AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

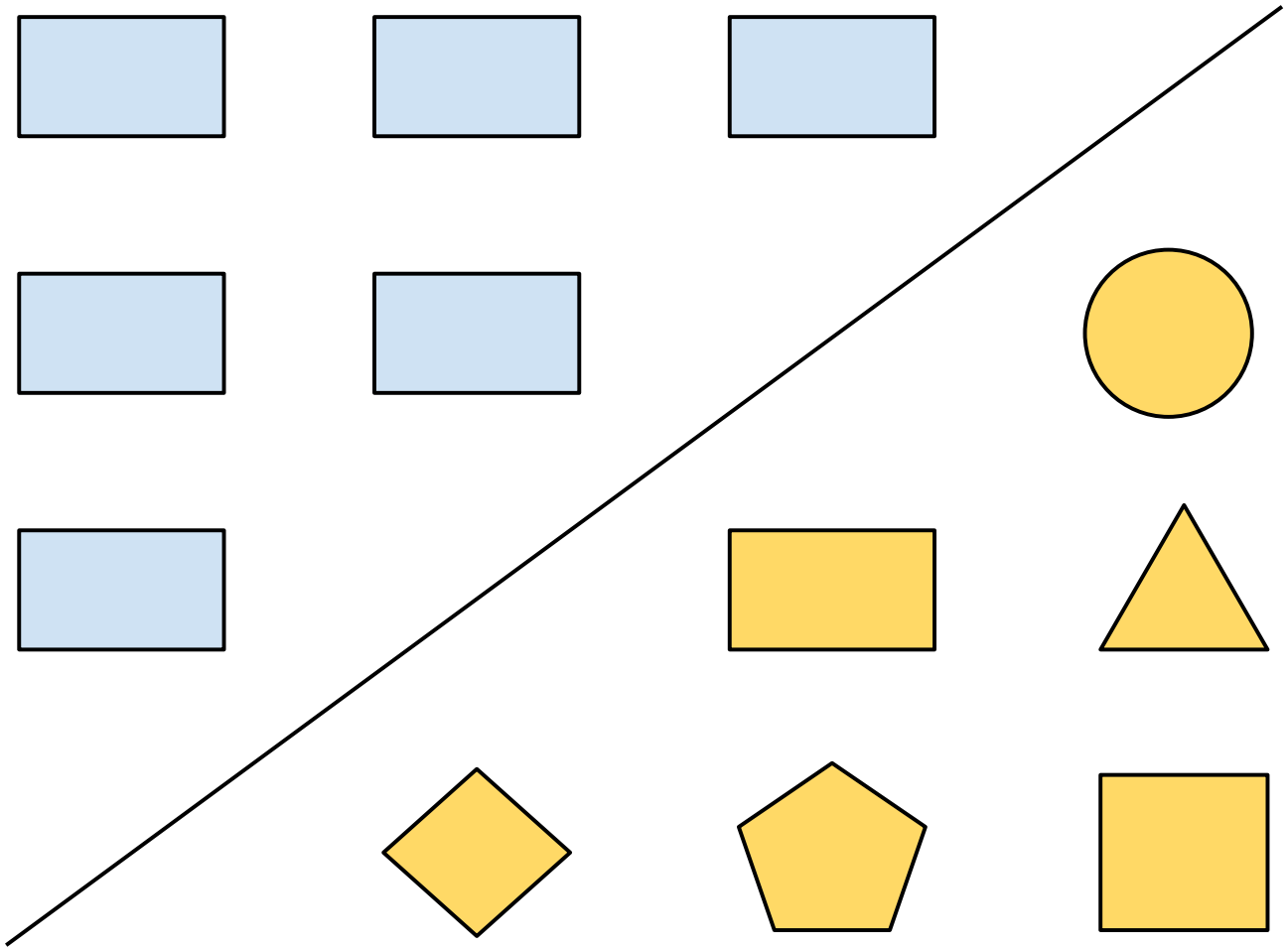
Projekt języka programowania wspierającego przetwarzanie rozproszone na platformach heterogenicznych.

Kajetan Rzepecki

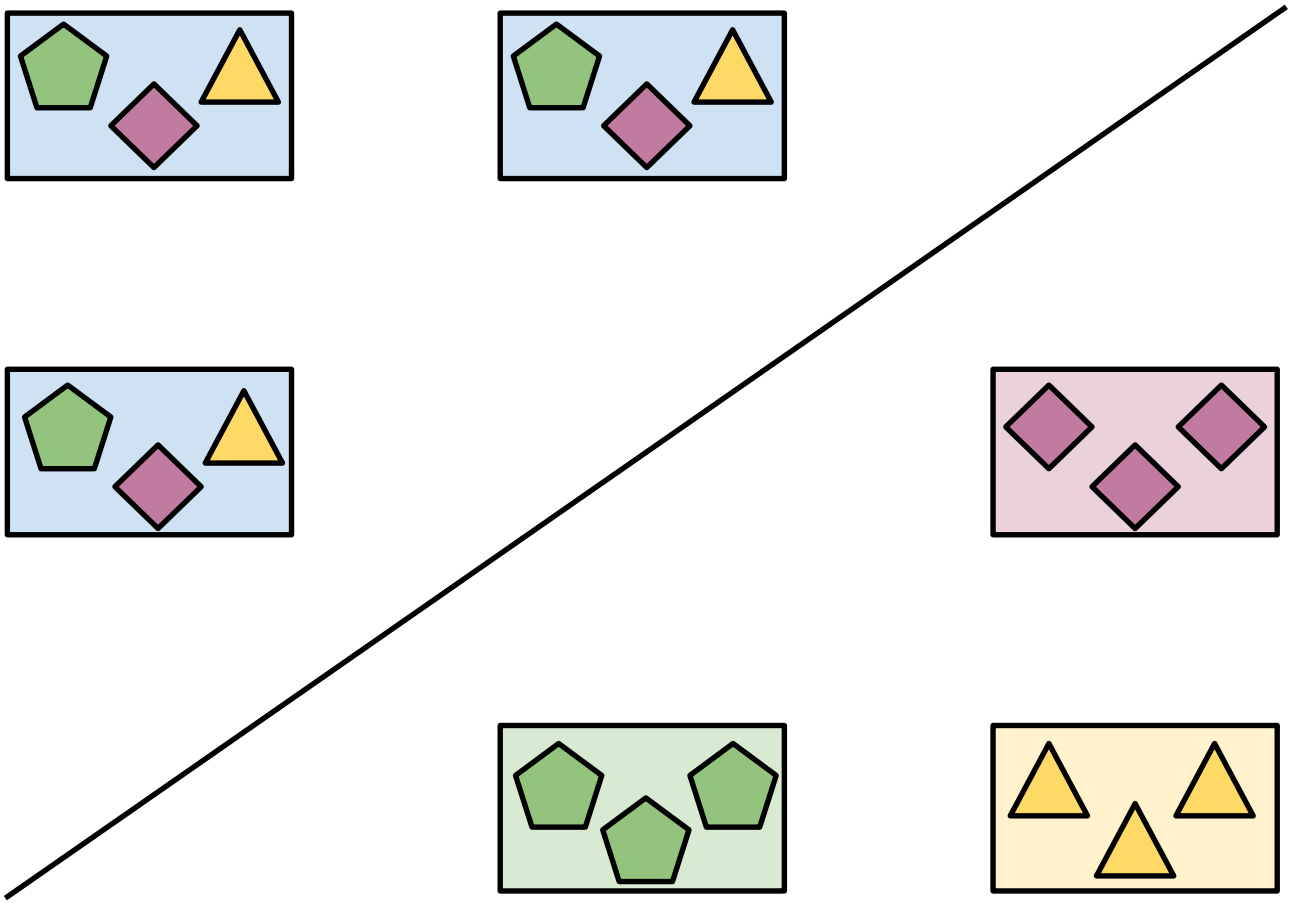
Wydział EAIiIB
Katedra Informatyki Stosowanej

31 maja 2015

Problem - Heterogeniczność



Problem - Heterogeniczność



Problem - Na czym się skupiamy

- Skalowalność
- Dynamiczność
- Niezawodność
- Konfiguracja

Problem - Na czym się skupiamy

- Skalowalność
- Dynamiczność
- Niezawodność
- Konfiguracja

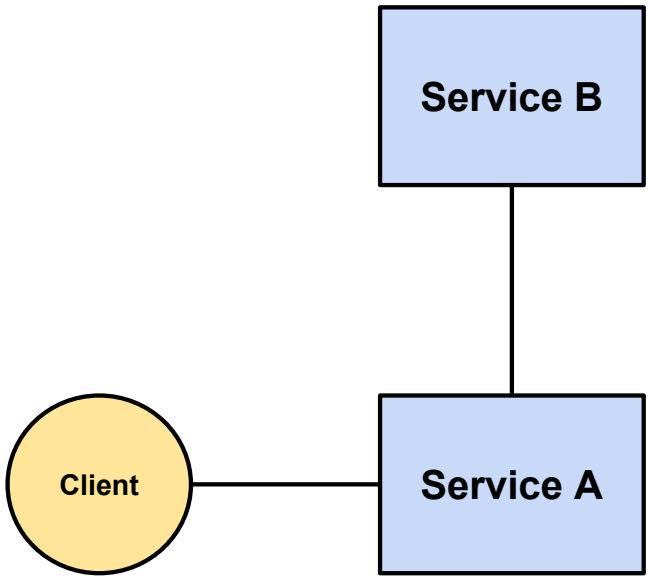
- Heterogeniczność?
- Świadomość platformy?

Problem - State of the Art

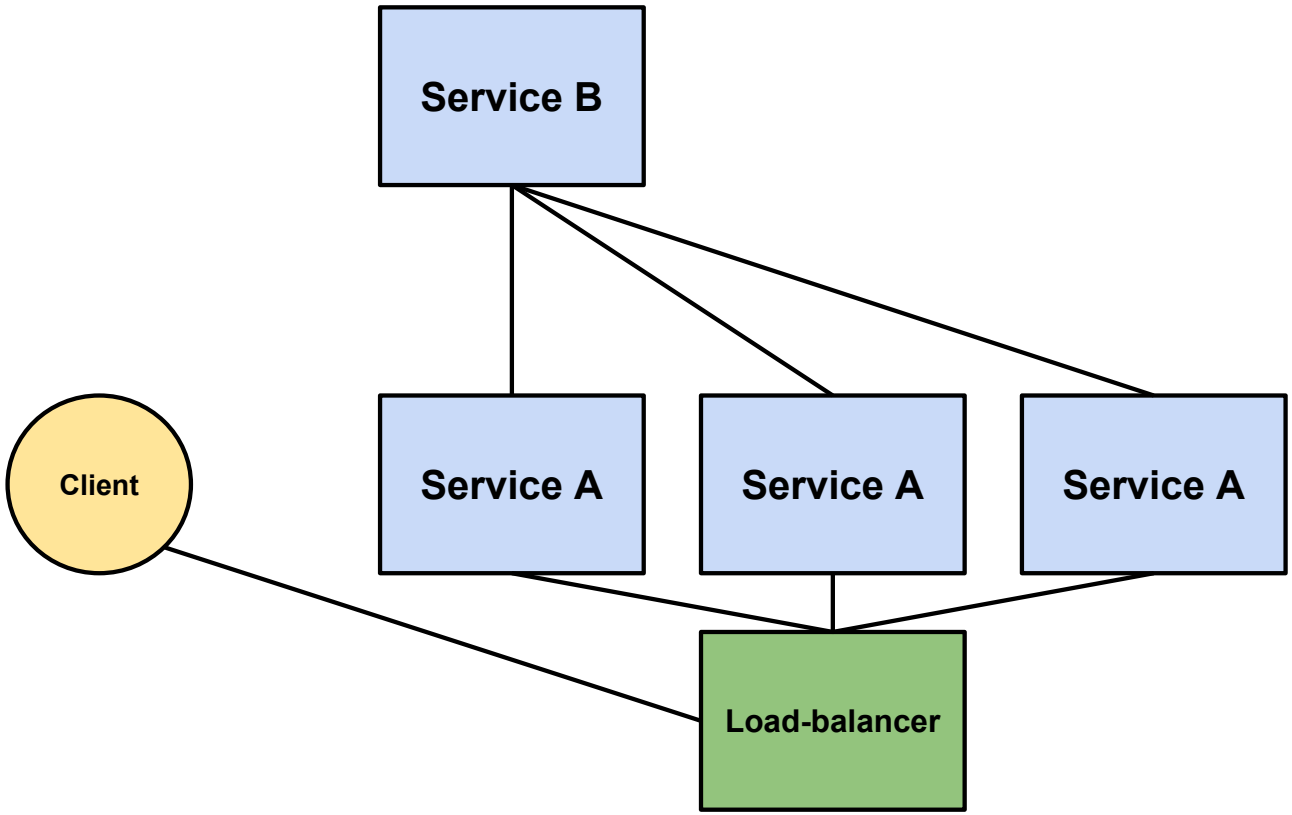
Problem - State of the Art

- Heterogeniczność?
- Świadomość platformy?

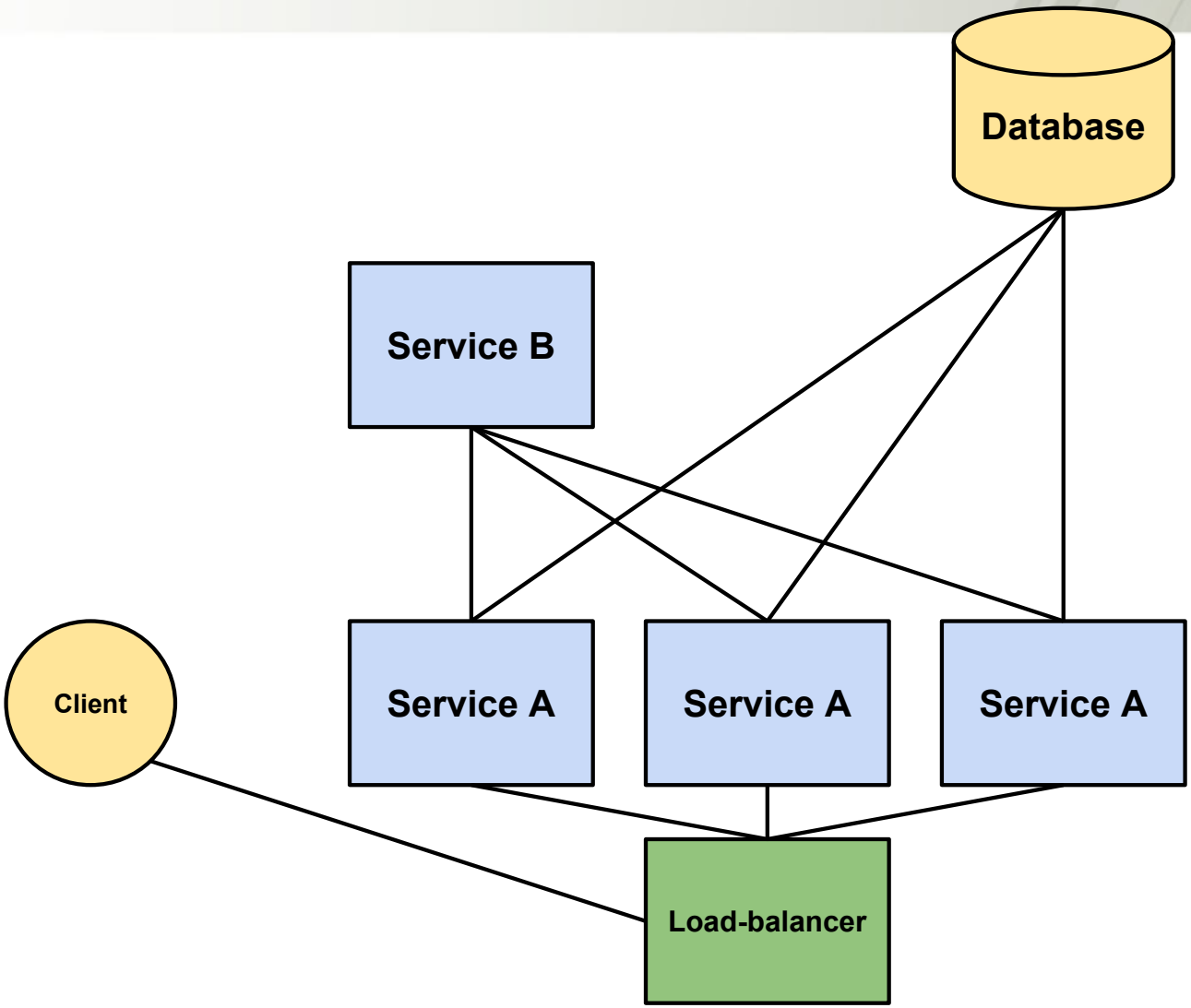
Problem - Przykład



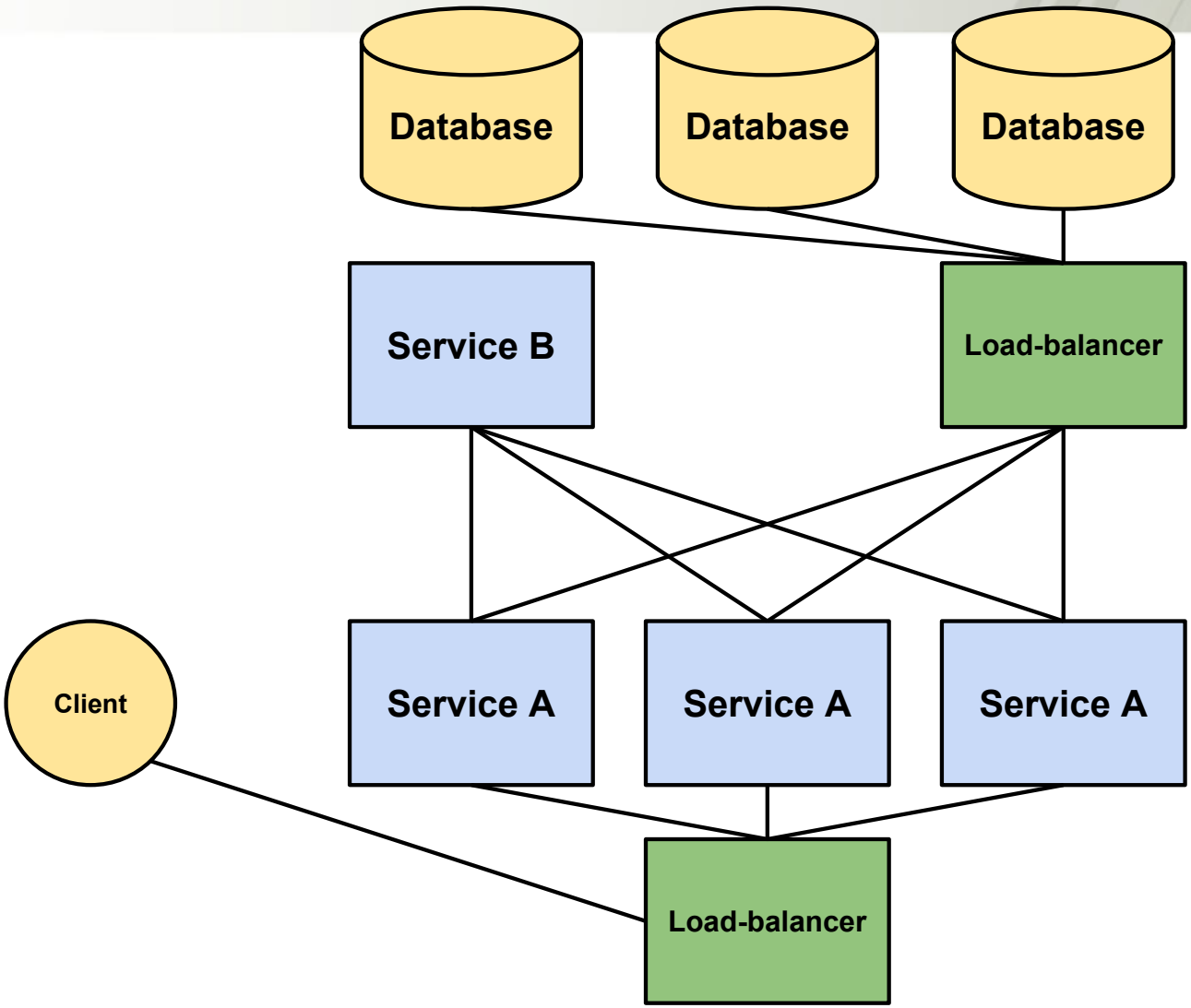
Problem - Przykład



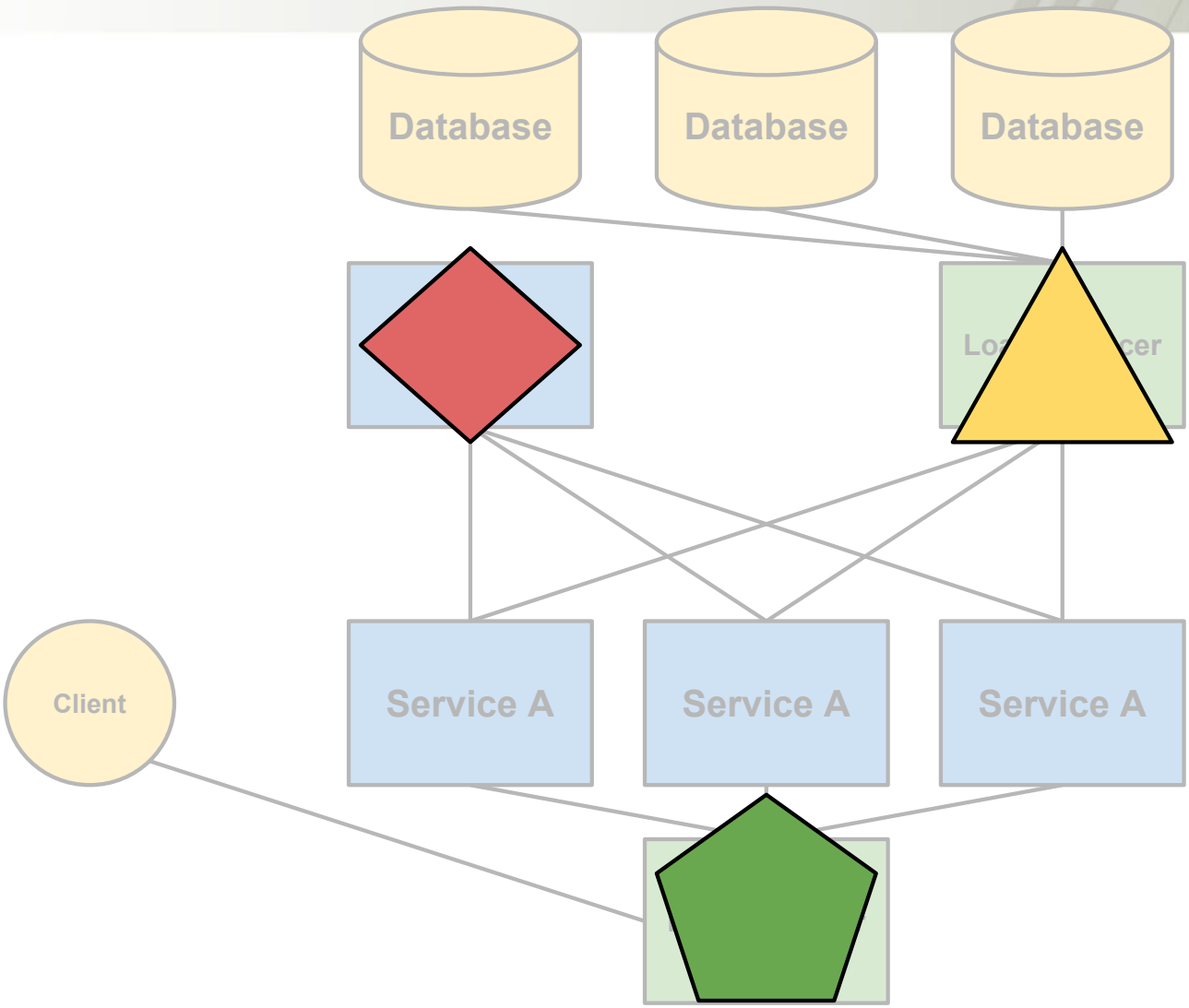
Problem - Przykład



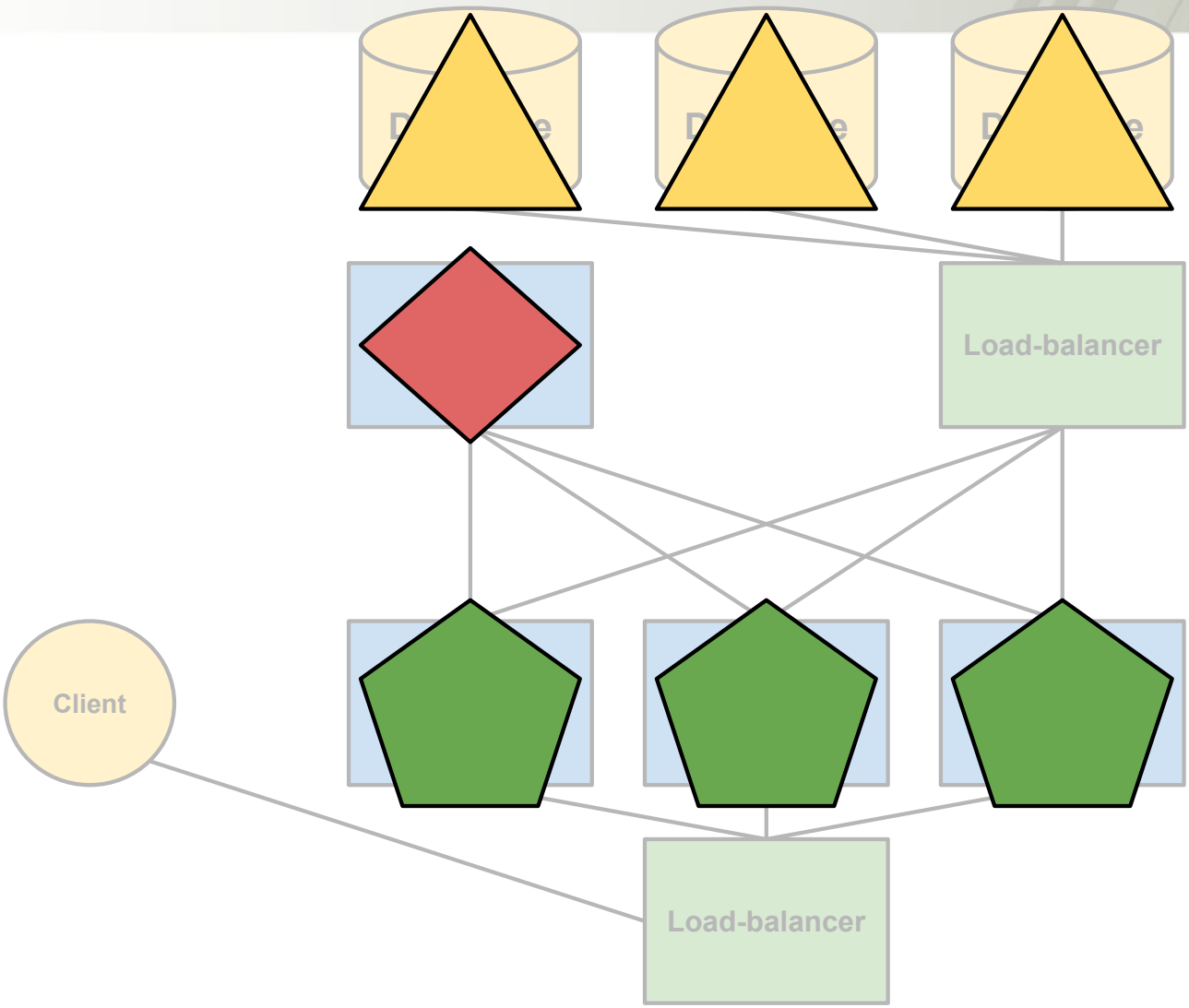
Problem - Przykład



Problem - Heterogeniczność bez świadomości



Problem - Heterogeniczność bez świadomości



Problem - Cel pracy

- Projekt języka programowania

Problem - Cel pracy

- Projekt języka programowania
- Kompilator
- System uruchomieniowy + dystrybucja
- System modułowy + wiedza

Problem - Cel pracy

- Projekt języka programowania
- Kompilator
- System uruchomieniowy + dystrybucja
- System modułowy + wiedza
- Integracja z wieloma platformami

Architektura - Prosty rdzeń języka

- Przenośny i integrowalny z wieloma platformami

Architektura - Prosty rdzeń języka

- Przenośny i integrowalny z wieloma platformami
- λ calculus - funkcje:

```
(lambda (x) x)
```

Architektura - Prosty rdzeń języka

- Przenośny i integrowalny z wieloma platformami
- λ calculus - funkcje:

```
(lambda (x) x)
```

- Kontynuacje:

```
(+ 1 (reset (* 2 (shift k (k (k 4)))))) ; 17 wtf!?
```

Architektura - Prosty rdzeń języka

- Przenośny i integrowalny z wieloma platformami
- λ calculus - funkcje:

```
(lambda (x) x)
```

- Kontynuacje:

```
(+ 1 (reset (* 2 (shift k (k (k 4)))))) ; 17 wtf!?
```

- Model Aktorowy - procesy:

```
(send (spawn do-something) 'message)
```

Architektura - System modułowy

- Oparty o reguły:

```
(whenever (and (module ?m) (provides ?m feature))
  (start-using ?m))
```

Architektura - System modułowy

- Oparty o reguły:

```
(whenever (and (module ?m) (provides ?m feature))
  (start-using ?m))
```

- Moduł dostarcza funkcjonalność vs. moduł wymaga funkcjonalności:

```
(module FProvider
  (provide f)
  (function (f) ...))
(module FUser
  (require FProvider))
```

Architektura - System modułowy

- Oparty o reguły:

```
(whenever (and (module ?m) (provides ?m feature))
  (start-using ?m))
```

- Moduł dostarcza funkcjonalność vs. moduł wymaga funkcjonalności:

```
(module FProvider
  (provide f)
  (function (f) ...))
(module FUser
  (require FProvider))
```

```
(module FProvider
  (function (f) ...))
(module FUser
  (require ?m
    (and (module ?m)
         (provides ?m f))))
```

Architektura - Przetwarzanie wiedzy

- Automatyczne odkrywanie i propagacja wiedzy

```
(module Foo                    (module Foo)
  (function (bar) ...))        (provides Foo bar)
```

Architektura - Przetwarzanie wiedzy

- Automatyczne odkrywanie i propagacja wiedzy

```
(module Foo                          (module Foo)
  (function (bar) ...))              (provides Foo bar)
```

- Dynamiczność, niezawodność i skalowalność

Architektura - Przetwarzanie wiedzy

- Automatyczne odkrywanie i propagacja wiedzy

```
(module Foo                          (module Foo)
  (function (bar) ...))              (provides Foo bar)
```

- Dynamiczność, niezawodność i skalowalność
- Przetwarzanie złożonych zdarzeń

Architektura - Przetwarzanie wiedzy

- Automatyczne odkrywanie i propagacja wiedzy

```
(module Foo                          (module Foo)
  (function (bar) ...))              (provides Foo bar)
```

- Dynamiczność, niezawodność i skalowalność
- Przetwarzanie złożonych zdarzeń
- Świadomość platformy

```
(@ big-oh (log N)
  (function (hillis-steele xs) ; runs on O(N) GPU cores
    ...))
```

Implementacja - Kompilator

Implementacja - System regułowy

- algorytm Rete + rozszerzenia

Implementacja - System regułowy

- algorytm Rete + rozszerzenia
- wnioskowanie w przód:

```
(whenever (and (module ?m)
               (provides ?m ?f))
  (list ?m ?f))
```

Implementacja - System regułowy

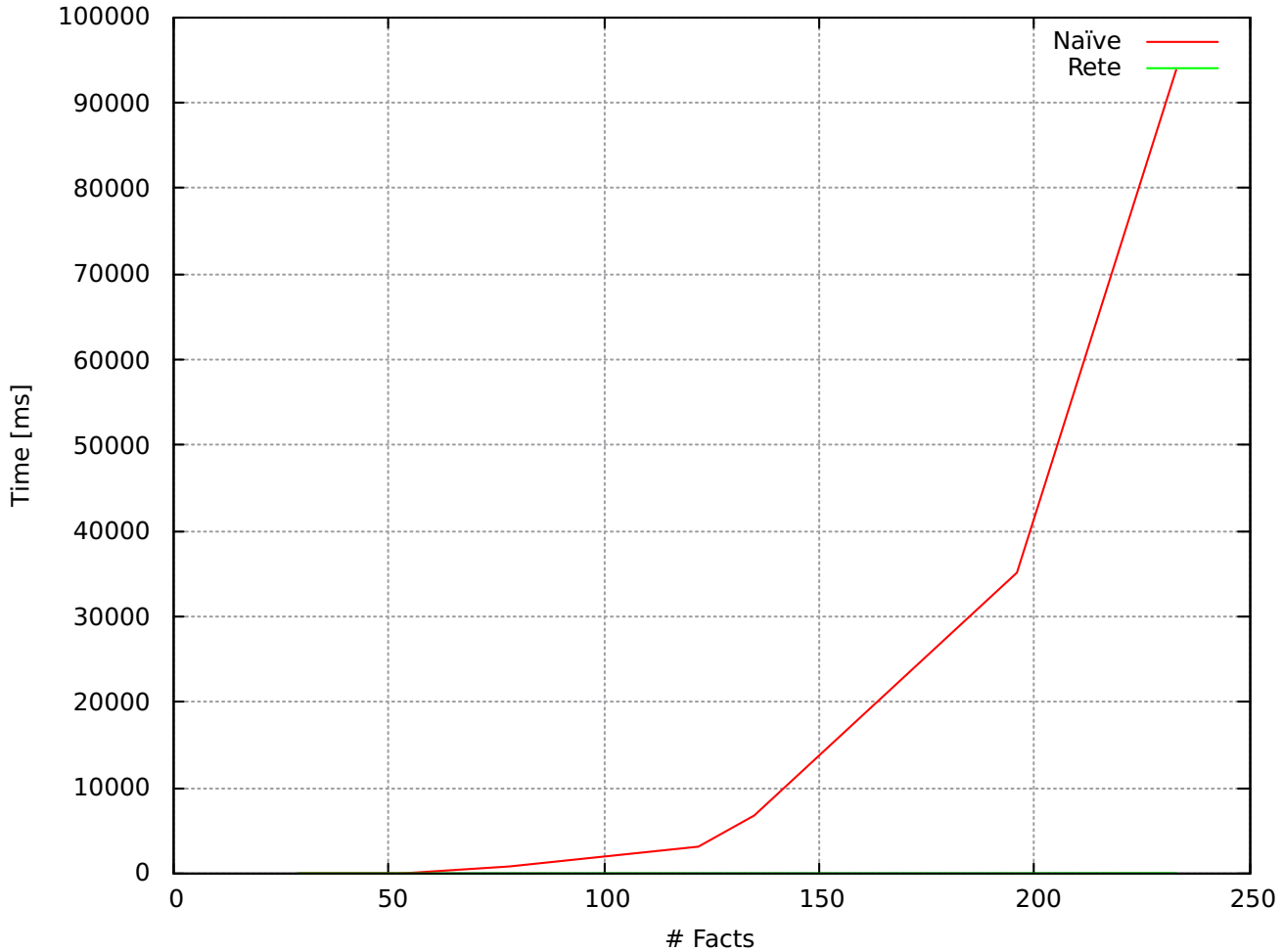
- algorytm Rete + rozszerzenia
- wnioskowanie w przód:

```
(whenever (and (module ?m)
               (provides ?m ?f))
  (list ?m ?f))
```

- wnioskowanie wstecz:

```
(select (?m ?f)
        (and (module ?m)
             (provides ?m ?f)))
```

Wyniki


Naïve
Rete
Time [ms]
Facts
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
0
50
100
150
200
250

Wyniki

- TODO: obrazek z różnymi urządzeniami gadającymi ze sobą.
- TODO: Lista rzeczy, które udało się wykonać.
- TODO: Lista przyszłych kierunków rozwoju?

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Dziękuję za uwagę.

Kajetan Rzepecki