Master-Slave 패턴

아꿈사 http://cafe.naver.com/architect1
TTF http://www.npteam.net

Master-Slave 패턴 컴포넌트

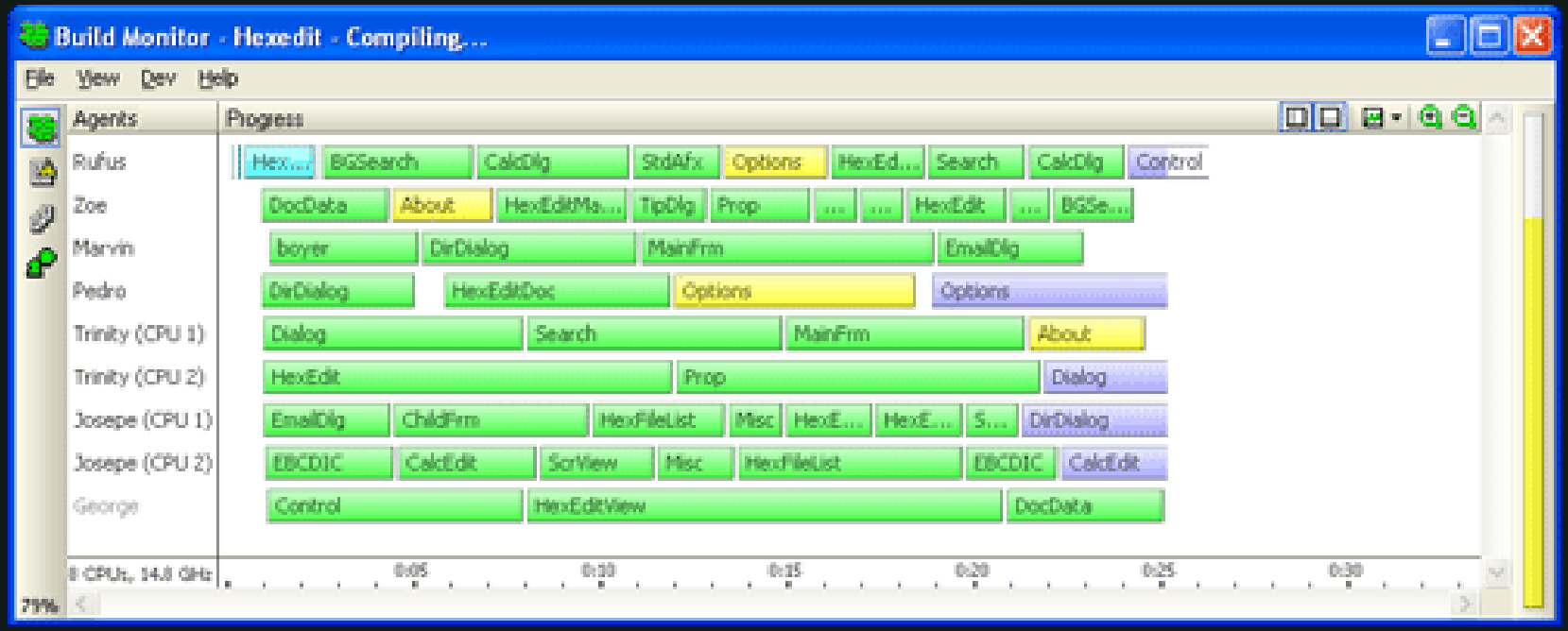
Master Component
  - 마스터 컴포넌트

Slave Component
  - 슬레이브 컴포넌트

마스터 컴포넌트는 자신과 동등한 역할을 하는
슬레이브 컴포넌트에 작업을 분산하고,
슬레이브에서 반환된 결과로부터 최종 결과를 계산한다.

Master-Slave Pattern - Context

- 정황(Context)
  - 작업을 의미적으로 동일한 여러 서브태스크들로 분할한다.



이미지 출처 : http://www.xoreax.com/slow_build_time_challenge.htm

Master-Slave Pattern - Problem

- 문제(Problem)
  - 분할 정복은 문제들을 해결하기 위한 공통원칙.
  - 독립적으로 처리되는 여러 개의 동등한 서브태스크들로 작업을 분할한다.

이미지 출처 : http://withman.net/326

Master-Slave Pattern - Problem

- 문제(Problem)
  - 분할된 프로세스들에 의해 제공된 결과들로부터 전체 계산 결과가 산출된다.

이미지 출처 : http://eureka0000.tistory.com/entry/%EB%9D%BC%EC%8A%A4%EA%B8%B8%EB%A1%9C

Master-Slave Pattern – Force

- 문제에 내포된 영향력(Force) - 01

  - 클라이언트는 계산이 분할 정복 원칙에 근거하고 있다는 사실을 알 필요가 없다.
    → 결과를 Master 컴포넌트에 넘기기

  - 클라이언트나 서브태스크의 처리는 작업을 분할하고 최종 결과를 모으는 알고리즘에 좌우되어서는 안 된다.
    → Master 컴포넌트에서 해야할 일!

Master-Slave Pattern – Force

- 문제에 내포된 영향력(Force) - 02

  - 서브태스크들을 처리하기 위해서는
    서로 다르지만 의미적으로 동일한 구현들을
    사용하는 것이 좋다.

  - 서브태스크들을 처리할 때 간혹 조정이
    필요하기도 하다.
    → 유한 요소법을 사용하는 모의 실험
    애플리케이션의 경우가 그렇다.

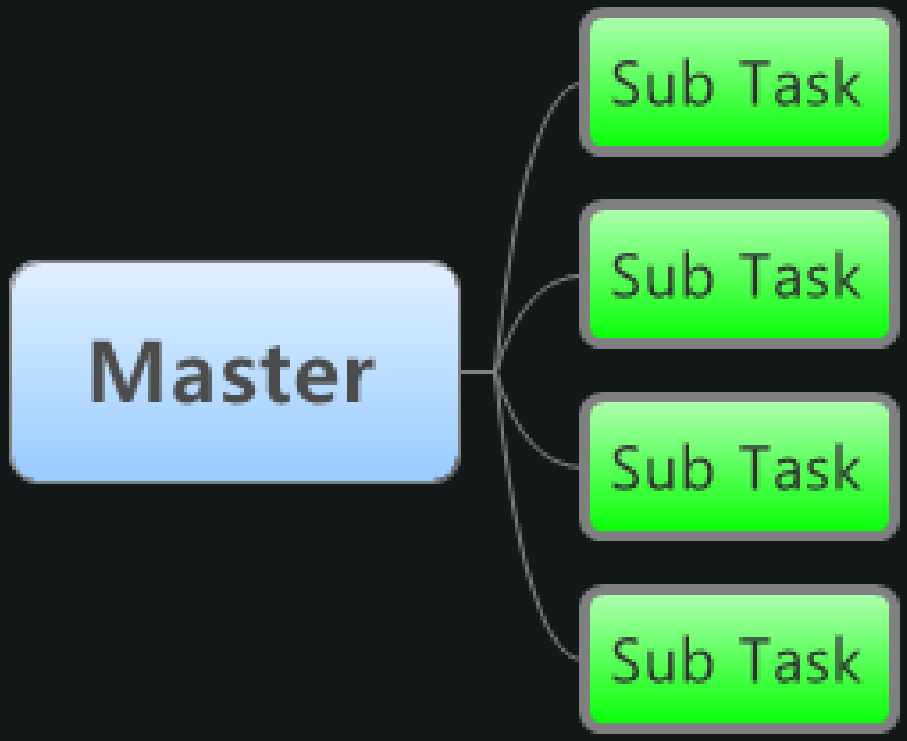
Master-Slave Pattern – Solution

- 해법(Solution)
  - 서비스의 클라이언트들과 개별 서브태스크들의 프로세싱 간에 조정(coordination) 인스턴스들을 도입한다.

이미지 출처 : MBC 손석희 100분 토론

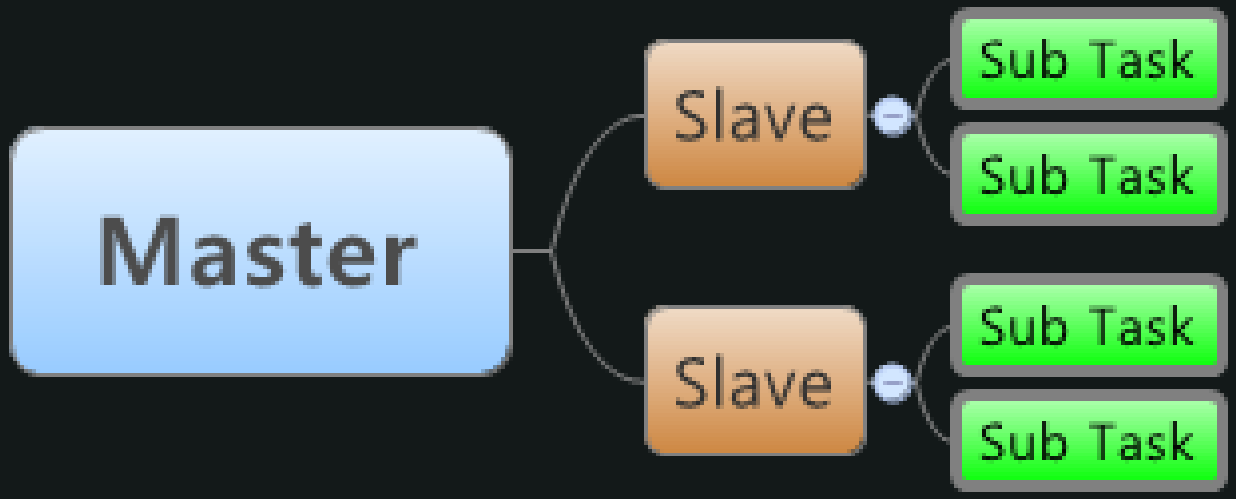
Master-Slave Pattern – Solution

- 해법(Solution)
  - 마스터 컴포넌트는 여러 개의 동일한 서브태스크들로 작업을 분할하고,



Master-Slave Pattern – Solution

- 해법(Solution)
  - 서브태스크를 여러 개의 슬레이브 컴포넌트에 위임한 다음, 각 슬레이브가 반환하는 결과로부터 최종 결과를 계산한다.



Master-Slave Pattern – Solution

- 응용 분야

  - 장애 허용성
    → 서비스에 장애 발생시 복제된 다른 서비스가 대체적으로 작동한다.

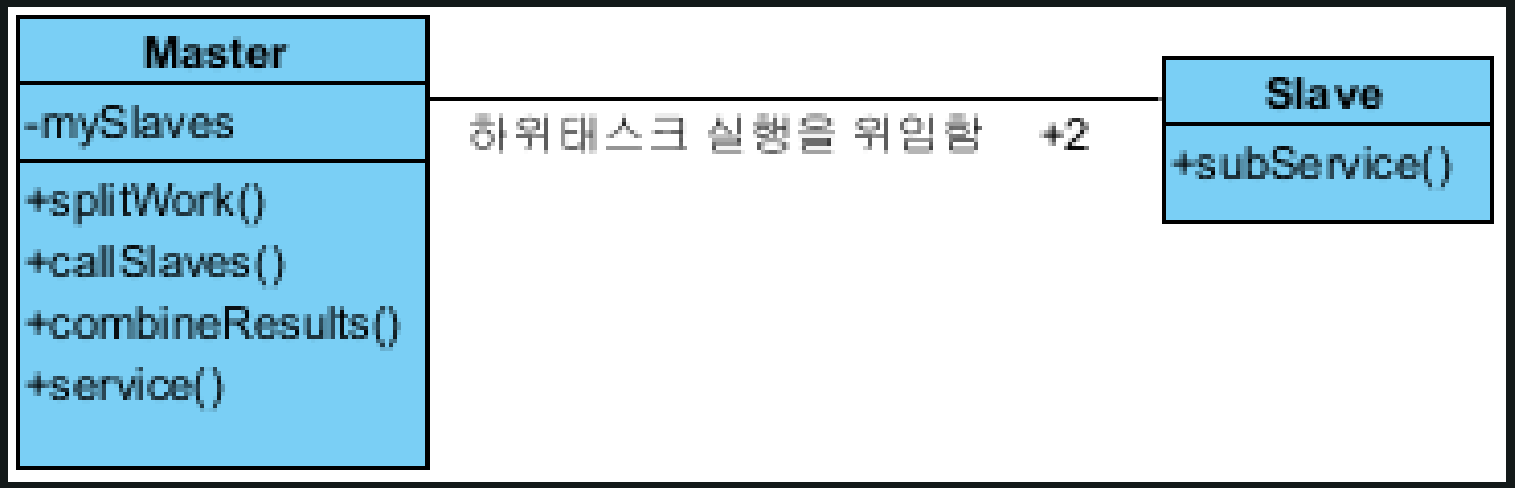
  - 병렬 컴퓨팅
    → 병렬로 실행되는 서브태스크로 나누어서 실행하고 결과를 종합하여 처리한다.

  - 계산 정확도
    → 서비스를 각기 다른 구현들에게 위임한다.

Master-Slave Pattern – Structure

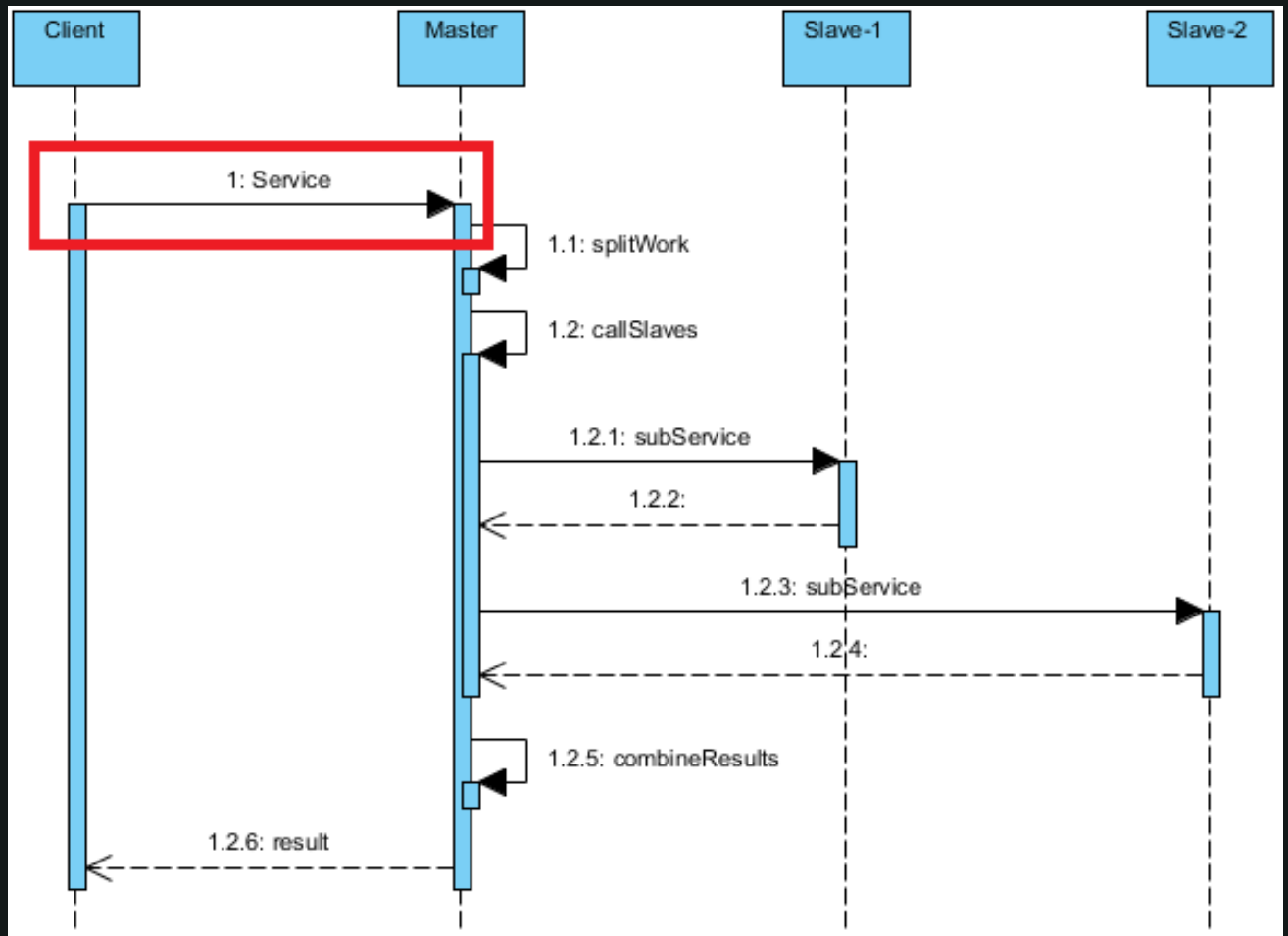
- 마스터 컴포넌트
  - 클라이언트가 서비스에 접근할 수 있는 인터페이스 제공

- 슬레이브 컴포넌트
  - 마스터에 정의된 서브태스크들을 처리할 수 있는 서브서비스를 제공한다.



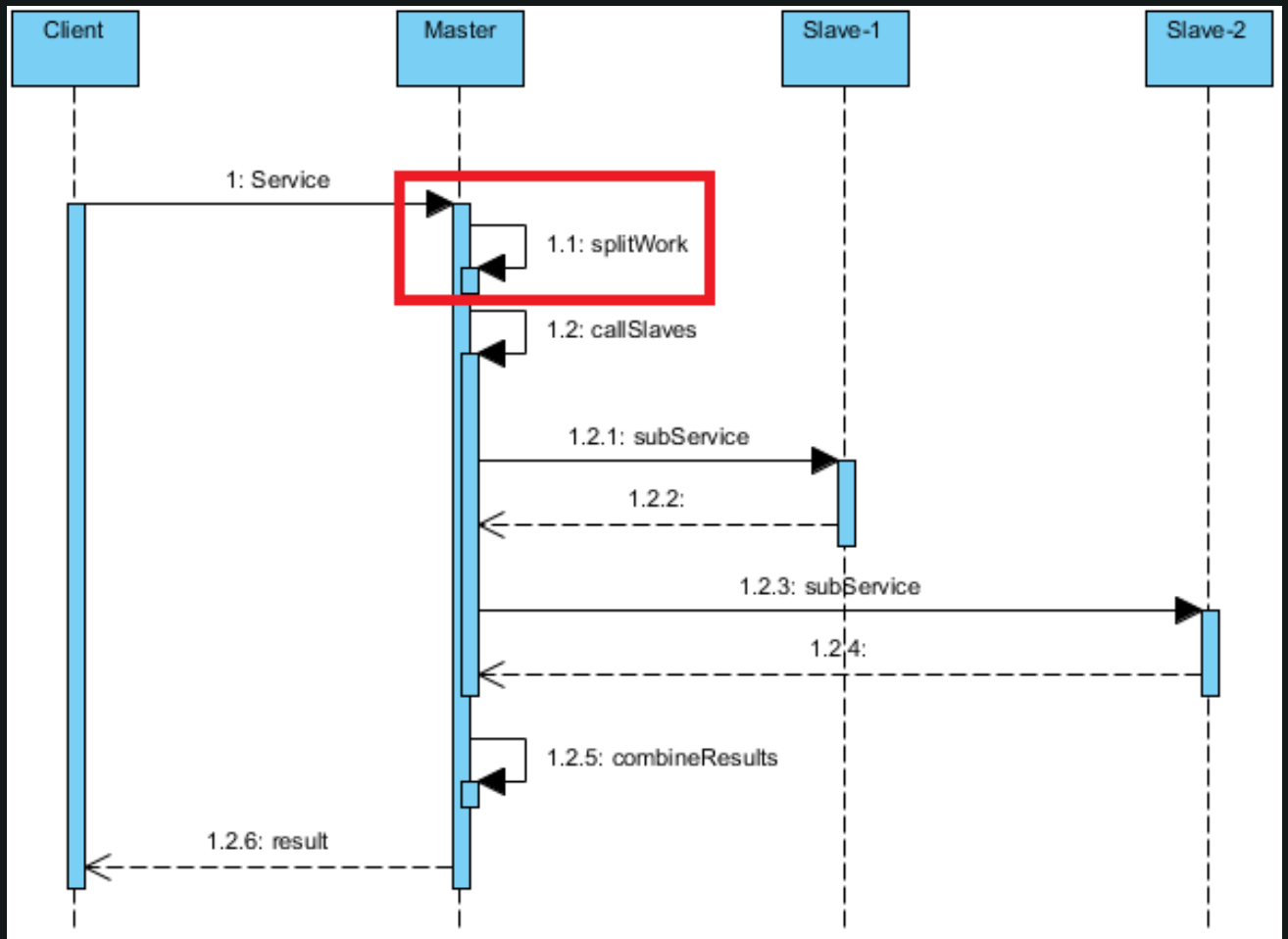
Master-Slave Pattern – Dynamic

- Master-Slave 동작 묘사
  - 클라이언트가 마스터에게 서비스를 요청한다.



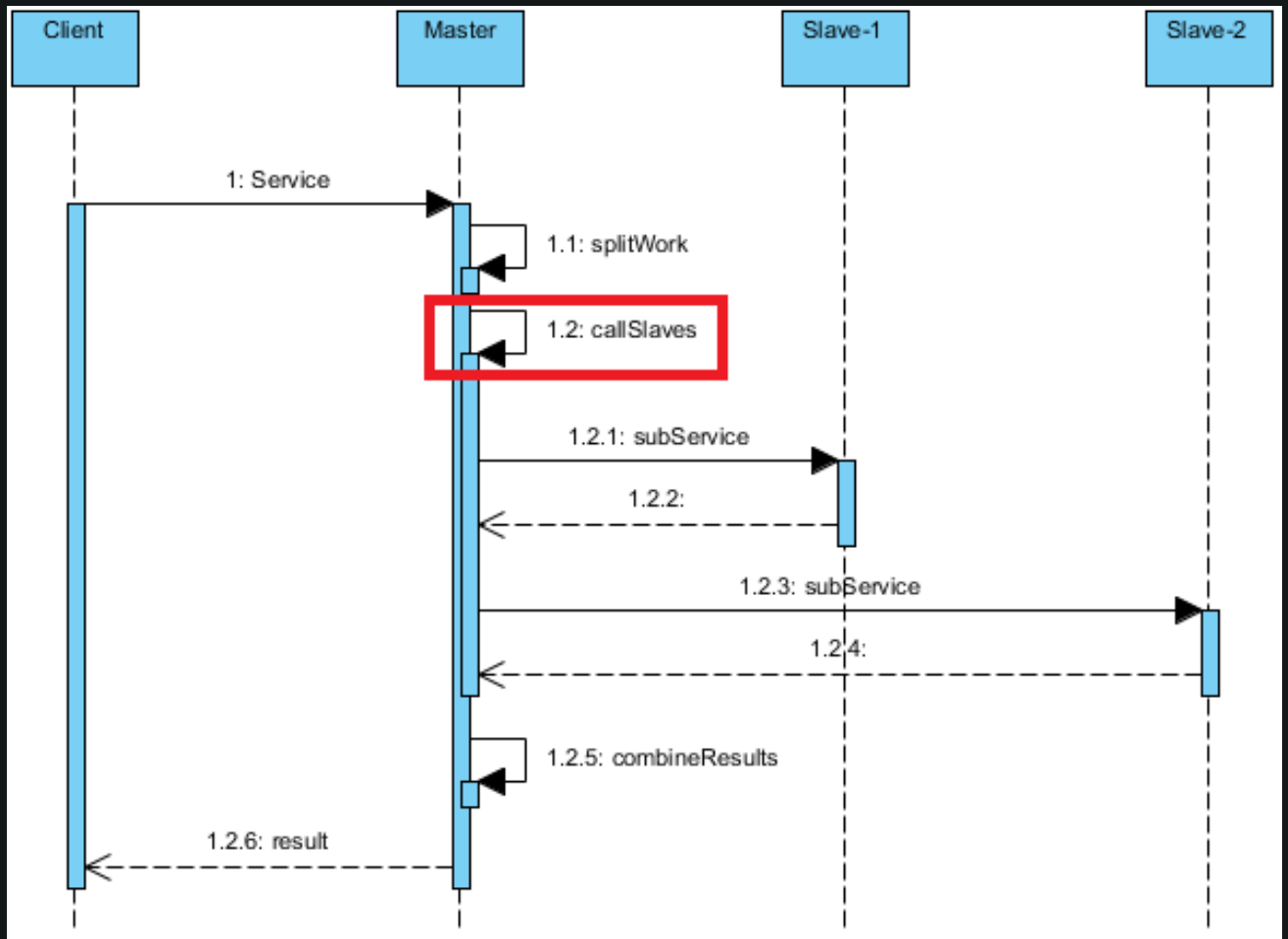
Master-Slave Pattern – Dynamic

- Master-Slave 동작 묘사
  - 마스터는 동등한 서브태스크들 몇 개로 태스크를 분할한다.



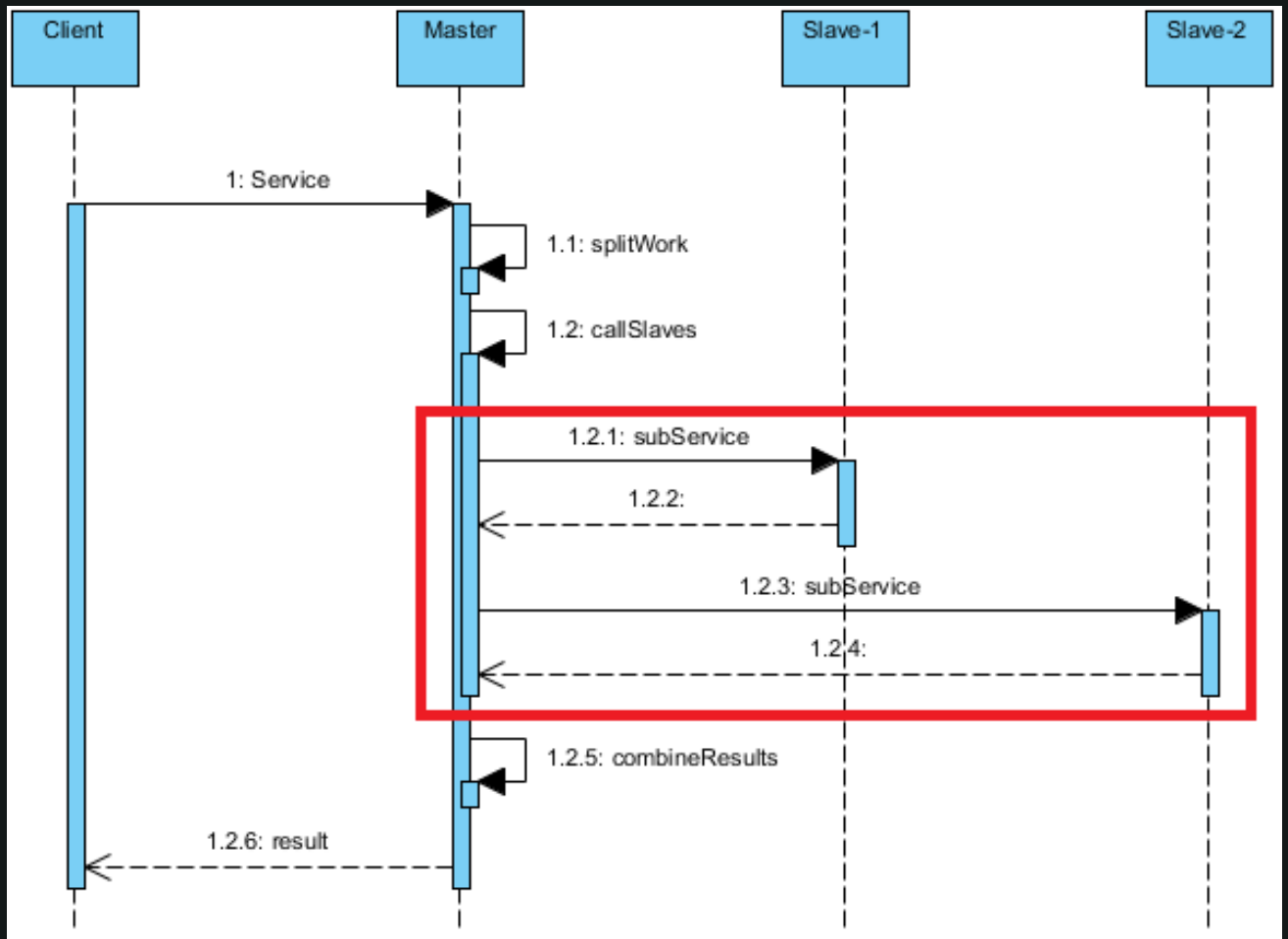
Master-Slave Pattern – Dynamic

- Master-Slave 동작 묘사
  - 마스터는 이 서브태스크들의 실행을 몇 개의 슬레이브 인스턴스들에 위임하고 실행을 시작한다.



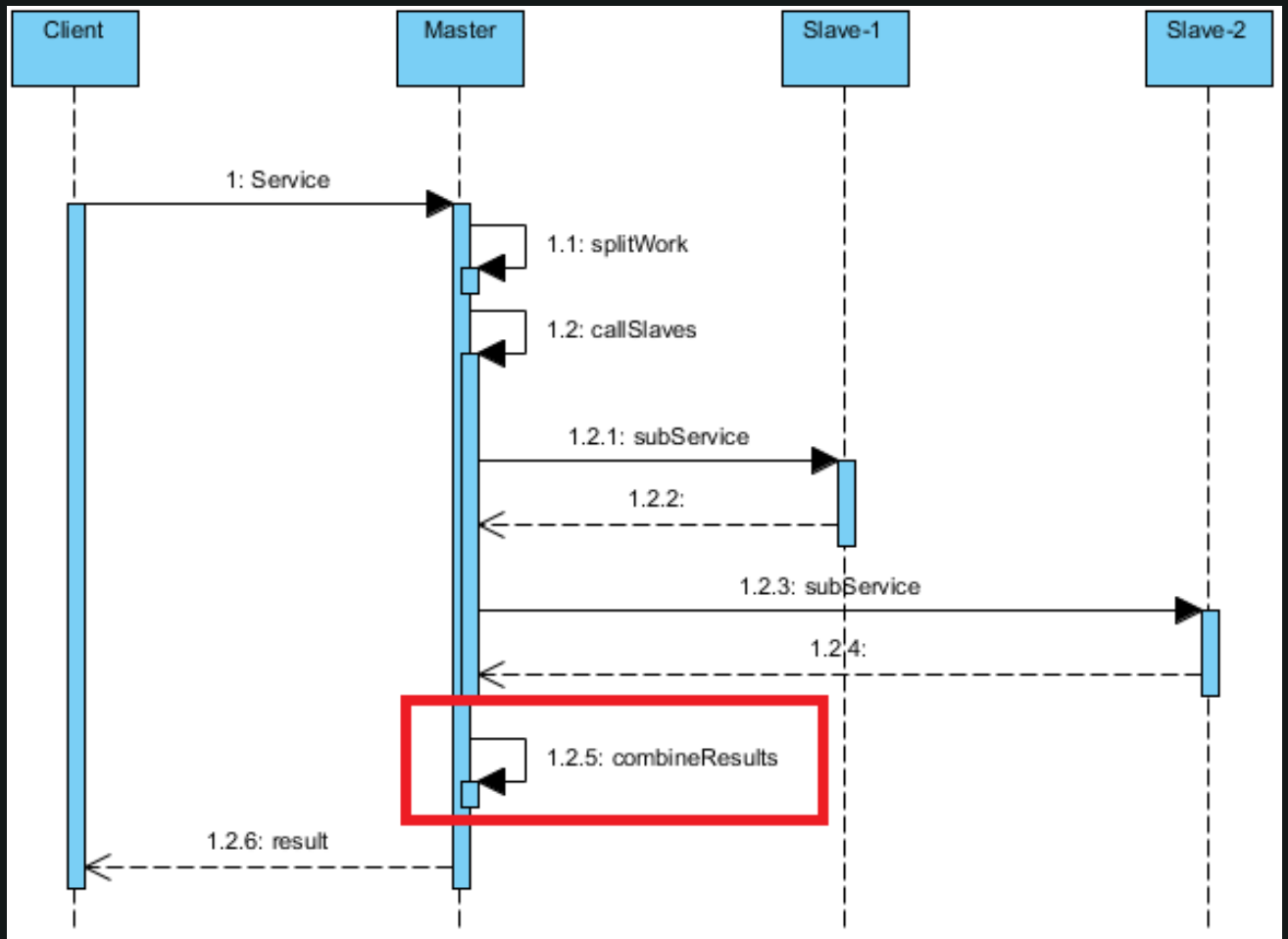
Master-Slave Pattern – Dynamic

- Master-Slave 동작 묘사
  - 슬레이브들은 각 서브태스크들의 프로세싱을 수행하고, 그 결과를 마스터에 반환한다.



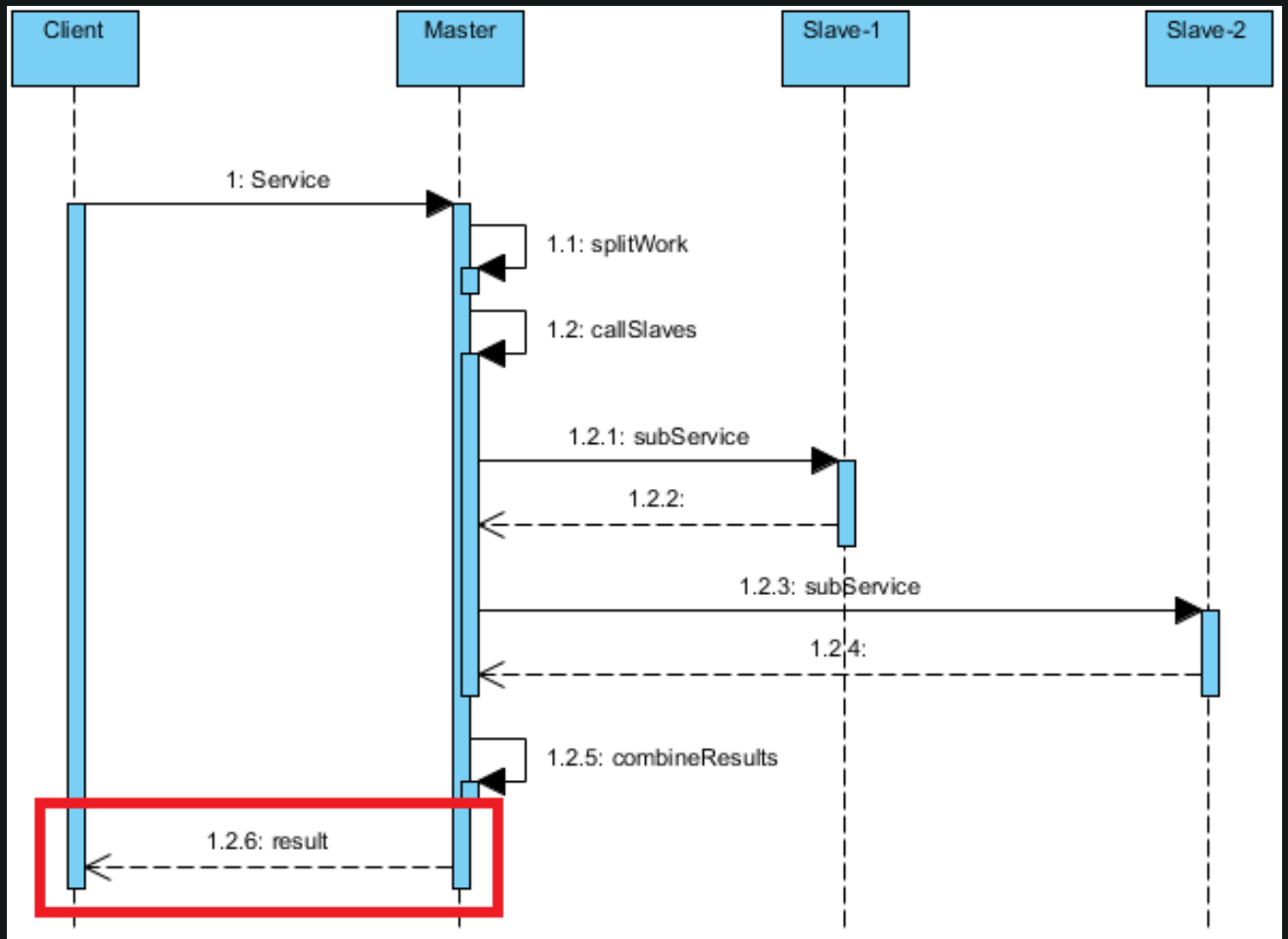
Master-Slave Pattern – Dynamic

- Master-Slave 동작 묘사
  - 마스터는 슬레이브들로 부터 받은 분할된 결과들을 모아 전체 태스크에 대한 최종 결과를 계산한다.



Master-Slave Pattern – Dynamic

- Master-Slave 동작 묘사
  - 마스터가 클라이언트에 최종 결과를 반환한다.



Master-Slave Pattern – Implementation

- 구현(Implementation)

  - **1단계** 작업을 분할한다.

    → 서브태스크로 나누는 방법을 정의한다.
    → 서브태스크 하나에서 처리될 서브서비스를 파악한다.

  - **2단계** 서브태스크의 결과들을 조합한다.

    → 개별 서브태스크들을 처리해서 얻은 결과들을 통하여 전체 서비스의 결과를 어떻게 계산할지 정의한다.

Master-Slave Pattern – Implementation

■ 구현(Implementation)

- **3단계** 마스터와 슬레이브 간의 협력을 정의한다.

  → 1단계에서 파악한 서브서비스 인터페이스를 정의한다.
  → 이 인터페이스는 슬레이브에 구현된다.
  → 마스터는 이 인터페이스를 사용하여 각 서브태스크의 처리를 위임한다.

- **4단계** 이전 단계에서 개발된 명세에 따라 **슬레이브 컴포넌트**를 **구현**한다.

- **5단계** 1~3단계에서 개발한 명세에 따라 **마스터 컴포넌트**를 **구현**한다.

Master-Slave Pattern – Variant

- 변형(Variant)

  - 장애 허용성을 위한 Master-Slave 패턴

    → 마스터는 정해진 개수의 복제된 구현물에 서비스의 실행을 단지 위임하기만 한다.

    장애 허용성(fault tolerance) : 최소한 하나의 슬레이브는 장애가 발생하지 않고 제대로 동작하여 유효한 결과를 제공할 수 있는 상황을 지원하는 것.

  - 병렬 계산을 위한 Master-Slave 패턴

    → Master-Slave 패턴의 가장 일반적인 형태. 서브태스크가 병렬적으로 독립적으로 실행되어, 마스터에 결과를 반환한다.

Master-Slave Pattern – Variant

- 변형(Variant)
  - 계산 정확도(computational accuracy)를 위한 Master-Slave 패턴

    → 최소한 3개의 서로 다른 구현물에 서비스를 위임한다.

    독립된 슬레이브로 이루어진 구현물들의 작업이 완료될 때까지 기다린다.

    마스터는 각각의 결과물을 여러가지 전략적인 방법으로 선택하여 결과를 산출한다.

Master-Slave Pattern – Variant

- 변형(Variant)
  - 슬레이브를 조정하는 Master-Slave 패턴

    → 한 슬레이브의 계산이 다른 슬레이브들의 계산 상태에 따라 좌우된다.

    예) 유한 요소 모의 실험의 경우 슬레이브들을 자체적으로 조정하기 위해 계산이 완료된 슬레이브는 다른 슬레이브들이 모두 완료될 때까지 일시중단 되어야 한다.

Master-Slave Pattern – Consequence

- 결과(Consequence)

  - 교환가능성과 확장성이 도입된다.

  - 역할의 분리가 가능하다.

  - 효율이 향상된다.

  - 실행가능성이 떨어진다.

Master-Slave Pattern – Consequence

- 결과(Consequence)

  - 머신 종속성의 제약을 받는다.

  - 구현이 어렵다.

  - 이식성이 떨어진다.

감사합니다.