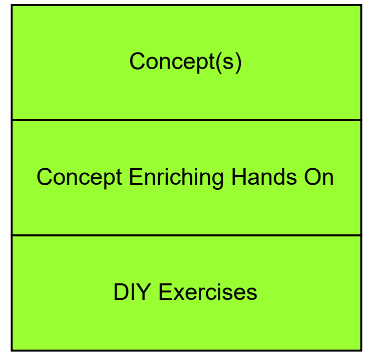
Teaching Principle



Concept(s)

Concept Enriching Hands On

DIY Exercises

Learning Map

Kafka Foundation

Tx Messages
using
CLI

Foundational

Day 1

Advanced/Internals

Day 2

Tx Messages
using
Application

Producer

Producer Application
using API

Day 3

Internals/Configurations

Day 4

Consumer

ConsumerApplication
using API

Day 5

Internals/Configurations

Day 6, 7

Custom
Serializers/Deserializers

Day 8

Tx Messages
In
Spring Boot
MicroServices

Producer

Day 9, 10

Consumer

Day 11,
12, 13, 14

Security

Day
15, 16, 17

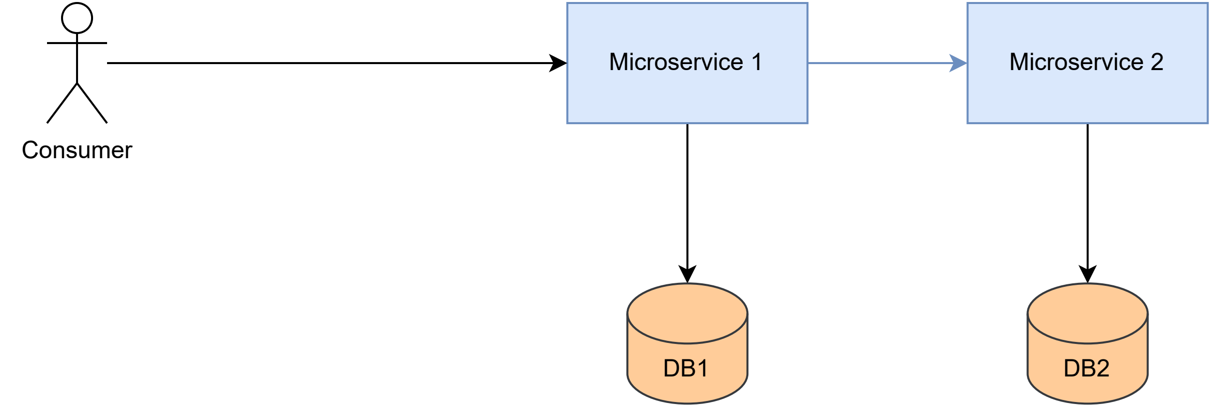
Streams

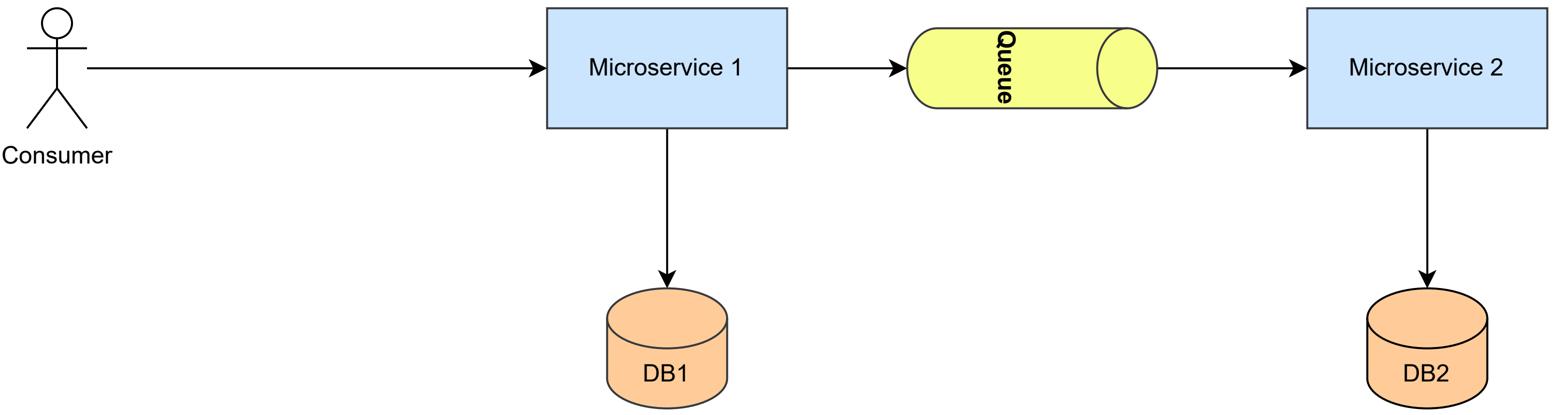
Day
, 18, 19, 20

Connectors

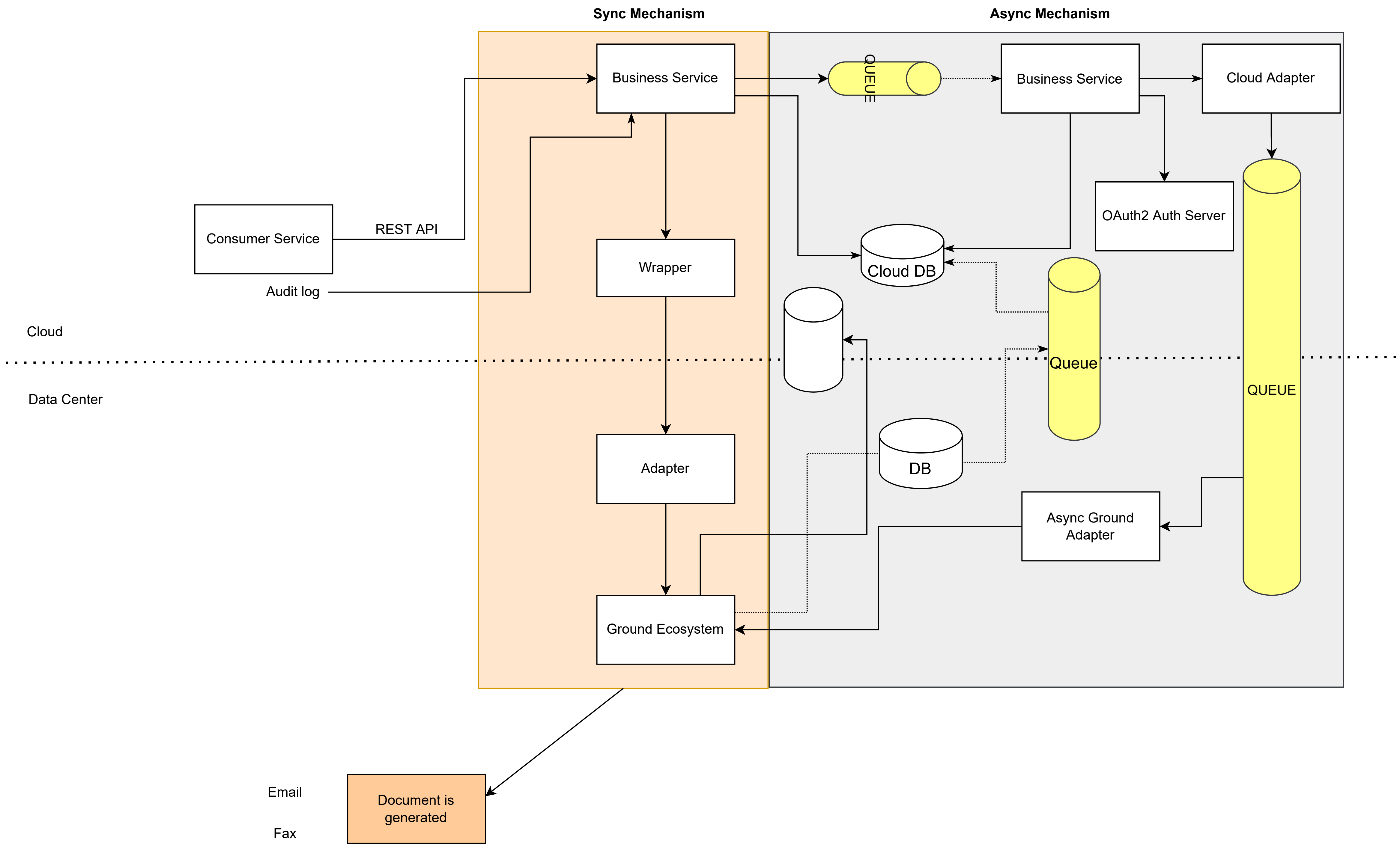
Monitoring

...

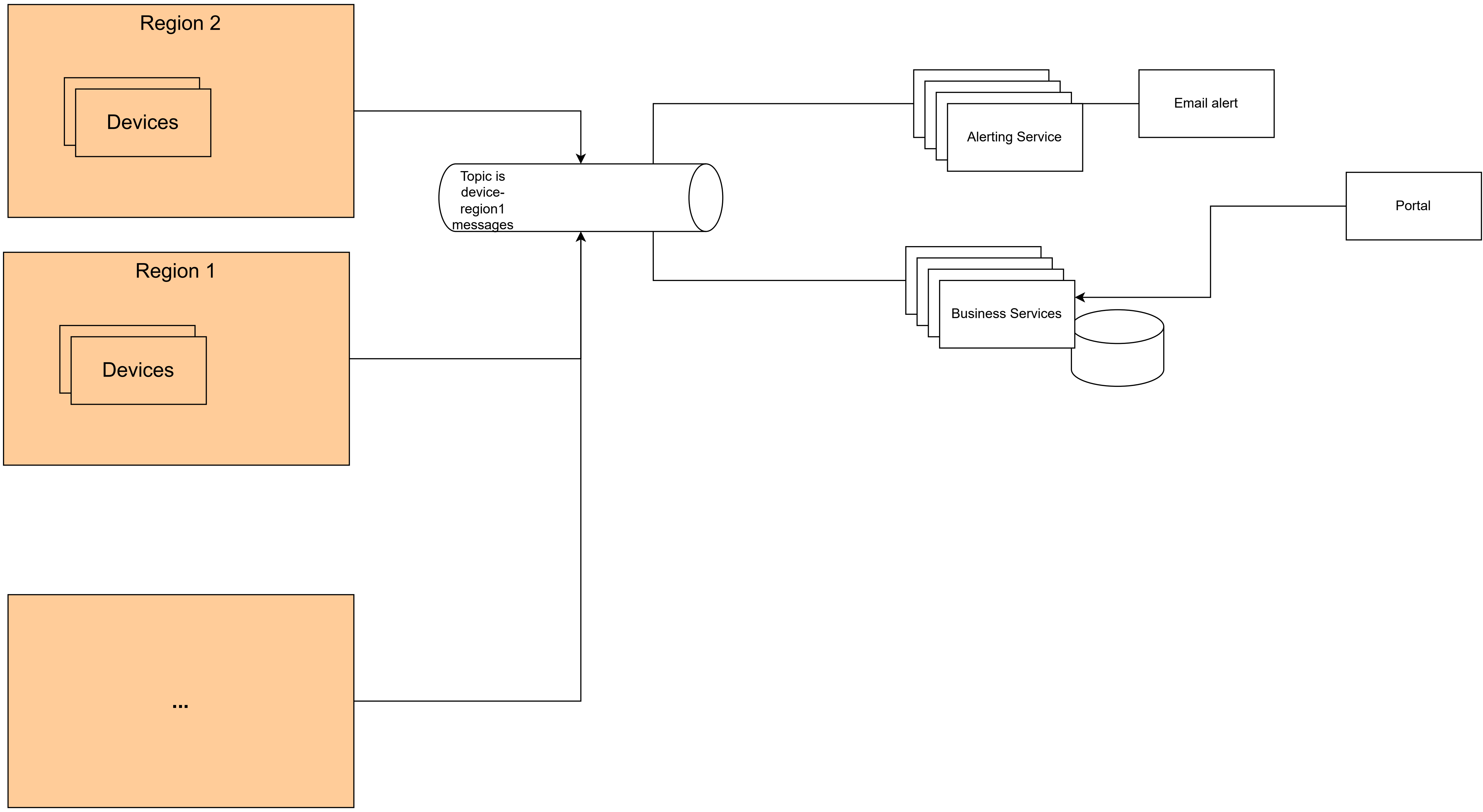

Consumer
Microservice 1
Microservice 2
DB1
DB2


Consumer
Microservice 1
Queue
Microservice 2
DB1
DB2

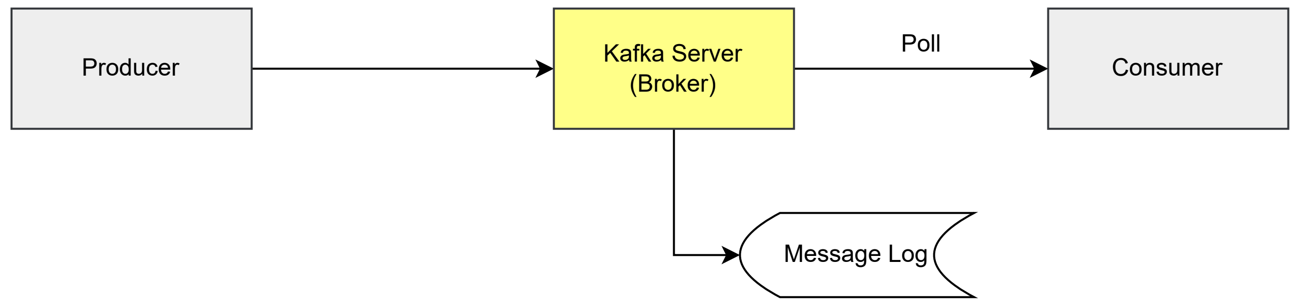
Use case 1



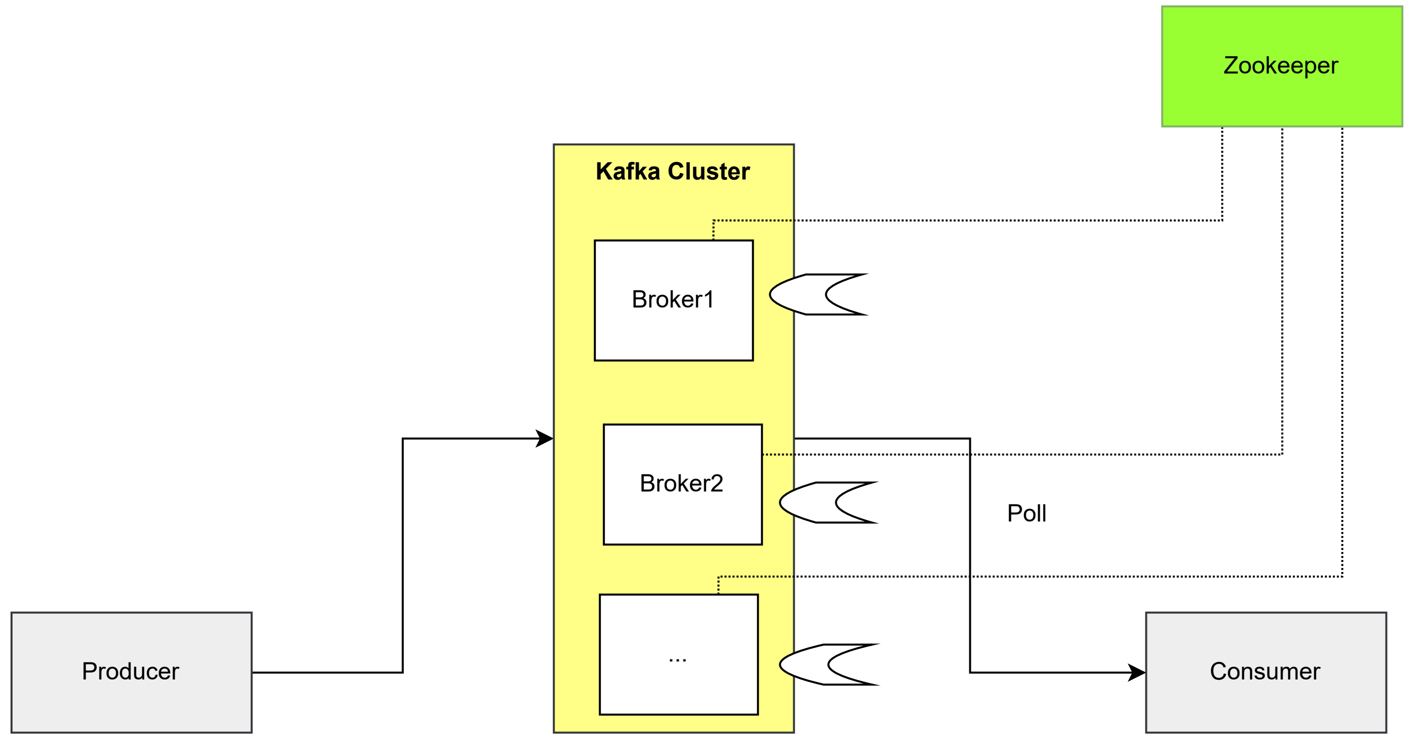
Use case 2



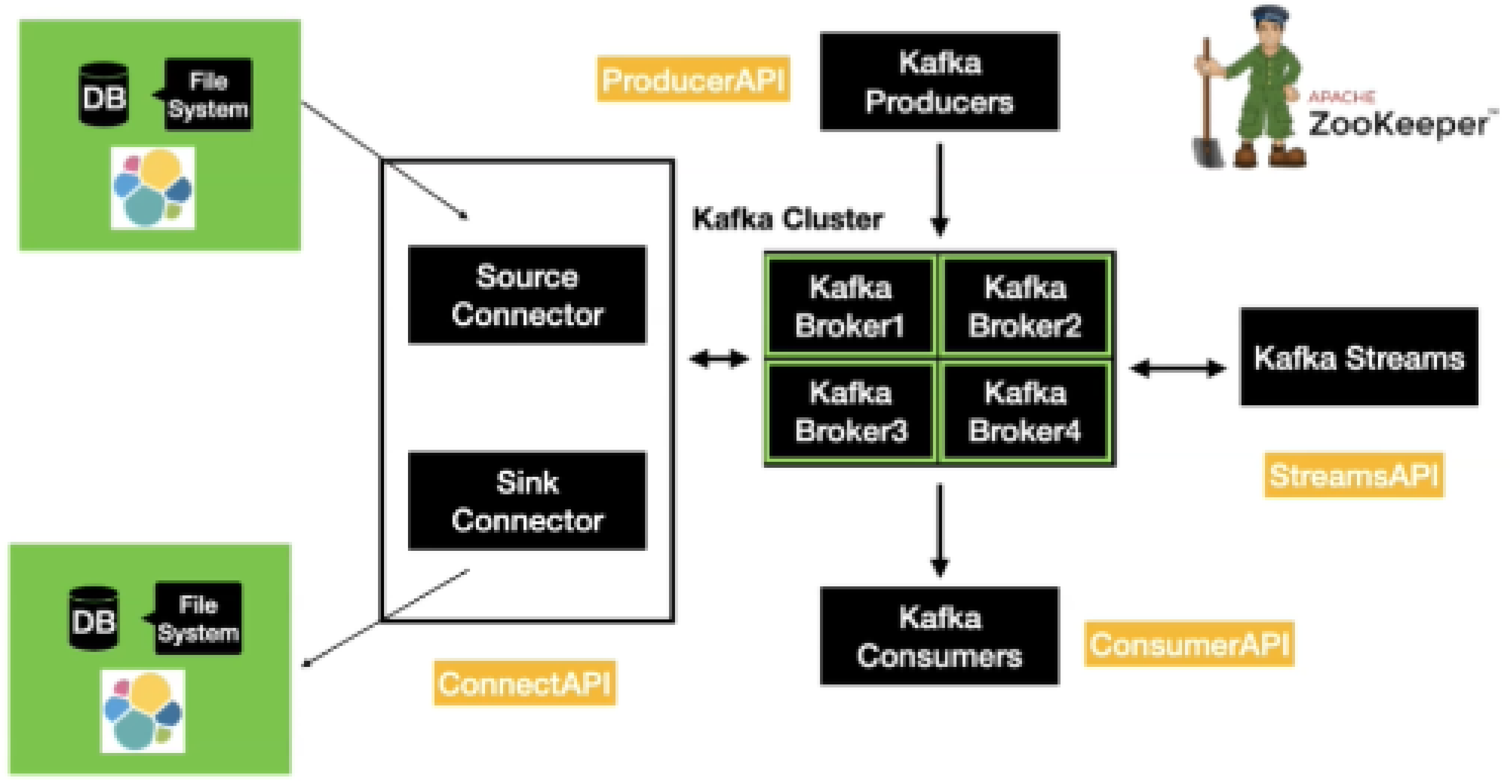
Basic Topology



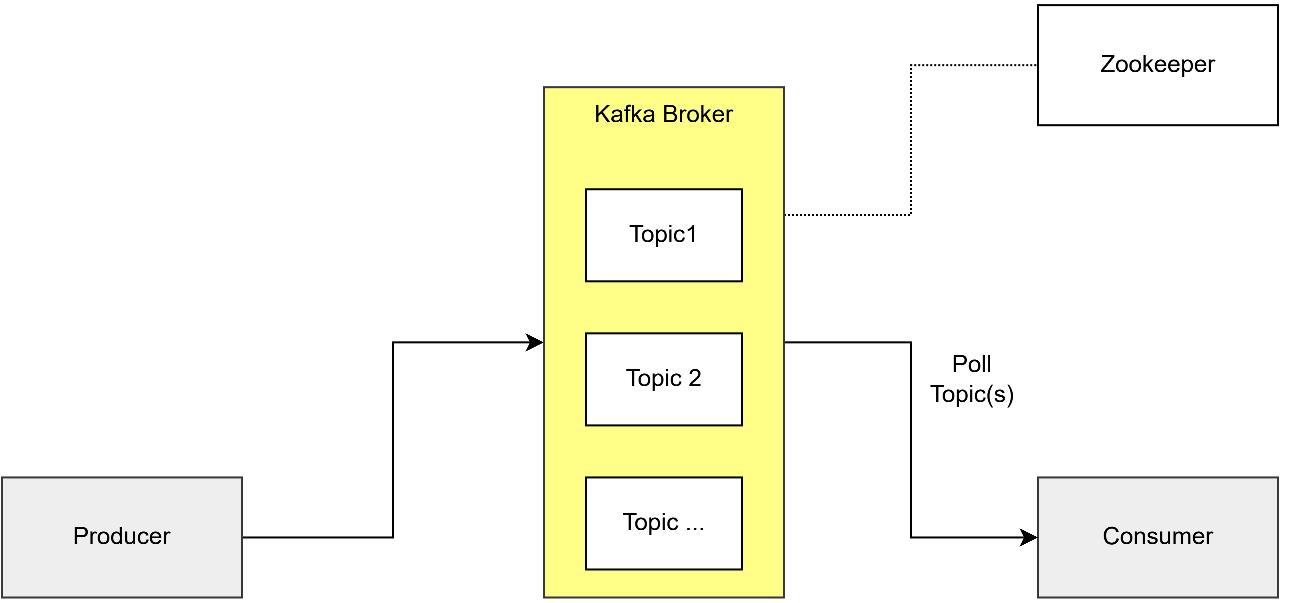
Distributed Topology



Kafka Landscape



Broker Anatomy



Kafka Broker

Topic1

Topic 2

Topic ...

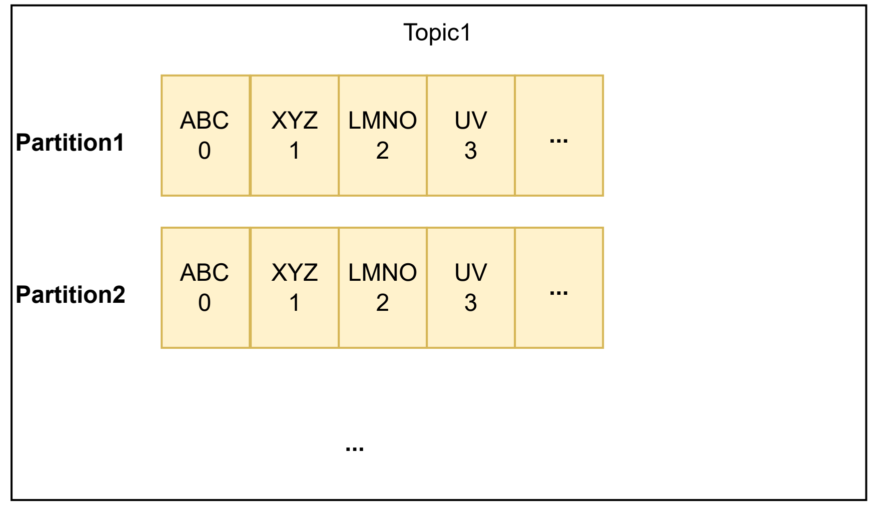
Zookeeper

Producer

Poll Topic(s)

Consumer

Topic Anatomy

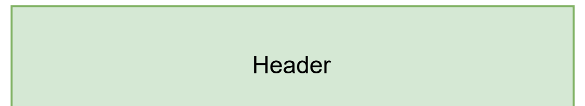


Topic1

Partition1	ABC 0	XYZ 1	LMNO 2	UV 3	...
Partition2	ABC 0	XYZ 1	LMNO 2	UV 3	...

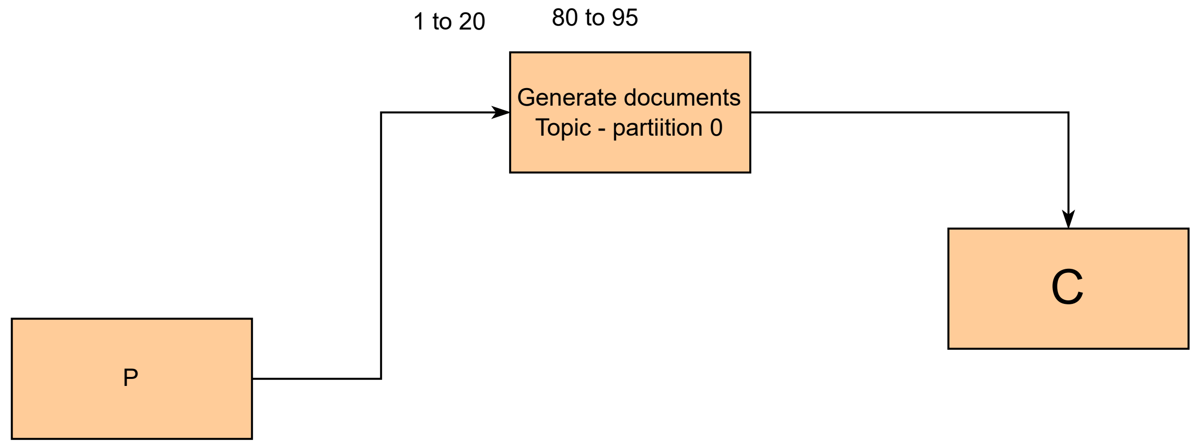
...

Message Anatomy



Header

Topic-Same



1 to 20

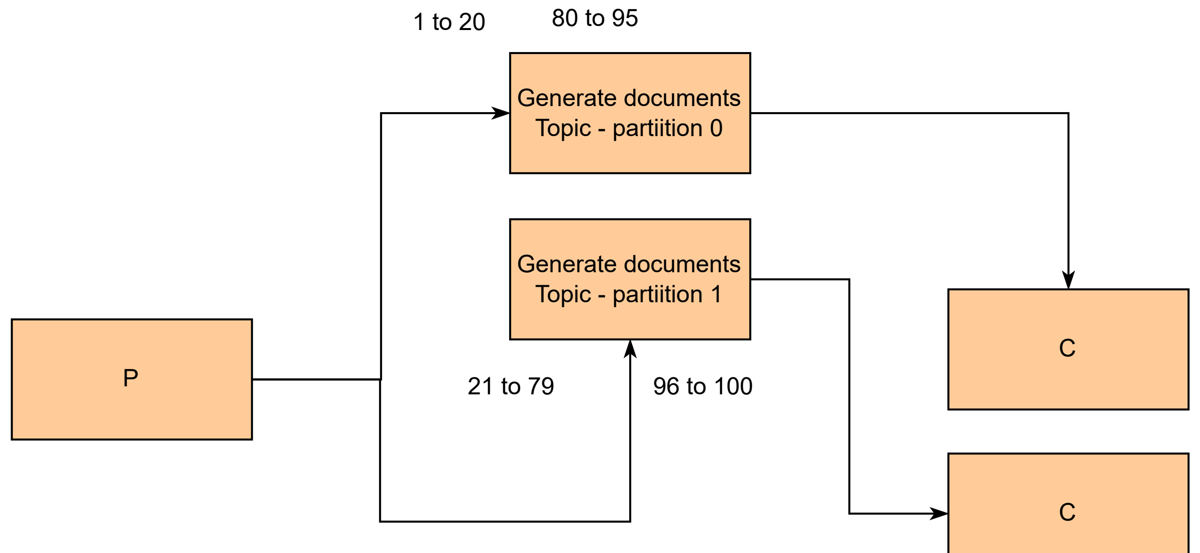
80 to 95

Generate documents Topic - partiition 0

P

C

Topic-Same



1 to 20

80 to 95

Generate documents Topic - partiition 0

Generate documents Topic - partiition 1

21 to 79

96 to 100

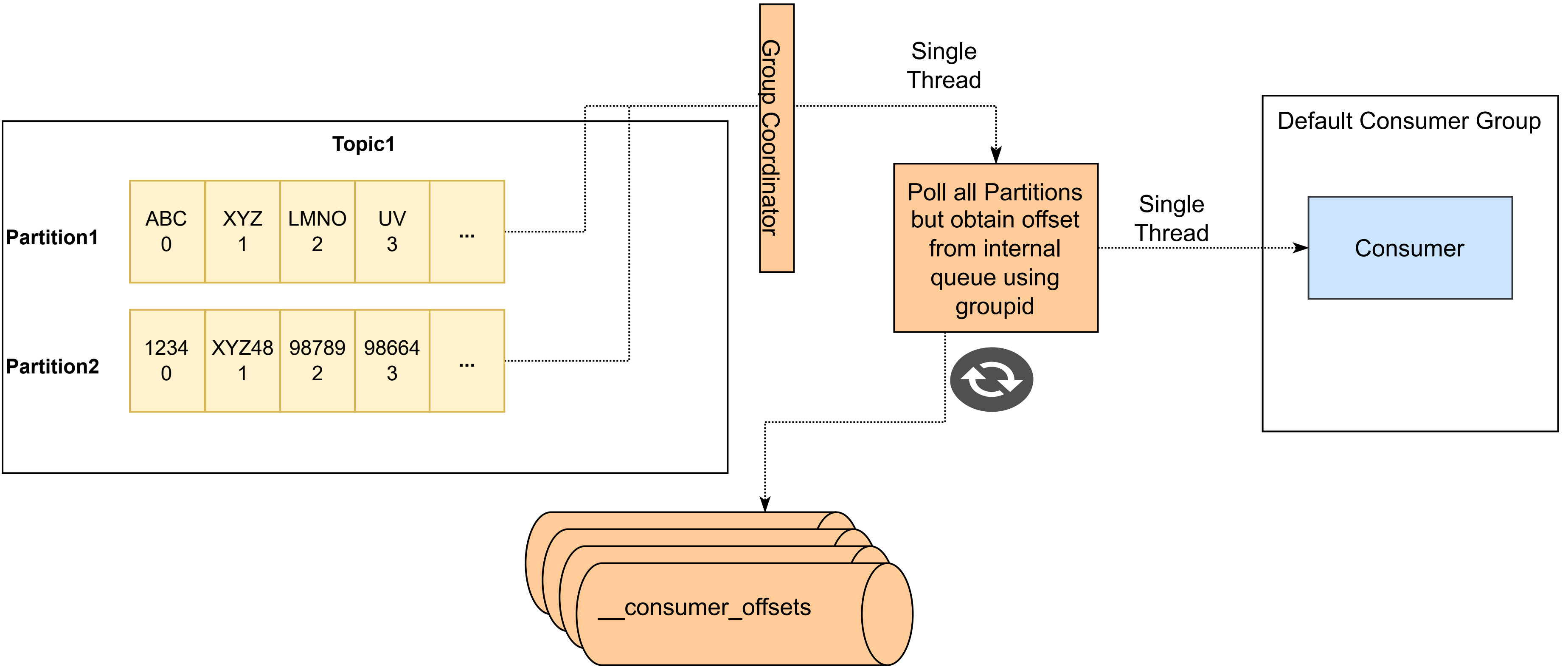
P

C

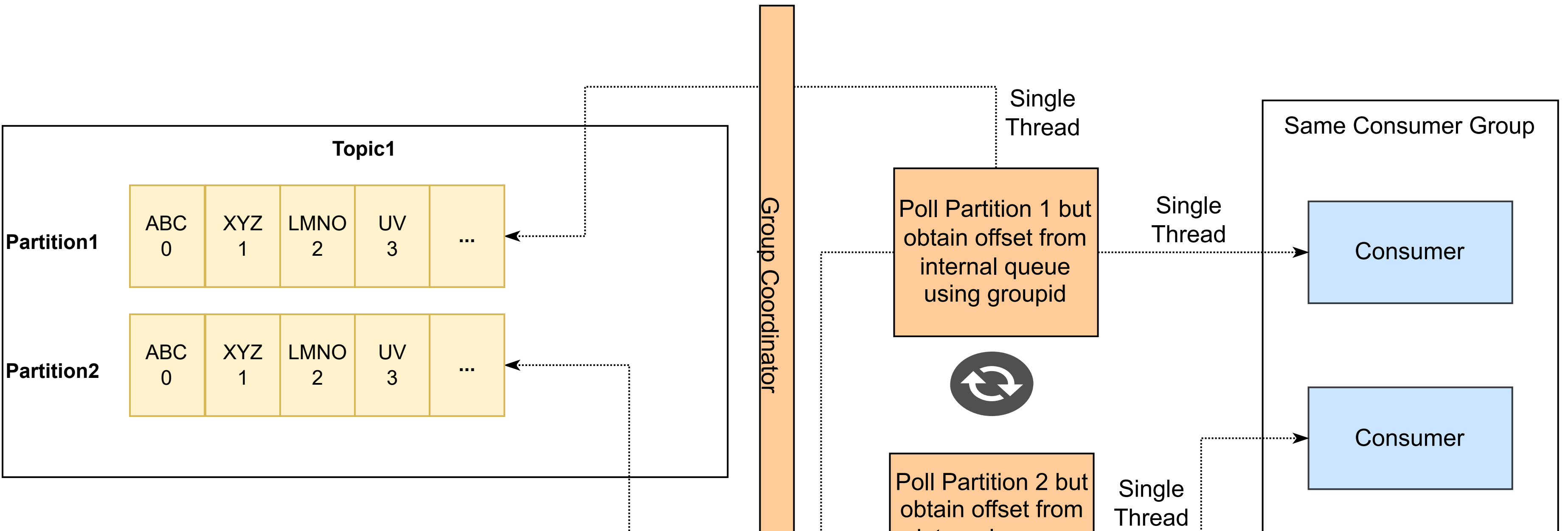
C

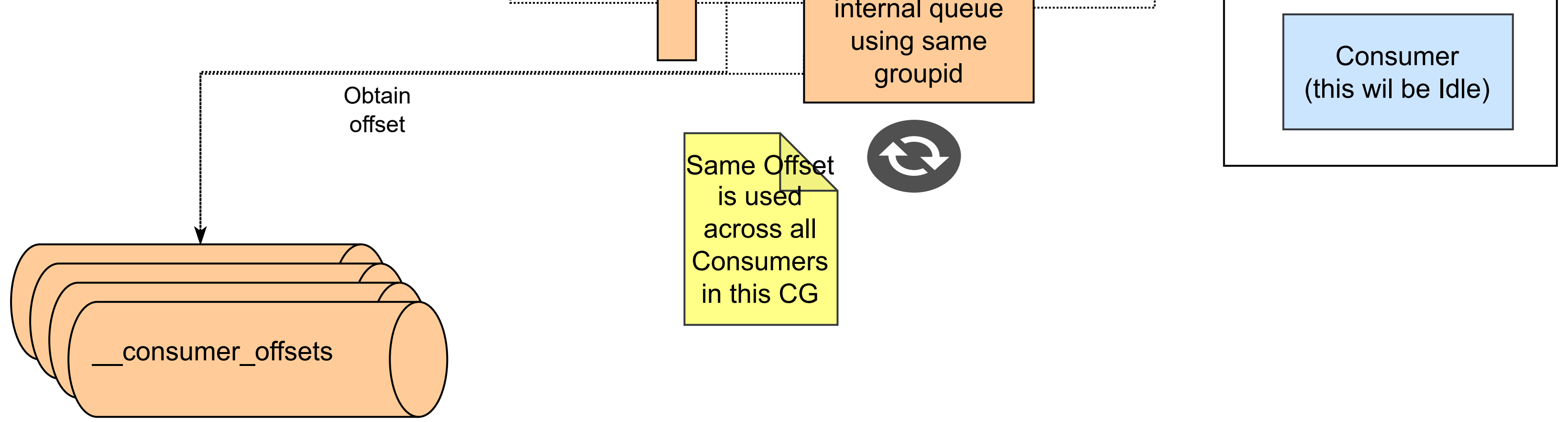
Key(optional)
Value
Timestamp

Default Consumer Group

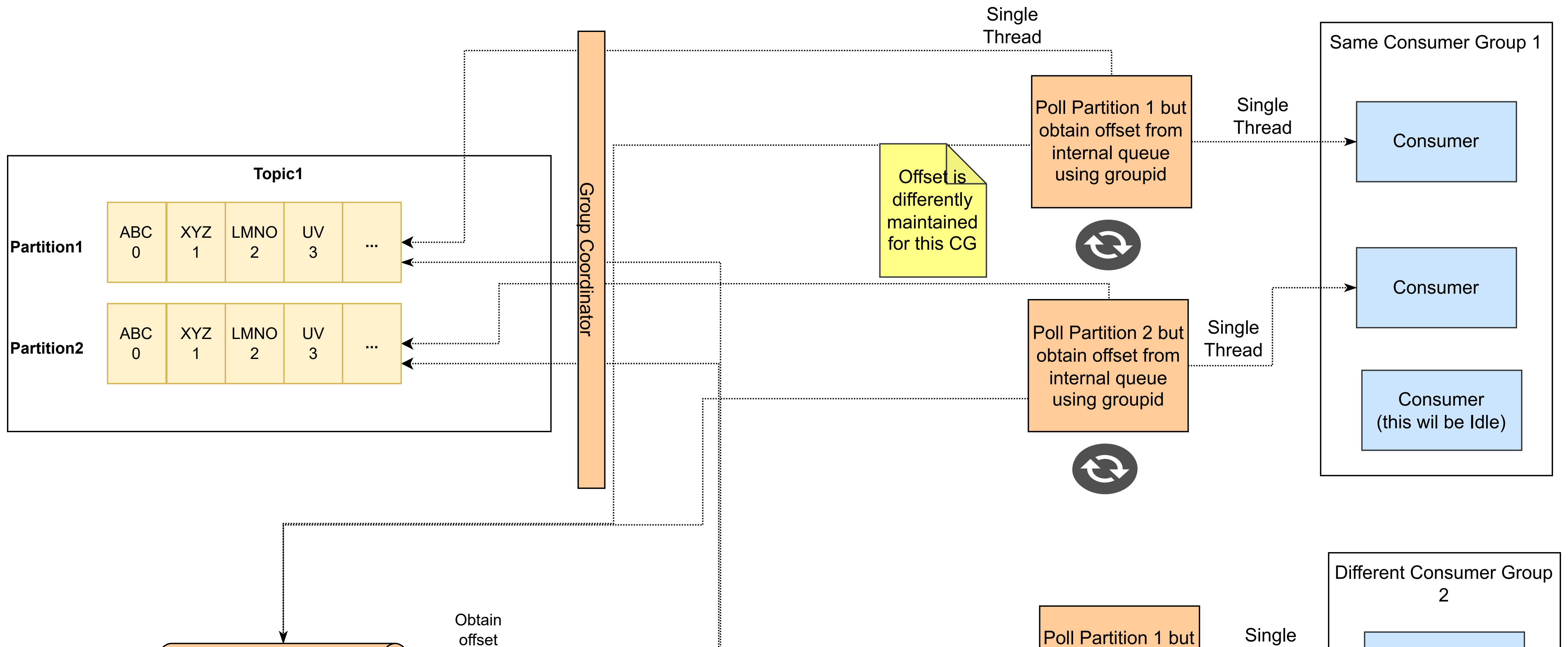


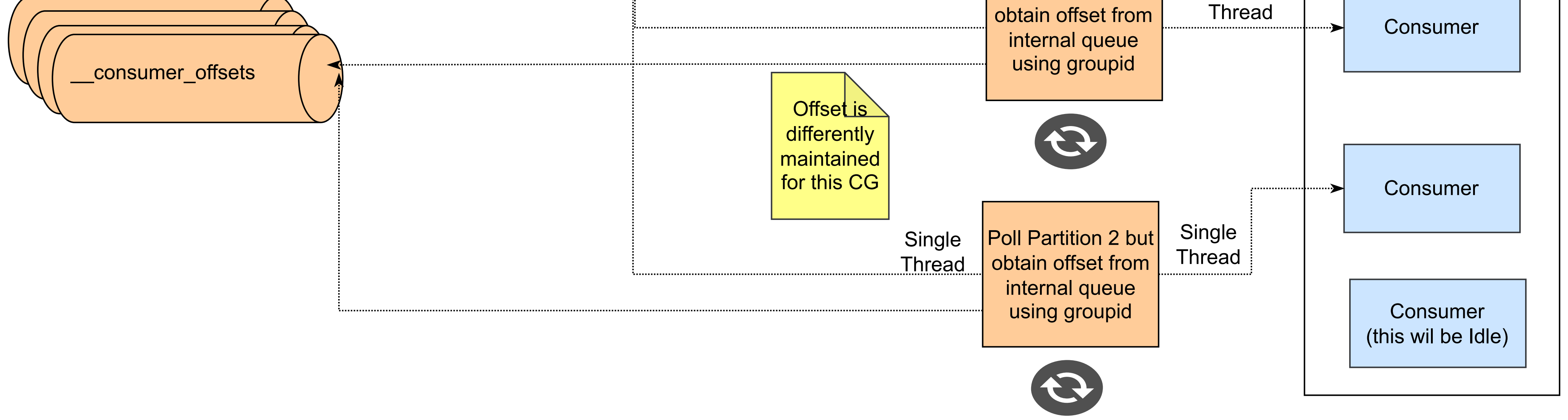
Using same Consumer Group to Read Messages at Scale



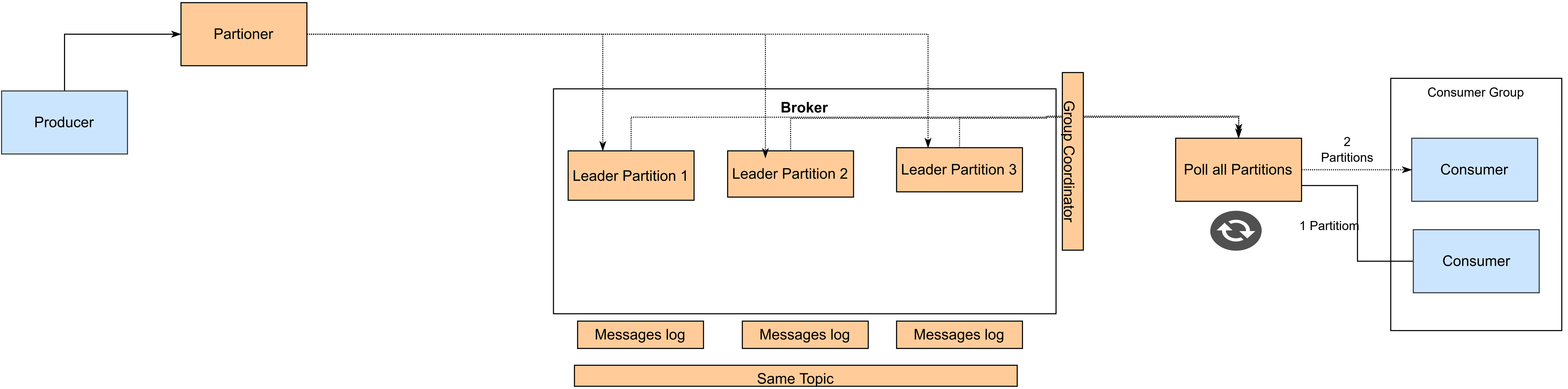


Using Different Consumer Group to Read Messages From Same Topic at Scale

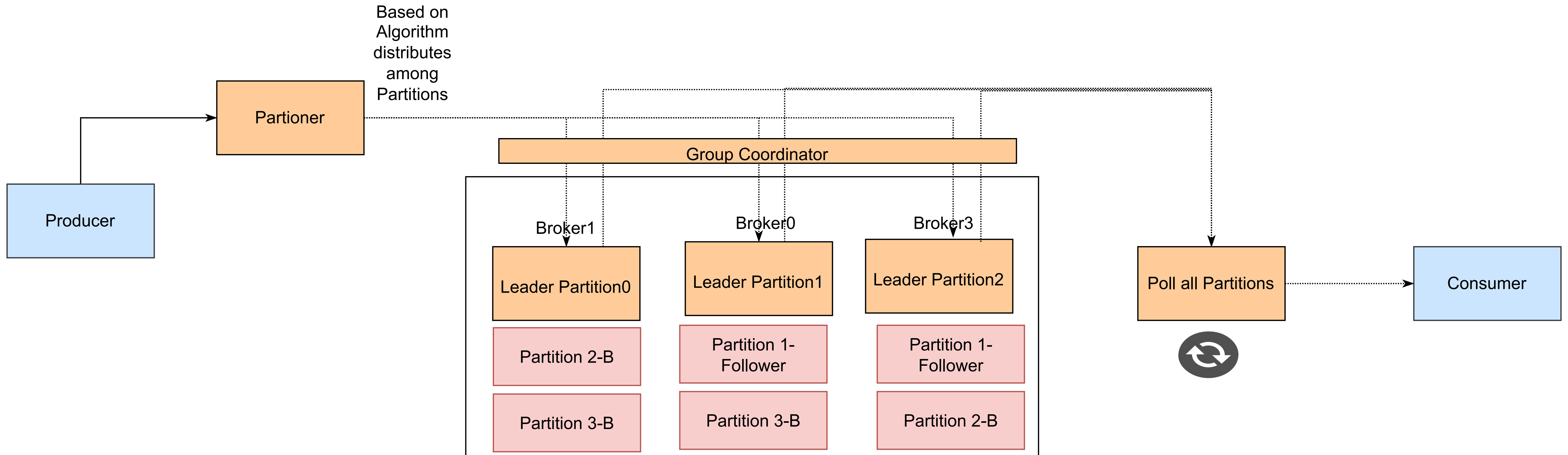


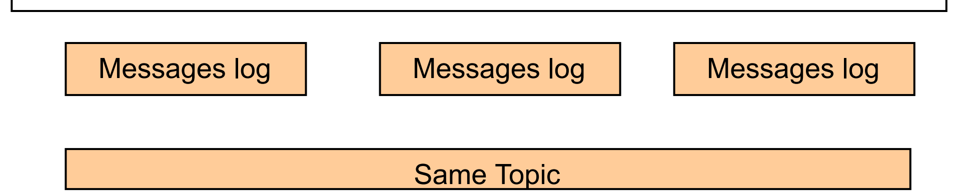

__consumer_offsets
obtain offset from internal queue using groupid
Thread
Consumer
Offset is differently maintained for this CG
Consumer
Single Thread
Poll Partition 2 but obtain offset from internal queue using groupid
Single Thread
Consumer (this wil be Idle)

Standalone Topology

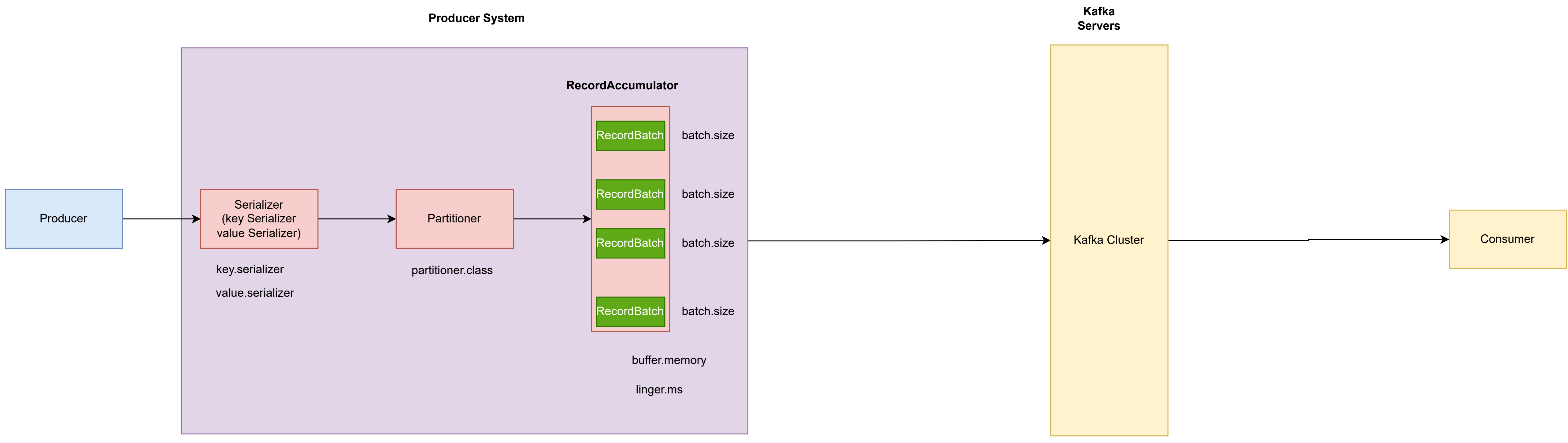


Distributed Topology

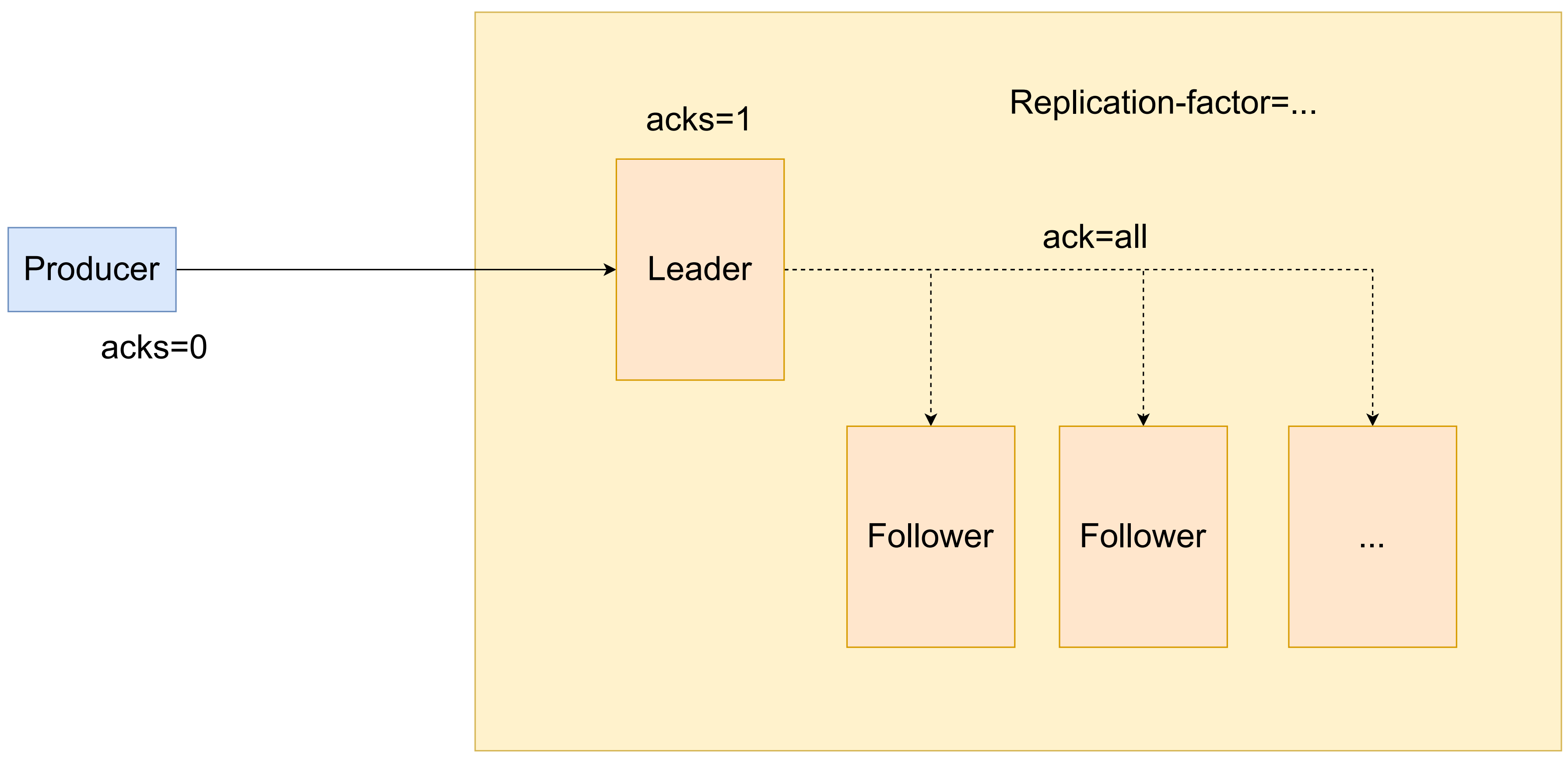



Messages log
Messages log
Messages log
Same Topic

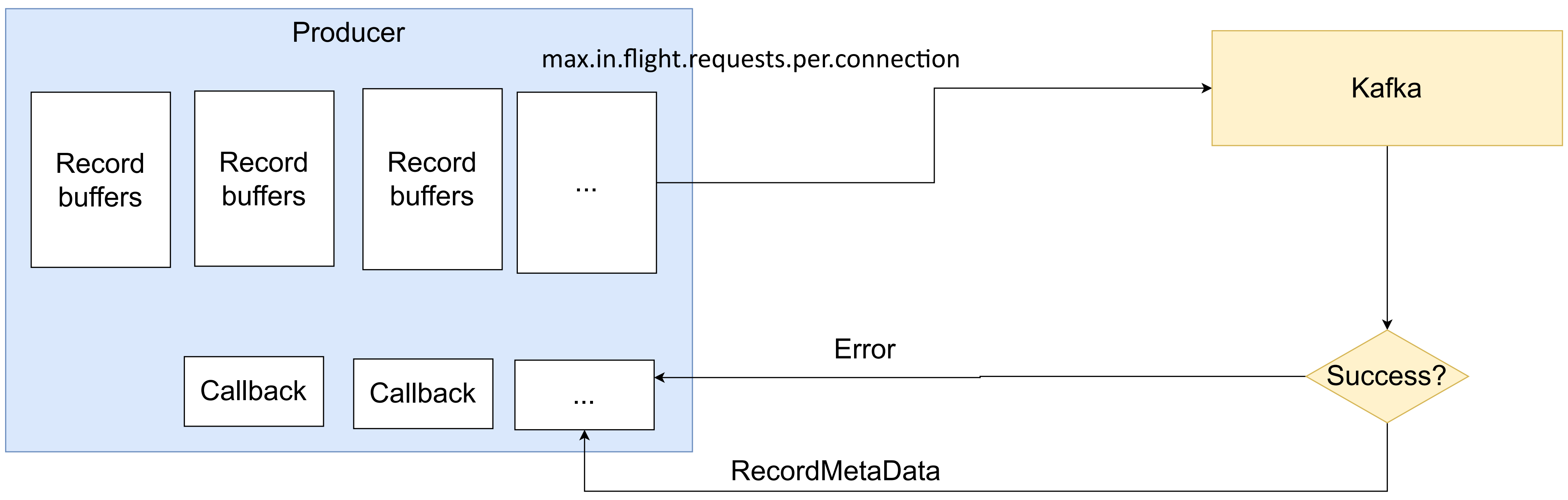
Producer Application Internals & Configurations



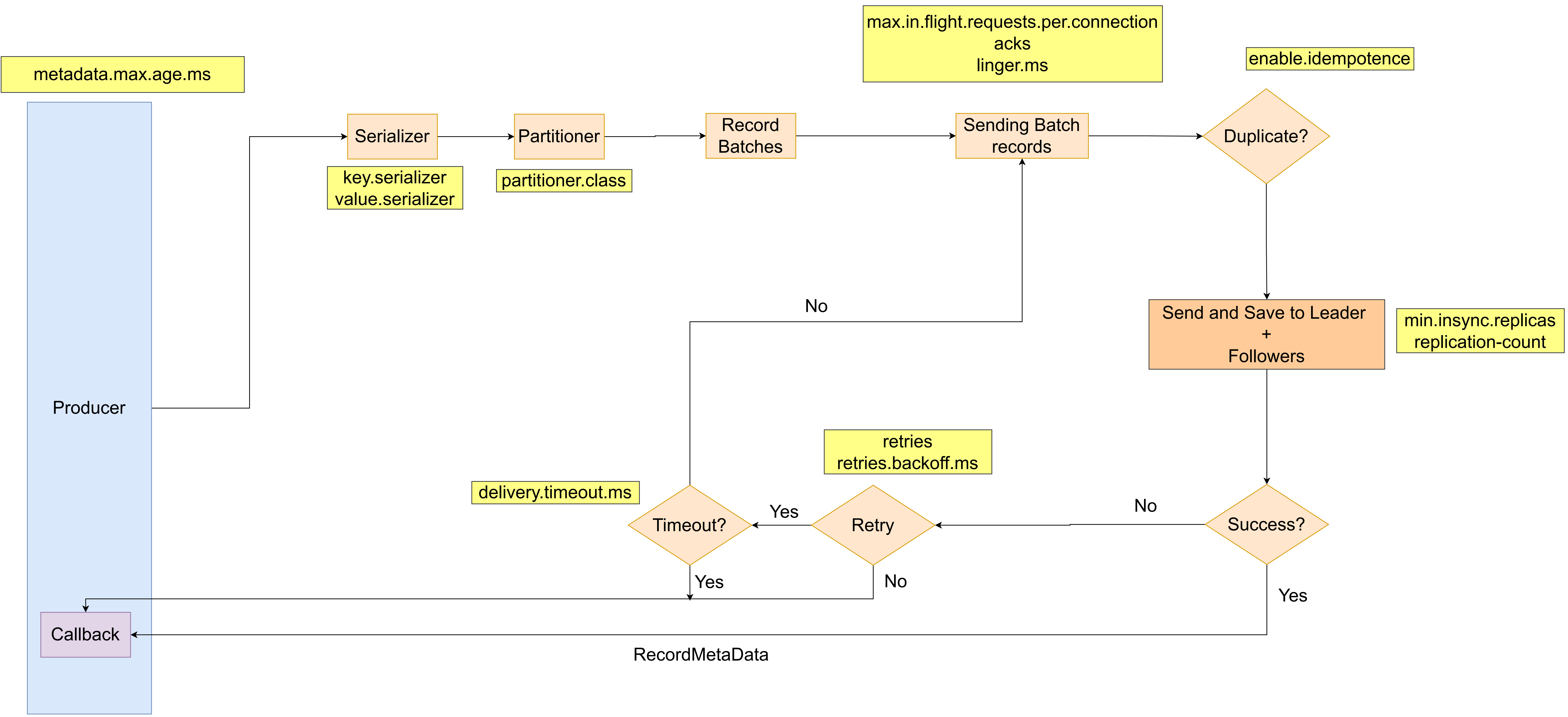
ack configuration



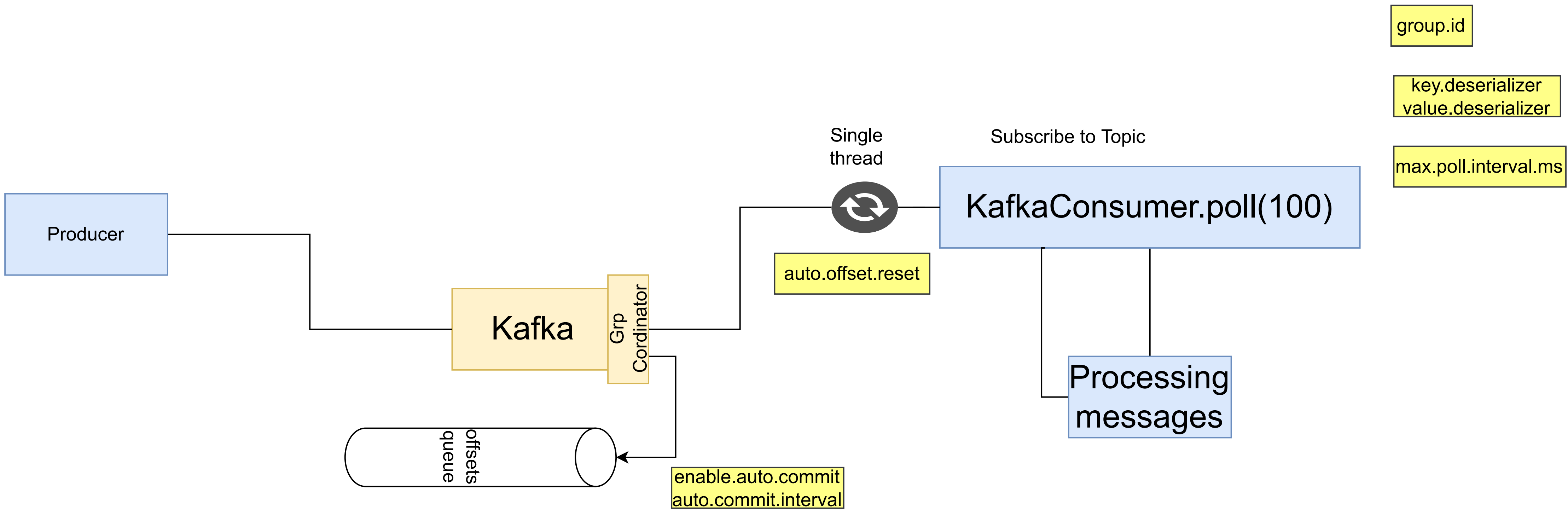
max.in.flight.requests.per.connection Configuration



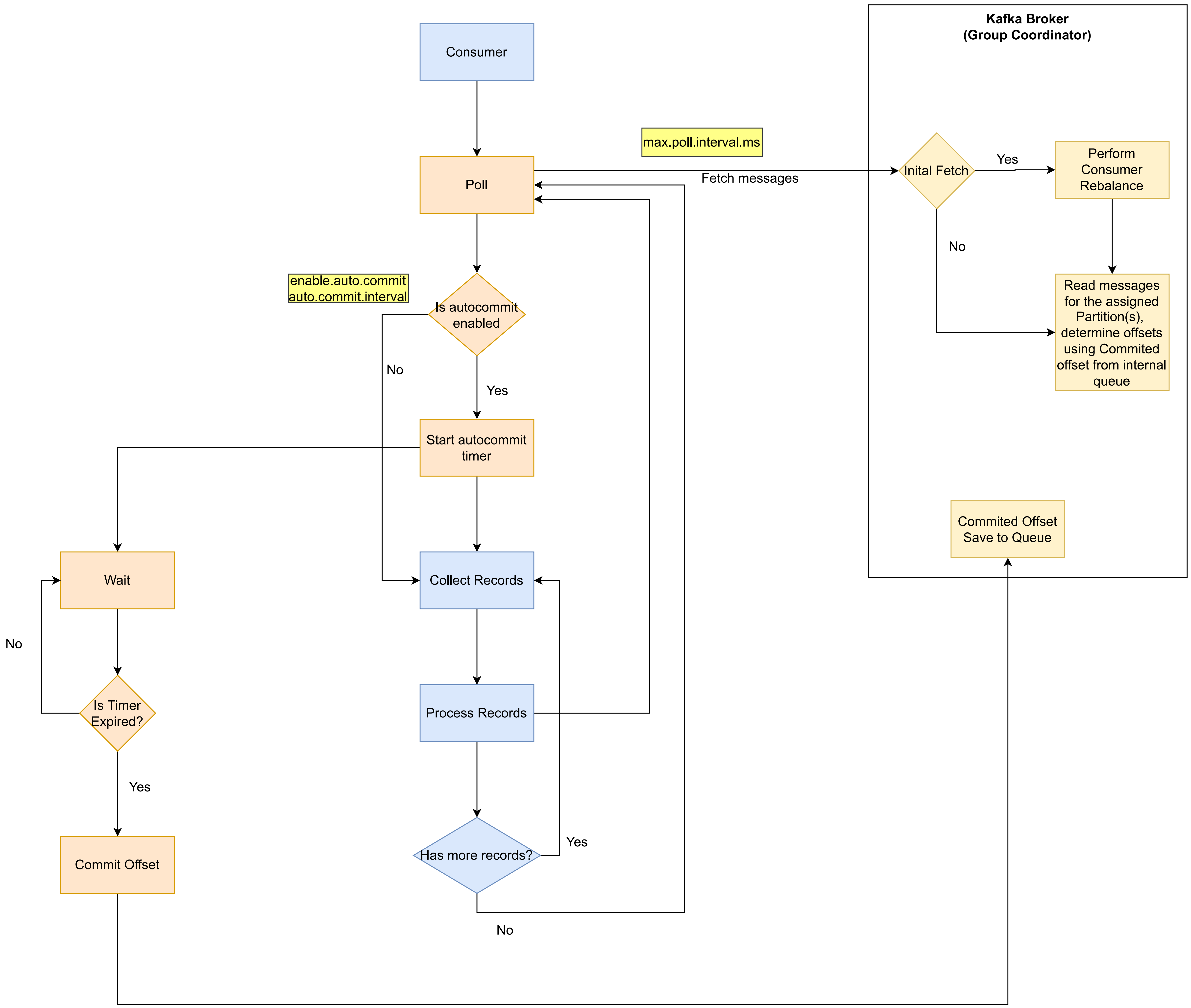
Consolidated Flow



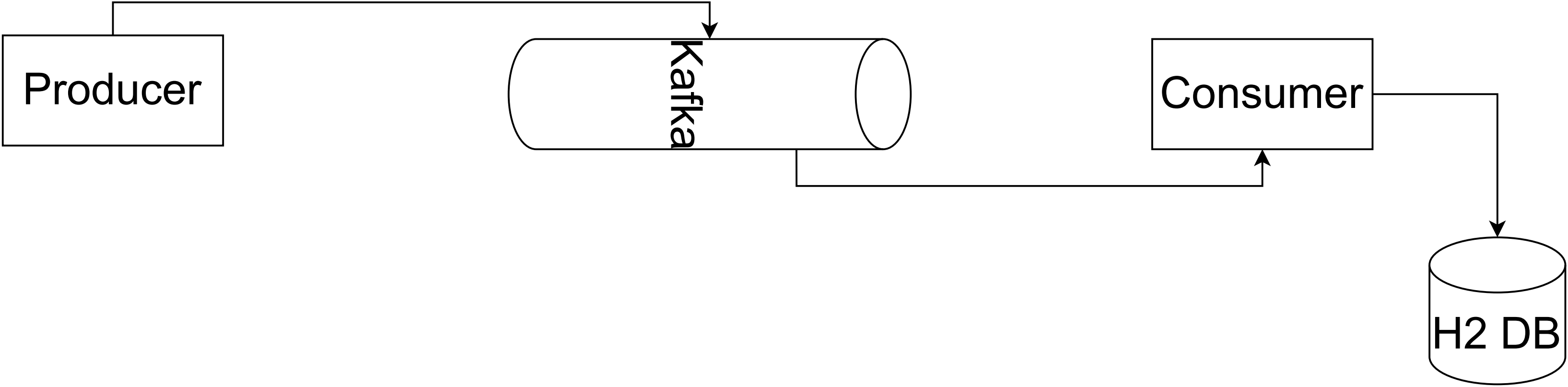
Consumer Pooling

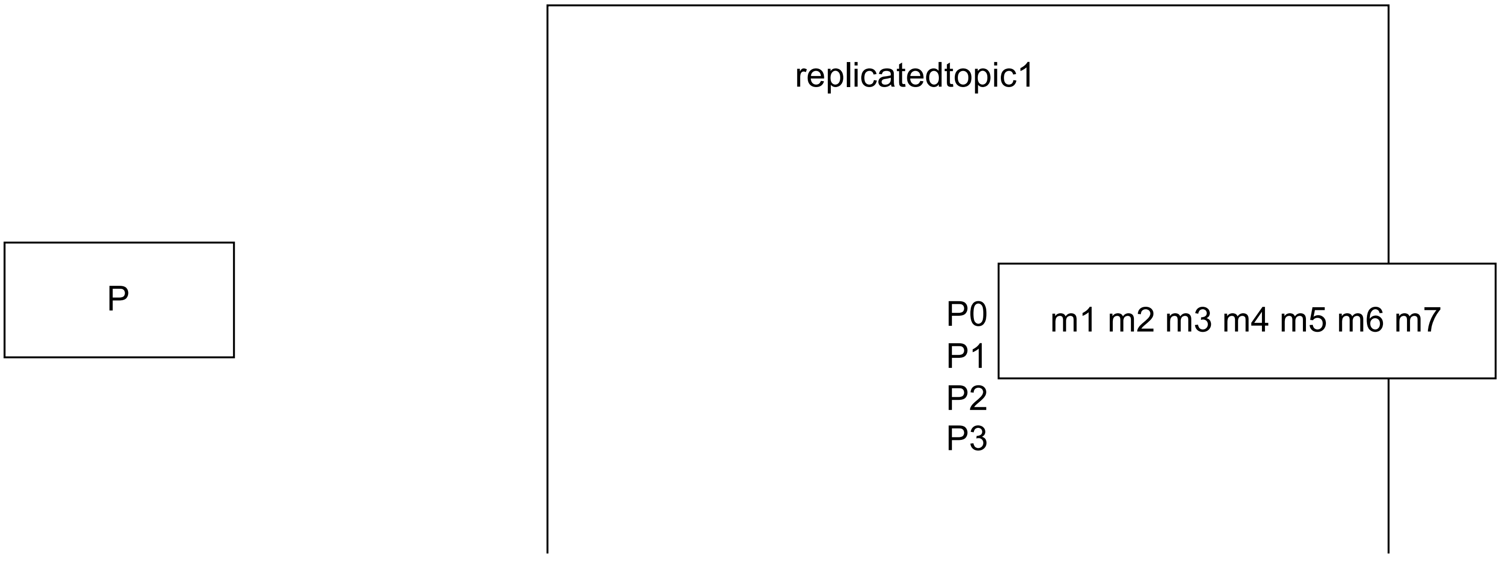


Consumer Internal Working



Persistency of read messages



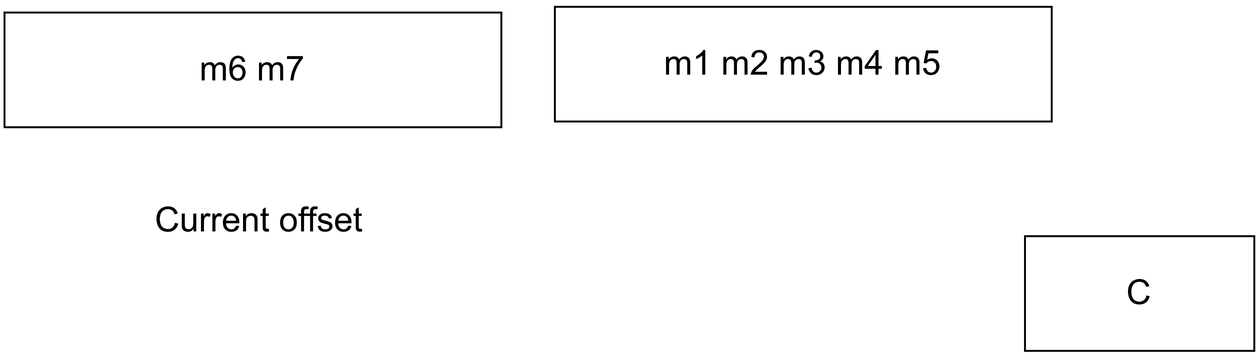


P

replicatedtopic1

P0
P1
P2
P3

m1 m2 m3 m4 m5 m6 m7

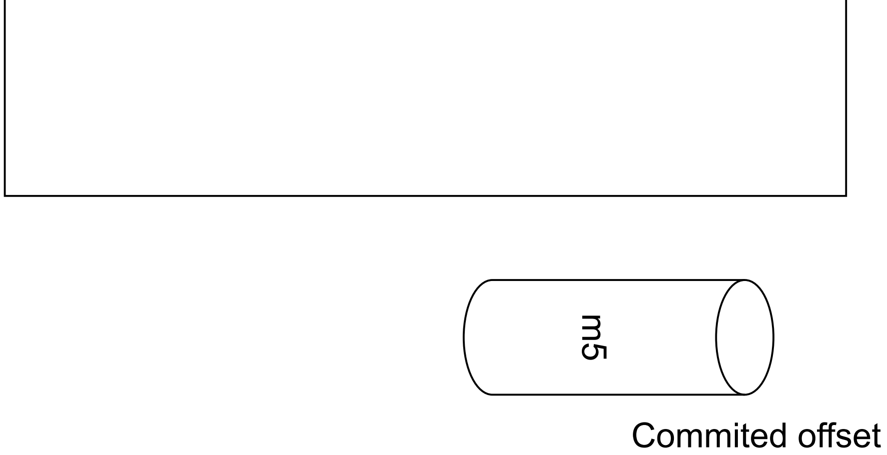


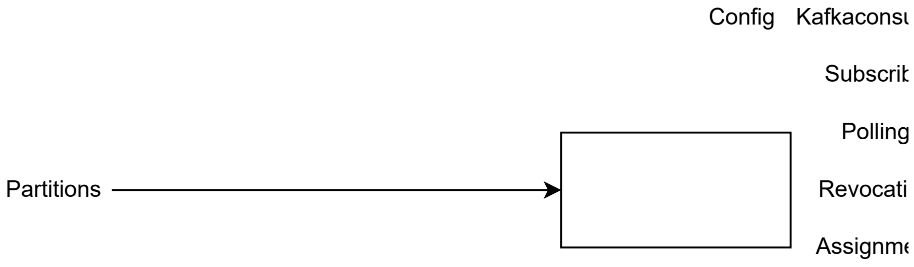
m6 m7

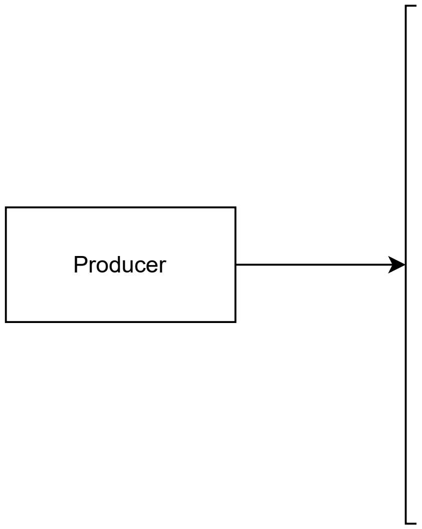
m1 m2 m3 m4 m5

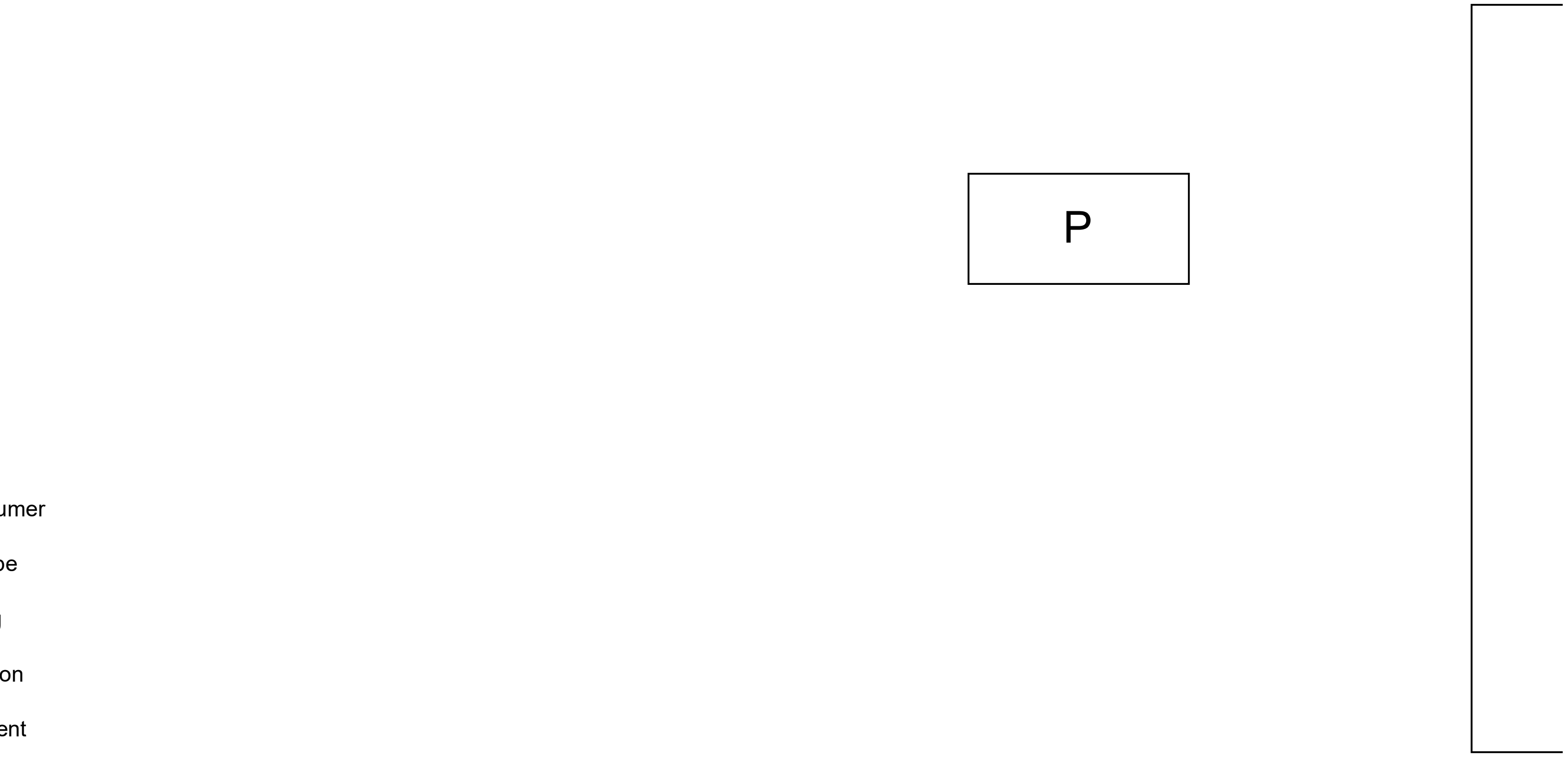
Current offset

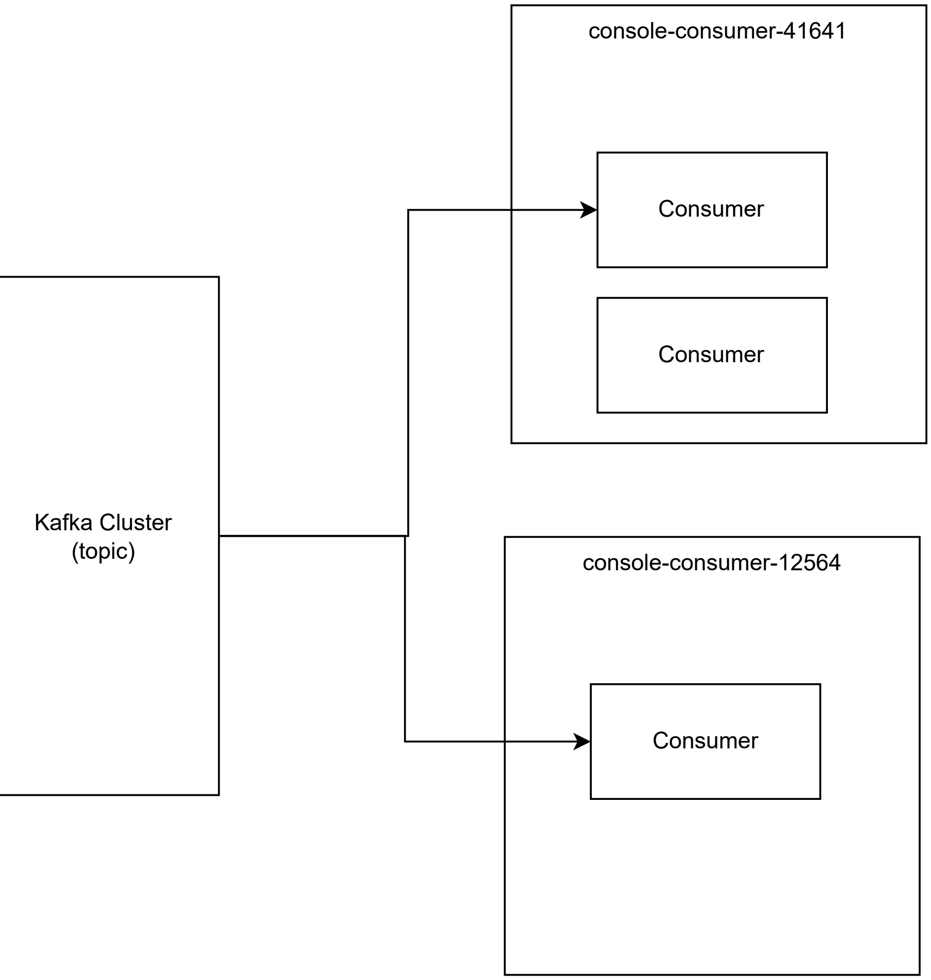
C

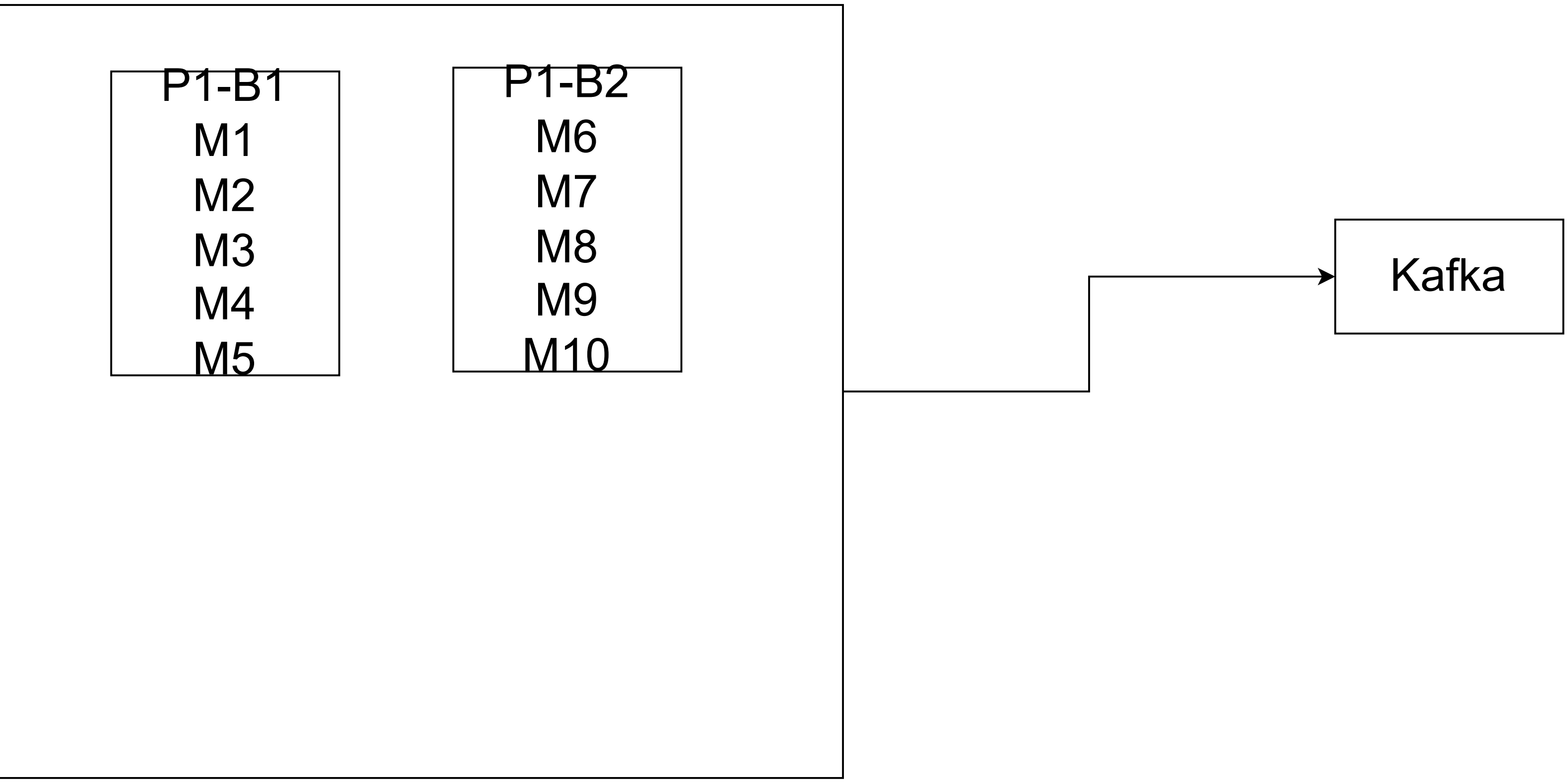

m5
Commited offset


Config
Kafkaconsu
Subscrib
Polling
Partitions
Revocati
Assignme


Producer


P
umer
pe
on
ent


console-consumer-41641
Consumer
Consumer
Kafka Cluster
(topic)
console-consumer-12564
Consumer


P1-B1
M1
M2
M3
M4
M5
P1-B2
M6
M7
M8
M9
M10
Kafka

commitSync
Commit Async

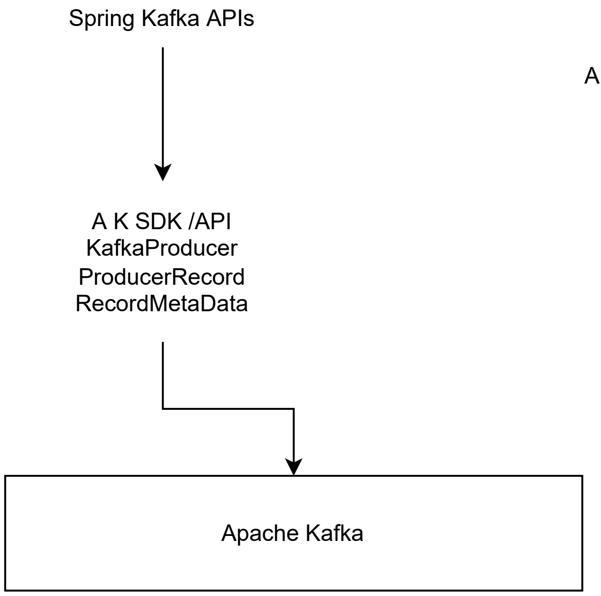
Last
polled
offset

Set of
offsets

Particula
offset

Properties file yml

AutoWiring of Kafka
objects



Spring Kafka APIs

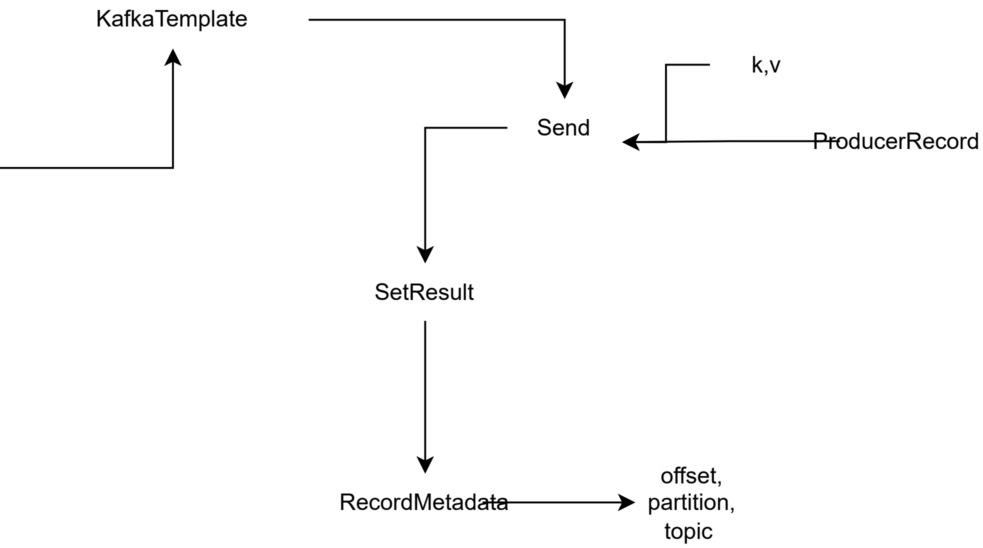
A K SDK /API
KafkaProducer
ProducerRecord
RecordMetaData

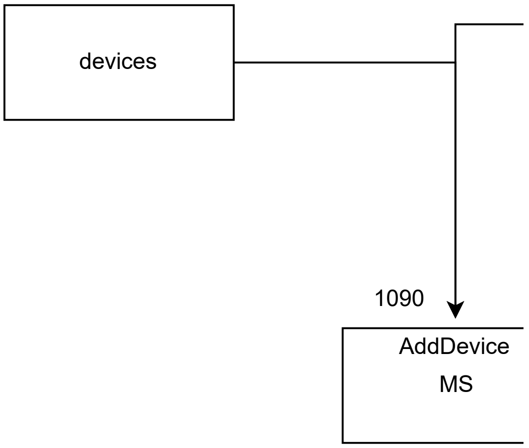
Apache Kafka

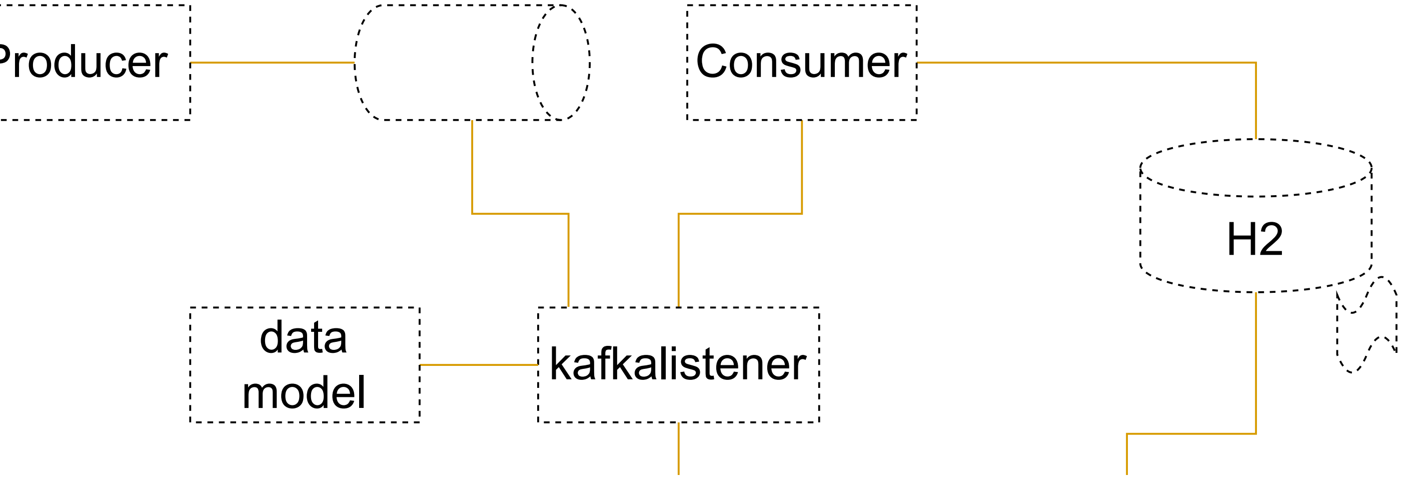
Application.yml

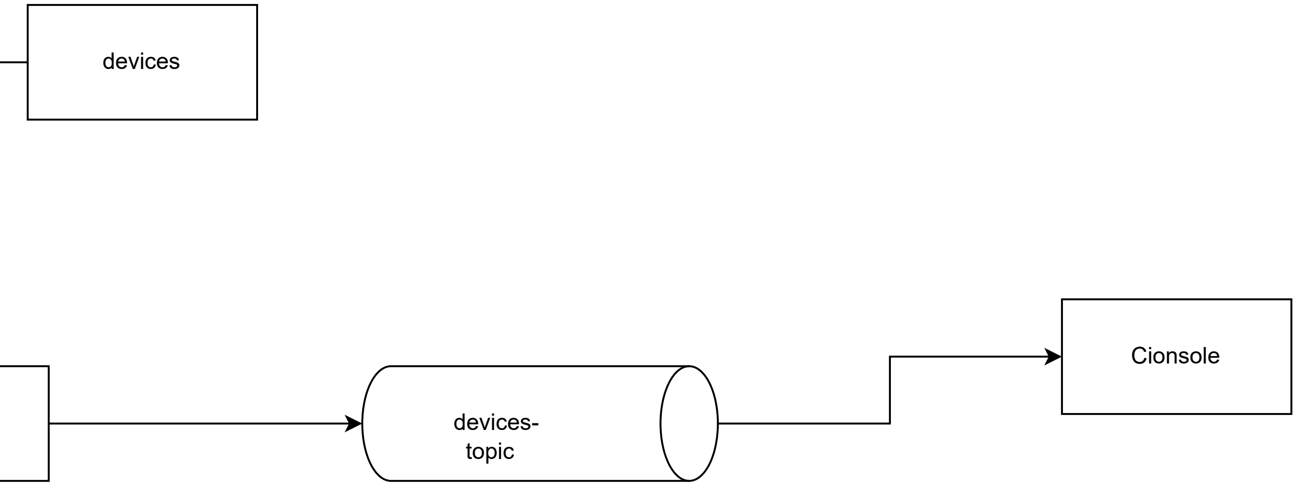
F

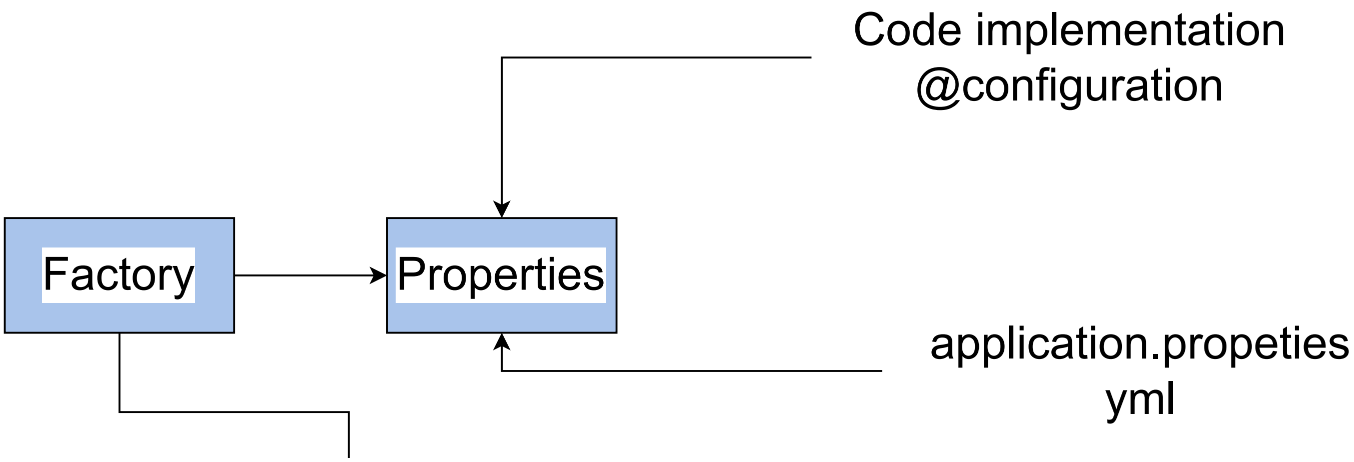
ar

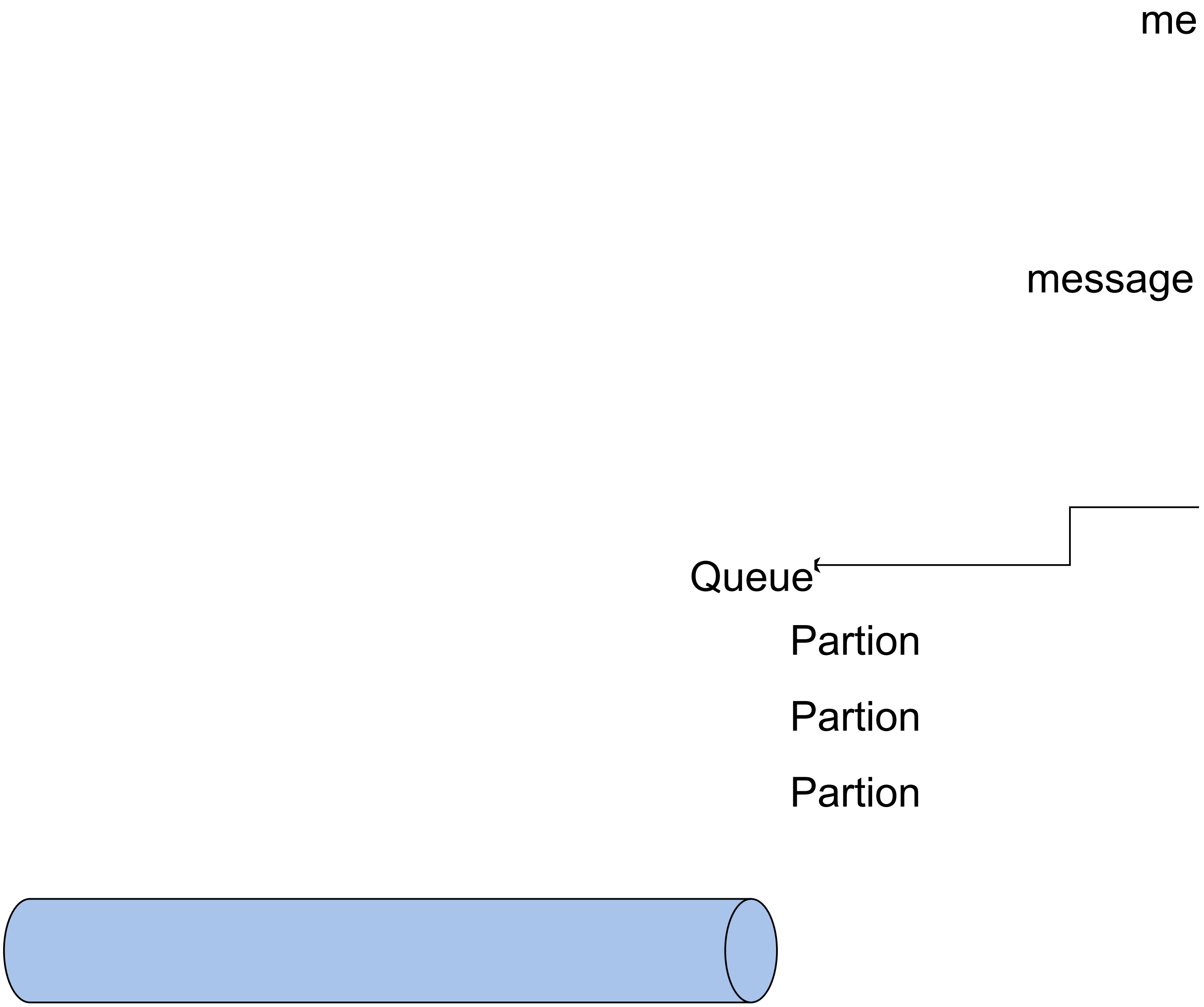

KafkaTemplate
k,v
Send
ProducerRecord
SetResult
RecordMetadata
offset,
partition,
topic


devices
1090
AddDevice
MS


Producer
Consumer
H2
data
model
kafkalistener


devices
devices-topic
Cionsole


Code implementation
@configuration
Factory
Properties
application.propeties
yml



me

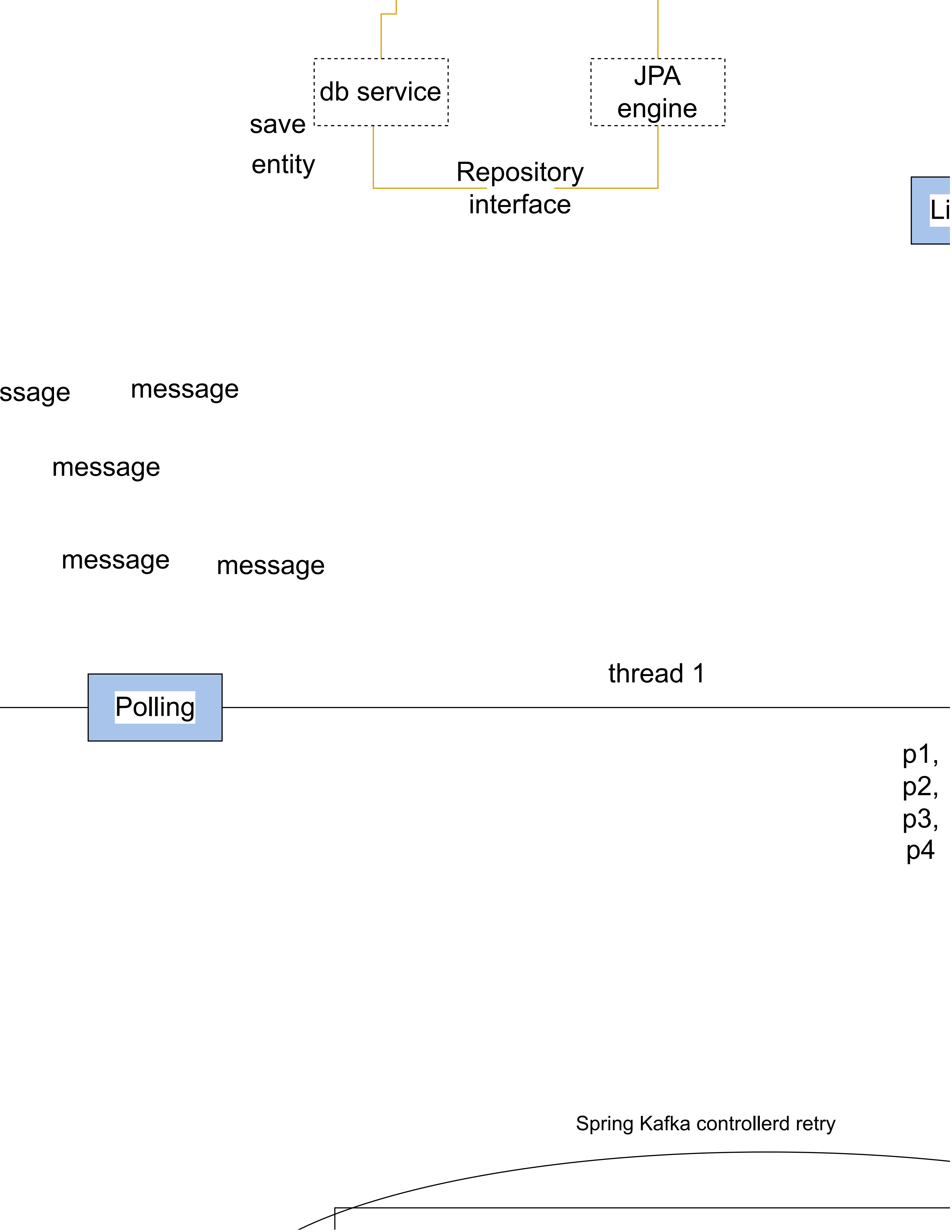
message

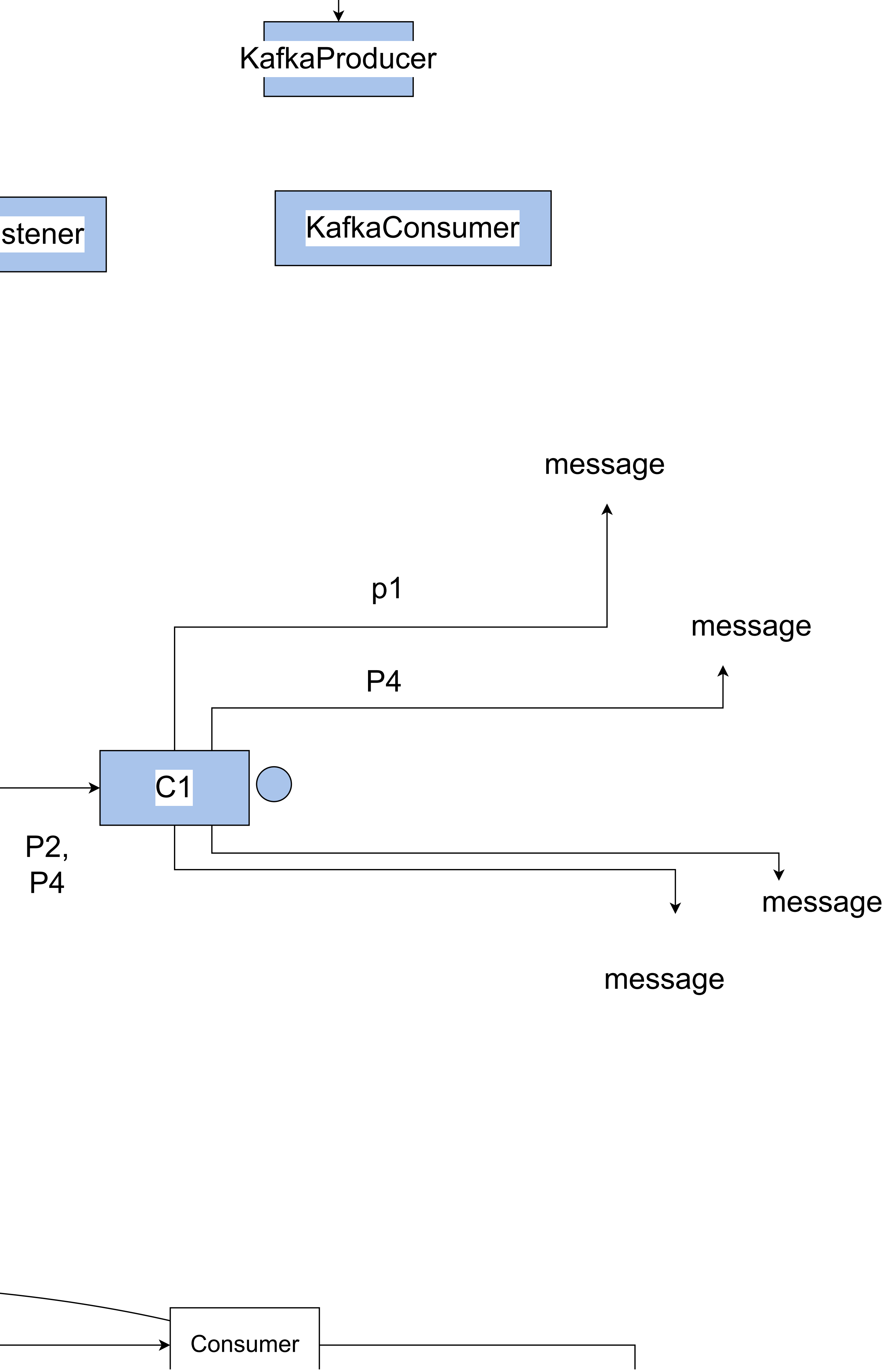
Queue

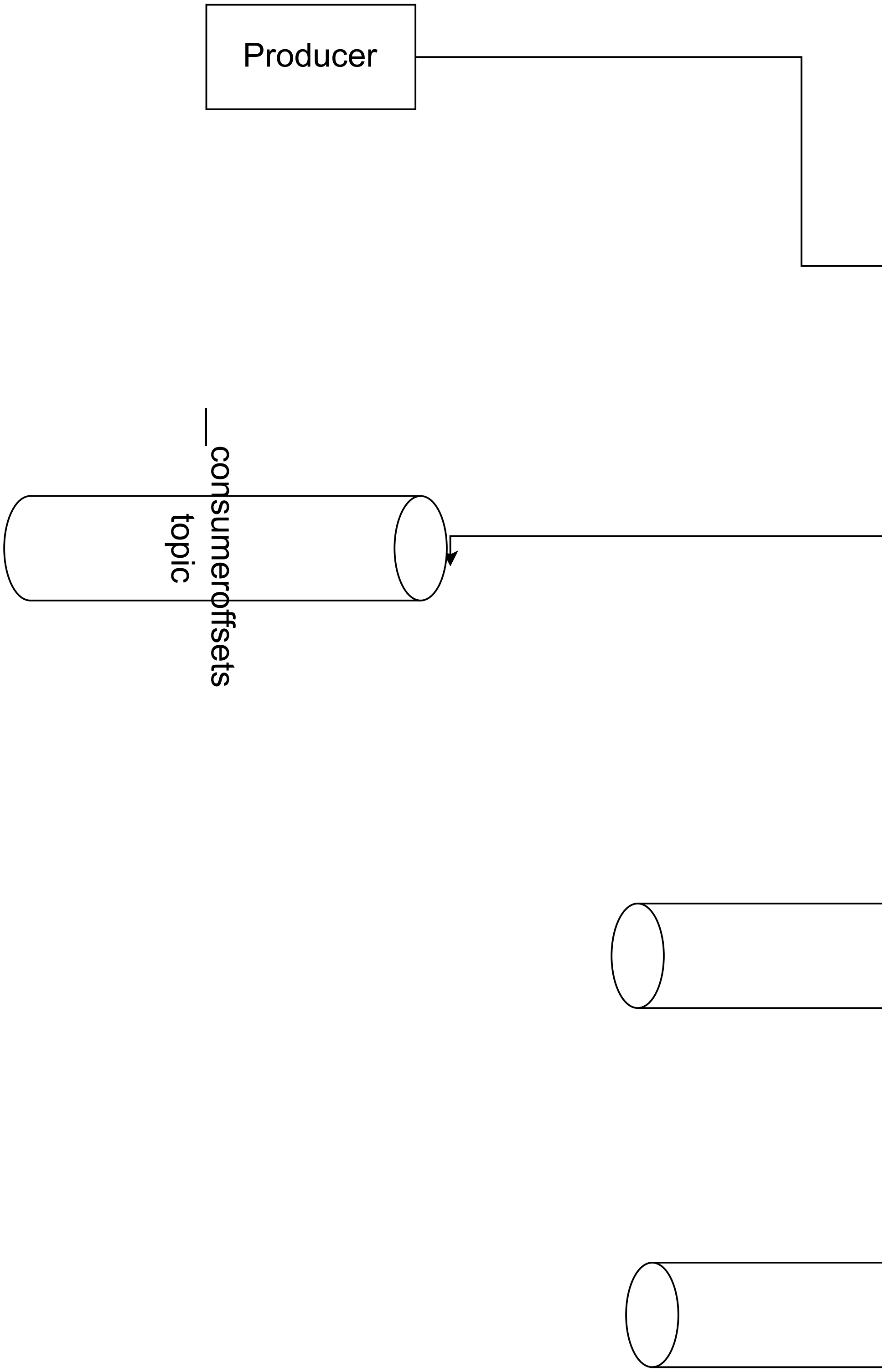
Partion

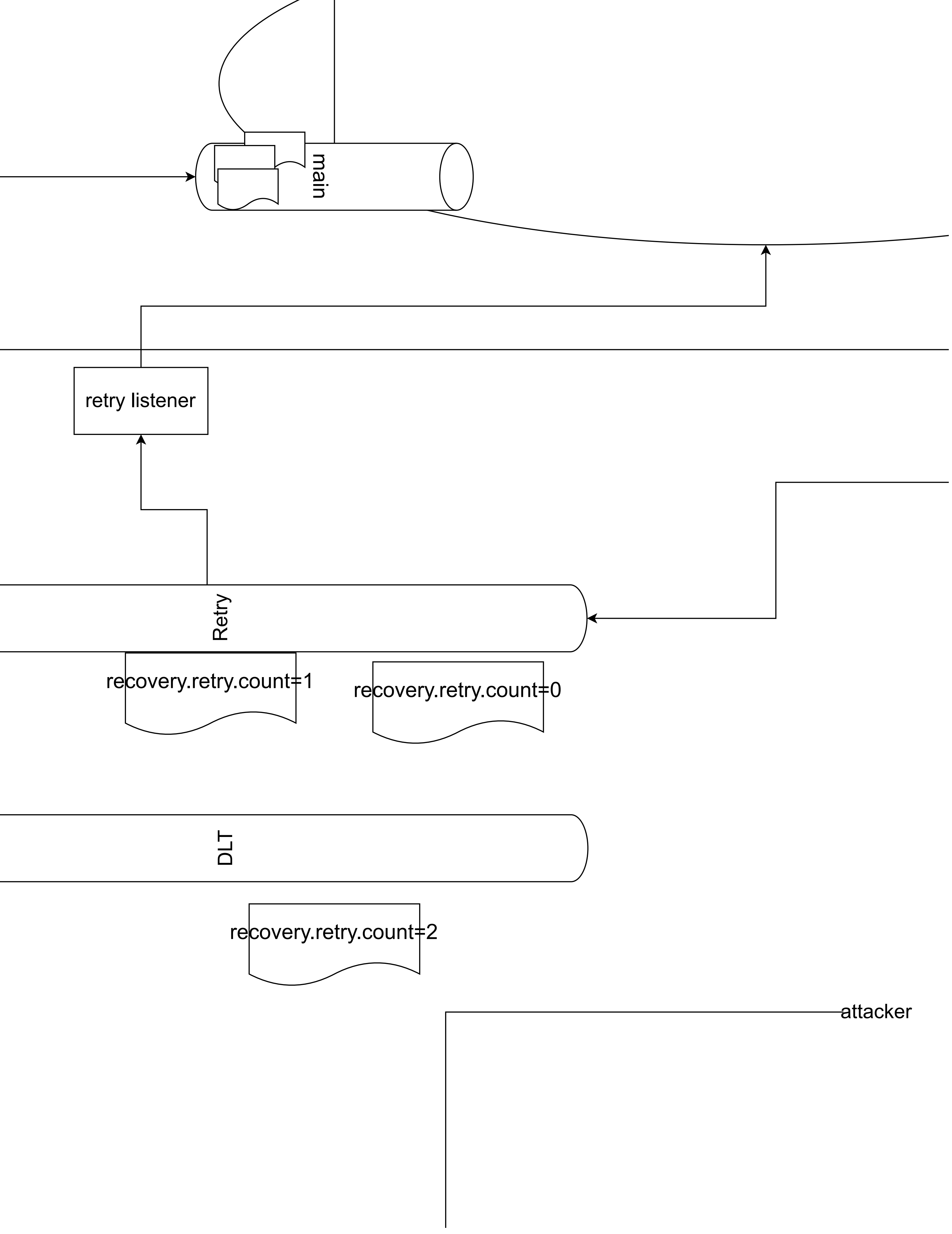
Partion

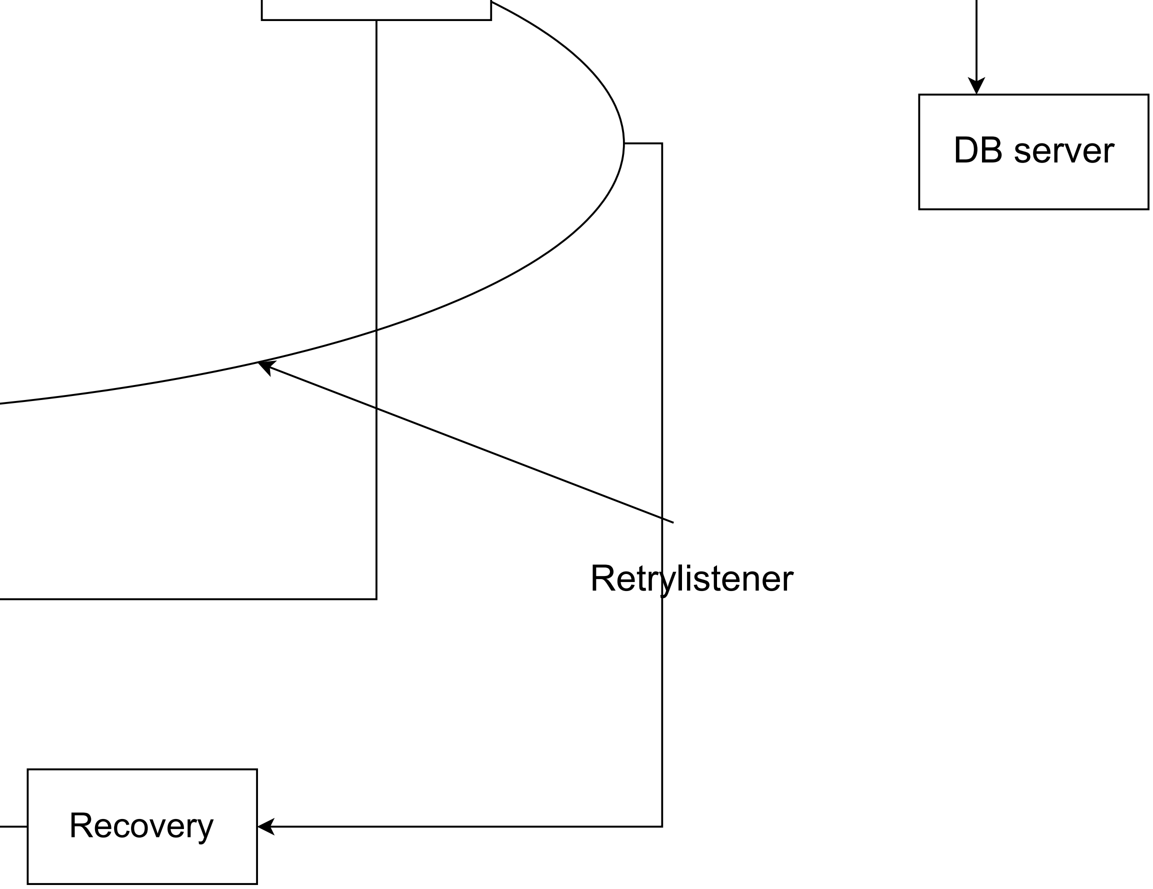
Partion

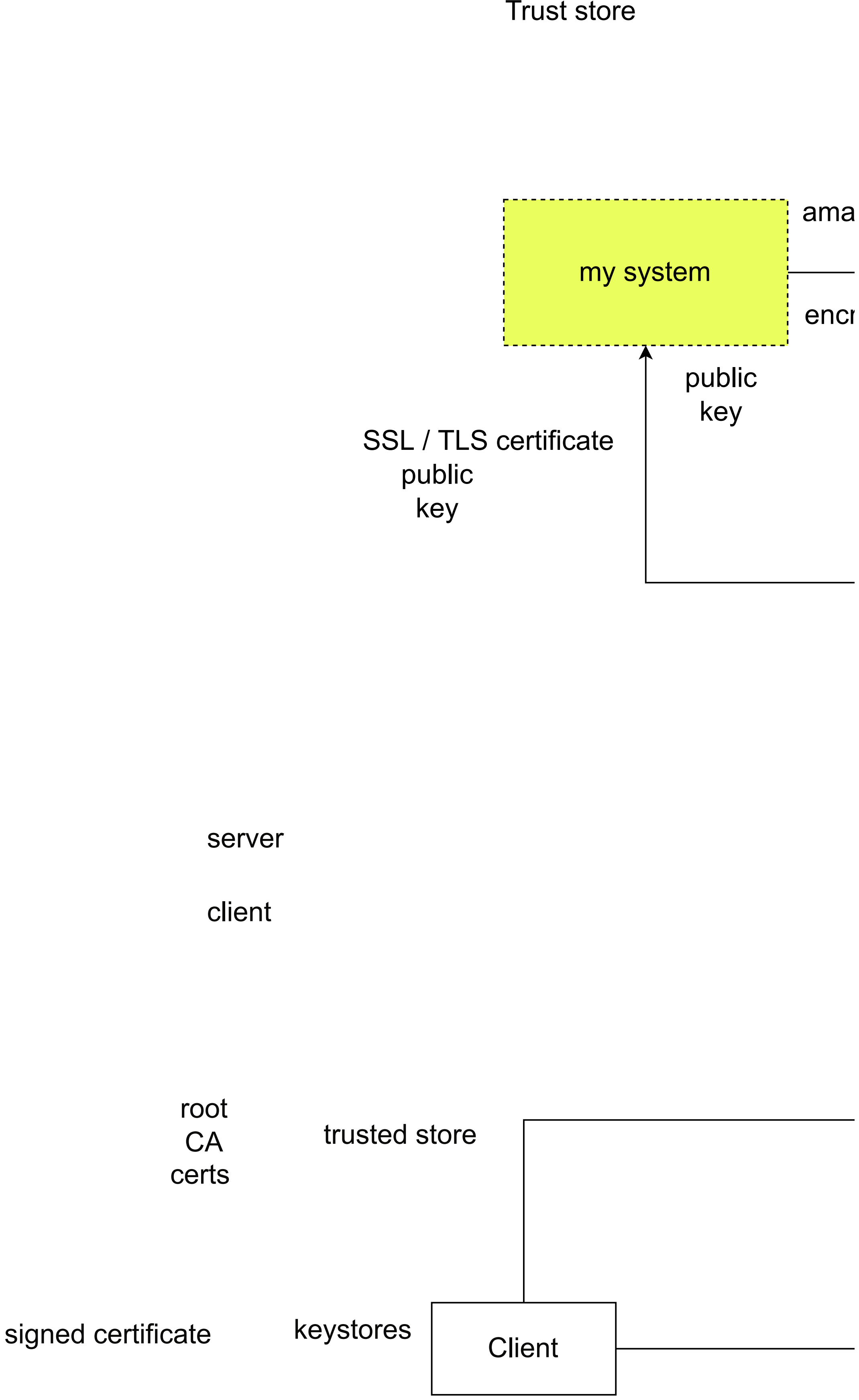

db service
JPA engine
save
entity
Repository interface
Li
ssage
message
message
message
message
Polling
thread 1
p1,
p2,
p3,
p4
Spring Kafka controllerd retry


KafkaProducer
stener
KafkaConsumer
message
p1
message
P4
C1
P2,
P4
message
message
Consumer


Producer
__consumeroffsets topic


main
retry listener
Retry
recovery.retry.count=1
recovery.retry.count=0
DLT
recovery.retry.count=2
attacker


DB server
Retrylistener
Recovery



Trust store

my system

ama

encr

public
key

SSL / TLS certificate
public
key

server

client

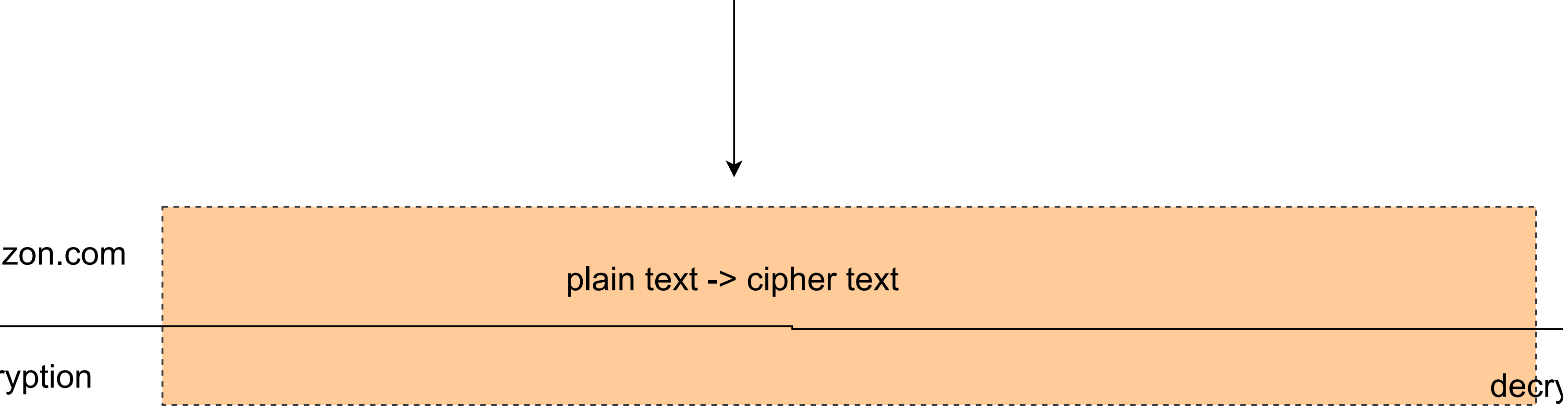
root
CA
certs

trusted store

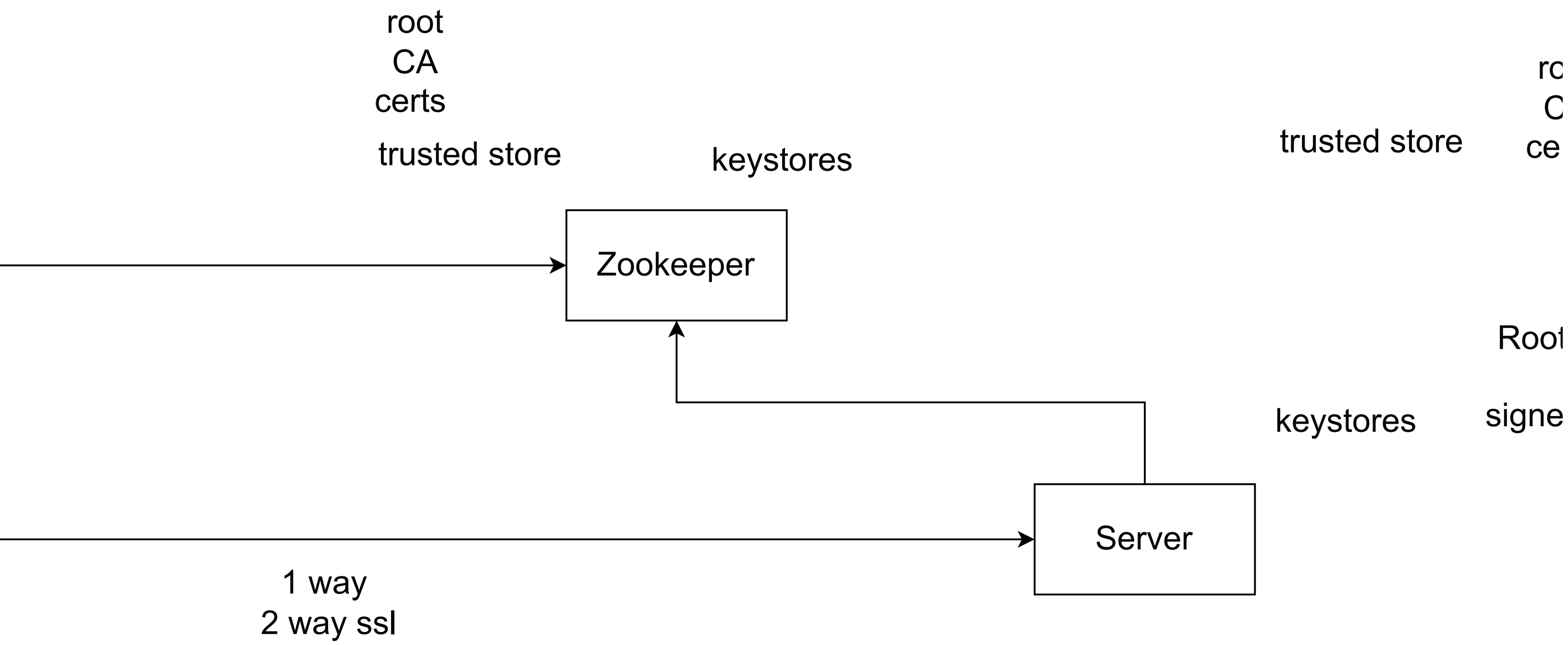
signed certificate

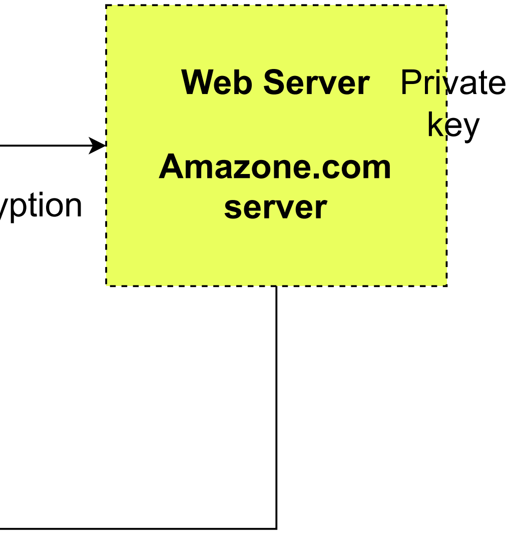
keystores

Client



SSL rough work area





public key

ot
A
rts

SSL
- Encryption/ Decruption

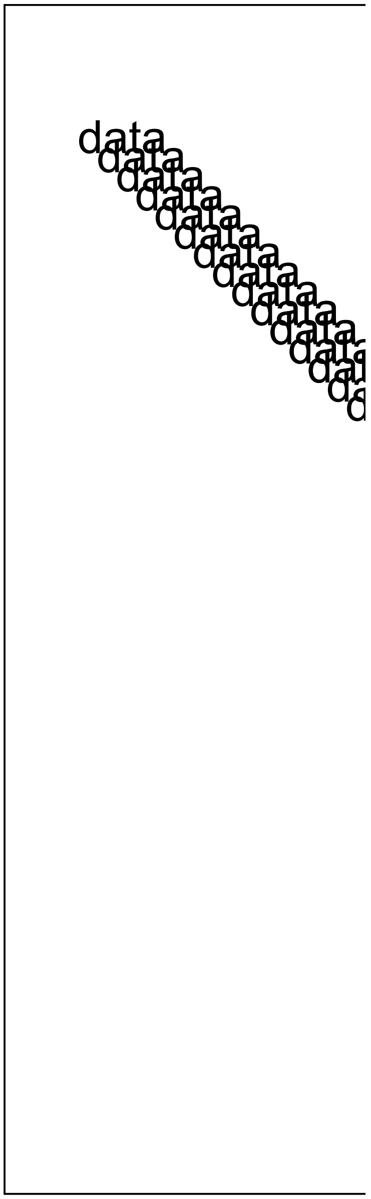
- Authentication

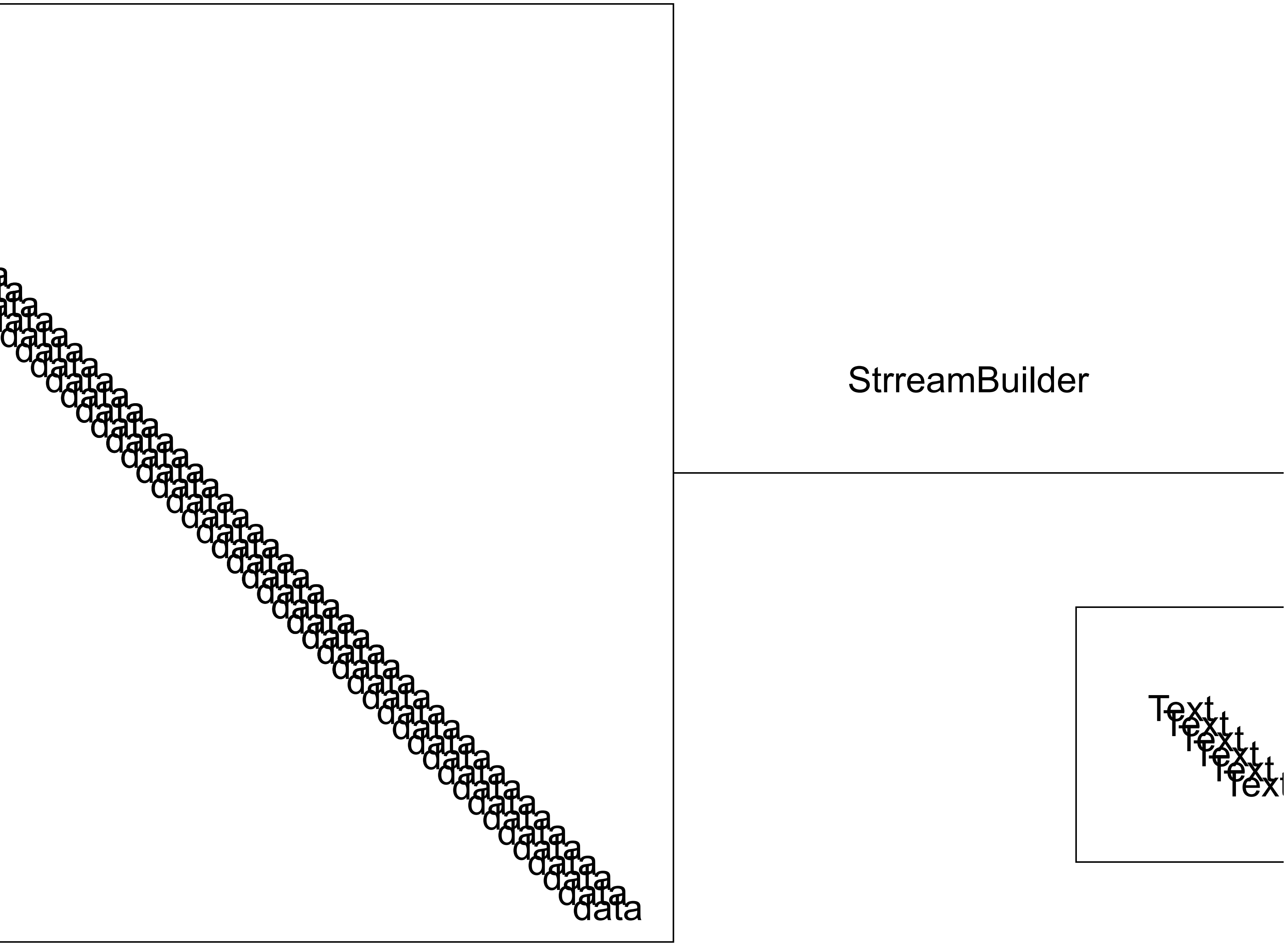
SSL certificates

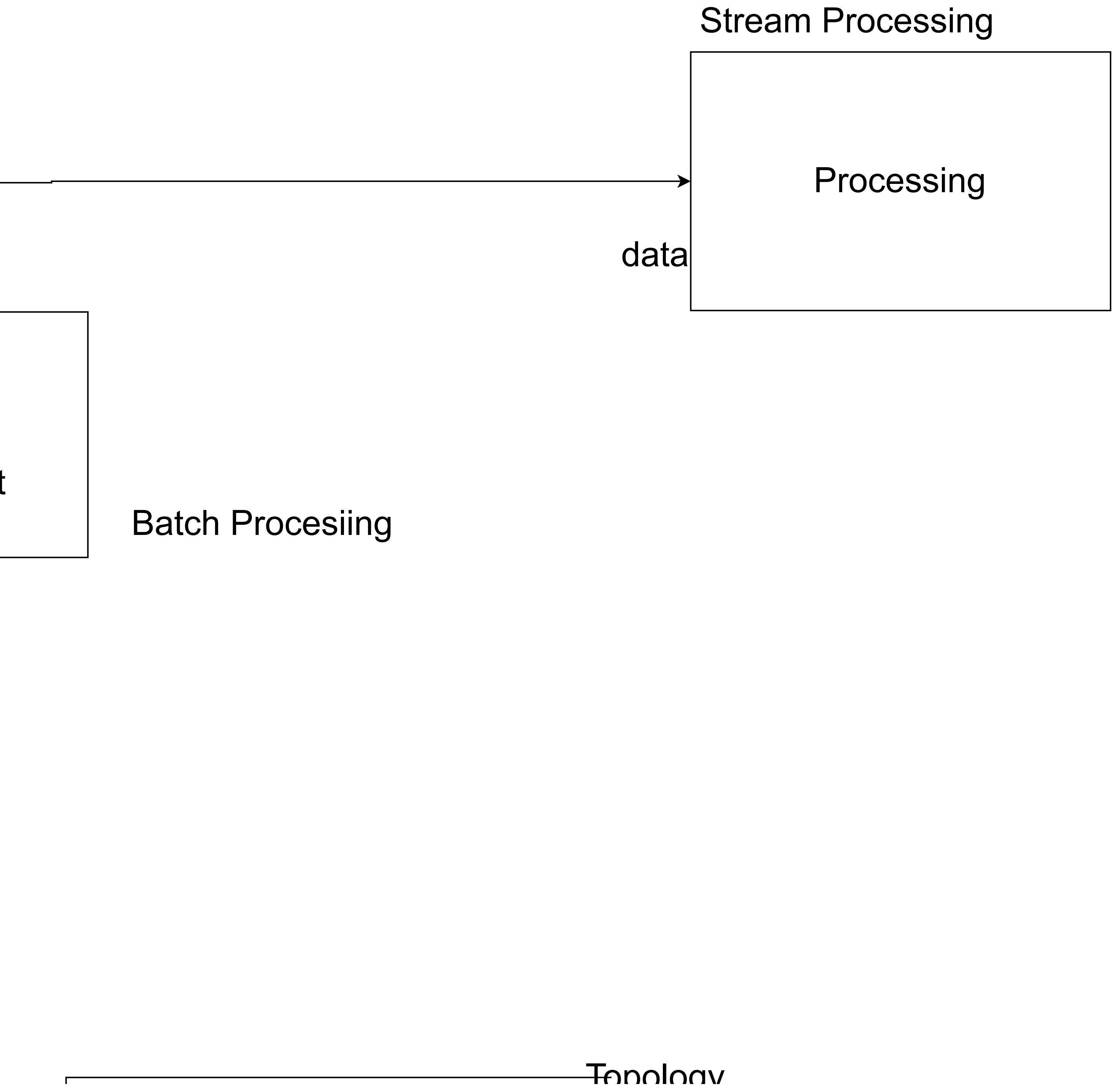
t CA cert

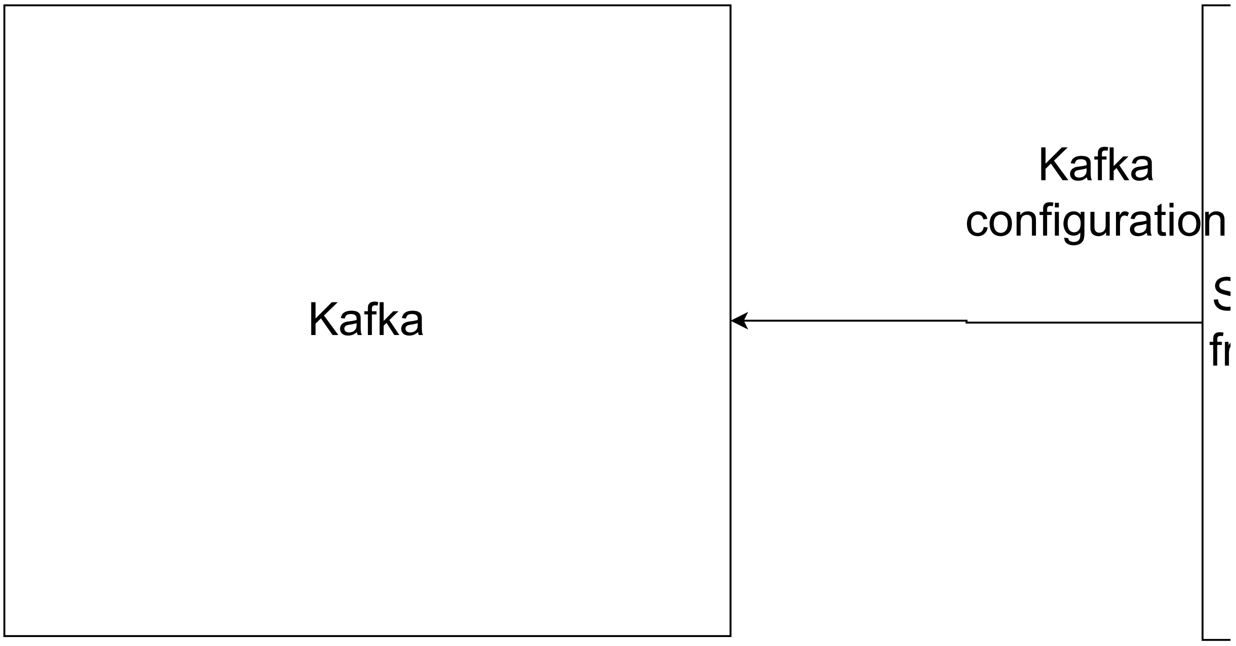
d certificate

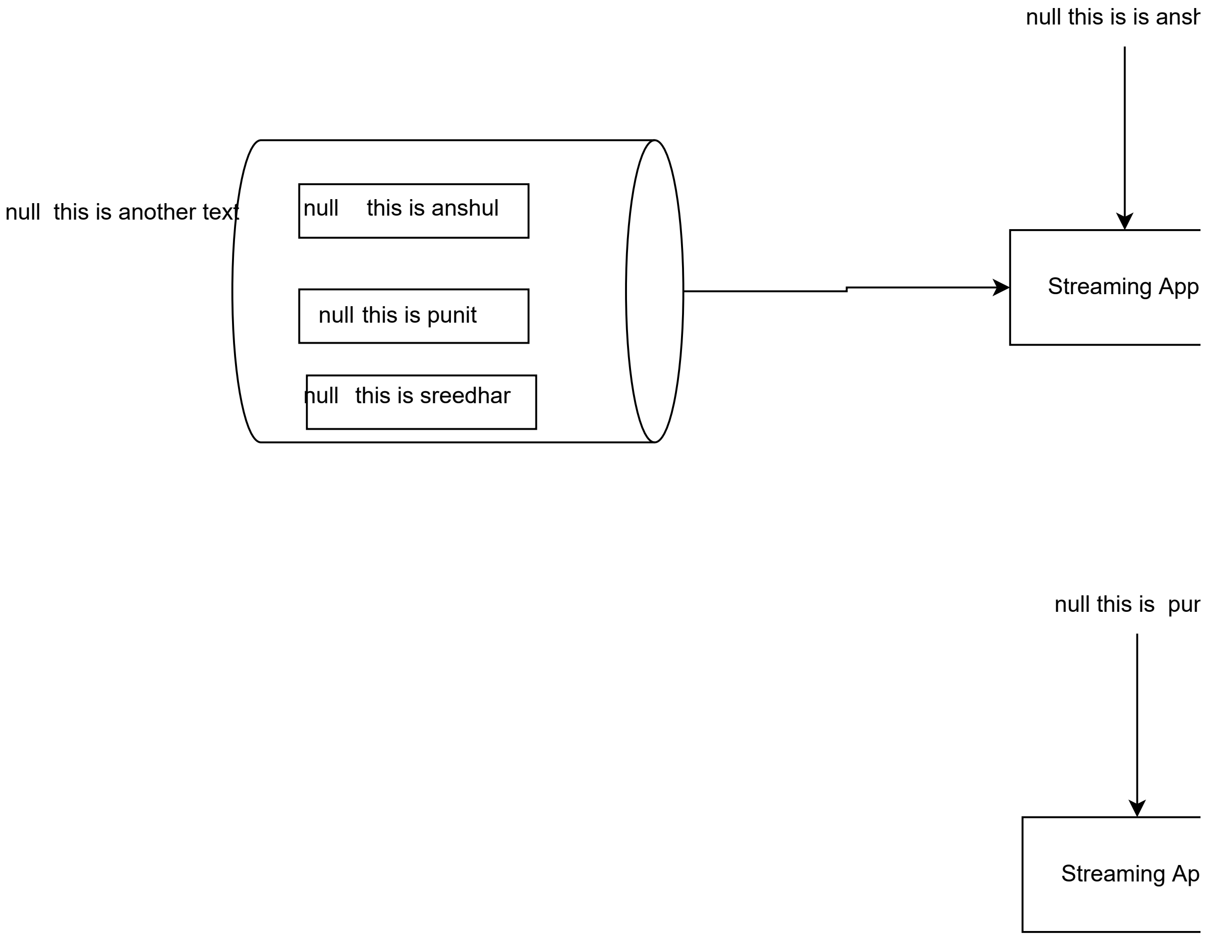
CA

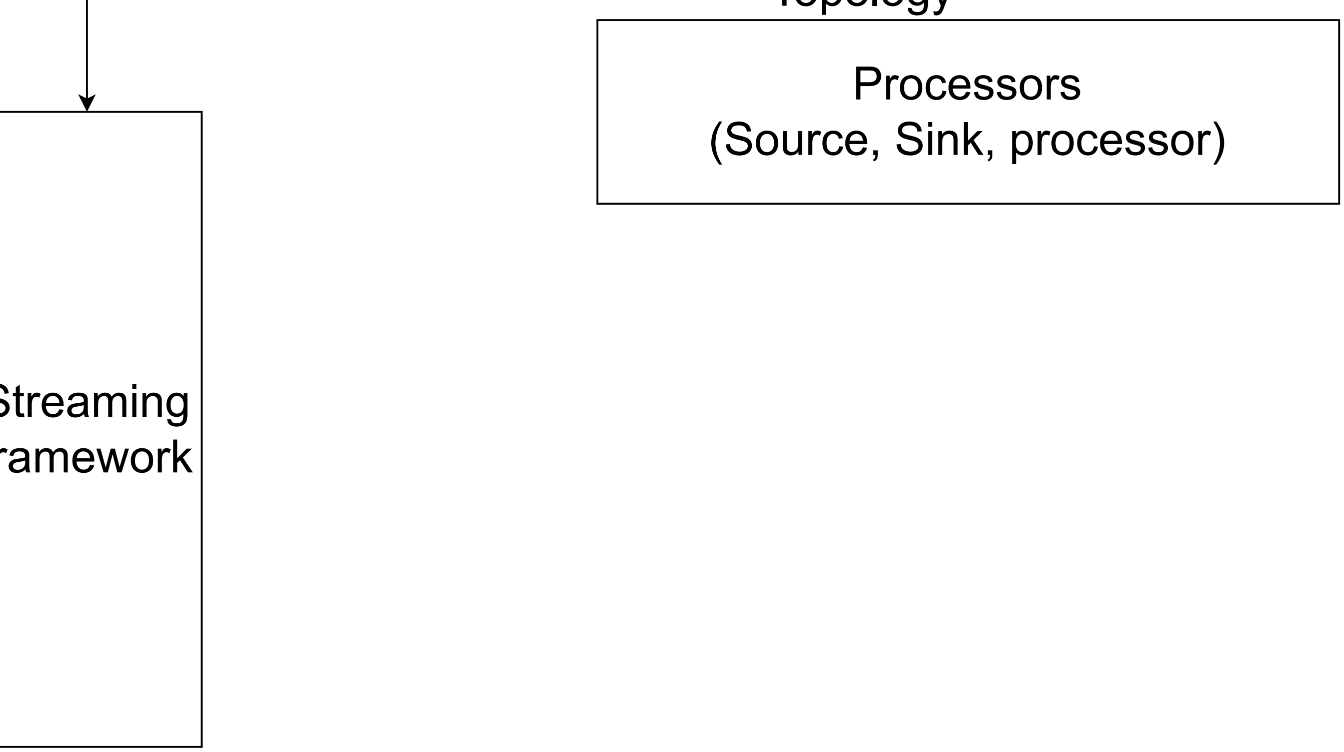

data
data
data
data
data
data
data
data
data
data
data


data
StrreamBuilder
Text


Stream Processing
Processing
data
t
Batch Procesiing
Topology


Kafka
Kafka configuration
S
fr

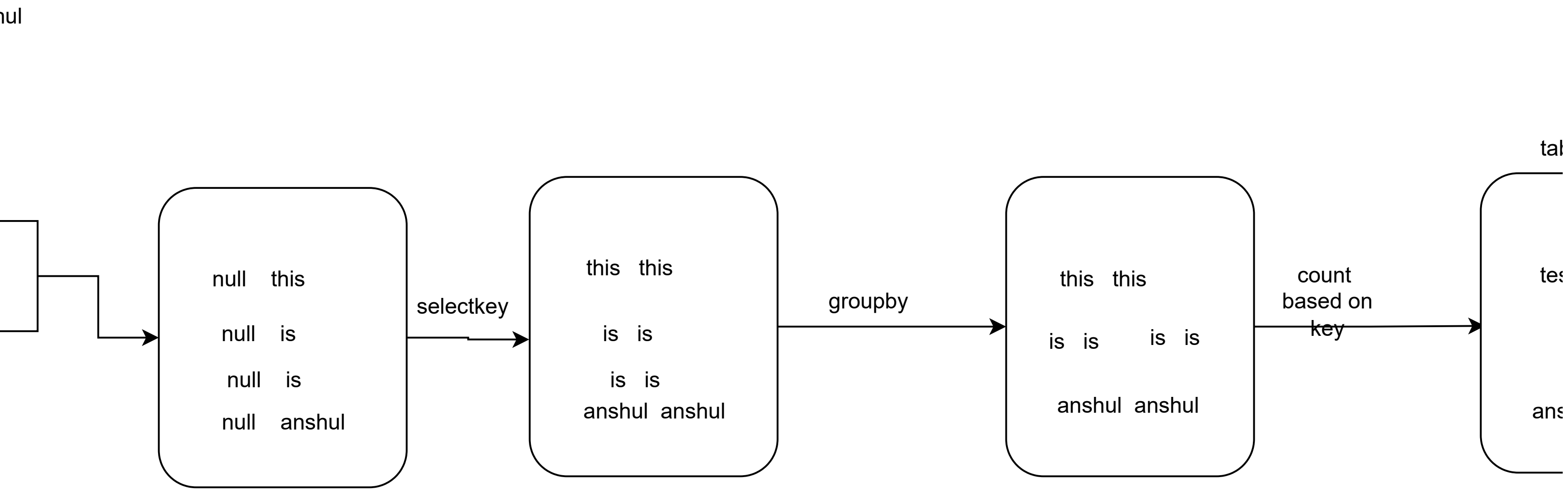

null this is is ansh
null this is another text
null this is anshul
null this is punit
null this is sreedhar
Streaming App
null this is pur
Streaming Ap



Topology

Processors
(Source, Sink, processor)

Streaming
Framework



hul

null this
null is
null is
null anshul

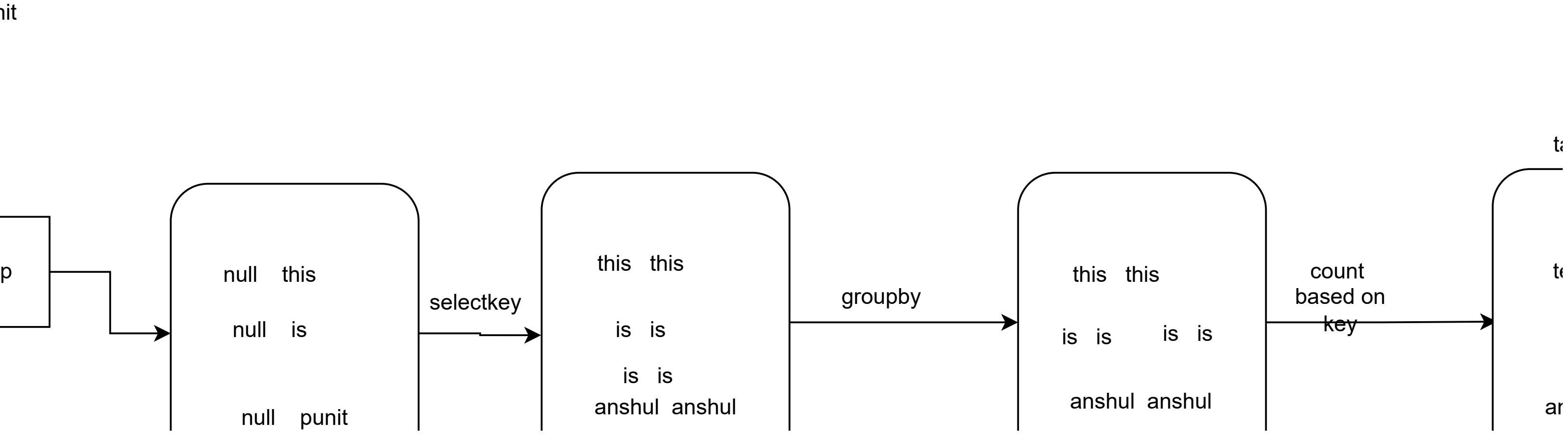
selectkey

this this
is is
is is
anshul anshul

groupby

this this
is is is is
anshul anshul

count
based on
key



nit

p

null this
null is
null punit

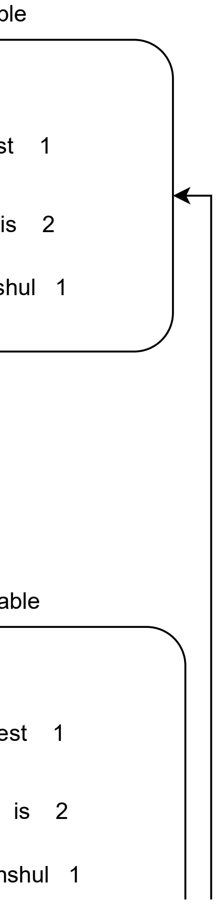
selectkey

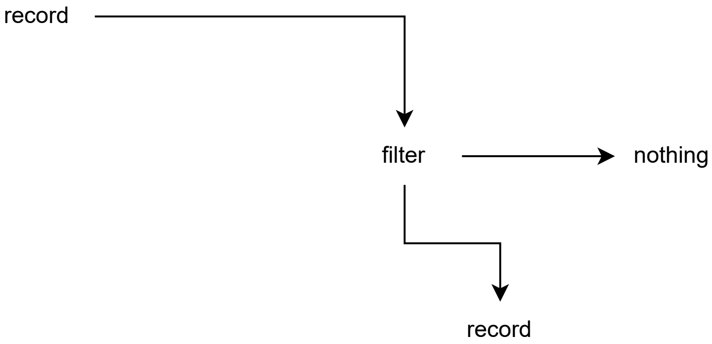
this this
is is
is is
anshul anshul

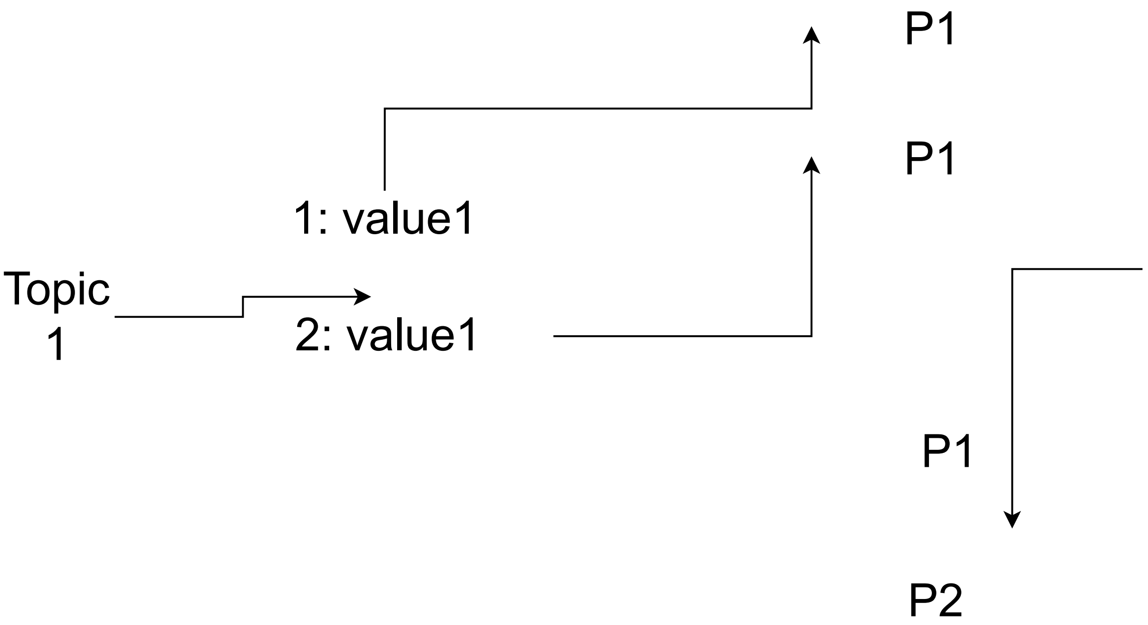
groupby

this this
is is is is
anshul anshul

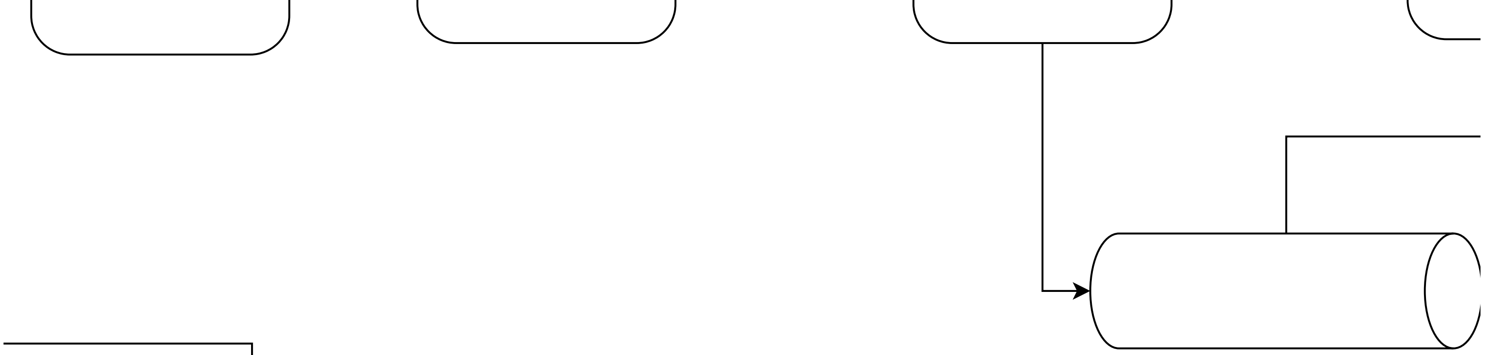
count
based on
key

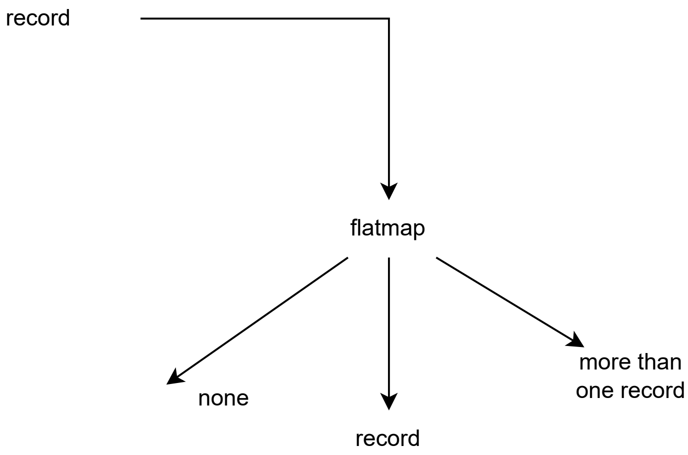

ble
st 1
is 2
shul 1
able
est 1
is 2
nshul 1





Repartition topic





map mapValue

selectkey

repartition

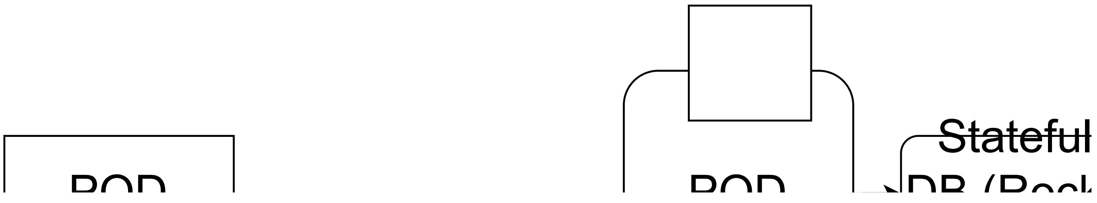
Changelog

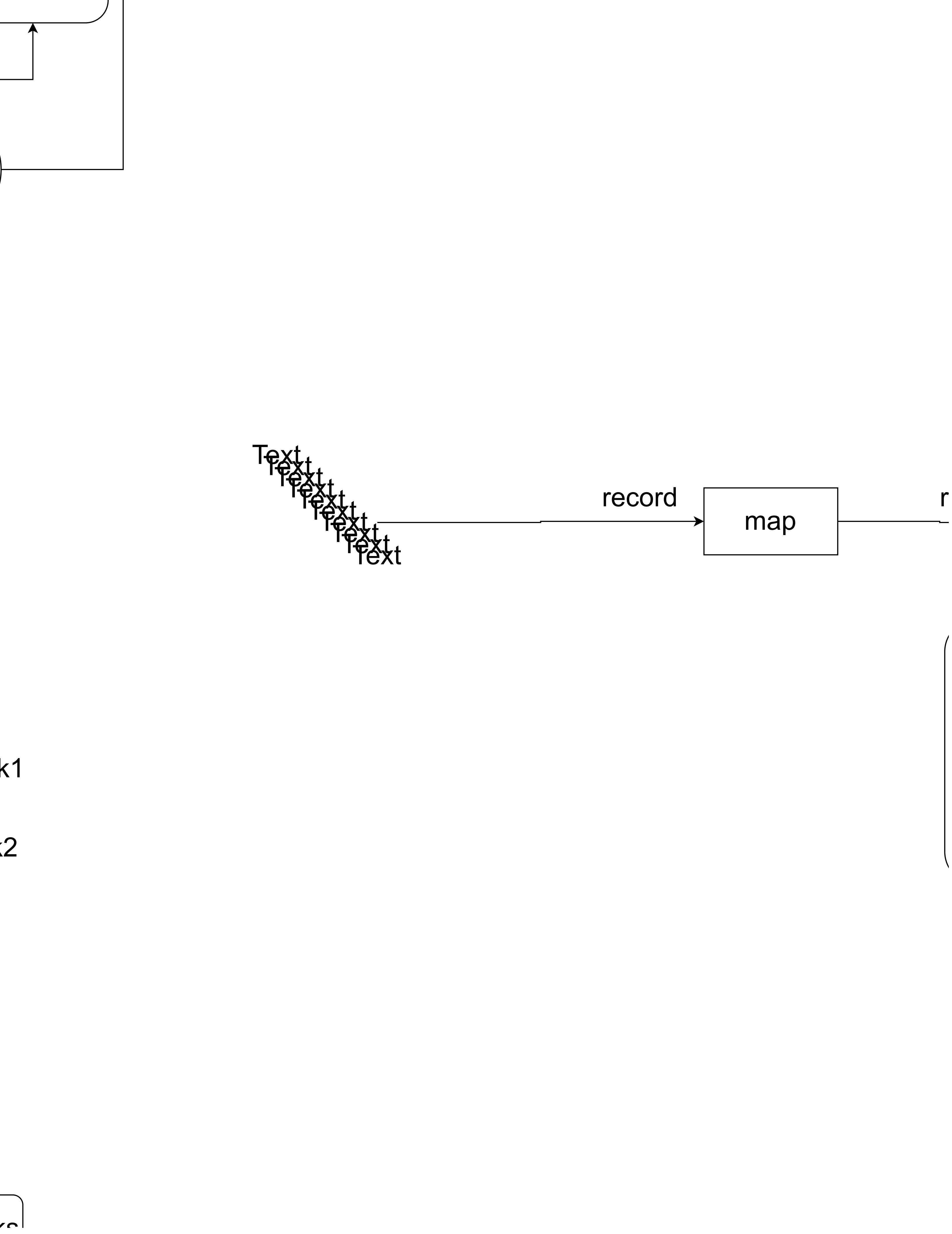
Thread 1 - Tasl

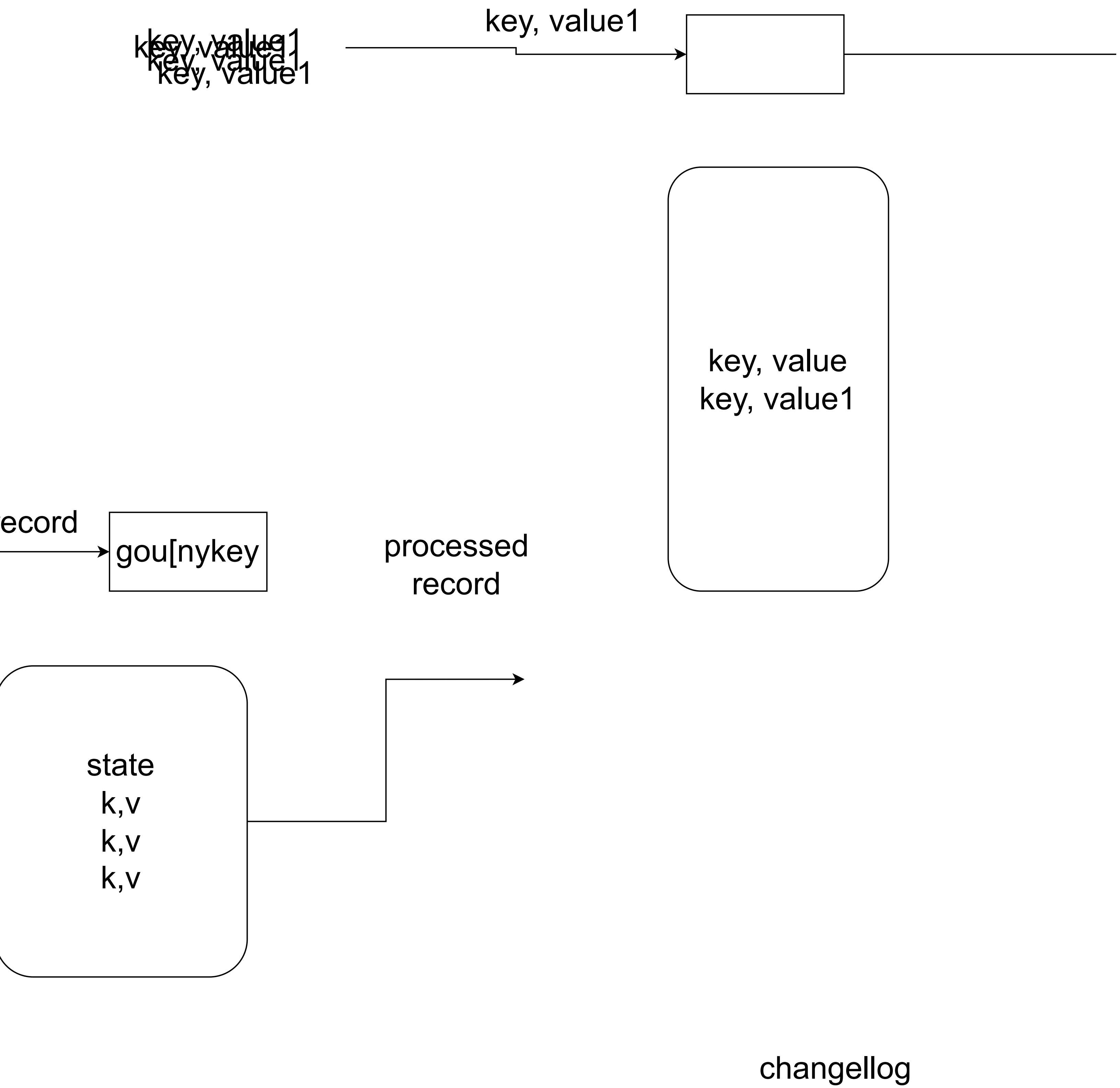
Thread 2 - Task

value1:value1.tupper

~~value2:~~value1.tupper




Text
Text
Text
Text
Text
Text
Text
Text
Text
record
map
r
k1
k2
ks


key, value1
key, value1
key, value1
key, value1
key, value1
key, value
key, value1
ecord
gou[nykey
processed
record
state
k,v
k,v
k,v
changellog

key, value

key, value

value1+value

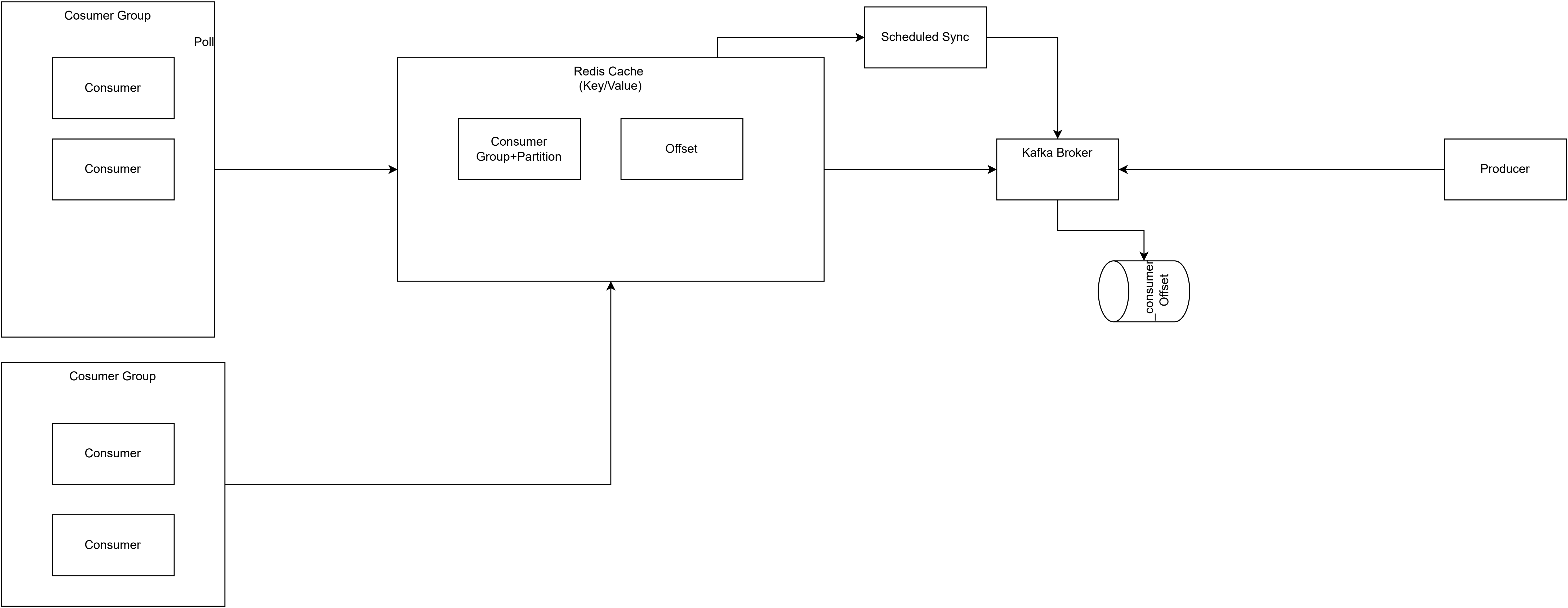
key, value1

Changelog

POD
POD
DB (Rock
DB)

KS

DIY Exercise - day 7



Cosumer Group

Poll

Consumer

Consumer

Cosumer Group

Consumer

Consumer

Redis Cache
(Key/Value)

Consumer
Group+Partition

Offset

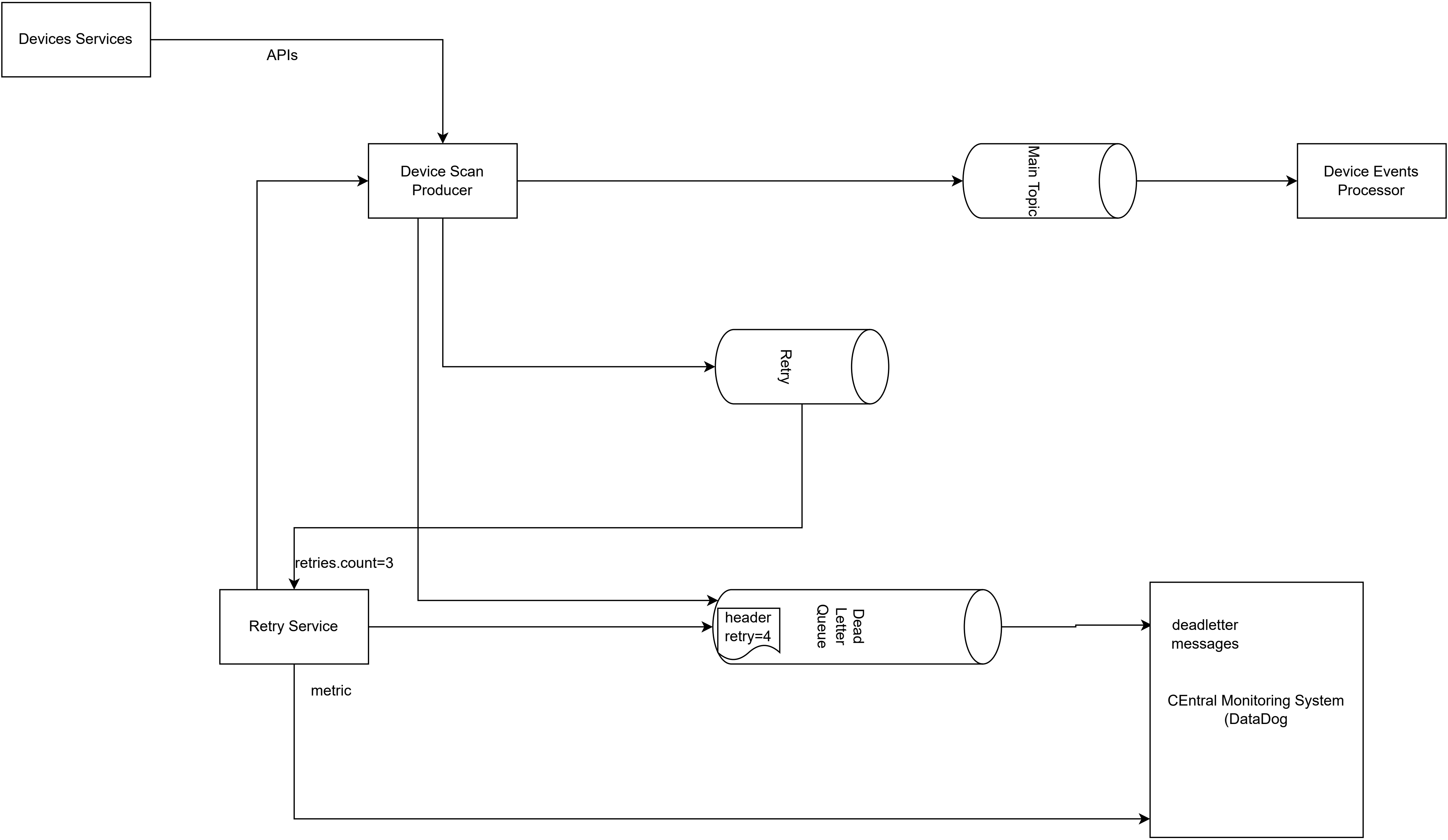
Scheduled Sync

Kafka Broker

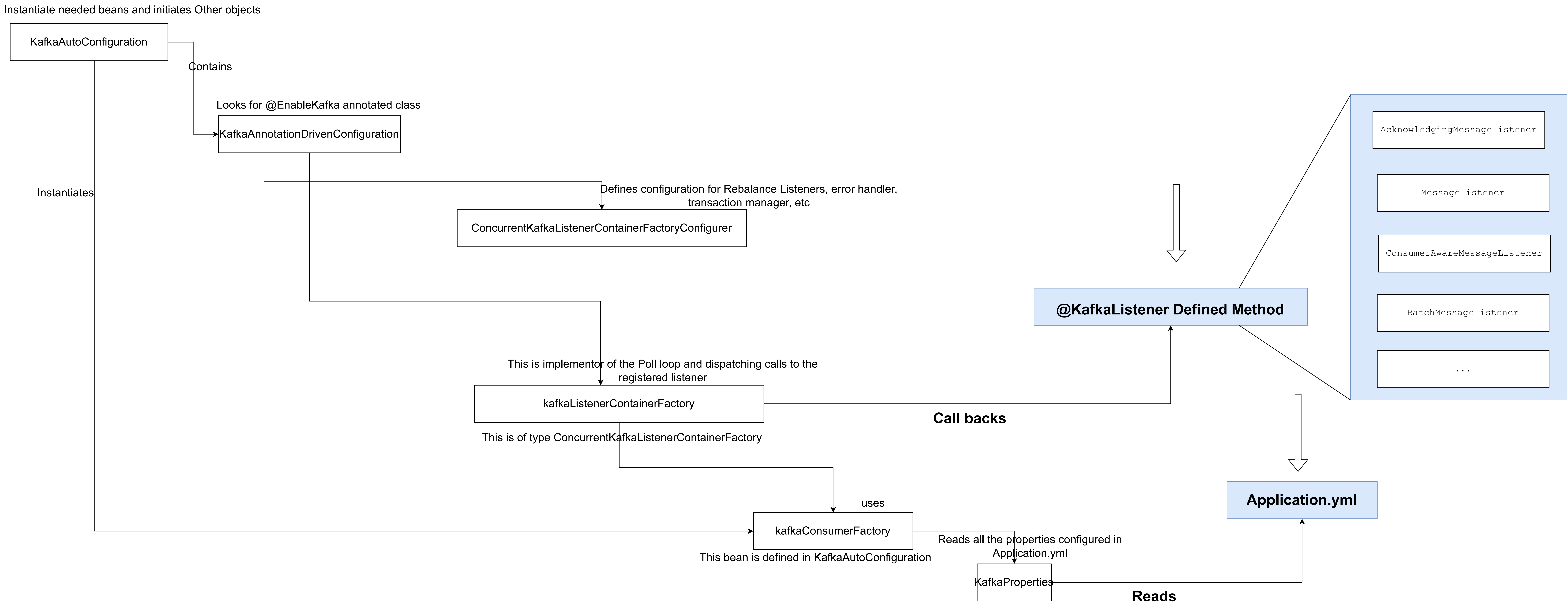
Producer

_consumer
Offset

Producer Retry Mechanism



Spring Kafka Consumer Internals



Instantiate needed beans and initiates Other objects

KafkaAutoConfiguration

Contains

Looks for @EnableKafka annotated class

KafkaAnnotationDrivenConfiguration

Instantiates

Defines configuration for Rebalance Listeners, error handler, transaction manager, etc

ConcurrentKafkaListenerContainerFactoryConfigurer

This is implementor of the Poll loop and dispatching calls to the registered listener

kafkaListenerContainerFactory

This is of type ConcurrentKafkaListenerContainerFactory

Call backs

@KafkaListener Defined Method

AcknowledgingMessageListener

MessageListener

ConsumerAwareMessageListener

BatchMessageListener

. . .

Application.yml

uses

kafkaConsumerFactory

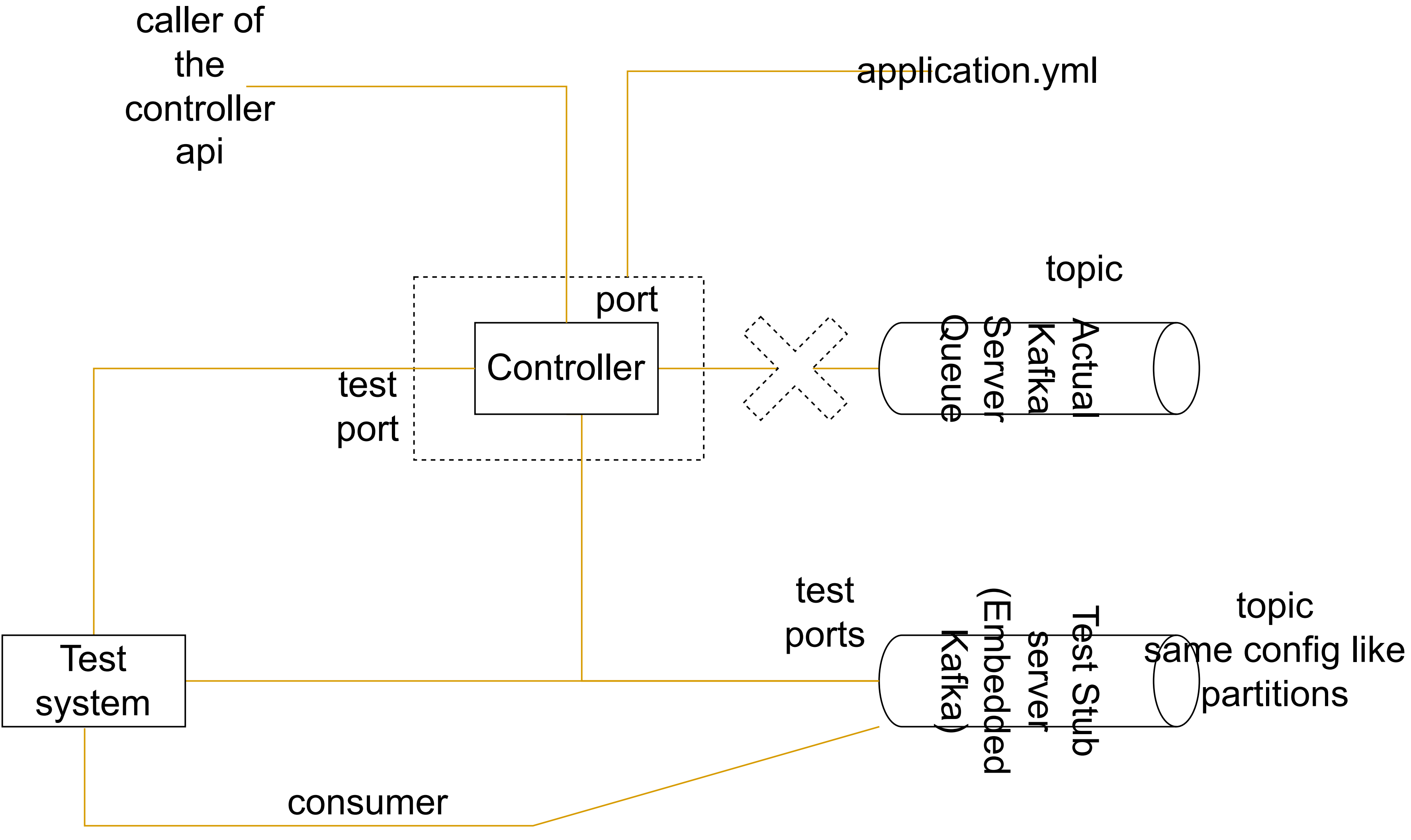
This bean is defined in KafkaAutoConfiguration

Reads all the properties configured in Application.yml

KafkaProperties

Reads

Producer Test



caller of the controller api

application.yml

port

Controller

test port

topic

Actual Kafka Server Queue

test ports

Test system

Test Stub server (Embedded Kafka)

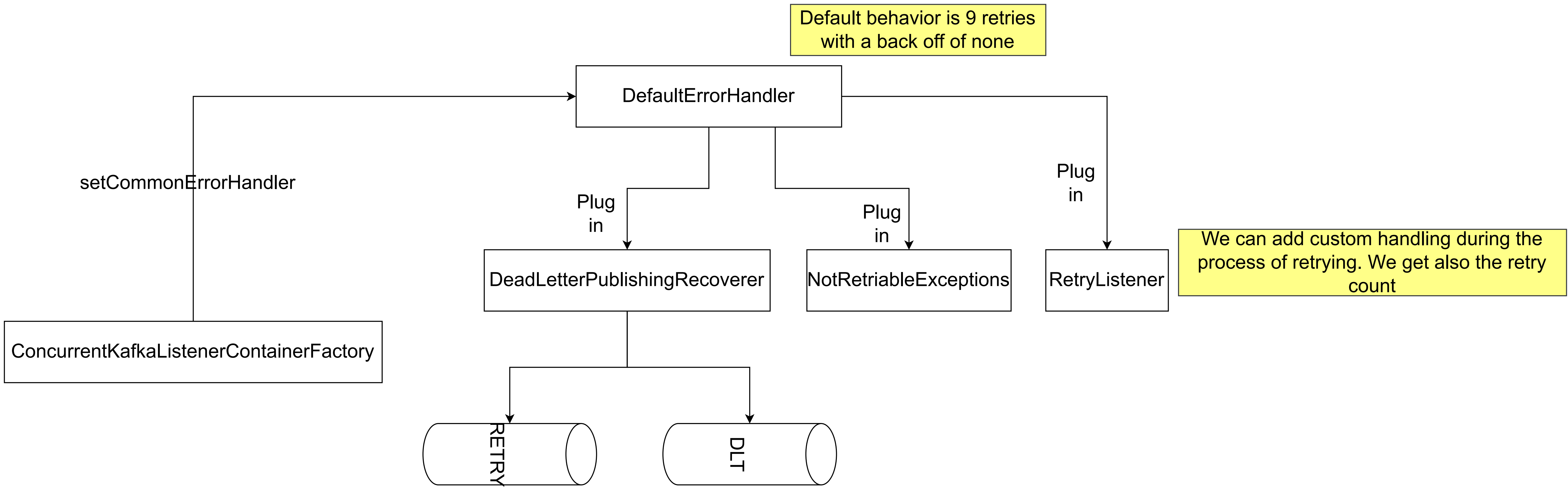
topic
same config like
partitions

consumer

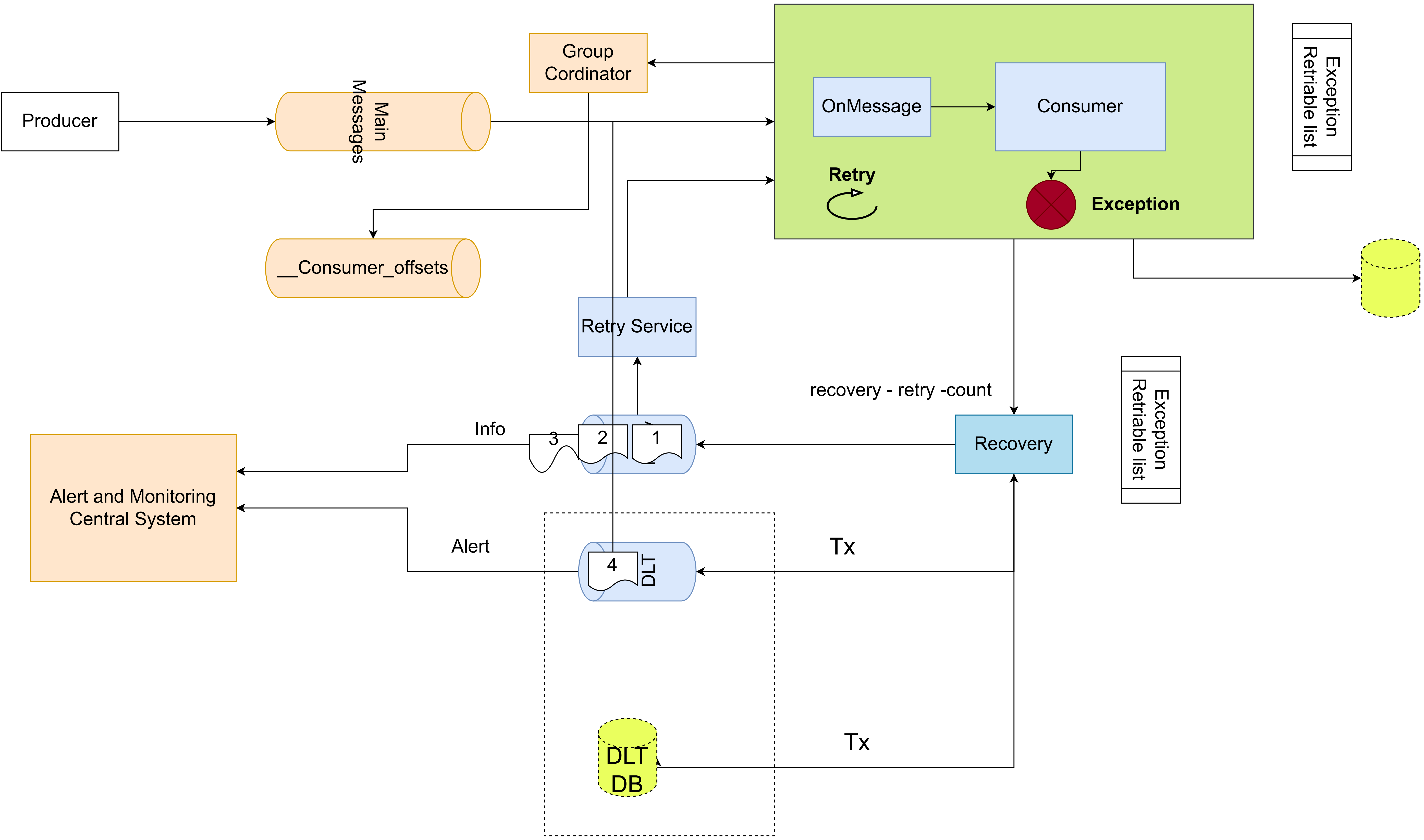
Integer ---- DeviceData

Consumer Error Handling & Recovery Mechanism

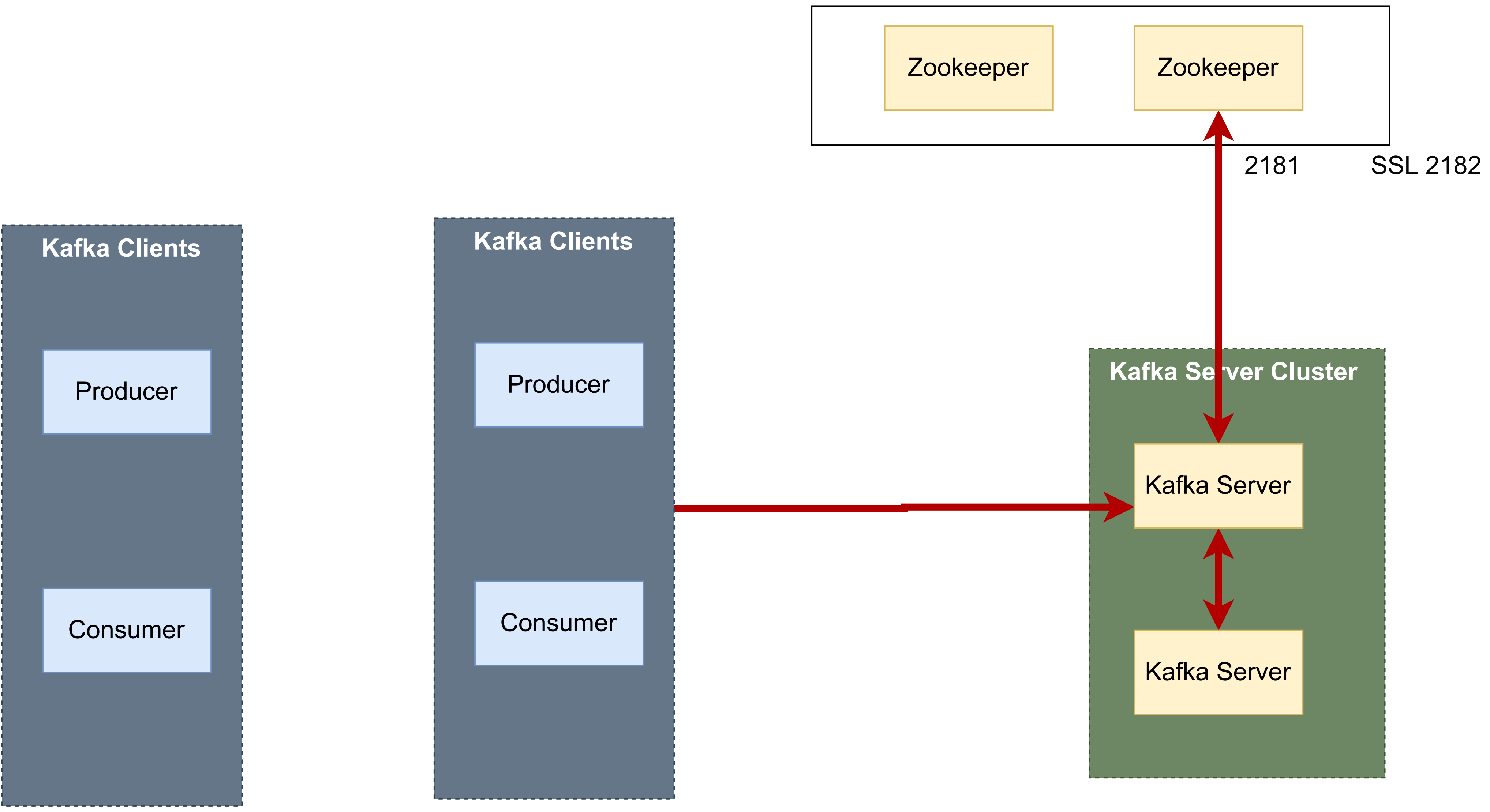
Spring Kafka Consumer Classes


Default behavior is 9 retries with a back off of none
DefaultErrorHandler
setCommonErrorHandler
Plug in
Plug in
Plug in
We can add custom handling during the process of retrying. We get also the retry count
DeadLetterPublishingRecoverer
NotRetriableExceptions
RetryListener
ConcurrentKafkaListenerContainerFactory
RETRY
DLT

Consumer Fail Recovery



Communications in Kafka Infra To Be Secured



Zookeeper

Zookeeper

2181

SSL 2182

Kafka Clients

Producer

Consumer

Kafka Clients

Producer

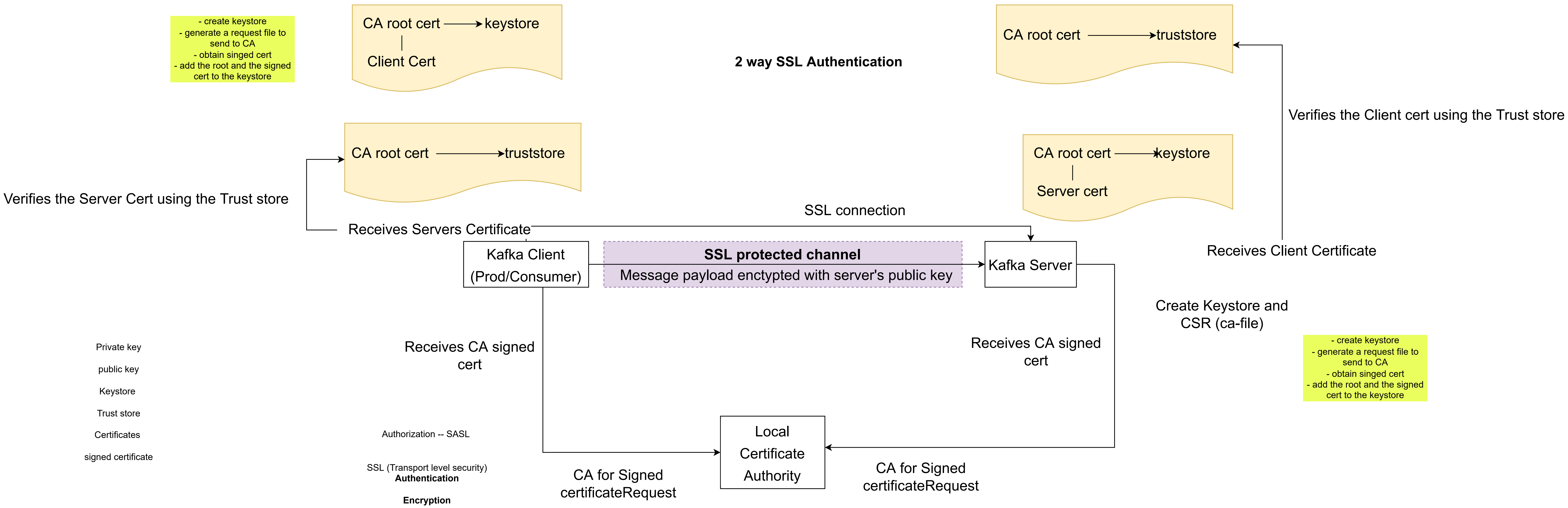
Consumer

Kafka Server Cluster

Kafka Server

Kafka Server

Securing Kafka Client and Server using SSL



- create keystore
- generate a request file to send to CA
- obtain singed cert
- add the root and the signed cert to the keystore

CA root cert → keystore

Client Cert

2 way SSL Authentication

CA root cert → truststore

Verifies the Client cert using the Trust store

CA root cert → truststore

CA root cert → keystore

Server cert

Verifies the Server Cert using the Trust store

Receives Servers Certificate

SSL connection

Kafka Client (Prod/Consumer)

SSL protected channel

Message payload enctypted with server's public key

Kafka Server

Receives Client Certificate

Create Keystore and CSR (ca-file)

- create keystore
- generate a request file to send to CA
- obtain singed cert
- add the root and the signed cert to the keystore

Receives CA signed cert

Receives CA signed cert

Private key

public key

Keystore

Trust store

Certificates

signed certificate

Authorization -- SASL

SSL (Transport level security)
Authentication

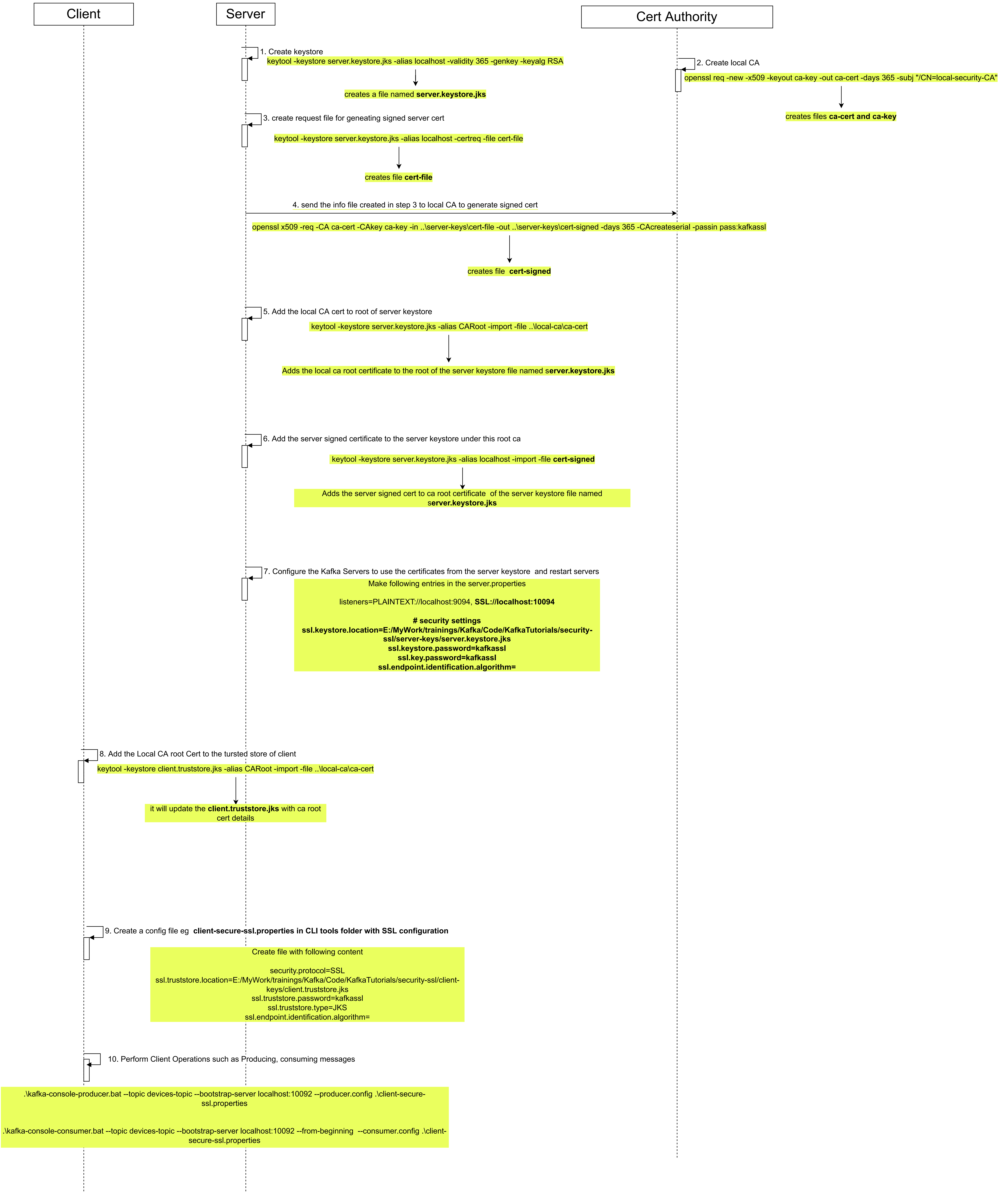
Encryption

Local Certificate Authority

CA for Signed certificateRequest

CA for Signed certificateRequest

SSL Setup for securing Kafka Server and Clients - 1 Way



1. Create keystore

```
keytool -keystore server.keystore.jks -alias localhost -validity 365 -genkey -keyalg RSA
```

creates a file named **server.keystore.jks**

2. Create local CA

```
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365 -subj "/CN=local-security-CA"
```

creates files **ca-cert and ca-key**

3. create request file for geneating signed server cert

```
keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-file
```

creates file **cert-file**

4. send the info file created in step 3 to local CA to generate signed cert

```
openssl x509 -req -CA ca-cert -CAkey ca-key -in ..\server-keys\cert-file -out ..\server-keys\cert-signed -days 365 -CAcreateserial -passin pass:kafkassl
```

creates file **cert-signed**

5. Add the local CA cert to root of server keystore

```
keytool -keystore server.keystore.jks -alias CARoot -import -file ..\local-ca\ca-cert
```

Adds the local ca root certificate to the root of the server keystore file named s**erver.keystore.jks**

6. Add the server signed certificate to the server keystore under this root ca

```
keytool -keystore server.keystore.jks -alias localhost -import -file cert-signed
```

Adds the server signed cert to ca root certificate of the server keystore file named **server.keystore.jks**

7. Configure the Kafka Servers to use the certificates from the server keystore and restart servers

Make following entries in the server.properties

```
listeners=PLAINTEXT://localhost:9094, SSL://localhost:10094

# security settings
ssl.keystore.location=E:/MyWork/trainings/Kafka/Code/KafkaTutorials/security-ssl/server-keys/server.keystore.jks
ssl.keystore.password=kafkassl
ssl.key.password=kafkassl
ssl.endpoint.identification.algorithm=
```

8. Add the Local CA root Cert to the tursted store of client

```
keytool -keystore client.truststore.jks -alias CARoot -import -file ..\local-ca\ca-cert
```

it will update the **client.truststore.jks** with ca root cert details

9. Create a config file eg **client-secure-ssl.properties in CLI tools folder with SSL configuration**

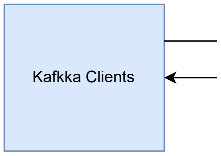
Create file with following content

```
security.protocol=SSL
ssl.truststore.location=E:/MyWork/trainings/Kafka/Code/KafkaTutorials/security-ssl/client-keys/client.truststore.jks
ssl.truststore.password=kafkassl
ssl.truststore.type=JKS
ssl.endpoint.identification.algorithm=
```

10. Perform Client Operations such as Producing, consuming messages

```
.\kafka-console-producer.bat --topic devices-topic --bootstrap-server localhost:10092 --producer.config .\client-secure-ssl.properties

.\kafka-console-consumer.bat --topic devices-topic --bootstrap-server localhost:10092 --from-beginning  --consumer.config .\client-secure-ssl.properties
```

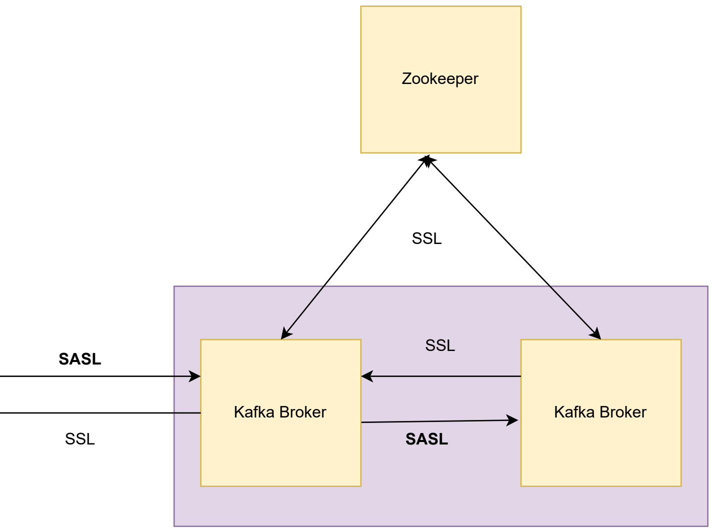
SASL T



User creation

ACL creation

SASL Topology



Important commands

Admin

```
./bin/kafka-configs.sh --zookeeper localhost:2182 --zk-tls-config-file ./bin/zookeeper-client.properties --alter --add-conf
```

Producer

```
./bin/kafka-configs.sh --zookeeper localhost:2182 --zk-tls-config-file ./bin/zookeeper-client.properties --alter --add-conf
```

Consumer

```
./bin/kafka-configs.sh --zookeeper localhost:2182 --zk-tls-config-file ./bin/zookeeper-client.properties --alter --add-con
```

Producer

```
./bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2182 --zk-tls-config-file ./bin/zoo
```

Consumer

```
./bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2182 --zk-tls-config-file ./bin/zooke
```

Consumer
to group

```
./bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2182 --zk-tls-config-file ./bin/zook
```

```
fig 'SCRAM-SHA-512=[password=kafkassl]' --entity-type users --entity-name broker-admin

fig 'SCRAM-SHA-512=[password=kafkassl]' --entity-type users --entity-name kafkaprod

fig 'SCRAM-SHA-512=[password=kafkassl]' --entity-type users --entity-name kafkacon
```

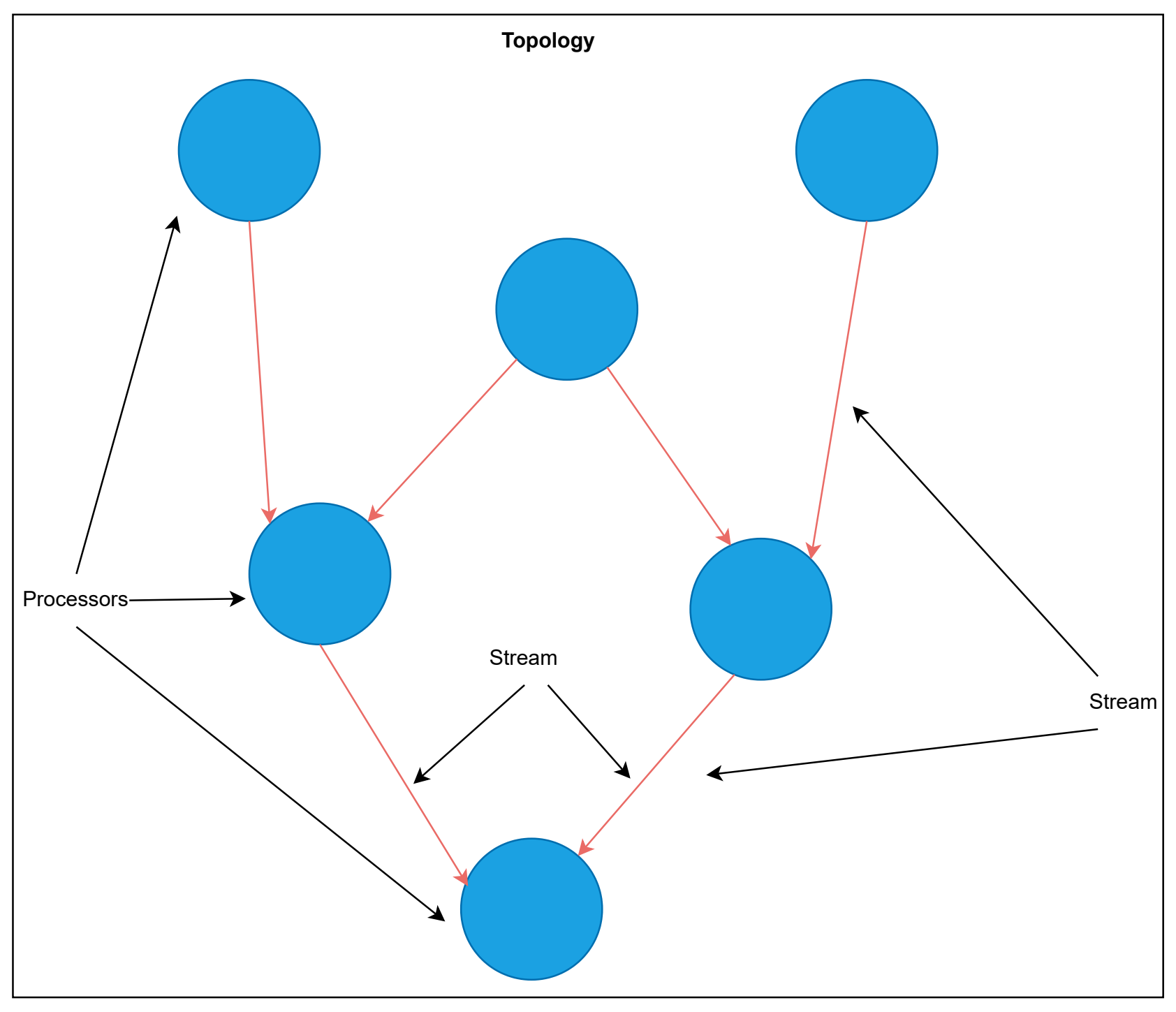
```
okeeper-client.properties --add --allow-principal User:kafkaprod --topic secure-topic --operation WRITE --operation DESCRIBE --

eeper-client.properties --add --allow-principal User:kafkacon --topic secure-topic --operation READ --operation DESCRIBE --opera
```

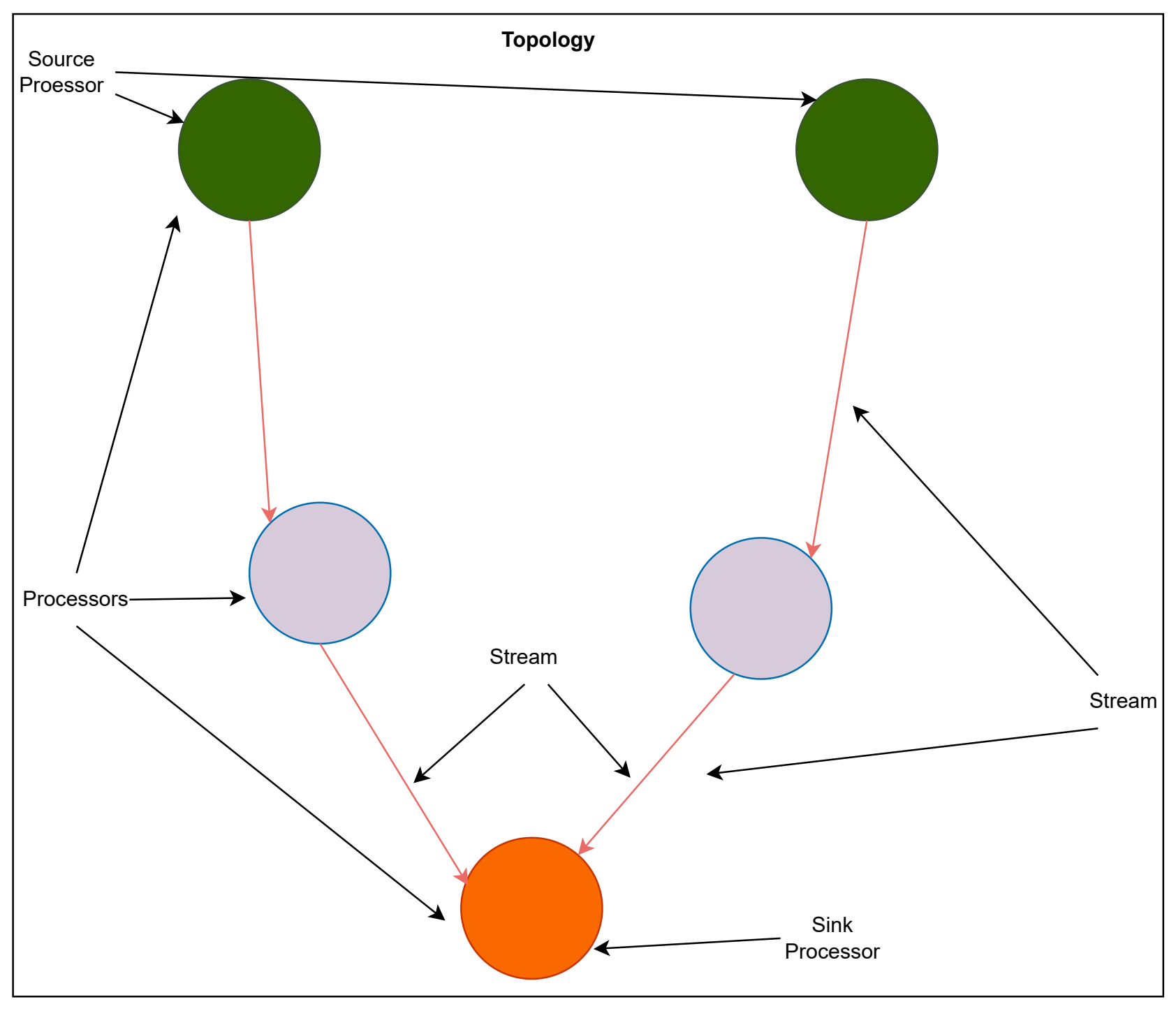
```
keeper-client.properties --add --allow-principal User:kafkacon --group secure-group --operation READ
```

operation DESCRIBECONFIGS

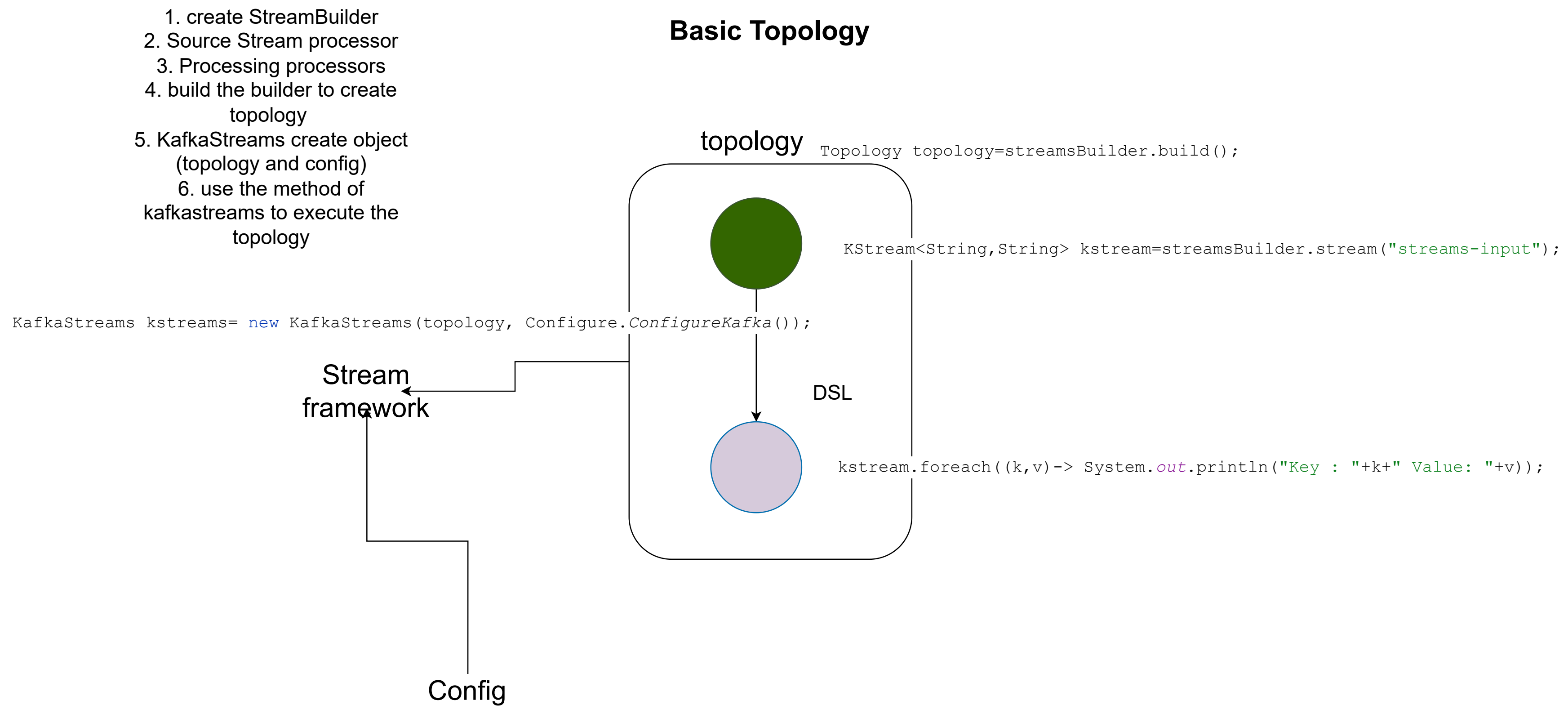
ation DESCRIBECONFIGS

Kafka Streams





Basic Topology



1. create StreamBuilder
2. Source Stream processor
3. Processing processors
4. build the builder to create topology
5. KafkaStreams create object (topology and config)
6. use the method of kafkastreams to execute the topology

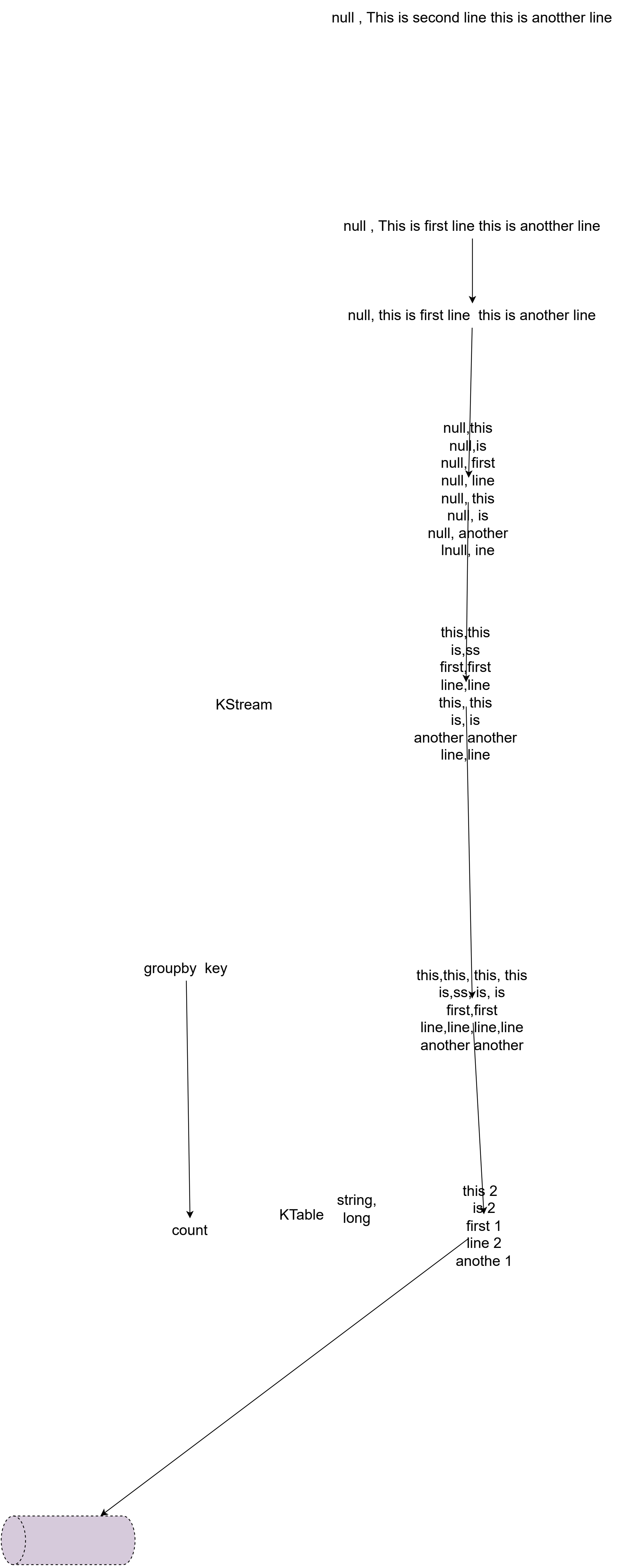
```
Topology topology=streamsBuilder.build();
```

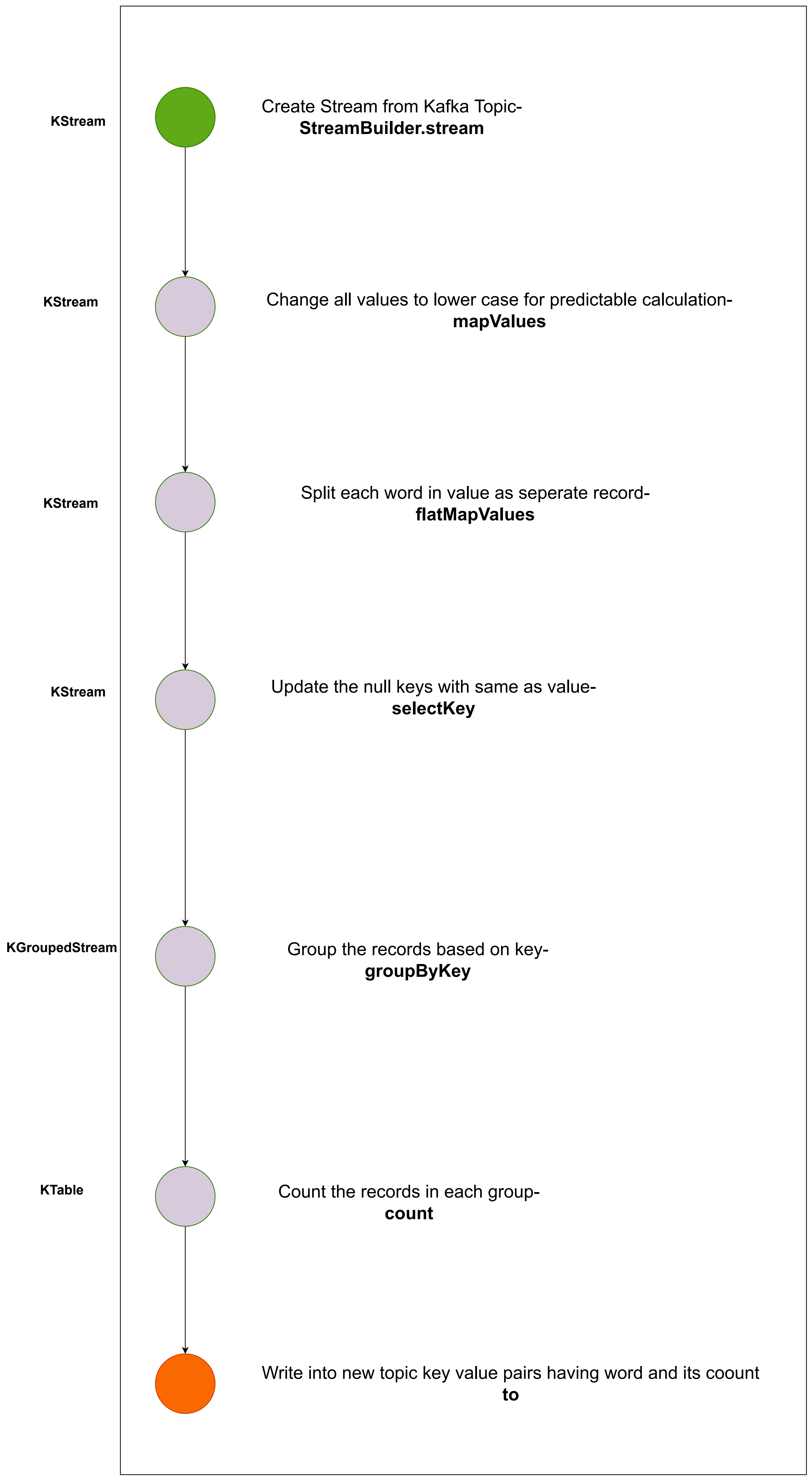
```
KStream<String,String> kstream=streamsBuilder.stream("streams-input");
```

```
KafkaStreams kstreams= new KafkaStreams(topology, Configure.ConfigureKafka());
```

```
kstream.foreach((k,v)-> System.out.println("Key : "+k+" Value: "+v));
```

WordCount Topology





KStream — Create Stream from Kafka Topic- **StreamBuilder.stream**

KStream — Change all values to lower case for predictable calculation- **mapValues**

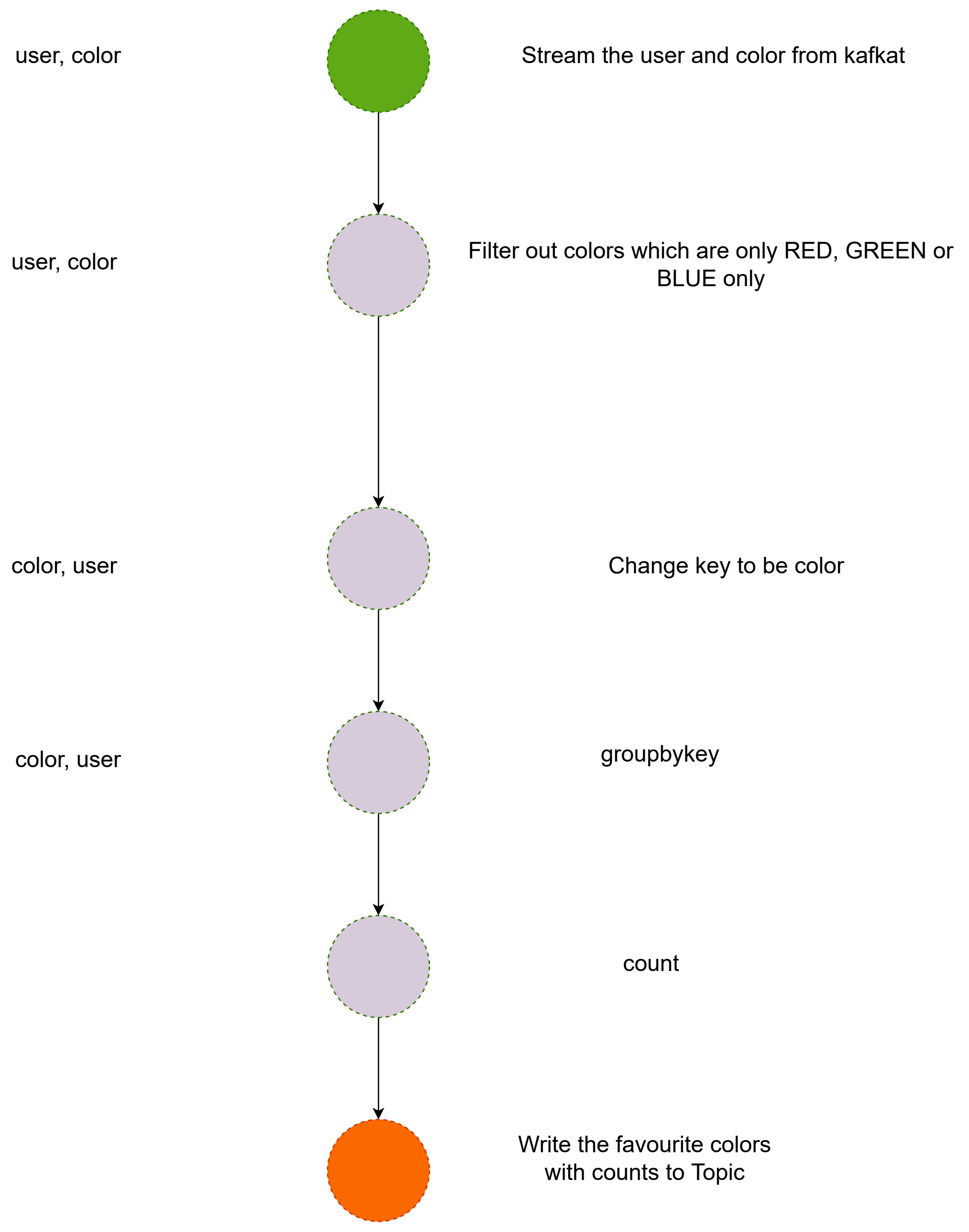
KStream — Split each word in value as seperate record- **flatMapValues**

KStream — Update the null keys with same as value- **selectKey**

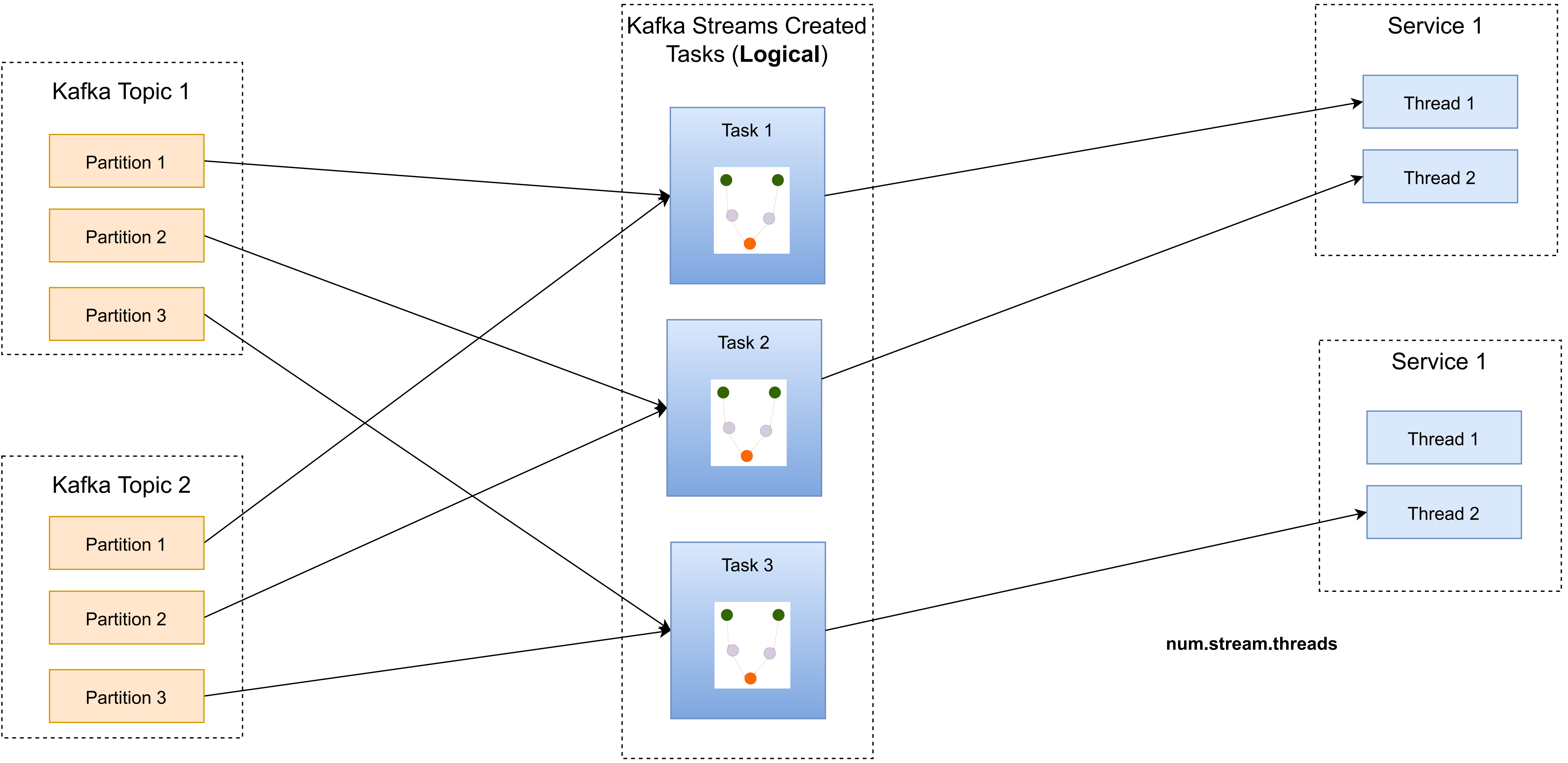
KGroupedStream — Group the records based on key- **groupByKey**

KTable — Count the records in each group- **count**

Write into new topic key value pairs having word and its coount **to**


user, color
Stream the user and color from kafkat
user, color
Filter out colors which are only RED, GREEN or BLUE only
color, user
Change key to be color
color, user
groupbykey
count
Write the favourite colors with counts to Topic

Kafka Streaming Architecture



Kafka Topic 1

Partition 1

Partition 2

Partition 3

Kafka Topic 2

Partition 1

Partition 2

Partition 3

Kafka Streams Created Tasks (**Logical**)

Task 1

Task 2

Task 3

Service 1

Thread 1

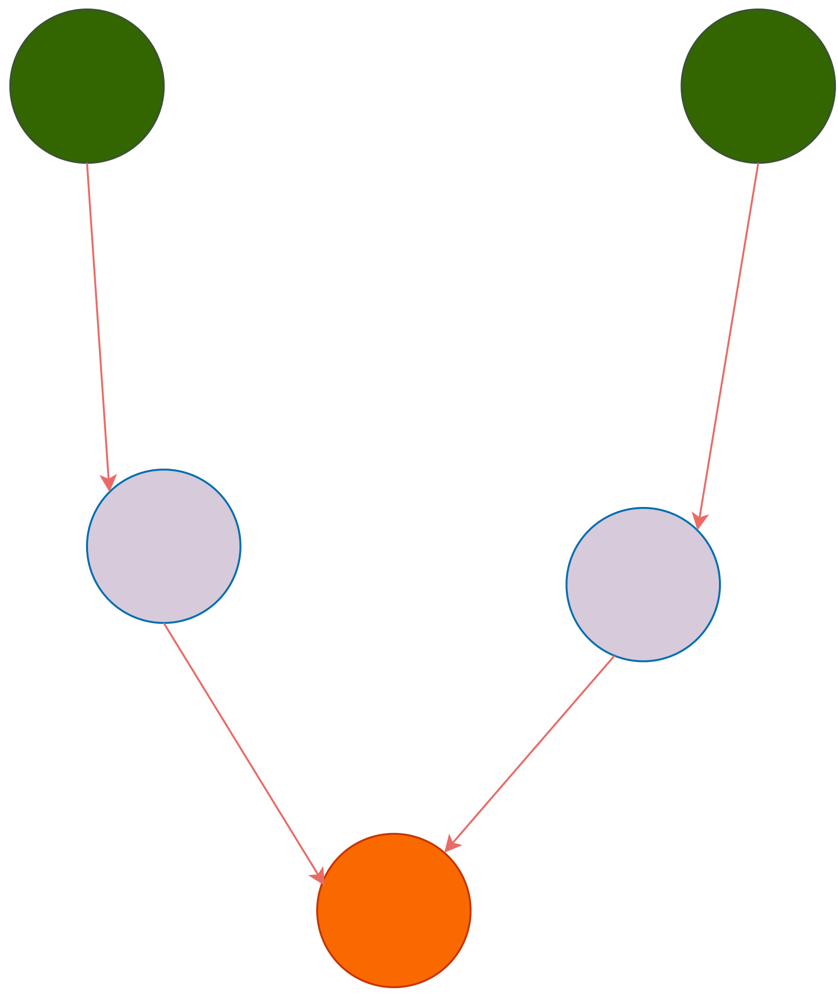
Thread 2

Service 1

Thread 1

Thread 2

num.stream.threads



Input	**KStream**	**KTable**
Ashok:45000 Aravind:90000	Ashok:45000 Aravind:90000	Ashok:45000 Aravind:90000
Centhil:100000	Ashok:45000 Aravind:90000 **Centhil:100000**	Ashok:45000 Aravind:90000 **Centhil:100000**
Ashok:75000	Ashok:45000 Aravind:90000 Centhil:100000 **Ashok:75000**	**Ashok:75000** Aravind:90000 Centhil:100000

username, address

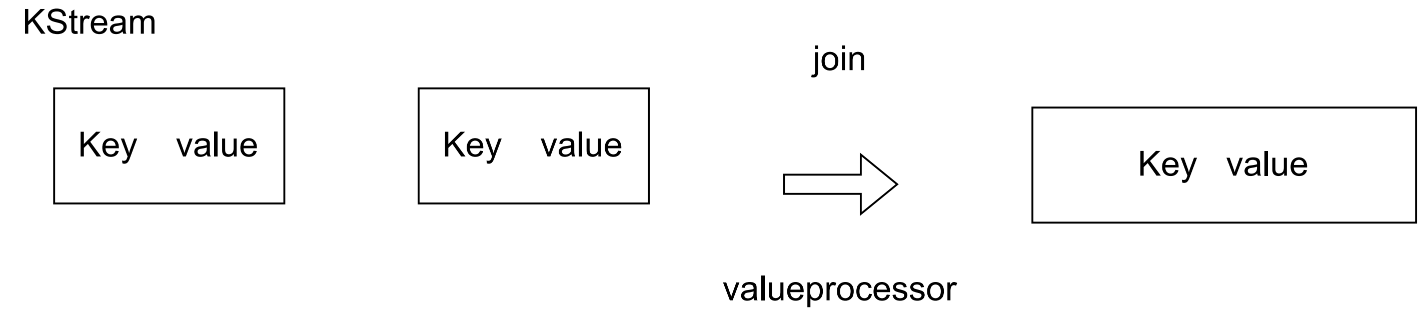
username, transaction description

username noof transactions

uname   "{address=<>,transaction desc=<>}

provide list of transactions done by each other

Produce enritched records having usernamme with puchase info added with address



Left topic	Right topic	Inner Join	Left Join
key leftva1	key rightval1	key left1+joined+rightval1	key left1+joined+rightval1
leftkeydiff leftva1	rightkey rightval1		leftkeydiff leftva1