Redis의 Redisson으로 분산락 적용하기

with AOP

김정욱

2024.10.20

목차

1. 왜 분산락이 필요한가?
2. Redis Redisson을 선택한 이유
3. AOP를 적용한 예제

왜 분산락이 필요한가?

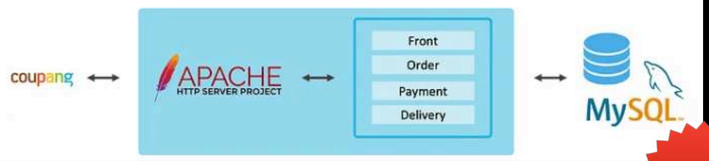
대 MSA의 시대

설립 초기에, 쿠팡은 단 한 개의 서비스 안에 모든 컴포넌트가 존재하는 모놀리식 아키텍처(Monolithic Architecture)로 구성되어 있었습니다. 모놀리식 아키텍처는 쿠팡의 빠른 성장을 뒷받침하기에는 한계를 가지고 있었을 뿐만 아니라 풀기 어려운 다양한 문제들을 야기했습니다.

이를 해결하기 위해 쿠팡은 2013년에 모놀리식 아키텍처로 구성된 서비스를 마이크로서비스 아키텍쳐(Microservices Architecture, MSA)로 전환하는 프로젝트를 진행하였습니다. 이러한 전환을 위해 쿠팡이 취한 전략이 무엇이고, 그 과정에서 나타난 문제를 어떻게 풀어나갔는지 소개하고자 합니다.

. . .

모놀리식 아키텍처



쿠팡 - 마이크로서비스 아키텍처로의 전환

coupang

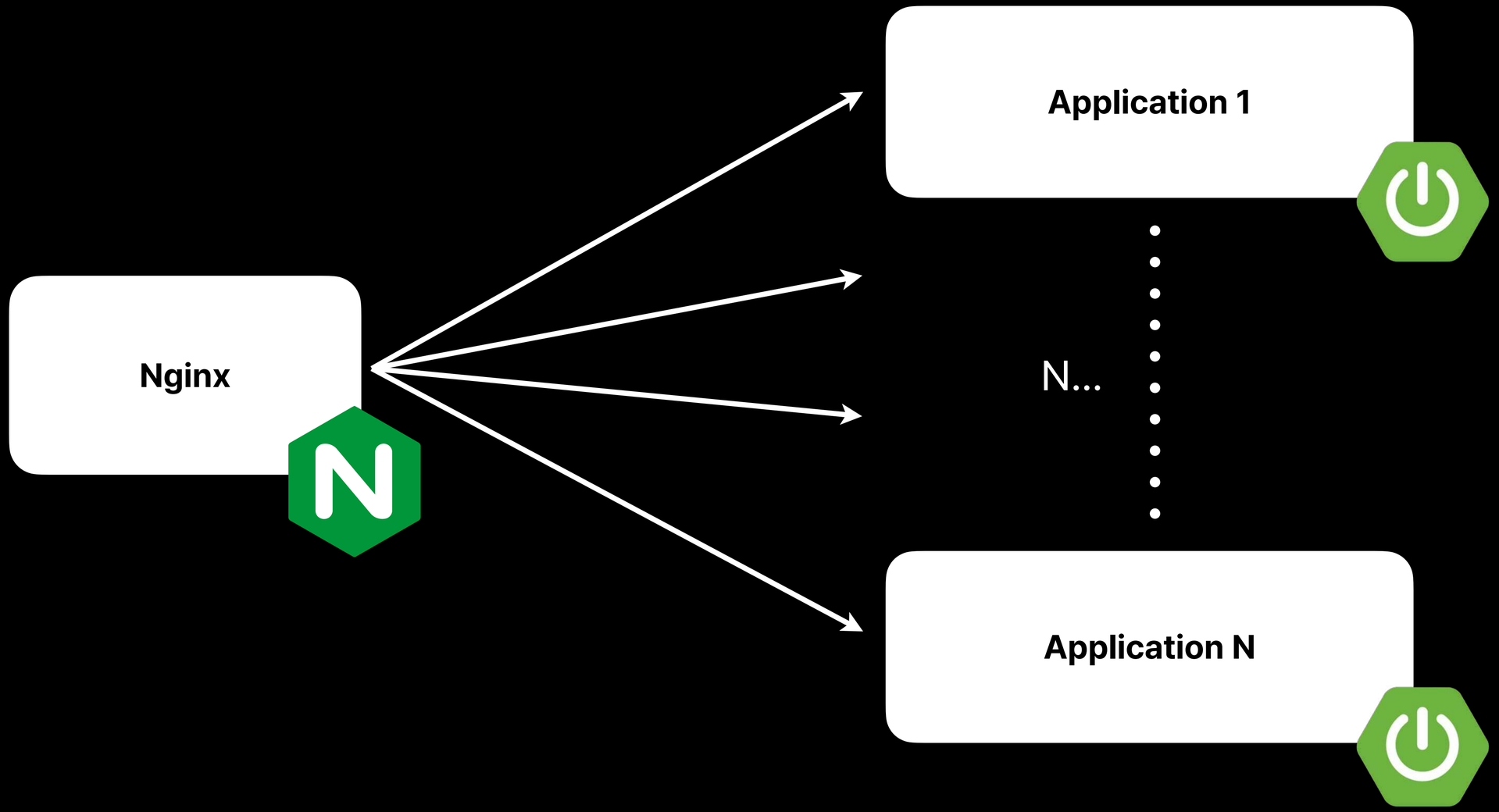
Core Banking

은행 최초 코어뱅킹 MSA 전환기 (feat. 지금 이자 받기)

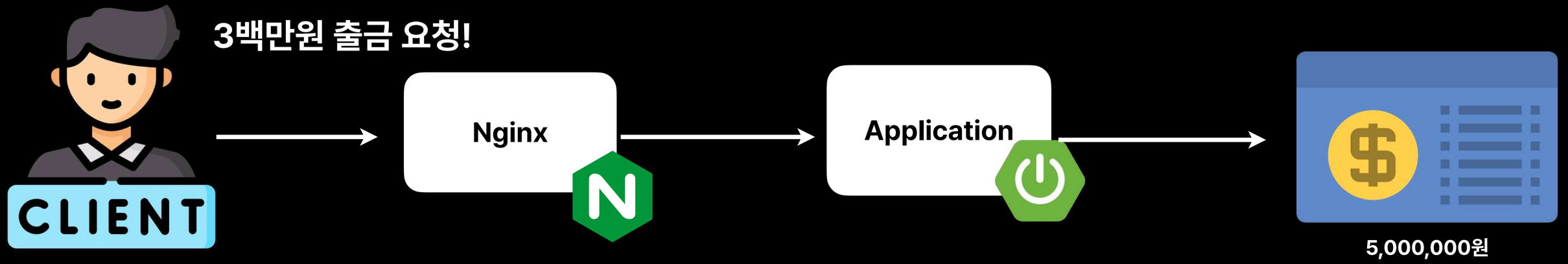
수십 년간 정체되어 있던 전통적인 은행 시스템의 모놀리식 소프트웨어 아키텍처를 MSA로 전환할 수 있을까요? 토스뱅크의 '코어뱅킹 MSA 전환' 사례를 통해 향후 은행 시스템이 나아가야 할 방향을 소개합니다.

Toss Slash 23 - 은행 최초 코어뱅킹 MSA 전환기 (feat. 지금 이자 받기)

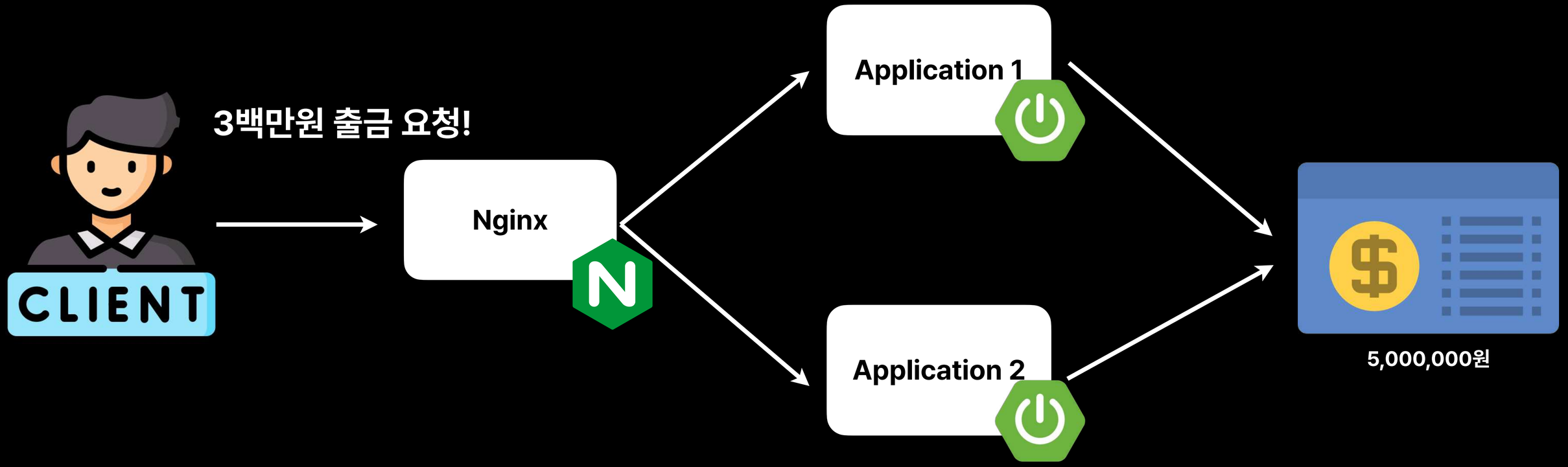
모놀리식 아키텍처 로드밸런싱



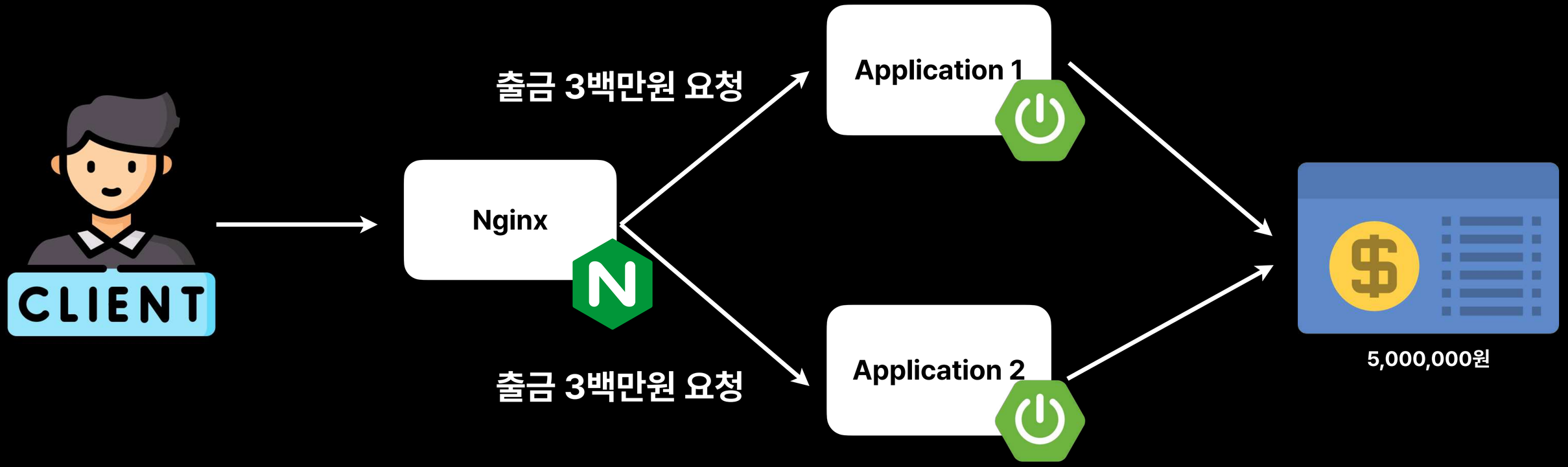
왜 문제가 발생하는가?



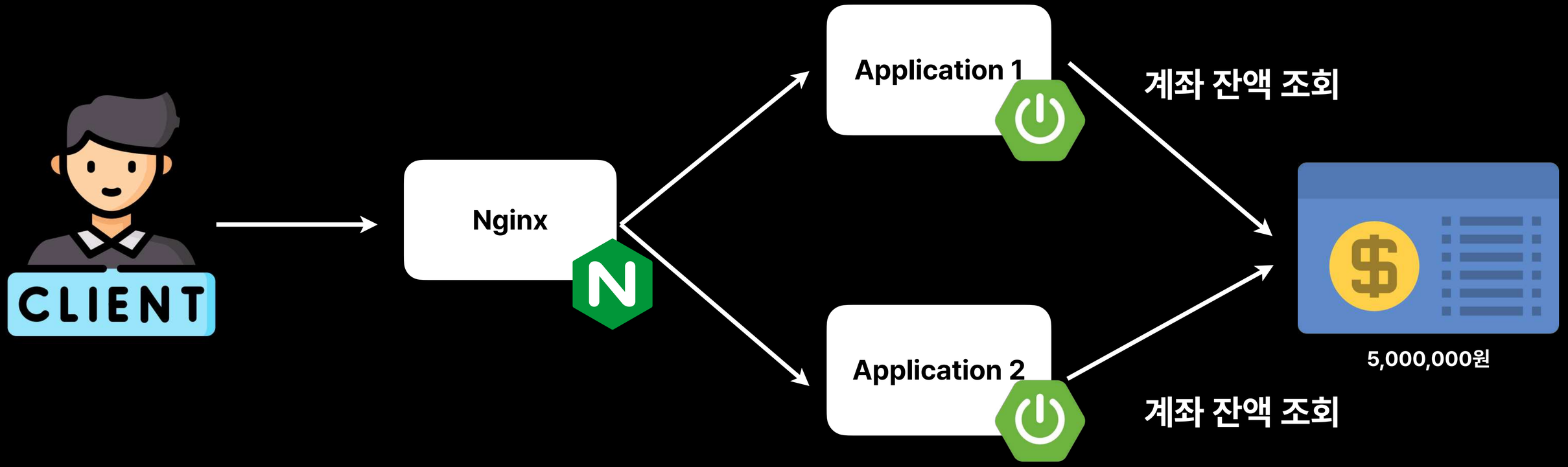
왜 문제가 발생하는가?


3백만원 출금 요청!
CLIENT
Nginx
Application 1
Application 2
5,000,000원

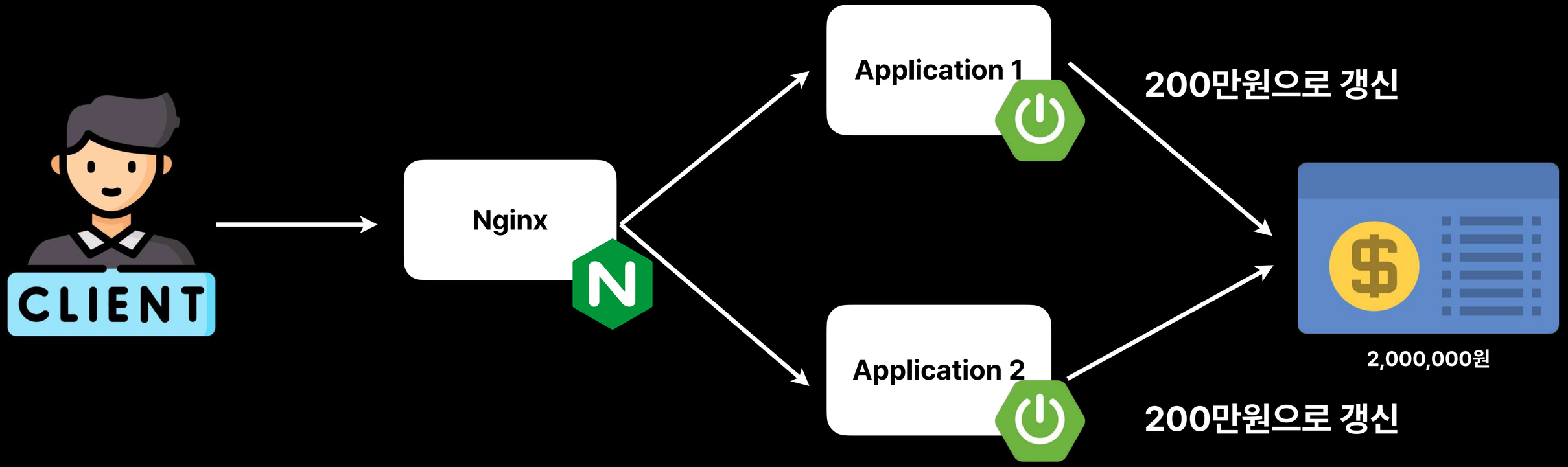
왜 문제가 발생하는가?


출금 3백만원 요청
Application 1
CLIENT
Nginx
N
출금 3백만원 요청
Application 2
5,000,000원

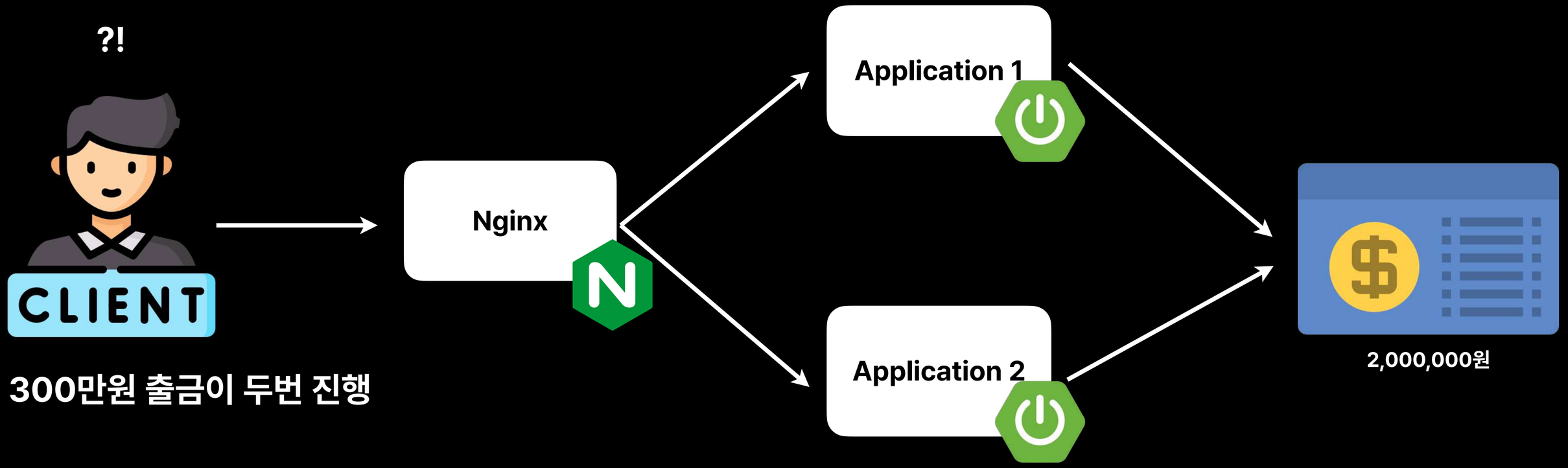
왜 문제가 발생하는가?


CLIENT
Nginx
Application 1
Application 2
계좌 잔액 조회
계좌 잔액 조회
5,000,000원

왜 문제가 발생하는가?

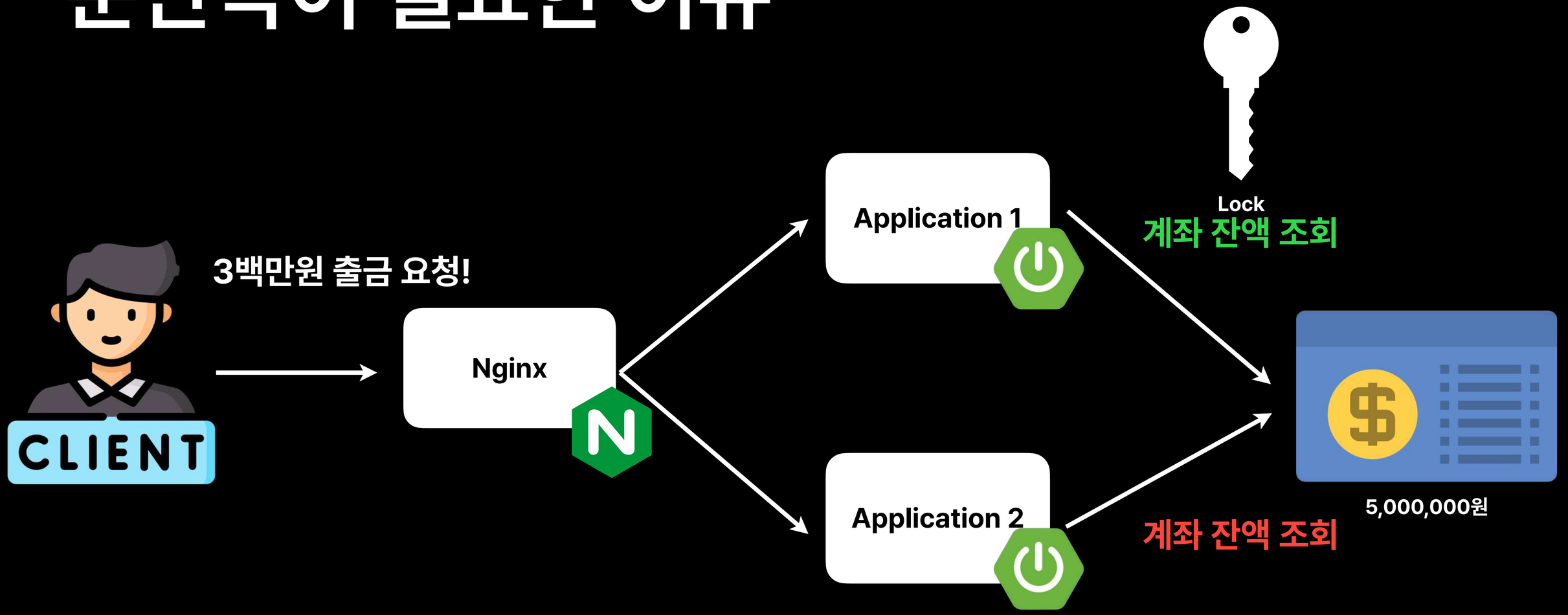

CLIENT
Nginx
Application 1
Application 2
200만원으로 갱신
200만원으로 갱신
2,000,000원

왜 문제가 발생하는가?

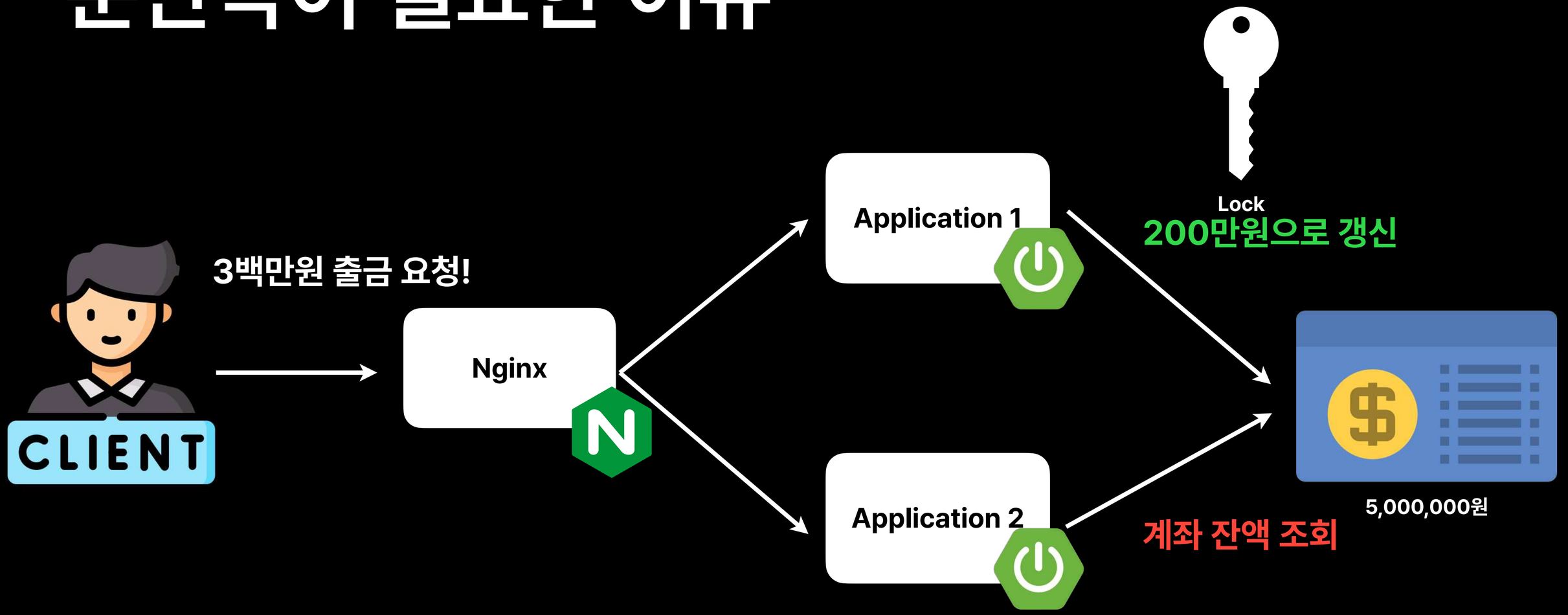

?!
CLIENT
300만원 출금이 두번 진행
Nginx
Application 1
Application 2
2,000,000원

데이터 일관성!

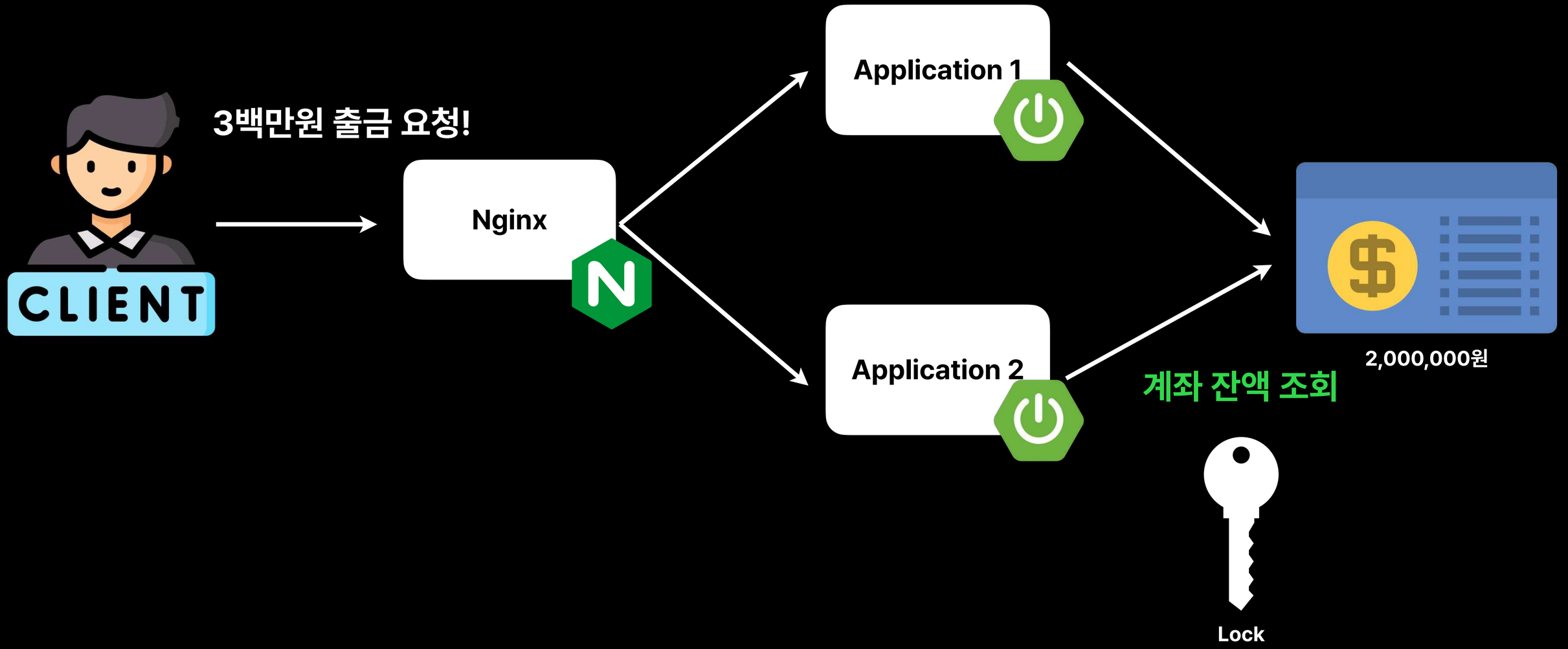
분산락이 필요한 이유


CLIENT
3백만원 출금 요청!
Nginx
Application 1
Application 2
Lock
계좌 잔액 조회
계좌 잔액 조회
5,000,000원

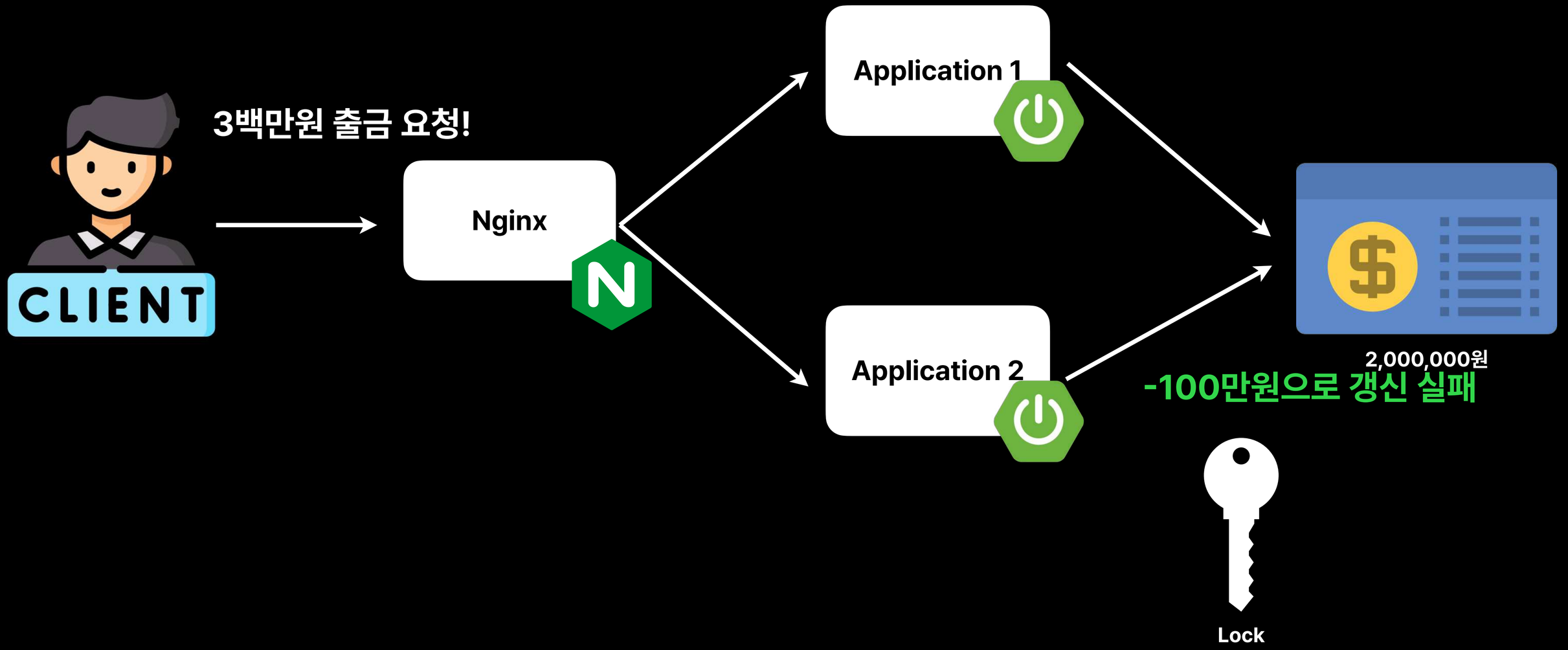
분산락이 필요한 이유


Lock
200만원으로 갱신
Application 1
3백만원 출금 요청!
Nginx
CLIENT
Application 2
계좌 잔액 조회
5,000,000원

분산락이 필요한 이유


3백만원 출금 요청!
CLIENT
Nginx
Application 1
Application 2
2,000,000원
계좌 잔액 조회
Lock

분산락이 필요한 이유


CLIENT
3백만원 출금 요청!
Nginx
Application 1
Application 2
2,000,000원
-100만원으로 갱신 실패
Lock

Redis Redisson을 선택한 이유

분산락 구현 도구

Zookeeper

MySQL

Redis

Lettuce VS Redisson

Lettuce

Redisson

Lettuce

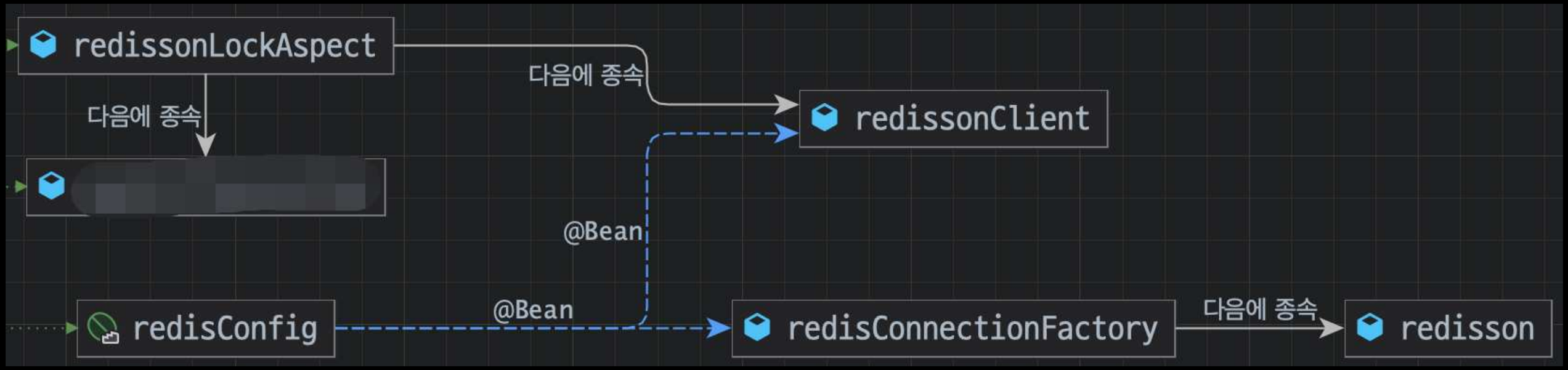
- Spring Data Redis 기본 구현체
- 기본적으로 Spin Lock을 사용
- setnx, setex 등을 이용해 분산락 직접 구현
- Lock에 대한 타임아웃이 없어 Unlock 호출을 하지 못할 경우 Dead Lock 유발

Redisson

- Pub/Sub 방식
- Lock의 lease time 설정 가능
- RLock 인터페이스 제공

AOP를 적용한 예제

구현한 Redis Redisson 구조



build.gradle

build.gradle

```
dependencies {
    /* Spring Boot */
    implementation 'org.springframework.boot:spring-boot-starter-aop'

    /* Redis */
    implementation 'org.redisson:redisson-spring-boot-starter:3.37.0'
}
```

https://mvnrepository.com/artifact/org.redisson/redisson-spring-boot-starter/3.37.0

Redis Config

RedisConfig.java

```java
@Configuration
@ConfigurationProperties(prefix = "spring.data.redis")
public class RedisConfig {
    private static final String REDISSON_HOST_PREFIX = "redis://";
    private String host = "127.0.0.1";
    private int port = 6379;

    @Bean
    public RedissonClient redissonClient() {
        RedissonClient redisson = null;
        Config config = new Config();
        config.useSingleServer().setAddress(REDISSON_HOST_PREFIX + host + ":" + port);
        redisson = Redisson.create(config);
        return redisson;
    }

    @Bean
    public RedissonConnectionFactory redisConnectionFactory(RedissonClient redisson) {
        return new RedissonConnectionFactory(redisson);
    }
}
```

AOP Annotation

key: 락의 이름(식별자)

timeUnit: 락의 시간 단위(초, 분, ...)

waitTime: 락을 기다리는 시간

leaseTime: 락 임대. 시간

RedissonLock

```java
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface RedissonLock {
    /**
     * 락의 이름
     */
    String key();

    /**
     * 락의 시간 단위
     */
    TimeUnit timeUnit() default TimeUnit.SECONDS;

    /**
     * 락을 기다리는 시간 (default - 5s)
     * 락 획득을 위해 waitTime 만큼 대기한다.
     */
    long waitTime() default 5L;

    /**
     * 락 임대 시간 (default - 3s)
     * 락을 획득한 후 leaseTime이 자나면 락을 해제한다.
     */
    long leaseTime() default 3L;
}
```

Aspect Component

RedissonLockAspect

```java
@Aspect
@Component
@RequiredArgsConstructor
public class RedissonLockAspect {

    private static final String REDISSON_LOCK_PREFIX = "LOCK:";

    private final RedissonClient redissonClient;

    @Around(value = "@annotation(com.seikim.redisredlock.RedissonLock)")
    public Object around(ProceedingJoinPoint joinPoint) throws Throwable {
        MethodSignature signature = (MethodSignature)joinPoint.getSignature();
        Method method = signature.getMethod();
        RedissonLock redissonLock = method.getAnnotation(RedissonLock.class);
        String key = REDISSON_LOCK_PREFIX + CustomSpringELParser
                .getDynamicValue(signature.getParameterNames(), joinPoint.getArgs(), redissonLock.key());
        RLock rLock = redissonClient.getLock(key);
        try {
            boolean available = rLock.tryLock(redissonLock.waitTime(), redissonLock.leaseTime(),
                    redissonLock.timeUnit());
            if (!available) {
                return false;
            }
            return joinPoint.proceed();
        } catch (InterruptedException e) {
            throw new InterruptedException();
        } finally {
            if (rLock.isHeldByCurrentThread()) {
                rLock.unlock();
            } else {
                log.info("Redisson Lock Already UnLock ServiceName: {} Key: {}", method.getName(), key);
            }
        }
    }
}
```

Aspect Component

RedissonLockAspect

```java
@Aspect
@Component
@RequiredArgsConstructor
public class RedissonLockAspect {

    private static final String REDISSON_LOCK_PREFIX = "LOCK:";

    private final RedissonClient redissonClient;

    // Method...
}
```

Aspect Component

RedissonLockAspect

```java
@Aspect
@Component
@RequiredArgsConstructor
public class RedissonLockAspect {

    // ETC ...

    @Around(value = "@annotation(com.seikim.redisredlock.RedissonLock)")
    public Object around(ProceedingJoinPoint joinPoint) throws Throwable {
        MethodSignature signature = (MethodSignature)joinPoint.getSignature();
        Method method = signature.getMethod();
        RedissonLock redissonLock = method.getAnnotation(RedissonLock.class);
        String key = REDISSON_LOCK_PREFIX + CustomSpringELParser
                .getDynamicValue(signature.getParameterNames(), joinPoint.getArgs(), redissonLock.key());
        RLock rLock = redissonClient.getLock(key);
        // ETC ...
    }
}
```

Aspect Component

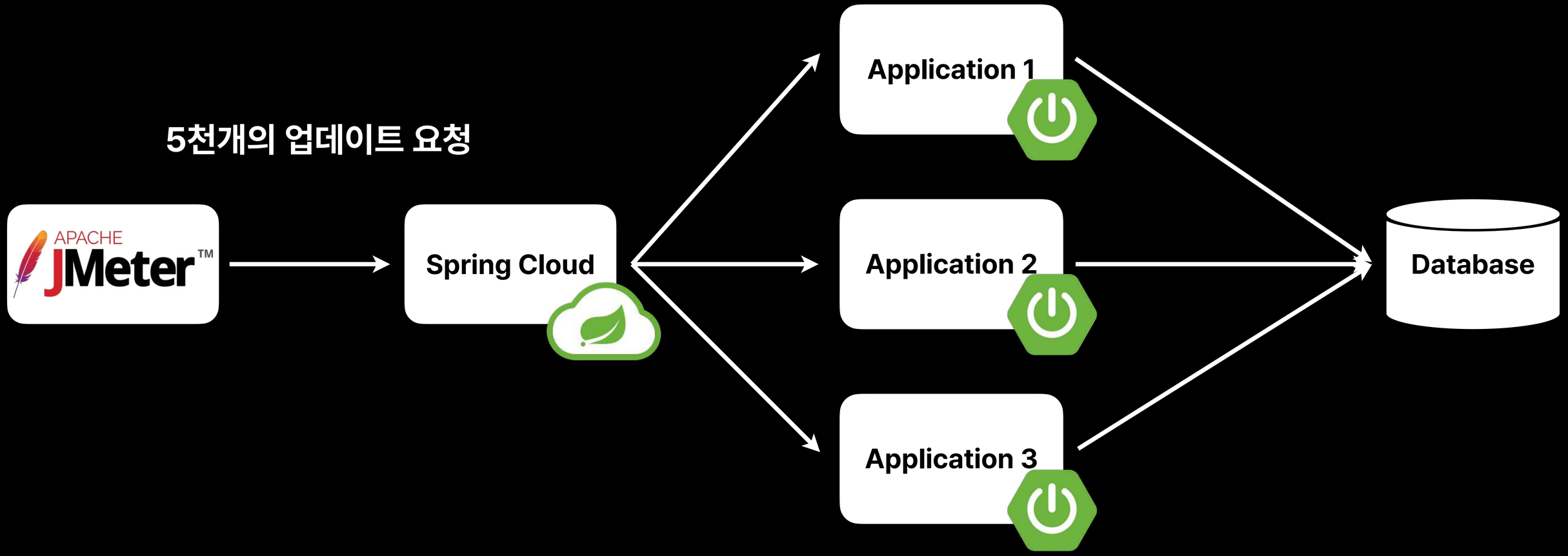
RedissonLockAspect

```java
@Aspect
@Component
@RequiredArgsConstructor
public class RedissonLockAspect {

    // ETC ...

    @Around(value = "@annotation(com.seikim.redisredlock.RedissonLock)")
    public Object around(ProceedingJoinPoint joinPoint) throws Throwable {
        // ETC ...
        RLock rLock = redissonClient.getLock(key);
        try {
            boolean available = rLock.tryLock(redissonLock.waitTime(), redissonLock.leaseTime(),
                    redissonLock.timeUnit());
            if (!available) {
                return false;
            }
            return joinPoint.proceed();
        } catch (InterruptedException e) {
            throw new InterruptedException();
        } finally {
            if (rLock.isHeldByCurrentThread()) {
                rLock.unlock();
            } else {
                log.info("Redisson Lock Already UnLock ServiceName: {} Key: {}", method.getName(), key);
            }
        }
    }
}
```

예제 시나리오


5천개의 업데이트 요청
APACHE
JMeter™
Spring Cloud
Application 1
Application 2
Application 3
Database

락 없이 동작

MemberService

```java
@Transactional
@Override
public int addLikeCount(final int memberId) {
    Member findMember = MemberServiceUtils.findById(memberRepository, memberId);
    findMember.addLikeCount();
    return findMember.getLikeCount();
}
```

락 없이 동작

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Add Like Reque...	5000	6	3	132	3.59	0.00%	247.8/sec	31.37	54.69	129.6
TOTAL	5000	6	3	132	3.59	0.00%	247.8/sec	31.37	54.69	129.6


```
{
    "createdAt": "2024-10-18T13:23:49.171982",
    "modifiedAt": "2024-10-18T13:29:00.348224",
    "deletedAt": null,
    "id": 1,
    "email": "workju1124@gmail.com",
    "username": "김세이",
    "profileImage": null,
    "introduction": null,
    "likeCount": 3401,
    "delete": false,
    "notDelete": true
}
```


likeCount → 3,401개

락 설정 후 동작

MemberService

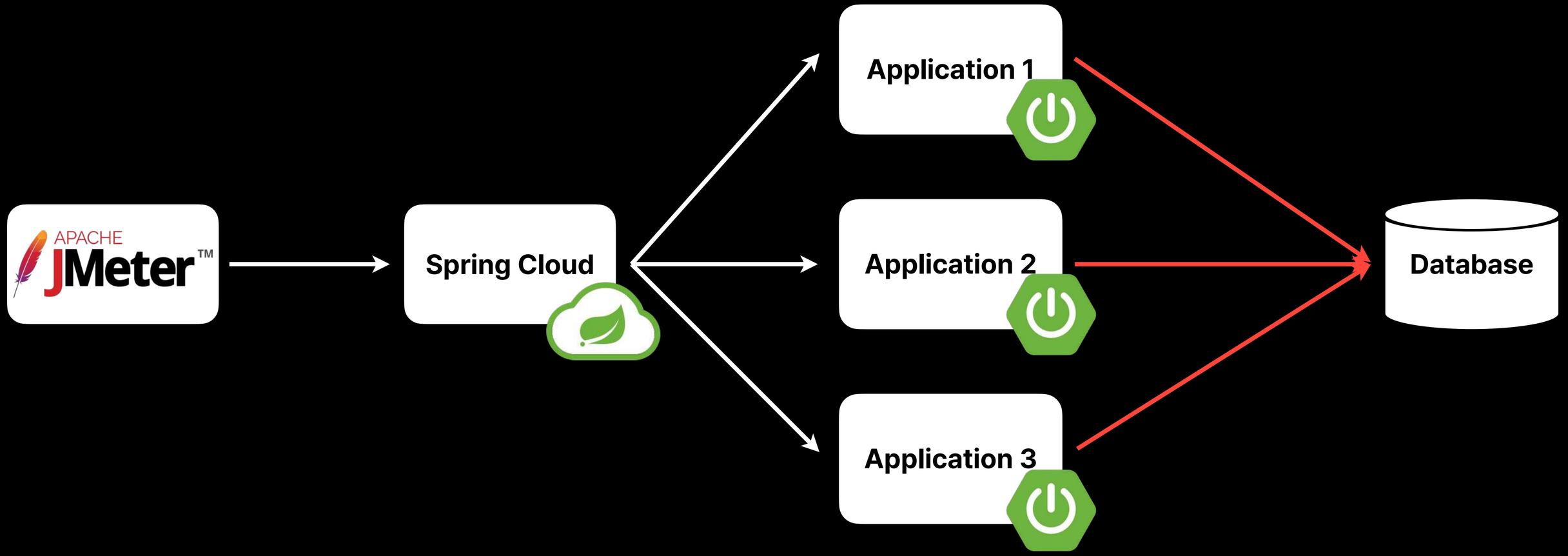
```java
@RedissonLock(key = "#memberId")
@Transactional
@Override
public int addLikeCount(final int memberId) {
    Member findMember = MemberServiceUtils.findById(memberRepository, memberId);
    findMember.addLikeCount();
    return findMember.getLikeCount();
}
```

락 설정 후 동작

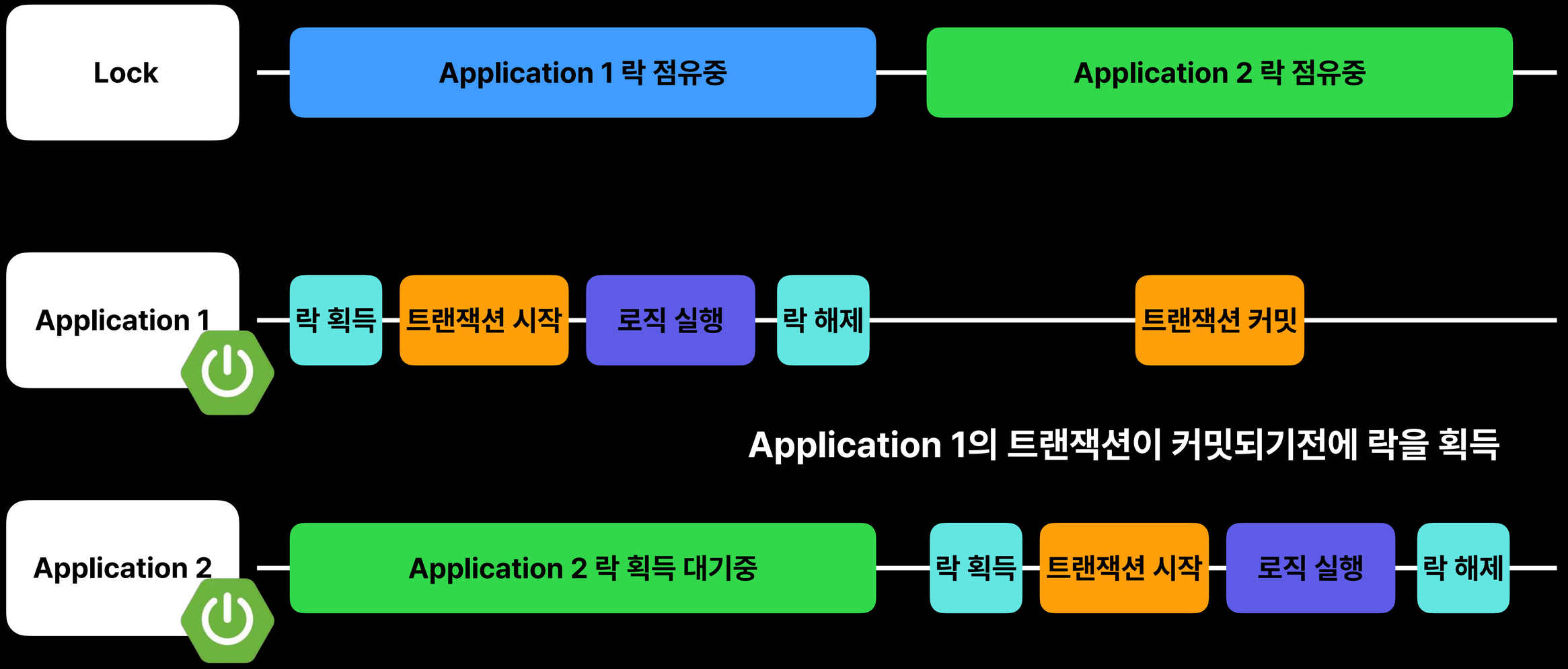

```
{
    "createdAt": "2024-10-18T14:54:26.248219",
    "modifiedAt": "2024-10-18T14:54:40.639505",
    "deletedAt": null,
    "id": 1,
    "email": "workju1124@gmail.com",
    "username": "김세이",
    "profileImage": null,
    "introduction": null,
    "likeCount": 1620,
    "notDelete": true,
    "delete": false
}
```


likeCount → 1,620개??

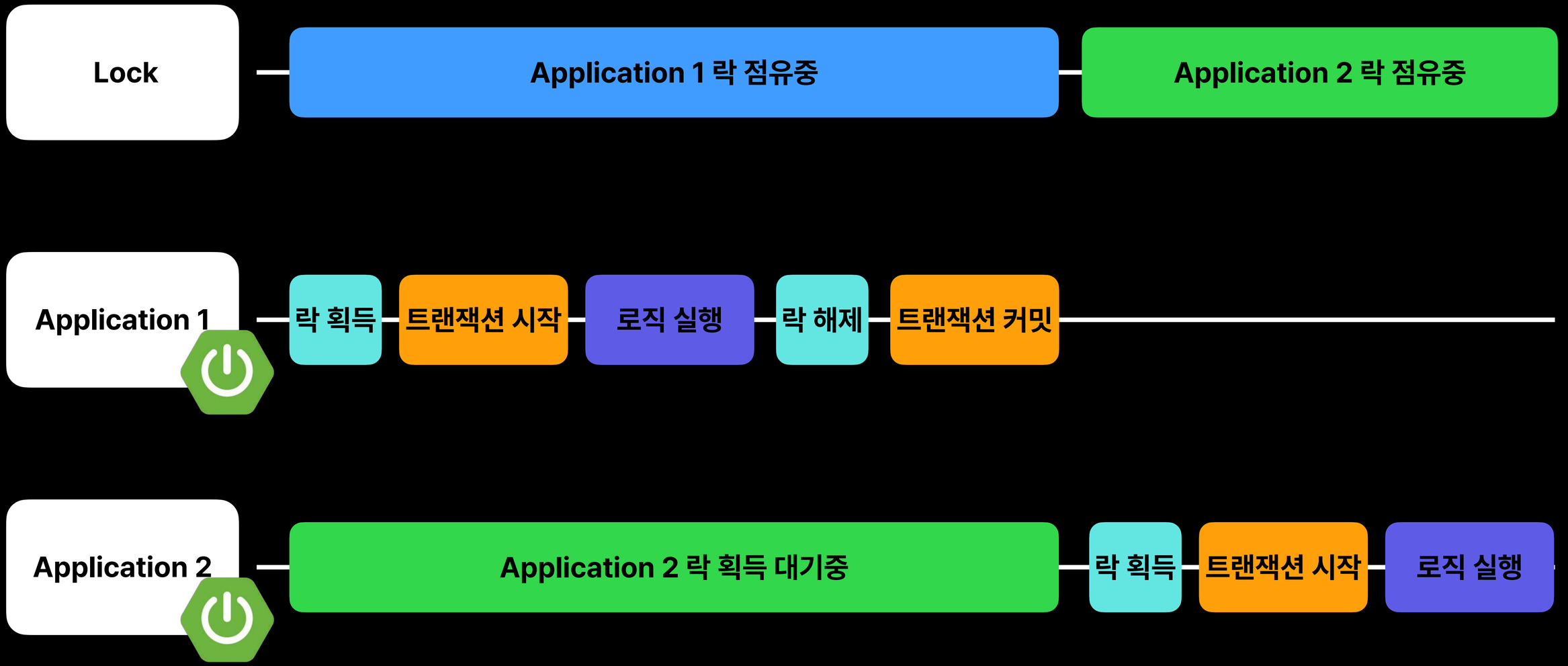
이유는?


APACHE
JMeter™
Spring Cloud
Application 1
Application 2
Application 3
Database

이유는?


Lock
Application 1 락 점유중
Application 2 락 점유중
Application 1
락 획득
트랜잭션 시작
로직 실행
락 해제
트랜잭션 커밋
Application 1의 트랜잭션이 커밋되기전에 락을 획득
Application 2
Application 2 락 획득 대기중
락 획득
트랜잭션 시작
로직 실행
락 해제

해결법은?


Lock
Application 1 락 점유중
Application 2 락 점유중
Application 1
락 획득
트랜잭션 시작
로직 실행
락 해제
트랜잭션 커밋
Application 2
Application 2 락 획득 대기중
락 획득
트랜잭션 시작
로직 실행

AOP For Transaction

AopForTransaction

```java
@Component
public class AopForTransaction {
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public Object proceed(final ProceedingJoinPoint joinPoint) throws Throwable {
        return joinPoint.proceed();
    }
}
```

AOP For Transaction

AopForTransaction

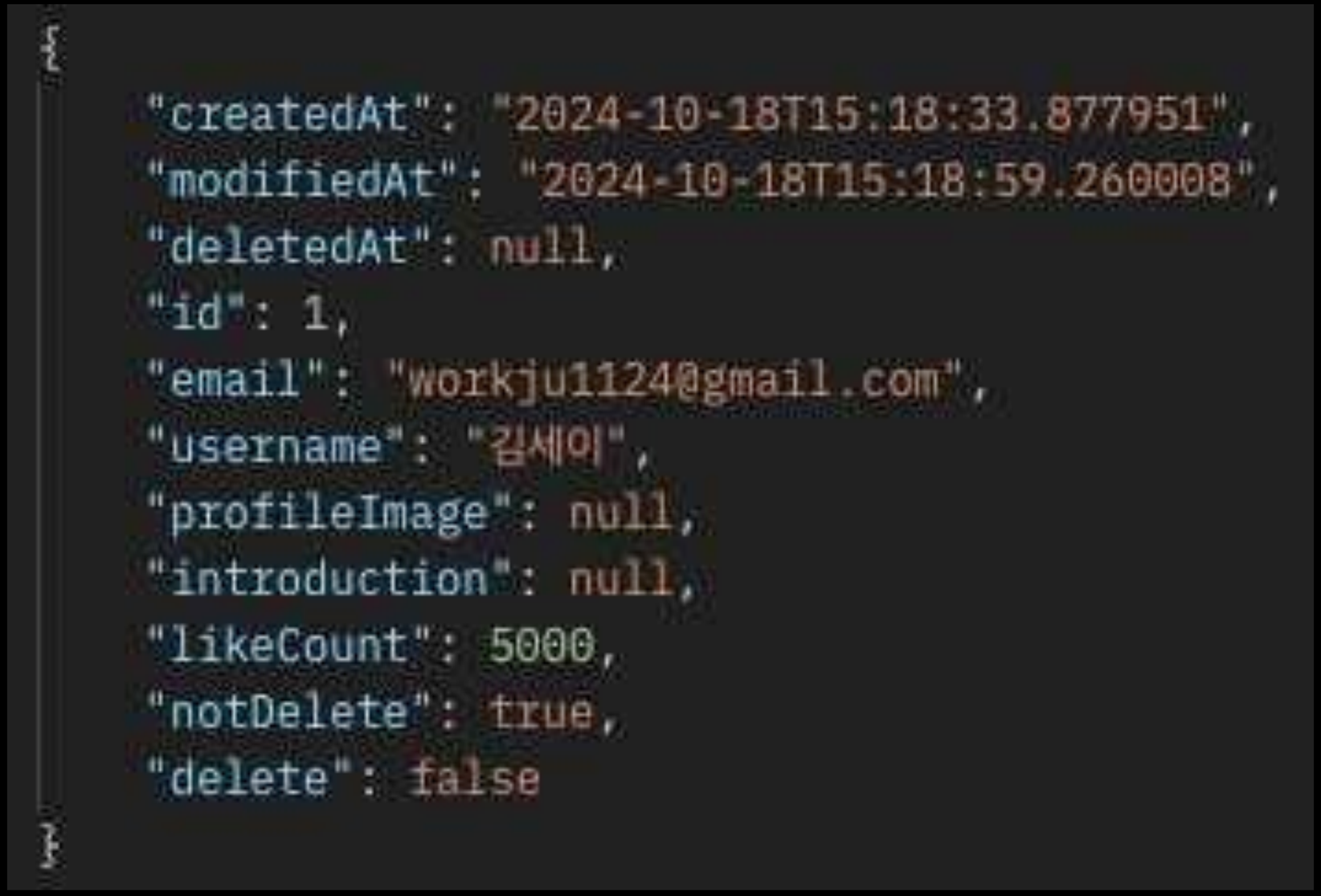
```java
@Aspect
@Component
@RequiredArgsConstructor
public class RedissonLockAspect {

    private static final String REDISSON_LOCK_PREFIX = "LOCK:";

    private final RedissonClient redissonClient;
    private final AopForTransaction transaction; // 추가

    @Around(value = "@annotation(com.seikim.redisredlock.RedissonLock)")
    public Object around(ProceedingJoinPoint joinPoint) throws Throwable {
        // ETC ...
        try {
            boolean available = rLock.tryLock(redissonLock.waitTime(), redissonLock.leaseTime(),
                    redissonLock.timeUnit());
            if (!available) {
                return false;
            }
            return transaction.proceed(joinPoint); // 수정
        } catch (InterruptedException e) {
            // ETC...
        }
    }
}
```

AOP For Transaction



Redis Redisson + Spring AOP

Ref

- https://helloworld.kurly.com/blog/distributed-redisson-lock/
- https://velog.io/@hkyo96/Redis-분산락을-이용한-동시성-문제-해결
- https://velog.io/@profoundsea25/Spring에서-Redis-분산-락-적용하기-Redisson-사용

Thanks for Watching