Apache Kafka

Daniel Hinojosa

About Me...

Daniel Hinojosa
Programmer, Consultant, Trainer

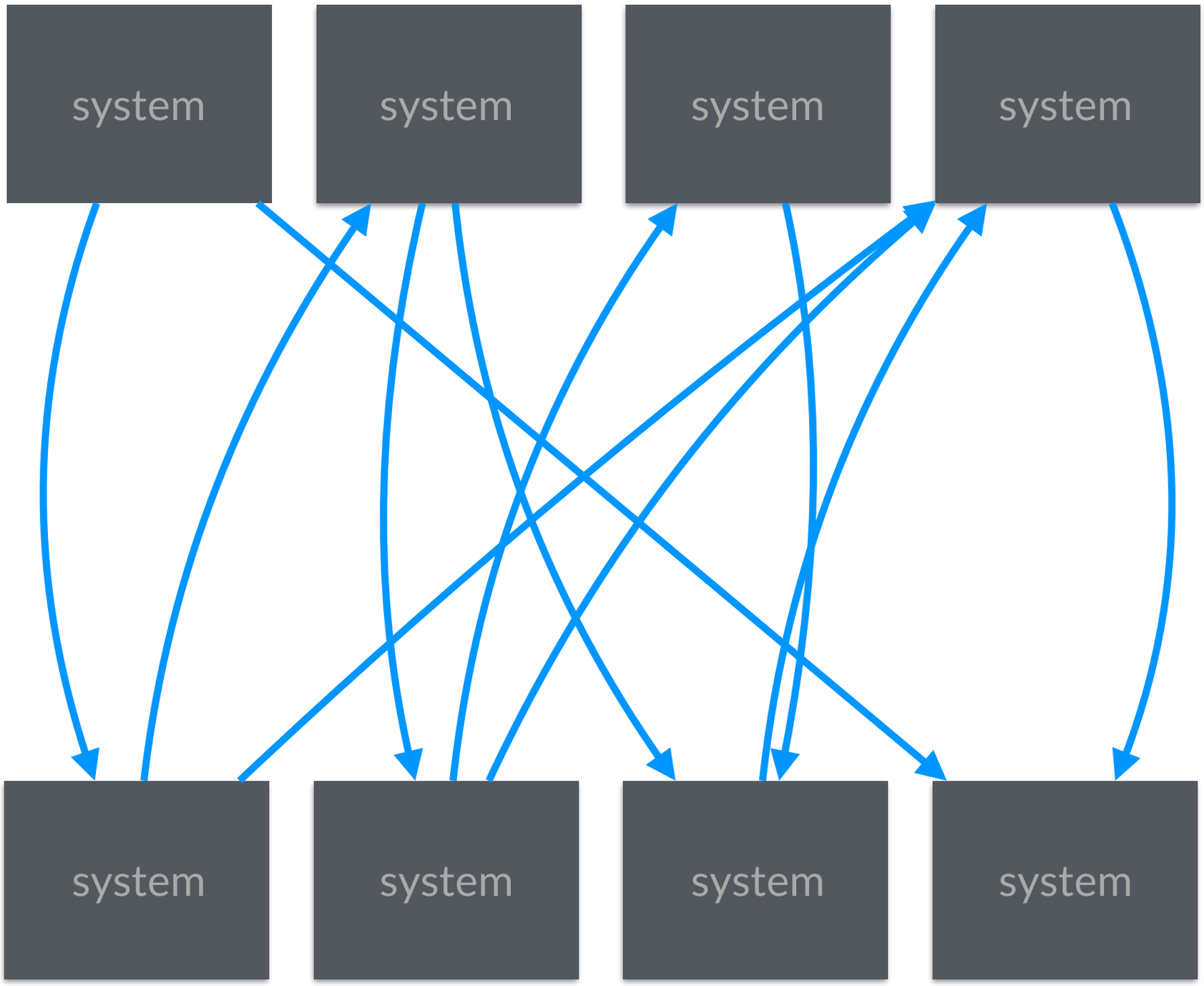
Testing in Scala (Book)
Beginning Scala Programming (Video)
Scala Beyond the Basics (Video)

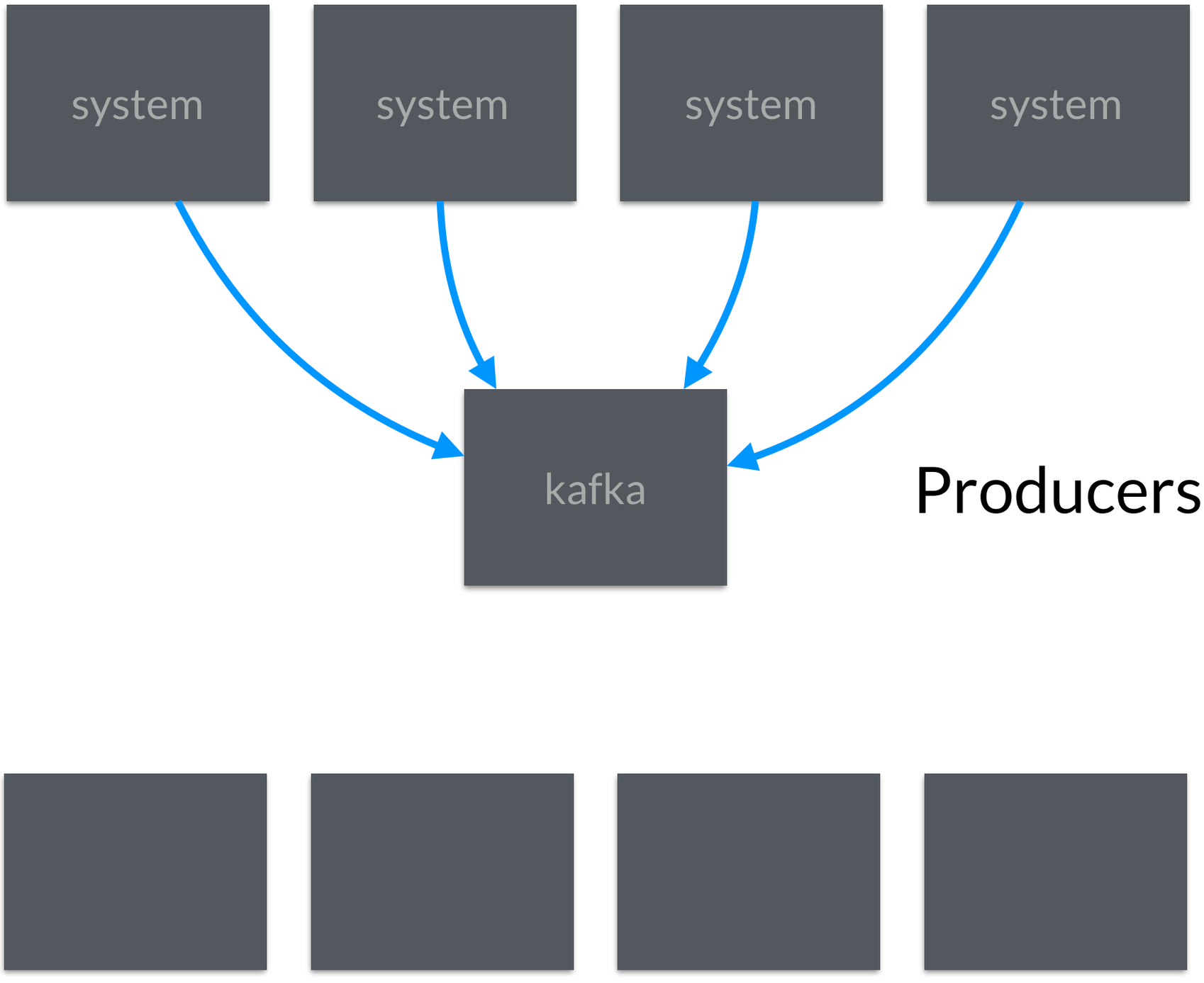
Contact:
dhinojosa@evolutionnext.com
@dhinojosa

Slides and Code Available @:

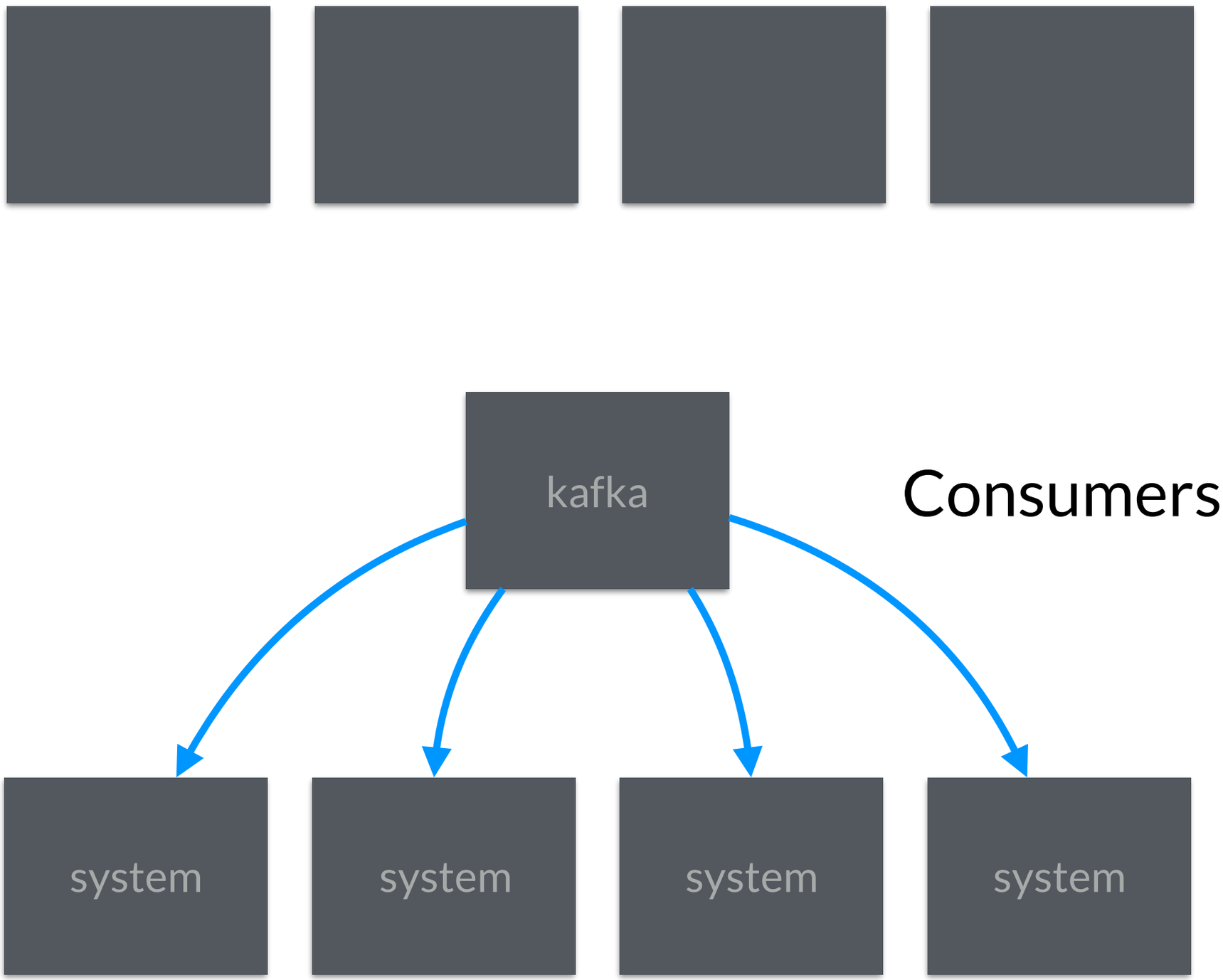
https://github.com/dhinojosa/kafka-study

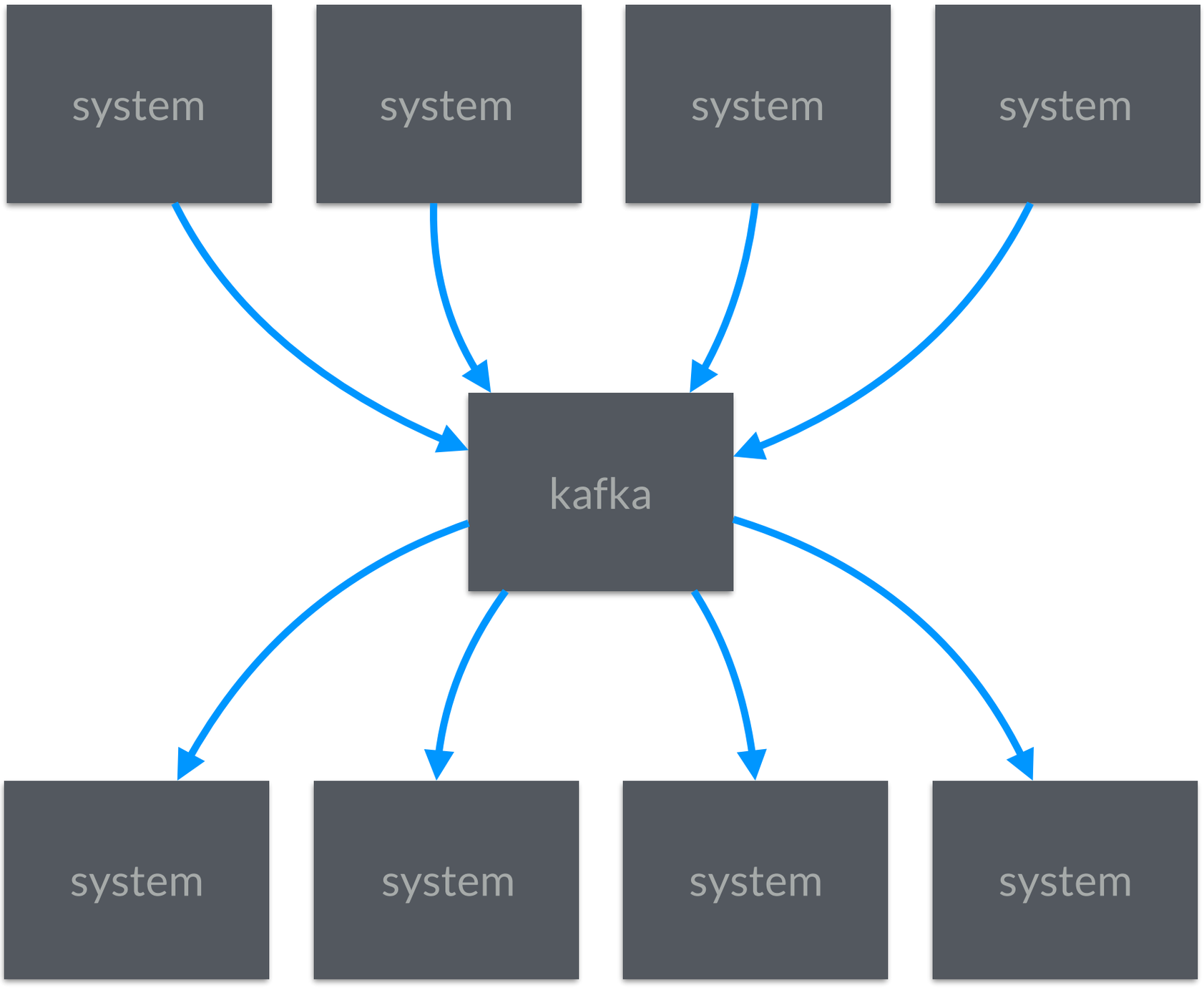
Kafka Introduction

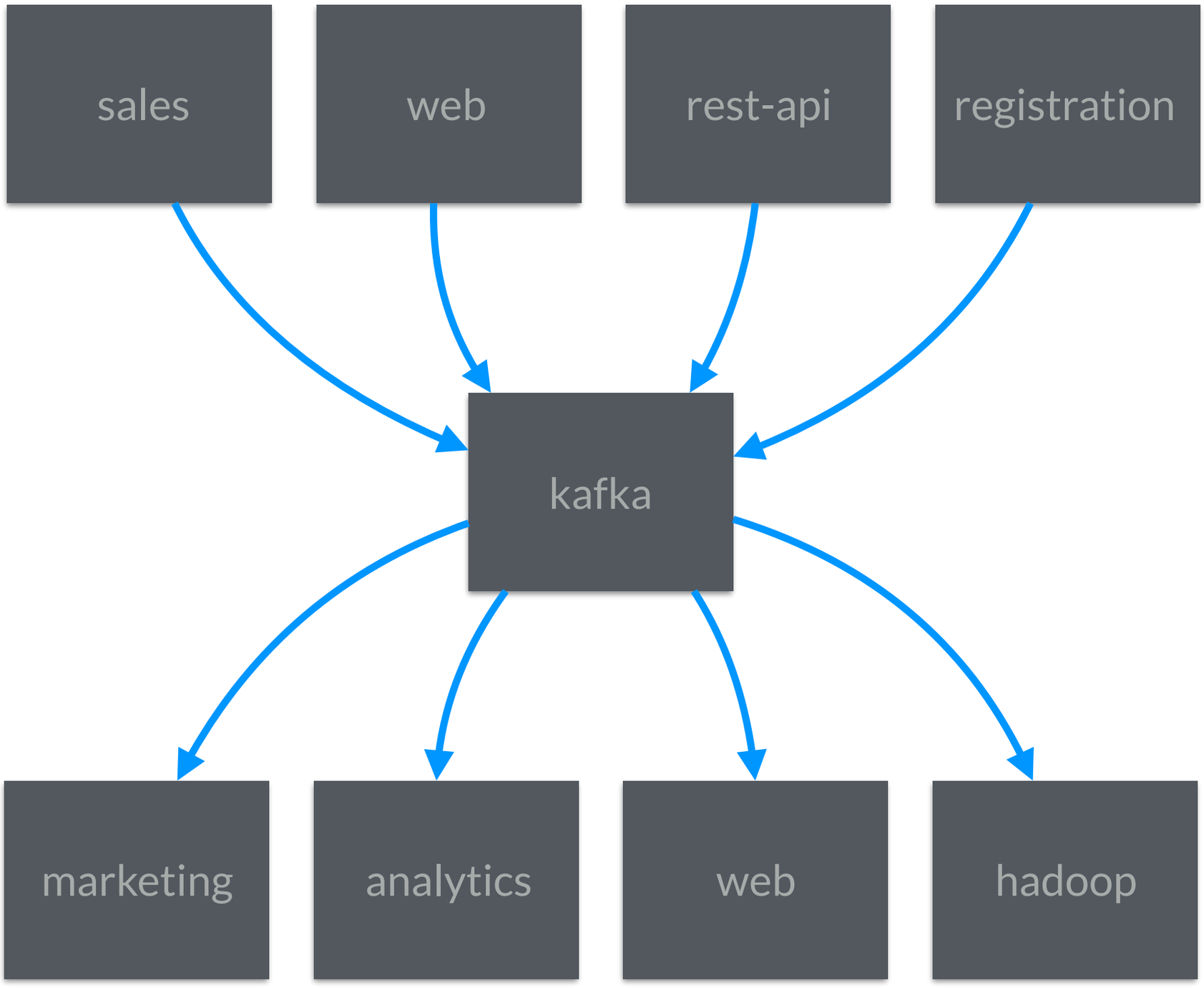

system
system
system
system
system
system
system
system



Producers


kafka
Consumers
system
system
system
system


system
system
system
system
kafka
system
system
system
system



Oversimplied: A Producer can be a Consumer

About Kafka

Handles millions of messages per seconds, high throughput, high volume

Distributed and Replicated Commit-log

Real Time Data Processing

Stream processing

How messages are produced

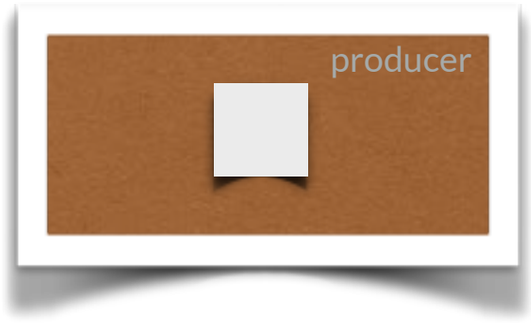
kafka broker: 0

Partition 0:

[0] [1] [2] [3] [4]

topic-a

producer



Retention: The **data is temporary**

How messages are consumed

consumer-1_offset

kafka broker: 0

Partition 0:

[0] [1] [2]

topic-a

consumer-1

consumer-1_offset

kafka broker: 0

Partition 0:

[0] [1] [2] [3] [4]

topic-a

consumer-1

from-beginning

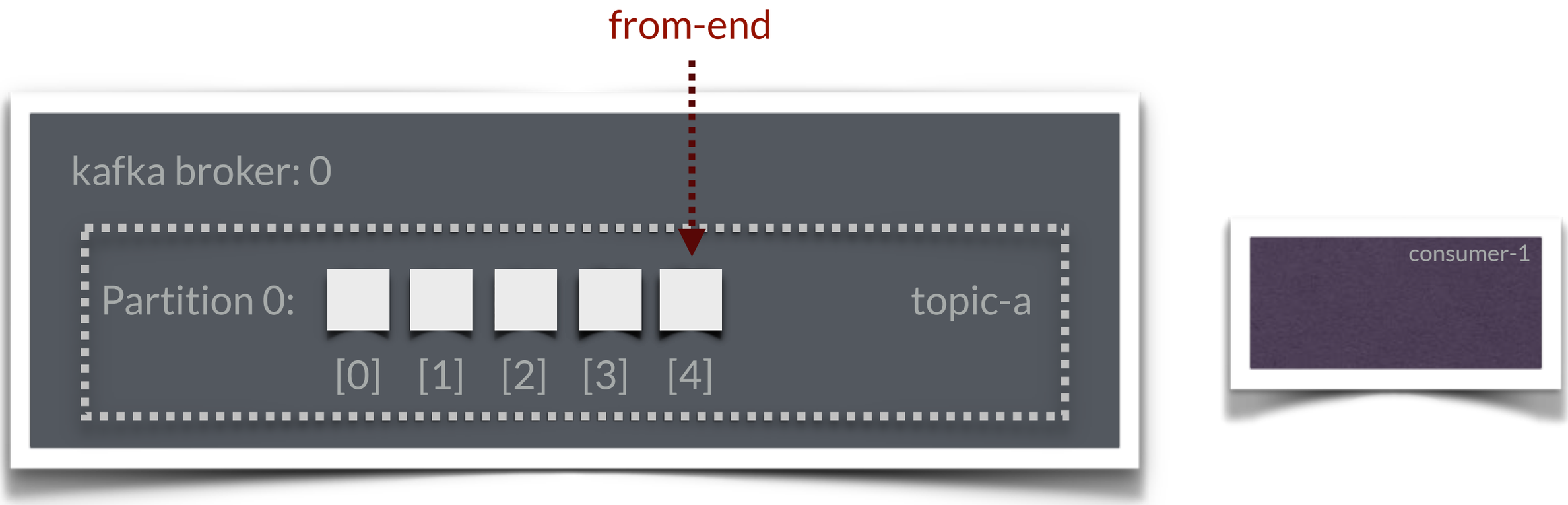
kafka broker: 0

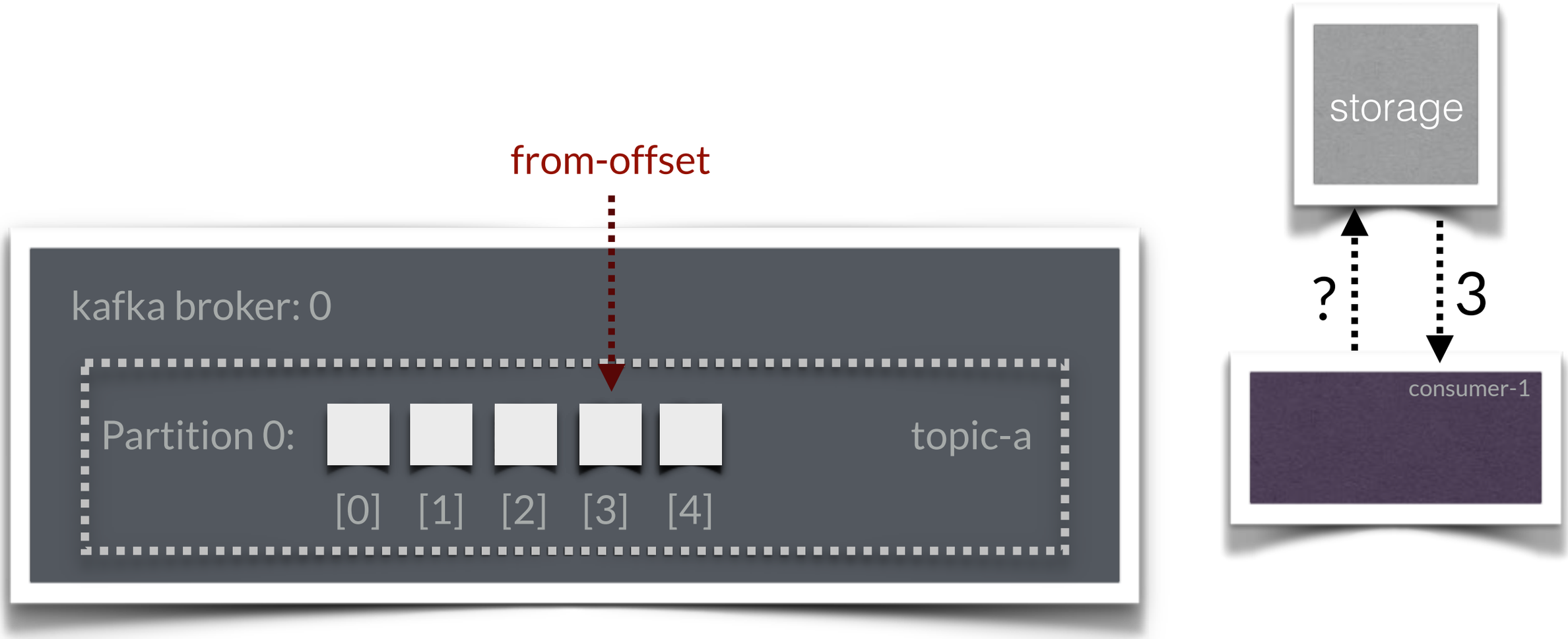
Partition 0:

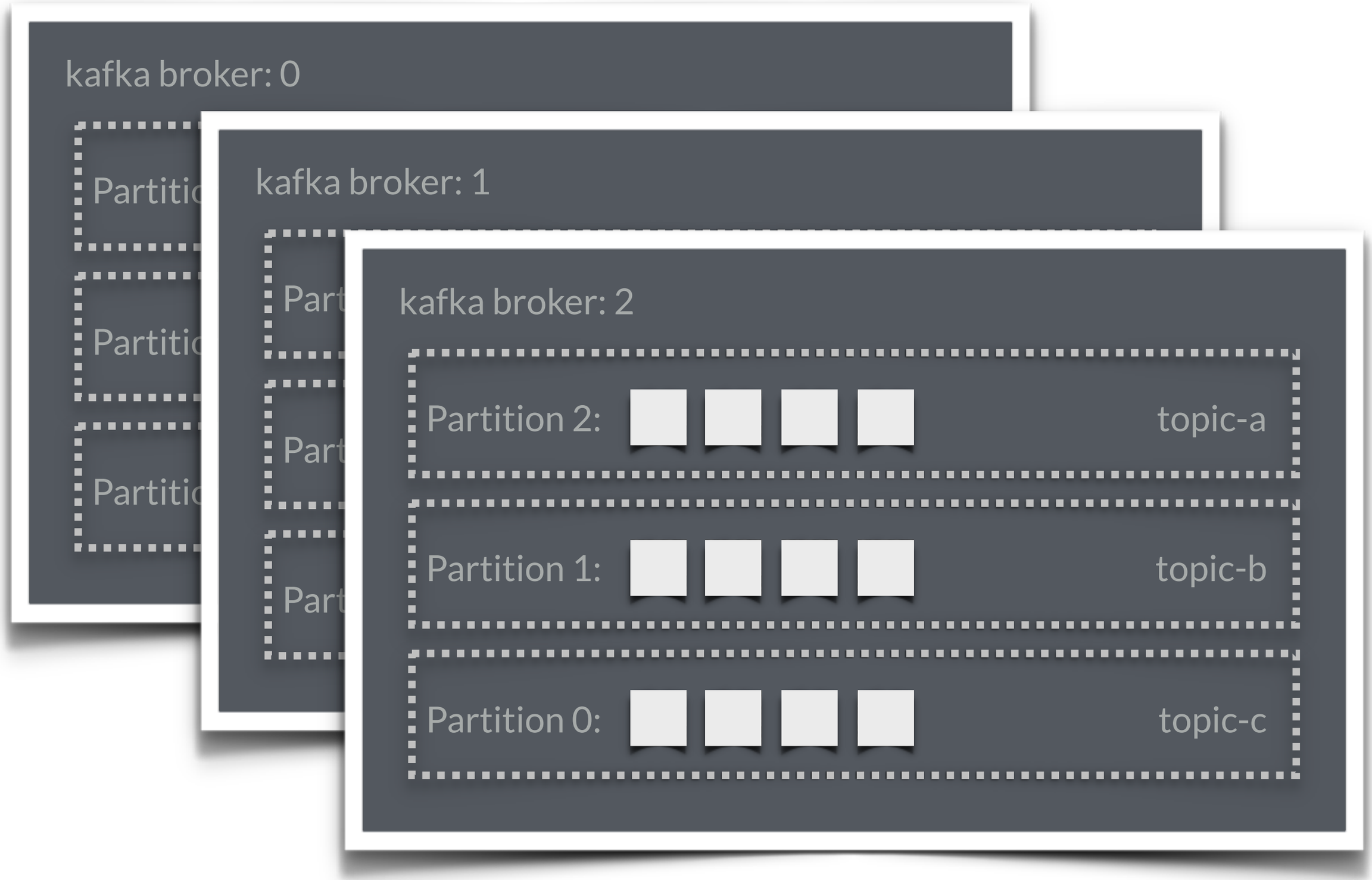
[0] [1] [2] [3] [4]

topic-a

consumer-1


from-end
kafka broker: 0
Partition 0:
[0]
[1]
[2]
[3]
[4]
topic-a
consumer-1
consumer-1


from-offset
kafka broker: 0
Partition 0:
topic-a
[0]
[1]
[2]
[3]
[4]
storage
?
3
consumer-1



Each partition is on a different broker,
therefore a single topic is scaled

kafka b

Partit

Partit

Partit

Partit

kafka b

Parti

Parti

Parti

Parti

kafka broker: 2

Partition 0: topic-a

Partition 1: topic-a

Partition 2: topic-a

Partition 0: topic-b

Partition 1: topic-b

Partition 2: topic-b

Partition 0: topic-c

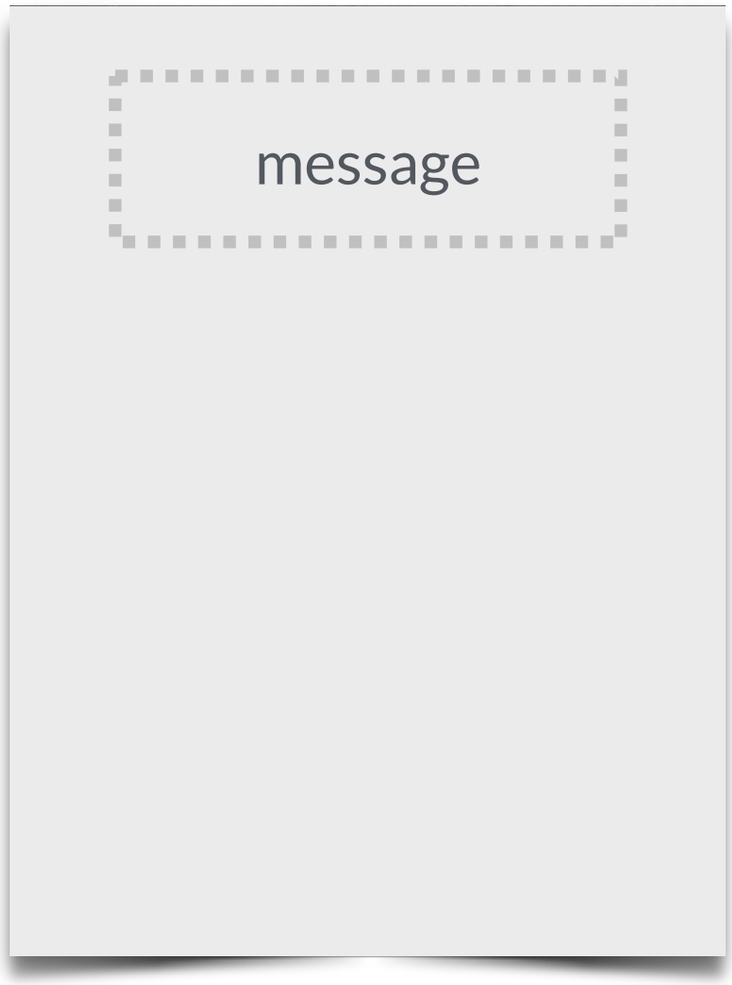
Partition 1: topic-c

Partition 2: topic-c

What is a message?

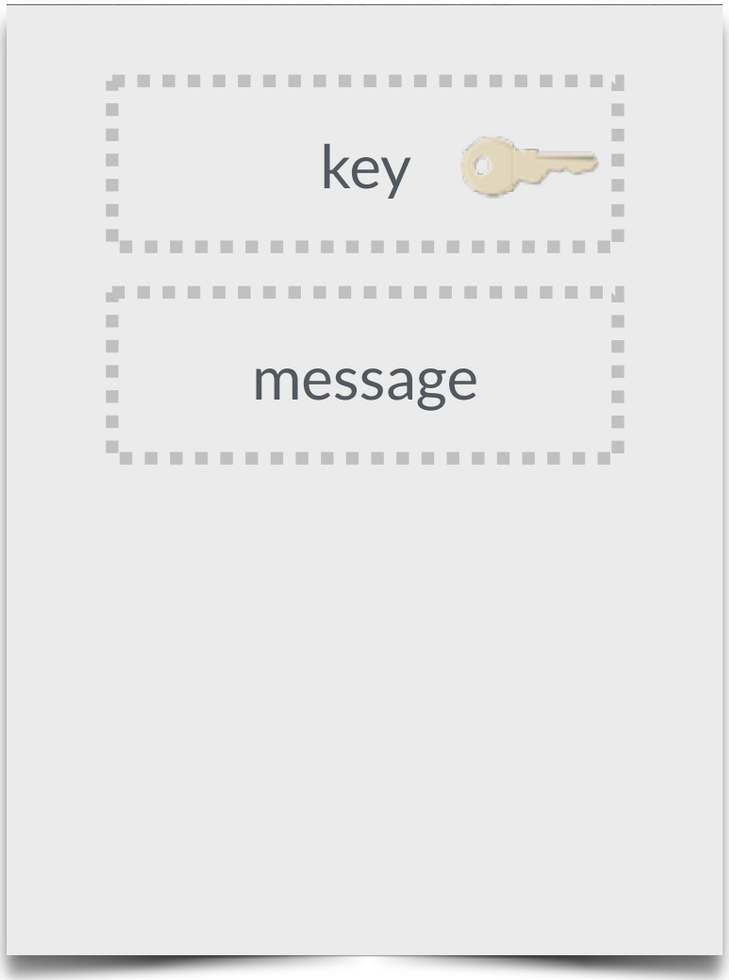
A Kafka Message

- Similar to a *row* or a *record*
- Message is an array of bytes
- No special serialization, that is done at the producer or consumer



A Kafka Message Key

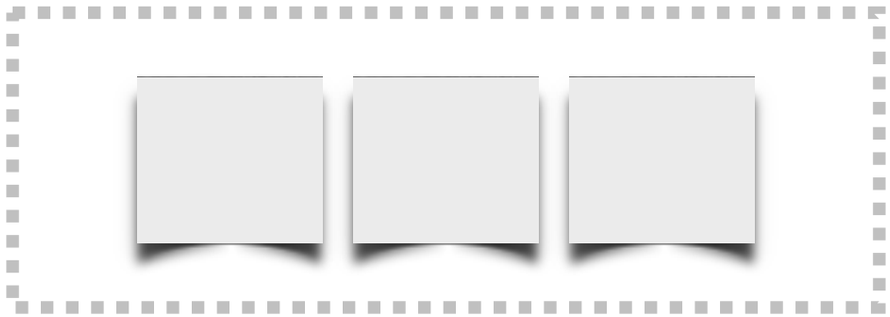
- Message may contain a *key* for better distribution to partitions
- The *key* is also an array of bytes
- If a *key* is provided, a partitioner will hash the key and map it to a single partition
- Therefore it is the only time that something is guaranteed to be in order



A Kafka Batch

Kafka Batch

- A collection of messages, that is sent, configured in *bytes*
- Sent to the same *topic* and the same *partition*
- *Avoids overhead of sending multiple message over the wire*



Kafka Compare/ Contrast with Sub/Pub

- Known as the distributed "git log"
- Ability to replayed in a consistent manner
- Kafka is also stored durable, in order *, and deterministic. *
- Data can also be distributed for resiliency
- Scalable and Elastic

Kafka Architecture

Apache Zookeeper

Zookeeper Essence

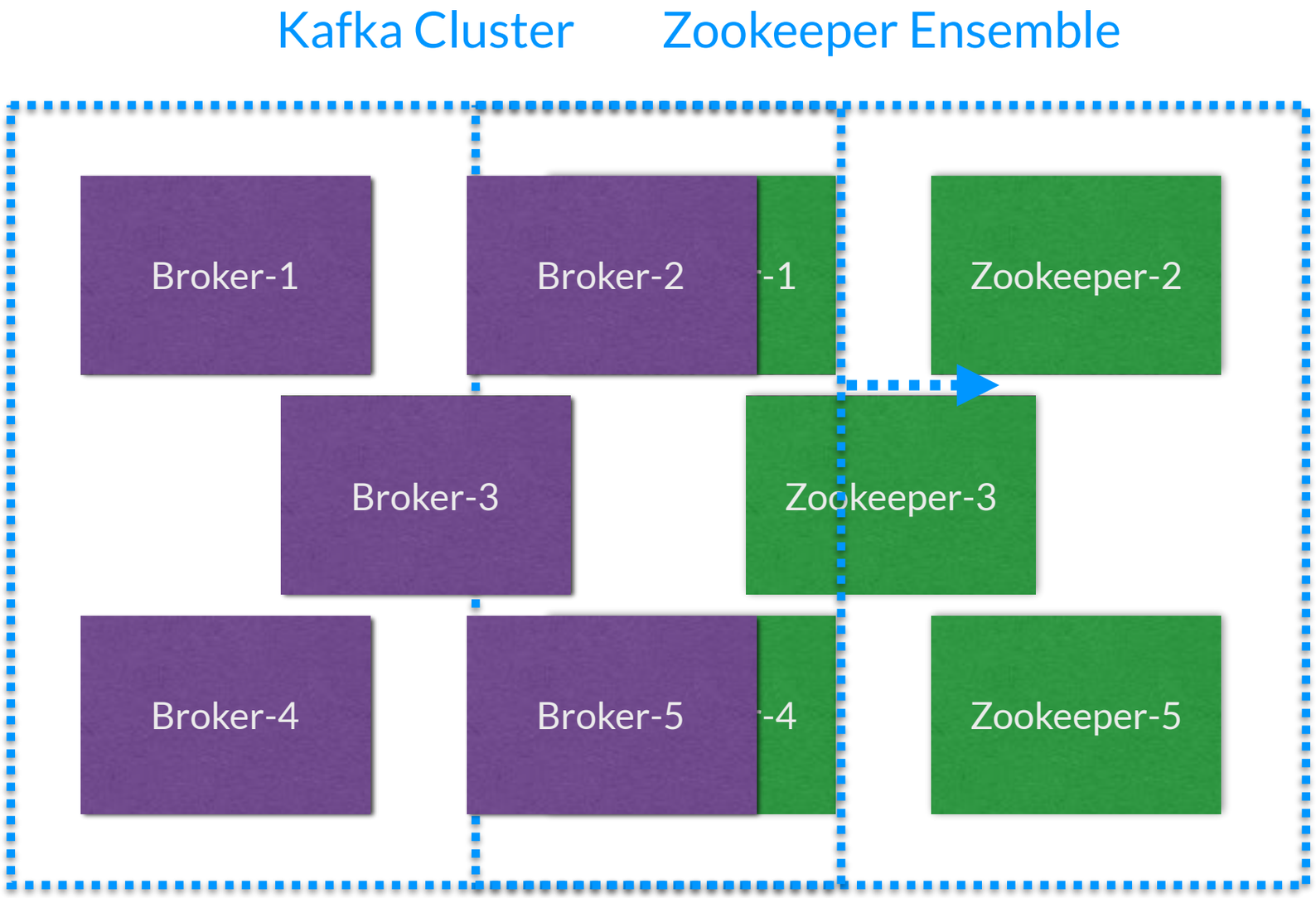
Centralized Coordination Service

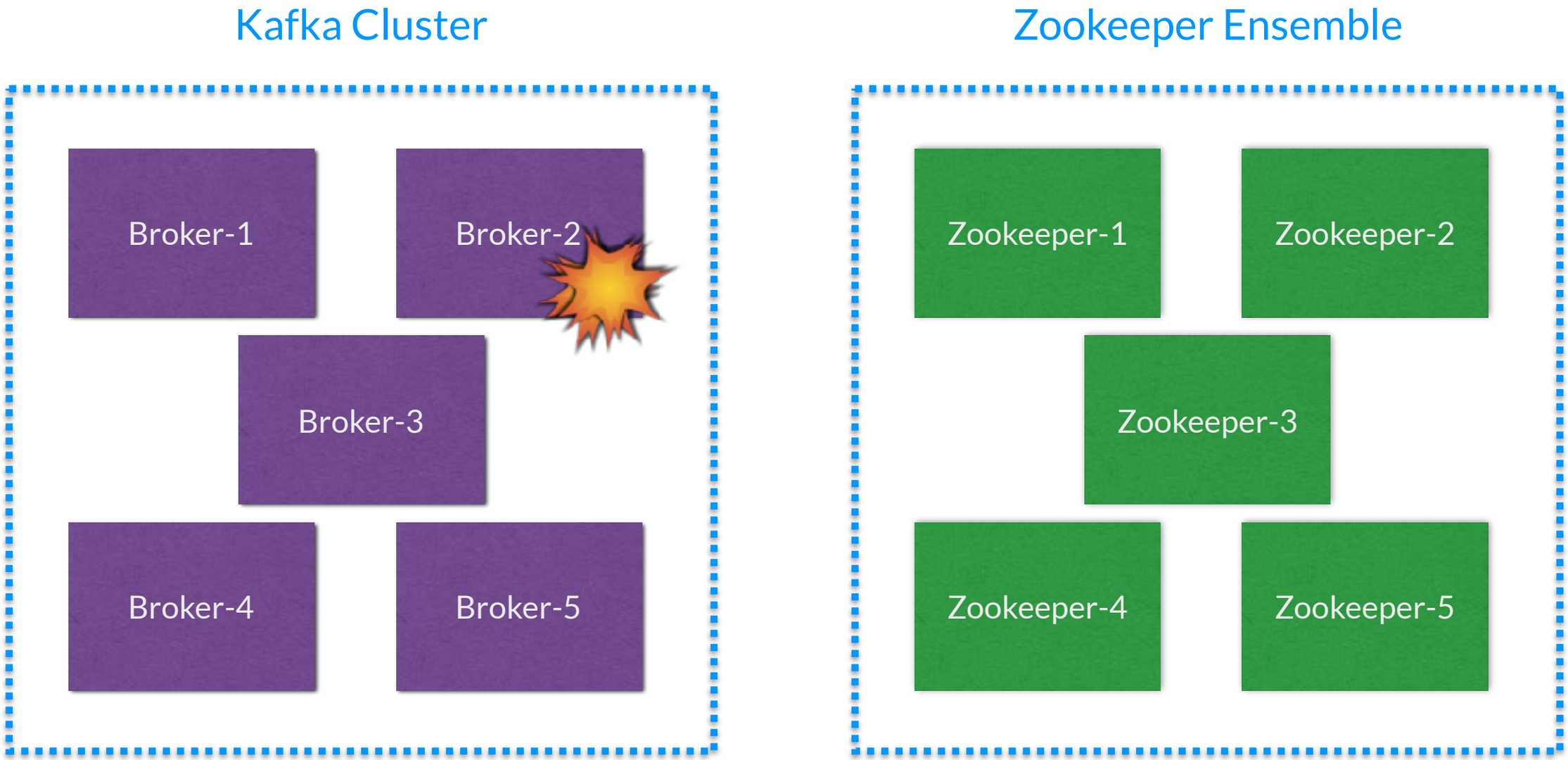
Maintains:

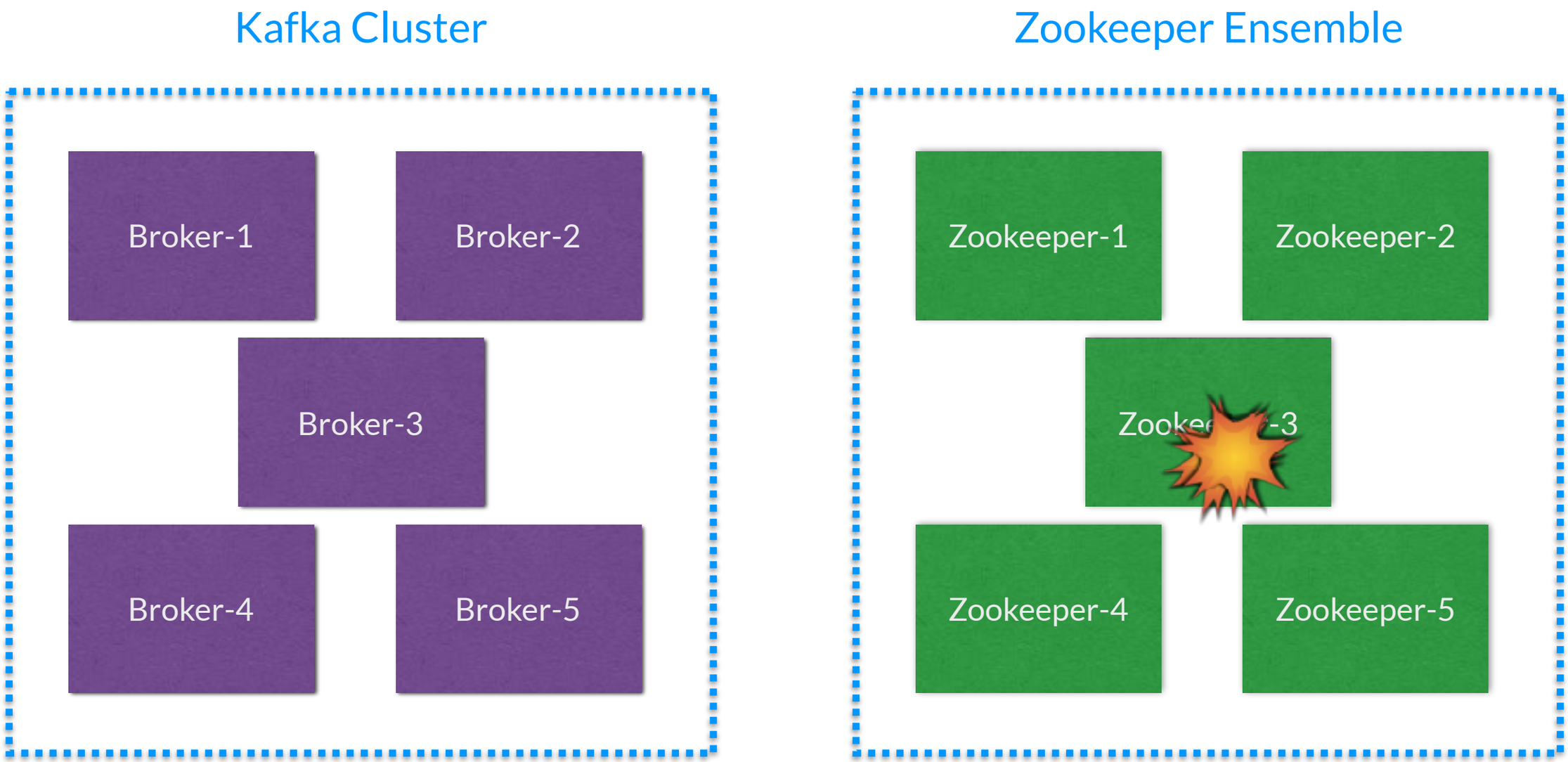
~ Metadata

~ Naming

~ Configuration


Kafka Cluster
Zookeeper Ensemble
Broker-1
Broker-2
Zookeeper-2
Broker-3
Zookeeper-3
Broker-4
Broker-5
Zookeeper-5


Kafka Cluster
Broker-1
Broker-2
Broker-3
Broker-4
Broker-5
Zookeeper Ensemble
Zookeeper-1
Zookeeper-2
Zookeeper-3
Zookeeper-4
Zookeeper-5


Kafka Cluster
Broker-1
Broker-2
Broker-3
Broker-4
Broker-5
Zookeeper Ensemble
Zookeeper-1
Zookeeper-2
Zookeeper-3
Zookeeper-4
Zookeeper-5

Requirements

Hardware Requirements

Do not co-locate other applications due to memory page cache pollution and will degrade performance

Performant drive is required

Storage capacity will need to be calculated by the expected messages per day and retention

Slower networks can degrade the rate in which messages are produced

Cloud Requirements

Analyze by data retention

Analyze performance need by the producers

If low latency is required, SSD should be considered

Ephemeral Storage may be required (Elastic Block Storage)

Kafka Guarantees

Kafka Guarantees

Messages sent by a producer to a particular topic partition will be appended in the order they are sent.

A consumer instance sees records in the order they are stored in the log.

For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any records committed to the log.

Setup

Zookeeper Setup

Setting up Instances in AWS

Notes about Zookeeper

Create a directory `/var/lib/zookeeper`

Requires an ensemble:

- Odd number of servers
  - Required for redundancy and respond to requests
  - 3 nodes can respond to one node missing, 5 nodes with 2
  - Recommended you don’t go above 7

/usr/local/zookeeper/conf/zoo.cfg

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
```

/usr/local/zookeeper/conf/zoo.cfg

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
```

`tickTime` - a millisecond standard of unit for zookeeper

/usr/local/zookeeper/conf/zoo.cfg

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
```

`dataDir` - the data directory for a zookeeper instance

/usr/local/zookeeper/conf/zoo.cfg

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
```

`clientPort` - ports for clients to connect to this zookeeper instance

Clustering with Zookeeper

/usr/local/zookeeper/conf/zoo.cfg

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
initLimit=20
syncLimit=5
server.1=zoo1.example.com:2888:3888
server.2=zoo1.example.com:2888:3888
server.3=zoo1.example.com:2888:3888
```

`initLimit` - amount of time allow followers to connect with leader in tickTime units, therefore:

```
initLimit (20) * tickTime (2000 ms) = (40000 ms) = 40 seconds
```

/usr/local/zookeeper/conf/zoo.cfg

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
initLimit=20
syncLimit=5
server.1=zoo1.example.com:2888:3888
server.2=zoo1.example.com:2888:3888
server.3=zoo1.example.com:2888:3888
```

`syncLimit` - how far out of sync followers can be with the leader in tickTime units, therefore:

```
syncLimit (5) * tickTime (2000 ms) = (10000 ms) = 10 seconds
```

/usr/local/zookeeper/conf/zoo.cfg

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
initLimit=20
syncLimit=5
server.1=zoo1.example.com:2888:3888
server.2=zoo1.example.com:2888:3888
server.3=zoo1.example.com:2888:3888
```

`server.x` - name of the server
`hostname` - the hostname of the server
`peerPort` - port where the servers communicate with one another
`leaderPort` - is the TCP port where leader elections are performed

/var/lib/zookeeper/myid

```
2
```

Describe who you are in this instance, just one number, and that's it.

Kafka Setup

Brokers on AWS

/etc/hosts

```
127.0.0.1   localhost ... ip-31-35-1-222
::1         localhost6 ...
```

/usr/local/kakfa/config/server.properties

```
broker.id=2
listeners=PLAINTEXT://0.0.0.0:9092
advertised.listeners=PLAINTEXT://<exposed-port>:9092
zookeeper.connect=<zookeeper0>:2181,<zookeeper1>:2181,<zookeeper2>:2181
log.dirs=/tmp/kafka-logs
```

Running Kafka

```
> /usr/local/kafka/bin/kafka-server-start.sh -daemon \
    /usr/local/kafka/config/server.properties
```

Stopping Kafka

```
> /usr/local/kafka/bin/kafka-server-stop.sh
```

Verifying Instances

```
> /usr/local/zookeeper/bin/zkCli.sh -server <zookeeper>:2181
ls /brokers
```

Kafka Topic Creation

Creating a Topic

```
> /usr/bin/kafka-topics --create --zookeeper zoo:2181
    --replication-factor 2 --partitions 4 --topic <topic>
```

Listing the Topics

```
> /usr/bin/kafka-topics --zookeeper <zookeeper>:2181 --list
```

Kafka CLI writing and reading

Sending a message CLI

```
> /usr/local/kafka/bin/kafka-console-producer.sh --broker-list
<kafka-broker>:9092 --topic test
```

Reading Back The Messages

```
> /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server <kafka-broker>:
9092 --topic <topic> --from-beginning
```

Viewing how partitions are distributed

```
> /usr/local/kafka/bin/kafka-topics.sh --describe --topic <topic-name>
    --zookeeper <zookeeper>:2181
```

Kafka Programming

Producers

Demo: Producers

Acks

Acks

`acks` controls how many partition replicas must receive the record before the write is considered a success.

Acks

acks	description
0	No acknowledgment, assume all is well
1	At least one replica will Producer will receive a success response, error if unsuccessful, up to client to hand
all	All replicas must acknowledge. Higher latency, safest

Demo: Ack Producers

Kafka Programming

Consumers & Groups

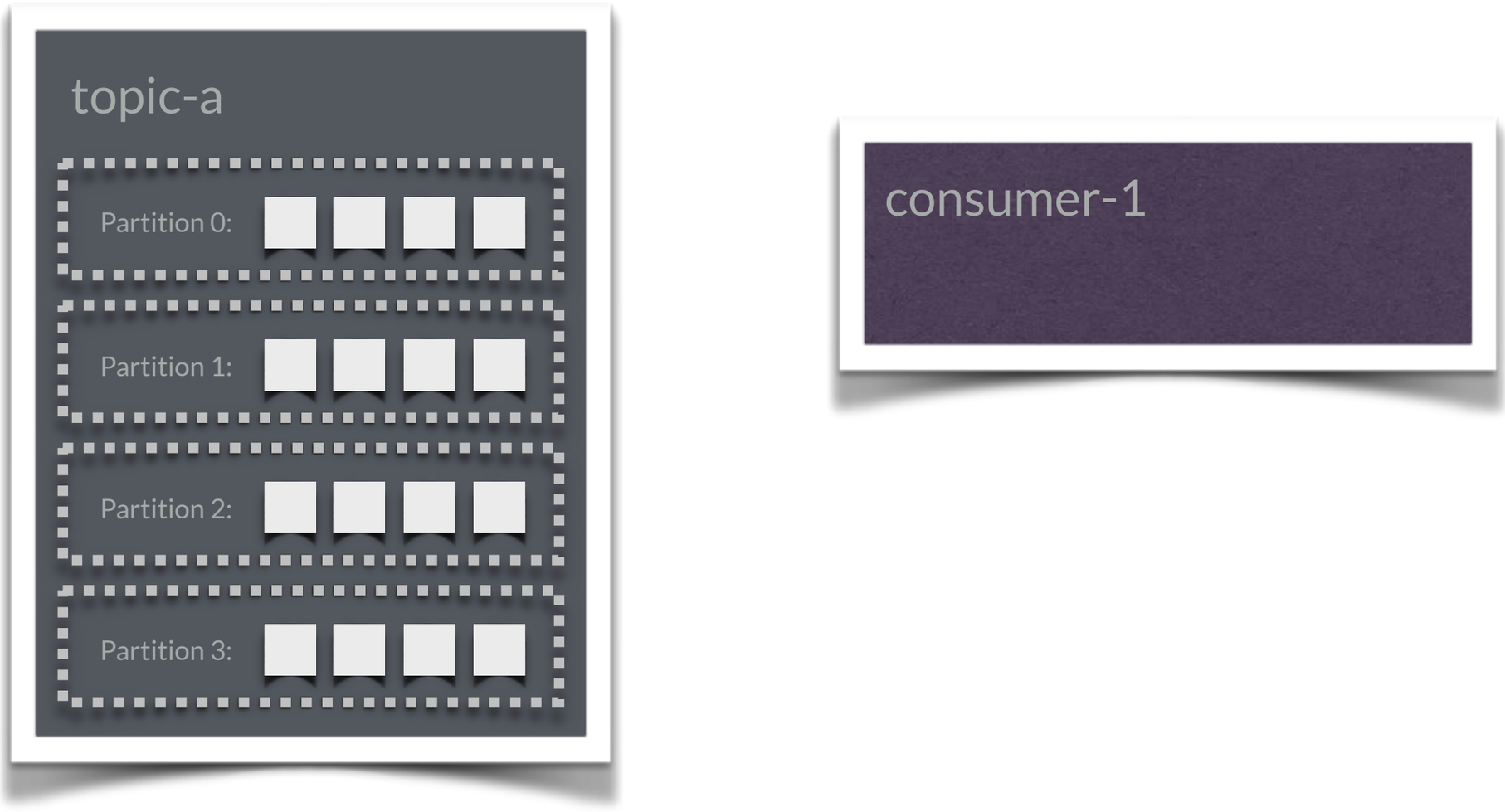
Kafka Consumer Groups

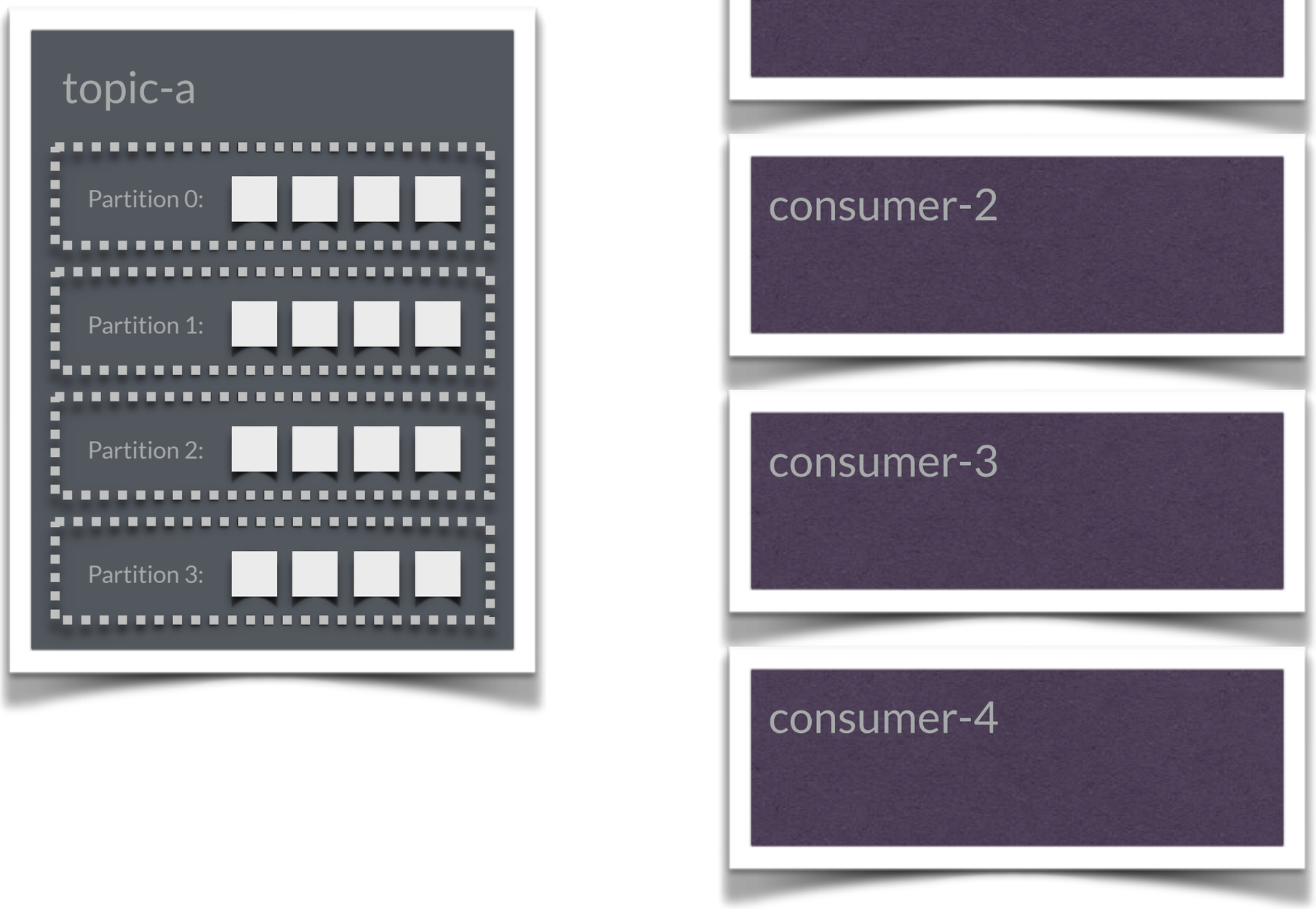
Consumers are typically done as a group

A single consumer will end up inefficient with large amounts of data

A consumer may never catch up

Demo: Simple Consumers


topic-a
Partition 0:
Partition 1:
Partition 2:
Partition 3:
consumer-1


topic-a
Partition 0:
Partition 1:
Partition 2:
Partition 3:
consumer-1
consumer-2
consumer-3
consumer-4

Kafka's Goal

Kafka scales to large amount of different consumers without affecting performance

Kafka Consumer Threading

There are no multiple consumers that belong to the same group in one thread.
One consumer per one thread.

There are not multiple threads running one consumer,
One consumer per one thread.

__consumer_offsets

Topic on Kafka brokers that contain information about consumer and their offsets, stored in Kafka’s data directory

Synchronous commit

Synchronous Commit

After turning off auto-commit

Committing and blocking

Asynchronous commit

Asynchronous Commit

After turning off auto-commit

Committing and and letting it commit asynchronously

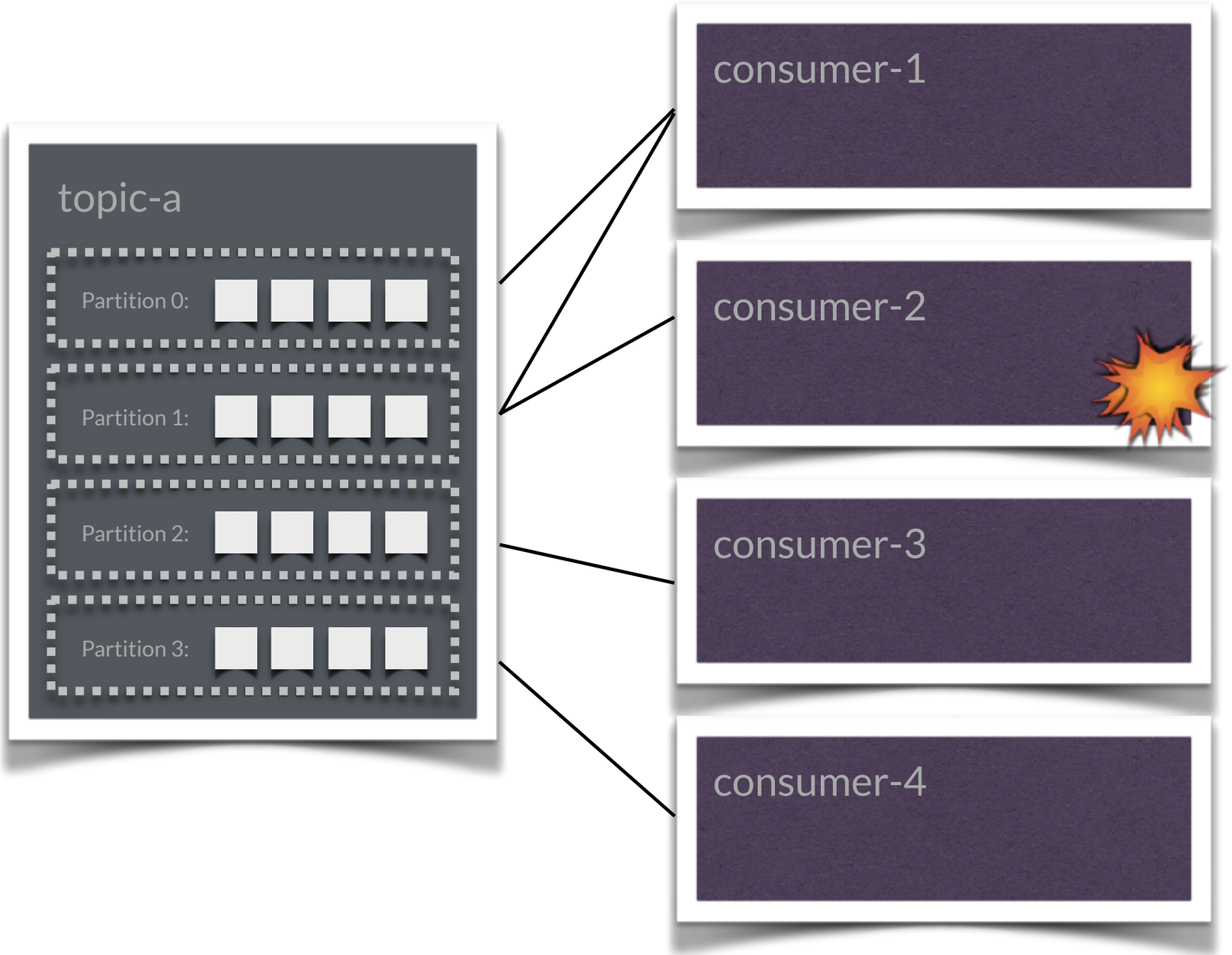
Partition Rebalancing

Partition Rebalance

When one partition is moved from one consumer to another, this is known as a *rebalance*.

A way to mitigate when either consumers go down, or when consumers or added.

Although unavoidable, this will cause an unfortunate pause, and it ***will lose state***.


topic-a
Partition 0:
Partition 1:
Partition 2:
Partition 3:
consumer-1
consumer-2
consumer-3
consumer-4

Demo: Partition Rebalancing

Groups and Heartbeats

Groups and Heartbeats

At regular intervals, consumers will send heartbeats to the broker, to let it know it is alive and still reading data

Heartbeats are sent to a Kafka broker called the *group coordinator*

They are sent when the consumer `polls` and consumes a record

Leaving the Group

When a consumer leaves the group, it lets the *group coordinator* know it is done

The Kafka broker then triggers a rebalance of the group.

Demo: Proper Consumer Groups

Retention Settings

Retention

Durable Storage over a Period of Time

Can either be configured in time or size

Once reached messages are expunged

Retention Formula

```
log.retention.hours = 120
log.retention.bytes = 3221225472
```

Default Hours are set to (168 hours) or less than one week

You may also use `log.retention.minutes` or `log.retention.ms`, whichever is smallest wins

Retention Evaluation

Durable Storage over a Period of Time

Can either be configured in time or size

Once reached messages are expunged

Compaction

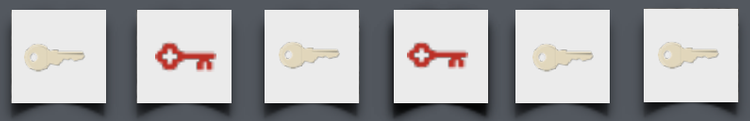
Compaction

A form of retention where messages of the same key where only the latest message will be retained.

Compaction is performed by a *cleaner thread*.

kafka broker: 0

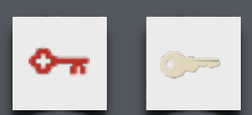
Partition 0:



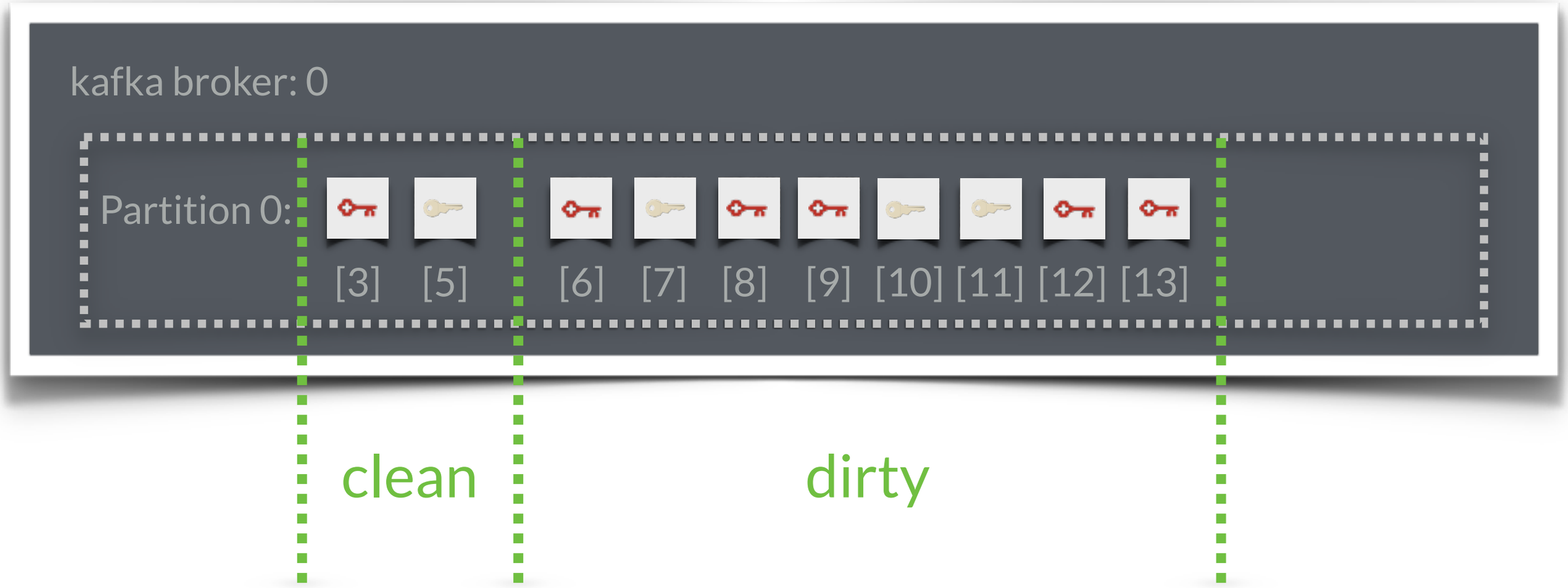
[0] [1] [2] [3] [4] [5]

kafka broker: 0

Partition 0:



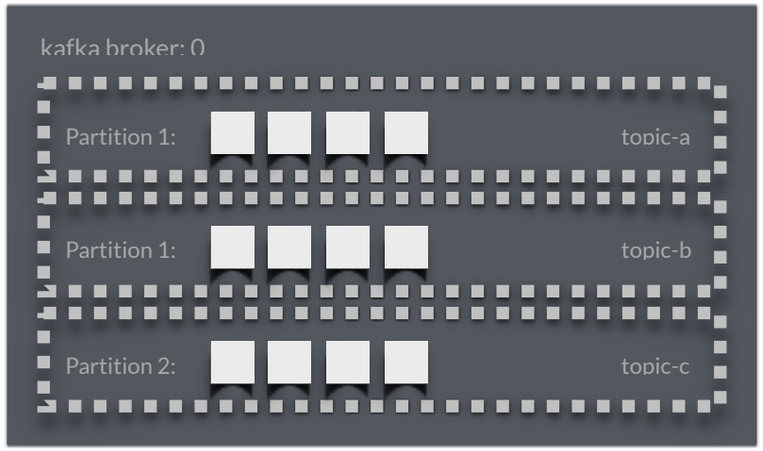
[3] [5]

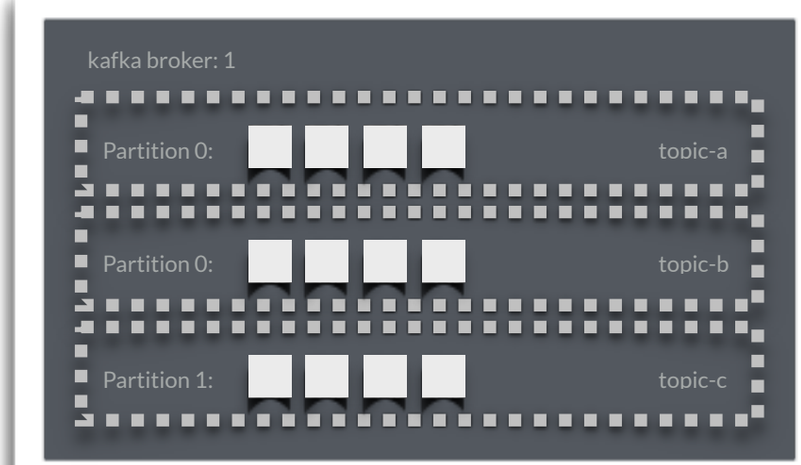


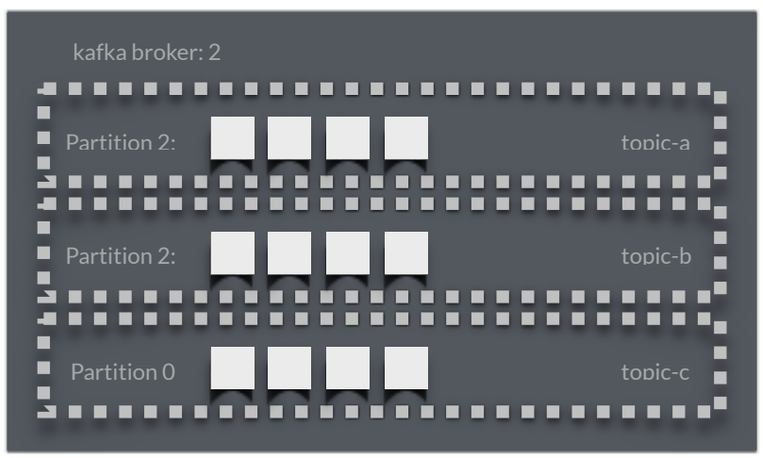
Kafka will start compacting when 50% of the topic contains dirty records

Resiliency

Ephemeral Nodes

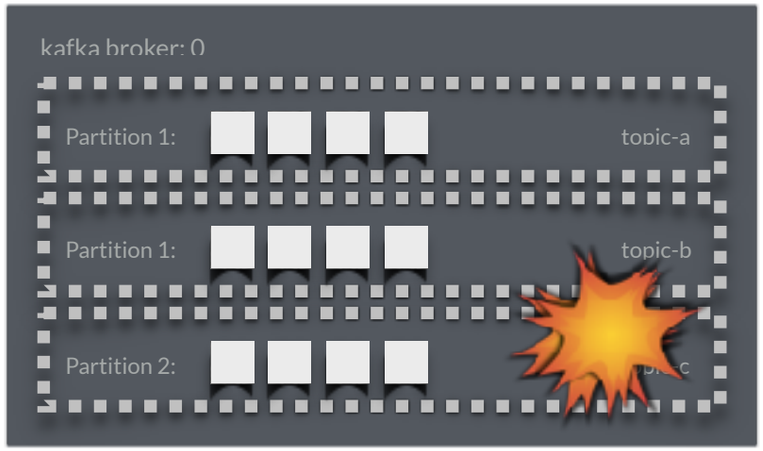


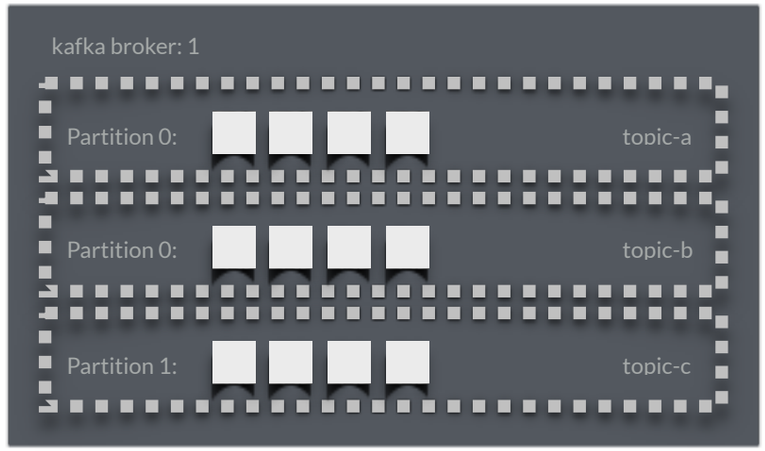


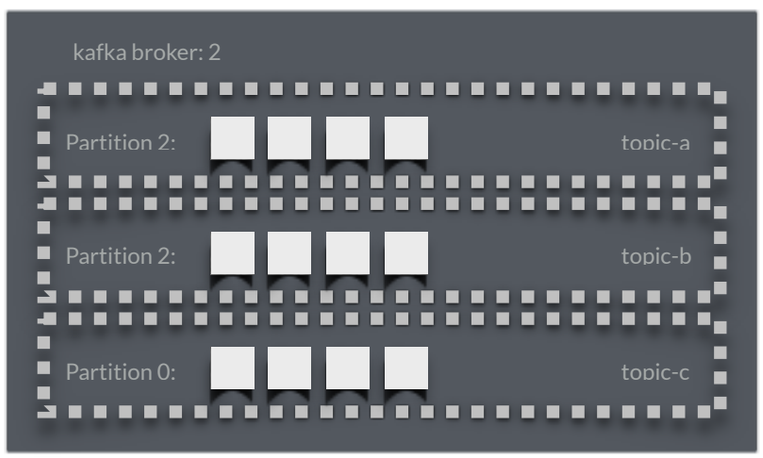



```
broker/ids
0
1
2
```

Ephemeral Nodes








```
broker/ids
1
2
```

ISR (In sync replicas)

Given a leader partition, an in sync replica is one that has been kept up to date within the last 10 seconds

This is configurable

During a crash, the closes ISR will retain control for failover

Streaming


TK45
115m
ÖNIK.C.

Stream Processing

Endless supply of data

“Replayable" supply of data

Real Time Processing for Fraud Detection, High End Sales Detection, Internet of Things, and More.

Will require built-in accumulator table to perform real time data processing, like *counting*, and *grouping*

Demo:Stream Processing

Conclusion