if(kakao)2021

스마트 메시지 서비스 개발기

설계부터 카프카 스트림즈 활용까지

우영화 Dane.W, 최원영 Cory.Doras
카카오

스마트 메시지 소개

서비스 설계와 개발

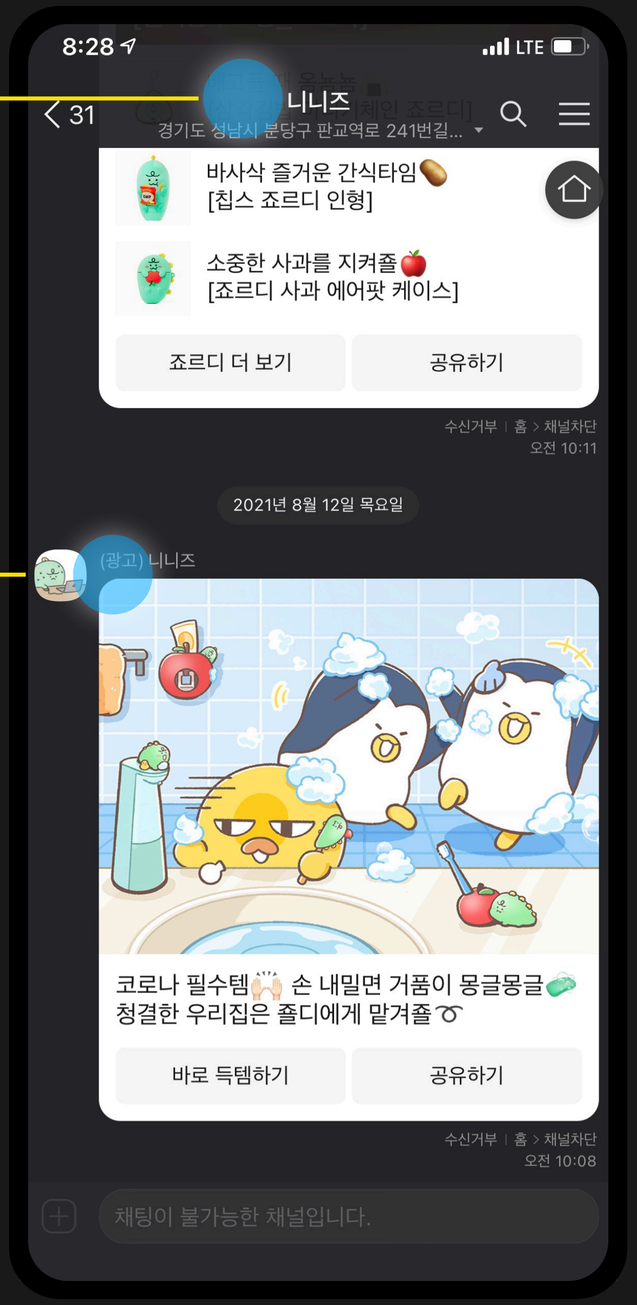
카프카 스트림즈를 선택한 이유

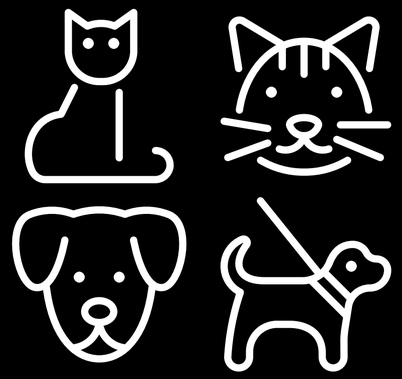
스마트 메시지에 카프카 스트림즈 적용

스마트 메세지 소개

채널

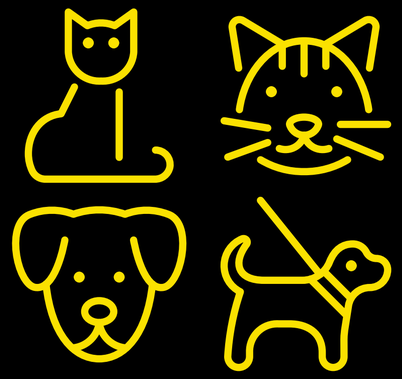
광고메세지


8:28
LTE
31
니니즈
경기도 성남시 분당구 판교역로 241번길...
바사삭 즐거운 간식타임
[칩스 죠르디 인형]
소중한 사과를 지켜죨
[죠르디 사과 에어팟 케이스]
죠르디 더 보기
공유하기
수신거부 | 홈 > 채널차단
오전 10:11
2021년 8월 12일 목요일
(광고) 니니즈
코로나 필수템 손 내밀면 거품이 몽글몽글
청결한 우리집은 죨디에게 맡겨죨
바로 득템하기
공유하기
수신거부 | 홈 > 채널차단
오전 10:08
채팅이 불가능한 채널입니다.

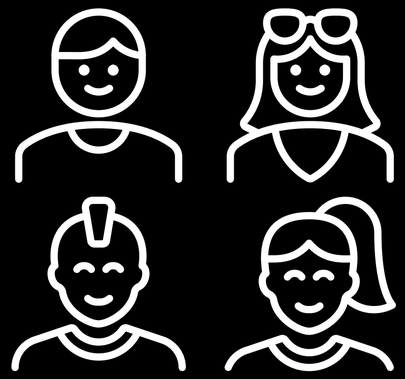


소재최적화

유저타게팅

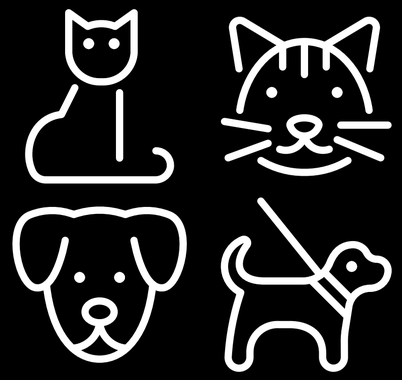


소재최적화



유저타게팅

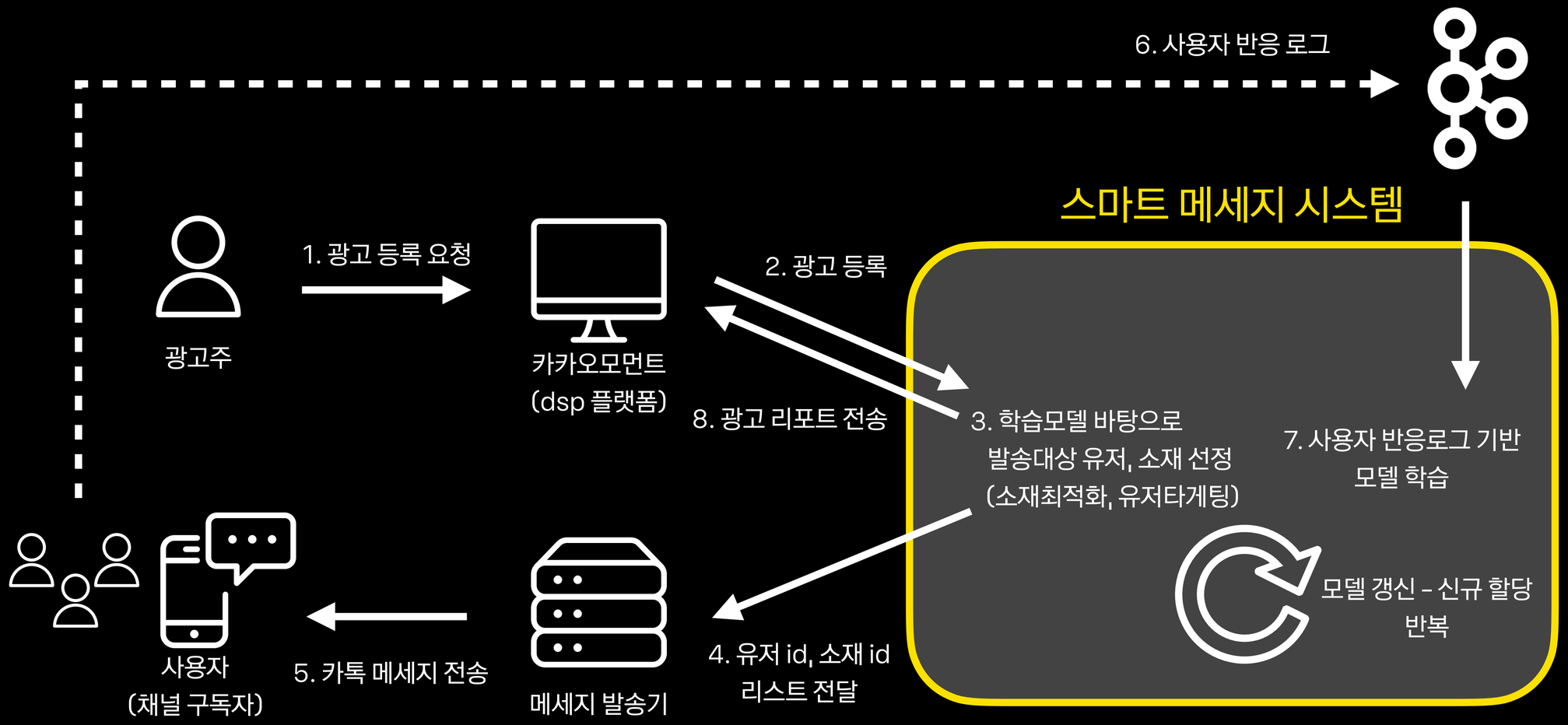
뭘 좋아할 지 몰라 광고 소재(시안)을 여러 개 만들었는데,
어떤 소재가 가장 광고 효과가 좋을까?



소재최적화

유저타게팅

내 채널 구독자는 30만 명인데, 광고 예산은 10만 명분 어치..
누구한테 보내야 반응을 잘 해줄까?


6. 사용자 반응 로그
스마트 메세지 시스템
광고주
1. 광고 등록 요청
카카오모먼트
(dsp 플랫폼)
2. 광고 등록
8. 광고 리포트 전송
3. 학습모델 바탕으로
발송대상 유저, 소재 선정
(소재최적화, 유저타게팅)
7. 사용자 반응로그 기반
모델 학습
모델 갱신 - 신규 할당
반복
사용자
(채널 구독자)
5. 카톡 메세지 전송
메세지 발송기
4. 유저 id, 소재 id
리스트 전달

서비스 설계와 개발

프로젝트 탄생 배경

Legacy 시스템 운영의 한계

- 운영 인력 부족
- 운영 담당 인력 중 서비스 개발에 참여했던 사람 부재

  → 보수적인 코드 수정, 소극적 이슈 대응

- 소재 최적화와 유저 타게팅이 별도 기능으로 분리되어 동시 사용 불가능

  → 데이터 컬렉션도 두벌, 서버 처리 로직 코드도 두벌, ...

프로젝트 탄생 배경

때마침..

- 인력 충원 : 데이터 처리 전담 파트 결성
- 소재 최적화와 유저 타게팅의 통합 모델 아이디어

V1 → V2 전환 결정

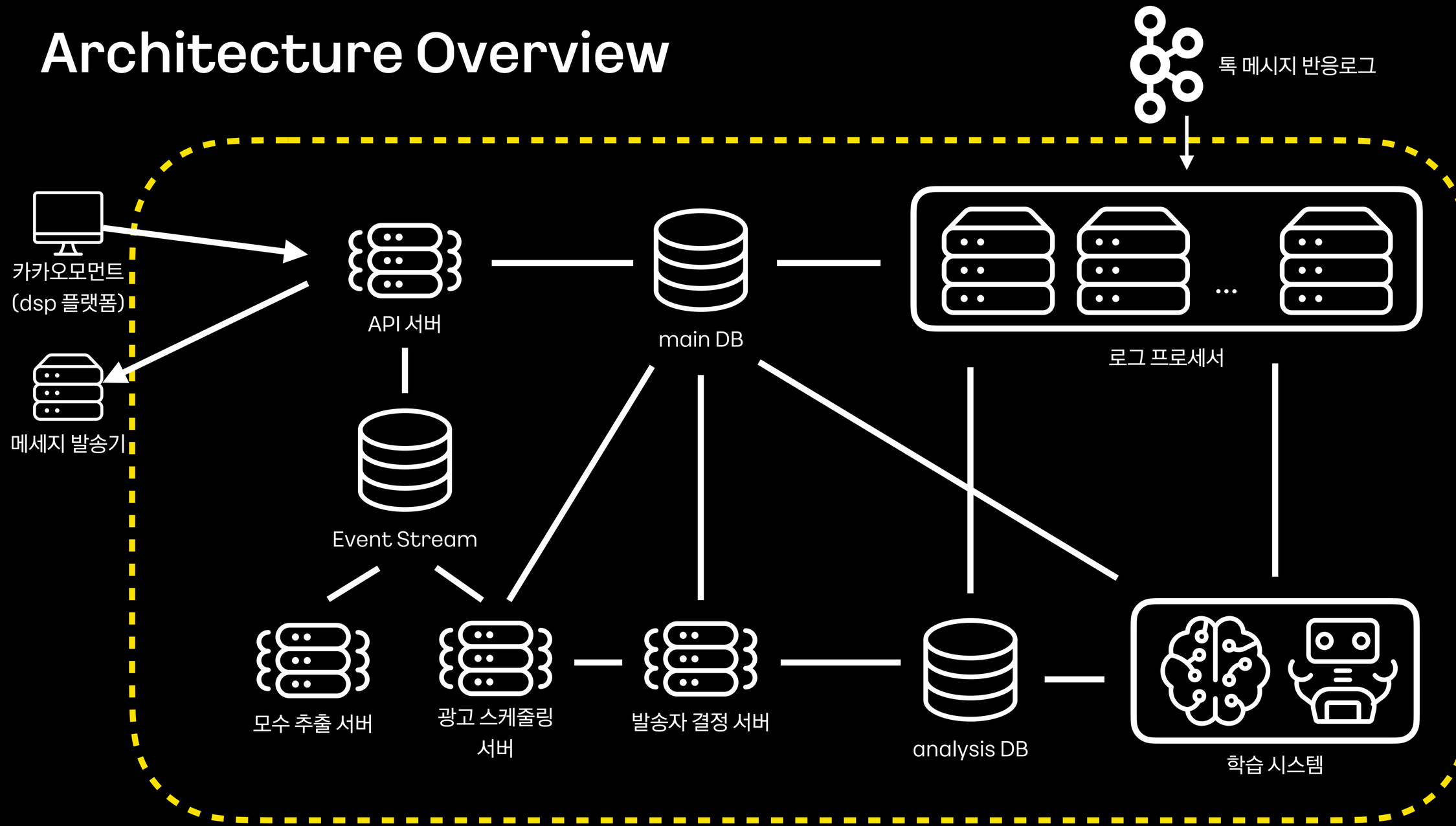
Project Goal

1. 6인의 개발자, 6개월의 개발 기간

2. 클러스터 환경에 최적화된 MSA 시스템

3. 운영 편의성을 최대한 고려한 시스템

Architecture Overview



주요 시스템 요소

Golang-based Server

Event-driven Architecture

Job Scheduling with Delay Queue

Kafka Streams for Data Pipeline

1. Golang-based Server

Go : MSA (Micro Service Architecture) 시대의 언어

- 기능별로 서버를 분리: 무겁고 복잡한 프레임워크에 대한 필요성 감소
- 필요한 라이브러리만 조합하여 가벼우면서 빠른 성능 보장
- 효율적인 CI/CD: 빠른 빌드, 쉬운 의존성 관리
- 높은 생산성: 단순한 문법과 낮은 난이도로 인한 빠른 개발속도, 코드 리뷰 용이
- 라이브러리 선택지가 많다는 건 장점이자 단점..

  웹 프레임워크 - echo, gin 사용

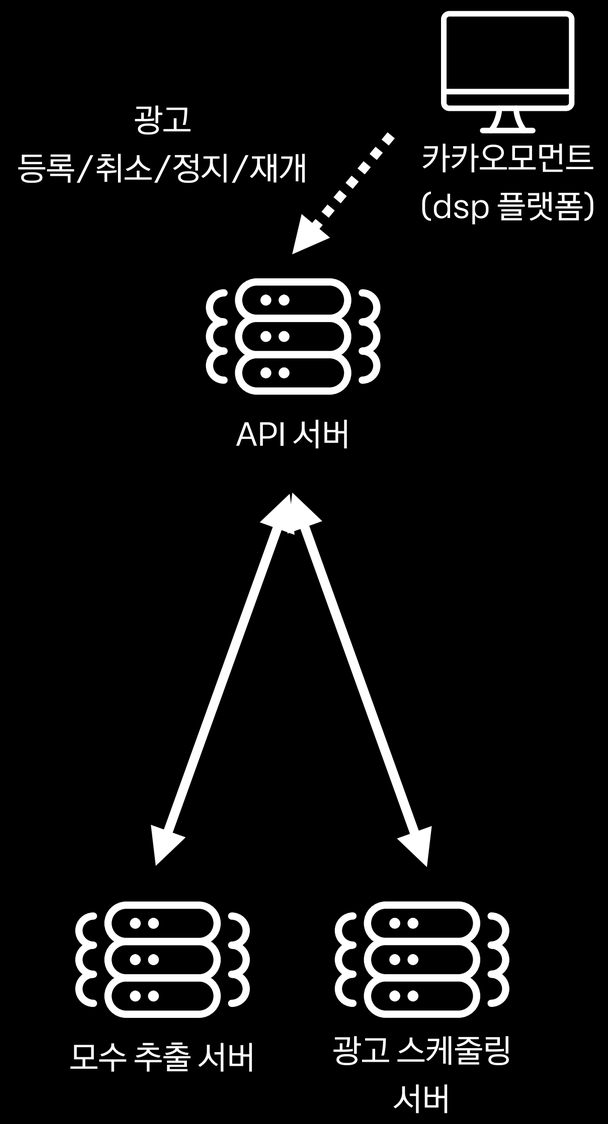
Legacy Spring → Go 전환 효과

- 서버 코드 LOC 60% 감소
- API 서버 기준 pod 메모리 사용량 1/3 수준으로 감소

2. Event-driven Architecture

설계 초안

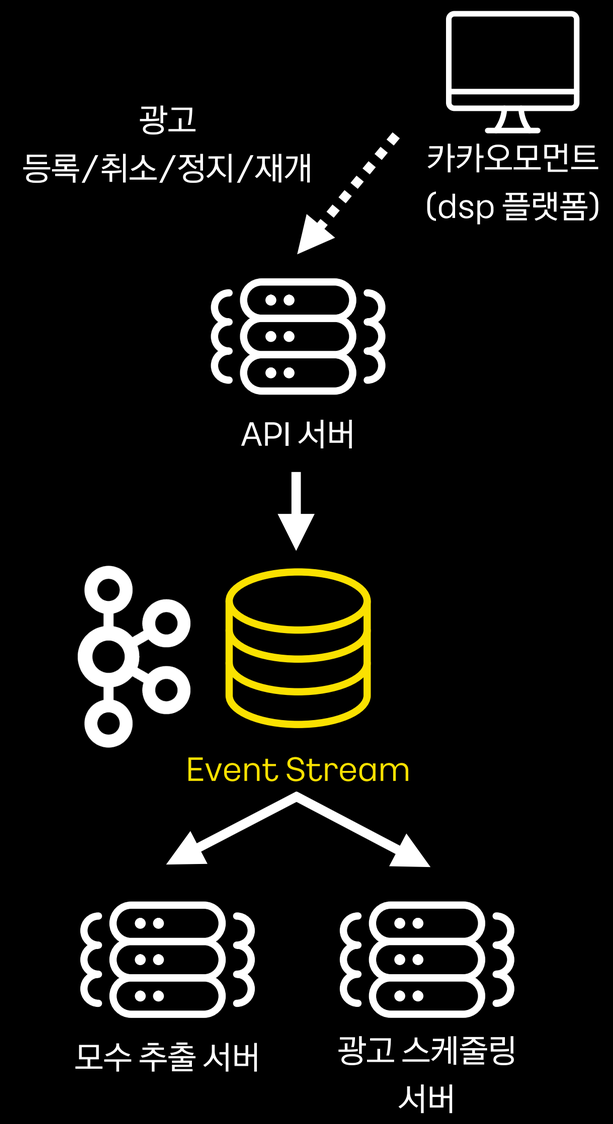
- API서버와 광고 집행 서버들이 HTTP 통신
- 광고 집행 관련정보 (광고 생성 / 일시정지 / 취소) 는 민감정보
- 유실/중복처리의 가능성 존재



2. Event-driven Architecture

Kafka 기반 Event Sourcing Architecture 로 전환

- Event 를 Server Pod 장애로부터 안전하게 저장
  → 유실 방지
- Kafka Transaction 사용
  → API서버 처리도중 이슈 발생하더라도 중복 캠페인 처리 방지
- CQRS (Command and Query Responsibility Segregation)
  → API 서버는 main DB 조회만 수행, 광고 스케줄링 서버가 DB 수정
  → API 서버 개발자와 광고 집행 서버 개발자 간 업무 coupling 제거
  → Event handling 모듈 추가가 필요한 경우 유연한 확장 가능
- REST 기반 CRUD 방식에 비해 event 처리 확인에 걸리는 시간 증가
  → Latency에 민감한 API 가 아니며 event 트래픽이 높지 않음



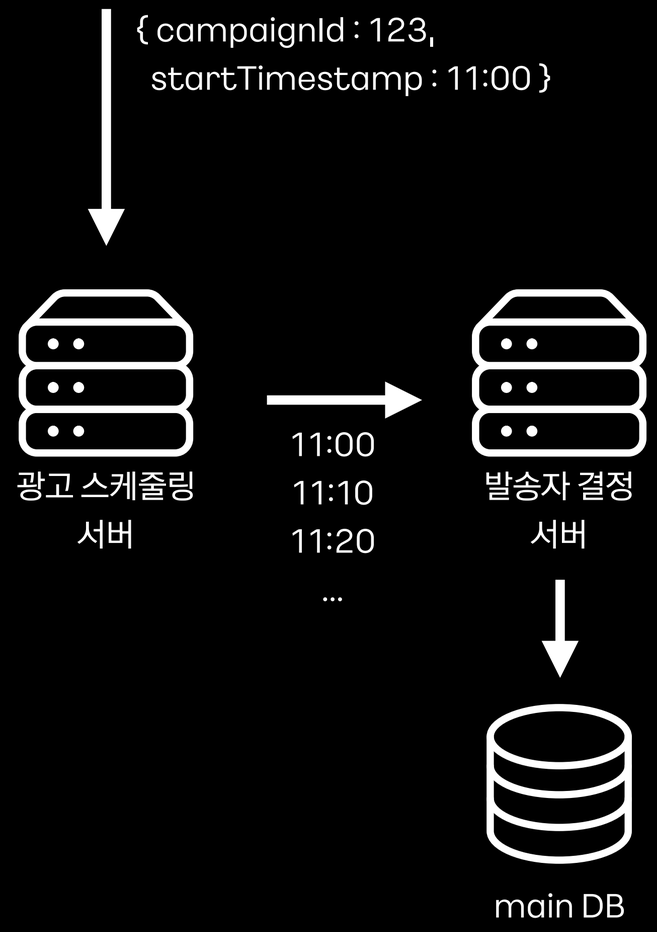
3. Delay Queue 를 이용한 job scheduling

광고 스케줄링 로직 동작 방식

- 광고가 mini-batch 형태로 진행됨

  → 매 배치마다 갱신된 모델을 사용하기 때문

- 광고 집행 시작 시점과, 실제 job 발생 시점 사이 interval 존재
- 미리 설정된 시간에 맞추어 발송자 결정 서버에 job을 요청



3. Delay Queue 를 이용한 job scheduling

예정된 작업이 남아있는 상태에서 서버 장애가 발생한다면?

- Kubernetes가 서버는 새로 띄워준다

문제는 스케줄 되어있던 job 정보들..

- 메모리에만 저장하는 경우

  → 장애로 서버 종료시 유실

- main DB 에 보관한다면?

  → 서버 pod가 여러개 떠있을 때 중복 처리 방지 필요

3. Delay Queue 를 이용한 job scheduling

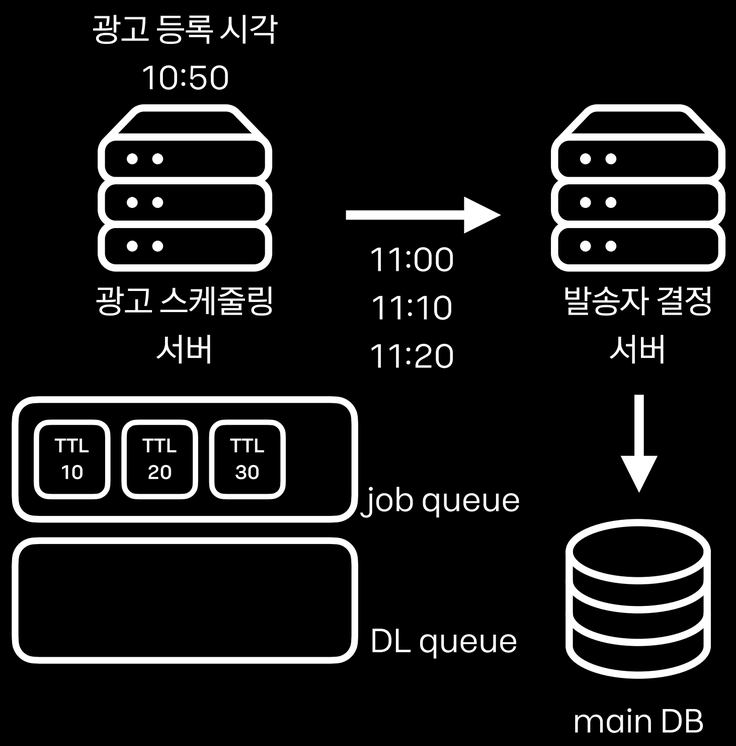
RabbitMQ 기반 delay queue

- Job 스케줄 정보를 프로세스 메모리 외부에 안전하게 저장
- Delay Queue: Message TTL + Dead Letter Exchanges

  Message 에 TTL 설정 후 queue에 넣으면,

  설정한 TTL이 지난 후 DLX 로 이동한다
- DLX를 별도의 queue 에 bind

  → 일정 지연 시간 후 메세지가 도달하는 delay queue



3. Delay Queue 를 이용한 job scheduling

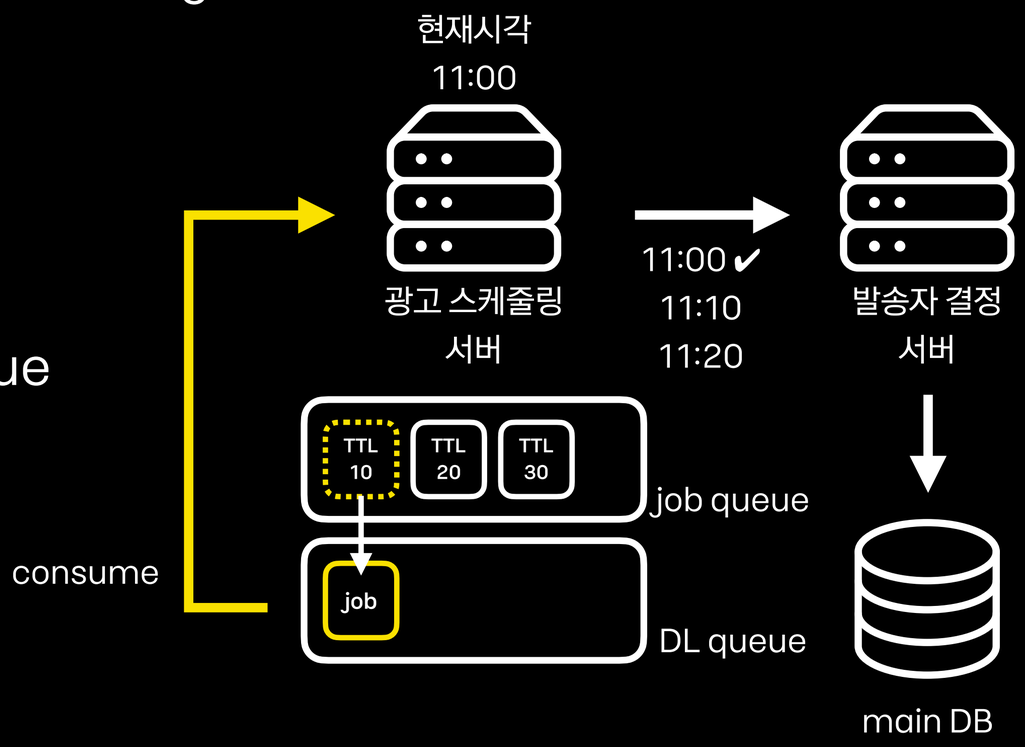
RabbitMQ 기반 delay queue

- Job 스케줄 정보를 프로세스 메모리 외부에 안전하게 저장
- Delay Queue: Message TTL + Dead Letter Exchanges

  Message 에 TTL 설정 후 queue에 넣으면,

  설정한 TTL이 지난 후 DLX 로 이동한다
- DLX를 별도의 queue 에 bind

  → 일정 지연 시간 후 메세지가 도달하는 delay queue



3. Delay Queue 를 이용한 job scheduling

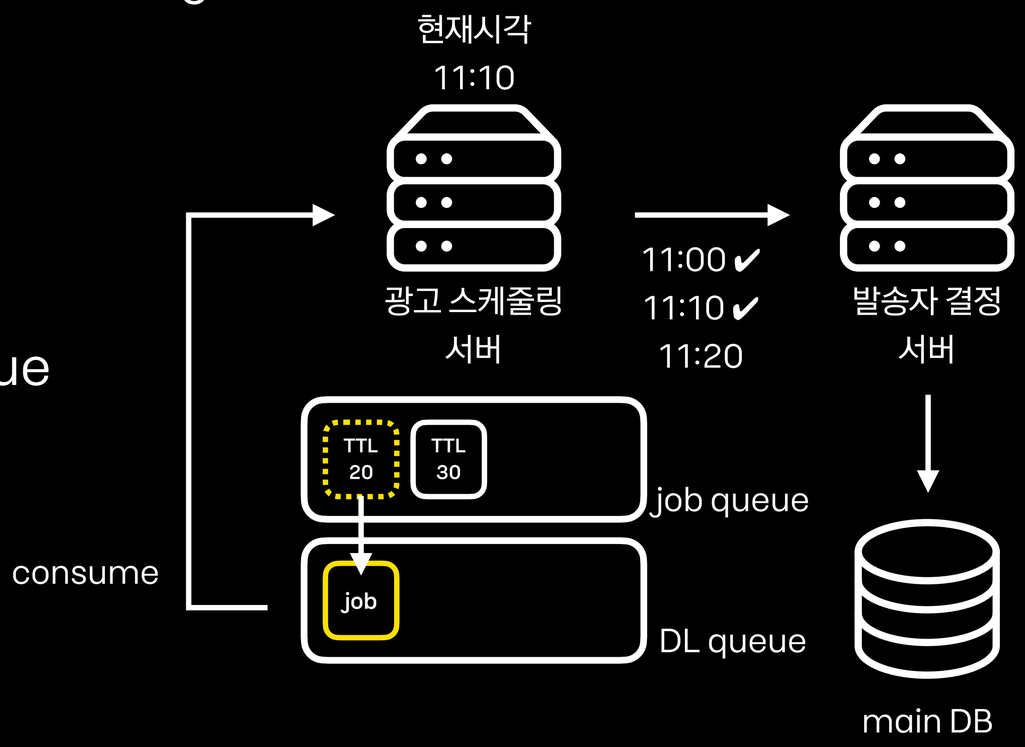
RabbitMQ 기반 delay queue

- Job 스케줄 정보를 프로세스 메모리 외부에 안전하게 저장
- Delay Queue: Message TTL + Dead Letter Exchanges

  Message 에 TTL 설정 후 queue에 넣으면,

  설정한 TTL이 지난 후 DLX 로 이동한다
- DLX를 별도의 queue 에 bind

  → 일정 지연 시간 후 메세지가 도달하는 delay queue



3. Delay Queue 를 이용한 job scheduling

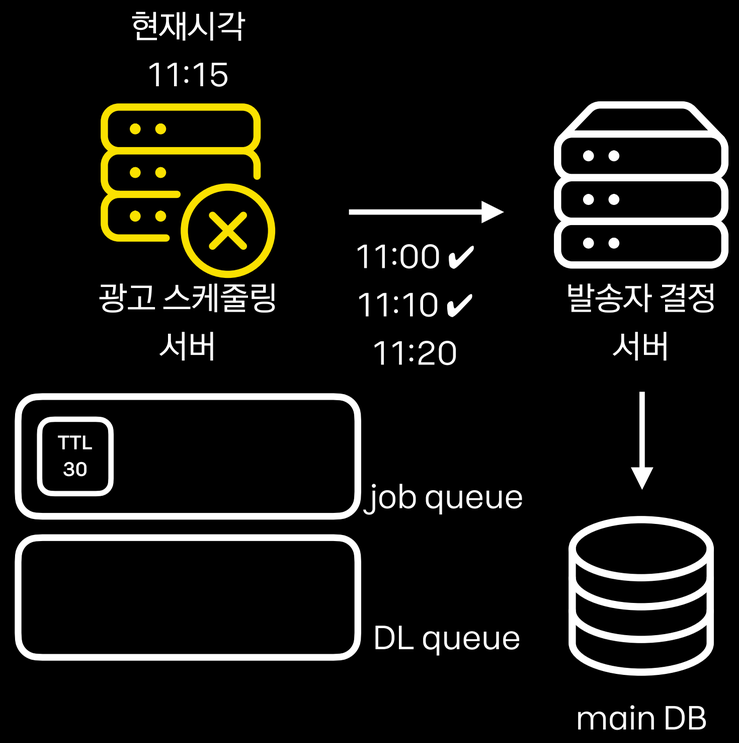
RabbitMQ 기반 delay queue

- Job 스케줄 정보를 프로세스 메모리 외부에 안전하게 저장
- Delay Queue: Message TTL + Dead Letter Exchanges

  Message 에 TTL 설정 후 queue에 넣으면,

  설정한 TTL이 지난 후 DLX 로 이동한다
- DLX를 별도의 queue 에 bind

  → 일정 지연 시간 후 메세지가 도달하는 delay queue



3. Delay Queue 를 이용한 job scheduling

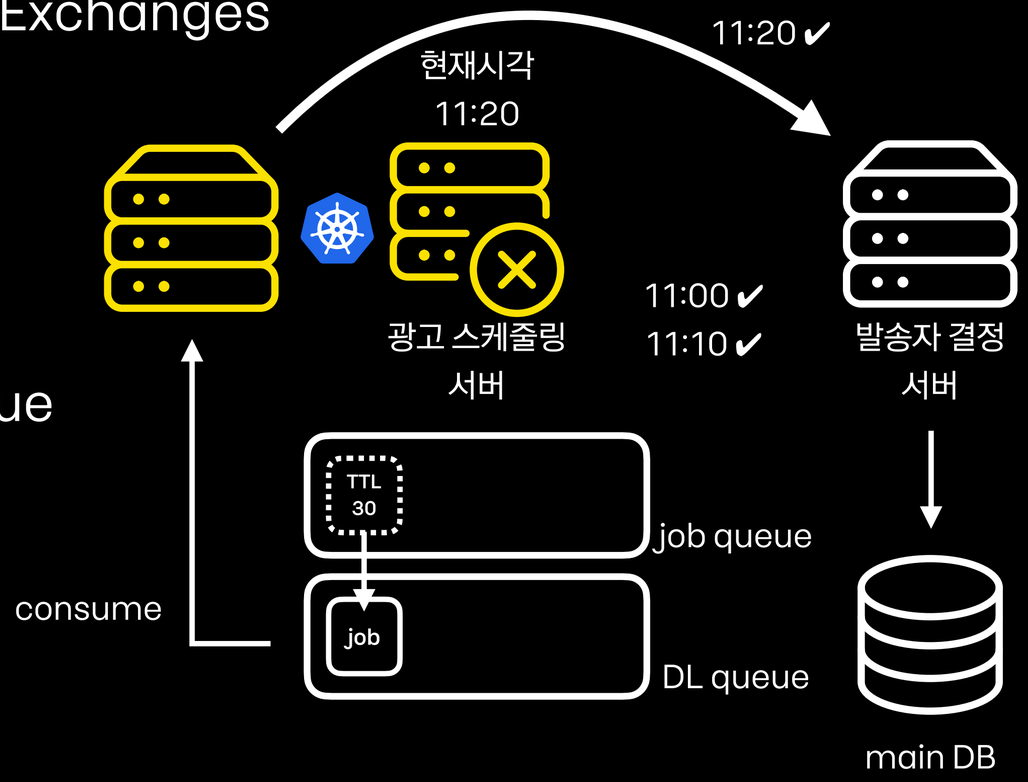
RabbitMQ 기반 delay queue

- Job 스케줄 정보를 프로세스 메모리 외부에 안전하게 저장
- Delay Queue: Message TTL + Dead Letter Exchanges

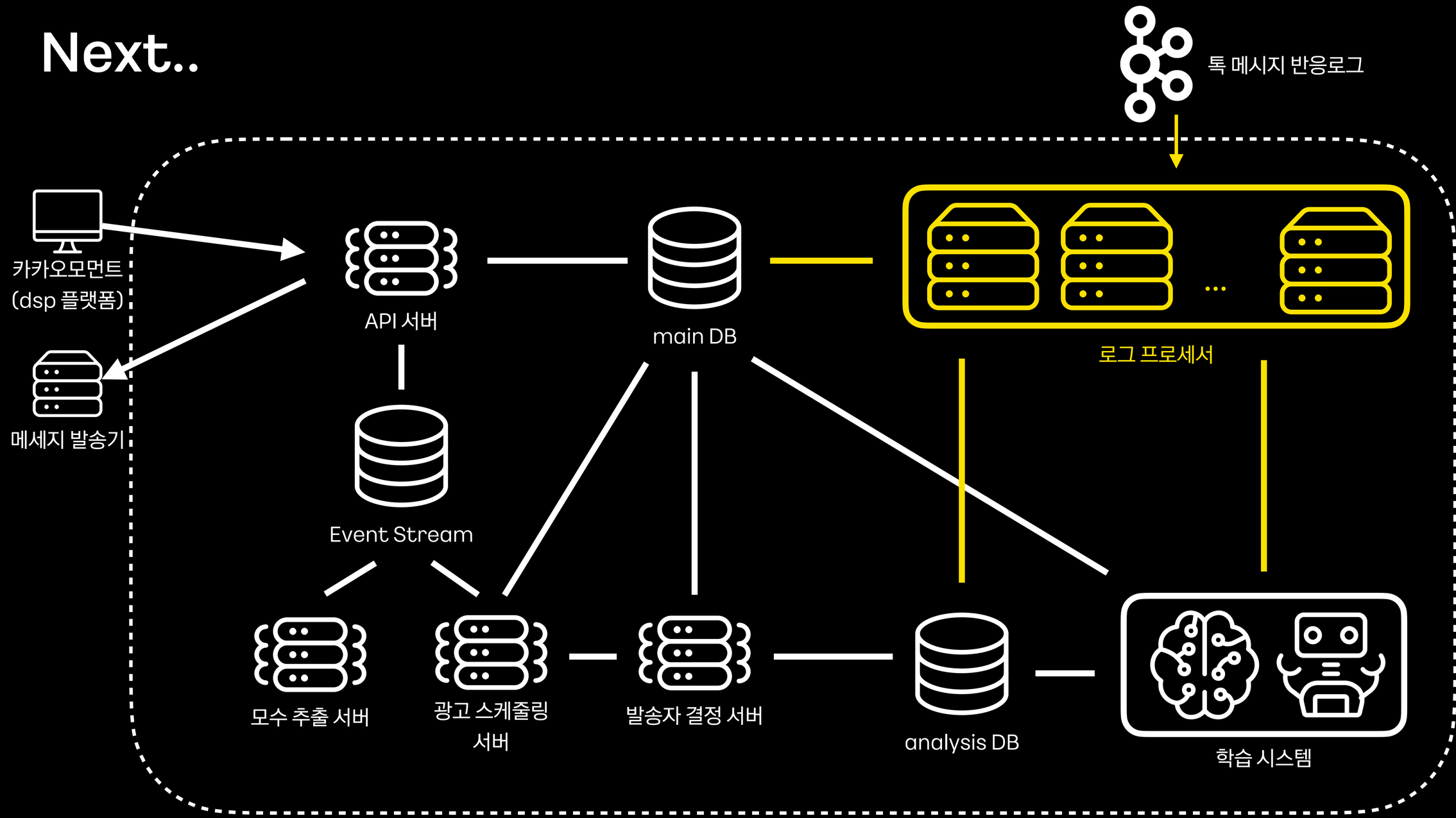
  Message 에 TTL 설정 후 queue에 넣으면,

  설정한 TTL이 지난 후 DLX 로 이동한다
- DLX를 별도의 queue 에 bind

  → 일정 지연 시간 후 메세지가 도달하는 delay queue



Next..


톡 메시지 반응로그
카카오모먼트
(dsp 플랫폼)
메세지 발송기
API 서버
main DB
로그 프로세서
Event Stream
모수 추출 서버
광고 스케줄링
서버
발송자 결정 서버
analysis DB
학습 시스템

카프카 스트림즈를 선택한 이유

카프카 상태기반처리 운영

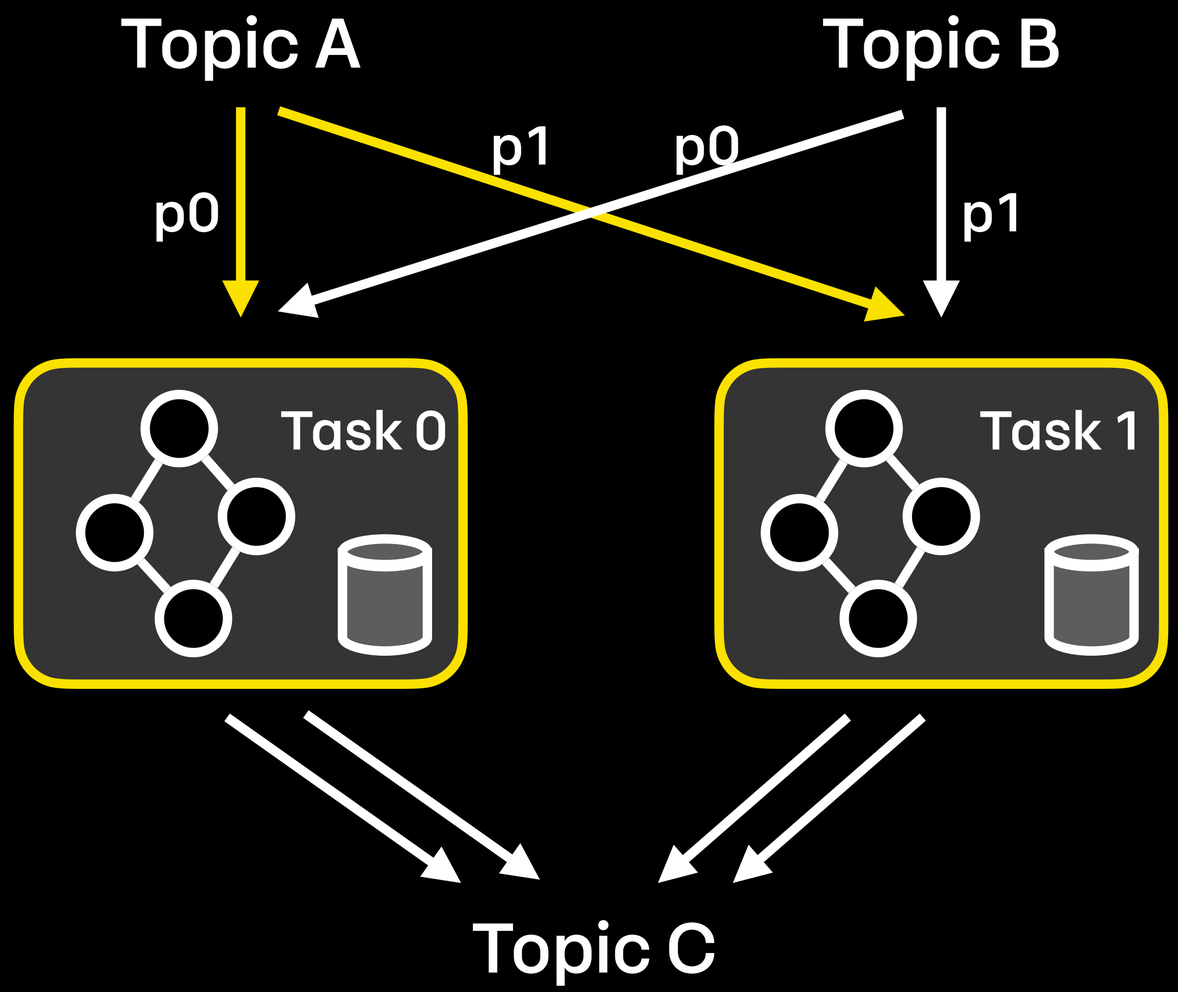
Kafka Stateful maintenance

스트림 데이터 처리 기술 비교

	카프카 스트림즈	스파크 구조적 스트리밍
카프카 클러스터와 연동	지원(공식 릴리즈)	지원
스트리밍 소스	카프카 토픽	소켓, HDFS, 카프카 토픽 등
상태기반처리 방법	메모리 / rocksDB + changelog 토픽	메모리 / 디스크
데이터 처리 보장	Exactly-once	Exactly-once
릴리즈 주체	아파치 카프카	아파치 스파크
운영하는 방법	masterless	Driver + Executor
지원하는 언어	Java	Scala, Java, Python, R, SQL
데이터 처리 방법	Continuous Streaming	Micro-batching

카프카 스트림즈 내부 아키텍처

- 라이브러리로 제공
- 스트림 처리를 위한 DSL 또는 프로세서API 제공
- 오픈소스 아파치 카프카가 공식적으로 릴리즈
- 카프카 파티션 개수만큼 유연하게 스케일 아웃
- 별도의 클러스터나 스케쥴링 도구 필요 없음
- 상태기반 처리를 위해 로컬에 rocksDB를 사용
- 상태를 기록하기 위해 변경로그(changelog)를 토픽으로 안전하게 저장



카프카 스트림즈DSL과 프로세서API

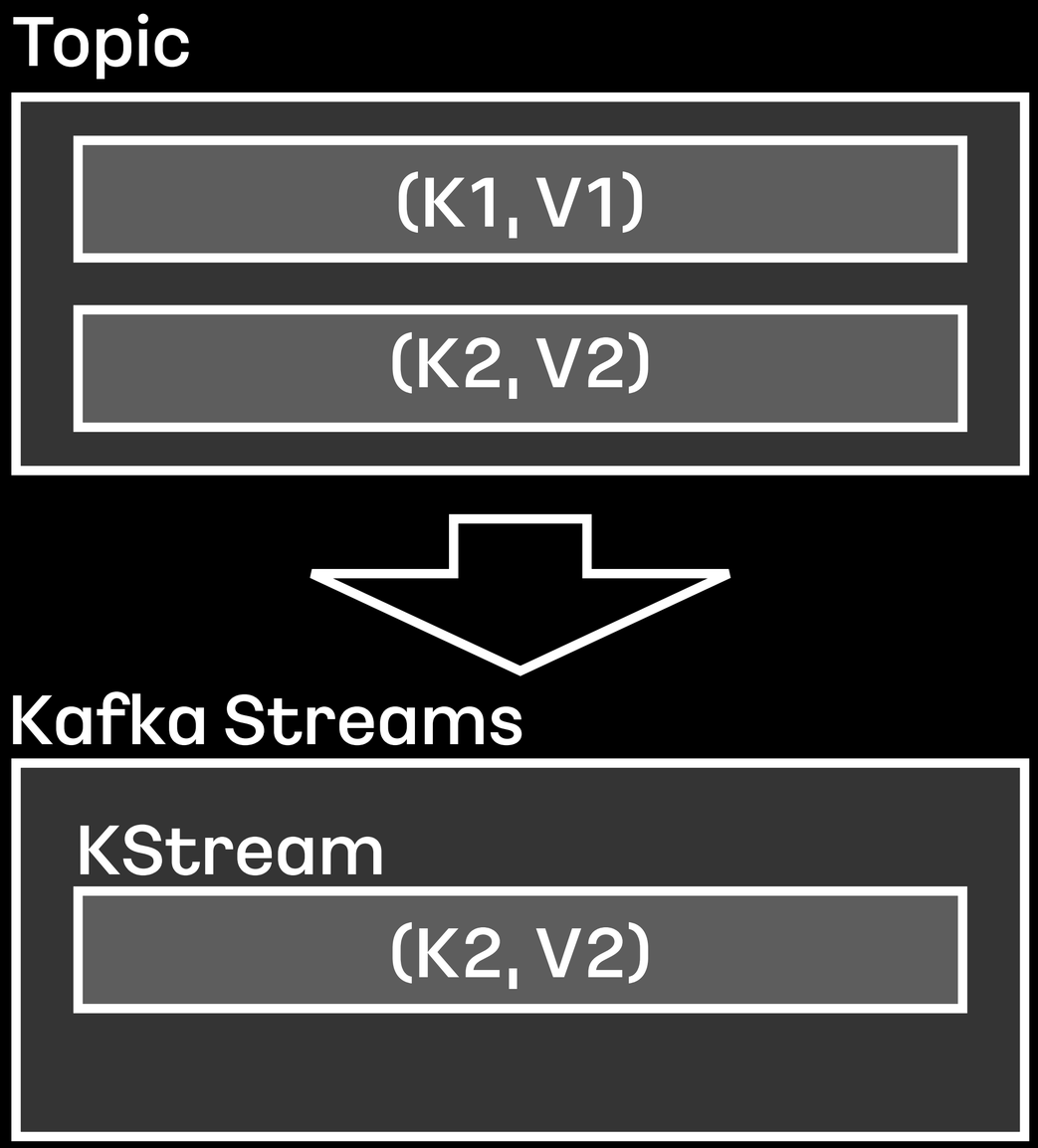
스트림즈DSL

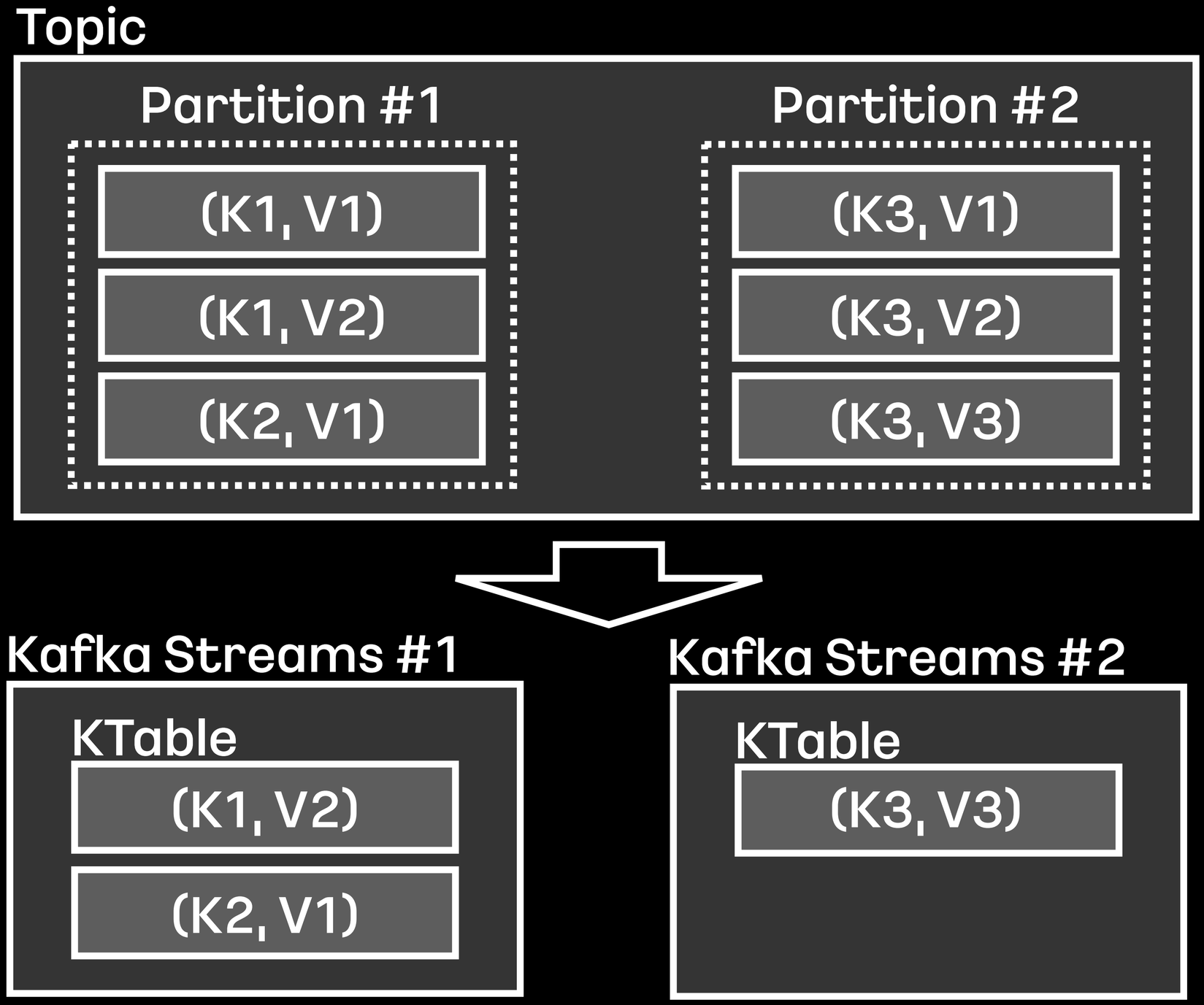
- Stateful, Stateless 처리를 위한 대부분의 메서드들(map, window,join, aggregation 등)이 제공됨
- 메시지 키, 메시지 값 기반 토폴로지 처리 수행.
- KStream, KTable, GlobalKTable 제공

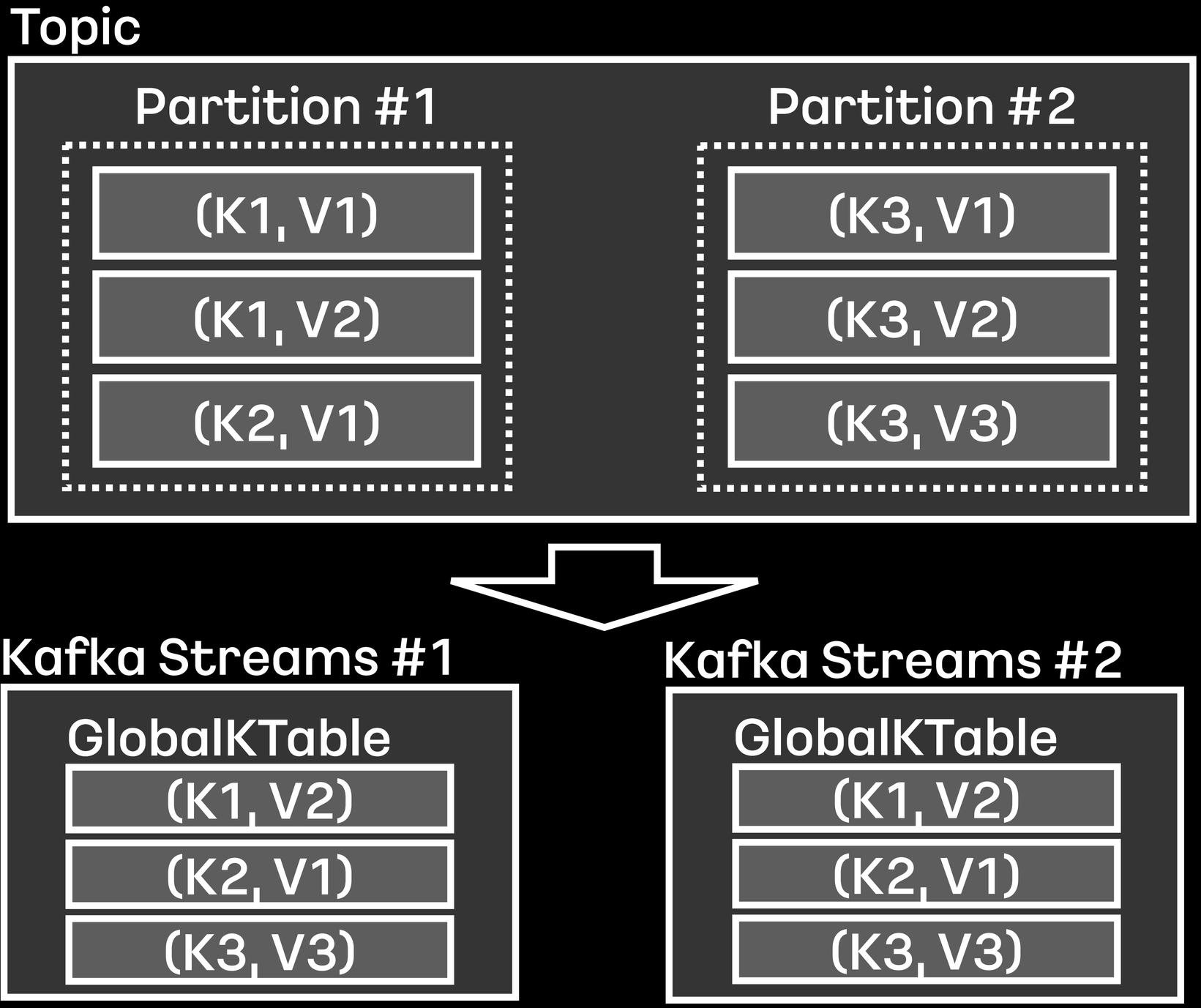
프로세서API

- 스트림즈DSL에서 제공하지 않는 스트림 처리(scheduler 등)를 레코드 단위로 구현 가능
- 메시지 키, 메시지 값, 헤더, 타임스탬프 사용하여 스트림 데이터 처리

```
KStream<String, String> advertiseStream = builder.stream(AdvertiseTopic);
KStream<String, String> smartMessageStream = advertiseStream.filter(((key, value)
                                                -> isSmartMessage(value)));
smartMessageStream.to(FilteredAdvertiseTopic);
```


Topic
(K1, V1)
(K2, V2)
Kafka Streams
KStream
(K2, V2)


Topic
Partition #1
(K1, V1)
(K1, V2)
(K2, V1)
Partition #2
(K3, V1)
(K3, V2)
(K3, V3)
Kafka Streams #1
KTable
(K1, V2)
(K2, V1)
Kafka Streams #2
KTable
(K3, V3)


Topic
Partition #1
(K1, V1)
(K1, V2)
(K2, V1)
Partition #2
(K3, V1)
(K3, V2)
(K3, V3)
Kafka Streams #1
GlobalKTable
(K1, V2)
(K2, V1)
(K3, V3)
Kafka Streams #2
GlobalKTable
(K1, V2)
(K2, V1)
(K3, V3)

카프카 스트림즈DSL과 프로세서API

스트림즈DSL

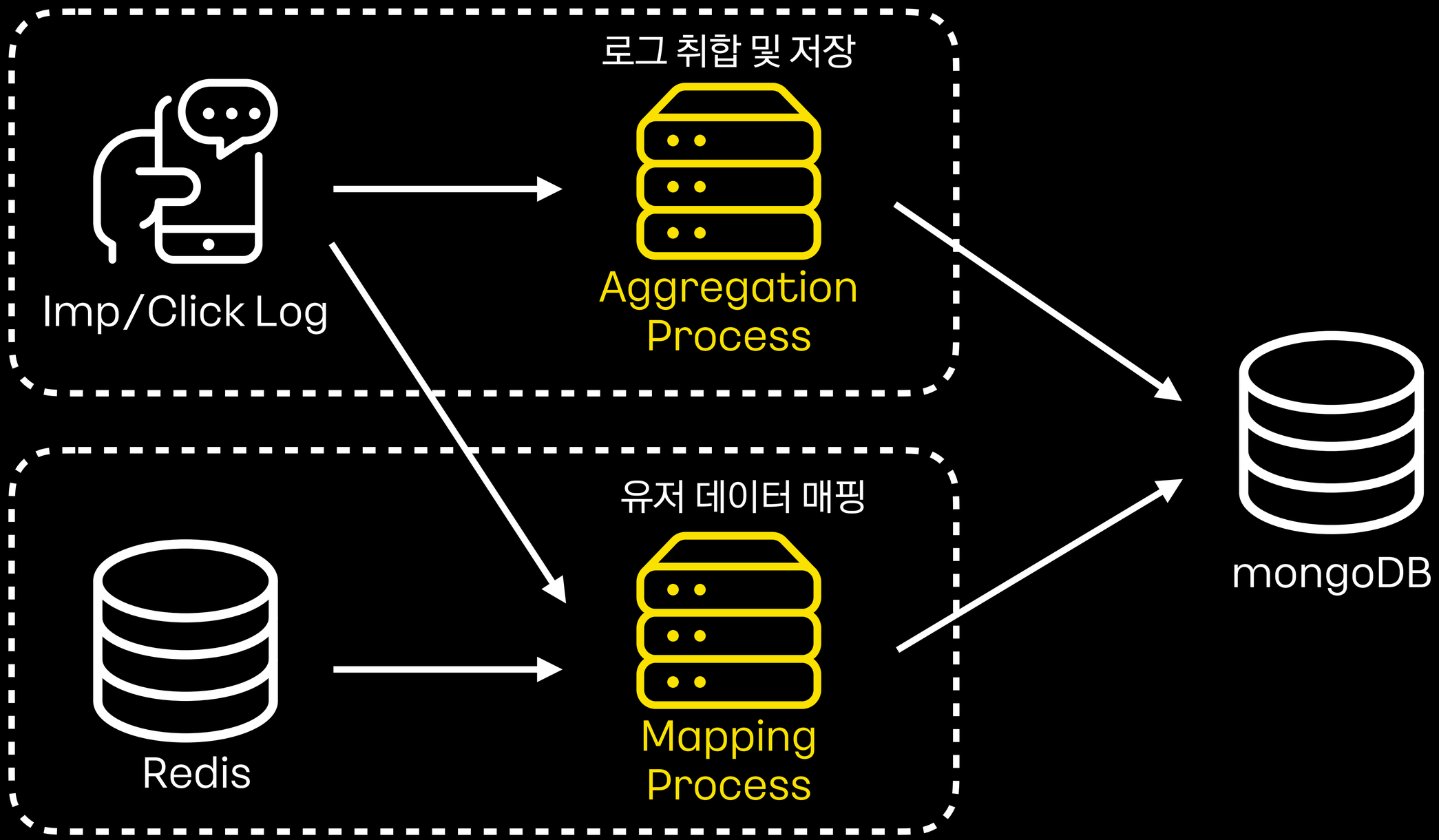
- Stateful, Stateless 처리를 위한 대부분의 메서드들(map, window,join, aggregation 등)이 제공됨
- 메시지 키, 메시지 값 기반 토폴로지 처리 수행.
- KStream, KTable, GlobalKTable 제공

프로세서API

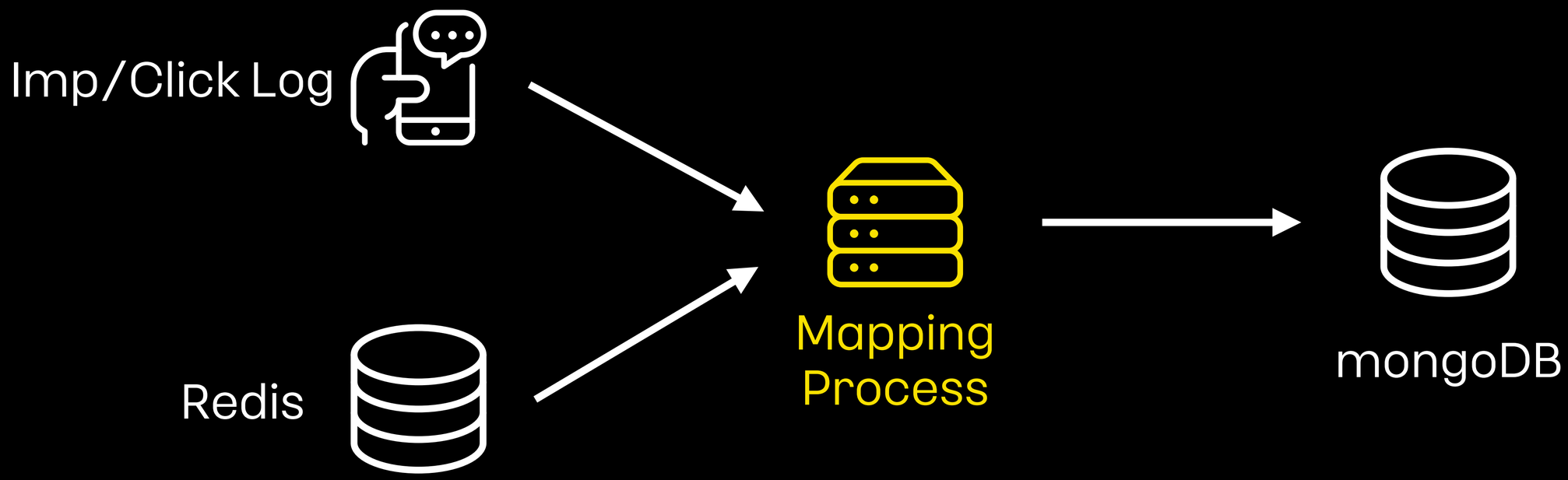
- 스트림즈DSL에서 제공하지 않는 스트림 처리(scheduler 등)를 레코드 단위로 구현 가능
- 메시지 키, 메시지 값, 헤더, 타임스탬프 사용하여 스트림 데이터 처리

```
KStream<String, String> advertiseStream = builder.stream(AdvertiseTopic);
KStream<String, String> smartMessageStream = advertiseStream.filter(((key, value)
                                                -> isSmartMessage(value)));
smartMessageStream.to(FilteredAdvertiseTopic);
```

스마트 메시지에서 카프카 스트림즈 적용


로그 취합 및 저장
Imp/Click Log
Aggregation
Process
유저 데이터 매핑
Redis
Mapping
Process
mongoDB

광고 로그와 유저 정보의 결합




```
KStream<String, String> reactionMessage = builder.stream(reactionMessages);
KStream<String, String> validMessage = reactionMessage.filter((k, v) -> MessageUtil.isJson(v));
KStream<String, String> userJoinedMessage = validMessage.map((key,value) -> joinUserInformation(key, value));
userJoinedMessage.foreach((key,value) -> mongodbConnector.save(key,value));
```

Aggregation 프로세싱

```
KStream<String, String> userJoinedMessage = builder.stream(userJoinedTopic);

KTable<Windowed<String>, String> aggregationMessageTable = userJoinedMessage
                        .groupByKey()
                        .windowedBy(TimeWindows.of(Duration.ofMinutes(1)))
                        .aggregate(() -> "{}", new UserAggregator());

KStream<String, String> aggregatedStream = aggregationMessageTable
                        .toStream()
                        .map((windowedId, value) ->
                             getStreamAggregation(windowedId.key(), value));

aggregatedStream.foreach((key,value) -> mongodbConnector.save(key,value));
```

스트림즈 애플리케이션 테스트 코드

```
TopologyTestDriver testDriver = new TopologyTestDriver(streams.setTopology());


TestInputTopic<String, String> inputTopic =  testDriver.createInputTopic(userJoinedTopic, stringSer(), stringSer());
TestOutputTopic<String, String> outputTopic = testDriver.createOutputTopic(userAggregatedTopic, stringDer(), stringDer());


inputTopic.pipeInput(aggregateKey, "{\"car\":0.1}");


KeyValue<String, String> expectedKeyValue = new KeyValue<>(aggregateKey, "{\"car\":{\"aggr\":0.1}}}");
assertEquals(expectedKeyValue, outputTopic.readKeyValue());
```

감사합니다