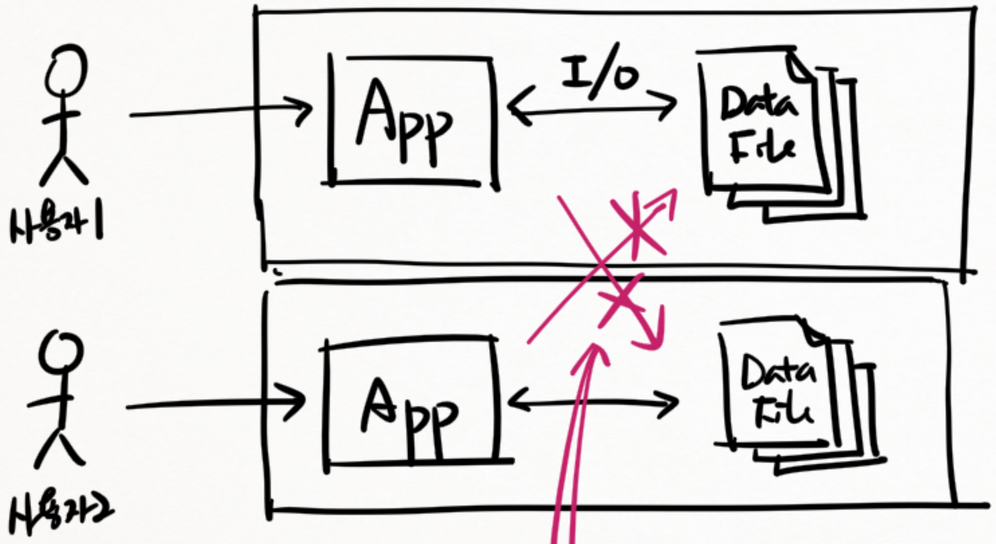
37. 네트워킹을 통한 데이터 공유

① 현황



프로그램 간의 Data File 공유가 안된다.

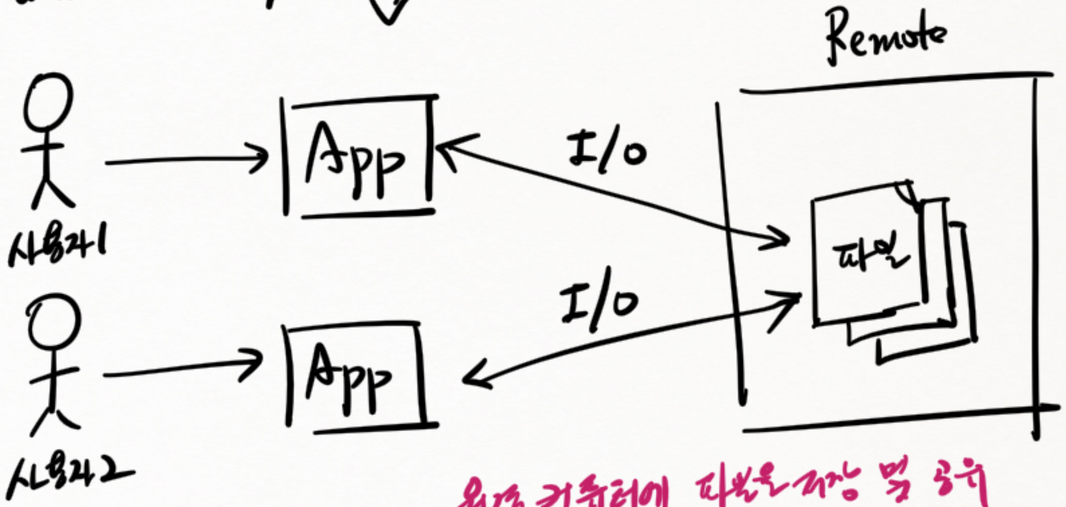
App을 실행하는 컴퓨터

데이터 파일이 로컬에 저장된다.

⇓

App 간에 데이터를 공유할 수 없다!

② 데이터 파일을 별도의 컴퓨터에 보관/공유 ⇓ 해결책?

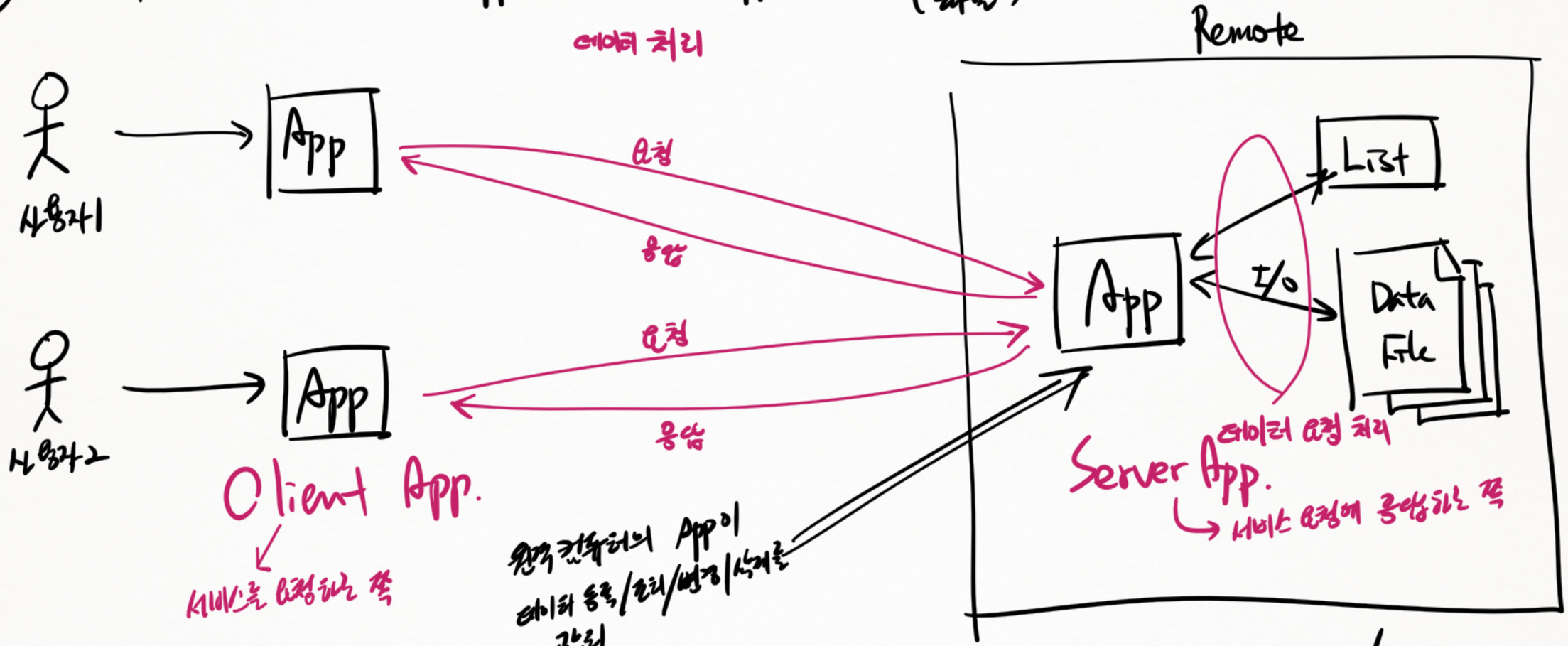


- 원격 컴퓨터에 파일을 저장 및 공유

⇓

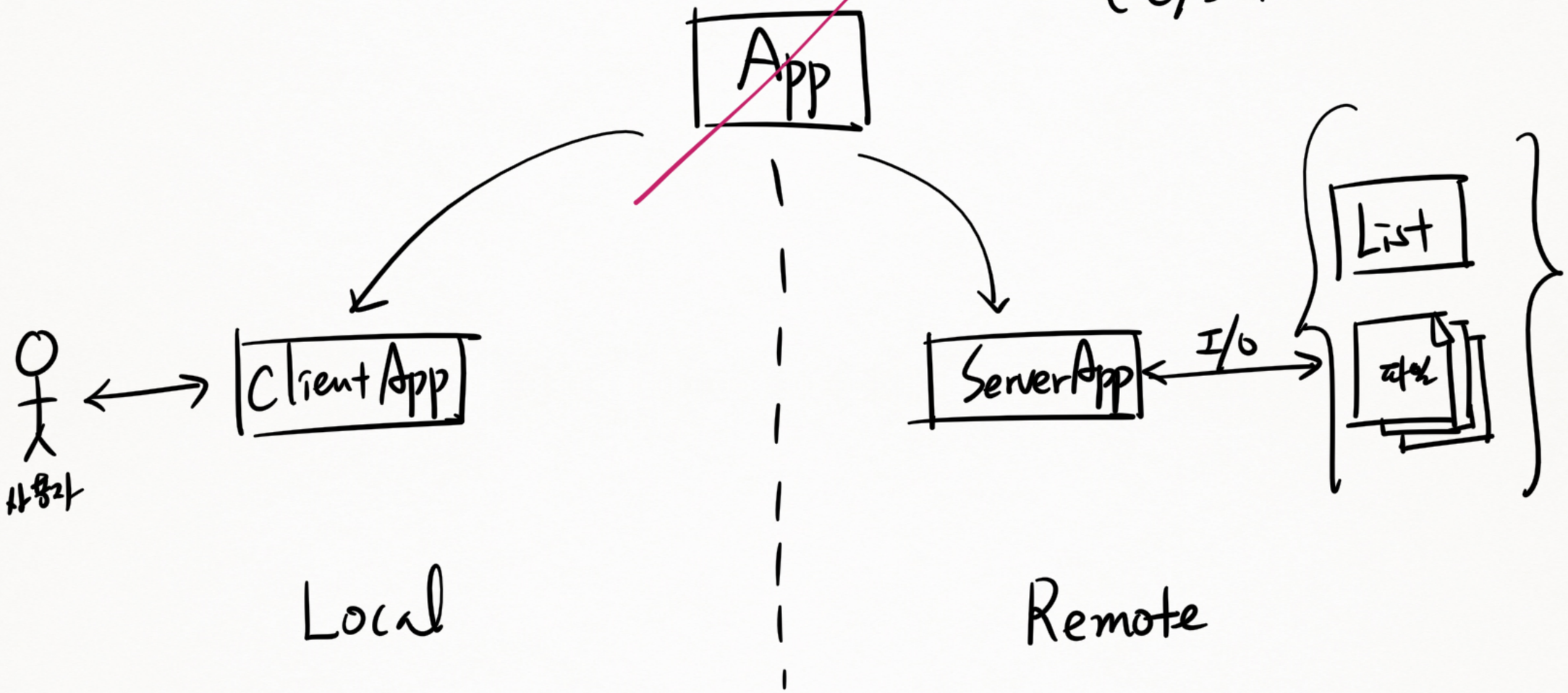
여러 App이 동시에 같은 파일을 접근한다 보면 파일의 데이터가 깨질 수 있다.

③ 데이터 관리 기능을 별도의 App.으로 분리 — App이 직접 (데이터 파일)에 접근하는 것을 막는다.

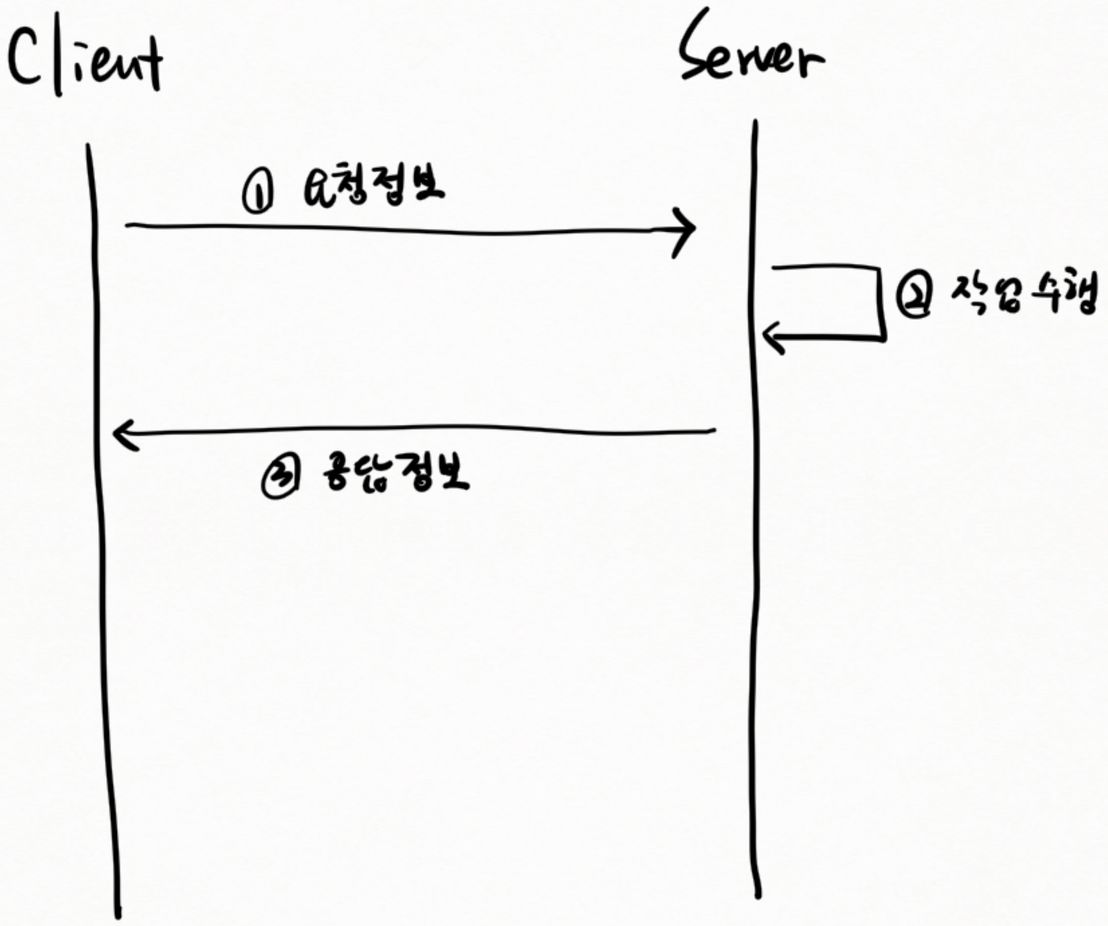


↳ 같은 (파일/데이터)를 동시에 접근할 때 발생할 수 있는 문제를 피할 수 있다!

* System Architecture : Client/Server Architecture
(C/S Architecture)



* client / server 통신 규칙 (protocol)



* client / server 통신 규칙(protocol)

① 요청 정보 (JSON 문자열)

```
{
    "command" : "테이블이름/명령",
    "data"    : "JSON 문자열"
}
```

"테이블이름 / 명령"

게시글 : board
회원 : member
독서록 : reading

서버의 DAO 메서드명.

⇓ 예

```
{
    "command" : "board/insert",
    "data" : "{\"title\":\"...\",
              \"content\":\"...\",
              }"  ⋮
}
```

⇐

JSON 문자열

command 값 : 일반 문자열
data 값 : JSON 문자열

* client / server 통신 규칙 (protocol)

② 응답 정보 (JSON 문자열)

```
{
    "status" : "실행결과",
    "result" : "JSON 문자열"
}
```

⇓ 성공 에

```
{
    "status" : "success",
    "result" : "{\"no\" : 12, ...}"
}
```

* 실행결과

"success" : 성공

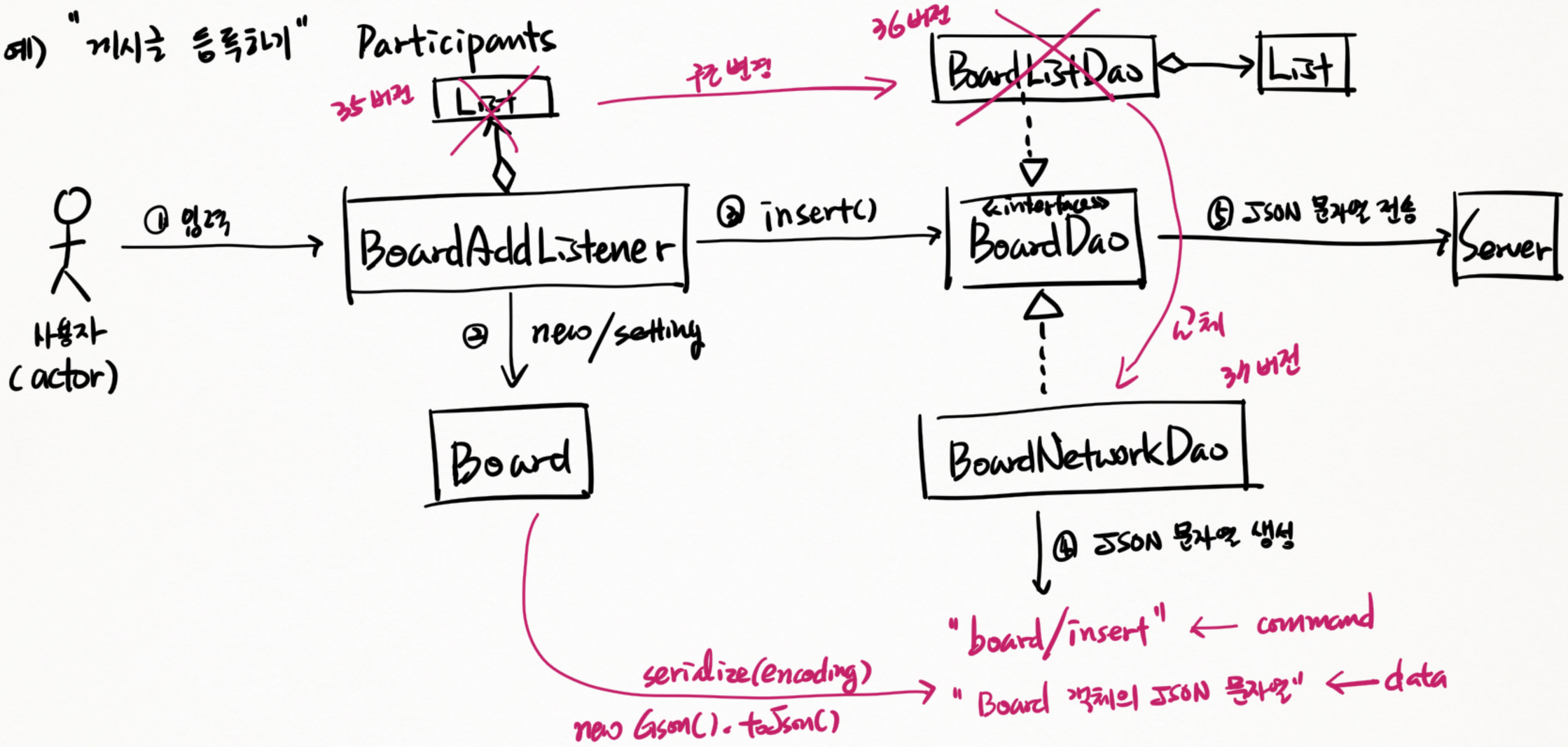
"failure" : 실패

⇓ 실패 에

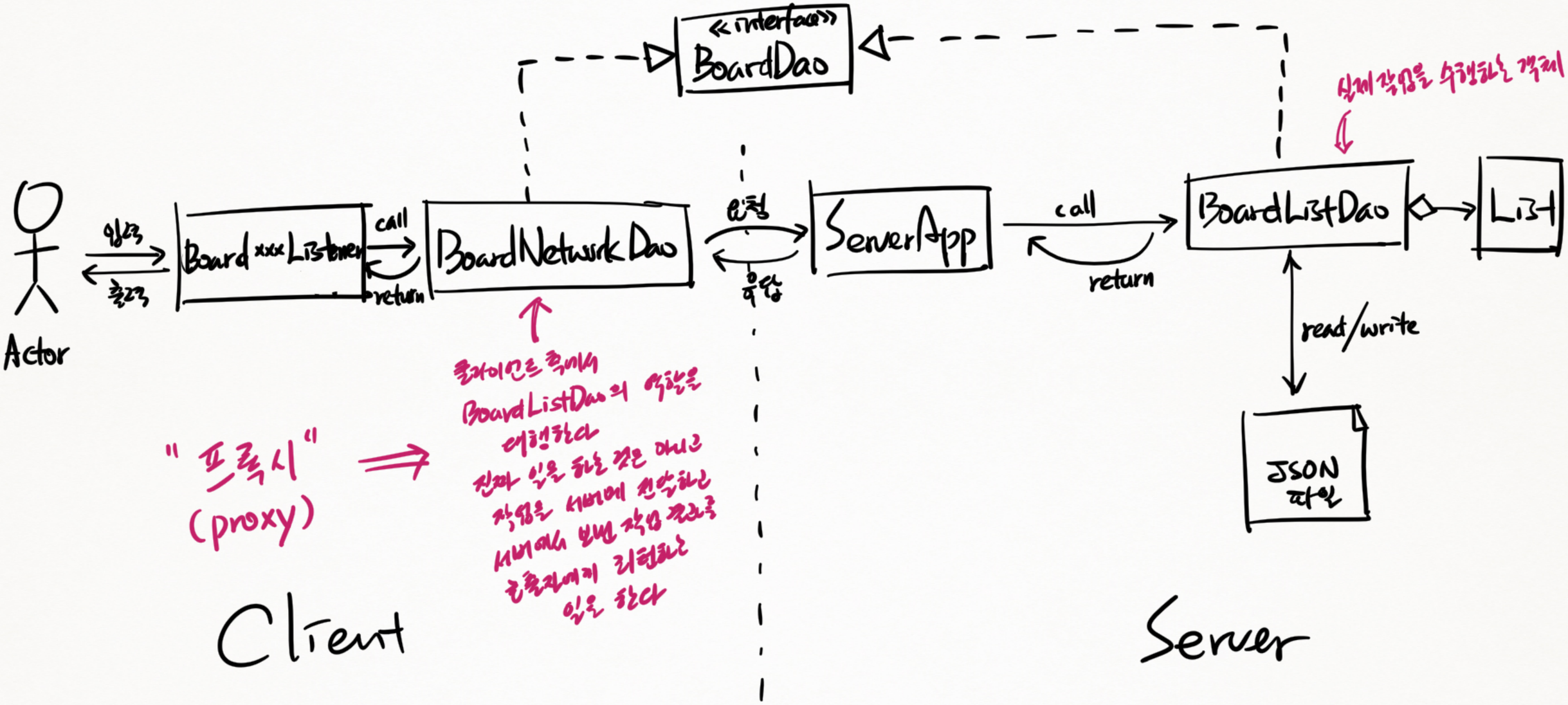
```
{
    "status" : "failure",
    "result" : "해당 번호의 데이터가 없습니다!"
}
```

* 요청 정보 보내기 작업에 참여하는 객체들과 실행 흐름

예) "게시글 등록하기" Participants

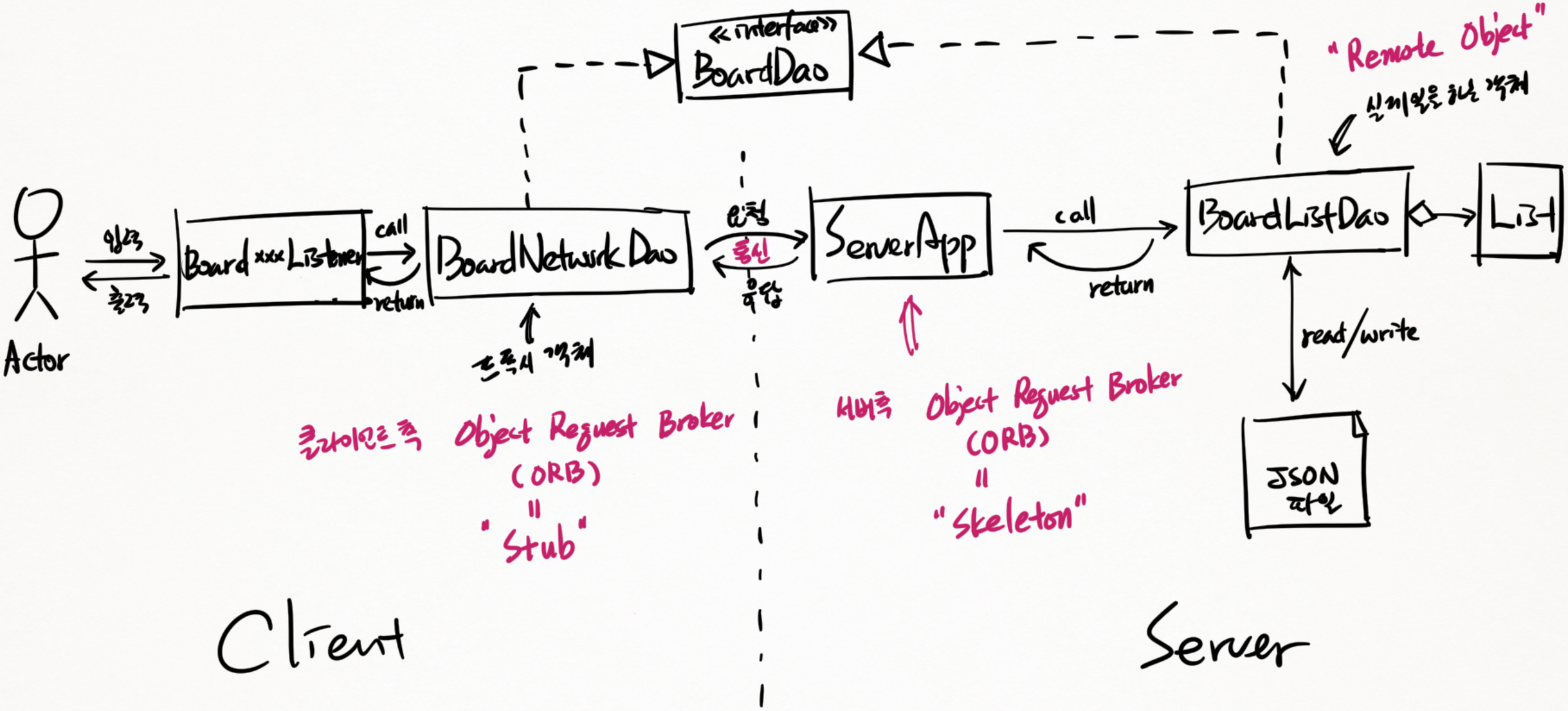


* DAO 와 Proxy 패턴 (GoF)



* DAO 와 Proxy 패턴 (GoF)

각 역할자의 명칭



* 분산 컴퓨팅

① 중앙 집중식 컴퓨팅 → ② 분산 컴퓨팅 ⇒ Rest API (RESTful)

↓

AJAX, XML/JSON

Web → HTTP

- 전용 API

Web Service
- WSDL
- SOAP
- UDDI
- XML

+Web

Java EJB

이기종 통합

CORBA (Common ORB Architecture)
- IIOP

OOP { C++, Pascal

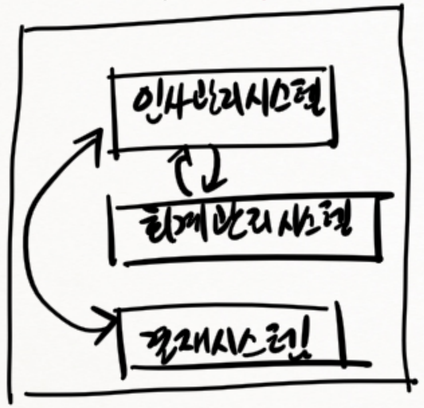
RMI (Remote Method Invocation)

C

RPC (Remote Procedure Call)

해결책

서버 (메인 프레임)



크기↓ 가격↓ 용량↓ (down sizing)

(워크스테이션) 서버1 — 인사관리시스템

↳ Unix

↳ Solaris, HP-UX, IBM AIX, ...

(워크스테이션) 서버2 — 회계관리시스템

(워크스테이션) 서버3 — 판매시스템

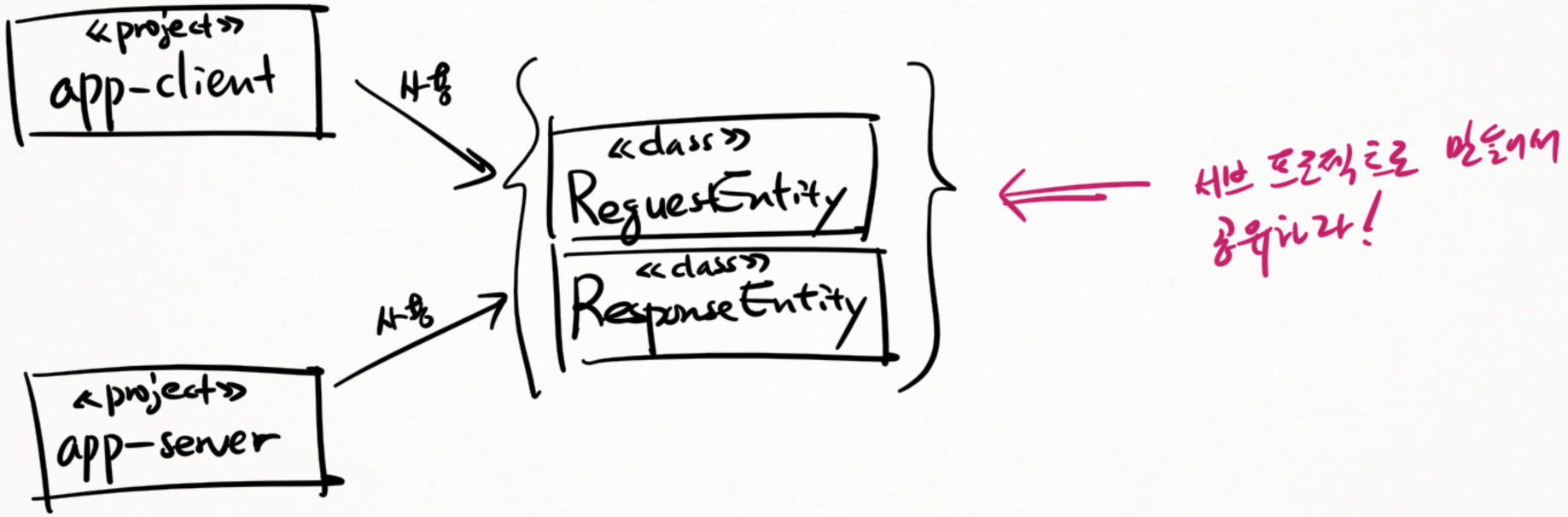
call

call

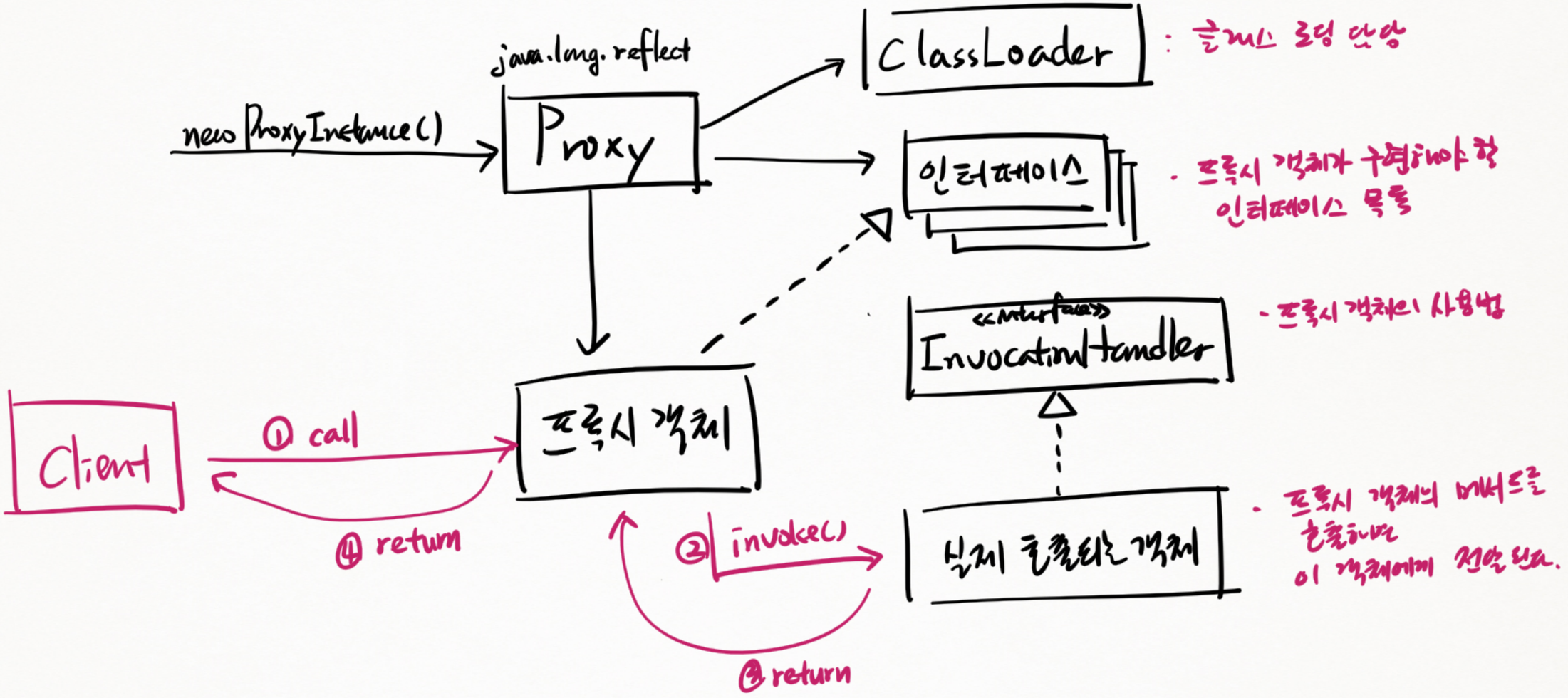
문제점
- 서버에 문제 발생
  ↓
  모든 업무시스템이 정지
- 사용자 증가 → 서버 증설 (RAM, HDD, CPU)
  ↓
  서버 장비는 고가
  ↓
  H/W 유지비가 많이든다.

문제점
- 다른 시스템의 코드를 실행할 수 없다
  ↓
  네트워크를 이용하여 데이터 통신
  ↓
  개발자/개발사 별로 구현
  ↓
  표준화 X → 유지보수 어렵다

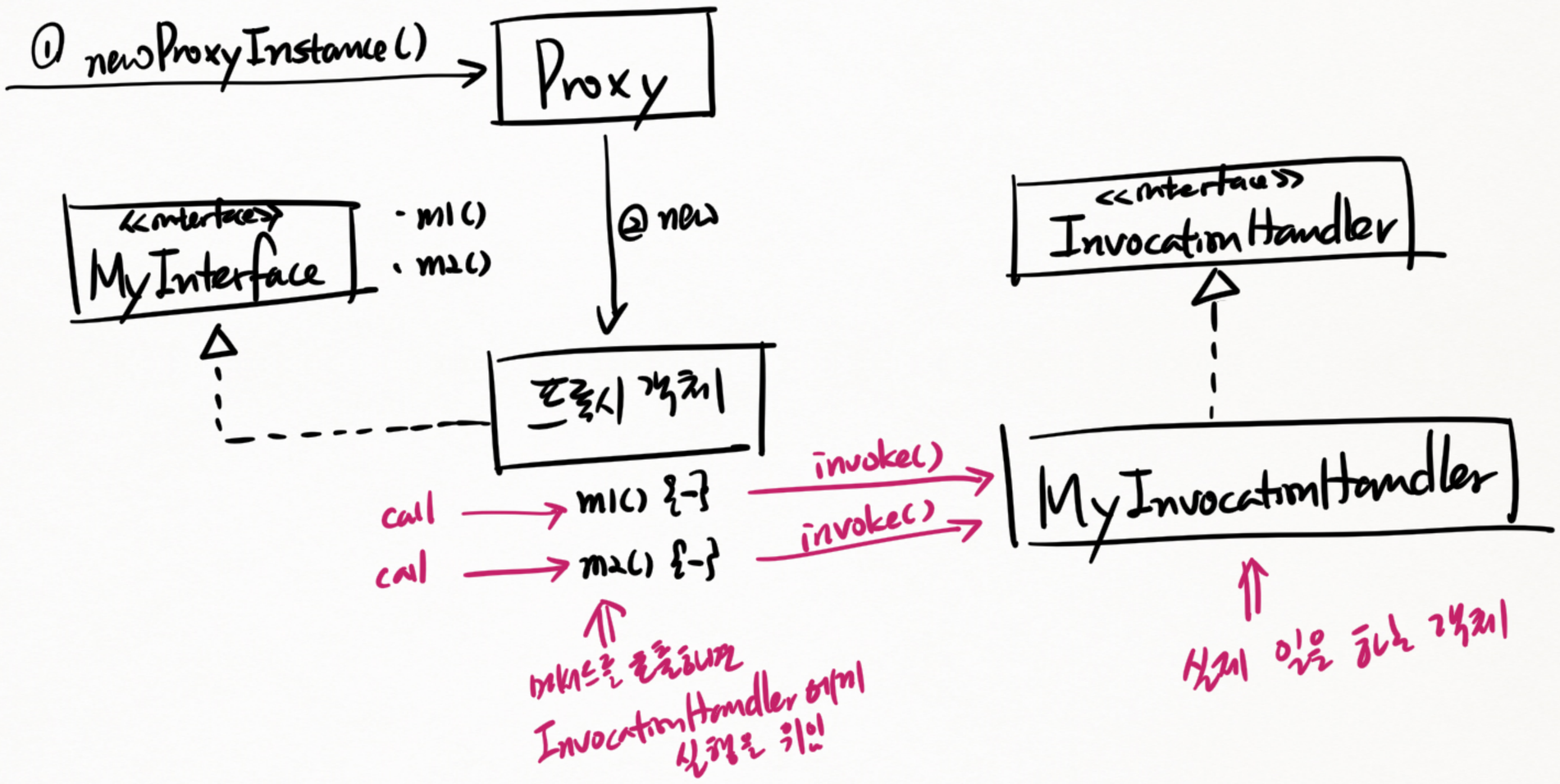
* Project 간에 클래스 공유하기



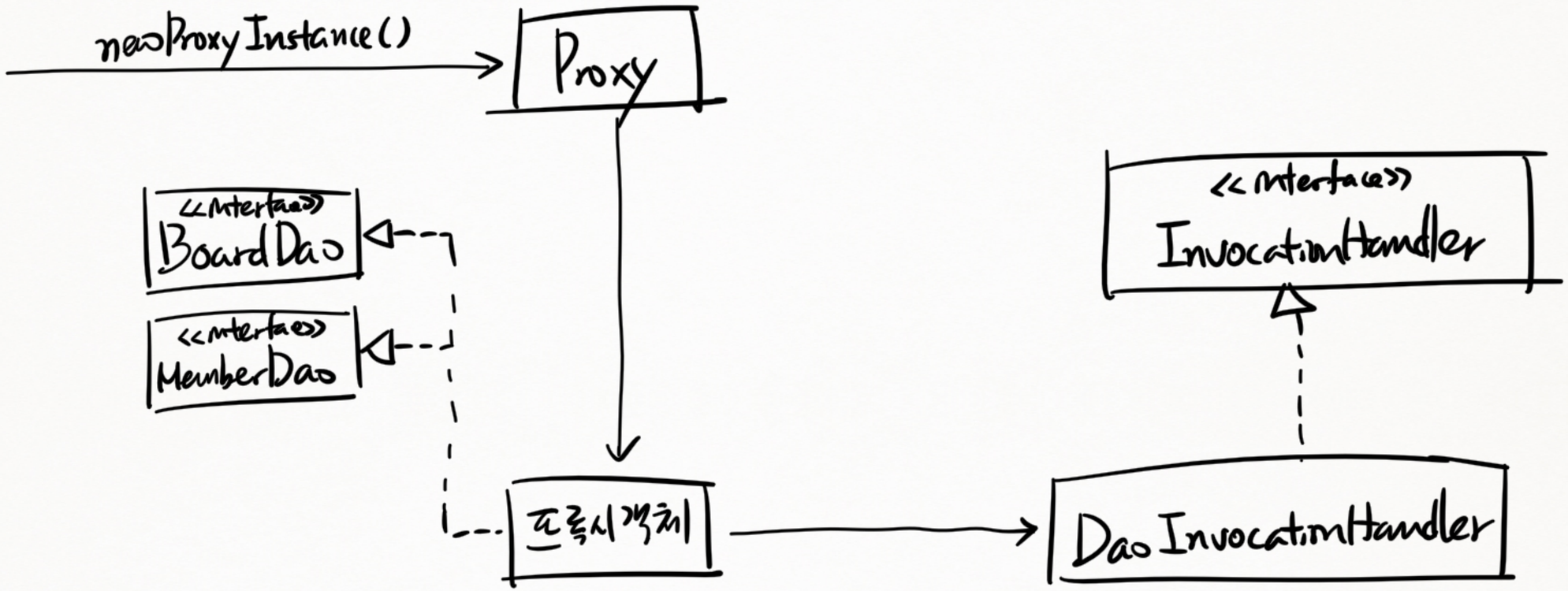
38. 프록시 객체 자동 생성



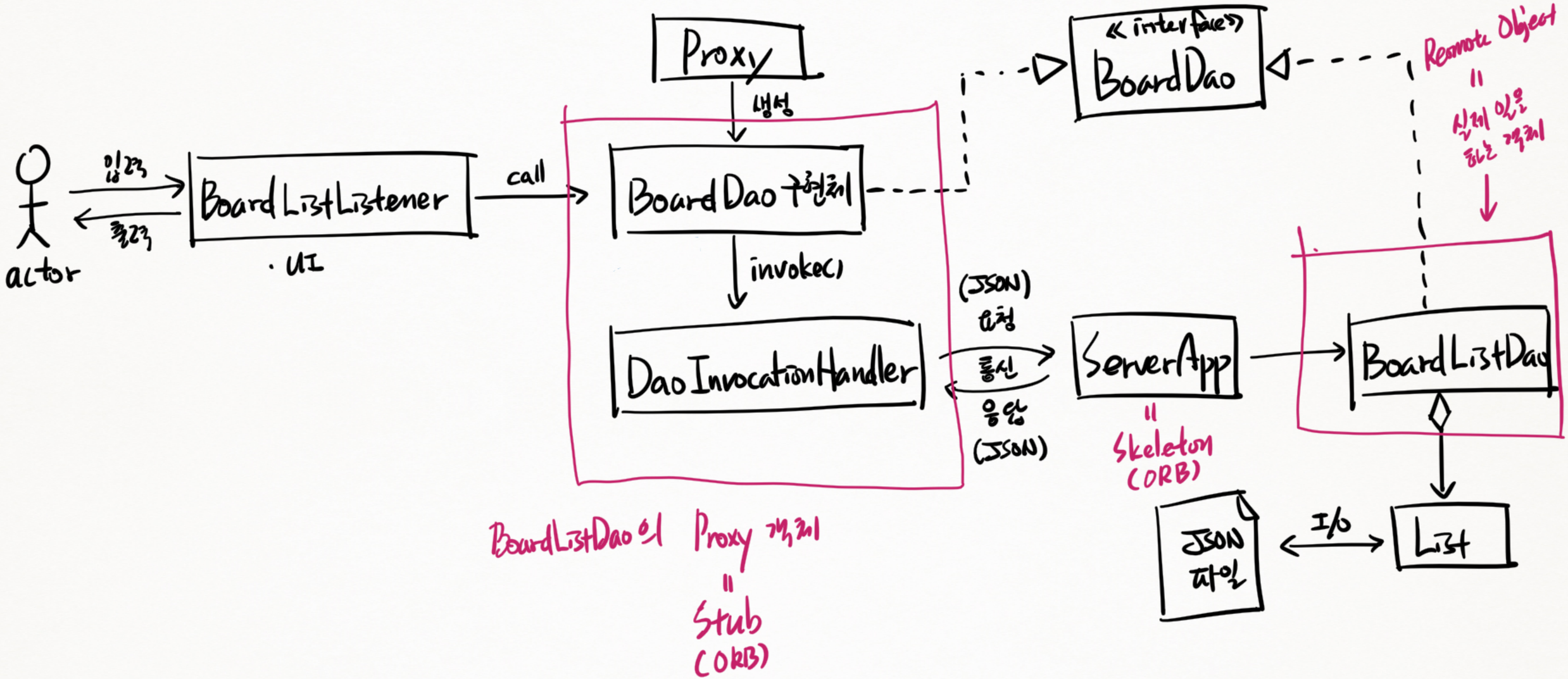
38. 프록시 객체 자동 생성 — 예1



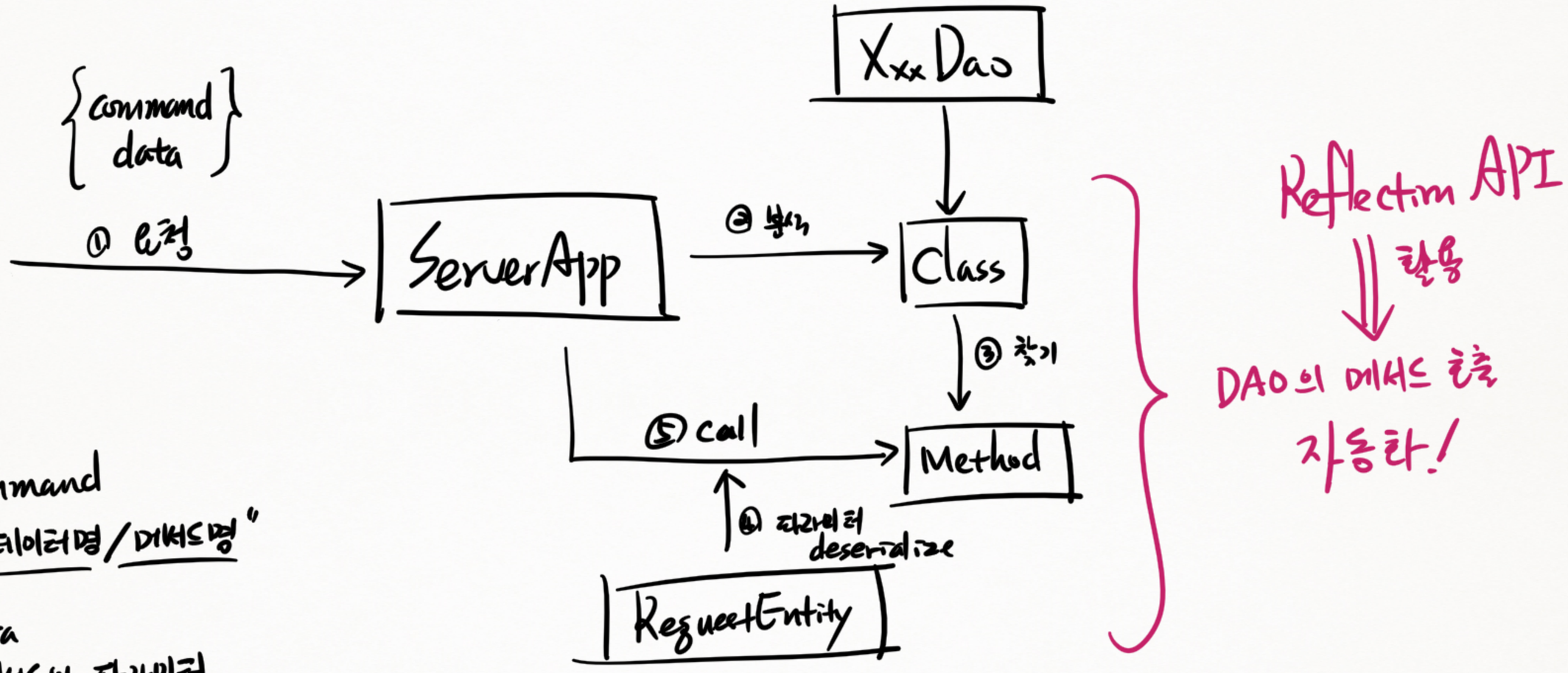
38. 프록시 객체 자동 생성 - Participants



38. 프록시 객체 자동 생성 - Participants Ⅱ



39. Reflection API를 사용해서 DAO 객체의 메서드 호출을 자동화 하기



* command
  "테이터명/메서드명"

* data
  메서드의 파라미터

40. 예외 처리 적용

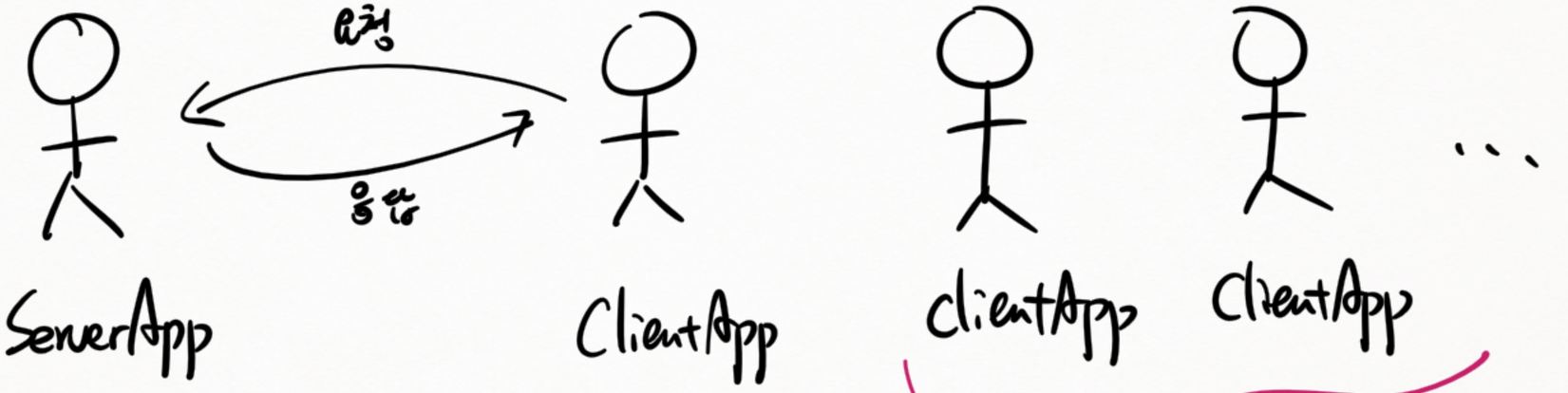
① client 측

```
try {
    ≡
    menu.execute();
    ≡
} catch (Exception e){
    ≡
}
```

② server 측

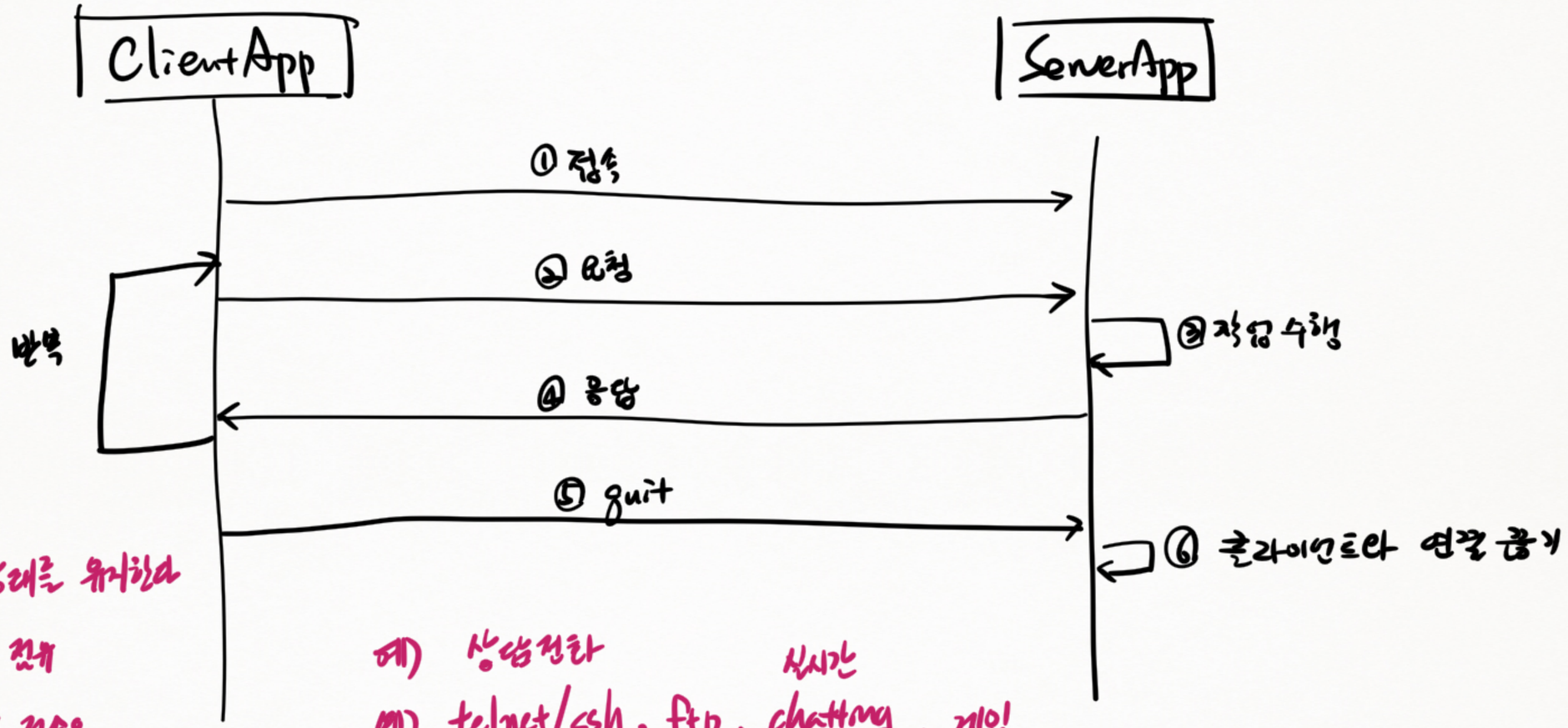
```
try {
    ≡
    dao.invoke(—);
    ≡
} catch (Exception e){
    ≡
}
```

4. 여러 클라이언트 요청을 순차적으로 처리하기 - Stateful 방식



먼저 접속한 client가
접속을 끊을 때까지 대기!

4. 여러 클라이언트 요청을 순차적으로 처리하기 - Stateful 방식



예) 상담전화

실시간

예) telnet/ssh, ftp, chatting, 게임

* 특징

① 접속해 연결된 상태를 유지한다

↓

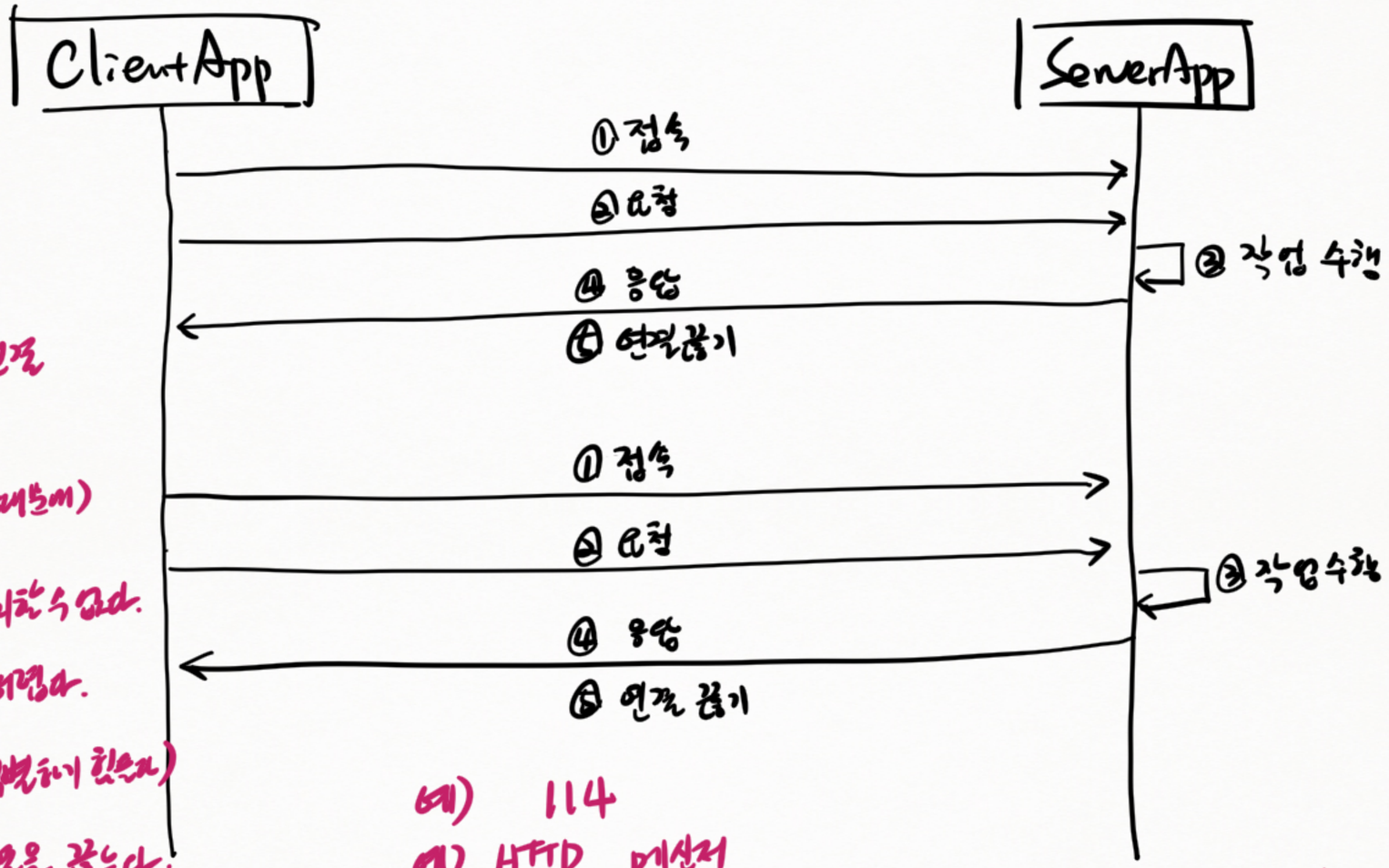
· 서버 자원을 점유

↓

· 동시에 많은 클라이언트 접속을 유지할 수 없다

② 클라이언트 정보 유지 → 클라이언트가 요청한 작업 결과 유지 → 연결 작업을 처리하기 수월.

42. 여러 클라이언트 요청을 순차적으로 처리하기 — Stateless 방식



* 특징

① 요청할 때마다 매번 연결

↓

· 요청 시간이 길어진다.
(연결 시간이 포함되기 때문에)

↓

클라이언트 작업 관리를 위임할 수 없다.

↓

연결 작업을 수행하기 어렵다.
(매번 연결하기 때문에
서버는 클라이언트를 식별하기 힘들다)

② 응답 후 클라이언트와 연결을 끊는다.

↳ 서버에 클라이언트 정보를 유지하지 않는다

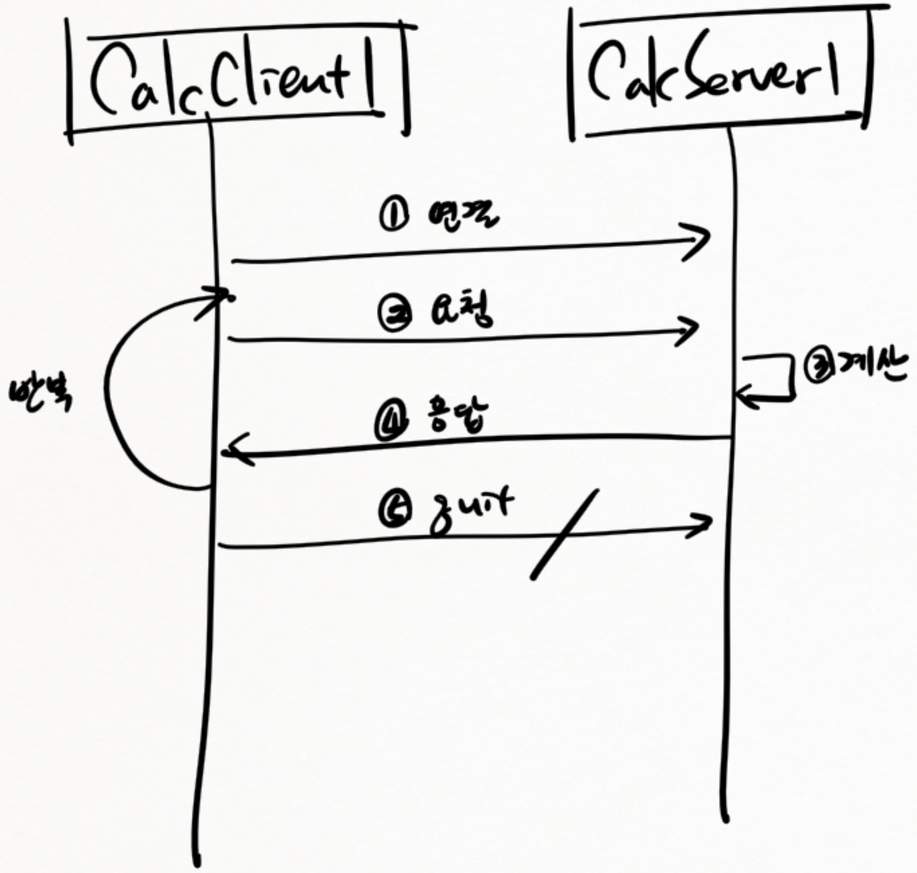
↳ 서버 자원을 덜 사용 → Stateful 방식보다 더 많은 클라이언트 동시 접속을 처리할 수 있다.

예) 114

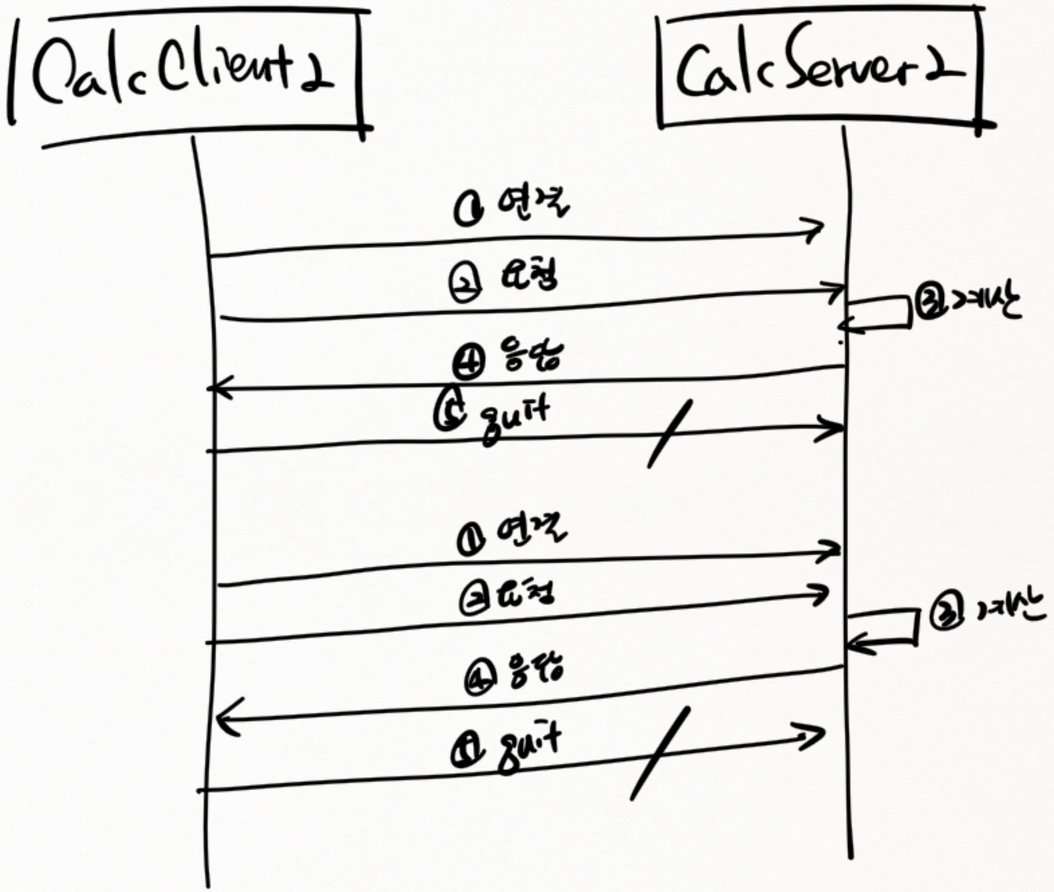
예) HTTP, 메신저
(웹)

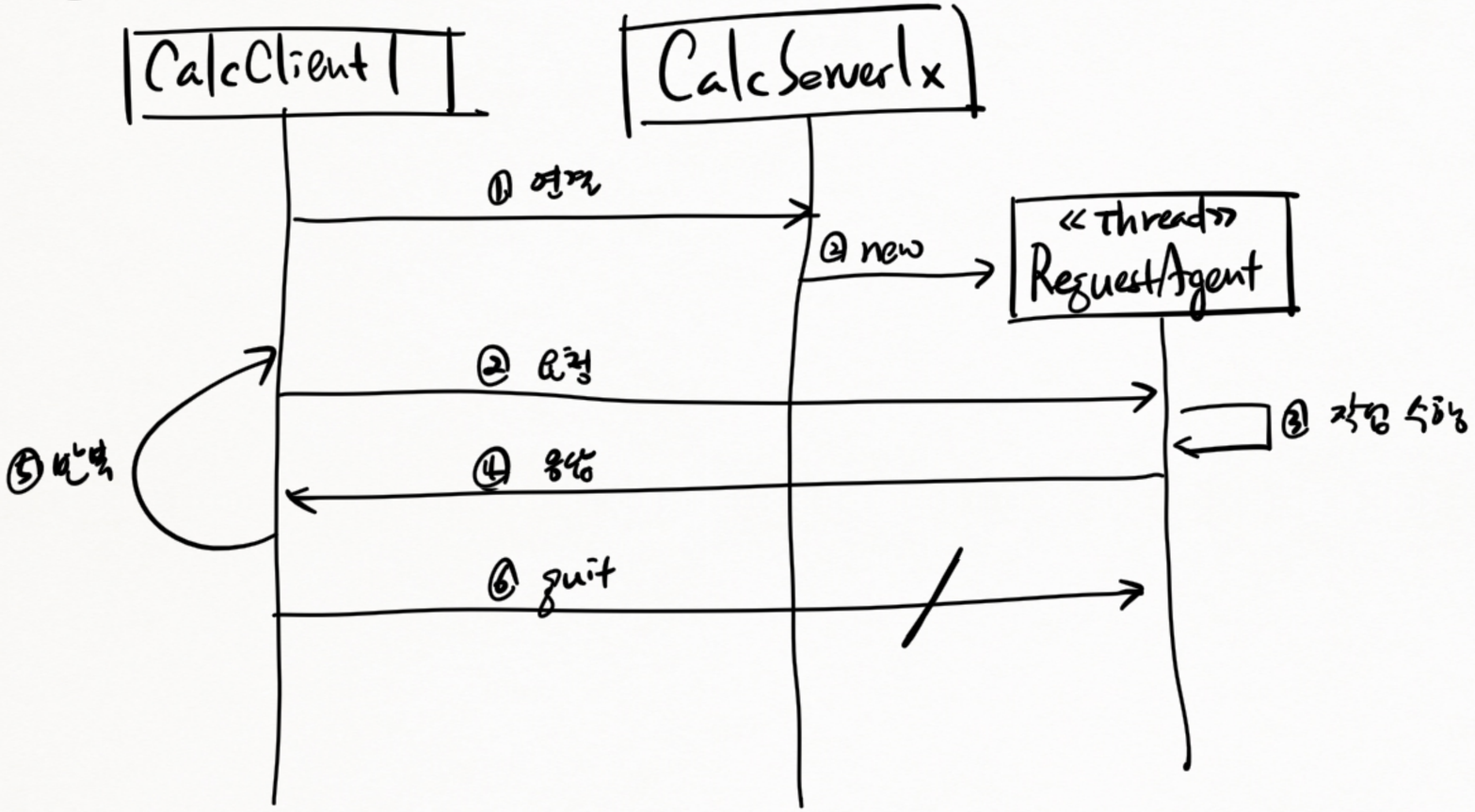
* Stateful vs Stateless

① Stateful



② Stateless



* Stateful vs Stateless

③ Stateful + Thread