Wstęp do Programowania potok funkcyjny

Marcin Kubica

2010/2011

Outline

Środowisko i ramki.

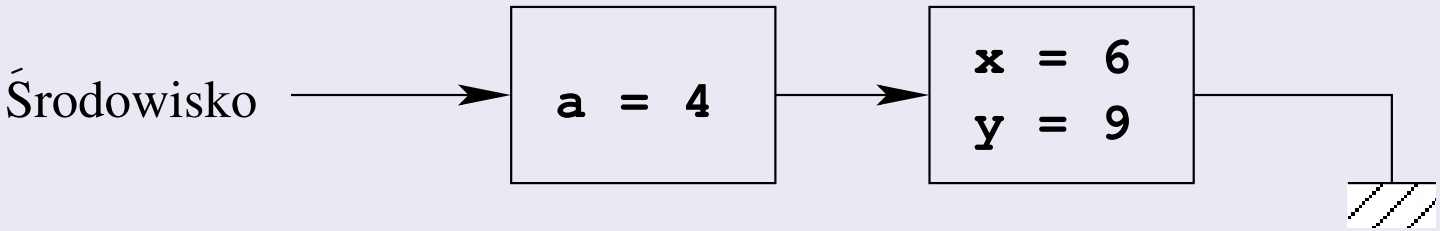
Definition

- *Środowisko* — przypisuje nazwom ich wartości.
- Istnieje wiele środowisk:
  - globalnie zdefiniowane pojęcia,
  - na potrzeby definicji lokalnych,
  - na potrzeby wywołania procedur.
- Implementacja środowiska — jednokierunkowa lista wskaźnikowa *ramek*.
- Środowisko = wskaźnik do pierwszej ramki.
- *Środowisko otaczające* i wskaźnik do niego.

Środowisko i ramki.

Definition

- Ramka zawiera:
  - jeden lub więcej identyfikatorów,
  - związane z nimi wartości,
  - wskaźnik do środowiska otaczającego.
- Wartości prostych typów wbudowanych (jedno słowo maszynowe) są pamiętane w ramce.
  Dla wartości złożonych, ramka zawiera wskaźnik do wartości.
- Przeglądamy kolejne ramki środowiska.
  Pierwsza ramka, która zawiera nazwę, określa jej wartość.



Definicje globalne

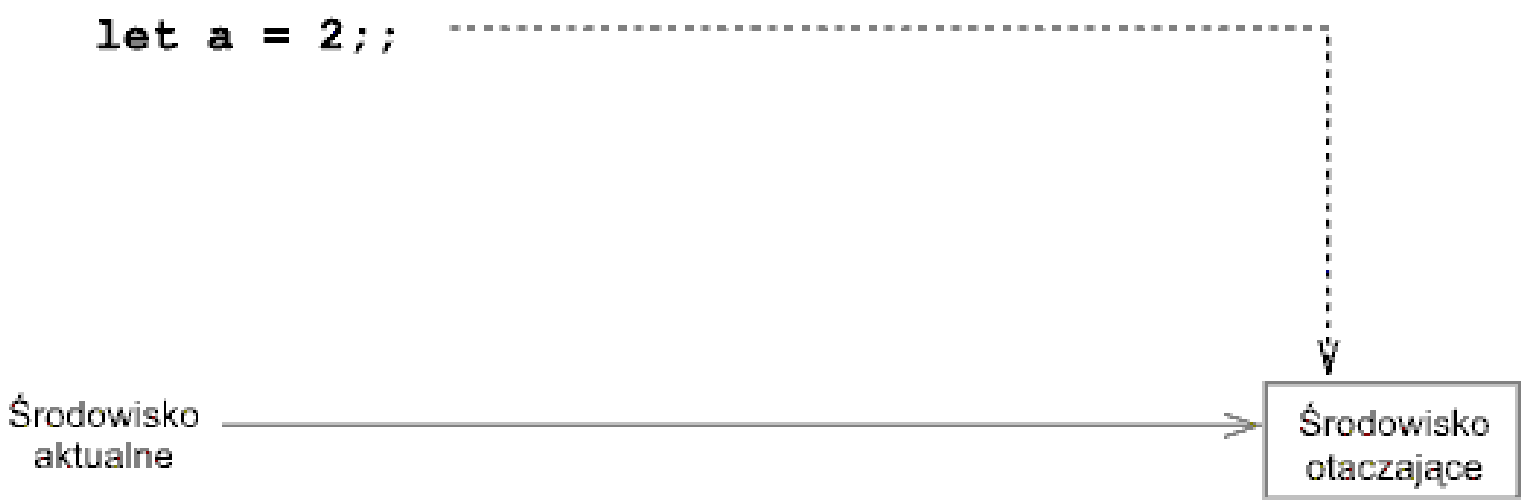
- Definicje globalne rozszerzają *środowisko globalne*.
- Dodanie nowej ramki zawierającej zdefiniowane wartości.
- Definicje równoczesne (spójnik `and`) — wiele nazw w ramce.
- Dotychczasowe środowisko globalne = środowisko otaczające.
- Przesłanianie wcześniej zdefiniowanych wartości.

Definicje globalne

Example

Definicje globalne

Example



Definicje globalne

Example

```
let a = 2;;
```

Definicje globalne

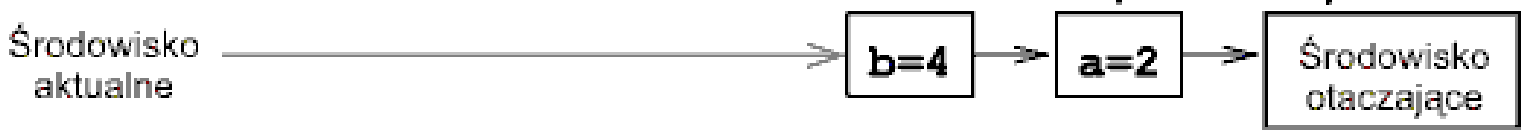
Example

```
let a = 2;;
let b = 2 * a;;
```

Definicje globalne

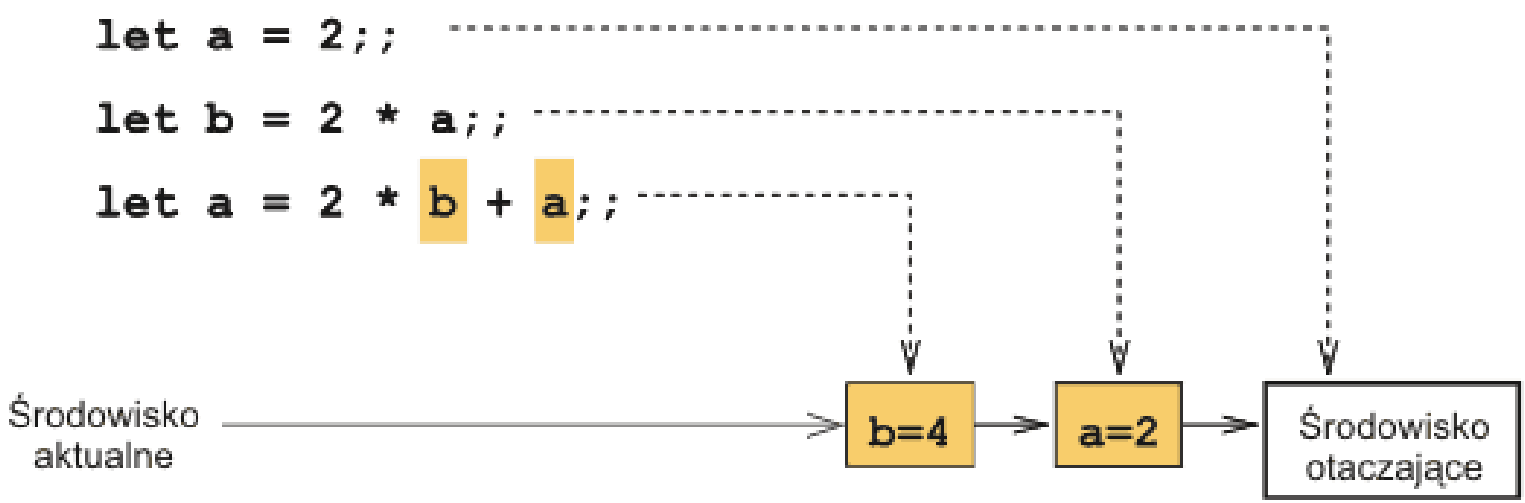
Example

```
let a = 2;;
let b = 2 * a;;
```



Definicje globalne

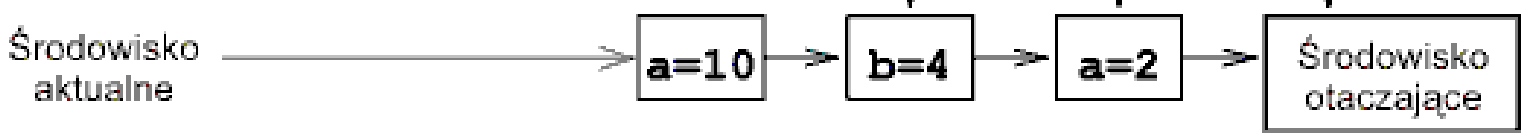
Example



Definicje globalne

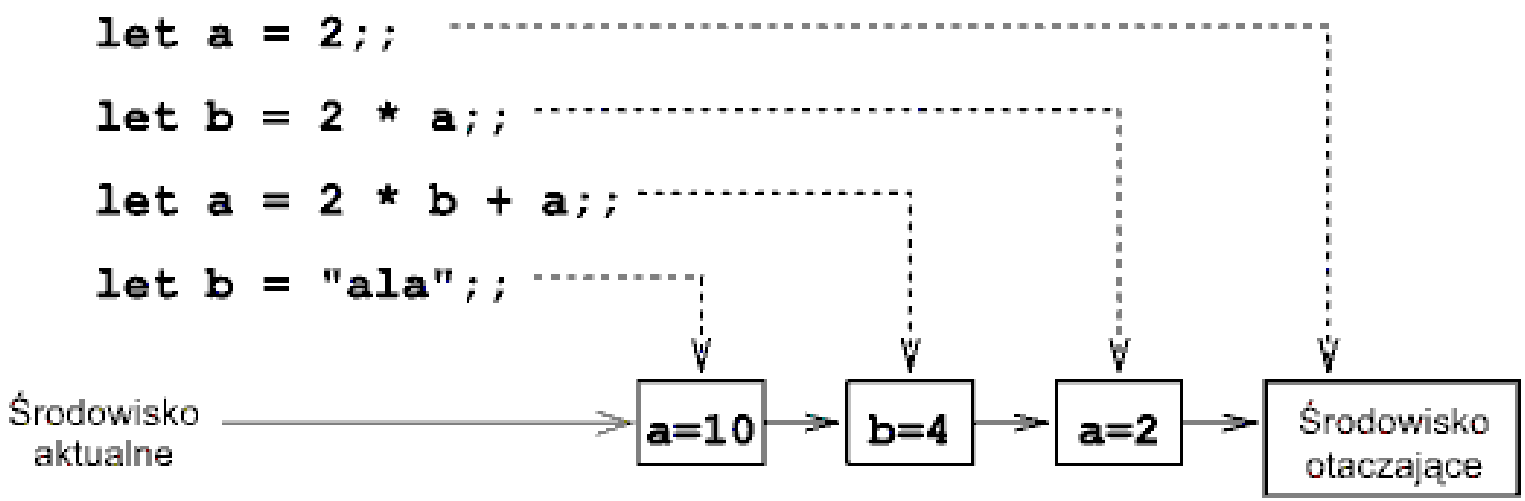
Example

```
let a = 2;;
let b = 2 * a;;
let a = 2 * b + a;;
```



Definicje globalne

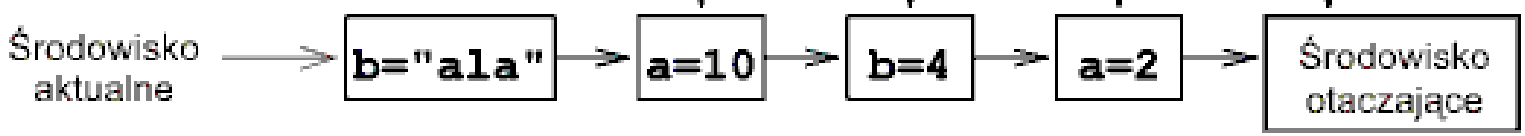
Example



Definicje globalne

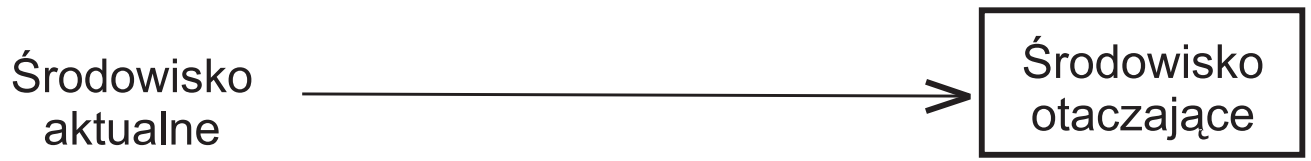
Example

```
let a = 2;;
let b = 2 * a;;
let a = 2 * b + a;;
let b = "ala";;
```



Definicje globalne

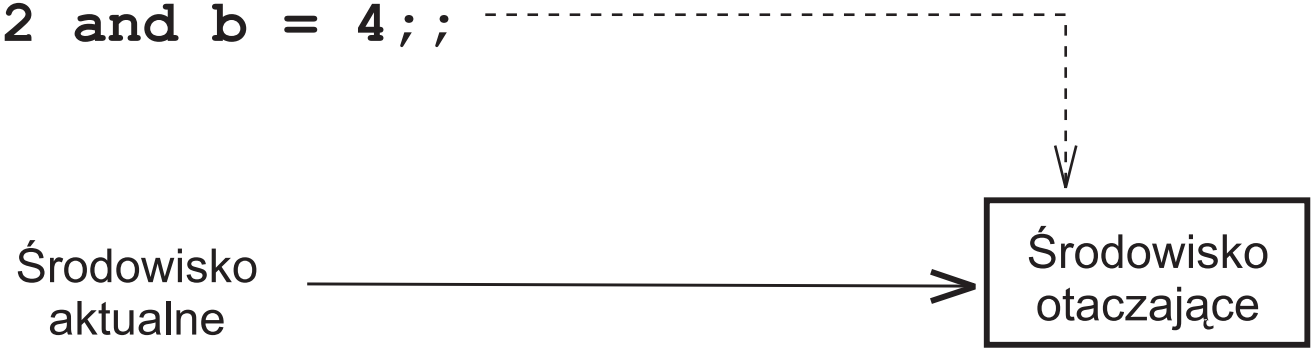
Example



Definicje globalne

Example

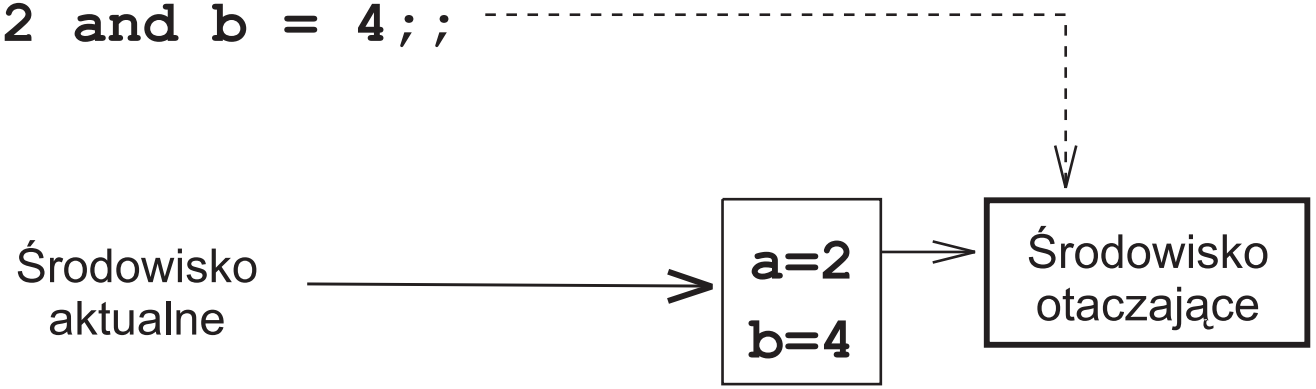
```
let a = 2 and b = 4;;
```



Definicje globalne

Example

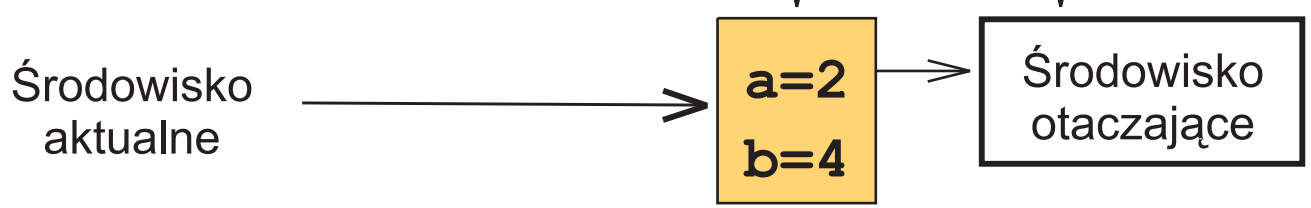
```
let a = 2 and b = 4;;
```



Definicje globalne

Example

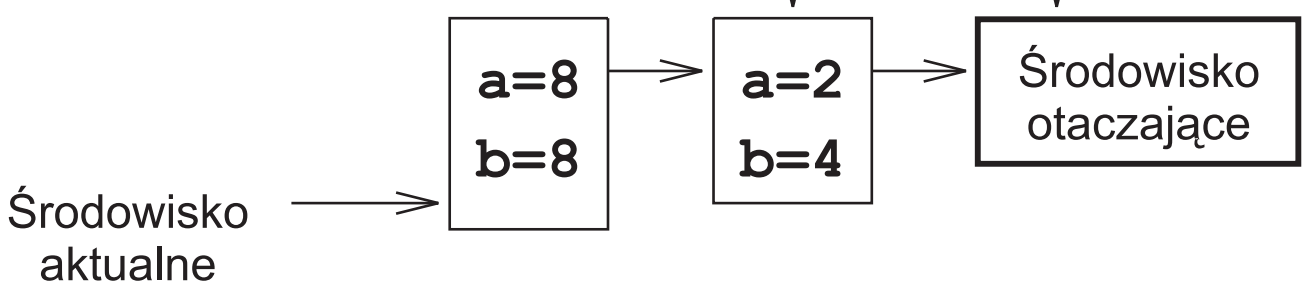
```
let a = 2 and b = 4;;
let a = 2 * b and b = 4 * a;;
```



Definicje globalne

Example

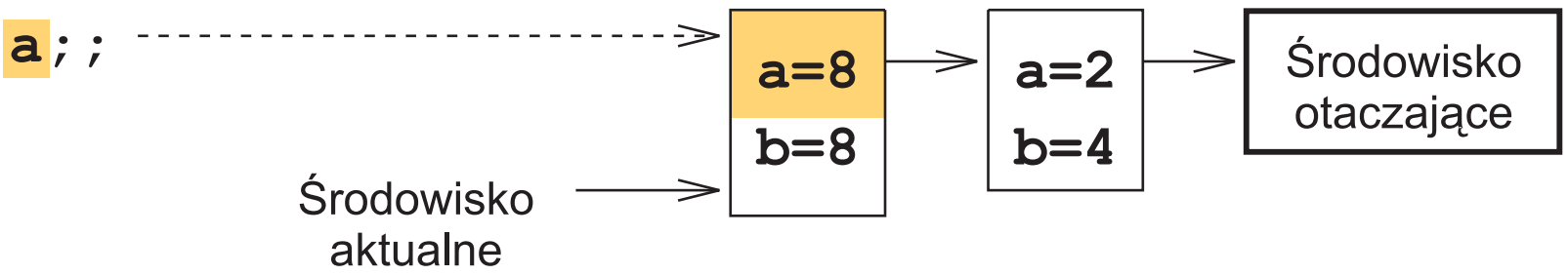
```
let a = 2 and b = 4;;

let a = 2 * b and b = 4 * a;;
```



Definicje globalne

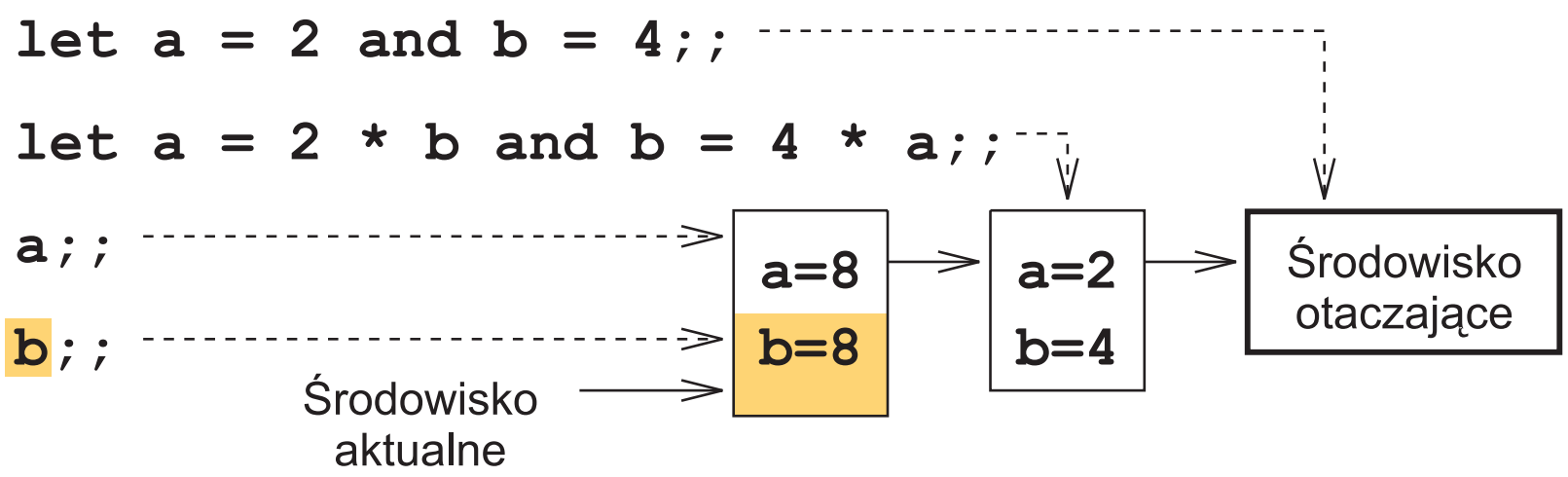
Example

```
let a = 2 and b = 4;;

let a = 2 * b and b = 4 * a;;

a;;
```



Definicje globalne

Example



Wartości typów danych

- Wartości prostych typów wbudowanych (np. `int`, `bool`, `float`) są pamiętane w ramkach wprost.
- Wartości złożonych typów danych — ramka zawiera wskaźnik do wartości.
- Wskaźnik czy wartość — jedno słowo maszynowe.
- Kopiowanie wskaźników, a nie złożonych wartości.
- Raz skonstruowane wartości nie ulegają zmianie.
- Możliwość współdzielenia fragmentów struktur danych.

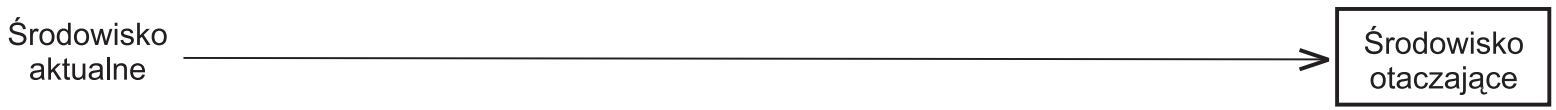
Wartości typów danych

Wartości złożonych typów danych są generalnie reprezentowane jako rekordy złożone ze wskaźników do wartości składowych:

- Element produktu kartezjańskiego — n-tka wartości/wskaźników.
- Rekord — analogicznie.
- Warianty bez argumentów — stała reprezentująca wariant.
  Warianty z argumentami — para: stała reprezentująca wariant, wartość lub wskaźnik do argumentu.
- Lista pusta — stała.
  Lista niepusta — para: głowa i ogon.

Wartości typów danych

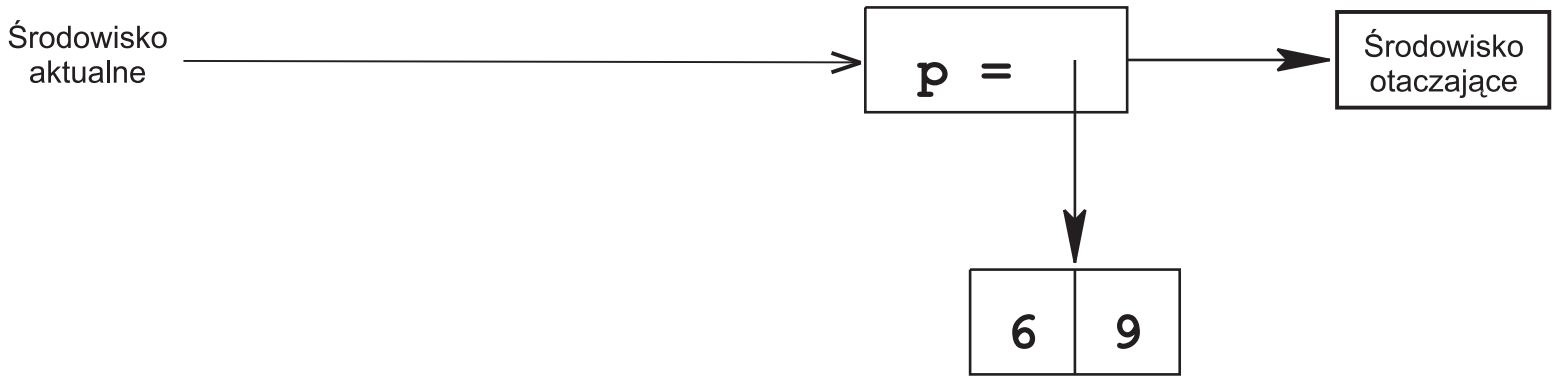
Example



Wartości typów danych

Example

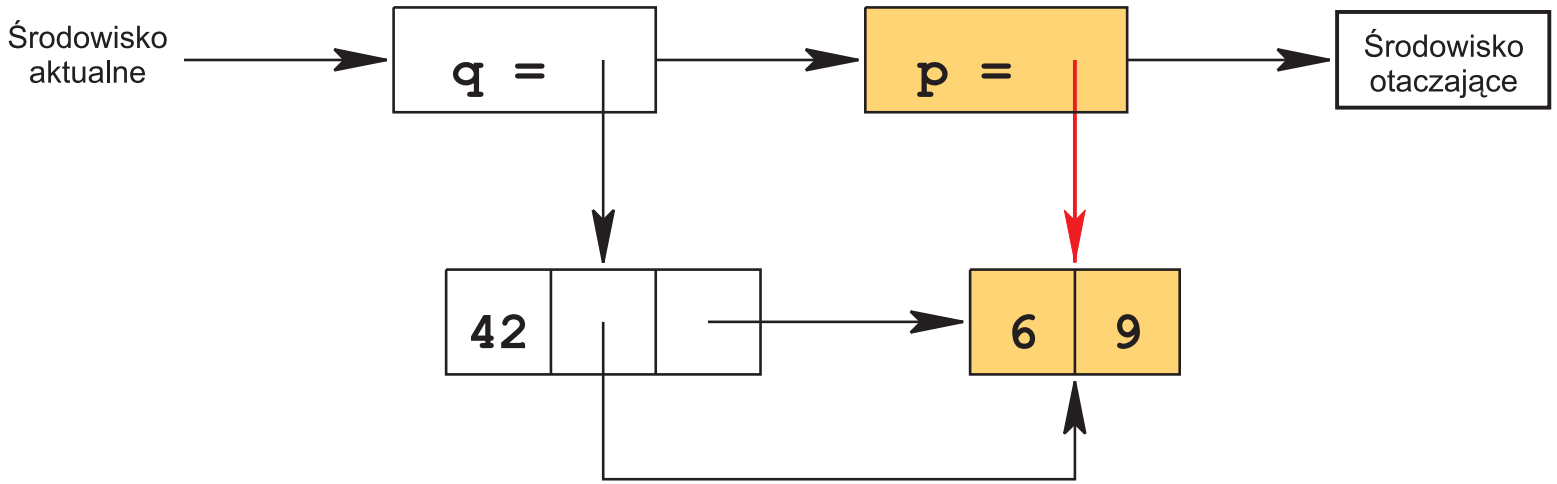
```
let p = (6,9);;
```



Wartości typów danych

Example

```
let p = (6,9);;

let q = (42, p, p);;
```



Wartości proceduralne

- Na wartość proceduralną składają się:
  - nazwy parametrów formalnych procedury (kod),
  - treść procedury (kod),
  - wskaźnik do środowiska, w którym zdefiniowano procedurę, (określony w trakcie obliczeń, procedury lokalne).
- Reprezentacja wartości proceduralnych — para wskaźników:
  - do kodu skompilowanej procedury,
  - do środowiska, w którym procedura została zdefiniowana.
- Procedury rekurencyjne — *środowisko, w którym zdefiniowano procedurę* zawiera jej własną definicję.
- Uproszczenie: procedury wieloargumentowe będą otrzymywać wszystkie swoje argumenty na raz (a nie po jednym).

Zastosowanie procedury do argumentów

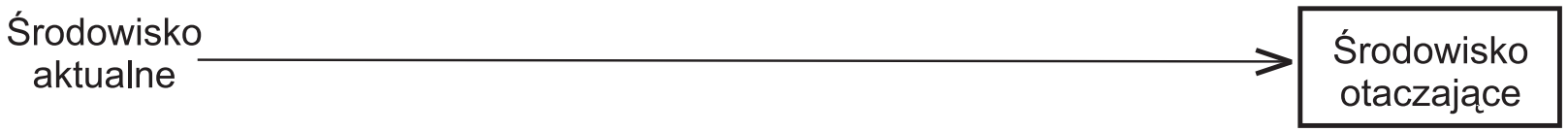
Zastosowanie procedury (wywołanie):

- wyliczenie wartości procedury i jej argumentów,
- stworzenie nowej ramki:
  - nazwy parametrów formalnych $\rightarrow$ wartości argumentów,
  - środowisko otaczające = środowisko, w którym zdefiniowano procedurę,
- wyliczenie w takim środowisku treści procedury.

Zastosowanie procedury do argumentów

Example

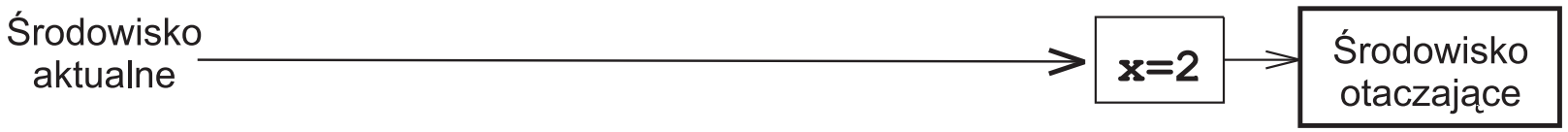
```
let x = 2;;
let square x = x * x;;
let pitagoras x y = square x + square y;;
```



Zastosowanie procedury do argumentów

Example

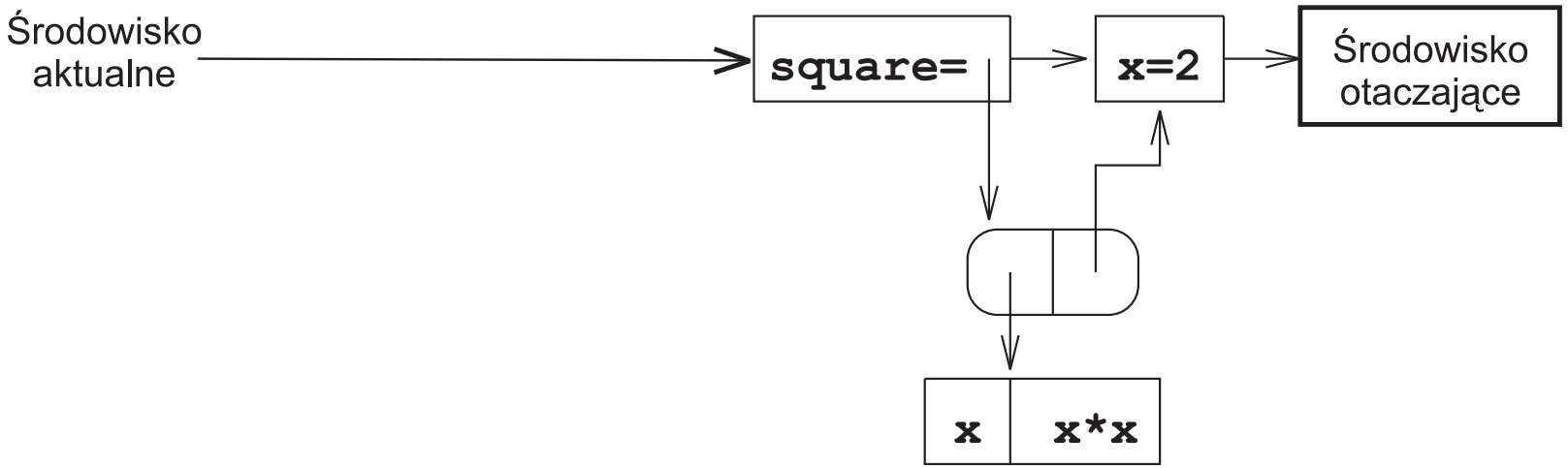
```
let x = 2;;
let square x = x * x;;
let pitagoras x y = square x + square y;;
```



Zastosowanie procedury do argumentów

Example

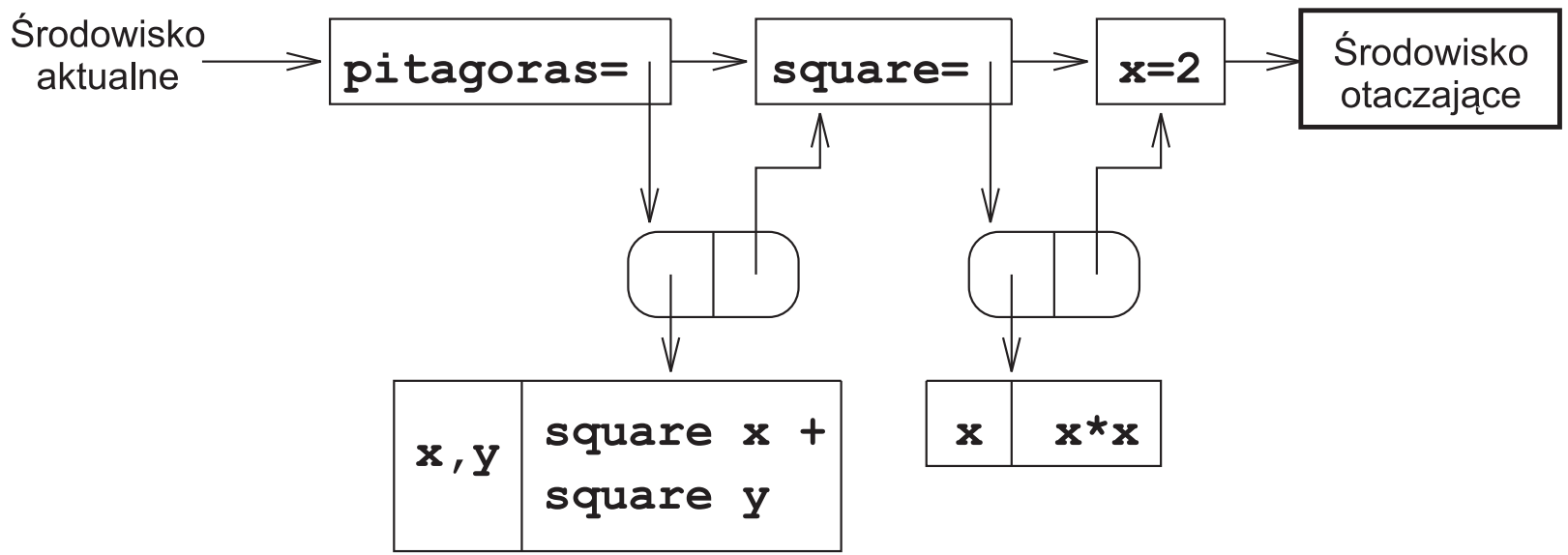
```
let x = 2;;
let square x = x * x;;
let pitagoras x y = square x + square y;;
```



Zastosowanie procedury do argumentów

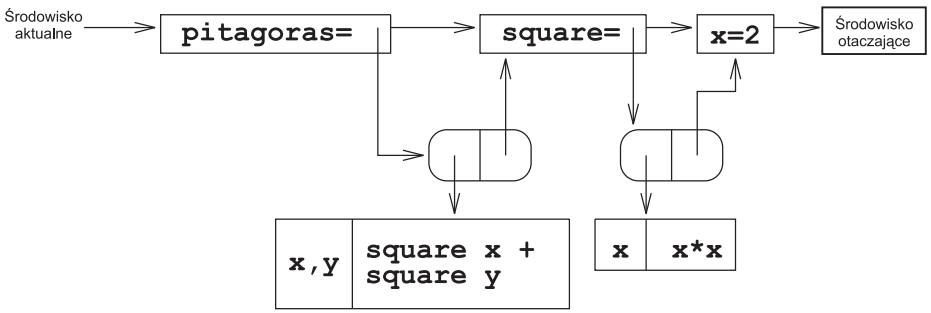
Example

```
let x = 2;;
let square x = x * x;;
let pitagoras x y = square x + square y;;
```



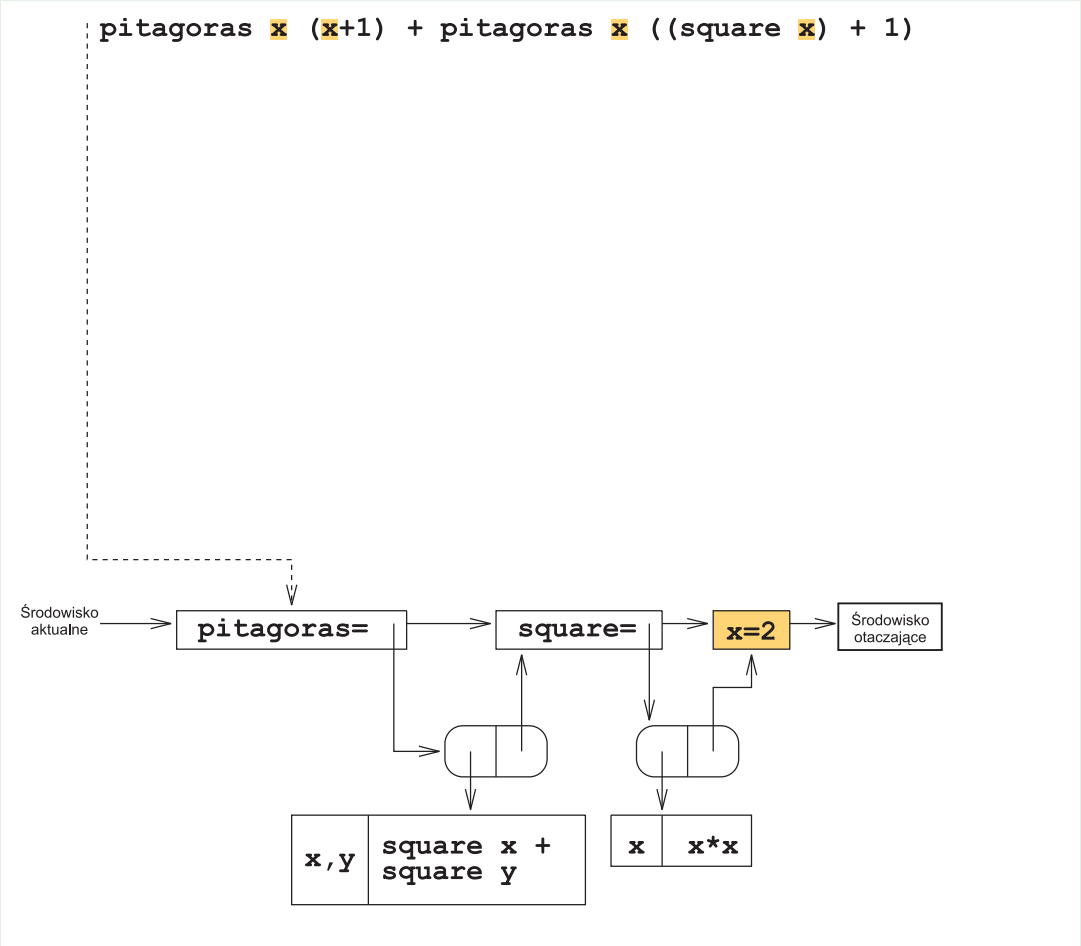
Zastosowanie procedury do argumentów

Example



Zastosowanie procedury do argumentów

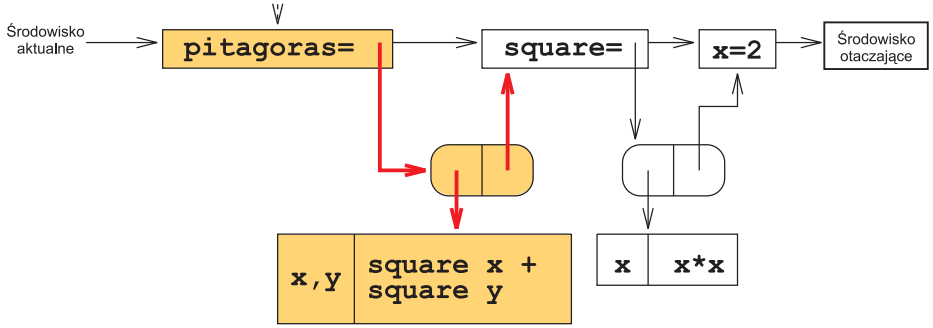
Example



Zastosowanie procedury do argumentów

Example

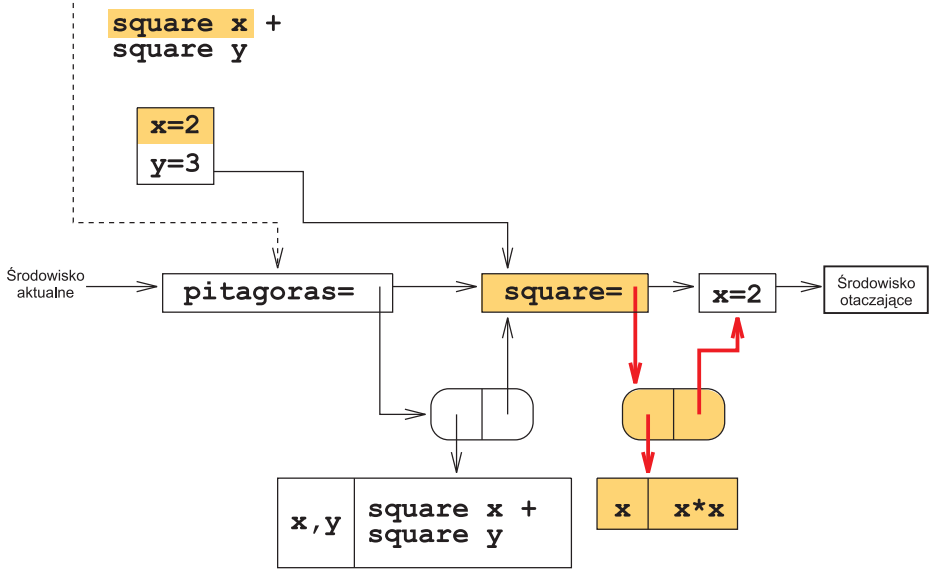
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1)
```



Zastosowanie procedury do argumentów

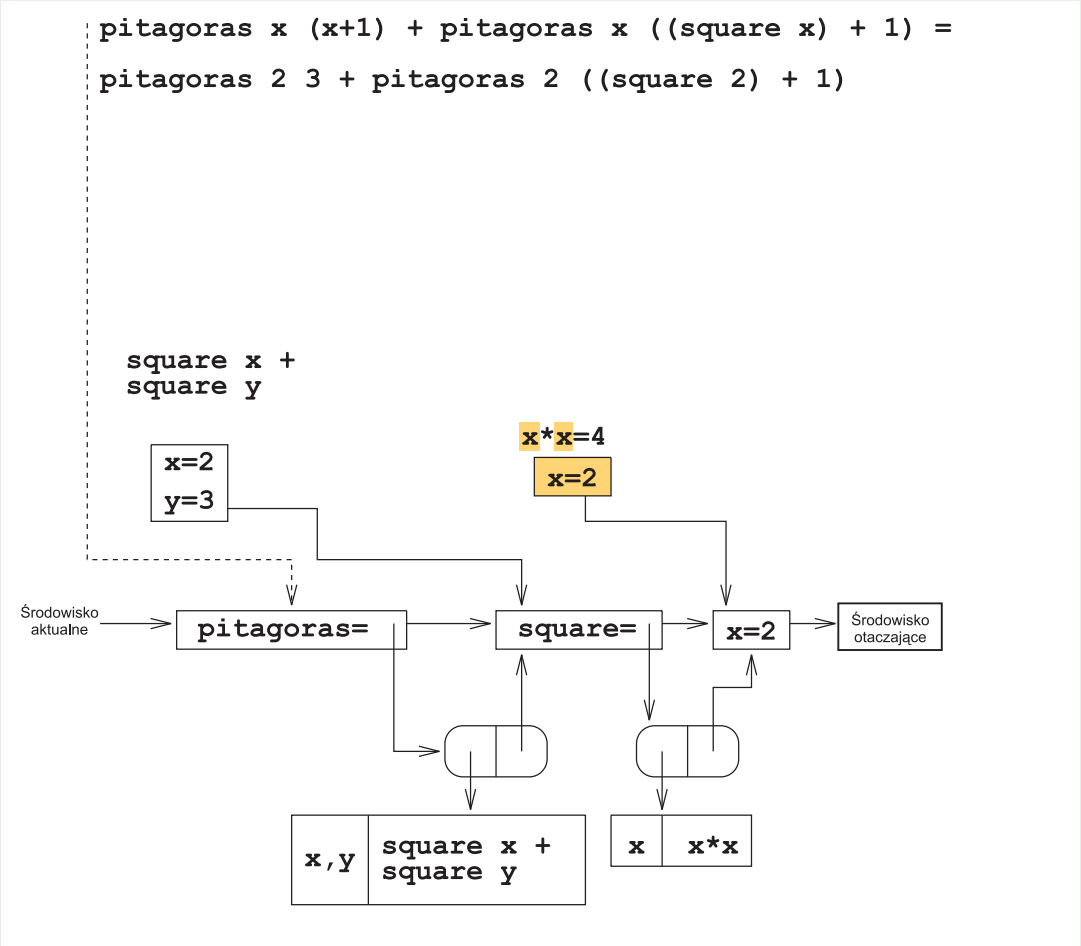
Example

```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1)
```



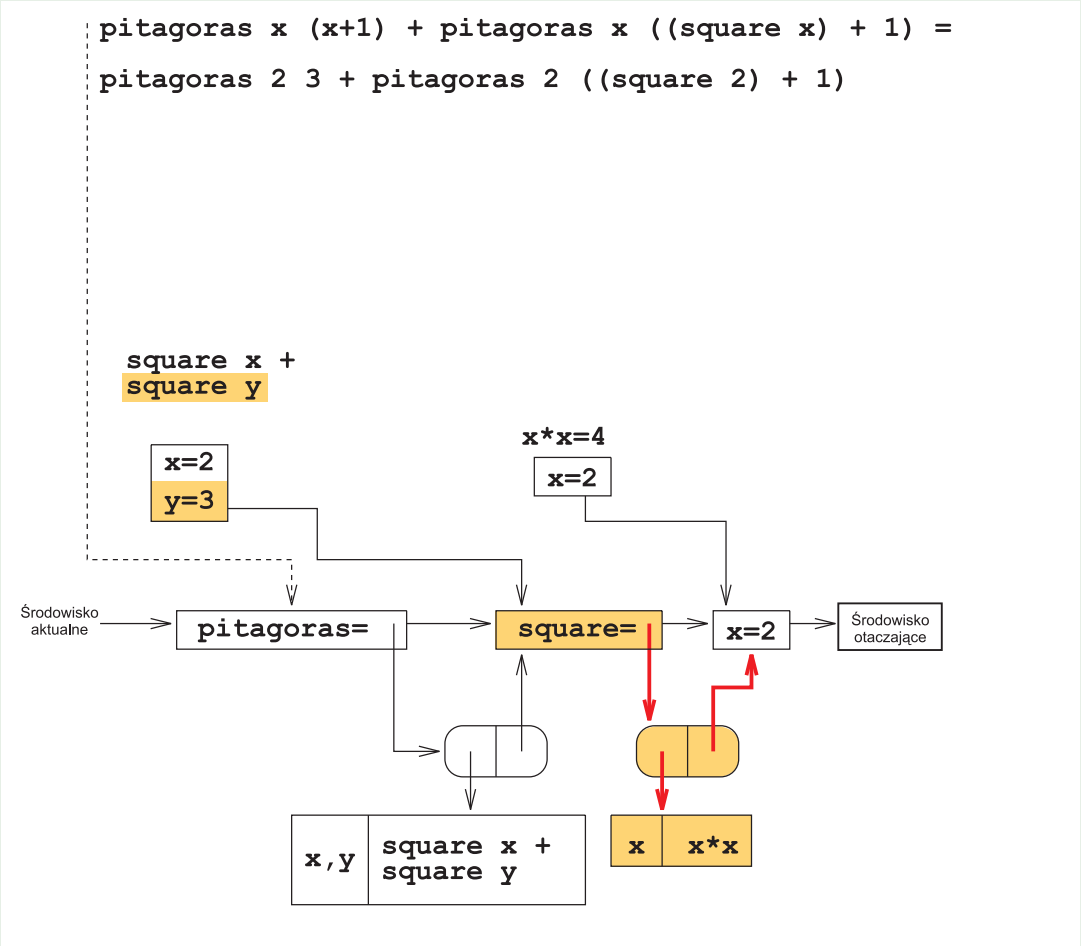
Zastosowanie procedury do argumentów

Example



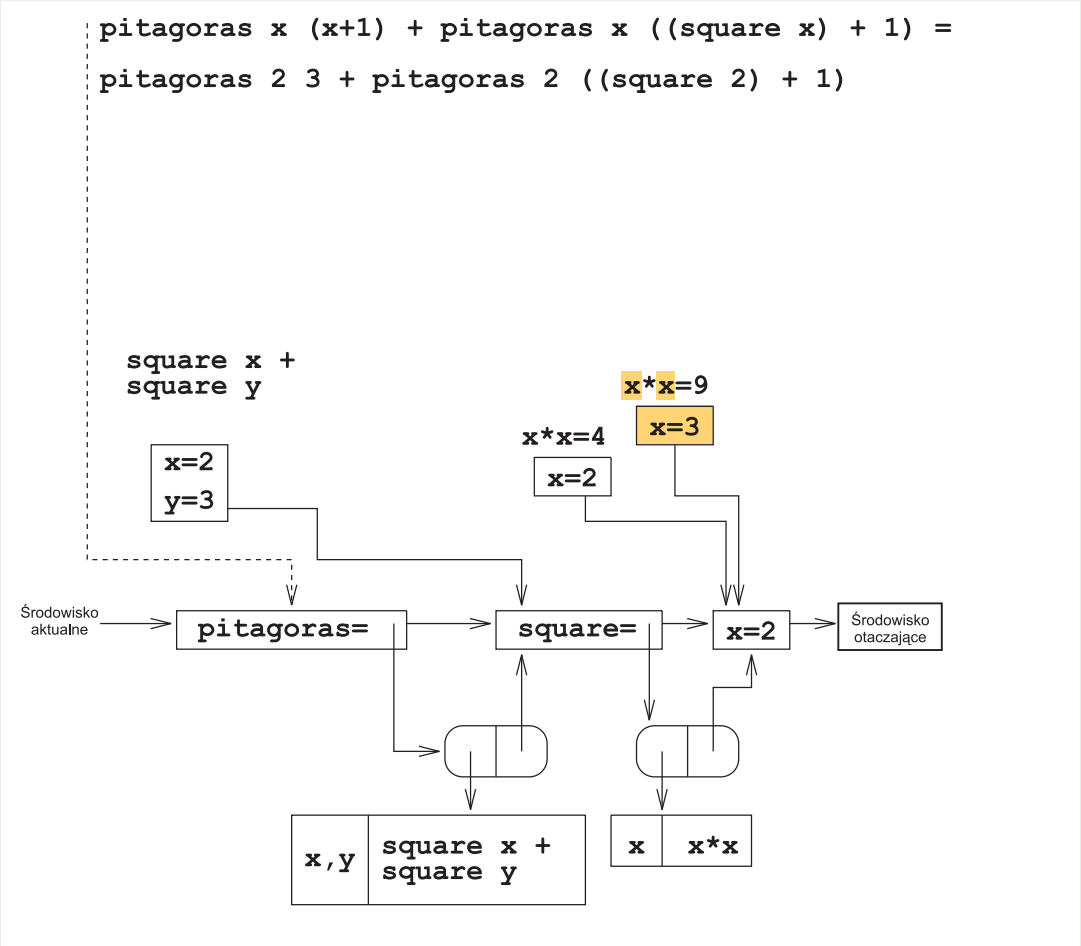
Zastosowanie procedury do argumentów

Example



Zastosowanie procedury do argumentów

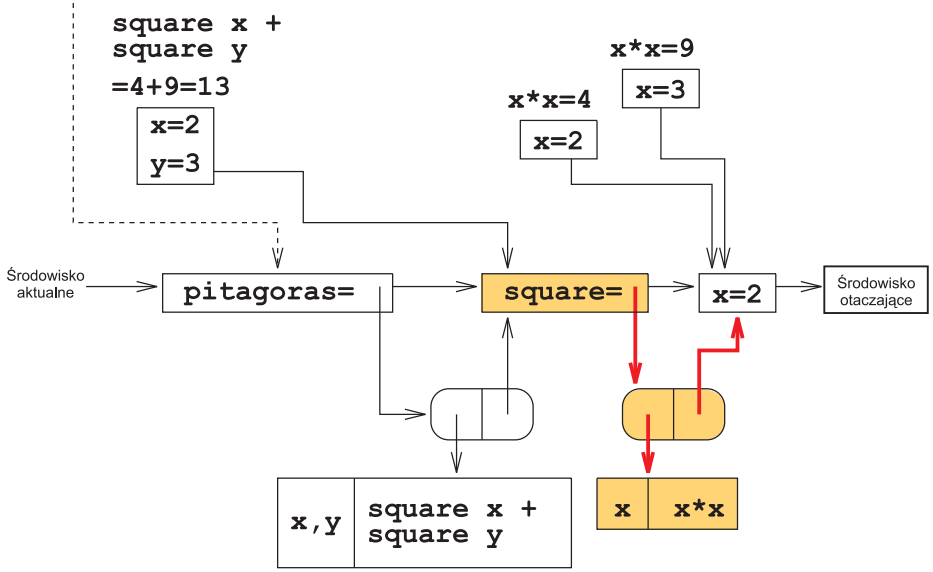
Example



Zastosowanie procedury do argumentów

Example

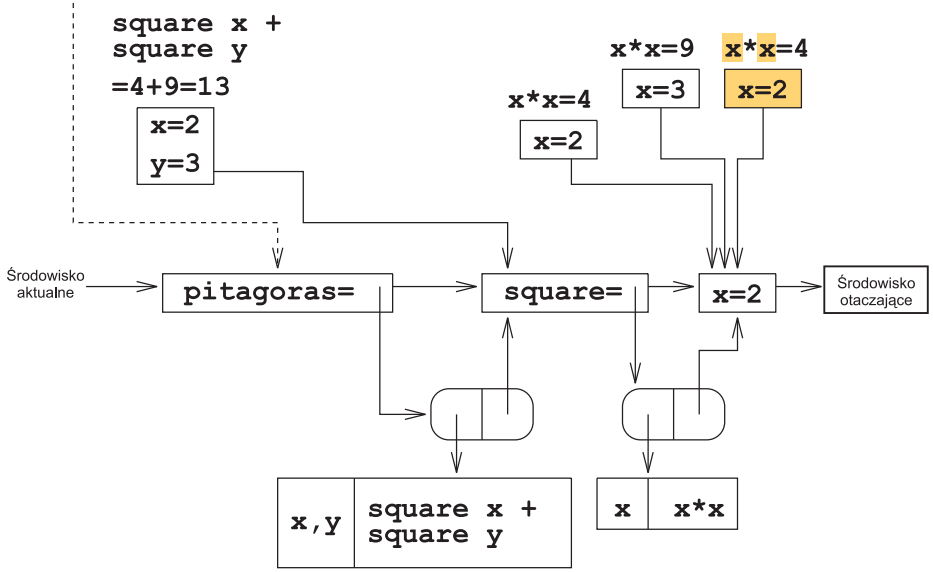
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1) =
13 + pitagoras 2 ((square 2)+ 1)
```



Zastosowanie procedury do argumentów

Example

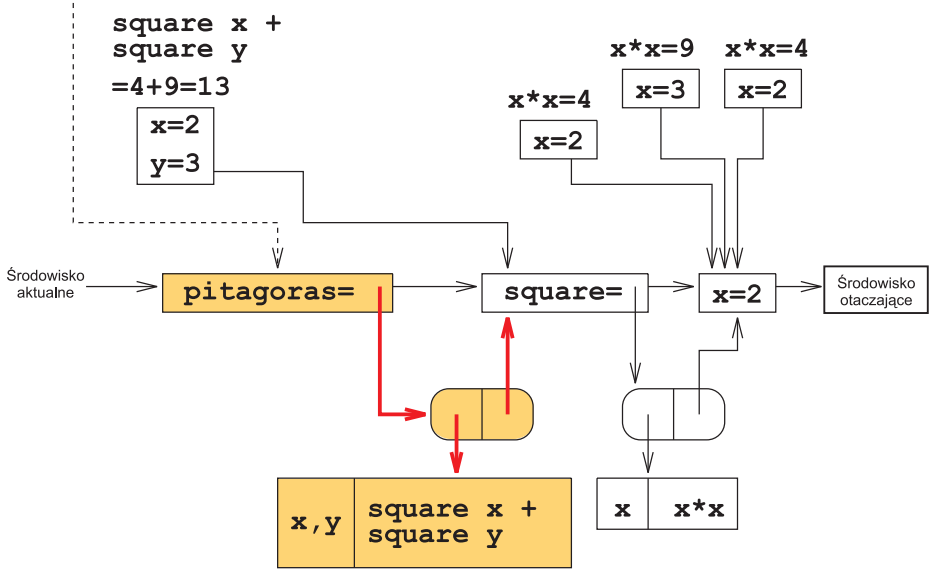
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1) =
13 + pitagoras 2 ((square 2)+ 1)
```



Zastosowanie procedury do argumentów

Example

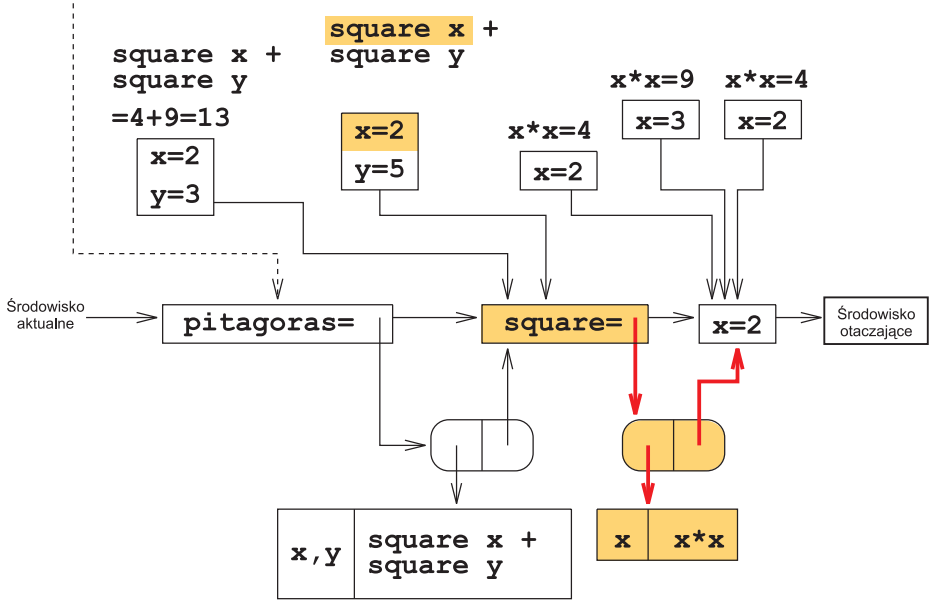
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1) =
13 + pitagoras 2 ((square 2)+ 1) =
13 + pitagoras 2 5
```



Zastosowanie procedury do argumentów

Example

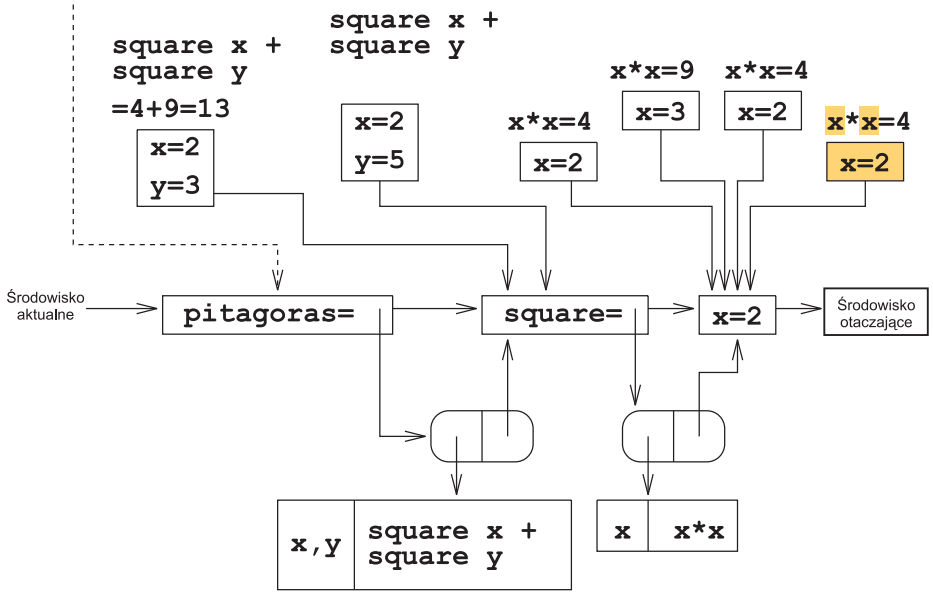
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1) =
13 + pitagoras 2 ((square 2)+ 1) =
13 + pitagoras 2 5
```



Zastosowanie procedury do argumentów

Example

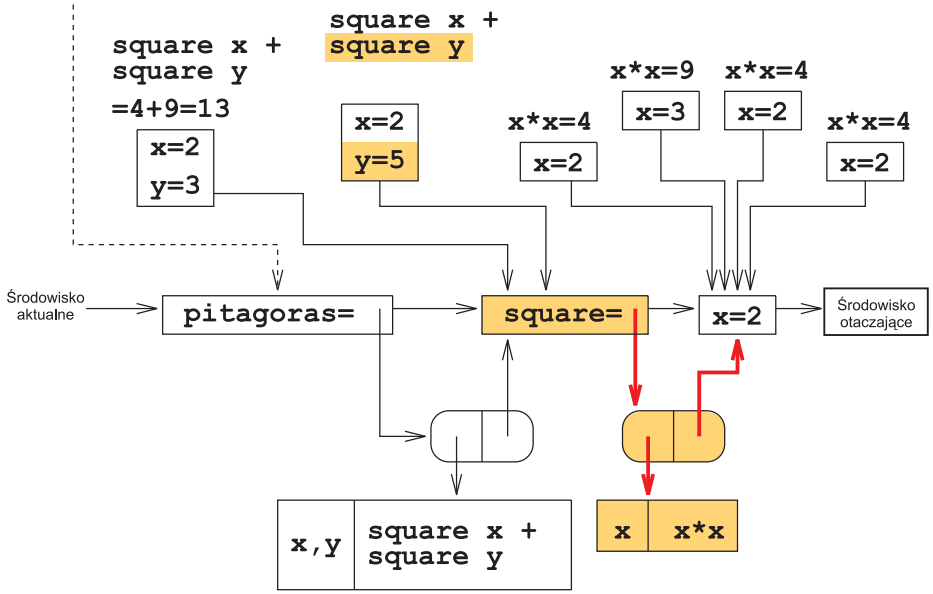
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1) =
13 + pitagoras 2 ((square 2)+ 1) =
13 + pitagoras 2 5
```



Zastosowanie procedury do argumentów

Example

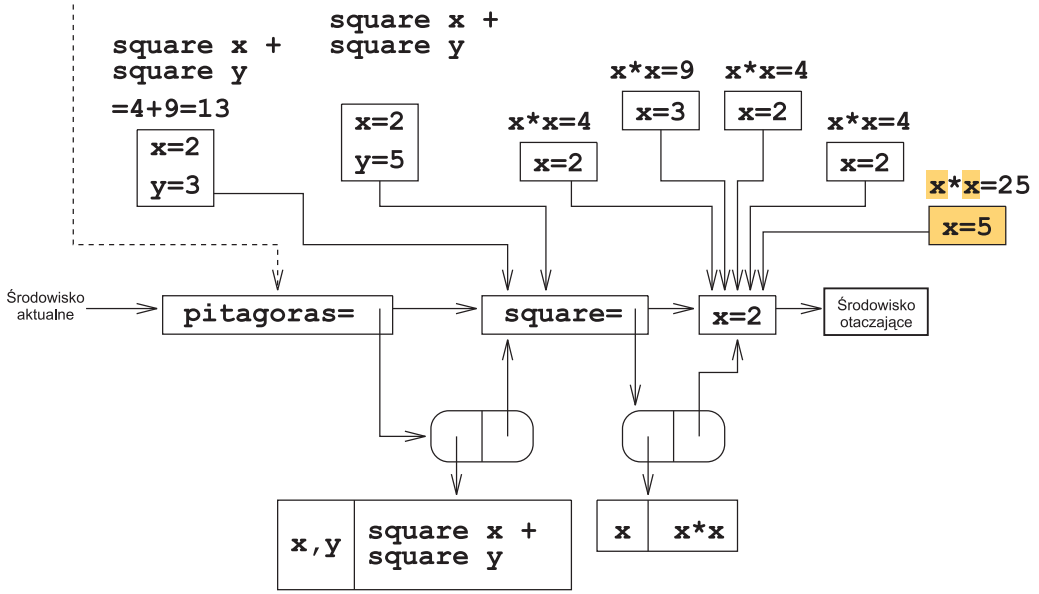
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1) =
13 + pitagoras 2 ((square 2)+ 1) =
13 + pitagoras 2 5
```



Zastosowanie procedury do argumentów

Example

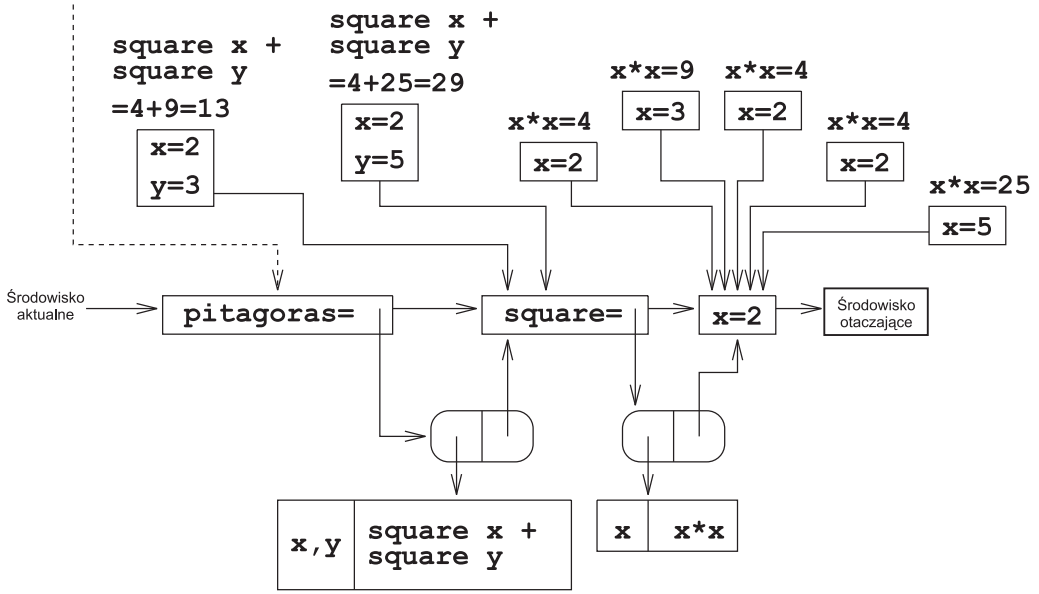
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1) =
13 + pitagoras 2 ((square 2)+ 1) =
13 + pitagoras 2 5
```



Zastosowanie procedury do argumentów

Example

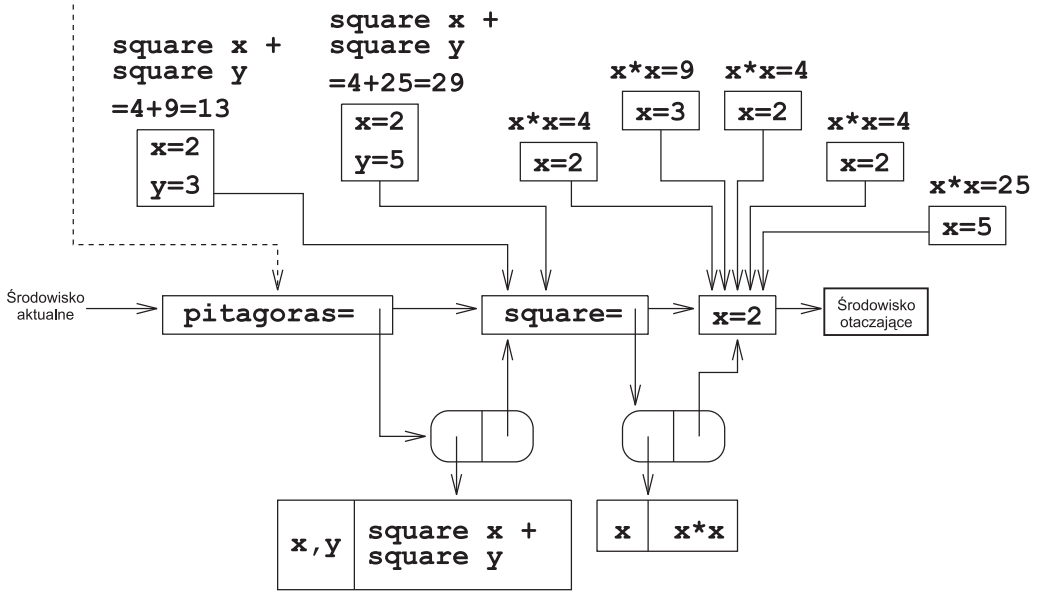
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1) =
13 + pitagoras 2 ((square 2)+ 1) =
13 + pitagoras 2 5
```



Zastosowanie procedury do argumentów

Example

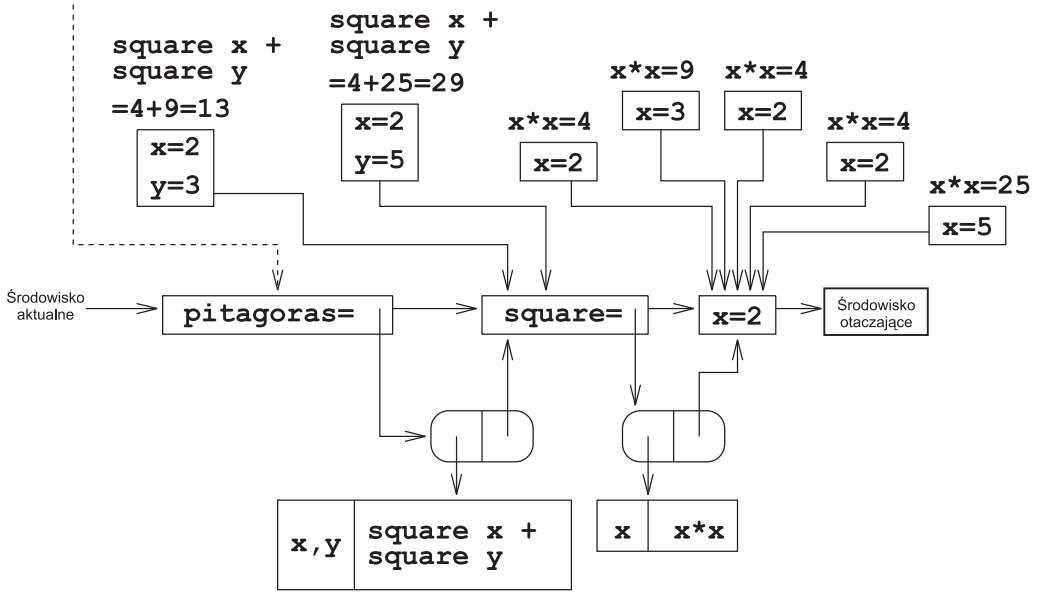
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1) =
13 + pitagoras 2 ((square 2)+ 1) =
13 + pitagoras 2 5  =
13 + 29
```



Zastosowanie procedury do argumentów

Example

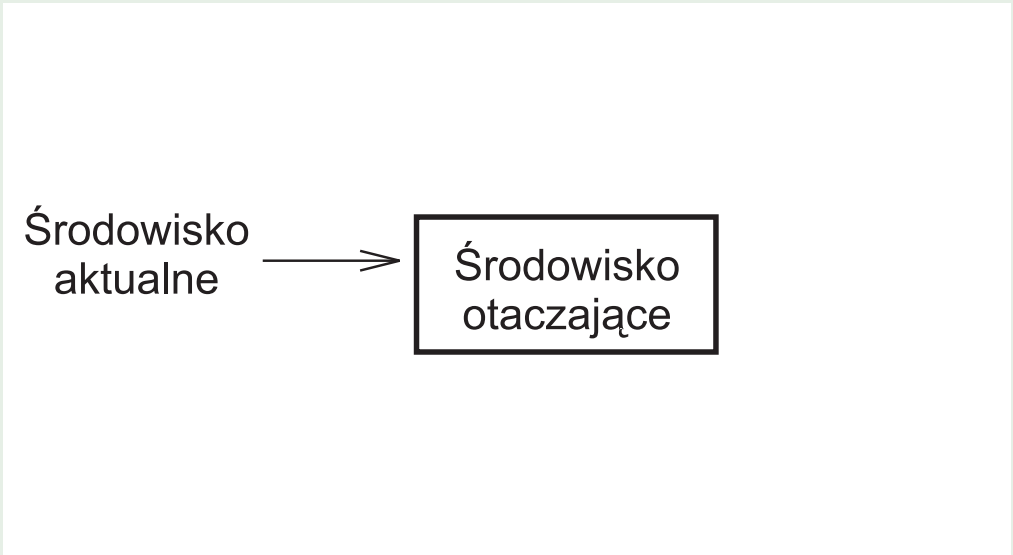
```
pitagoras x (x+1) + pitagoras x ((square x) + 1) =
pitagoras 2 3 + pitagoras 2 ((square 2) + 1) =
13 + pitagoras 2 ((square 2)+ 1) =
13 + pitagoras 2 5  =
13 + 29 =
42
```



Zastosowanie procedury do argumentów

Example

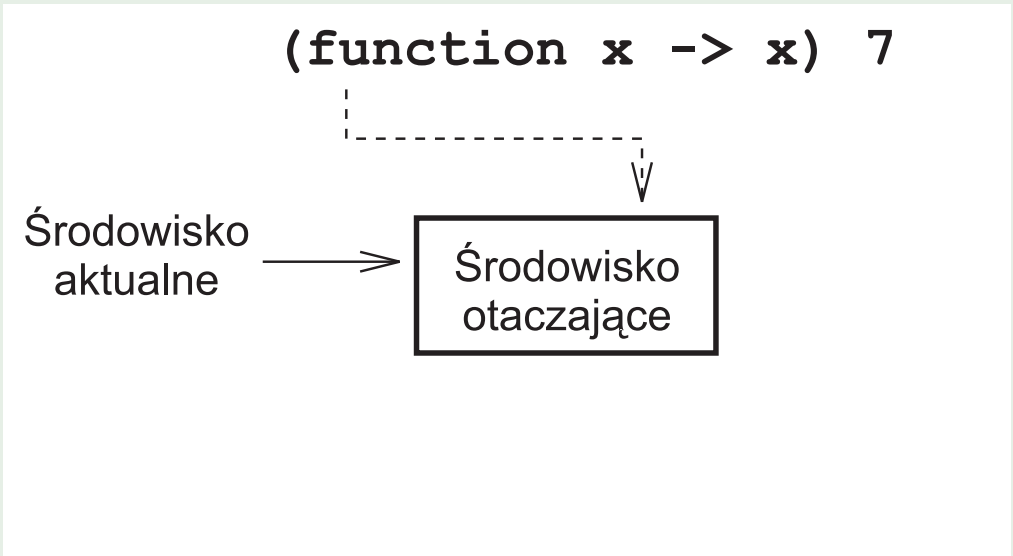
Obliczenie λ-abstrakcji:



Zastosowanie procedury do argumentów

Example

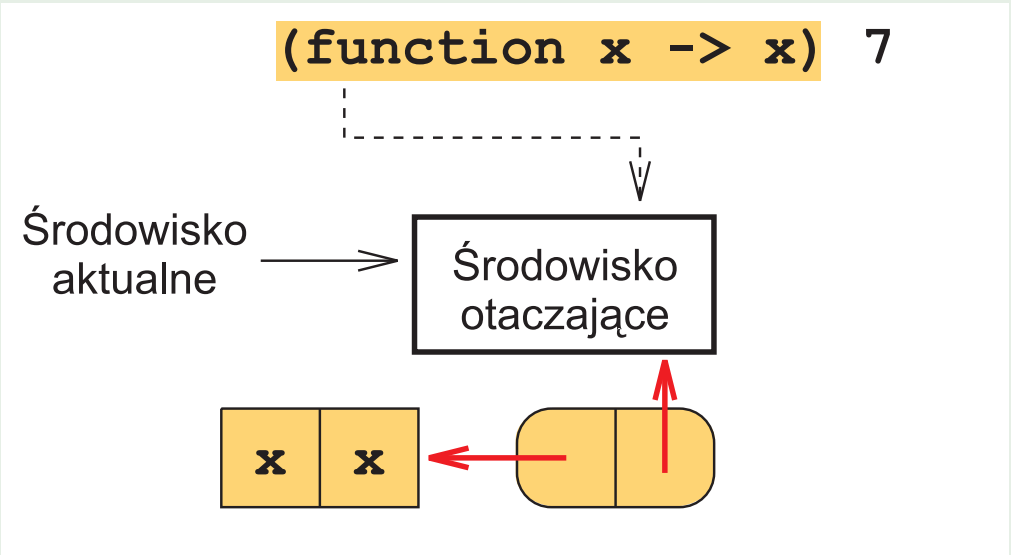
Obliczenie λ-abstrakcji:



Zastosowanie procedury do argumentów

Example

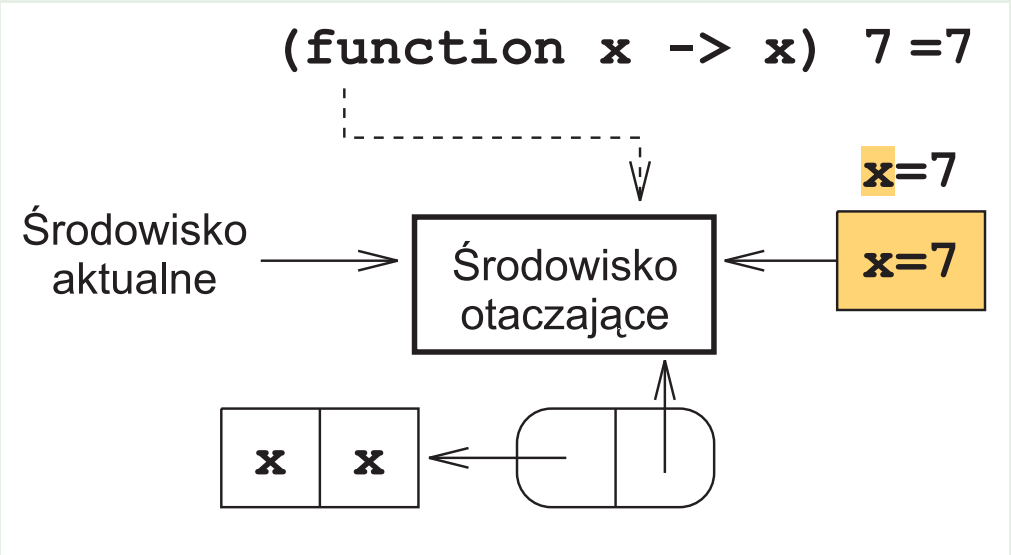
Obliczenie λ-abstrakcji:



Zastosowanie procedury do argumentów

Example

Obliczenie λ-abstrakcji:

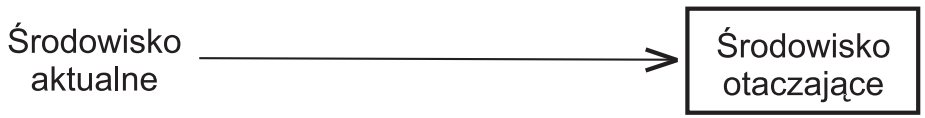


Zastosowanie procedury do argumentów

Example

Zastosowanie procedury rekurencyjnej:

```
let rec silnia n =
  if n < 2 then 1 else n * silnia (n - 1);;
silnia 3;;
```

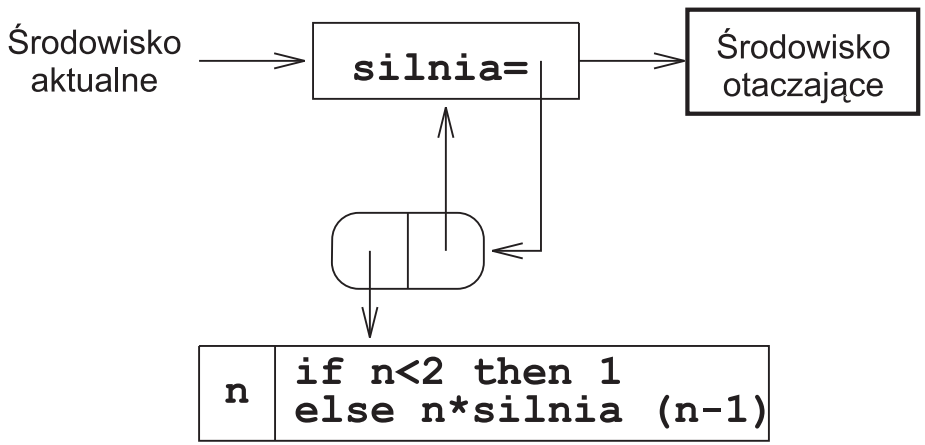


Zastosowanie procedury do argumentów

Example

Zastosowanie procedury rekurencyjnej:

```
let rec silnia n =
  if n < 2 then 1 else n * silnia (n - 1);;
silnia 3;;
```

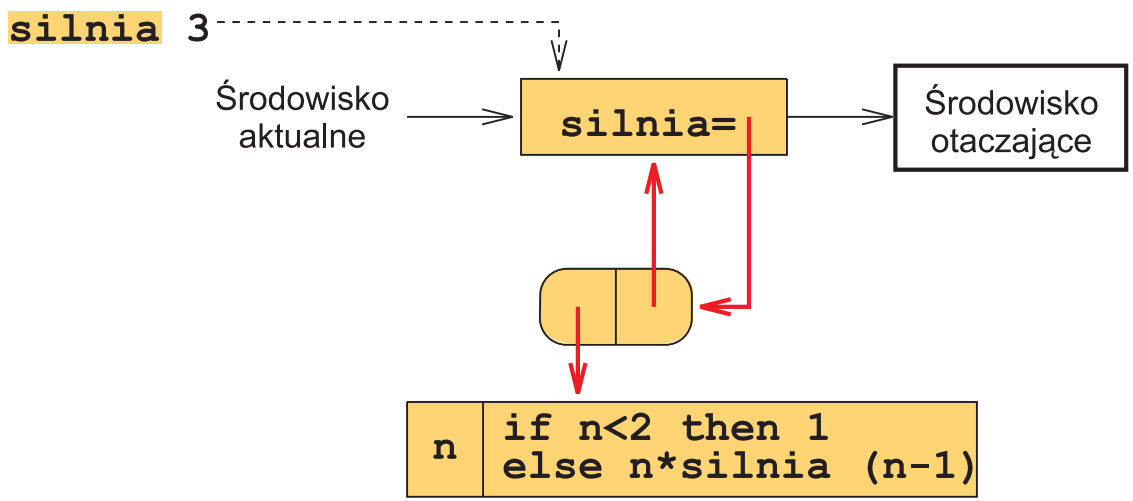


Zastosowanie procedury do argumentów

Example

Zastosowanie procedury rekurencyjnej:

```
let rec silnia n =
  if n < 2 then 1 else n * silnia (n - 1);;
silnia 3;;
```

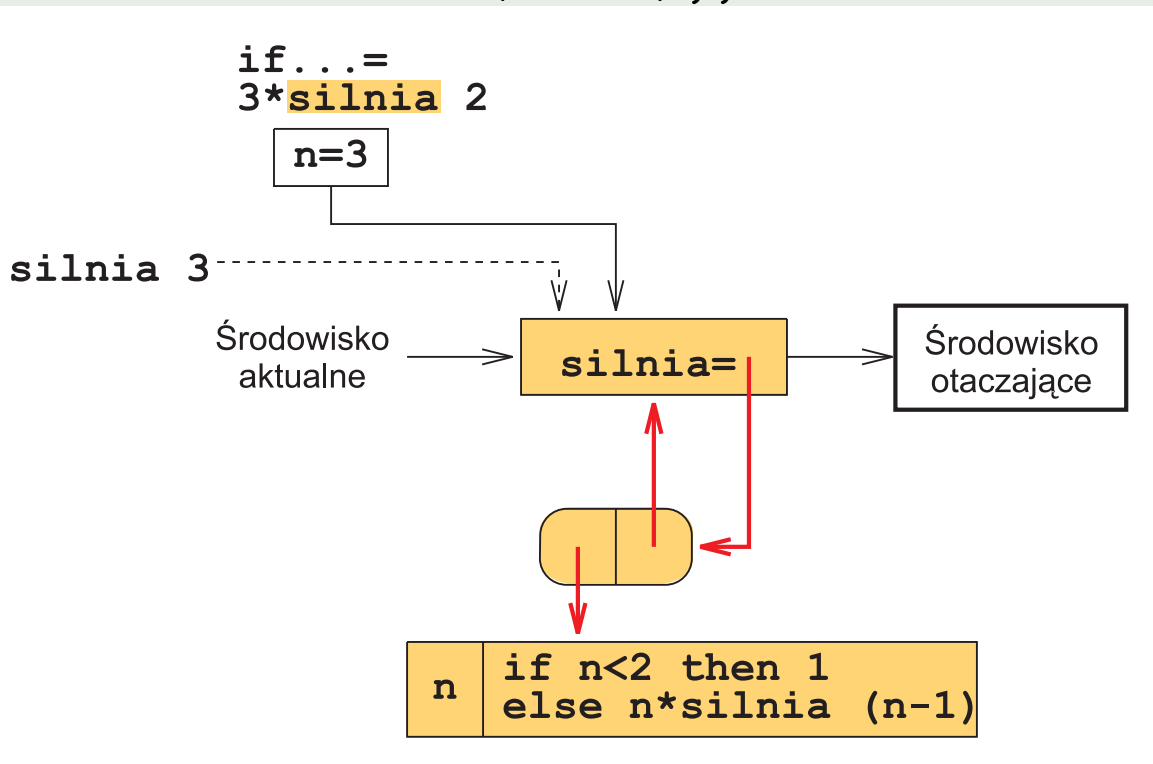


Zastosowanie procedury do argumentów

Example

Zastosowanie procedury rekurencyjnej:

```
let rec silnia n =
  if n < 2 then 1 else n * silnia (n - 1);;
silnia 3;;
```

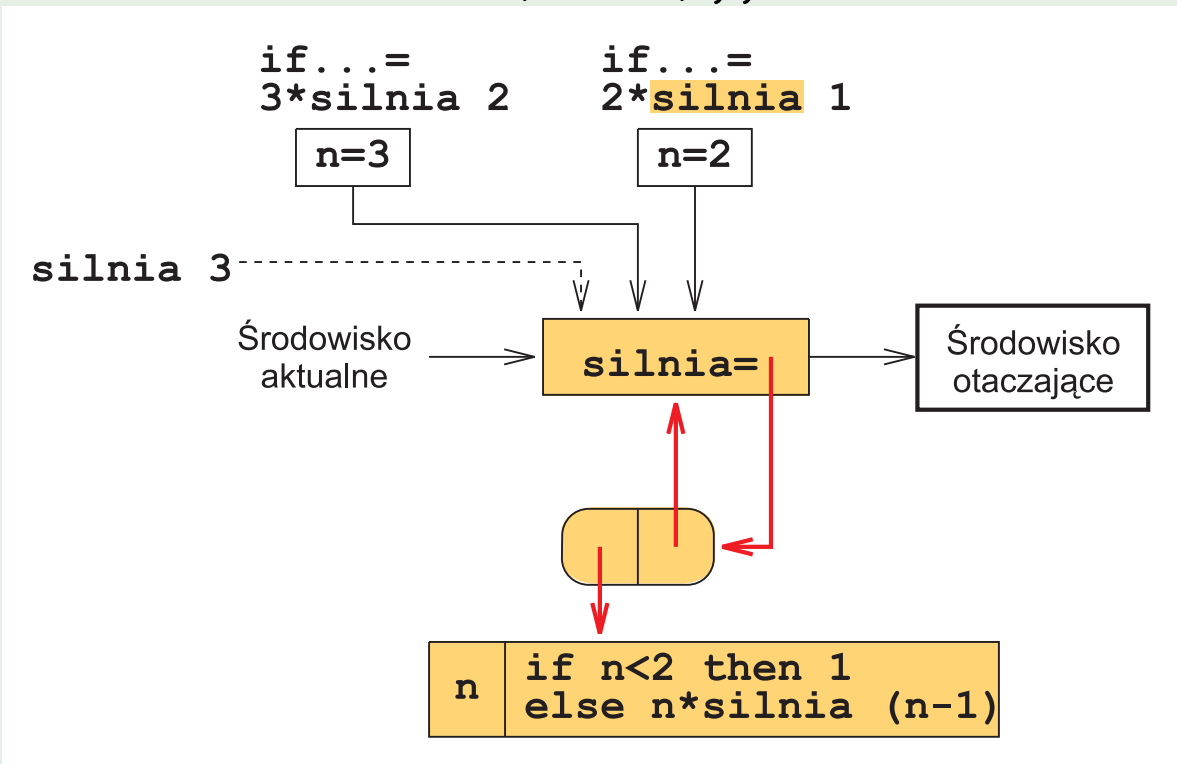


Zastosowanie procedury do argumentów

Example

Zastosowanie procedury rekurencyjnej:

```
let rec silnia n =
  if n < 2 then 1 else n * silnia (n - 1);;
silnia 3;;
```

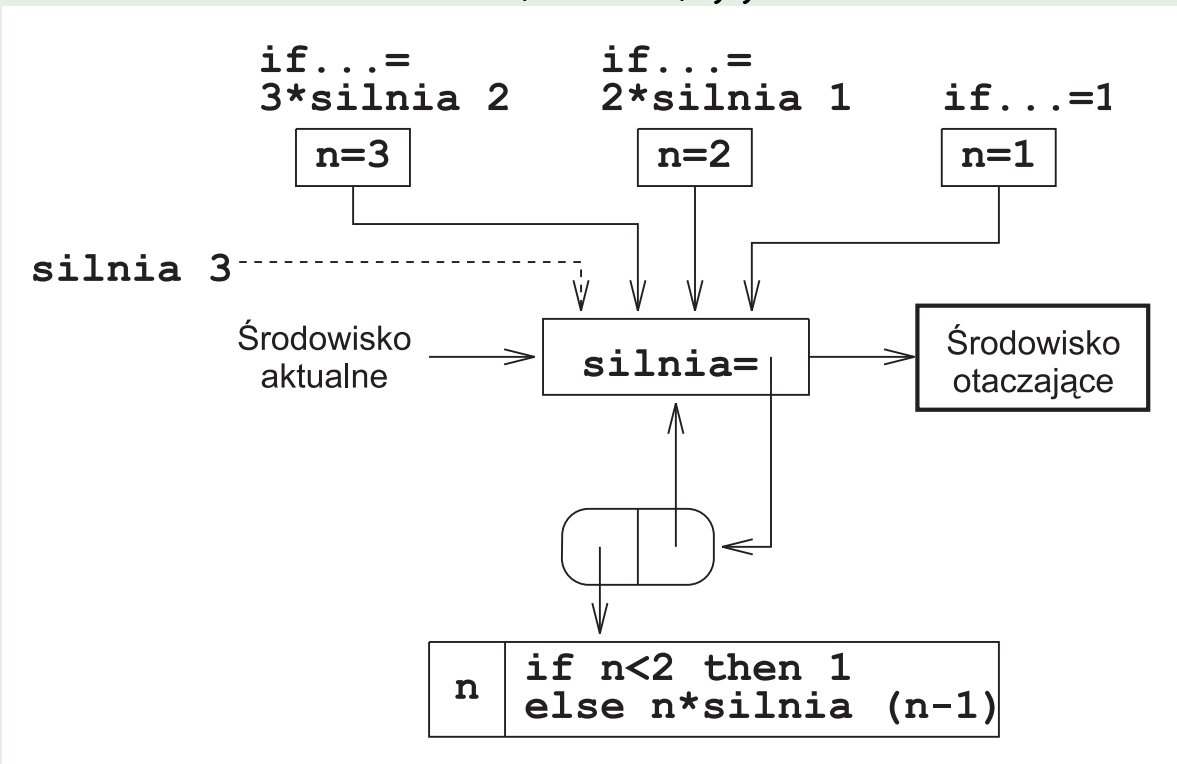


Zastosowanie procedury do argumentów

Example

Zastosowanie procedury rekurencyjnej:

```
let rec silnia n =
  if n < 2 then 1 else n * silnia (n - 1);;
silnia 3;;
```

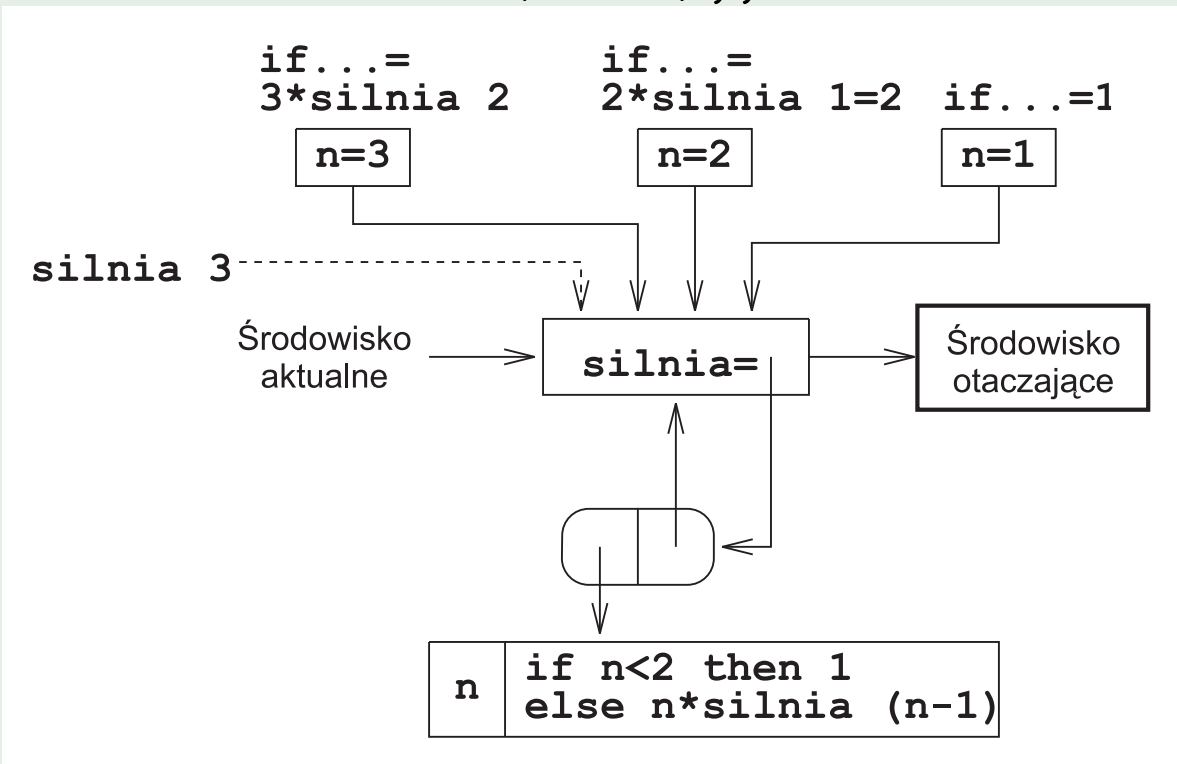


Zastosowanie procedury do argumentów

Example

Zastosowanie procedury rekurencyjnej:

```
let rec silnia n =
  if n < 2 then 1 else n * silnia (n - 1);;
silnia 3;;
```

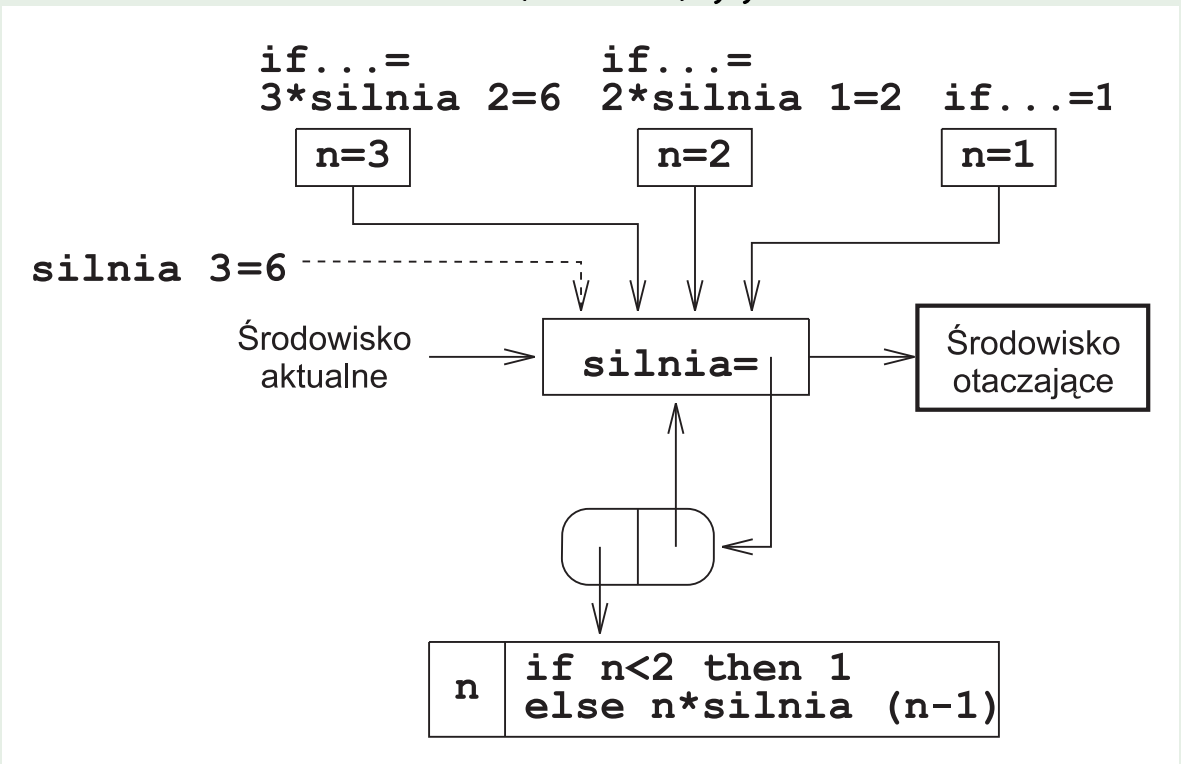


Zastosowanie procedury do argumentów

Example

Zastosowanie procedury rekurencyjnej:

```
let rec silnia n =
  if n < 2 then 1 else n * silnia (n - 1);;
silnia 3;;
```



Rekurencja ogonowa

Rekurencja ogonowa:

- Wartości zwracane przez wszystkie wywołania rekurencyjne, bez żadnego przetwarzania, są przekazywane jako wynik danej procedury.
- W momencie wywołania rekurencyjnego nie musimy tworzyć nowej ramki.
- Zmieniają się jedynie wartości argumentów w ramce.
- Mniejszy koszt pamięciowy — tylko jedna ramka.

Jak przekształcić rekurencję w ogonową:

- Rekurencyjna procedura pomocnicza.
- Dodatkowe argumenty — akumulator.

Rekurencja ogonowa

Rekurencja ogonowa:

- Wartości zwracane przez wszystkie wywołania rekurencyjne, bez żadnego przetwarzania, są przekazywane jako wynik danej procedury.
- W momencie wywołania rekurencyjnego nie musimy tworzyć nowej ramki.
- Zmieniają się jedynie wartości argumentów w ramce.
- Mniejszy koszt pamięciowy — tylko jedna ramka.

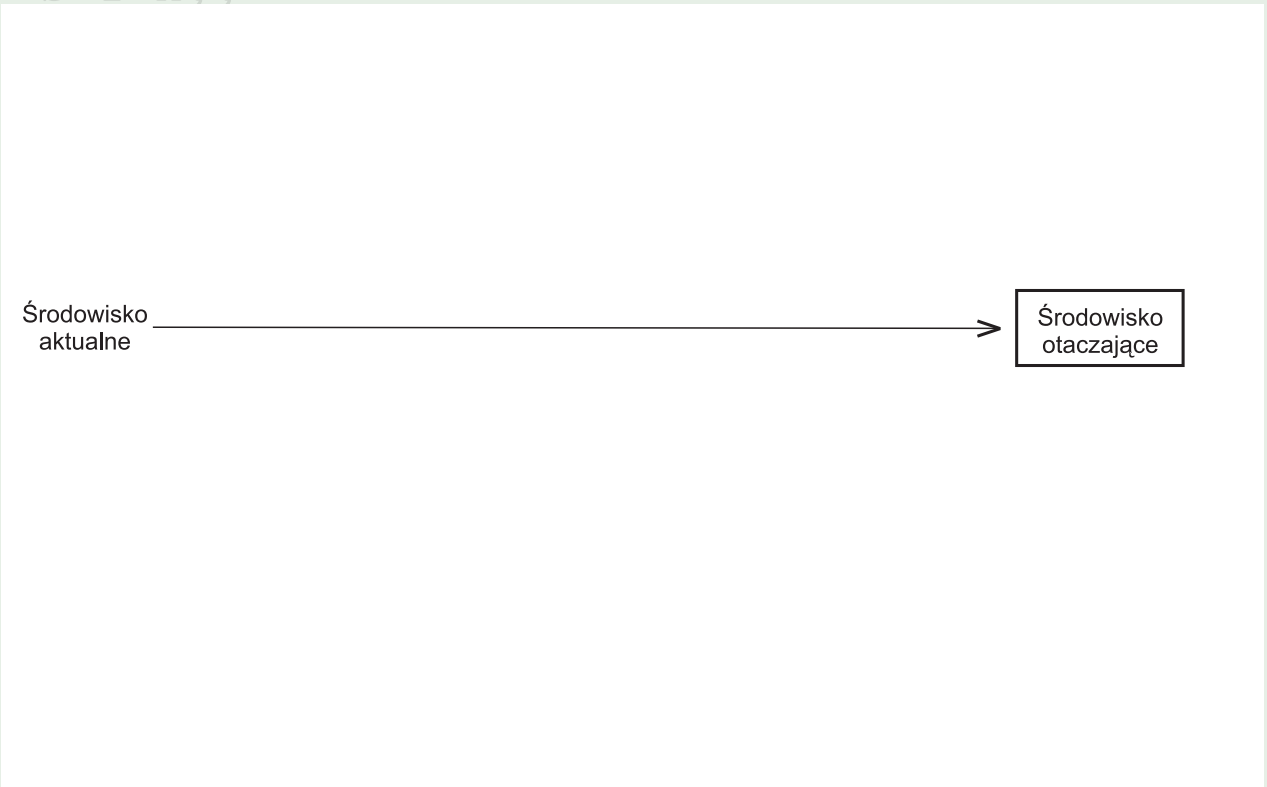
Jak przekształcić rekurencję w ogonową:

- Rekurencyjna procedura pomocnicza.
- Dodatkowe argumenty — akumulator.

Rekurencja ogonowa

Example (Silnia ogonowo)

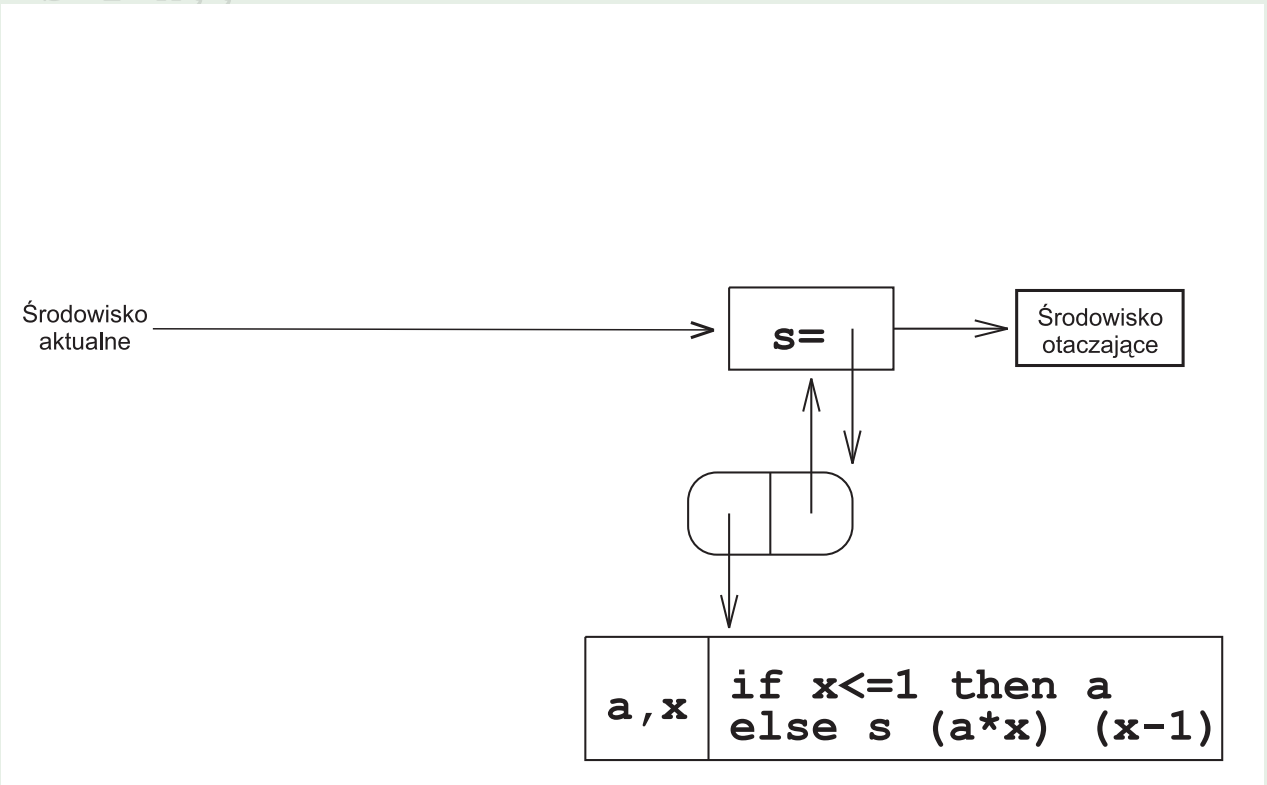
```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

Example (Silnia ogonowo)

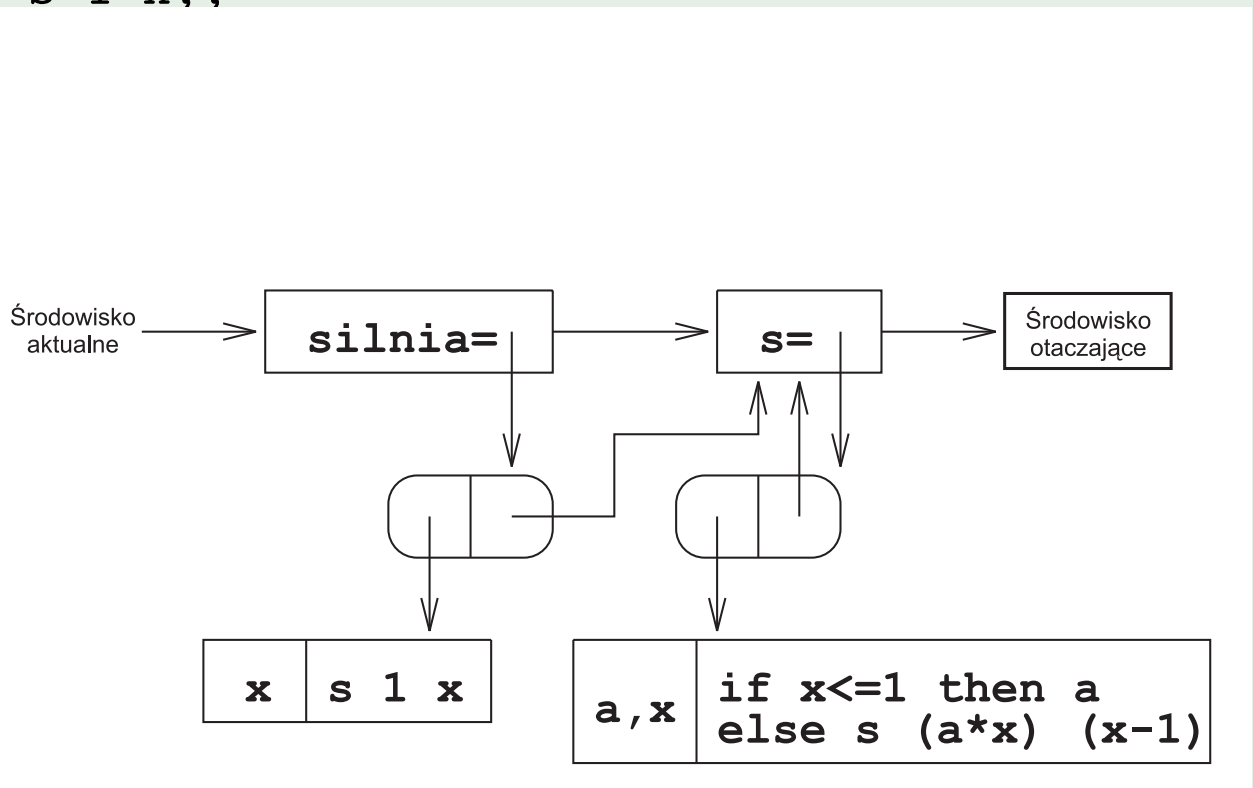
```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

Example (Silnia ogonowo)

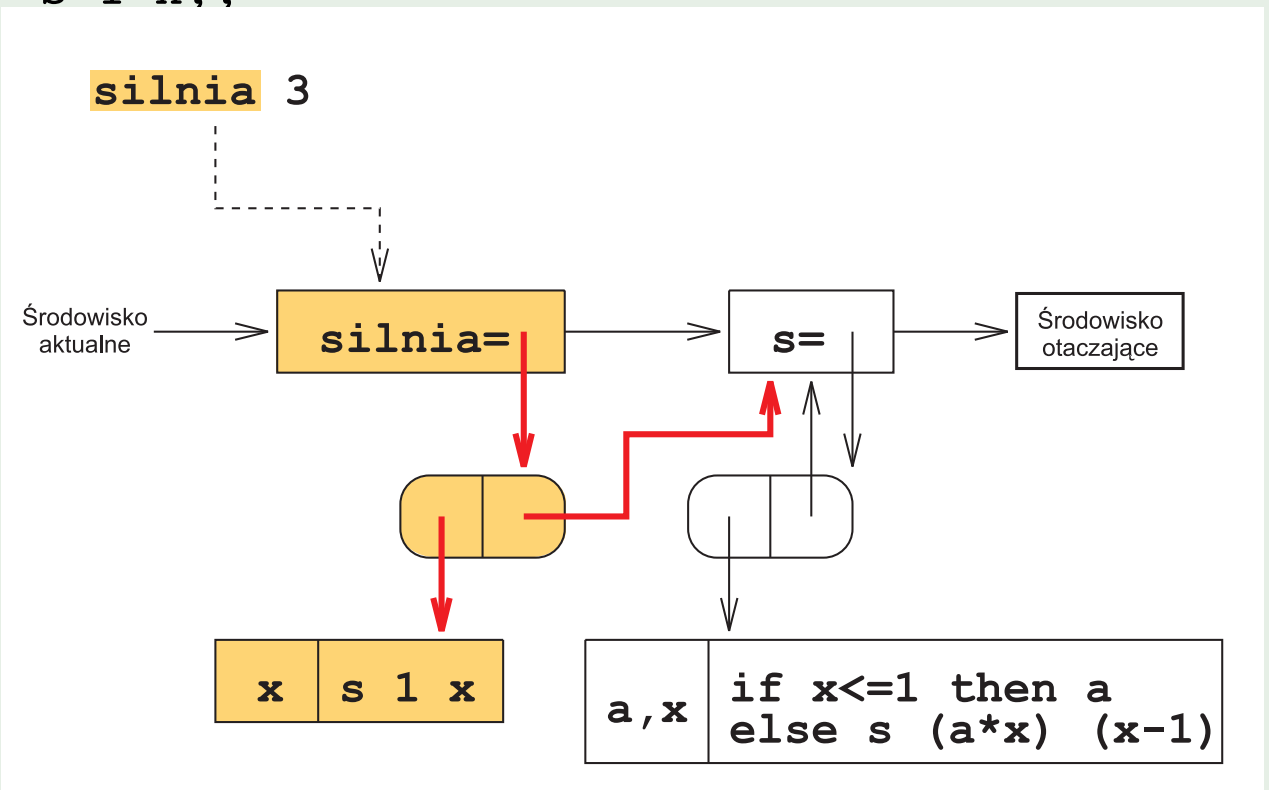
```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

Example (Silnia ogonowo)

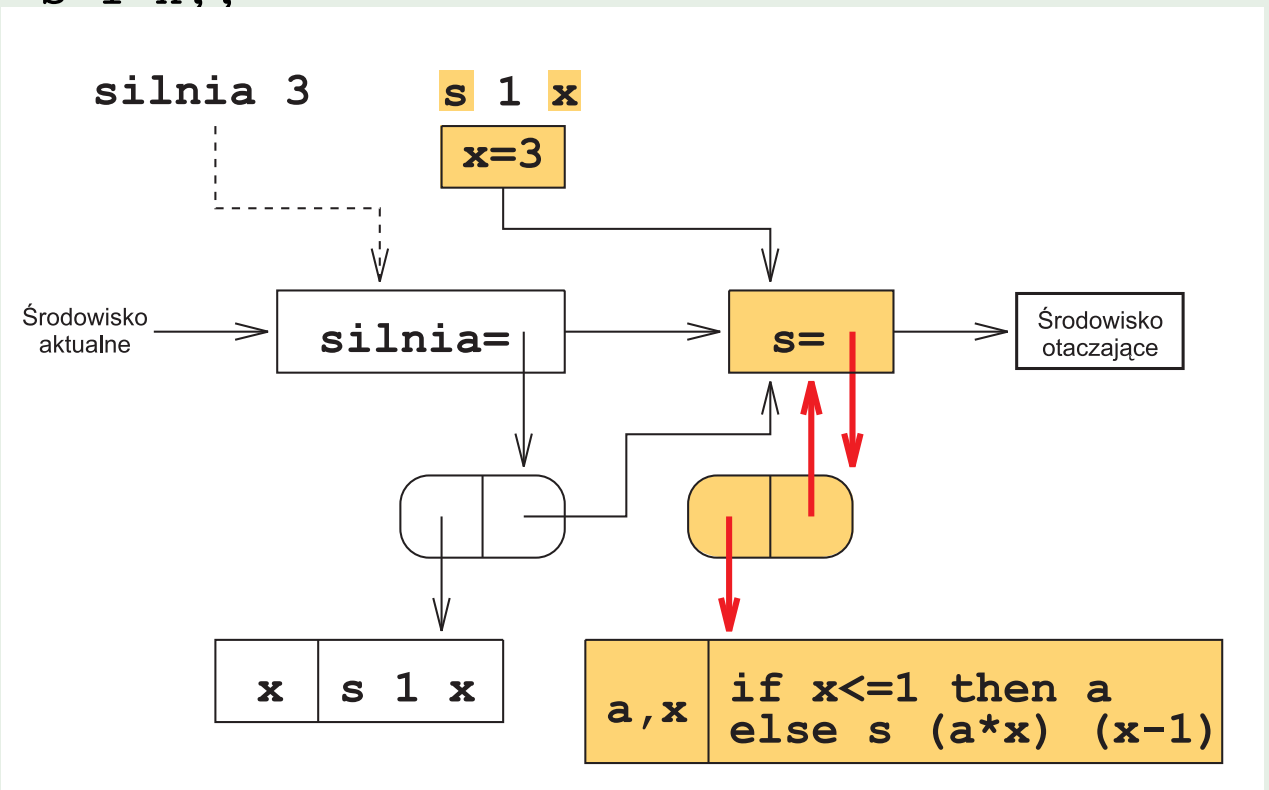
```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

Example (Silnia ogonowo)

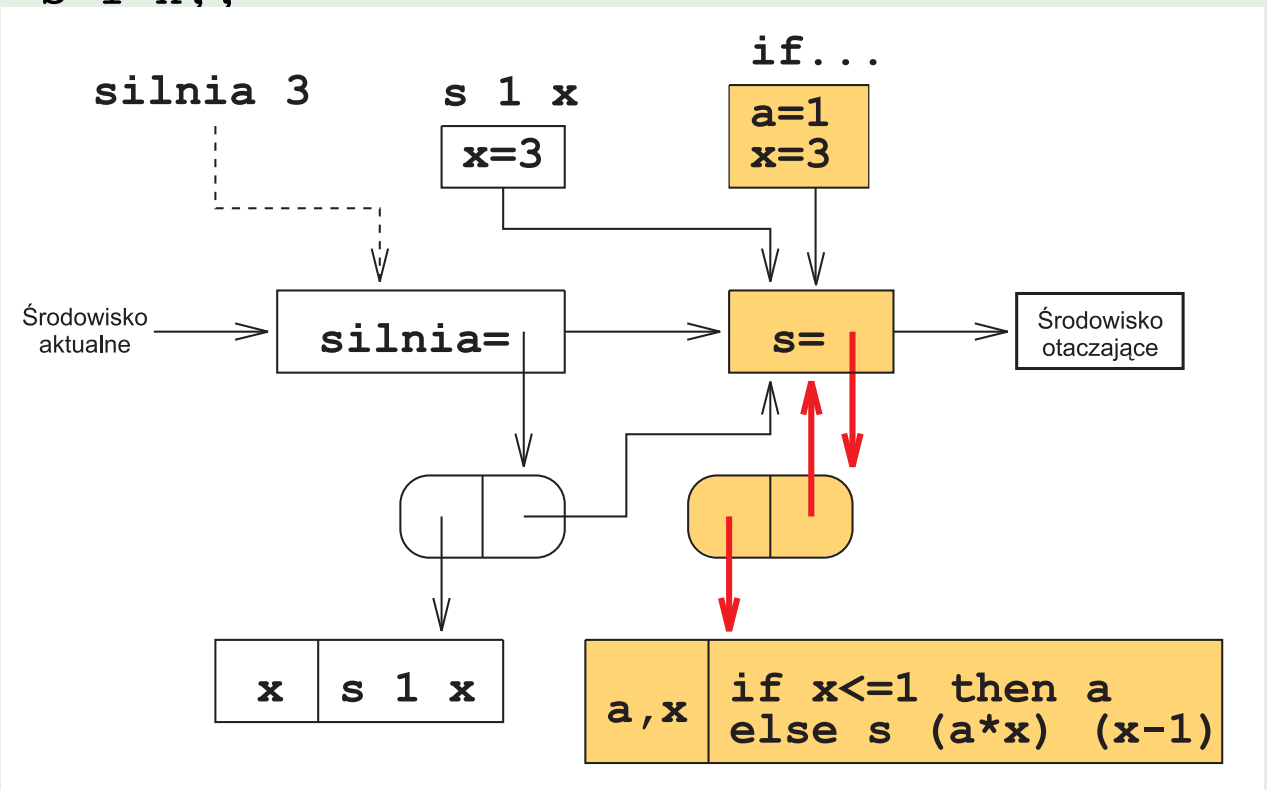
```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

Example (Silnia ogonowo)

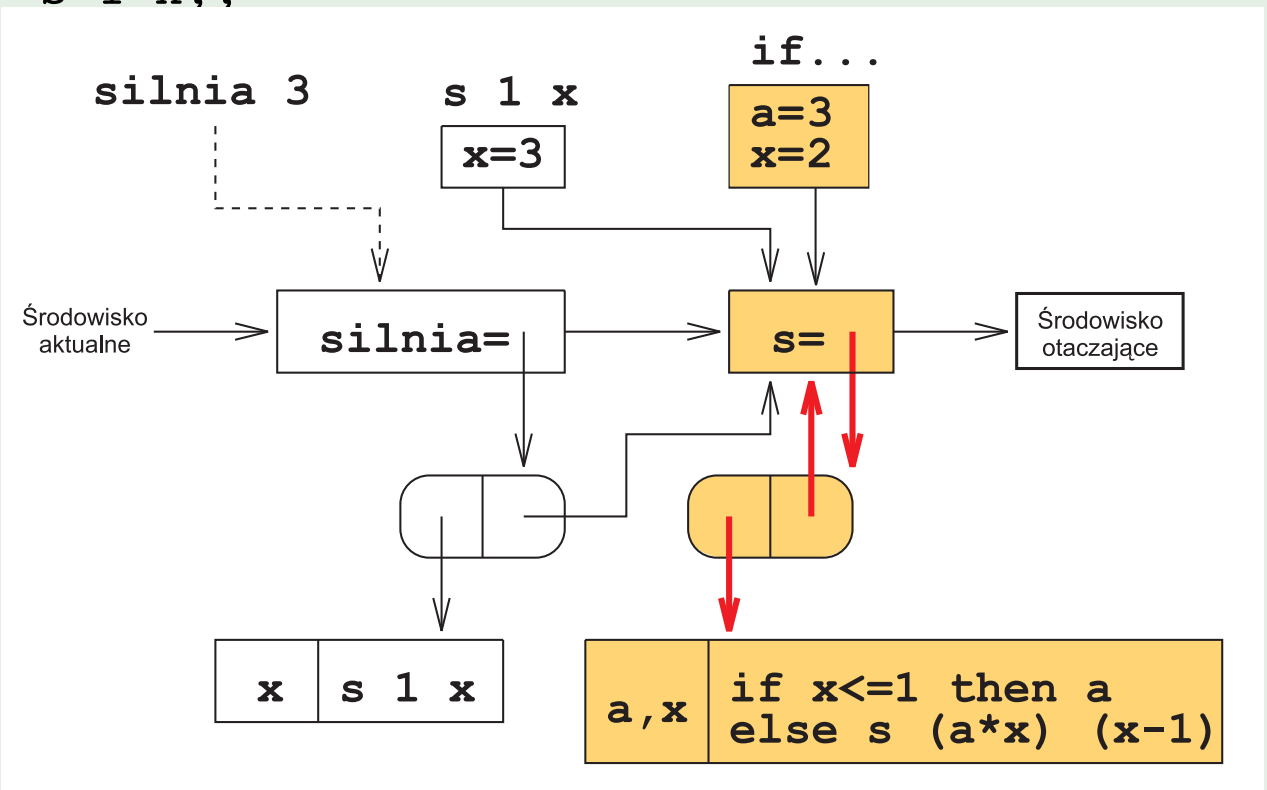
```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

Example (Silnia ogonowo)

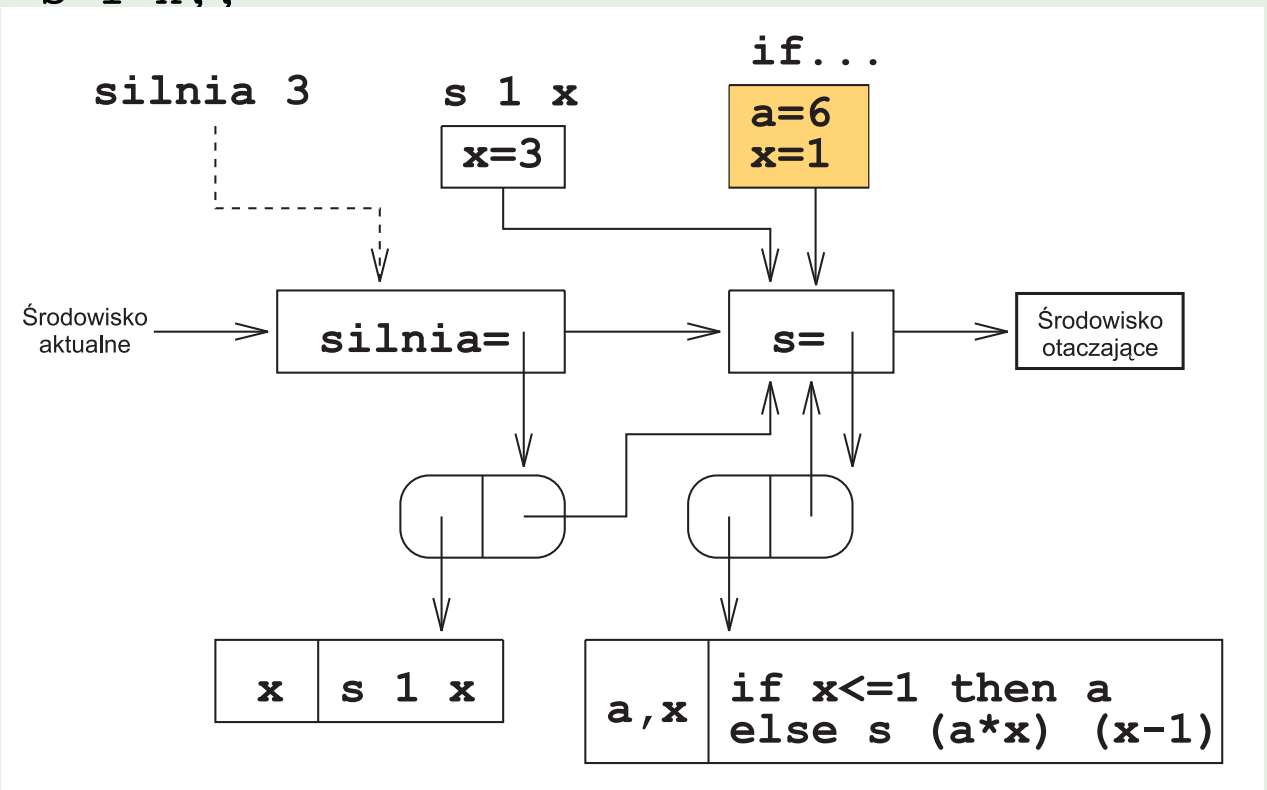
```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

Example (Silnia ogonowo)

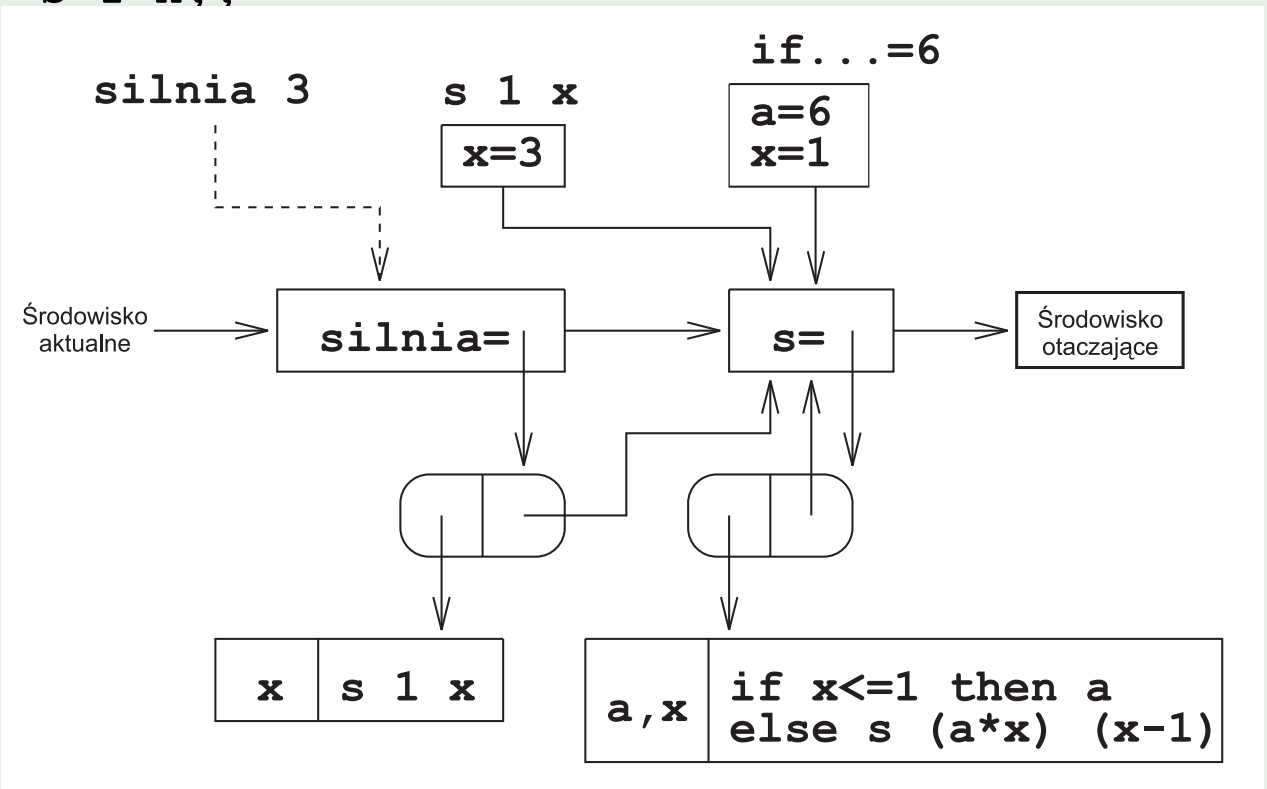
```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

Example (Silnia ogonowo)

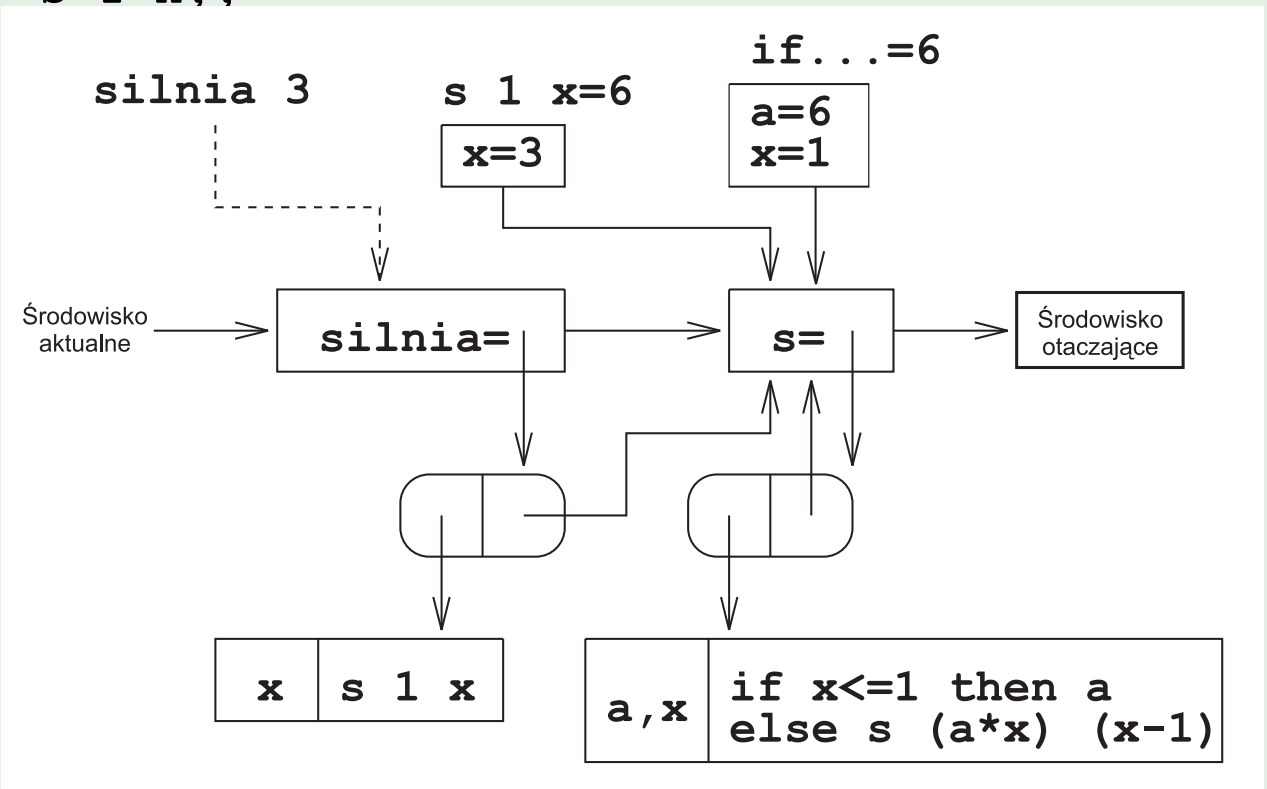
```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

Example (Silnia ogonowo)

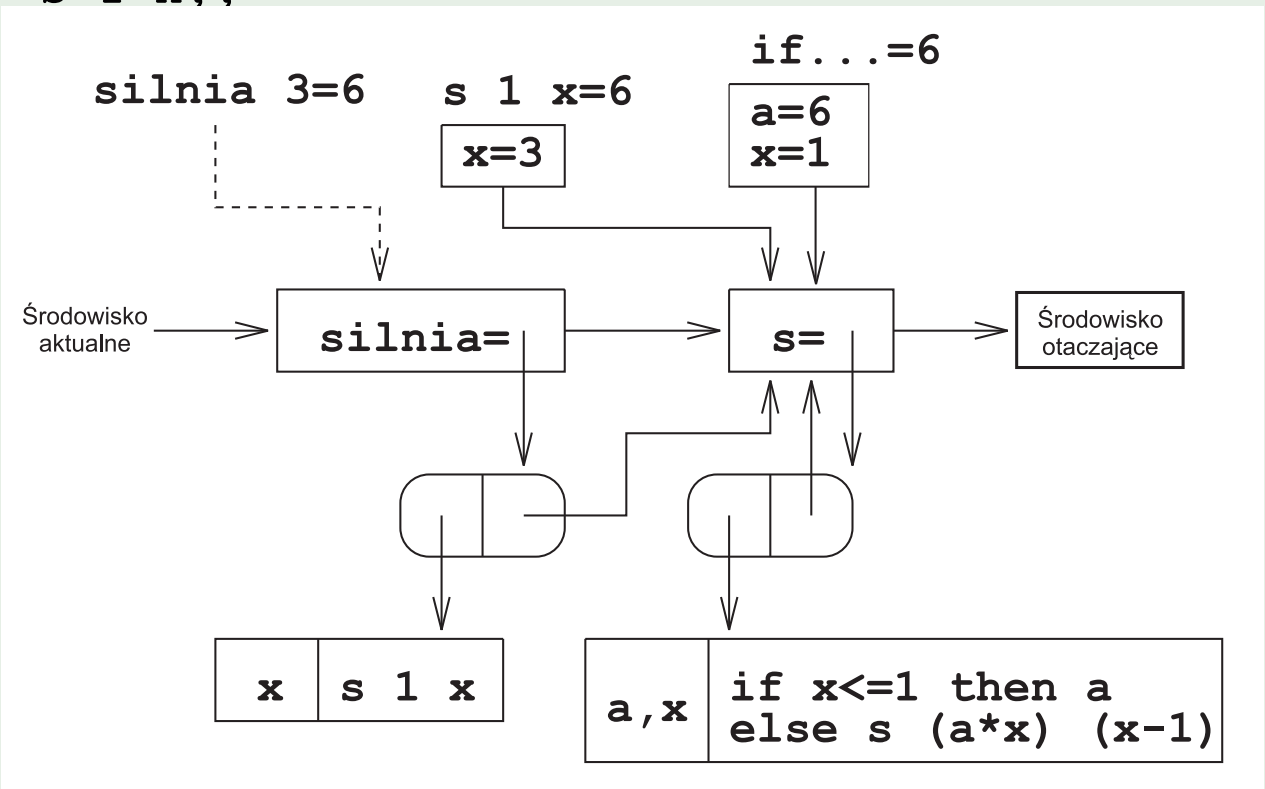
```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

Example (Silnia ogonowo)

```
let rec s a x =
  if x <= 1 then a else s (a * x) (x - 1);;
let silnia x = s 1 x;;
silnia 3;;
```



Rekurencja ogonowa

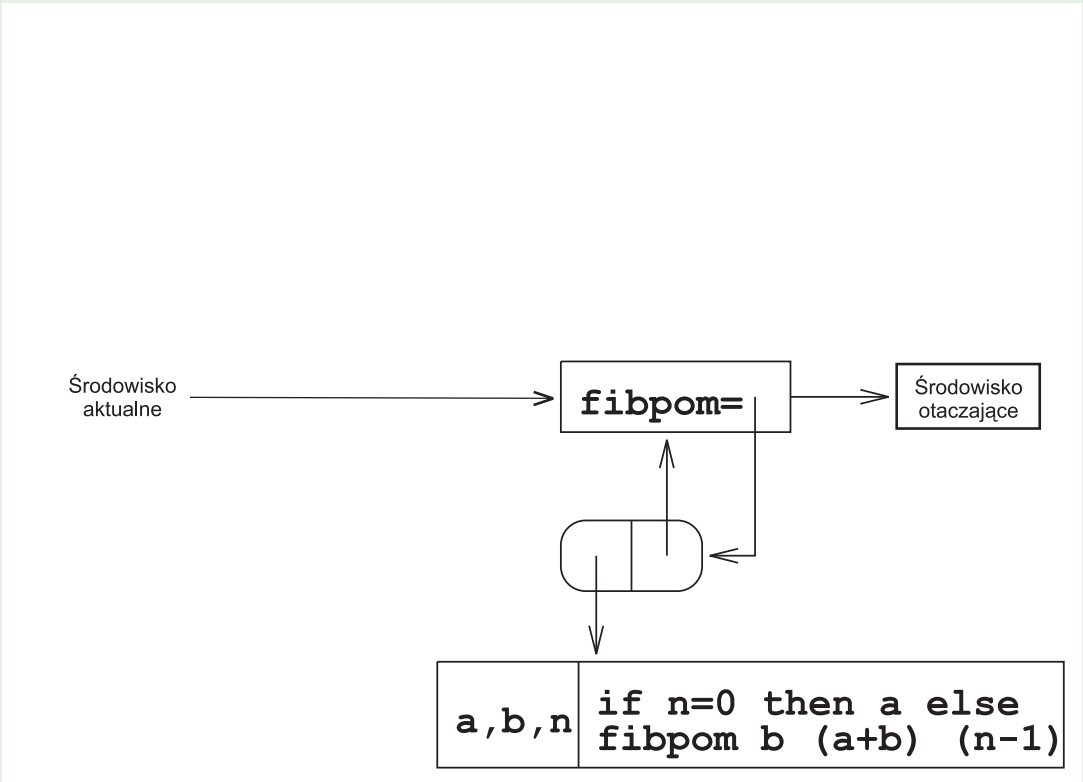
Example (Liczby Fibonacciego ogonowo)

```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```

Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

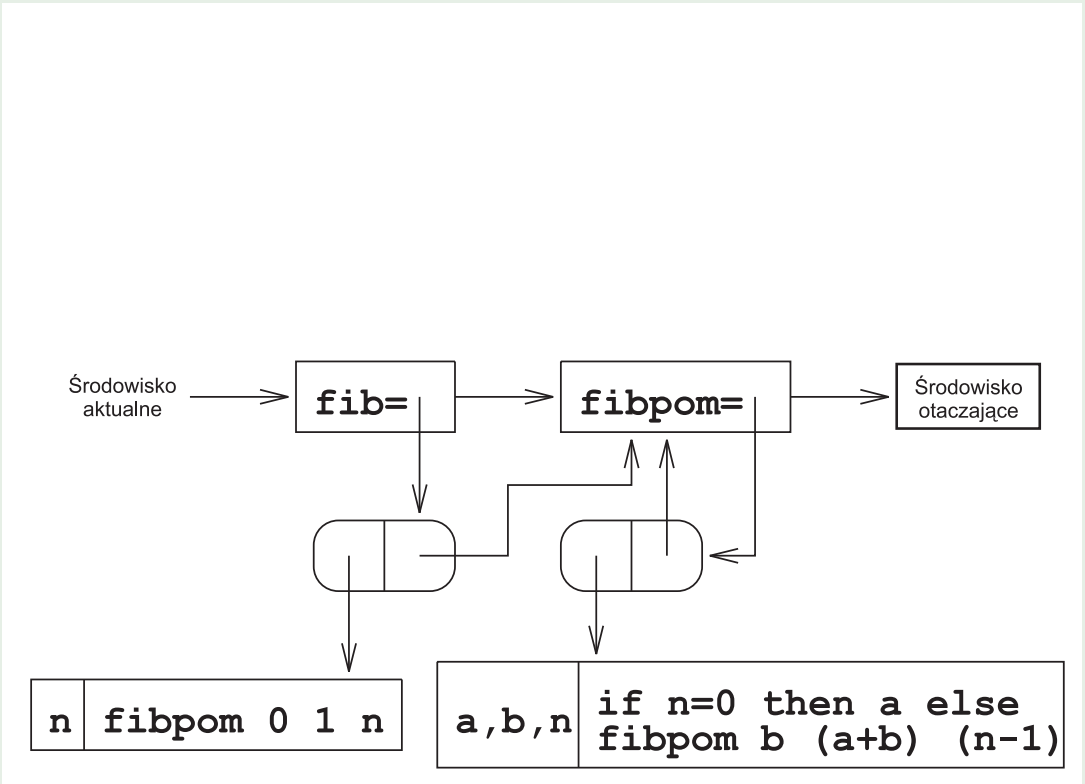
```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

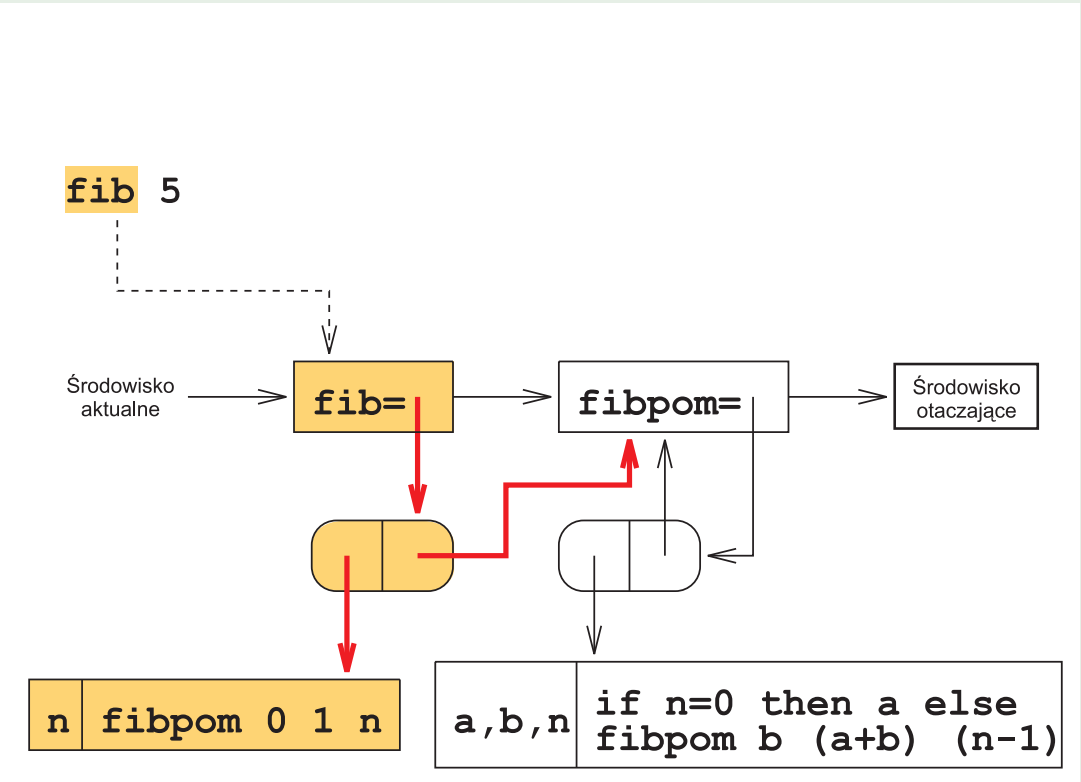
```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

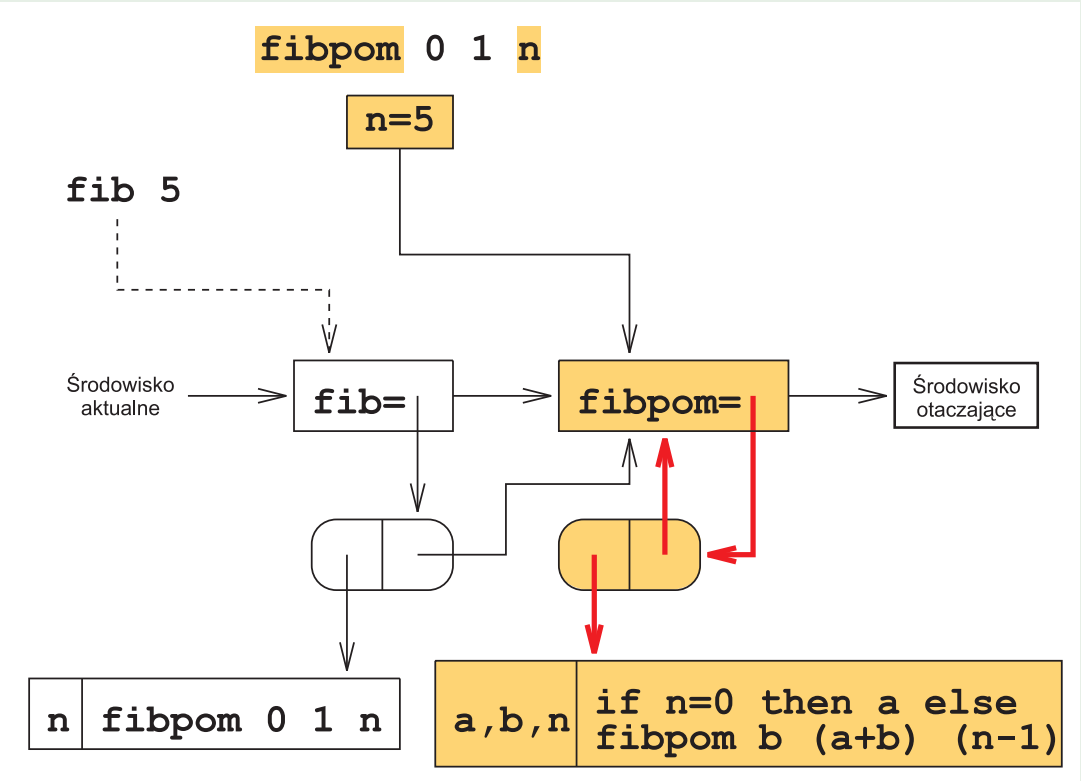
```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

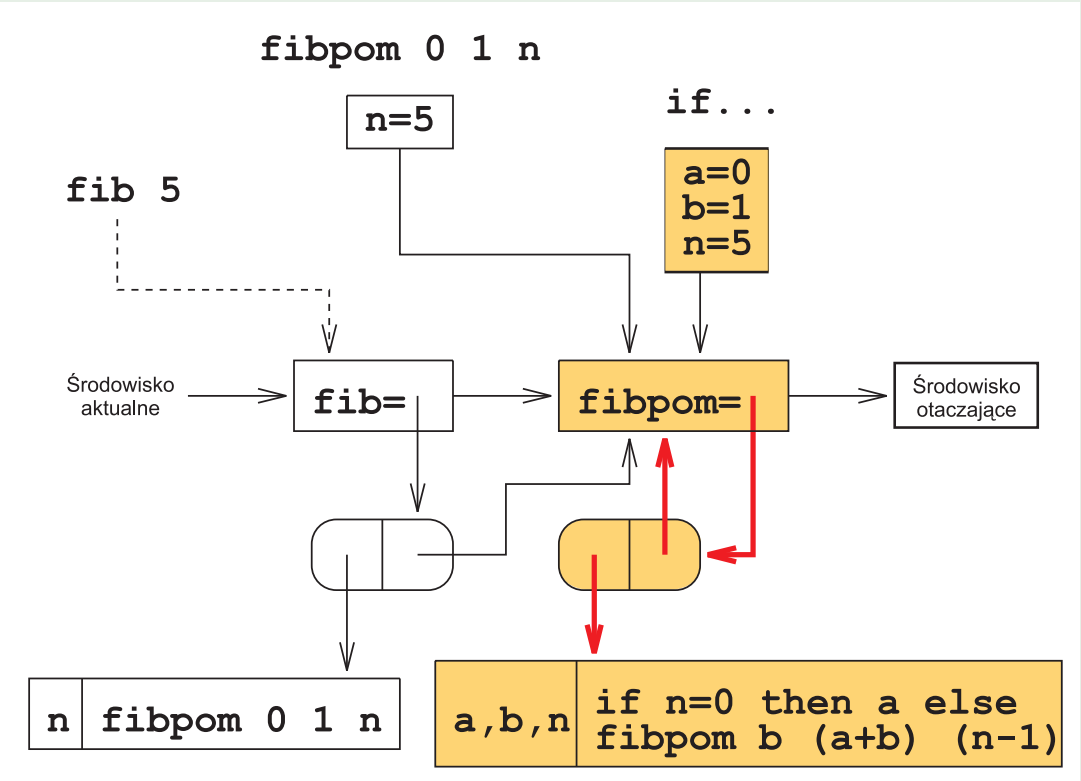
```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

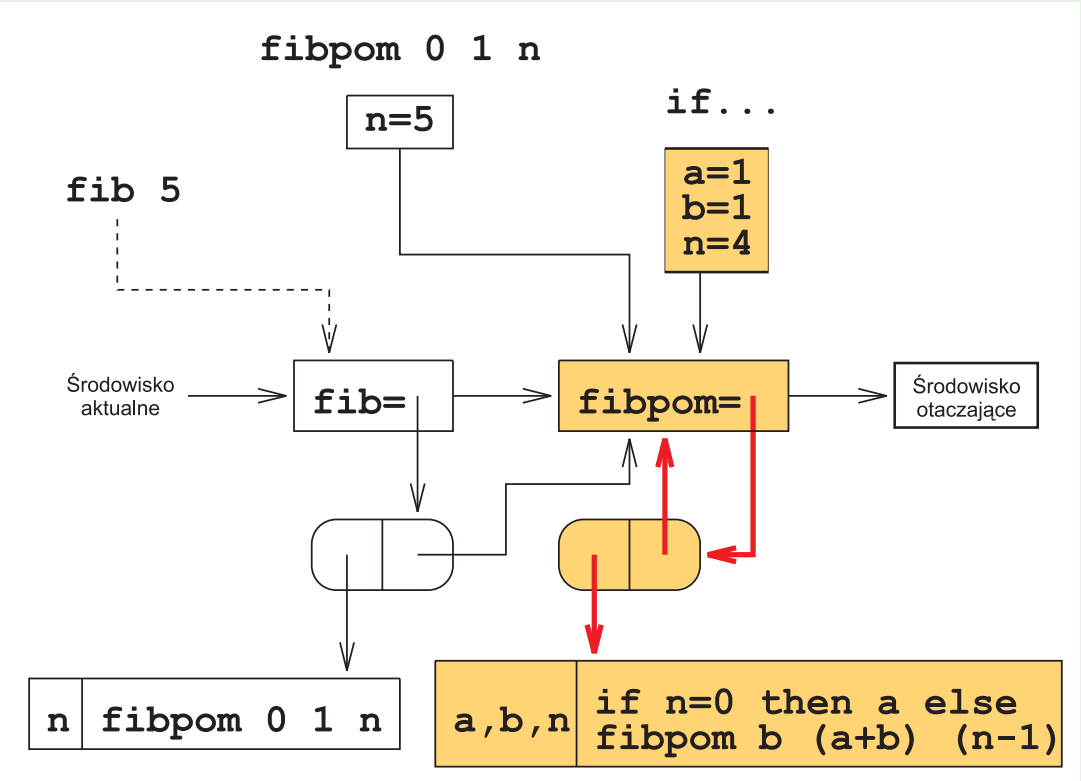
```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

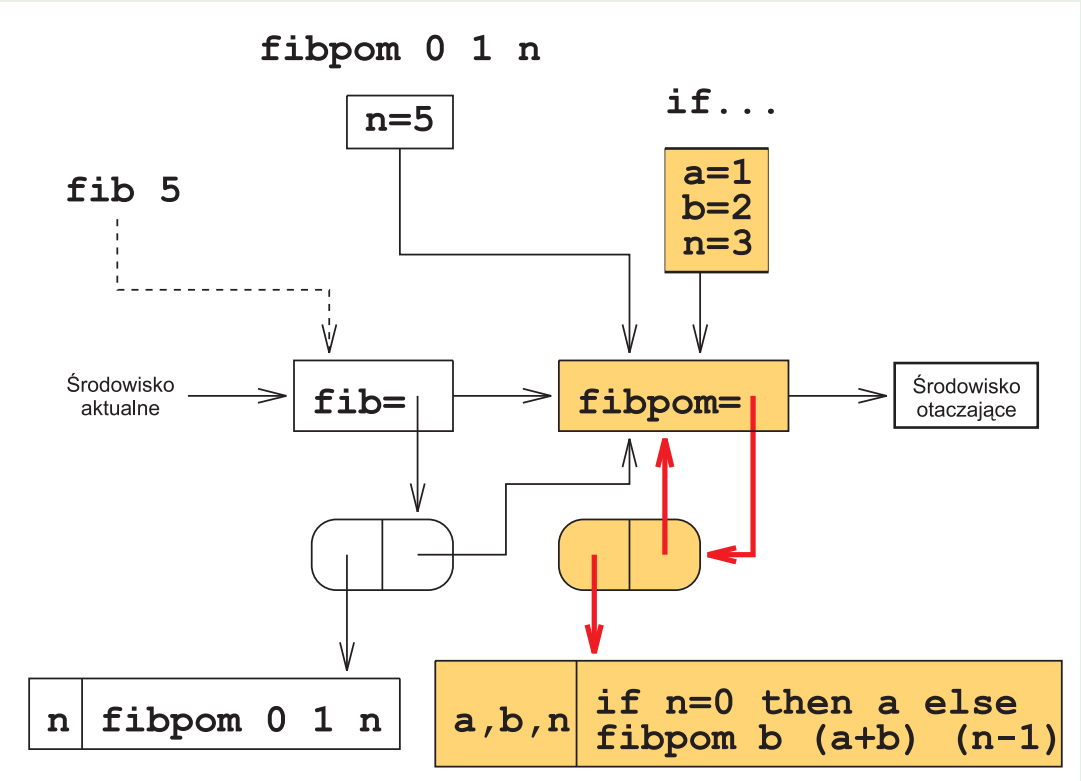
```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

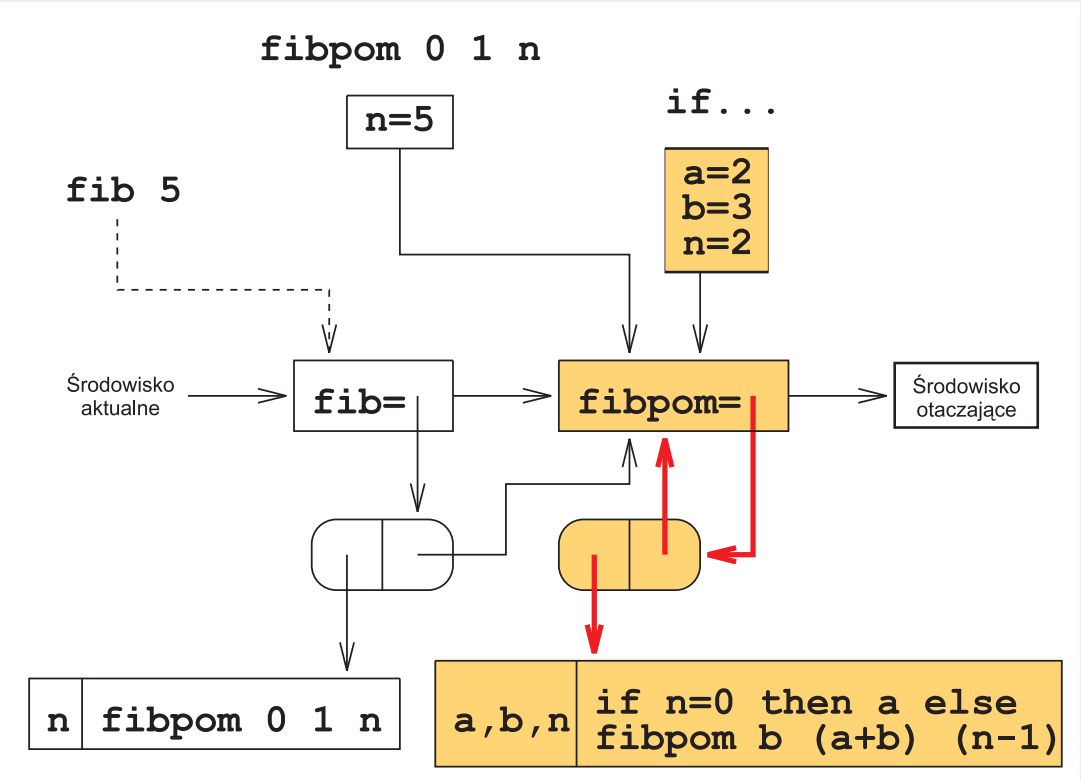
```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

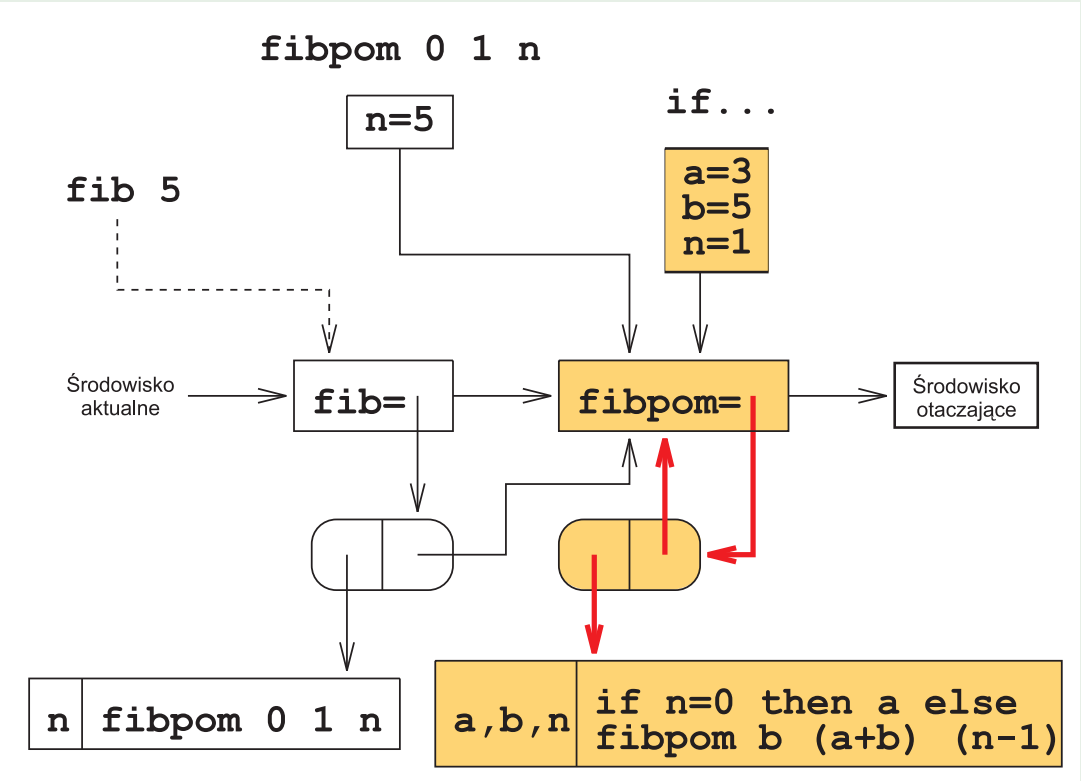
```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

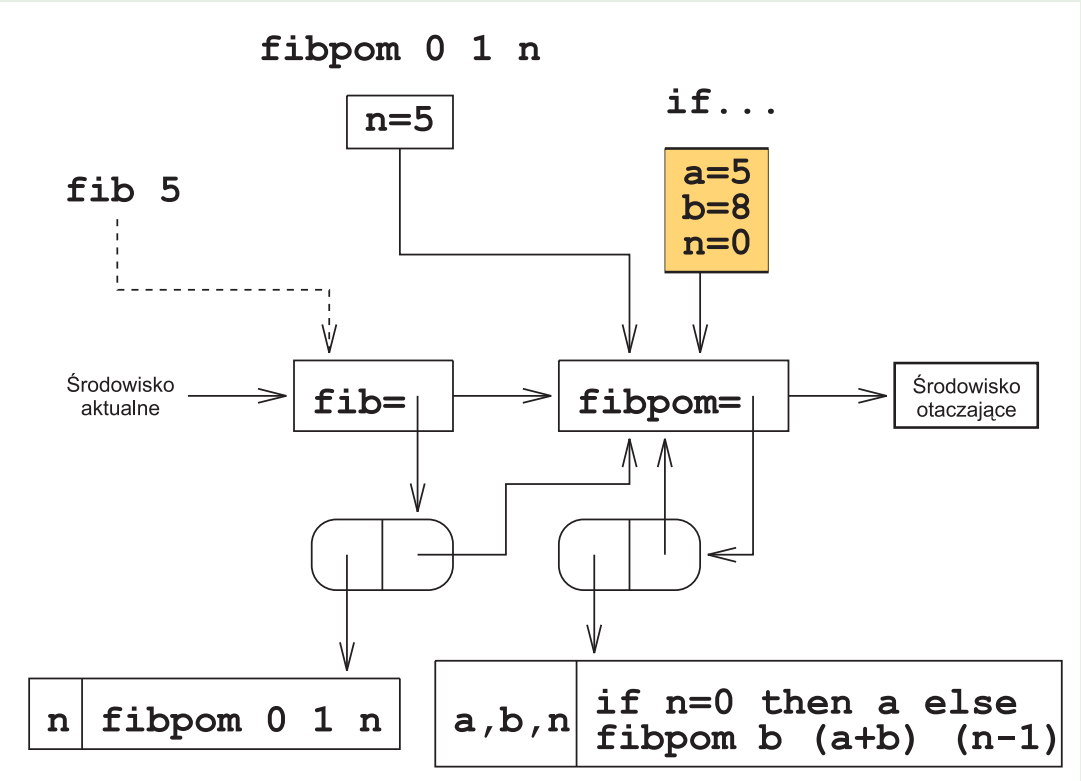
```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

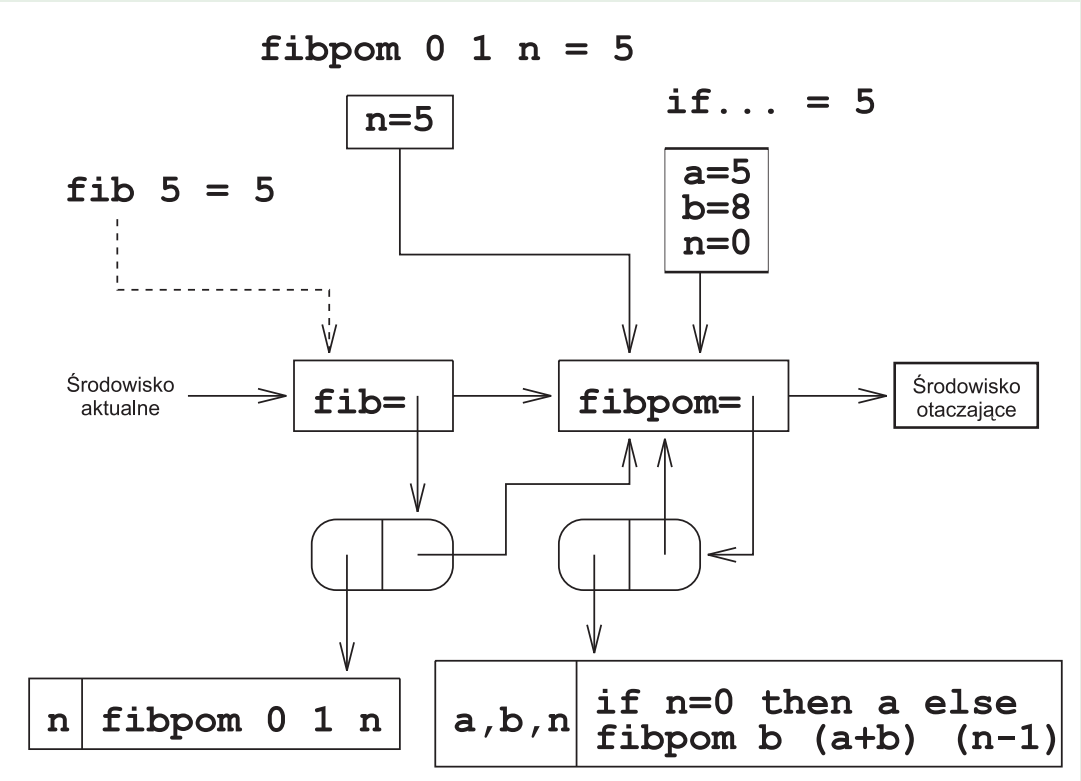
```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Rekurencja ogonowa

Example (Liczby Fibonacciego ogonowo)

```
let rec fibpom a b n =
  if n = 0 then a else fibpom b (a + b) (n - 1);;
let fib n = fibpom 0 1 n;;
fib 5;;
```



Definicje lokalne

Obliczenie wyrażenia `let ...in ...`:

- utworzenie ramki zawierającej definicje lokalne,
- środowisko otaczające = środowisko aktualne,
- obliczenie wyrażenia po `in`,
- przywrócenie poprzedniego środowiska,
- Zagnieżdżone definicje lokalne — powstaje więcej ramek.

Definicje lokalne

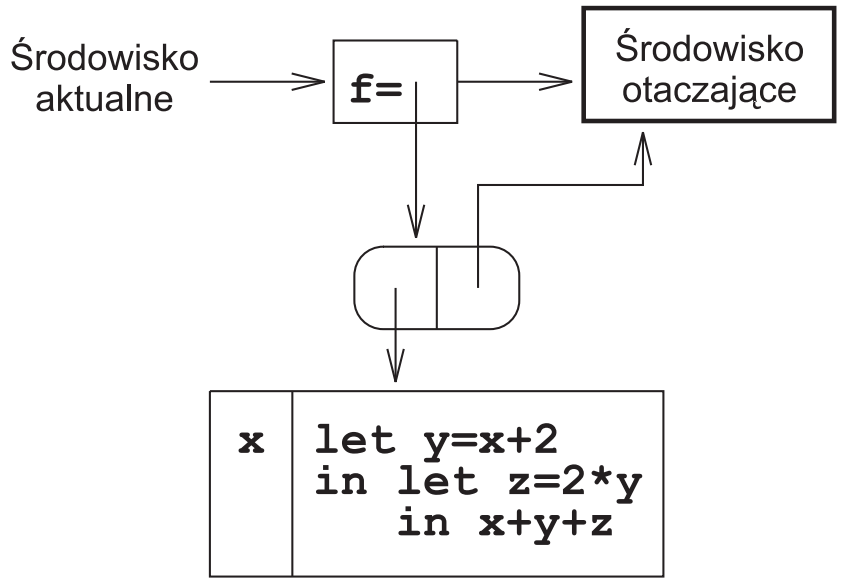
Example

```
let f x =
  let y = x + 2
  in
    let z = 2 * y
    in
      x + y + z;;
f 9;;
```

Definicje lokalne

Example

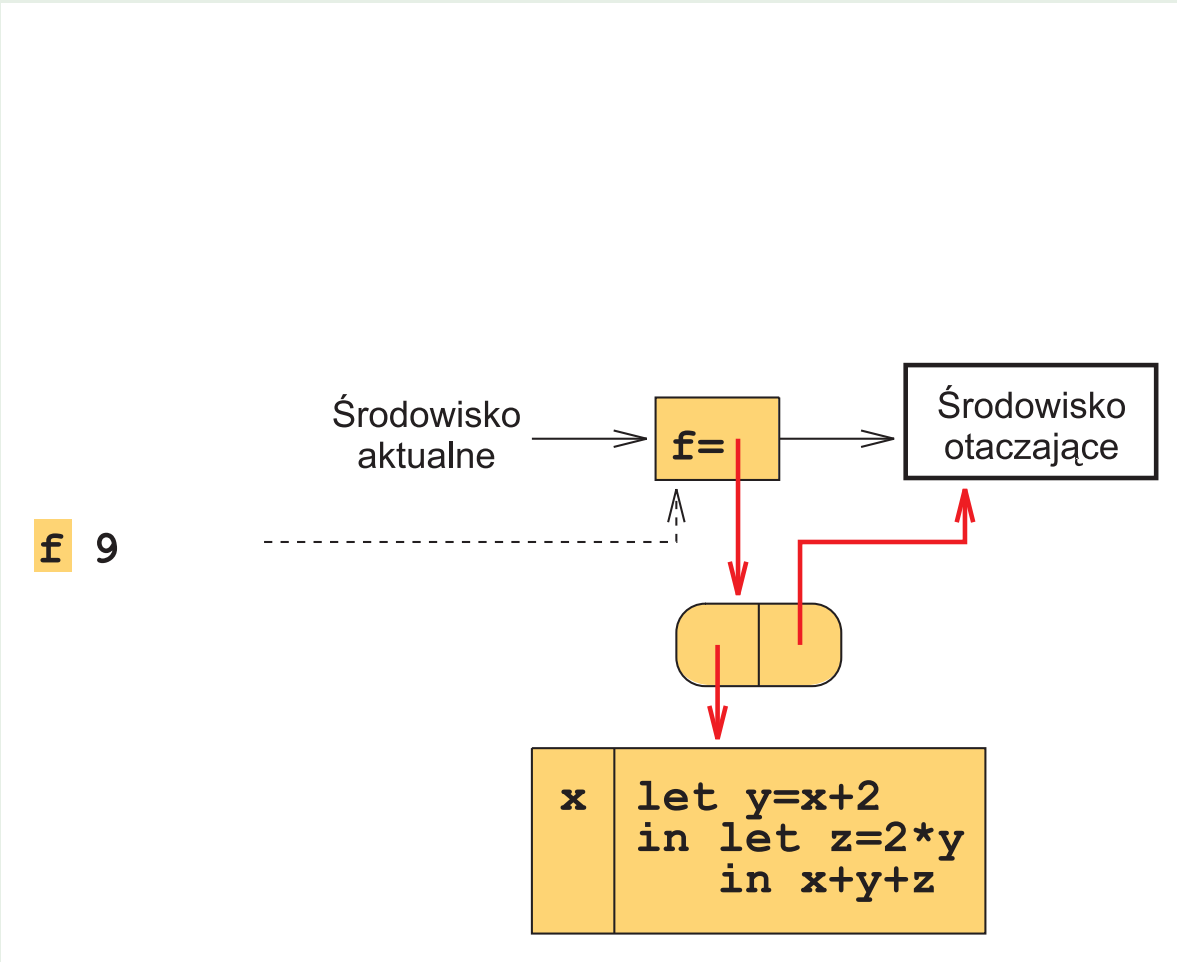
```
let f x =
  let y = x + 2
  in
    let z = 2 * y
    in
      x + y + z;;
f 9;;
```



Definicje lokalne

Example

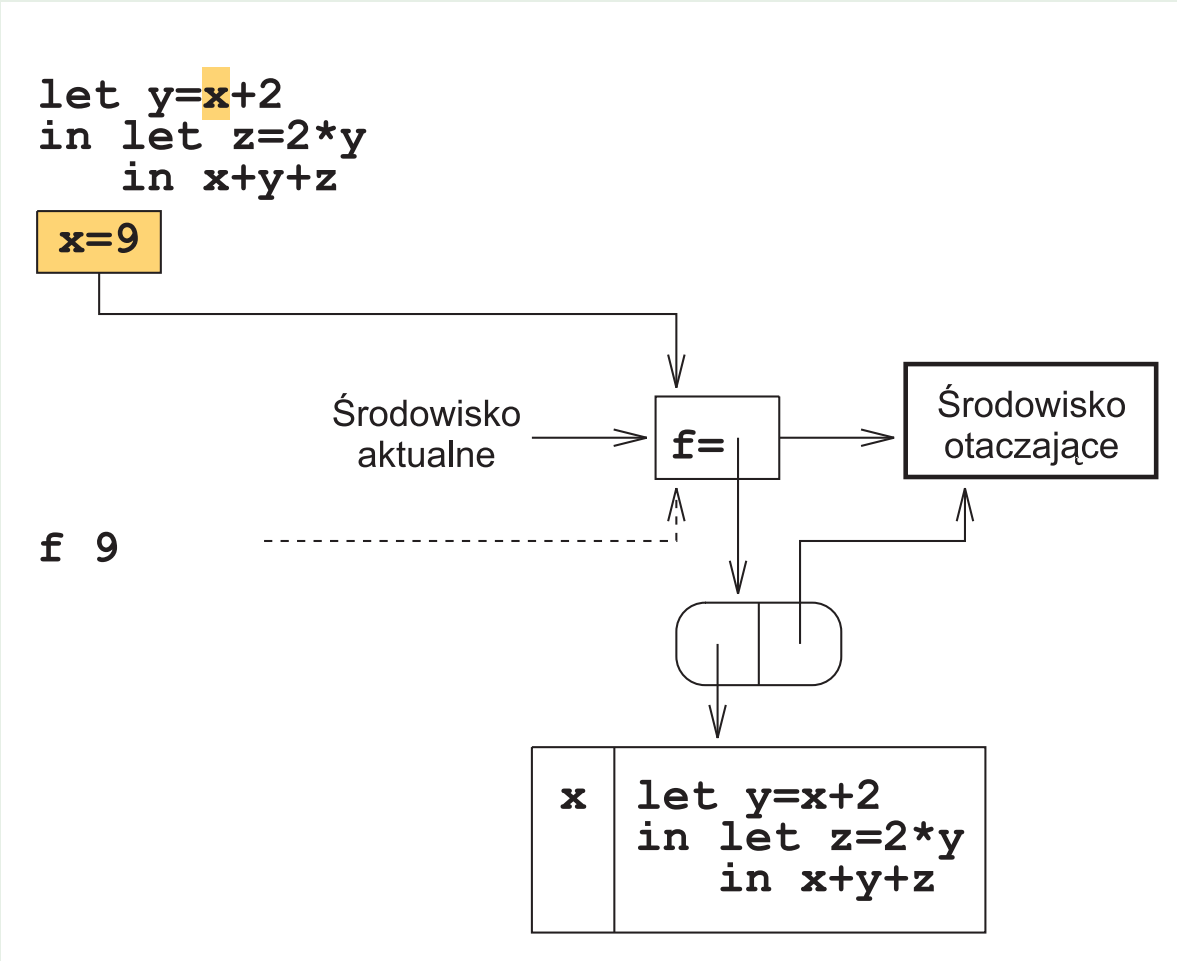
```
let f x =
  let y = x + 2
  in
    let z = 2 * y
    in
      x + y + z;;
f 9;;
```



Definicje lokalne

Example

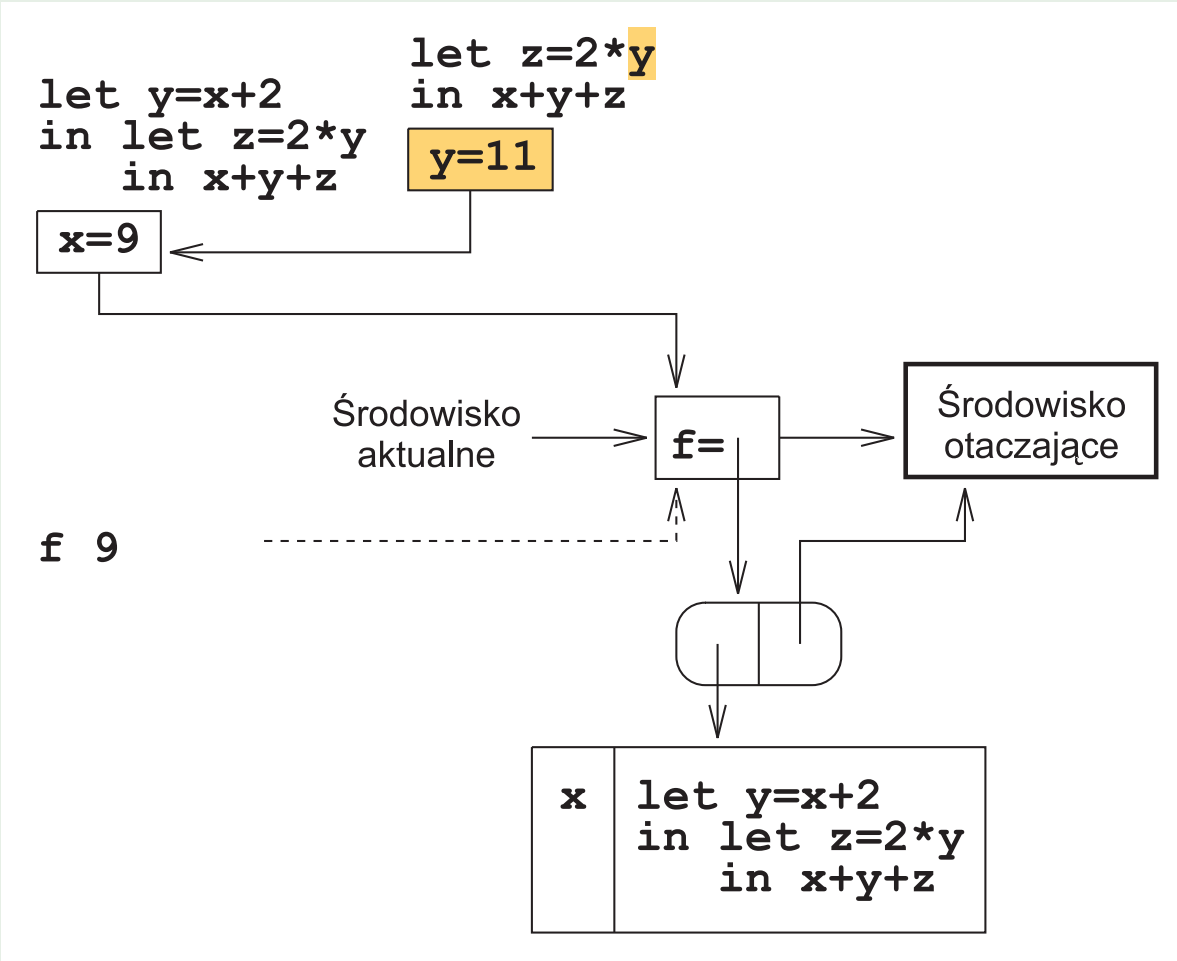
```
let f x =
  let y = x + 2
  in
    let z = 2 * y
    in
      x + y + z;;
f 9;;
```



Definicje lokalne

Example

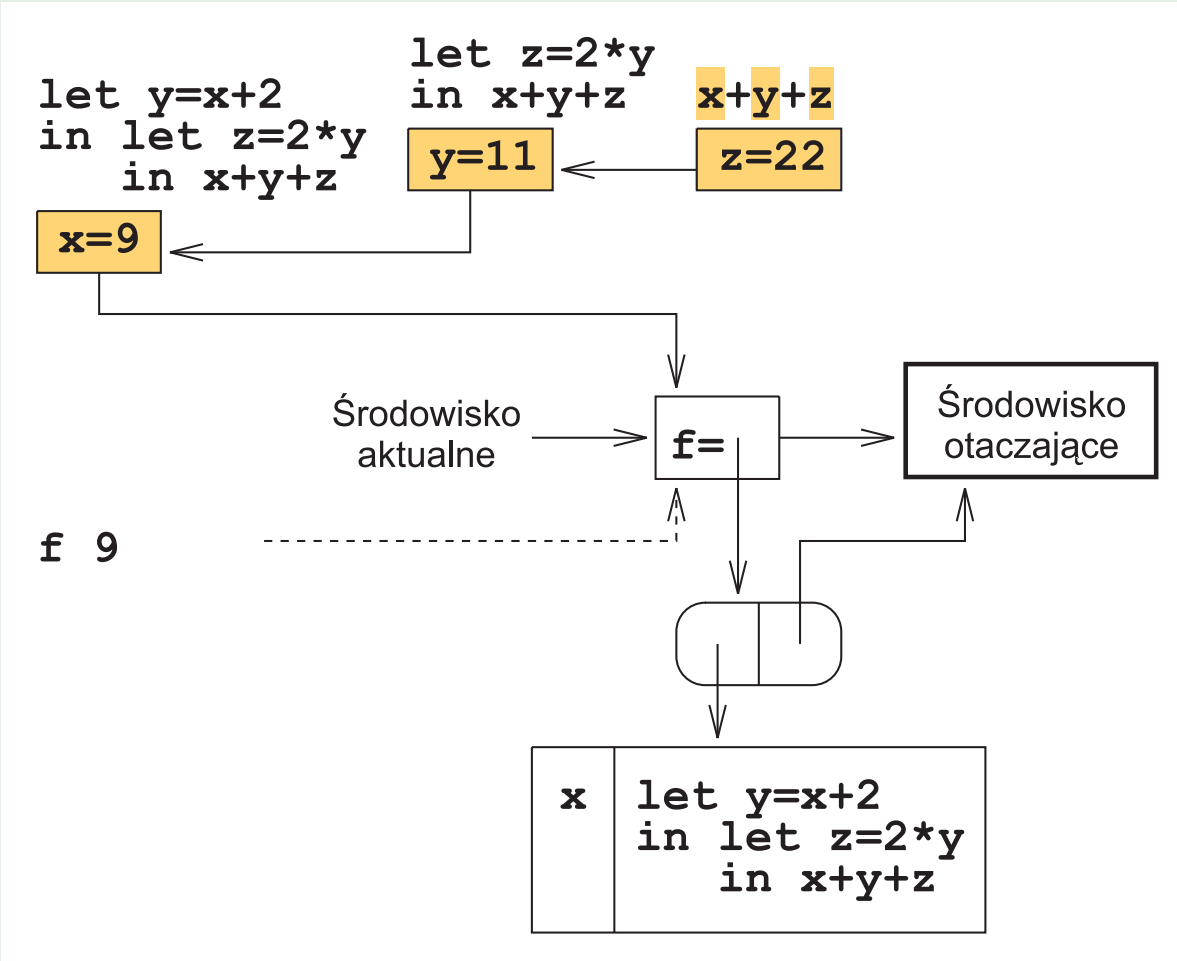
```
let f x =
  let y = x + 2
  in
    let z = 2 * y
    in
      x + y + z;;
f 9;;
```



Definicje lokalne

Example

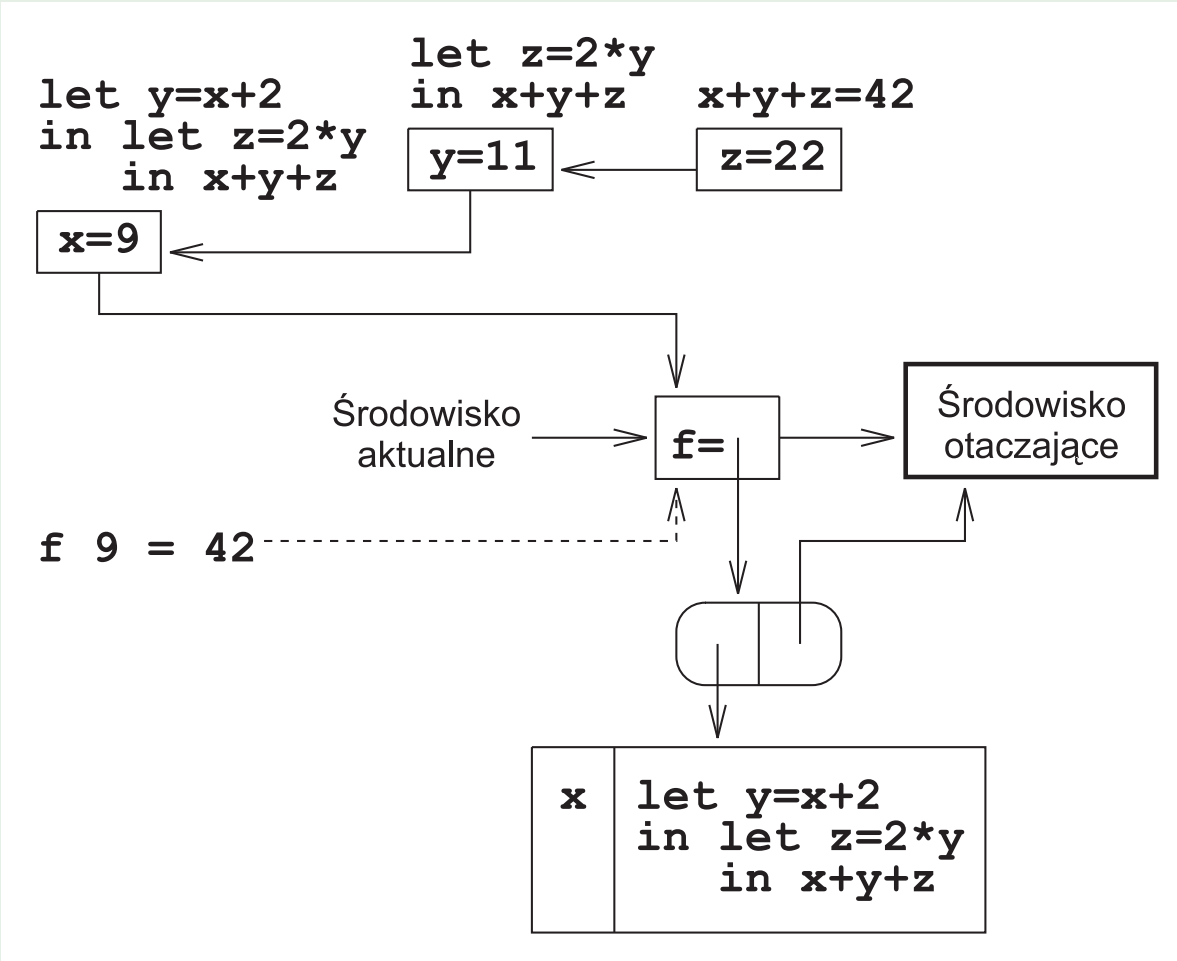
```
let f x =
  let y = x + 2
  in
    let z = 2 * y
    in
      x + y + z;;
f 9;;
```



Definicje lokalne

Example

```
let f x =
  let y = x + 2
  in
    let z = 2 * y
    in
      x + y + z;;
f 9;;
```



Definicje lokalne

Example

```
let f =
  let g x = x - 8
  in
    fun x -> g (2 * x);;
let h x = f (x * x);;
h 5;;
```

Procedura g:

- jest widoczna tylko wewnątrz f,
- ma jedną instancję.

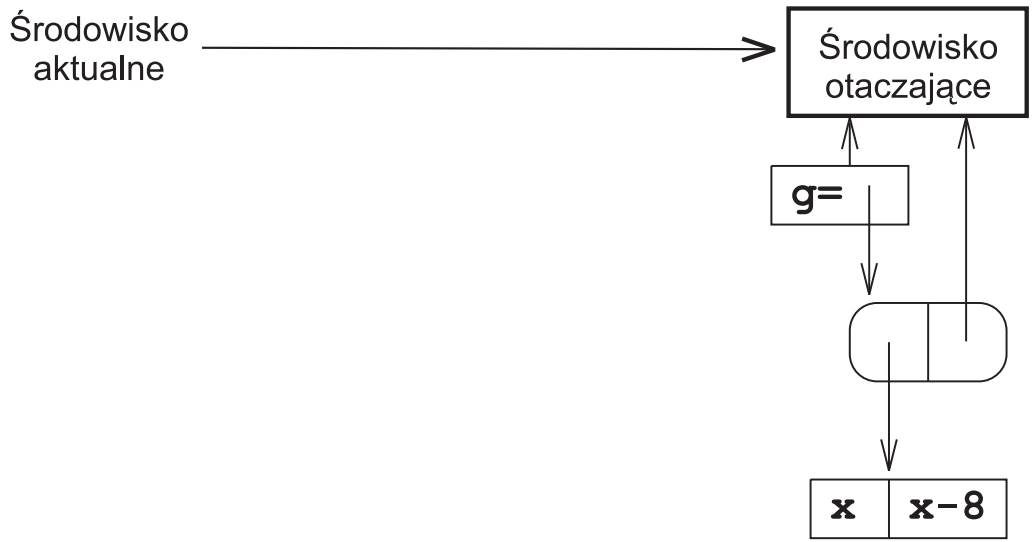
Definicje lokalne

Example

```
let f =
  let g x = x - 8
  in
    fun x -> g (2 * x);;
let h x = f (x * x);;
h 5;;
```

Procedura g:

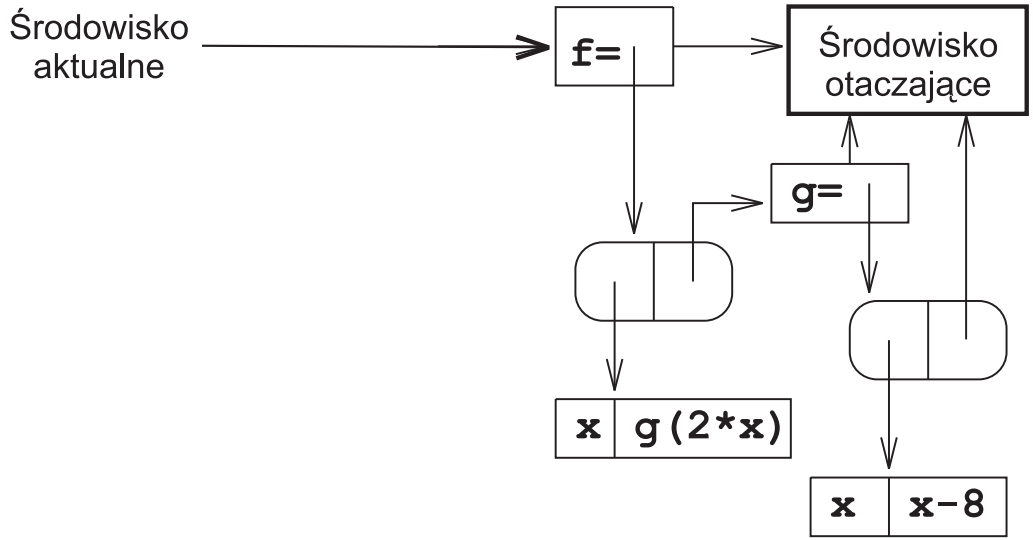
- jest widoczna tylko wewnątrz f,
- ma jedną instancję.



Definicje lokalne

Example

```
let f =
  let g x = x - 8
  in
    fun x -> g (2 * x);;
let h x = f (x * x);;
h 5;;
```



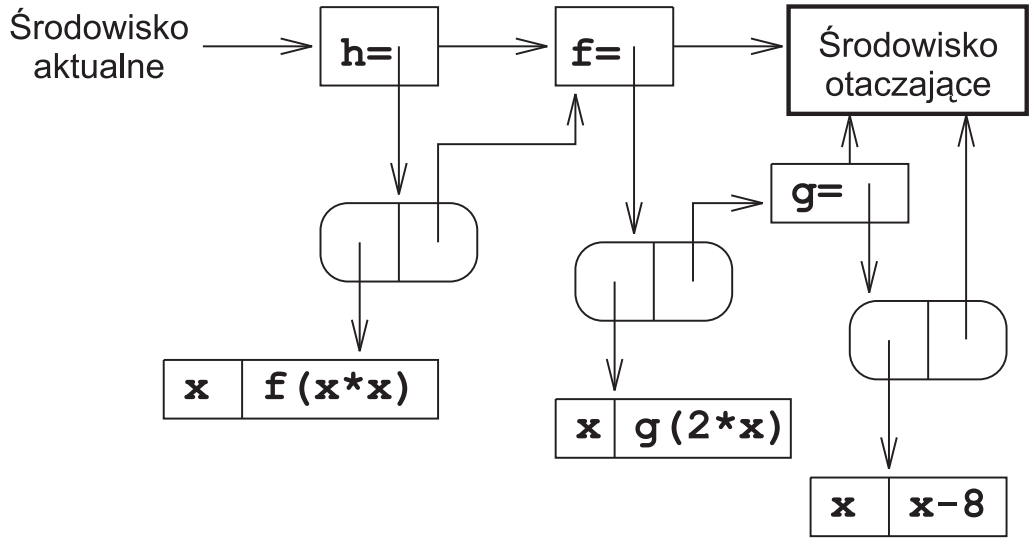
Procedura g:

- jest widoczna tylko wewnątrz f,
- ma jedną instancję.

Definicje lokalne

Example

```
let f =
  let g x = x - 8
  in
    fun x -> g (2 * x);;
let h x = f (x * x);;
h 5;;
```



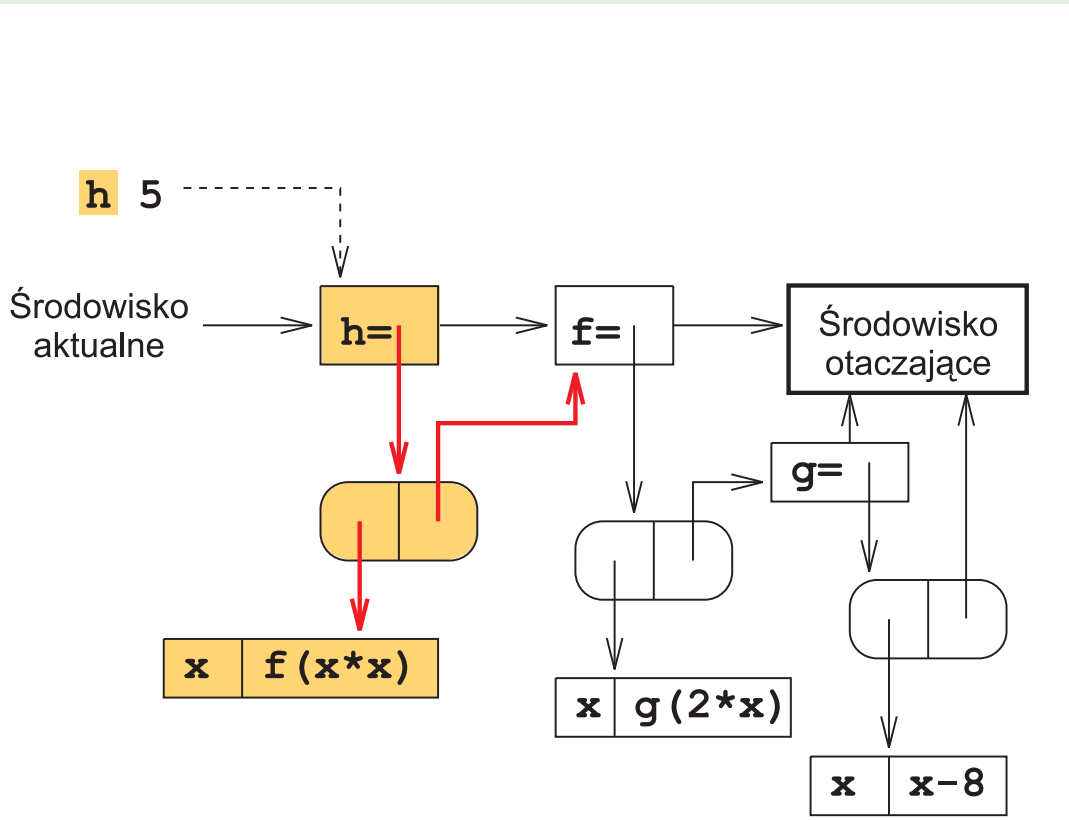
Procedura g:

- jest widoczna tylko wewnątrz f,
- ma jedną instancję.

Definicje lokalne

Example

```
let f =
  let g x = x - 8
  in
    fun x -> g (2 * x);;
let h x = f (x * x);;
h 5;;
```



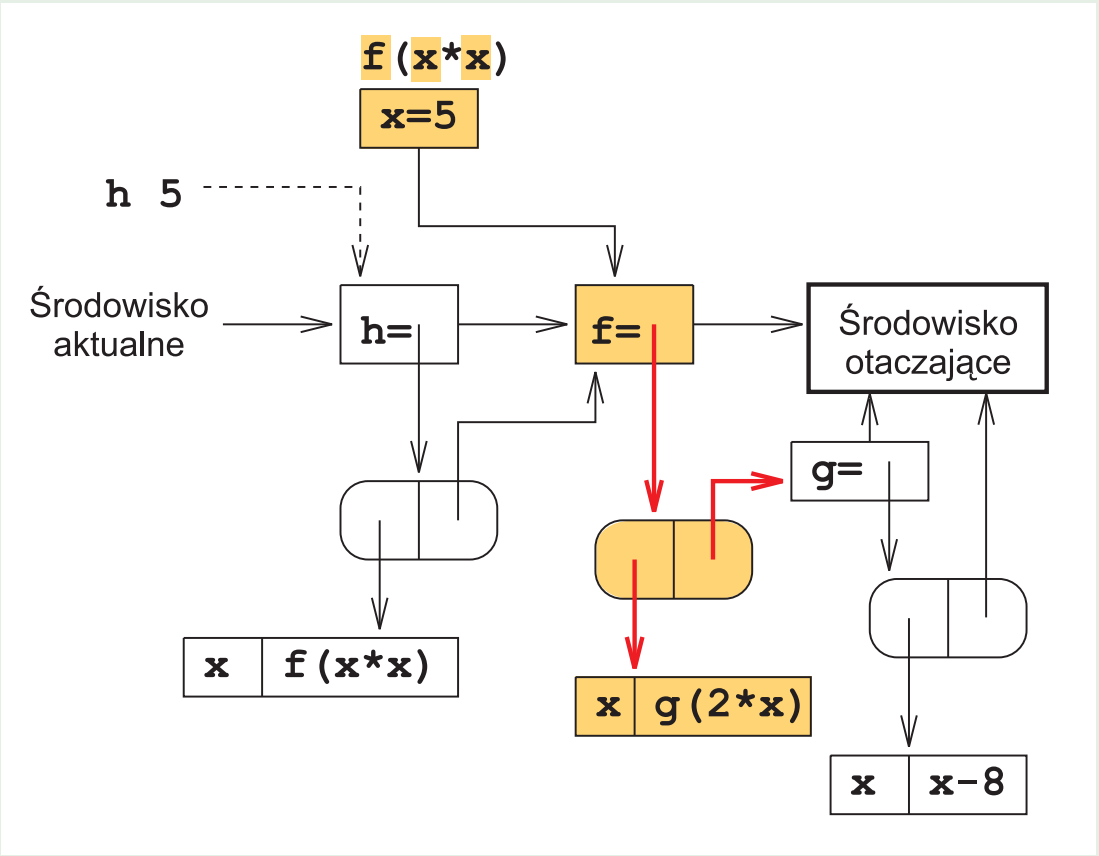
Procedura g:

- jest widoczna tylko wewnątrz f,
- ma jedną instancję.

Definicje lokalne

Example

```
let f =
  let g x = x - 8
  in
    fun x -> g (2 * x);;
let h x = f (x * x);;
h 5;;
```



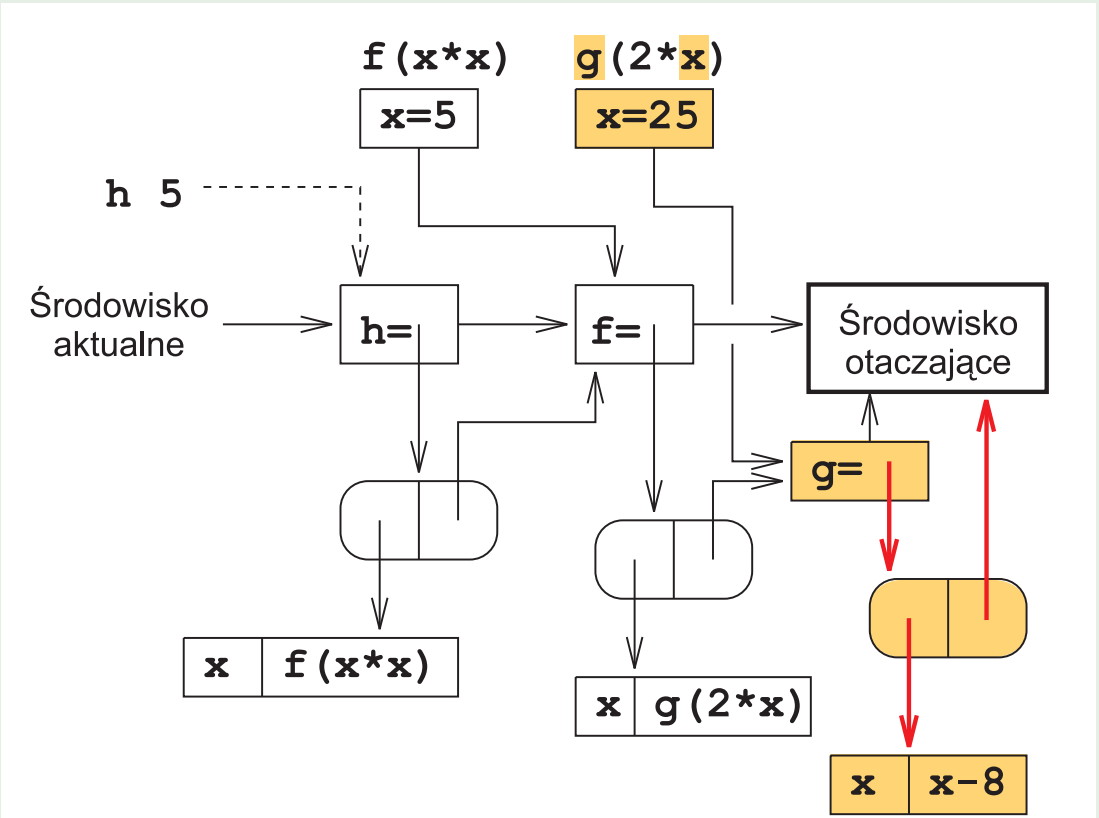
Procedura g:

- jest widoczna tylko wewnątrz f,
- ma jedną instancję.

Definicje lokalne

Example

```
let f =
  let g x = x - 8
  in
    fun x -> g (2 * x);;
let h x = f (x * x);;
h 5;;
```



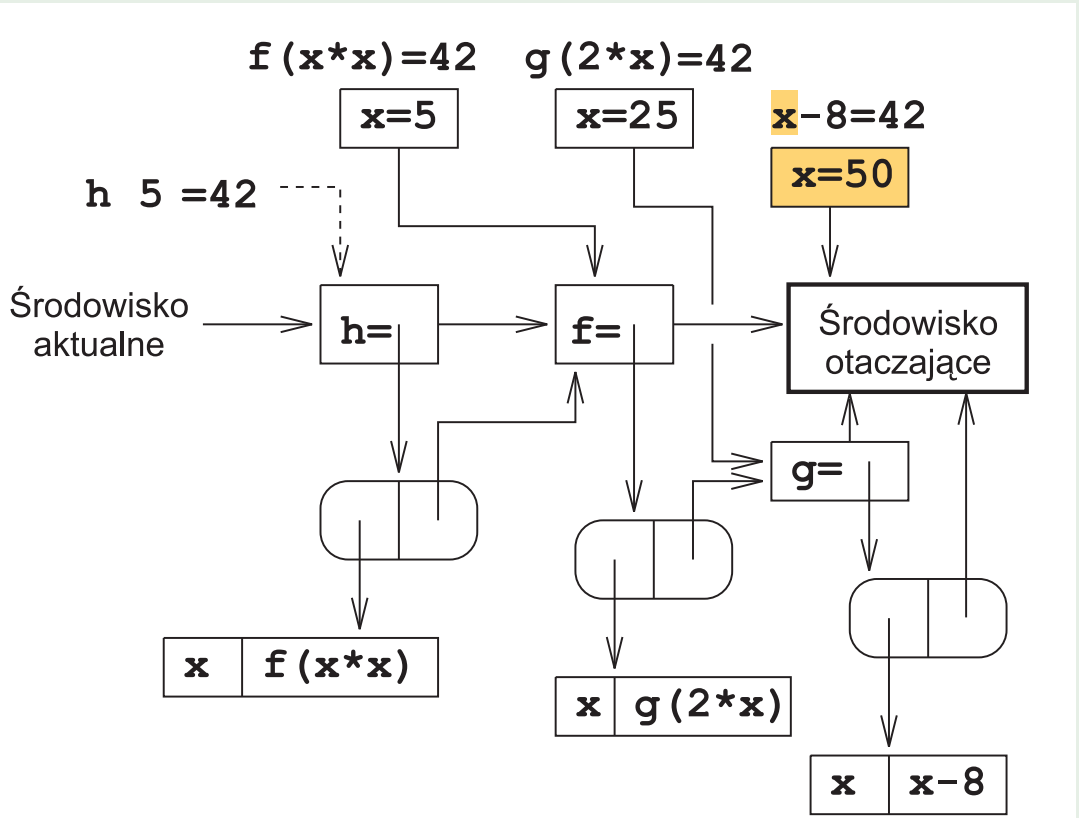
Procedura g:

- jest widoczna tylko wewnątrz f,
- ma jedną instancję.

Definicje lokalne

Example

```
let f =
  let g x = x - 8
  in
    fun x -> g (2 * x);;
let h x = f (x * x);;
h 5;;
```



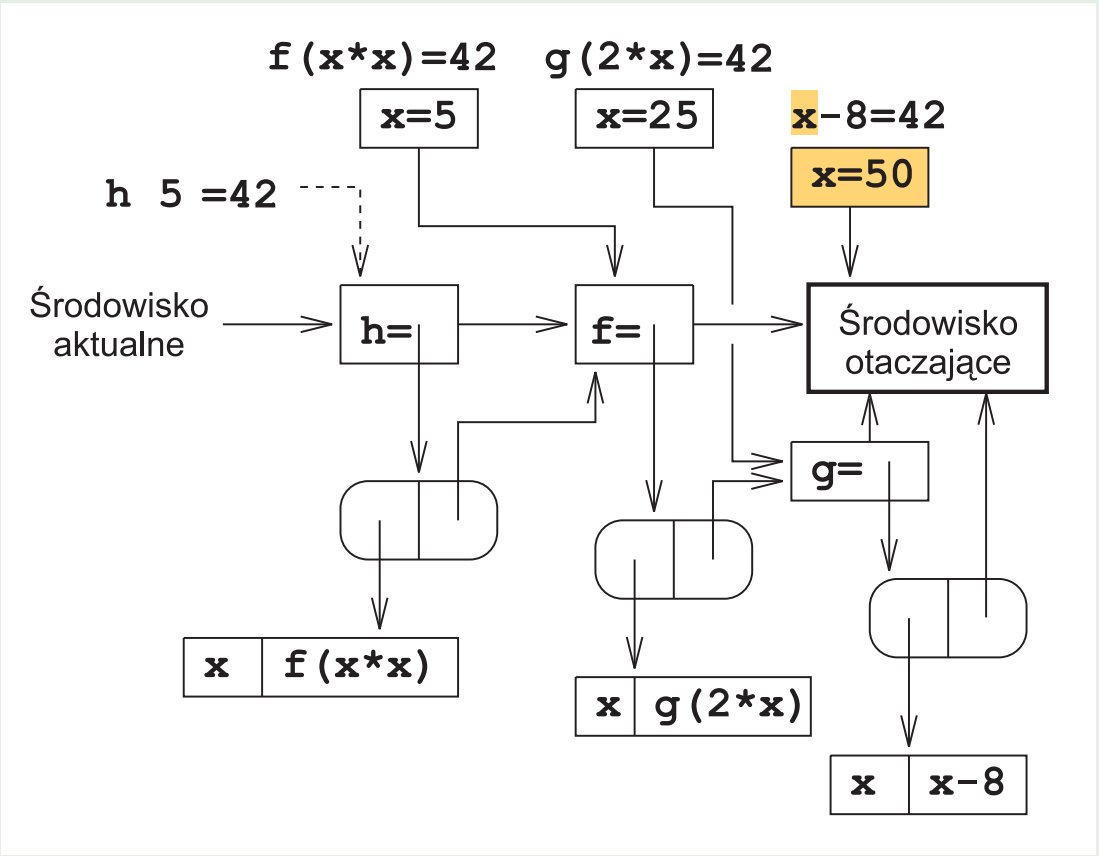
Procedura g:

- jest widoczna tylko wewnątrz f,
- ma jedną instancję.

Definicje lokalne

Example

```
let f =
  let g x = x - 8
  in
    fun x -> g (2 * x);;
let h x = f (x * x);;
h 5;;
```



Procedura g:

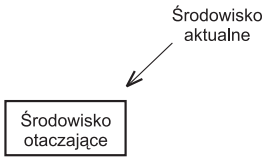
- jest widoczna tylko wewnątrz f,
- ma jedną instancję.

Definicje lokalne

Example

```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```

- Osobne instancje procedury g.
- g korzysta z argumentów f.

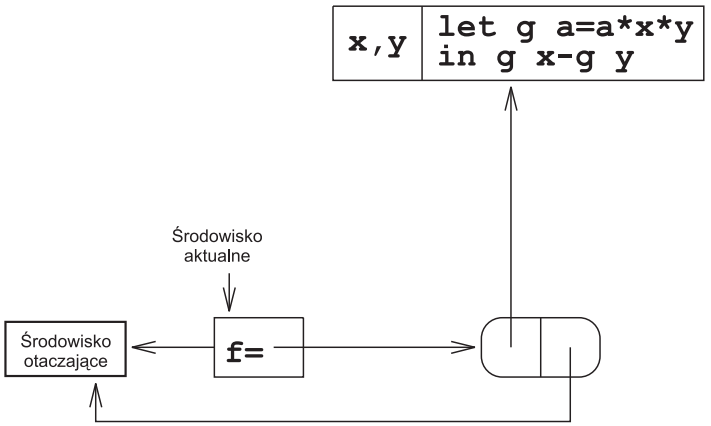


Definicje lokalne

Example

```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```

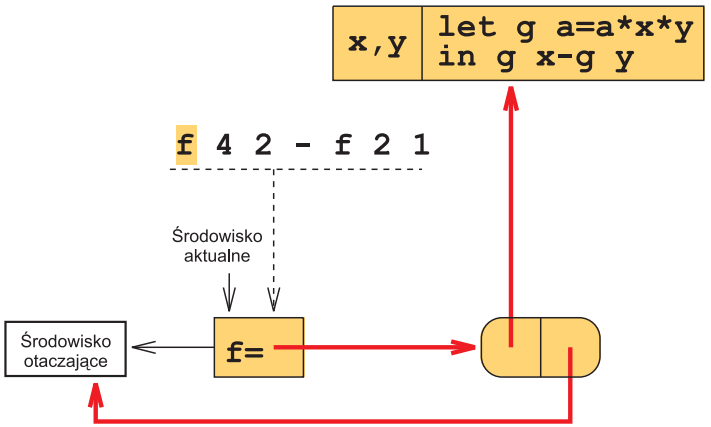
- Osobne instancje procedury g.
- g korzysta z argumentów f.



Definicje lokalne

Example

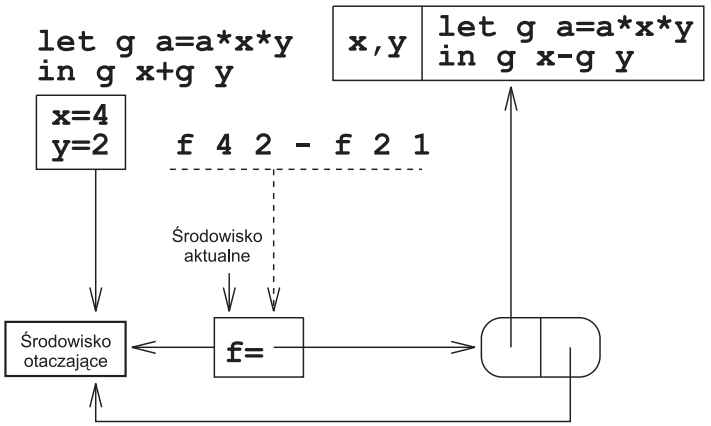
```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```



Definicje lokalne

Example

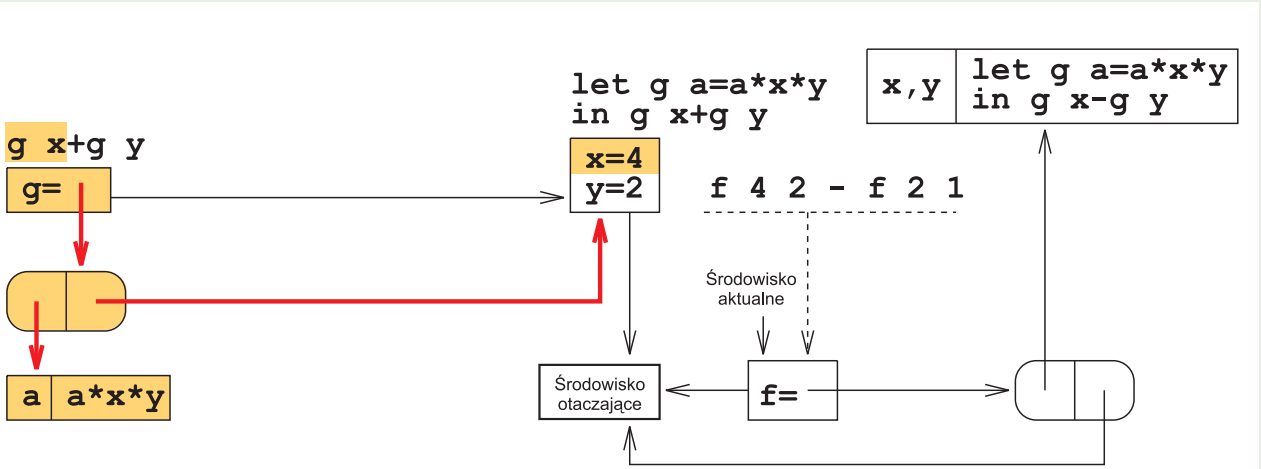
```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```



Definicje lokalne

Example

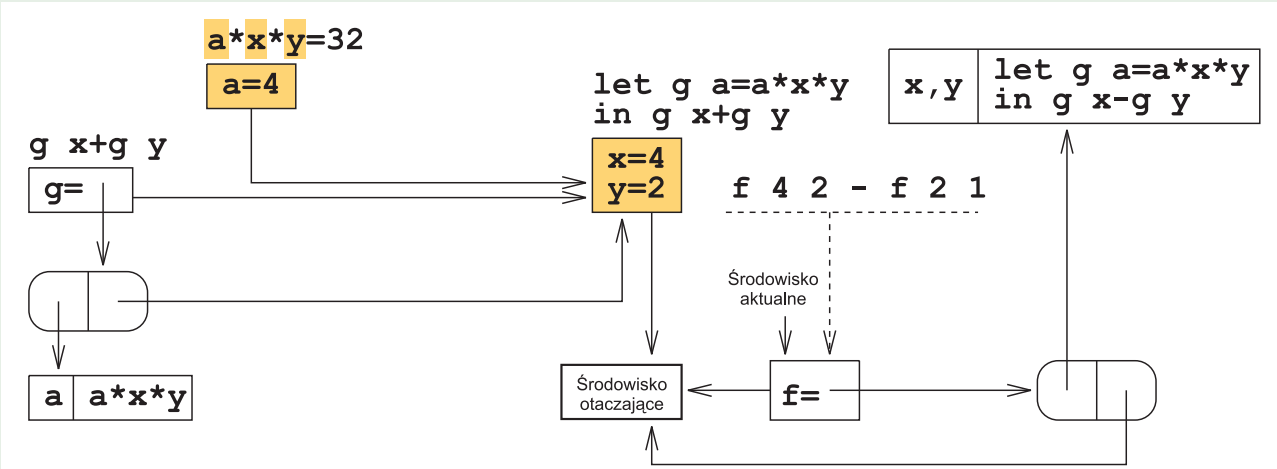
```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```



Definicje lokalne

Example

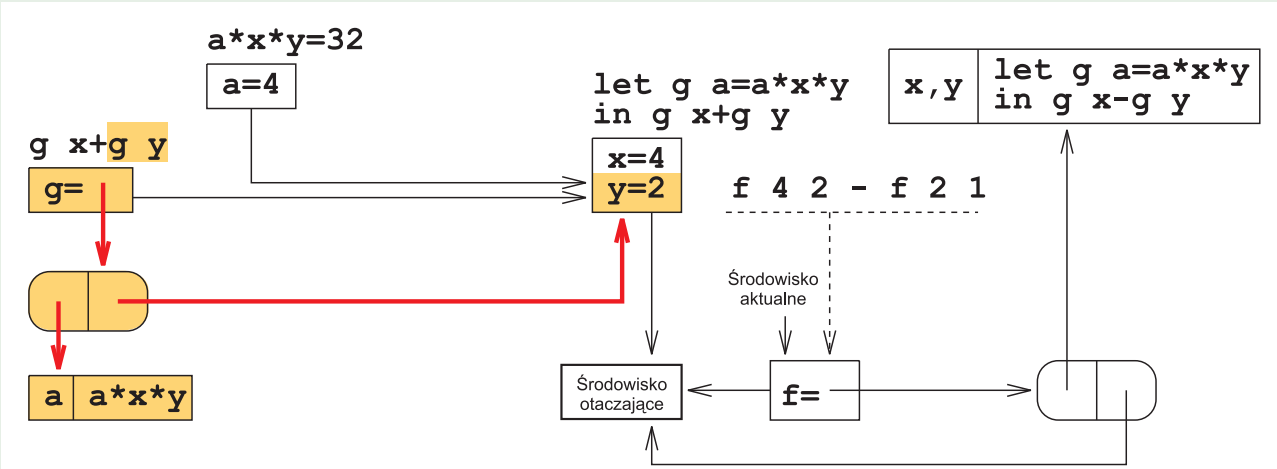
```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```



Definicje lokalne

Example

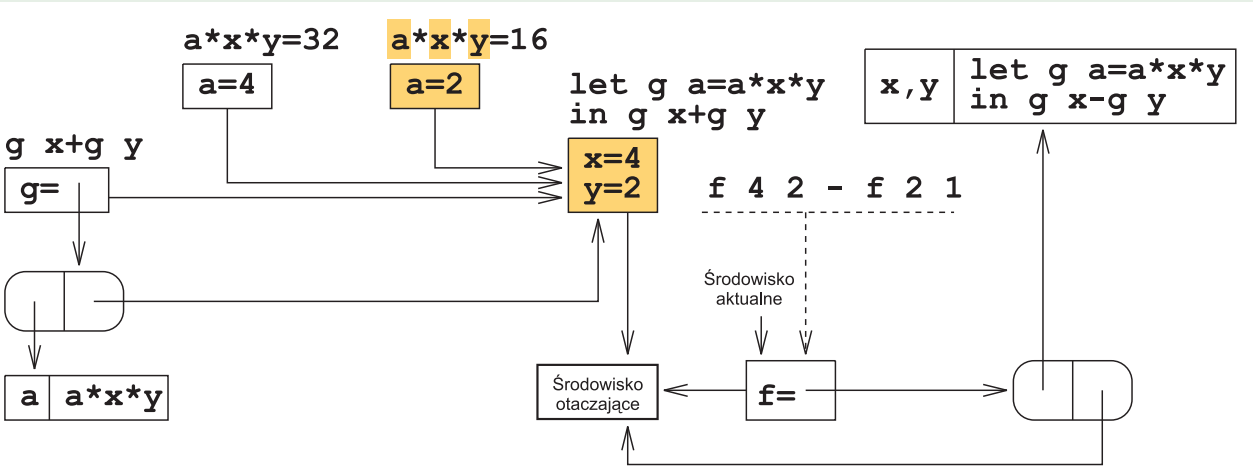
```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```



Definicje lokalne

Example

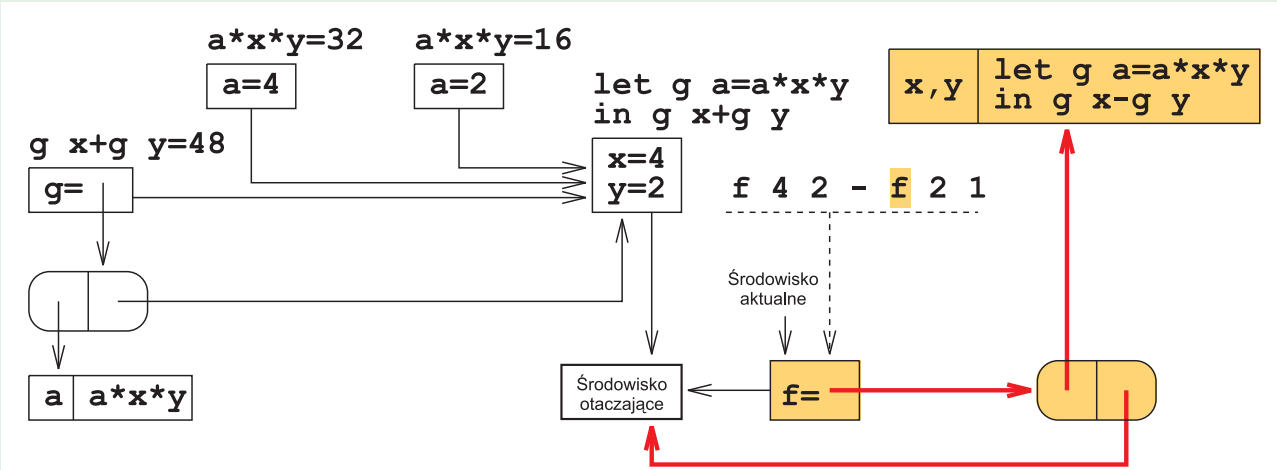
```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```



Definicje lokalne

Example

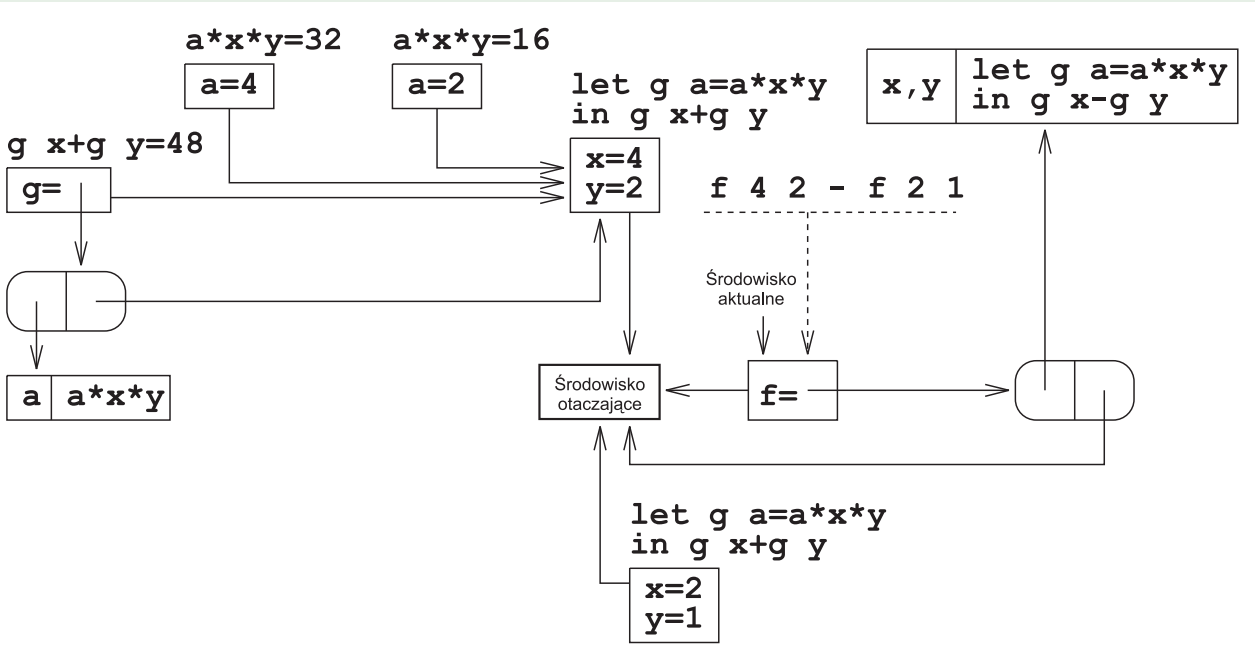
```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```



Definicje lokalne

Example

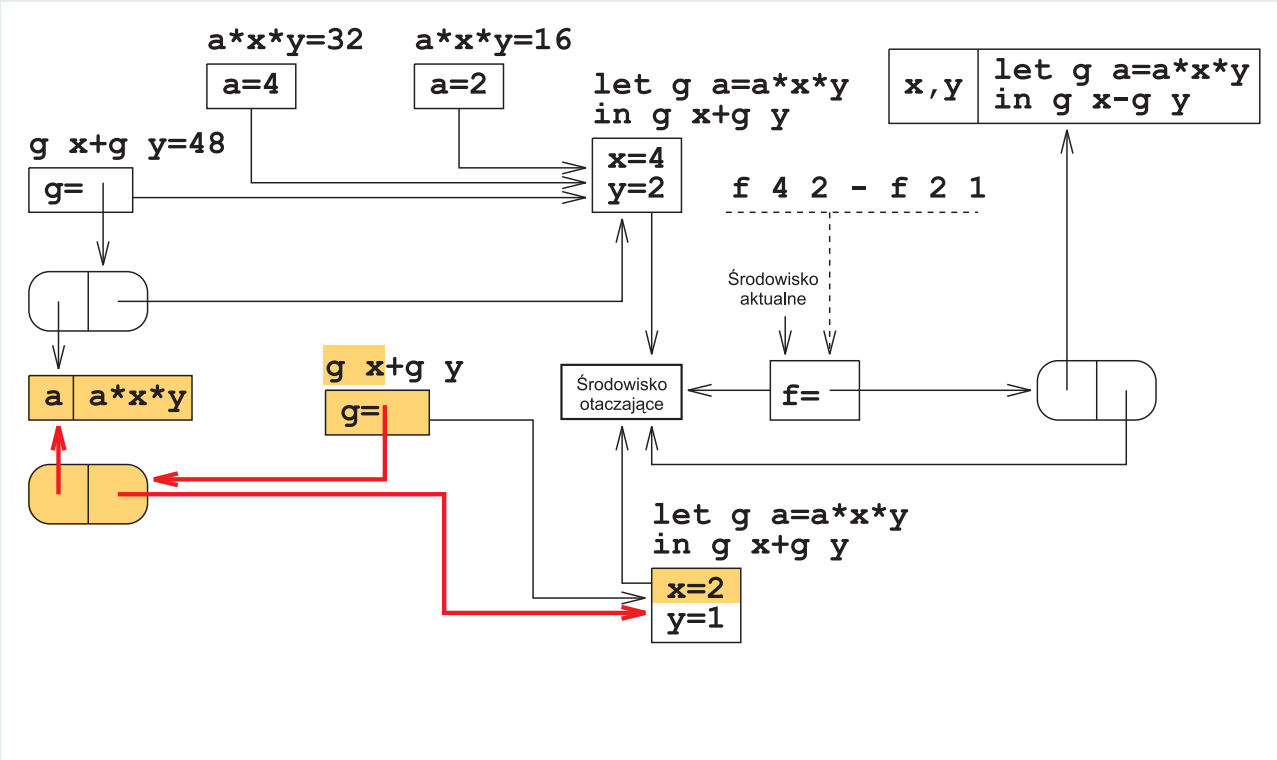
```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```



Definicje lokalne

Example

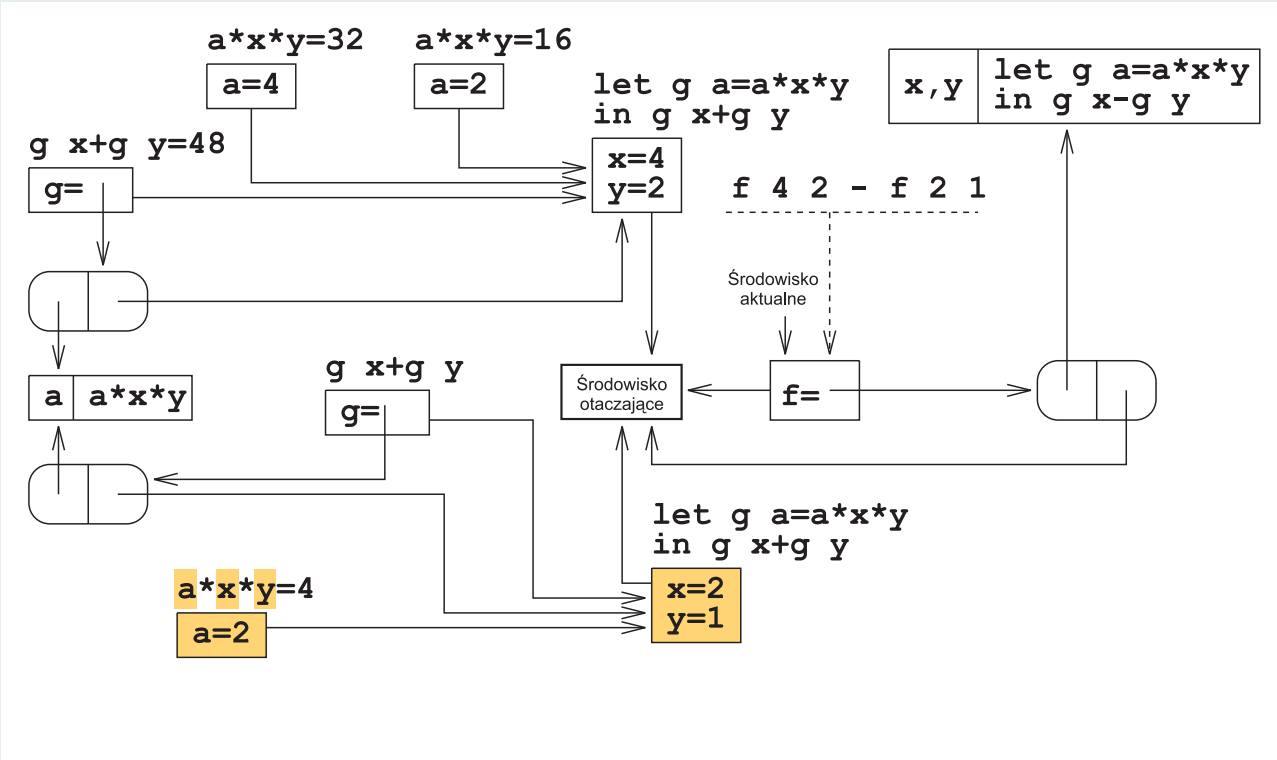
```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```



Definicje lokalne

Example

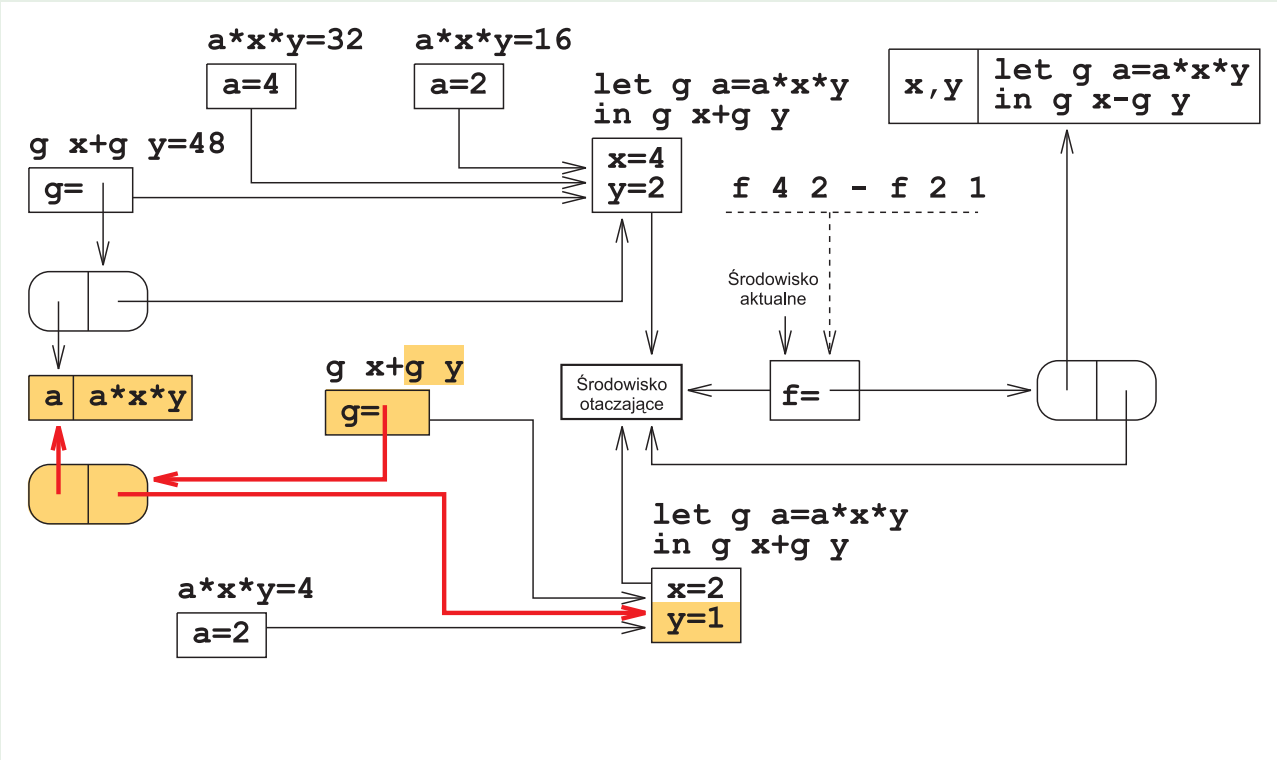
```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```



Definicje lokalne

Example

```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```

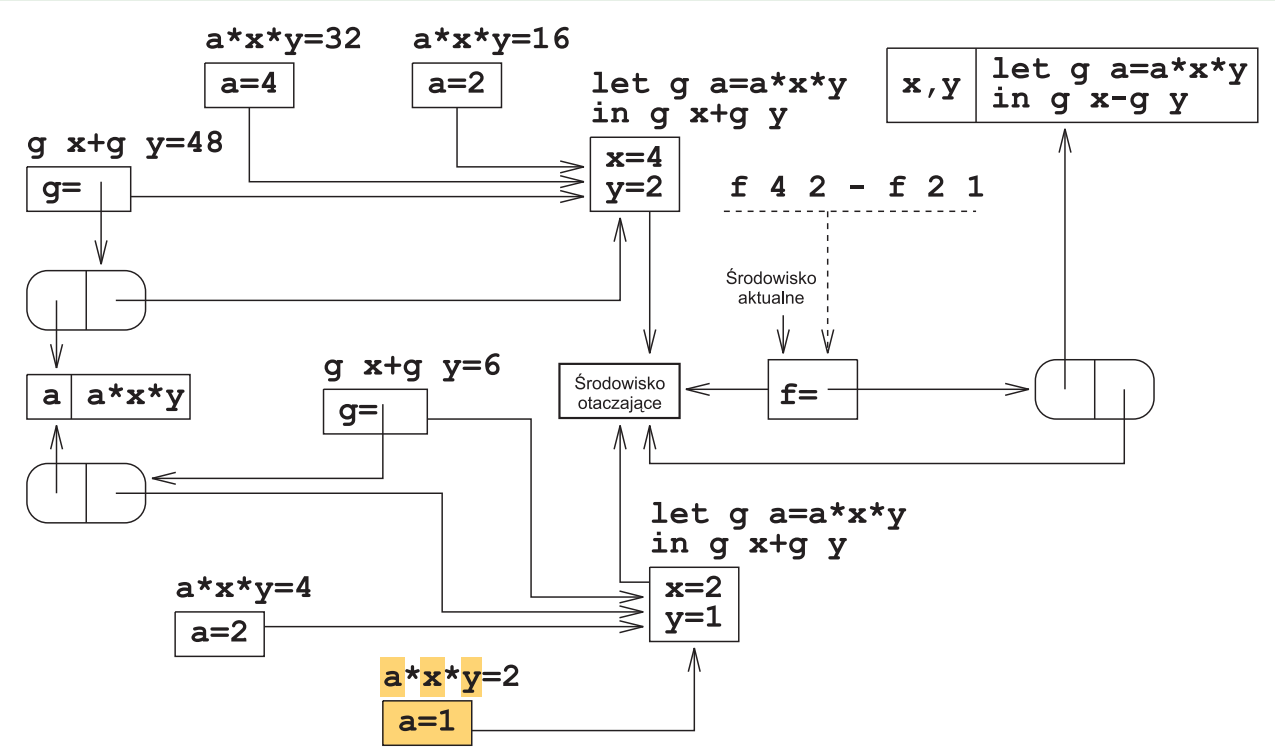


Definicje lokalne

Example

```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```

- Osobne instancje procedury g.
- g korzysta z argumentów f.

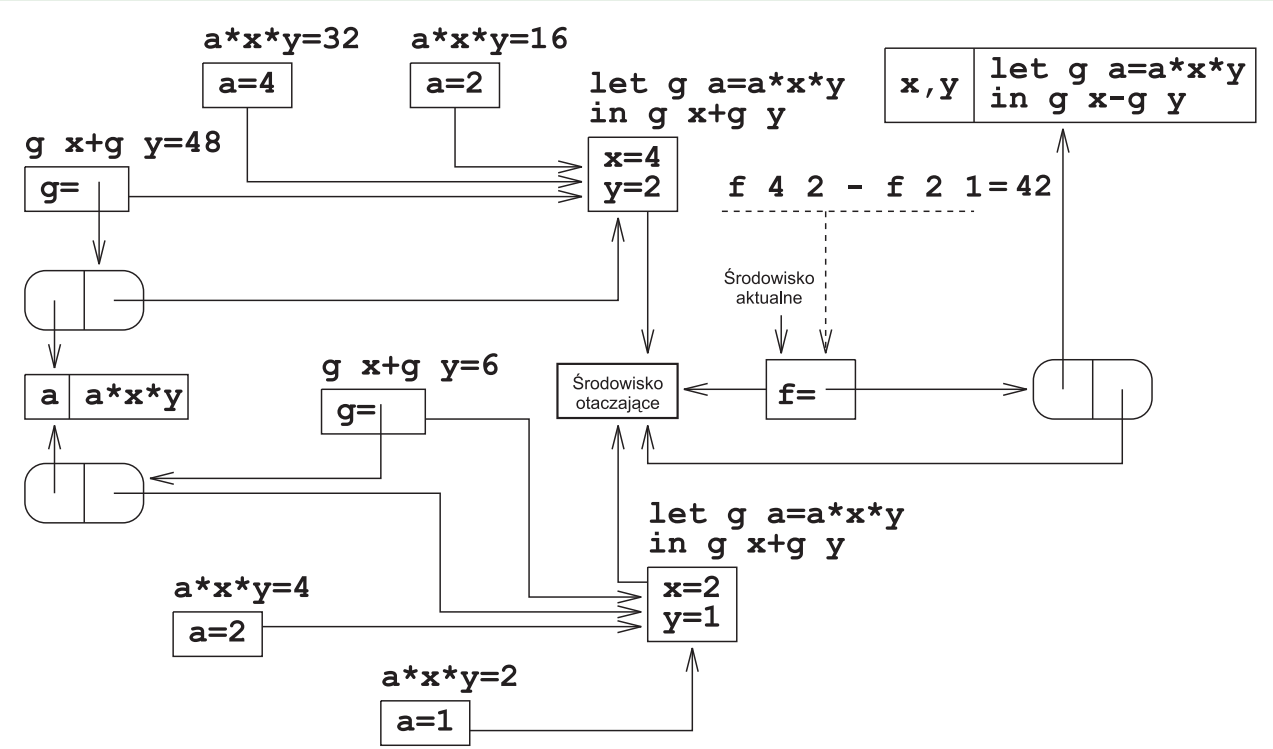


Definicje lokalne

Example

```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```

- Osobne instancje procedury g.
- g korzysta z argumentów f.

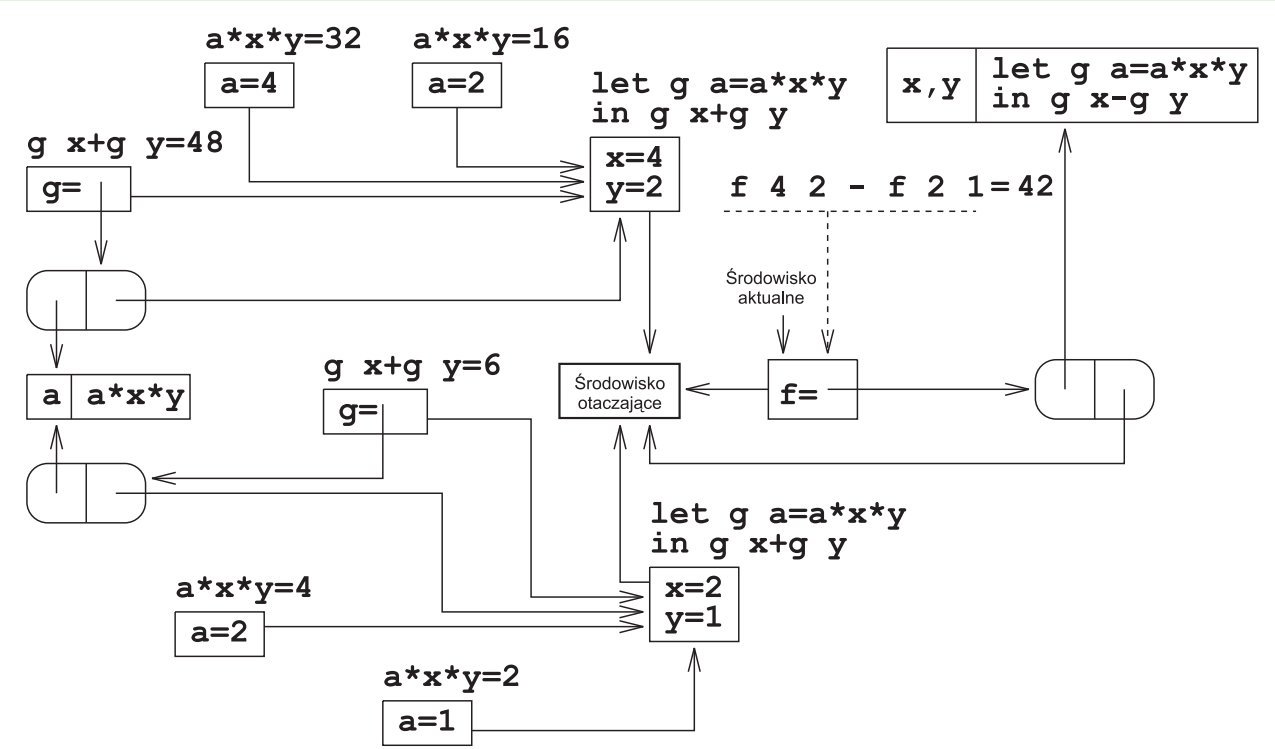


Definicje lokalne

Example

```
let f x y =
  let g a = a * x * y
  in g x + g y;;
f 4 2 - f 2 1;;
```

- Osobne instancje procedury g.
- g korzysta z argumentów f.



Ramki i rekordy aktywacji

Rekordy aktywacji:

- Co to jest?
- Zawierają m.in. wartości zmiennych lokalnych.
- Tworzą stos.
- Niszczony w momencie powrotu z wywołania procedury.

Ramki:

- Nawet po zakończeniu wywołania procedury nie muszą być zbędne.
- Mogą na nią wskazywać wartości proceduralne (gdy wynikiem procedury jest procedura mająca dostęp do lokalnych definicji).
- Wskaźniki na środowisko otaczające tworzą drzewo.

Ramki i rekordy aktywacji

Rekordy aktywacji:

- Co to jest?
- Zawierają m.in. wartości zmiennych lokalnych.
- Tworzą stos.
- Niszczony w momencie powrotu z wywołania procedury.

Ramki:

- Nawet po zakończeniu wywołania procedury nie muszą być zbędne.
- Mogą na nią wskazywać wartości proceduralne (gdy wynikiem procedury jest procedura mająca dostęp do lokalnych definicji).
- Wskaźniki na środowisko otaczające tworzą drzewo.

Ramki a rekordy aktywacji

Example

```
let f x =
  let g y = x * y
  in g;;
let h = f 6;;
h 7;;
```

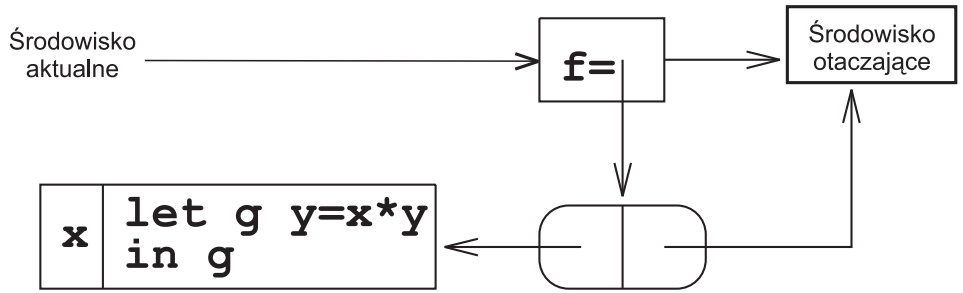
Ramka z definicją g powstaje w wywołaniu f 6, ale jest potrzebna do obliczenia h 7.

Ramki a rekordy aktywacji

Example

```
let f x =
  let g y = x * y
  in g;;
let h = f 6;;
h 7;;
```

Ramka z definicją g powstaje w wywołaniu f 6, ale jest potrzebna do obliczenia h 7.

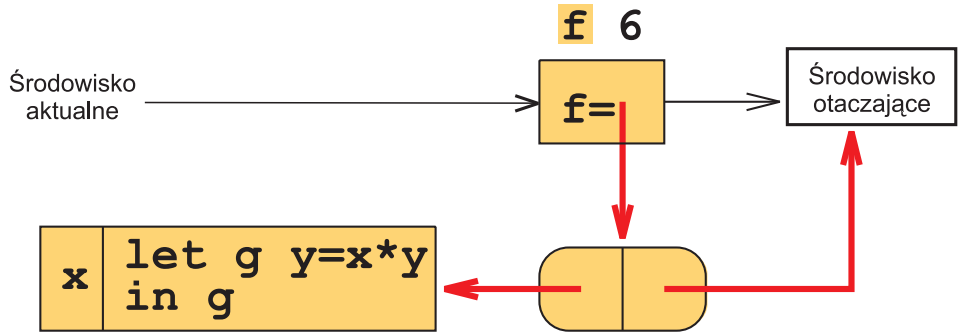


Ramki a rekordy aktywacji

Example

```
let f x =
  let g y = x * y
  in g;;
let h = f 6;;
h 7;;
```

Ramka z definicją g powstaje w wywołaniu f 6, ale jest potrzebna do obliczenia h 7.

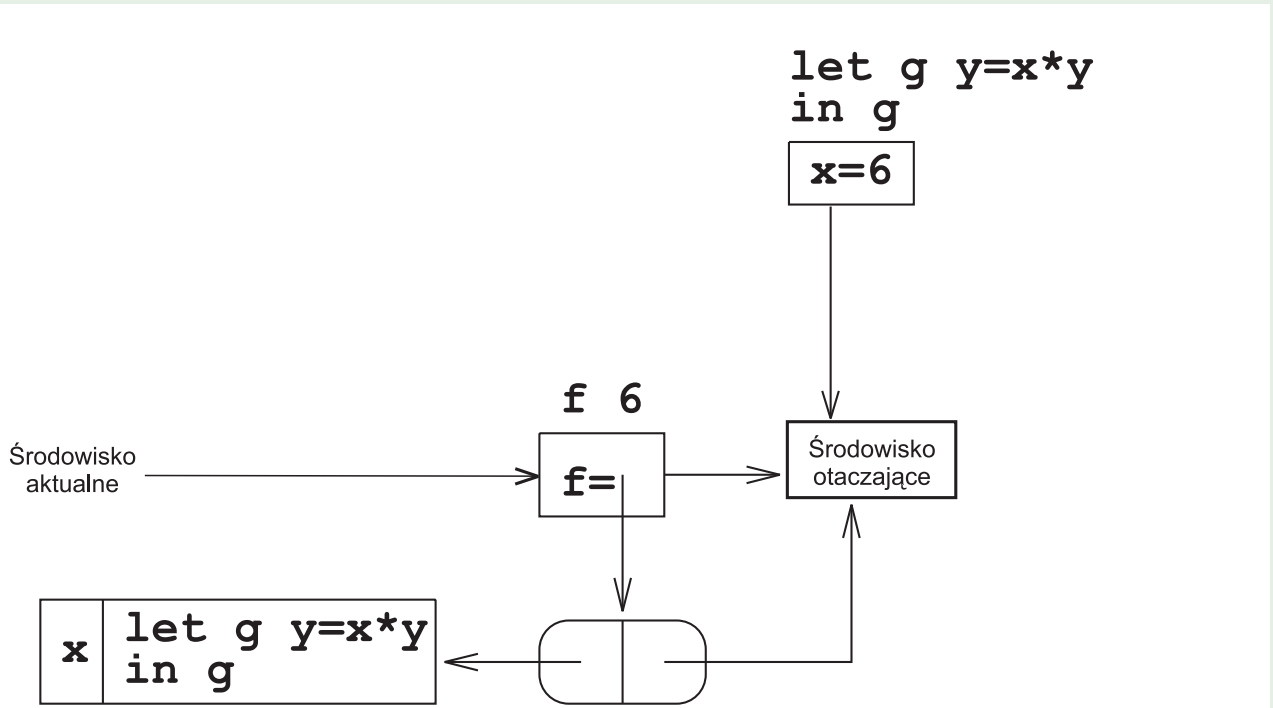


Ramki a rekordy aktywacji

Example

```
let f x =
  let g y = x * y
  in g;;
let h = f 6;;
h 7;;
```

Ramka z definicją g powstaje w wywołaniu f 6, ale jest potrzebna do obliczenia h 7.

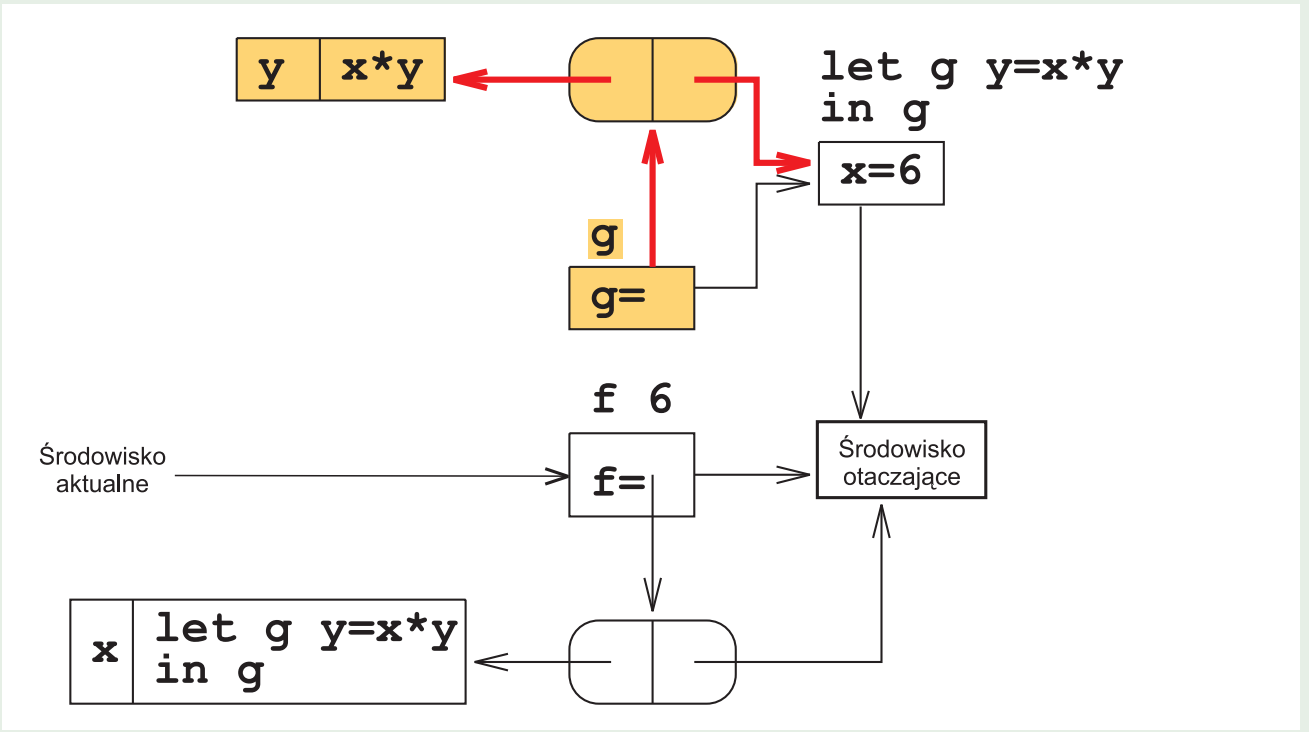


Ramki a rekordy aktywacji

Example

```
let f x =
  let g y = x * y
  in g;;
let h = f 6;;
h 7;;
```

Ramka z definicją g powstaje w wywołaniu f 6, ale jest potrzebna do obliczenia h 7.

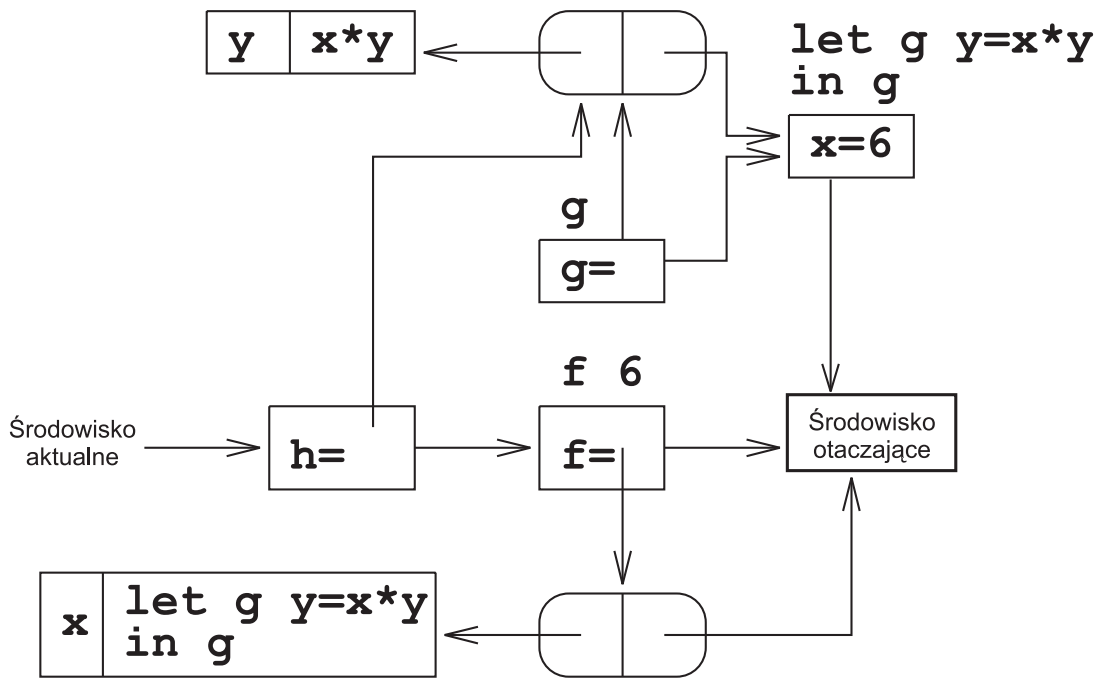


Ramki a rekordy aktywacji

Example

```
let f x =
  let g y = x * y
  in g;;
let h = f 6;;
h 7;;
```

Ramka z definicją g powstaje w wywołaniu f 6, ale jest potrzebna do obliczenia h 7.

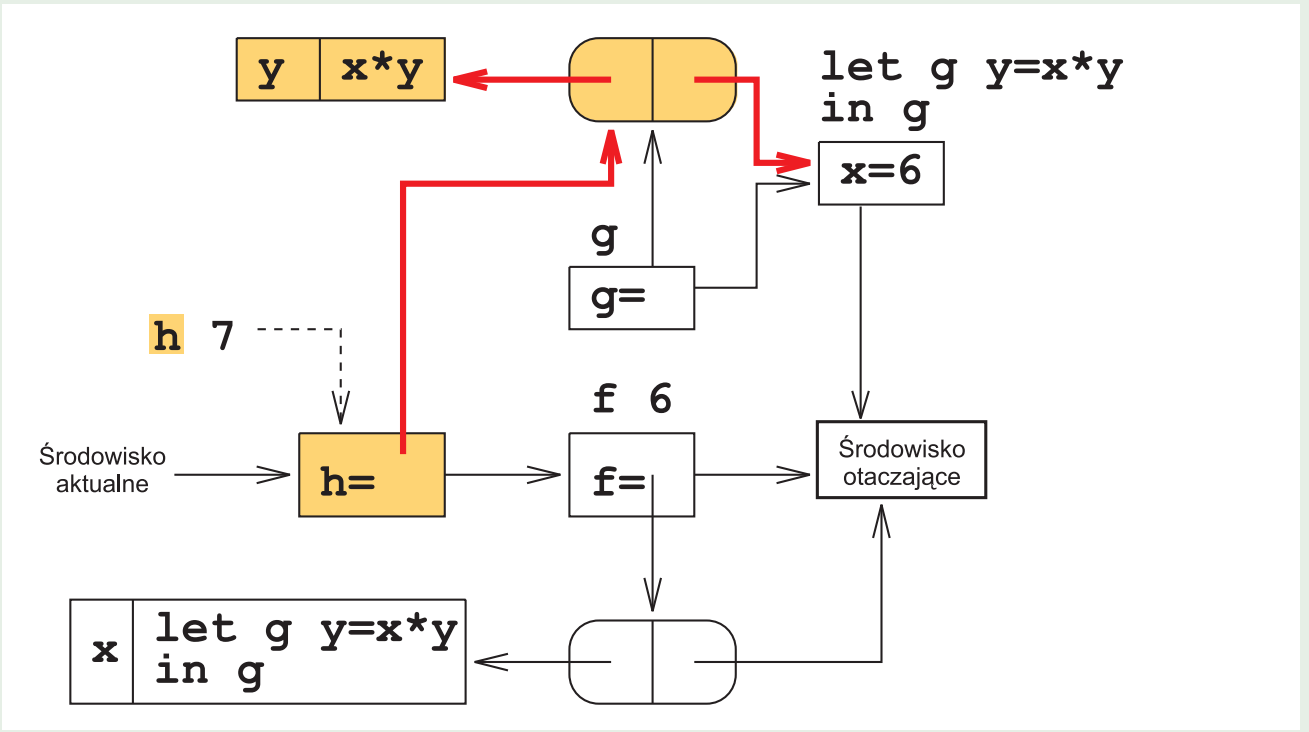


Ramki a rekordy aktywacji

Example

```
let f x =
  let g y = x * y
  in g;;
let h = f 6;;
h 7;;
```

Ramka z definicją g powstaje w wywołaniu f 6, ale jest potrzebna do obliczenia h 7.

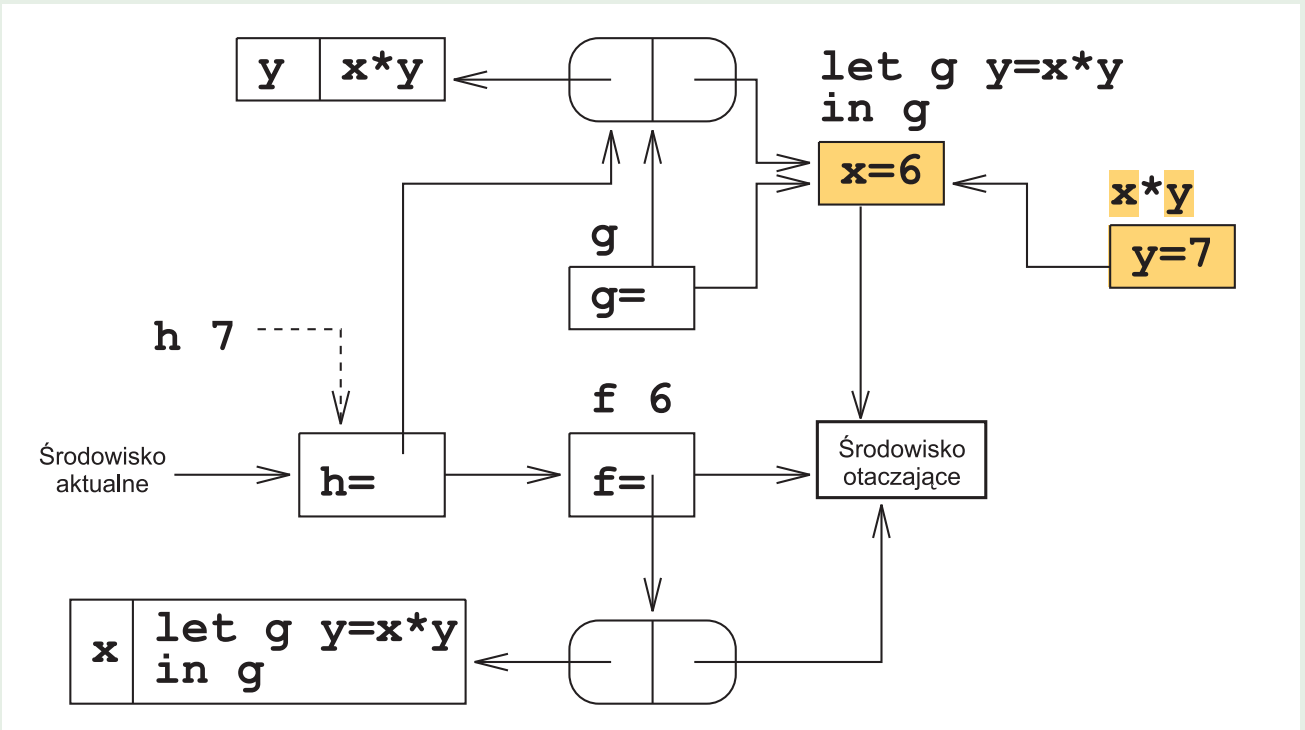


Ramki a rekordy aktywacji

Example

```
let f x =
  let g y = x * y
  in g;;
let h = f 6;;
h 7;;
```

Ramka z definicją g powstaje w wywołaniu f 6, ale jest potrzebna do obliczenia h 7.

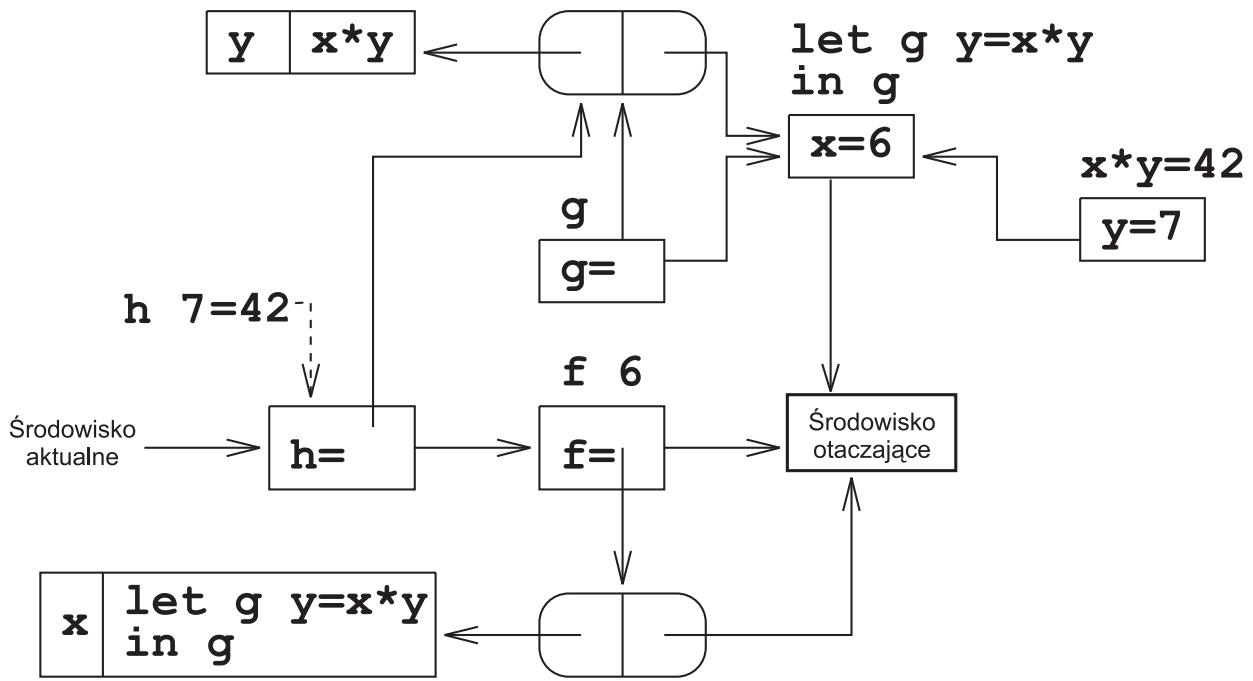


Ramki a rekordy aktywacji

Example

```
let f x =
  let g y = x * y
  in g;;
let h = f 6;;
h 7;;
```

Ramka z definicją g powstaje w wywołaniu f 6, ale jest potrzebna do obliczenia h 7.

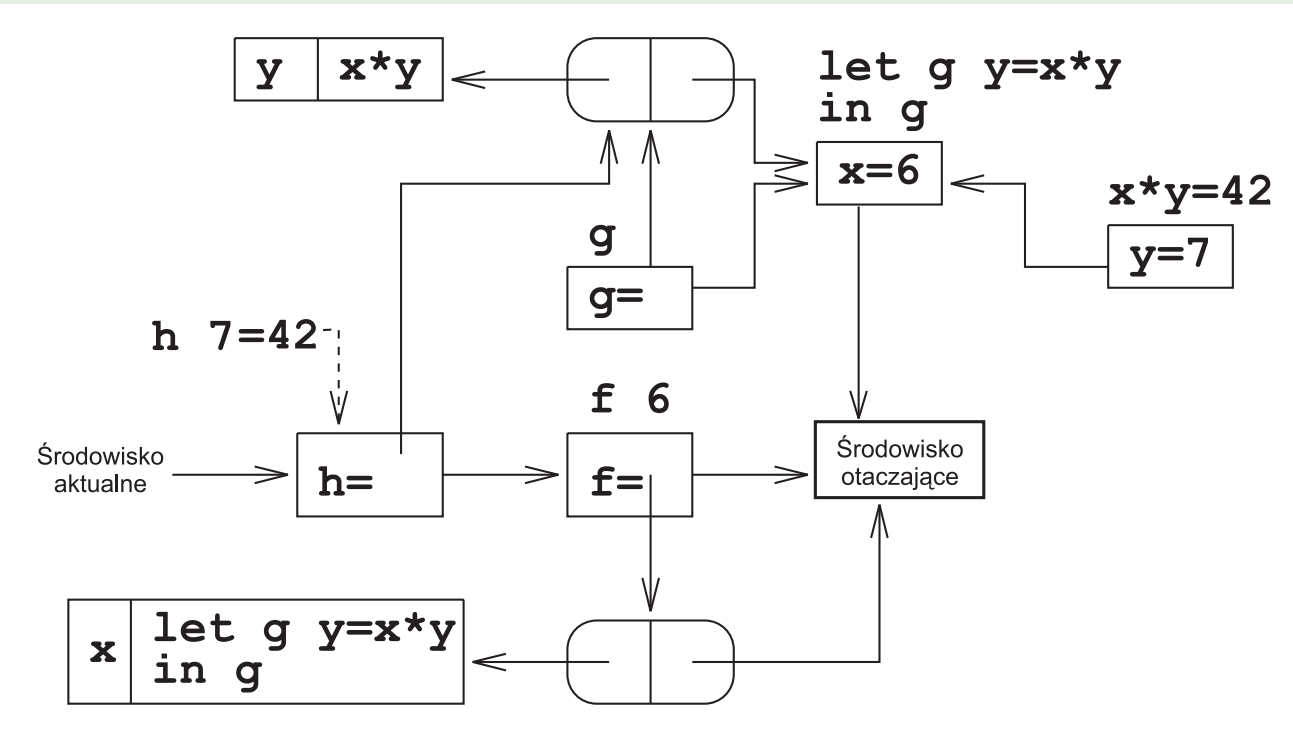


Ramki a rekordy aktywacji

Example

```
let f x =
  let g y = x * y
  in g;;
let h = f 6;;
h 7;;
```

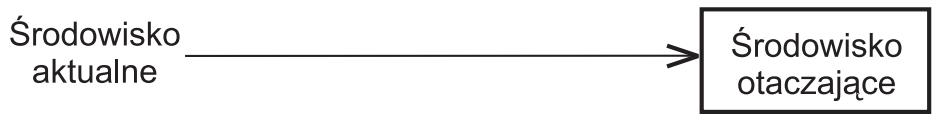
Ramka z definicją g powstaje w wywołaniu f 6, ale jest potrzebna do obliczenia h 7.



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

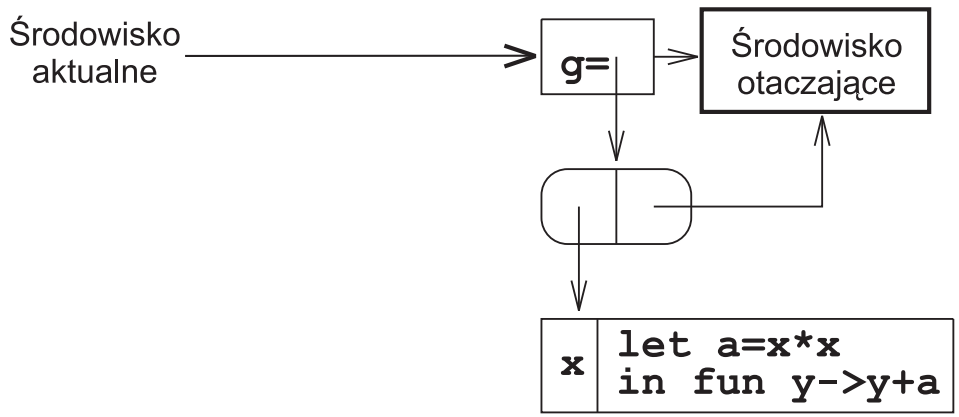
```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

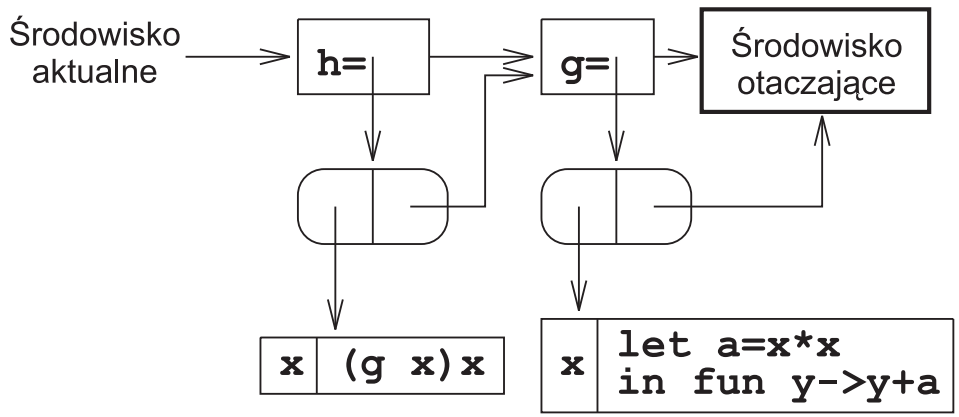
```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

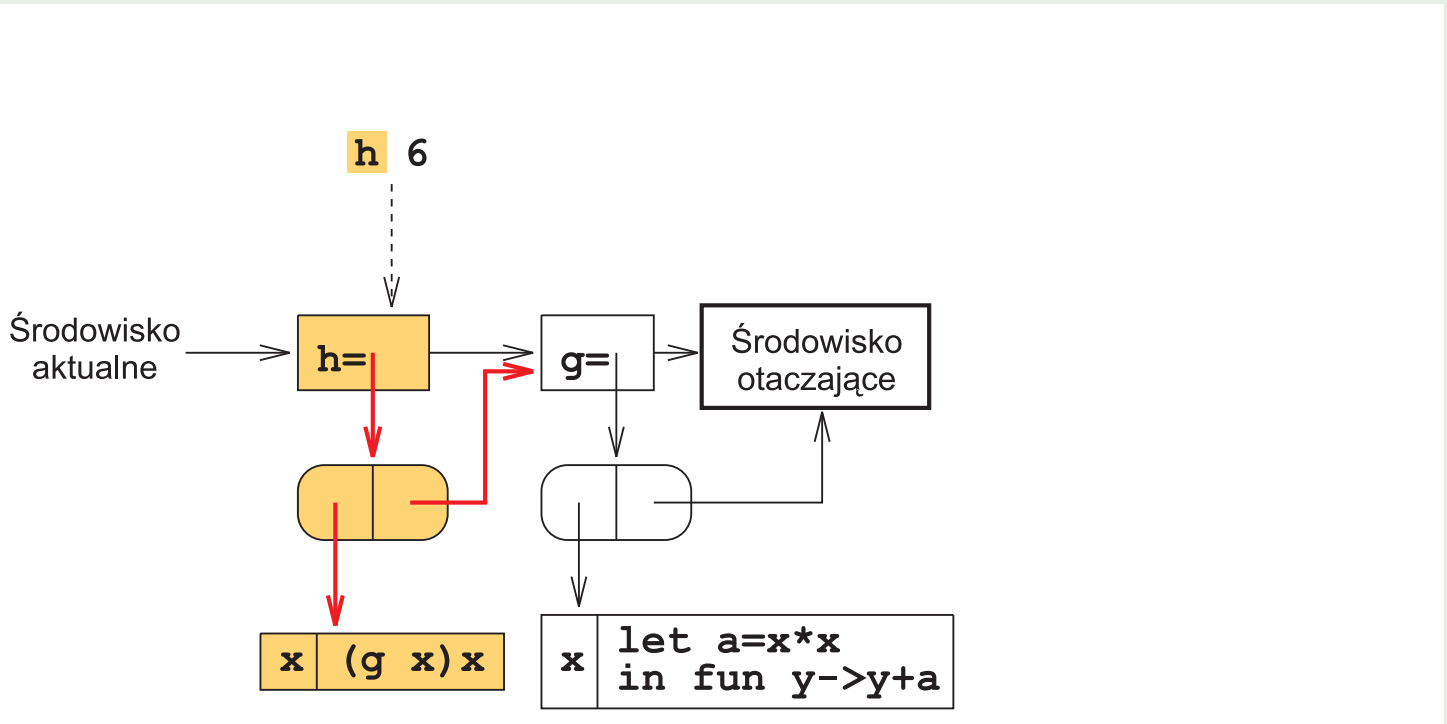
```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

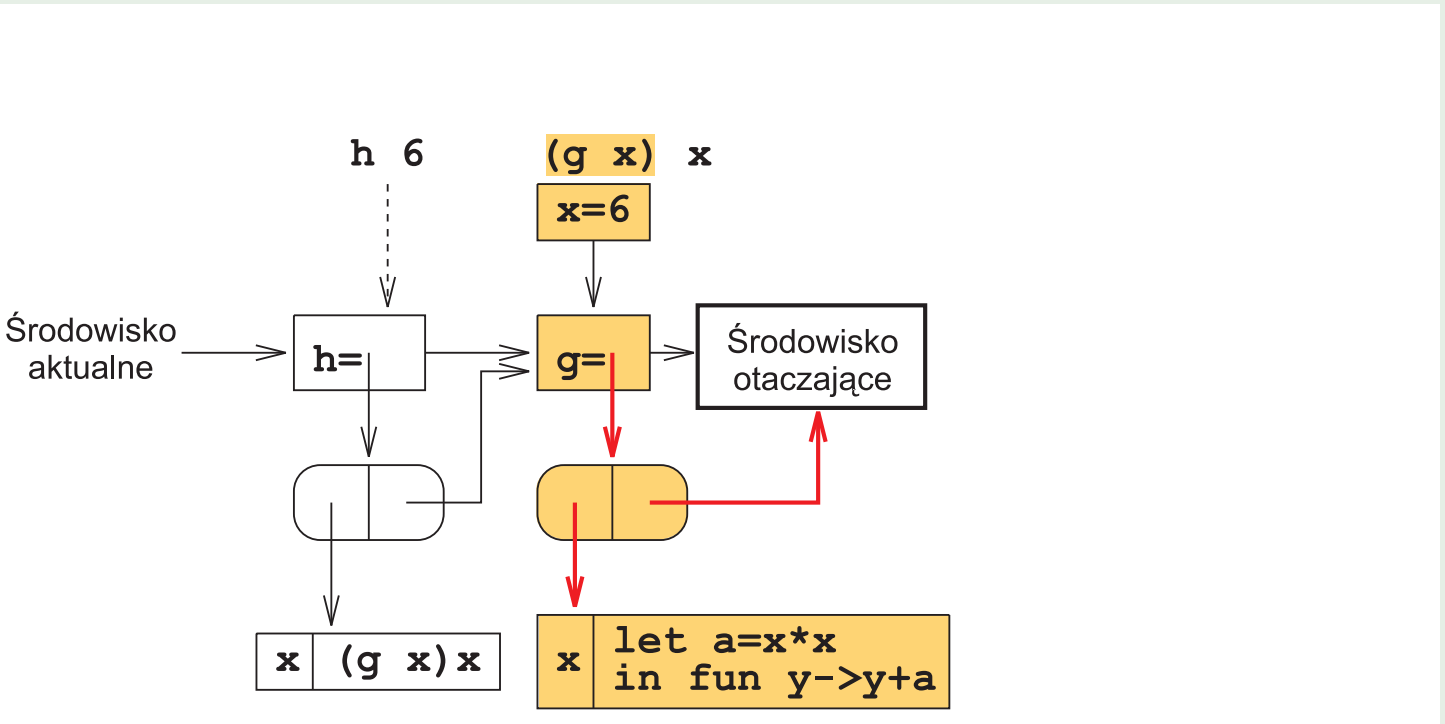
```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

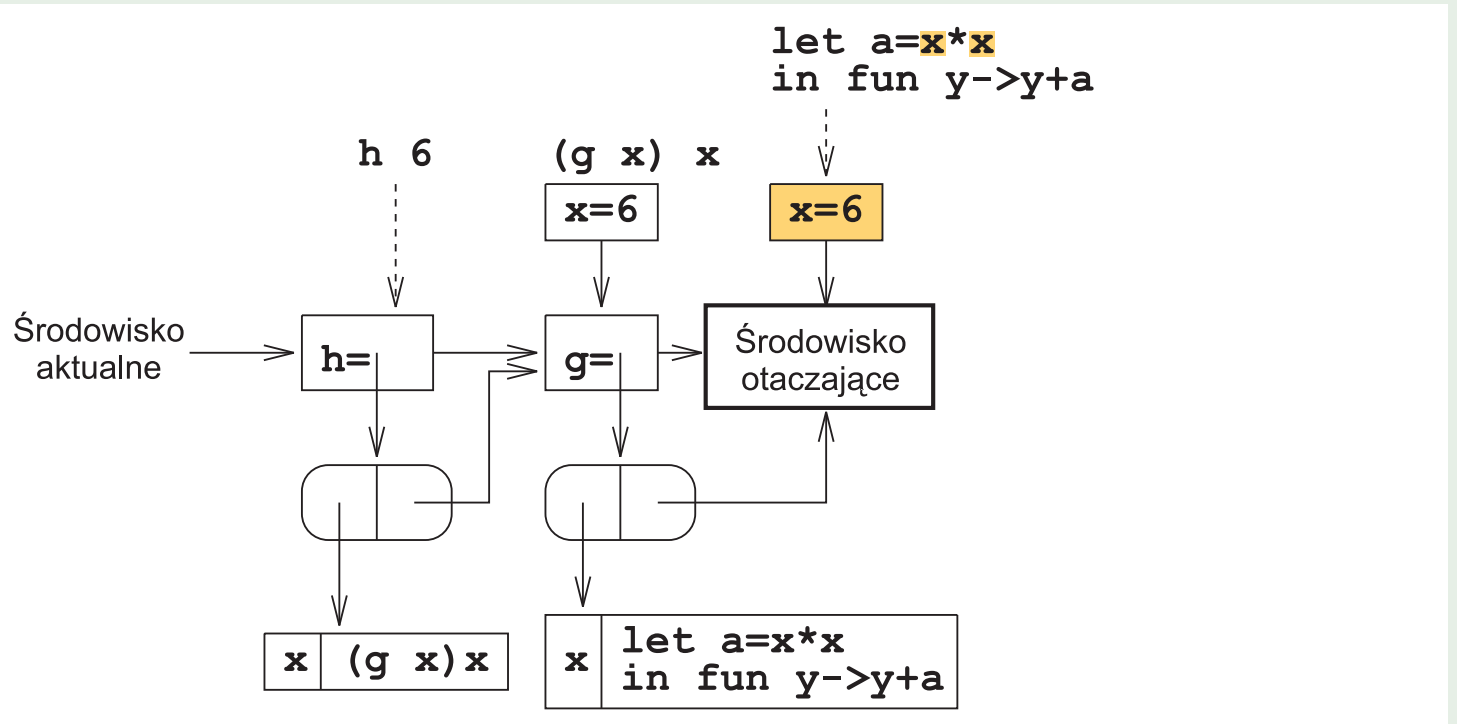
```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

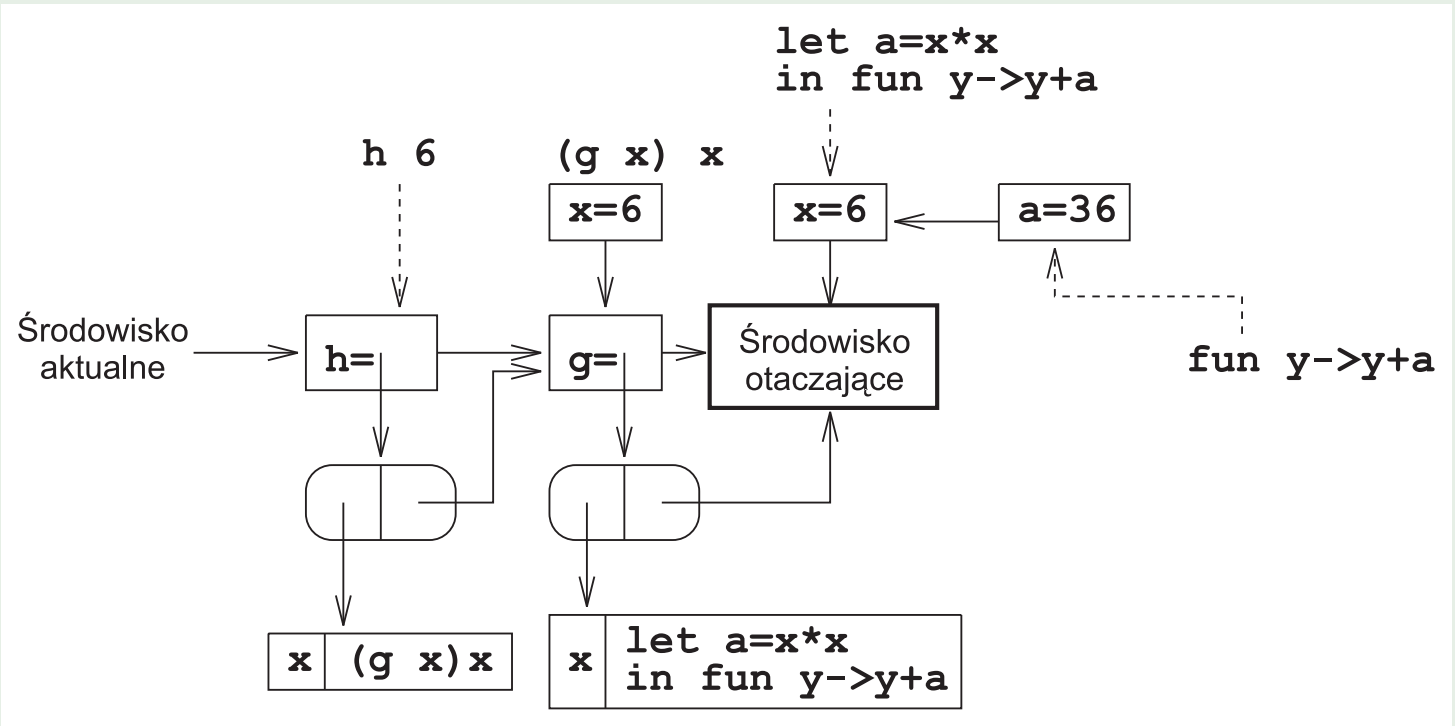
```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

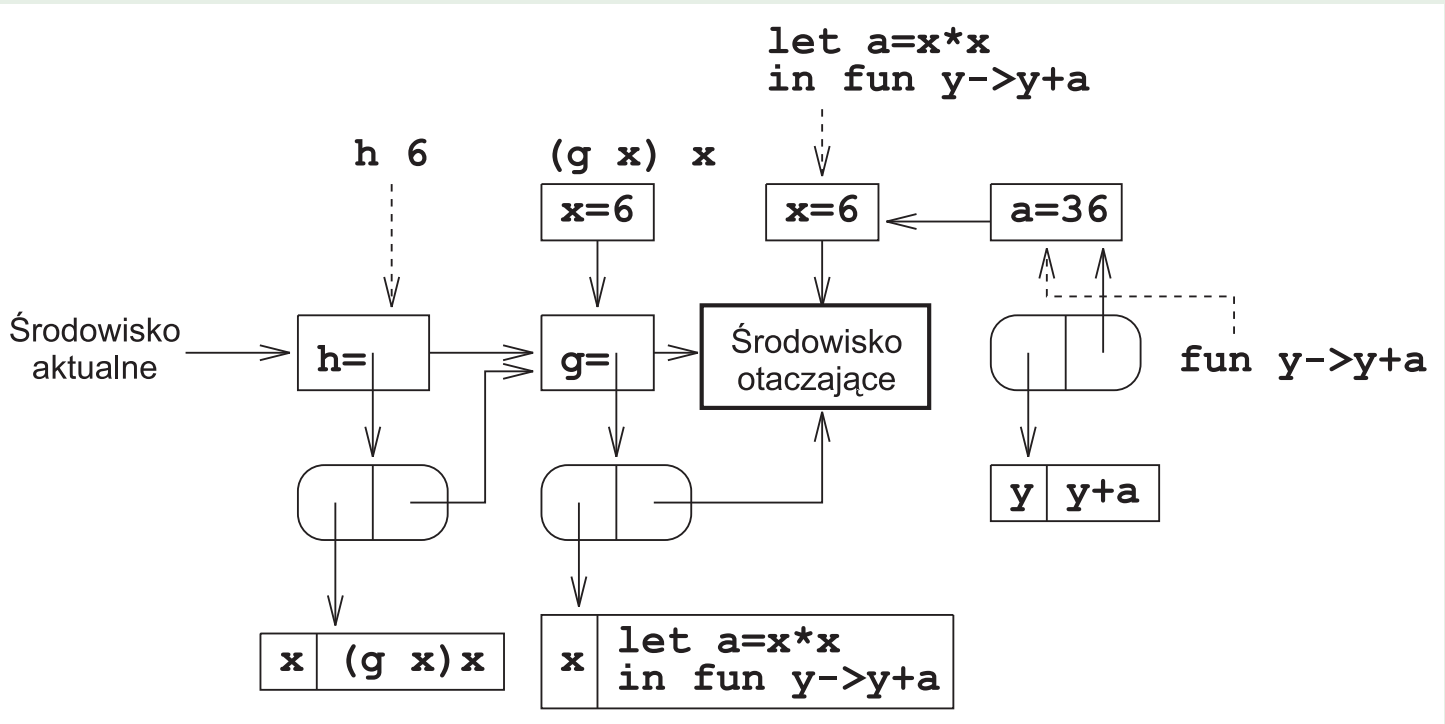
```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

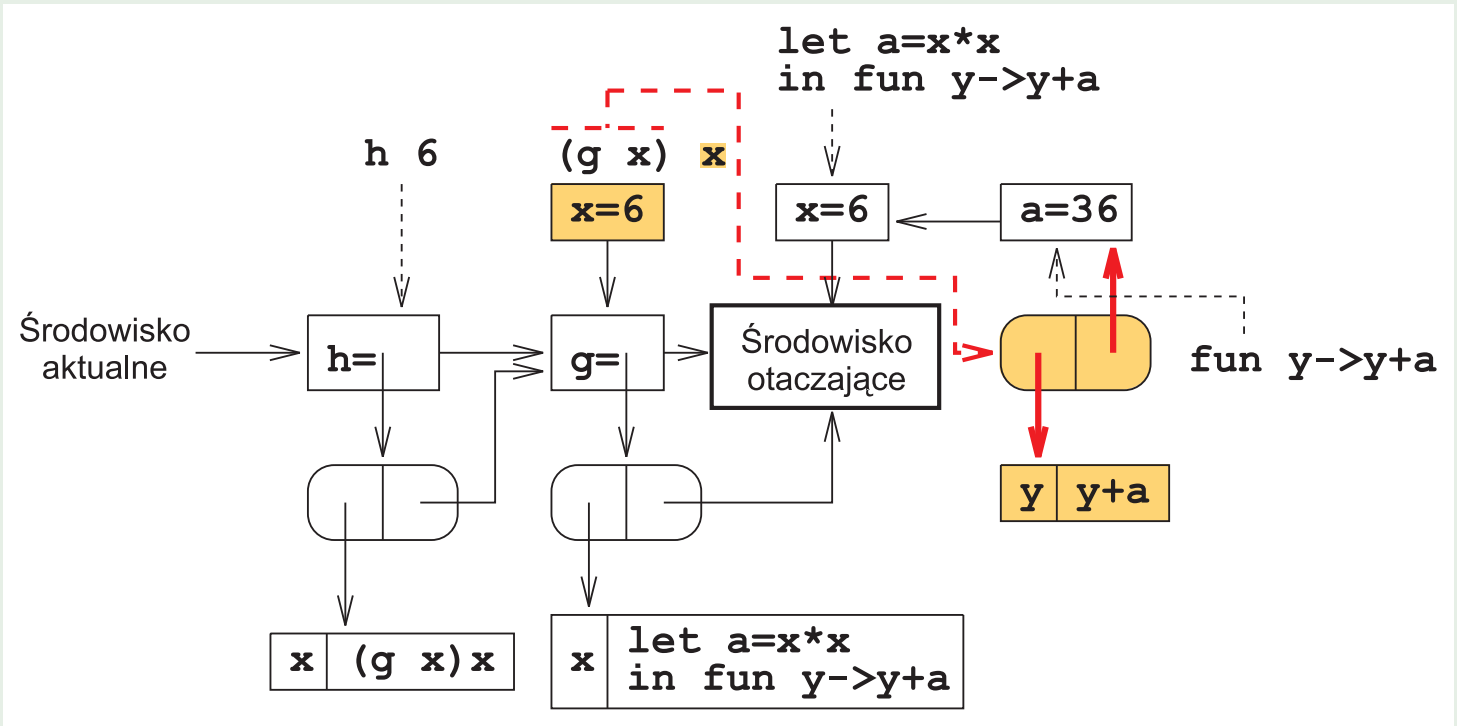
```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

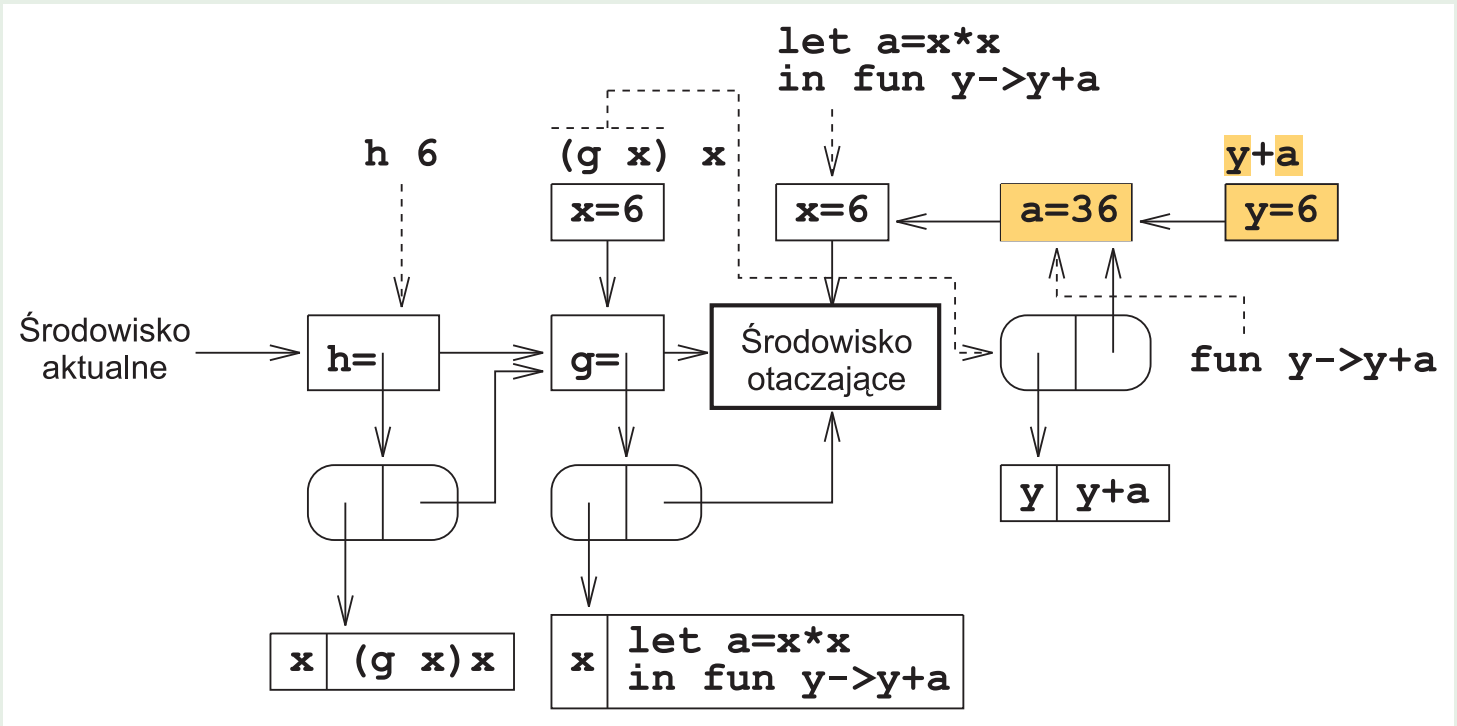
```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

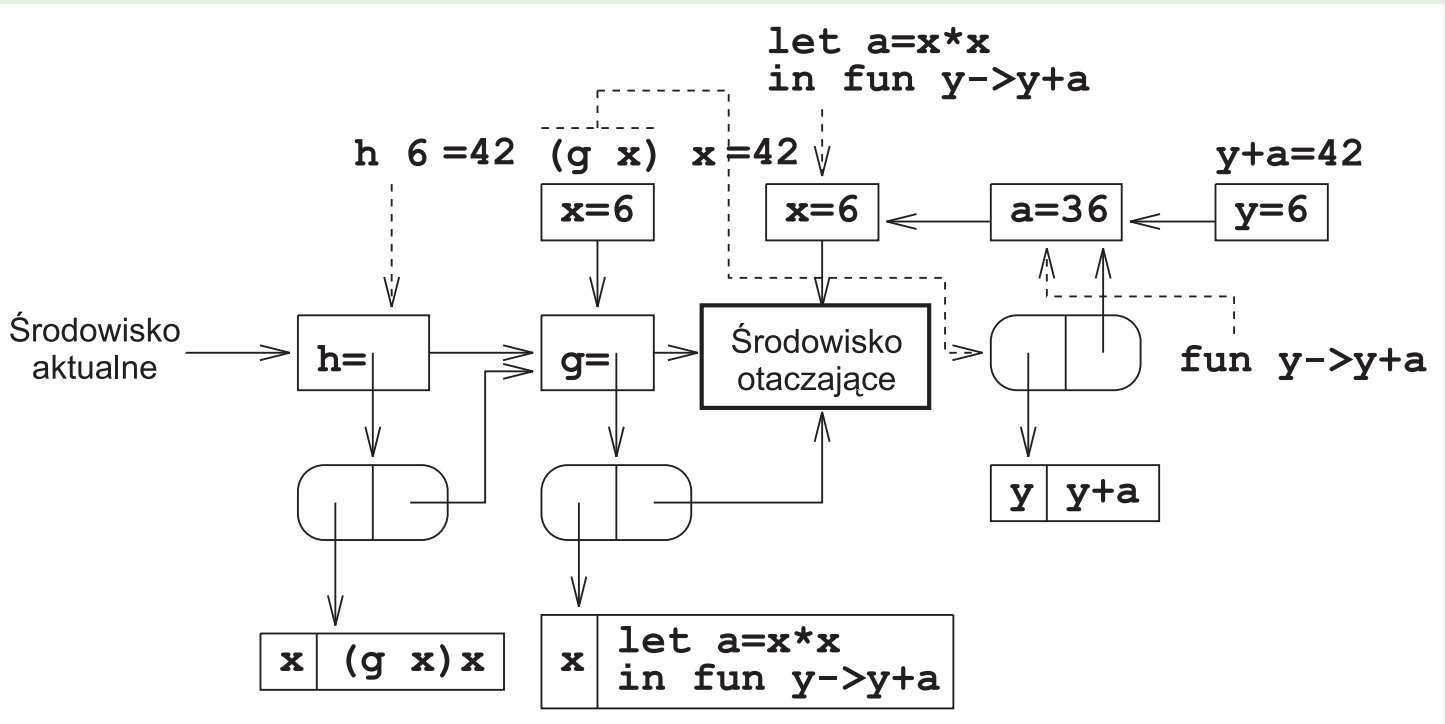
```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Ramki a rekordy aktywacji

Example (Stopniowe przekazywanie argumentów)

```
let g x =
  let a = x * x
  in fun y -> y + a;;
let h x = (g x) x;;
h 6;;
```



Odśmiecanie

Odśmiecanie:

- Uruchamiane, gdy brak wolnej pamięci.
- Usuwa niedostępne ramki i struktury danych.
- Przeszukiwanie grafu wskaźników.
- Punkty startowe — środowiska w których obliczane są właśnie wyrażenia.
- Koszt czasowy odśmiecania można pominąć (jest wliczony w stały koszt przydziału pamięci).
- Koszt pamięciowy:
  - ograniczamy się tylko do potrzebnych fragmentów danych i ramek,
  - nie wliczamy danych,
  - uwzględniamy współdzielone fragmenty danych — skomplikowane,
  - minimalna ilość pamięci konieczna do przeprowadzenia obliczeń.