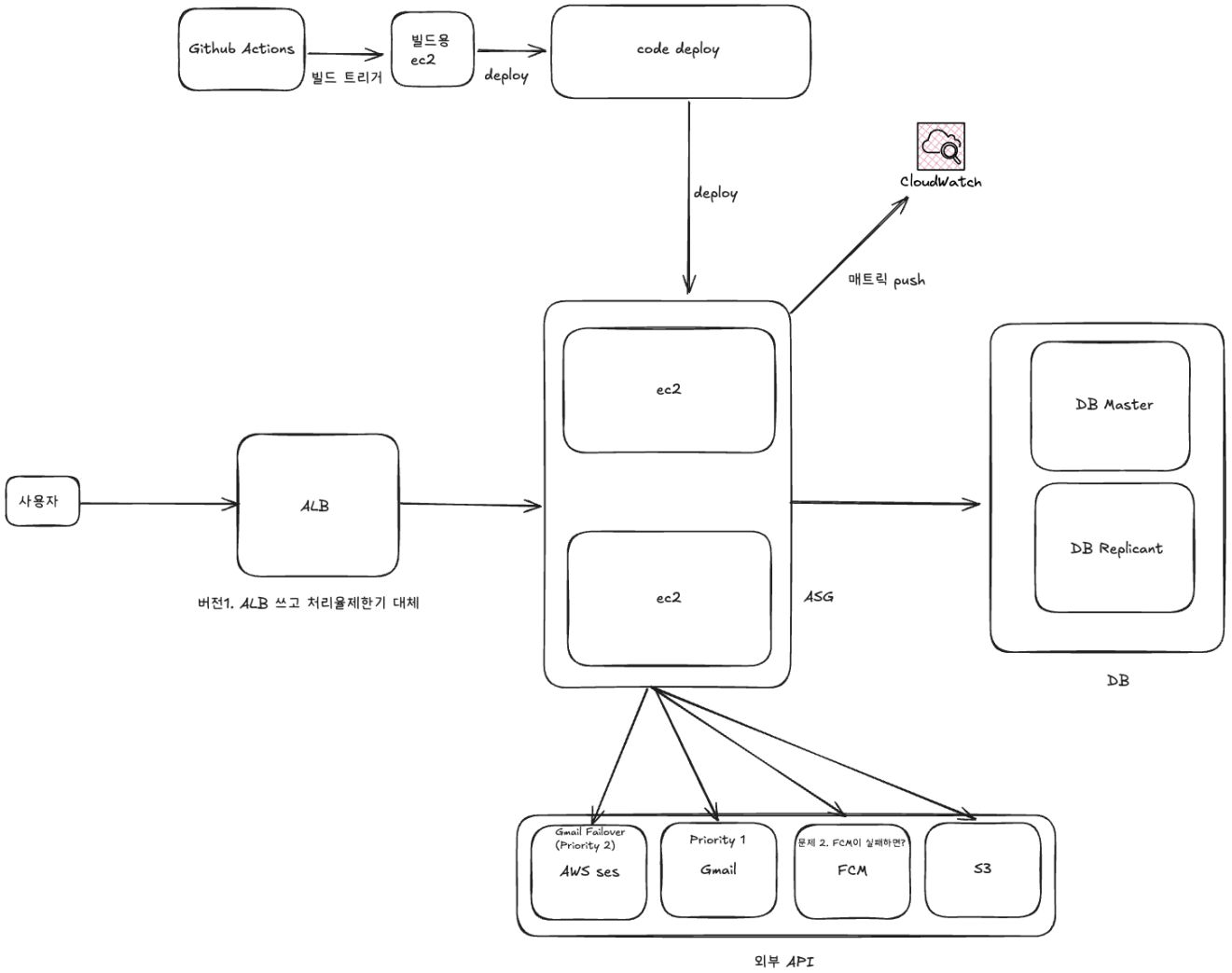
알림 아키텍쳐 개선기

7기 BE 투다

지난 이야기

- 메시지 그 자체를 데이터베이스에 저장하여 영속성을 보장하자
  - 알림을 보낼때마다 데이터베이스를 조회하게 된다
  - 향후 병목이 생기는 지점이 무엇이 있을지 생각해보았고 공유해보려고함

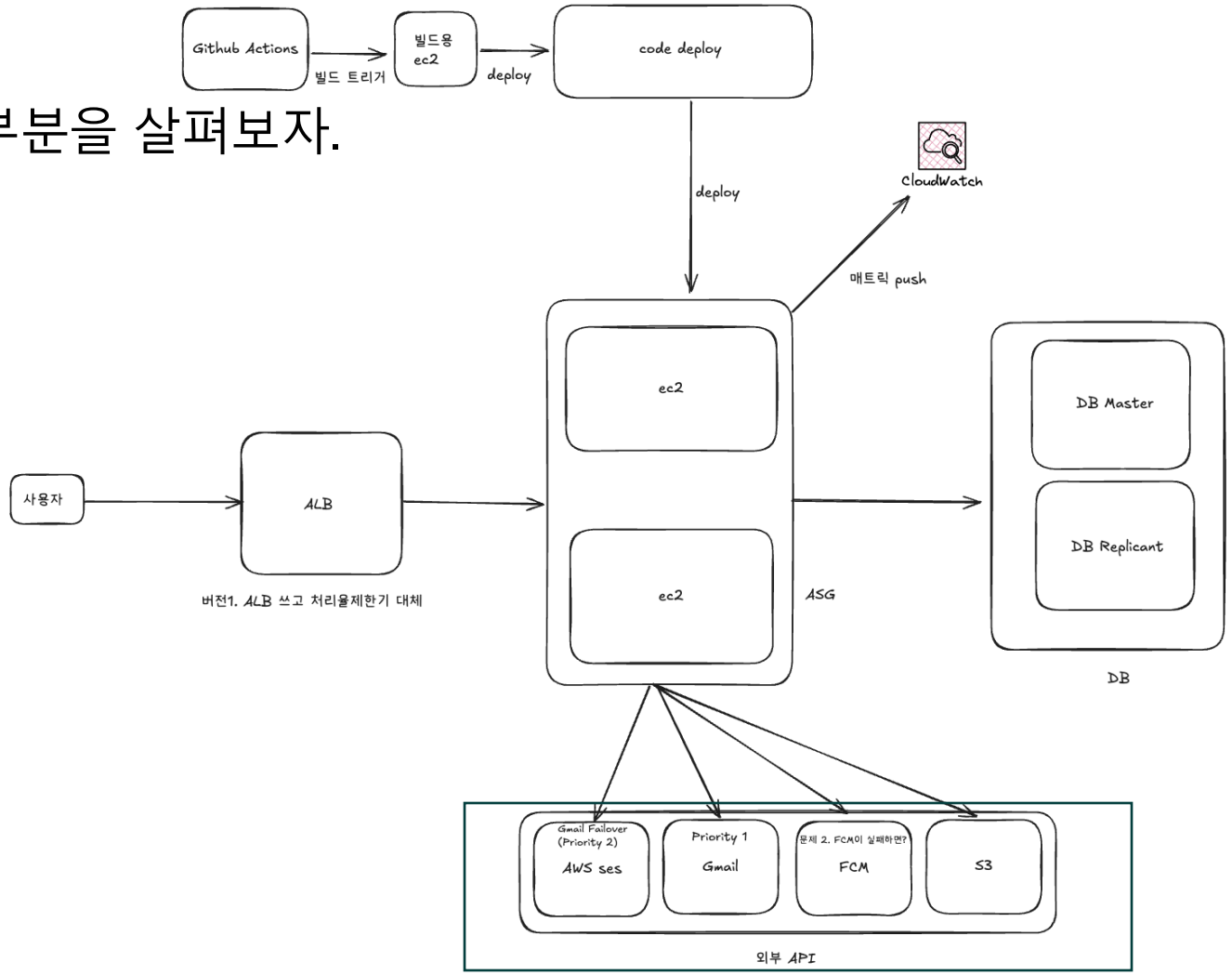
현재 아키텍쳐



현재 아키텍쳐

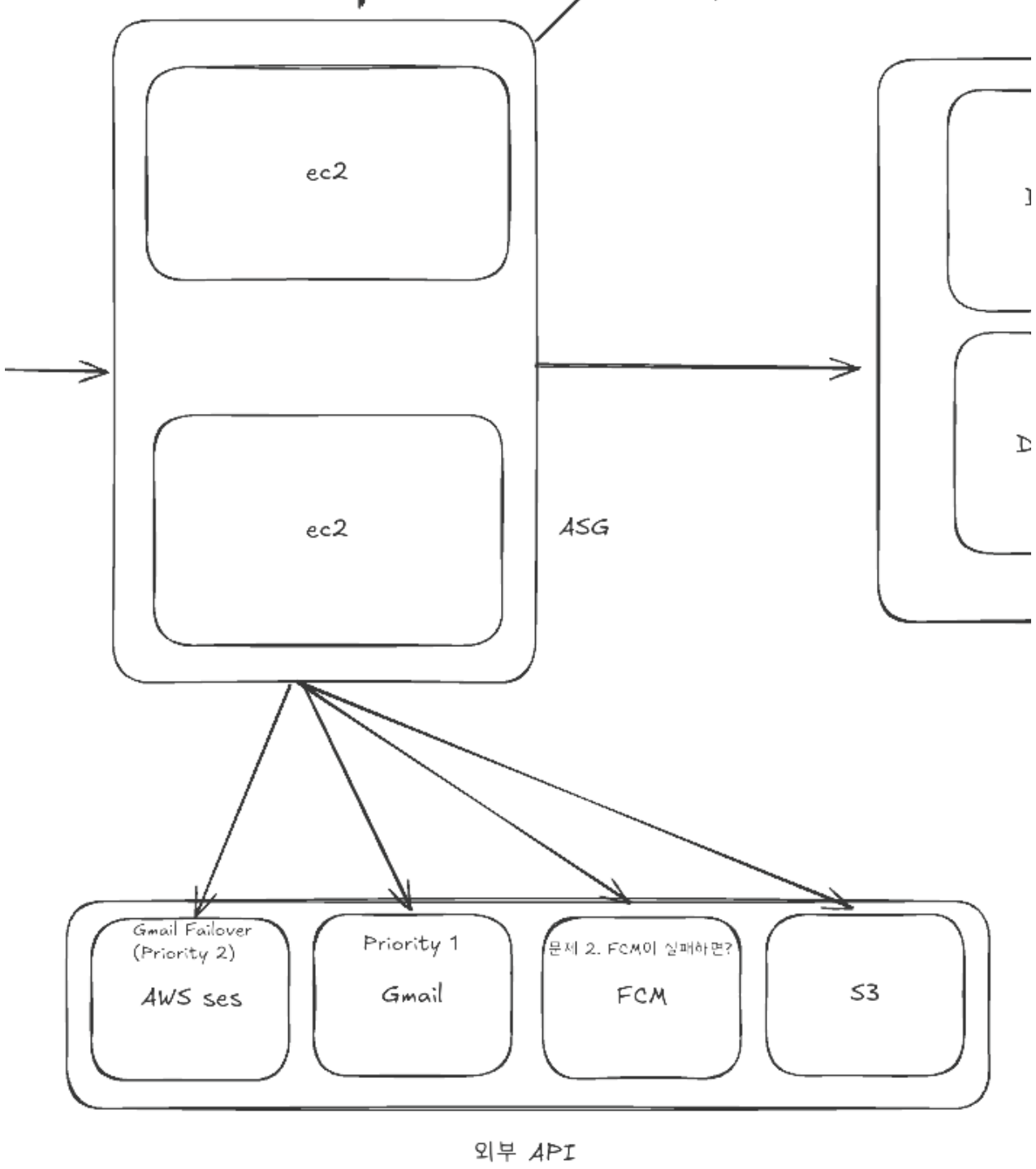
현재 아키텍쳐

- 특히 중요한 알림 부분을 살펴보자.



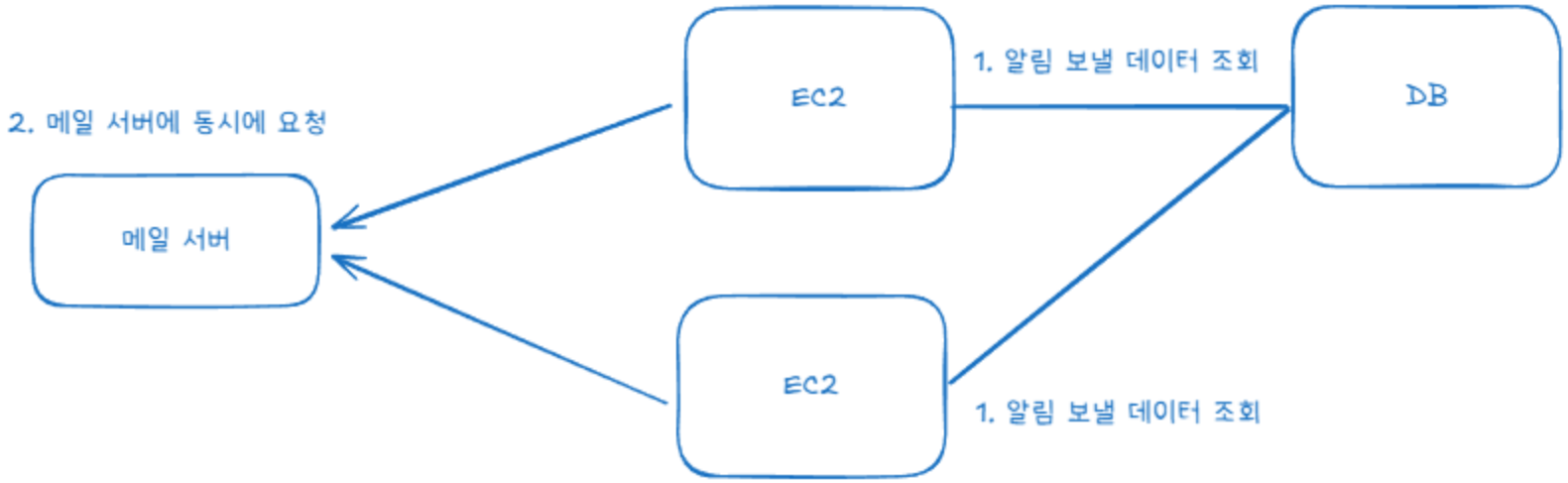
현재 아키텍쳐

- 현 알림 전송 프로세스(DB 풀링 방식)
  - 알림은 1분마다 스케줄링하여 전송한다.
  - 서버는 분산되어 있다.



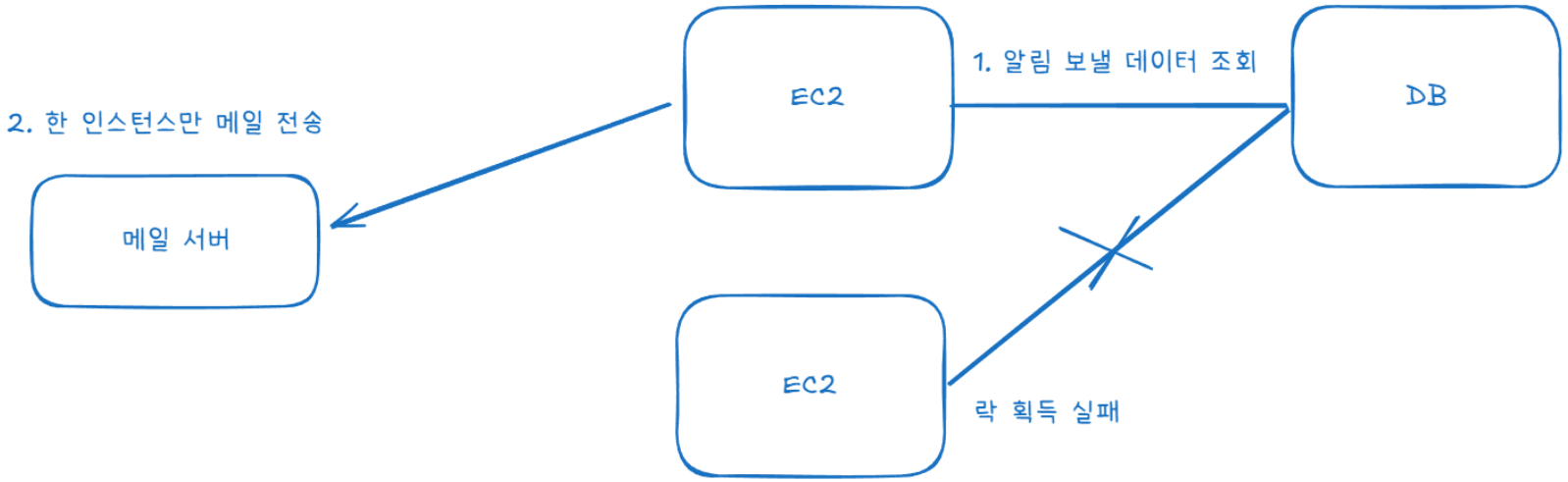
문제 1. 중복 알림 전송

- 각각의 분산된 서버들이 스케줄링 한다면?
  - 1분마다 모든 알림 이벤트들을 조회하여 전송한다
  - 중복된 메일을 전송한다



중복 알림 전송 해결

- 락을 걸어서 해결하자 (비관적락을 예로 들자)
  - 1개의 인스턴스가 1분마다 모든 알림 이벤트를 조회한다.
  - 락을 획득한 1개의 인스턴스만 메일을 전송한다



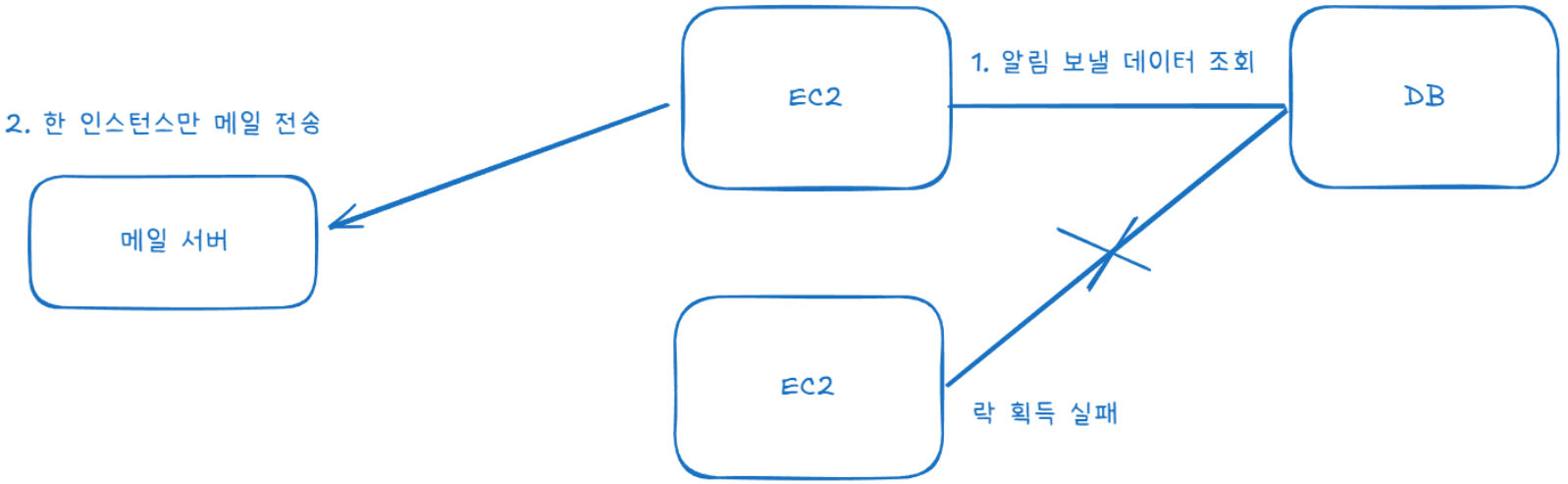
만약 알림 데이터가 많아진다면?

1분동안 10만개 이상의 알림 이벤트들이 쌓인다고 가정하자. 이는 곧 1분마다 DB에서 10만개의 조회를 하는 것이다.

> 주의) 데이터가 10만개인 것이 중요한 것이 아니다. 데이터가 많아졌을때 대응할 수 있는 아키텍쳐를 설계하는 것에 집중하기 위함이다. 10만개 정도라면 실제로는 크게 느리지 않을 수 있다.

현 아키텍쳐가 데이터가 많아져도 대응할 수 있는 구조인지 살펴보자

문제 2. 확장성 부재



한개의 인스턴스가 모든 알림 데이터를 처리하고 있군..

문제 2. 확장성 부재

문제점 1. 데이터베이스의 부담

DB에서 10만개의 데이터를 가져와야하면, 내부적으로 인덱스의 유무와는 불과하게
리프노드에서 10만개를 순회하며 데이터를 적재해야한다.

- 이는 1분마다 CPU,메모리를 많이 사용하는 동작이다.

문제점 2. 애플리케이션의 부담

- 10만개를 순회하며, 요청을 해야하기때문에 10만개의 데이터를 가져와야한다. 이는
  메모리상에서 부담이 된다.

1. 항상 한개의 인스턴스만 일한다 - **확장성의 부재**
2. 모든 데이터를 다 가져온다
   - 메모리, CPU, 커넥션 오래 점유 등 리소스 소요가 크다

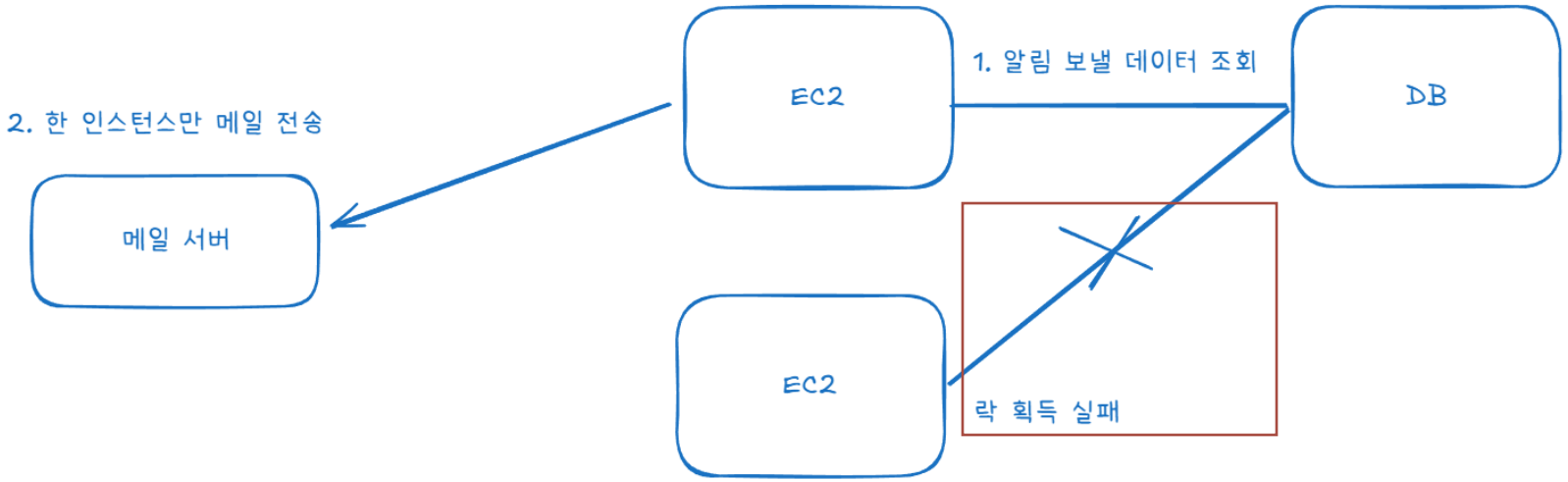
개선 1. 배치 적용

모든 데이터를 가져와 처리하는 문제를 개선해보자.

- 이를 위해 **DB 배치 풀링** 방식으로 전환하자

기존 락 방식의 문제점

락으로 중복 문제를 해결했었다.
그런데 기존 방식의 락은 비관적 락이다.
-> 병목이 생긴다



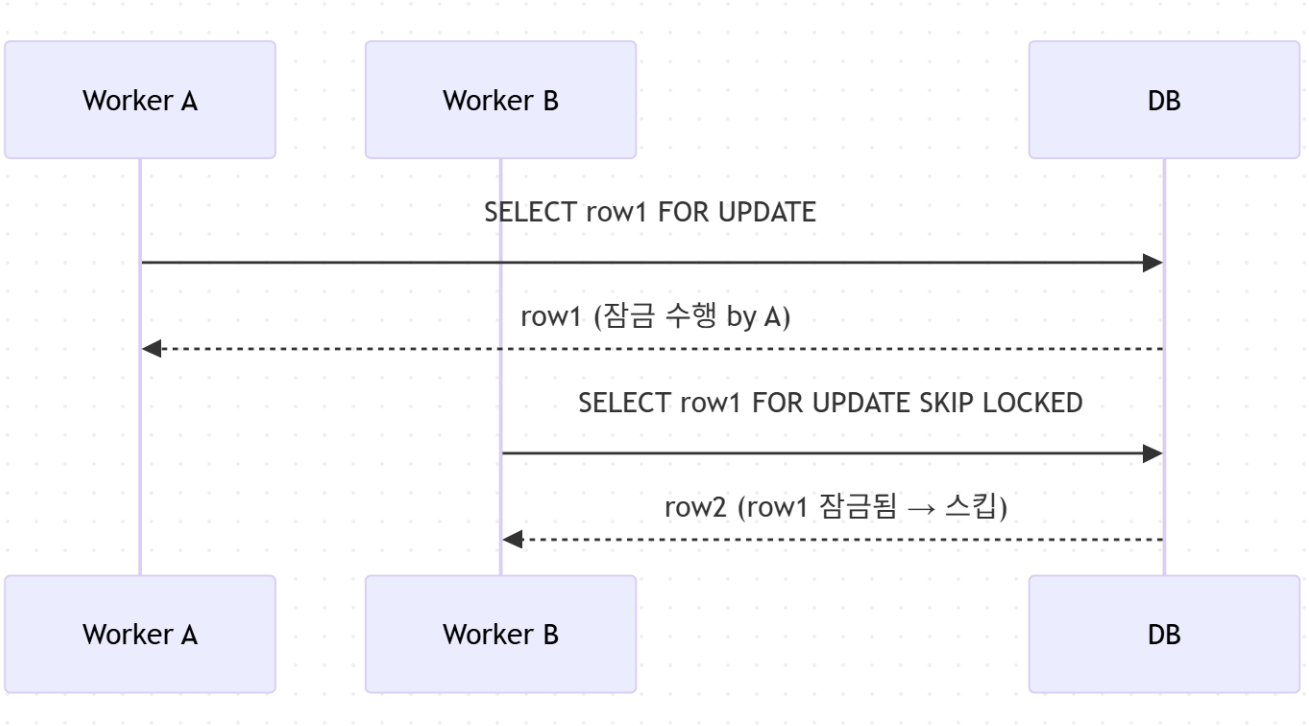
기다리는 인스턴스는 놀게 된다.

개선 2. 락 개선

기다리는 락이 아닌, **생략**하는 락은 없을까?

is 스킵락(skip lock)

스킵락을 사용하면 락을 걸어도
기다리지 않아 병목이 적다

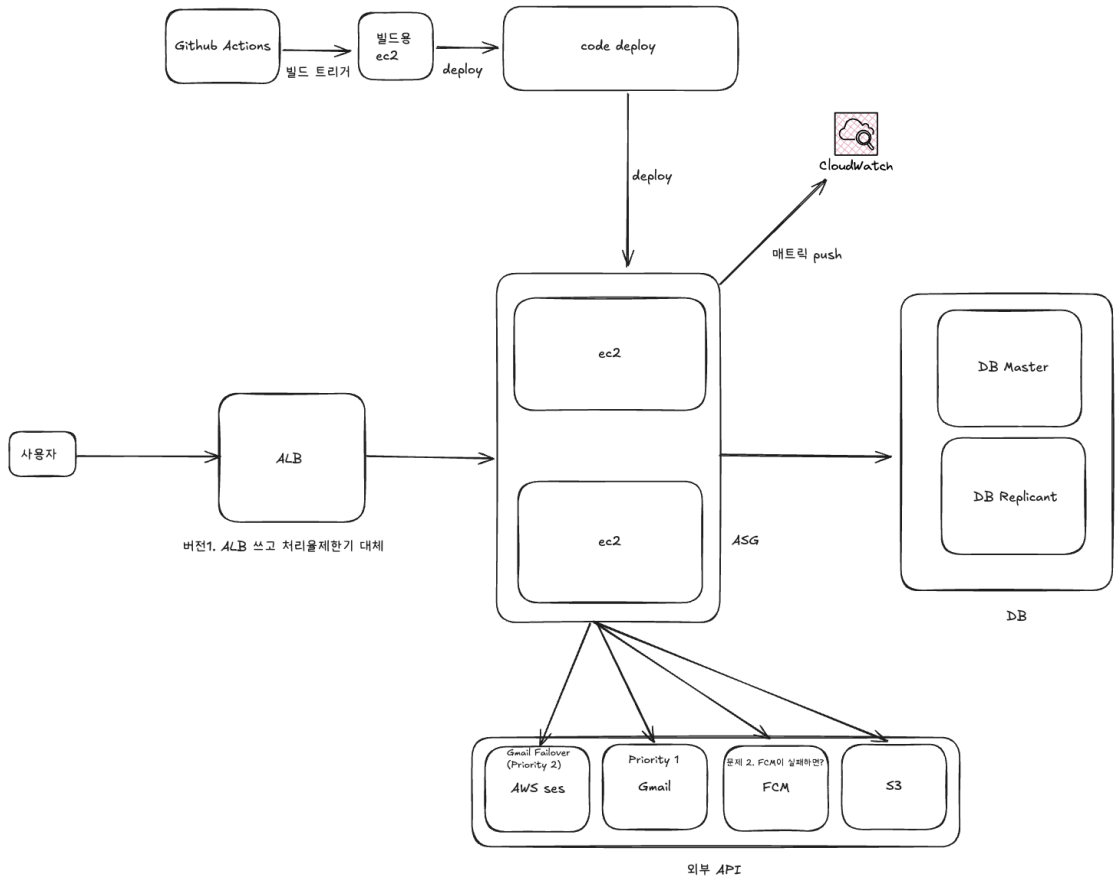


해치웠나?

skiplock을 적용했다면 배치 풀링으로 1000개씩 100번 반복할 것이다.
skiplock이기 때문에 락이 걸려있으면 빠르게 다음 청크로 넘어갈 것이다.
그러나 **100번** 요청한다는 점에서 여전히 DB 병목을 막을 수 없다.

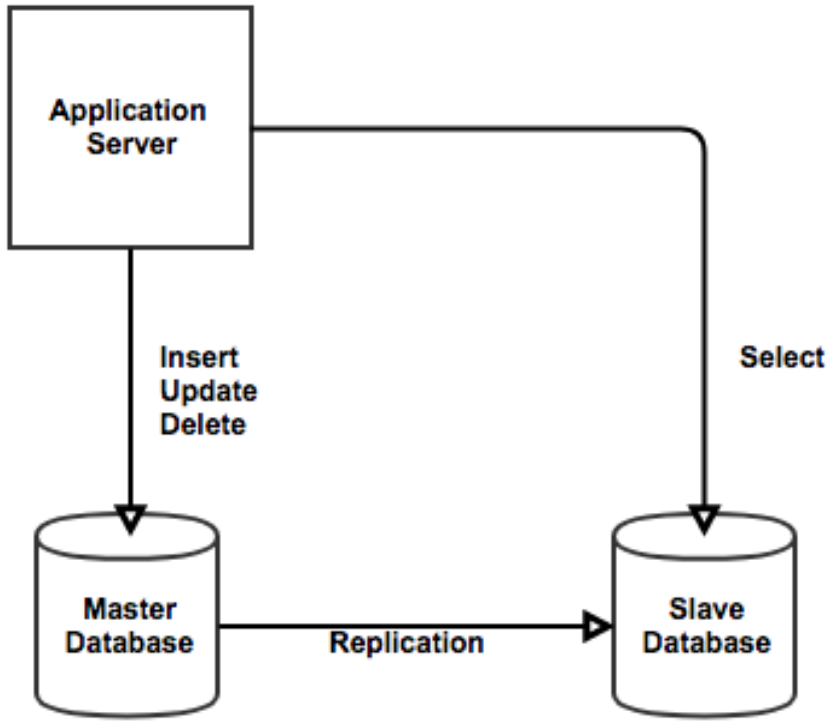
개선 3. DB의 부담 줄이기

데이터베이스의 부담을 줄이는 방법은 다양하다.
이번에도 확장이란 키워드로 접근해보자.



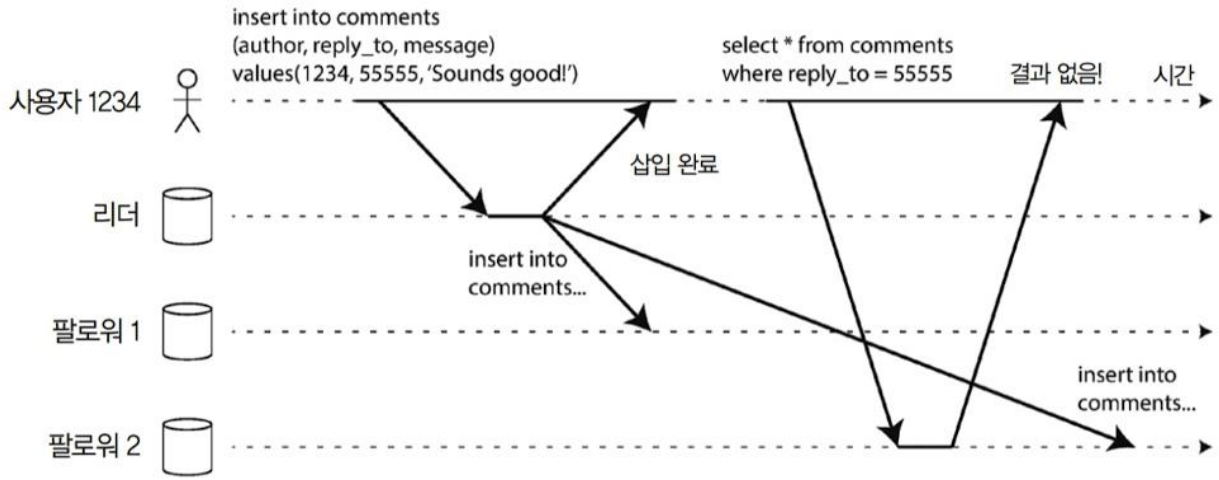
개선 3. DB의 부담 줄이기

기존 우리 웹 애플리케이션에서는 쓰기가 적고 정합성이 중요하기 문에
쓰기 확장이 크게 필요없는 **Master Slave** 구조를 선택하였다.



복제 지연

Master Slave 구조에서는 복제 지연 문제가 발생한다.
복제지연은 쓰기 작업시에
다른 slave DB들까지 반영되기에 시간이 소요되는 것



복제 지연

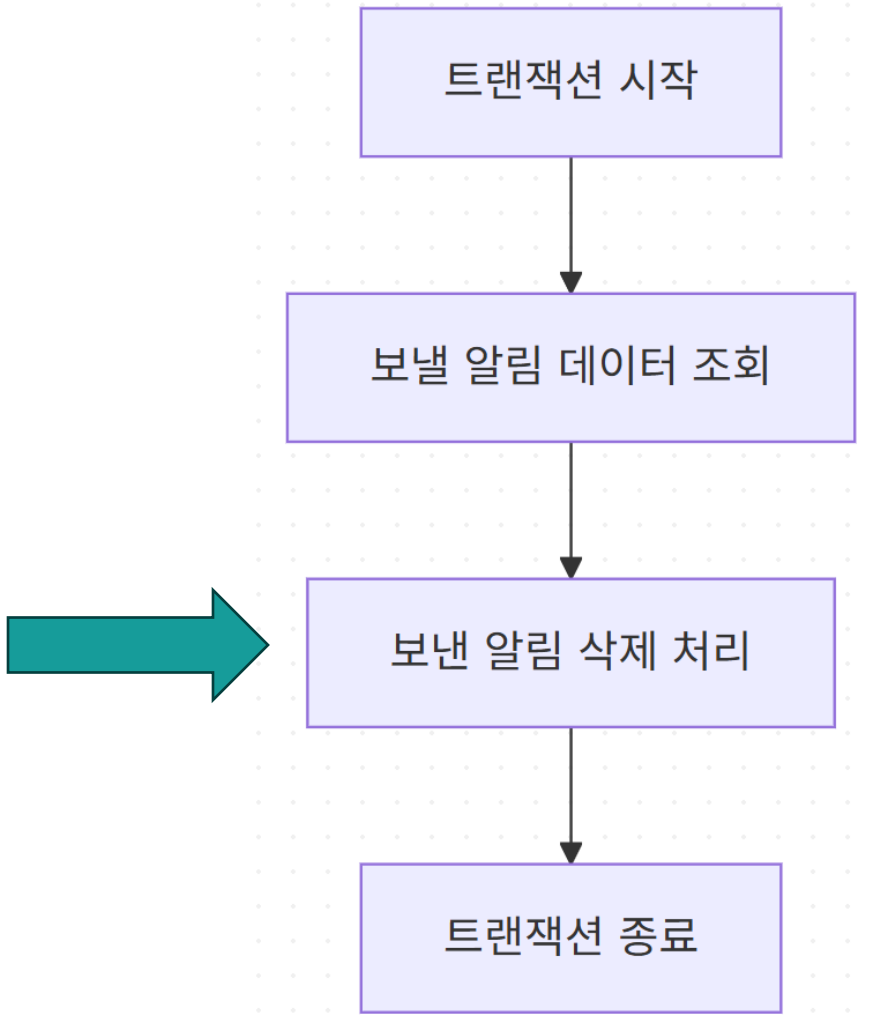
엥? 알림 조회만 하는데
쓰기로 인해 생기는 복제지연 문제가 왜생기나요?

복제 지연

사실은 거짓말을 했습니다..

복제 지연

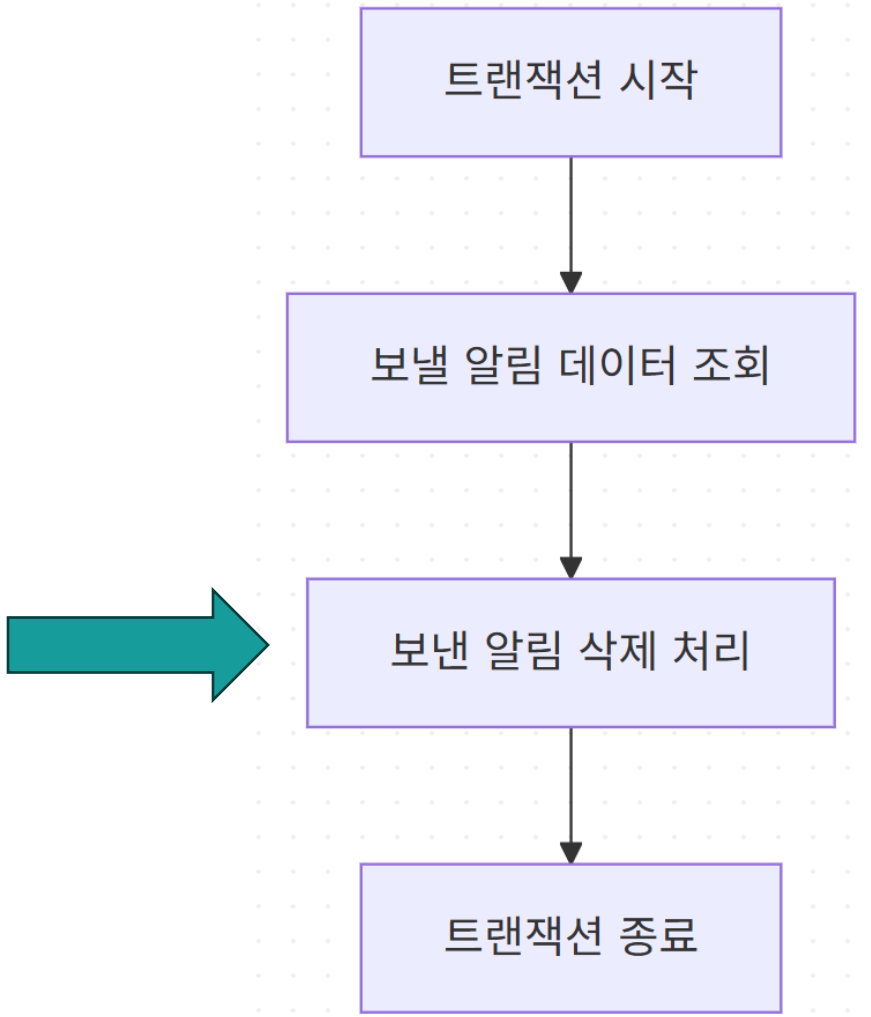
실제로는 쓰기도 하고 있다



복제 지연

실제로는 쓰기도 하고 있다

그렇다면, 복제 지연으로 인해
이미 보낸 알림을 다른 인스턴스들이 조회가능하다
- 재전송 문제 발생



복제 지연 해결

복제 지연 해결 방안 비교 표

구분	방법	장점	단점	특징
지연 자체 감소	Semi-Sync 복제	지연 감소, 데이터 유실 위험 감소	쓰기 성능 하락 가능	최소 1 Slave ACK 대기
	Slave 병목 제거(SSD, Parallel Replication)	처리 속도 향상, 지연 감소	운영 비용 증가	`slave_parallel_workers` 등 튜닝 필요
	네트워크 최적화	전송 지연 축소	리전 제약	Master–Slave 거리 최소화
	Binlog 압축	복제 전송량 감소	CPU 부하 증가	MySQL 8+ 기능
지연 우회 (설계적 회피)	Master 강제 읽기	일관성 유지	Master 부하 증가	Read-After-Write 필수 케이스
	캐시/이벤트 기반 최신성 유지	DB 지연 영향 감소	복잡성 증가	Redis/Kafka 등 활용
	Lag 기반 Smart Routing	지연 시 자동 우회	구성 복잡	ProxySQL 등 필요
	모델링 변경(쓰기 전용/조회 전용 분리)	복제 부담 감소	스키마 복잡	트랜잭션 패턴 최적화
구조 개편(근본적)	Group Replication/Galera	강한 일관성, 자동 Failover	쓰기 성능 저하	합의 기반 복제
	Sharding	수평 확장, write 분산	운영 복잡	대규모 시스템용

여러 방법이 있으나 근본적인 해결을 위한 **Master 강제 읽기** 도입!

복제시 생기는 문제점

중복된 알람이 생길 수 있기 때문에 Master를 통해 강제로 읽는다

- **결국 Master만** 사용한다는 것 아닌가..?

복제시 생기는 문제점

중복된 알람이 생길 수 있기 때문에 Master를 통해 강제로 읽는다

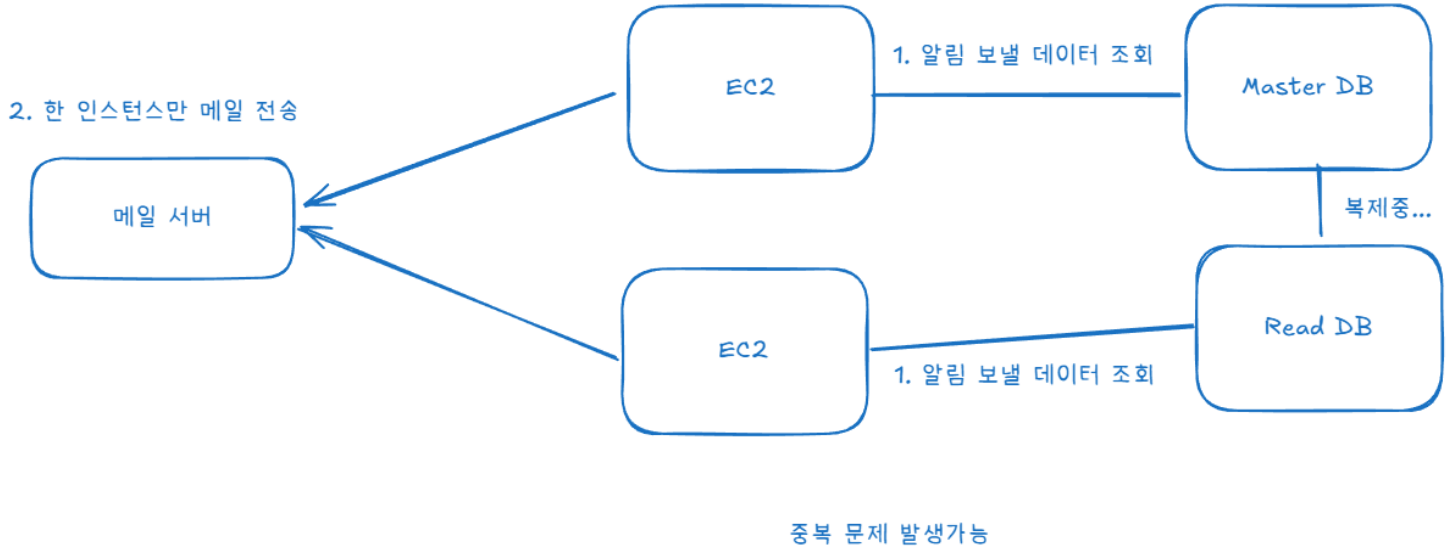
- **결국 Master만** 사용한다는 것 아닌가..?

OK 포기할게요.

멱등성 보장

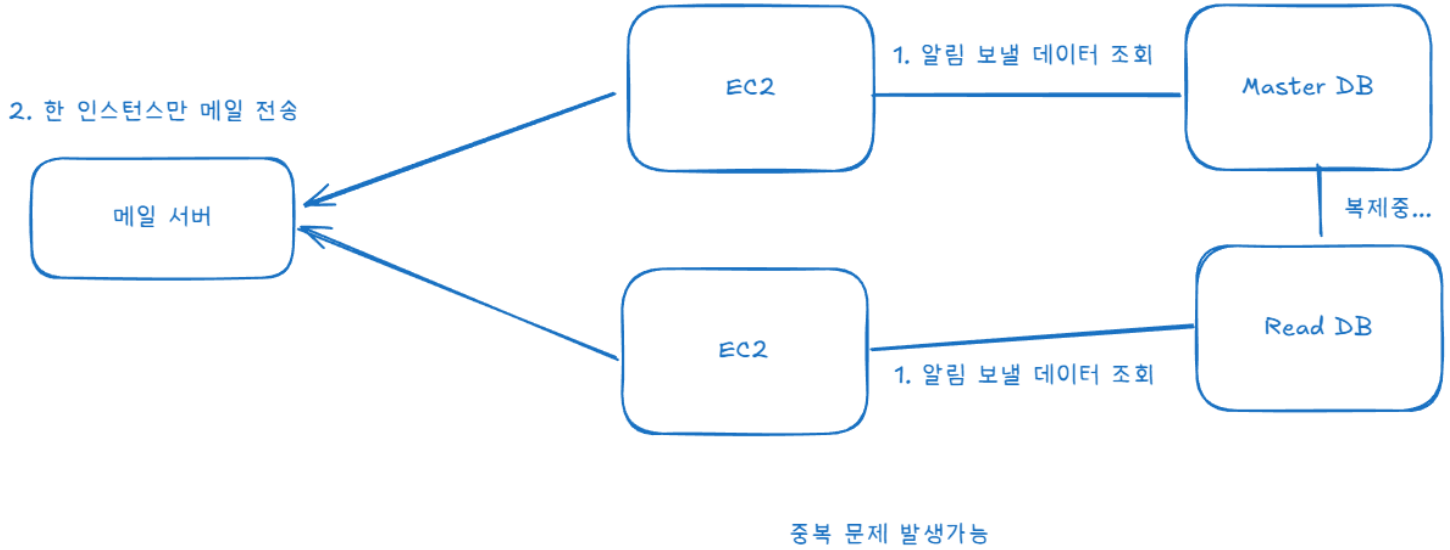
이제 skiplock도 사용하지 못한다
(락은 쓰기 서버에만 적용되기때문)
다른 방법을 찾아보자

- 중복 전송이 문제 아닌가?
- 복제지연을 허용한다면?



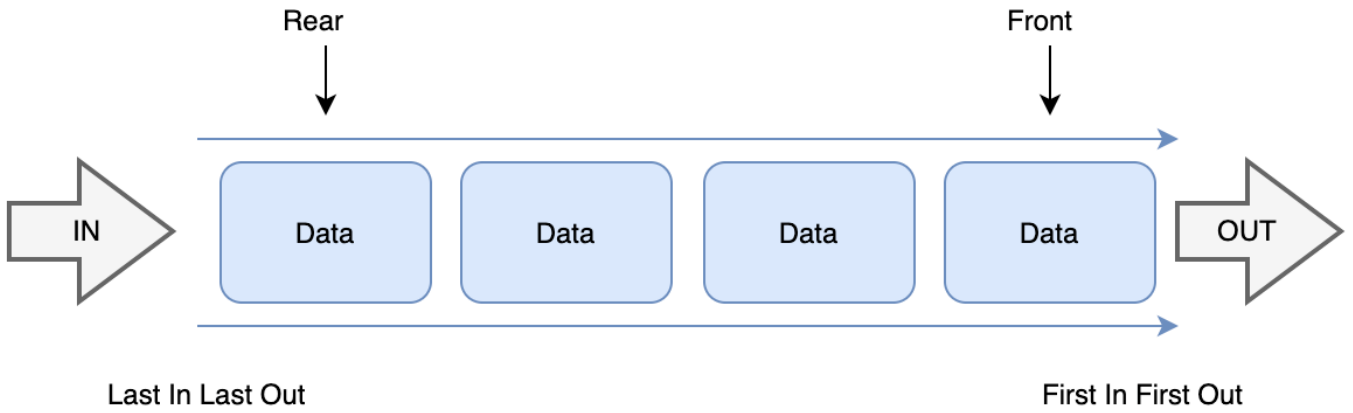
멱등성 보장

뭔가 **빠른 조회**를 지원하는
중간 계층이 있으면 되지 않을까..?



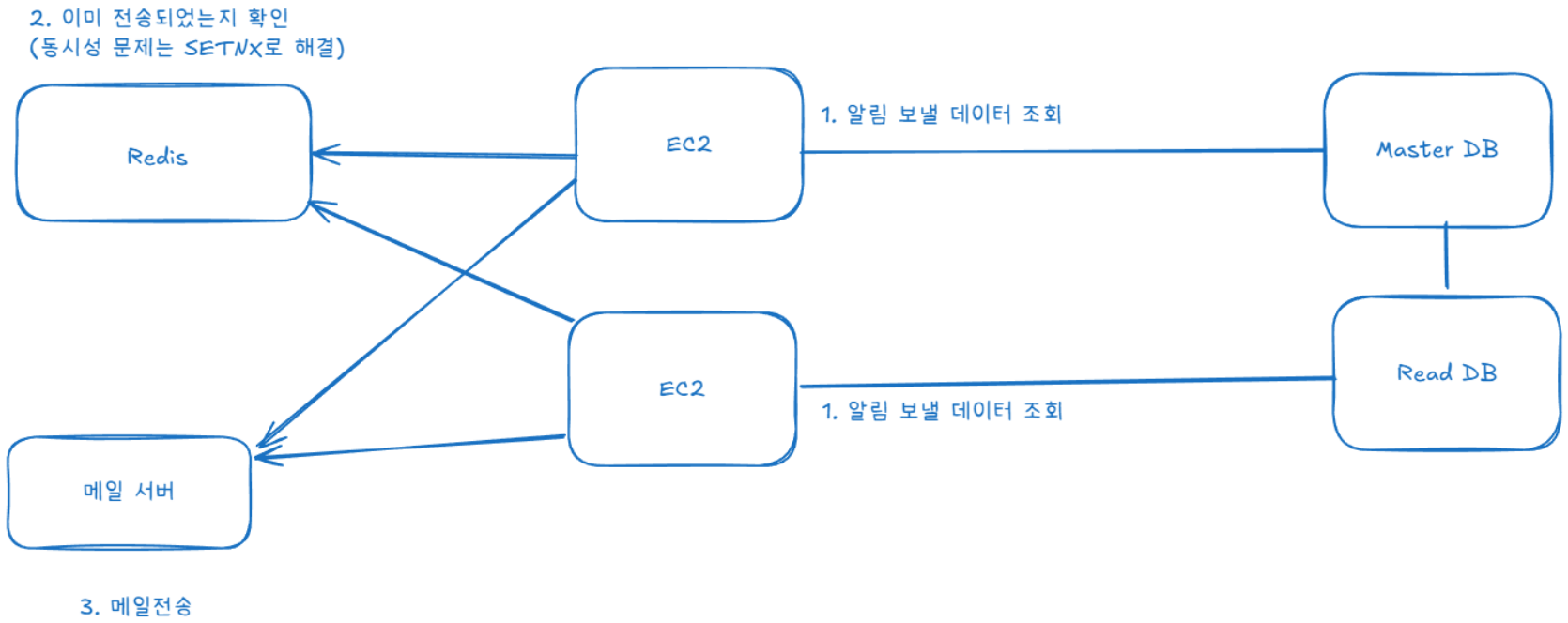
멱등성 보장

빠른 조회?


Rear
Front
IN
Data
Data
Data
Data
OUT
Last In Last Out
First In First Out

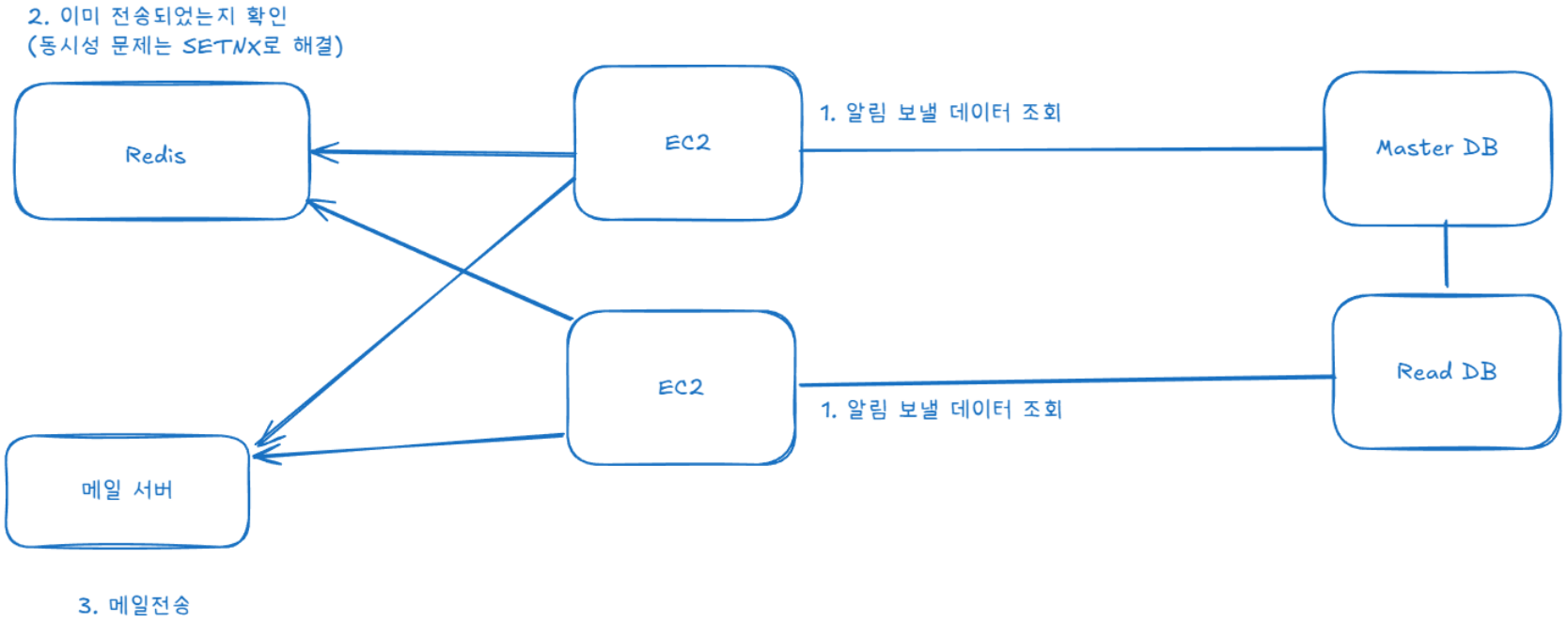
멱등성 보장

- Redis를 도입하자.
  - 멱등성 보장을 위한 저장소 역할



멱등성 보장

- 나머지는 기존 구조와 동일하다
  1. 알림을 보낼 데이터(outbox)를 조회한다
  2. redis에게 락을 최근 3분동안 이 알림이 이미 보내졌는지 확인한다
  3. 보내졌다면 skip, 그렇지 않다면 전송 및 redis에 기록한다



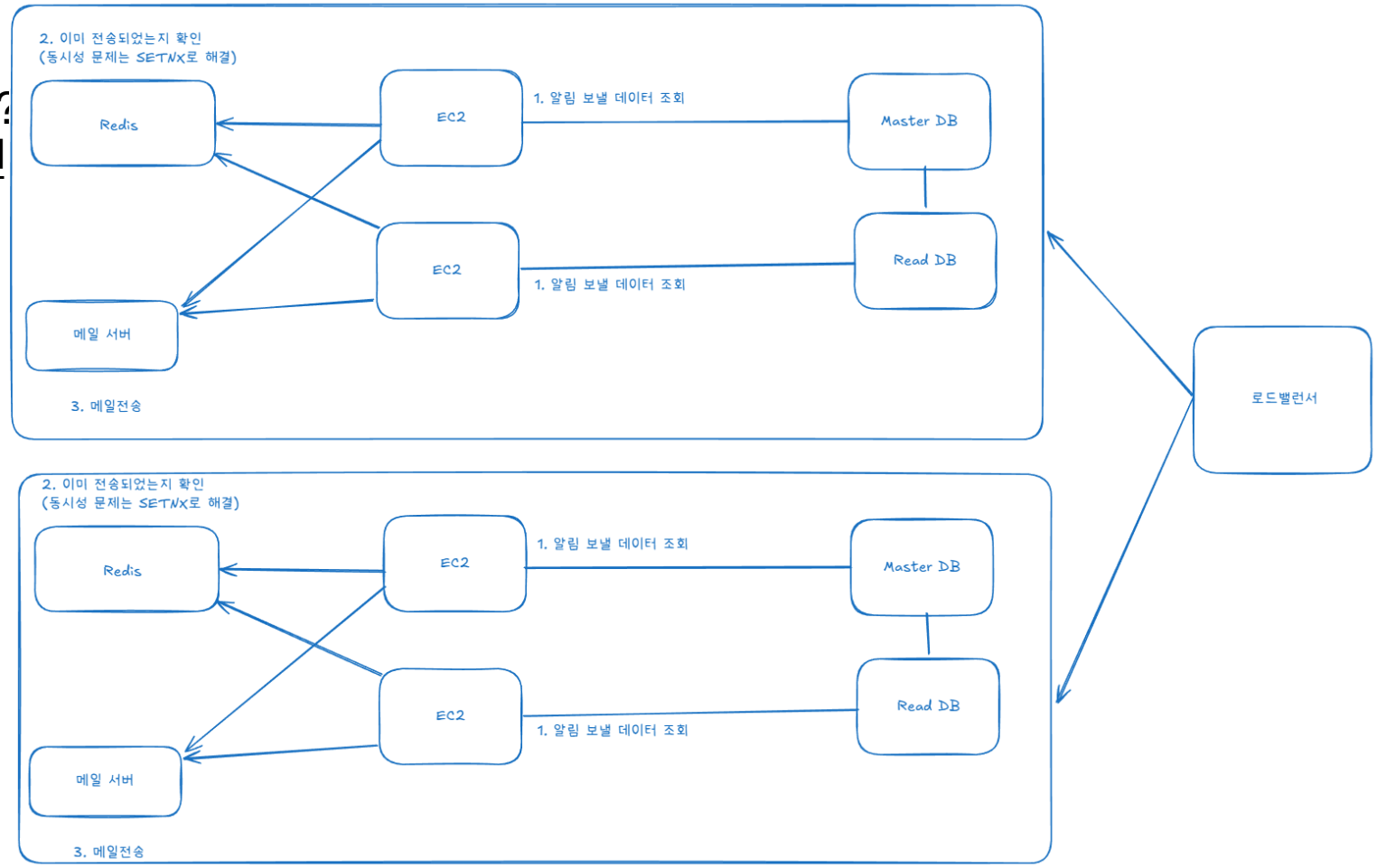
마무리

- 쓰기는 어쩔 수 없이 마스터 DB의 부하가 크다
  - 배치로 완화하자
- 조회는 어느정도의 복제 지연을 허용함으로써 분산 가능해졌다.
  - 레디스로 멱등성을 확보하여 안정성을 향상시켰다.
  - 만약, 레디스가 장애가 나면 어떻게 될까?

- **완벽한 한번만 보장은 없다. 99.9% 정도는 가능할지도..**

더 생각해볼수 있는 것

- 쓰기 분산은 불가능한가?
  - 알림용 구조,데이터베이스를 둔다면?
  - 알림 아키텍쳐 그 자체를 분산한다면



감사합니다