveDeleteReqStatus = GoogleDriveDataManager.deleteUserData(deleteRequestUser);
        Boolean googleMapsDeleteReqStatus = GoogleMapsDataManager.deleteUserData(deleteRequestUser);
        Boolean gmailDeleteReqStatus = GMailDataManager.deleteUserData(deleteRequestUser);
        Boolean googleSearchDeleteReqStatus = GoogleSearchDataManager.deleteUserData(deleteRequestUser);
        /*
        perform other business logic and logging
        */
        return youtubeDeleteReqStatus && googleDriveDeleteReqStatus && googleMapsDeleteReqStatus
                && gmailDeleteReqStatus && googleSearchDeleteReqStatus;
    }

    public static String handleReadRequest(User readRequestUser, String format) {
        String youtubeData = YoutubeDataManager.fetchUserData(readRequestUser, format);
        String googleDriveData = GoogleDriveDataManager.fetchUserData(readRequestUser, format);
        String googleMapsData = GoogleMapsDataManager.fetchUserData(readRequestUser, format);
        String gmailData = GMailDataManager.fetchUserData(readRequestUser, format);
        String googleSearchData = GoogleSearchDataManager.fetchUserData(readRequestUser, format);
        String userDataLocation = Utils.storeDataAndReturnLocation(youtubeData, googleDriveData, googleMapsData,
                                                                   gmailData, googleSearchData);
        /*
        perform other business logic and logging
        */
        return userDataLocation;
    }
}
```

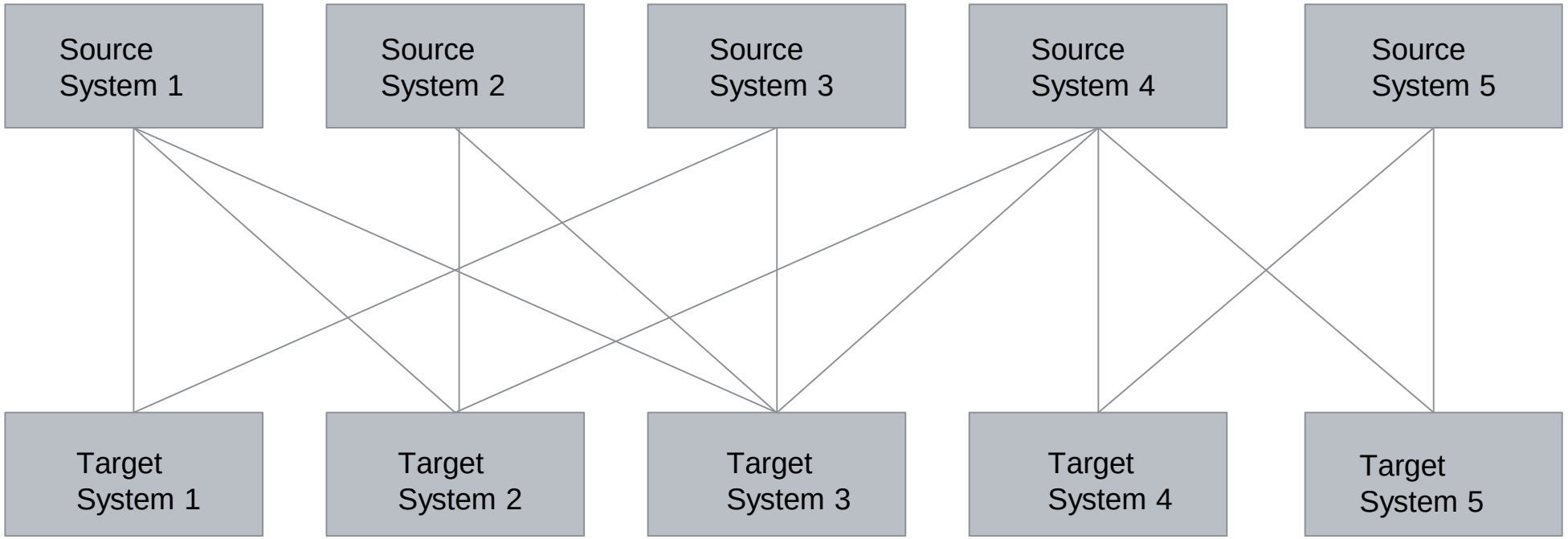
Drawbacks

1. Less Performant
2. Tight Coupling
3. Less Responsive

Asynchronous Programming

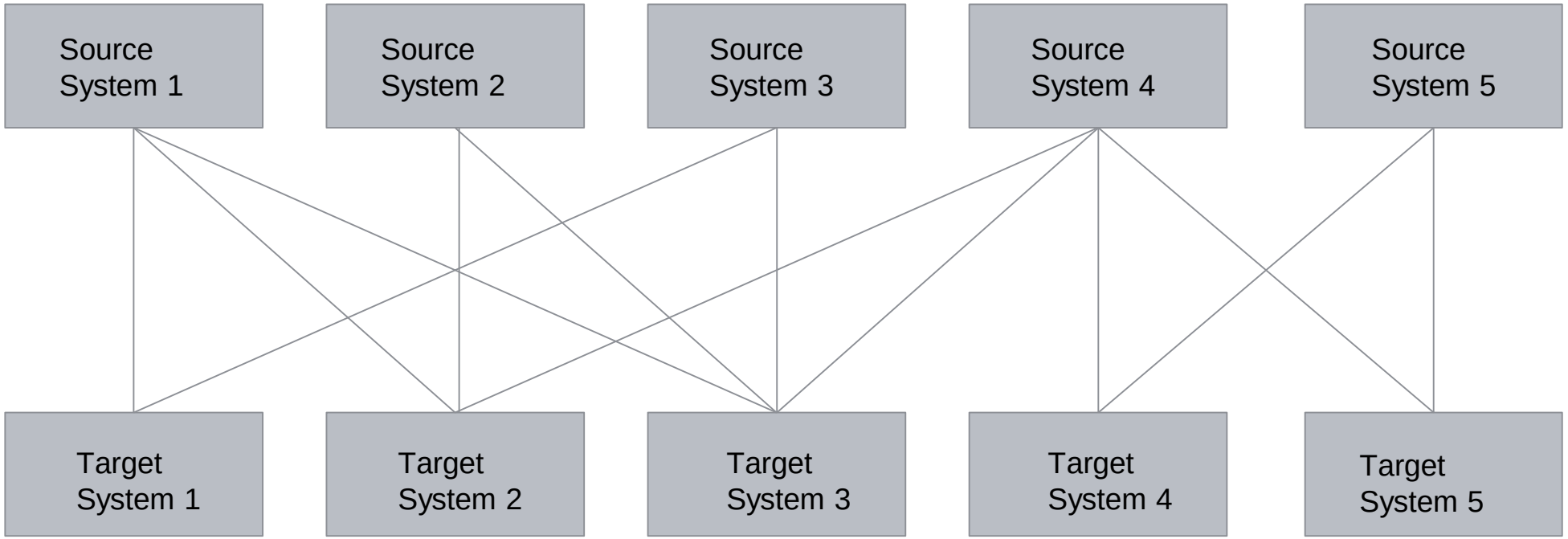
Messaging Systems

Need for messaging systems



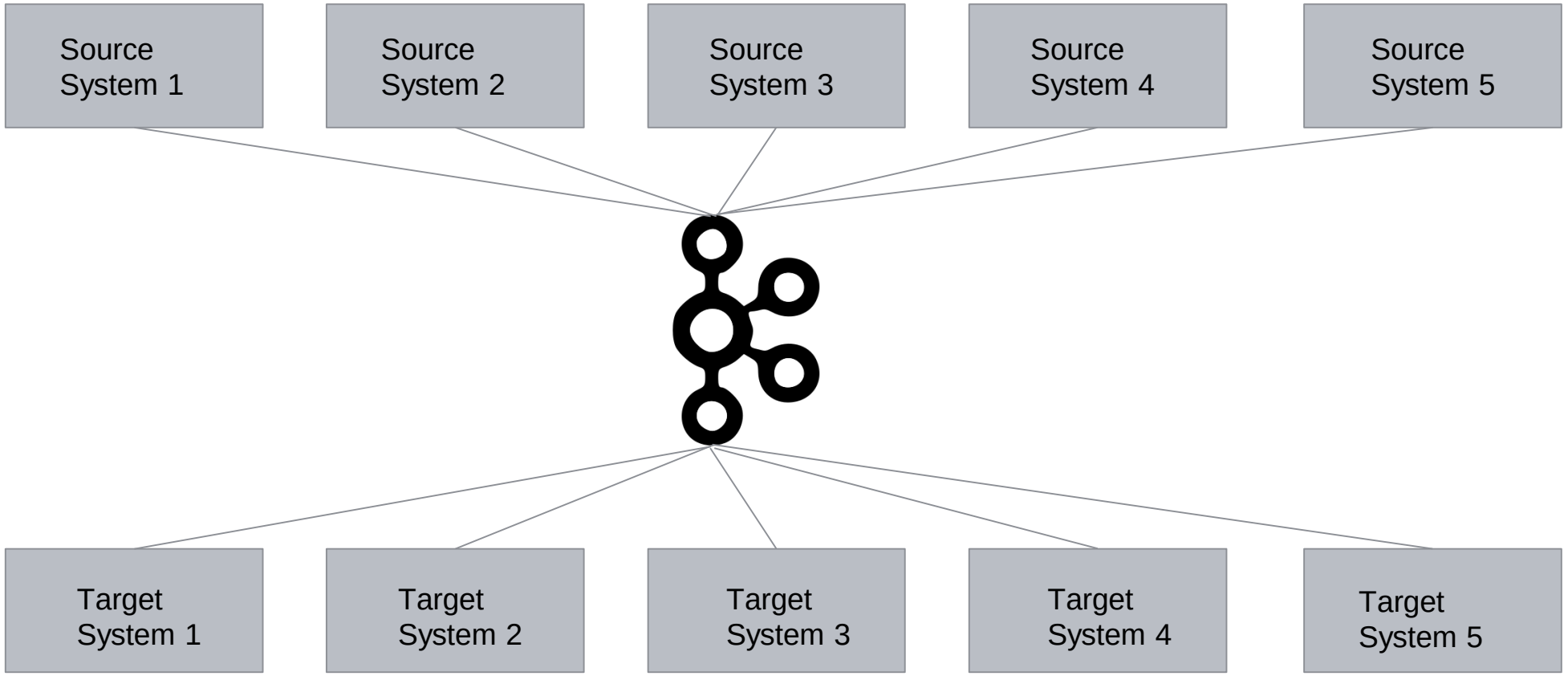
Communication is required between different systems in the real-time scenario, which is done by using data pipelines.

Complications around this approach



- With n source and m target systems, you can have up to m*n different integrations between them.
- Every integration has its own drawbacks
  - Which protocol to choose (TCP, HTTP, REST, FTP, etc)
  - Data format
  - How can data be parsed (binary, json, csv, etc)
- Handling these many pipelines is difficult. No more room to scale.

Solution - Messaging system



- Decouples the data pipelines.
- Makes the communication b/w systems simpler and manageable.
- Client libraries available for NodeJS, Java, C++, Python, Ruby, PHP and many more.

Kafka 101 - Introduction to Kafka

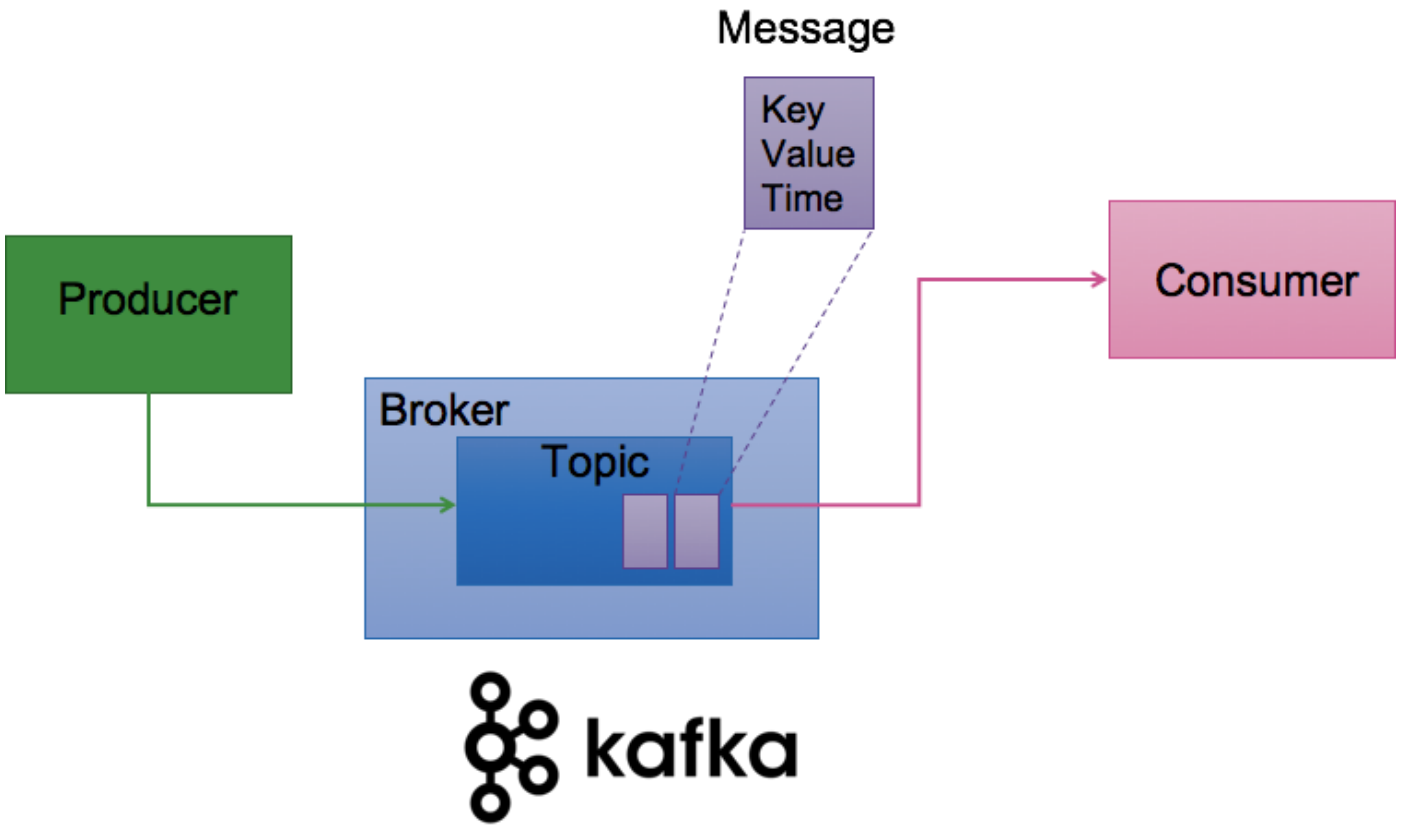
What is Kafka?

- Apache Kafka is a distributed *publish-subscribe* messaging system. It is -
  - Scalable
  - Durable
  - Fault-tolerant
  - Fast

- It was originally developed at LinkedIn and later became a part of Apache Projects.

Kafka Terminologies

- ***Producer***
  - produces messages to topic
- *Message*
  - == byte array
- ***Topic***
  - resides within ***Broker*** and it is *partitioned*
- ***Partitions***
  - are replicated
- ***Kafka Broker***
  - forms Kafka Cluster
- ***Consumer***
  - consumes together in consumer groups



Kafka Terminologies

Topic	Partition	Message/Record	Offset
A ***topic*** is category or feed name to which records are published	Topics are broken up into ordered commit logs called ***partitions***	***Record*** is just an array of bytes sent by the producer	Every record produced has an ***offset*** local to the partition

Producer[1]	Consumer[1]	Broker[1]	Zookeeper[1]
A ***producer*** can be any application that can publish messages to a topic	A ***consumer*** can be any application that subscribes to a topic & consume messages	Kafka cluster is set of servers, each of which is called a ***broker***	***Zookeeper*** is used for managing and coordinating kafka brokers

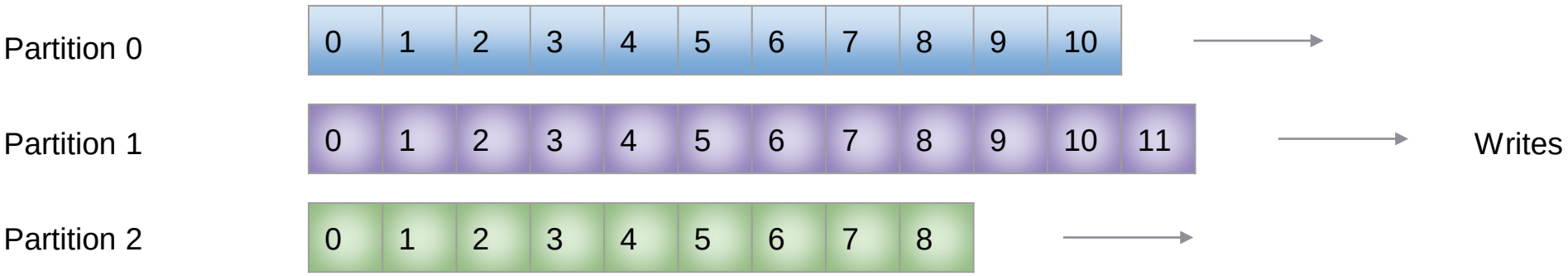
1. Communication between these components is done via high performance simple binary API over TCP protocol

intuit turbotax quickbooks mint

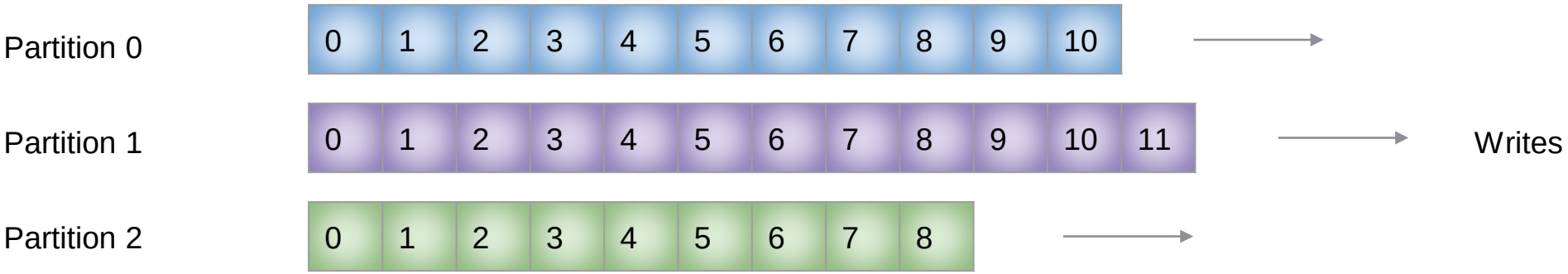
Hands-on

Topic, Partitions, Offsets

- Topic - a particular stream of data
  - Similar to a table in database
  - You can have as many topics as you want
  - A topic is identified by its name
- Topics are split into partitions
  - The partitions are ordered
  - Every message within a partition gets an incremental id called “Offset”
- Layout of topics



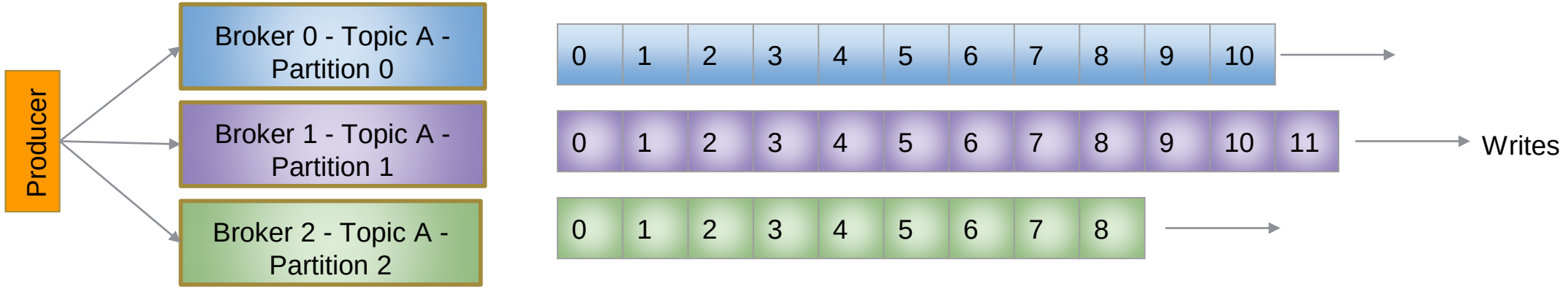
Topic, Partitions, Offsets



- Offset has a meaning only for a particular partition. For instance , offset 5 in partition 0 does not have the same data, as offset 5 in partition 1.
- Order is maintained only within a partition and not across partition.
- Data will be assigned to the partitions randomly if we don't provide key. *We will talk more on this in later sections.*
- Data is retained for a configurable amount of time (one week by default).
- Data once written to the partition cannot be changed - Immutability.

Producers

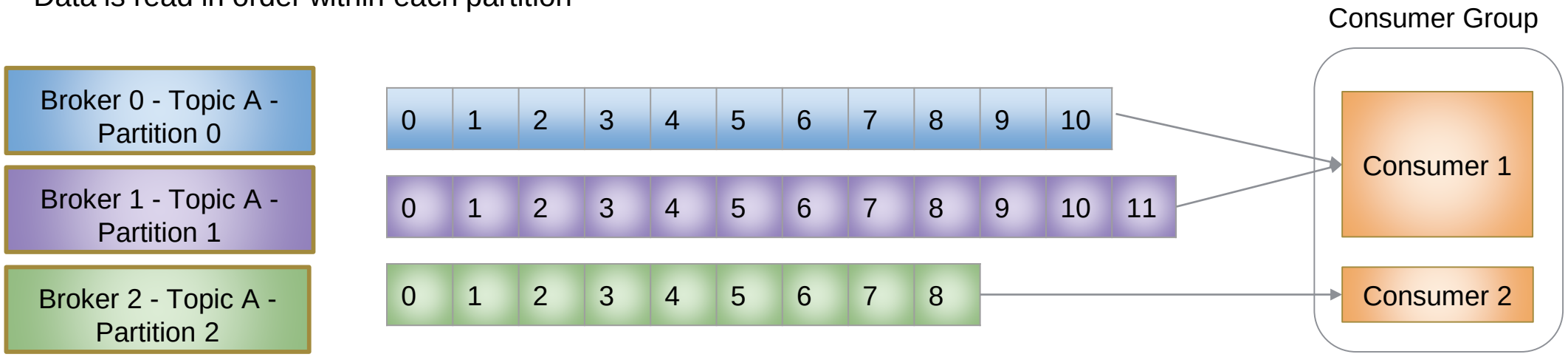
- **Producers write data to** topics **made of partitions**
- Producers knows automatically to which broker and partition to write to.
- In case of broker failure, Producers will recover automatically.



- **Producer can choose to receive ack for data writes**
  - acks=0 - Fire and forget - data loss possible
  - acks=1 - ack from leader - limited loss
  - acks=all - ack from leader & replicas - no loss
- **Producer can send key with message which can be string, number, object, etc**
  - key is null - round robin across brokers
  - key is sent - partitioning based on keyr

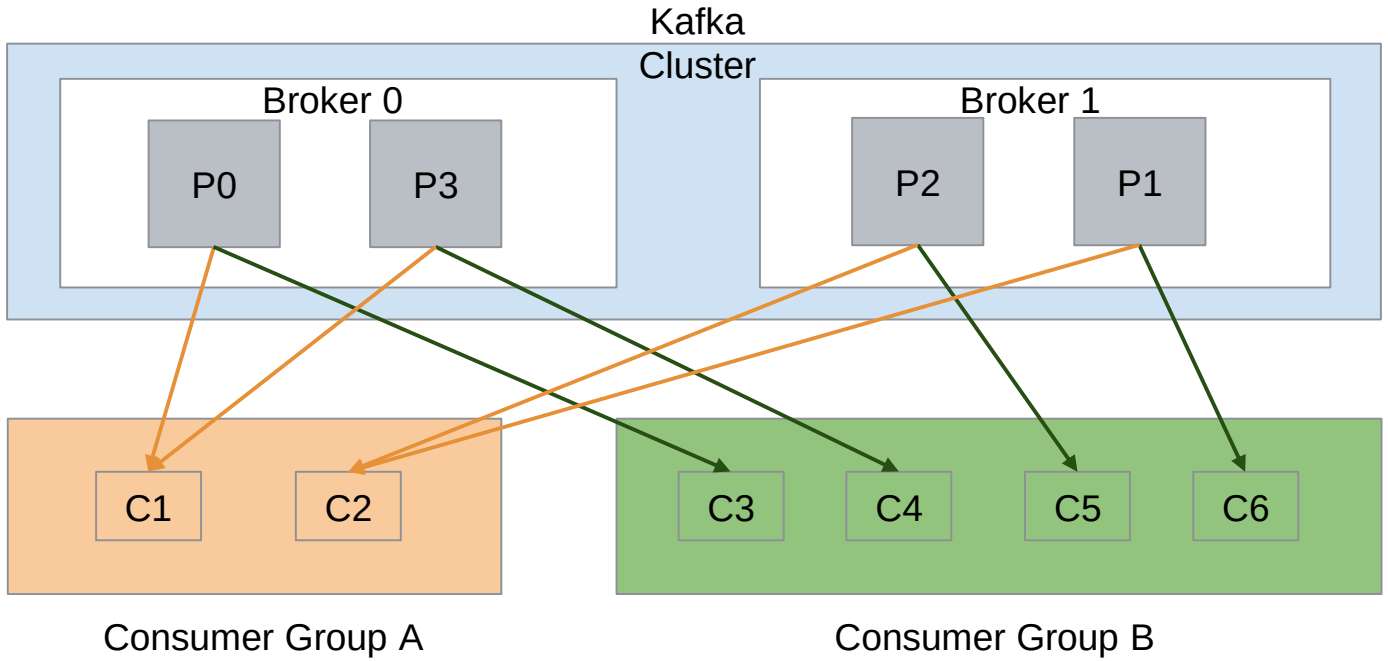
Consumers

- **Consumer can read data from a topic using the topic name.**
- Consumers know which broker to read from.
- In case of broker failure, consumers know how to recover.
- Data is read in order within each partition



Consumer Group

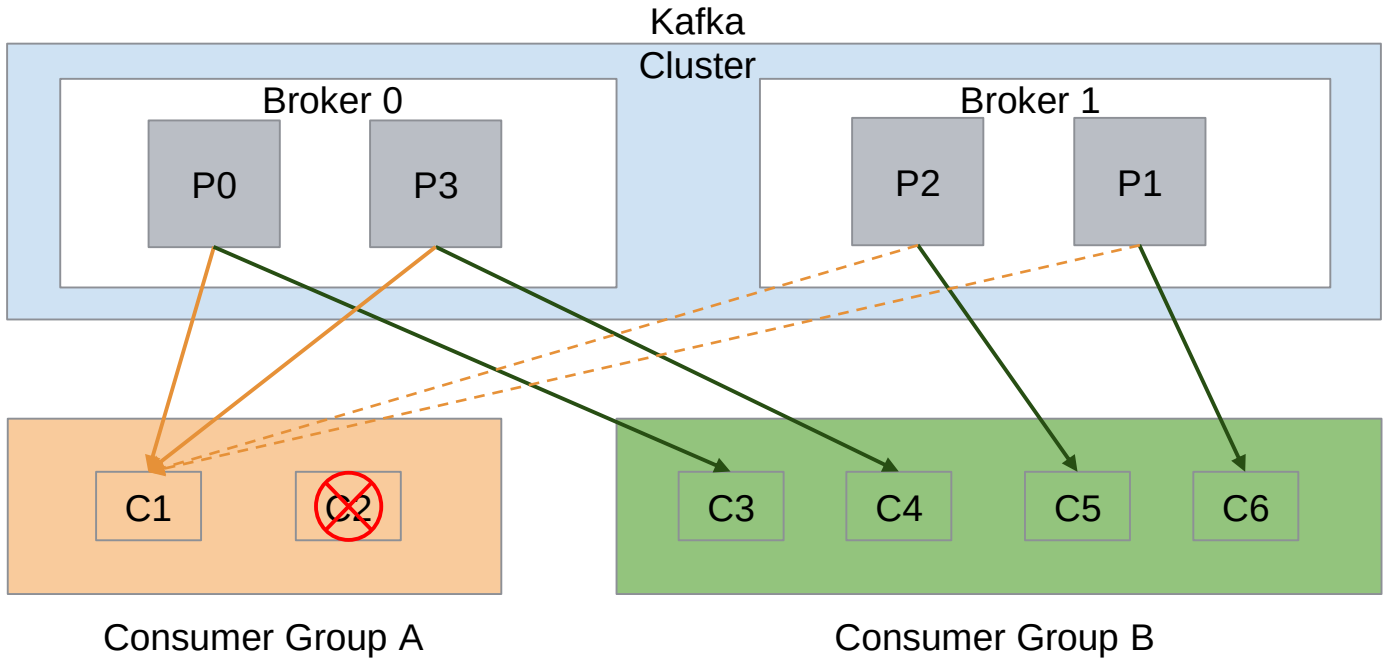
- Several consumers form a group to share the work
- Each consumer in the group and read from one or more partitions basis active consumer to partition ratio.
- At most one consumer can read a partition. Hence, if # of consumers > partitions implies idle consumers.
- Consumers will automatically use a group coordinator and consumer coordinator to assign consumer to a partition.



Consumer Groups provide isolation to topic and partitions

Consumer Group

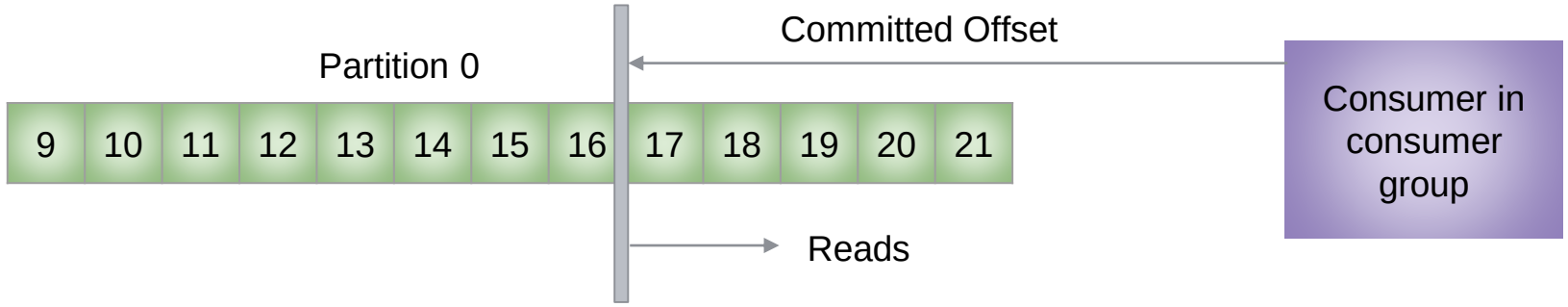
- Several consumers form a group to share the work
- Each consumer in the group and read from one or more partitions basis active consumer to partition ratio.
- At most one consumer can read a partition. Hence, if # of consumers > partitions implies idle consumers.
- Consumers will automatically use a group coordinator and consumer coordinator to assign consumer to a partition.



Can rebalance themselves

Consumer Offset

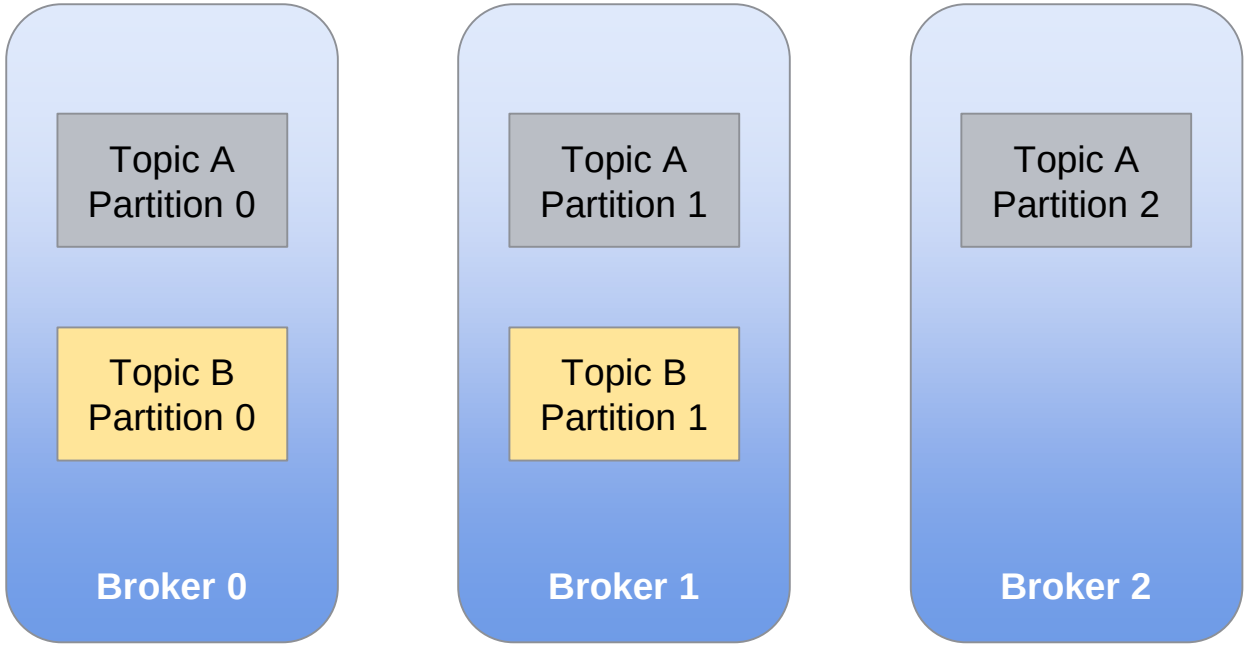
- Kafka stores the offsets at which the consumer group has been reading.
- The offsets committed are stored in a Kafka topic named __*consumer__offsets*.
- Once a consumer in the group processes the data received from Kafka, it commits the offsets to __*consumer__offsets*.
- When a consumer goes down, it restarts the reading from where it left off (made possible by committed consumer effects)



Broker

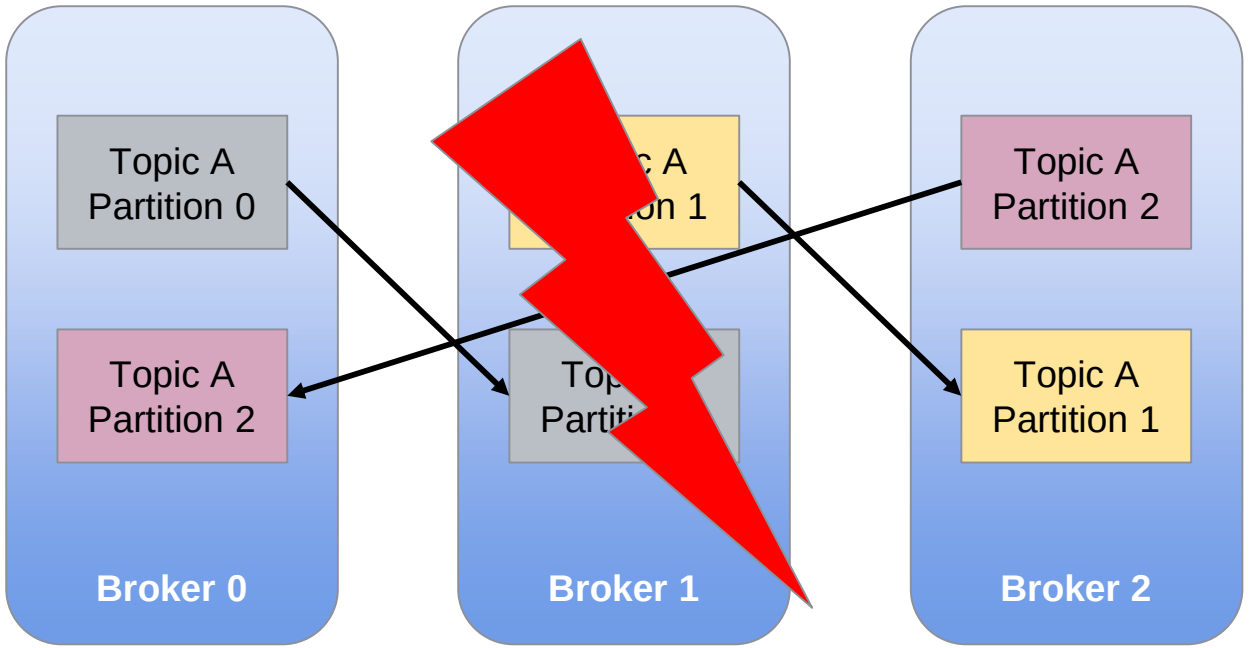
- Kafka cluster is set of servers, each of which is called a *broker*.
- Every broker is identified by its ID (an integer value).
- Each broker contains topic partitions.
- Once connected to any single broker, kafka client will be automatically be connected to the cluster.
- In the **example** below, we have taken 3 brokers. For a big cluster, there could be over 100s of brokers.

Topic distribution in Brokers



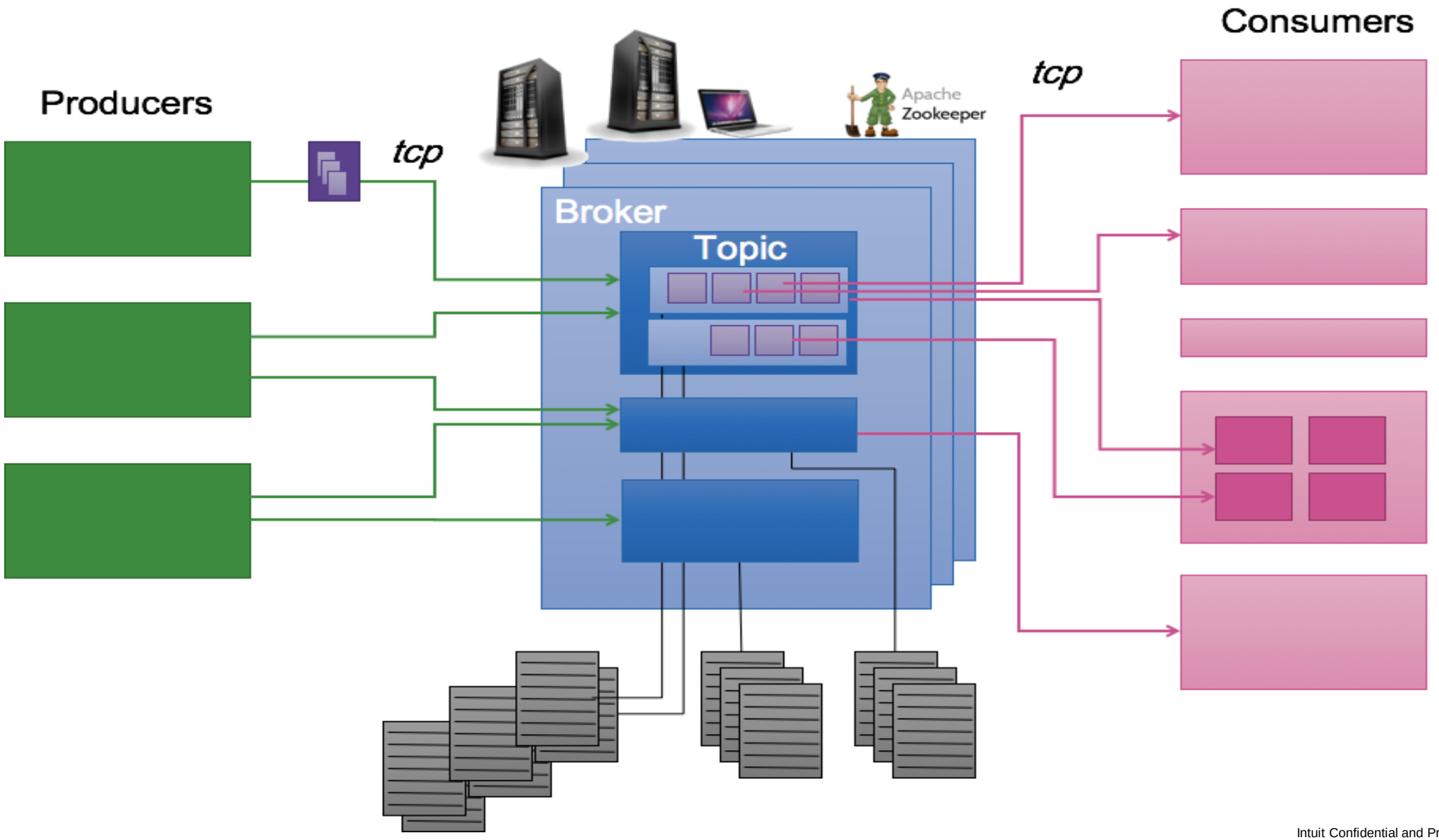
- Topic A with 3 partitions
- Topic B with 2 partitions

Replication factor



- Topics should have replication factor more than 1 for resiliency. **[Topic A has replication factor as 2]**
- If a broker is down, another broker will serve the data.
- **Use case** - Broker 1 dies | **Result** - Broker 0 and 2 can still serve the data
- For a topic with replication factor N, Kafka can tolerate N-1 server failures w/o losing any message committed to the log.

Kafka Architecture

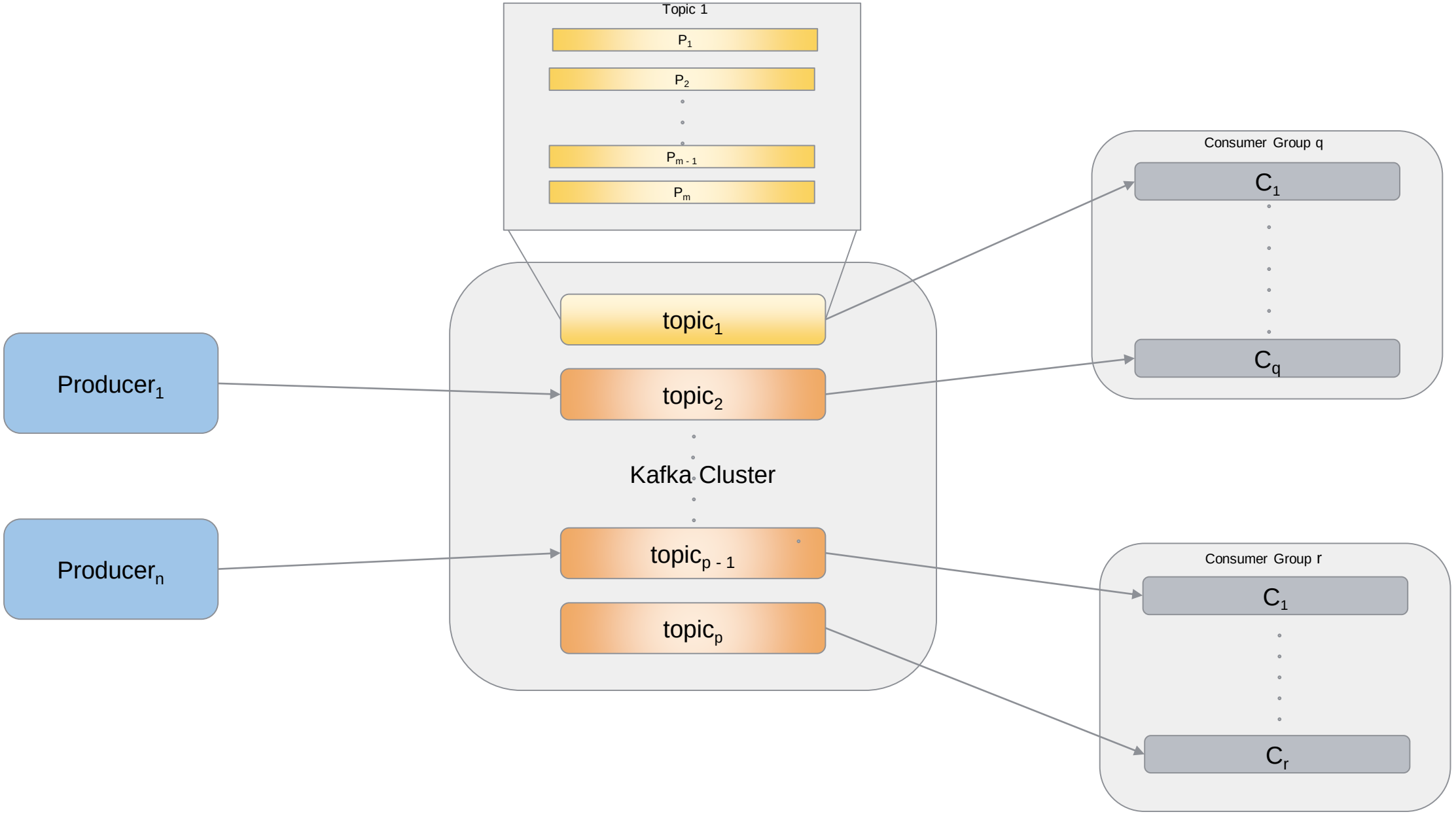


Kafka @Intuit

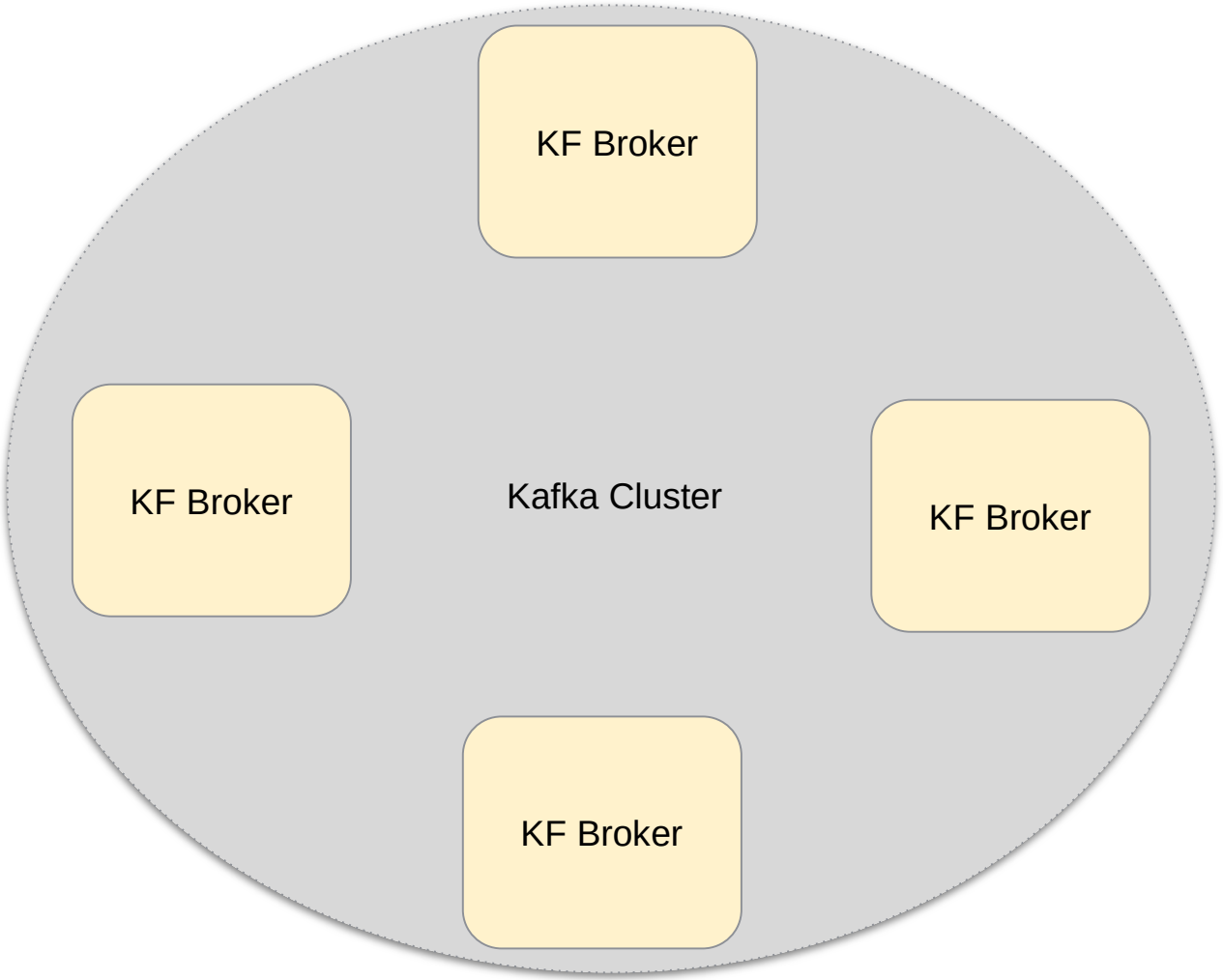
Alone has 15,398 partitions in Prod
Processes 98M messages per day

Your opportunity to ask and learn

Kafka 101 - Rewind


Topic 1
P_1
P_2
P_{m-1}
P_m
Consumer Group q
C_1
C_q
$topic_1$
$topic_2$
Kafka Cluster
$topic_{p-1}$
$topic_p$
$Producer_1$
$Producer_n$
Consumer Group r
C_1
C_r

Kafka Cluster

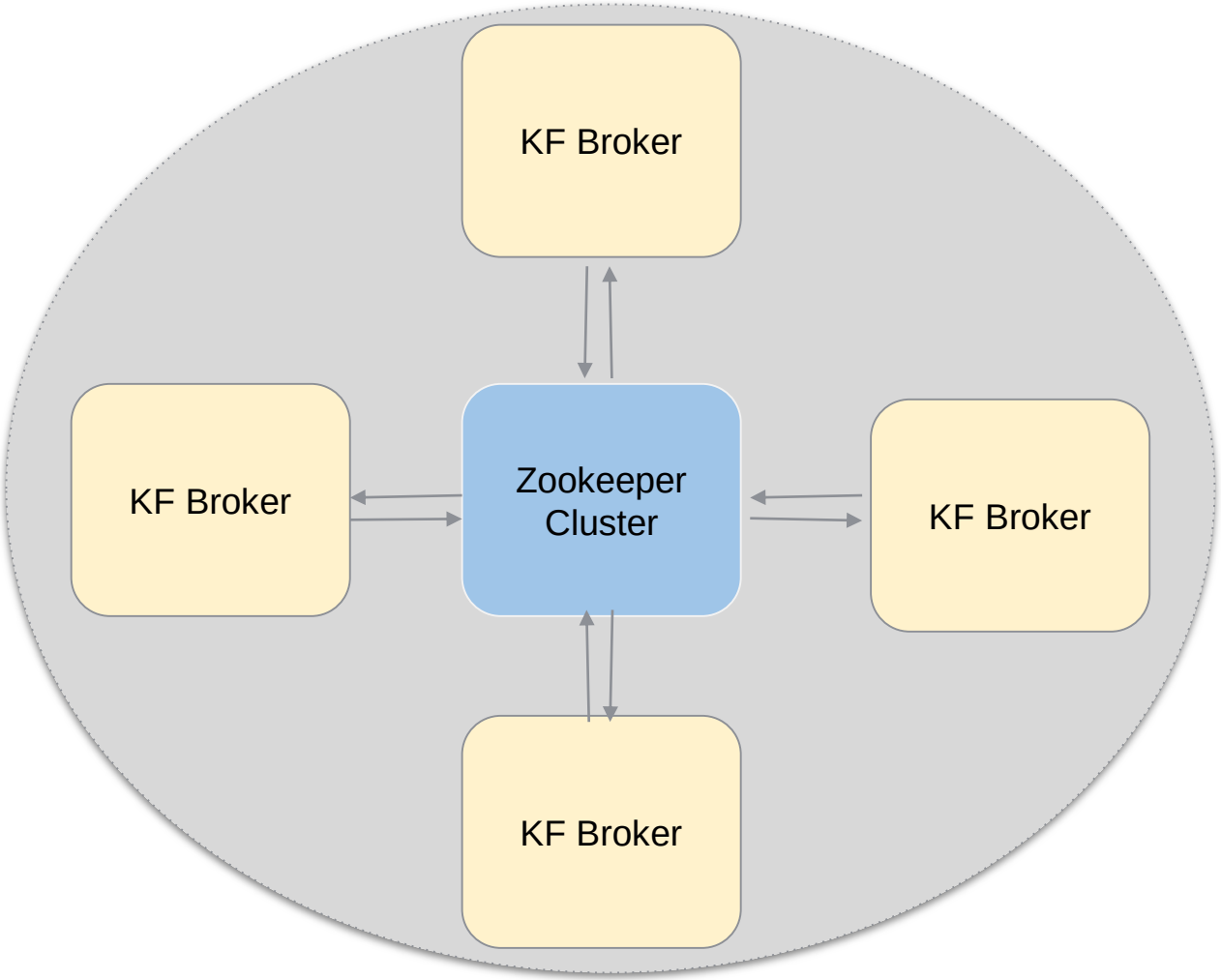


- **Multiple brokers**
  - Handles failures via redundancy.
- **Each brokers are primarily identical in responsibilities.**
  - Peer to Peer system.
  - Some brokers have special responsibilities.
- **Peer to peer system ?**
  - How are states changes communicated?
    - Distance Vector routing protocol 😖
  - How are failures detected?

Detection Of Broker Failures

- Send heartbeat to every other broker ?
- Send heartbeat to a leader ?
  - Selection of leader ?
- Challenges
  - Split Brain Problem.
  - Consensus.
  - What if all brokers go down?

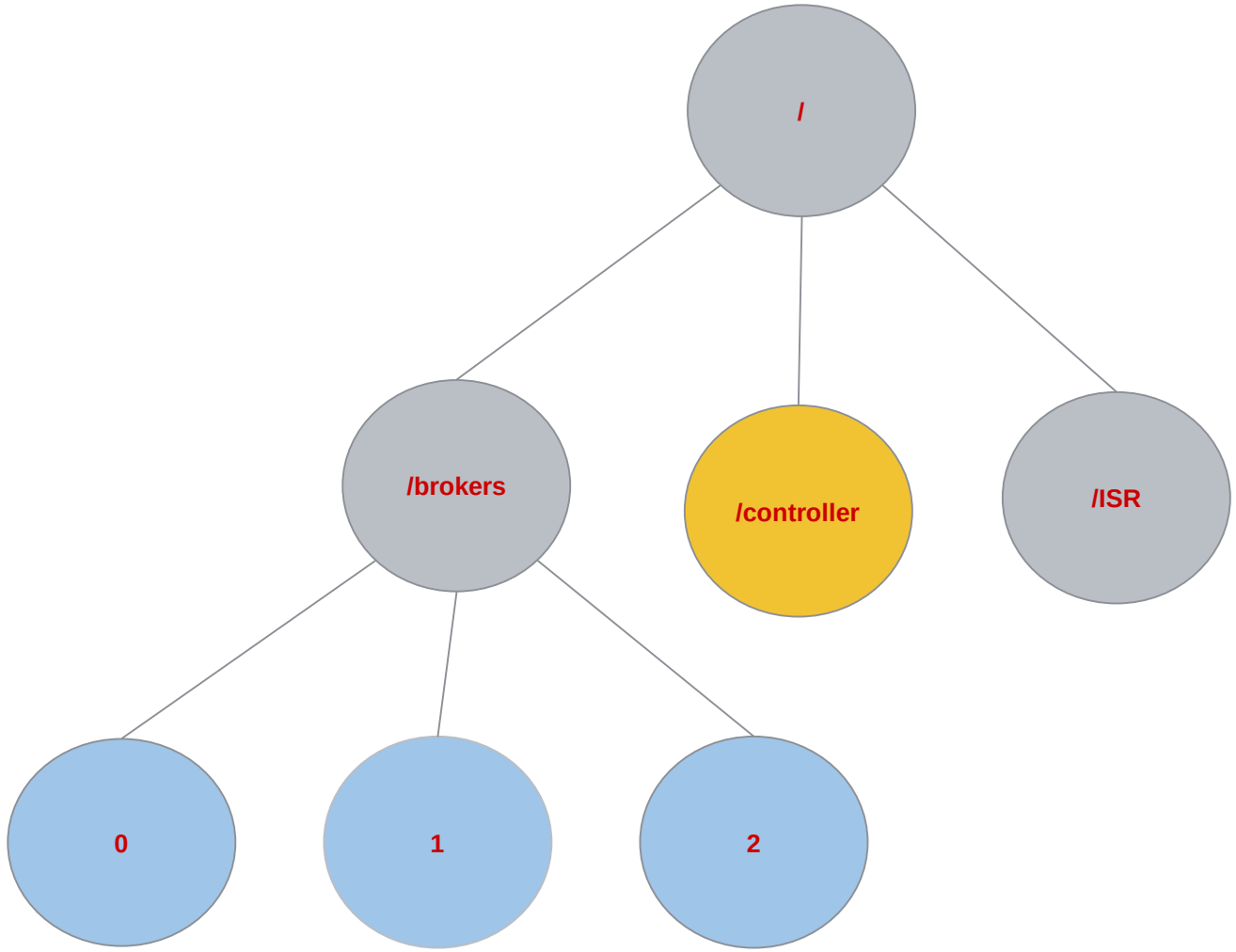
Detection Of Broker Failures



Zookeeper

- Centralized service with
  - Distributed hierarchical key-value store.
  - Support for distributed synchronization.
- In cluster mode
  - Modification requests always to the leader.
  - A successful write requires acknowledgment from more than half of the nodes in the cluster. - Resiliency :)

Zookeeper Primitives



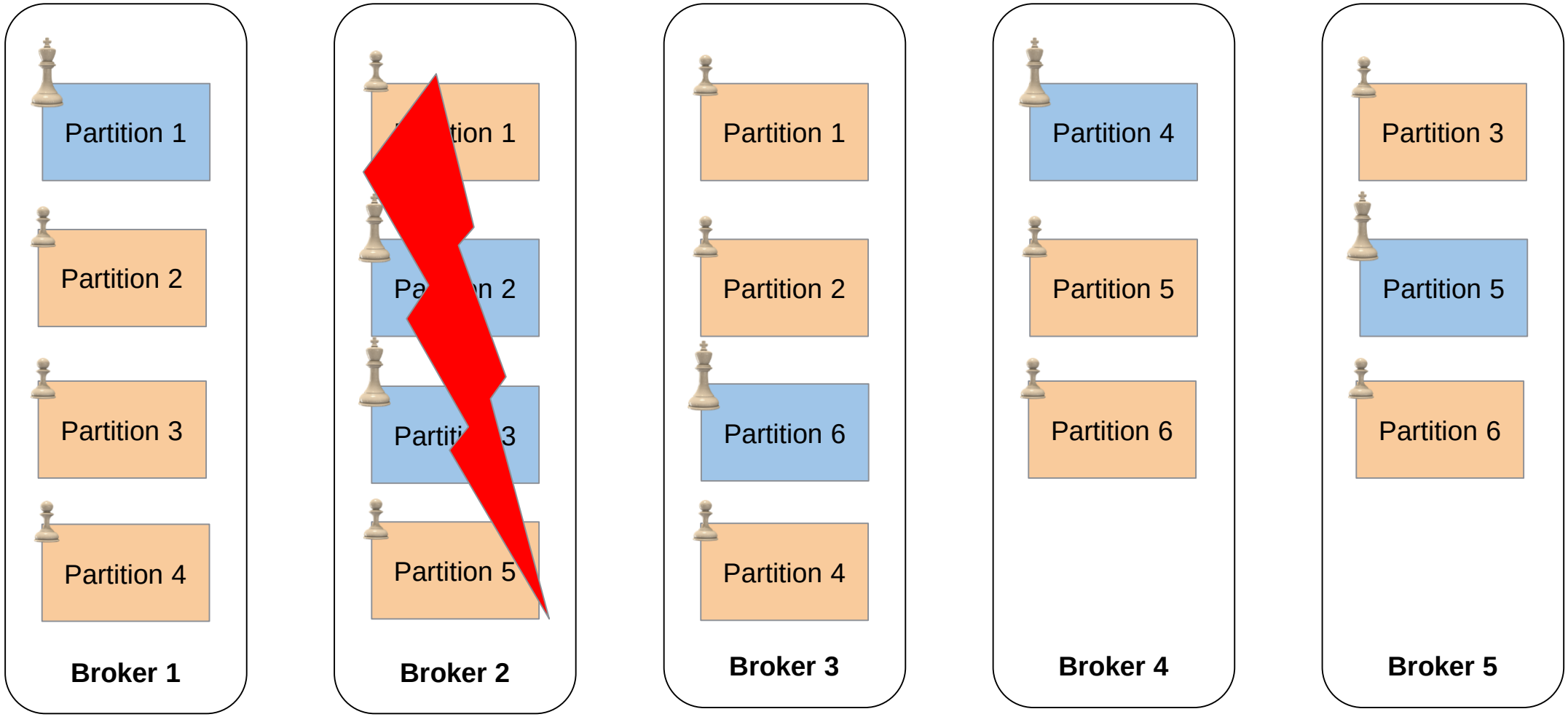
Data Model

- Like a standard file system.
- Called Znodes
- Not just the leaves, all znodes can contain data.

Zookeeper Primitives

- Ephemeral Node
  - As soon as session with the client is terminated, the node gets deleted.
  - Kafka utilizes the mechanism to determine if a broker is down.
  - A combination of ephemeral and sequential nodes are used for leader election recipe.
- Watches
  - A mechanism to notify all subscribers of a change in the path.
  - Kafka utilizes this to notify all brokers in the cluster that a node has gone down.
- Persistent Nodes
  - Generally used to manage information.
  - Kafka uses it to store metadata like consumer group leader / partition assignments -> For recoverability.

Coordination amongst Brokers



- **Leader for partition 2 and 3 is down due to broker 2 failure**

Coordination amongst Brokers

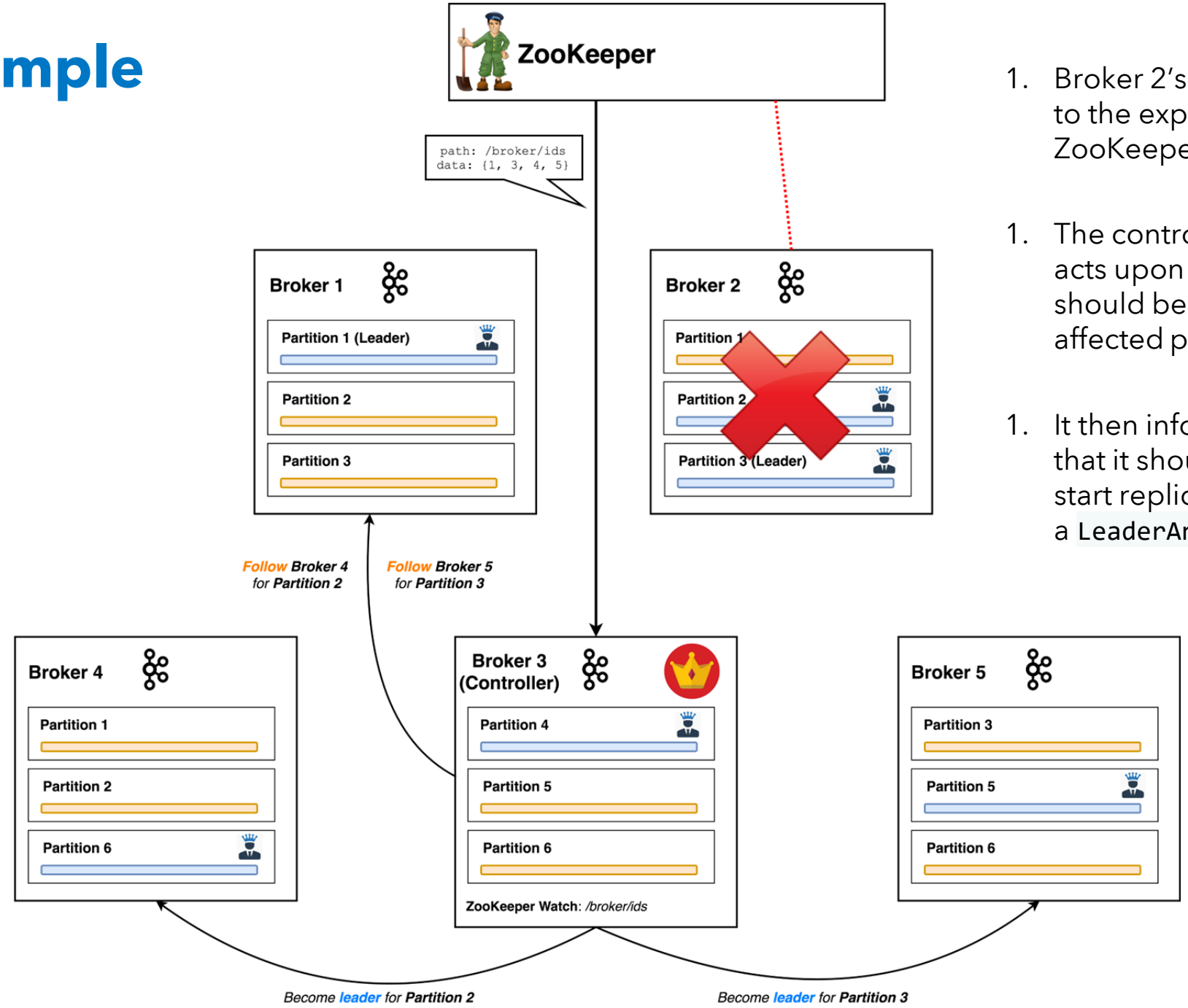
- Leader for partition 2 and 3 is down due to broker 2 failure
  - How should the system react in this case?
- Results in down time if broker where the partition leader lived goes down
- Should the ZK or the follower broker find substitute leaders?
  - If ZK
    - It should know how Kafka works. Does it know?
  - If follower
    - Multiple followers exist, which follower?


Distributed System
Problems
Kafka
Controller

Controller - Workhorse of kafka cluster

- It is a normal broker with special responsibility of -
  - keeping track of nodes in the cluster and appropriately handling nodes that leave, join or fail
  - rebalancing partitions and assigning new partition leaders
  - create/delete a topic, add partitions (and assign them leaders)
- ZooKeeper watches are crucial to Kafka - they serve as input to Kafka Brokers esp. Controller
  - gets notified of failing, new, re-joining brokers in the cluster
- The state of topic partitions that controller holds/controls is persisted in ZK
  - If Controller broker goes down, Kafka Controller Election happens and other broker becomes Controller
- Broadcast the latest state of topic partitions to all other brokers

Example

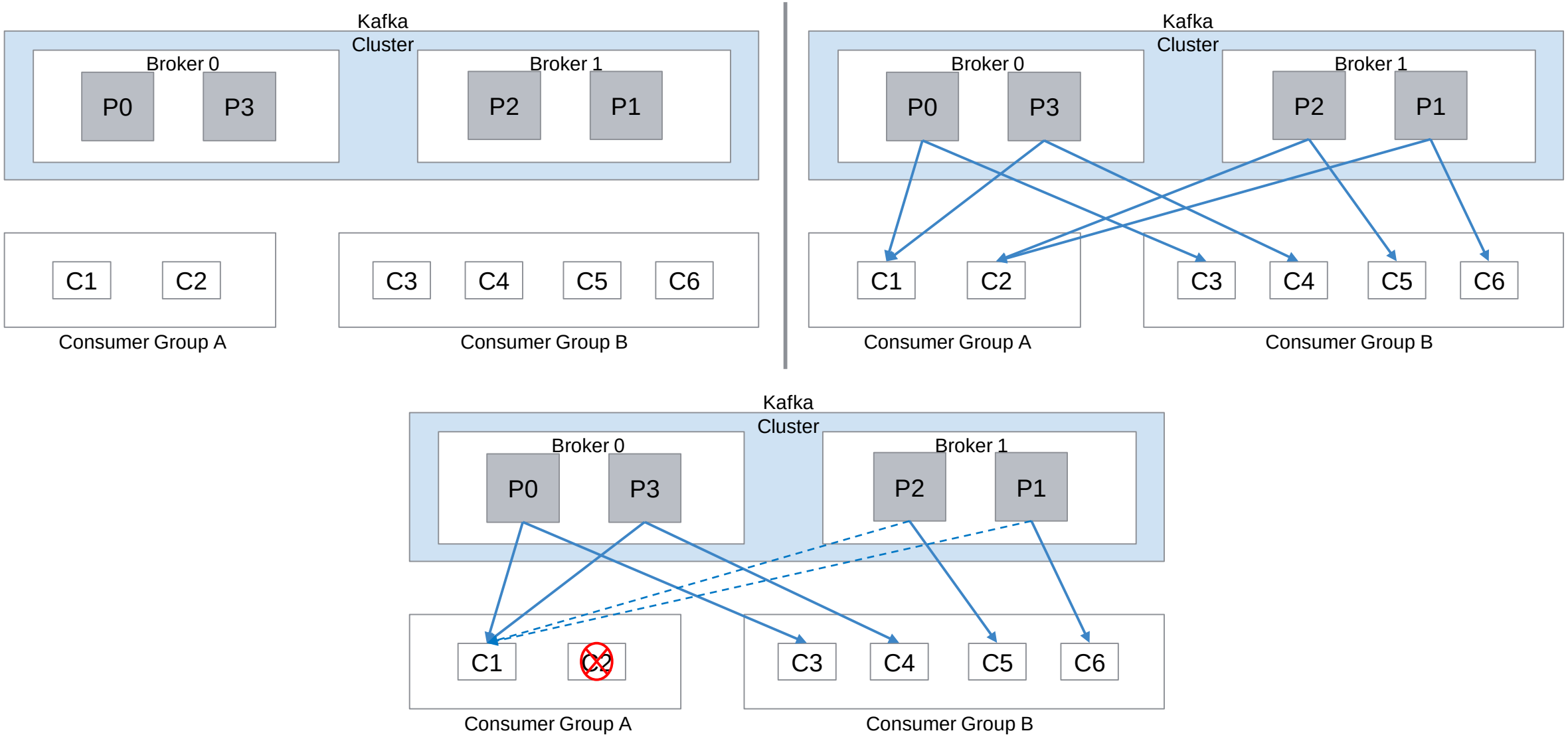


1. Broker 2's id is deleted from the list due to the expiry of the faulty broker's ZooKeeper Session

1. The controller gets notified of this and acts upon it. It decides which nodes should become the new leaders for the affected partitions.

1. It then informs every associated broker that it should either become a leader or start replicating from the new leader via a `LeaderAndIsr` request.

Consumers coming up and going down all the time



Consumers coming up and going down all the time

- Someone needs to be aware of which all consumers are alive and which are not?
    - Who should keep track of the consumer group's health?
- Number of consumers and partitions vary all the time - how should they be assigned?
- Consumers fail all the time - many a times intentionally for AMI upgrades
    - Who should substitute for its assigned partition? and on what basis?
- New consumers come for various reasons - a faulty machine's replacement, deployment, scalability, etc.
    - It shouldn't be lying idle and starts consuming messages.
- What we need?
    1. Ability to check consumer's health
    2. And lead efforts to assign partitions to consumers

Consumer Group Coordinator

- Group coordinator is responsible for managing the state of the consumer group
  - receives periodic heartbeats from all consumers in a consumer group
  - marks consumers as dead if periodic heartbeats aren't received
- Mediate partition assignment when members arrive or depart, and when topic partition metadata changes
  - Signals the group of changes and rebalances the group for consumers to rejoin partitions
- In the case that the group coordinator broker shutdowns, the Zookeeper will be notified and the election starts to promote a new group coordinator from the active brokers automatically.

Consumer Group Leader

- Consumer group leader is responsible for -
  - Receiving list of active consumers from the group coordinator
  - Assigning subset of partitions to each consumer active in the group
  - Report list of assignments to group coordinator which sends this information to all the consumers

Your opportunity to ask and learn