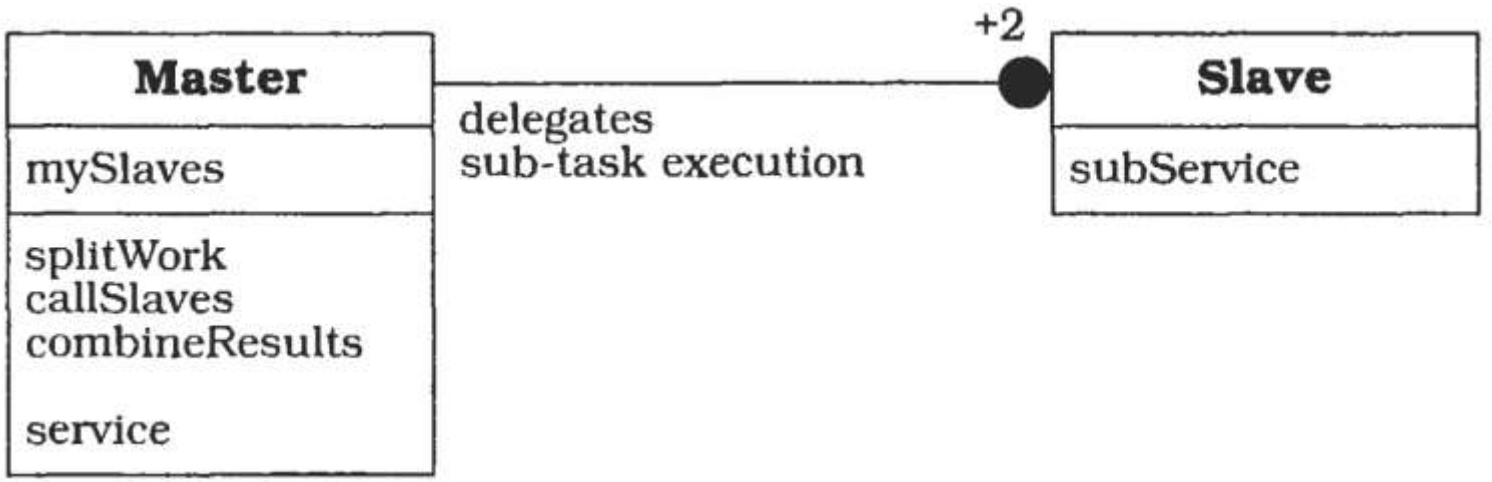
Master-Slave

Základní vlastnosti

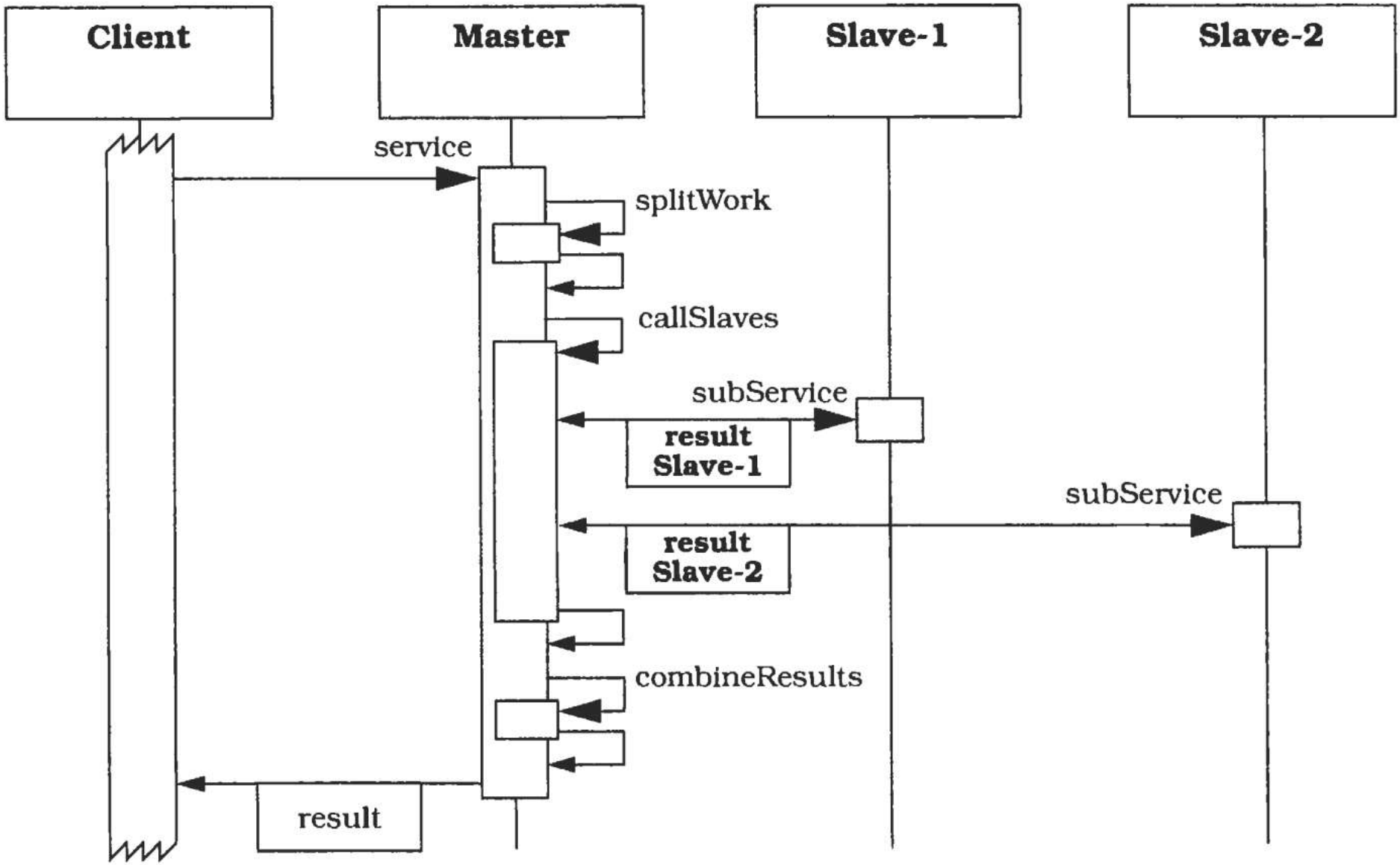
- **Princip Divide and conquer (rozděl a panuj)**
  - problém rozdělíme na shodné podproblémy
  - => ty vyřešíme nezávisle
  - z jejich výsledků pak spočítáme celkový výsledek

Architektura

- **Master (koordinační komponenta)**
  - poskytuje klientovi rozhraní pro zadání úlohy
  - rozdělení práce do shodných podúloh
  - podúlohy přidělí otrokům (slaves)
  - odstartuje a řídí výpočet podúloh - často běží paralelně
  - spočte celkový výsledek z výsledků podúloh
- **Slaves (podřízené prvky)**
  - rozhraní pro zadání podúlohy
  - provádí výpočet podúlohy



Architektura



Poznámky k implementaci

- **Jak budou otroci přistupovat k datům / vracet výsledky?**
  - dostanou jako argument / výsledek vrátí jako návratovou hodnotu
  - přistup / zápis do sdíleného kontejneru
- **Jak rozdělit na podúlohy?**
- **Jak sloučit výsledky?**
- **Jak zacházet s chybami?**

Poznámky k implementaci

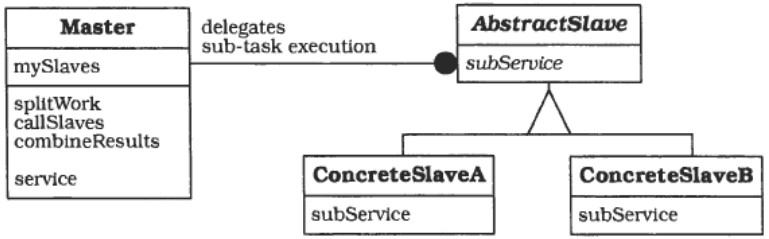
- klient je odstíněn od organizace výpočtu
- slave je zodpovědný pouze za řešení podúlohy
- rozdělení úloh a spočtení celkového výsledku má na starosti pouze master
- klient ani zpracování podúloh by nemělo záviset na algoritmu pro rozdělení zadání

Příklad

- **úloha**
  - spočtení součinu matic
- **základní myšlenka**
  - aplikujeme princip rozděl a panuj
  - každý slave spočítá **paralelně** 1 řádek výsledku
- **implementace**
  - https://github.com/VL-CZ/master-slave-design-pattern

Varianty

- **pro paralelní výpočty - nejčastější**
  - otroci pracují paralelně
  - master musí počkat na doběhnutí všech slaves
- **pro zvýšení přesnosti výsledku**
  - otroci mají různé způsoby implementace
  - všichni dostanou stejné zadání
- **pro zvýšení odolnosti vůči chybám**
  - otroci mají stejnou implementaci i zadání



- **Implementace otroků:**
  - pomocí vláken
    - ideální je použít např. Thread-pool
  - pomocí procesů

Využití

- **Násobení matic**
  - každý řádek výsledku spočítá jeden slave
- **Zpracování obrázků**
  - např. převod obrázku na černobílý, nebo počítání diskrétní kosinové transformace
  - každý slave pracuje s blokem N x N pixelů
- **Problém obchodního cestujícího**
  - (bez vracení do poč. bodu)
  - https://en.wikipedia.org/wiki/Travelling_salesman_problem
  - každý slave může např. začít v jiném vrcholu

Výhody

- **oddělení zodpovědností**
  - Slave – řešení podúloh
  - Master – rozdělení na podúlohy, zavolání otroků, spočtení celkového výsledku
- **snadná zaměnitelnost a rozšiřitelnost částí**
  - můžeme změnit implementaci jedné části (master/slave) bez zásahu do druhé
- **efektivita**
  - paralelní zpracování úloh je typicky rychlejší

Nevýhody

- **není vhodné pro všechny typy úloh**
  - nelze rozdělit na podúlohy
  - nemusí se vyplatit režie
- **závislost na HW**
  - např. na počtu jader
- **Problémy s implementací**
  - typické problémy s multi-threadingem (např. data races)
  - může být těžké převést do nezávislých podúloh
  - detekce a obsluha chyb