대규모 엔터프라이즈 시스템 개선 경험기

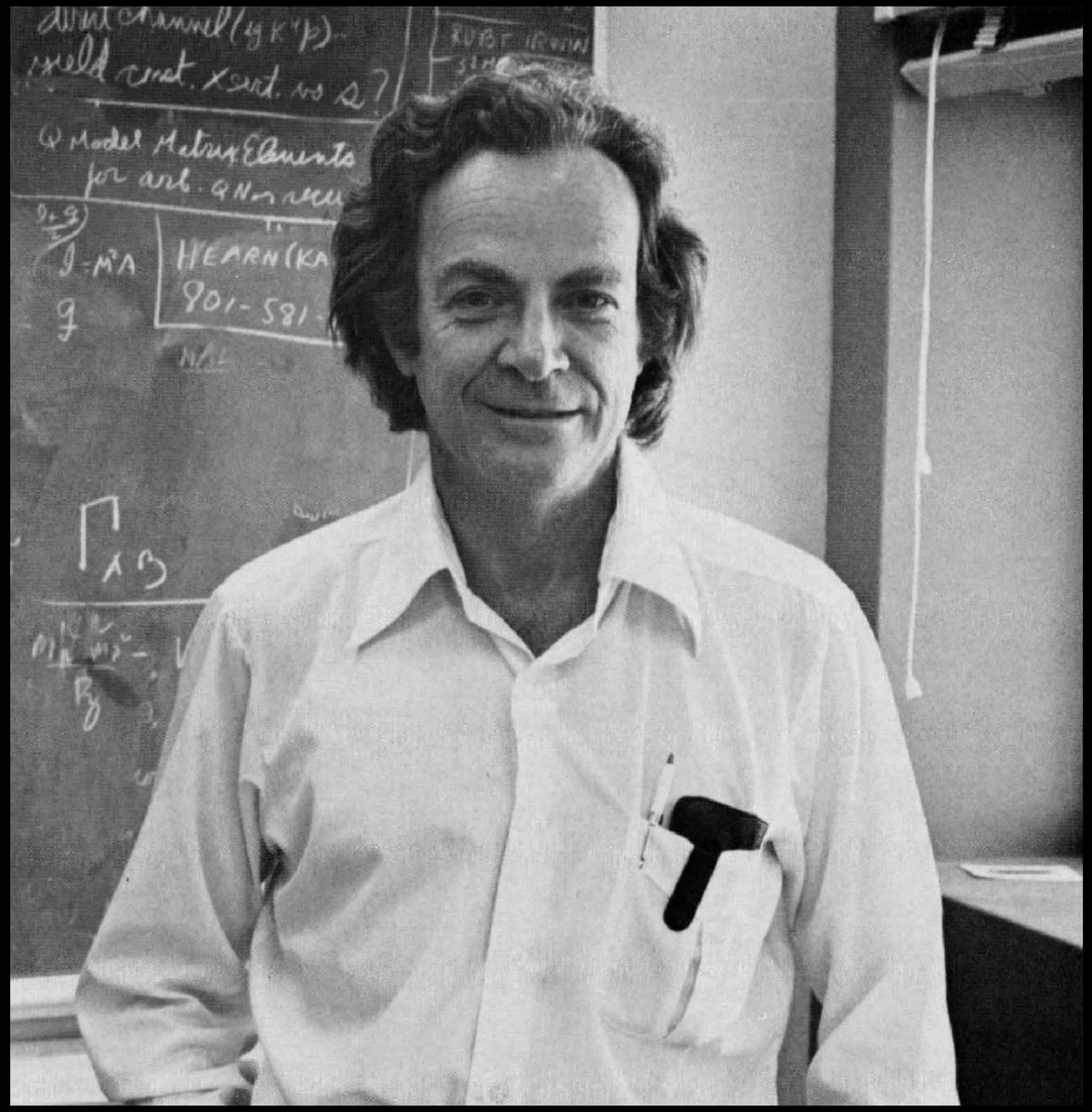
1부 달리는 기차의 바퀴 갈아 끼우기

임형태, 2023-04-22

개발자에게 개발이란?

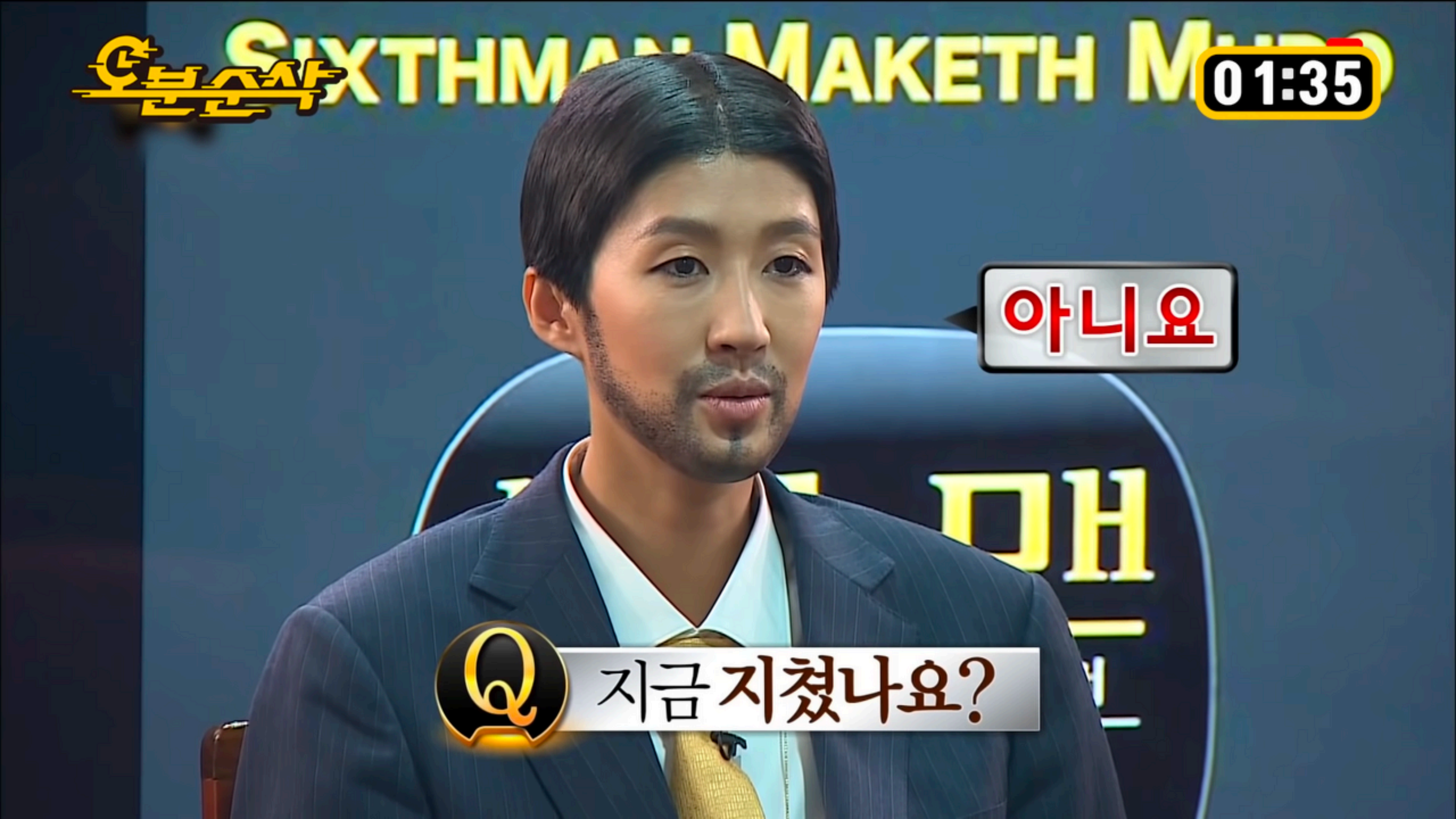
The Feynman Problem-Solving Algorithm

1. write down the problem
2. think very hard
3. write down the answer.




나
기능구현
장애


나
리팩토링
장애


오늘은 선생
SIXTHMAN MAKETH MUDO
01:35
아니요
Q
지금 지쳤나요?

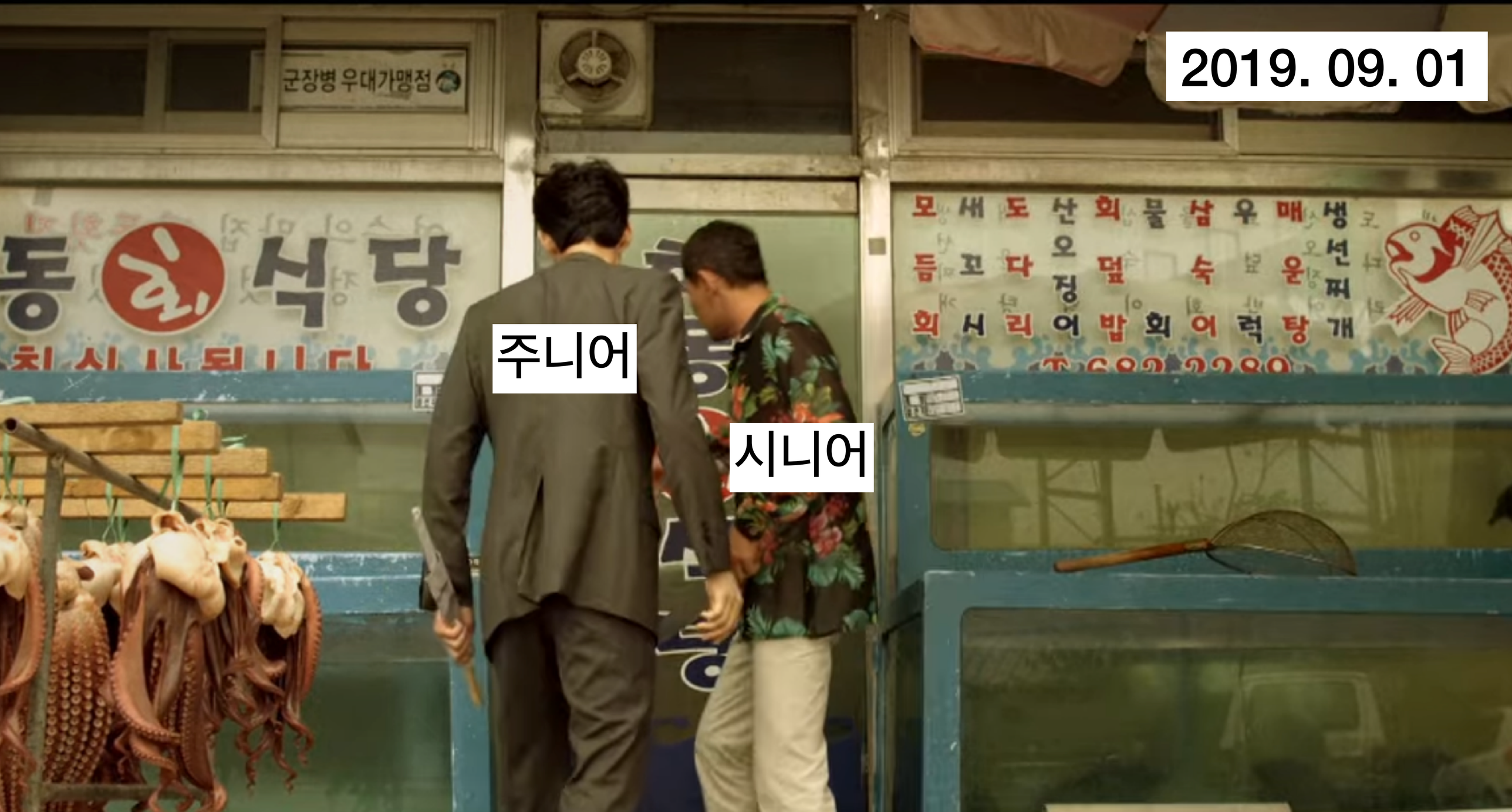
두려움

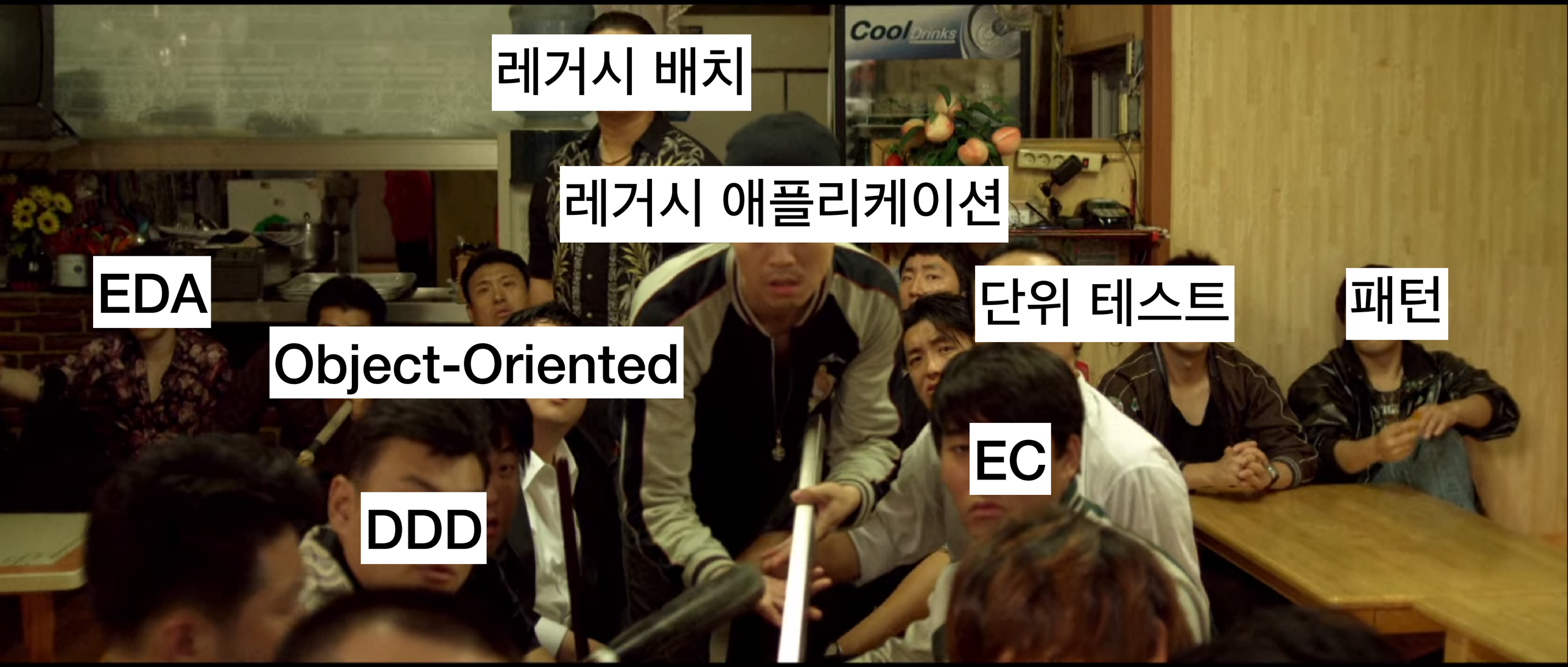
매 순간 개발자가 극복해야 하는 것

“유지보수성이 목표다.
여기서 유지보수성이란 **두려움** 없이, **주저함** 없이,
저항감 없이 코드를 변경할 수 있는 **능력**을 말한다.”

조영호, 오브젝트: 코드로 이해하는 객체지향 설계


기획팀
요구사항
개발자
레거시
시스템


2019. 09. 01
군장병 우대가맹점
식당
주니어
시니어


레거시 배치
레거시 애플리케이션
EDA
단위 테스트
패턴
Object-Oriented
EC
DDD

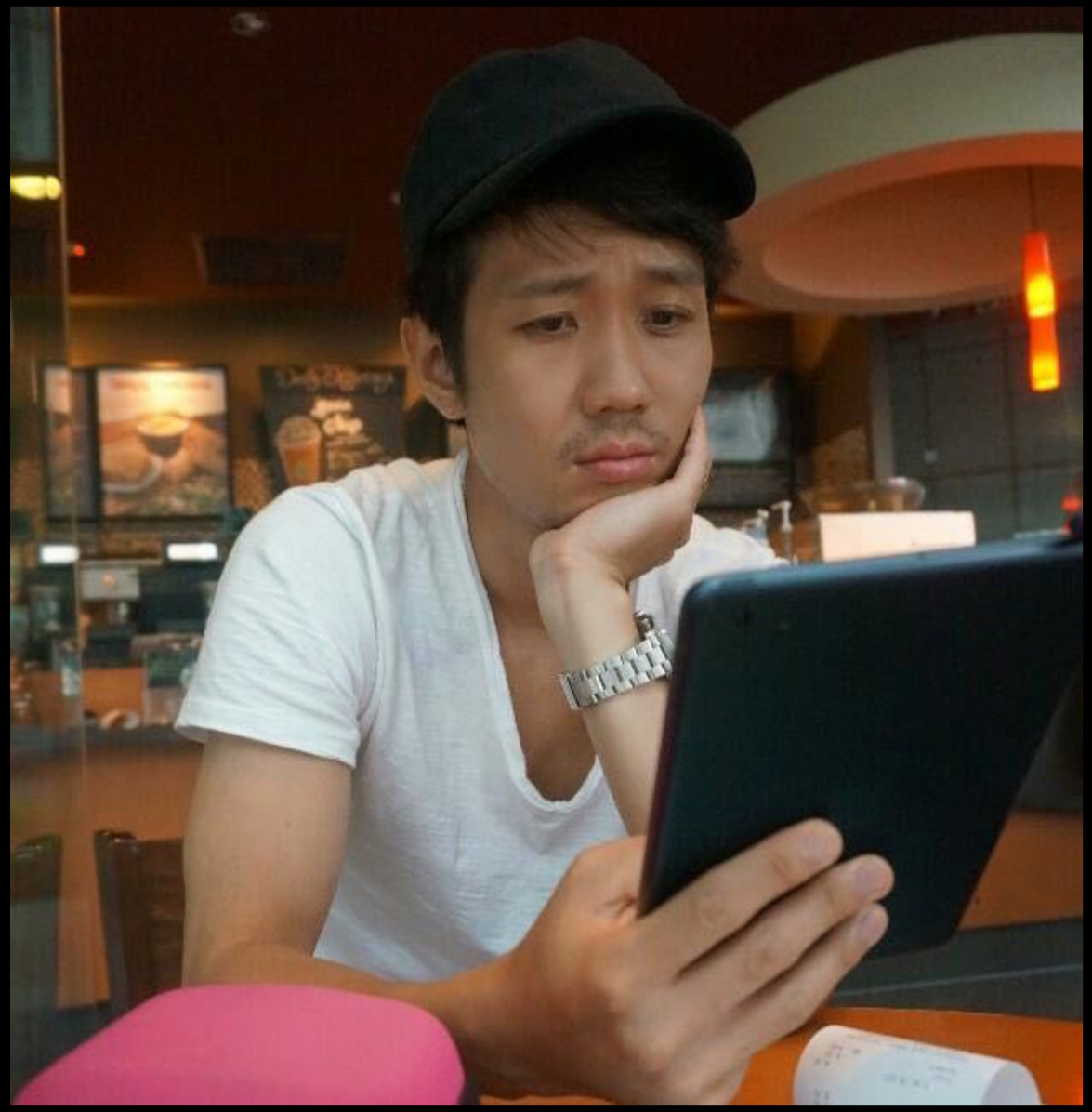
용어 설명

구글과 ChatGPT 에게 물어보세요

임형태 (PL;CO)

- 네이버 **쇼핑카탈로그** 개발
- 11번가(SK플래닛) **페이먼트** 개발
- 이스트소프트 **알약서버** 개발

- KSUG 일꾼단



placebo.coder@gmail.com

대규모 엔터프라이즈 시스템 개선 경험기

1부 달리는 기차의 바퀴 갈아 끼우기

- 전통적인 시스템의 결합도
- Refactoring VS Rewriting
- 새로운 코드 : 새 술, 새 부대에
- 새로운 시스템으로의 이동
- 품질과 끈기

전통적인 시스템에서의 높은 결합도

데이터베이스 공유

서비스와 소프트웨어 진화의 벽

- SW 구현과 관리의 효율
  → **무결성과 공유의 책임**을 DB에 위임
- 서비스, SW, 조직이 성장
  → **예측비용**이 증가
- **특이점**을 넘어선 경우
  → 기술 부채 **급증**

“소프트웨어 구조는 해당 소프트웨어를 개발한
조직의 커뮤니케이션 구조를 닮게 된다.”

콘웨이의 법칙(Conway's law)

행위 중복

코드 중복의 오해

- 소스코드 중복 = 기술 부채
- 소스코드 1 개 & API N 개
  : 이메일 유효성 검사 1 개 & 이메일 입력 API 5 개
- 소스코드 공유 ≒ 기술 부채 회피

“중복 코드를 추상화하여 끄집어내는 것은 **DRY (Don't Repeat Yourself)** 를 사용하는 좋은 출발점이지만, 그것이 전부는 아닙니다. 여러분이 중복 코드를 피하려고 하는 것은 사실 각 **기능과 요구사항을 한 번만 구현**하려고 노력하는 것입니다.”

브렛 맥래프린, 게리 폴리스, 데이빗 웨스트, 헤드 퍼스트 **Object-Oriented Analysis & Design**

데이터 무결성 관리 비용

서비스 성장과 무결성

- 데이터 무결성 = 서비스 품질
- 데이터 무결성을 DB 위임
  → 일정 규모 이하에서는 개발 효율 부과
- 서비스 성장과 행위 중복 증가
  → 데이터 무결성 유지, 관리 비용 증가

Refactoring VS Rewriting

더 이상 **유지보수 되지 않는** 개발 도구와
비지니스 코드와의 결합도가 높음

리팩토링을 위한 **최소한**의 테스트 작성과
실행 비용, 시간, 범위도 **예측할 수 없음**

애플리케이션 **간의** 결합도를 개선하기 위해 **새롭게 추상화** 된 구조와 모델이 필요

오로지 개발팀만의 노력

외부의 이해와 지원없이 합니다


And winter is coming.
겨울이 오고 있어요

FIG APPLICATON

Layered Architecture

Port & Adapter Pattern

Modular Monolith

FIG 애플리케이션은
서비스 **요구사항 달성**이 목적
+ 우리 개발자들의 요구사항

요구사항 해결을 위한
좋은 설계 품질을 가지고 있어야 함


그것이 알고싶다

실행 가능한
서비스 과제가 시작될 때


COMING
SOON...
makeameme.org

새로운 시스템으로의 이동

차세대 프로젝트

BIGBANG (Cut-over) 프로젝트

차세대 프로젝트

내 주변에서 빅뱅이 시작되기 어려운 이유

- 요구사항은 **지속적으로 지원**해야 하며
  이는 **SW 품질보다 우선**하여 예산과 인력을 사용
- 시스템에서 **일정 규모 이상을 변경**하기 위해서는
  대규모의 인력과 많은 시간을 소비
- 저수준의 SW 품질로 인한 **고통은**
  기획팀, 사업팀이 아닌 **개발팀의 몫**이다.

다른 **전략**이 필요

Strangler Pattern

Strangler Fig Application

FIG 1.

Tree

FIG 2.

Strangler Branch

FIG 3.

Strangled Tree

FIG 4.

Strangler Fig

“The most important reason to consider a strangler fig application over a cut-over rewrite is reduced risk.”

Martin Fowler, StranglerFigApplication


이게 교살자 패턴이야
이제 우리 이제 레거시
탈출 가능한거지??
레거시
탈출 가능한거지?

Object Oriented

Eventual **Consistency**

Event Driven Architecture

Domain Driven **Design**

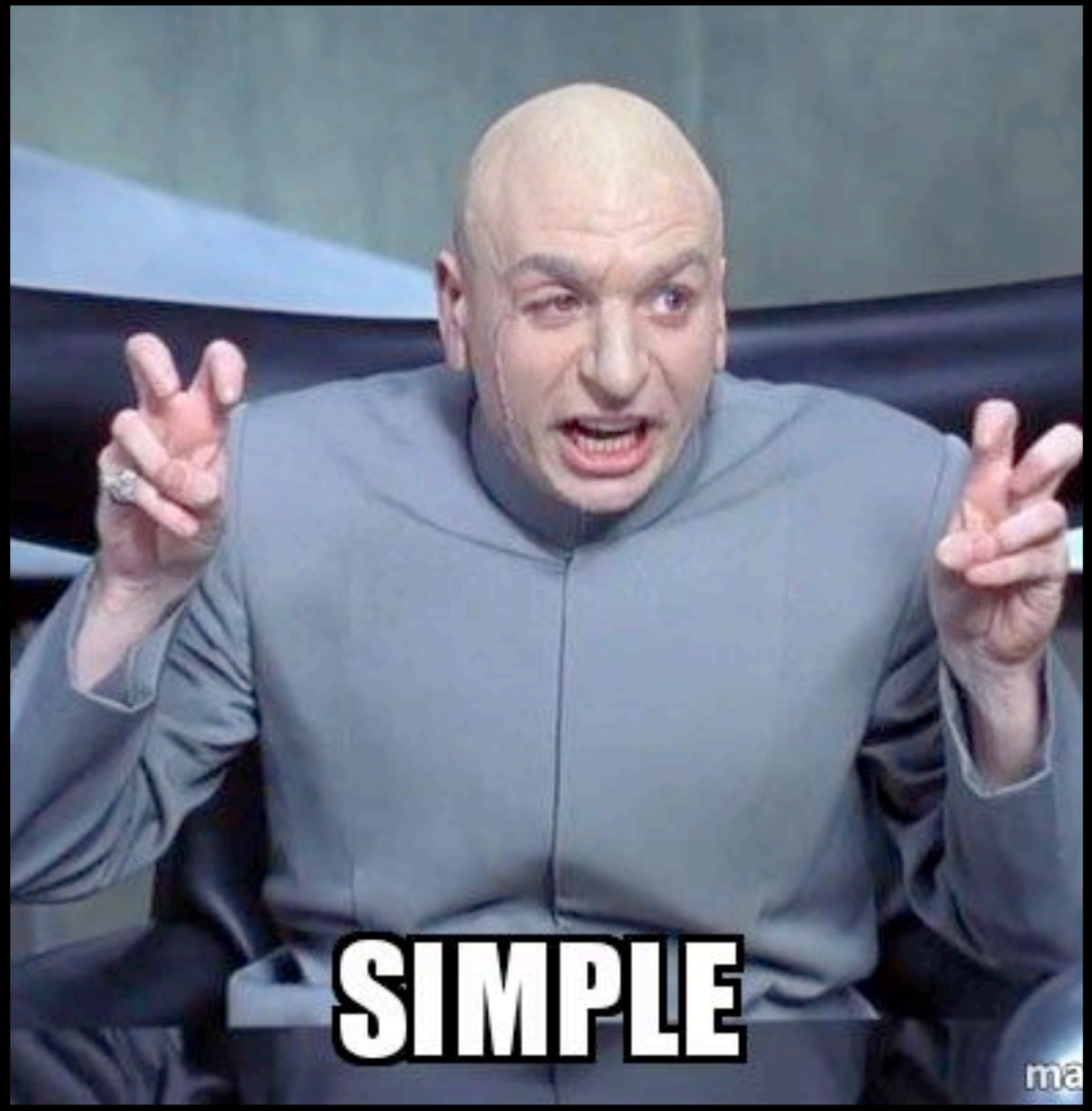
단순함

Keep **I**t **S**imple **S**tupid

단순함 ≠ 쉽다

개념의 부재와 단순함

Object-Oriented
& Domain Driven Design



개념의 부재

일관성의 필요

- OO 와 DDD 는 소프트웨어 설계에 일관성과 맥락을 부여
- 일관성과 맥락은 추상화 기본 재료
- 추상화로부터 얻은 모델은 주요 관심사
- 모델은 소프트웨어에 단순함을 부여

개념의 부재

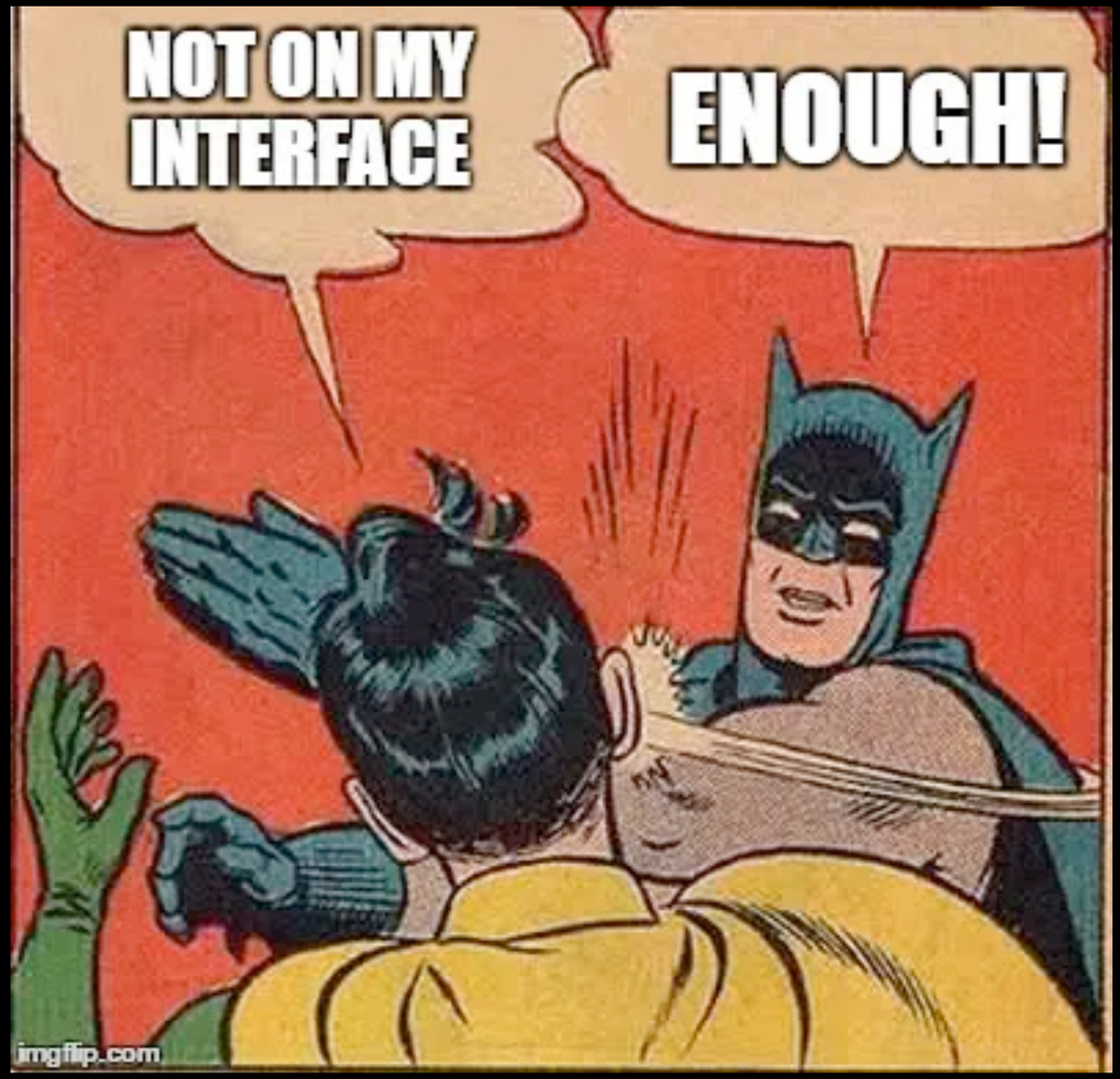
일관성의 필요

체계적이고 **논리적인** 과정에서
하나씩 **발견**해나가야 함

- 모델은 소프트웨어에 **단순함**을 부여

인터페이스와 이벤트

Event Driven Architecture & Event Consistency



인터페이스란?

“일반적으로 인터페이스란 어떤 **두 사물**이 마주치는 경계 지점에서 **서로 상호작용**할 수 있게 이어주는 방법이나 장치를 의미한다.”

조영호, 객체지향의 사실과 오해: 역할, 책임, 협력 관점에서 본 객체지향

이벤트란?

“An event can be defined as a significant change in state”

Wikipedia, Event-driven architecture, K. Mani Chandy Event-driven Applications, California Institute of Technology, 2006


이 뭔
소리야?

인터페이스로 메시지를 교환

≠

이벤트로 메시지를 전파

응답자

인터페이스

요청자

“subscribers express interest in one or more classes and only receive messages that are of interest, **without knowledge of which publishers**, if any, there are.”

Wikipedia, Publish–subscribe pattern

무결성과 Eventual Consistency

적절한 데이터 무결성 관리

- **모델, 개체(Entity) 단위**의 이벤트로 데이터 무결성 유지
- **일관된 단위**로 데이터 무결성 관리
  → 무결성 구현, 관리 **절차에 단순함**
- 무결성 단위 내에서의 **느슨한** 결합도
  → **빈약해진** 데이터 무결성
- 구현, 관리의 **단순함**과 무결성 **강도** 간의 **트레이드 오프**

무결성과 Eventual Consistency

적절한 데이터 무결성 관리

- **모델, 개체(Entity) 단위**의 이벤트로 데이터 무결성 유지
- **일관된 단위**로 데이터 무결성 관리

다양하고 풍부한 이벤트로 극복

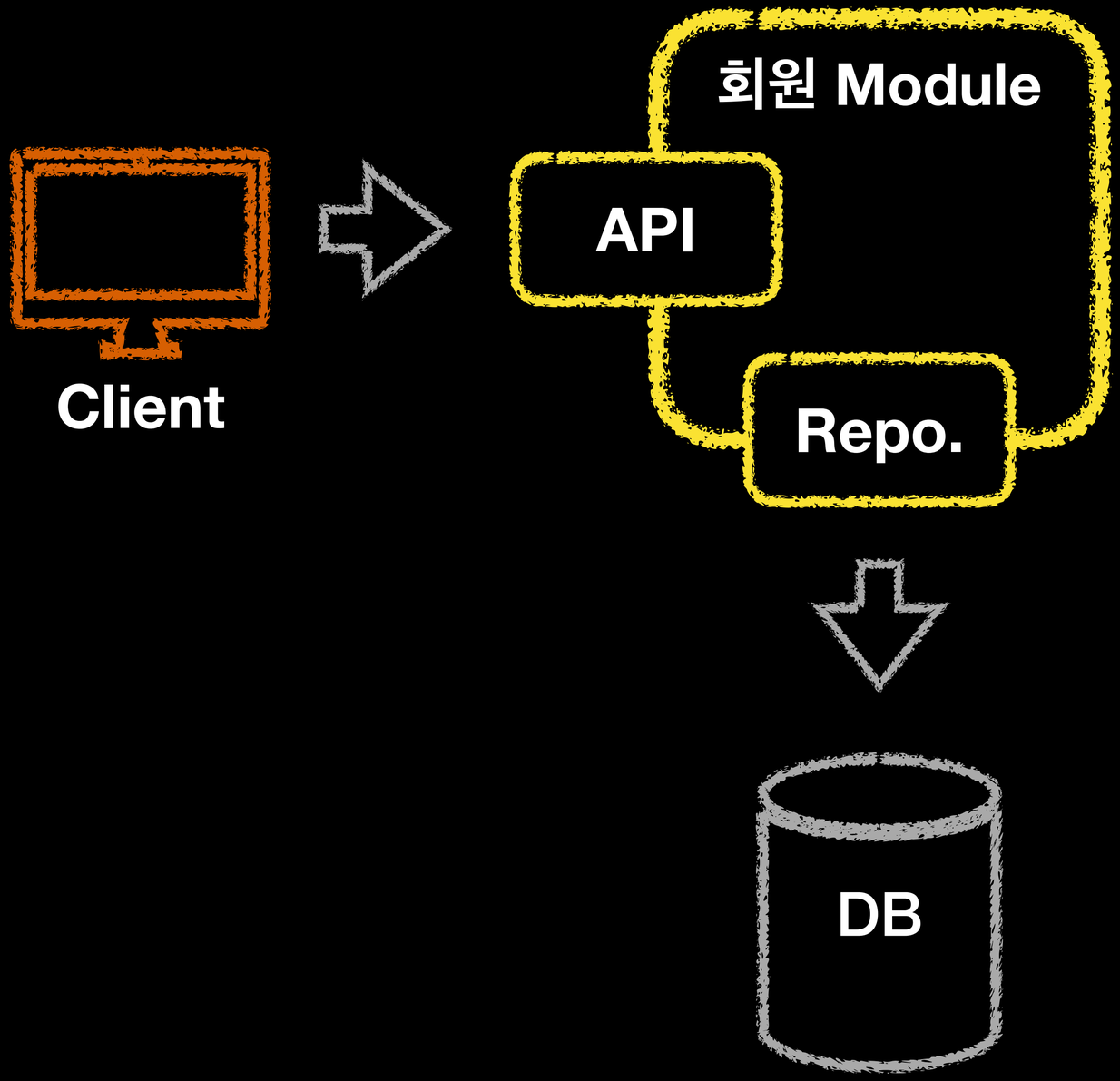
→ **빈약해진** 데이터 무결성

- 구현, 관리의 **단순함**과 무결성 **강도** 간의 **트레이드 오프**

진짜 새로운 시스템으로의 이동

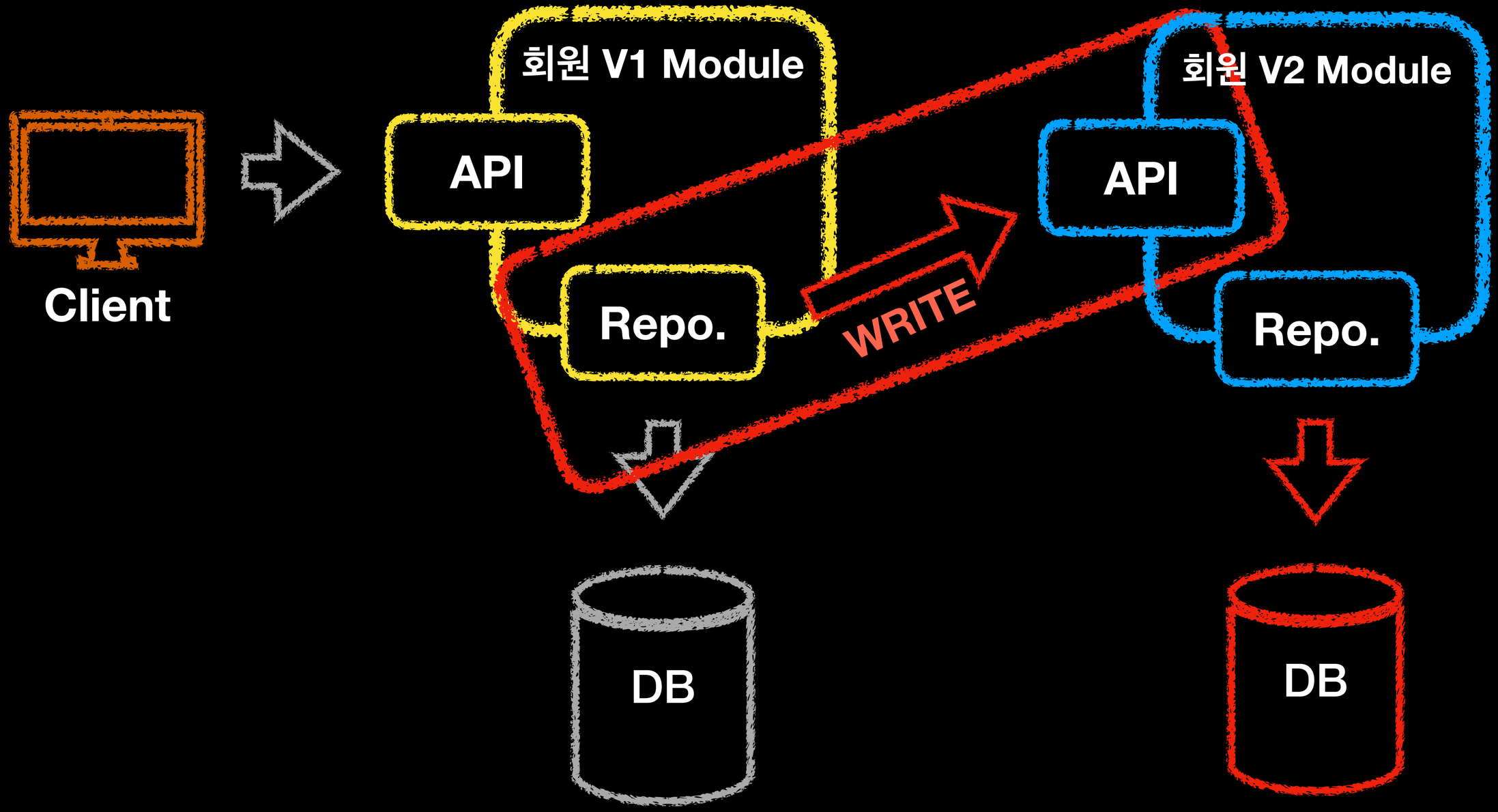
1. 레거시 시스템에서의 두번 쓰기

레거시 애플케이션



레거시 애플케이션

FIG 애플케이션



왜 **Repository** 에서
두번 쓰기를 수행해야 할까요?

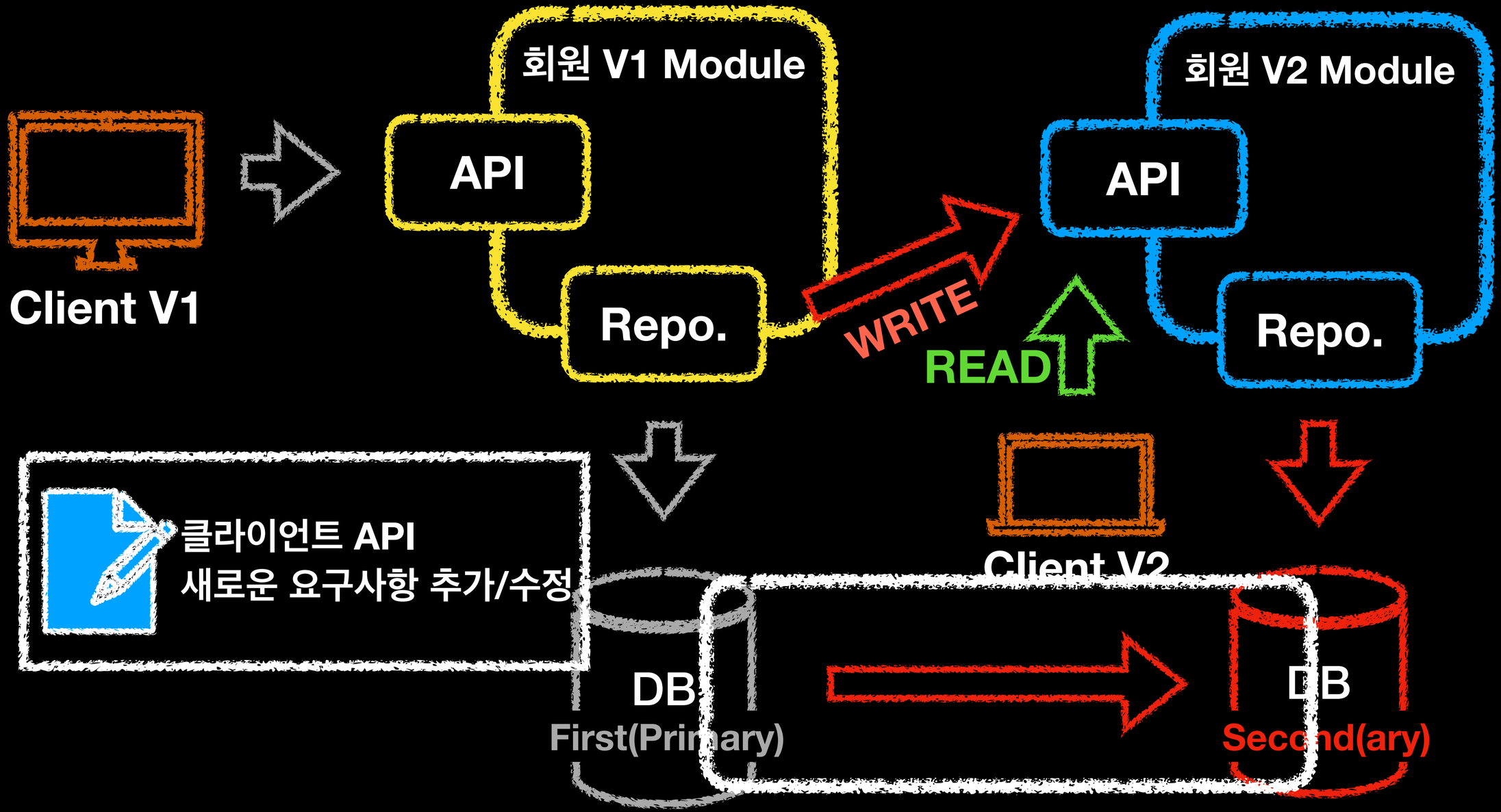
왜 API 에서 두번 쓰기 요청을 받을까요?

	레거시 애플리케이션	FIG 애플리케이션
참여자	**Repository**	**API**
책임	객체의 영속화와 인스턴스화	외부로부터 요청과 응답
FIG APP	또 다른 **저장소**	N/A
레거시 APP	N/A	또 다른 **클라이언트**

2. 데이터 마이그레이션

레거시 애플케이션

FIG 애플케이션



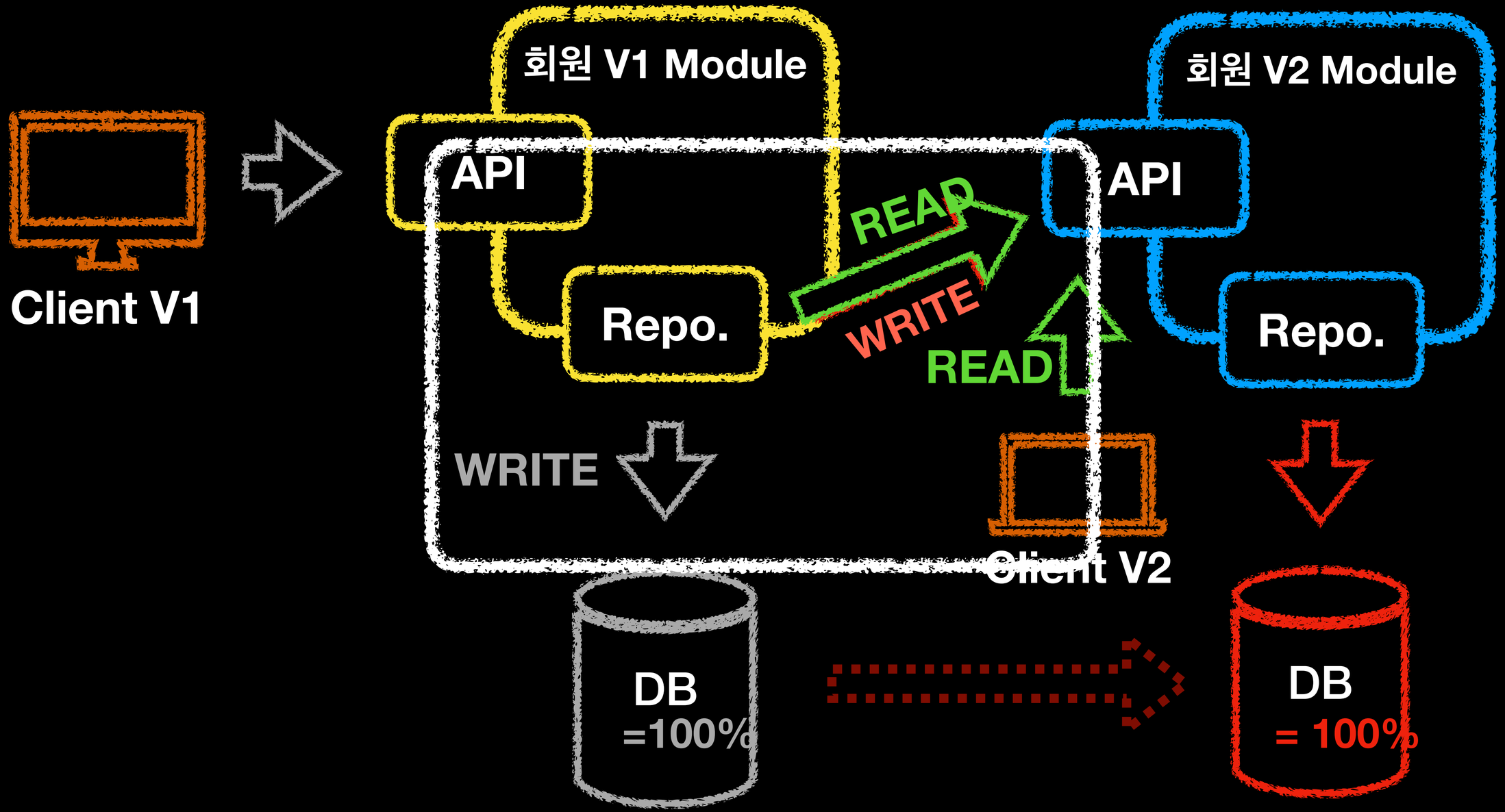
마이그레이션은 **반복**될 수 있음을 고려

추상화 모델의 성장은 곧 데이터 구성 변경

3. 레거시 시스템의 DB 읽기만 전환

레거시 애플케이션

FIG 애플케이션



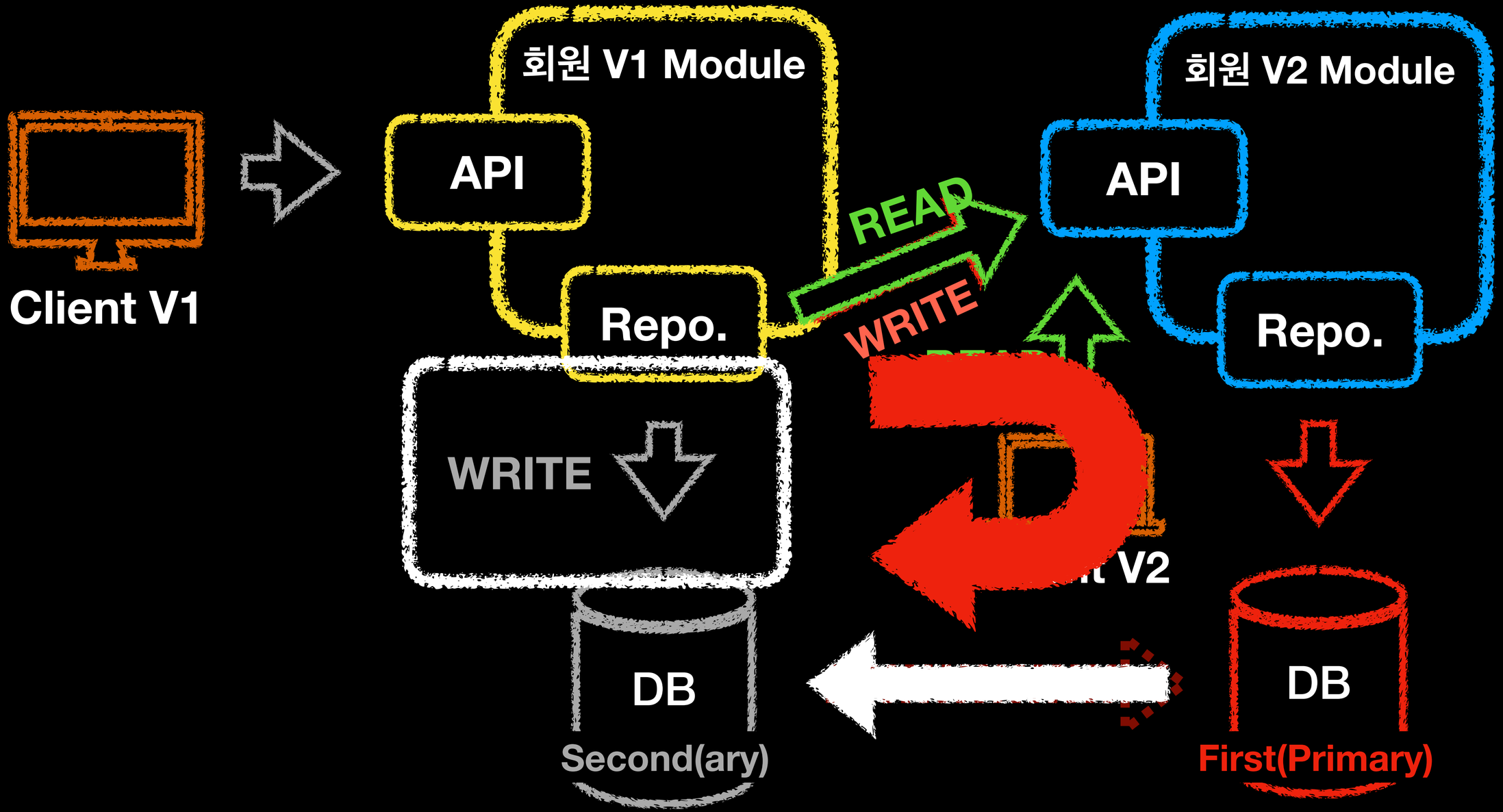
레거시 애플리케이션에서
DB 테이블 **쓰기는 왜 남겨** 둘까요?


- 아직 한발 남았다

4. 발행-구독 패턴으로 레거시 DB 쓰기

레거시 애플케이션

FIG 애플케이션



레거시 애플케이션

FIG 애플케이션

Second(ary)

First(Primary)

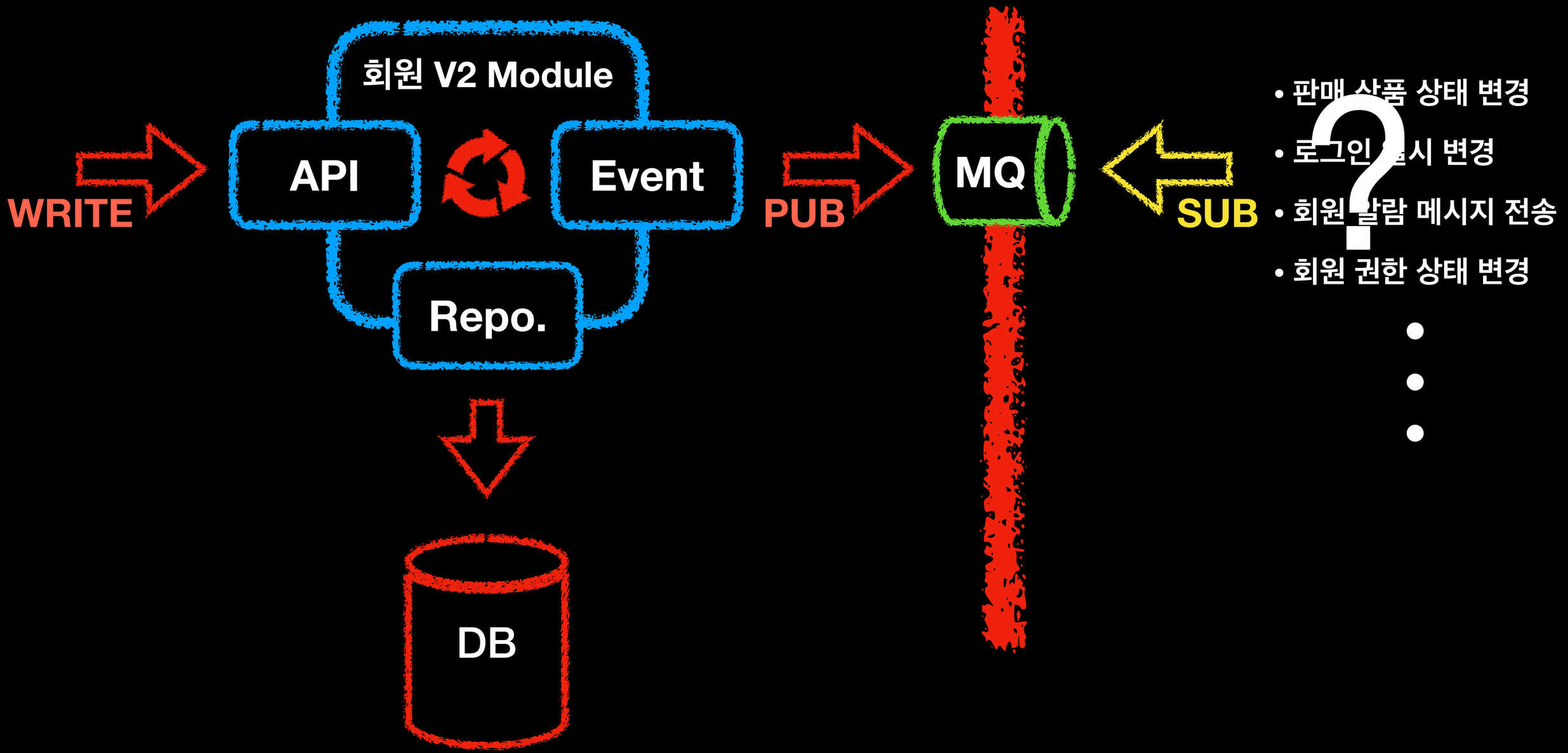
왜 안될까요?

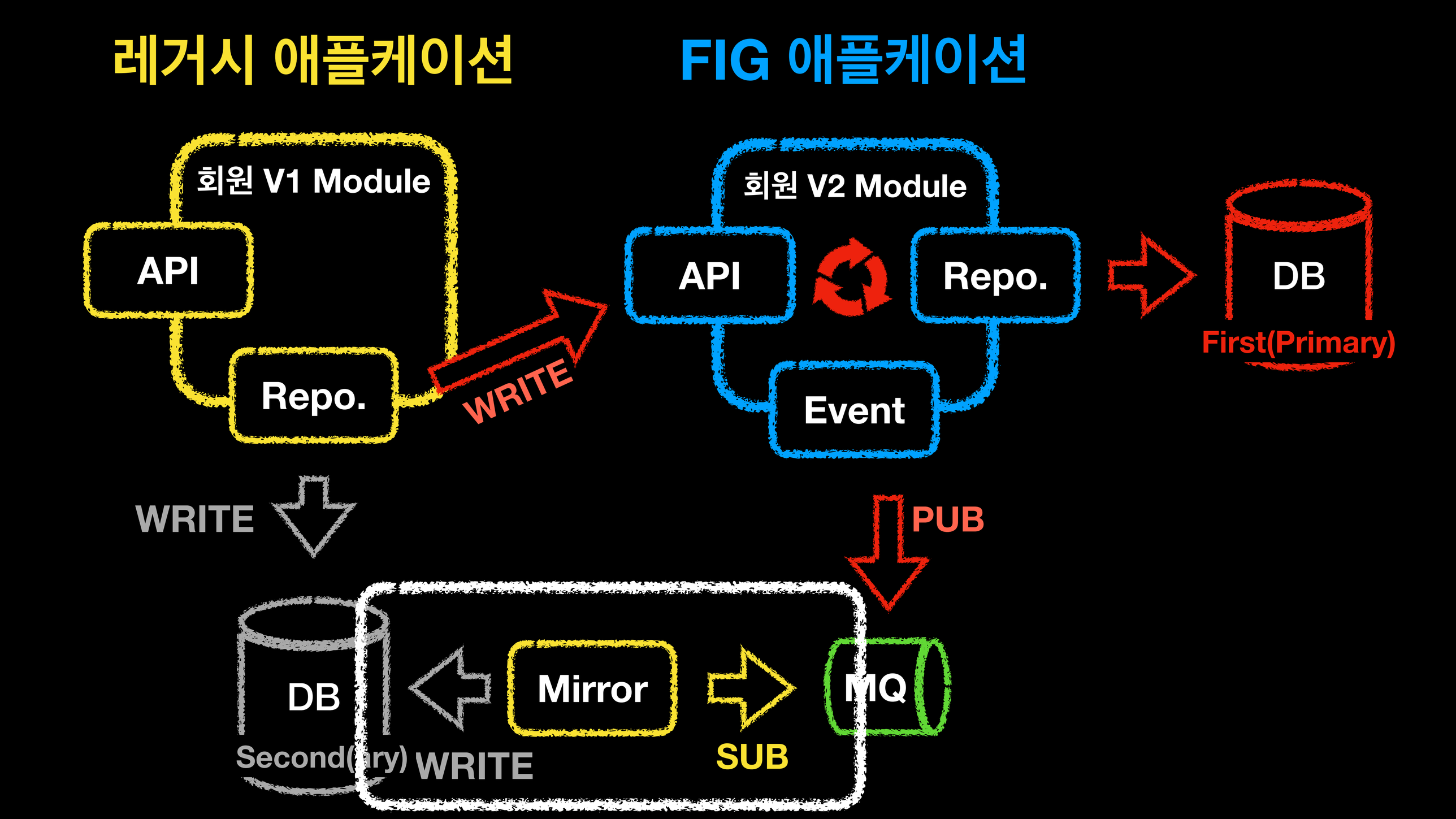
관리 **비용**의 증가

- FIG 와 레거시 간 **상호 참조**
  → 시스템의 **복잡도** 증가
- FIG 에 레거시 **‘만’**을 위한 기능
  → **일반화** 되지 않은 기능 추가 = 테이블마다 **개별 구현**
- 서비스 요구사항과 **관련없는** 구현 코드 증가
  → 요구사항 **변경 시 예측 비용** 증가

Publish–subscribe pattern

FIG 애플케이션


회원 V2 Module
API
Event
Repo.
WRITE
PUB
MQ
SUB
DB
• 판매 상품 상태 변경
• 로그인 방식 변경
• 회원 알람 메시지 전송
• 회원 권한 상태 변경


레거시 애플케이션
FIG 애플케이션
회원 V1 Module
API
Repo.
WRITE
회원 V2 Module
API
Repo.
Event
DB
First(Primary)
WRITE
PUB
DB
Second(ary)
Mirror
MQ
WRITE
SUB

역할과 의존성

레거시 시스템 DB 의 강인한 생명력

- FIG의 **책임**은 **요구사항**을 해결
  → 레거시 DB 데이터 동기화는 **FIG와는 별개**
- MIRROR 의 **책임** = 레거시 시스템 **데이터 무결성**
  → 레거시 애플레케이션 **소멸 후**에도 레거시 시스템 유지
- FIG 와 MIRROR, FIG와 레거시 시스템 간의 **의존성이 없음**
  → 요구사항 구현 시 레거시 시스템 수정 **비용 낮춤**

역할과 의존성

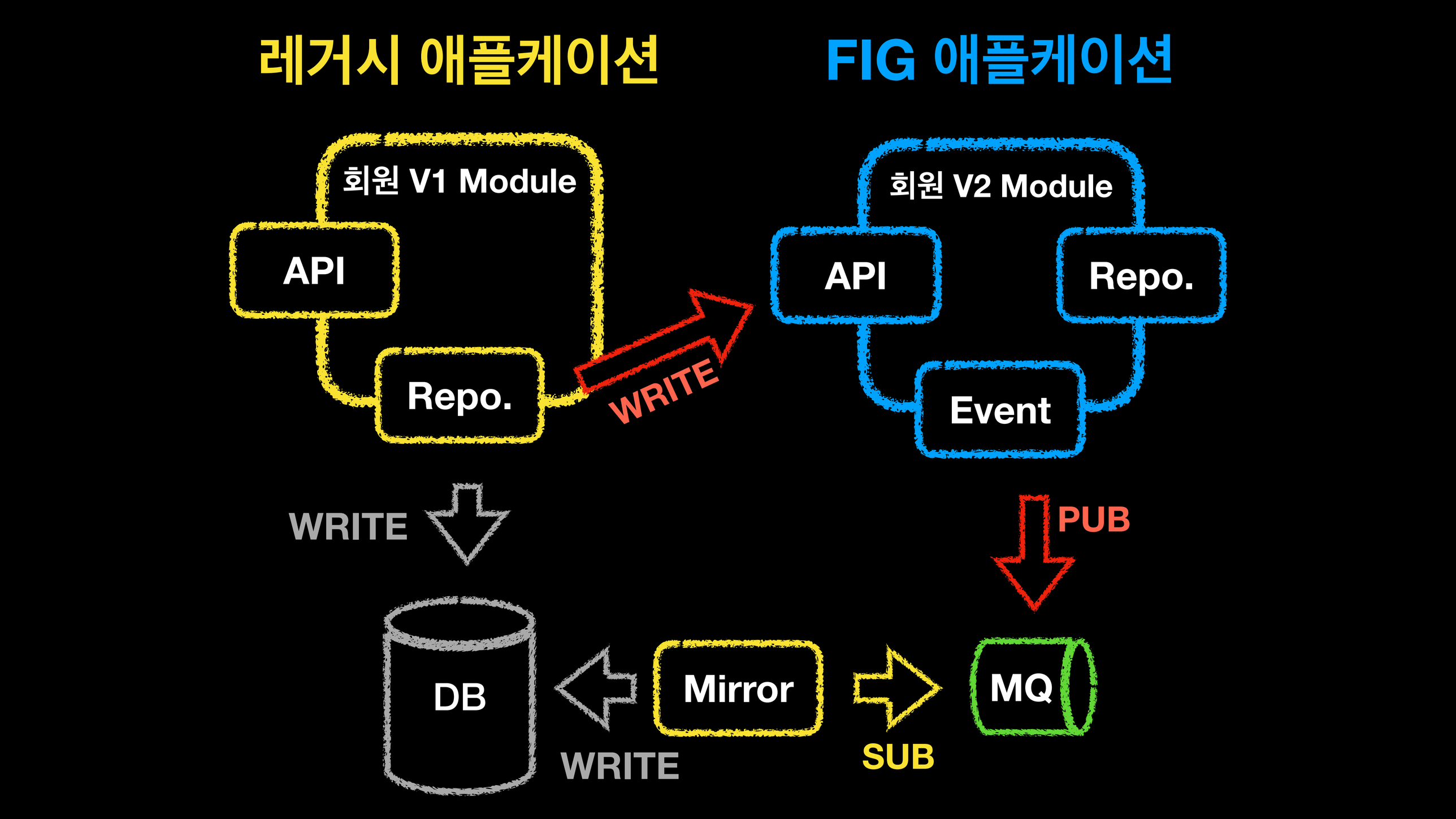
레거시 시스템 DB 의 강인한 생명력

- FIG의 **채임**으 **요구사항**을 해결

신규 시스템과 레거시 시스템 간에
높은 응집도, 낮은 결합도로 인지과부화 해소

- FIG 와 MIRROR, FIG와 레거시 시스템 간의 **의존성이 없음**
  → 요구사항 구현 시 레거시 시스템 수정 **비용 낮춤**

5. 레거시 시스템의 DB 쓰기 제거


레거시 애플케이션
FIG 애플케이션
회원 V1 Module
API
Repo.
WRITE
회원 V2 Module
API
Repo.
Event
WRITE
PUB
DB
Mirror
MQ
WRITE
SUB

WIRTE 제거를 왜 **마지막**에 하나요??

점진적인 개선

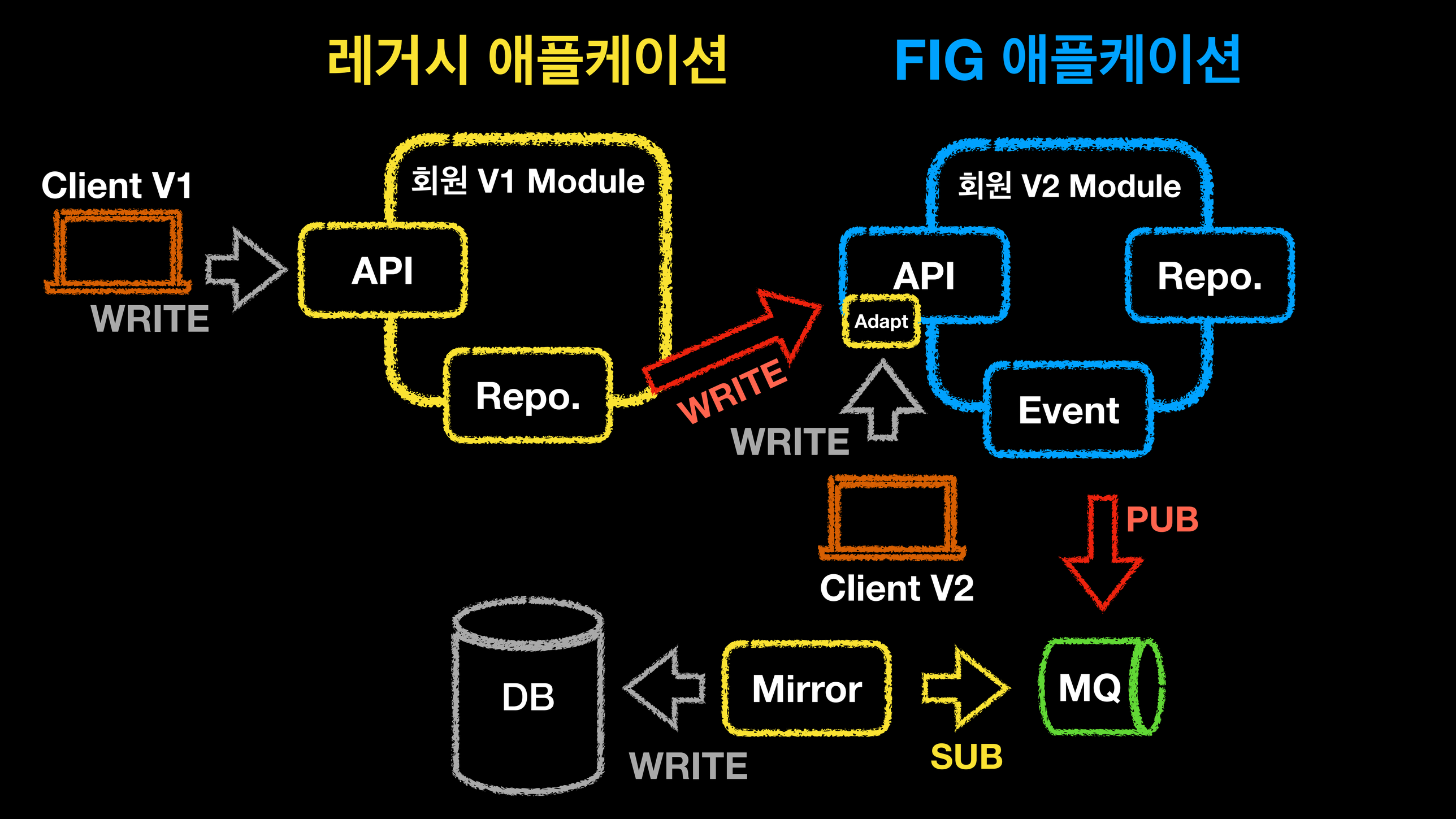
위험 줄이기

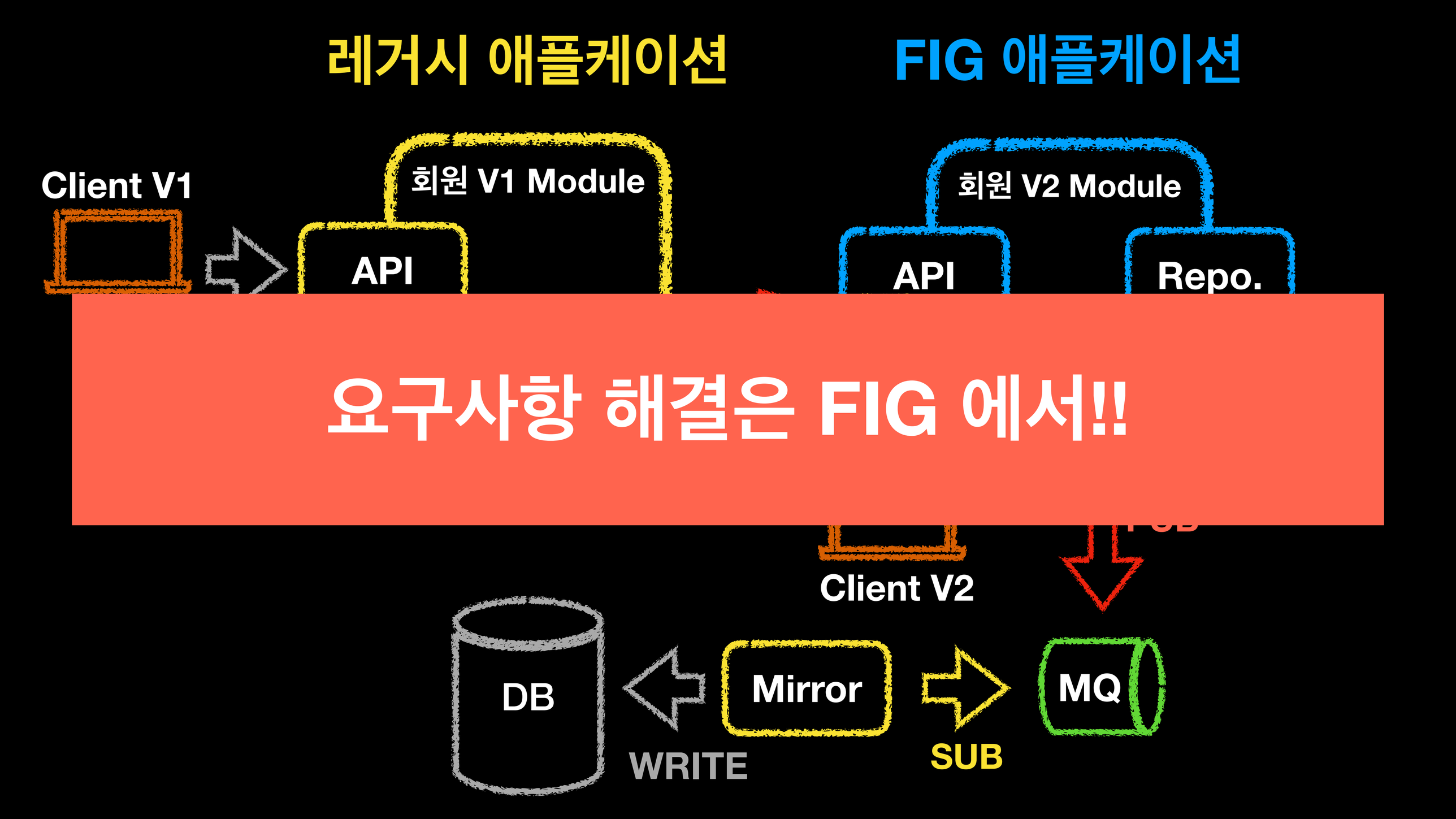
- Real-Time 과 **Near** Real Time 은 다름
- **안정적인** MIRROR 를 구성하기까지 시간이 필요
  → **불확실성 해소**
- 데이터 유실 = **장애**

6. 레거시 시스템에서의 추가 요구사항 구현 전략 (별책부록)

“일정 때문에 기존 클라이언트 전환을 못하겠는데,
기존 API에 기획 요건들 추가해주시겠어요?”

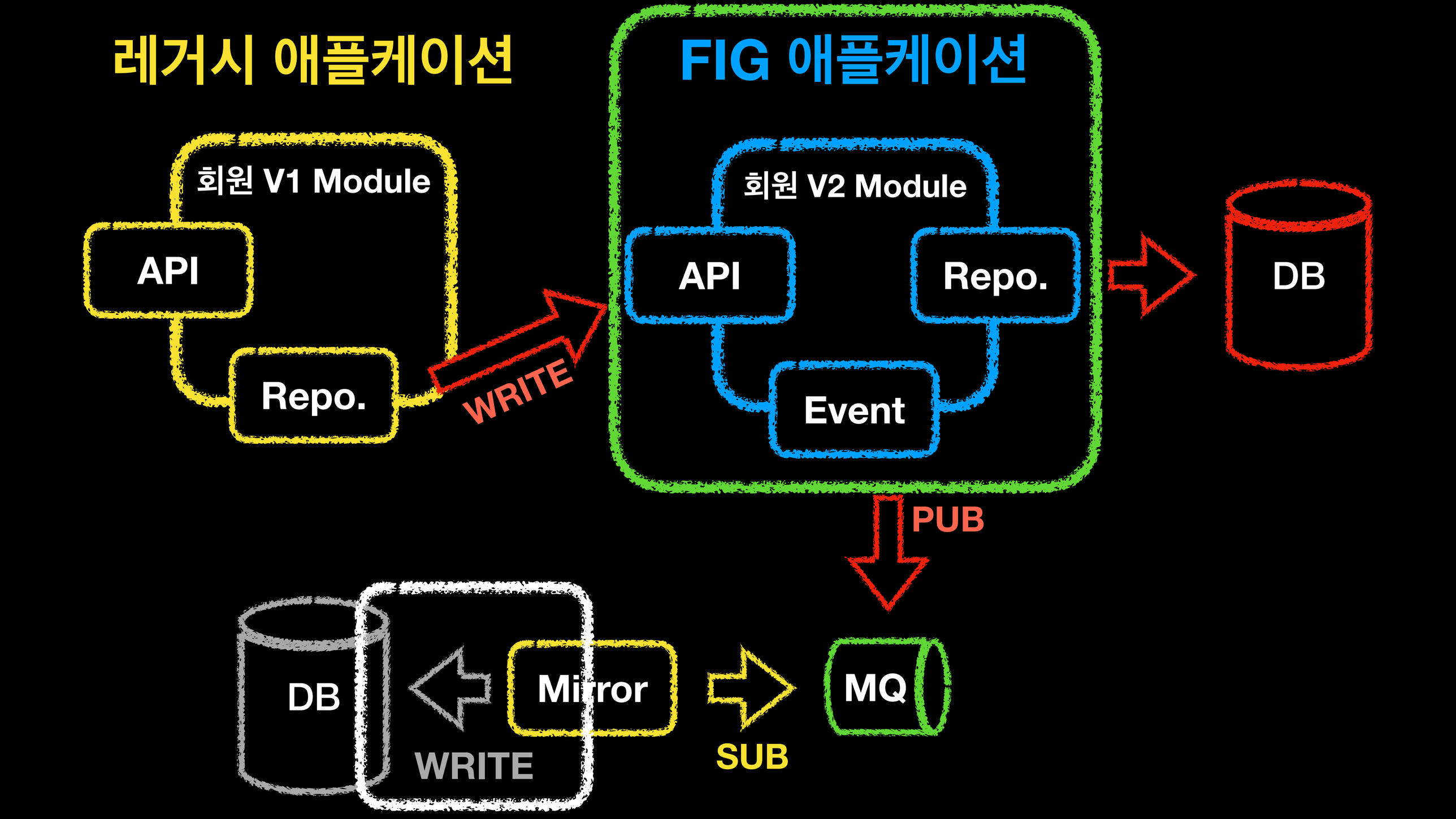
A 본부 B팀 개발자 C 군


레거시 애플케이션
FIG 애플케이션
Client V1
WRITE
회원 V1 Module
API
Repo.
WRITE
WRITE
회원 V2 Module
API
Adapt
Repo.
Event
Client V2
PUB
DB
WRITE
Mirror
SUB
MQ

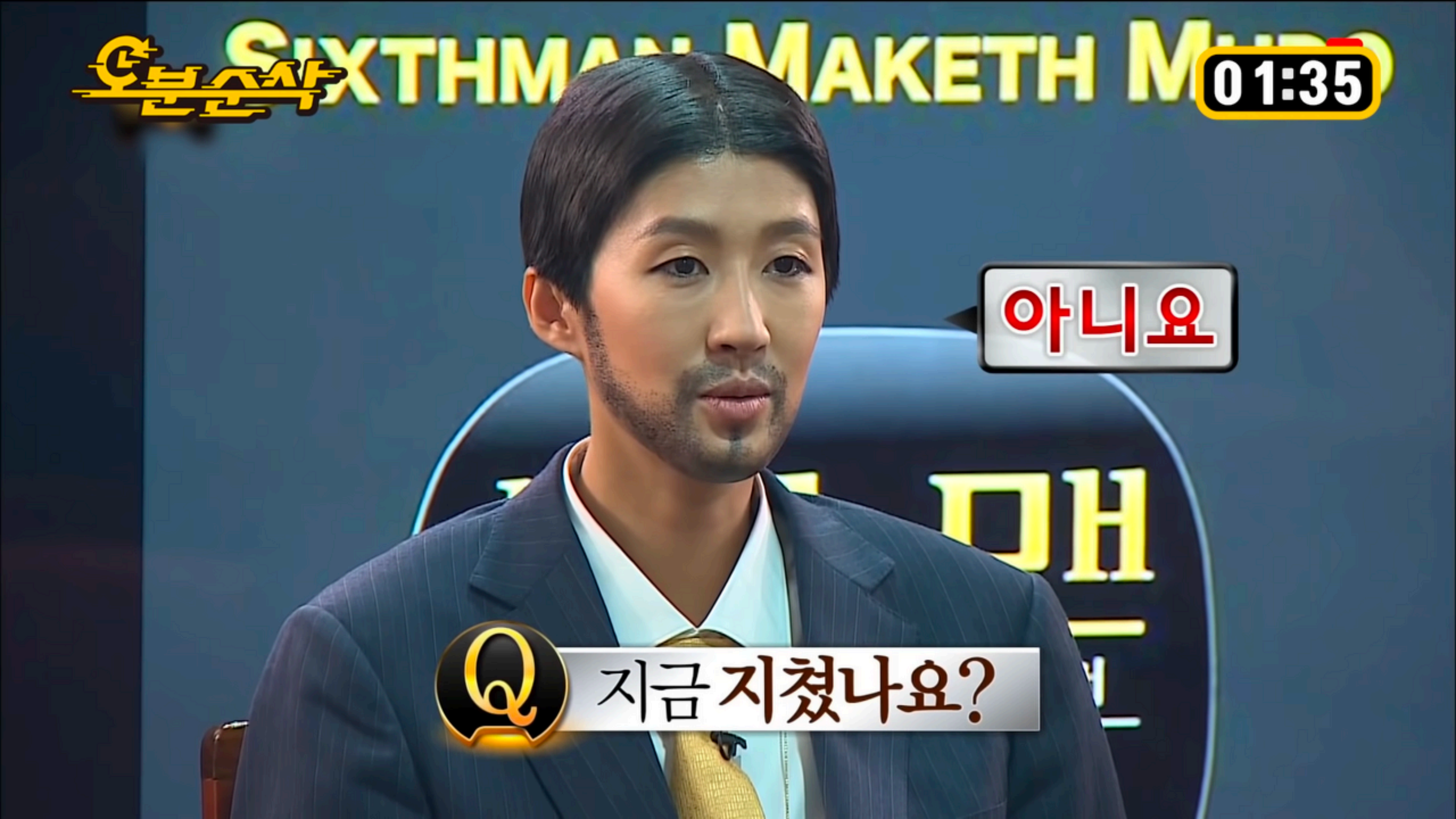

레거시 애플케이션
FIG 애플케이션
Client V1
회원 V1 Module
API
회원 V2 Module
API
Repo.
요구사항 해결은 FIG 에서!!
Client V2
DB
Mirror
MQ
WRITE
SUB

“이번에 신규로 추가된 데이터 연동이 필요해졌는데
기존 DB에 테이블과 필드 추가가 가능할까요?”

D 본부 E팀 개발자 F 군


레거시 애플케이션
회원 V1 Module
API
Repo.
WRITE
FIG 애플케이션
회원 V2 Module
API
Repo.
Event
DB
PUB
DB
Mirror
WRITE
SUB
MQ

7. 반복, 반복, 반복 … 퇴.사.냐.하.르.또..


오늘은 선생
SIXTHMAN MAKETH MUDO
01:35
아니요
Q
지금 지쳤나요?

품질과 끈기


이십세기
힛-트쏭
최단기간 1위! 스피드런 힛-트쏭10
2위 (99년) 핑클 '영원한 사랑'
ON
OFF
이십세기 최고 유행어
약속해줘

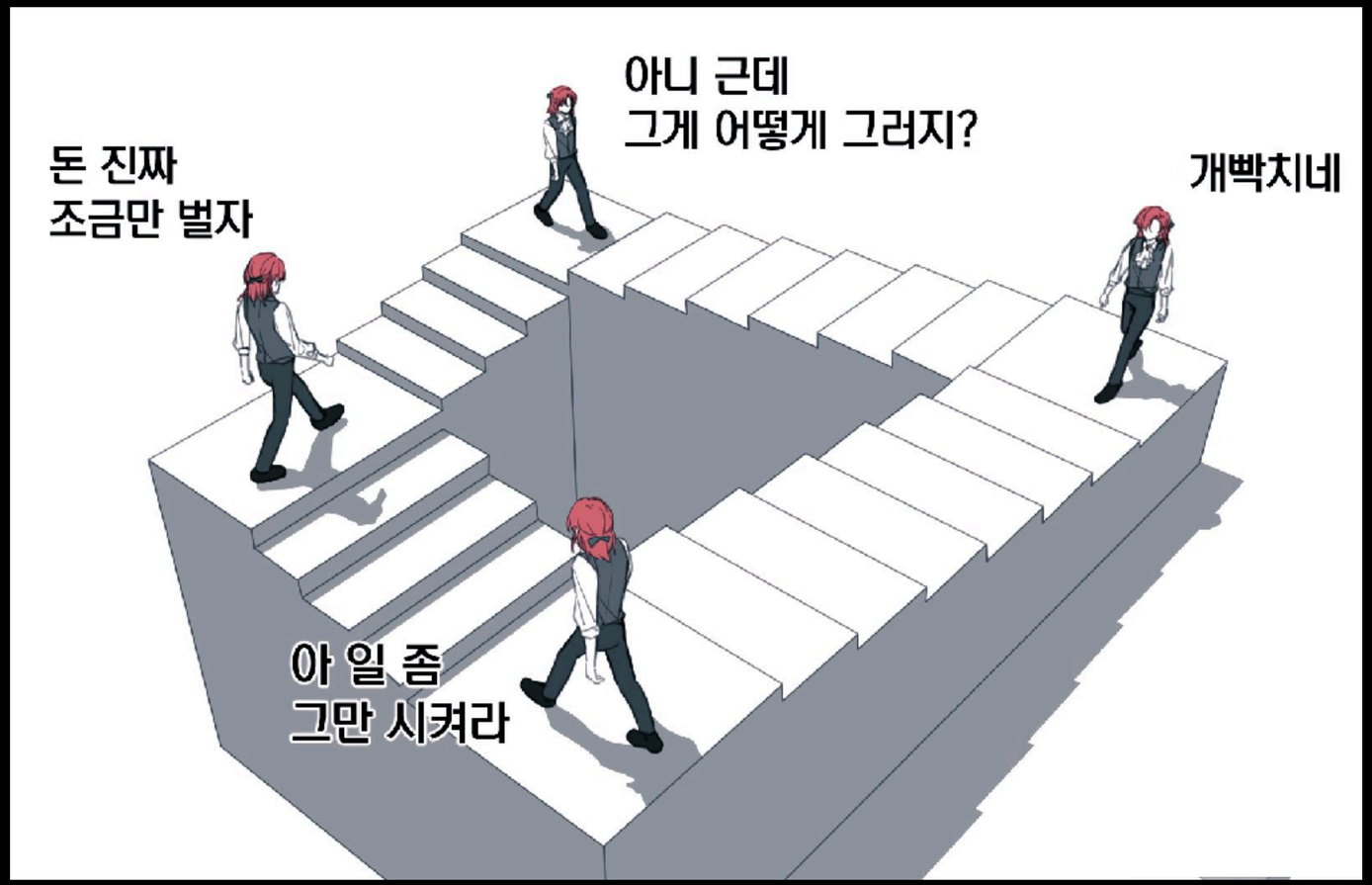
우리는 개발팀으로서 서비스 성장에
높은 품질의 **소프트웨어**와 **개발과정**으로
기여하는 것을 중요한 **책임**으로 여깁니다.

우리는 애플리케이션과 개발과정의
품질 개선**만을 목적으로 실행**하지 않으며
요구사항 기능 구현 과정에서 **함께 개선**하도록 합니다.

우리는 애플리케이션과 개발과정 품질을
개선하는데 필요한 기술과 자원은
팀 역량 안에서부터 실행합니다.

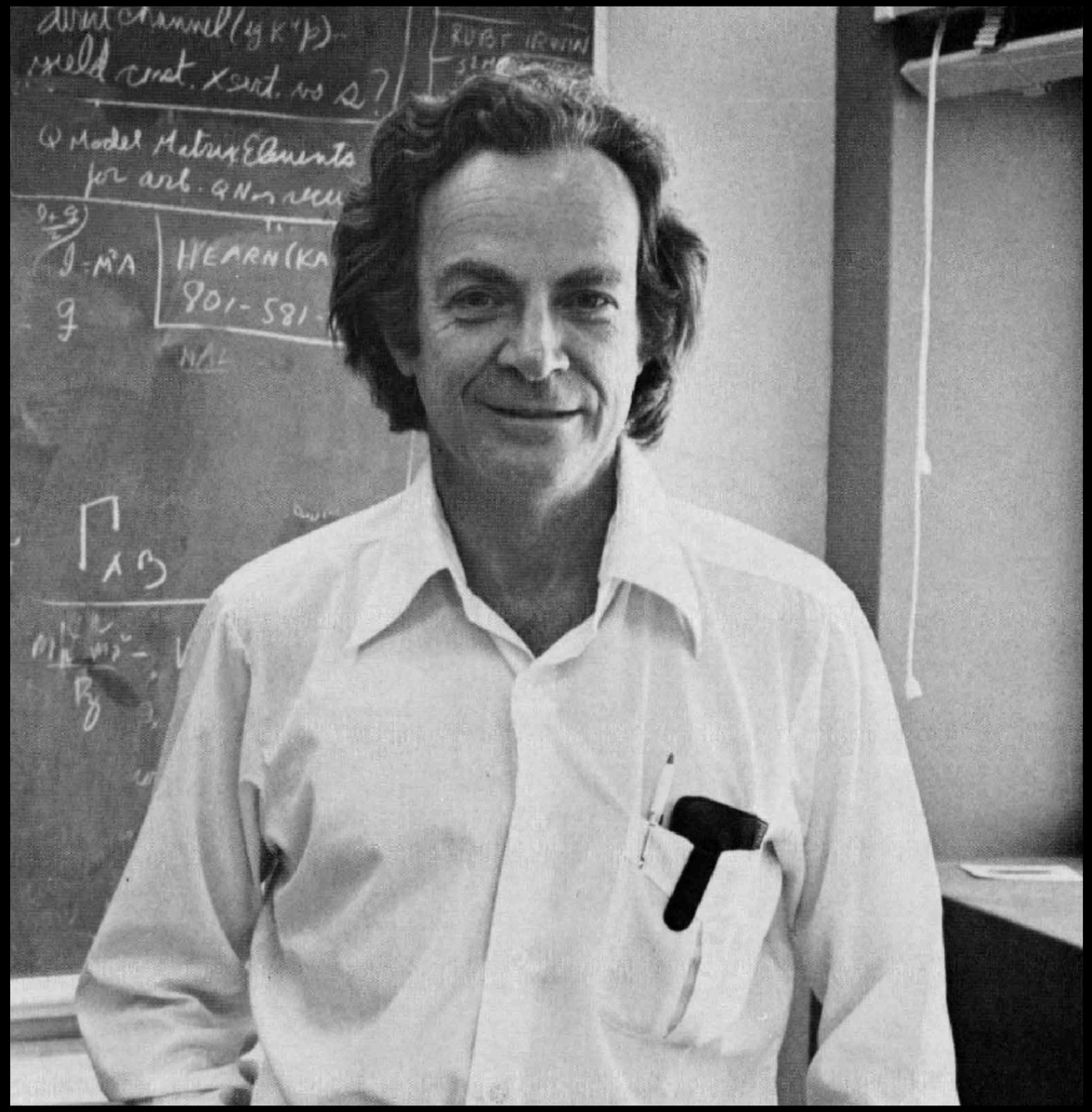
지속가능한 노력

높은 품질에 대한 책임과 끈기



The Feynman Problem-Solving Algorithm

1. write down the problem
2. think very hard
3. write down the answer.
4. ***repeat*... repeat... repeat...**



이제 남은 건?

동료가 돼라!

“개발자는 무언가를 만들때 보다
무언가를 고칠때 성장합니다.”

PL;CO, 시니어가 주니어를 고생시키 전에 하는 말

2부를 기대해주세요

Coming Soon!!

“뭔가 **제대로 진행**이 된다는 것은 뭔가 제대로 진행이 되지 않는 와중에 발생하는 **특별한** 경우일 뿐이다.”

존 올스퍼, 사이트 신뢰성 엔지니어링: 구글이 공개하는 서비스 개발과 운영 노하우


하하하하
힘은 전혀 안나지만
화이팅 하겠습니다