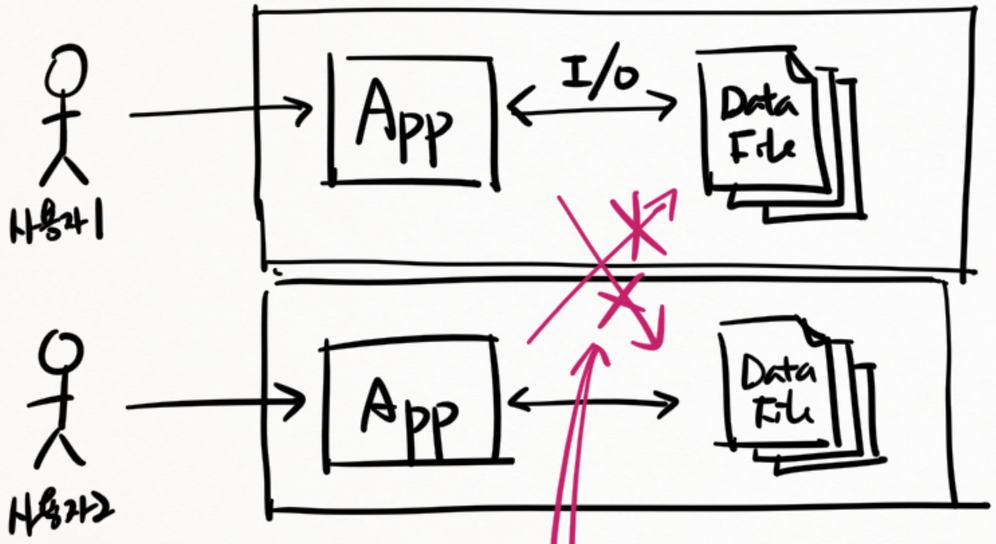
37. 네트워킹을 통한 데이터 공유

① 현황



프로그램 간의 Data File 공유가 안된다.

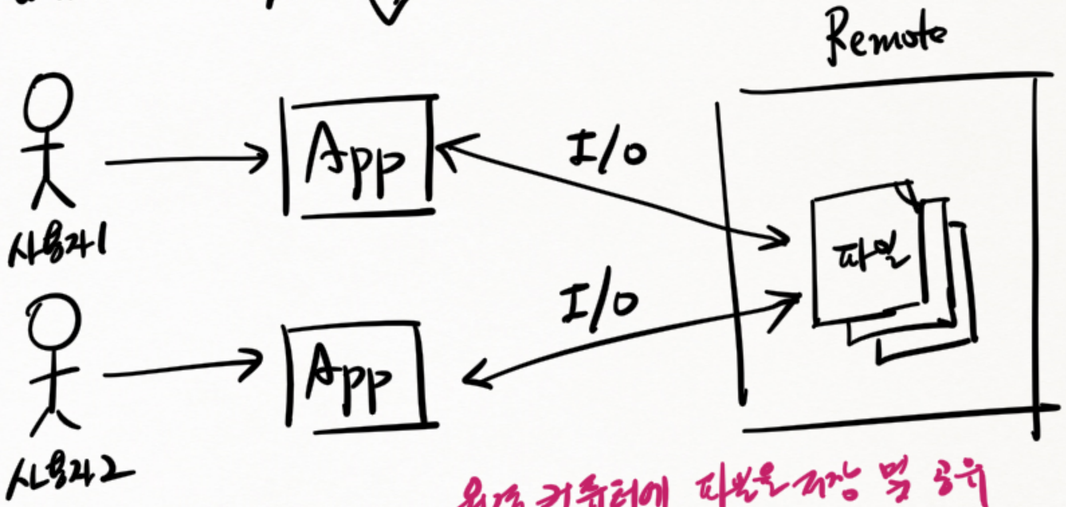
App을 실행하는 컴퓨터

데이터 파일이 로컬에 저장된다.

⇓

App 간에 데이터를 공유할 수 없다!

② 데이터 파일을 별도의 컴퓨터에 보관/공유 ⇓ 해결책?

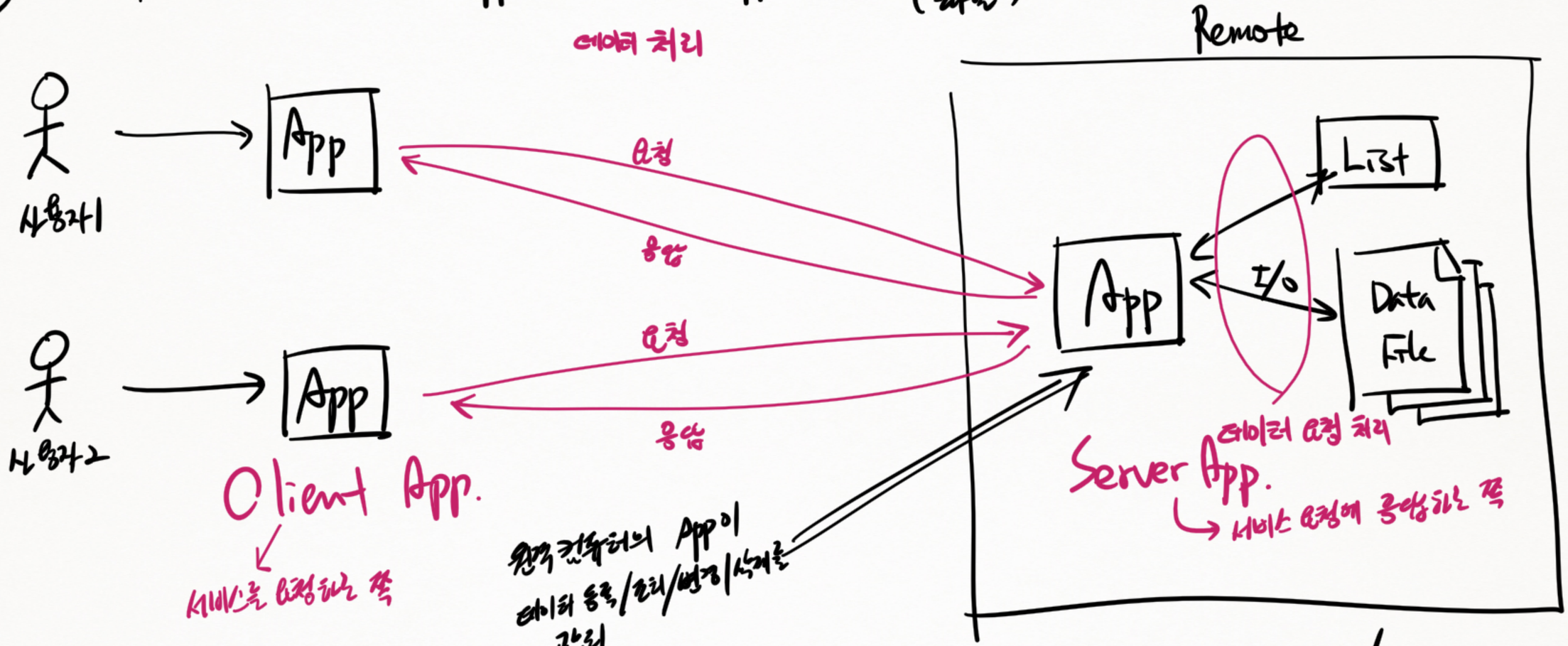


- 원격 컴퓨터에 파일을 저장 및 공유

⇓

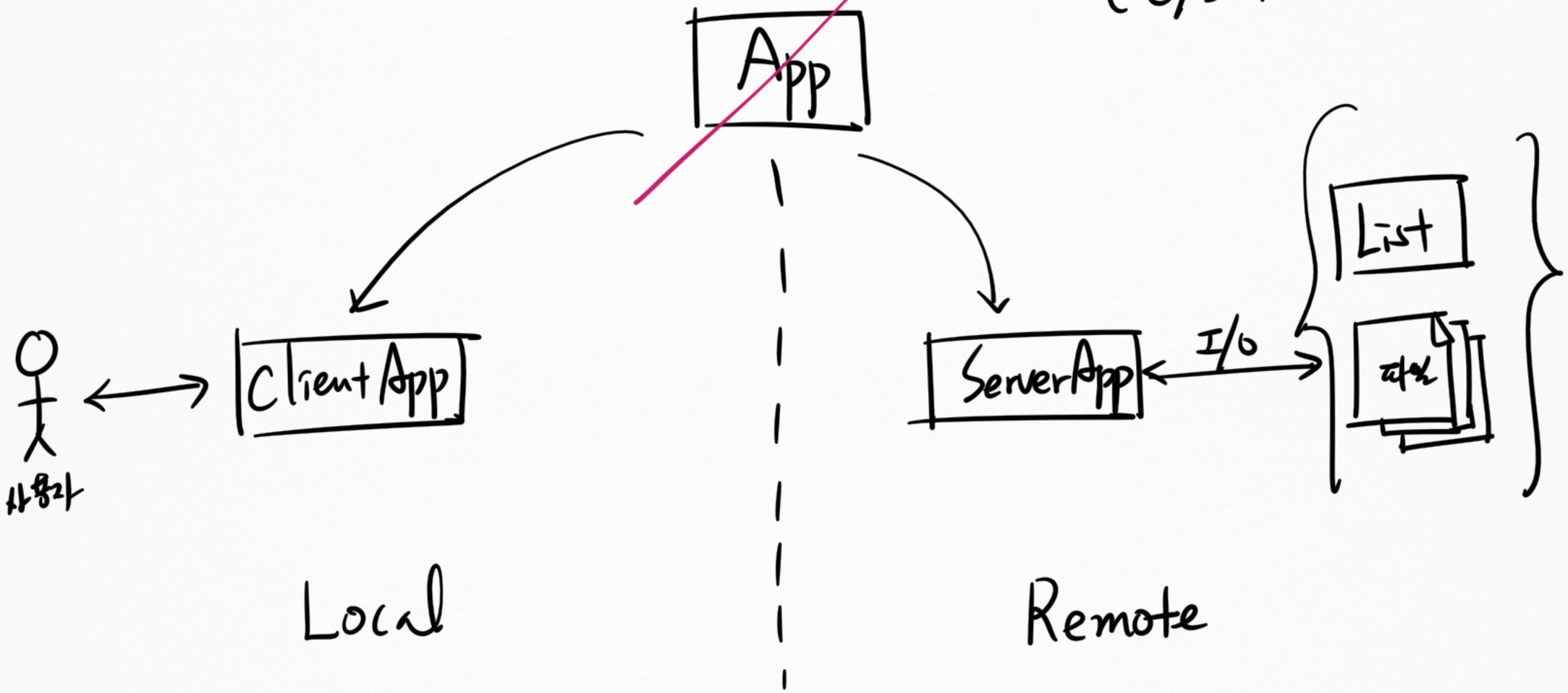
여러 App이 동시에 같은 파일을 접근한다 보면 파일의 데이터가 깨질 수 있다.

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

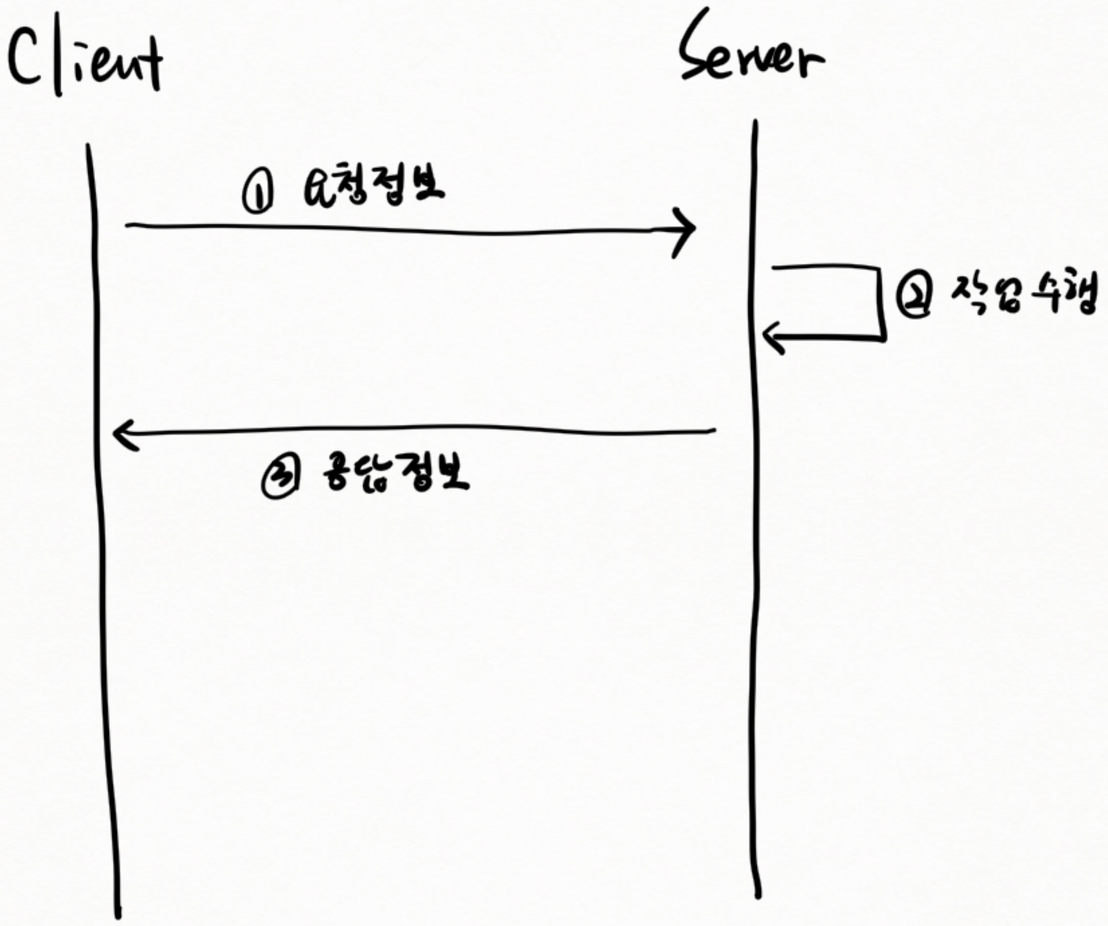


↳ 같은 (파일/데이터)를 동시에 접근할 때 발생할 수 있는 문제를 피할 수 있다!

* 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

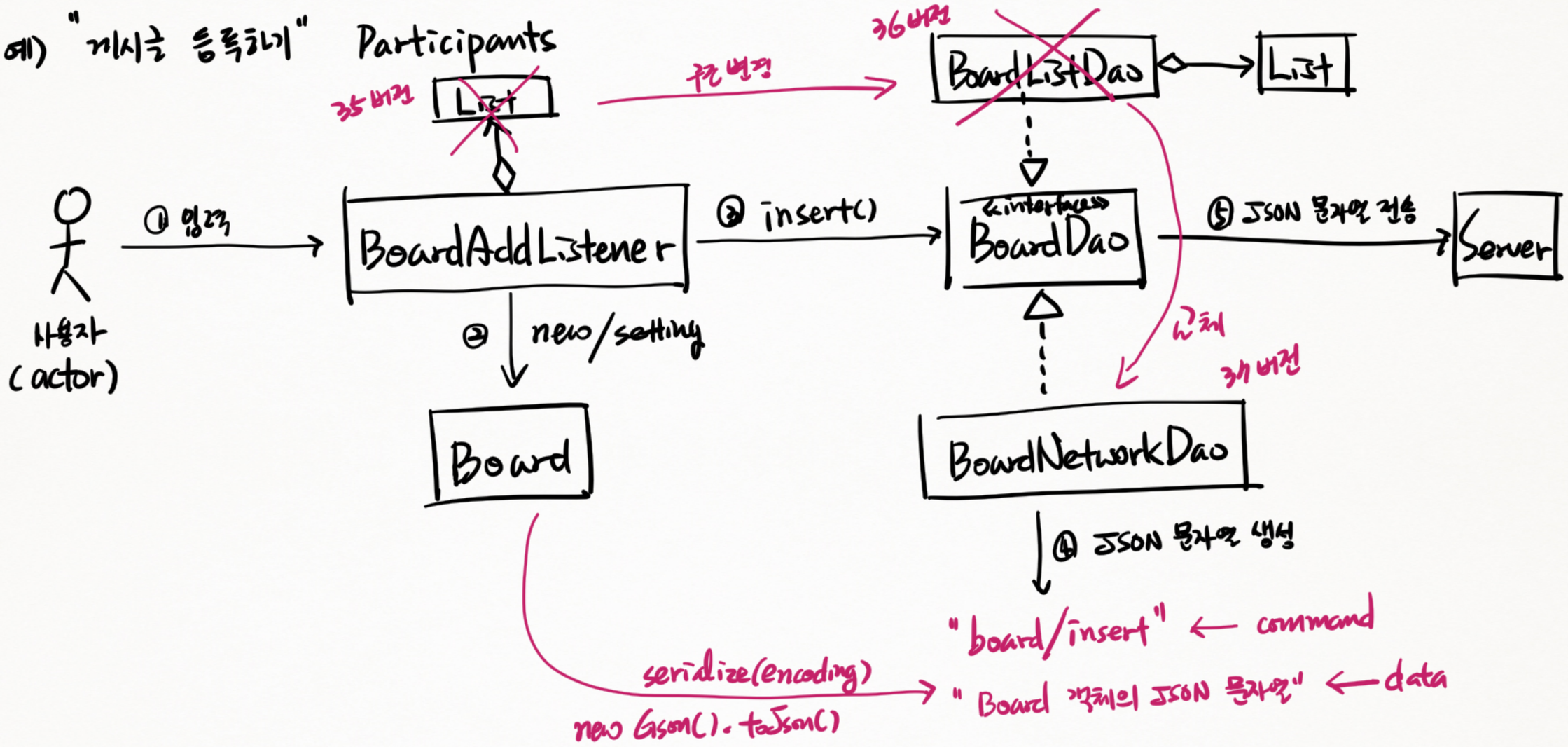
"failure" : 실패

⇓ 실패 에

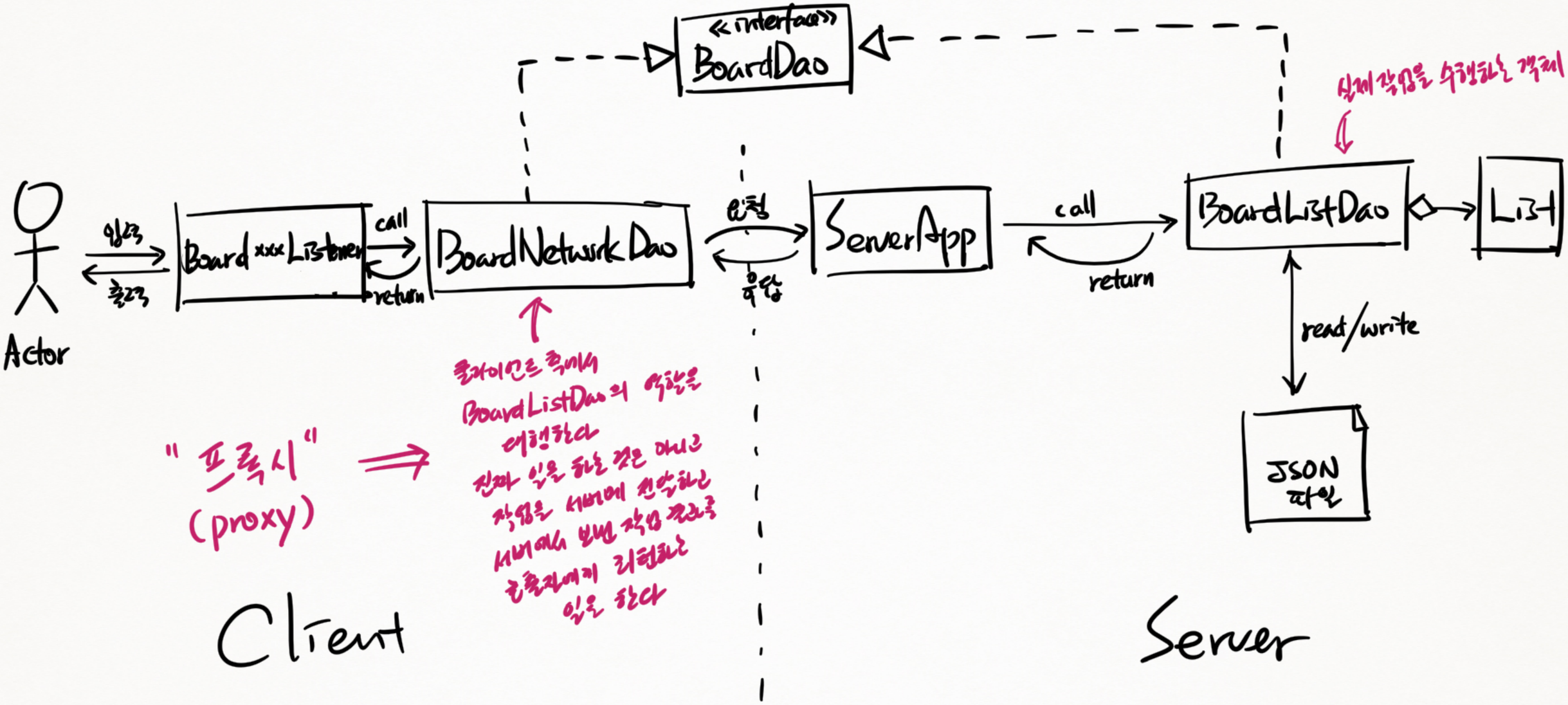
```
{
    "status" : "failure",
    "result" : "해당 번호의 데이터가 없습니다!"
}
```

* 요청 정보 보내기 작업에 참여하는 객체들과 실행 흐름

예) "게시글 등록하기" Participants

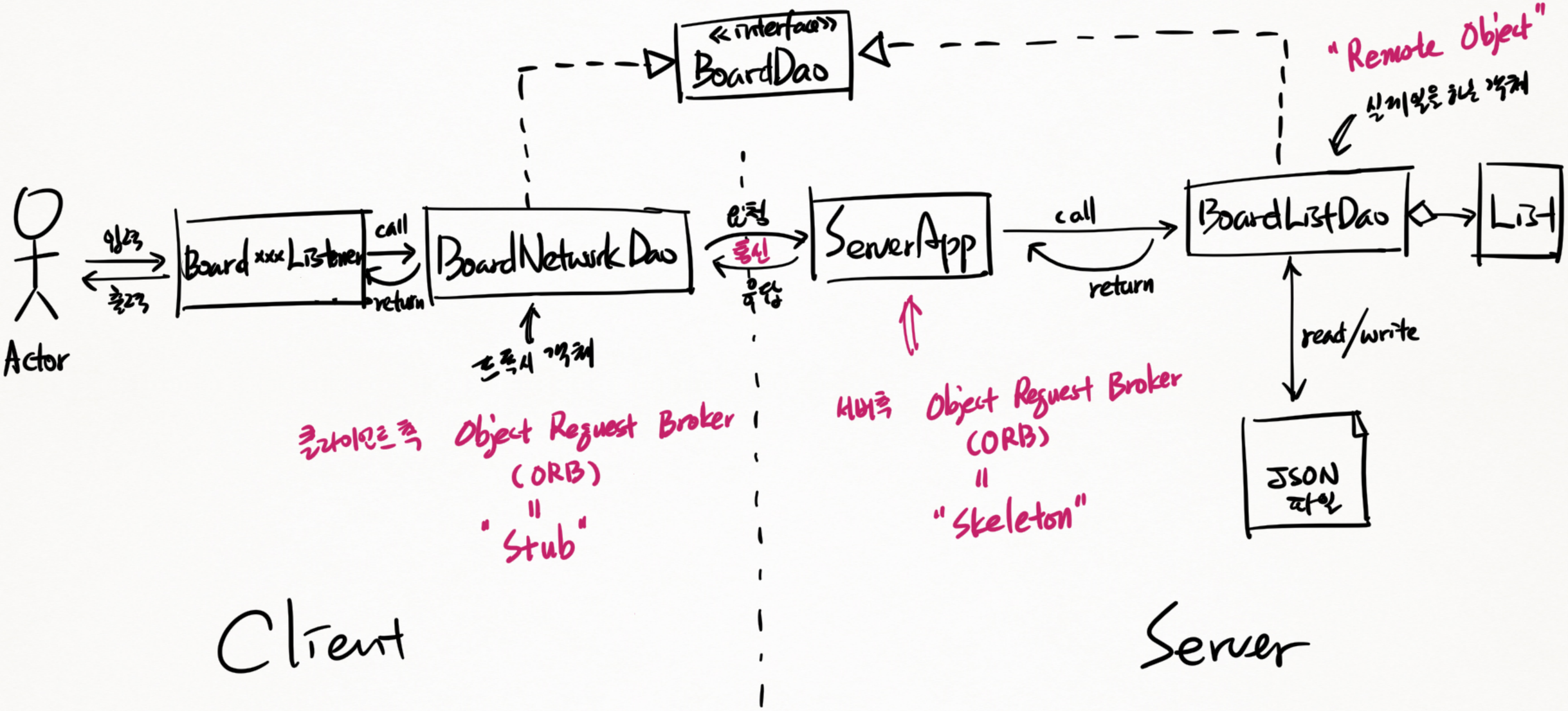


* DAO 와 Proxy 패턴 (GoF)



* 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

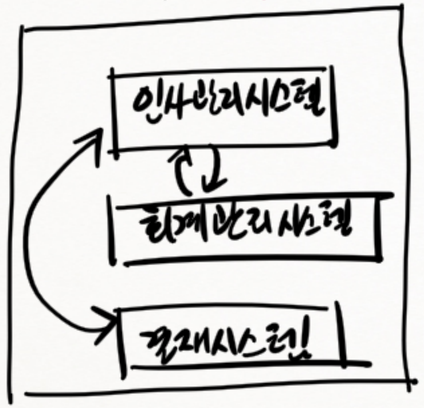
RMI (Remote Method Invocation)

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

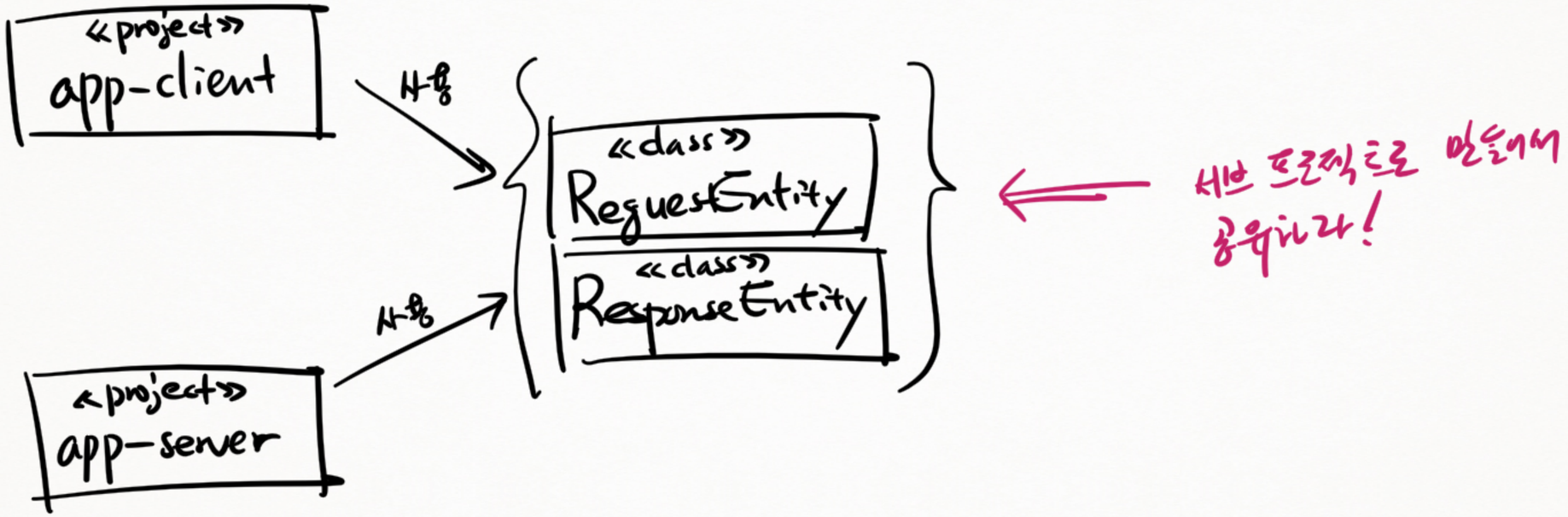
call

call

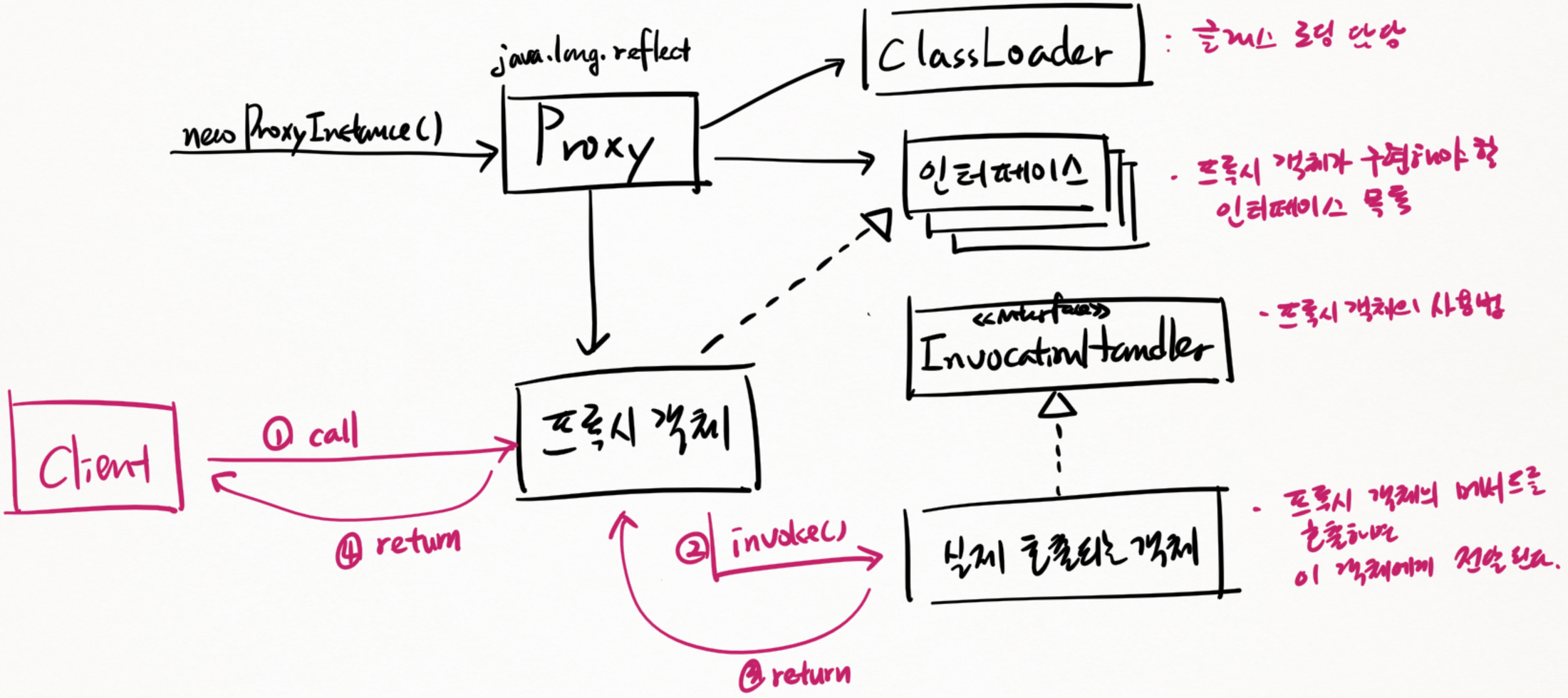
문제점
- 서버에 문제 발생
  ↓
  모든 업무시스템이 정지
- 사용자 증가 → 서버 증설 (RAM, HDD, CPU)
  ↓
  서버 장비는 고가
  ↓
  H/W 유지비가 많이든다.

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

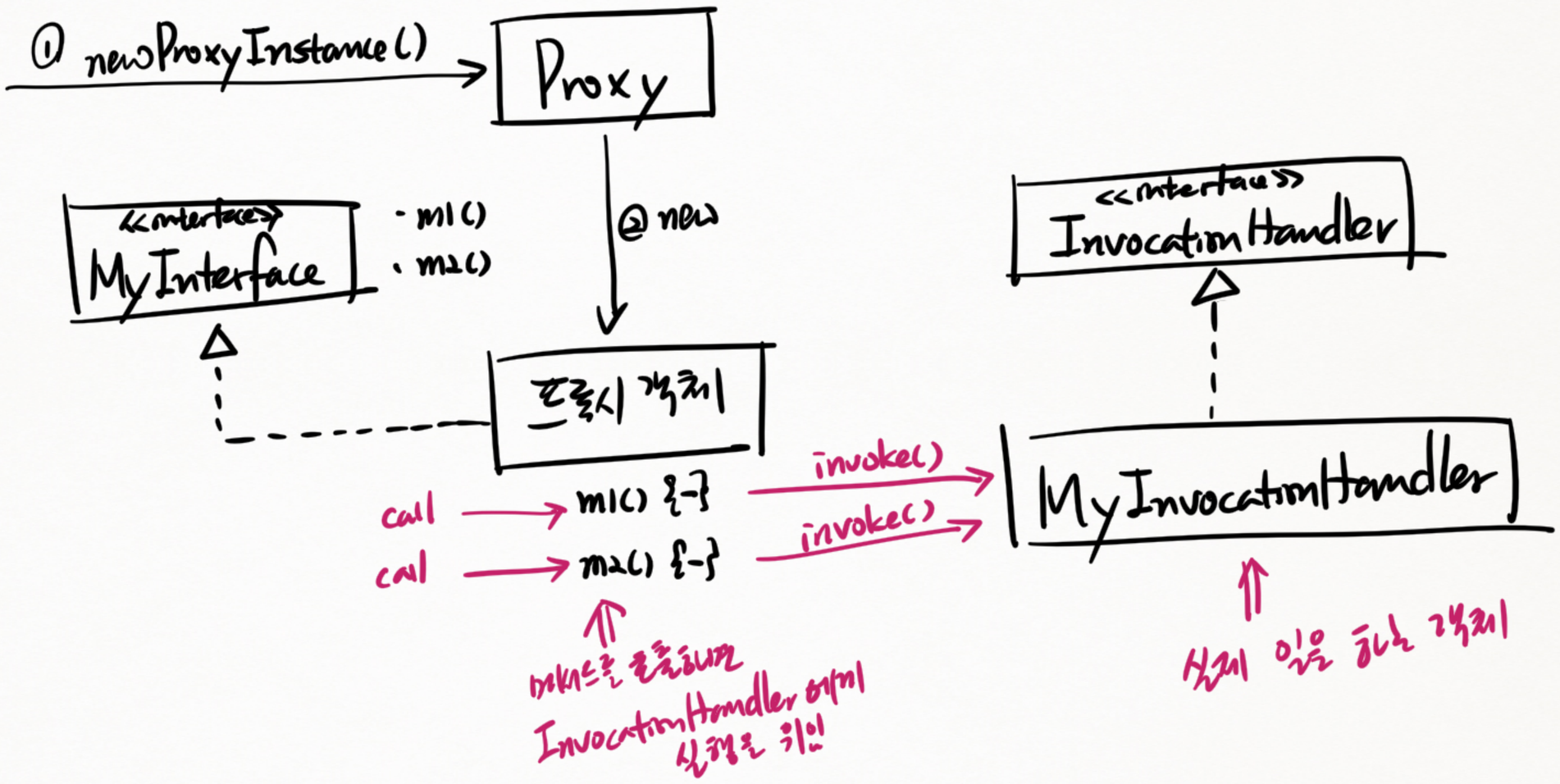
* Project 간에 클래스 공유하기



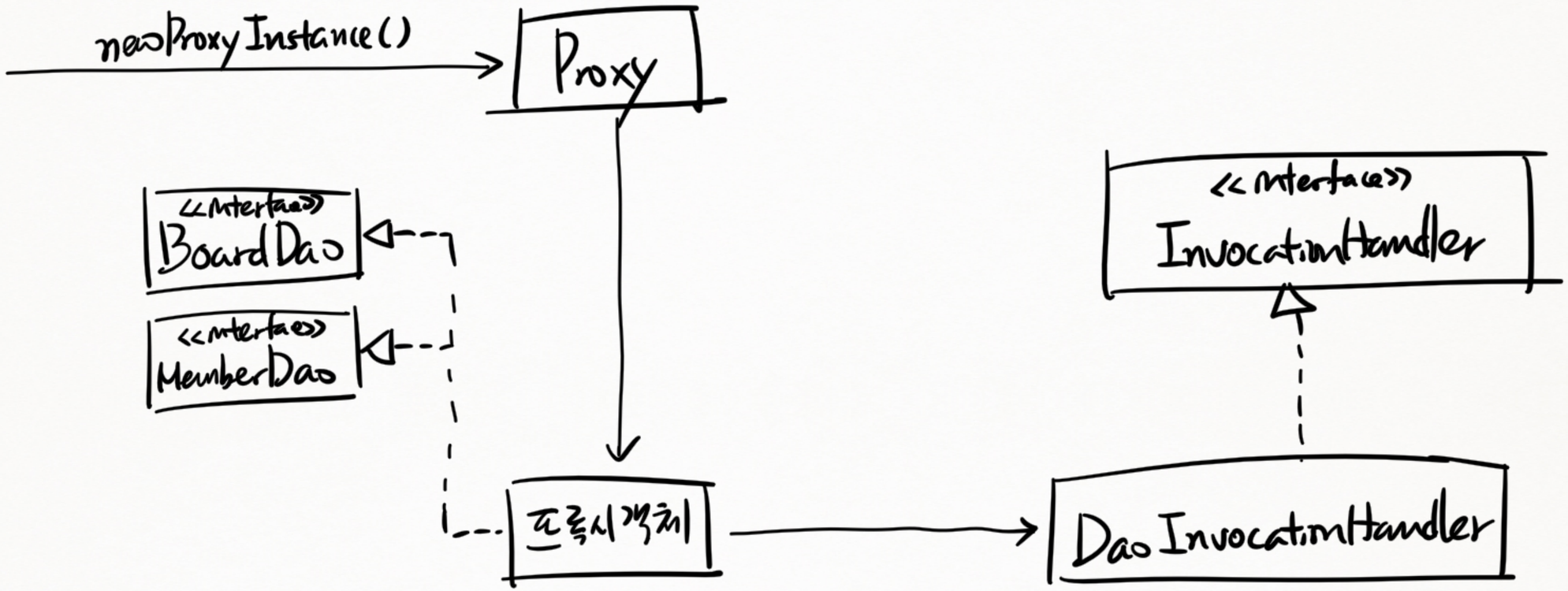
38. 프록시 객체 자동 생성



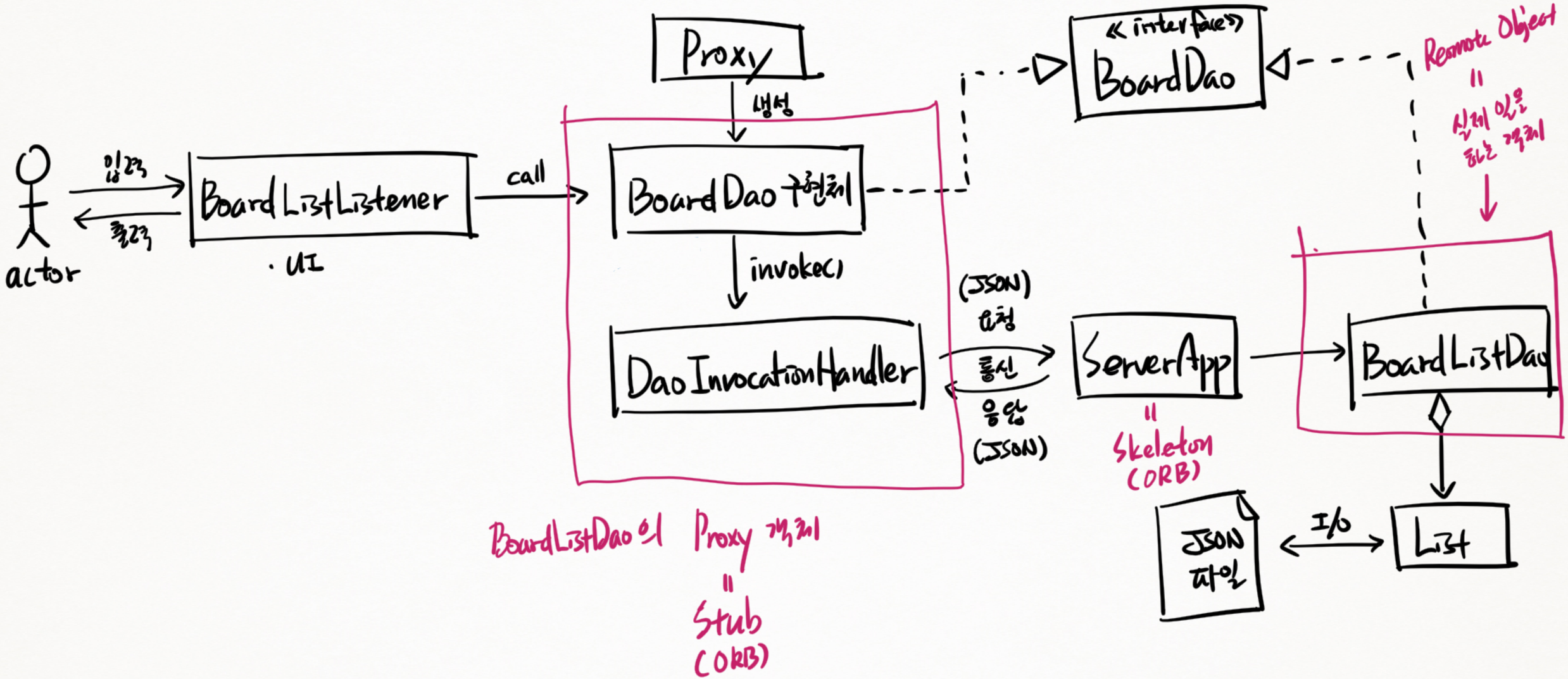
38. 프록시 객체 자동 생성 — 예1



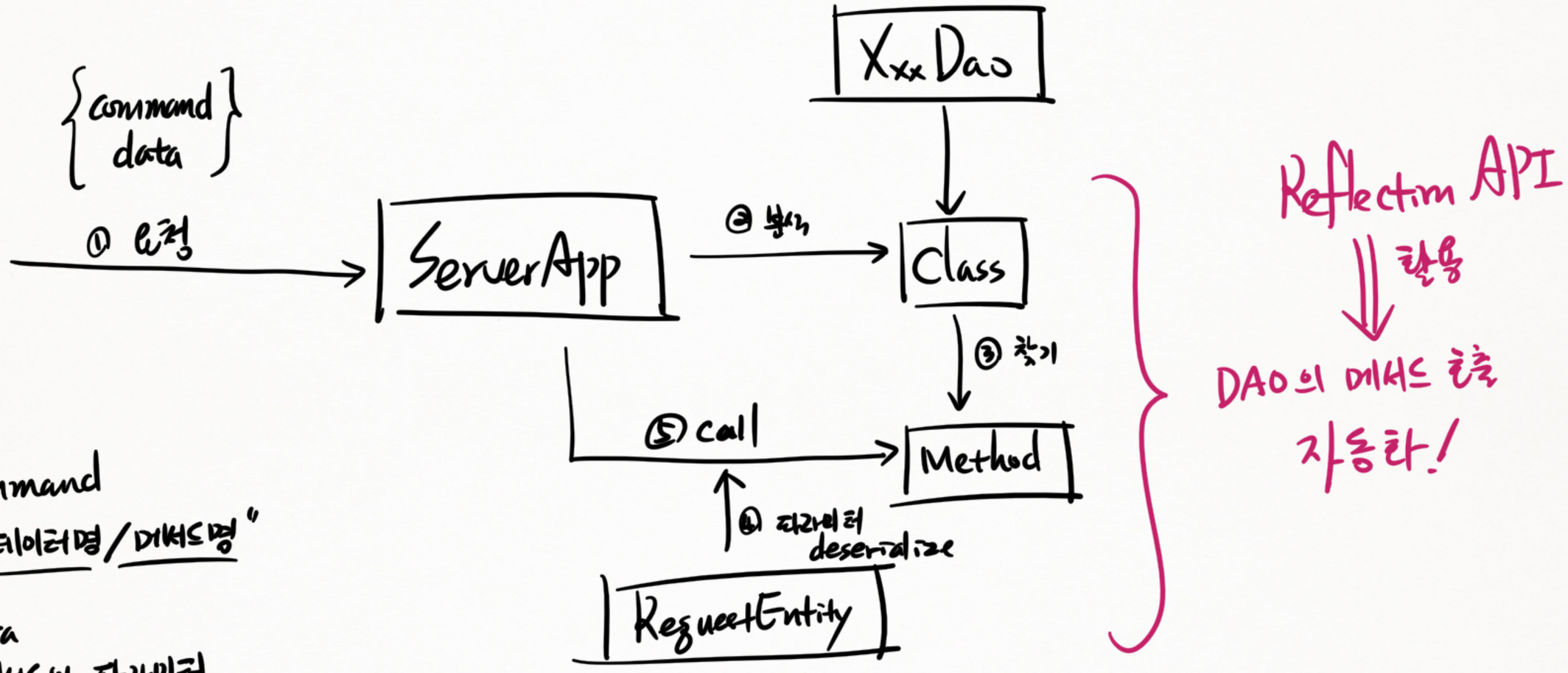
38. 프록시 객체 자동 생성 - Participants



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

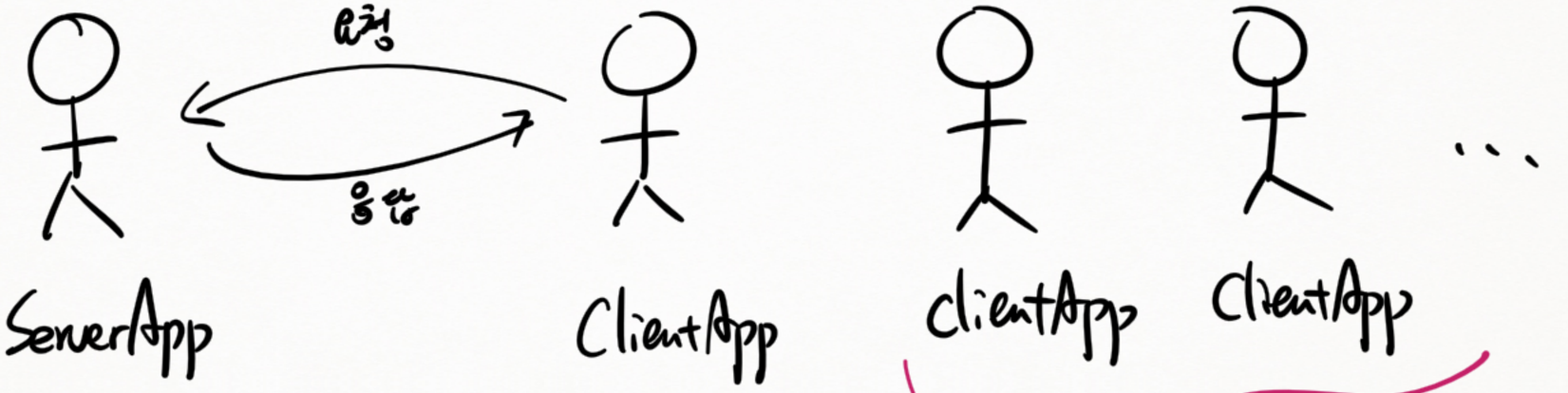
① client 측

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

② server 측

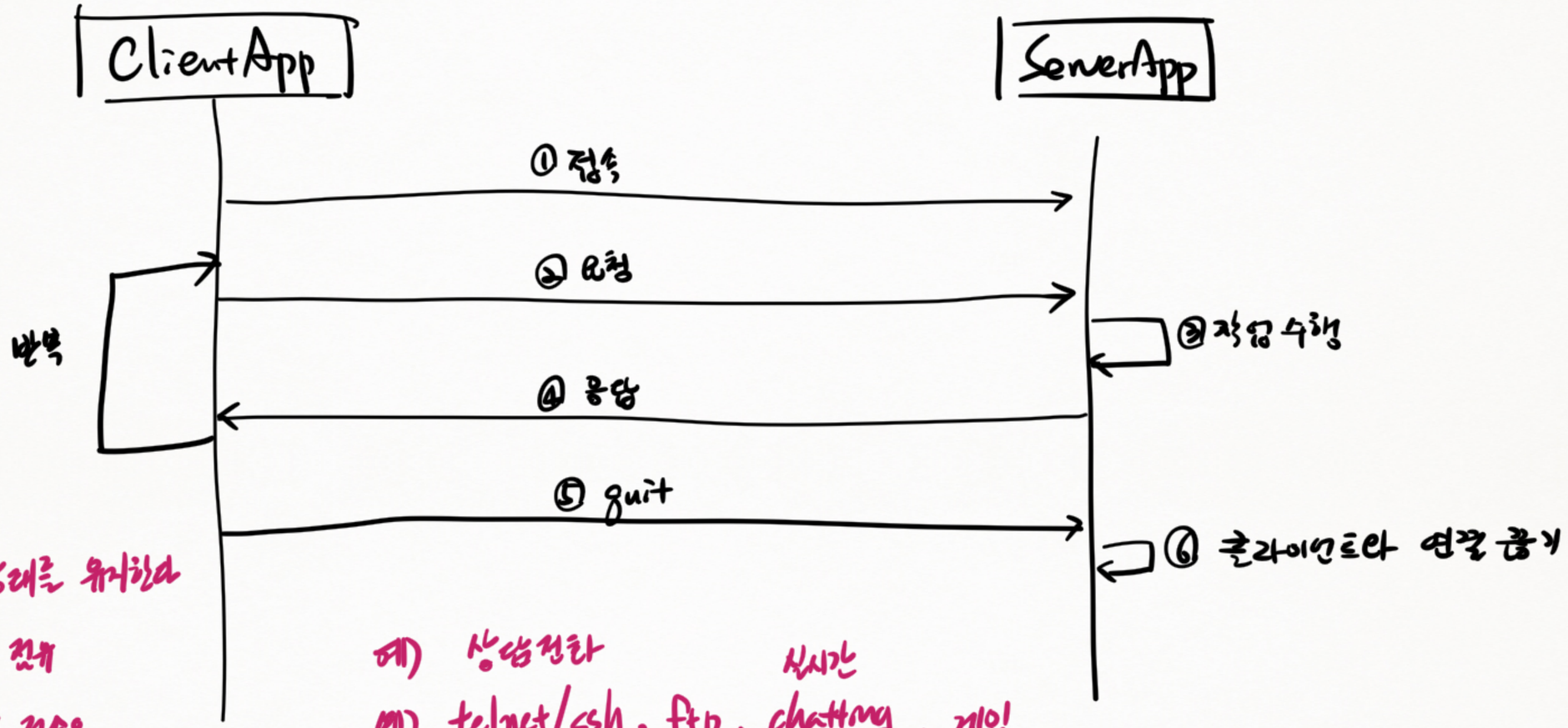
```
try {
    ≡
    dao.invoke(—);
    ≡
} catch (Exception e){
    ≡
}
```

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

↓

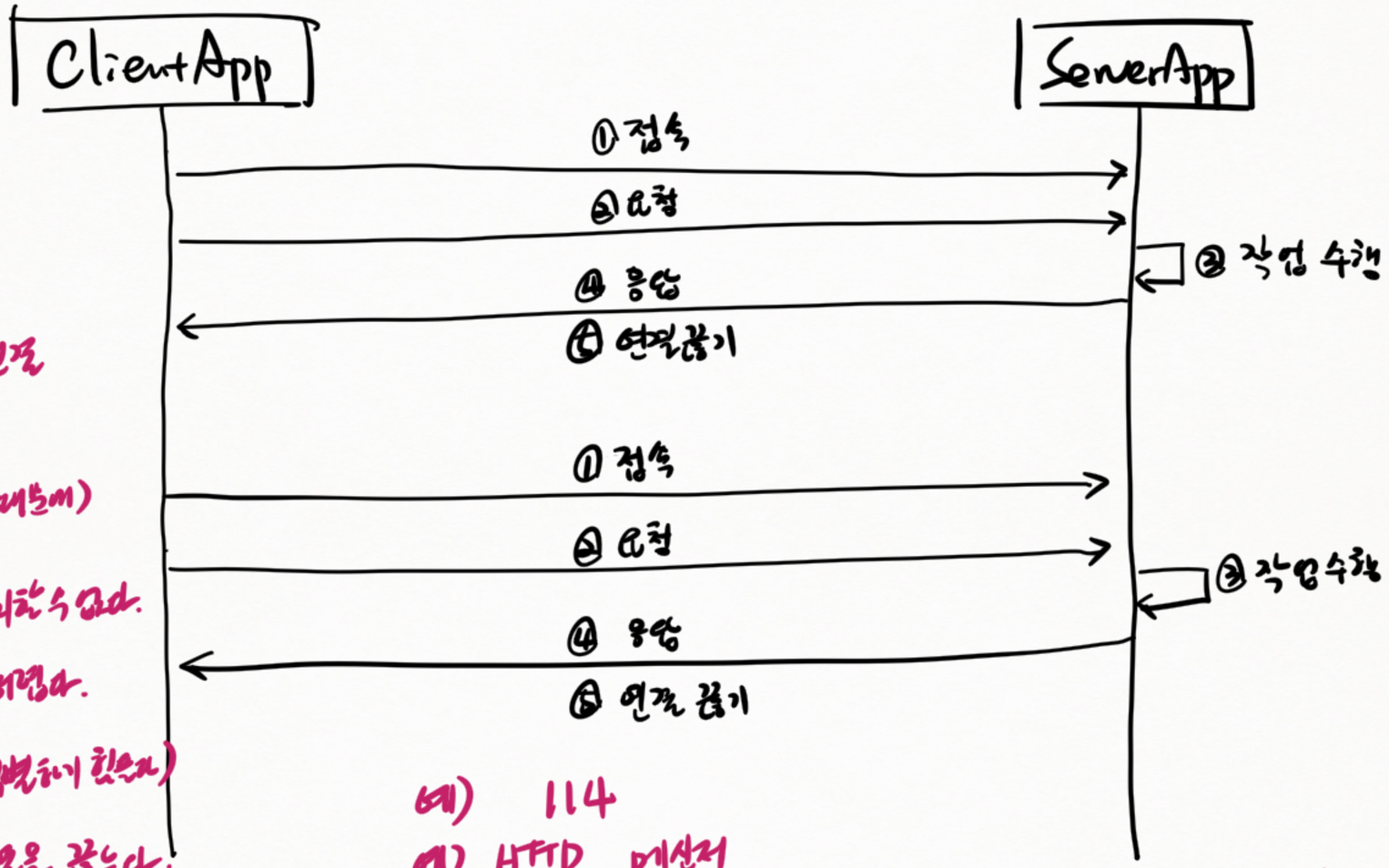
· 서버 자원을 점유

↓

· 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

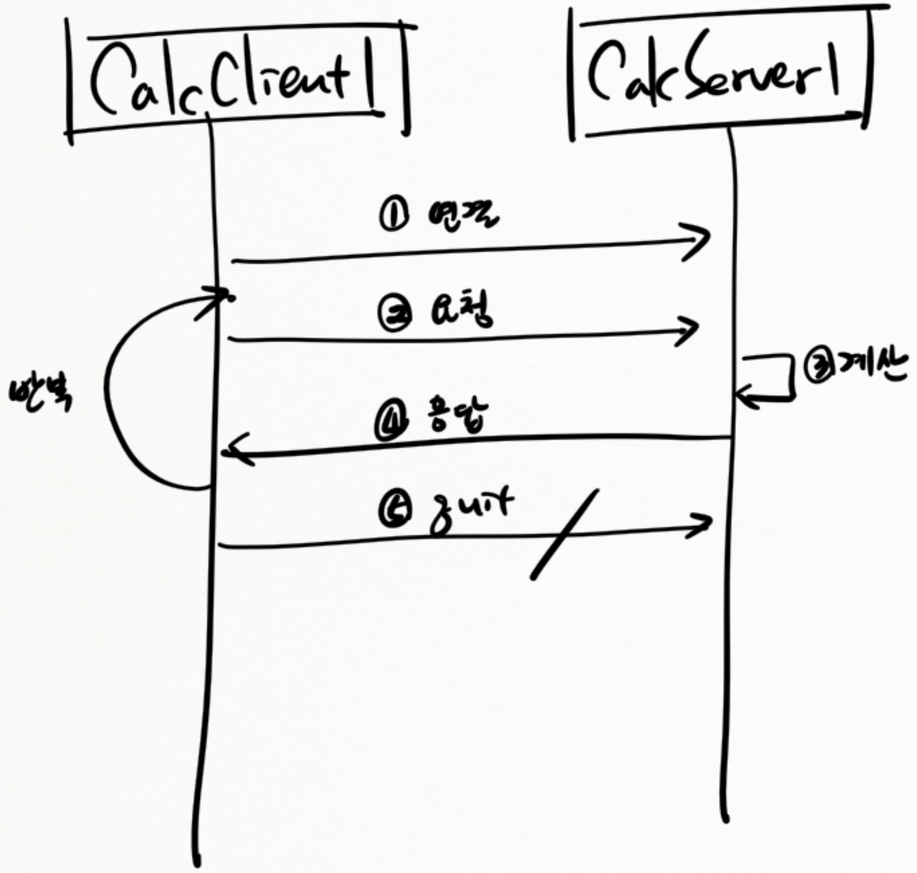
↳ 서버 자원을 덜 사용 → Stateful 방식보다 더 많은 클라이언트 동시 접속을 처리할 수 있다.

예) 114

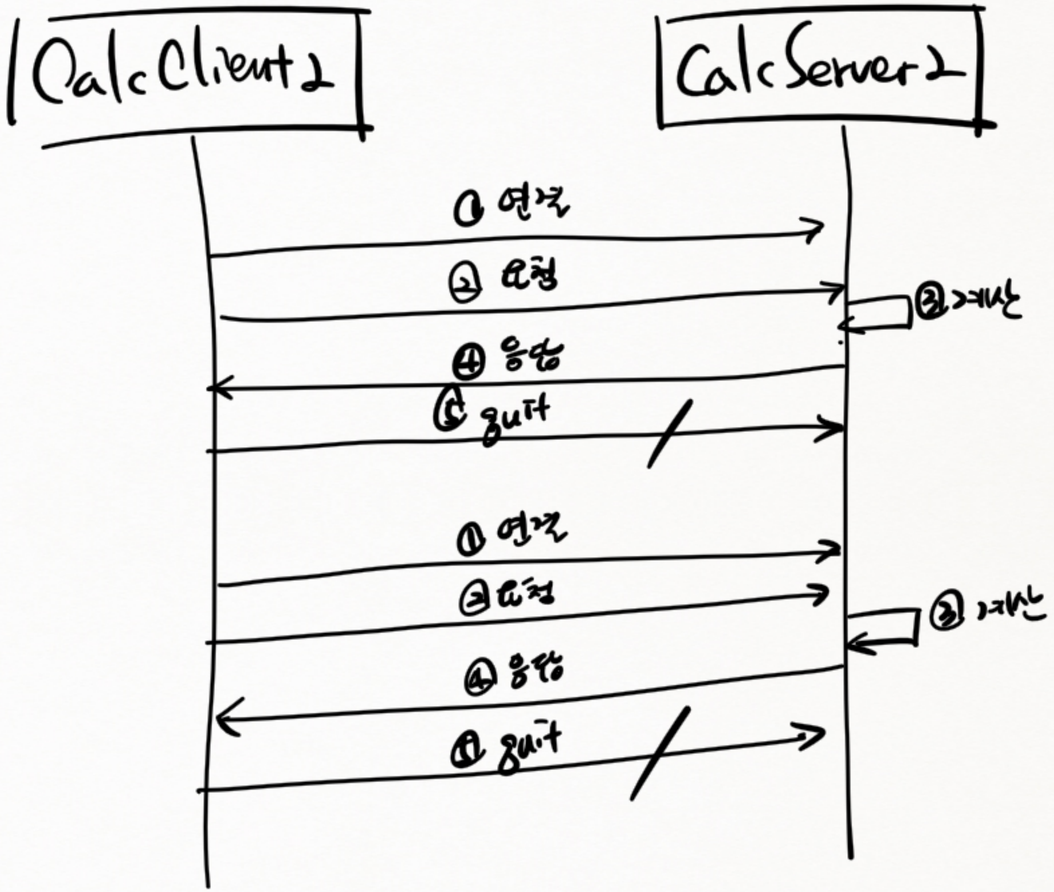
예) HTTP, 메신저
(웹)

* Stateful vs Stateless

① Stateful

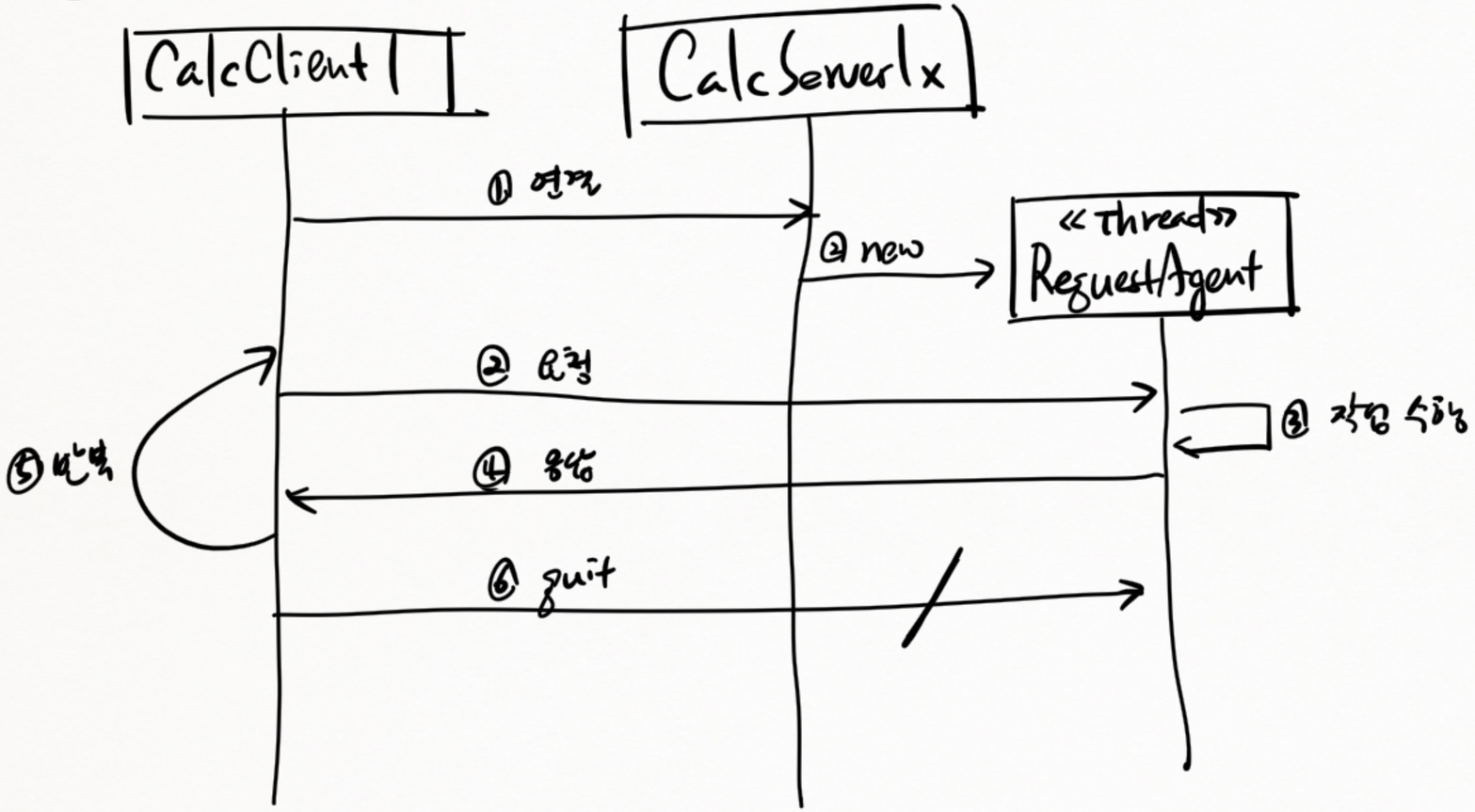


② Stateless



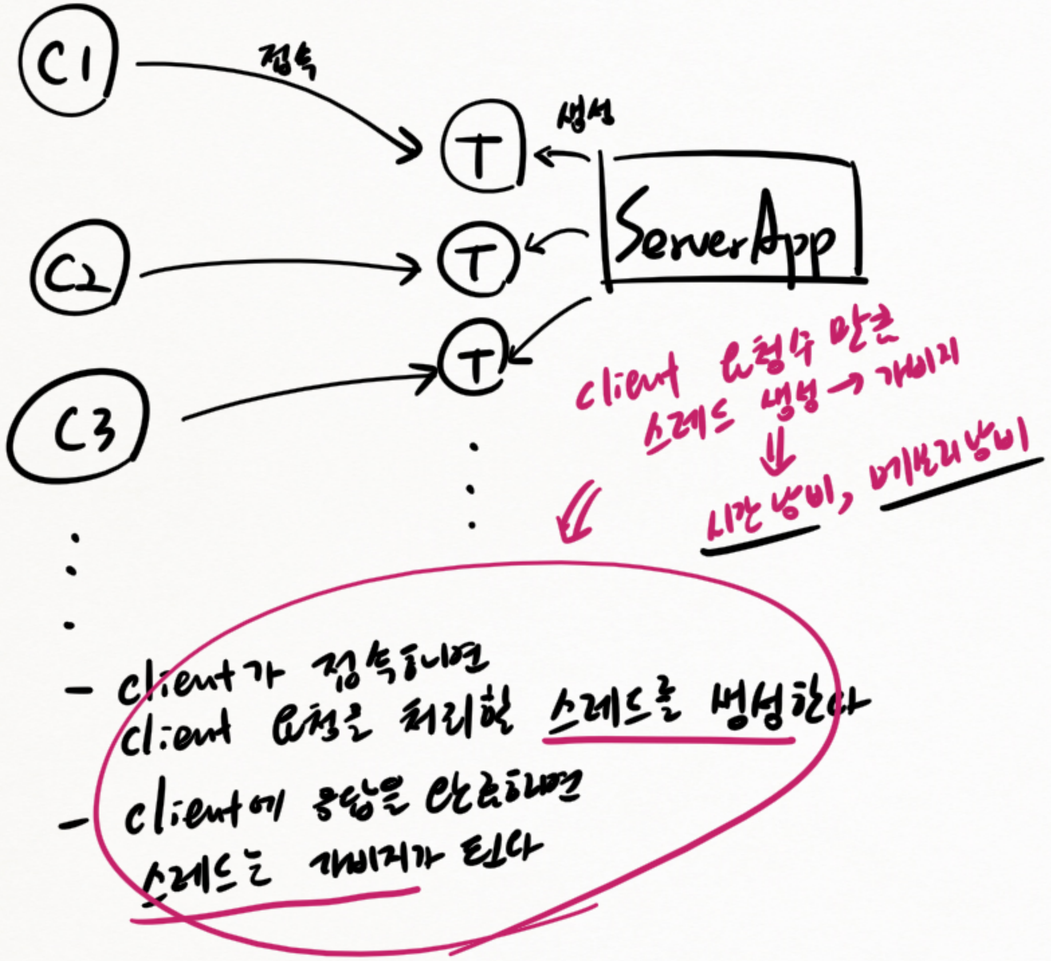
* Stateful vs Stateless

③ Stateful + Thread



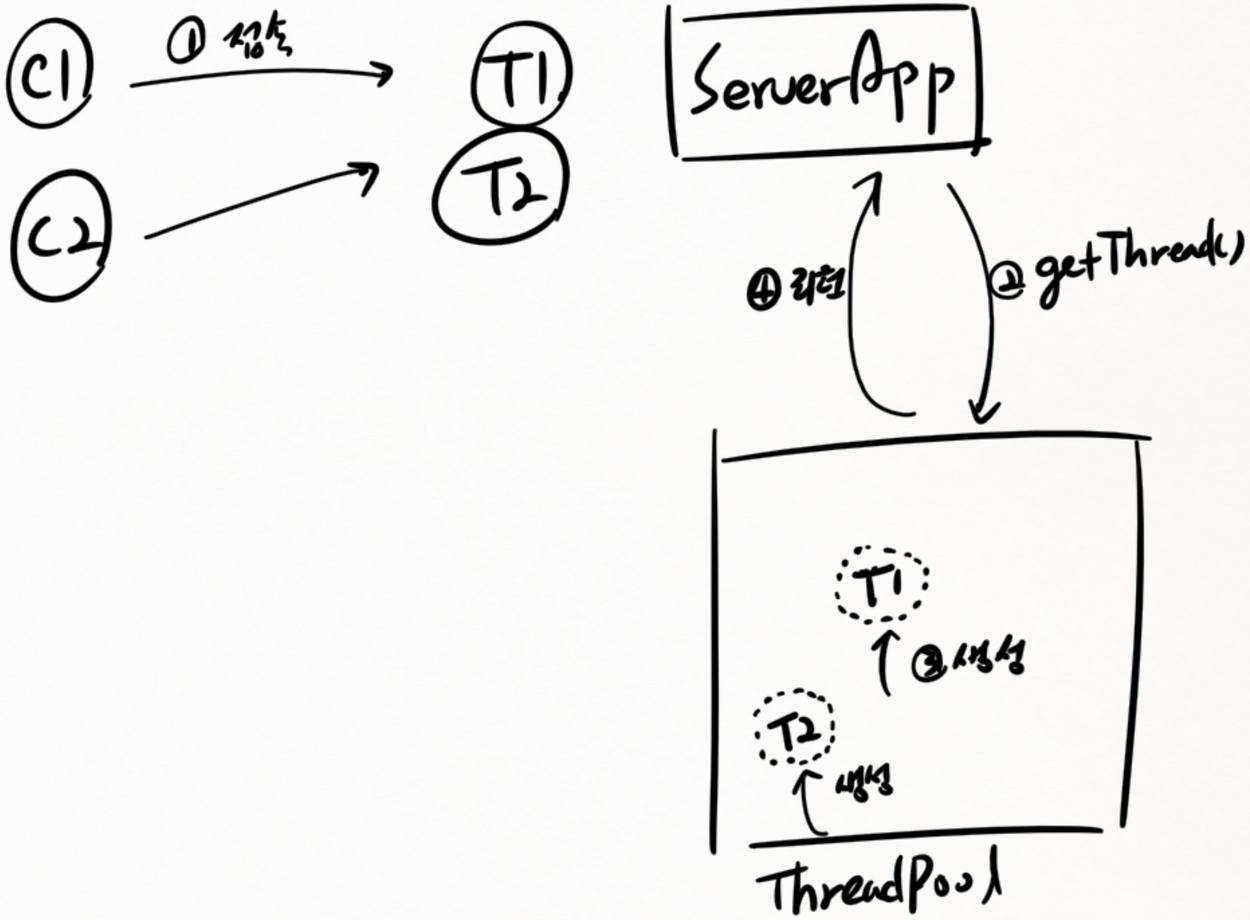
4. 스레드 재사용하기 : GoF의 Flyweight 적용

① 스레드풀 적용 전



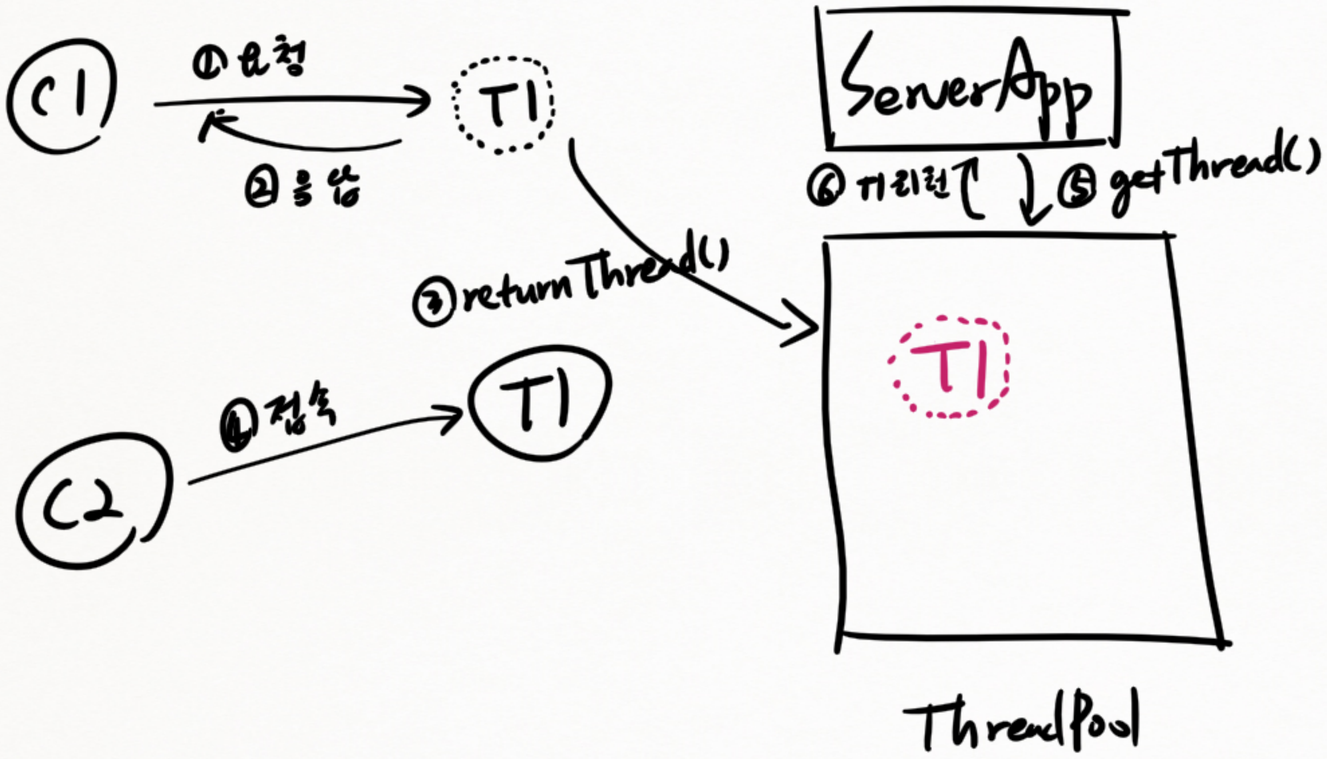
- client가 접속하면
  client 요청을 처리할 스레드를 생성한다
- client에 응답을 완료하면
  스레드는 가비지가 된다

② 스레드풀 적용 후



4. 스레드 재사용하기 : GoF의 Flyweight 적용

③ 스레드 반납



{ · 객체 생성 시간이 오래 걸리는 경우
· 동일한 객체를 반복적으로 사용하는 경우 }

⇓

- 객체 사용후 가비지로 버리지 않고 컬렉션에 보관.
- 객체가 필요할 때, 컬렉션에 보관된 객체를 리턴함으로써 "객체 재사용!"

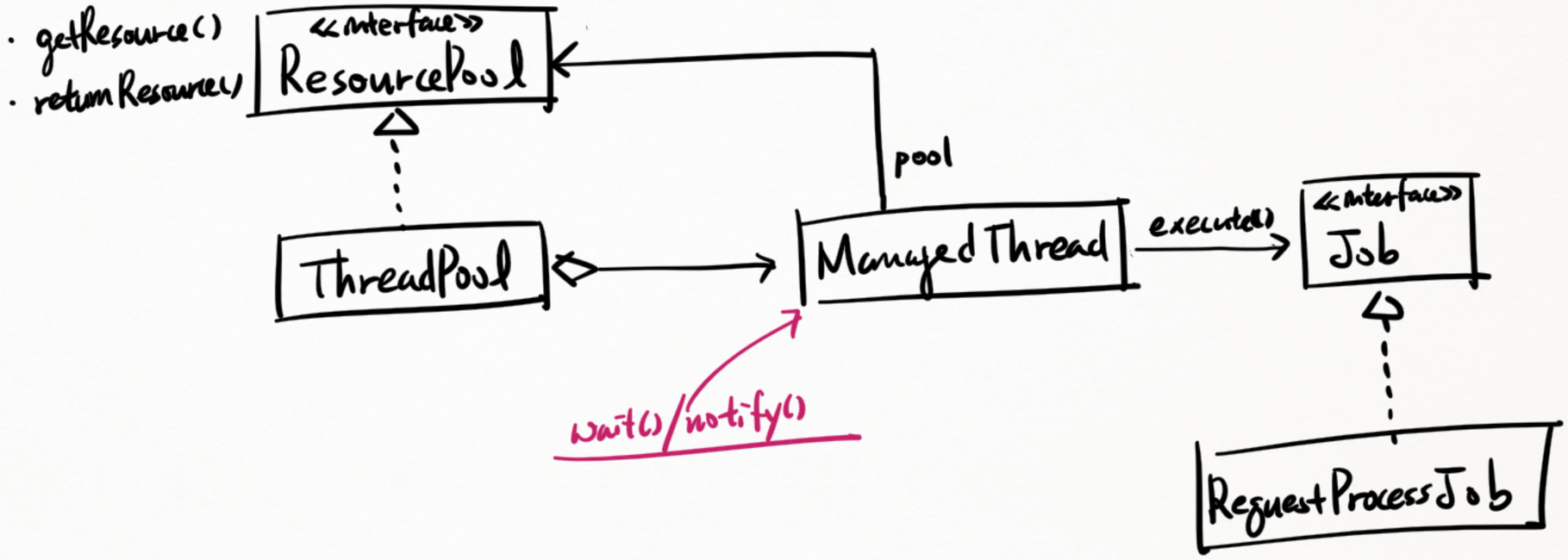
⇓

"Flyweight 패턴"

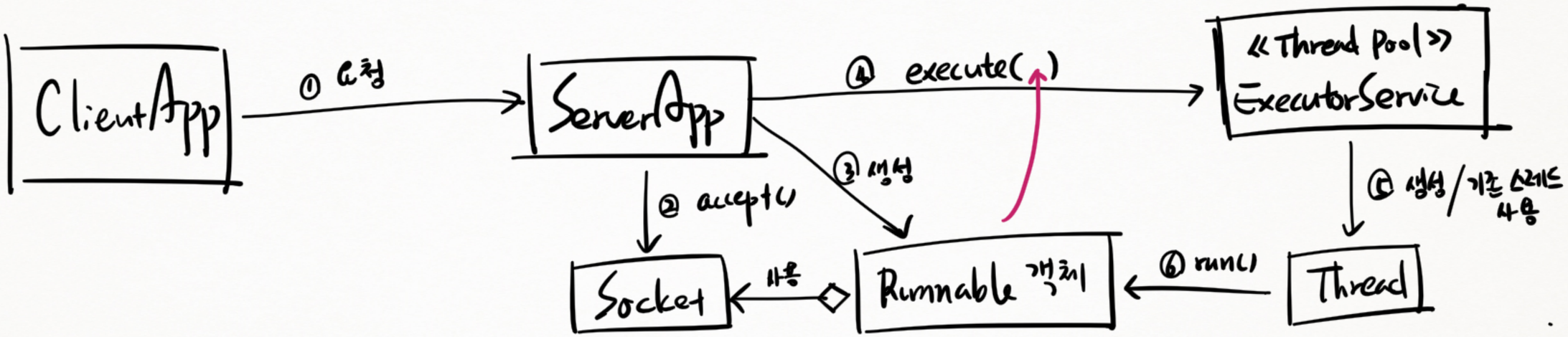
||

"Pooling 기법"

* Thread Pool Class Diagram

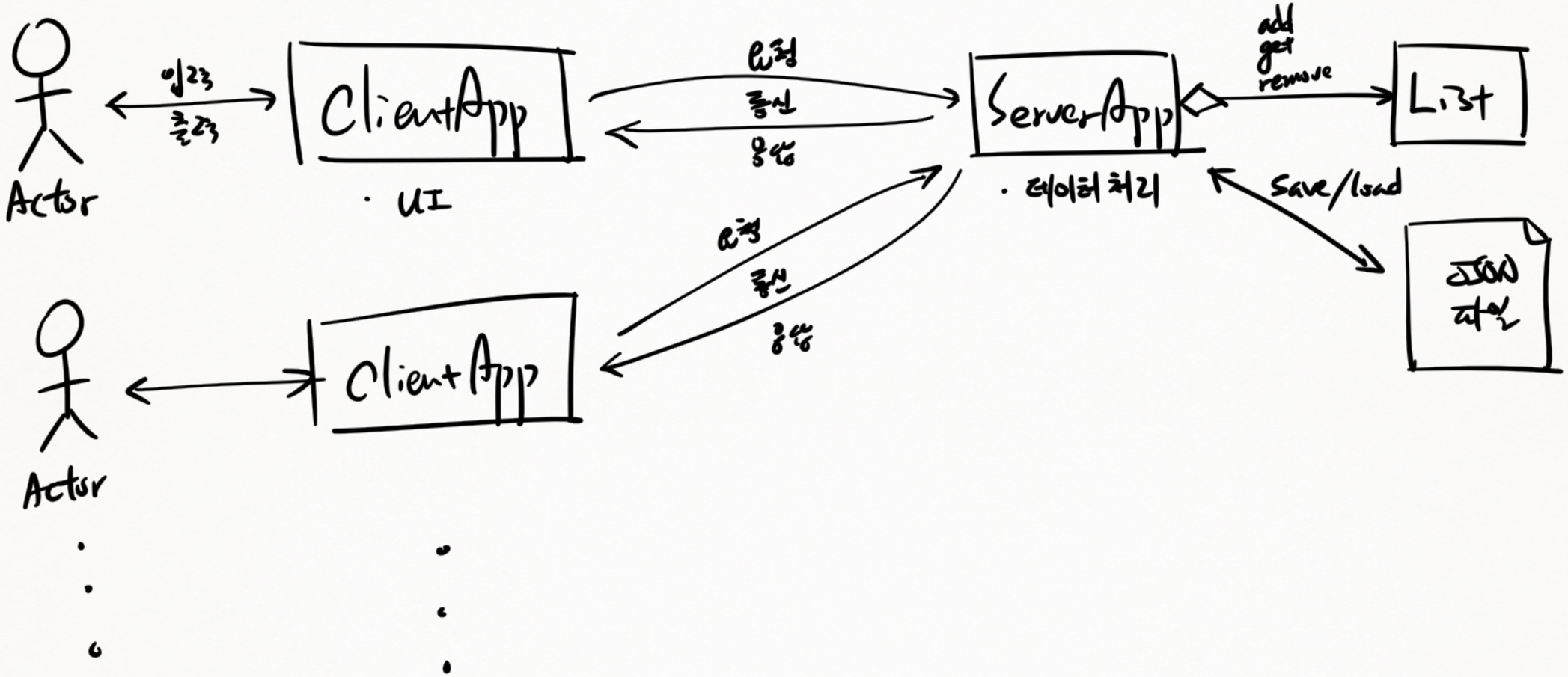


4. 자바 스레드를 작용하기 - Executors/ExecutorService 사용법



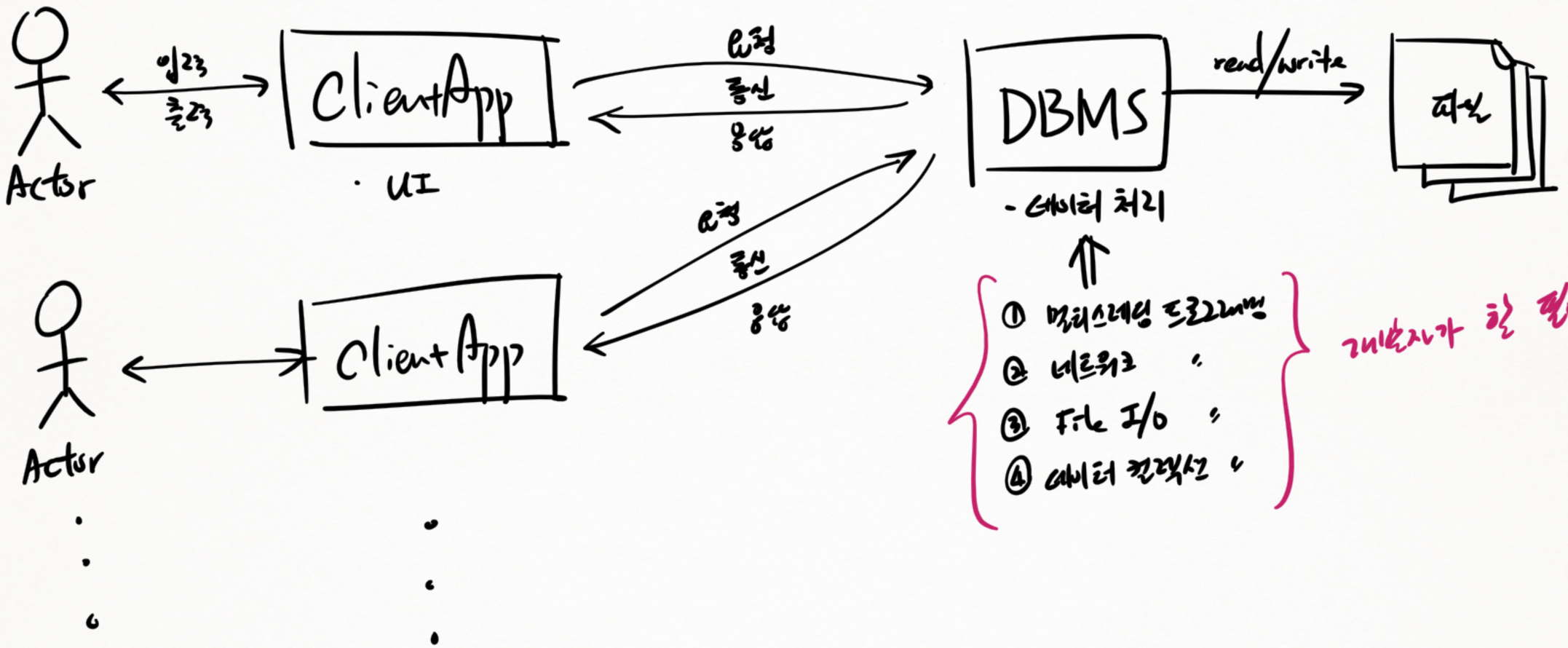
46. DBMS 도입하기

① 비전 방식



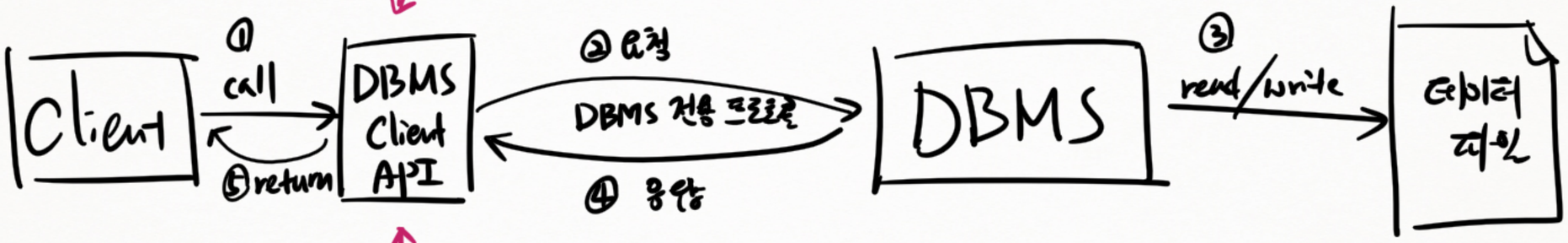
46. DBMS 도입하기

② 개선 방식



* DBMS 와 통신

JDBC 명세에 따라 만든 API 드라이버 의미로
"JDBC Driver"
→ JDBC 규칙에 따라 DBMS와의 통신을 이끈다는 의미!



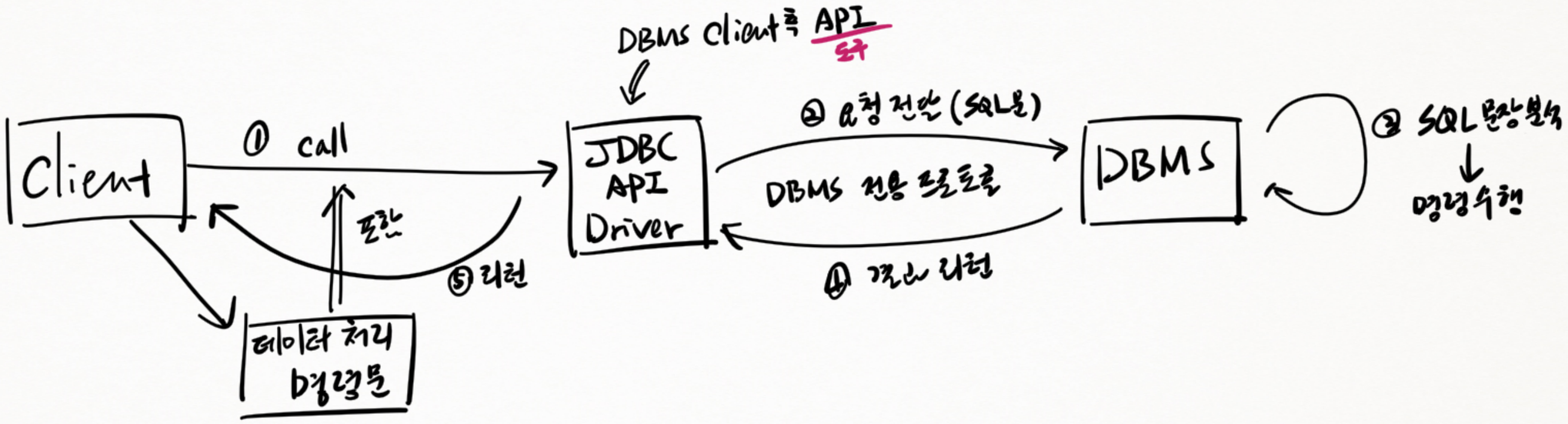
클라이언트 개발자는 통신 프로토콜을 알 필요가 없다.

- DBMS와의 통신을 담당
- DBMS 전용 프로토콜에 따라 통신
- DBMS에 상관없이 API 사용법을 통일시키기 위해 자바에서 정의한 JDBC 규칙에 따라 만들었음.
- DBMS 제조사에서 제공해준다.

* JDBC?

Java
Database
Connectivity

* DBMS에 데이터 처리를 요청하기



SQL 문법에 맞춰 작성한다.

* SQL

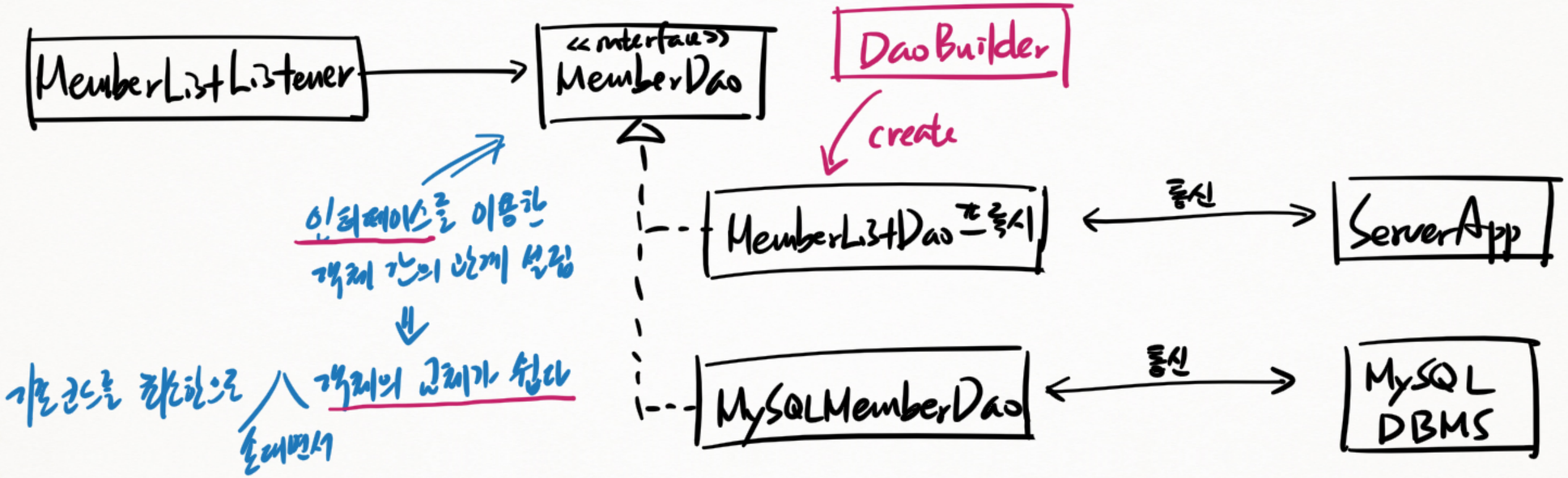
Structured Query Language

구조적인 문법을 갖춘 데이터 처리 명령 언어

· 표준언어
↳ DBMS 독립적이다.
(DBMS에 상관없이 사용할 수 있다.

⟹ 실수 : 표준SQL + DBMS 전용 쓰임

* DAO 와 DBMS


MemberListListener
<<interface>>
MemberDao
DaoBuilder
create
인터페이스를 이용한
객체 간의 관계 설립
⇓
객체의 교체가 쉽다
기존 코드를 확인하으로
훼손되면서
MemberListDao 프록시
통신
ServerApp
MySQLMemberDao
통신
MySQL
DBMS

47. SQL 삽입 공격

① 입력 값으로 SQL 생성할 때

```
update myapp_board set
    title='%s',
    content='%s'
where category=%d
  and board_no=%d
  and password='%s'
```

'1111' or '1'='1'
입력한 내용

⇒ and password='1111' or '1'='1'
SQL 조작 가능

"SQL 삽입 공격"

사용자가 값을 입력할 때
SQL 문법에 영향을 끼치는
문장을 삽입하여 공격하는 해킹기법

47. SQL 삽입 방식 - 해커로부터!

② 입력값과 SQL문을 분리하라

```
update myapp_board set
    title = ?,
    content = ?
where category = ?
  and board_no = ?
  and password = ?
```

값을 넣을 자리에 ?로 표시한다.

* in-parameter 값 설정

```
PreparedStatement pstmt = con.prepareStatement(SQL);

pstmt.setXxx(1, 값);
```

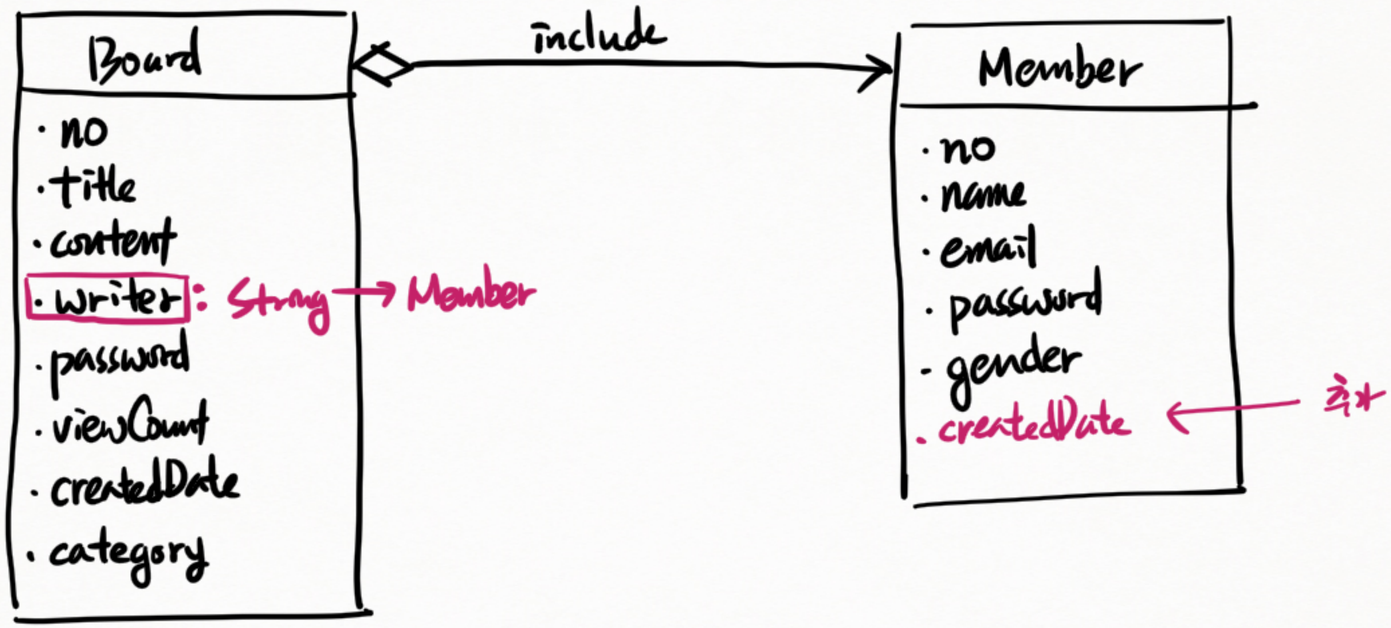
값 type

in-parameter 번호(1부터 시작)

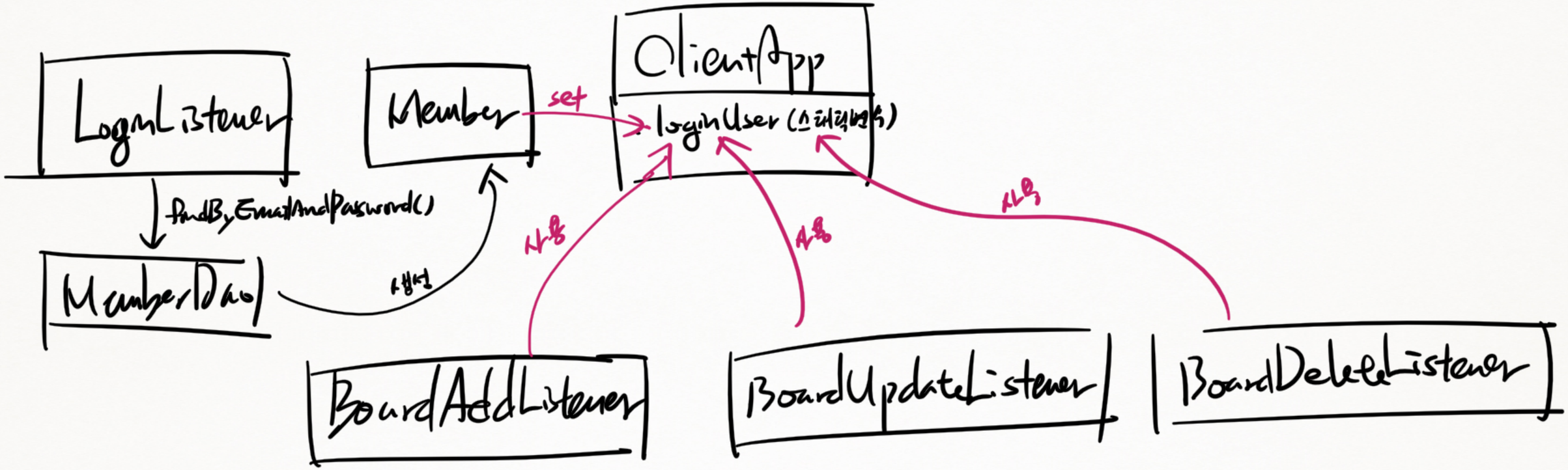
in-parameter에 들어갈 값

48. 엔티티 사용하기

«table»
myapp_board
board_no PK
writer FK
0..*
1
«table»
myapp_member
member_no PK

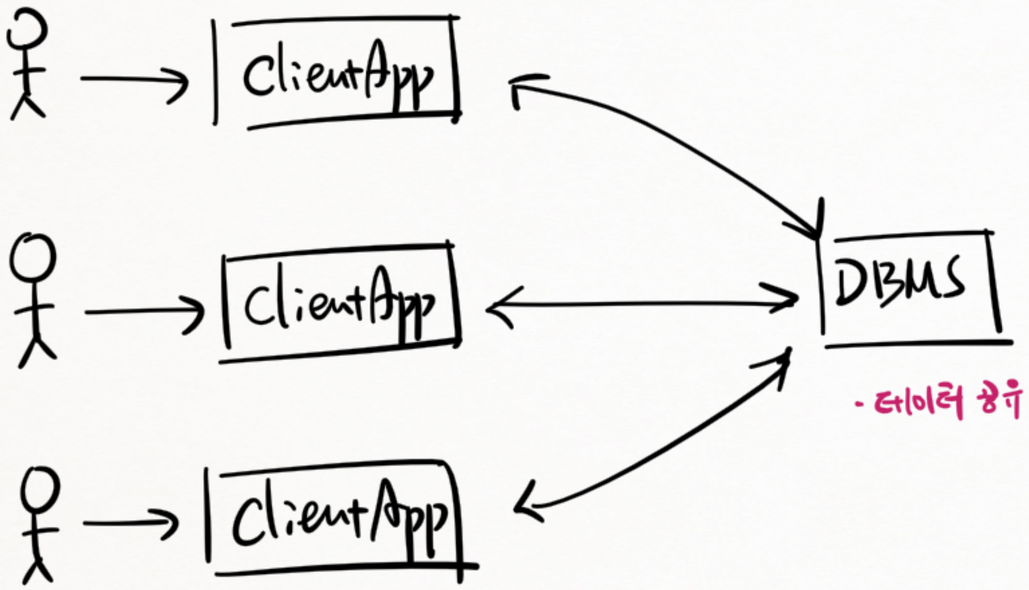

Board
· no
· title
· content
· writer : String → Member
· password
· viewCount
· createdDate
· category
include
Member
· no
· name
· email
· password
· gender
· createdDate
추가

49. 로그인 적용하기


LoginListener
Member
set
ClientApp
loginUser (스태틱변수)
findByEmailAndPassword()
MemberDao
생성
사용
사용
사용
BoardAddListener
BoardUpdateListener
BoardDeleteListener

50. Application Server Architecture

① C/S Architecture



- 데이터 공유

Client

- local 에 설치
  - ↓
  - ✓ 실행할 때 local 자원 사용
  - ✓ 기능 변경 → 재설치 필요
  - ✓ C/C++, VB, PowerBuilder, Delphi

Server

- client 에서 직접 접속
  - ↓
  - client 가 해킹을 당하면 서버가 위험
  - ↓
  - 보안에 취약

Global Business

↓

경쟁이 심화

↓

제품의 라이프사이클이 짧아짐

↓

업무프로세스의 변화가 잦아짐

↓

업무 변경이 잦아짐 → 업무시스템 변경이 잦아짐

↑

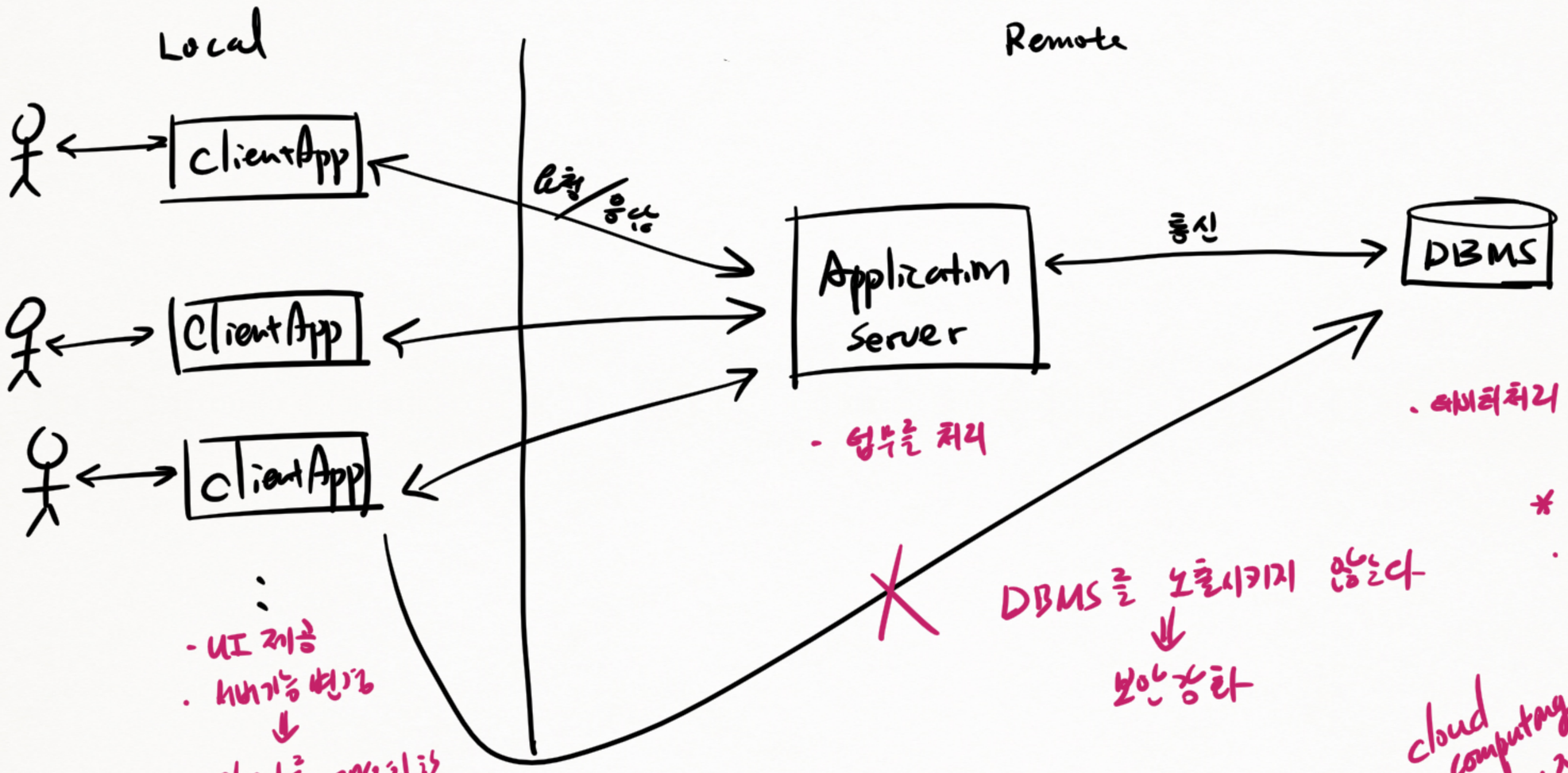
재설치가 잦아짐

↓

유지보수가 힘들어짐

50. Application Server Architecture

② Application Server Architecture



- 업무를 처리

- 데이터처리

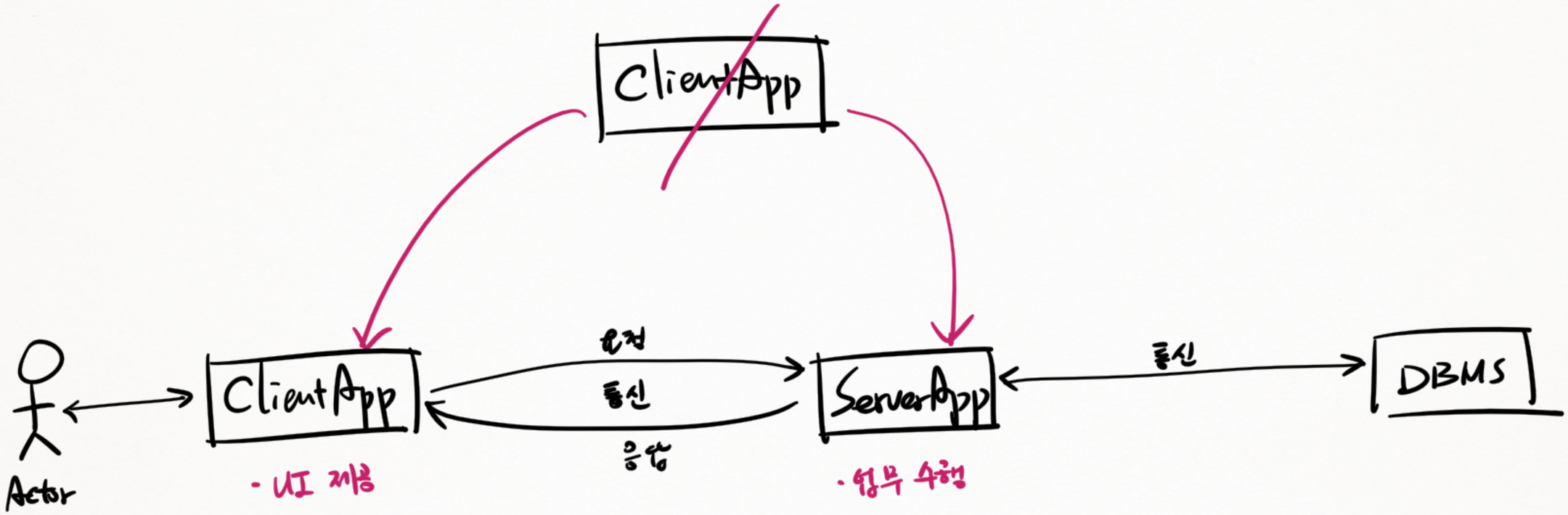
- UI 제공
- 서버기능 변경
  ↓
  client를 재배치할 필요가 없다.

DBMS를 노출시키지 않는다
⇓
보안강화

* 특징
- 모든 기능 서버에서 수행
  ↓
  서버에 부하가 집중
  ↓
  해결

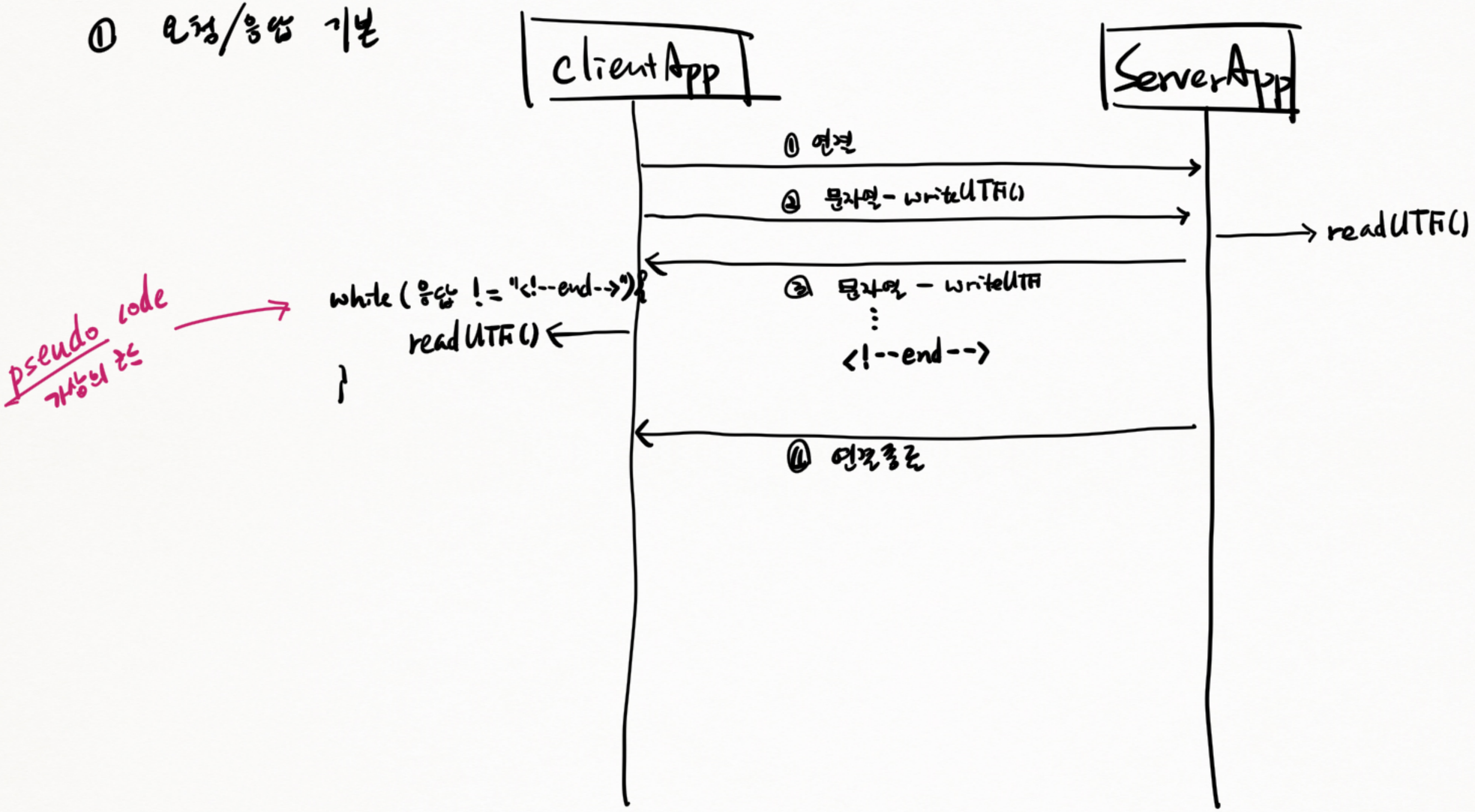
cloud computing 기술의 진보
- clustering H/W 기술
- 분산컴퓨팅 S/W 기술
- ⋮

50. Application Server Architecture



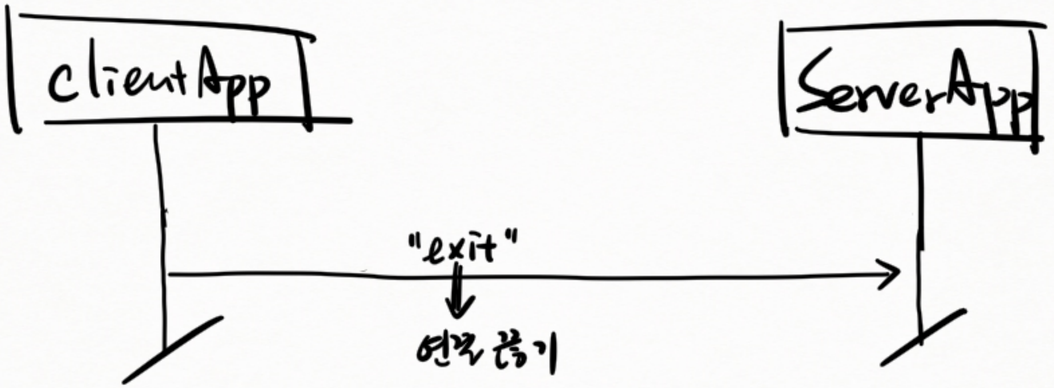
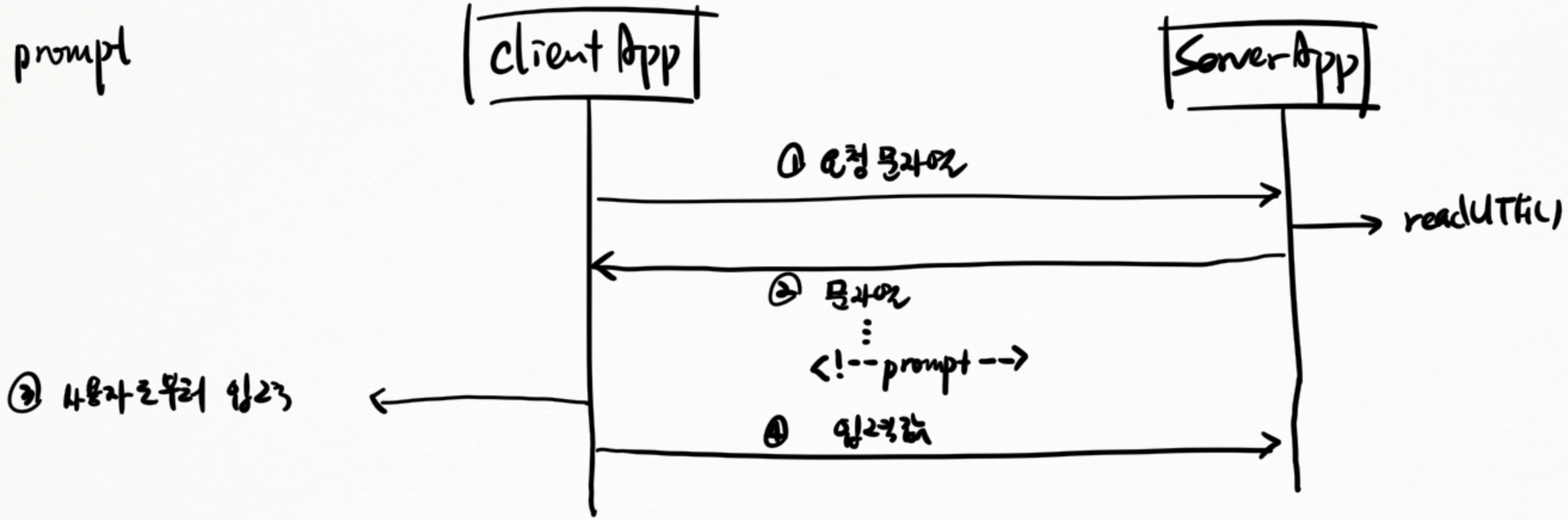
50. Application Server Architecture — Protocol

① 요청/응답 기본



50. Application Server Architecture – Protocol

② exit



③ prompt