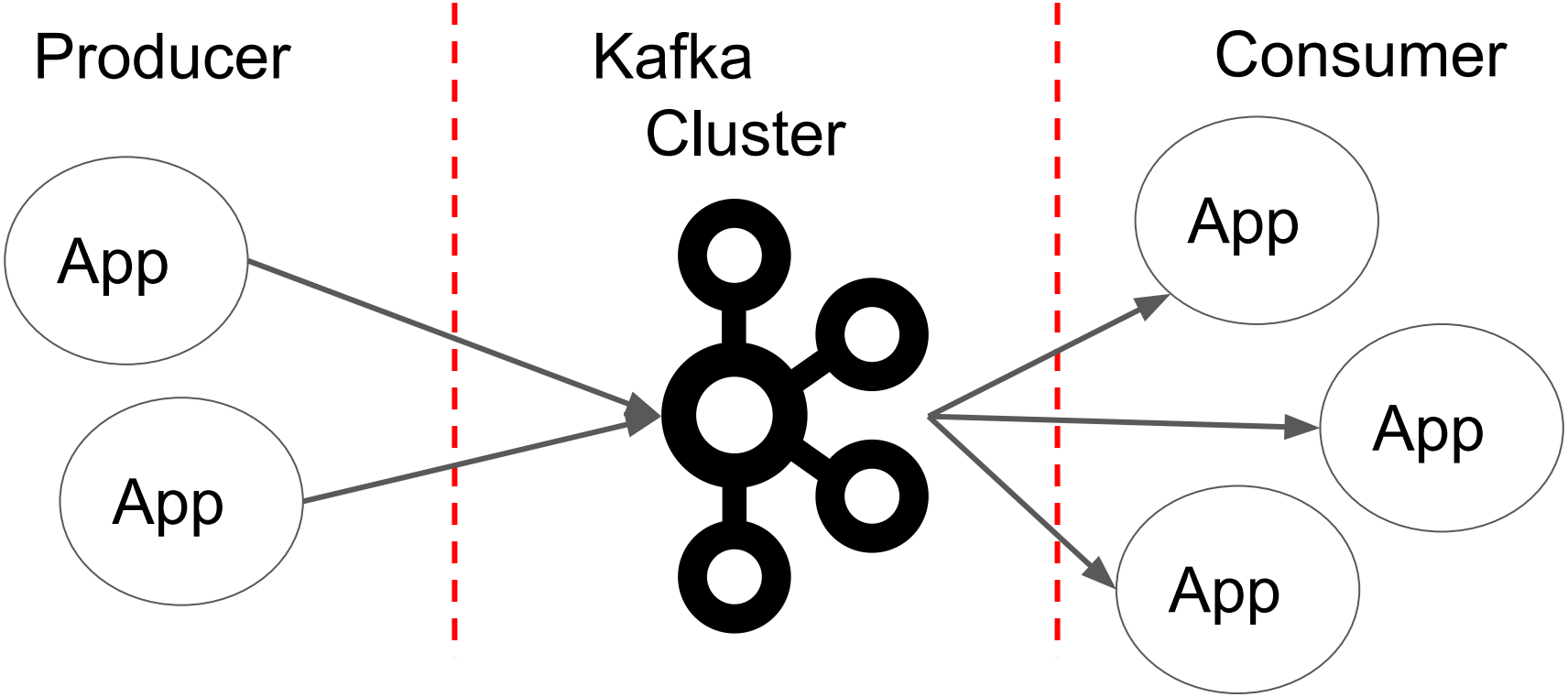
Kafka,
Producer와 Consumer

강 한구

Index

- Producer
- Broker
- Consumer
- 부록

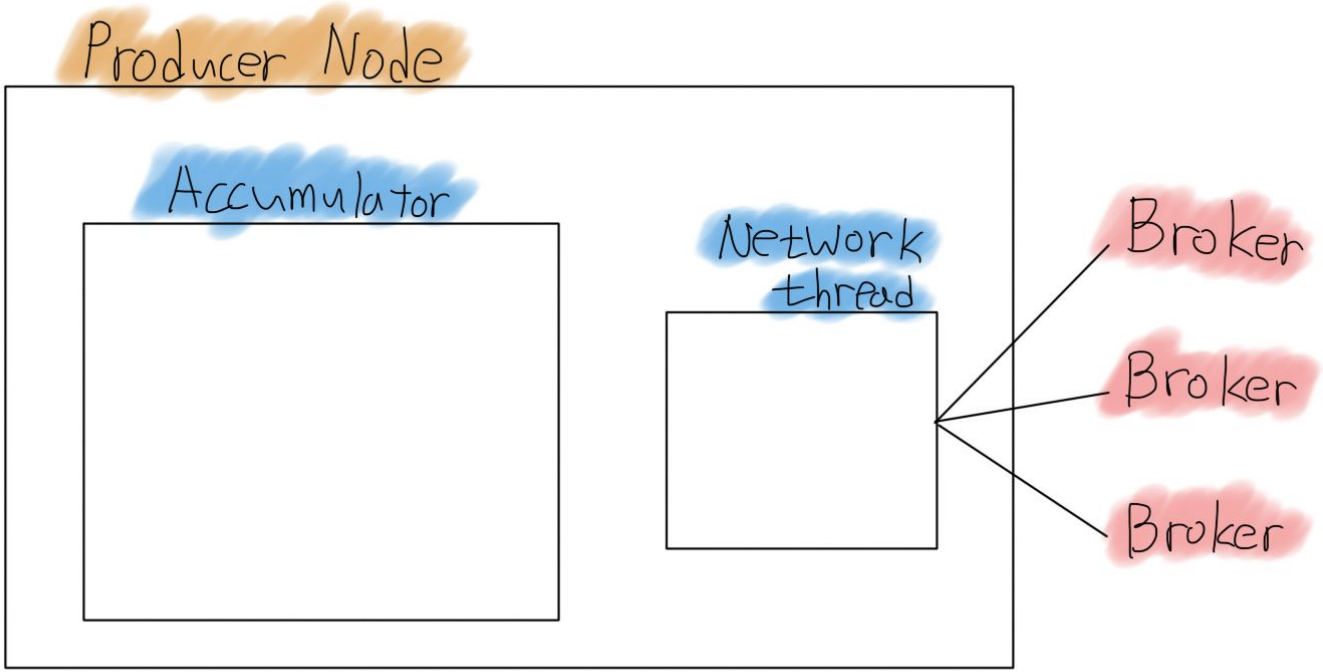

Producer
Kafka
Cluster
Consumer
App
App
App
App
App

Producer

Code

```
Producer<String, String> producer = new KafkaProducer<>(props);
for (int i = 0; i < 100; i++) {
    String data = Integer.toString(i);
    producer.send(new ProducerRecord("my-topic", data, data));
}
```

KafkaProducer 생성



KafkaProducer 생성

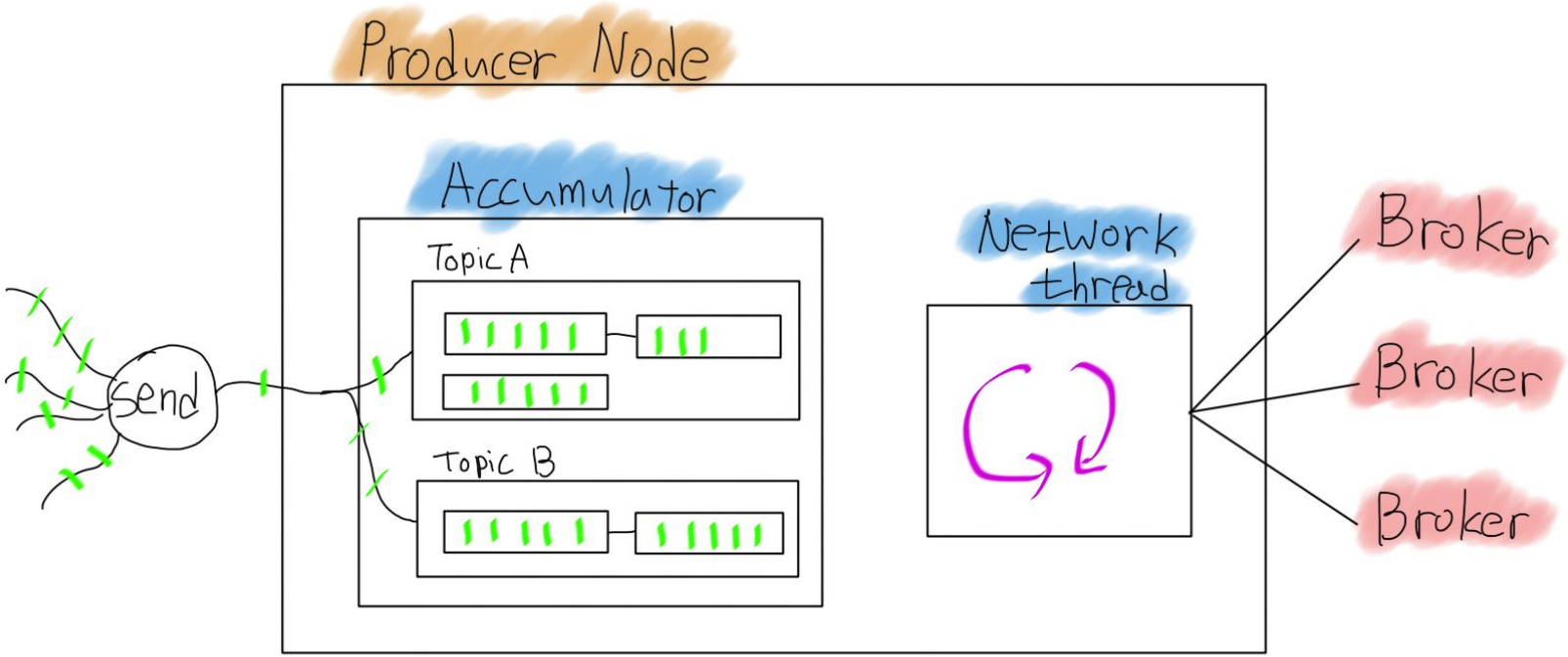
- Accumulator
  send한 Record를 메모리(RecordBatch)에 차곡차곡 쌓아주는 역할

- Network Thread
  Accumulator에 쌓인 RecordBatch를 Broker로 전송하는 역할

Code

```
Producer<String, String> producer = new KafkaProducer<>(props);
for (int i = 0; i < 100; i++) {
    String data = Integer.toString(i);
    producer.send(new ProducerRecord("my-topic", data, data));
}
```

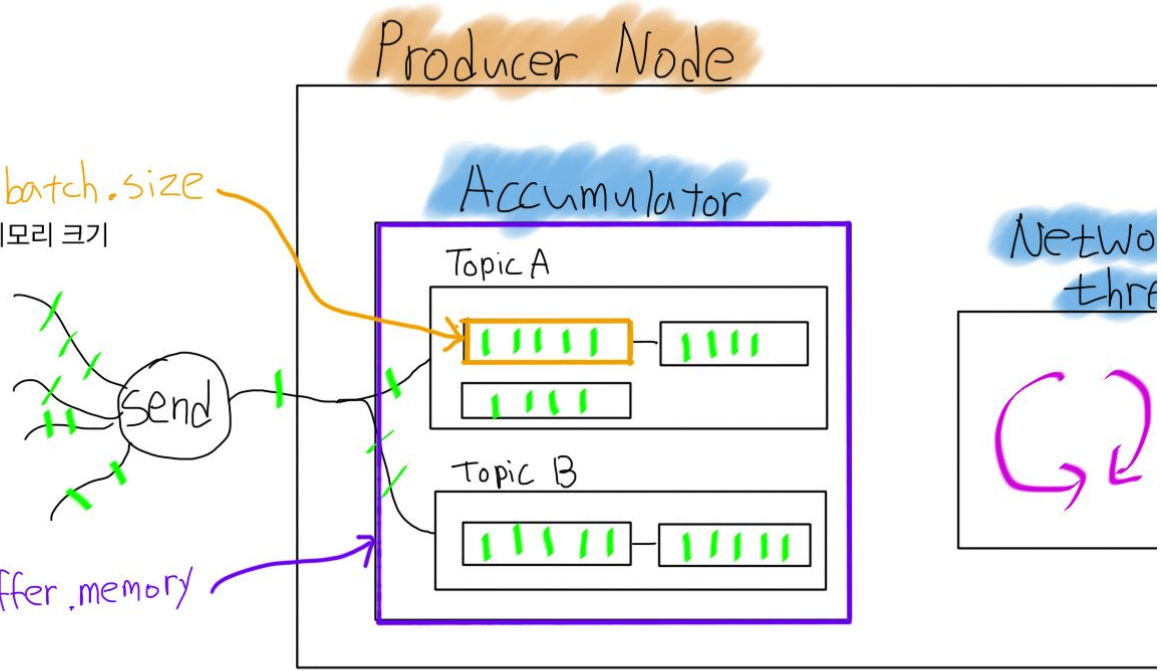
Accumulator



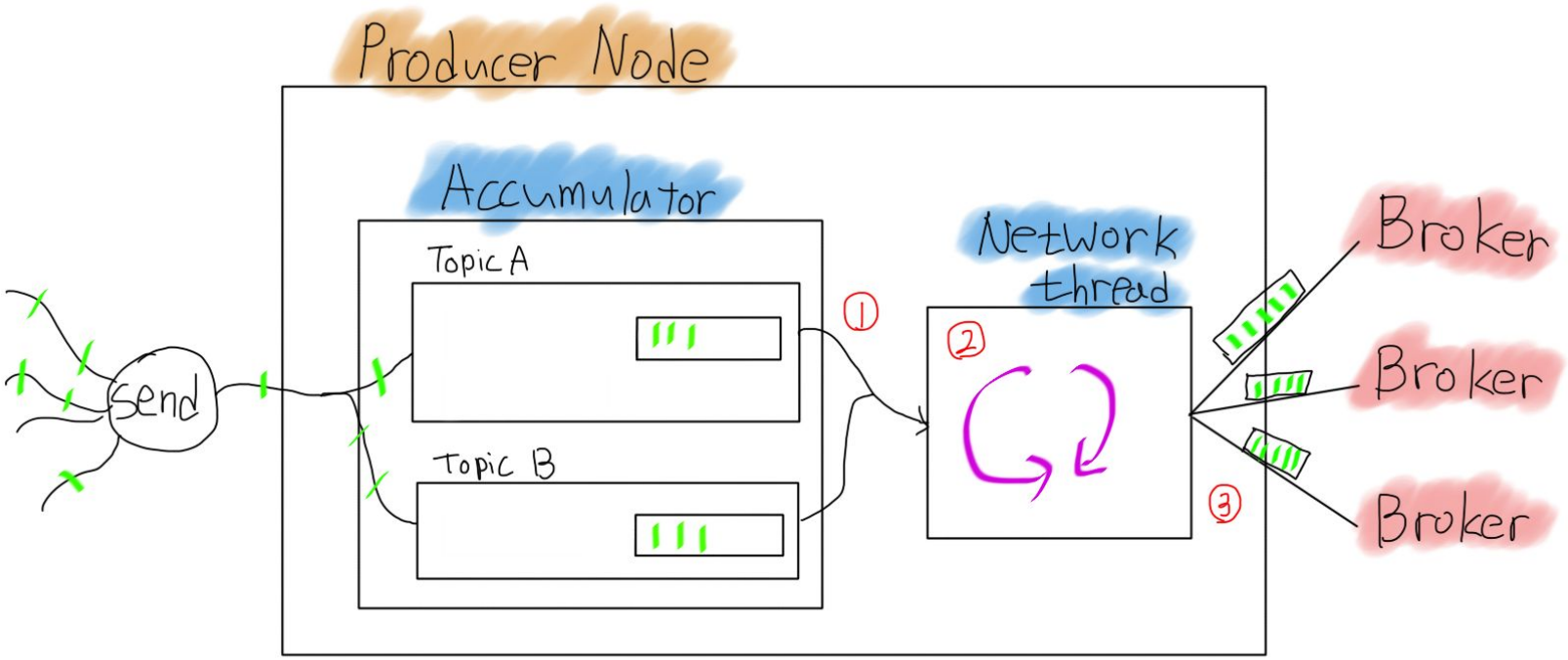
Accumulator

topic + partition 별로
record를 모아 둔 RecordBatch의 메모리 크기

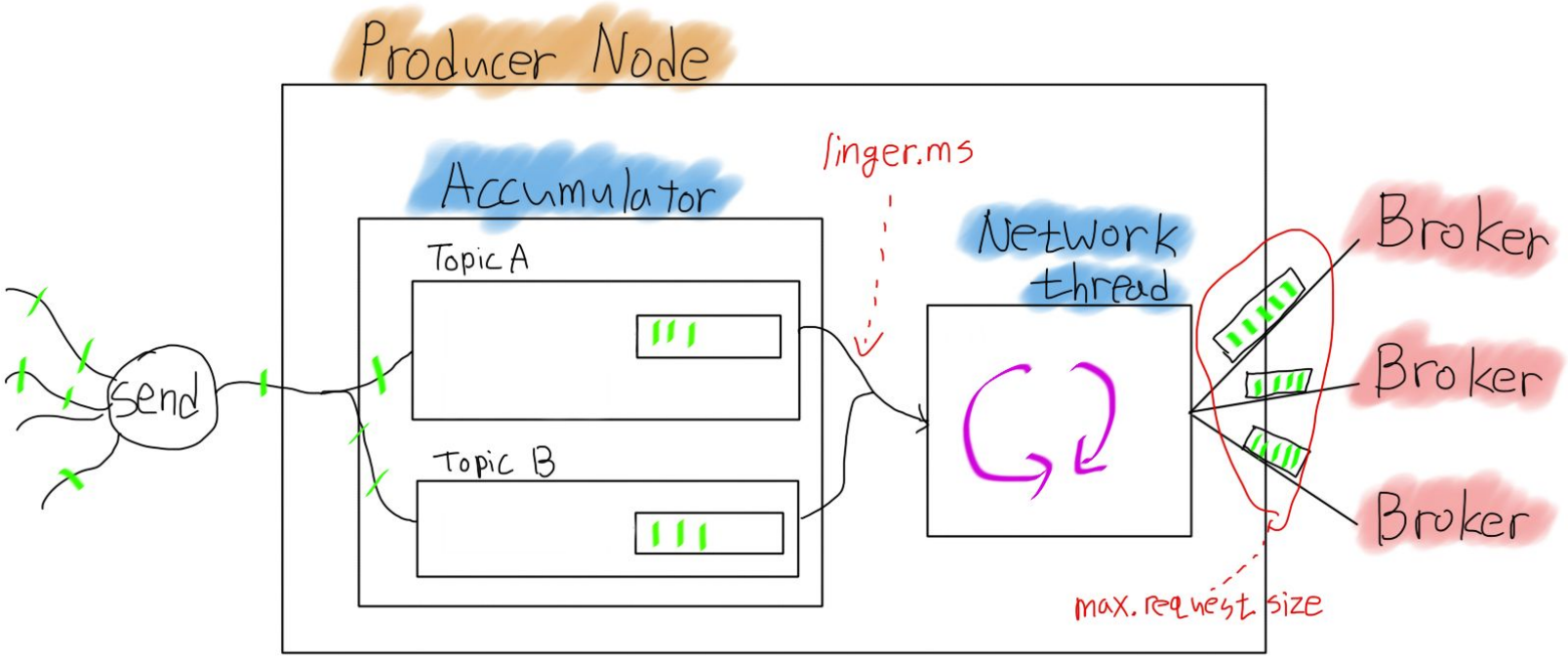
Accumulator에서 record를 쌓는데
사용 가능한 메모리 크기



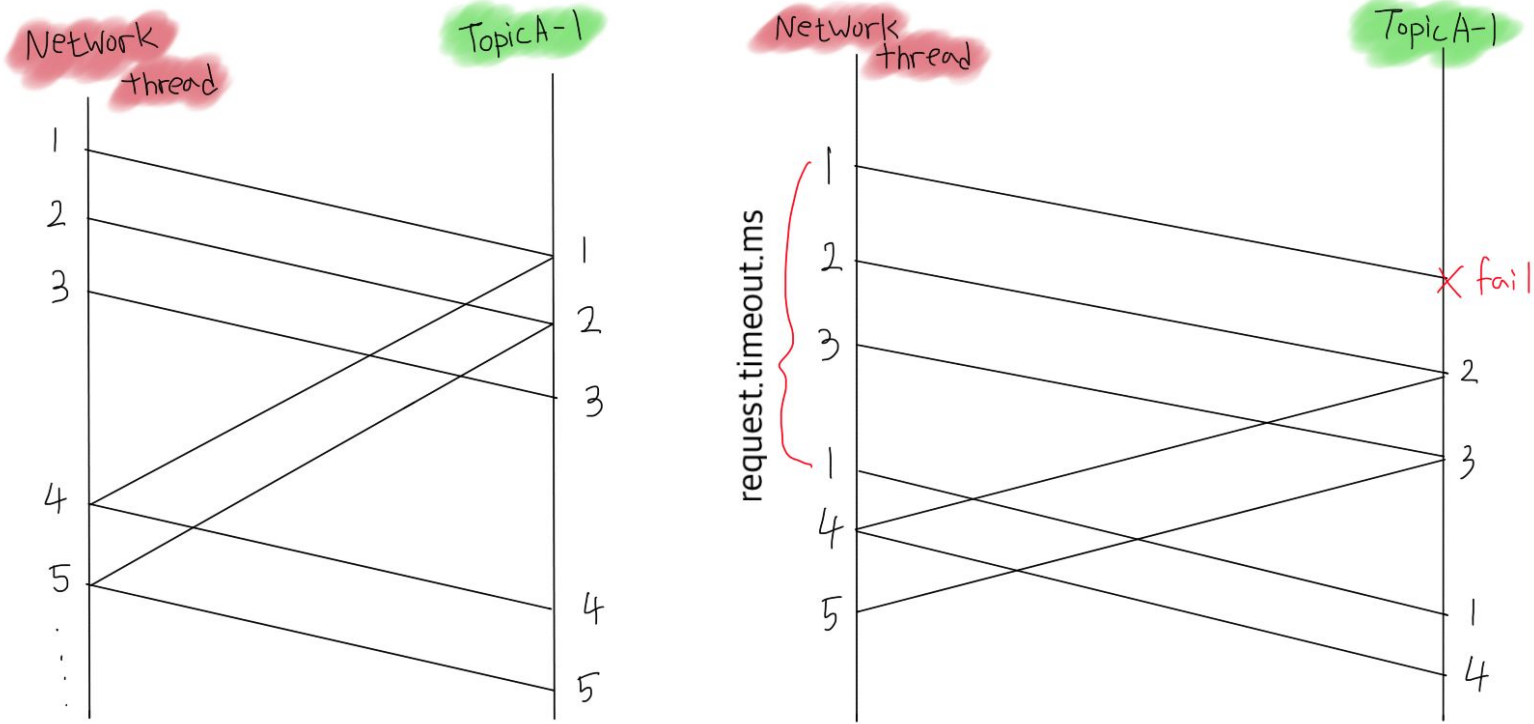
Network Thread



Network Thread

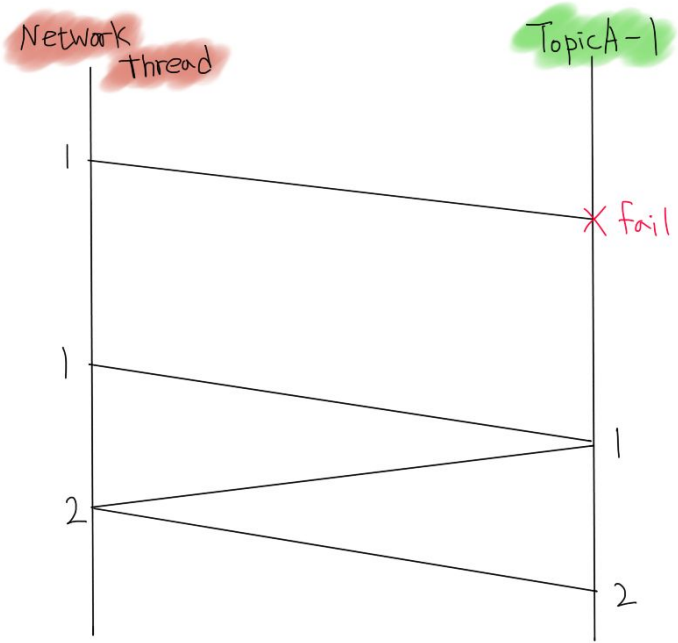


Network Thread



max.in.flight.requests.per.connection = 3

Network Thread



max.in.flight.requests.per.connection = 1

Broker

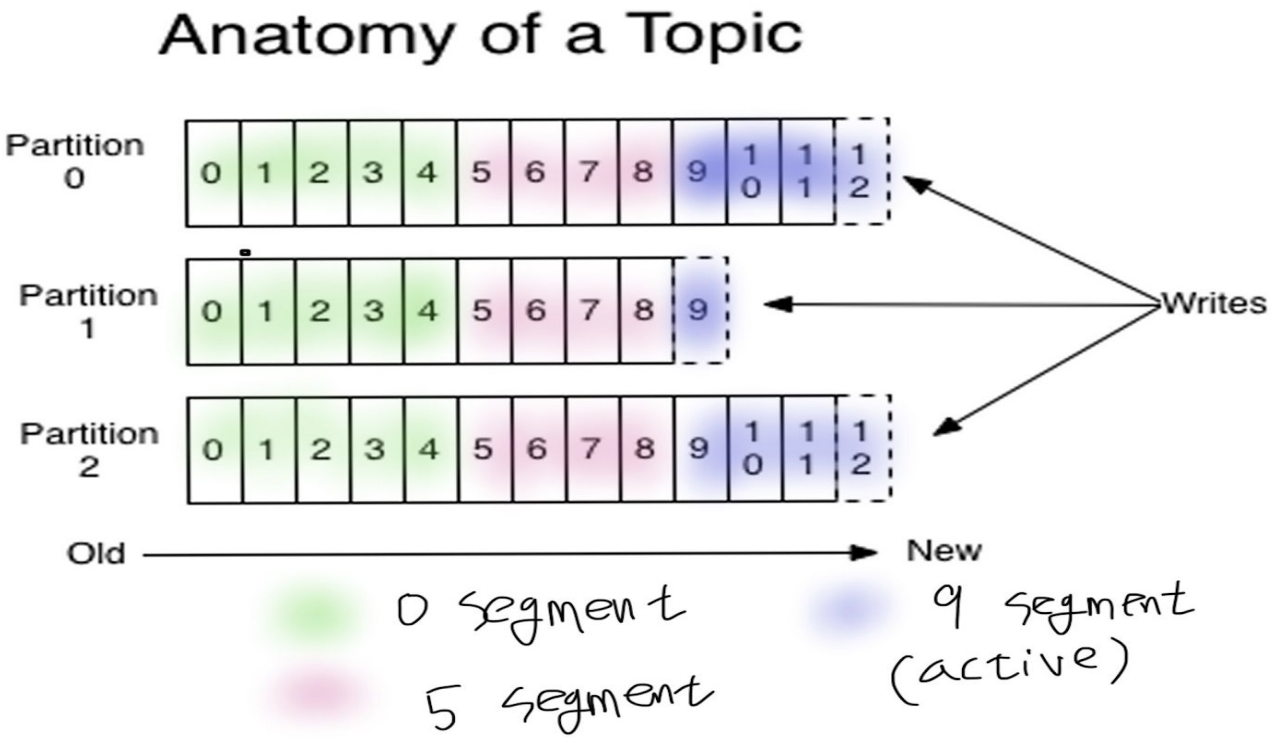
어떻게 저장되나

Topic

Partition

Segment

Offset



어떻게 저장되나

[Topic name]-[partition] 폴더 구조

```
[[redacted]1 /]$ tree /data* -d
/data1
└── kafka-logs
    ├── __consumer_offsets-10
    ├── __consumer_offsets-15
    ├── __consumer_offsets-18
    ├── __consumer_offsets-23
    ├── __consumer_offsets-30
    ├── [redacted]
    └── [redacted]
/data2
└── kafka-logs
    ├── __consumer_offsets-1
    ├── __consumer_offsets-26
    ├── __consumer_offsets-34
    ├── __consumer_offsets-44
    ├── __consumer_offsets-8
    └── [redacted]
/data3
└── kafka-logs
    ├── __consumer_offsets-0
    ├── __consumer_offsets-22
    ├── __consumer_offsets-36
    ├── __consumer_offsets-42
    ├── __consumer_offsets-43
    └── [redacted]
/data4
└── kafka-logs
    ├── __consumer_offsets-2
    ├── __consumer_offsets-29
    ├── __consumer_offsets-37
    ├── __consumer_offsets-49
    ├── [redacted]
    └── [redacted]
```

어떻게 저장되나

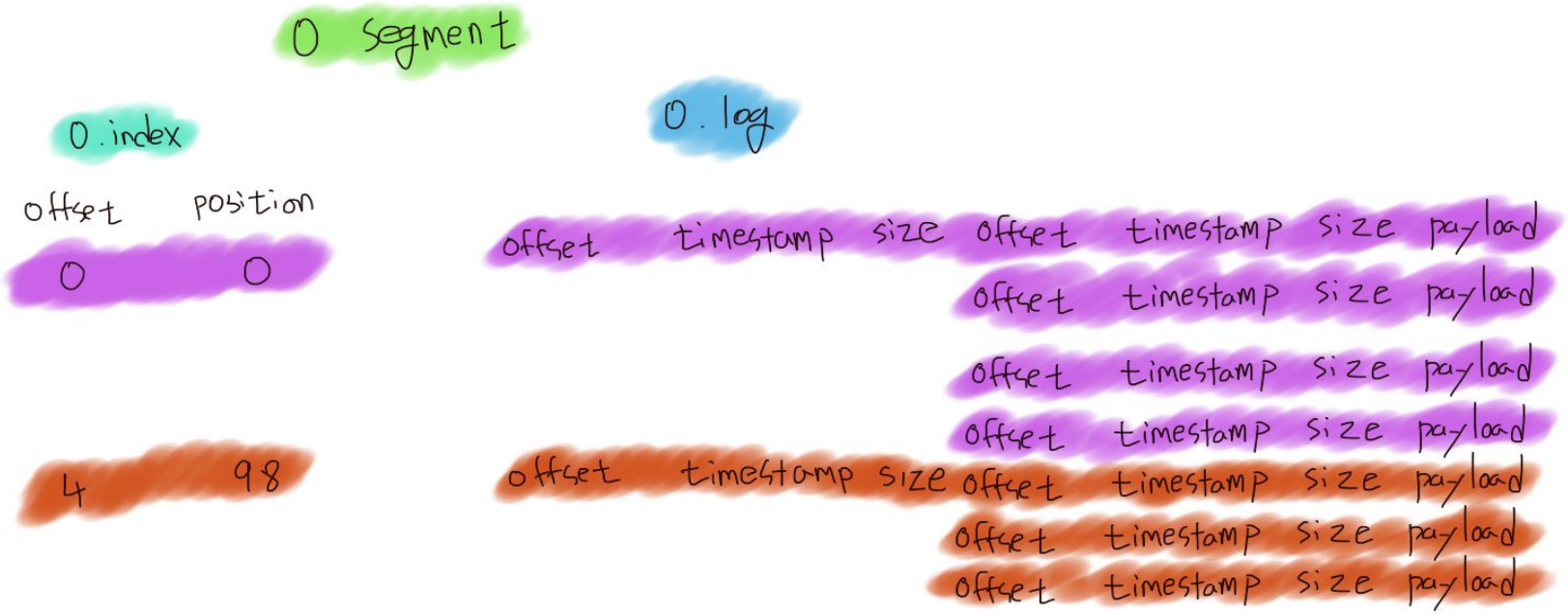
Segment 단위로 파일 저장

*.index, *.timeindex

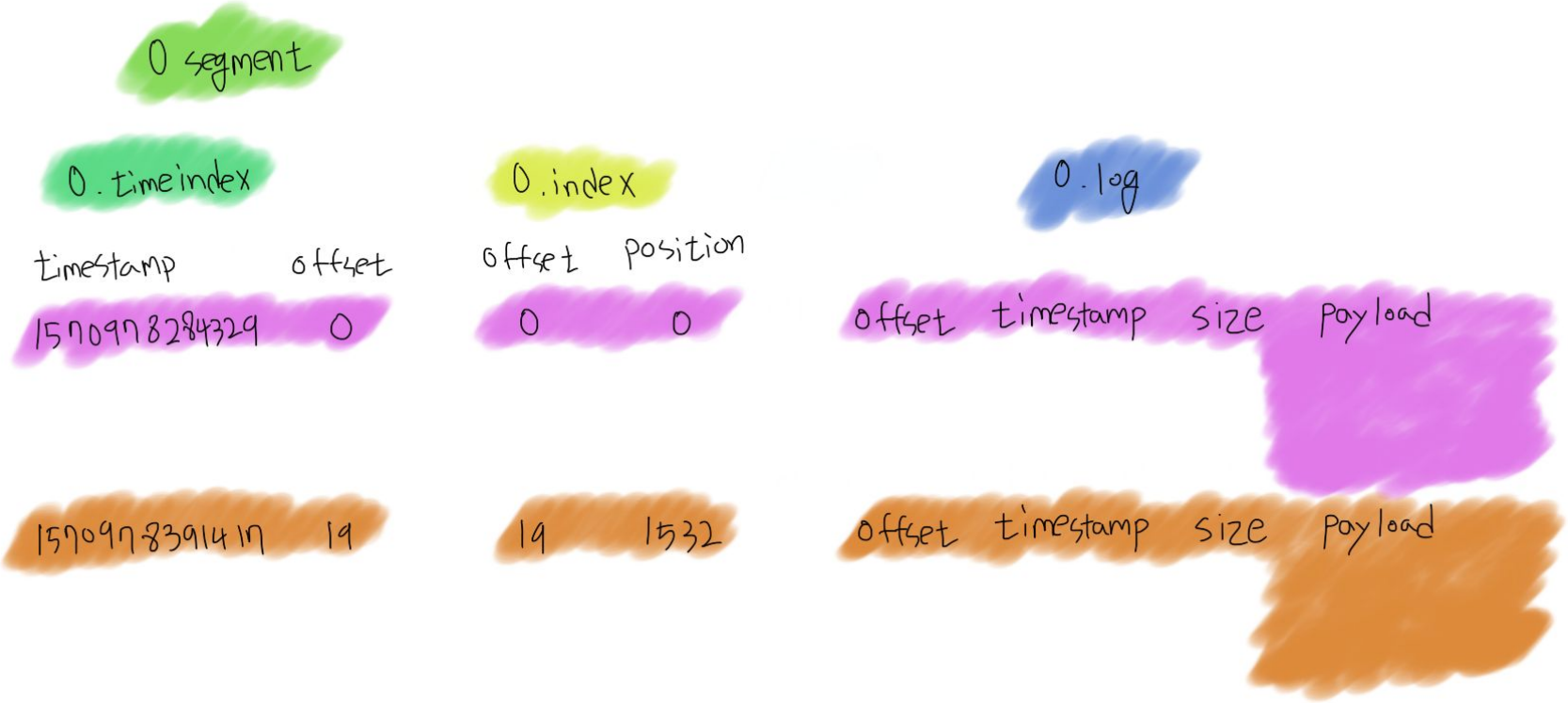
*.log, *.snapshot

```
-rw-r--r-- 1 root root 1.8M  2월  21 18:48 00000000000219311110.index
-rw-r--r-- 1 root root 1.0G  2월  21 18:48 00000000000219311110.log
-rw-r--r-- 1 root root 2.7M  2월  21 18:48 00000000000219311110.timeindex
-rw-r--r-- 1 root root 1.8M  2월  22 13:56 00000000000220688777.index
-rw-r--r-- 1 root root 1.0G  2월  22 13:56 00000000000220688777.log
-rw-r--r-- 1 root root 2.7M  2월  22 13:56 00000000000220688777.timeindex
-rw-r--r-- 1 root root 1.8M  2월  22 23:31 00000000000222069103.index
-rw-r--r-- 1 root root 1.0G  2월  22 23:31 00000000000222069103.log
-rw-r--r-- 1 root root 2.7M  2월  22 23:31 00000000000222069103.timeindex
-rw-r--r-- 1 root root 1.8M  2월  23 14:43 00000000000223448069.index
-rw-r--r-- 1 root root 1.0G  2월  23 14:43 00000000000223448069.log
-rw-r--r-- 1 root root 2.7M  2월  23 14:43 00000000000223448069.timeindex
-rw-r--r-- 1 root root 1.8M  2월  24 00:43 00000000000224827223.index
-rw-r--r-- 1 root root 1.0G  2월  24 00:43 00000000000224827223.log
-rw-r--r-- 1 root root 2.7M  2월  24 00:43 00000000000224827223.timeindex
-rw-r--r-- 1 root root 1.8M  2월  24 16:54 00000000000226205693.index
-rw-r--r-- 1 root root 1.0G  2월  24 16:54 00000000000226205693.log
-rw-r--r-- 1 root root   10  2월  24 00:43 00000000000226205693.snapshot
-rw-r--r-- 1 root root 2.7M  2월  24 16:54 00000000000226205693.timeindex
-rw-r--r-- 1 root root  10M  2월  24 17:41 00000000000227583737.index
-rw-r--r-- 1 root root 109M  2월  24 17:41 00000000000227583737.log
-rw-r--r-- 1 root root   10  2월  24 16:54 00000000000227583737.snapshot
-rw-r--r-- 1 root root  10M  2월  24 17:41 00000000000227583737.timeindex
-rw-r--r-- 1 root root   16  2월  24 08:33 leader-epoch-checkpoint
```

어떻게 저장되나 (index)



어떻게 저장되나 (timeindex)

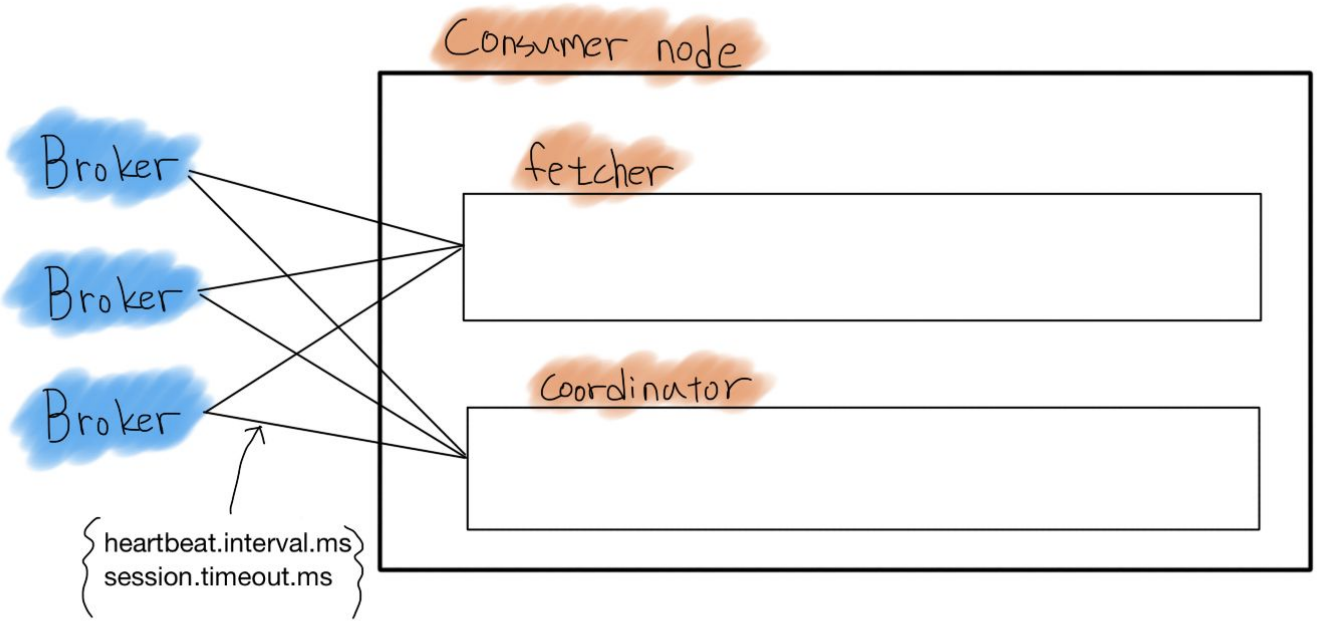


Consumer

Consumer Code

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("my-topic"));

while (true) {
    ConsumerRecords<String, String> records = consumer.poll(100);
    for (ConsumerRecord<String, String> record : records)
        System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(),
              record.key(), record.value());
}
```

KafkaConsumer 생성



KafkaConsumer 생성

- Fetcher

  poll 함수가 실행되면 적절한 크기의 records 리턴.
  내부에 적합한 records가 없다면 Broker에게 records를 요청 및 내부 저장.
  그리고 적절한 크기의 record 리턴하는 역할

- Coordinator

  어떤 토픽, 파티션을 consume 할지
  Broker의 group coordinator와 통신하는 역할
  heartbeat, offset commit, consumer group join 도 합니다~

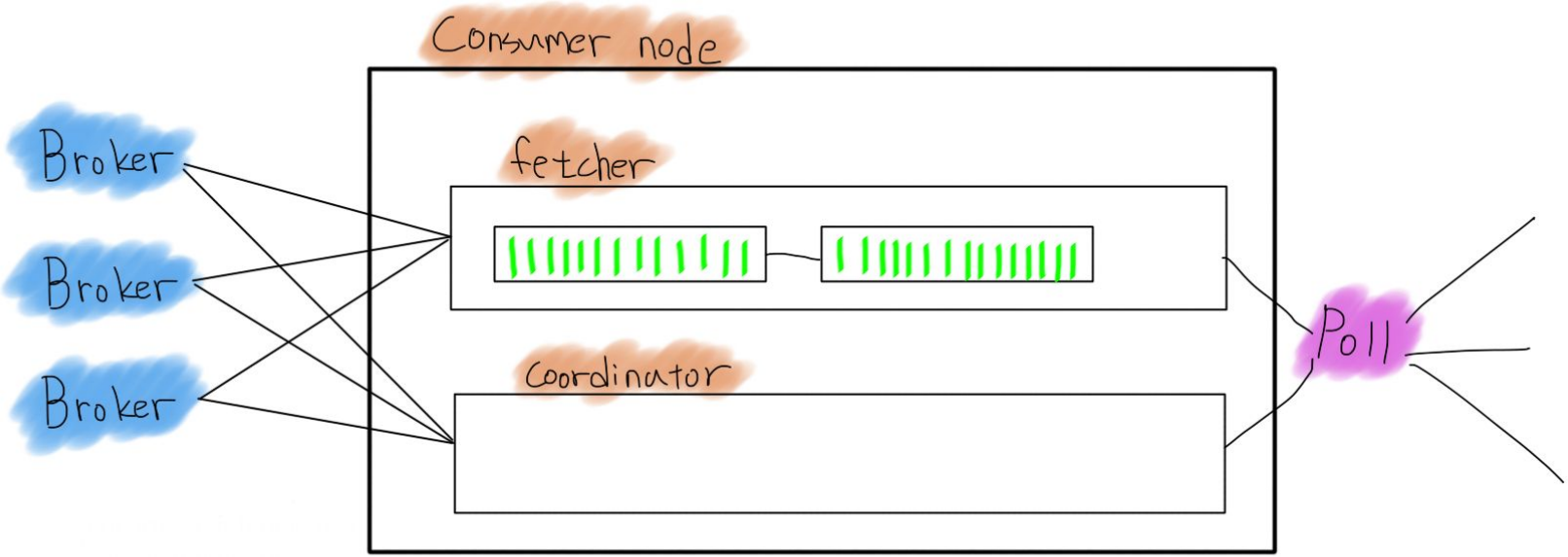
Code

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("my-topic"));

while (true) {
    ConsumerRecords<String, String> records = consumer.poll(100);
    for (ConsumerRecord<String, String> record : records)
        System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(),
              record.key(), record.value());
}
```

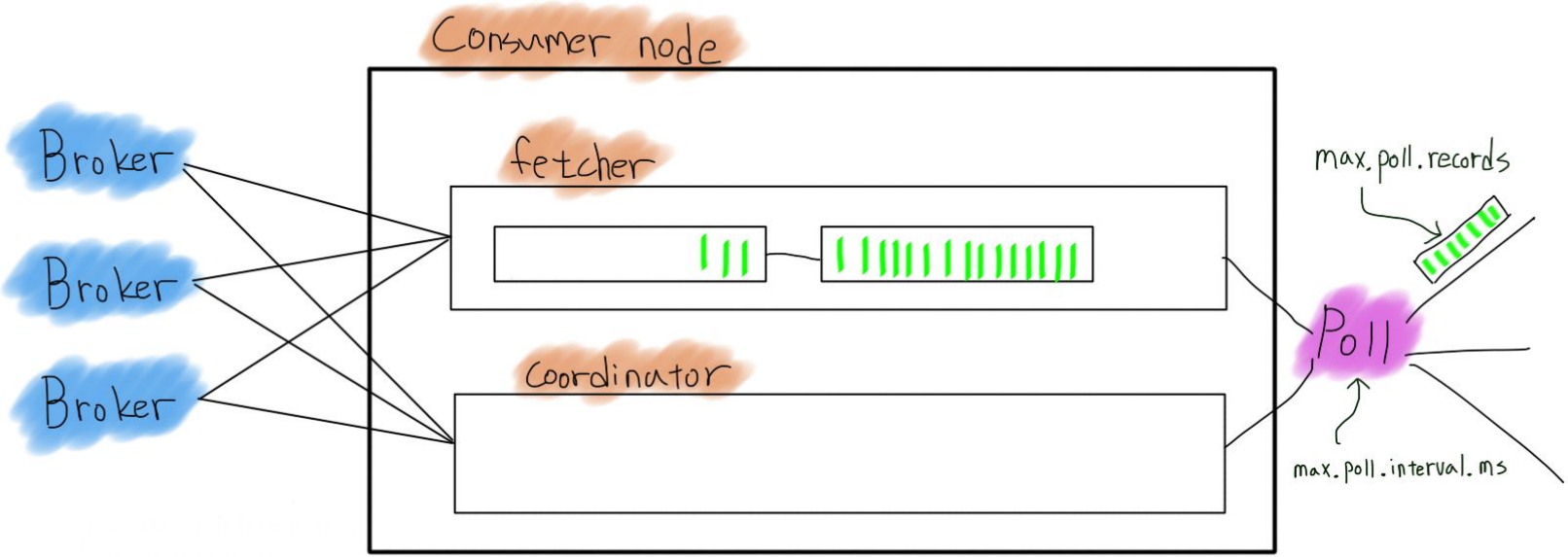
Code

```
// Topic, Partition, Offset의 정보를 갱신한다.
// ※ rebalance, auto commit offset도 이곳에서 실행
coordinator.poll(startMs, timeout);

// 이미 fetcher에 records가 있다면 바로 리턴
Map<TopicPartition, List<ConsumerRecord<K,V>>> records =
                                    fetcher.fetchedRecords();
if (!records.isEmpty())
    return records;

// records가 없다면 Broker에게 데이터 요청 및 내부 저장
fetcher.sendFetches();
client.poll(pollTimeout, nowMs, ...);

// rebalance가 필요하다면 빈 records 리턴
if (coordinator.needRejoin())
    return Collections.emptyMap();

// records 리턴
return fetcher.fetchRecords();
```

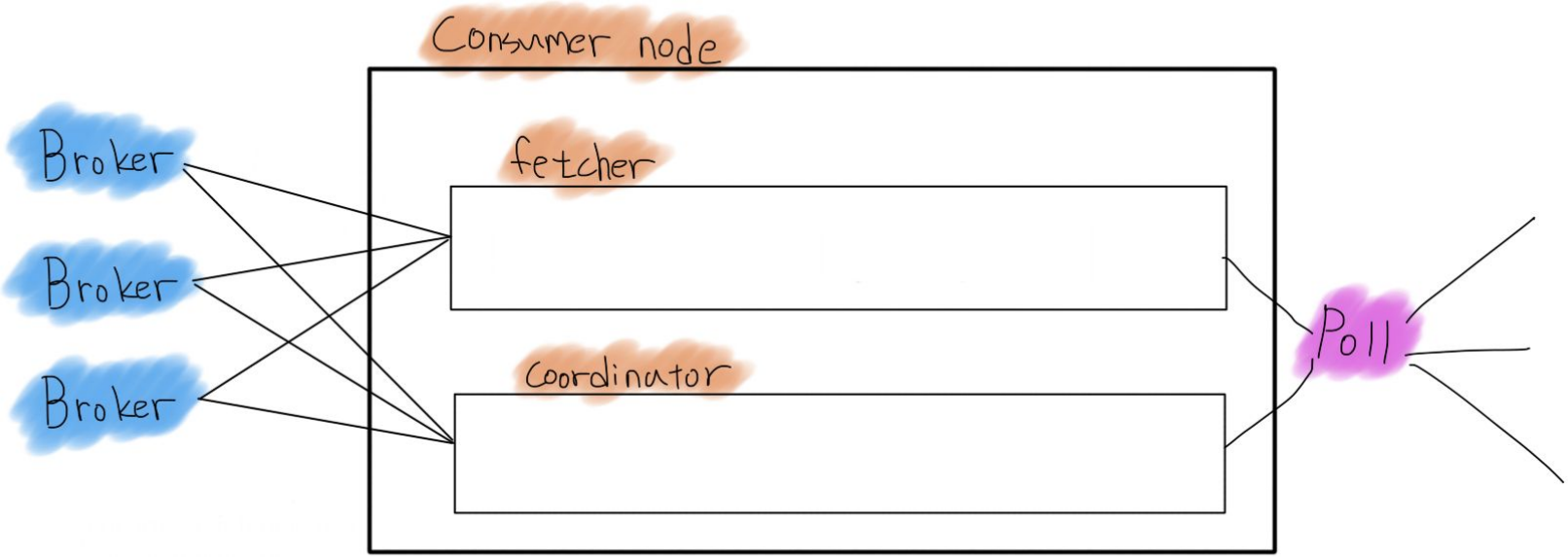
데이터 가져오기 (fetcher에 records가 있는 경우)



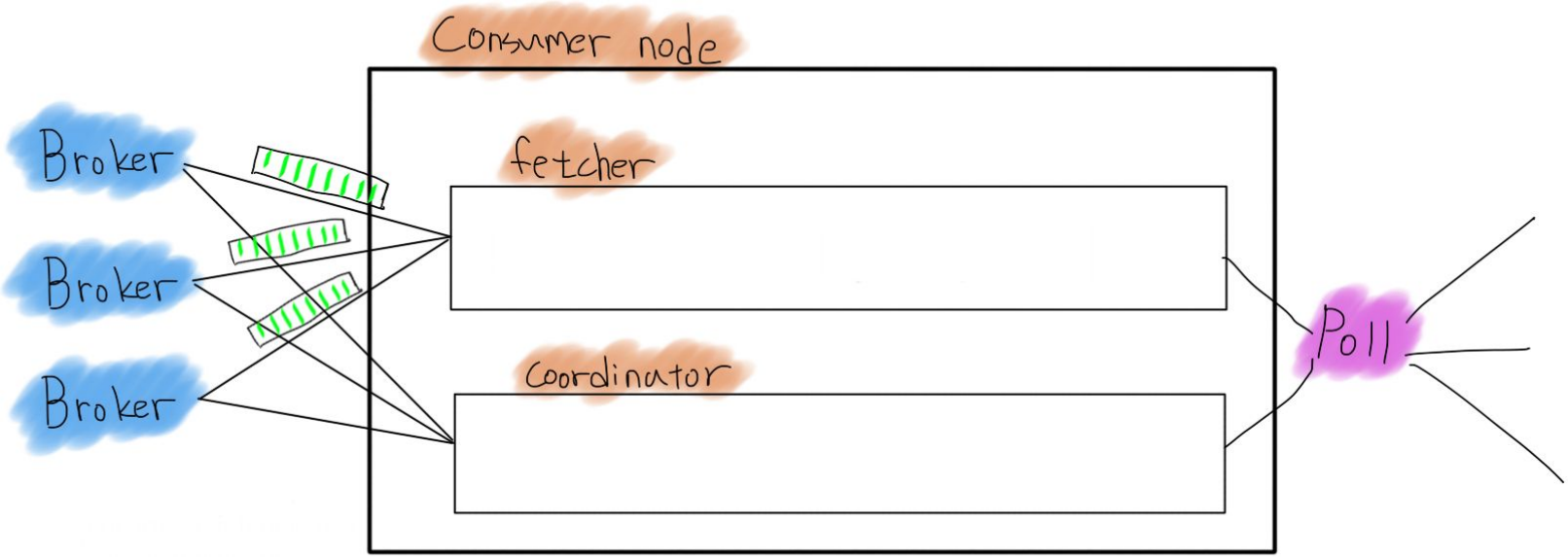
데이터 가져오기 (fetcher에 records가 있는 경우)



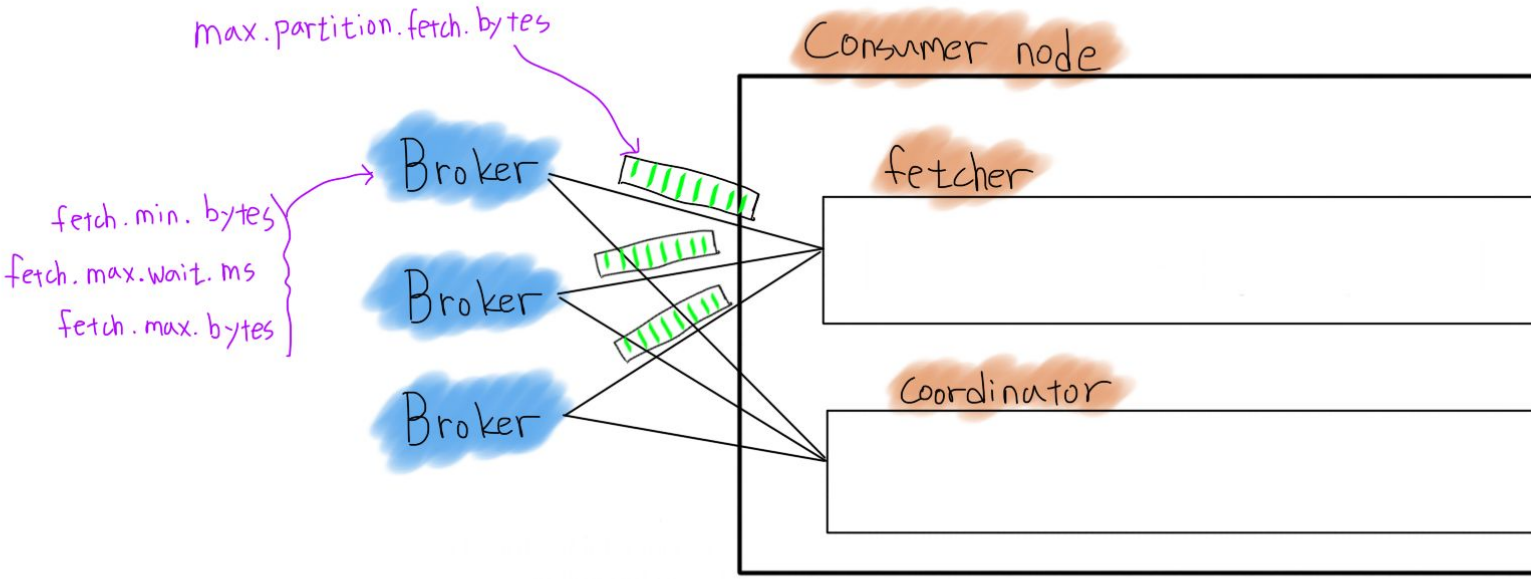
데이터 가져오기 (fetcher에 records가 없는 경우)



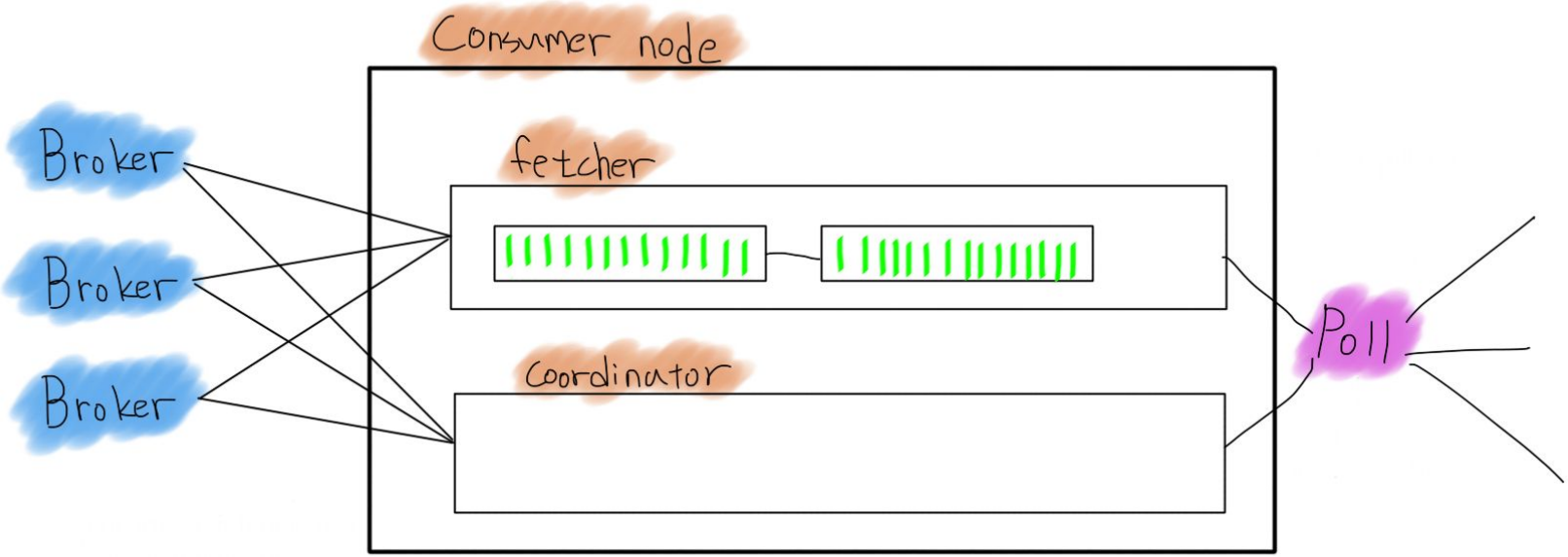
데이터 가져오기 (fetcher에 records가 없는 경우)



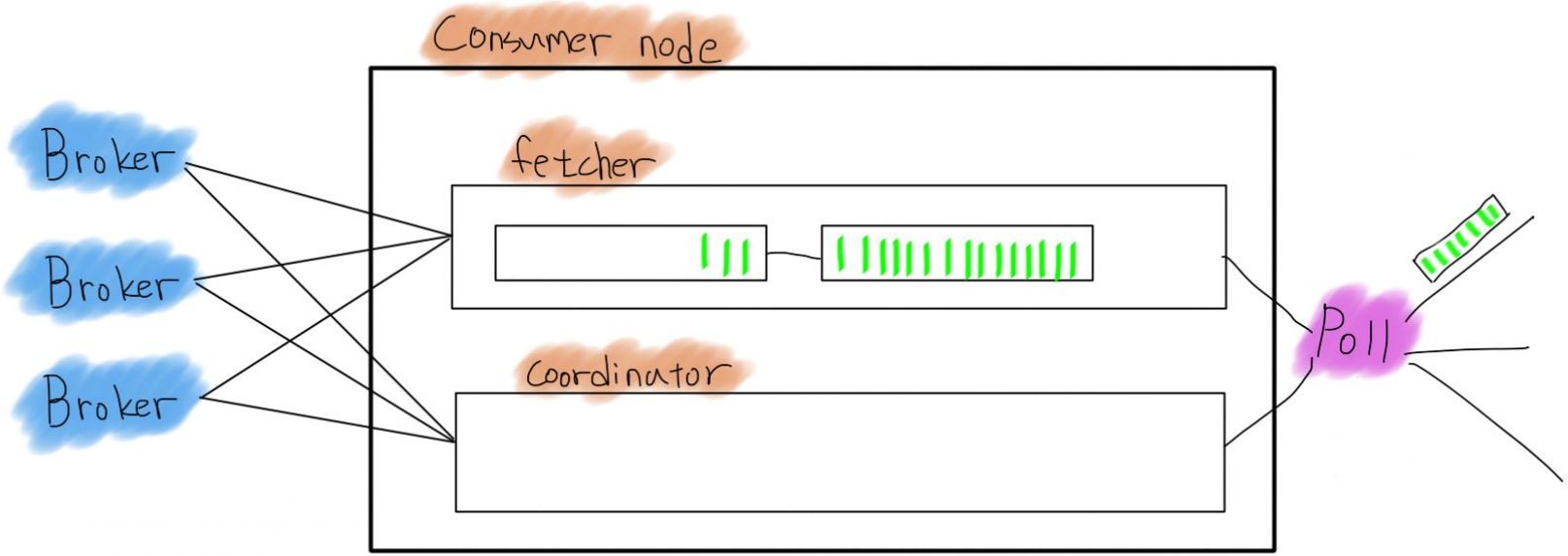
데이터 가져오기 (fetcher에 records가 없는 경우)



데이터 가져오기 (fetcher에 records가 없는 경우)



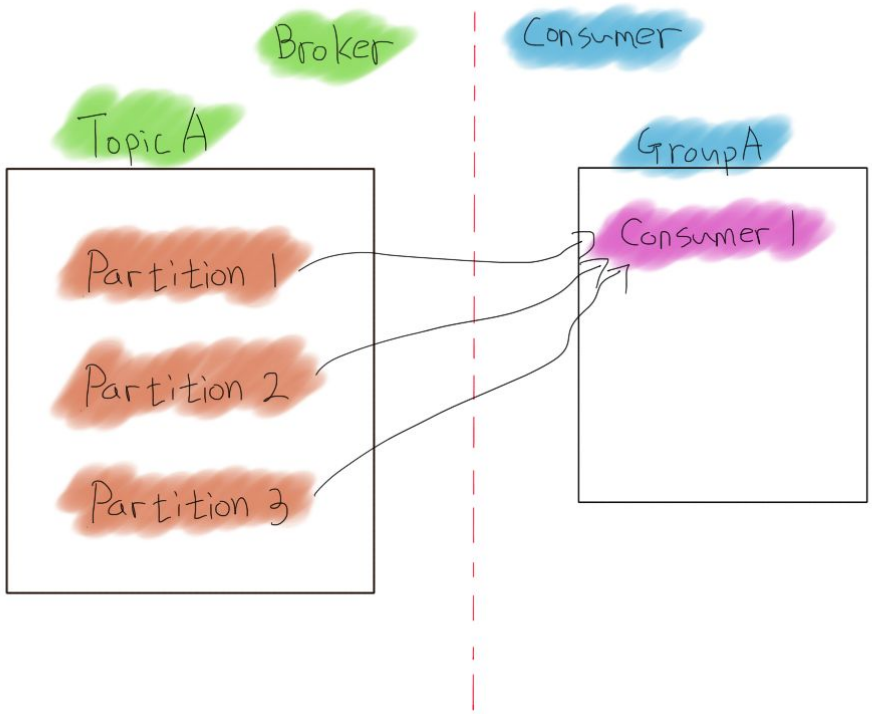
데이터 가져오기 (fetcher에 records가 없는 경우)



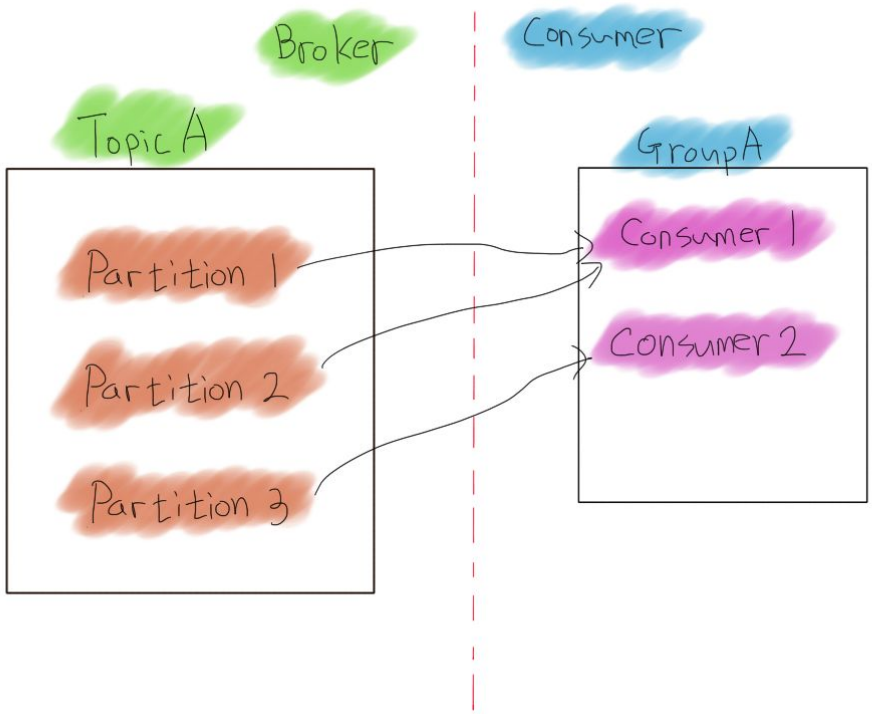
부록

Consumer Rebalance

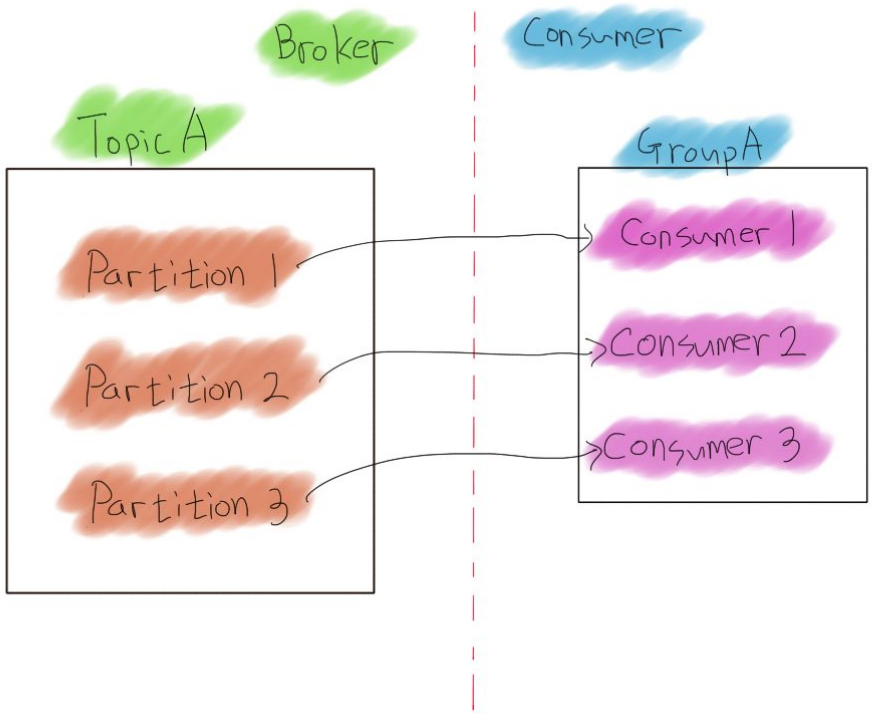
Consumer Rebalance



Consumer Rebalance



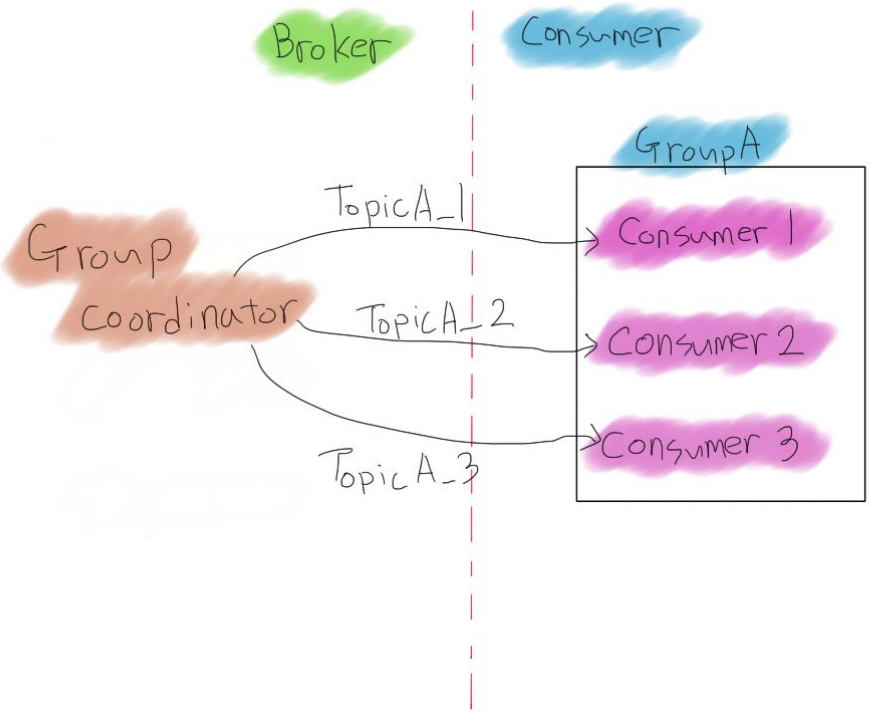
Consumer Rebalance



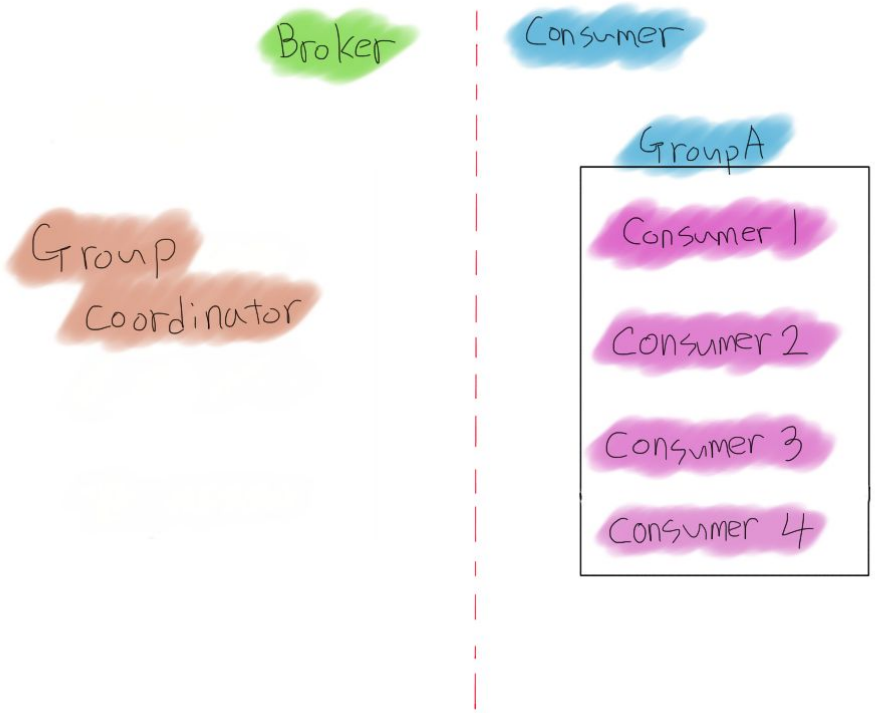
Consumer Rebalance

```
// Topic, Partition, Offset의 정보를 갱신한다.
// ※ rebalance, auto commit offset도 이곳에서 실행
coordinator.poll(startMs, timeout);

// 이미 fetcher에 records가 있다면 바로 리턴
Map<TopicPartition, List<ConsumerRecord<K,V>>> records =
                                        fetcher.fetchedRecords();
if (!records.isEmpty())
    return records;

// records가 없다면 Broker에게 데이터 요청 및 내부 저장
fetcher.sendFetches();
client.poll(pollTimeout, nowMs, ...);

// rebalance가 필요하다면 빈 records 리턴
if (coordinator.needRejoin())
    return Collections.emptyMap();

// records 리턴
return fetcher.fetchRecords();
```

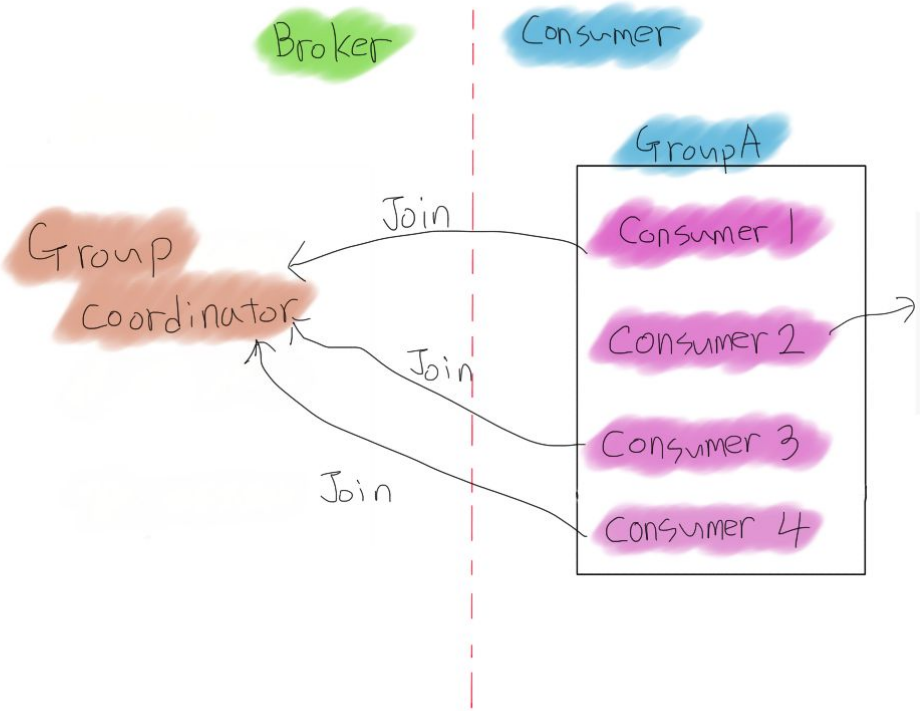
Consumer Rebalance



Consumer Rebalance

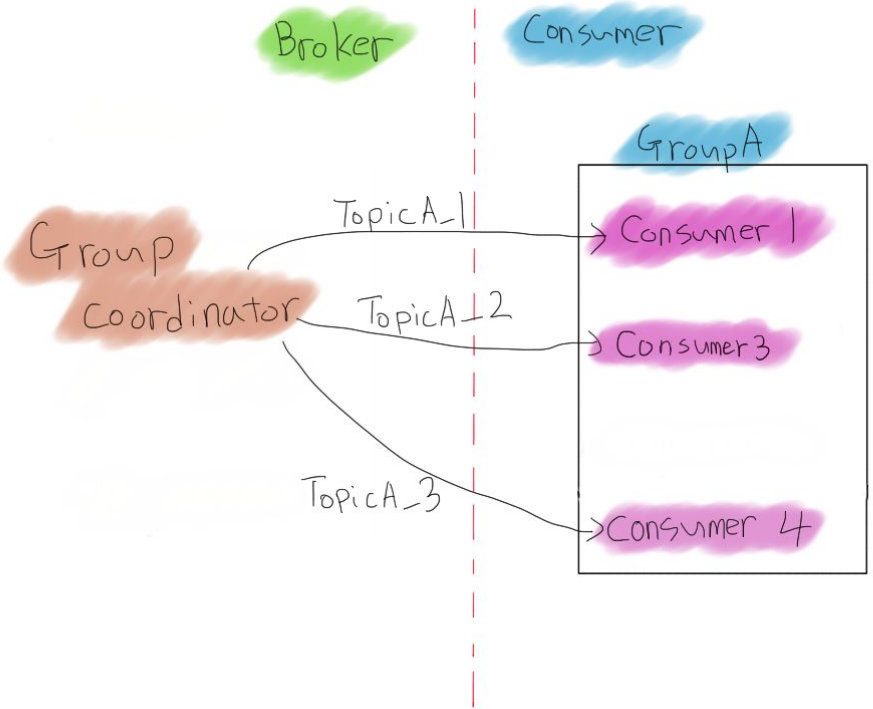


Consumer Rebalance



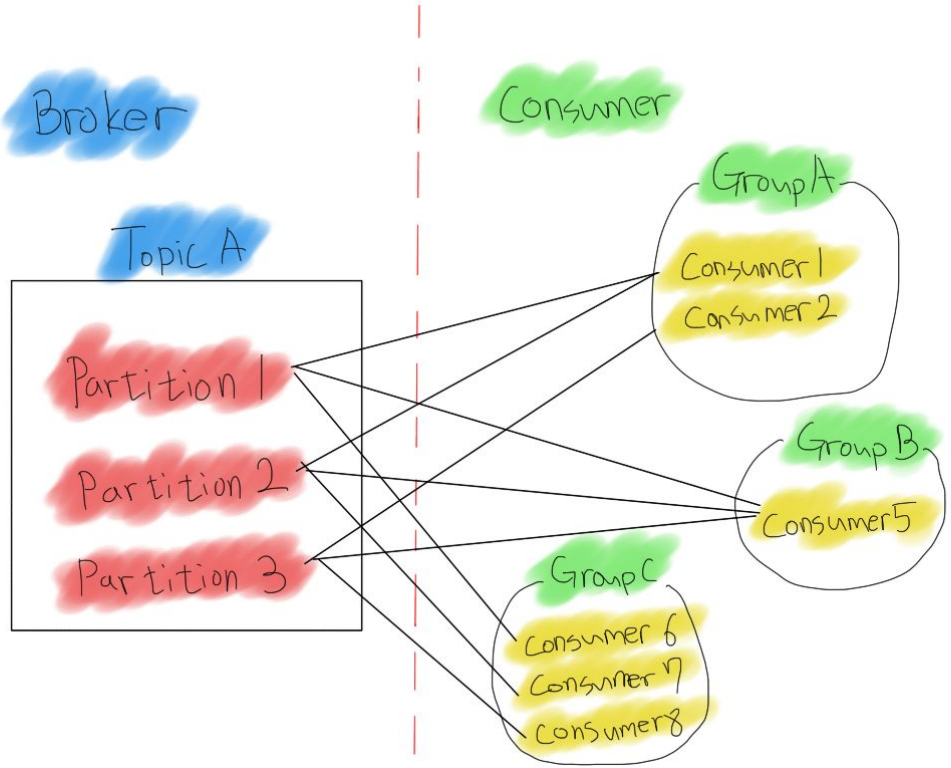

```
while (true) {
  val records = consumer.poll(100)
  for (record: records) {
    // do something...
  }
}
```

Consumer Rebalance



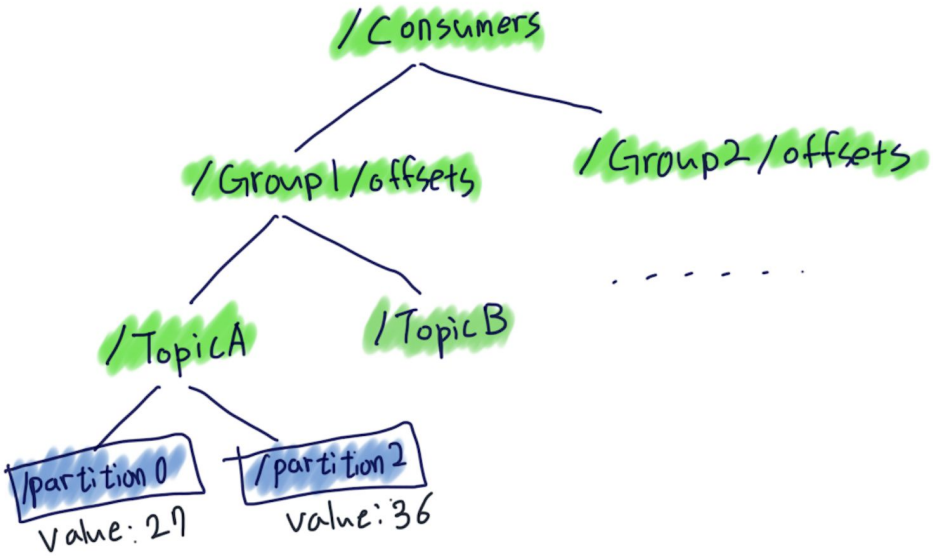
Consumer Group offset

Consumer Group offset



Consumer Group offset

Ver. 0.9 미만 에서는 zookeeper에 consumer offset을 저장



참조 : https://elang2.github.io/myblog/posts/2017-09-20-Kafak-And-Zookeeper-Offsets.html

Consumer Group offset

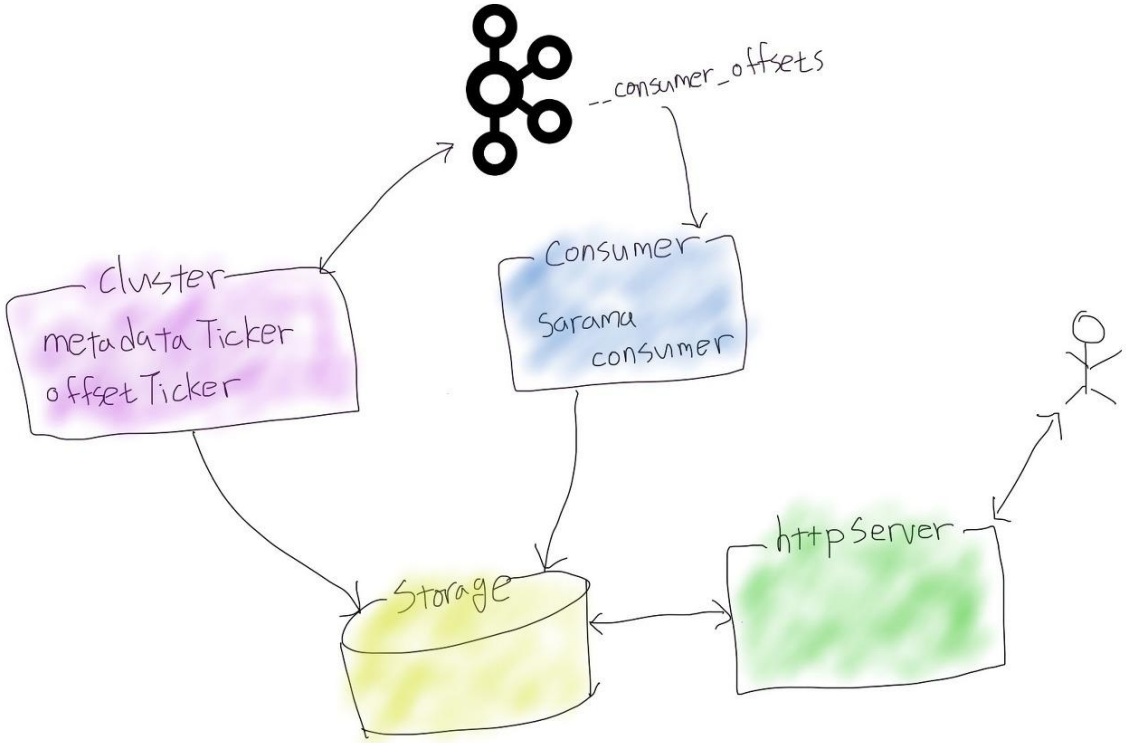
Ver. 0.9 이상 __consumer_offset 토픽을 사용

Group name	Topic name	partition	offset	Commit time
test-01	my-topic	1	0	1551191950
test-02	my-topic	1	0	1551193842
test-01	my-topic	1	10	1551203421
test-01	my-topic	1	19	1551243229

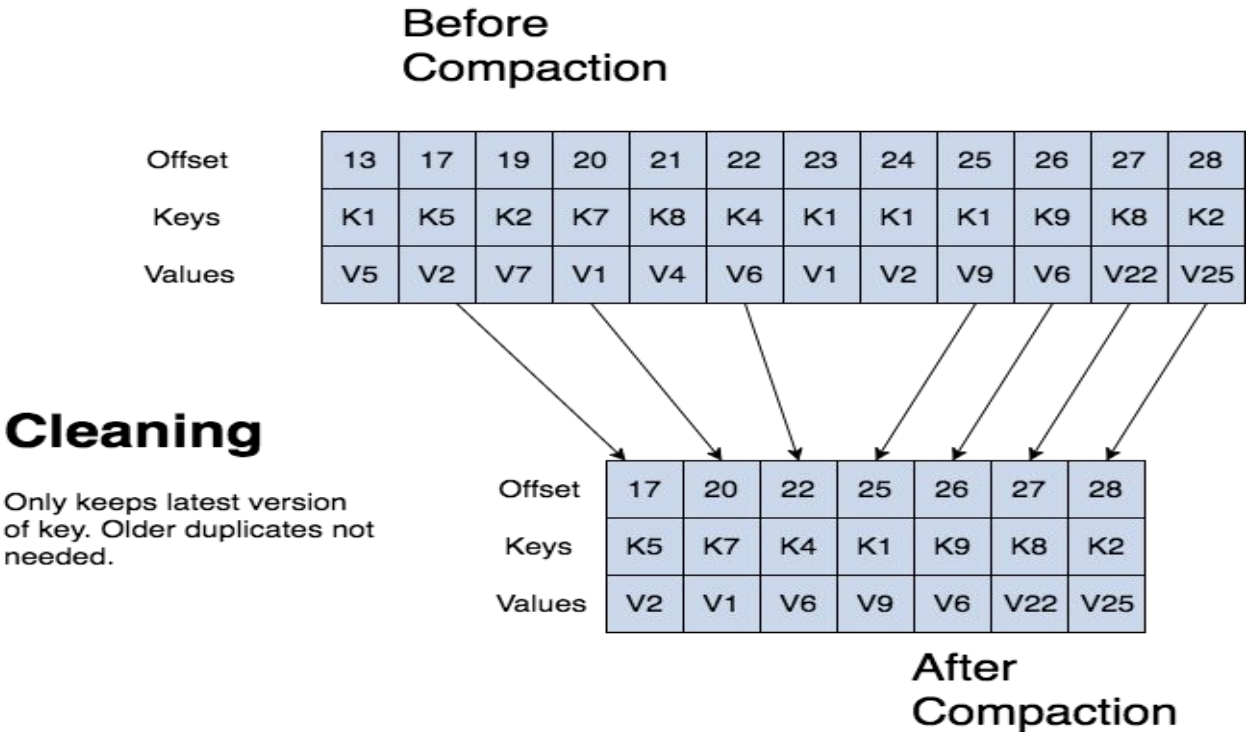
OLD

NEW

Burrow


__consumer_offsets
Cluster
metadata Ticker
offset Ticker
Consumer
Sarama
consumer
Storage
http Server

Consumer Group offset

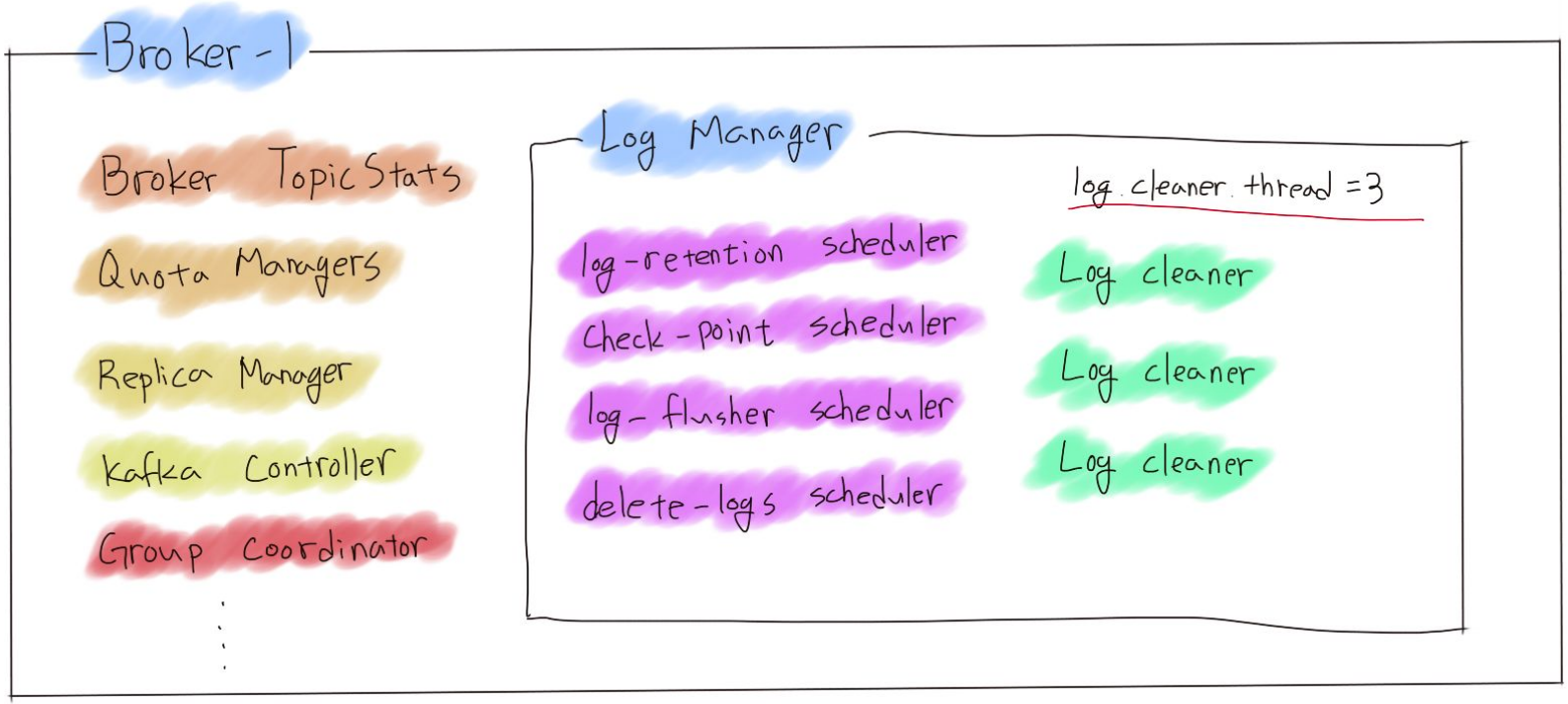


Cleaning

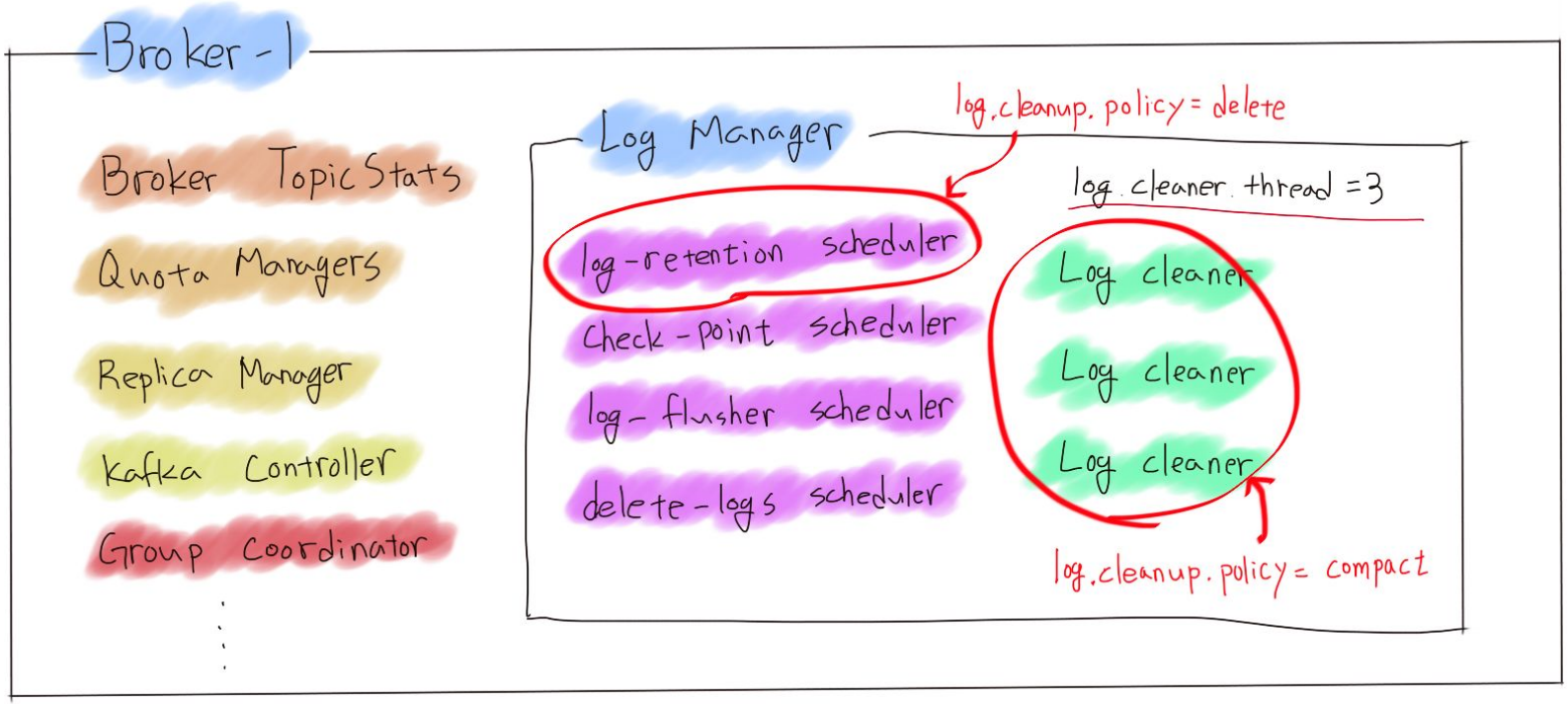
Only keeps latest version of key. Older duplicates not needed.

Log Manager

Log Manager

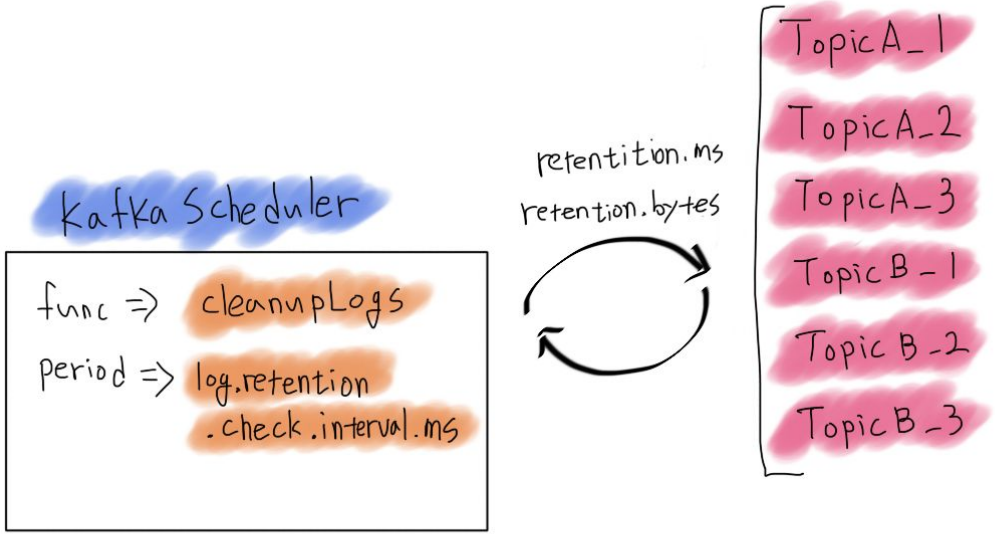

Broker-1
Broker TopicStats
Quota Managers
Replica Manager
Kafka Controller
Group Coordinator
Log Manager
log.cleaner.thread=3
log-retention scheduler
Check-point scheduler
log-flusher scheduler
delete-logs scheduler
Log cleaner
Log cleaner
Log cleaner

Log Manager

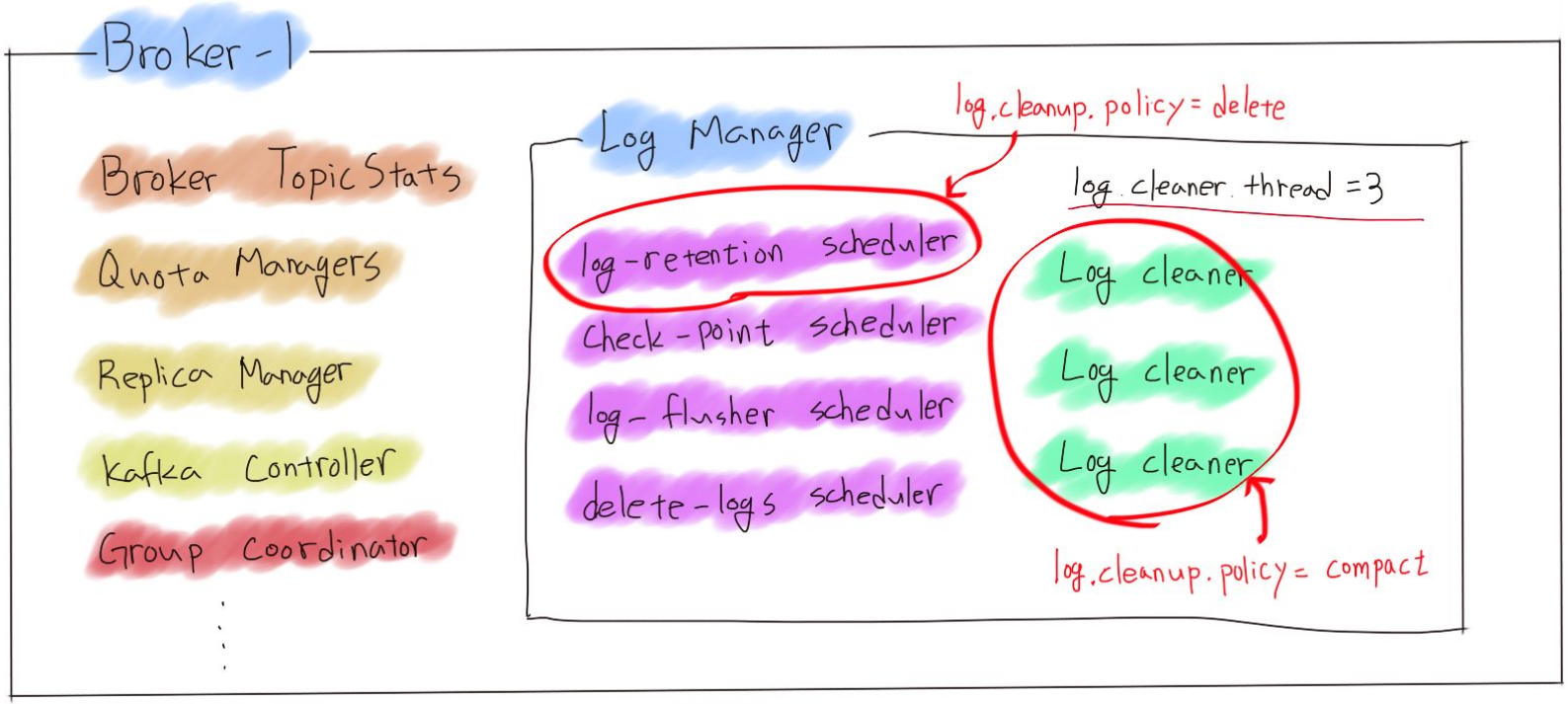

Broker-1
Broker Topic Stats
Quota Managers
Replica Manager
Kafka Controller
Group Coordinator
Log Manager
log.cleanup.policy = delete
log.cleaner.thread = 3
log-retention scheduler
Check-point scheduler
log-flusher scheduler
delete-logs scheduler
Log cleaner
Log cleaner
Log cleaner
log.cleanup.policy = compact

Log Manager

log.cleanup.policy = delete



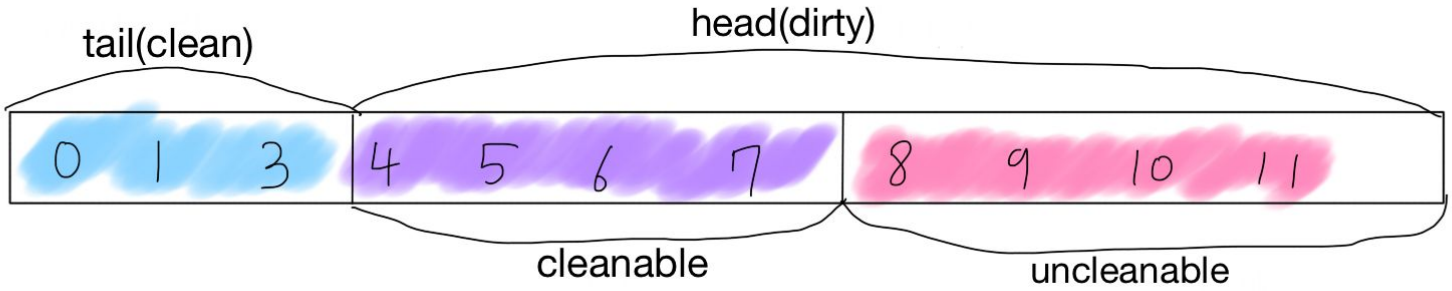
Log Manager


Broker-1
Broker Topic Stats
Quota Managers
Replica Manager
Kafka Controller
Group Coordinator
Log Manager
log.cleanup.policy = delete
log.cleaner.thread = 3
log-retention scheduler
Check-point scheduler
log-flusher scheduler
delete-logs scheduler
Log cleaner
Log cleaner
Log cleaner
log.cleanup.policy = compact

Log Manager

$$\text{min.cleanable.dirty.ratio} < \frac{\text{cleanable.size}}{\text{clean.size} + \text{cleanable.size}}$$

log.cleanup.policy = compact



Key	a	b	c	b	c	f	b	a	b	c	d
Value	1	2	3	4	9	8	7	12	3	4	5

0 segment 4 segment 8 segment (active)

Log Manager

log.cleanup.policy = compact

	0	1	3	4	5	6	7
K	a	b	c	b	c	f	b
V	1	2	3	4	9	8	7

key	offset
b	7
f	6
c	5

0	5	6	7
a	c	f	b
1	5	6	7

0 segment. cleaned

Log Manager

log.cleanup.policy = compact

	0	1	3	4	5	6	7
K	a	b	c	b	c	f	b
V	1	2	3	4	9	8	7

0 segment . deleted

4 segment . deleted

0	5	6	7
a	c	f	b
1	5	6	7

0 segment. swap

Log Manager

log.cleanup.policy = compact

	0	1	3	4	5	6	7
K	a	b	c	b	c	f	b
V	1	2	3	4	9	8	7

0 segment . deleted

4 segment . deleted

log . segment . delete . delay . ms

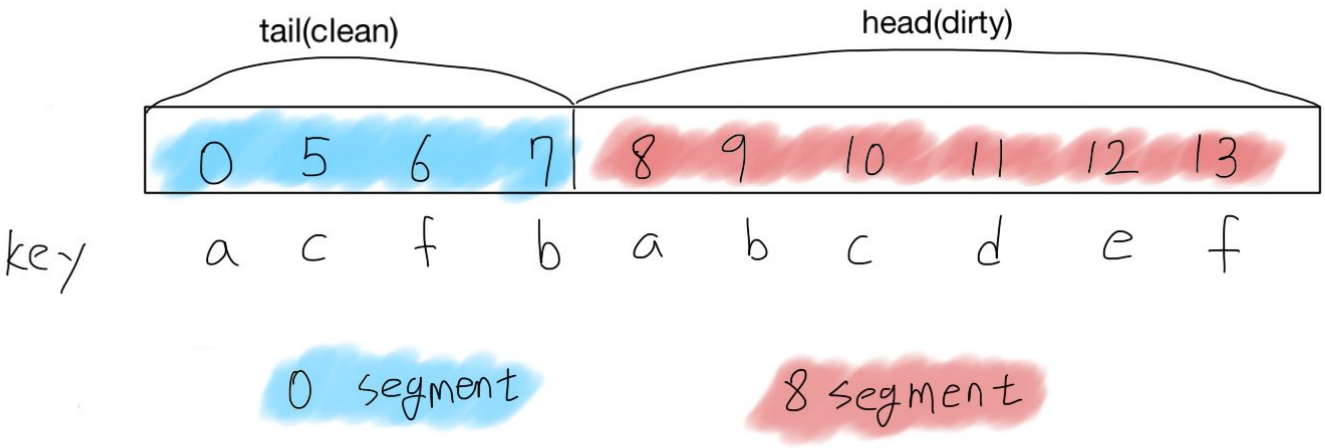
0	5	6	7
a	c	f	b
1	5	6	7

0 segment

Log Manager

log.cleanup.policy = compact

log.cleaner.min.compaction.lag.ms



못다한 이야기

1. Exactly-Once(Transaction)
2. Purgatory
3. Controller in Broker and Leader Election
4. Metrics

감사합니다.