Design Concepts

서동준, 윤민성, 하유성

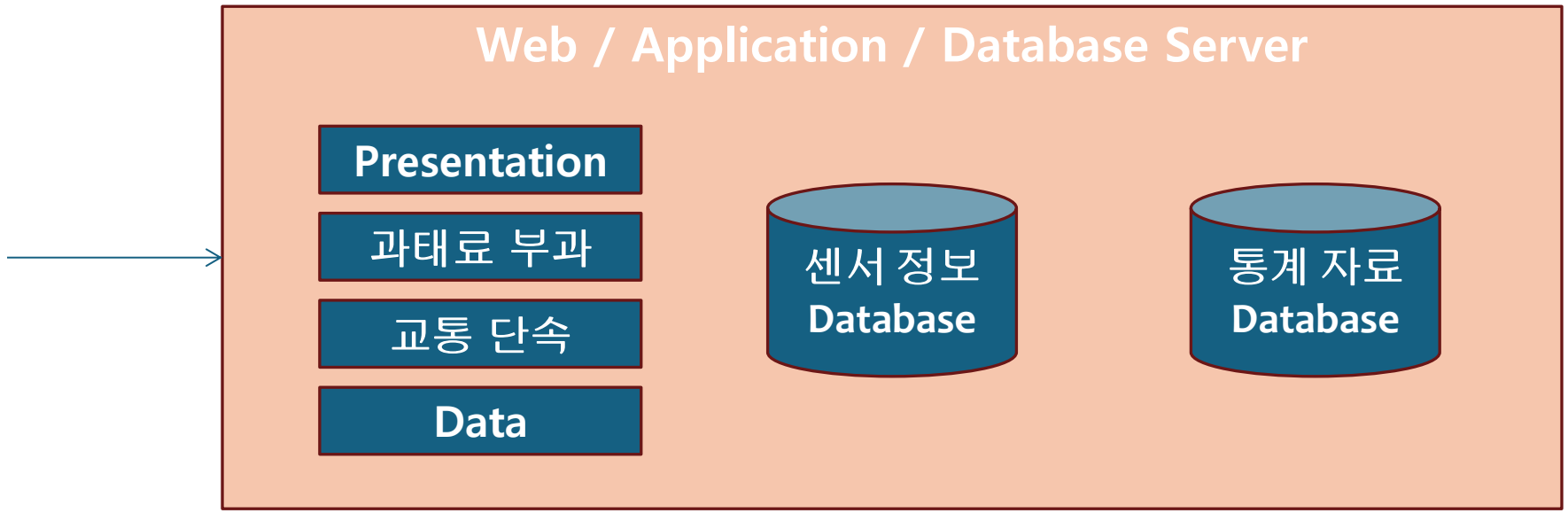
목차

- Deployment Pattern
  - NonDistributed
  - distributed
  - security
- Architecture Style
  - Server discovery pattern
  - Pipeline
  - Microservices
  - Master and slave
- Externally Developed Components
  - Application framework
  - Products
  - platform

목차

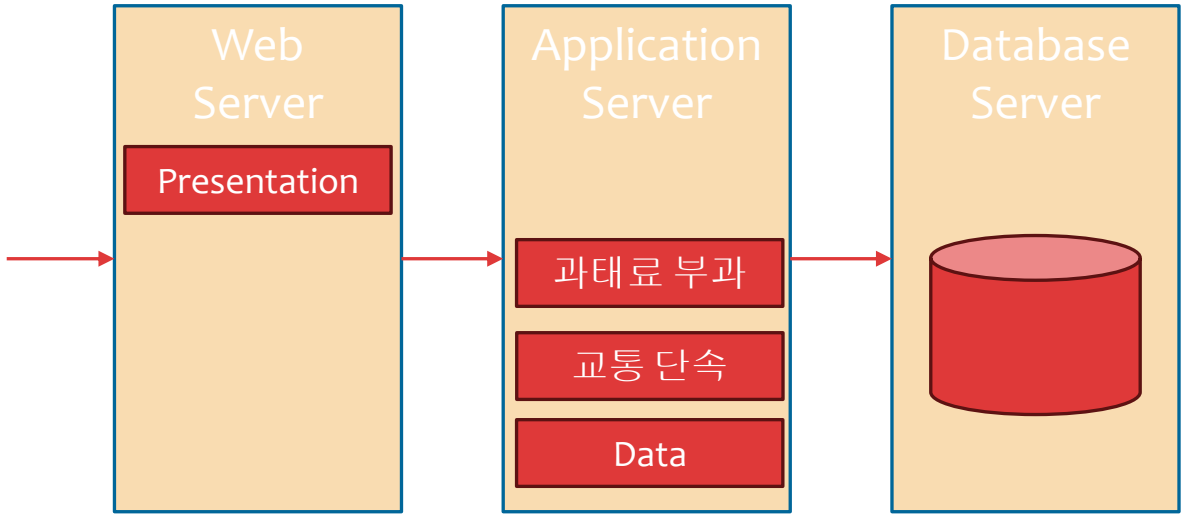
- Tactics
  - Testability
  - Usability
  - Maintainability
  - Availability
  - Security
  - Performance
  - Modifiability
  - Interoperability
  - Scalability
  - depolyability

Deployment Pattern
Nondistributed



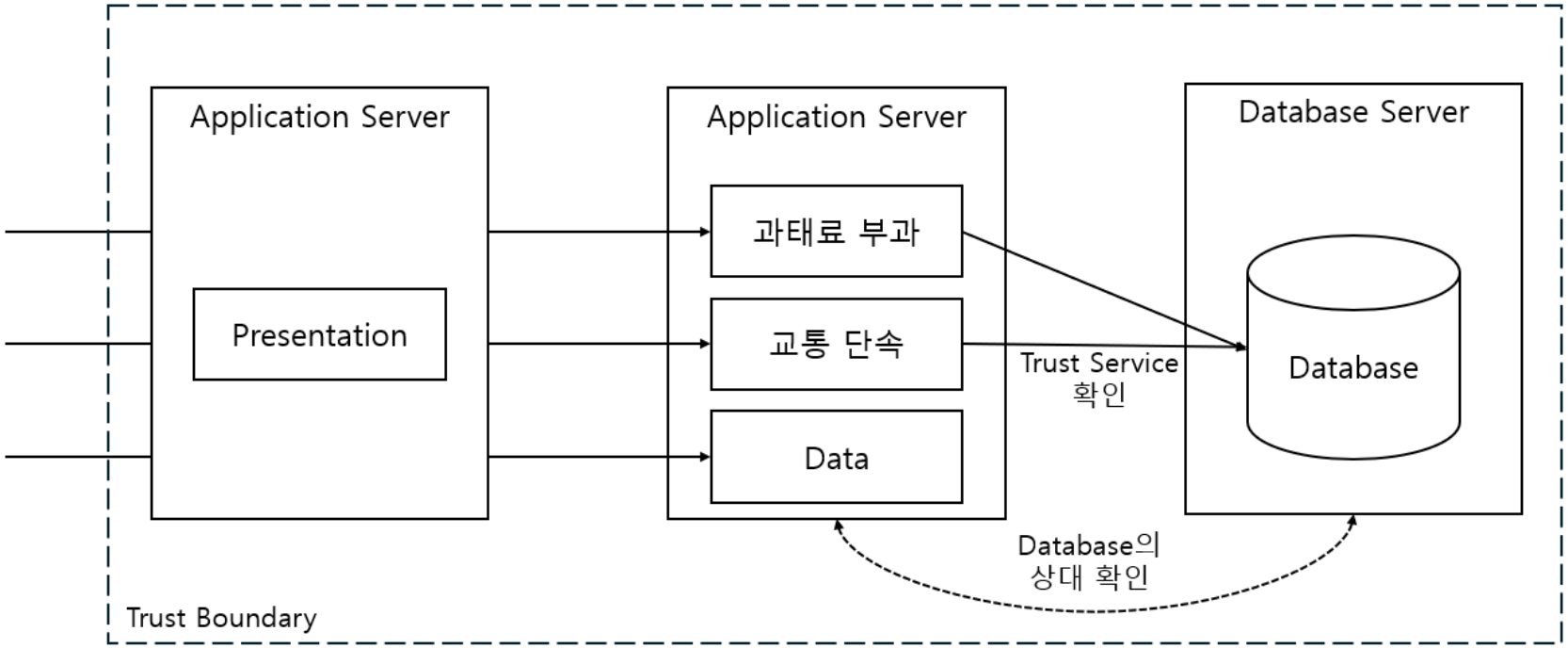
- 단일한 서버로 통합하여 운영하는 방식
- 관리가 용이, 구축시간이 짧음
- 서버에 문제가 생기면 전체 시스템이 다운

Deployment Pattern
Distributed



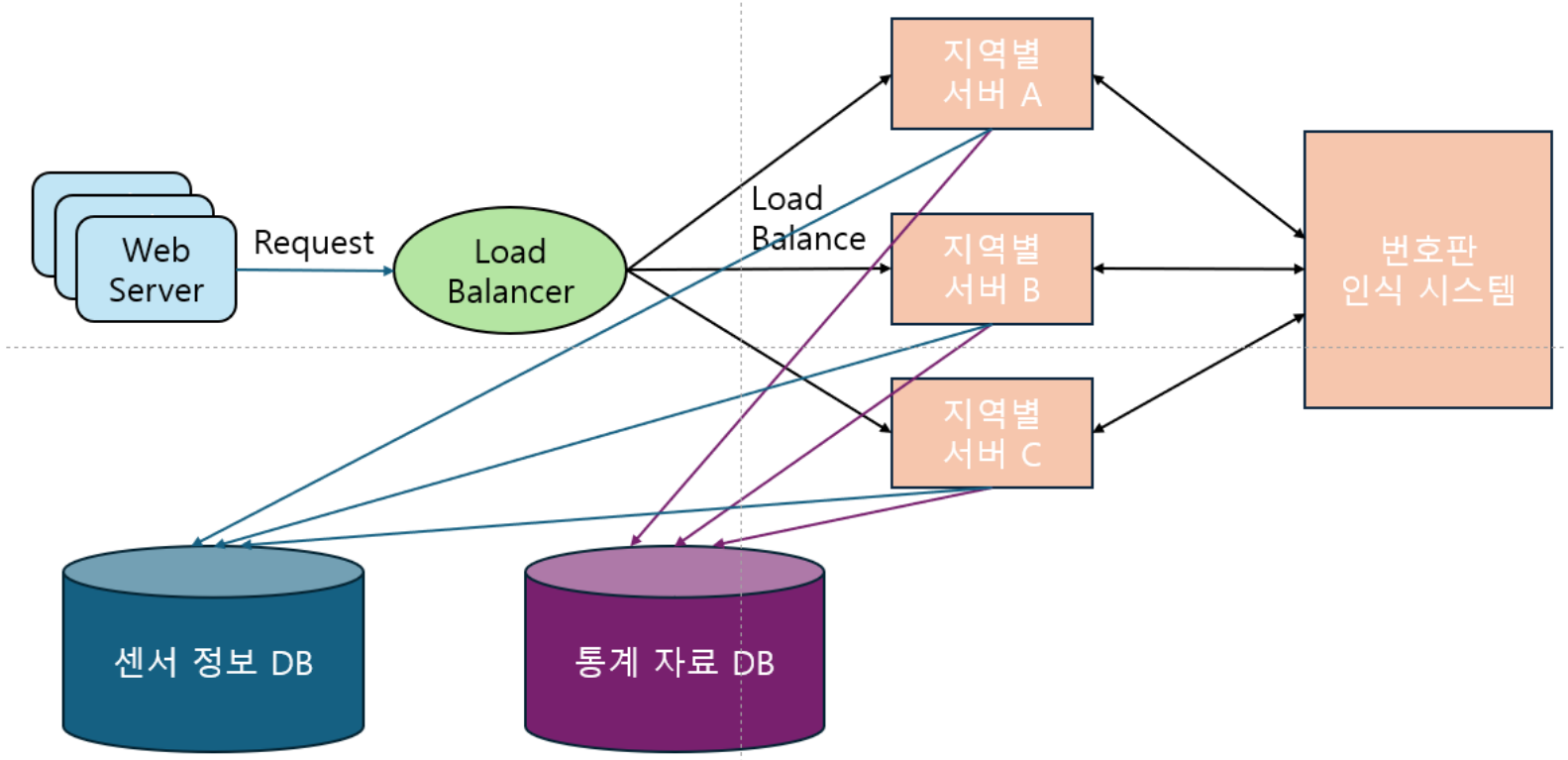
- 웹 서버를 이용하여 가시적 부분을 해결하고 어플리케이션 서버에 수행 동작을 분리하여 운영하는 방식
- 용도에 따른 구분으로 인하여 부하가 분산
- 관리 및 구축에 요구 사항이 발생

Deployment Pattern
Security



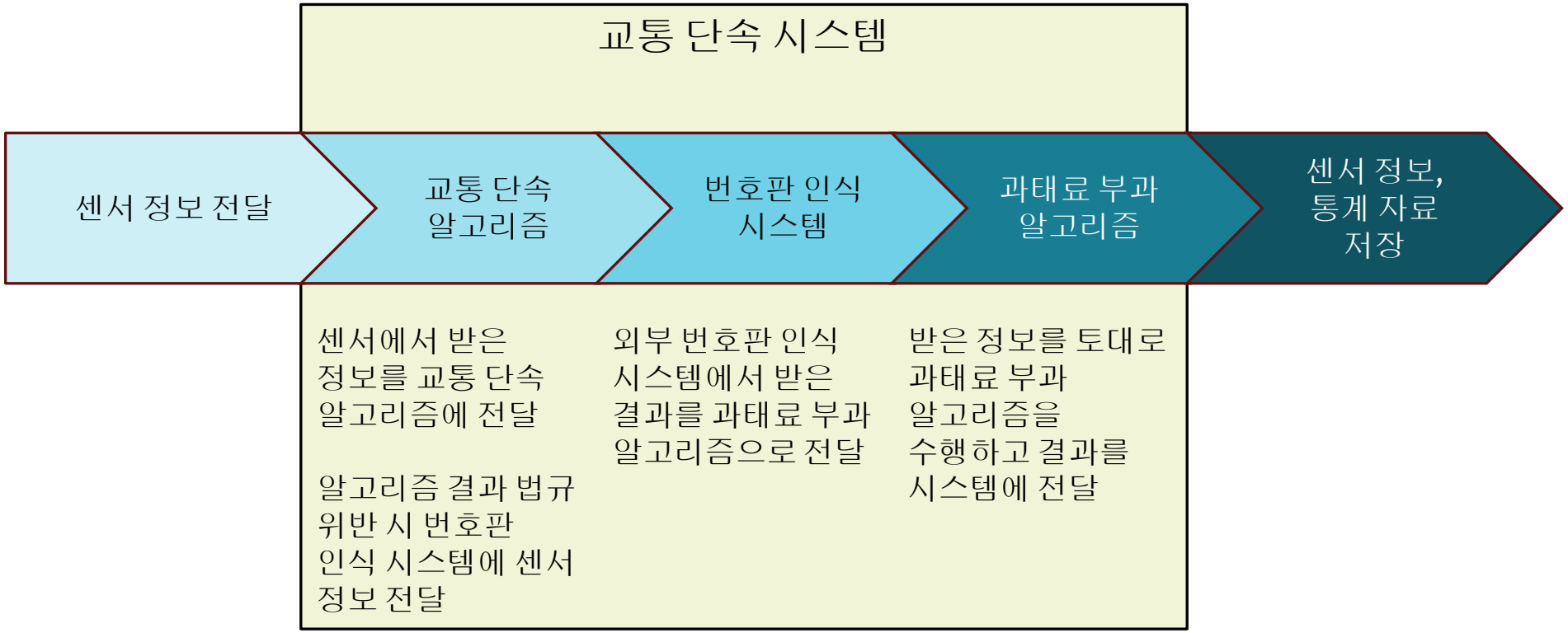
- **Application Server**와 **Database Server** 사이에 신뢰성 확인 후 전송
- 더 안전한 데이터 전송 가능
- 신뢰성 확인에 소모되는 시간 증가

Architecture Style
Server discovery pattern



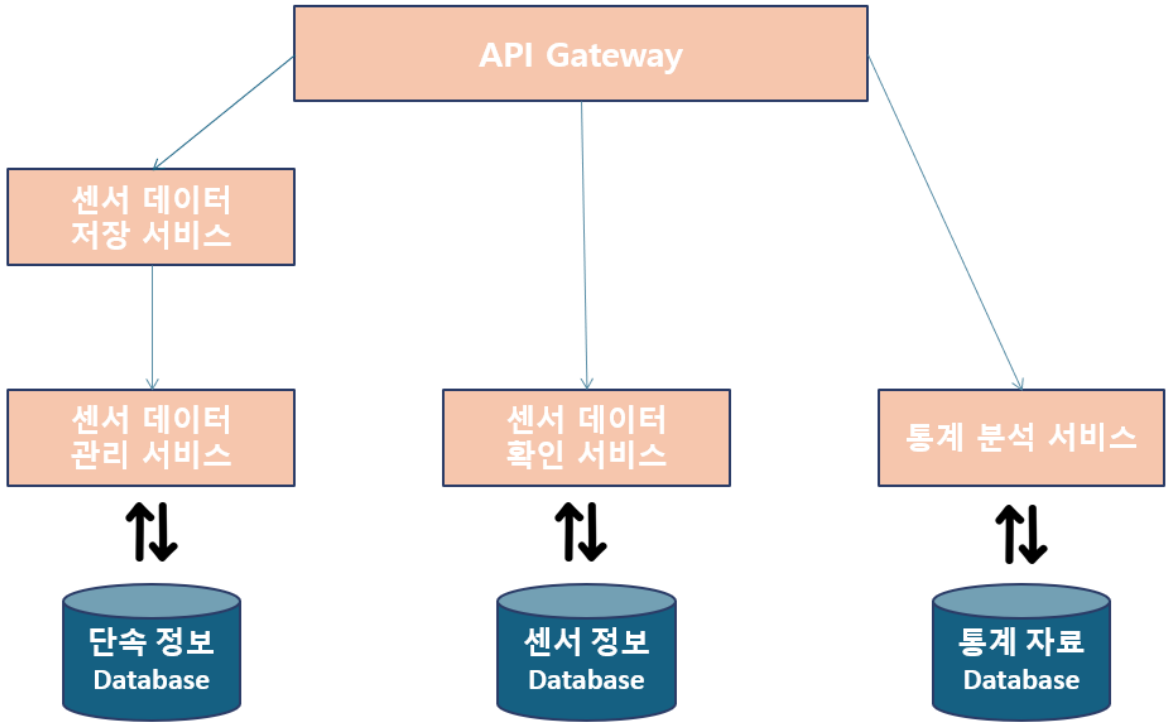
- **Load Balancer**를 통한 부하 분산으로 균일한 서비스를 제공
- 고장 및 과부하 등 상황에 효과적인 대처 가능
- 추가적인 기술이 필요, 기술로 인한 오버헤드 발생

Architecture Style
Pipeline



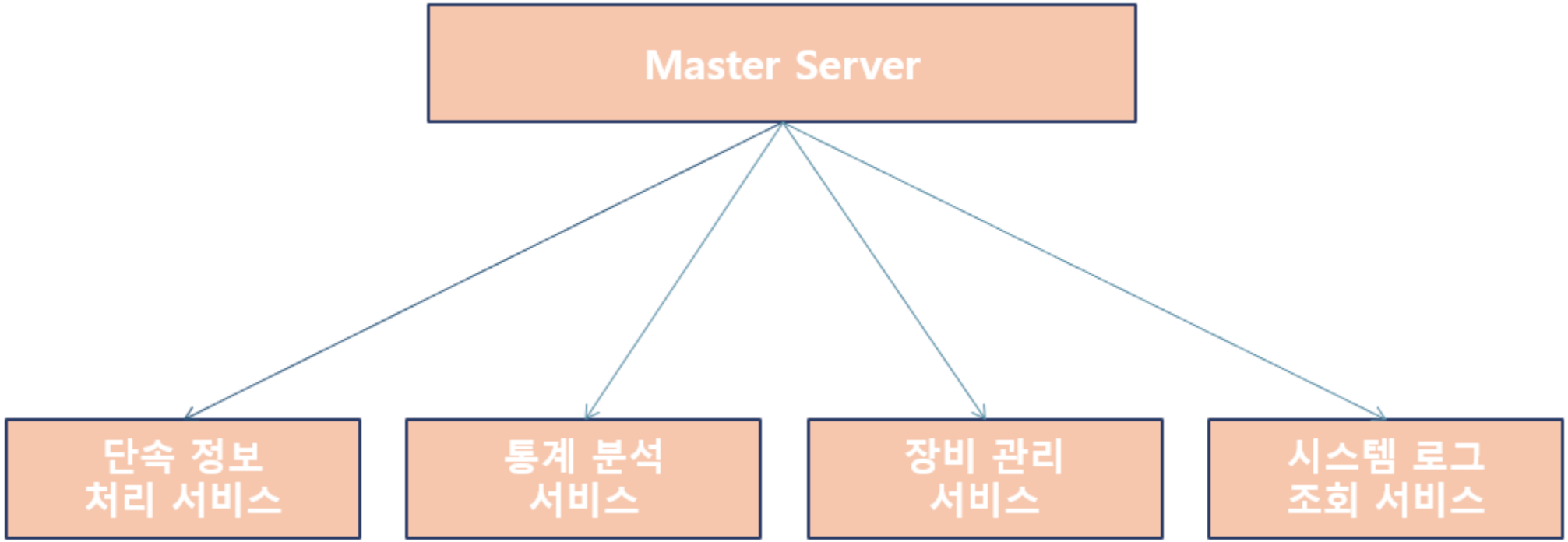
- 데이터 스트림 절차의 각 단계를 필터 컴포넌트로 캡슐화하여 파이프를 통해 데이터를 전송하는 패턴
- 여러 컴포넌트를 잘게 나누어 순차적 개발 가능, 필터 유지 및 재사용 용이
- 상태 정보를 공유하는데 비용이 많이 들고, 데이터 변환에 오버헤드가 발생

Architecture Style
Microservices



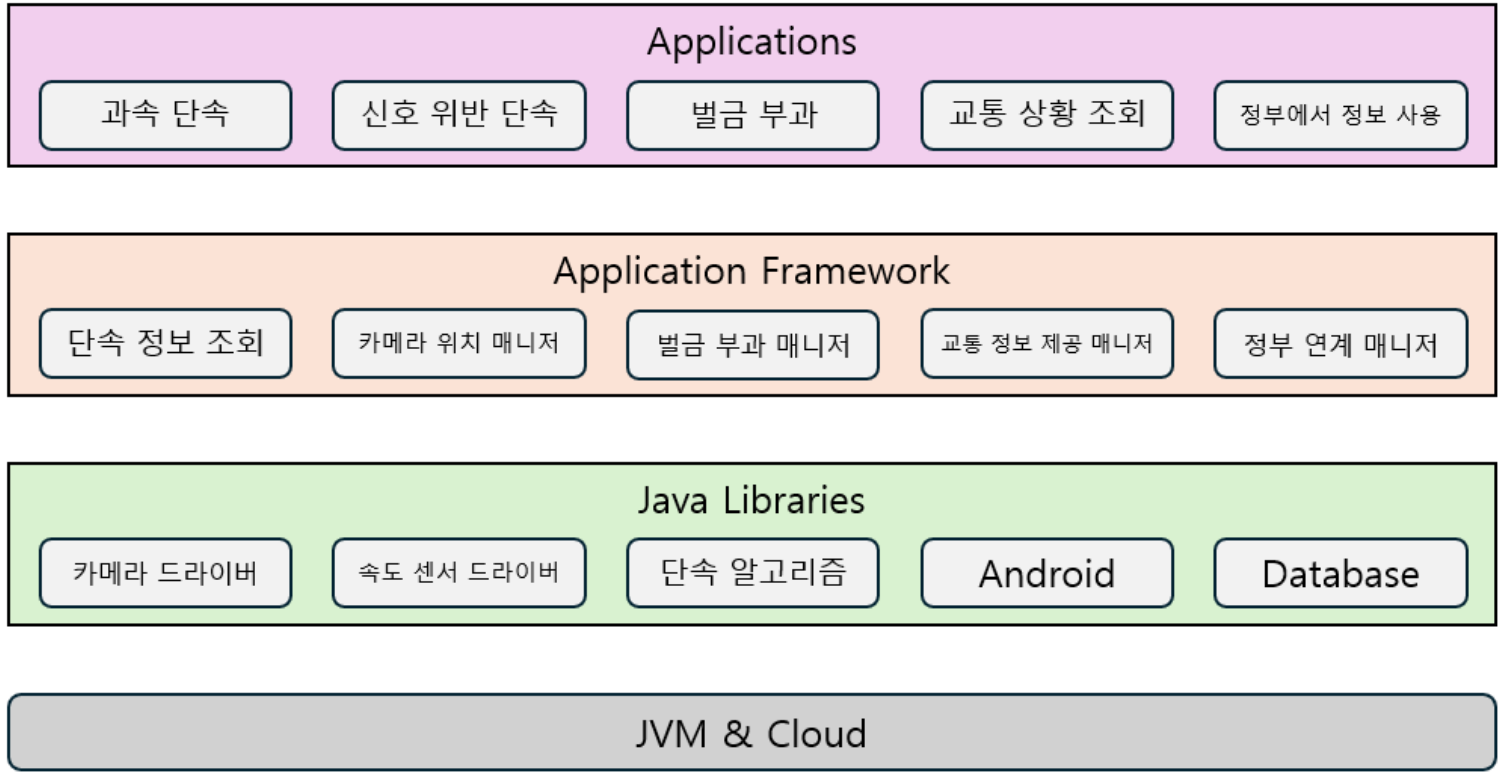
- 네트워크를 통해 서로 통신하는 작고 독립적인 서비스 집합으로 구축하는 스타일
- 확장성이 좋아서 유연성 증가, 새로운 기능의 빠른 출시가 가능
- 서비스 간 통신 복잡성과 데이터 일관성 문제로 인해 운영 및 관리가 어려움, 여러 서비스 인프라 운영으로 인해서 초기 비용이나 복잡성이 증가

Architecture Style
Master and slave



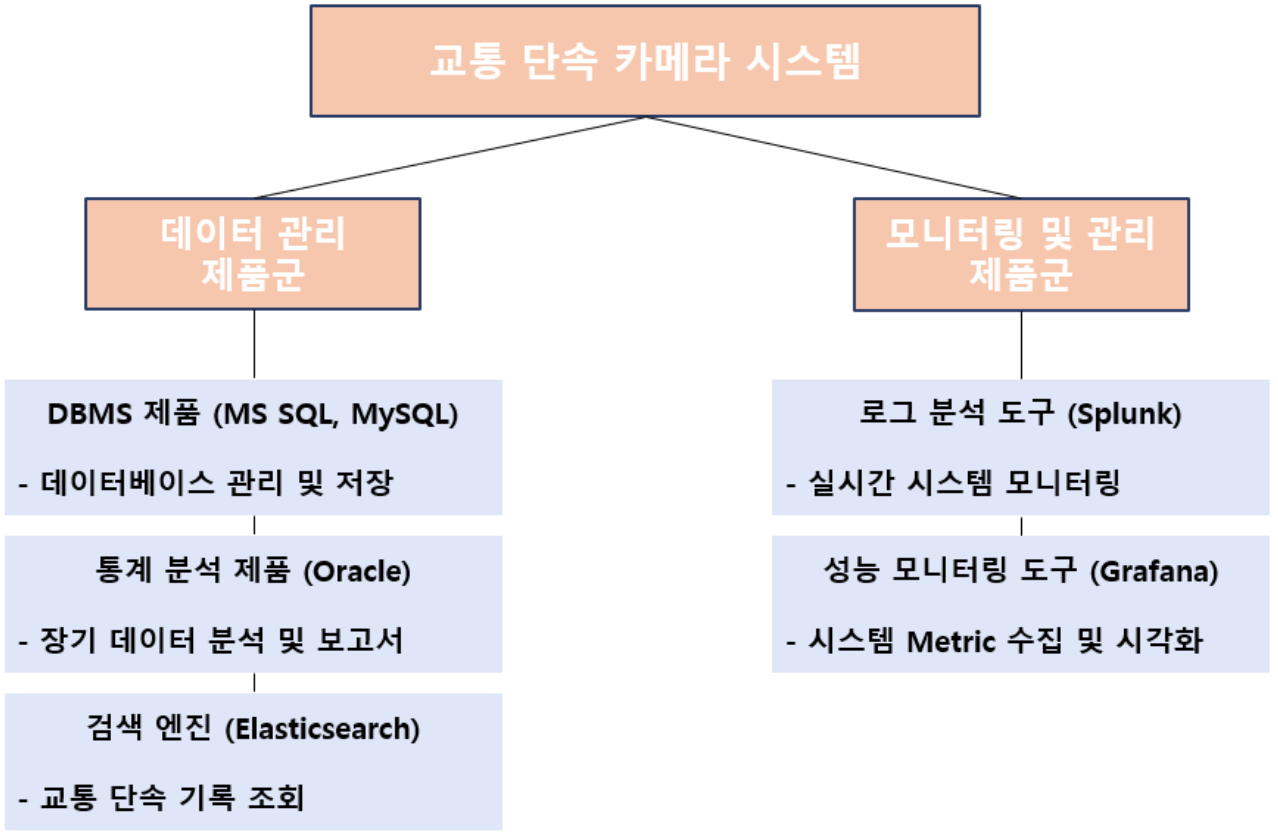
- **master** 서버가 주요 명령을 관리하며, 여러 개의 **slave** 서비스가 해당 명령을 받아 작업을 수행하는 구조, **master**는 작업을 분배하고, **slave**들은 그에 따른 작업을 처리
- **master**가 작업을 여러 **slave**에게 나누어 처리함으로써 부하 분산이 가능, 새로운 **slave**를 추가함으로써 시스템 성능을 쉽게 확장 가능
- **master** 서버에 장애가 발생하면 전체 시스템에 영향을 미칠 수 있음, **master**와 **slave** 간의 통신 지연이 성능 저하를 야기할 수 있음

Externally Developed Components
Application framework



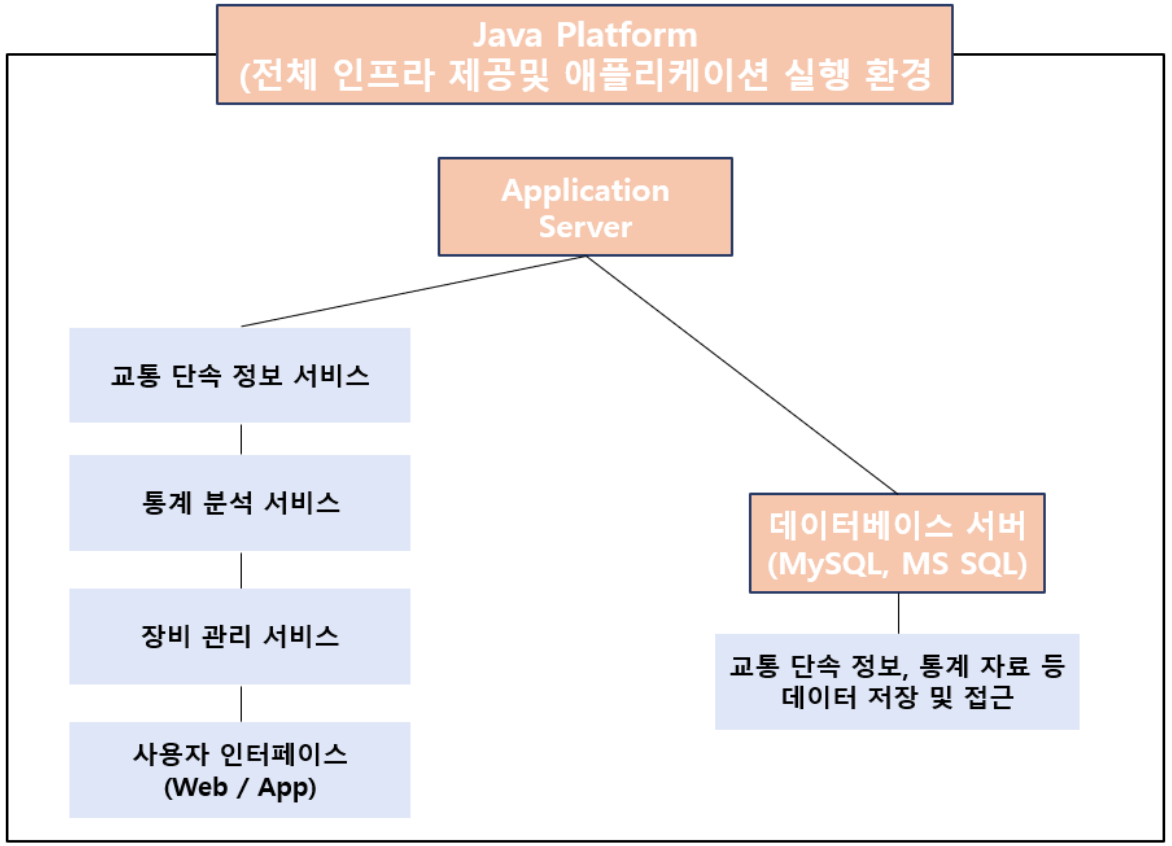
- **Application SW**의 표준 구조를 구현하는 데 사용하는 **SW Framework**
- 일관된 개발을 지원하고, 코드 작성 등이 효율적
- 개발 환경이 자유롭지 못 함

Externally Developed Components Products



- 시스템에 통합할 수 있는 독립적인 기능을 가진 소프트웨어 패키지
- 소규모 설정만으로 쉽게 통합 가능하여 개발 시간을 줄임
- 상용 제품 사용에 따른 추가적인 라이선스 비용 발생 및 외부 제품에 대한 의존성이 발생하기 때문에 시스템 커스터마이징에 대해서 제약이 있을 수 있음

Externally Developed Components Platform

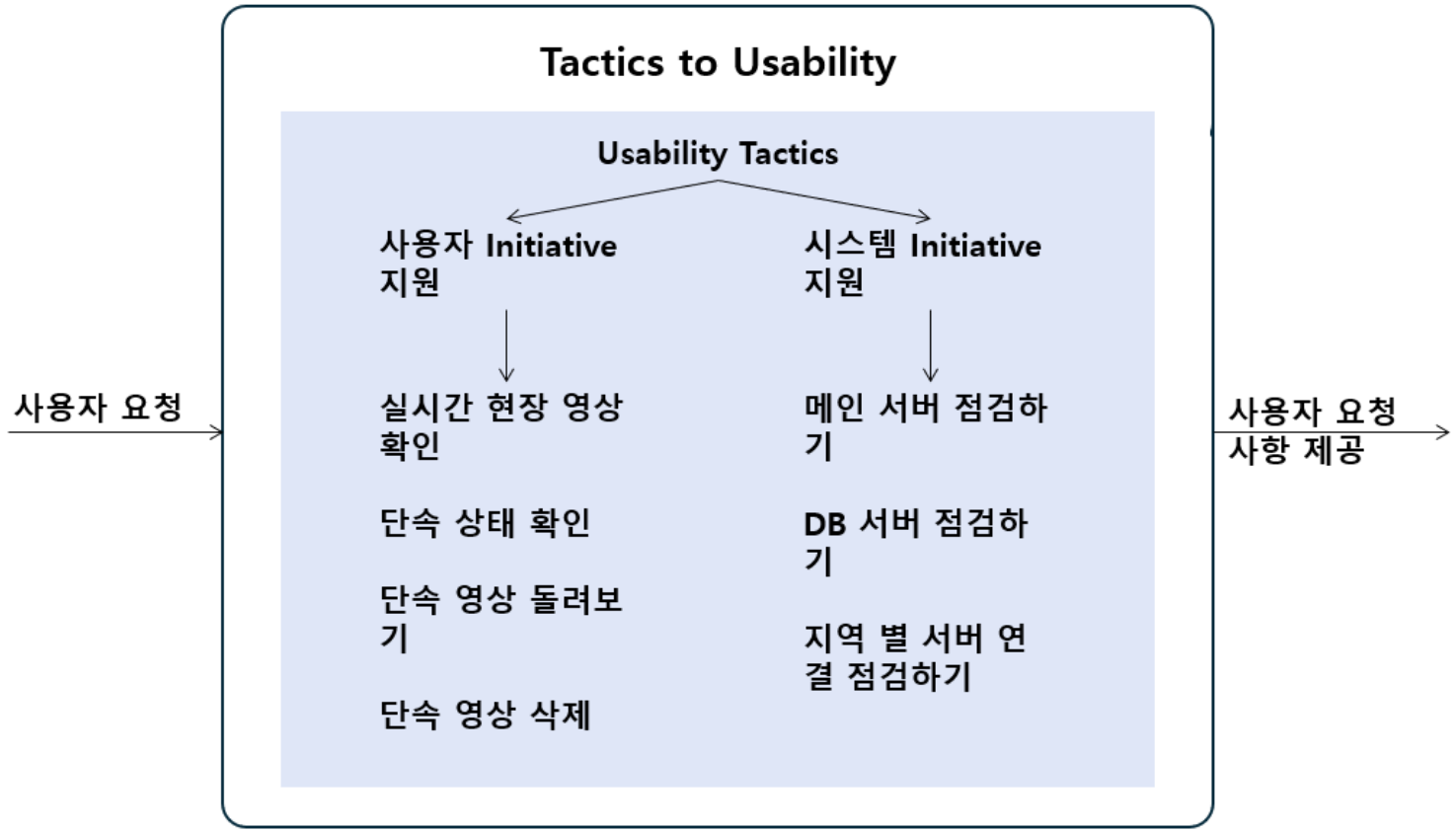


- 시스템을 개발하고 실행하기 위한 완전한 인프라를 제공하는 환경
- 통합된 개발 및 실행 환경을 제공하여 개발과 배포의 일관성을 유지할 수 있음
- 특정 플랫폼에 대한 의존성이 발생하여 다른 플랫폼으로 전환 시 많은 기회비용(노력이나 비용)필요할 수 있음

Tactics
Testabilility

- 시스템을 효과적으로 테스트될 수 있도록 설계하는 것에 초점을 맞춤
- 결함을 조기 발견하고 수정할 수 있음

Tactics
Usability



- 사용자가 시스템을 쉽게 이해하고 사용할 수 있게 하여, 사용자 경험을 개선하는 것에 중점을 맞춤
- 유저의 요청을 효과적으로 처리하는 것이 가능

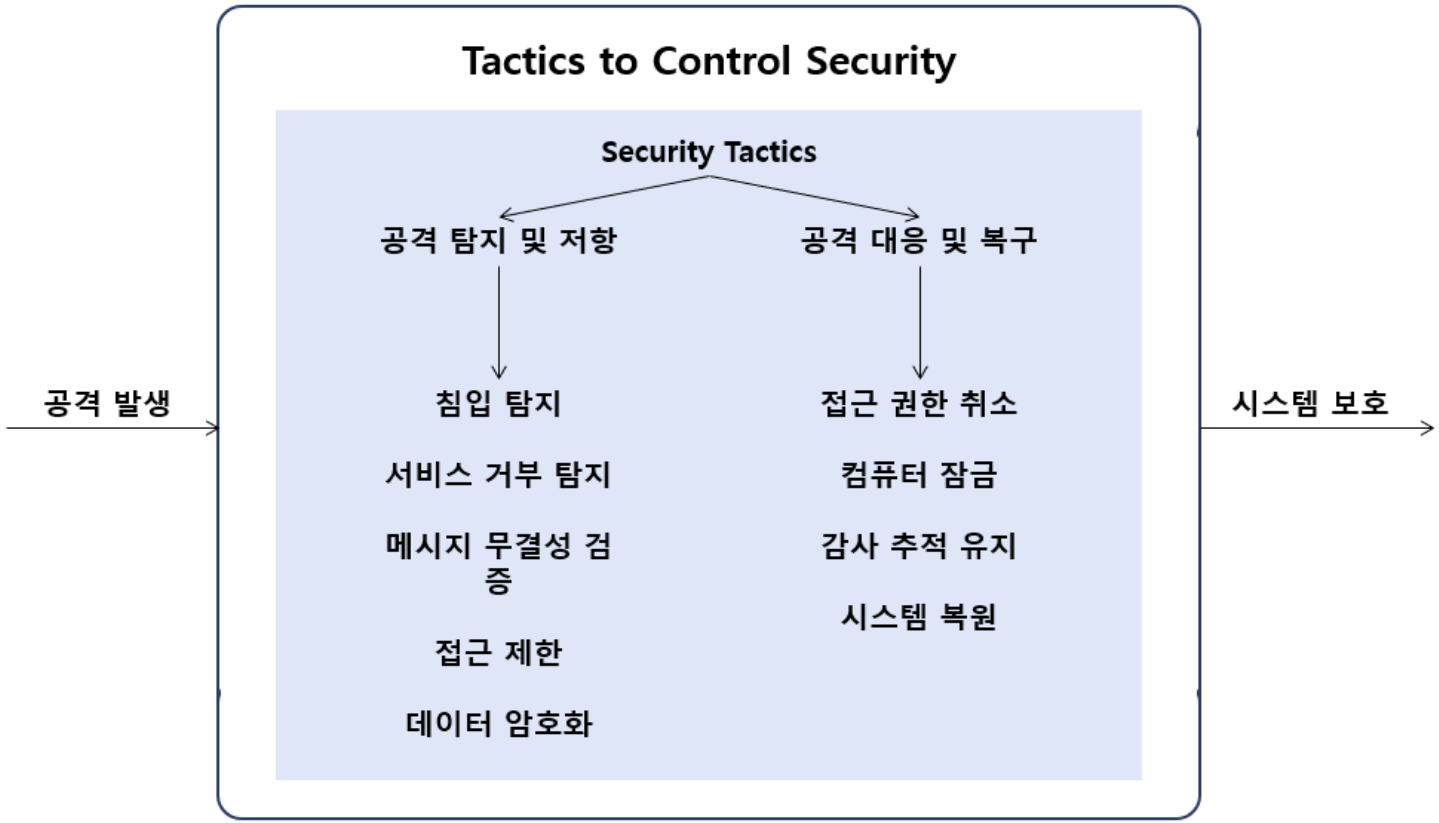
Tactics
Maintainability

- 시스템 고장 시 유지 보수에 특화된 설계
- 고장 시 보수 및 유지를 도와주어 효과적인 관리 가능
- 예외 상황에서 매뉴얼 등이 없어 취약하며 추가적인 조치를 요하여 시간을 더 많이 사용함

Tactics
Availability

- 장애 발생 시 시스템의 지속적인 운영을 보장하는 데 중점
- 장애 발생 시 신속한 탐지와 복구로 시스템의 가용성을 높이고 사용자 서비스 연속성을 보장
- 장애 예방 및 복구를 위한 추가적인 자원과 인프라 요구 사항이 존재하고, 시스템 복잡성이 증가

Tactics
security

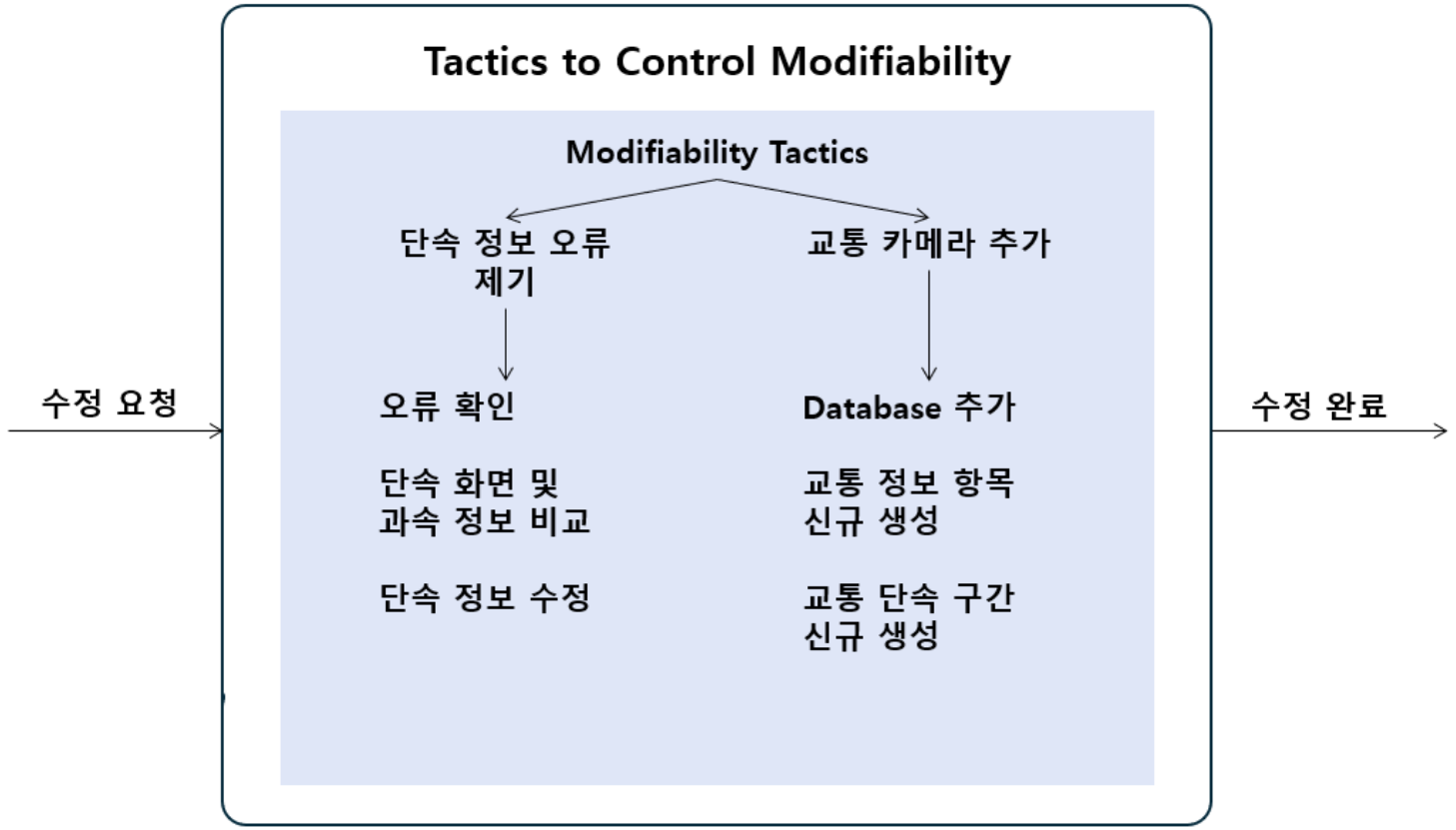


- 시스템을 외부의 공격으로부터 보호하고, 적절하게 대응하여 시스템을 안전하게 유지
- 다양한 공격 유형에 대한 탐지 및 대응 방식을 통해 시스템의 전반적인 보안성을 강화할 수 있음
- 보안을 강화하기 위한 추가적인 자원 소모 및 운영 복잡성이 증가

Tactics
Performance

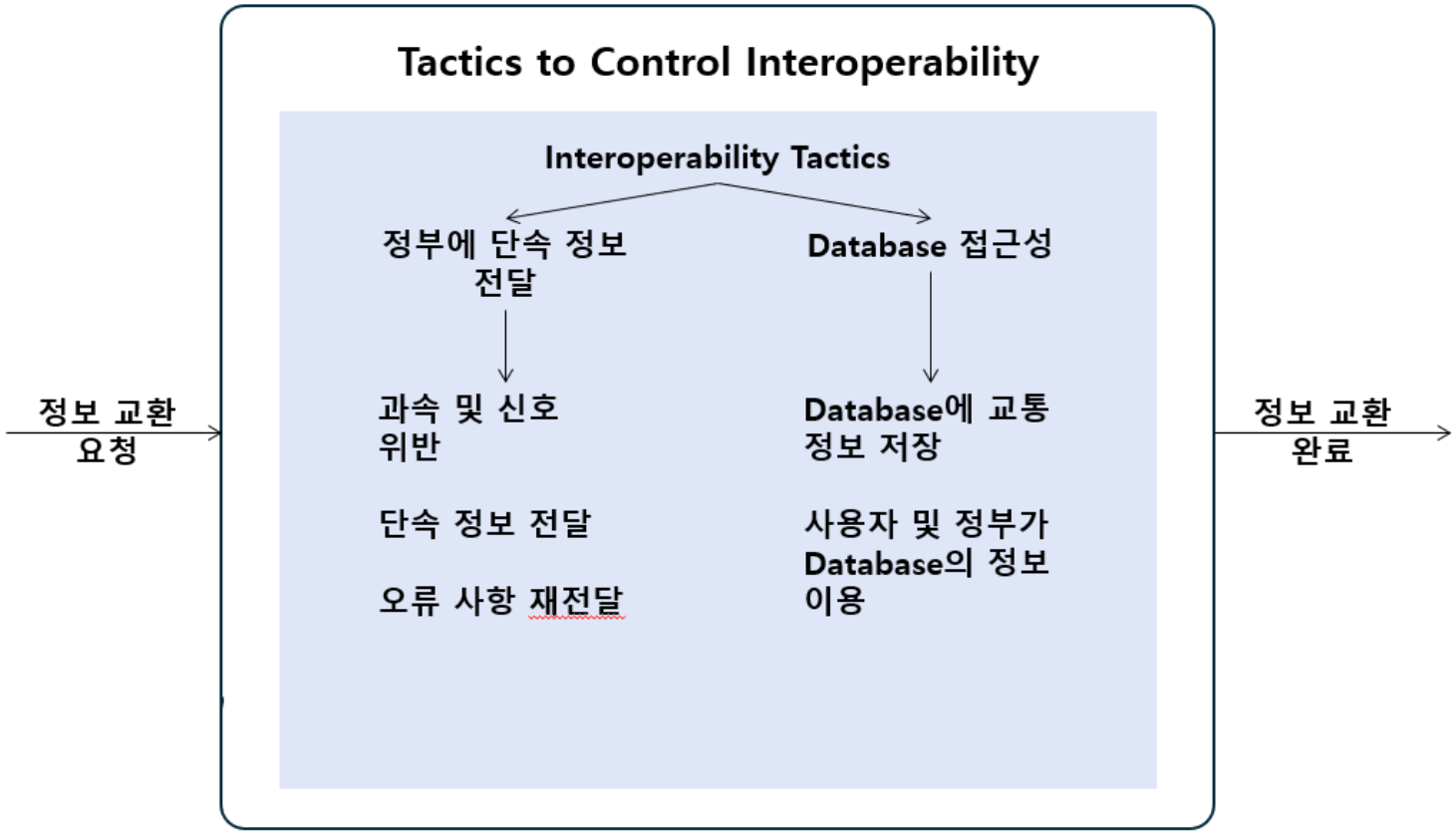
- 시스템의 성능을 최대화할 수 있는 설계
- 정확도 및 전달 시간에서 이익을 취할 수 있음
- 비교 및 업그레이드에 소모되는 추가 비용

Tactics
Modifiability



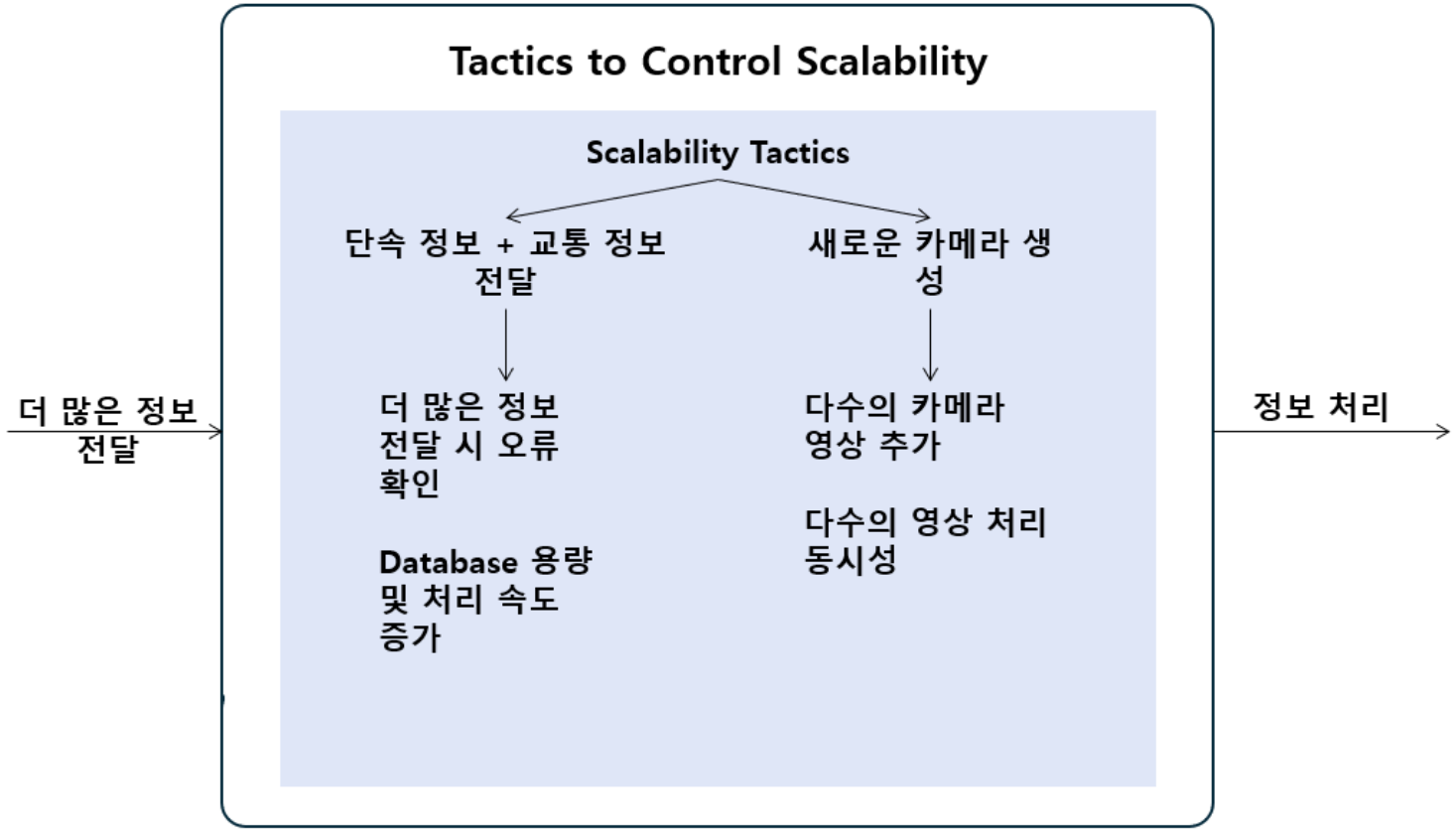
- 시스템의 수정이 용이한 설계
- **SW**는 지속적으로 진화하고 변화하기 때문에 대응이 쉬워짐
- 모듈의 크기가 크다면, 수정 비용이 커질 수 있음

Tactics Interoperability



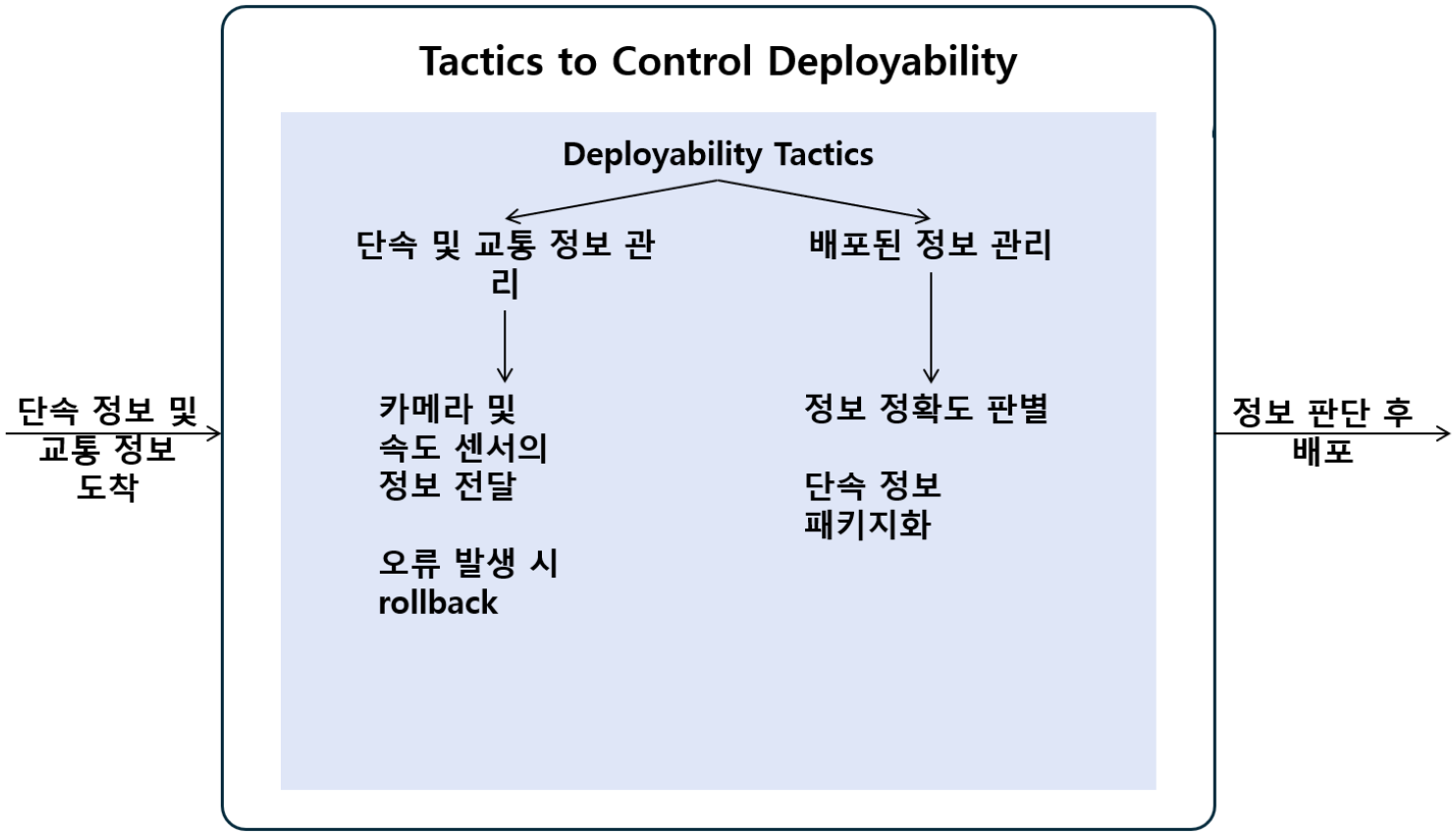
- 다른 디바이스나 모듈의 연결이 용이한 설계
- 시스템 간의 정보 교환이 정확해짐

Tactics
Scalability



- 증가하는 요청이나 정보에 따른 설계
- 더 많은 데이터, 사용자, 트래픽 등을 처리할 수 있도록 효과적으로 확장 가능해짐

Tactics
Deployability



- 시스템의 효율적이고 신속한 배포를 위한 설계
- 새로운 버전이나 정보를 빠르고 안정적으로 배포 가능