

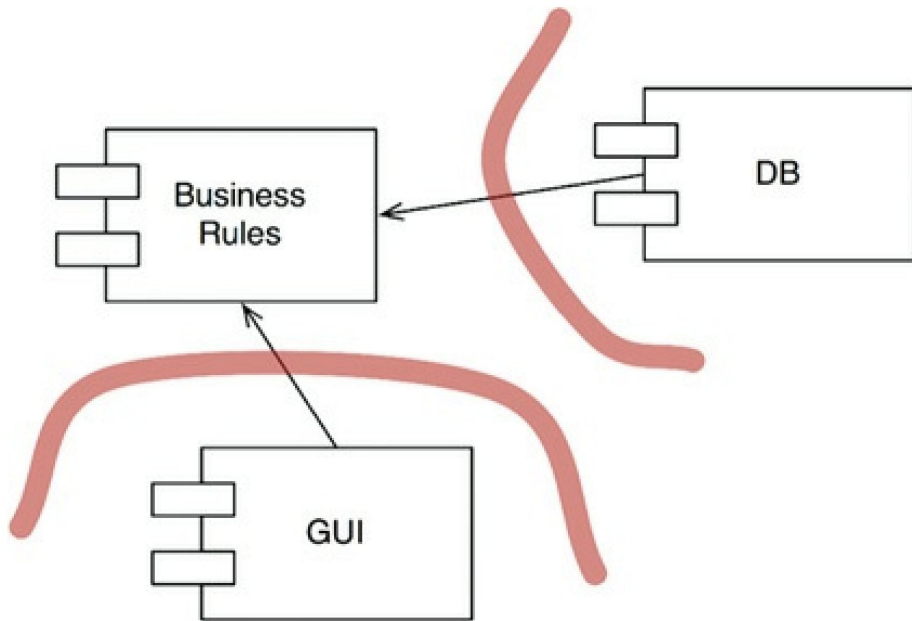
# 18장. 경계해부학



다양한 종류의 경계들을 살펴보기

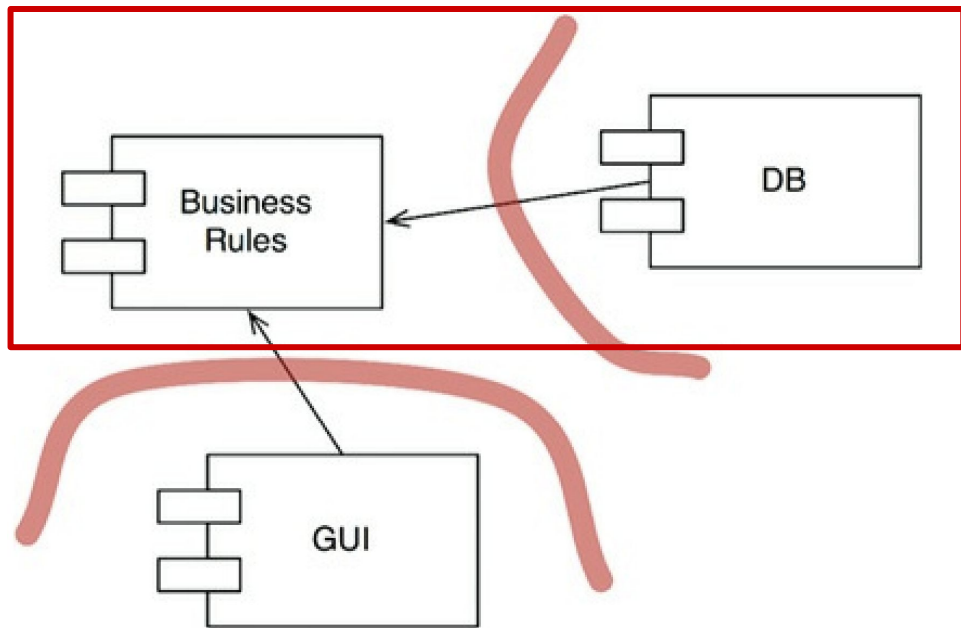
# 시스템 아키텍처

일련의 소프트웨어 **컴포넌트** + 이 컴포넌트들을 분리하는  
**경계**



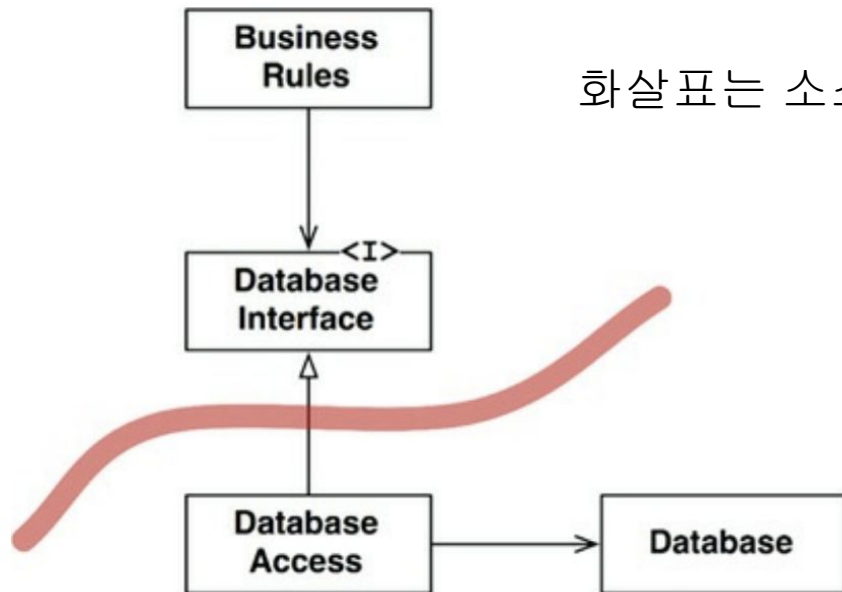
# 시스템 아키텍처

일련의 소프트웨어 **컴포넌트** + 이 컴포넌트들을 분리하는  
경계



# 경계

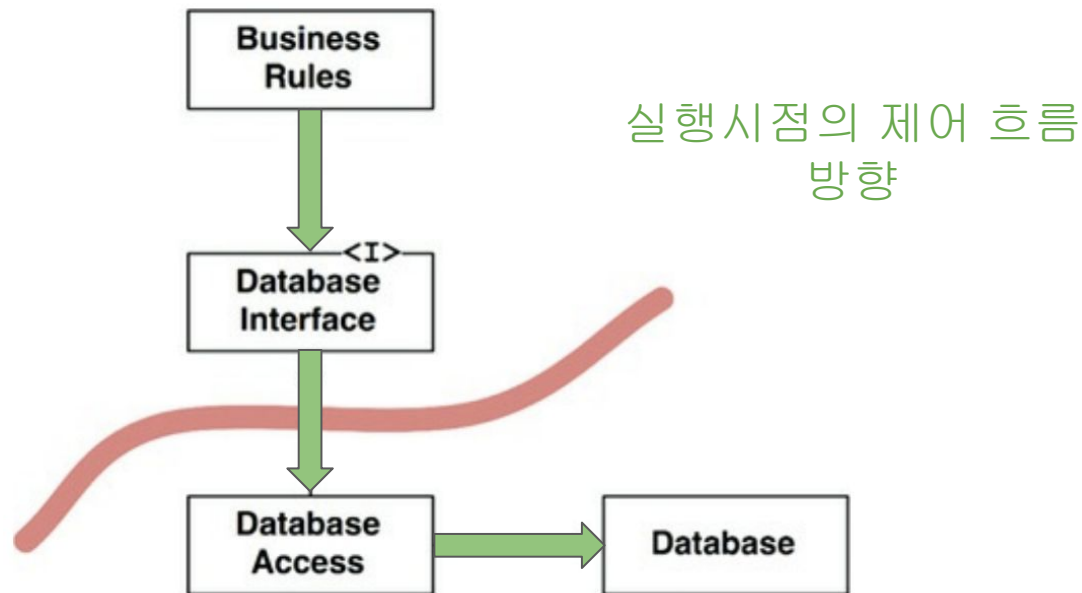
- 경계의 역할: 변경이 전파되는 것을 막는 방화벽



화살표는 소스코드 의존성

# 경계 횡단하기

- 런타임에 횡단



다양한 종류의 경계들을 살펴보기

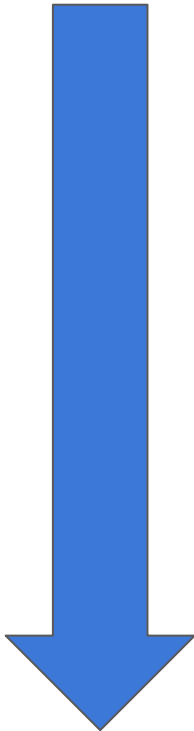
# 다양한 종류의 경계들을 살펴보기

- 소스 수준 분리 모드
  - 소스 코드에서만 경계를 확인 가능
- 배포 수준 결합 분리 모드
  - 배포 과정까지 분리
- 프로세스 수준 결합 분리 모드
  - 각 컴포넌트를 다른 프로세스에서 실행
- 서비스 수준 결합 분리 모드
  - 프로세스가 서로 다른 머신일 수도 있다



# 다양한 종류의 경계들을 살펴보기

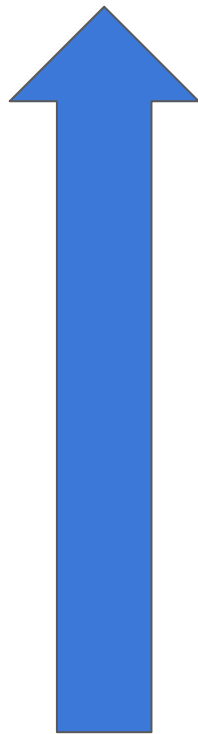
- 소스 수준 분리 모드
  - 소스 코드에서만 경계를 확인 가능
- 배포 수준 결합 분리 모드
  - 배포 과정까지 분리
- 프로세스 수준 결합 분리 모드
  - 각 컴포넌트를 다른 프로세스에서 실행
- 서비스 수준 결합 분리 모드
  - 프로세스가 서로 다른 머신일 수도 있다



물리적으로 더욱  
뚜렷한 경계

# 다양한 종류의 경계들을 살펴보기

- 소스 수준 분리 모드
  - 소스 코드에서만 경계를 확인 가능
- 배포 수준 결합 분리 모드
  - 배포 과정까지 분리
- 프로세스 수준 결합 분리 모드
  - 각 컴포넌트를 다른 프로세스에서 실행
- 서비스 수준 결합 분리 모드
  - 프로세스가 서로 다른 머신일 수도 있다

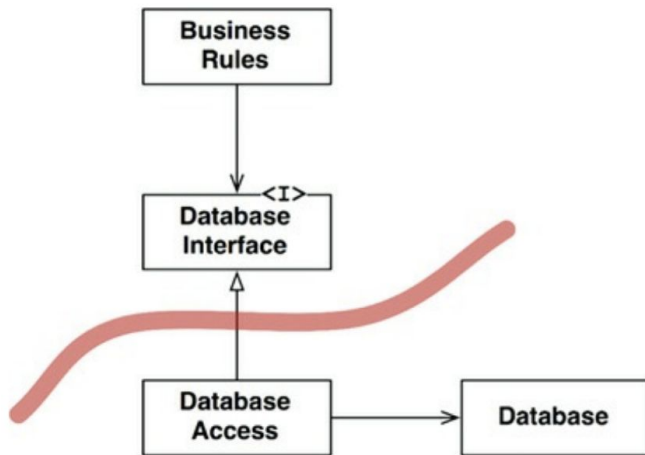


컴포넌트간 통신  
속도

소스 수준 분리 모드: 두려운 단일체

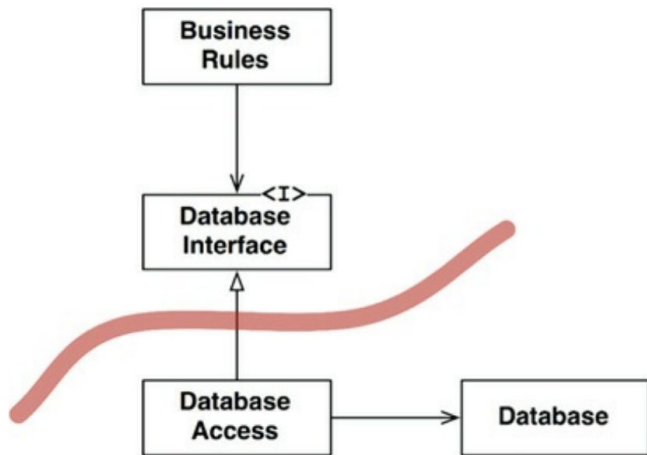
## 소스 수준 분리 모드: 두려운 단일체

- 가장 단순하고, 흔한 형태
- 소스 코드에서만 컴포넌트를 나눠 둔 형태
- 배포하고 보면, 단일체 (Monolith)라고 불리는 단일 실행 파일



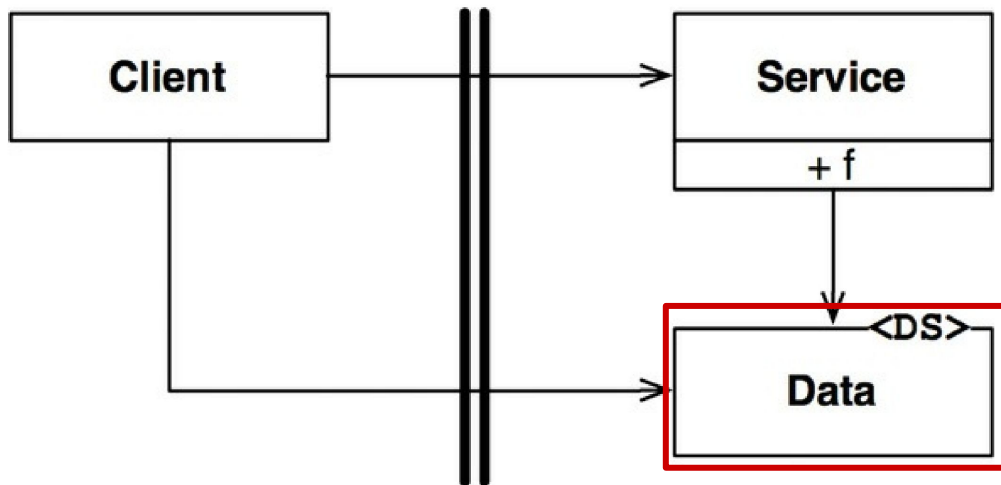
## 소스 수준 분리 모드: 두려운 단일체

- 의미가 없어 보일 수도 있지만, 사실 매우 가치있는 행위
- 거의 모든 경우, 동적 다형성 (= Interface + DIP)에 의존하여 내부 의존성을 관리



## 소스 수준 분리 모드: 두려운 단일체

- 가장 단순한 형태의 경계 횡단:
  - 저수준 Client의 고수준 Service 호출

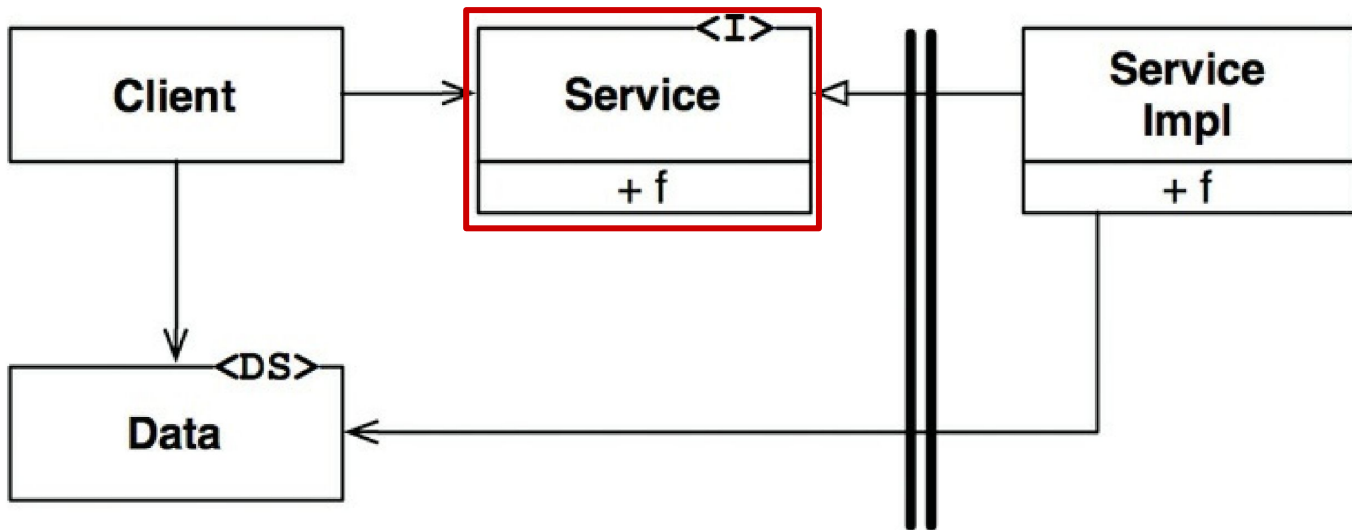


# 소스 수준 분리 모드: 두려운 단일체

- 보다 복잡한 경계:

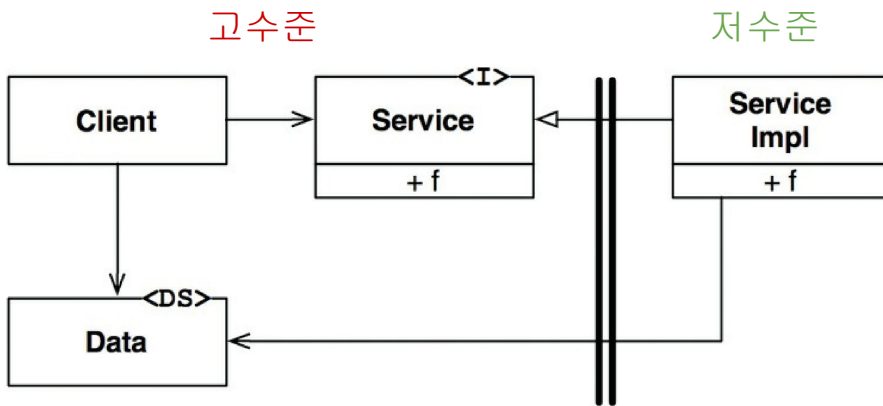
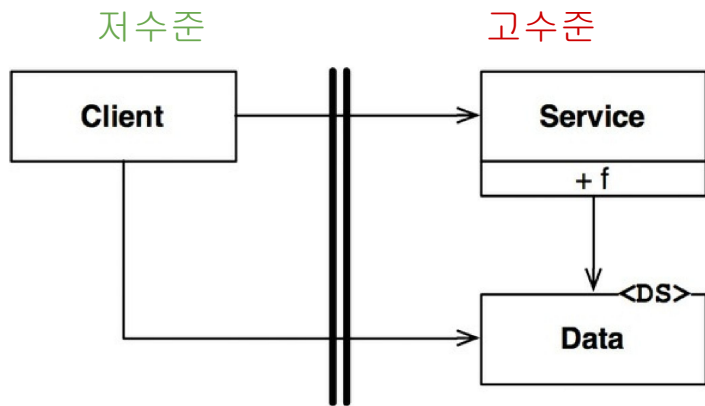
- 고수준 클라이언트의 저수준 서비스 호출
- ex) BusinessRules의 Database를 접근

동적 다형성 + 의존성  
역전



## 소스 코드 의존성 방향: 저수준이 고수준에 의존

- 저수준은 **Plugable**함 = 확장에 열려있음
- 고수준은 저수준의 세부사항으로부터 독립적





## 소스 수준 분리 모드: 두려운 단일체

- 단일체이므로, 하나의 프로세스에 모든 컴포넌트가 올라감
- 컴포넌트들이 같은 주소공간을 공유함
- 통신 속도는 매우 빠른 편: 단순한 함수 호출과 큰 차이 없음
- 컴포넌트 전달 (**Deliver**) 방식: 소스 코드 형태

배포 수준 결합 분리 모드: 배포형  
컴포넌트

# 배포 수준 결합 분리 모드: 배포형 컴포넌트

- 아키텍처의 경계가 물리적으로 드러나는 가장 간단한 형태
- 컴포넌트들을 동적 링크 라이브러리 형태로 배포하는 방식
  - ex) Jar 파일, 루비 Gem, .NET DLL 등
- 배포 과정:
  - 각 컴포넌트들을 바이너리로 배포
  - 배포된 바이너리들을 묶음
- 컴포넌트 전달 (**Deliver**) 방식: 바이너리와 같이 배포 가능한 형태
- 바이너리로 배포 가능한 컴포넌트들 = 배포 수준 컴포넌트 (Deployment-level Component) = Monolith

## 배포 수준 결합 분리 모드: 배포형 컴포넌트

- 컴포넌트간 통신 속도는 동적 링크와 런타임 로딩 등으로 최초에는 느릴 수 있으나, 이후에는 함수 호출과 큰 차이 없이, 매우 빠른 편.

프로세스 수준 결합 분리 모드

# 프로세스 수준 결합 분리 모드

- 훨씬 강한 수준의 아키텍처 경계를 가짐
- 각 컴포넌트들을 같은 머신의 서로 다른 로컬 프로세스에 띄우는 방식
  - 각 컴포넌트들은 OS가 제공하는 프로세스 간 통신 방법을 통해야만 함
  - 정적으로 링크된 단일체 or 동적으로 링크된 여러 컴포넌트
- 로컬 프로세스 = 최상위 컴포넌트 = 여러 저수준 컴포넌트들로 이뤄짐

## 프로세스 수준 결합 분리 모드

- 여전히 소스 코드 의존성 방향은 저수준 -> 고수준 방향이어야 함
- 고수준 컴포넌트의 프로세스가 저수준 컴포넌트 프로세스의 세부정보에 대해 알고 있으면 안됨
  - 예를 들어, 프로세스 아이디, 물리 주소 등을 알면 안됨
- 프로세스간 통신 비용은 비싸므로, 통신이 너무 빈번하지 않도록 해야함

서비스 수준 결합 분리 모드

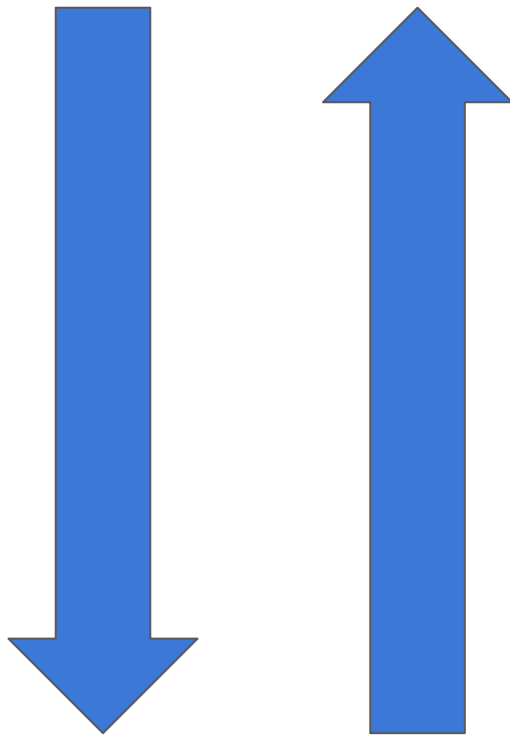


## 서비스 수준 결합 분리 모드

- 로컬 프로세스와는 달리, 각 컴포넌트가 서로 다른 머신의 프로세스인 경우까지 확장
- 가장 강력한 형태의 물리적 경계를 가짐
- 네트워크 통신을 통해, 컴포넌트간 경계 횡단이 이뤄짐
- 따라서, 너무 빈번한 통신은 피해야 함

# 결론

- 소스 수준 분리 모드
  - 소스 코드에서만 경계를 확인 가능
- 배포 수준 결합 분리 모드
  - 배포 과정까지 분리
- 프로세스 수준 결합 분리 모드
  - 각 컴포넌트를 다른 프로세스에서 실행
- 서비스 수준 결합 분리 모드
  - 프로세스가 서로 다른 머신일 수도 있다



## 결론

- Trade-Off를 고려하여, 여러 종류의 경계 전략을 혼합하여 사용