

Java/Spring 테스트를 추가하고 싶은 개발자들의 오답노트

방향성 탐색 (1)



Contents

- 01 h2
- 02 레이어드 아키텍처의 문제점
- 03 개선된 아키텍처

h2

“모든 테스트가 h2를 필요로 합니다.”

01

h2

h2

01

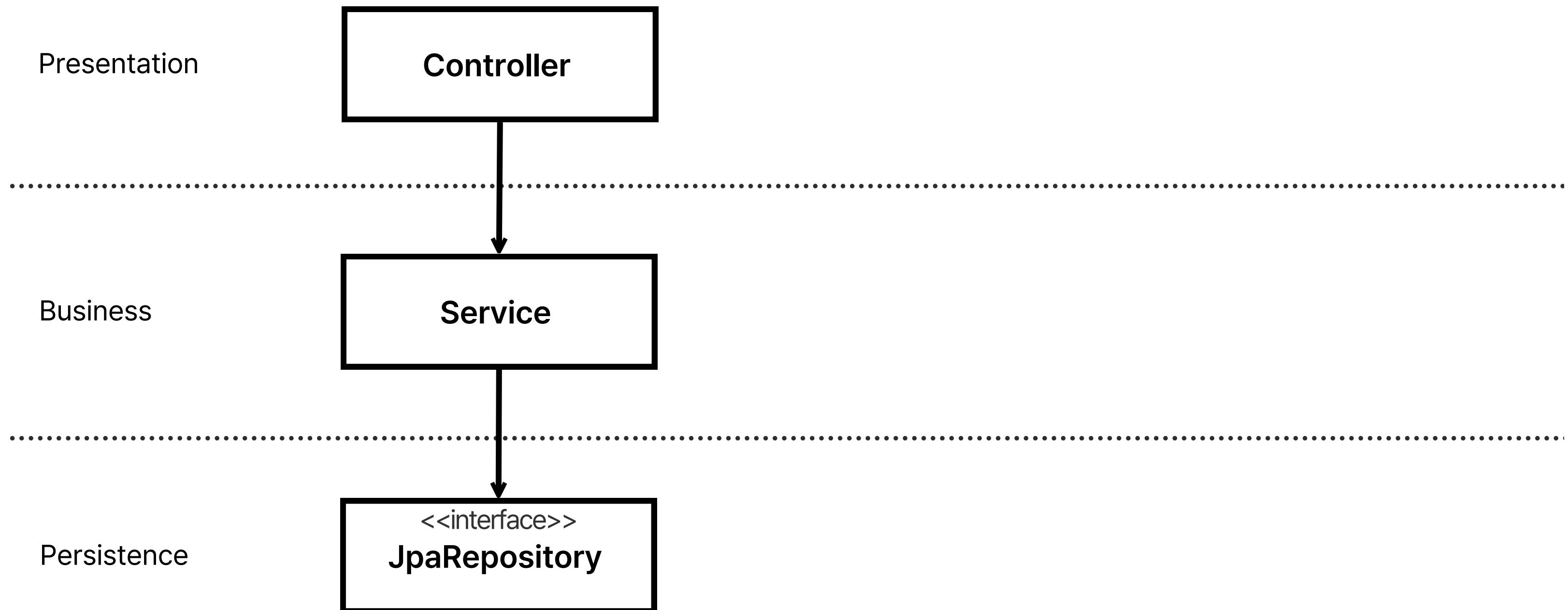
설계가 잘못되었을 확률

02

지금 작성한 테스트가 실제로 테스트가 필요한 본질이 아닐 확률

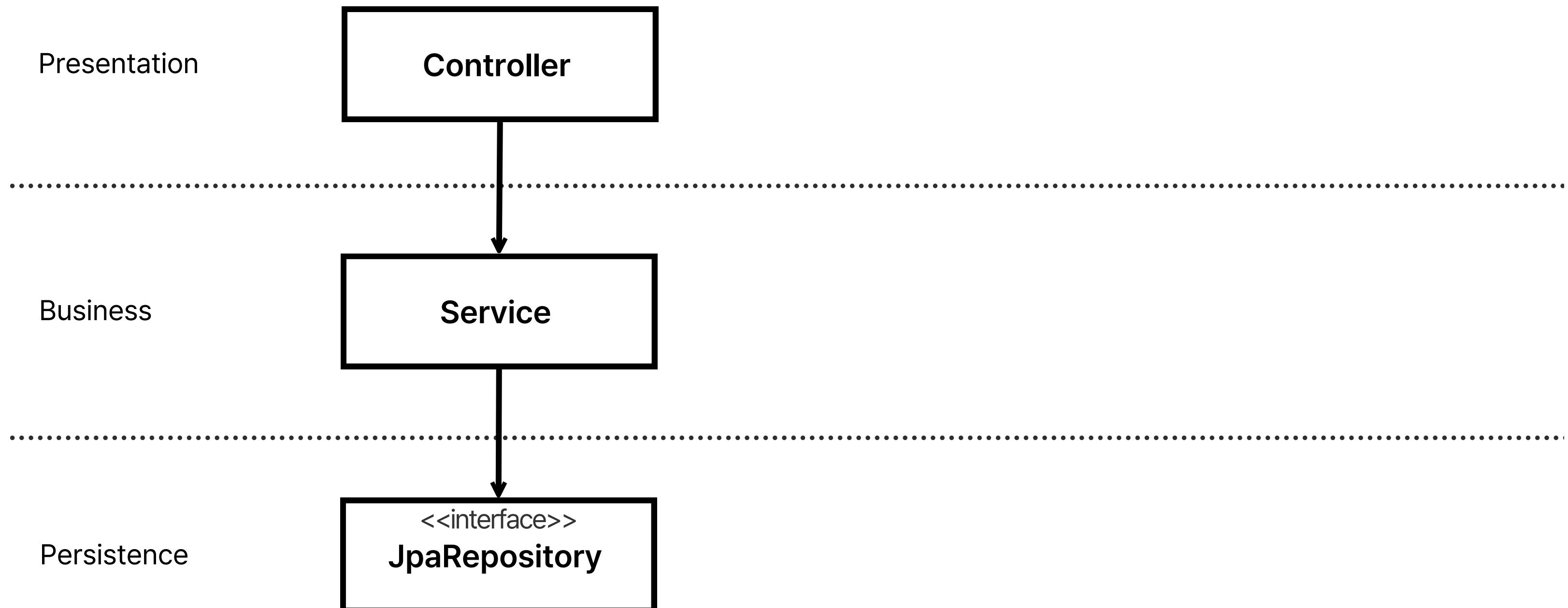
레이어드 아키텍처

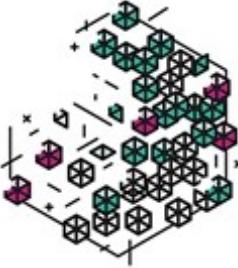
유사한 기능들을 같은 계층으로 묶어 관리하는 방식의 아키텍처 구조



레이어드 아키텍처의 장점 쉽다

기능 개발을 할 때 가시적인 무언가를 만들기에 가장 쉬운 방법.

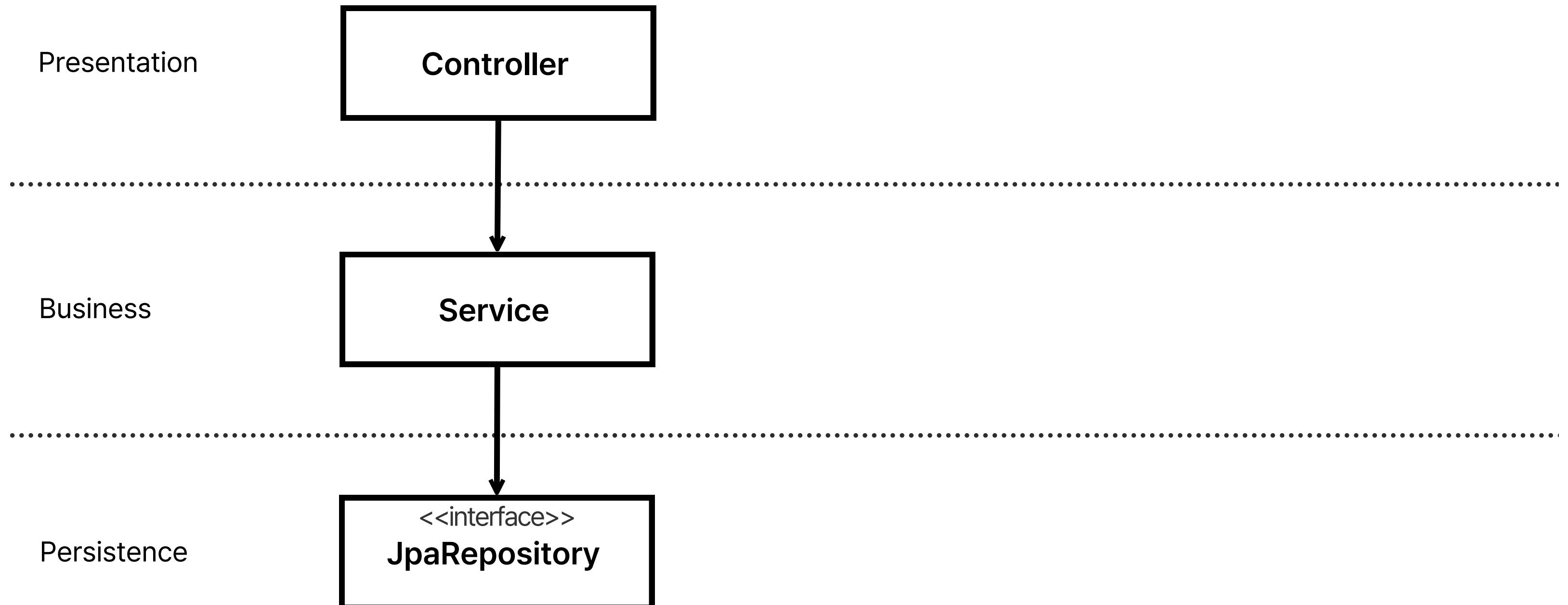


자바 코드로 구현하는
클린 웹 애플리케이션

레이어드 아키텍처의 단점 (1) DB 주도 설계

계층형 아키텍처는 데이터베이스 주도 설계를 유도한다. (중략) 모든 것이 영속성 계층을 토대로 만들어진다.

톰 훔버그, 만들면서 배우는 클린 아키텍처 자바 코드로 구현하는 클린 웹 애플리케이션, 박소은 옮김 조영호 감수, (위키북스, 2022-03-30), 2~3p



레이어드 아키텍처의 단점 (1) DB 주도 설계

계층형 아키텍처는 데이터베이스 주도 설계를 유도한다. (중략) 모든 것이 영속성 계층을 토대로 만들어진다.

톰 홈버그, 만들면서 배우는 클린 아키텍처 자바 코드로 구현하는 클린 웹 애플리케이션, 박소은 옮김 조영호 감수, (위키북스, 2022-03-30), 2~3p



여러분이 주문 시스템을 만들어야 한다면?

레이어드 아키텍처의 단점 (1) DB 주도 설계

계층형 아키텍처는 **데이터베이스 주도 설계**를 유도한다. (중략) 모든 것이 영속성 계층을 토대로 만들어진다.

톰 훔버그, 만들면서 배우는 클린 아키텍처 자바 코드로 구현하는 클린 웹 애플리케이션, 박소은 옮김 조영호 감수, (위키북스, 2022-03-30), 2~3p

```
@Getter  
@Setter  
@Entity  
@Table(name = "order")  
public class OrderEntity {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id;  
  
    @Column  
    private String orderId;  
  
    @Column  
    private Date orderDate;  
  
    // ...  
}
```

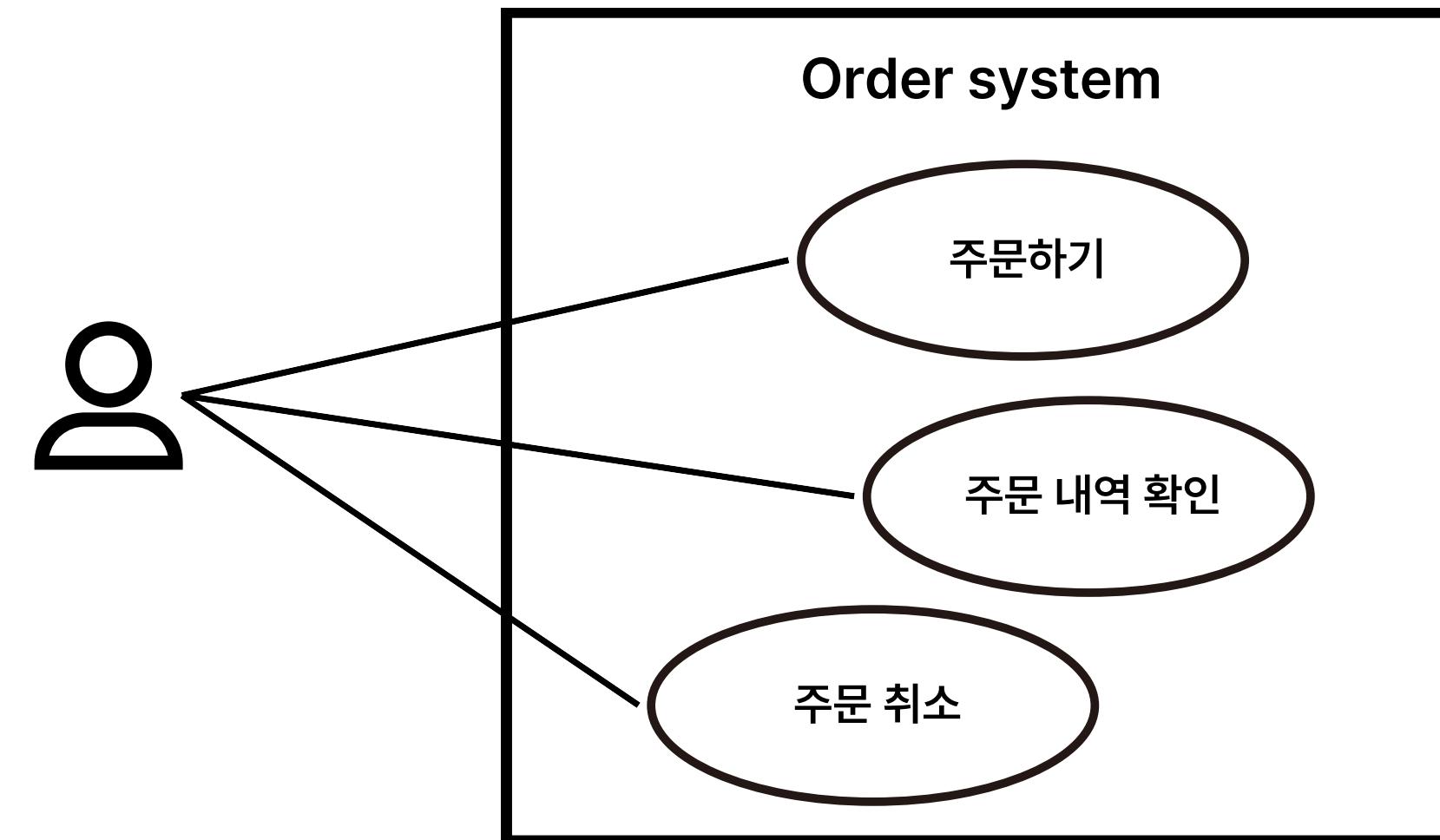
아마도 JPA Entity를 먼저 떠올리셨을 겁니다.

레이어드 아키텍처의 단점 (1) DB 주도 설계

계층형 아키텍처는 **데이터베이스 주도 설계**를 유도한다. (중략) 모든 것이 영속성 계층을 토대로 만들어진다.

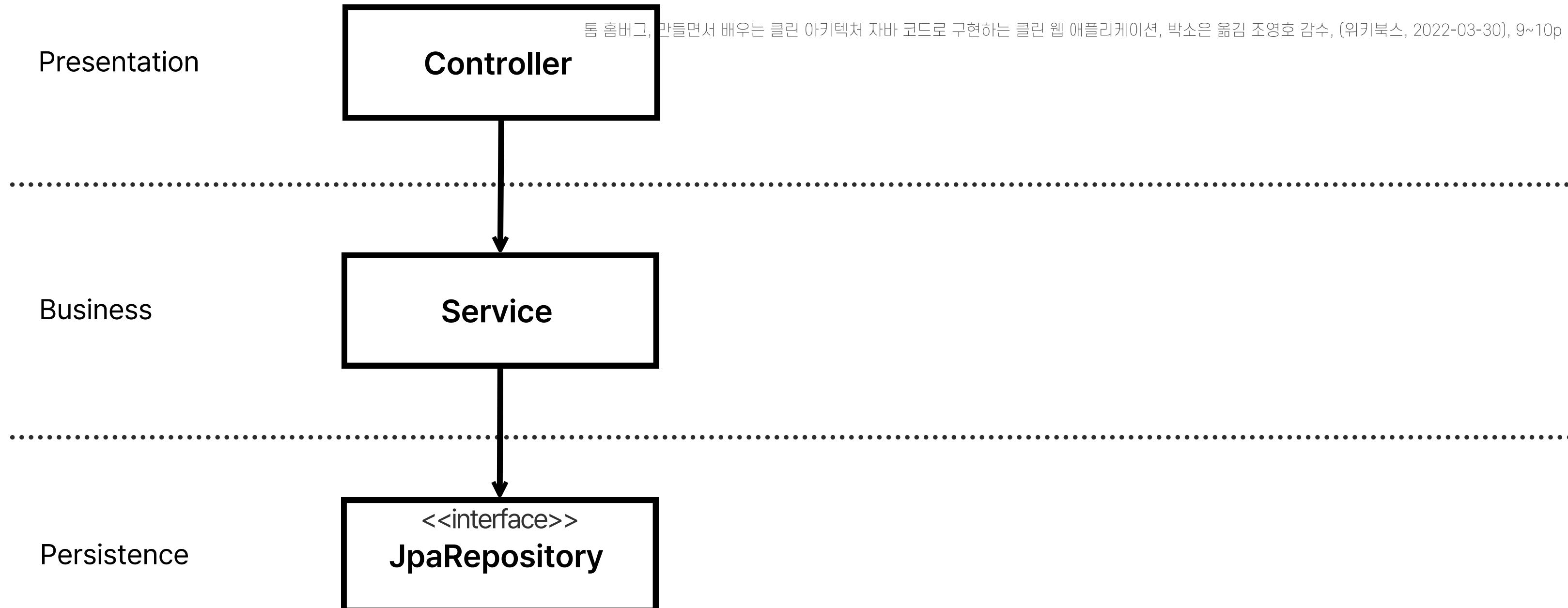
톰 홈버그, 만들면서 배우는 클린 아키텍처 자바 코드로 구현하는 클린 웹 애플리케이션, 박소은 옮김 조영호 감수, (위키북스, 2022-03-30), 2~3p

하지만 실은 주문 시스템에 필요한 **Use case**를 파악하는게 먼저입니다



레이어드 아키텍처의 단점 (2) 동시 작업

이러한 기대를 충족시키려면 아키텍처가 동시 작업을 지원해야 하지만 이렇게 하기란 쉽지는 않다. 그리고 계층형 아키텍처는 이런 측면에서 그다지 도움이 되지 않는다. (중략) 계층형 아키텍처에서는 이렇게 작업할 수 없다. 모든 것이 영속성 계층 위에 만들어지기 때문에 영속성 계층을 먼저 개발해야 하고, 그 다음에 도메인 계층을, 그리고 마지막으로 웹 계층을 만들어야 한다. 그렇기 때문에 특정 기능은 동시에 한 명의 개발자만 작업할 수 있다.

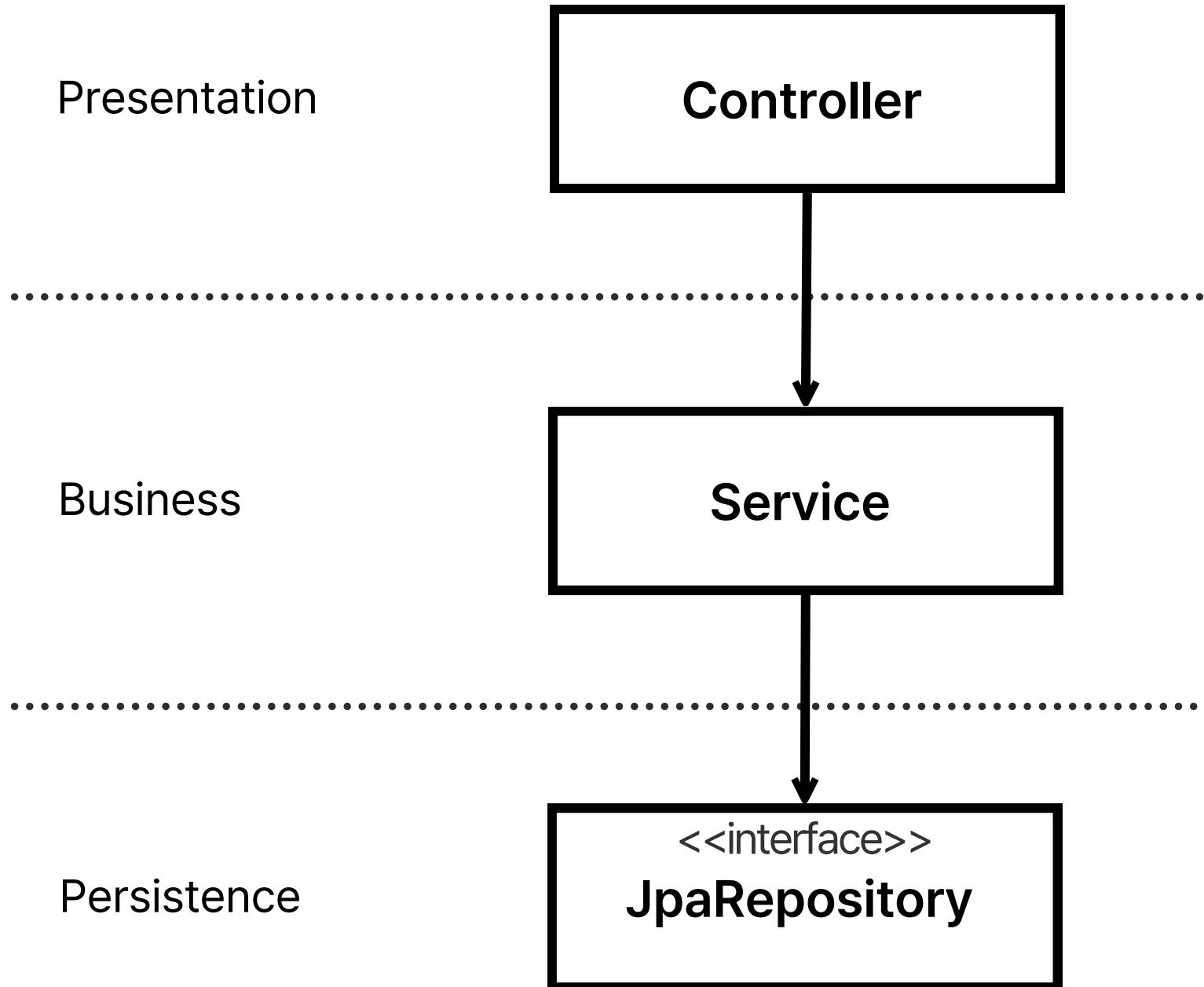




레이어드 아키텍처의 단점 (3) 죽은 도메인

계층형 아키텍처는 업무 도메인에 대해 아무것도 말해주지 않는다.

로버트 C. 마틴 저, 클린 아키텍처 소프트웨어 구조와 설계의 원칙, 송준이 옮김, (인사이트(insight), 2019-08-20), 318p



```

@Transactional
public UserEntity createUser(UserCreateDto userCreateDto) {
    UserEntity userEntity = new UserEntity();
    userEntity.setEmail(userCreateDto.getEmail());
    userEntity.setAddress(userCreateDto.getAddress());
    userEntity.setStatus(UserStatus.PENDING);
    userEntity.setCertificationCode(UUID.randomUUID().toString());
    userEntity = userRepository.save(userEntity);
    String certificationUrl = generateCertificationUrl(userEntity);
    sendCertificationEmail(userCreateDto.getEmail(), certificationUrl);
    return userEntity;
}

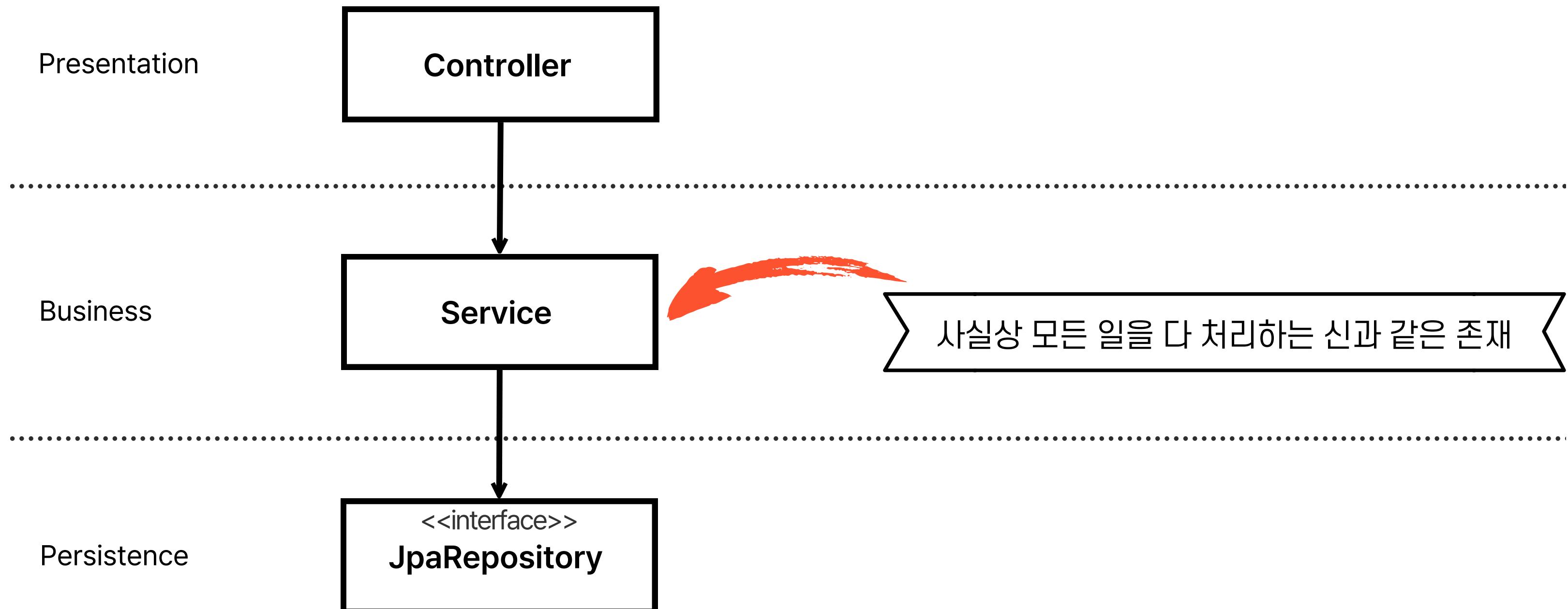
public void sendCertificationEmail(String email, String certificationUrl) {
    SimpleMailMessage message = new SimpleMailMessage();
    message.setTo(email);
    message.setSubject("Please certify your email address");
    message.setText("Please click the following link to certify your email address: " + certificationUrl);
    mailSender.send(message);
}
  
```

객체는 수동적이고 모든 코드가 함수 위주로 돌아간다.

레이어드 아키텍처의 단점 (3) 죽은 도메인

계층형 아키텍처는 업무 도메인에 대해 아무것도 말해주지 않는다.

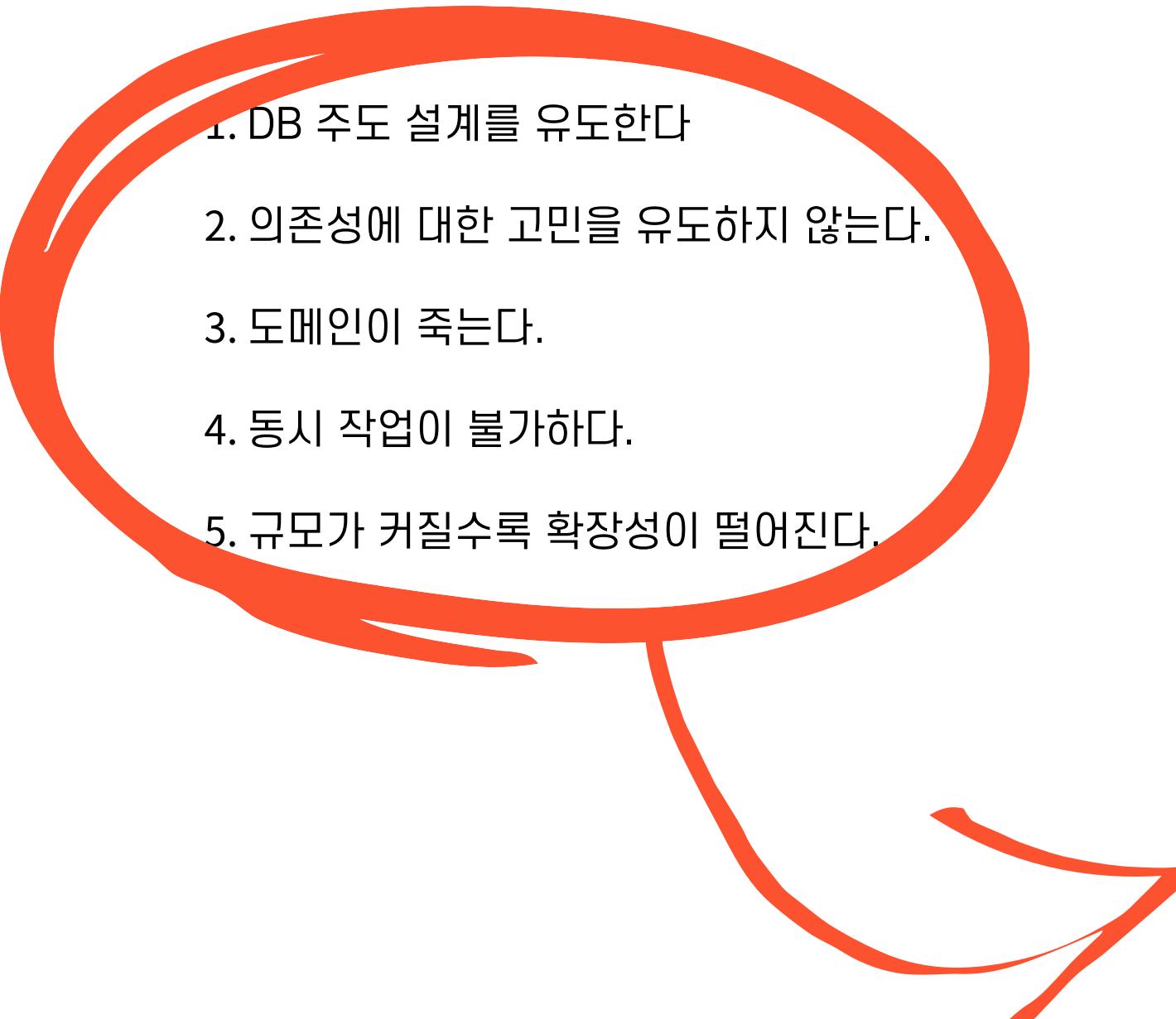
로버트 C. 마틴 저, 클린 아키텍처 소프트웨어 구조와 설계의 원칙, 송준이 옮김, (인사이트(insight), 2019-08-20), 318p



레이어드 아키텍처의 단점

1. DB 주도 설계를 유도한다
2. 의존성에 대한 고민을 유도하지 않는다.
3. 도메인이 죽는다.
4. 동시 작업이 불가하다.
5. 규모가 커질수록 확장성이 떨어진다.

레이어드 아키텍처의 단점

- 
1. DB 주도 설계를 유도한다
 2. 의존성에 대한 고민을 유도하지 않는다.
 3. 도메인이 죽는다.
 4. 동시 작업이 불가하다.
 5. 규모가 커질수록 확장성이 떨어진다.

절차지향적 사고를 유도

레이어드 아키텍처의 단점

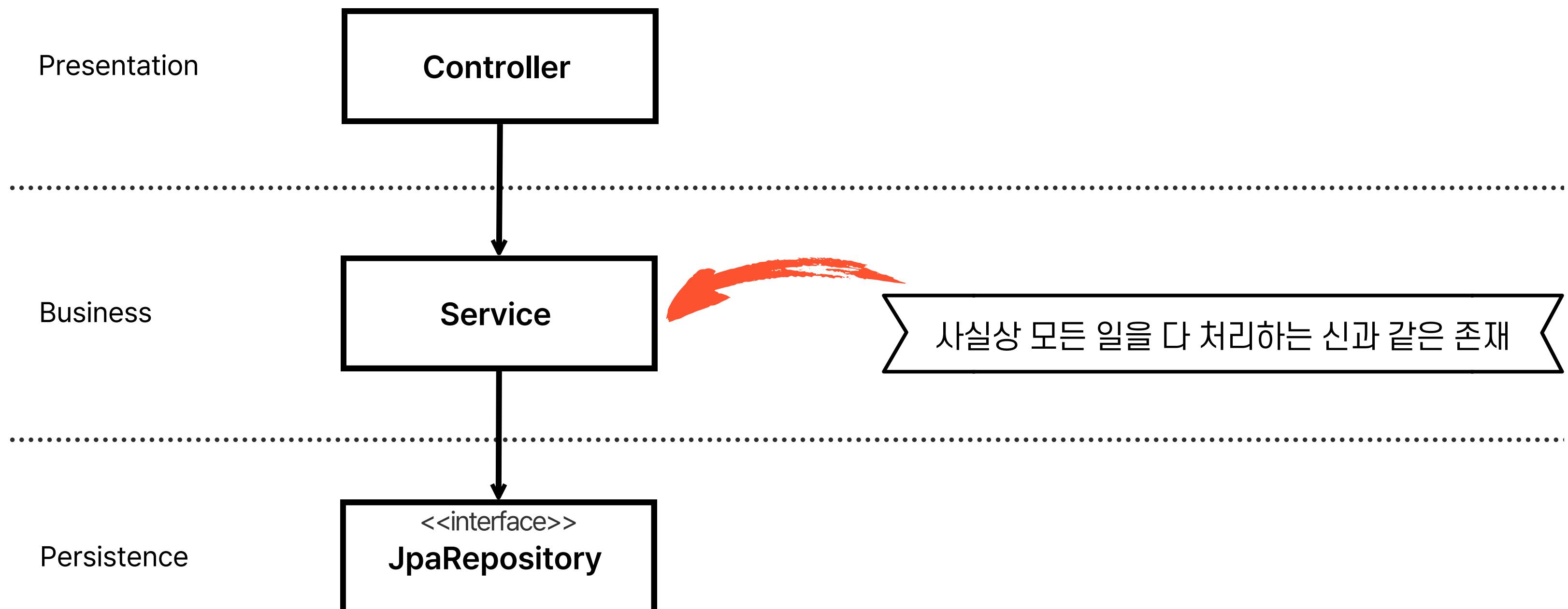
1. DB 주도 설계를 유도한다
2. 의존성에 대한 고민을 유도하지 않는다.
3. 도메인이 죽는다.
4. 동시 작업이 불가하다.
5. 규모가 커질수록 확장성이 떨어진다.

절차지향적 사고를 유도

낮은 Testability & Bad SOLID

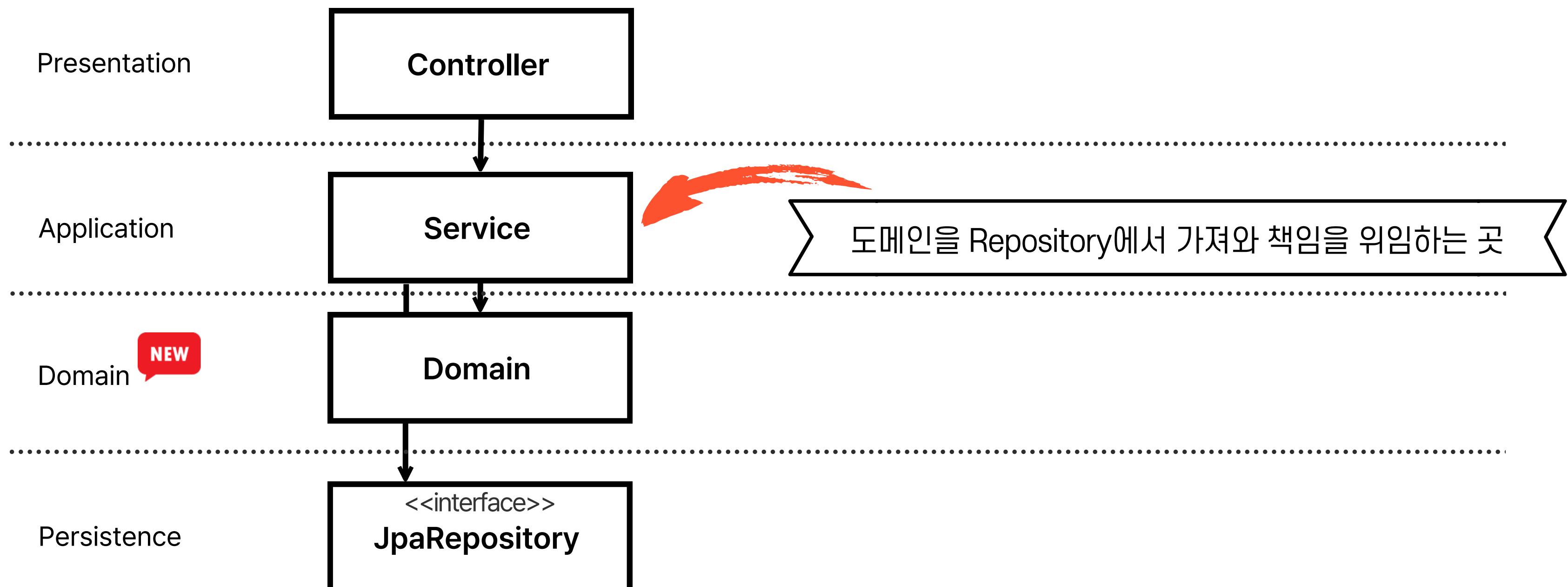
개선된 아키텍처

1. 죽은 도메인



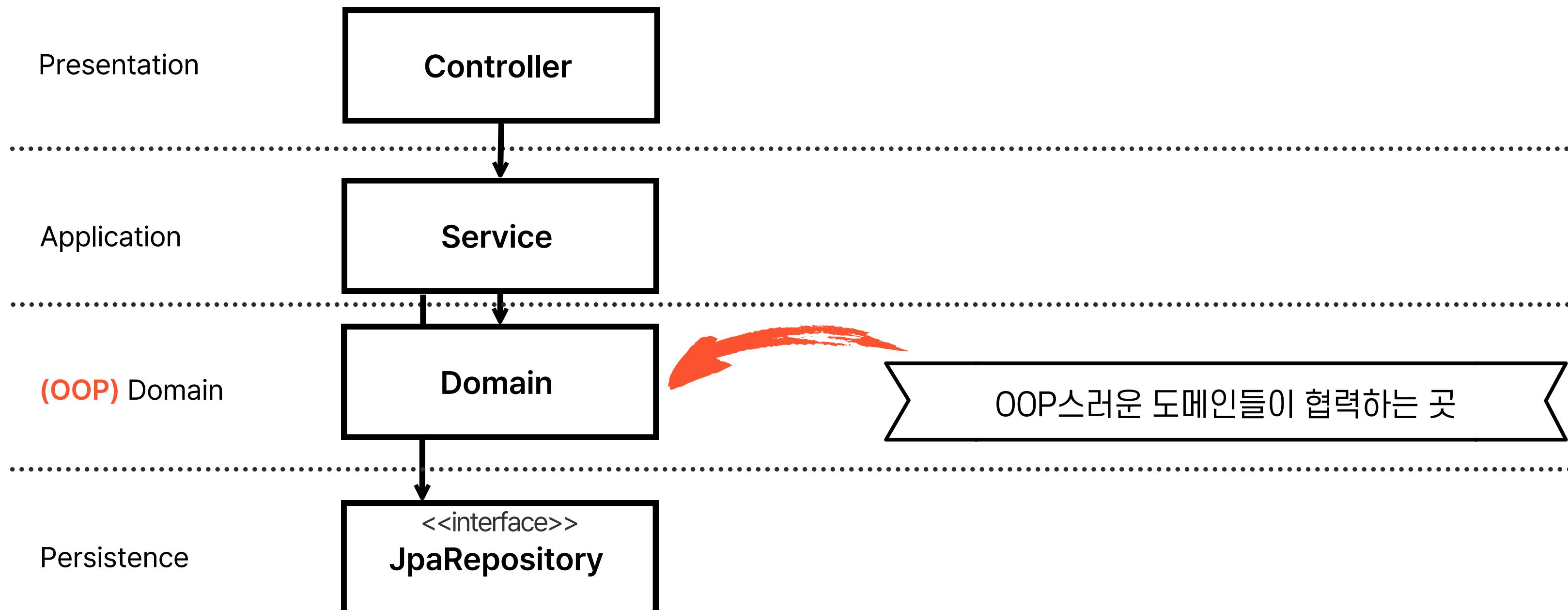
개선된 아키텍처

1. 죽은 도메인



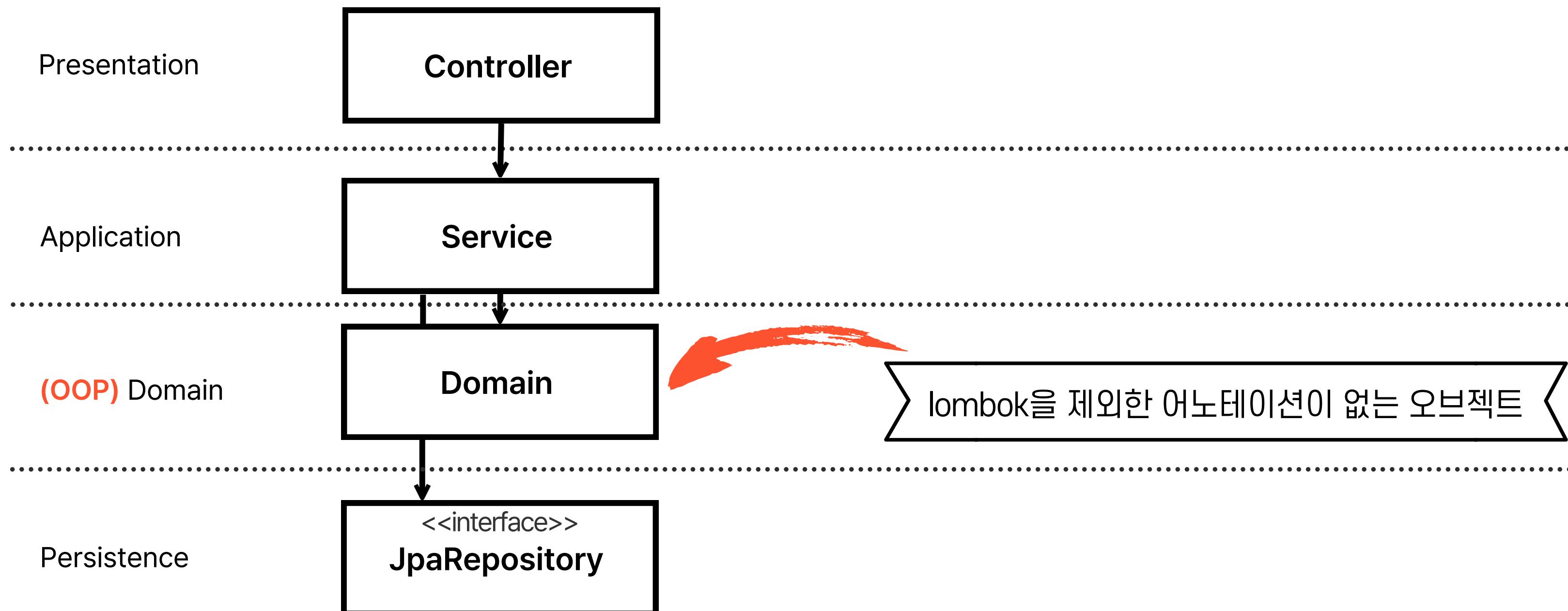
개선된 아키텍처

1. 죽은 도메인



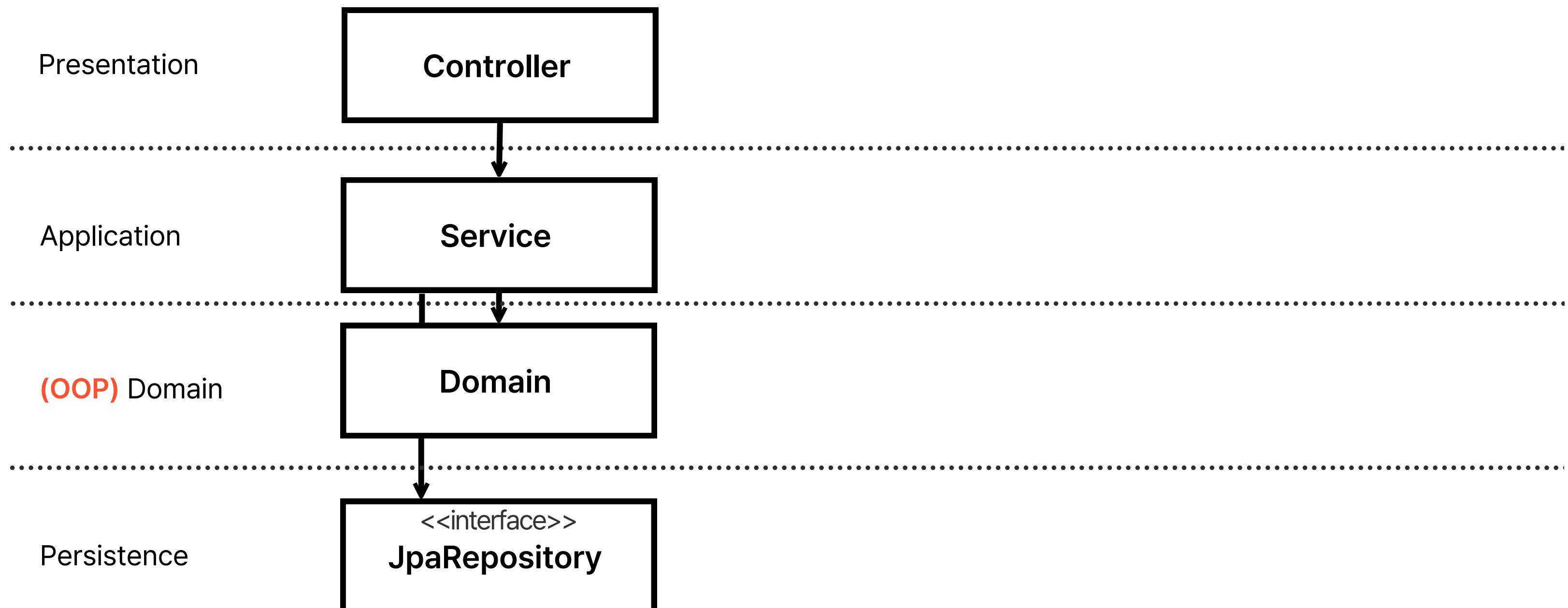
개선된 아키텍처

1. 죽은 도메인



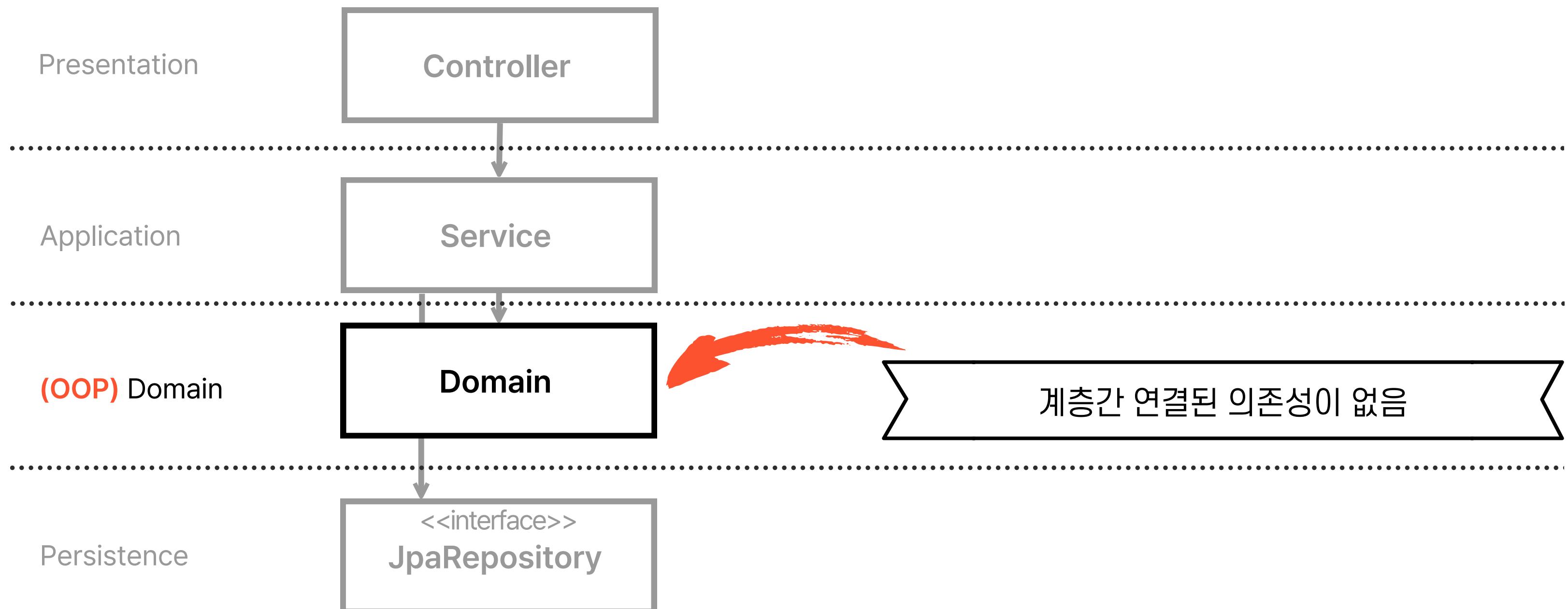
개선된 아키텍처

2. SOLID & 동시 작업



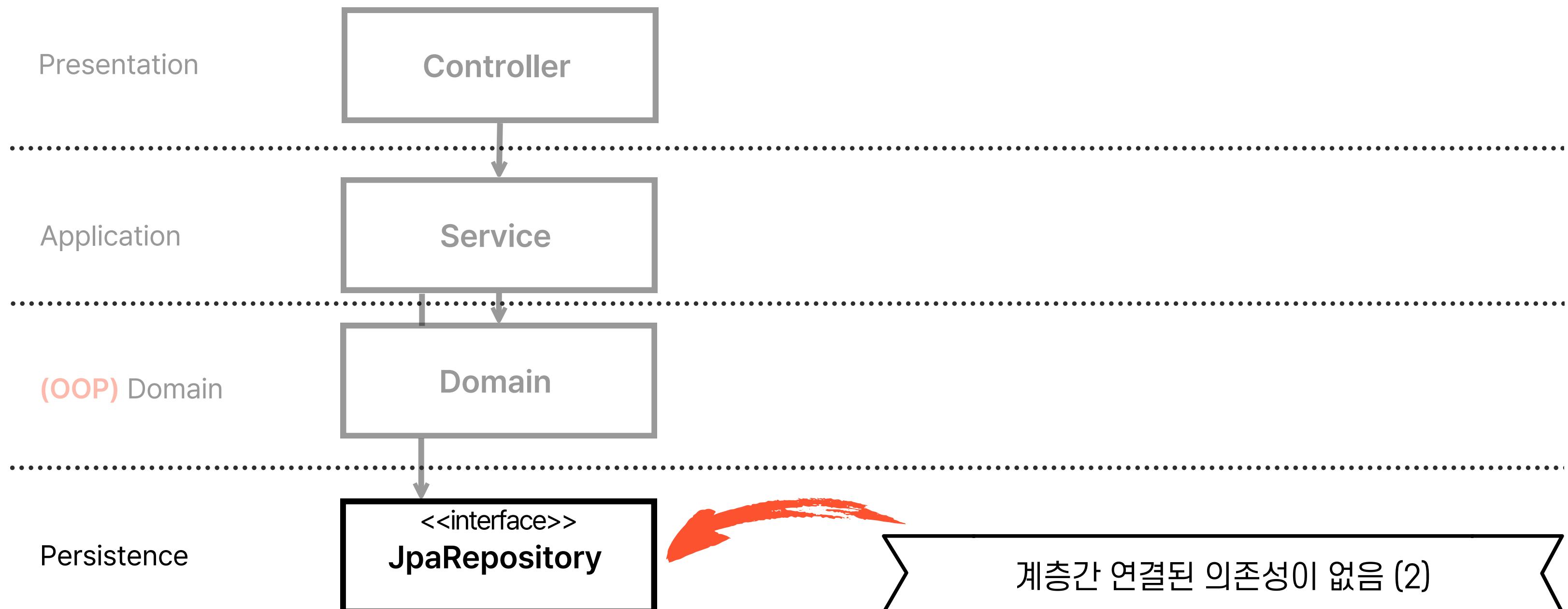
개선된 아키텍처

3. 낮은 Testability



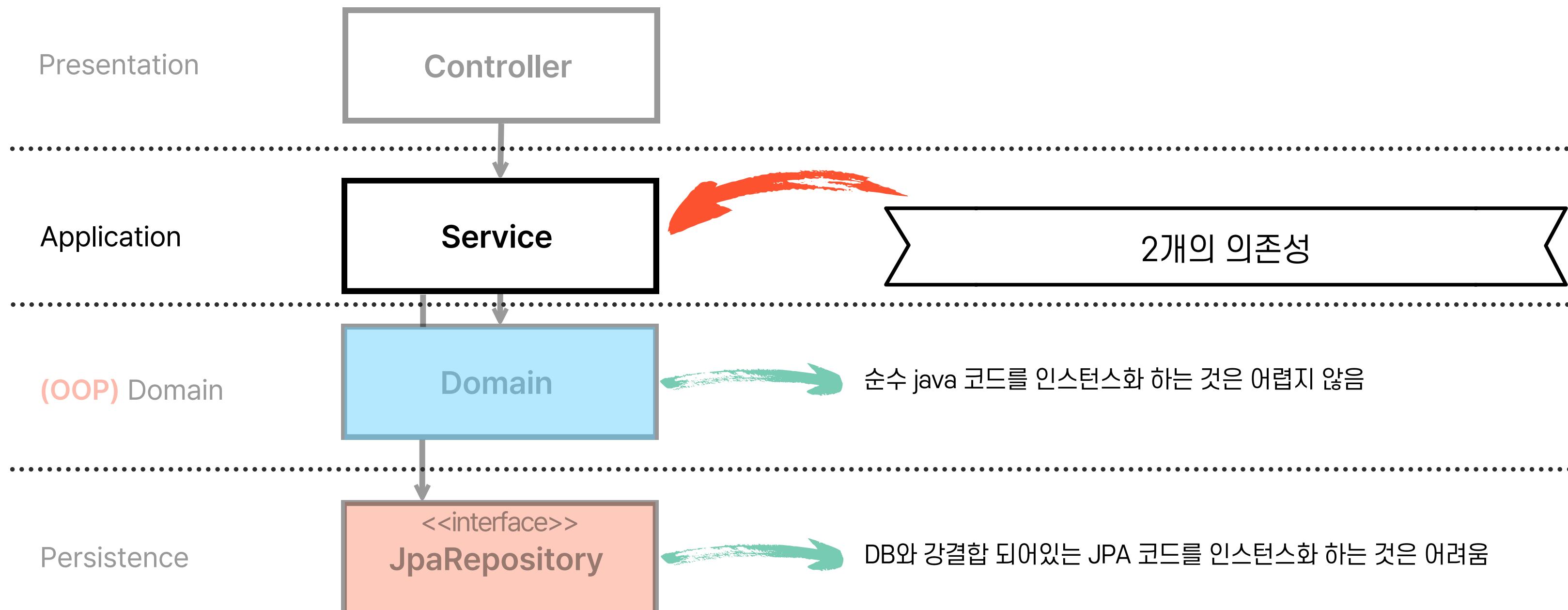
개선된 아키텍처

3. 낮은 Testability



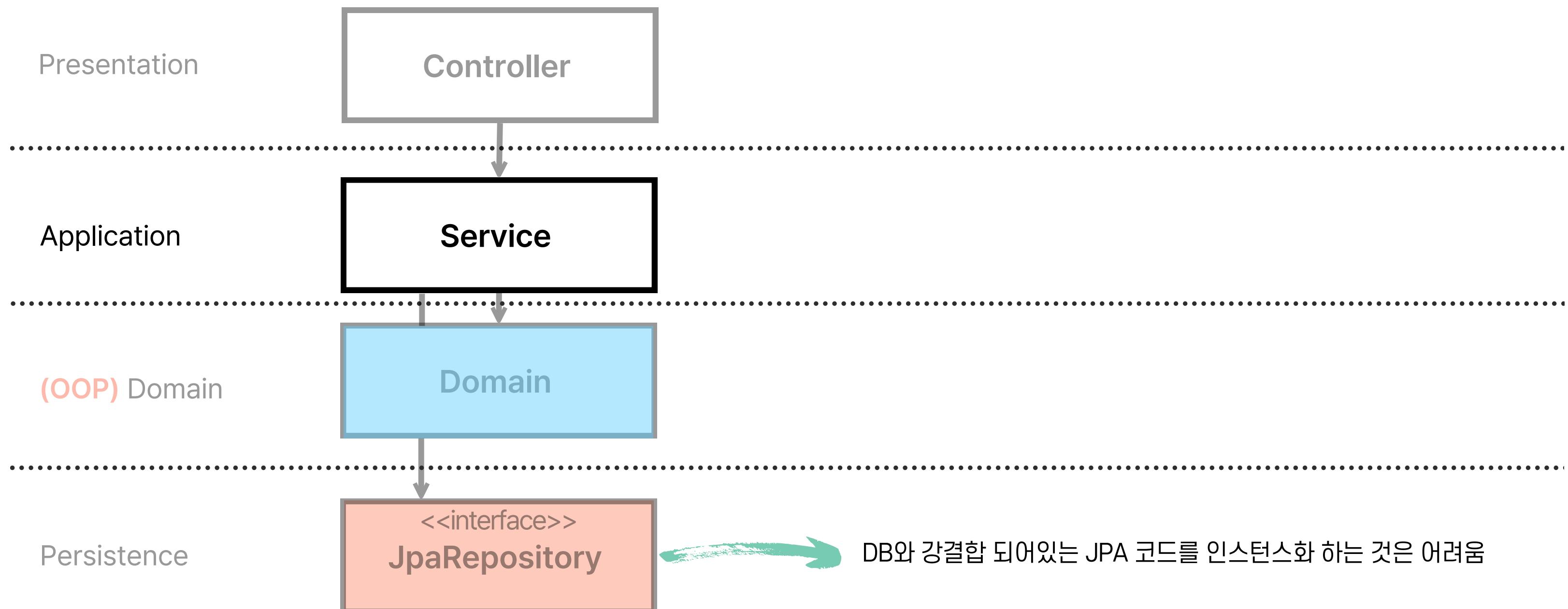
개선된 아키텍처

3. 낮은 Testability



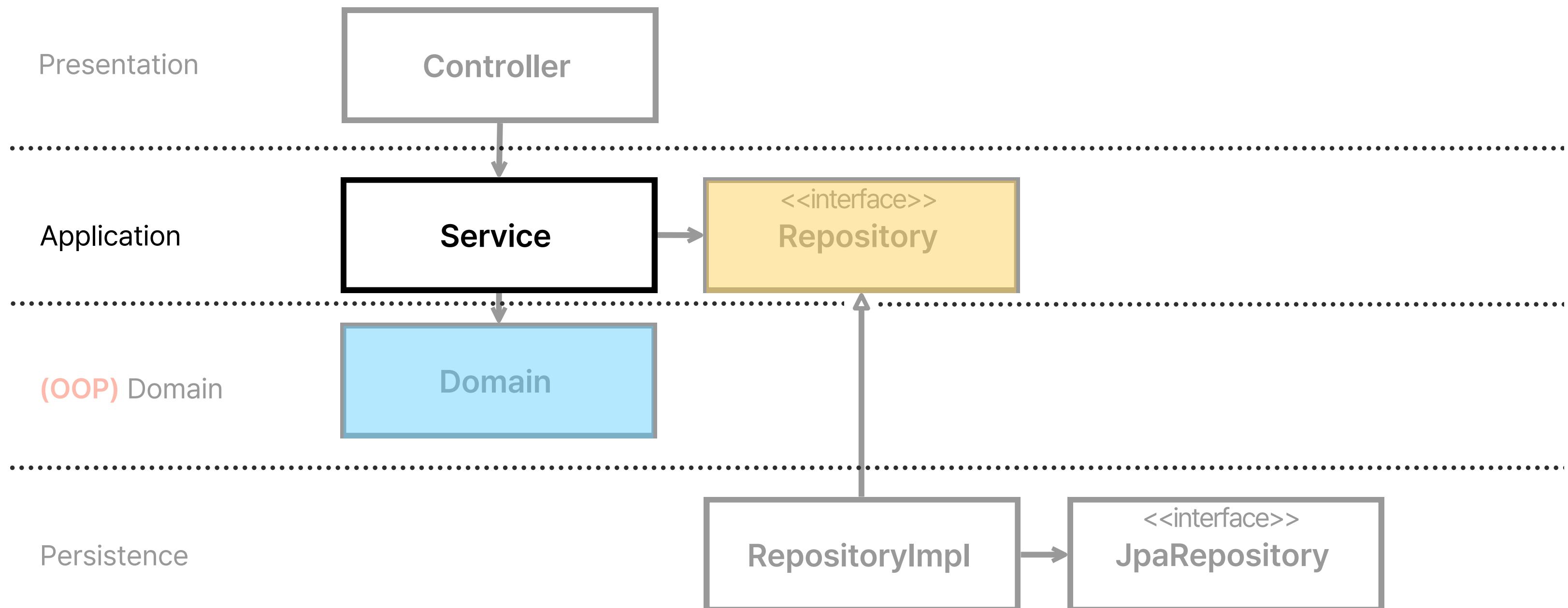
개선된 아키텍처

3. 낮은 Testability



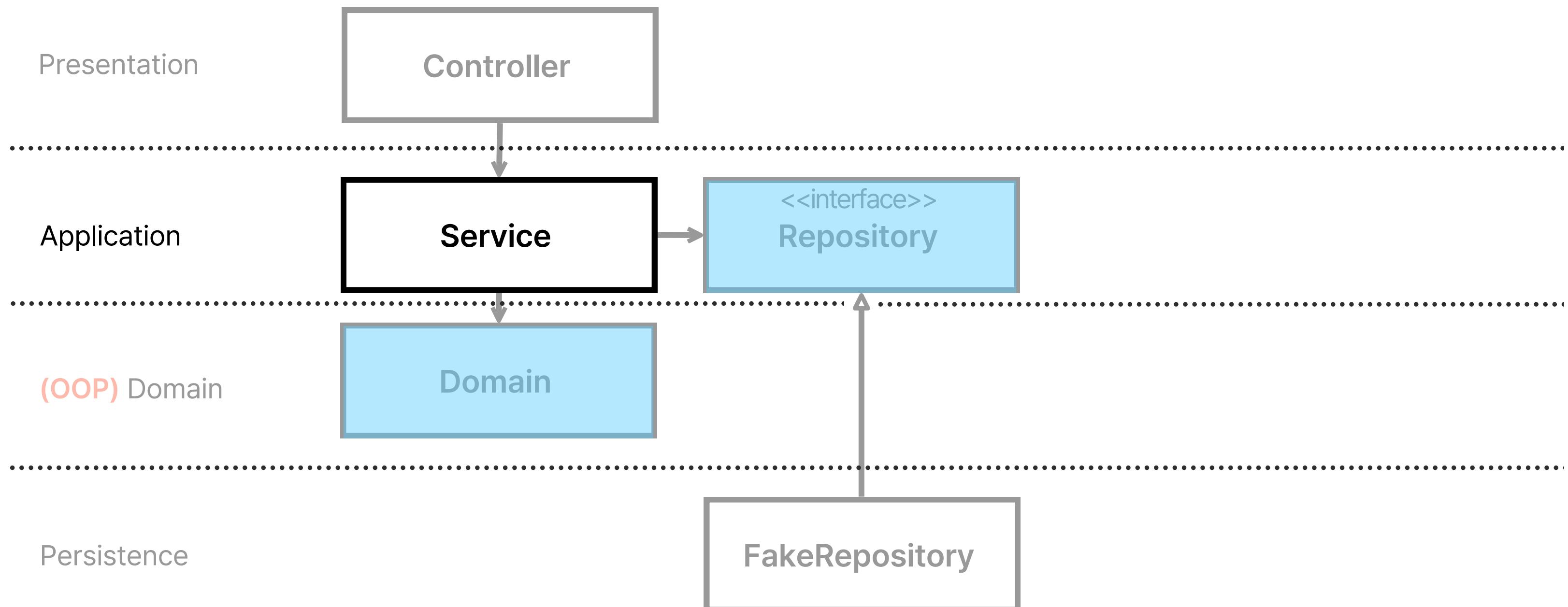
개선된 아키텍처

3. 낮은 Testability



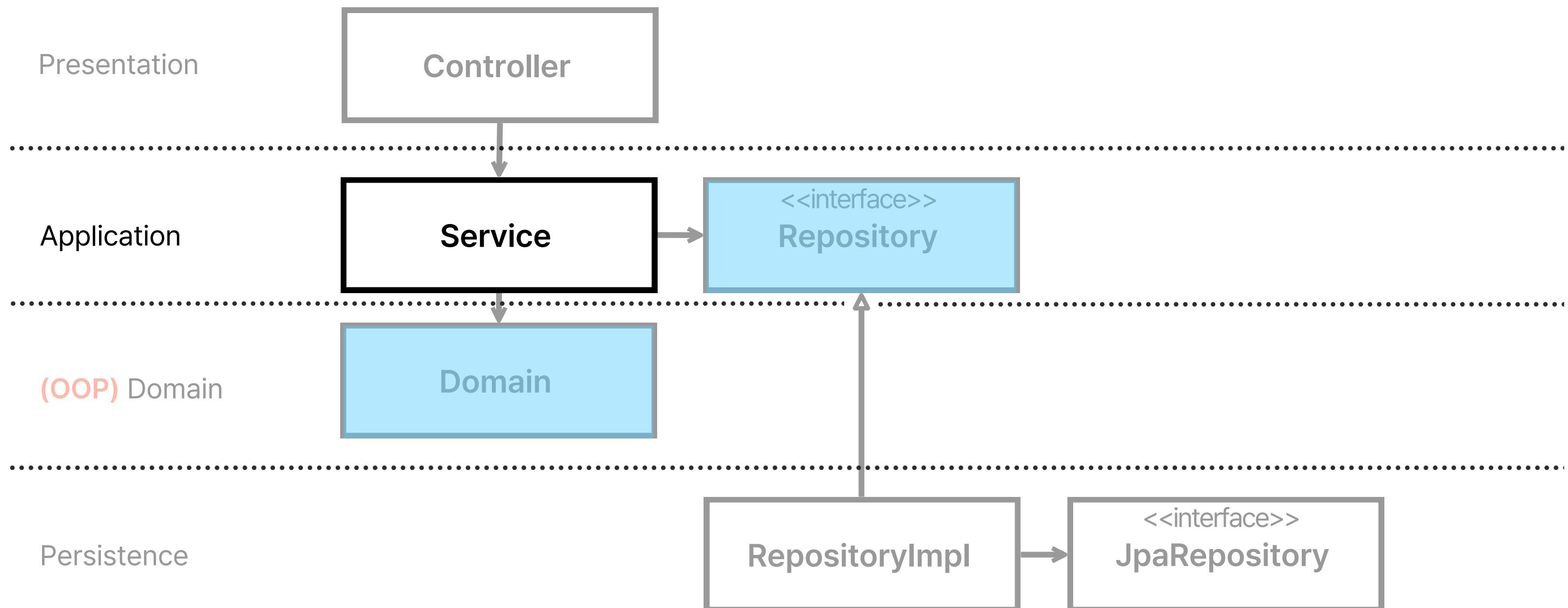
개선된 아키텍처

3. 낮은 Testability



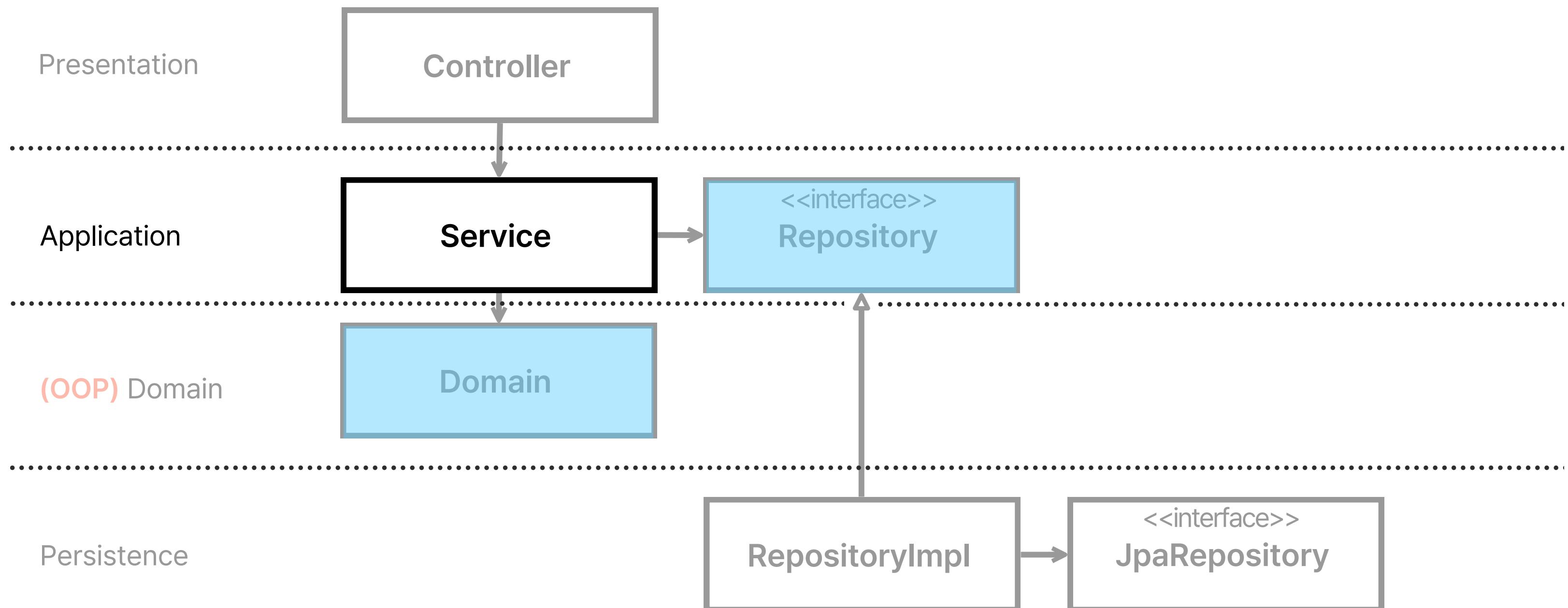
개선된 아키텍처

3. 낮은 Testability



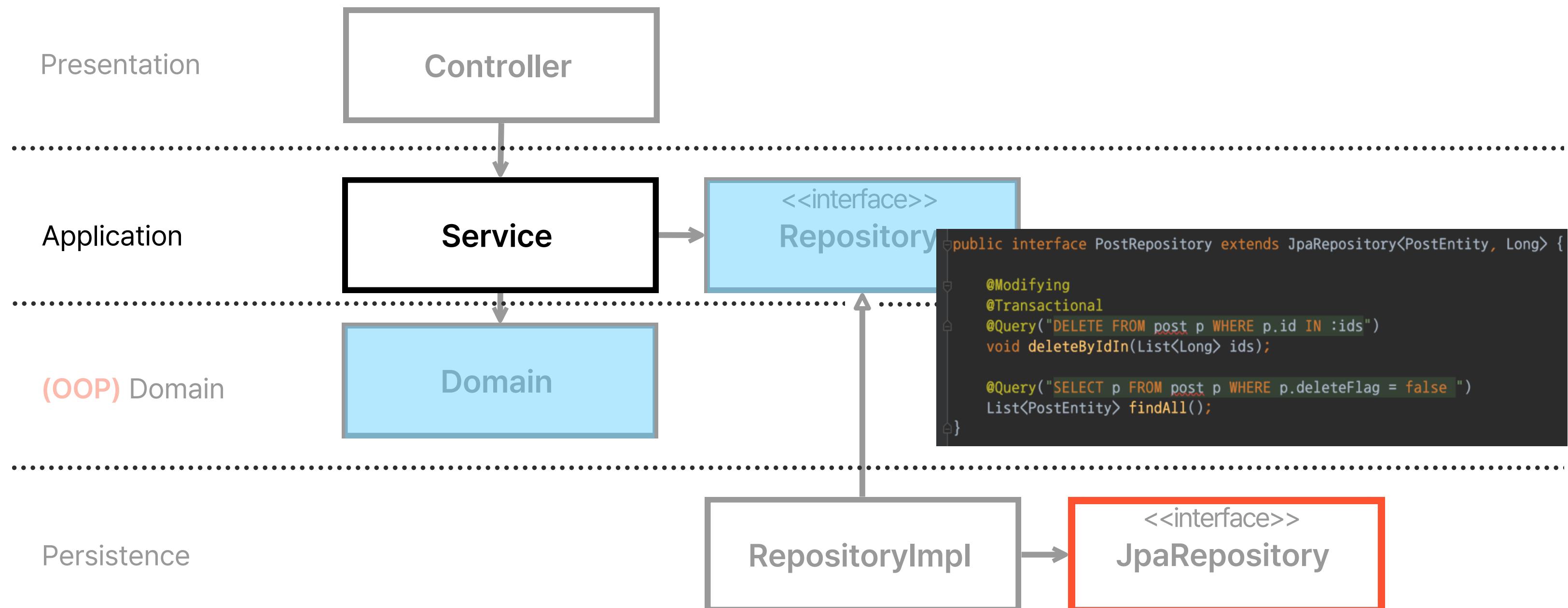
개선된 아키텍처

3. 낮은 Testability



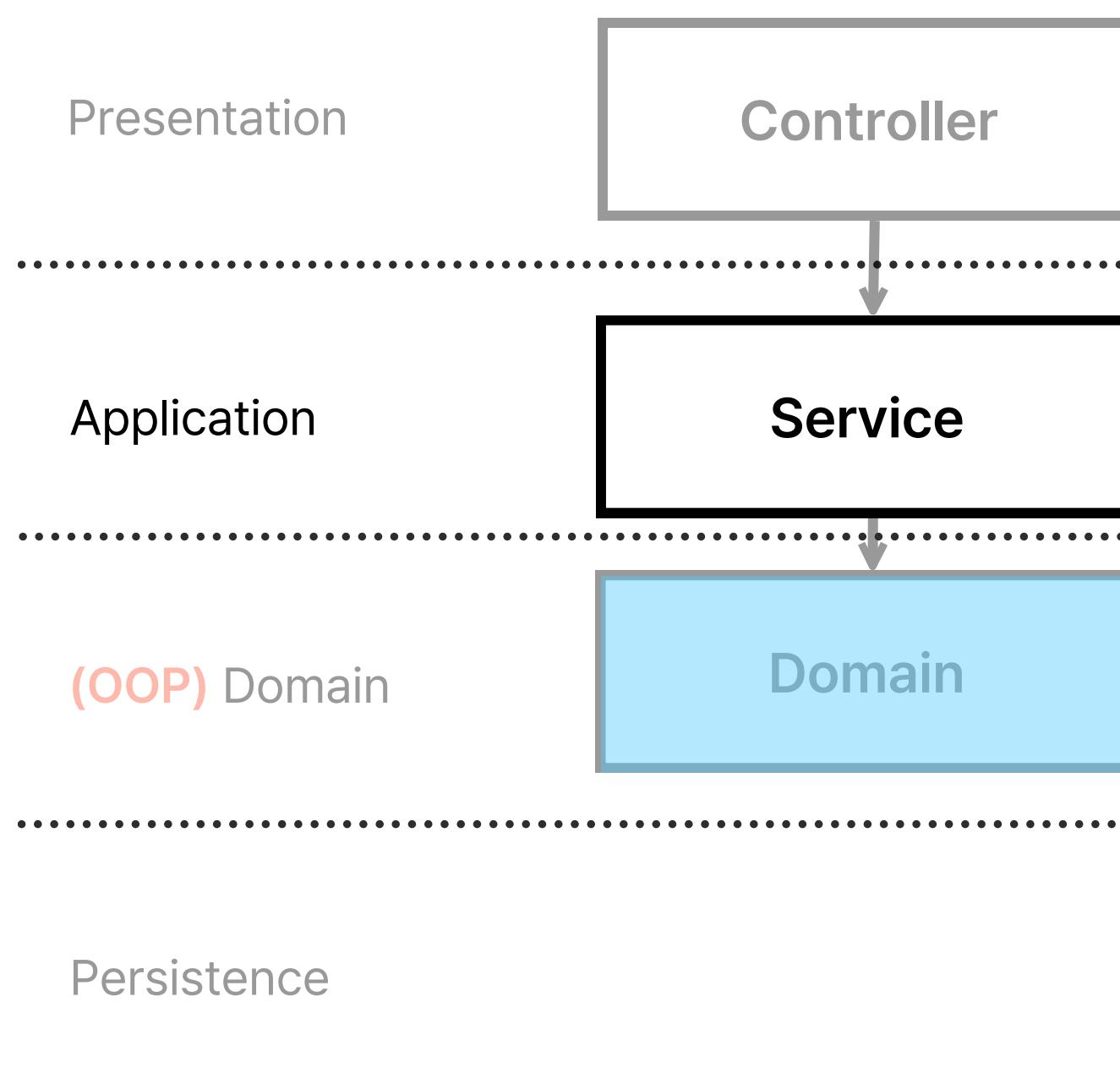
개선된 아키텍처

3. 낮은 Testability



개선된 아키텍처

3. 낮은 Testability



```

@RequiredArgsConstructor
class PostRepositoryImpl implements PostRepository {

    private final PostJpaRepository postJpaRepository;

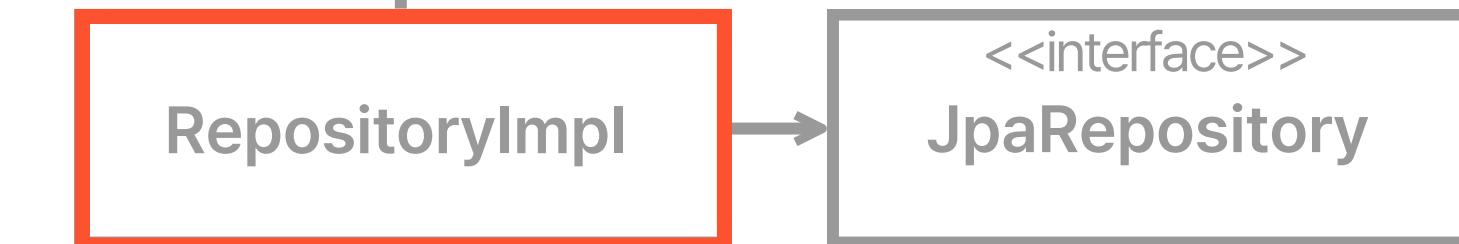
    @Override
    public Post getById(long id) {
        return postJpaRepository.findById(id).Optional<PostEntity>
            .orElseThrow(NotFoundException::new).toDomain();
    }

    @Override
    public Optional<Post> findById(long id) {
        return postJpaRepository.findById(id).map(PostEntity::toDomain);
    }

    @Override
    public List<Post> findAll() {
        return postJpaRepository.findAll().List<PostEntity>
            .stream().Stream<PostEntity>
            .map(PostEntity::toDomain).Stream<Post>
            .collect(Collectors.toList());
    }

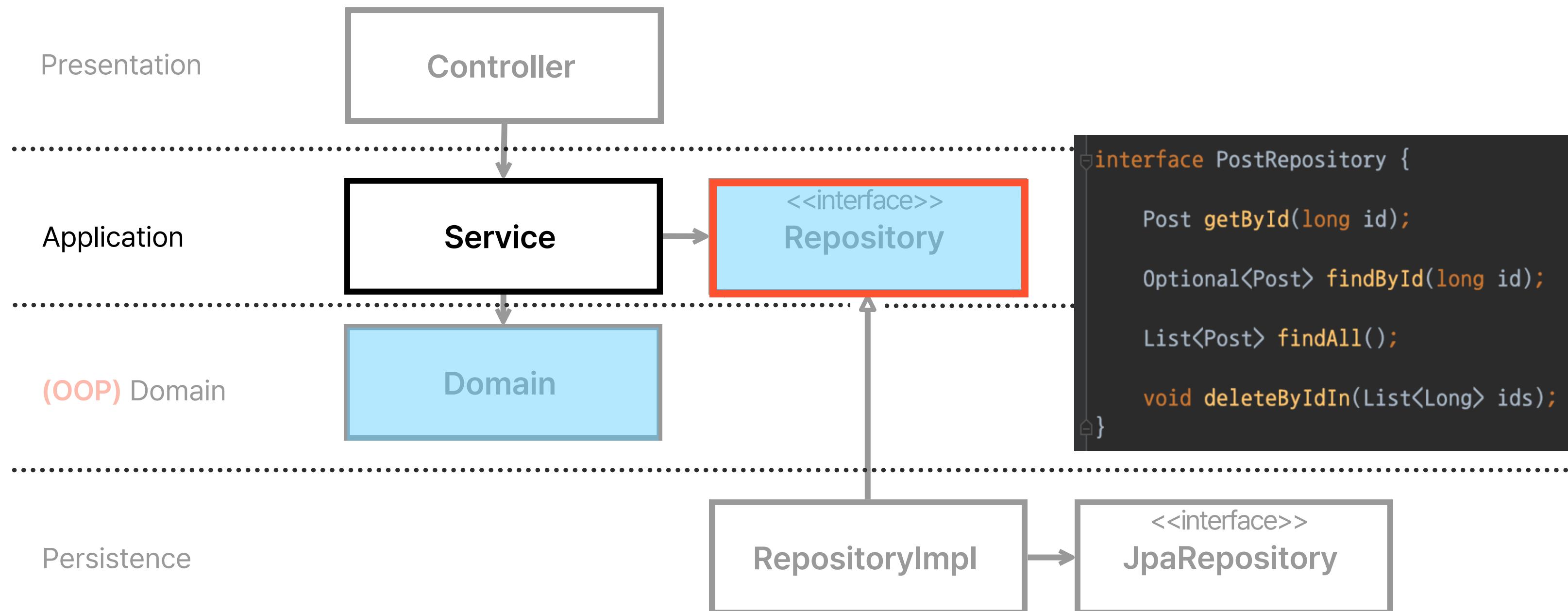
    @Override
    public void deleteByIdIn(List<Long> ids) {
        postJpaRepository.deleteByIdIn(ids);
    }
}

```



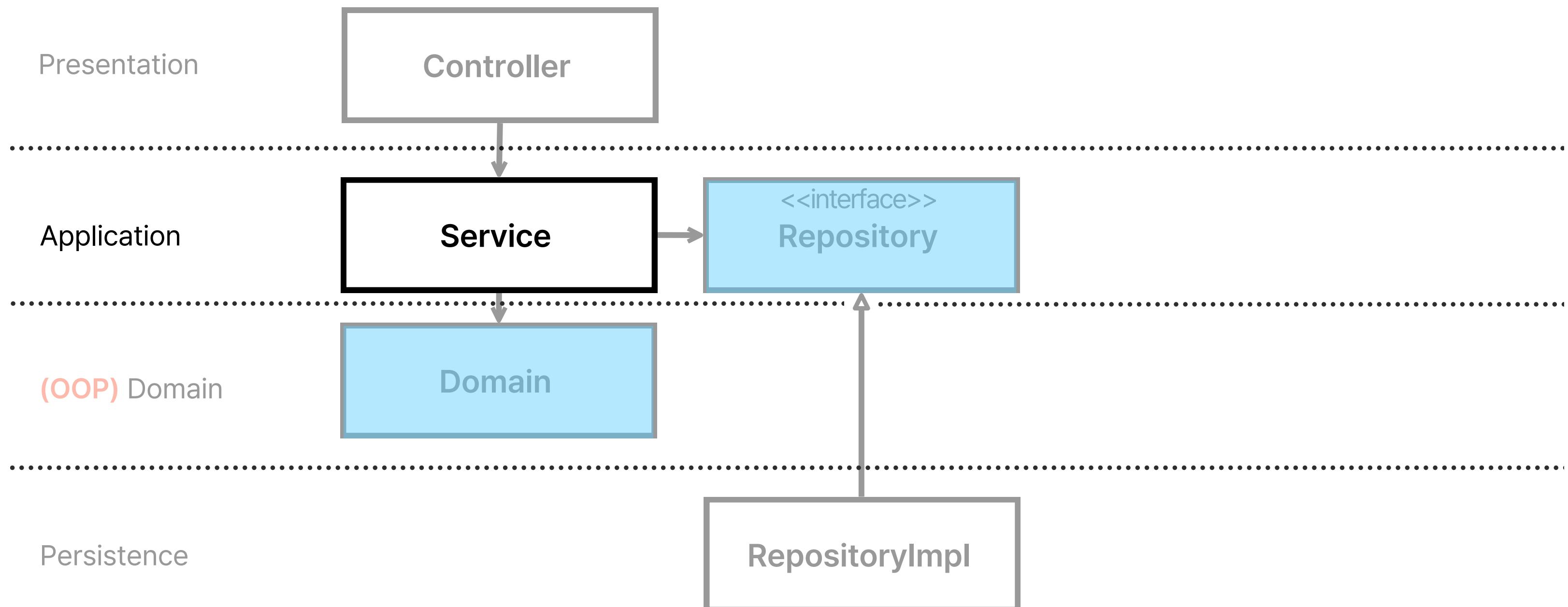
개선된 아키텍처

3. 낮은 Testability



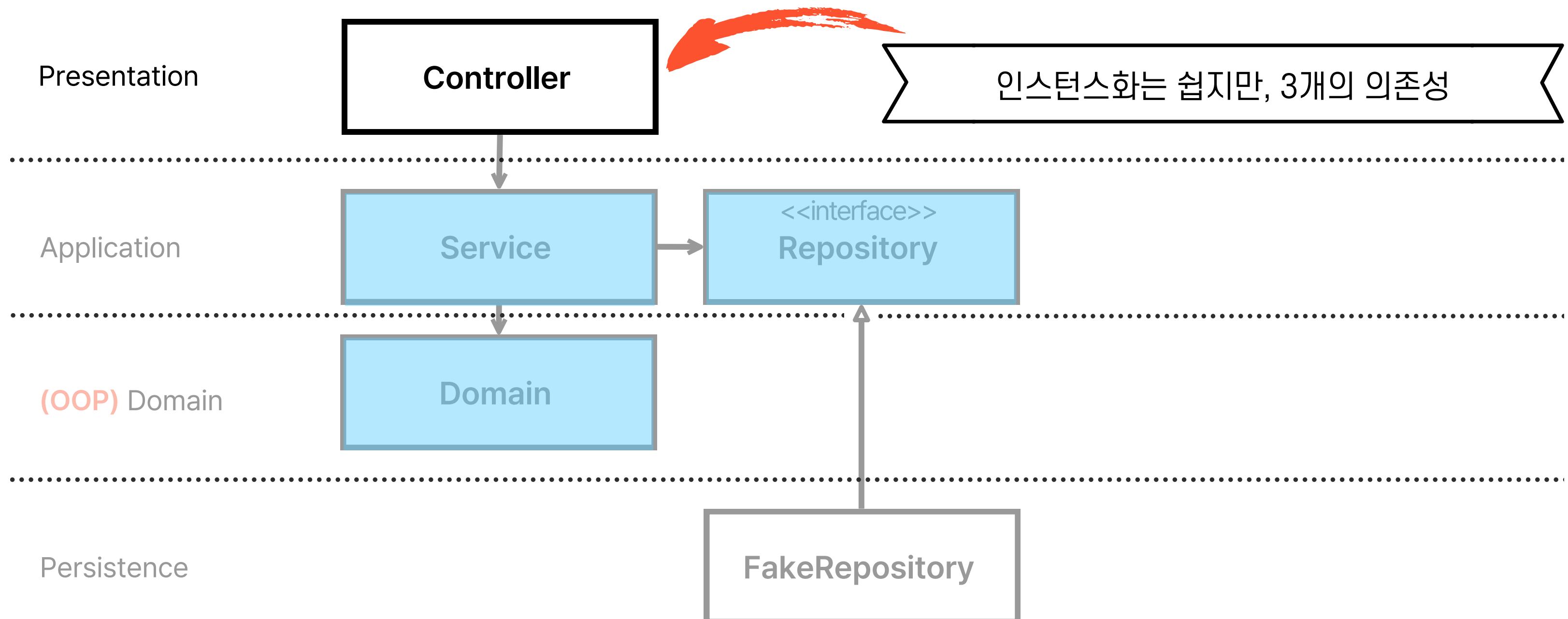
개선된 아키텍처

3. 낮은 Testability



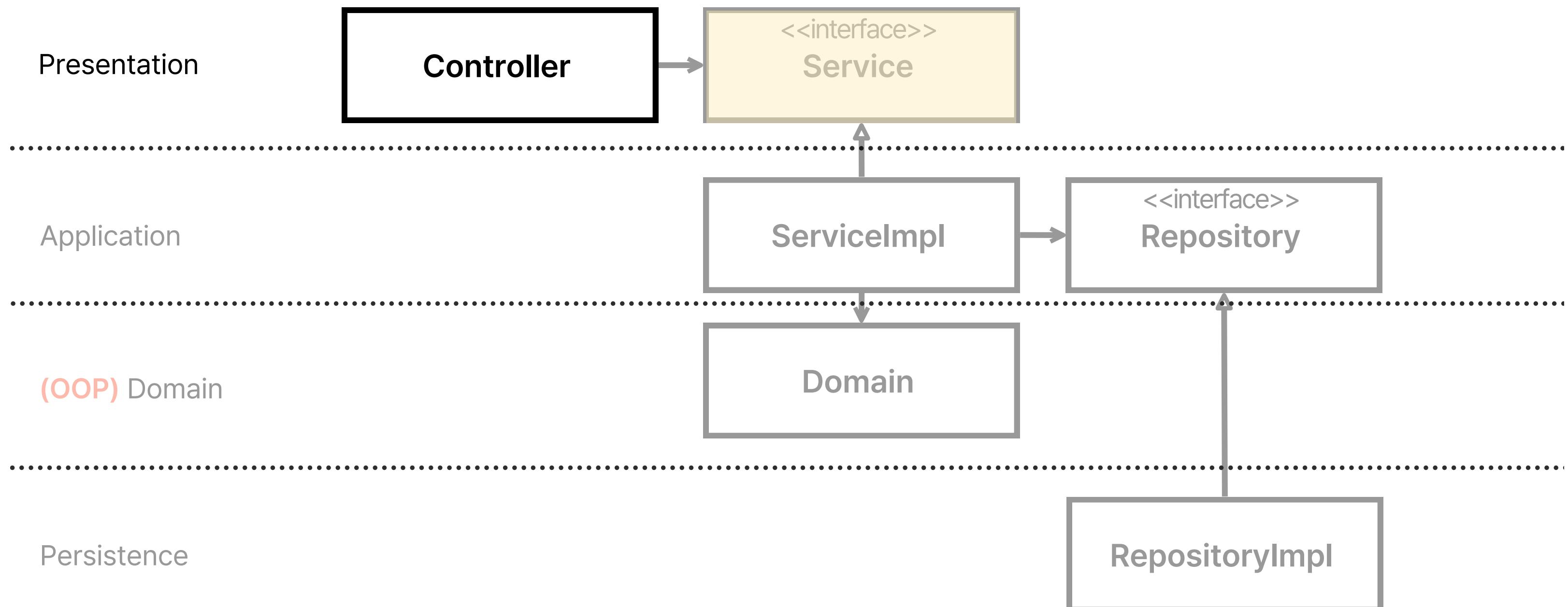
개선된 아키텍처

3. 낮은 Testability



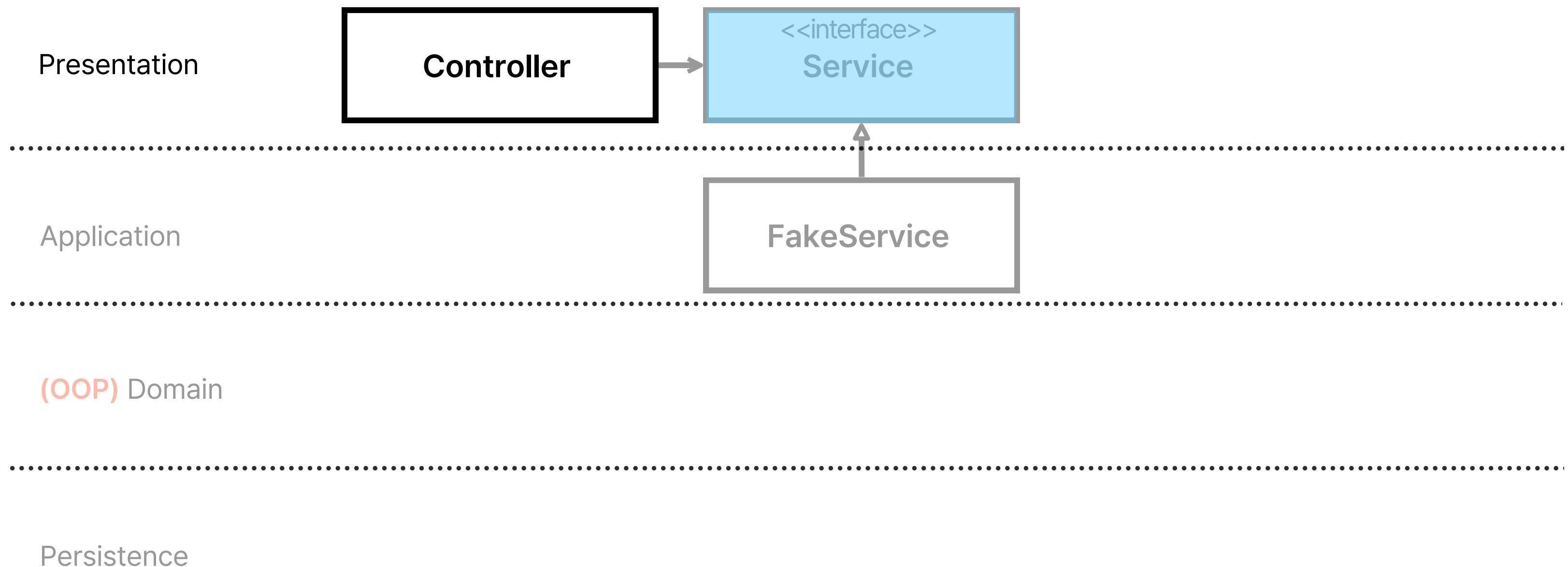
개선된 아키텍처

3. 낮은 Testability

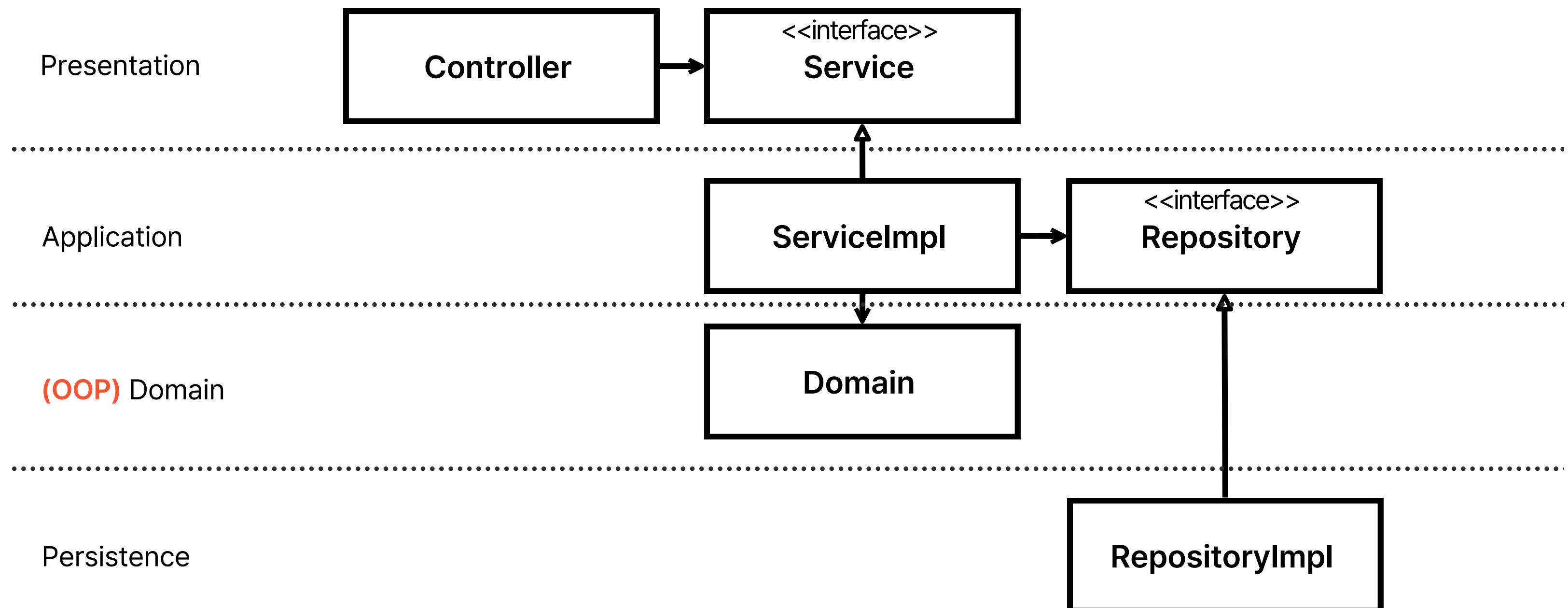


개선된 아키텍처

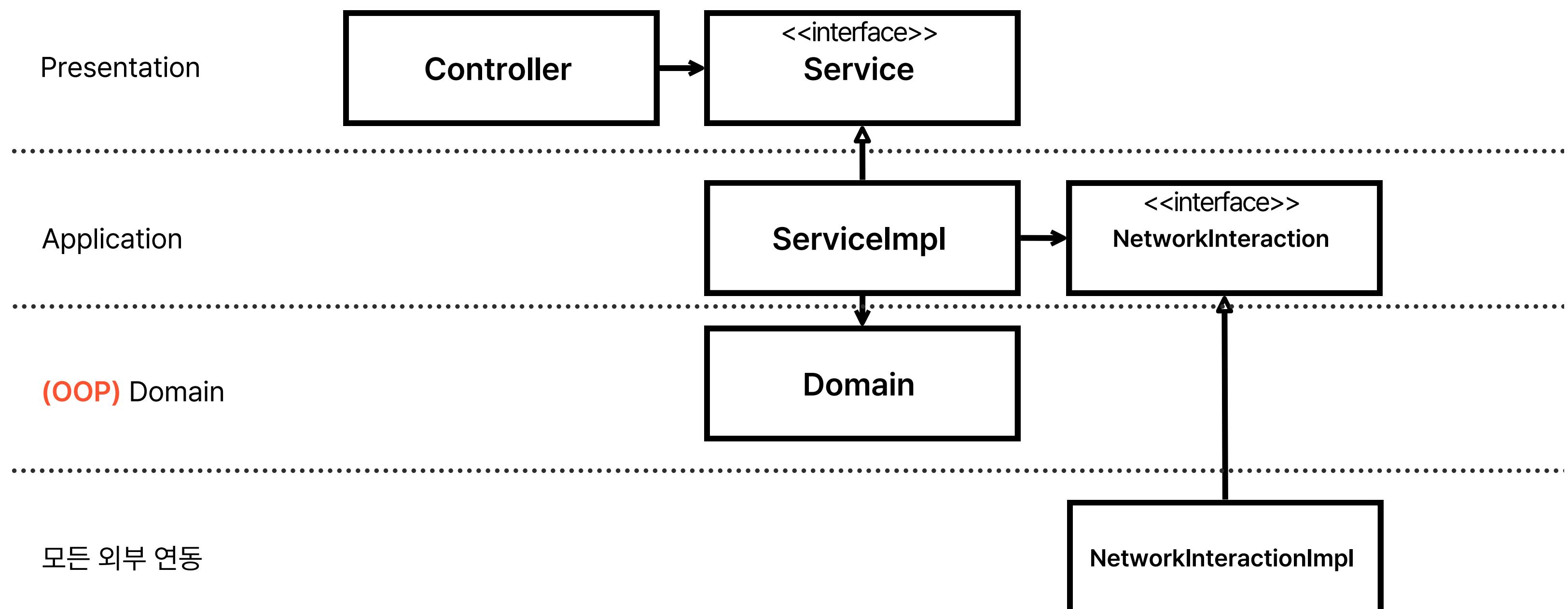
3. 낮은 Testability



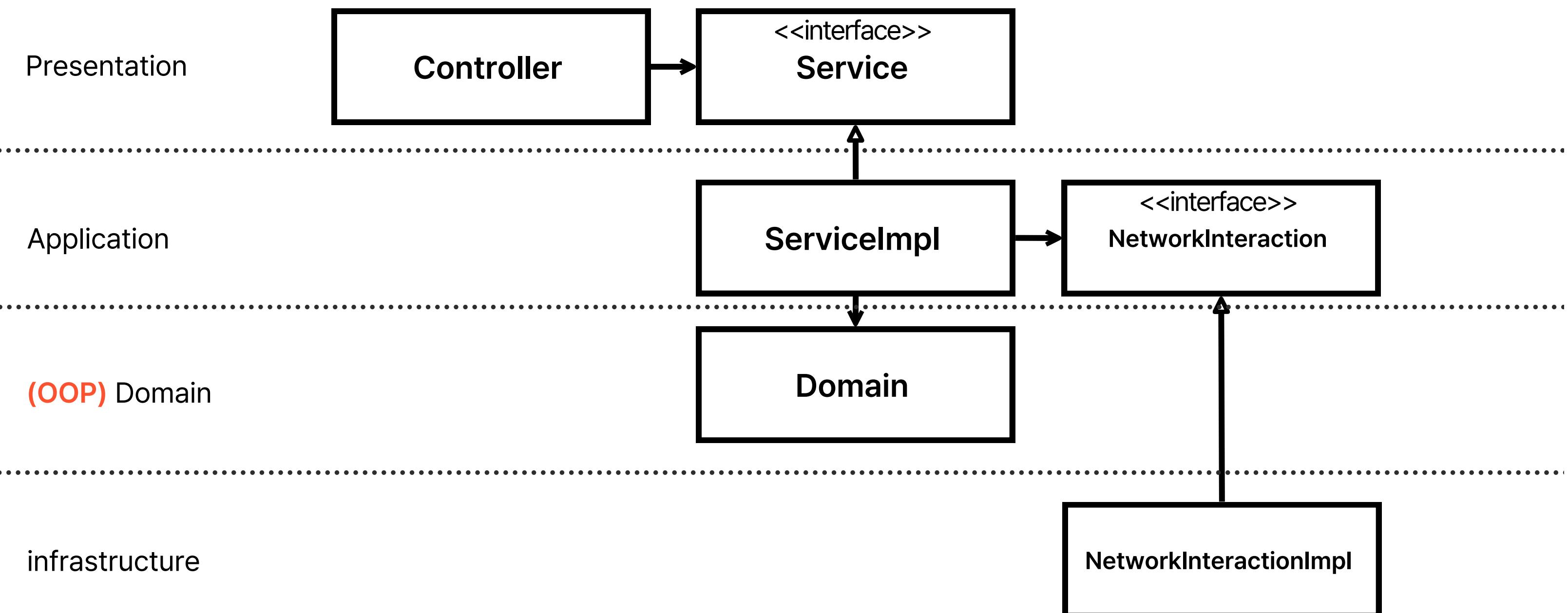
개선된 아키텍처



개선된 아키텍처



개선된 아키텍처



RETURN;

**Testcase
with architecture**

Java/Spring 테스트를 추가하고 싶은 개발자들의 오답노트