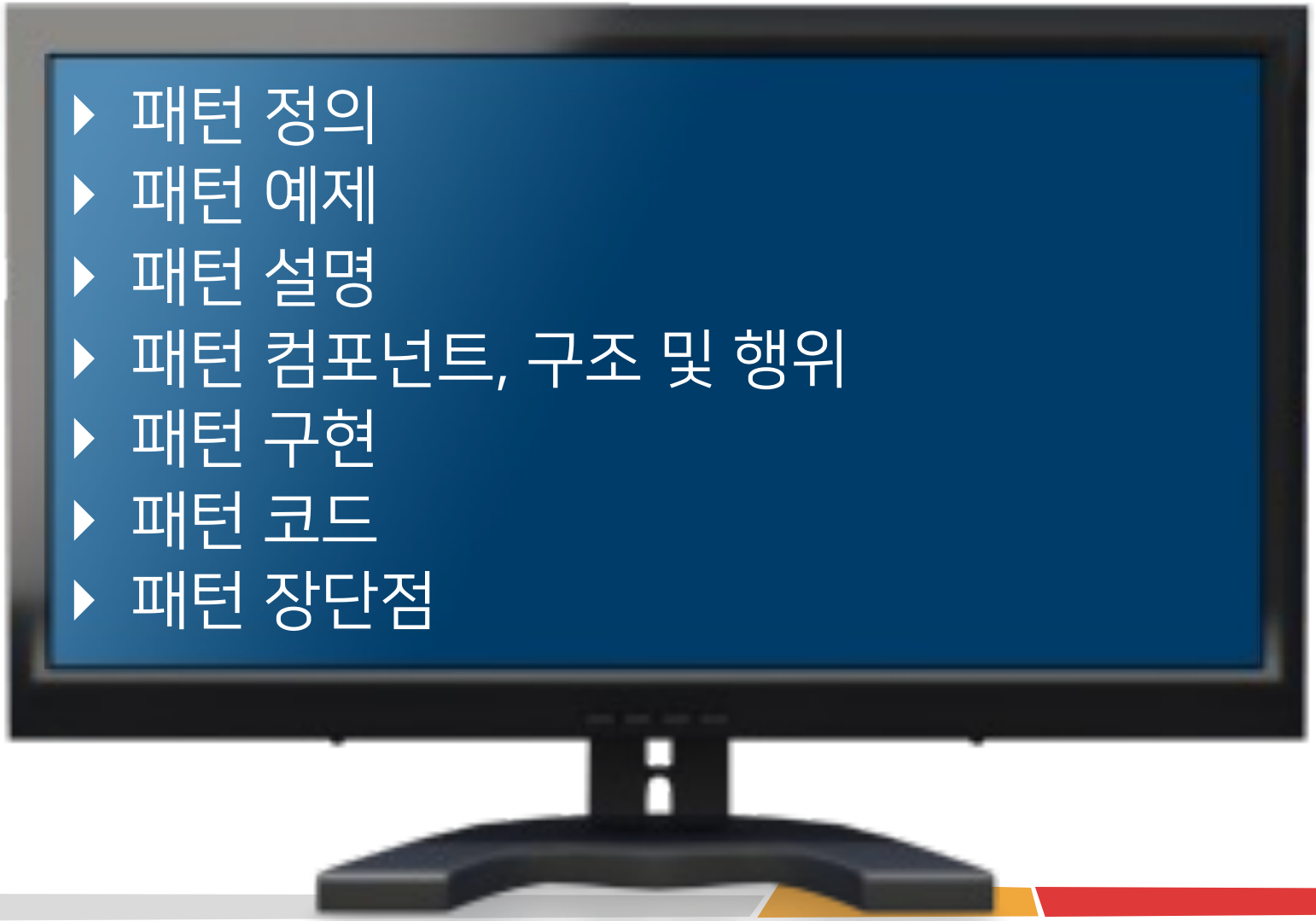
Spring 2023

소프트웨어 아키텍처 패턴: Master-Slave

Seonah Lee

Gyeongsang National University

Master-Slave 패턴



Master-Slave Pattern: Definition

- 정의
  - **Master** 컴포넌트는 자신과 동등한 역할을 하는 **Slave** 컴포넌트에 작업을 분산하고, 이 **Slave** 컴포넌트들에서 반환된 결과들로부터 최종 결과를 계산하는 패턴
- 예제
  - 행렬 곱셈
  - 이미지 변환 코드
  - 두 신호의 상관 계산
  - 병렬 계산을 위한 프로세스 제어
  - 분산 서비스

Master-Slave Pattern: Example

- 순회판매원 문제(**Traveling Salesman Problem**)
  - 주어진 위치들 간에 최적화된 순회 경로를 찾음(각 위치 한번만 방문)
  - 경로들을 비교해서 그 중에 가장 최적의 근사한 해법을 찾아 나감

Master-Slave Pattern: Description

- 정황(**Context**)
  - 작업을 의미적으로 동일한 여러 서브태스크들로 분할
- 문제(**Problem**)
  - 분할-정복(**Divide-and-Conquer**) 원칙에 따라 독립적으로 처리하는 하위 작업으로 작업 분할
  - 각기 분할된 프로세스에 의해 제공된 결과에서 전체 계산 결과 산출

Master-Slave Pattern: Description

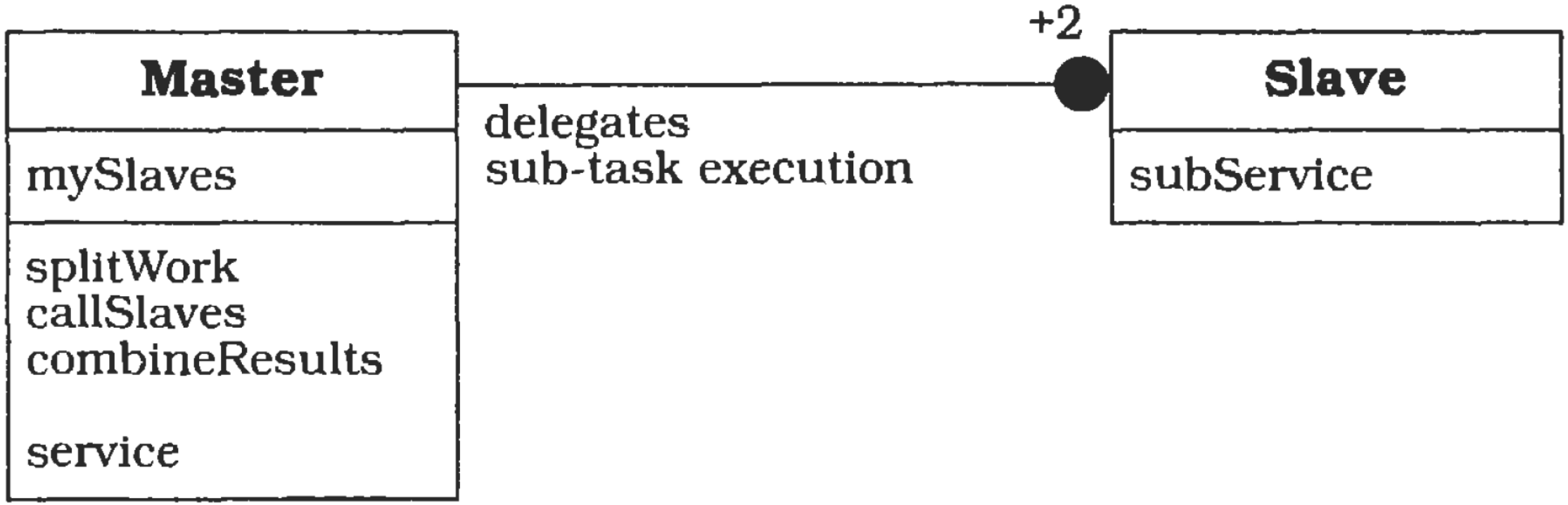
- 해법 **(Solution)**
  - 개별 서브태스크들의 프로세싱 간에 조정**(Coordination)** 인스턴스 도입
  - 작업 서비스 대상자는 작업이 분할-정복 원칙에 근거한다는 사실을 알 필요가 없음
  - 하위 작업 처리는 작업을 분할하고 최종 결과를 모으는 알고리즘에 좌우되어서는 안됨
  - 하위 작업 처리는 서로 다르지만 의미적으로 동일한 구현을 사용하는 것 권장
  - 하위 작업 처리에서 간혹 조정**(Coordination)**이 필요함

Master-Slave Pattern: Components

Master
• 여러 슬레이브 컴포넌트들에게 작업을 분할. (모든 Slave들에 하나의 동일한 인터페이스 제공) • (독립적이지만 의미적으로 동일한) 슬레이브들의 실행을 개시 • 슬레이브가 반환한 결과를 계산

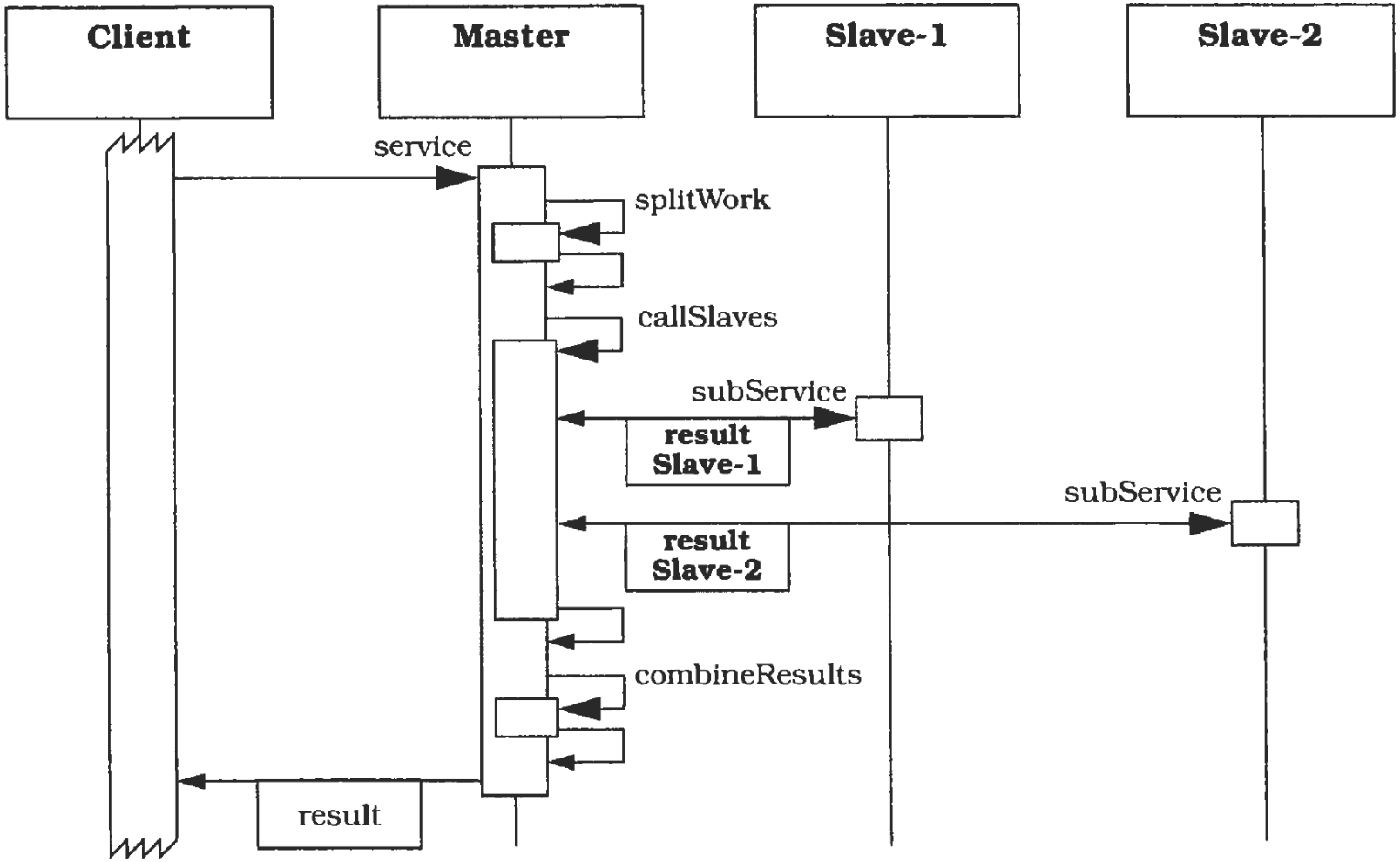
Slave
• 마스터를 대신해 작업할 서브 서비스를 구현

Master-Slave Pattern: Structure



Master-Slave Pattern: Behavior

- Slave-1과 Slave-2를 호출



Master-Slave Pattern: Realization

- 구현 순서
  1. 작업을 분할
     - 어떻게 작업을 동등한 하위작업으로 나눌 수 있는지 정의
  2. 하위작업의 결과를 조합
     - 조합하여 어떻게 최종 결과를 계산할 것인지 정의
  3. **Master**와 **Slave**간의 협력을 정의
     - **Slave**의 인터페이스 정의:
       - 슬레이브의 인터페이스에 하위작업을 매개변수로 포함시킴
       - 혹은 마스터가 하위작업을 저장하고 슬레이브가 저장된 하위작업을 가져올 **Repository**를 정의
     - **Master**는 이 인터페이스를 사용해 개별 하위 작업의 처리를 위임

Master-Slave Pattern: Realization

- 구현 순서
  4. 앞 단계의 명세에 따라 **Slave** 컴포넌트 구현
  5. 앞 단계들의 명세에 따라 **Master** 컴포넌트 구현
     - 작업을 일정한 개수의 하위 작업들로 나눔
     - 슬레이브에게 작업 전체의 실행을 위임
     - 필요한 개수만큼, 혹은 가능한 개수 만큼의 하위 작업을 나눔
       - 예: 지원 가능한 프로세스들의 개수

Master-Slave Pattern: Application

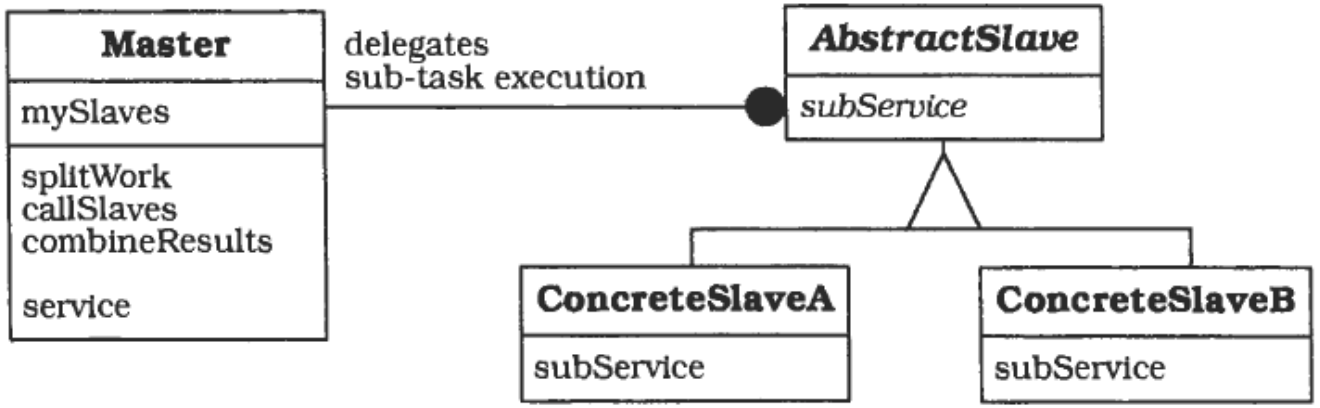
- 장애 허용성(**Fault tolerance**)
  - 서비스의 실행을 여러 개의 복제된 구현들에 위임
  - 분할된 서비스의 실행들에 장애가 발생하는지 탐지하고 처리
- 응용
  - **Master**는 정해진 개수의 복제된 **Slave**에 서비스 실행을 단순 위임
  - 최소한 하나의 **Slave**는 장애가 발생하지 않고 항상 제대로 동작하여 클라이언트에 유효한 결과를 제공할 수 있는 상황 제공
    - 예외를 일으키거나 클라이언트가 처리할 특수한 예외값 반환
  - **Master**는 **Slave** 장애를 검사하기 위해 타임아웃(**time-out**) 방식을 채택
  - 이 경우, **Master** 자체에 장애가 발생하는 상황에는 대처하지 못함

Master-Slave Pattern: Application

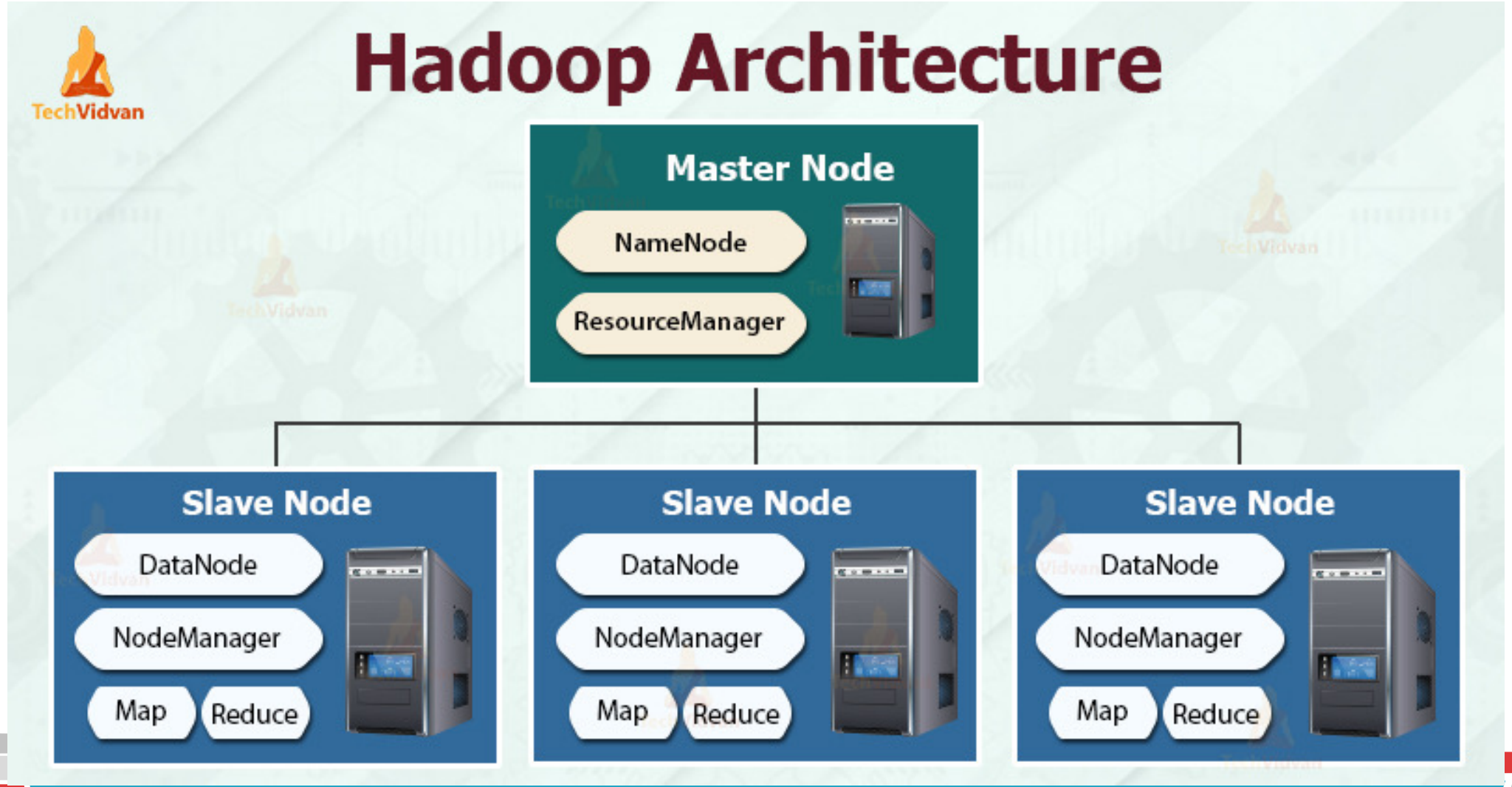
- 병렬 컴퓨팅(**Parallel computing**)
  - 복잡한 **Task**를 병렬로 실행되는 일정 개수의 동일한 **Subtask**들로 분할
  - 이 **Subtask**들이 처리해 얻게 되는 결과들을 활용해 최종 결과를 얻음
- 응용
  - **Master**는 최종 결과를 계산하기 전에 모든 **Subtask** 실행 완료를 대기
  - 혹은 **Slave**가 자체 결과를 종료할 때마다 계산 실행도 가능
  - 해당 응용은 하드웨어 아키텍처에 강하게 좌우
  - 또한 머신의 위상 구조와 프로세서 상호연결의 속도도 고려

Master-Slave Pattern: Application

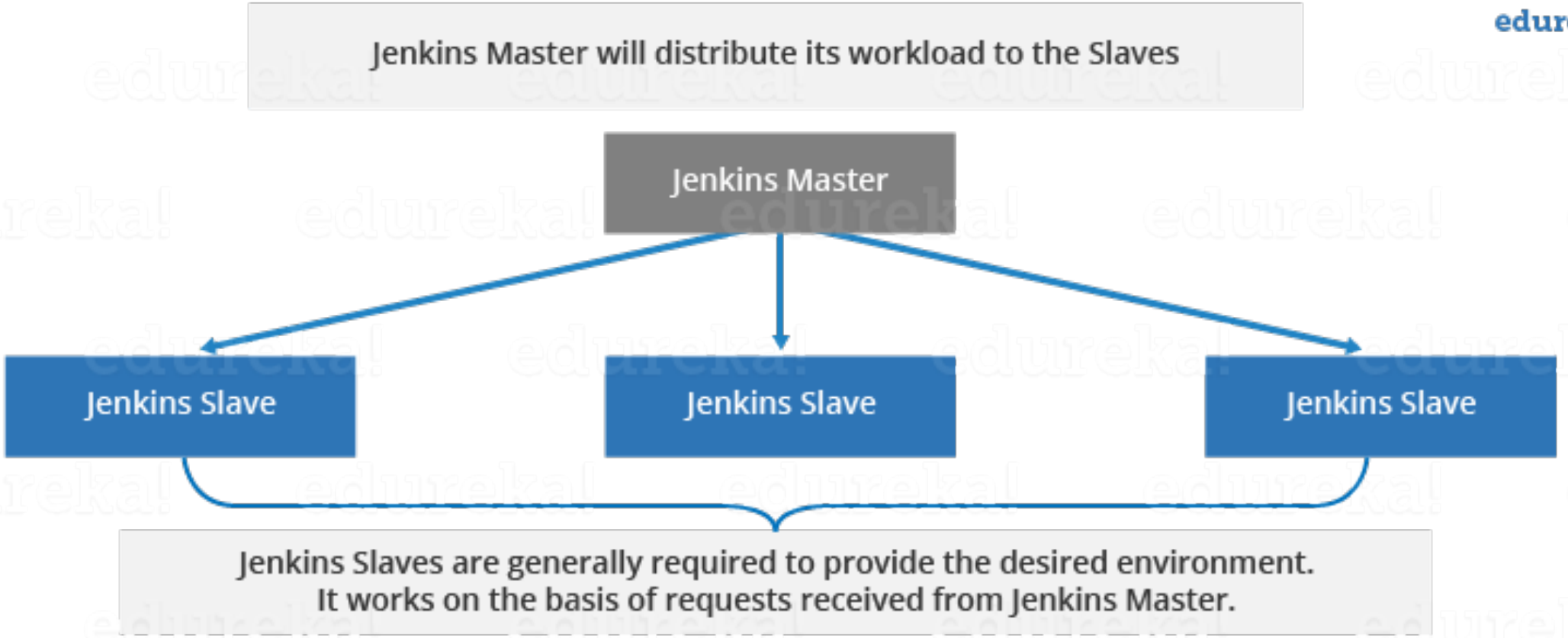
- 계산 정확도(**Computational Accuracy**)
  - 서비스 실행을 각기 다른 몇 개의 구현들에 위임
  - 정확하지 못한 결과들을 탐지하고 처리
- 응용
  - 서비스 실행은 최소한 세 개의 서로 다른 **Slave**에 위임
  - **Master**는 모든 **Slave**가 작업을 마칠 때까지 기다려 결과를 판별



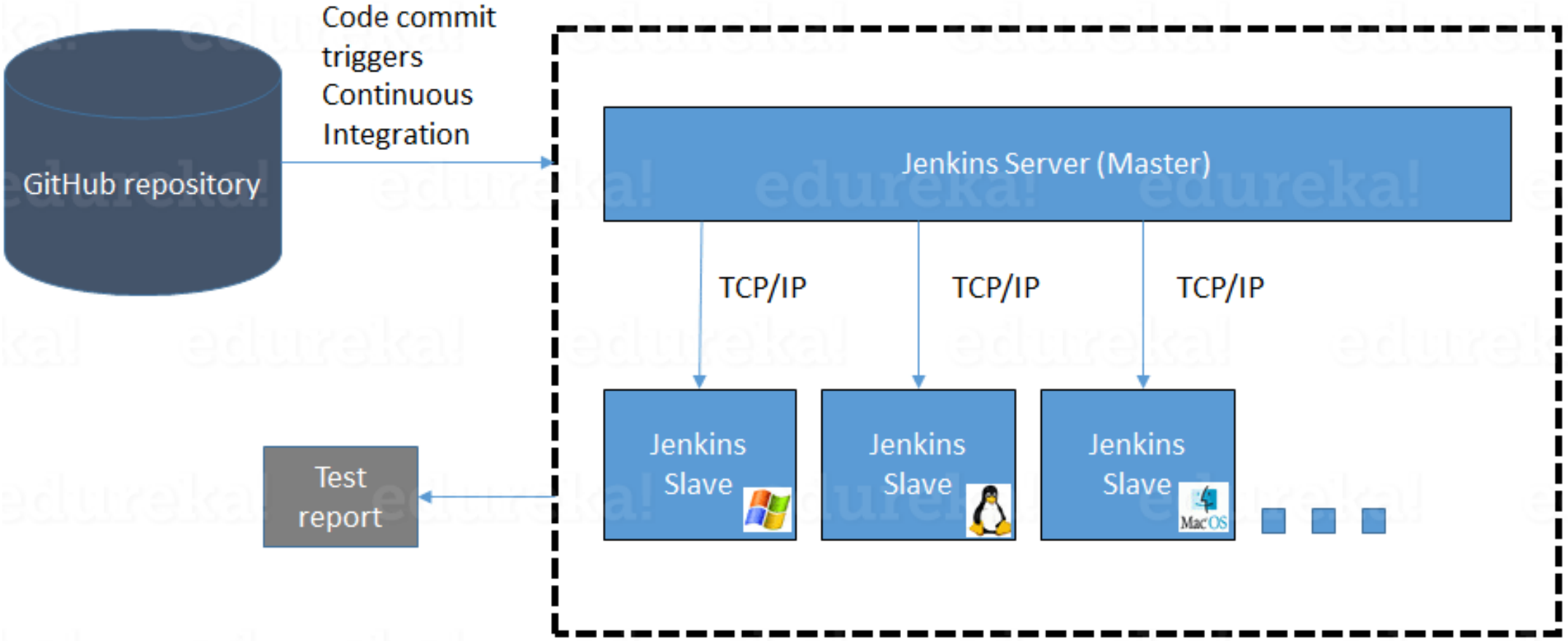
Master-Slave Pattern: Case Studies



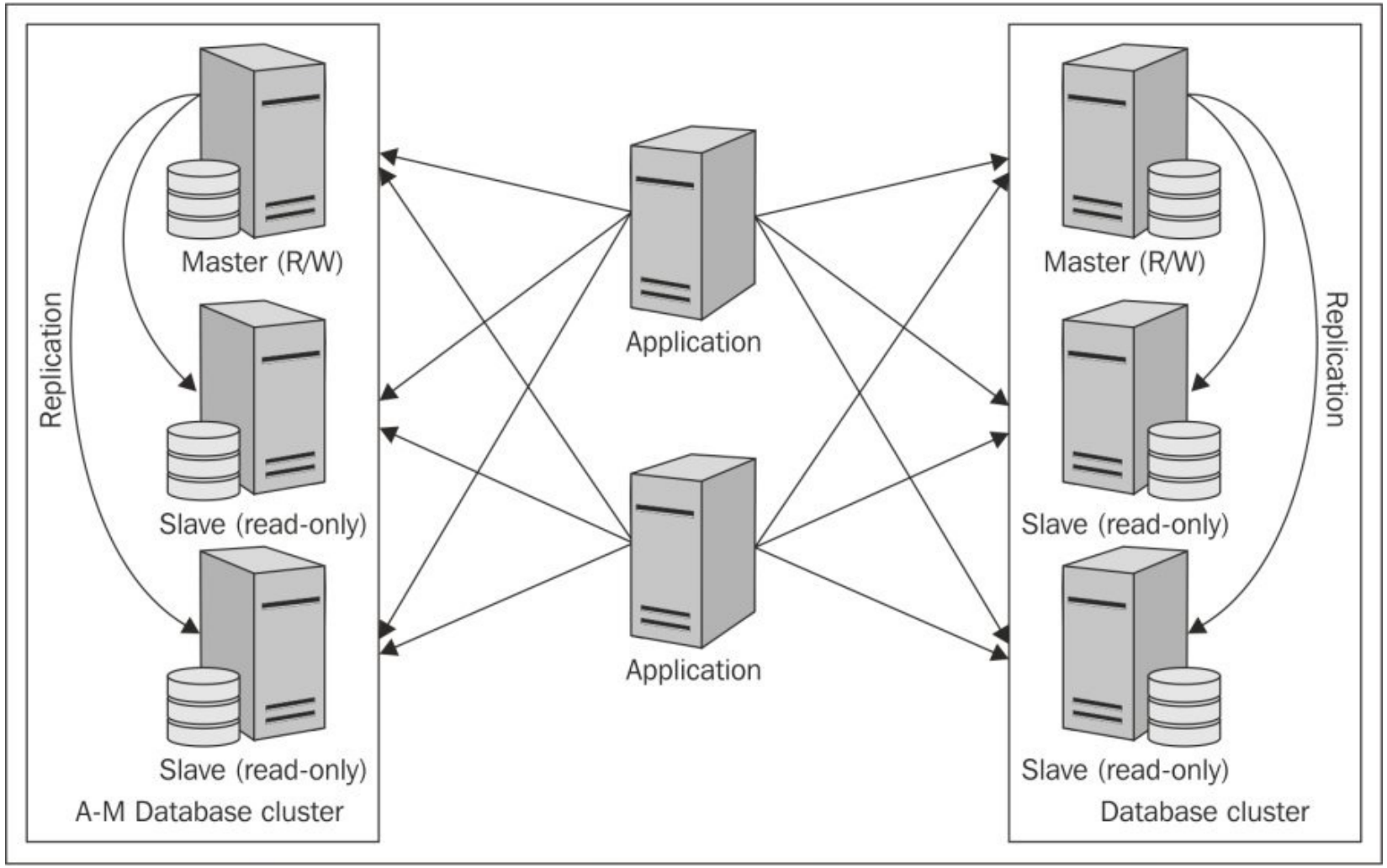
Master-Slave Pattern: Case Studies



Master-Slave Pattern: Case Studies



Master-Slave Pattern: Case Studies



Master-Slave Pattern: Benefits

- 교환가능성(**Exchangeability**)와 확장성(**Extensibility**)을 도입
  - 추상 **Slave**를 도입하면, **Slave** 대체 용이
- 관심사항(**Concerns**)의 분리가 가능
  - **Master**를 도입하며, **Slave** 및 클라이언트 코드 분리
- 효율이 향상
  - 병렬 계산을 신중히 구현하면, 특정 계산에서의 성능 향상 가능

Master-Slave Pattern: Liabilities

- 실행가능성(**Feasibility**)이 떨어짐
  - **Master**의 작업 분할, 할당, 대기, 결과 계산 등을 위한 처리 시간과 공간 필요
- 머신 종속성의 제약을 받음
  - 병렬 계산을 위해서는 하드웨어에 종속, 변경용이성과 이식성 하락
- 구현이 어려움
  - 작업 분할, **Master**와 **Slave**협력, 병렬 진행, 오류 처리 등을 주의 깊게 고려 필요
- 이식성이 떨어짐
  - 하드웨어에 종속된다는 것은 다른 하드웨어 이동이 어려움, 특히 병렬 계산!

Question?

Seonah Lee
saleese@gmail.com