Introduction to Kafka (and why you care)

Richard Nikula
VP, Product Development and Support
Nastel Technologies, Inc.

Introduction

EMA NASTEL LEADER 2017 ADVANCED IT ANALYTICS

- **Richard Nikula**
  - VP of Product Development and Support

- **Involved in “MQ” since early 90’s**
  - Primarily at the technology layer
  - Various certifications

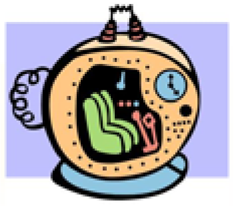
- **About Nastel Technologies**
  - Founded in 1994
  - Middleware-centric Application Performance Management software supplier
  - Core competency : Messaging Middleware, Java Application Servers, ESB's and other SOA technologies

About this session

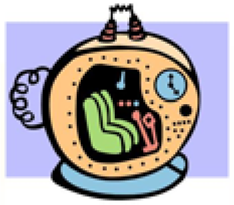
- Apache Kafka is showing up everywhere and is likely already being used today somewhere in your organization. In this session we will cover the fundamentals of Kafka. The basics of producers, consumers and message processing will be explained, along with several examples including clustered configuration. We will also look at several typical use cases.
- This session is targeted at technical resources familiar with IBM MQ. Session will compare Kafka to IBM MQ-based messaging to help you prepare for when your expertise is needed in a hybrid IBM MQ/IIB/Kafka environment.
- This session is not an exhaustive tutorial to Kafka and only touches on programming concepts.

BACKGROUND

Time Travel



1993

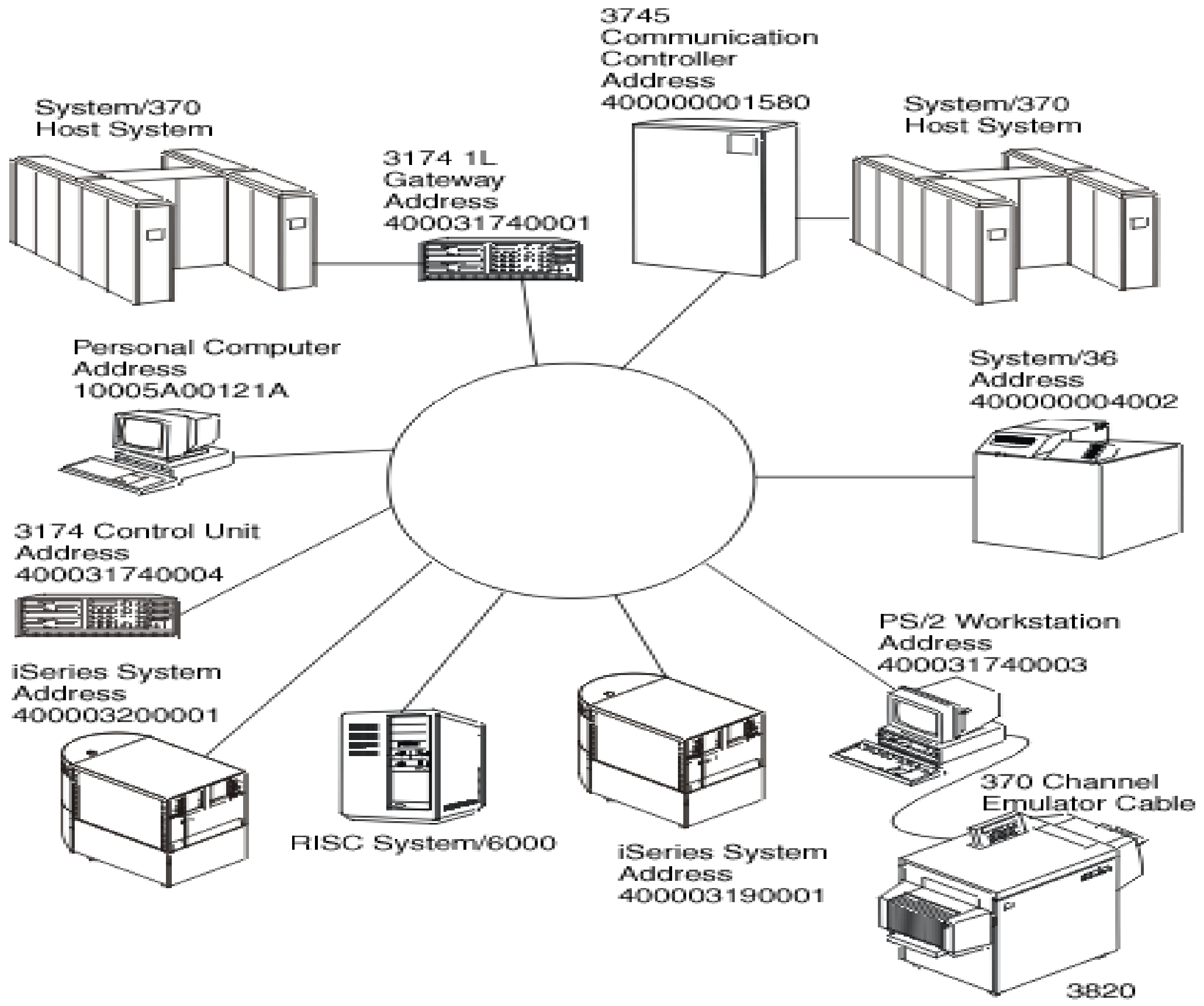


Your workstation

Operating system	PC DOS 4.01
CPU	Intel 80386SX @ 16 MHz
Memory	2 MB ~ 6 MB

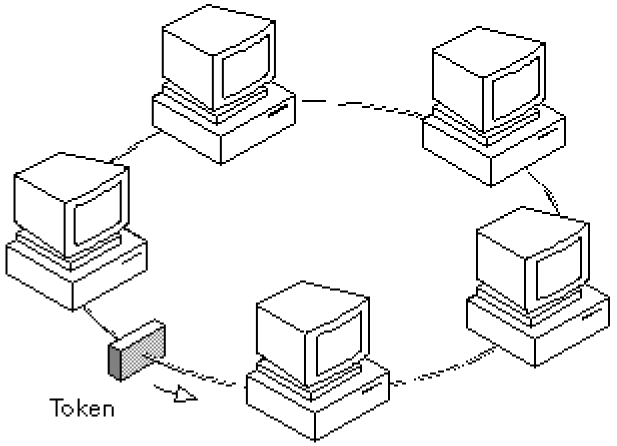
7 color
Reverse video and blink
Graphical display capable

Your Enterprise

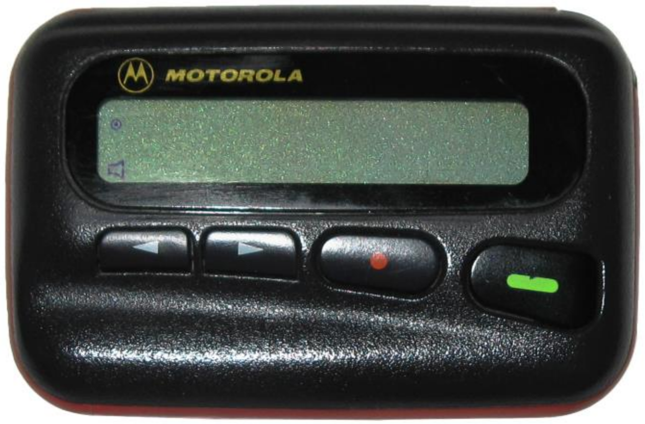


Your Network

9



Modern consumer electronics



Text Messaging and PDA functions become available on cell phones

The Birth of MQ

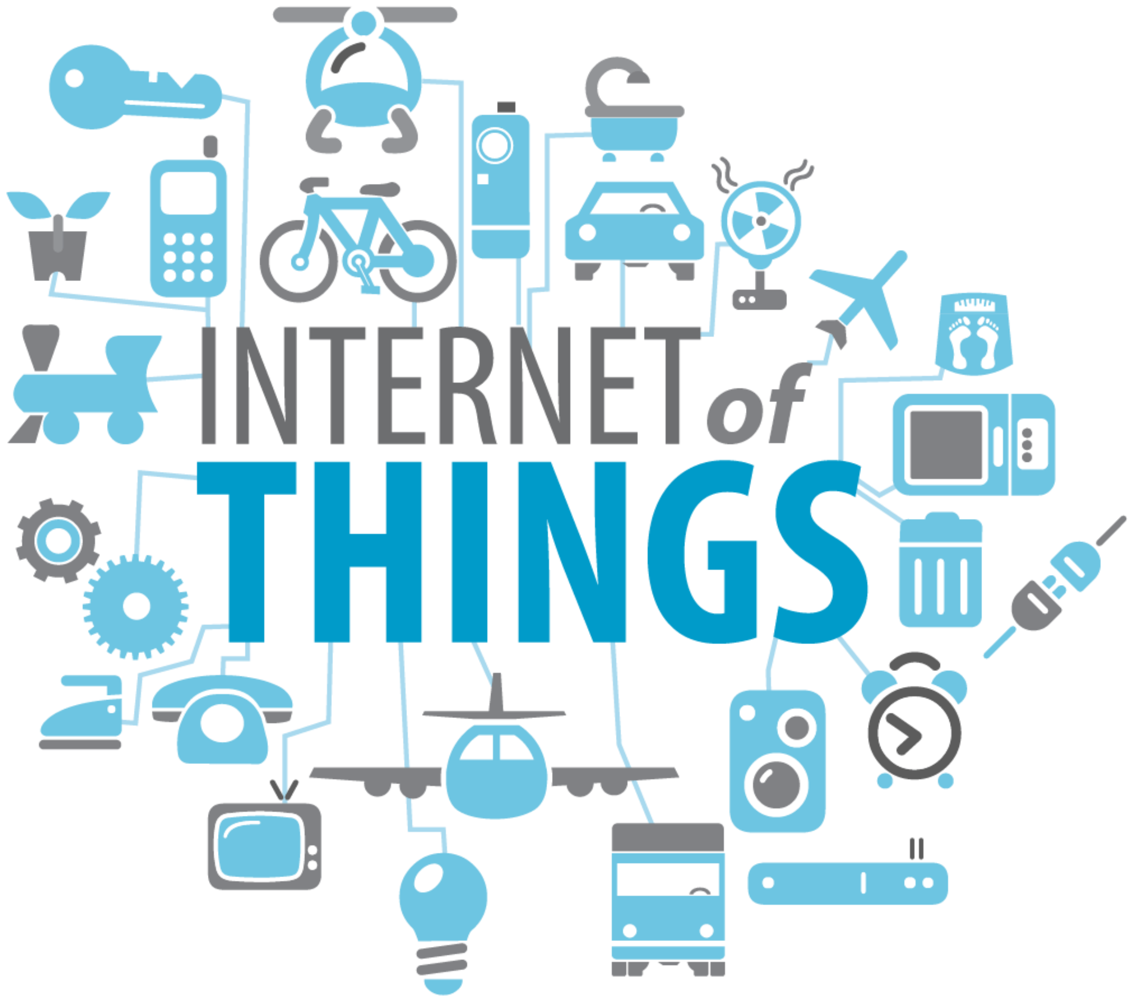
- MQ Solved many of the problems that existed in this environment

1. A consistent API across a number of disparate operating systems
2. Ability to interact between applications using different architectures (EBCIC/ASCII - Big/Little Endian)
3. Asynchronous support for systems and applications that ran at different speeds
4. Independent operations for systems that lost network connectivity
5. Ability to recover from regular system outages
6. Many more…
7. Publish / subscribe & Clustering came later

Your Environment today

Other Legacy Systems

Internet of Things



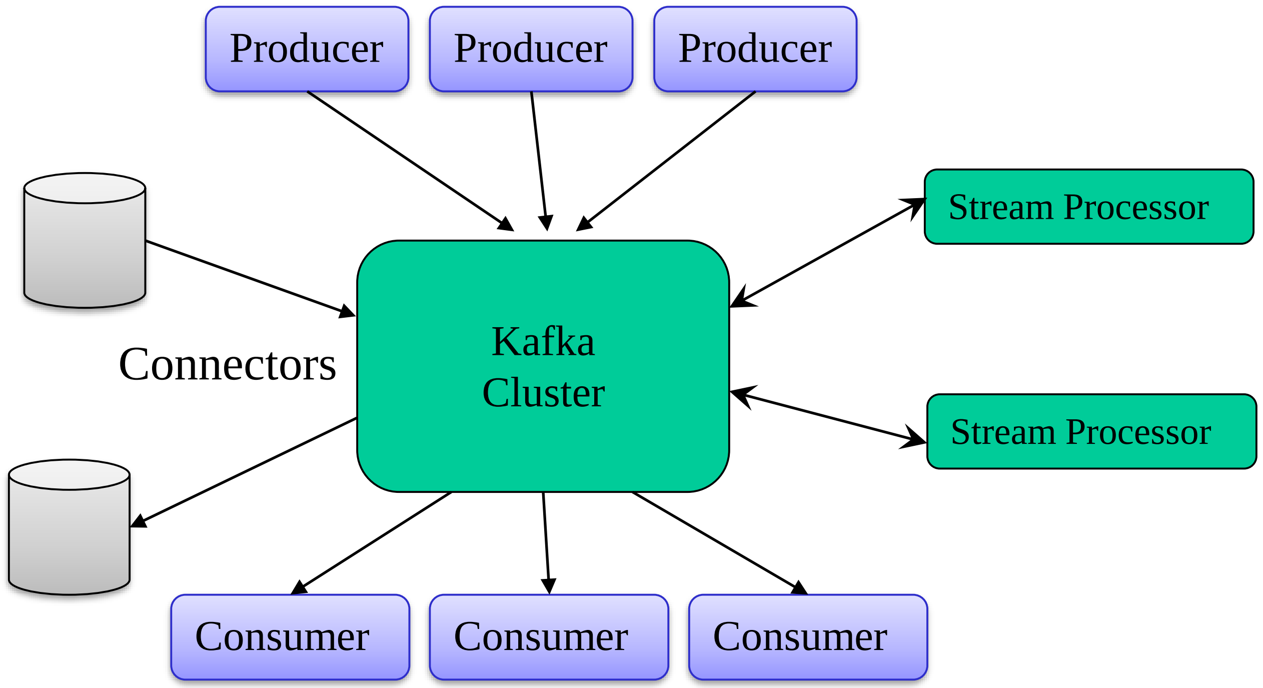
Kafka History and reference

- **Developed by LinkedIn**
- **Wanted a system that was not restricted by the past and exploited technologies commonly available**
- **Key requirements**
  - High speed
  - Fault tolerant
  - Infinitely scalable
  - Distributed access
- **Became open source in 2011 under Apache**
- **https://kafka.apache.org**
- **In the rest of this presentation, we will introduce Kafka concepts which will demonstrate how it achieves these objectives**

BASIC CONCEPTS

Basic constructs

- Producer
- Consumer
- Cluster
- Broker
- Streams
- Connectors



More constructs

- Topics

❖ As you would expect, a unique string to which produces produce "messages" and to which subscribers subscribe

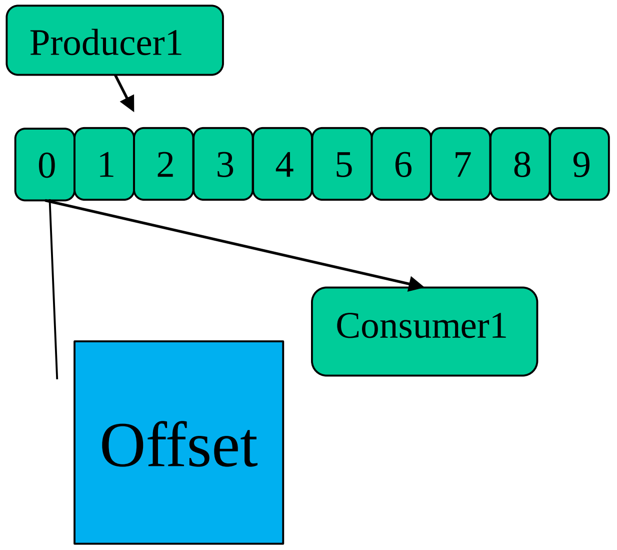
- Logs

❖ The basic construct of Kafka. Messages are persisted as a sequence of items to logs (just as MQ captures transaction logs).

❖ Logs consist of one or more immutable sequence of "messages"

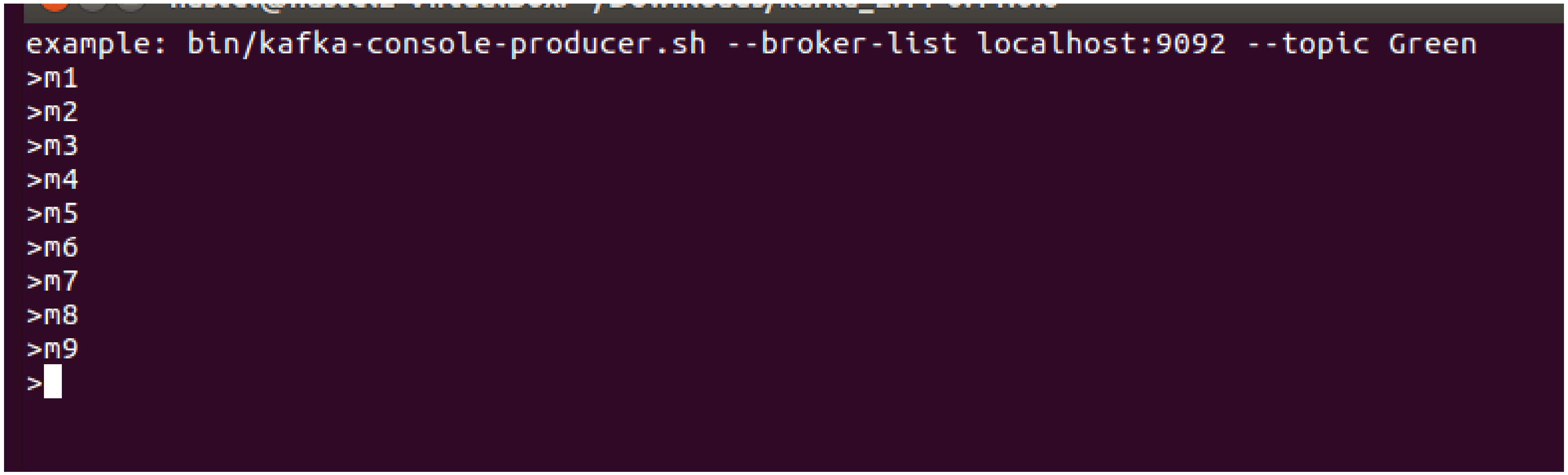
Putting it Together in a Simple Example

Topic: “Green”



INTRODUCTION BY EXAMPLE

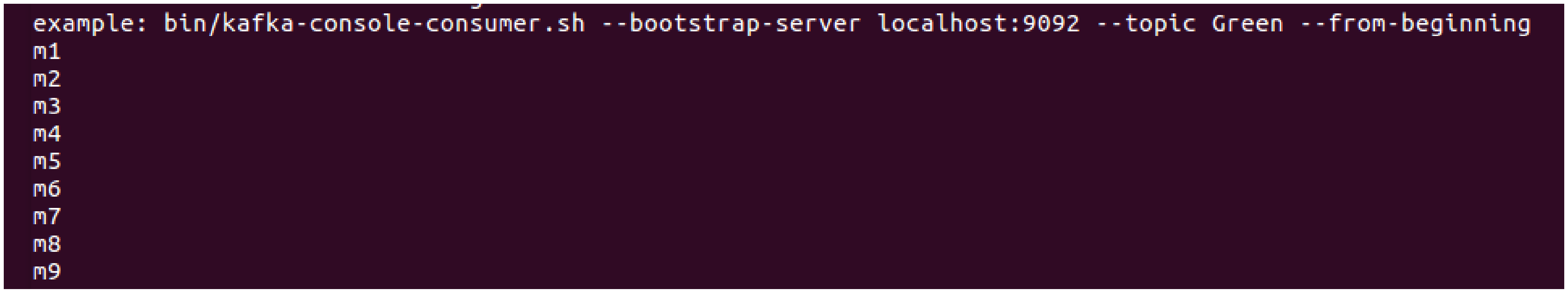
Simple Example - Producer



```
example: bin/kafka-console-producer.sh --broker-list localhost:9092 --topic Green
>m1
>m2
>m3
>m4
>m5
>m6
>m7
>m8
>m9
>
```

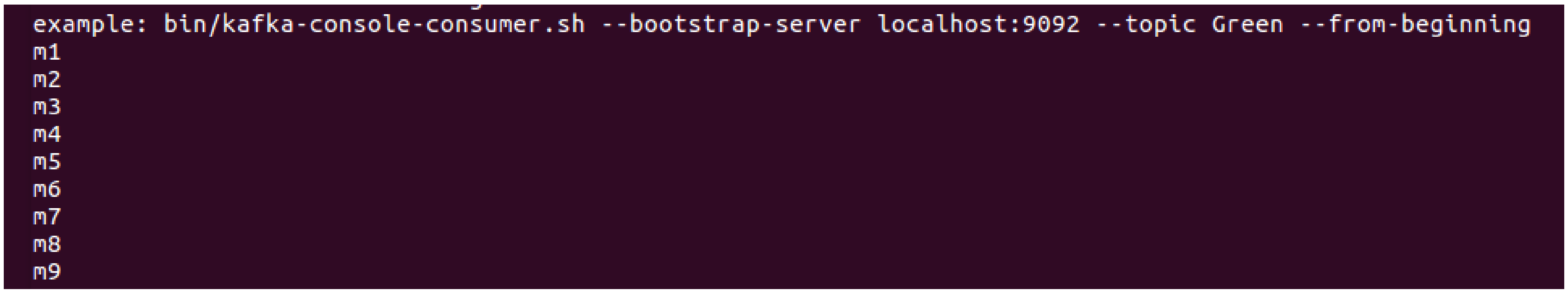
- **Simple console based console example putting to topic Green**

Simple Example – Consumer 1


```
example: bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Green --from-beginning
m1
m2
m3
m4
m5
m6
m7
m8
m9
```

- **A sample console consumer reads the messages**

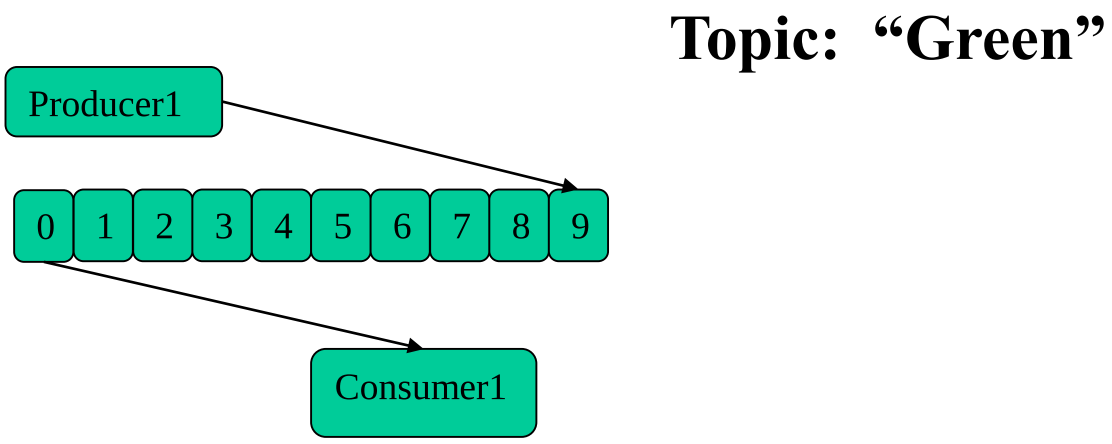
Simple Example – Consumer 2

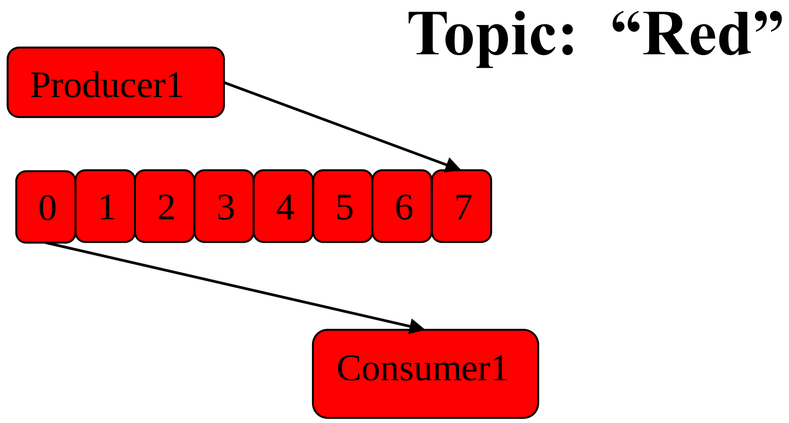


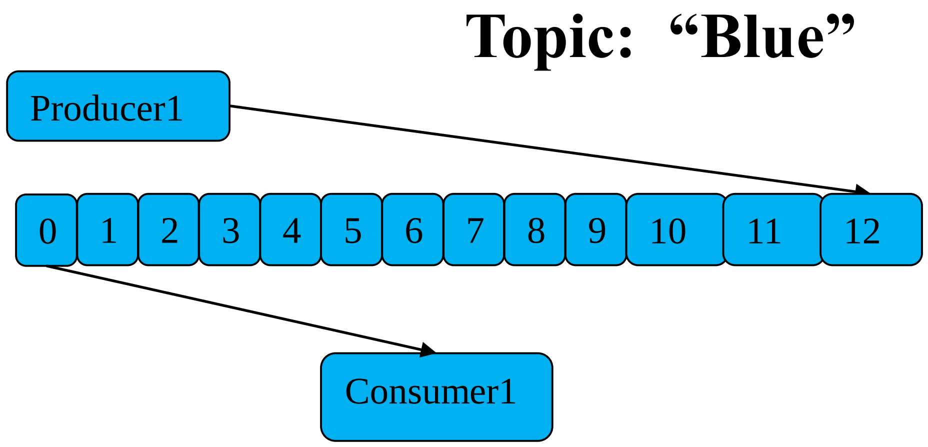
```
example: bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Green --from-beginning
m1
m2
m3
m4
m5
m6
m7
m8
m9
```

- **Now we start a 2nd consumer for Green**
- **Did you expect this result? Probably not if you are thinking of IBM MQ which only publishes messages for that topic if a subscription exists when published**
- **The messages in Kafka do not go away simply because they were read by a consumer like a queue (more later)**

Topics and Logs Revisited

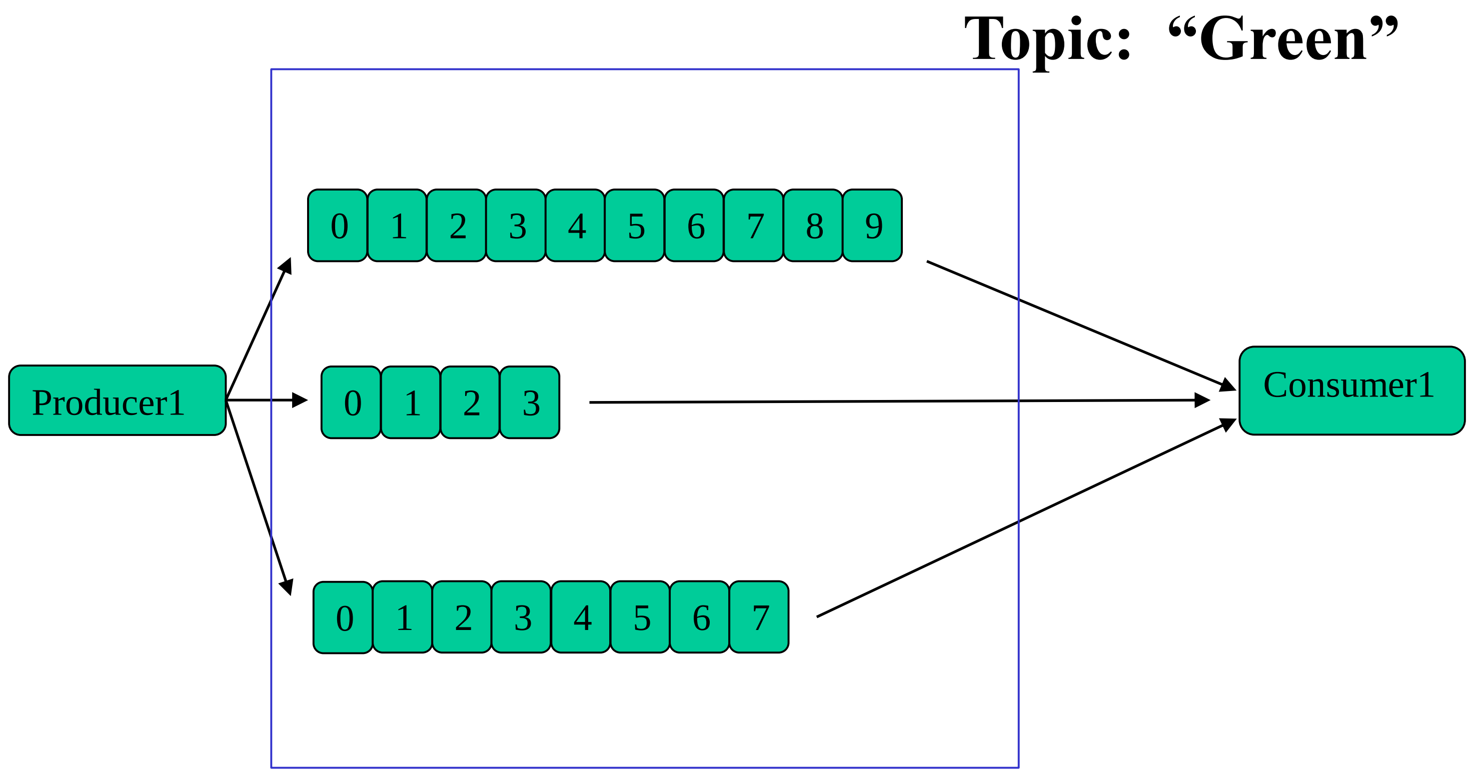






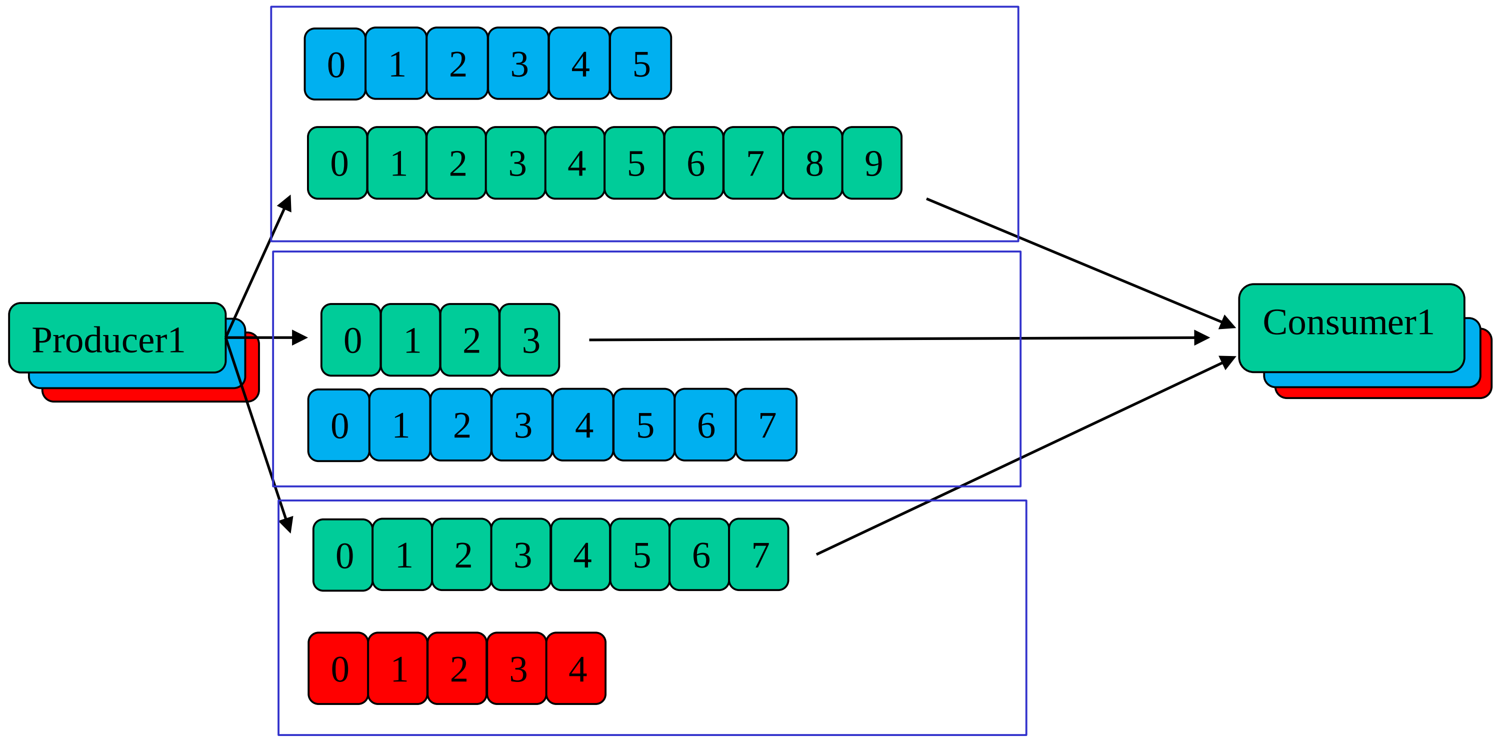
EXPANDED CONCEPTS

Partitions



- Partitions divide the topic storage across multiple logs

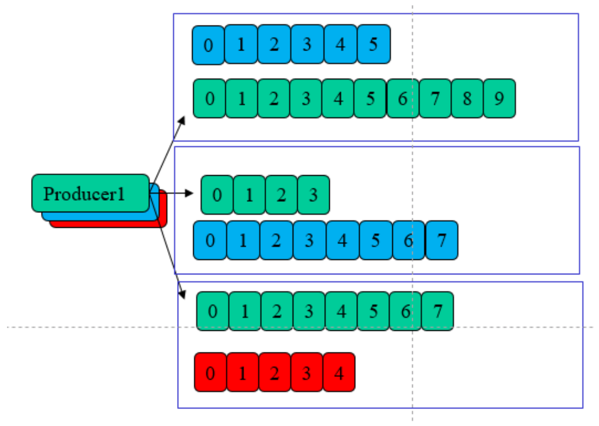
Distributed Partitions



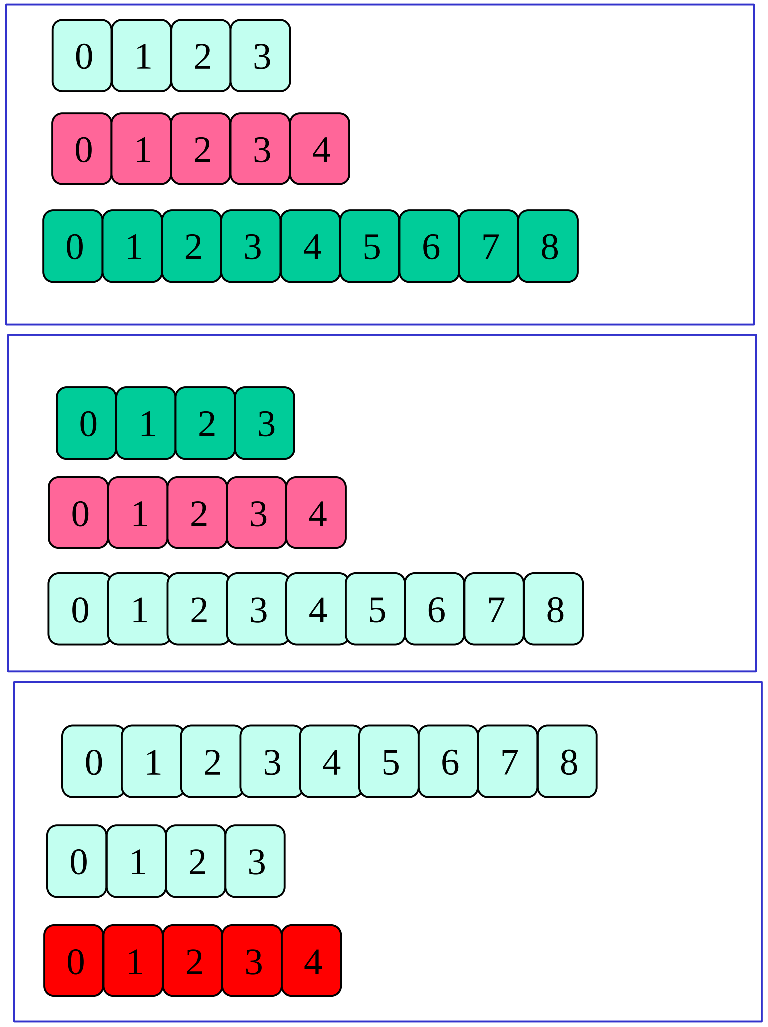
- **Partitions divide the topic storage across multiple logs**

Partition Assignment

- ❖ Default Partitioner
- ▪ Kafka partition specified
- ▪ If key specified, hash of the key
- ▪ Round robin assignment
- ❖ Custom Partitioner
- ▪ Assign based on various criteria

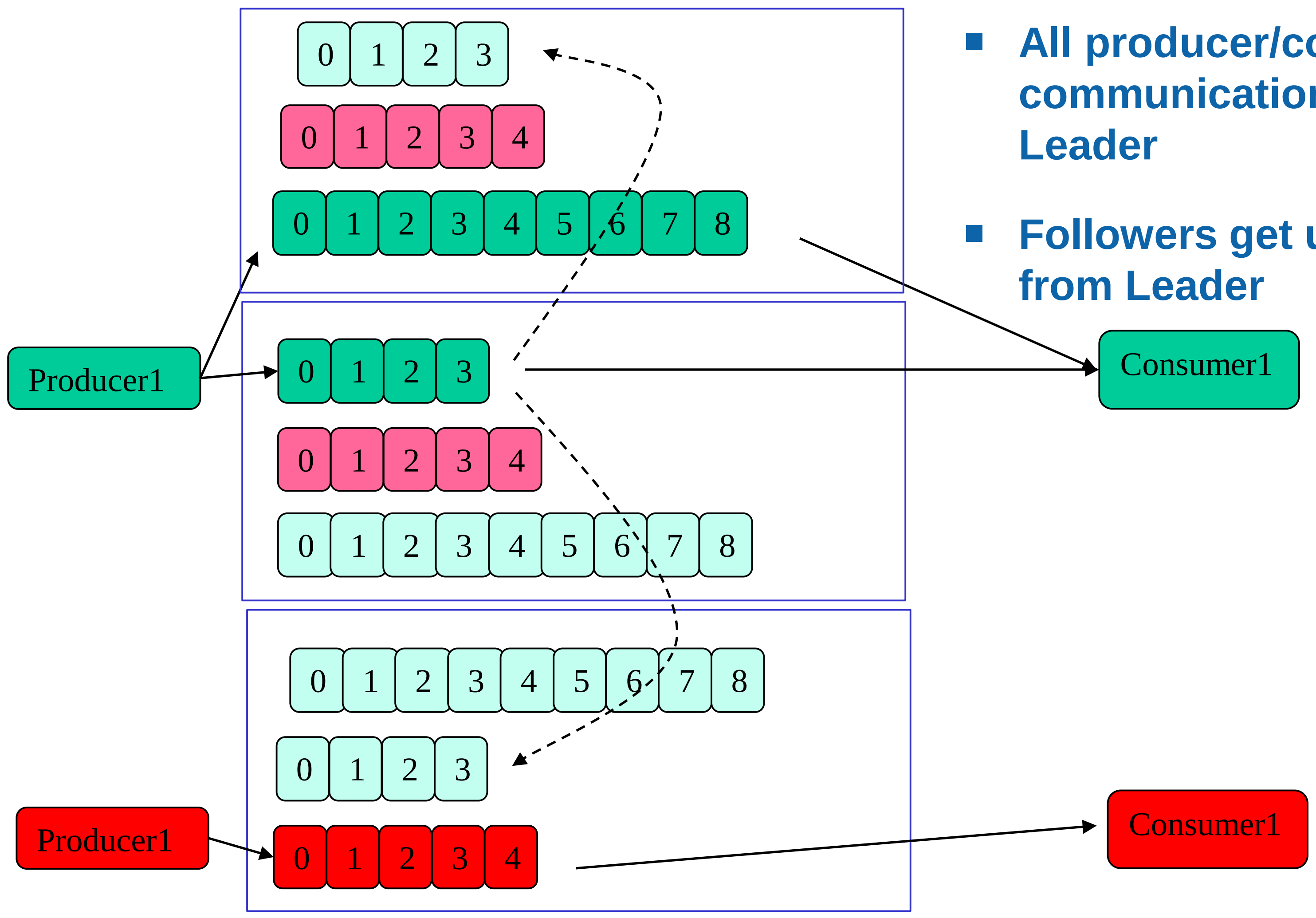


Replicated Partitions



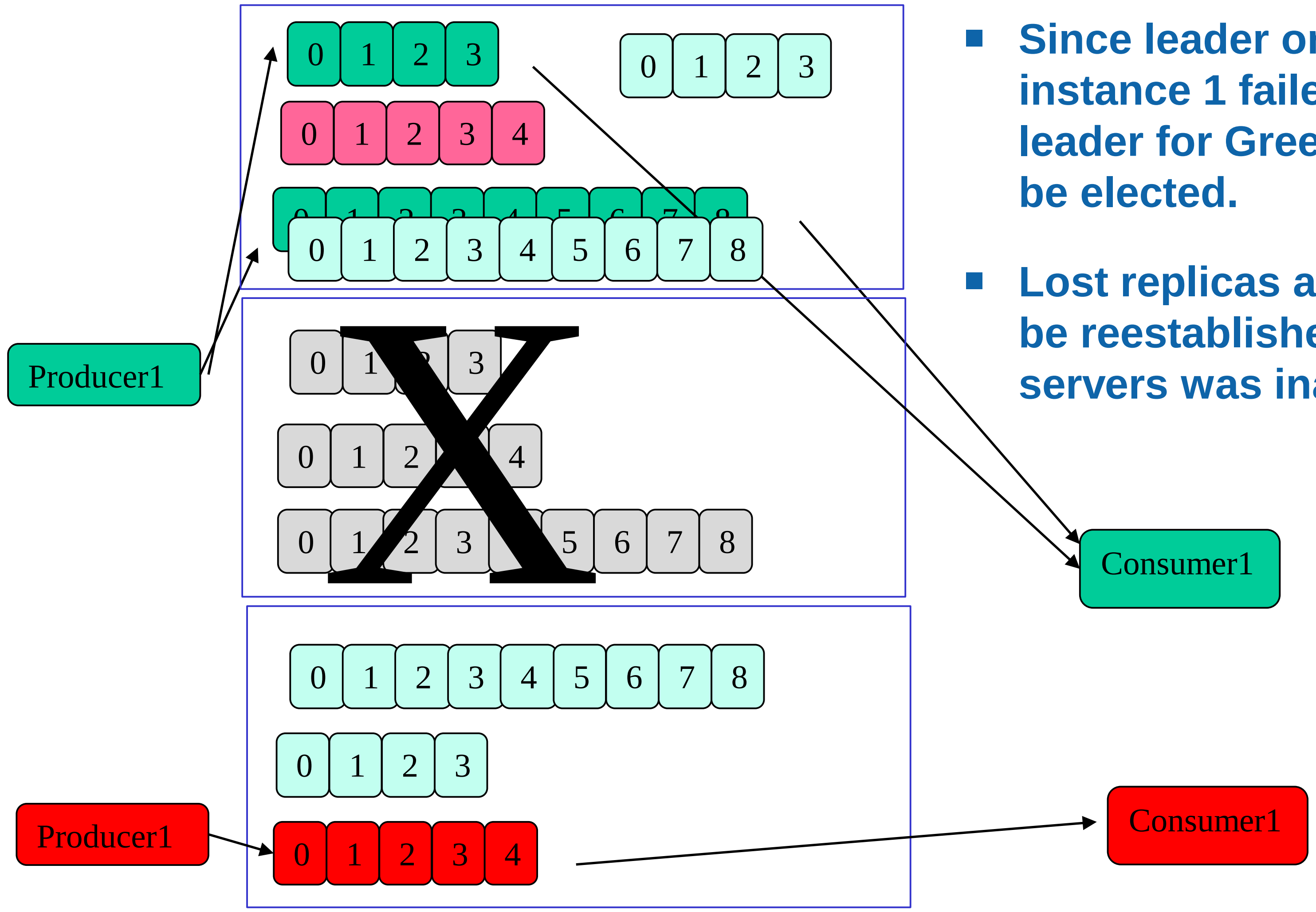
- **Replication Factor determines the numbers of copies of each partition created**
- **Failure of (replication-factor – 1) nodes does not result in loss of data**
- **Sufficient server instances are required to provide segregation of instances (in this example, 3 partitions on 3 brokers leaves limited room for failover)**

Leaders and Followers



- All producer/consumer communication is via the Leader
- Followers get updates from Leader

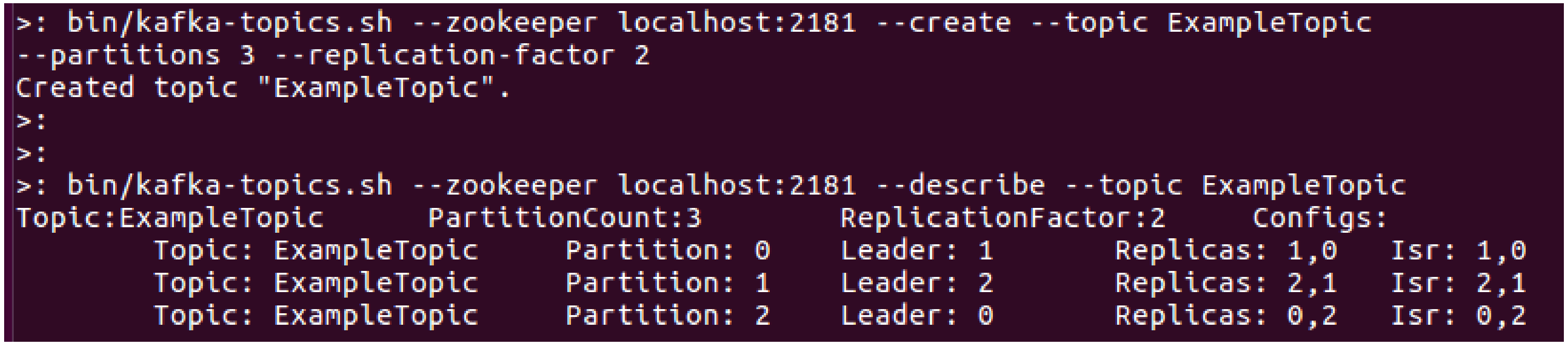
Replication Failover



- **Since leader on Green instance 1 failed, new leader for Green has to be elected.**
- **Lost replicas also need to be reestablished (why 3 servers was inadequate)**

Topic Describe

- **Create a topic with 3 partitions and a replication factor of 2**



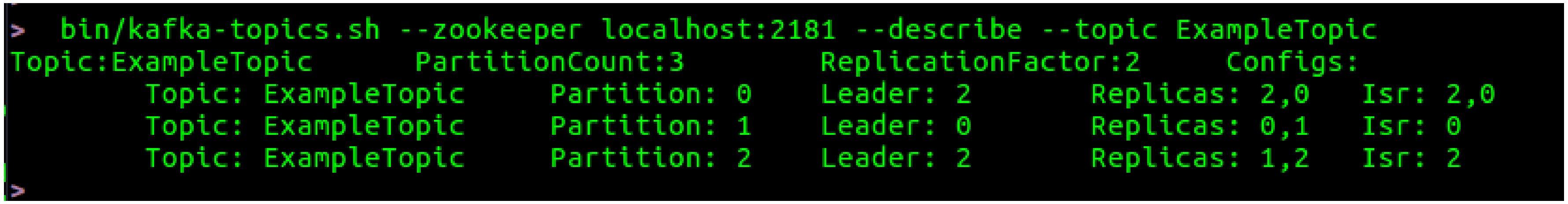
```
>: bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic ExampleTopic
--partitions 3 --replication-factor 2
Created topic "ExampleTopic".
>:
>:
>: bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic ExampleTopic
Topic:ExampleTopic      PartitionCount:3        ReplicationFactor:2     Configs:
        Topic: ExampleTopic     Partition: 0    Leader: 1       Replicas: 1,0   Isr: 1,0
        Topic: ExampleTopic     Partition: 1    Leader: 2       Replicas: 2,1   Isr: 2,1
        Topic: ExampleTopic     Partition: 2    Leader: 0       Replicas: 0,2   Isr: 0,2
```

3 partitions

Leaders

Replicas (leader and followers)

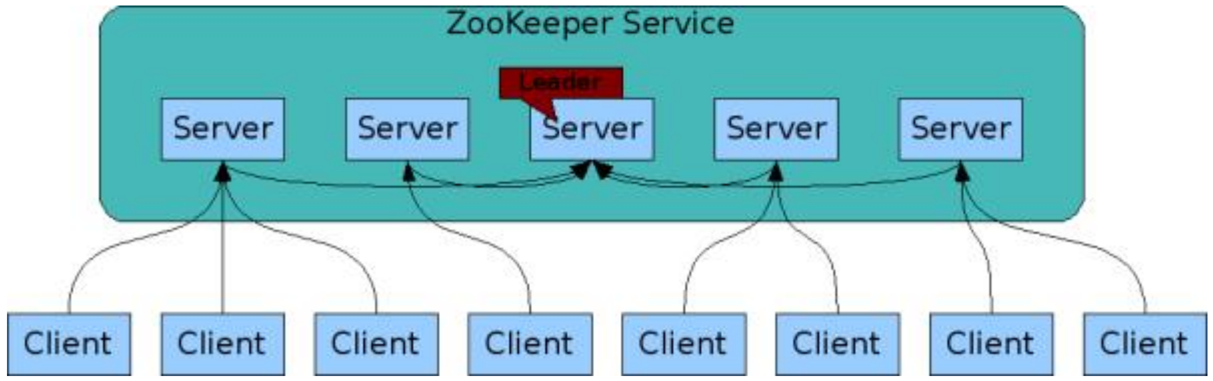
In Sync Replicas



```
>  bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic ExampleTopic
Topic:ExampleTopic      PartitionCount:3        ReplicationFactor:2     Configs:
        Topic: ExampleTopic     Partition: 0    Leader: 2       Replicas: 2,0   Isr: 2,0
        Topic: ExampleTopic     Partition: 1    Leader: 0       Replicas: 0,1   Isr: 0
        Topic: ExampleTopic     Partition: 2    Leader: 2       Replicas: 1,2   Isr: 2
>
```

Behind the Scenes

- How is possible to coordinate all of this across the kafka brokers?
- Apache Zookeeper
- A distributed coordination service
- Coordinates the state of Kafka for the brokers, publishers and consumers

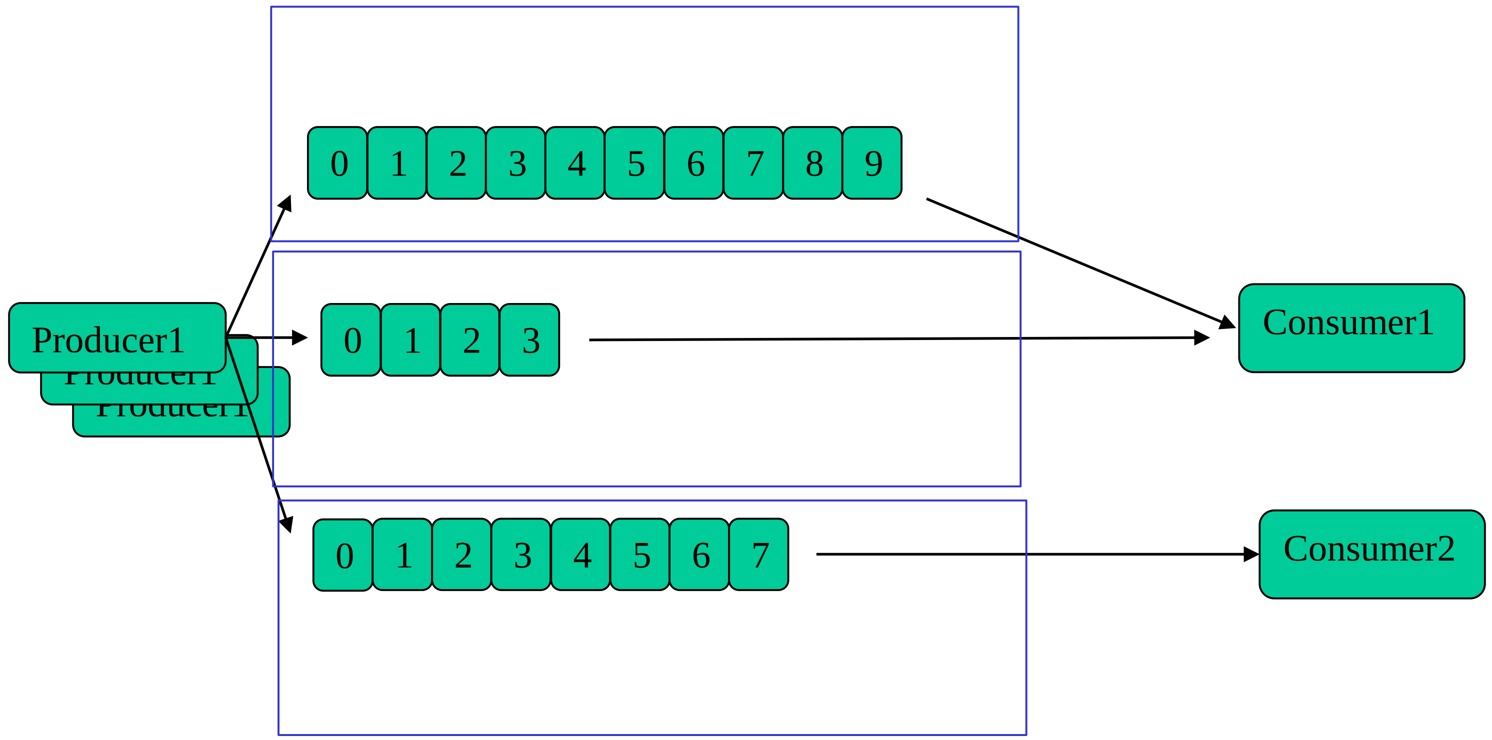


SCALABILITY

Scalability options

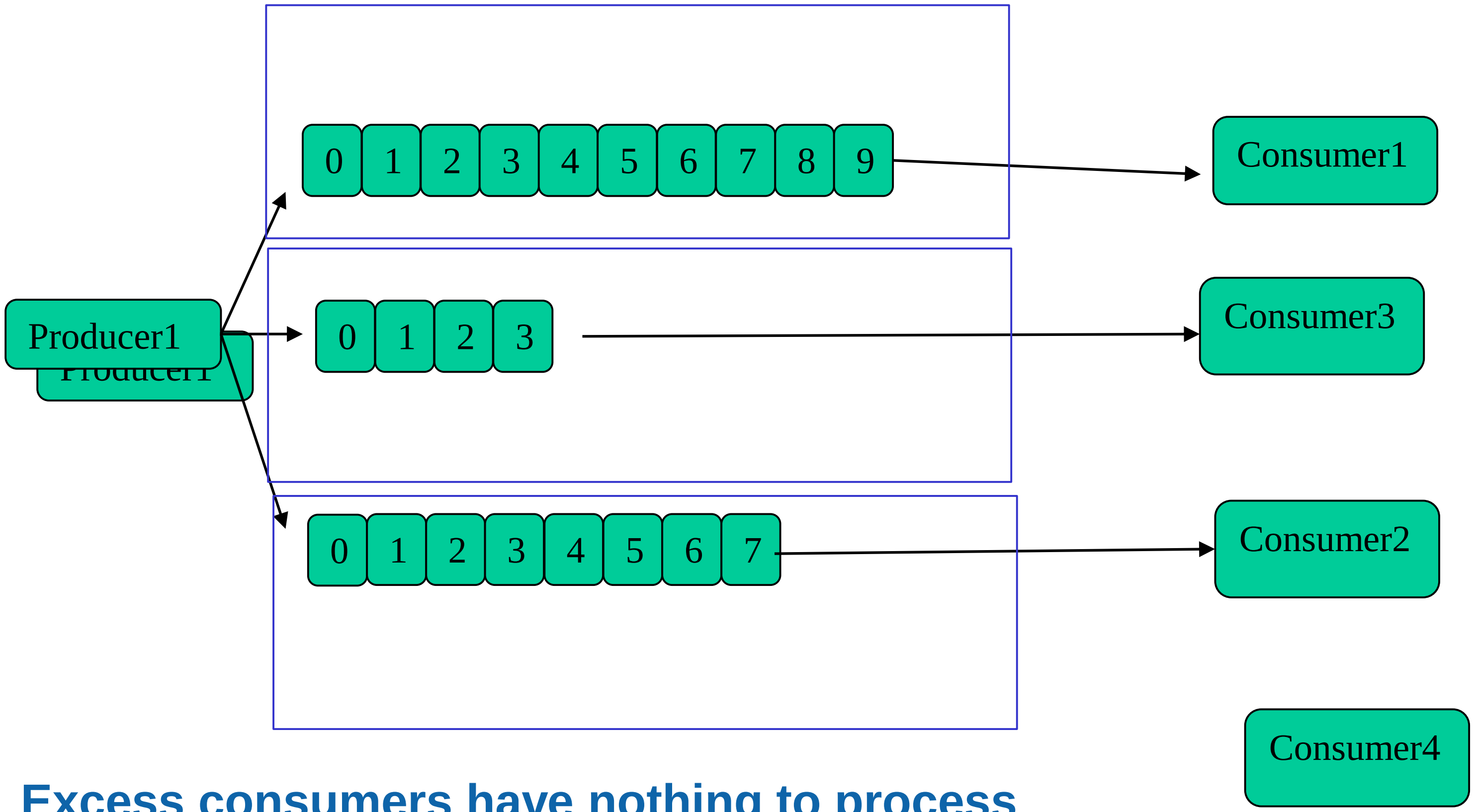
- Increasing consumers and producers
- Consumer groups
- Adding Partitions
- Message Key partitioning
- Adding Topics
- Streaming processes

Consumer Groups



- **Up to 1 consumer per partition in the group**

Consumer Groups



- **Excess consumers have nothing to process**

RUNTIME CONSIDERATIONS

Data Retention

- Retention by Time
  - log.retention.(hours|minutes|ms)
  - Default is 1 week
- Retention by Partition Size
  - log.retention.bytes
  - Applied per partition
- Applies to the log segment

Producer Considerations

- **Fire and Forget**
  - Send without any confirmation
- **Asynch Send**
  - Max inflight
- **Synchronous**
  - All writes complete with verification
- **Replication Acknowledgement level**
- **(Automatic) retry logic**
- **Partition Assignment**

Consumer Considerations

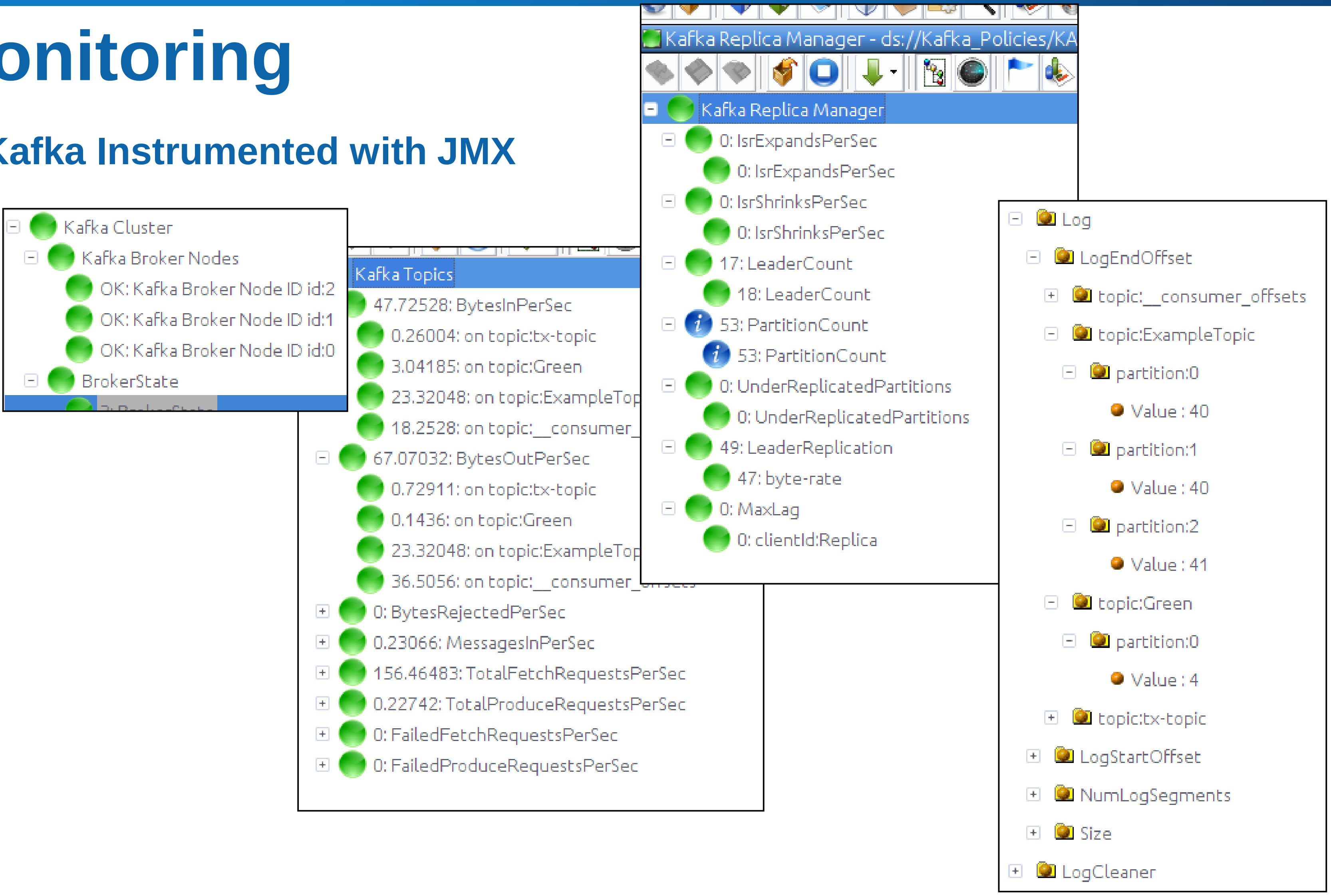
- Consumer Groups
- Commit Strategy
- Committed Offset (the last message committed by the consumer)
- Batch size
- Rebalancing
- Timely Interaction with Kafka

General Considerations

- Log Retention
  - "Kafka's performance is effectively constant with respect to data size so storing data for a long time is not a problem“
- Latency between producers and consumers
- Brokers health (running?)
- Distributed server health
- Runaway producers
- Other components in the stack
- “Poison” messages
- Partitioner Effectiveness
- Rebalance

Monitoring

- **Kafka Instrumented with JMX**



```
Kafka Cluster
  Kafka Broker Nodes
    OK: Kafka Broker Node ID id:2
    OK: Kafka Broker Node ID id:1
    OK: Kafka Broker Node ID id:0
  BrokerState
```

```
Kafka Topics
47.72528: BytesInPerSec
  0.26004: on topic:tx-topic
  3.04185: on topic:Green
  23.32048: on topic:ExampleTop
  18.2528: on topic:__consumer_
67.07032: BytesOutPerSec
  0.72911: on topic:tx-topic
  0.1436: on topic:Green
  23.32048: on topic:ExampleTop
  36.5056: on topic:__consumer_offsets
0: BytesRejectedPerSec
0.23066: MessagesInPerSec
156.46483: TotalFetchRequestsPerSec
0.22742: TotalProduceRequestsPerSec
0: FailedFetchRequestsPerSec
0: FailedProduceRequestsPerSec
```

```
Kafka Replica Manager - ds://Kafka_Policies/KA
Kafka Replica Manager
  0: IsrExpandsPerSec
    0: IsrExpandsPerSec
  0: IsrShrinksPerSec
    0: IsrShrinksPerSec
  17: LeaderCount
    18: LeaderCount
  53: PartitionCount
    53: PartitionCount
  0: UnderReplicatedPartitions
    0: UnderReplicatedPartitions
  49: LeaderReplication
    47: byte-rate
  0: MaxLag
    0: clientId:Replica
```

```
Log
  LogEndOffset
    topic:__consumer_offsets
    topic:ExampleTopic
      partition:0
        Value : 40
      partition:1
        Value : 40
      partition:2
        Value : 41
    topic:Green
      partition:0
        Value : 4
    topic:tx-topic
  LogStartOffset
  NumLogSegments
  Size
LogCleaner
```

“SOME” CODE

Simple Producer

```
String topicName = “MyKafkaTopic";
String key = “KeyToUse";
String value = “Some Text to Send";

Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092,localhost:9093");
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);

ProducerRecord<String, String> record =
         new ProducerRecord<> (topicName,  key,  value);

producer.send(record);

producer.close();
```

KAFKA VERSUS MQ

MQ to Kafka Summary

- No MQI – just code
- Conditions MQ handles out of the box may require coding
- Kafka has no “queues”, just publish and subscribe
- Publishes with no subscribers still publish
- ‘Guaranteed’ messaging
- failover for lost nodes (including file systems)
- resource consumption models
- Less “admin” involvement

KAFKA AND IBM

Kafka Integration

- **IIB (10.0.0.7) Kafka consumer and producer nodes**
- **MQ Connector**
  - https://github.com/ibm-messaging/kafka-connect-mq-source
- **MQ Bridge (Bluemix)**

SUMMARY

So why do you care?

- Kafka is an alternative to MQ that is likely already in use in your organization
- Kafka is part of the middleware space and needs to be managed
- Kafka can and will integrate with your existing applications creating a hybrid messaging environment
- Kafka has advantages to MQ for some message patterns but may not work for all
- Since it is messaging, there is an expectation you will be the expert…

Questions & Answers