
Kafka

프로듀서 & 컨슈머

Index

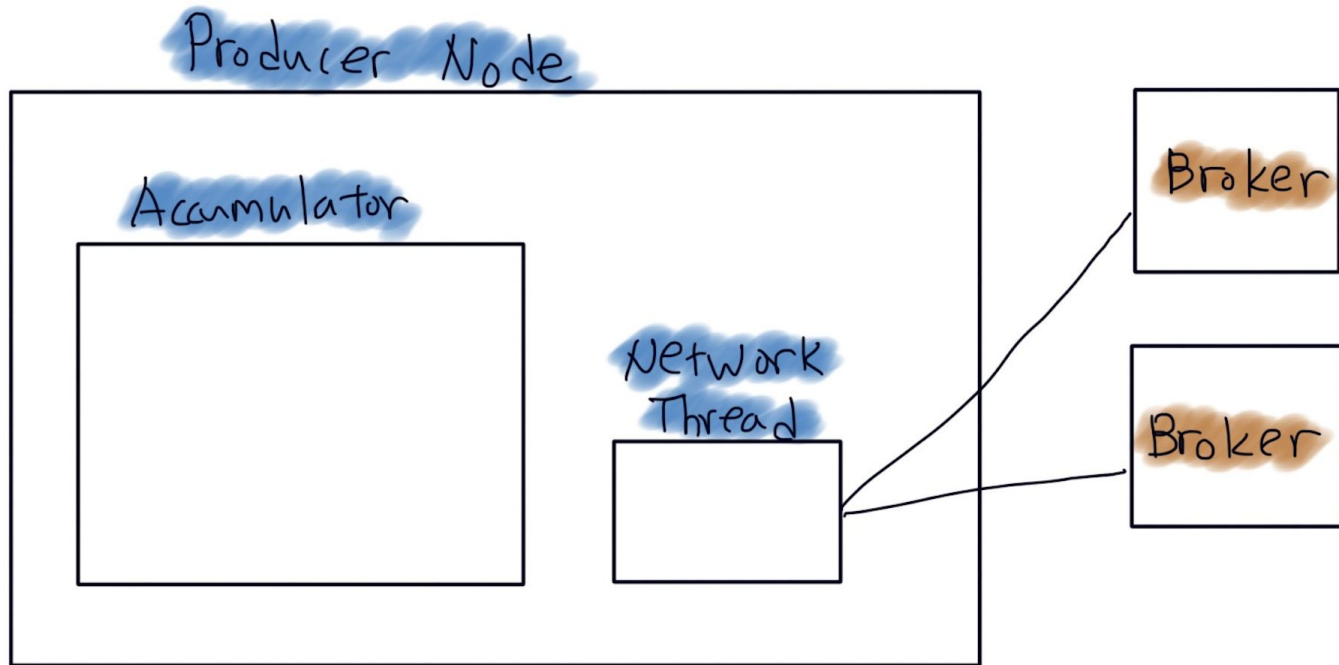
- Producer
- Broker
- Consumer
- Q&A

Producer

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

사용자가 send한 Record를 메모리(RecordBatch)에 차곡차곡 쌓아주는 역할

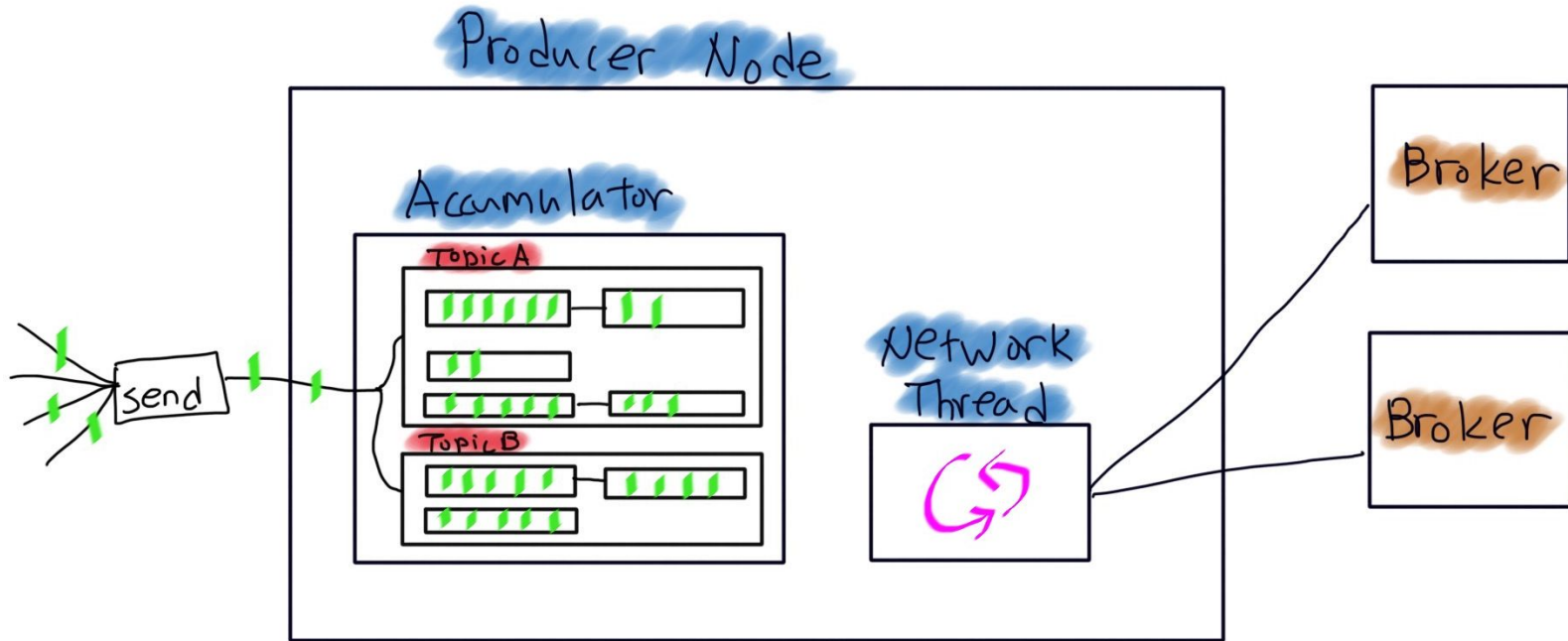
- Network Thread

Accumulator에서 만든 RecordBatch를 Broker로 전송하는 역할

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));  
}
```

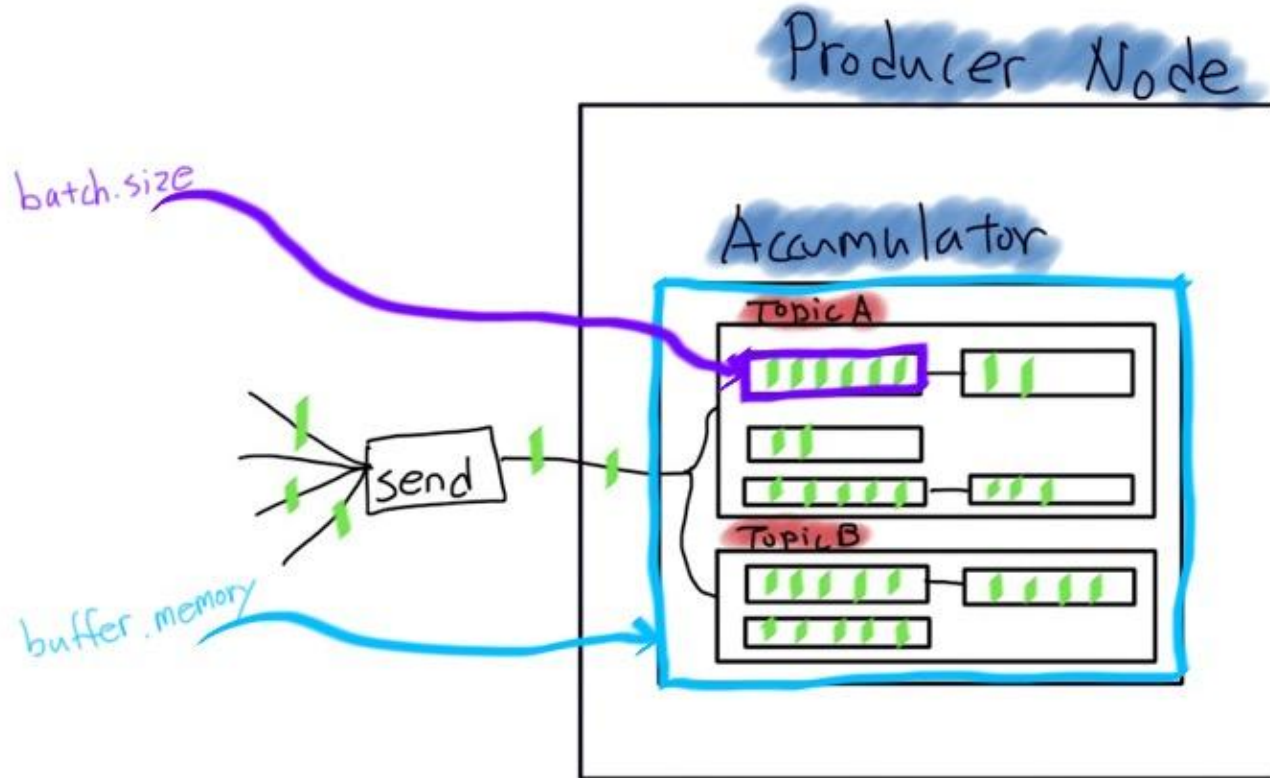
Accumulator



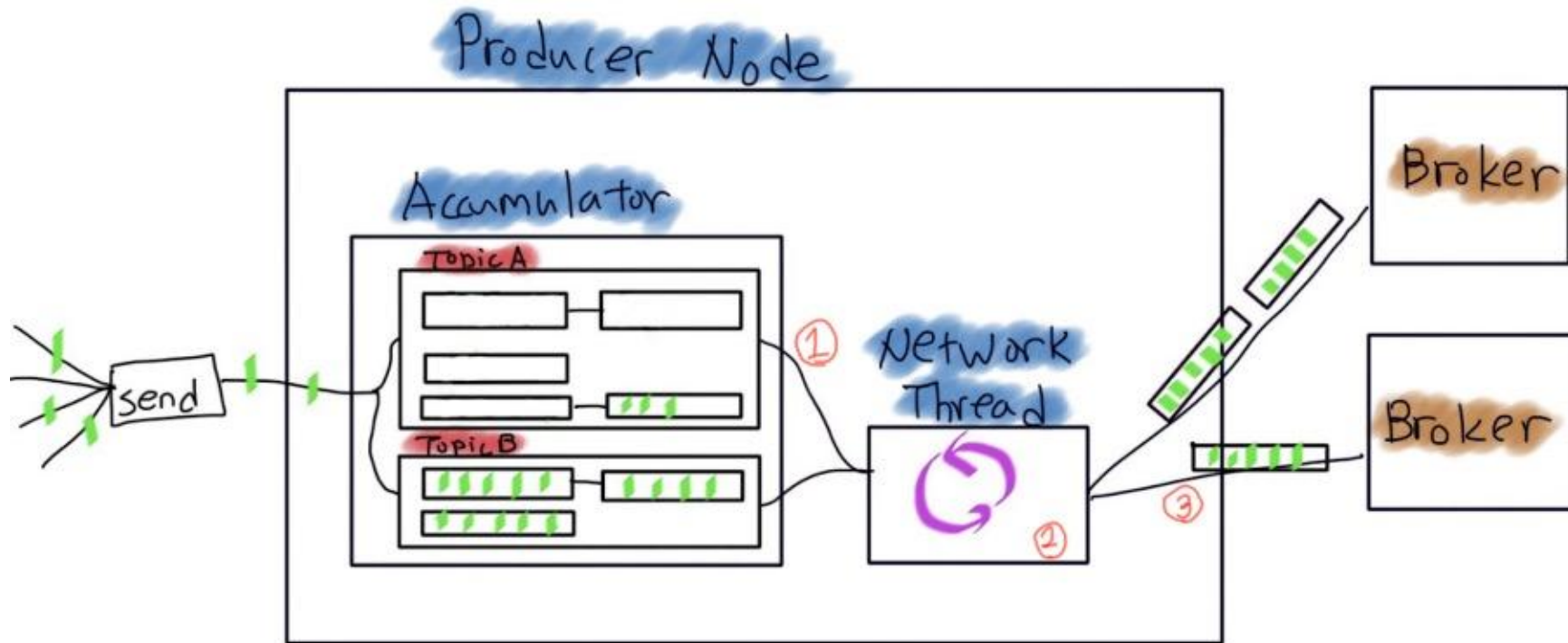
Accumulator

record를 topic + partition
별로 모아두는 대기 공간
(?)

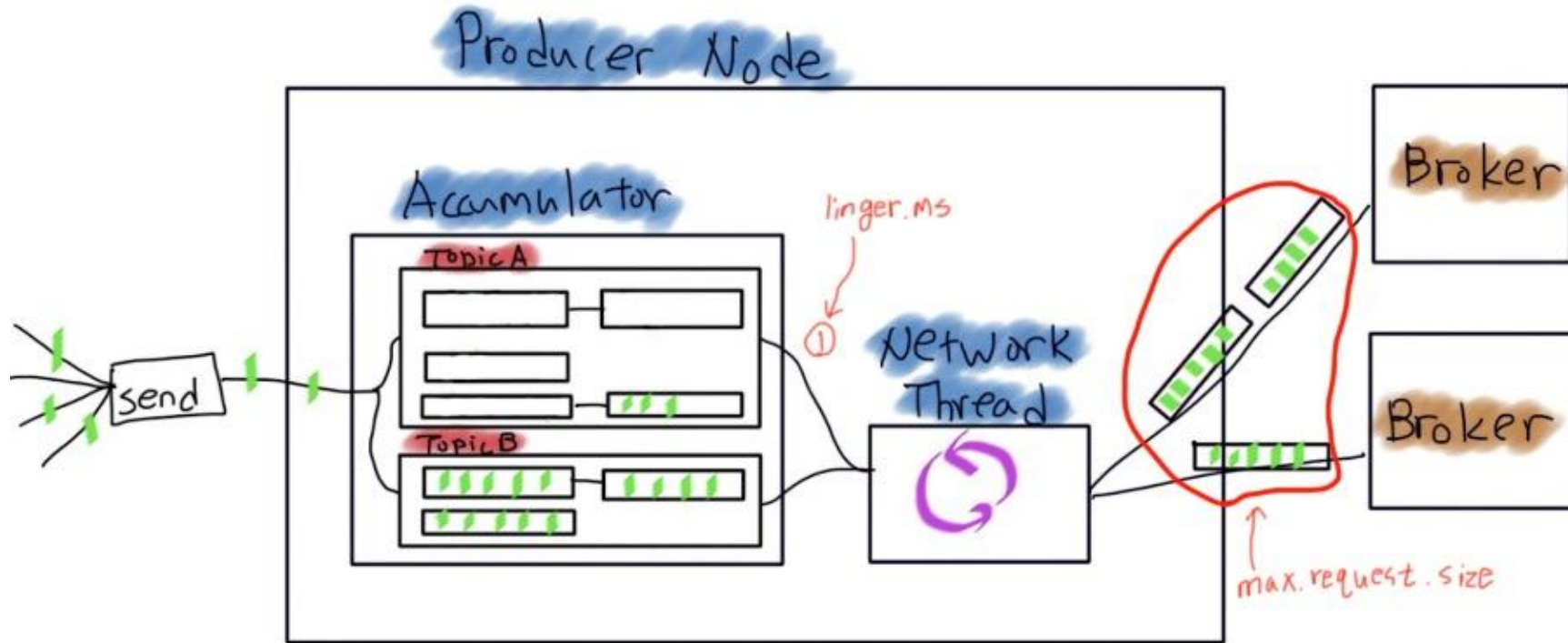
Accumulator에서
사용하는
메모리 크기



Network Thread

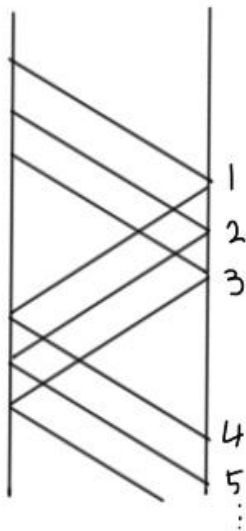


Network Thread

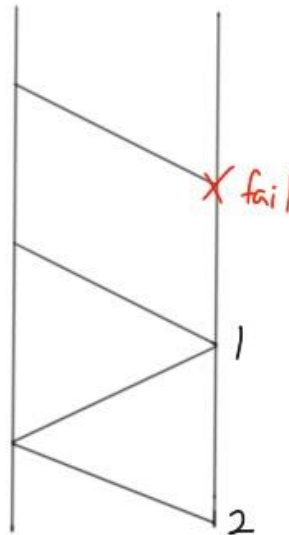
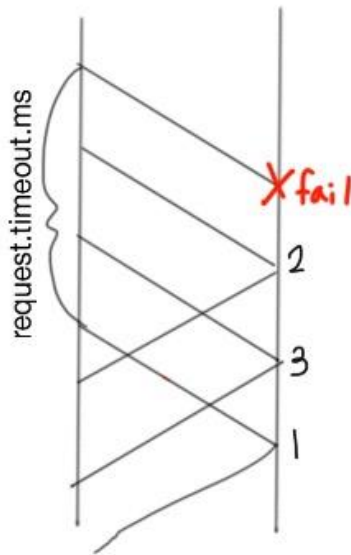


Network Thread

왼쪽이 Network Thread, 오른쪽이 Broker



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



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

Broker

어떻게 저장되나

[Topic name]-[partition] 폴더 구조

```
[ 1 /]$ tree /data* -d
/data1
└─ kafka-logs
   ├── __consumer_offsets-10
   ├── __consumer_offsets-15
   ├── __consumer_offsets-18
   ├── __consumer_offsets-23
   ├── __consumer_offsets-30
   └─ [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]
```

어떻게 저장되나

Segment 단위로 파일 저장

*.index, *.log, *.timeindex

```
-rw-r--r-- 1 root root 1.8M 2월 21 18:48 000000000000219311110.index
-rw-r--r-- 1 root root 1.0G 2월 21 18:48 000000000000219311110.log
-rw-r--r-- 1 root root 2.7M 2월 21 18:48 000000000000219311110.timeindex
-rw-r--r-- 1 root root 1.8M 2월 22 13:56 000000000000220688777.index
-rw-r--r-- 1 root root 1.0G 2월 22 13:56 000000000000220688777.log
-rw-r--r-- 1 root root 2.7M 2월 22 13:56 000000000000220688777.timeindex
-rw-r--r-- 1 root root 1.8M 2월 22 23:31 000000000000222069103.index
-rw-r--r-- 1 root root 1.0G 2월 22 23:31 000000000000222069103.log
-rw-r--r-- 1 root root 2.7M 2월 22 23:31 000000000000222069103.timeindex
-rw-r--r-- 1 root root 1.8M 2월 23 14:43 000000000000223448069.index
-rw-r--r-- 1 root root 1.0G 2월 23 14:43 000000000000223448069.log
-rw-r--r-- 1 root root 2.7M 2월 23 14:43 000000000000223448069.timeindex
-rw-r--r-- 1 root root 1.8M 2월 24 00:43 000000000000224827223.index
-rw-r--r-- 1 root root 1.0G 2월 24 00:43 000000000000224827223.log
-rw-r--r-- 1 root root 2.7M 2월 24 00:43 000000000000224827223.timeindex
-rw-r--r-- 1 root root 1.8M 2월 24 16:54 000000000000226205693.index
-rw-r--r-- 1 root root 1.0G 2월 24 16:54 000000000000226205693.log
-rw-r--r-- 1 root root 10 2월 24 00:43 000000000000226205693.snapshot
-rw-r--r-- 1 root root 2.7M 2월 24 16:54 000000000000226205693.timeindex
-rw-r--r-- 1 root root 10M 2월 24 17:41 000000000000227583737.index
-rw-r--r-- 1 root root 109M 2월 24 17:41 000000000000227583737.log
-rw-r--r-- 1 root root 10 2월 24 16:54 000000000000227583737.snapshot
-rw-r--r-- 1 root root 10M 2월 24 17:41 000000000000227583737.timeindex
-rw-r--r-- 1 root root 16 2월 24 08:33 leader-epoch-checkpoint
```

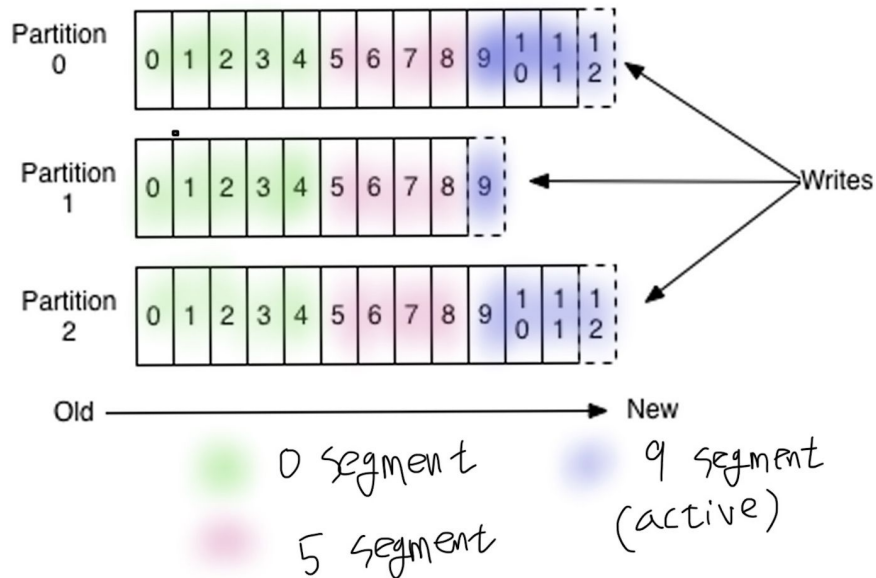
어떻게 저장되나

Topic

Partition

Segment

Anatomy of a Topic



어떻게 저장되나

OOO . INDEX

OFFSET, POSITION

0, 0

1, 3

2, 6

3, 11

OOO . LOG

OFFSET, POSITION, SIZE, PAYLOAD

0, 0, 3, ONE

1, 3, 3, TWO

2, 6, 5, THREE

3, 11, 4, FOUR

출처: <https://theboard.blog/how-kafkas-storage-internals-work-3a29b02e026>

어떻게 저장되나

000 segment

000.index

000.log

offset	position
--------	----------

0	0
---	---

1	28
---	----

offset	timestamp	size	payload
--------	-----------	------	---------

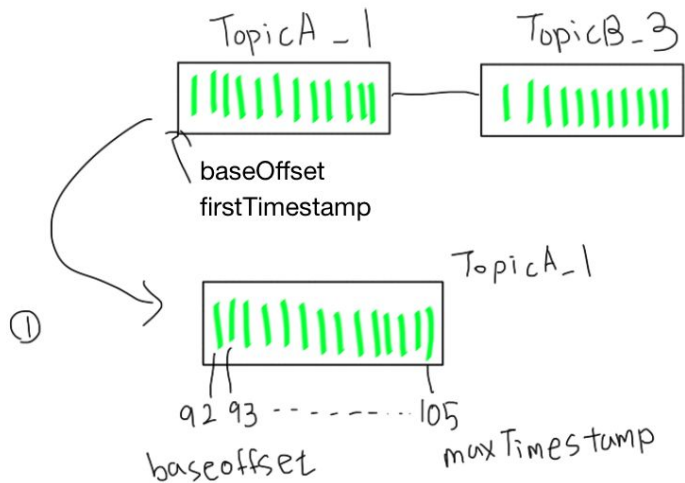
offset	timestamp	size	offset	timestamp	size	payload
--------	-----------	------	--------	-----------	------	---------

offset	timestamp	size	payload
--------	-----------	------	---------

⋮

offset	timestamp	size	payload
--------	-----------	------	---------

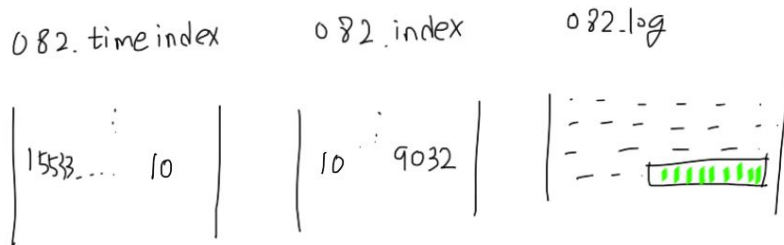
어떻게 저장되나



② Topic A, 1번 파티션 active segment 탐색

★ 이때 segment 롤링 할지 말지 결정

③ index, time index, log 기록



```
val appendedBytes = log.append(records)

if(bytesSinceLastIndexEntry > indexIntervalBytes) {
    offsetIndex.append(firstOffset, physicalPosition)
    timeIndex.maybeAppend(maxTimestampSoFar, offsetOfMaxTimestamp)
    bytesSinceLastIndexEntry = 0
}
```

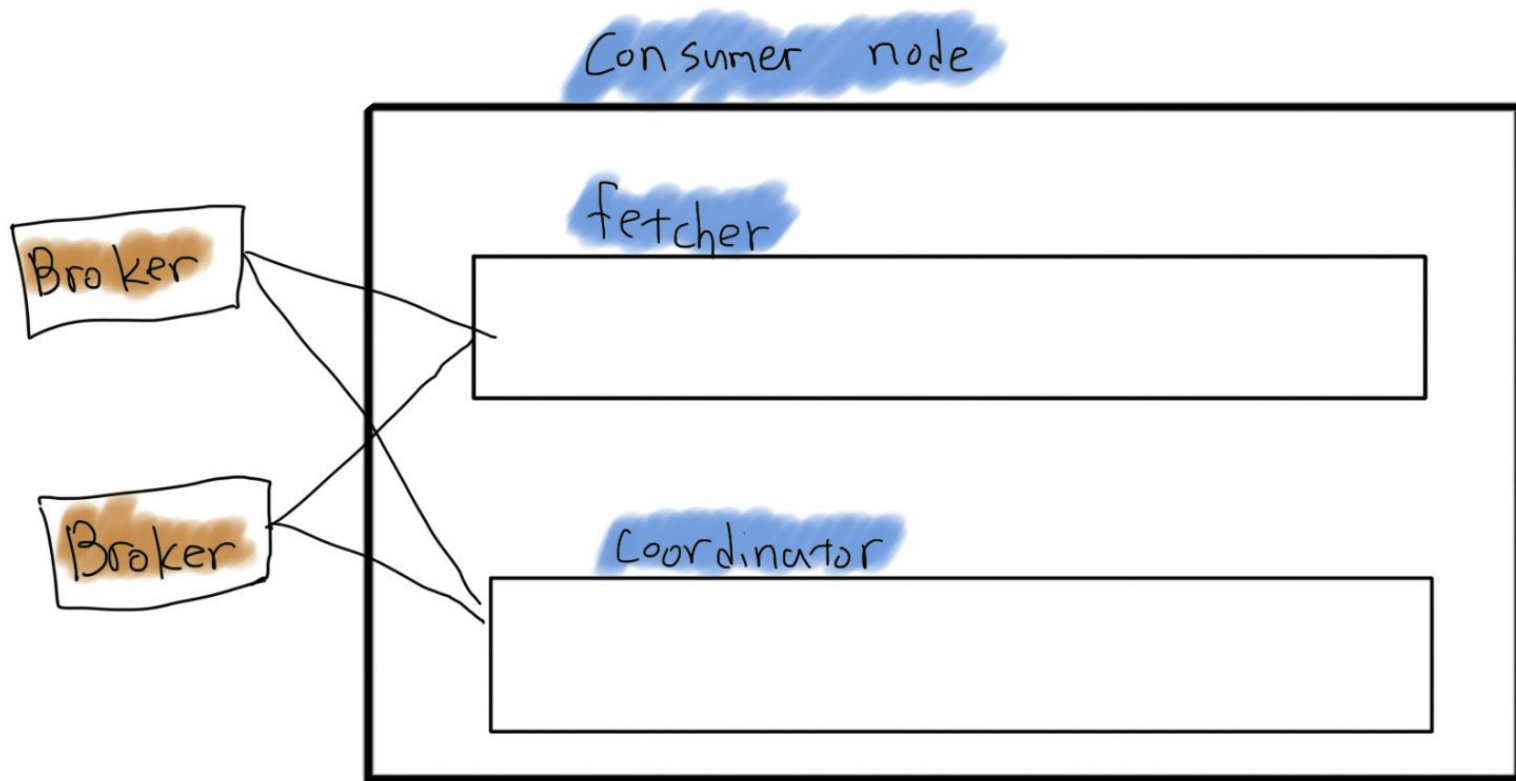
Consumer

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());
}
```

Kafka Consumer 생성



Kafka Consumer 생성

- Fetcher

poll 함수가 실행되면 적절한 크기의 records 리턴하고,
내부에 records가 없다면 Broker에게 records를 요청하고 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~

데이터 가져오기

```
// 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;

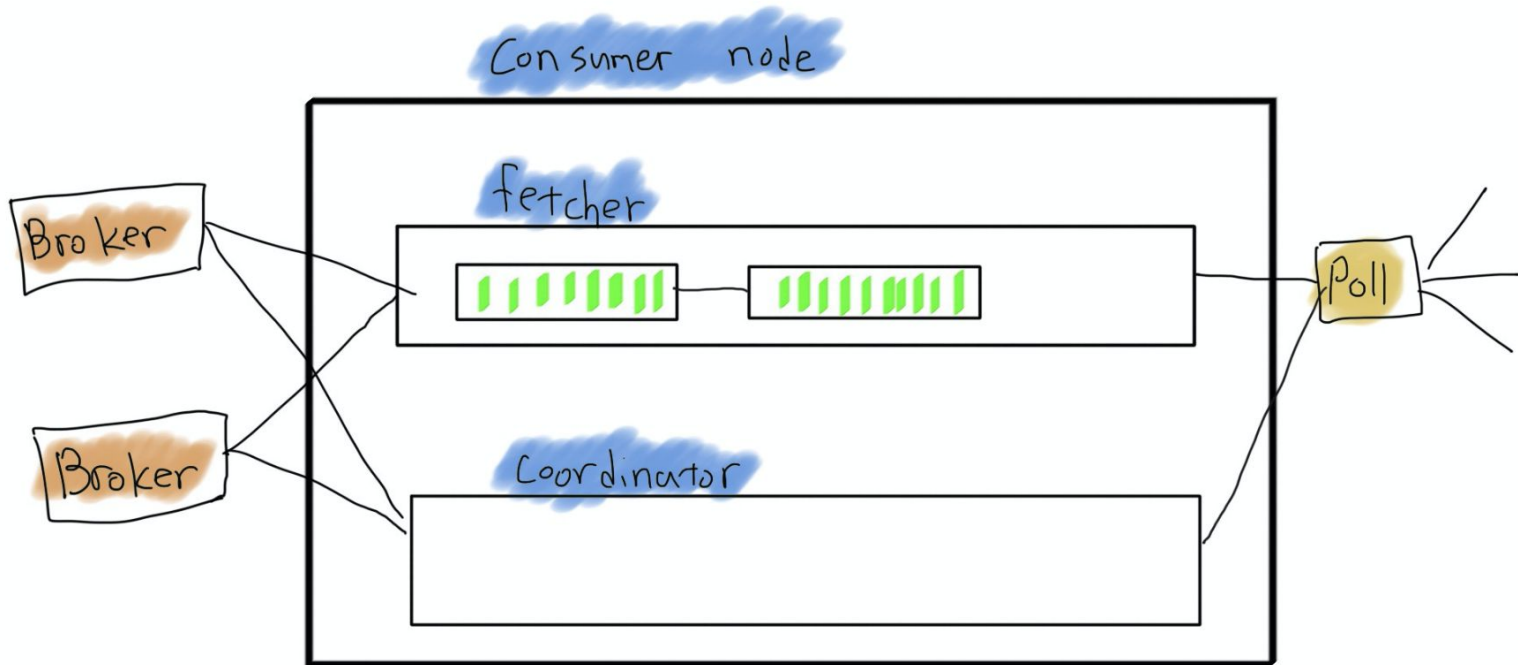
// records가 없다면 Broker에게 데이터 요청 및 내부 저장
fetcher.sendFetches();
client.poll(pollTimeout, nowMs, ...);

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

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

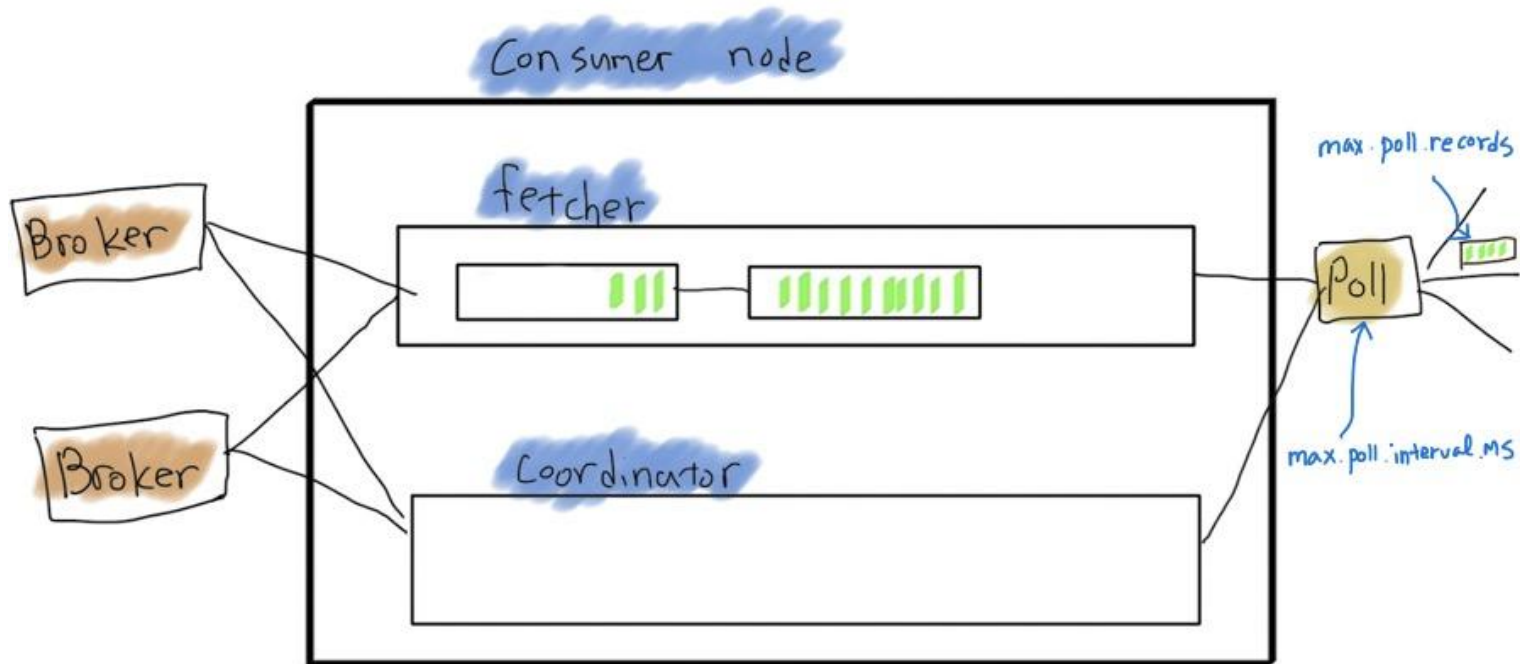
데이터 가져오기

(fetcher에 records가 있는 경우)



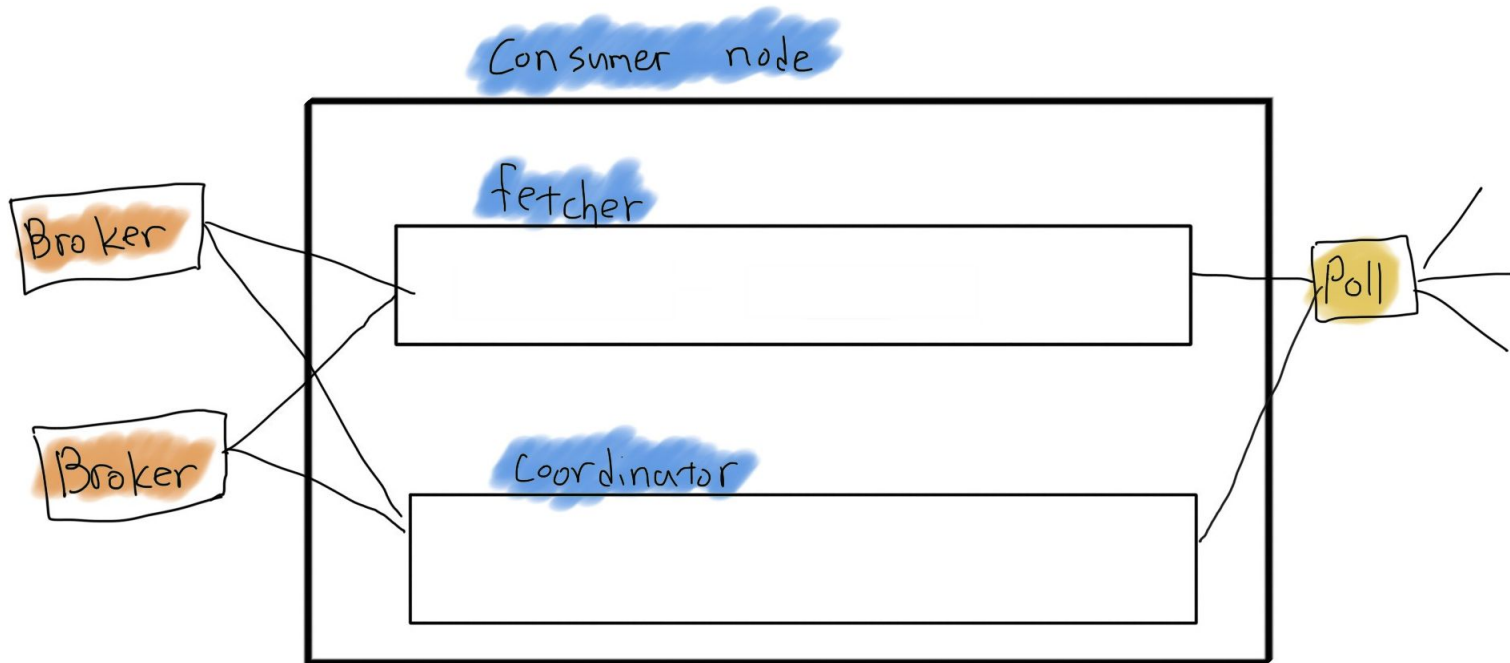
데이터 가져오기

(fetcher에 records가 있는 경우)



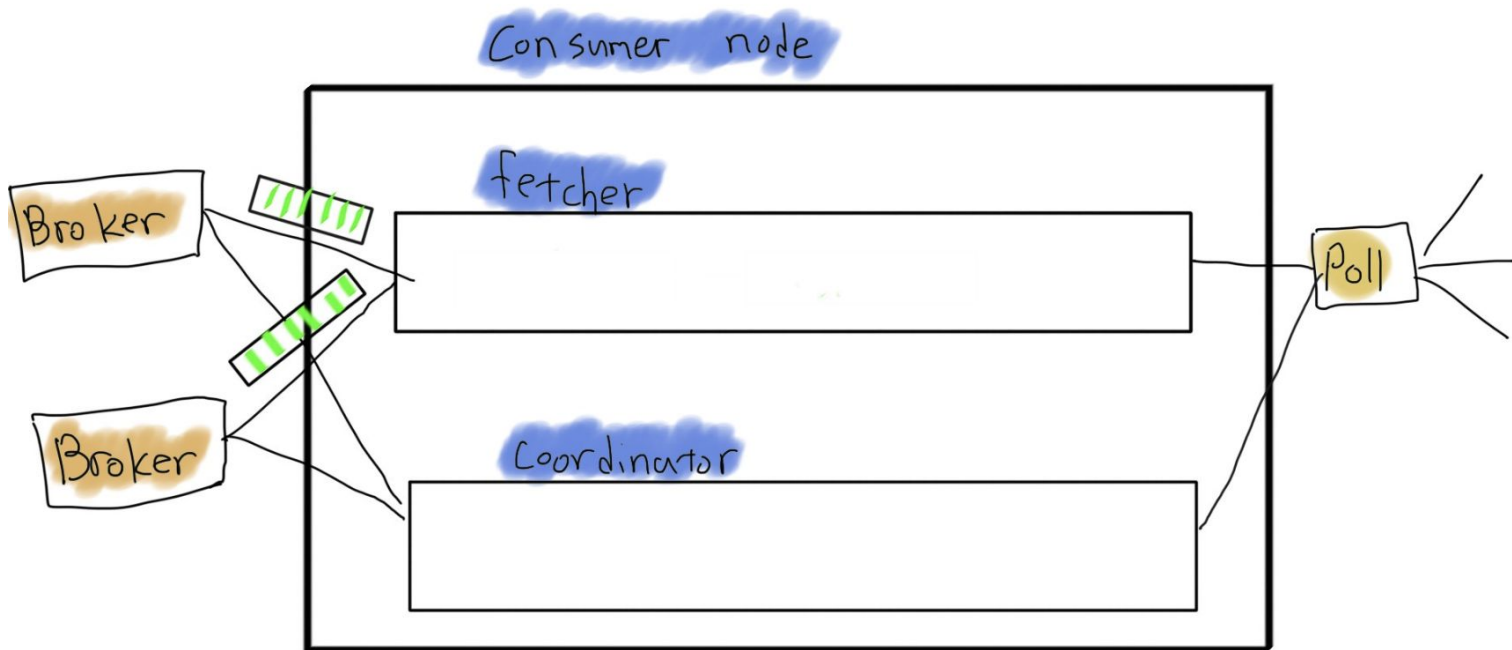
데이터 가져오기

(fetcher에 records가 없는 경우)



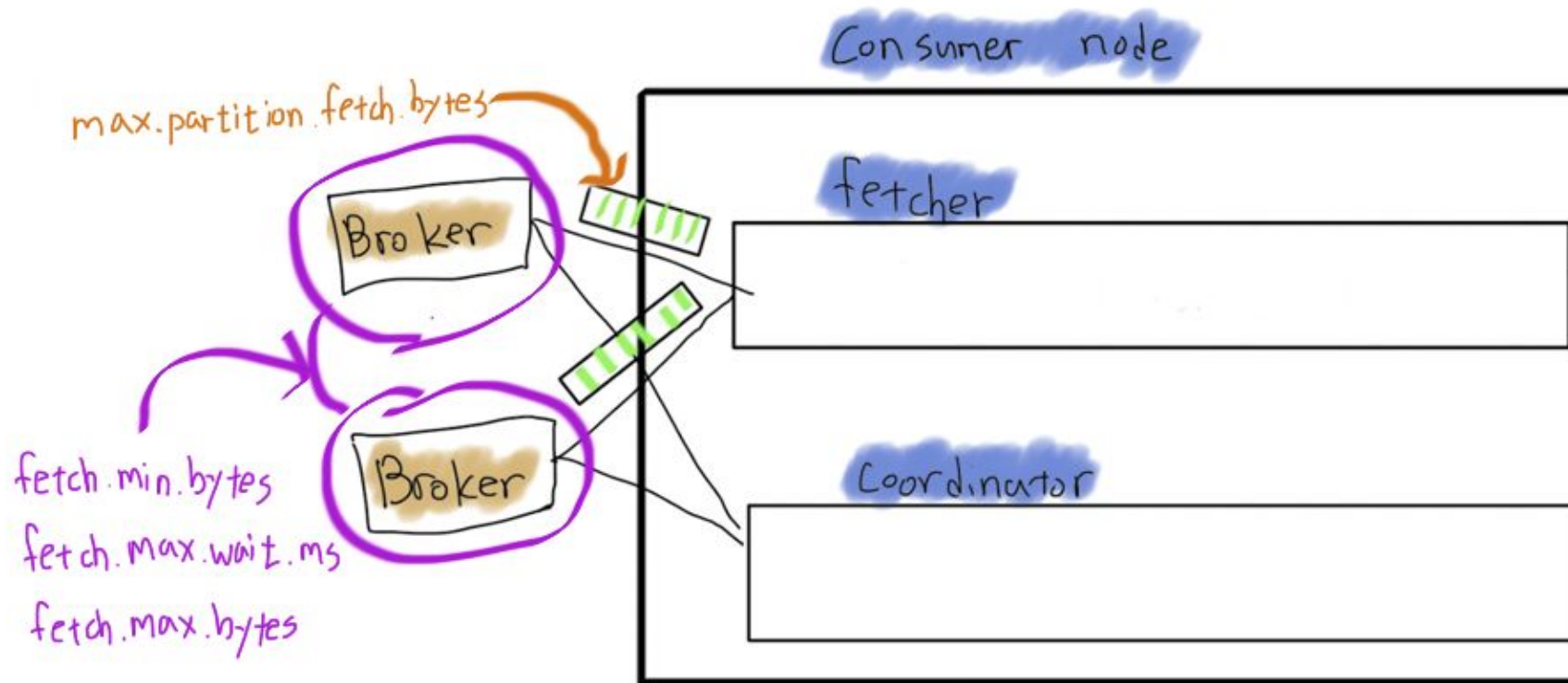
데이터 가져오기

(fetcher에 records가 없는 경우)



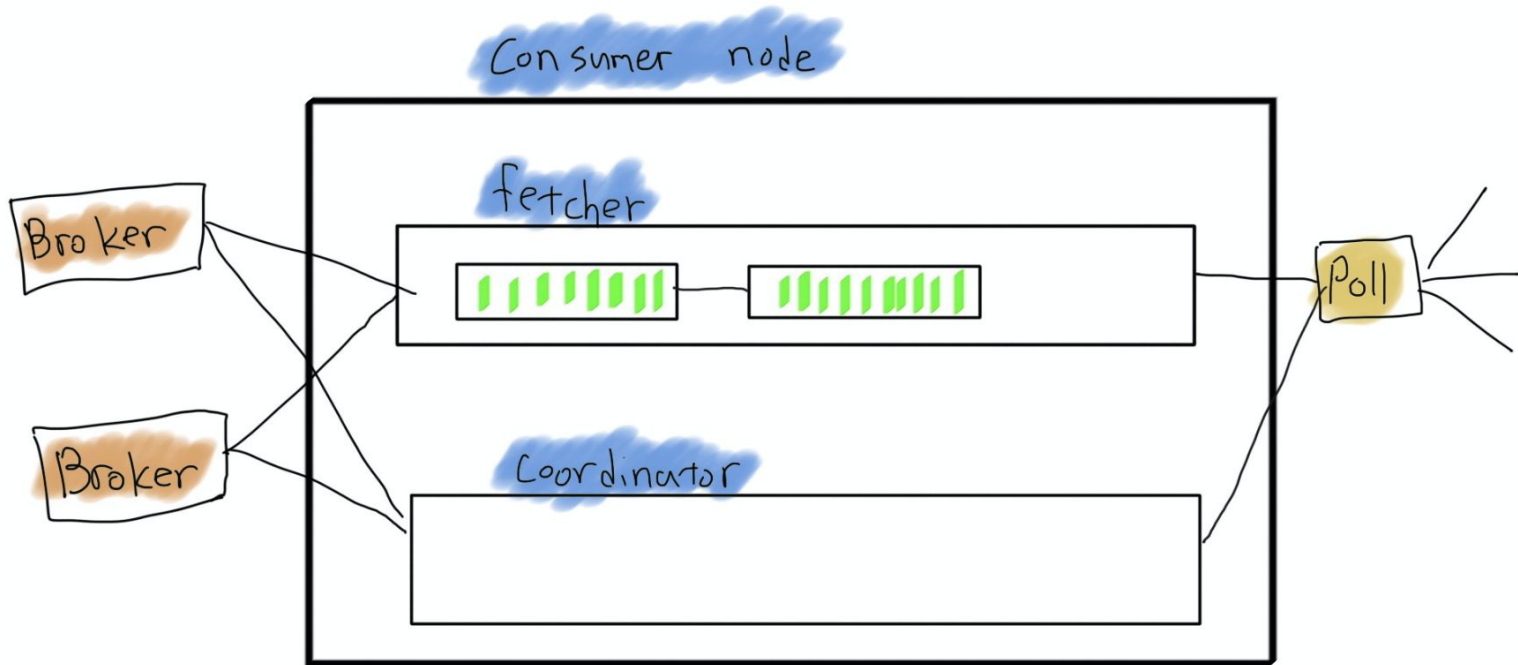
데이터 가져오기

(fetcher에 records가 없는 경우)



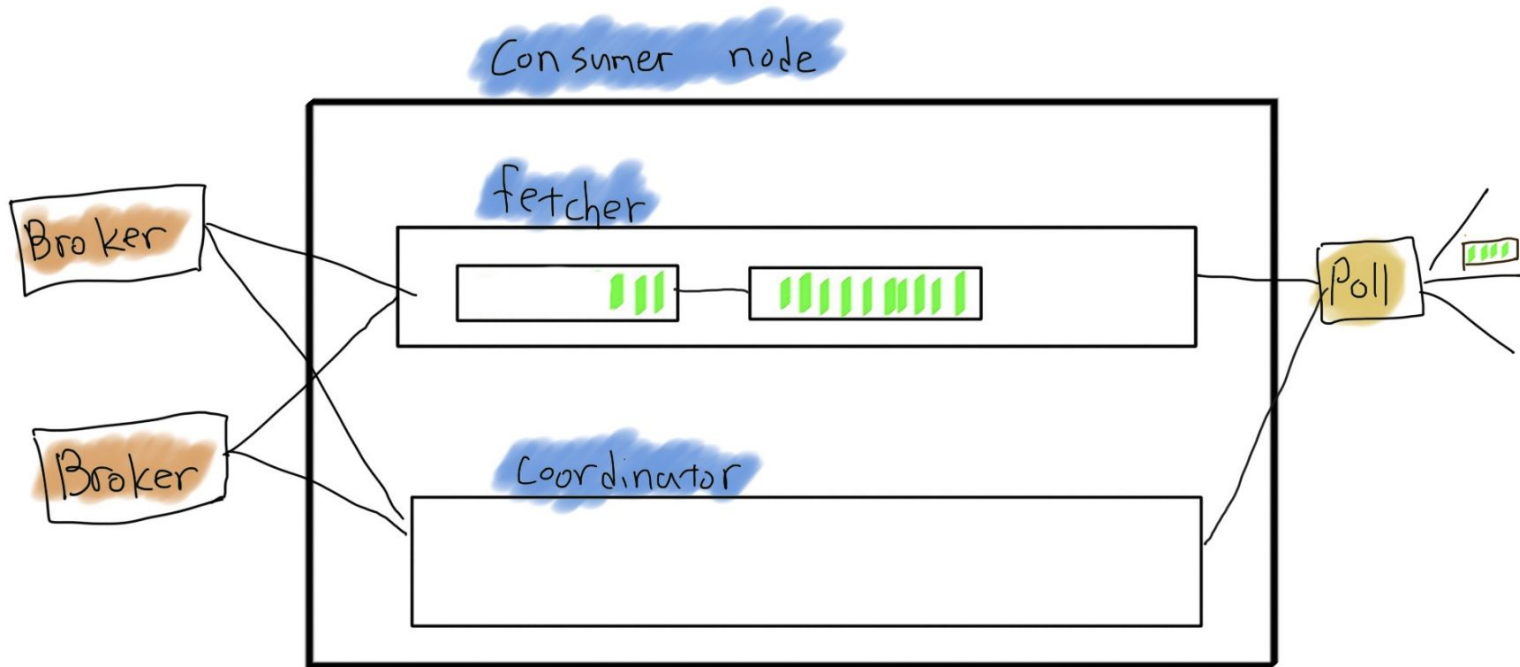
데이터 가져오기

(fetcher에 records가 없는 경우)



데이터 가져오기

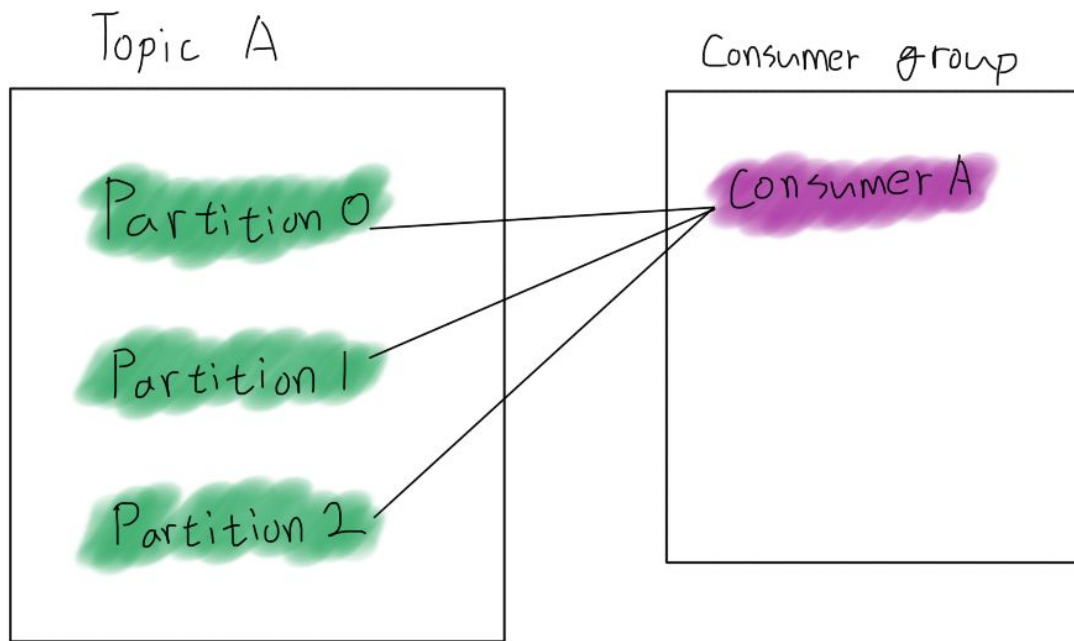
(fetcher에 records가 없는 경우)



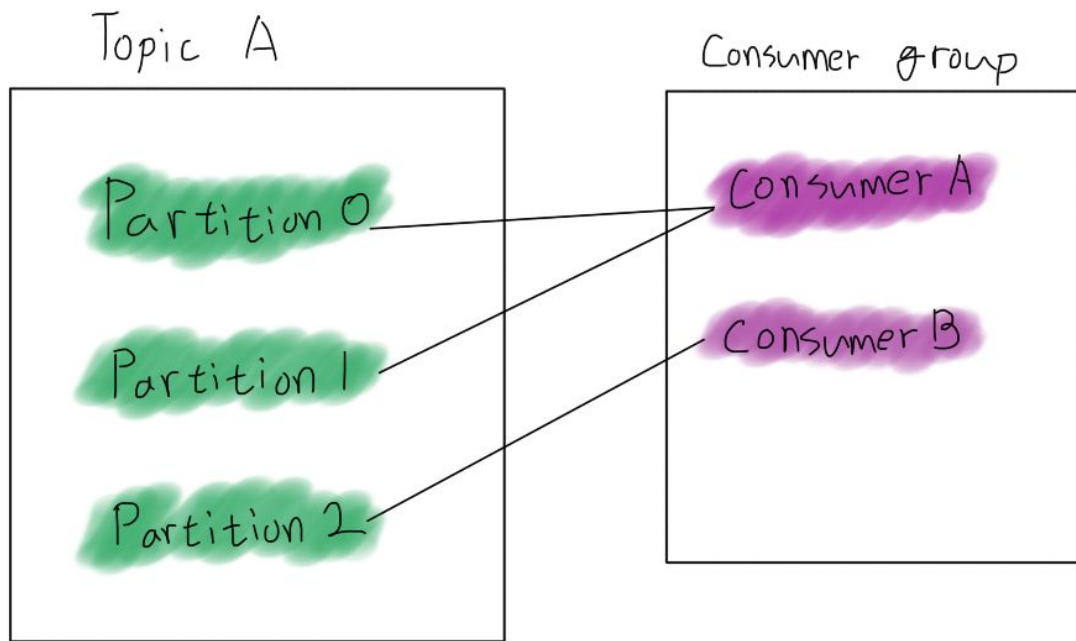
부부

Consumer Rebalance

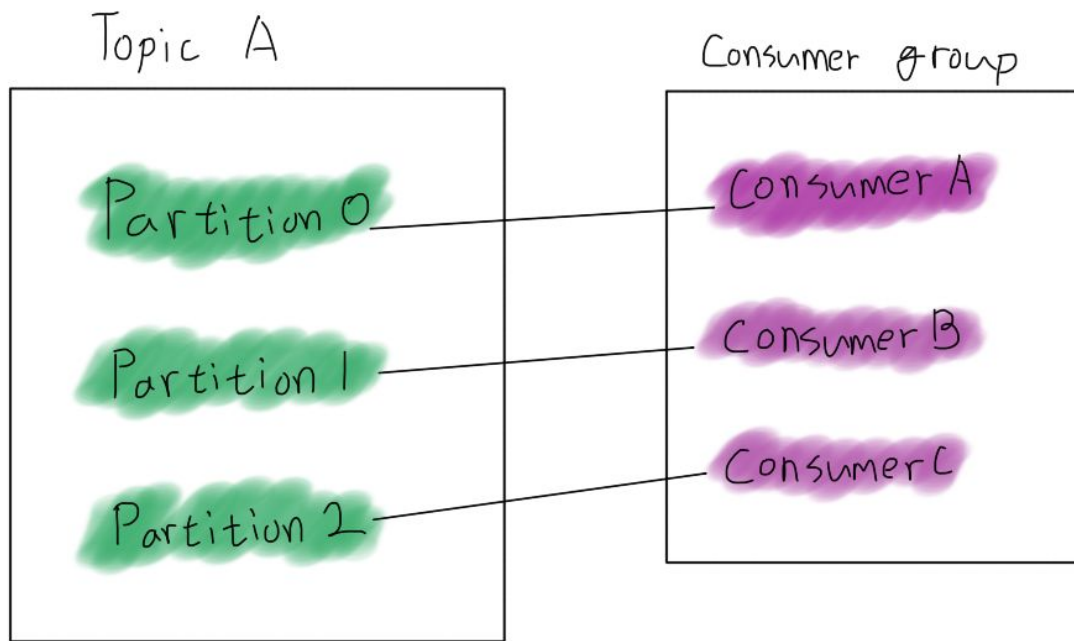
Consumer Rebalance



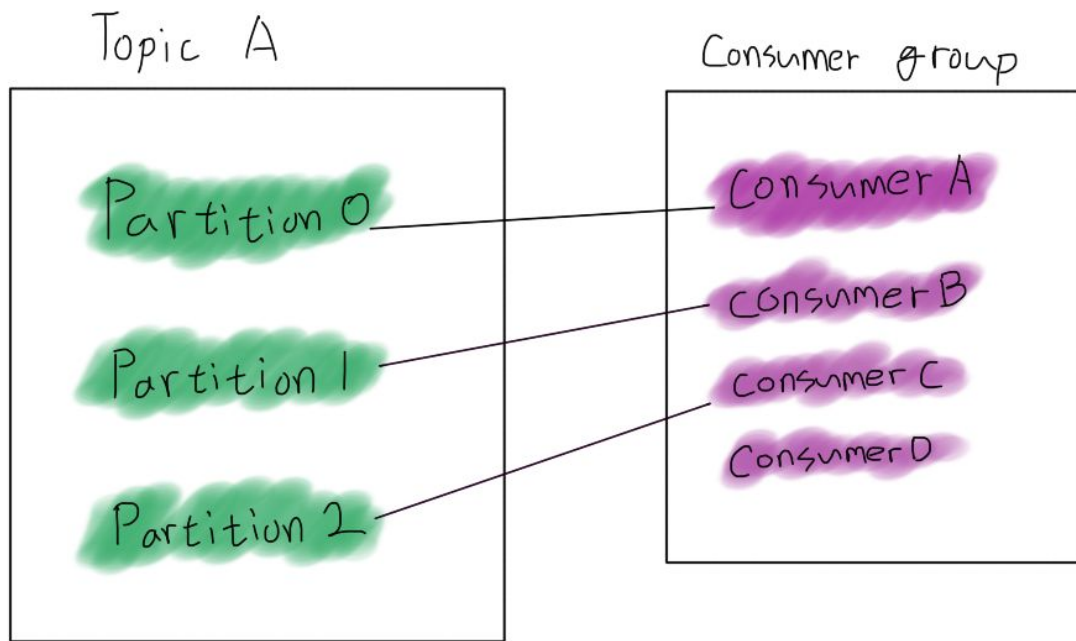
Consumer Rebalance



Consumer Rebalance



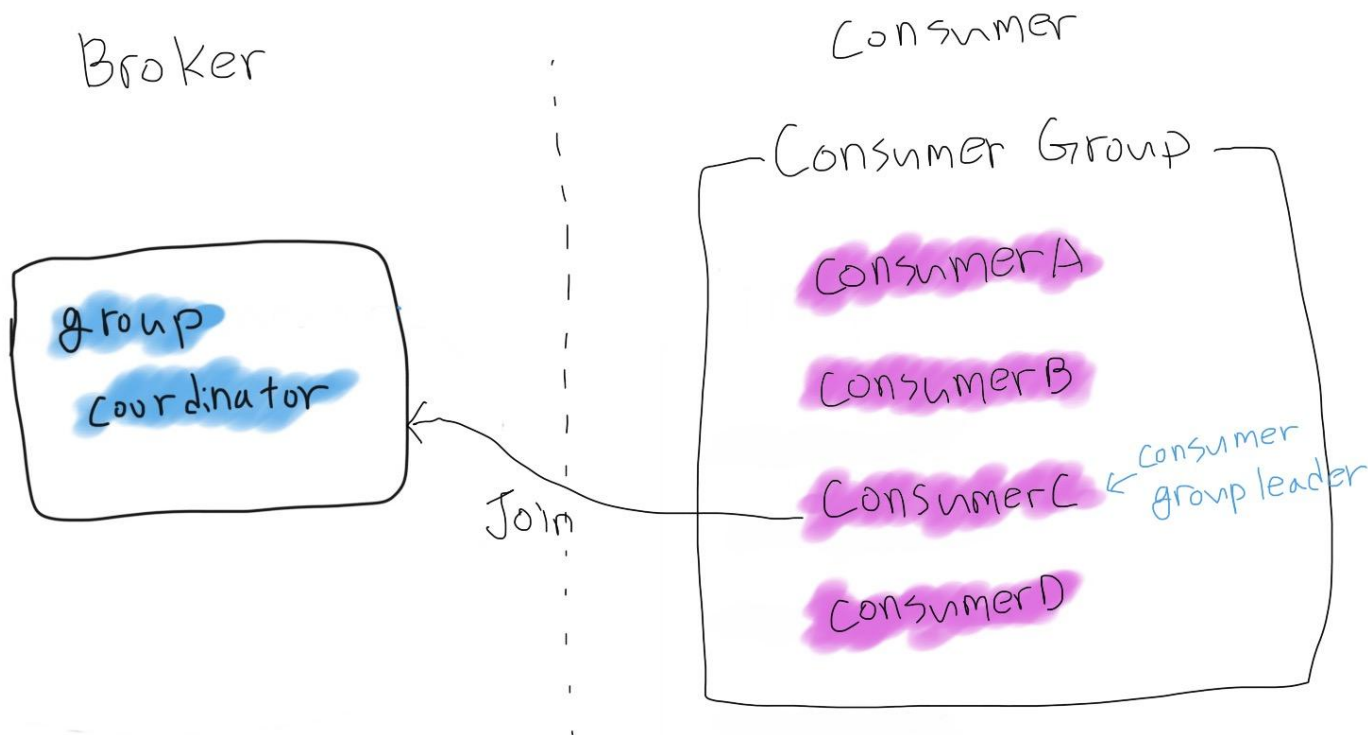
Consumer Rebalance



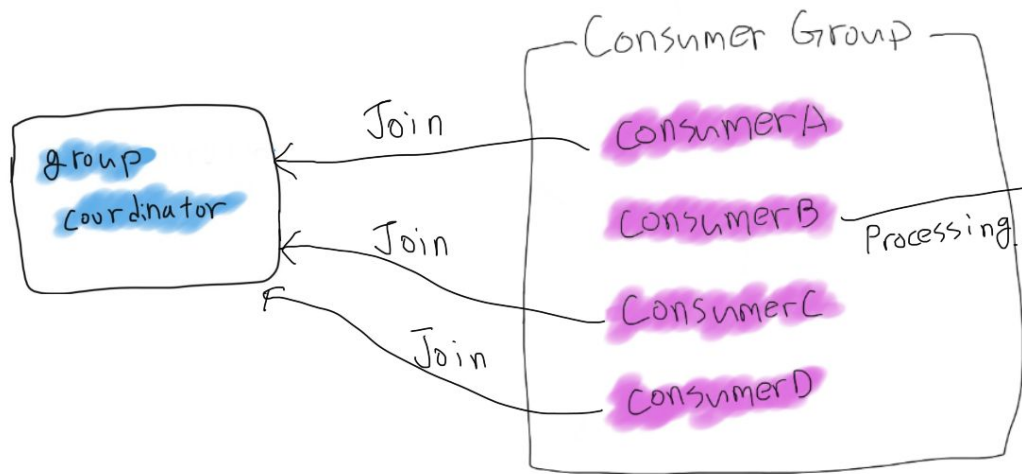
Consumer Rebalance

```
// 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

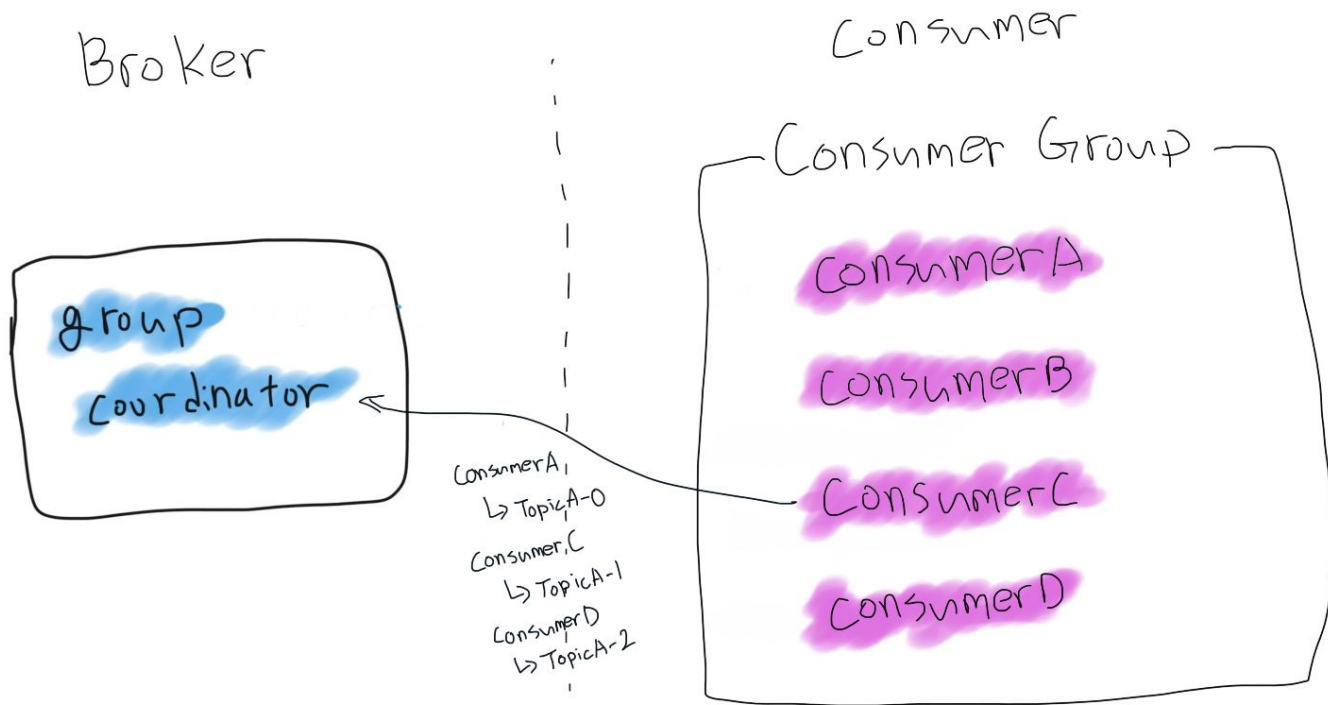


Consumer Rebalance

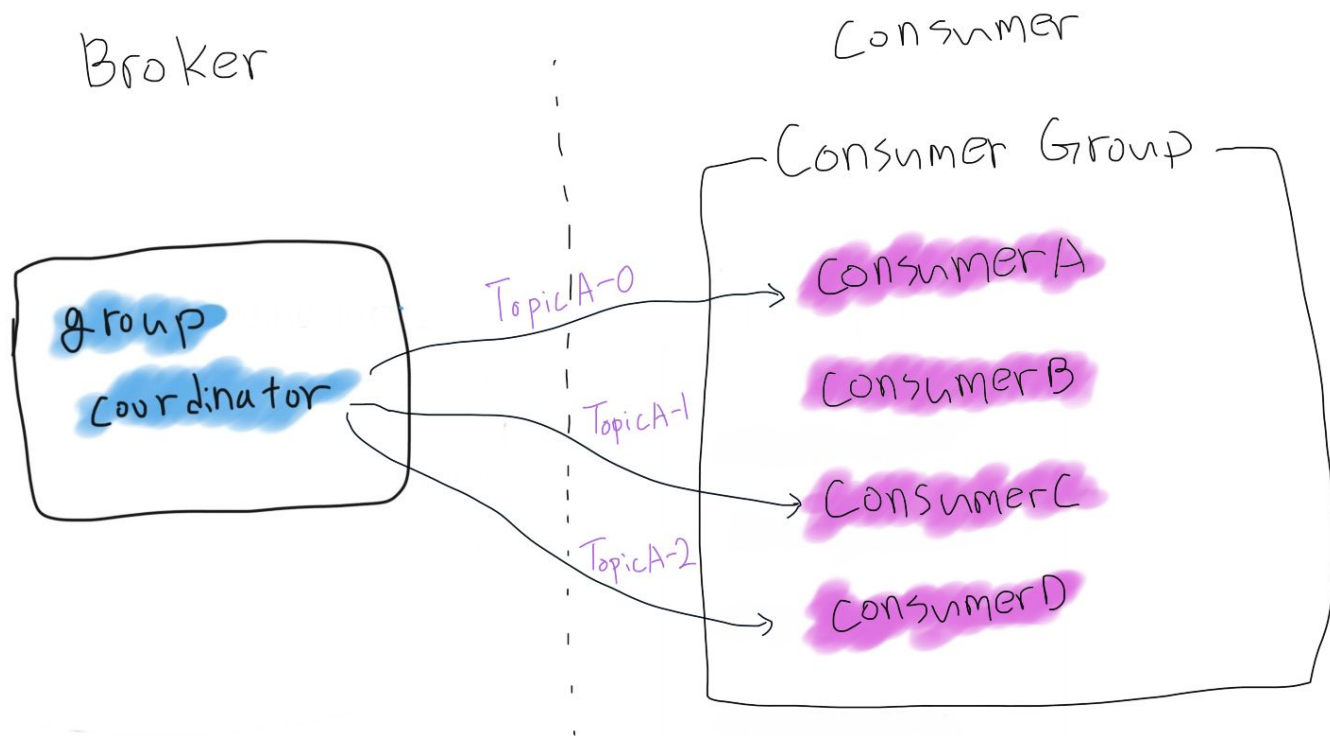


```
while (true) {  
    val records = consumer.poll(100);  
    for (record: records) {  
        // do something...  
    }  
}
```

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, ...);

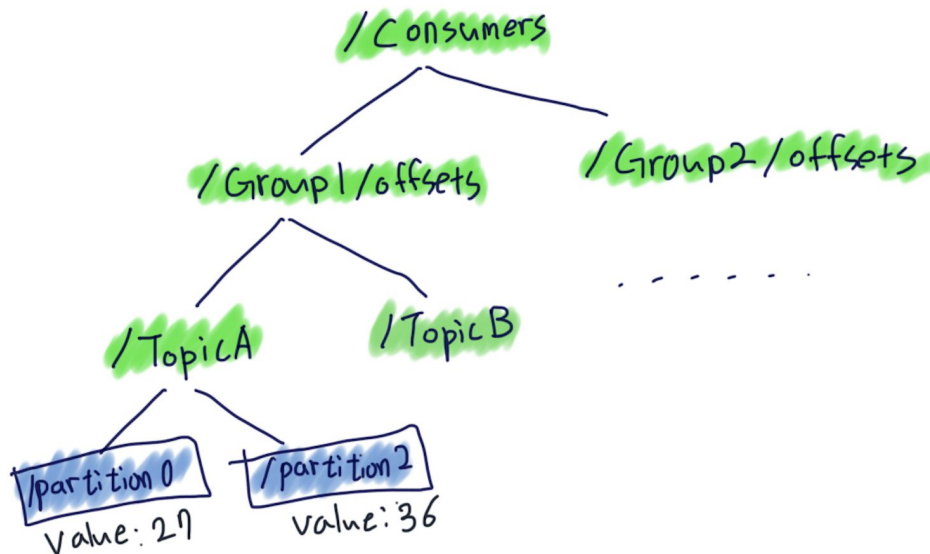
// rebalance가 필요하다면 빈 records 리턴
if (coordinator.needRejoin())
    return Collections.emptyMap();

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

Consumer Group offset

Consumer Group offset

Ver. 0.9 미만에서는 zookeeper에 consumer offset을 저장



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

Consumer Group offset

```
$> bin/kafka-consumer-groups.sh --bootstrap-server <host:port> --group  
<group> --topic <topic> --reset-offsets --to-earliest --execute
```

```
consumerGroupSummary.state match {  
  case "Empty" | "Dead" =>  
    ① val partitionsToReset = getPartitionsToReset(groupId)  
    ② val preparedOffsets = prepareOffsetsToReset(groupId, partitionsToReset)  
    ③ getConsumer.commitSync(preparedOffsets.asJava)  
  case currentState =>  
    // Error  
}
```


Consumer Group offset

Before Compaction

Offset	13	17	19	20	21	22	23	24	25	26	27	28
Keys	K1	K5	K2	K7	K8	K4	K1	K1	K1	K9	K8	K2
Values	V5	V2	V7	V1	V4	V6	V1	V2	V9	V6	V22	V25

Cleaning

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



Offset	17	20	22	25	26	27	28
Keys	K5	K7	K4	K1	K9	K8	K2
Values	V2	V1	V6	V9	V6	V22	V25

After Compaction

못다한 이야기

1. Exactly-Once(Transaction)
2. Log Cleaner and Log Compaction
3. Purgatory
4. Controller in Broker and Leader Election
5. Metrics
6. Kafka Connect

감사합니다.