

# Модел на средите и изчислителни процеси

Трифон Трифонов

Функционално програмиране, 2024/25 г.

9–16 октомври 2024 г.

Тази презентация е достъпна под лиценза Creative Commons Признание-Некомерсиално-Споделяне на споделеното 4.0 Международен 

## Среди в Scheme

- Връзката между символите и техните оценки се записват в речник, който се нарича среда.

## Среди в Scheme

- Връзката между символите и техните оценки се записват в речник, който се нарича среда.
- Всеки символ има най-много една оценка в дадена среда.

## Среди в Scheme

- Връзката между символите и техните оценки се записват в речник, който се нарича среда.
- Всеки символ има най-много една оценка в дадена среда.
- В даден момент могат да съществуват много среди.

## Среди в Scheme

- Връзката между символите и техните оценки се записват в речник, който се нарича среда.
- Всеки символ има най-много една оценка в дадена среда.
- В даден момент могат да съществуват много среди.
- Символите винаги се оценяват в една конкретна среда.

## Среди в Scheme

- Връзката между символите и техните оценки се записват в речник, който се нарича среда.
- Всеки символ има най-много една оценка в дадена среда.
- В даден момент могат да съществуват много среди.
- Символите винаги се оценяват в една конкретна среда.
- **Символите могат да има различни оценки в различни среди.**

## Среди в Scheme

- Връзката между символите и техните оценки се записват в речник, който се нарича среда.
- Всеки символ има най-много една оценка в дадена среда.
- В даден момент могат да съществуват много среди.
- Символите винаги се оценяват в една конкретна среда.
- **Символите могат да има различни оценки в различни среди.**
- При стартиране Scheme по подразбиране работи в глобалната среда.

## Среди в Scheme

- Връзката между символите и техните оценки се записват в речник, който се нарича среда.
- Всеки символ има най-много една оценка в дадена среда.
- В даден момент могат да съществуват много среди.
- Символите винаги се оценяват в една конкретна среда.
- **Символите могат да има различни оценки в различни среди.**
- При стартиране Scheme по подразбиране работи в глобалната среда.
- В глобалната среда са дефинирани символи за стандартни операции и функции.

# Пример за среда

- (define a 8)

E
a : 8

# Пример за среда

- (`define a 8`)
- `r` → Грешка!

E
a : 8

# Пример за среда

- (`define a 8`)
- `r` → Грешка!
- (`define r 5`)

E
a : 8
r : 5

# Пример за среда

- (`define a 8`)
- `r` → Грешка!
- (`define r 5`)
- (`+ r 3`) → 8

E
a : 8
r : 5

# Пример за среда

- (`define a 8`)
- `r` → Грешка!
- (`define r 5`)
- (`+ r 3`) → 8
- (`define (f x) (* x r))`

E	
a	: 8
r	: 5
f	: Параметри : x Тяло : (* x r) Среда : E

# Пример за среда

- (`define a 8`)
- `r` → Грешка!
- (`define r 5`)
- (`+ r 3`) → 8
- (`define (f x) (* x r)`)
- (`f 3`) → 15



# Пример за среда

- (`define a 8`)
- `r` → Грешка!
- (`define r 5`)
- (`+ r 3`) → 8
- (`define (f x) (* x r)`)
- (`f 3`) → 15
- (`f r`) → 25

E	
a	: 8
r	: 5
f	: Параметри : x Тяло : (* x r) Среда : E

# Функции и среди

- Всяка функция  $f$  пази указател към средата  $E$ , в която е дефинирана.

# Функции и среди

- Всяка функция  $f$  пази указател към средата  $E$ , в която е дефинирана.
- При извикване на  $f$ :

# Функции и среди

- Всяка функция  $f$  пази указател към средата  $E$ , в която е дефинирана.
- При извикване на  $f$ :
  - създава се нова среда  $E_1$ , която разширява  $E$

# Функции и среди

- Всяка функция  $f$  пази указател към средата  $E$ , в която е дефинирана.
- При извикване на  $f$ :
  - създава се нова среда  $E_1$ , която разширява  $E$
  - в  $E_1$  всеки символ означаващ формален параметър се свързва с оценката на фактическия параметър

# Функции и среди

- Всяка функция  $f$  пази указател към средата  $E$ , в която е дефинирана.
- При извикване на  $f$ :
  - създава се нова среда  $E_1$ , която разширява  $E$
  - в  $E_1$  всеки символ означаващ формален параметър се свързва с оценката на фактическия параметър
  - тялото на  $f$  се оценява в  $E_1$

# Дърво от среди

- Всяка среда пази указател към своя „родителска среда“, която разширява

# Дърво от среди

- Всяка среда пази указател към своя „родителска среда“, която разширява
- така се получава дърво от среди

# Дърво от среди

- Всяка среда пази указател към своя „родителска среда“, която разширява
- така се получава дърво от среди
- при оценка на символ в дадена среда E

# Дърво от среди

- Всяка среда пази указател към своя „родителска среда“, която разширява
- така се получава дърво от среди
- при оценка на символ в дадена среда E
  - първо се търси оценката му в E

# Дърво от среди

- Всяка среда пази указател към своя „родителска среда“, която разширява
- така се получава дърво от среди
- при оценка на символ в дадена среда E
  - първо се търси оценката му в E
  - ако символът не е дефиниран в E, се преминава към родителската среда

# Дърво от среди

- Всяка среда пази указател към своя „родителска среда“, която разширява
- така се получава дърво от среди
- при оценка на символ в дадена среда E
  - първо се търси оценката му в E
  - ако символът не е дефиниран в E, се преминава към родителската среда
  - при достигане на най-горната среда, ако символът не е дефиниран и в нея се извежда съобщение за грешка

# Извикване на дефинирана функция

- (`define r 5`)

E
r : 5

# Извикване на дефинирана функция

- (`define r 5`)
- (`define a 3`)

E
r : 5
a : 3

# Извикване на дефинирана функция

- `(define r 5)`
- `(define a 3)`
- `(define (f x) (* x r))`

E	
r	: 5
a	: 3
Параметри : x	
f	: Тяло : (* x r)
Среда : E	

# Извикване на дефинирана функция

- `(define r 5)`
- `(define a 3)`
- `(define (f x) (* x r))`

{E}      (f a)

E	
r	: 5
a	: 3
f	: Параметри : x Тяло : (* x r) Среда : E

# Извикване на дефинирана функция

- (`define r 5`)
- (`define a 3`)
- (`define (f x) (* x r))`

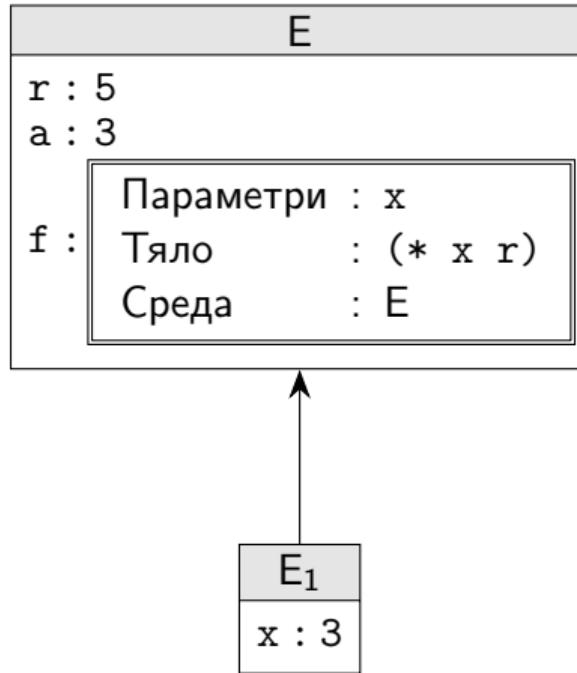
{E}      (f a)  
           ↓  
{E}      (f 3)

E	
r	: 5
a	: 3
Параметри : x	
f	: Тяло : (* x r)
Среда : E	

# Извикване на дефинирана функция

- (`define r 5`)
- (`define a 3`)
- (`define (f x) (* x r)`)

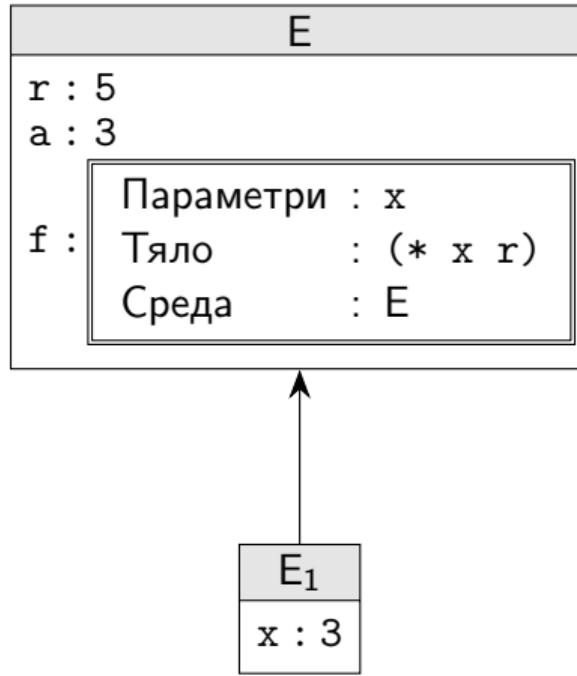
{E}        (f a)  
             ↓  
{E}        (f 3)  
             ↓  
{E<sub>1</sub>}     (\* x r)



# Извикване на дефинирана функция

- (`define r 5`)
- (`define a 3`)
- (`define (f x) (* x r)`)

{E}        (f a)  
             ↓  
{E}        (f 3)  
             ↓  
{E<sub>1</sub>}     (\* x r)  
             ↓  
         15



# Какво е рекурсия?

# Какво е рекурсия?



"Matryoshka dolls" от User:Fanghong (оригинал) и User:Gnomz007 (производна), CC BY SA-3.0

# Какво е рекурсия?



"Sierpinsk triangle, the evolution in five iterations" от Solkoll, Обществено достояние чрез Общомедия

# Какво е рекурсия?

# Какво е рекурсия?

Повторение чрез позоваване на себе си

# Какво е рекурсия?

Повторение чрез позоваване на себе си

Рекурсивна функция: дефинира се чрез себе си

$$n! = \begin{cases} 1, & \text{при } n = 0, \\ n \cdot (n - 1)!, & \text{при } n > 0. \end{cases} \quad \begin{array}{l} \text{(база)} \\ \text{(стъпка)} \end{array}$$

# Какво е рекурсия?

Повторение чрез позоваване на себе си

Рекурсивна функция: дефинира се чрез себе си

$$n! = \begin{cases} 1, & \text{при } n = 0, \\ n \cdot (n - 1)!, & \text{при } n > 0. \end{cases} \quad \begin{array}{l} \text{(база)} \\ \text{(стъпка)} \end{array}$$

- Дава се отговор на най-простата задача (база, дъно)
- Показва се как сложна задача се свежда към една или няколко по-прости задачи от същия вид (стъпка)

# Рекурсивни уравнения

Какво означава „да дефинираме функция чрез себе си“?

# Рекурсивни уравнения

$$f(x) = x^2 - 2$$

$$x = f(x)$$

$$x = x^2 - 2$$

$$x^2 - x - 2 = 0$$

$$x = 2$$

$$x = -1$$

Какво означава „да дефинираме функция чрез себе си“?

Да разгледаме *рекурсивното уравнение*, в което  $F$  е неизвестно:

$$F(n) = \underbrace{\begin{cases} 1, & \text{при } n = 0, \\ n \cdot F(n-1), & \text{при } n > 0. \end{cases}}_{\Gamma(F)(n)}$$

# Рекурсивни уравнения

Какво означава „да дефинираме функция чрез себе си“?

Да разгледаме *рекурсивното уравнение*, в което  $F$  е неизвестно:

$$F(n) = \underbrace{\begin{cases} 1, & \text{при } n = 0, \\ n \cdot F(n - 1), & \text{при } n > 0. \end{cases}}_{\Gamma(F)(n)}$$

$n!$  е „най-малкото“ решение на уравнението  $F = \Gamma(F)$ .

# Най-малка неподвижна точка

## Теорема (Knaster-Tarski)

Ако  $\Gamma$  е изчислим оператор, то уравнението  $F = \Gamma(F)$  има единствено най-малко решение  $f$

# Най-малка неподвижна точка

## Теорема (Knaster-Tarski)

Ако  $\Gamma$  е изчислим оператор, то уравнението  $F = \Gamma(F)$  има единствено най-малко решение  $f$  (най-малка неподвижна точка на  $\Gamma$ ).

# Най-малка неподвижна точка

## Теорема (Knaster-Tarski)

Ако  $\Gamma$  е изчислим оператор, то уравнението  $F = \Gamma(F)$  има единствено най-малко решение  $f$  (**най-малка неподвижна точка на  $\Gamma$** ). Нещо повече, решението точно съответства на рекурсивна програма пресмятща  $f$  чрез  $\Gamma$ .

# Най-малка неподвижна точка

## Теорема (Knaster-Tarski)

Ако  $\Gamma$  е изчислим оператор, то уравнението  $F = \Gamma(F)$  има единствено най-малко решение  $f$  (**най-малка неподвижна точка на  $\Gamma$** ). Нещо повече, решението точно съответства на рекурсивна програма пресмятща  $f$  чрез  $\Gamma$ .

```
(define (fact n)
  (if (= n 0) 1
      (* n (fact (- n 1)))))
```

# Най-малка неподвижна точка

## Теорема (Knaster-Tarski)

Ако  $\Gamma$  е изчислим оператор, то уравнението  $F = \Gamma(F)$  има единствено най-малко решение  $f$  (най-малка неподвижна точка на  $\Gamma$ ). Нещо повече, решението точно съответства на рекурсивна програма пресмятща  $f$  чрез  $\Gamma$ .

```
(define (fact n)
  (if (= n 0) 1
      (* n (fact (- n 1)))))
```

$$F(x) = x + C$$

Кое е най-малкото решение на уравнението  $F(x) \approx F(x + 1) - 1$ ?

# Най-малка неподвижна точка

## Теорема (Knaster-Tarski)

Ако  $\Gamma$  е изчислим оператор, то уравнението  $F = \Gamma(F)$  има единствено най-малко решение  $f$  (**най-малка неподвижна точка на  $\Gamma$** ). Нещо повече, решението точно съответства на рекурсивна програма пресмятща  $f$  чрез  $\Gamma$ .

```
(define (fact n)
  (if (= n 0) 1
      (* n (fact (- n 1))))))
```

Кое е **най-малкото решение** на уравнението  $F(x) = F(x + 1) - 1$ ?

```
(define (g x) (- 1 (g (+ x 1))))
(g 0) —> ?
```

# Най-малка неподвижна точка

## Теорема (Knaster-Tarski)

Ако  $\Gamma$  е изчислим оператор, то уравнението  $F = \Gamma(F)$  има единствено най-малко решение  $f$  (най-малка неподвижна точка на  $\Gamma$ ). Нещо повече, решението точно съответства на рекурсивна програма пресмятща  $f$  чрез  $\Gamma$ .

```
(define (fact n)
  (if (= n 0) 1
      (* n (fact (- n 1))))))
```

Кое е най-малкото решение на уравнението  $F(x) = F(x + 1) - 1$ ?

```
(define (g x) (- 1 (g (+ x 1))))
```

$(g 0) \rightarrow ?$

$g$  е „празната функция“, т.е.  $\text{dom}(g) = \emptyset$ .

# Оценка на рекурсивна функция

(fact 4)

# Оценка на рекурсивна функция

```
(fact 4)
      ↓
(if (= 4 0) 1 (* 4 (fact (- 4 1))))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (if (= 3 0) 1 (* 3 (fact (- 3 1)))))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (* 3 (fact 2)))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (* 3 (fact 2)))
  ↓
(* 4 (* 3 (if (= 2 0) 1 (* 2 (fact (- 2 1)))))))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (* 3 (fact 2)))
  ↓
(* 4 (* 3 (* 2 (fact 1))))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (* 3 (fact 2)))
  ↓
(* 4 (* 3 (* 2 (fact 1))))
  ↓
(* 4 (* 3 (* 2 (if (= 1 0) 1 (* 1 (fact (- 1 1))))))))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (* 3 (fact 2)))
  ↓
(* 4 (* 3 (* 2 (fact 1))))
  ↓
(* 4 (* 3 (* 2 (* 1 (fact 0)))))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (* 3 (fact 2)))
  ↓
(* 4 (* 3 (* 2 (fact 1))))
  ↓
(* 4 (* 3 (* 2 (* 1 (fact 0)))))
  ↓
(* 4 (* 3 (* 2 (* 1 (if (= 0 0) 1 (* 0 (fact (- 0 1)))))))))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (* 3 (fact 2)))
  ↓
(* 4 (* 3 (* 2 (fact 1))))
  ↓
(* 4 (* 3 (* 2 (* 1 (fact 0)))))
  ↓
(* 4 (* 3 (* 2 (* 1 1))))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (* 3 (fact 2)))
  ↓
(* 4 (* 3 (* 2 (fact 1))))
  ↓
(* 4 (* 3 (* 2 (* 1 (fact 0)))))
  ↓
(* 4 (* 3 (* 2 (* 1 1))))
  ↓
(* 4 (* 3 (* 2 1)))
```

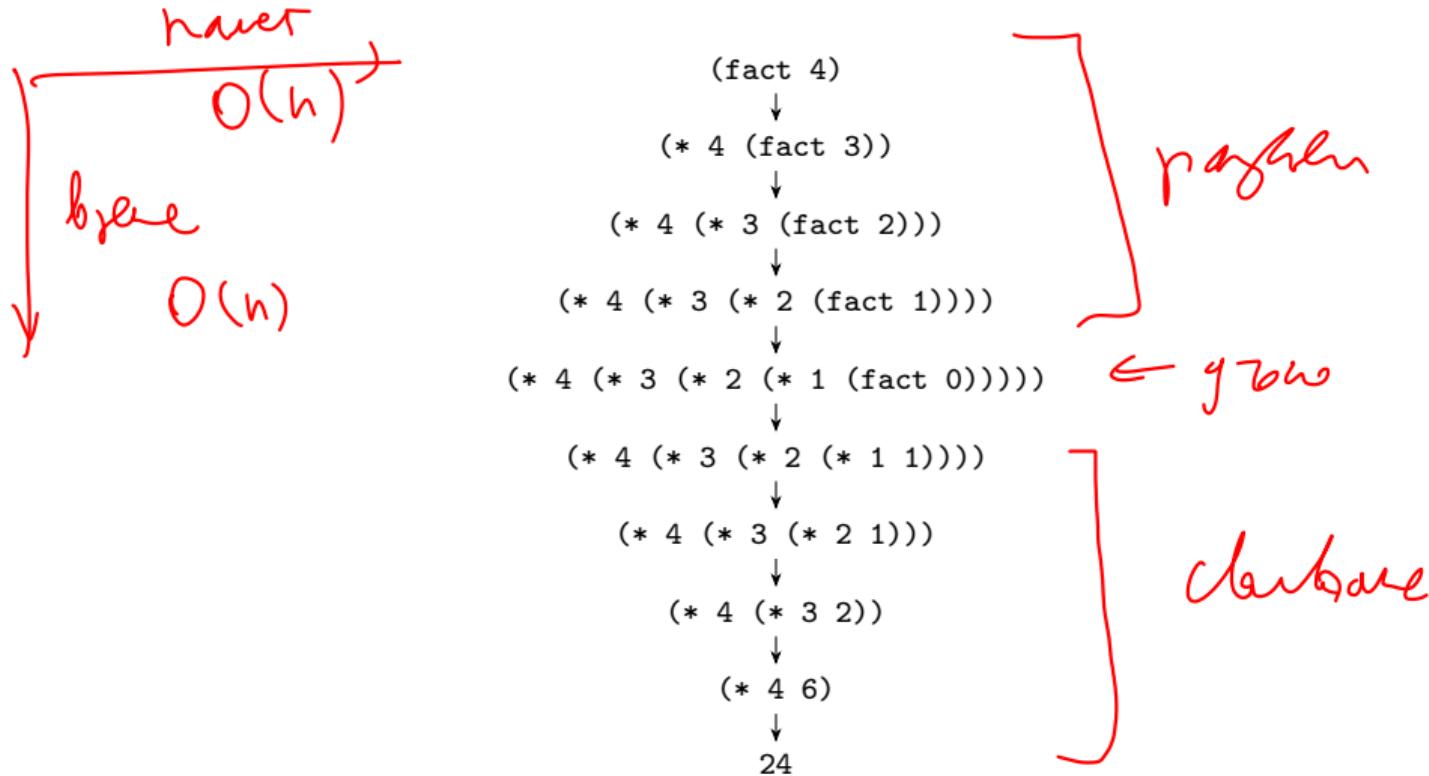
# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (* 3 (fact 2)))
  ↓
(* 4 (* 3 (* 2 (fact 1))))
  ↓
(* 4 (* 3 (* 2 (* 1 (fact 0)))))
  ↓
(* 4 (* 3 (* 2 (* 1 1))))
  ↓
(* 4 (* 3 (* 2 1)))
  ↓
(* 4 (* 3 2))
```

# Оценка на рекурсивна функция

```
(fact 4)
  ↓
(* 4 (fact 3))
  ↓
(* 4 (* 3 (fact 2)))
  ↓
(* 4 (* 3 (* 2 (fact 1))))
  ↓
(* 4 (* 3 (* 2 (* 1 (fact 0)))))
  ↓
(* 4 (* 3 (* 2 (* 1 1))))
  ↓
(* 4 (* 3 (* 2 1)))
  ↓
(* 4 (* 3 2))
  ↓
(* 4 6)
```

# Оценка на рекурсивна функция



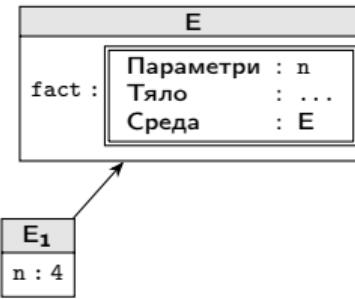
# Оценка на рекурсивна функция в среда

{E} (fact 4)



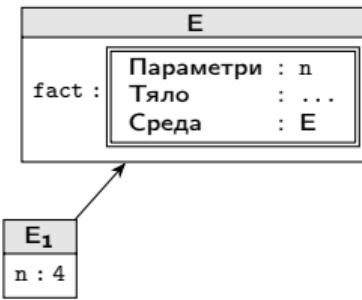
# Оценка на рекурсивна функция в среда

{E}            (fact 4)  
              ↓  
{E<sub>1</sub>}        (if (= n 0) 1 (\* n (fact (- n 1))))



# Оценка на рекурсивна функция в среда

{E}      (fact 4)  
        ↓  
{E<sub>1</sub>}      (\* 4 (fact 3))

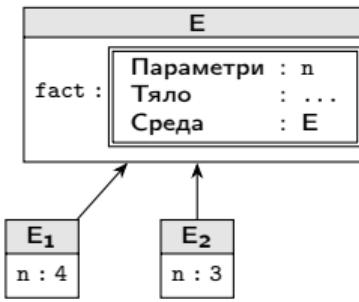


# Оценка на рекурсивна функция в среда

```

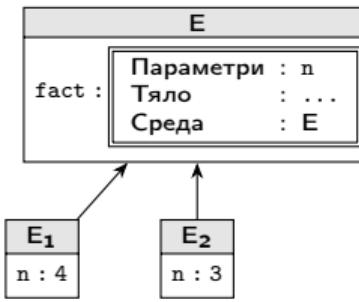
{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (if (= n 0) 1 (* n (fact (- n 1))))))

```



# Оценка на рекурсивна функция в среда

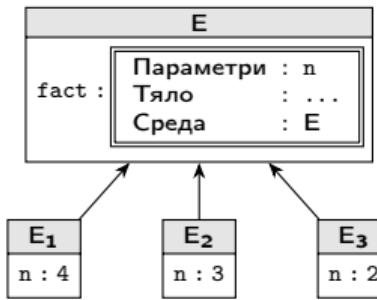
```
{E}      (fact 4)
        ↓
{E1}    (* 4 (fact 3))
        ↓
{E2}    (* 4 (* 3 (fact 2)))
```



# Оценка на рекурсивна функция в среда

```

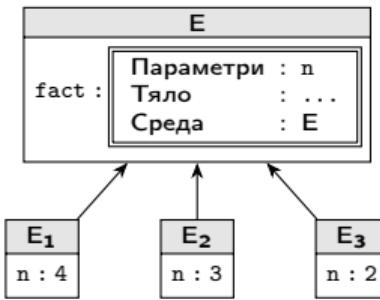
{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (* 3 (fact 2)))
         ↓
{E3}    (* 4 (* 3 (if (= n 0) 1 (* n (fact (- n 1)))))))
  
```



# Оценка на рекурсивна функция в среда

```

{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (* 3 (fact 2)))
         ↓
{E3}    (* 4 (* 3 (* 2 (fact 1))))
  
```

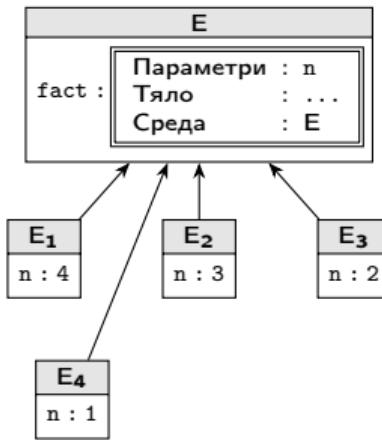


# Оценка на рекурсивна функция в среда

```

{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (* 3 (fact 2)))
         ↓
{E3}    (* 4 (* 3 (* 2 (fact 1))))
         ↓
{E4}    (* 4 (* 3 (* 2 (if (= n 0) 1 (* n (fact (- n 1)))))))

```

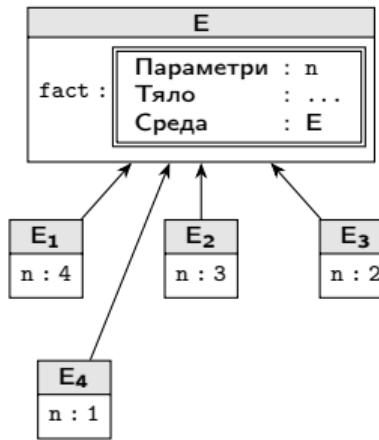


# Оценка на рекурсивна функция в среда

```

{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (* 3 (fact 2)))
         ↓
{E3}    (* 4 (* 3 (* 2 (fact 1))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 (fact 0)))))


```

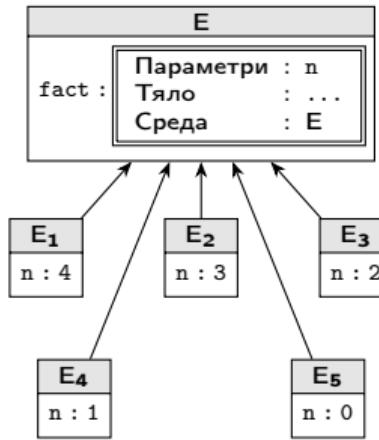


# Оценка на рекурсивна функция в среда

```

{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (* 3 (fact 2)))
         ↓
{E3}    (* 4 (* 3 (* 2 (fact 1))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 (fact 0)))))
         ↓
{E5}    (* 4 (* 3 (* 2 (* 1 (if (= n 0) 1 (* n (fact (- n 1))))))))

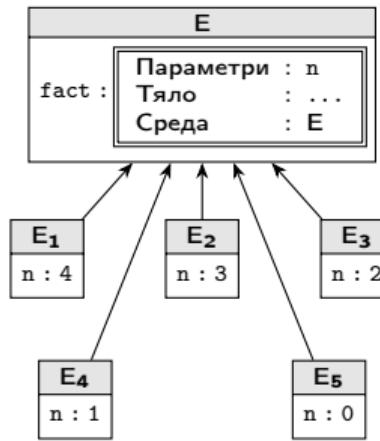
```



# Оценка на рекурсивна функция в среда

```

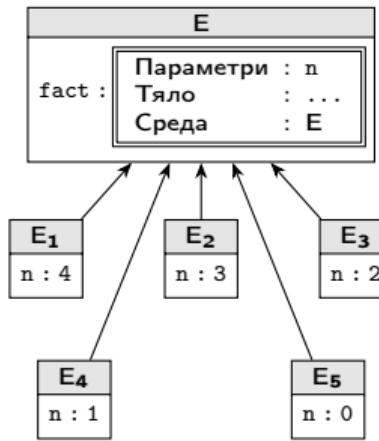
{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (* 3 (fact 2)))
         ↓
{E3}    (* 4 (* 3 (* 2 (fact 1))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 (fact 0)))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 1))))
  
```



# Оценка на рекурсивна функция в среда

```

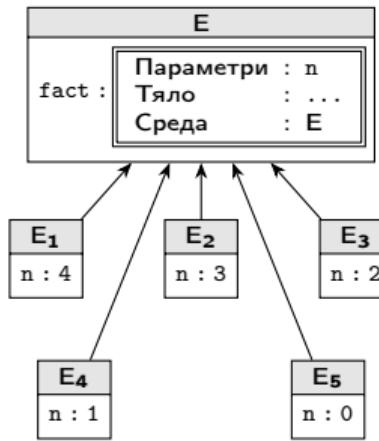
{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (* 3 (fact 2)))
         ↓
{E3}    (* 4 (* 3 (* 2 (fact 1))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 (fact 0)))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 1))))
         ↓
{E3}    (* 4 (* 3 (* 2 1)))
  
```



# Оценка на рекурсивна функция в среда

```

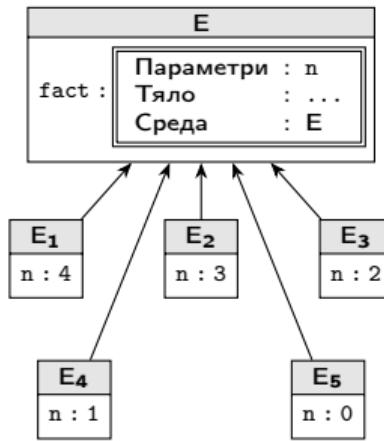
{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (* 3 (fact 2)))
         ↓
{E3}    (* 4 (* 3 (* 2 (fact 1))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 (fact 0)))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 1))))
         ↓
{E3}    (* 4 (* 3 (* 2 1)))
         ↓
{E2}    (* 4 (* 3 2))
  
```



# Оценка на рекурсивна функция в среда

```

{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (* 3 (fact 2)))
         ↓
{E3}    (* 4 (* 3 (* 2 (fact 1))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 (fact 0)))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 1))))
         ↓
{E3}    (* 4 (* 3 (* 2 1)))
         ↓
{E2}    (* 4 (* 3 2))
         ↓
{E1}    (* 4 6)
  
```

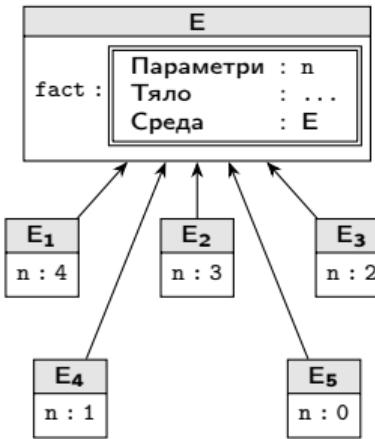


# Оценка на рекурсивна функция в среда



```

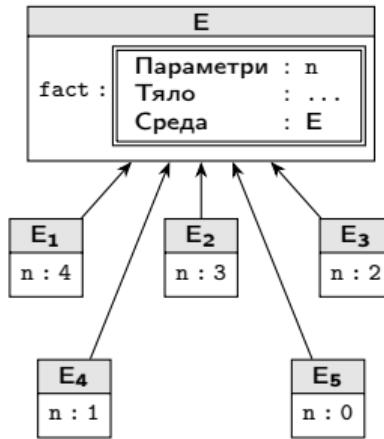
{E}      (fact 4)
         ↓
{E1}     (* 4 (fact 3))
         ↓
{E2}     (* 4 (* 3 (fact 2)))
         ↓
{E3}     (* 4 (* 3 (* 2 (fact 1))))
         ↓
{E4}     (* 4 (* 3 (* 2 (* 1 (fact 0)))))
         ↓
{E4}     (* 4 (* 3 (* 2 (* 1 1))))
         ↓
{E3}     (* 4 (* 3 (* 2 1)))
         ↓
{E2}     (* 4 (* 3 2))
         ↓
{E1}     (* 4 6)
         ↓
{E}      24
  
```



# Оценка на рекурсивна функция в среда

```

{E}      (fact 4)
         ↓
{E1}    (* 4 (fact 3))
         ↓
{E2}    (* 4 (* 3 (fact 2)))
         ↓
{E3}    (* 4 (* 3 (* 2 (fact 1))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 (fact 0)))))
         ↓
{E4}    (* 4 (* 3 (* 2 (* 1 1))))
         ↓
{E3}    (* 4 (* 3 (* 2 1)))
         ↓
{E2}    (* 4 (* 3 2))
         ↓
{E1}    (* 4 6)
         ↓
{E}      24
  
```



## Линеен рекурсивен процес

# Факториел с цикъл

## Факториел на C++

```
int fact(int n)  {
    int r = 1;
    for( int i = 1 ; i <= n ; i++ )
        r *= i;
    return r;
}
```

# Факториел с цикъл

## Факториел на C++

```
int fact(int n) {  
    int r = 1;  
    for( int i = 1 ; i <= n ; i++ )  
        r *= i;  
    return r;  
}
```

## Превод на Scheme

```
(define (for n r i)  
  (if (<= i n)  
      (for n (* r i) (+ i 1))  
      r))  
  
(define (fact n)  
  (for n 1 1))
```

# Факториел с цикъл

## Факториел на C++

```
int fact(int n) {  
    int r = 1;  
  
    for( int i = 1 ; i <= n ; i++ )  
        r *= i;  
  
    return r;  
}
```

## Превод на Scheme

```
(define (for n r i )  
  (if  (<= i n)  
      (for n (* r i) (+ i 1) )  
      r ))  
  
(define (fact n)  
  (for n 1 1))
```

# Факториел с цикъл

## Факториел на C++

```
int fact(int n) {  
    int r = 1;  
  
    for( int i = 1 ; i <= n ; i++ )  
        r *= i;  
  
    return r;  
}
```

## Превод на Scheme

```
(define (for n [r] i )  
  (if  (<= i n)  
      (for n (* r i) (+ i 1) )  
      r ))  
  
(define (fact n)  
  (for n [1] 1 ))
```

# Факториел с цикъл

## Факториел на C++

```
int fact(int n) {  
    int r = 1;  
  
    for(int i = 1; i <= n ; i++)  
        r *= i;  
  
    return r;  
}
```

## Превод на Scheme

```
(define (for n r i)  
  (if (<= i n)  
      (for n (* r i) (+ i 1) )  
      r ))  
  
(define (fact n)  
  (for n 1 1))
```

# Факториел с цикъл

## Факториел на C++

```
int fact(int n) {  
    int r = 1;  
    for( int i = 1 ; [i <= n]; i++ )  
        r *= i;  
    return r;  
}
```

## Превод на Scheme

```
(define (for n r i )  
  (if (<= i n)  
      (for n (* r i) (+ i 1) )  
      r ))  
  
(define (fact n)  
  (for n 1 1))
```

# Факториел с цикъл

## Факториел на C++

```
int fact(int n) {  
    int r = 1;  
    for( int i = 1 ; i <= n ; i++)  
        r *= i;  
    return r;  
}
```

## Превод на Scheme

```
(define (for n r i)  
  (if (<= i n)  
      (for n (* r i) (+ i 1))  
      r))  
  
(define (fact n)  
  (for n 1 1))
```

# Факториел с цикъл

## Факториел на C++

```
int fact(int n) {  
    int r = 1;  
    for( int i = 1 ; i <= n ; i++ )  
        r *= i;  
    return r;  
}
```

## Превод на Scheme

```
(define (for n [r] i )  
  (if  (<= i n)  
      (for n [(* r i)] (+ i 1) )  
      r ))  
  
(define (fact n)  
  (for n 1 1))
```

# Факториел с цикъл

## Факториел на C++

```
int fact(int n) {  
    int r = 1;  
    for( int i = 1 ; i <= n ; i++ )  
        r *= i;  
    return r;  
}
```

## Превод на Scheme

```
(define (for n r i)  
  (if (<= i n)  
      (for n (* r i) (+ i 1))  
      r))  
  
(define (fact n)  
  (for n 1 1))
```

# Оценка на итеративен факториел

(fact 4)

# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
```

# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(if (<= 1 4) (for 4 (* 1 1) (+ 1 1)) 1)
```

# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(for 4 1 2)
```

# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(for 4 1 2)
  ↓
(if (<= 2 4) (for 4 (* 1 2) (+ 2 1)) 2)
```

# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(for 4 1 2)
  ↓
(for 4 2 3)
```

# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(for 4 1 2)
  ↓
(for 4 2 3)
  ↓
(if (<= 3 4) (for 4 (* 2 3) (+ 3 1)) 6)
```

# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(for 4 1 2)
  ↓
(for 4 2 3)
  ↓
(for 4 6 4)
```

# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(for 4 1 2)
  ↓
(for 4 2 3)
  ↓
(for 4 6 4)
  ↓
(if (<= 4 4) (for 4 (* 6 4) (+ 4 1)) 24)
```

# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(for 4 1 2)
  ↓
(for 4 2 3)
  ↓
(for 4 6 4)
  ↓
(for 4 24 5)
```

# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(for 4 1 2)
  ↓
(for 4 2 3)
  ↓
(for 4 6 4)
  ↓
(for 4 24 5)
  ↓
(if (<= 5 4) (for 4 (* 24 5) (+ 5 1)) 24)
```

# Оценка на итеративен факториел

base  
 $O(n)$

```

(fact 4)
↓
(for 4 1 1)
↓
(for 4 1 2)
↓
(for 4 2 3)
↓
(for 4 6 4)
↓
(for 4 24 5)
↓
24
    
```

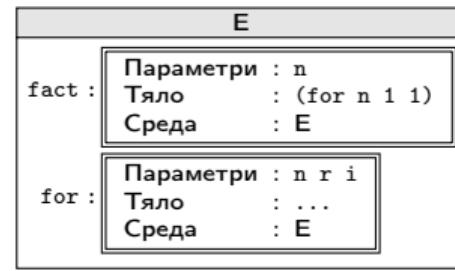
# Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(for 4 1 2)
  ↓
(for 4 2 3)
  ↓
(for 4 6 4)
  ↓
(for 4 24 5)
  ↓
24
```

## Линеен итеративен процес

# Оценка на итеративен факториел със среди

{E} (fact 4)



# Оценка на итеративен факториел със среди

{E}      (fact 4)  
           ↓  
 {E<sub>0</sub>}    (for n 1 1)



# Оценка на итеративен факториел със среди

{E}      (fact 4)  
           ↓  
 {E<sub>0</sub>}    (for 4 1 1)



# Оценка на итеративен факториел със среди

```

{E}      (fact 4)
         ↓
{E0}   (for 4 1 1)
         ↓
{E1}   (if (<= i n) (for n (* r i) (+ i 1)) r)
  
```



# Оценка на итеративен факториел със среди

```

{E}      (fact 4)
         ↓
{E₀}    (for 4 1 1)
         ↓
{E₁}    (for 4 1 2)
  
```



# Оценка на итеративен факториел със среди

```

{E}      (fact 4)
         ↓
{E₀}     (for 4 1 1)
         ↓
{E₁}     (for 4 1 2)
         ↓
{E₂}     (if (<= i n) (for n (* r i) (+ i 1)) r)
  
```



# Оценка на итеративен факториел със среди

```

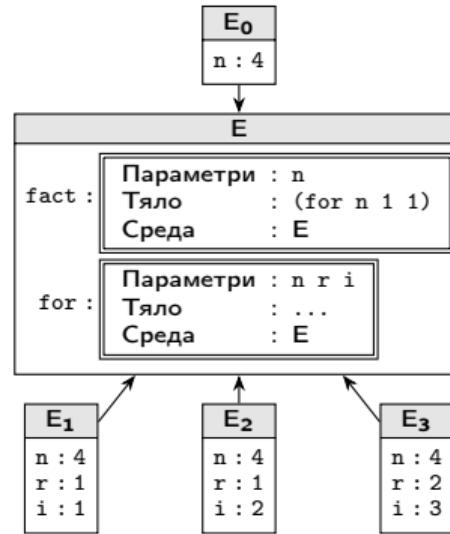
{E}      (fact 4)
         ↓
{E₀}    (for 4 1 1)
         ↓
{E₁}    (for 4 1 2)
         ↓
{E₂}    (for 4 2 3)
  
```



# Оценка на итеративен факториел със среди

```

{E}      (fact 4)
         ↓
{E₀}    (for 4 1 1)
         ↓
{E₁}    (for 4 1 2)
         ↓
{E₂}    (for 4 2 3)
         ↓
{E₃}    (if (<= i n) (for n (* r i) (+ i 1)) r)
  
```



# Оценка на итеративен факториел със среди

```

{E}      (fact 4)
         ↓
{E₀}    (for 4 1 1)
         ↓
{E₁}    (for 4 1 2)
         ↓
{E₂}    (for 4 2 3)
         ↓
{E₃}    (for 4 6 4)
  
```



# Оценка на итеративен факториел със среди

```

{E}      (fact 4)
         ↓
{E₀}    (for 4 1 1)
         ↓
{E₁}    (for 4 1 2)
         ↓
{E₂}    (for 4 2 3)
         ↓
{E₃}    (for 4 6 4)
         ↓
{E₄}    (if (<= i n) (for n (* r i) (+ i 1)) r)
  
```



# Оценка на итеративен факториел със среди

```

{E}      (fact 4)
         ↓
{E₀}    (for 4 1 1)
         ↓
{E₁}    (for 4 1 2)
         ↓
{E₂}    (for 4 2 3)
         ↓
{E₃}    (for 4 6 4)
         ↓
{E₄}    (for 4 24 5)
  
```



# Оценка на итеративен факториел със среди

```

{E}      (fact 4)
         ↓
{E₀}    (for 4 1 1)
         ↓
{E₁}    (for 4 1 2)
         ↓
{E₂}    (for 4 2 3)
         ↓
{E₃}    (for 4 6 4)
         ↓
{E₄}    (for 4 24 5)
         ↓
{E₅}    (if (<= i n) (for n (* r i) (+ i 1)) r)
  
```



# Оценка на итеративен факториел със среди

```

{E}      (fact 4)
         ↓
{E₀}    (for 4 1 1)
         ↓
{E₁}    (for 4 1 2)
         ↓
{E₂}    (for 4 2 3)
         ↓
{E₃}    (for 4 6 4)
         ↓
{E₄}    (for 4 24 5)
         ↓
{E₅}    24
  
```



# Рекурсивен и итеративен процес

```
(define (fact n)
  (if (= n 0) 1
      (* n (fact (- n 1)))))

(fact 4)
↓
(* 4 (fact 3))
↓
(* 4 (* 3 (fact 2)))
↓
(* 4 (* 3 (* 2 (fact 1))))
↓
(* 4 (* 3 (* 2 (* 1 (fact 0)))))
↓
(* 4 (* 3 (* 2 (* 1 1))))
↓
(* 4 (* 3 (* 2 1)))
↓
(* 4 (* 3 2))
↓
(* 4 6)
↓
24
```

```
(define (for n r i)
  (if (<= i n)
      (for n (* r i) (+ i 1))
      r))

(fact 4)
↓
(for 4 1 1)
↓
(for 4 1 2)
↓
(for 4 2 3)
↓
(for 4 6 4)
↓
(for 4 24 5)
↓
24
```

```
(define (fact n)
  (for n 1 1))
```

# Рекурсивен и итеративен процес

```
(define (fact n)
  (if (= n 0) 1
      (* n (fact (- n 1)))))
```

(fact 4)  
 ↓  
 (\* 4 (fact 3))  
 ↓  
 (\* 4 (\* 3 (fact 2)))  
 ↓  
 (\* 4 (\* 3 (\* 2 (fact 1))))  
 ↓  
 (\* 4 (\* 3 (\* 2 (\* 1 (fact 0))))))  
 ↓  
 (\* 4 (\* 3 (\* 2 (\* 1 1))))  
 ↓  
 (\* 4 (\* 3 (\* 2 1)))  
 ↓  
 (\* 4 (\* 3 2))  
 ↓  
 (\* 4 6)  
 ↓  
 24

```
(define (for n r i)
  (if (<= i n)
      (for n (* r i) (+ i 1))
      r))
```

(fact 4)  
 ↓  
 (for 4 1 1)  
 ↓  
 (for 4 1 2)  
 ↓  
 (for 4 2 3)  
 ↓  
 (for 4 6 4)  
 ↓  
 (for 4 24 5)  
 ↓  
 24

```
(define (fact n)
  (for n 1 1))
```

# Опашкова рекурсия

- Функциите, в които има отложени операции генерираят същински **рекурсивни процеси**

# Опашкова рекурсия

- Функциите, в които има отложени операции генерираят същински **рекурсивни процеси**
- Рекурсивно извикване, при което няма отложена операция се нарича **опашкова рекурсия**

# Опашкова рекурсия

- Функциите, в които има отложени операции генерират същински **рекурсивни процеси**
- Рекурсивно извикване, при което няма отложена операция се нарича **опашкова рекурсия**
- Функциите, в които всички рекурсивни извиквания са опашкови генерират **итеративни процеси**

# Опашкова рекурсия

- Функциите, в които има отложени операции генерират същински **рекурсивни процеси**
- Рекурсивно извикване, при което няма отложена операция се нарича **опашкова рекурсия**
- Функциите, в които всички рекурсивни извиквания са опашкови генерират **итеративни процеси**
- При итеративните процеси липсва етап на свиването на рекурсията

# Опашкова рекурсия

- Функциите, в които има отложени операции генерират същински **рекурсивни процеси**
- Рекурсивно извикване, при което няма отложена операция се нарича **опашкова рекурсия**
- Функциите, в които всички рекурсивни извиквания са опашкови генерират **итеративни процеси**
- При итеративните процеси липсва етап на свиването на рекурсията
- Опашковата рекурсия се използва за симулиране на цикли

# Опашкова рекурсия

- Функциите, в които има отложени операции генерираят същински **рекурсивни процеси**
- Рекурсивно извикване, при което няма отложена операция се нарича **опашкова рекурсия**
- Функциите, в които всички рекурсивни извиквания са опашкови генерираят **итеративни процеси**
- При итеративните процеси липсва етап на свиването на рекурсията
- Опашковата рекурсия се използва за симулиране на цикли
- В Scheme опашковата рекурсия **по стандарт** се интерпретира като цикъл

# Опашкова рекурсия

- Функциите, в които има отложени операции генерираят същински **рекурсивни процеси**
- Рекурсивно извикване, при което няма отложена операция се нарича **опашкова рекурсия**
- Функциите, в които всички рекурсивни извиквания са опашкови генерираят **итеративни процеси**
- При итеративните процеси липсва етап на свиването на рекурсията
- Опашковата рекурсия се използва за симулиране на цикли
- В Scheme опашковата рекурсия **по стандарт** се интерпретира като цикъл
  - т.е. не се заделя памет за всяко рекурсивно извикване

# Рекурсивен и итеративен процес

```
(define (fact n)
  (if (= n 0) 1
      (* n (fact (- n 1)))))

(fact 4)
↓
(* 4 (fact 3))
↓
(* 4 (* 3 (fact 2)))
↓
(* 4 (* 3 (* 2 (fact 1))))
↓
(* 4 (* 3 (* 2 (* 1 (fact 0)))))
↓
(* 4 (* 3 (* 2 (* 1 1))))
↓
(* 4 (* 3 (* 2 1)))
↓
(* 4 (* 3 2))
↓
(* 4 6)
↓
24
```

```
(define (for n r i)
  (if (<= i n)
      (for n (* r i) (+ i 1))
      r))

(fact 4)
↓
(for 4 1 1)
↓
(for 4 1 2)
↓
(for 4 2 3)
↓
(for 4 6 4)
↓
(for 4 24 5)
↓
24
```

```
(define (fact n)
  (for n 1 1))
```

# Оценка на итеративен факториел със среди

```

{E}      (fact 4)
         ↓
{E₀}    (for 4 1 1)
         ↓
{E₁}    (for 4 1 2)
         ↓
{E₂}    (for 4 2 3)
         ↓
{E₃}    (for 4 6 4)
         ↓
{E₄}    (for 4 24 5)
         ↓
{E₅}    24
  
```



## Вложени дефиниции

- (define (<функция> {<параметър>}) {<дефиниция>} <тяло>)

## Вложени дефиниции

- (define (<функция> {<параметър>}) {<дефиниция>}) <тяло>)
- При извикване на <функция> първо се оценяват всички <дефиниция> и след това се оценява <тяло>

## Вложени дефиниции

- (define (<функция> {<параметър>}) {<дефиниция>}) <тяло>)
- При извикване на <функция> първо се оценяват всички <дефиниция> и след това се оценява <тяло>
  - Първо се създава среда  $E_1$ , в която формалните параметри се свързват с оценките на фактическите

## Вложени дефиниции

- (define (<функция> {<параметър>}) {<дефиниция>}) <тяло>)
- При извикване на <функция> първо се оценяват всички <дефиниция> и след това се оценява <тяло>
  - Първо се създава среда  $E_1$ , в която формалните параметри се свързват с оценките на фактическите
  - След това се създава среда  $E_2$ , която разширява  $E_1$ , за вложените дефиниции

## Вложени дефиниции

- (define (<функция> {<параметър>}) {<дефиниция>}) <тяло>)
- При извикване на <функция> първо се оценяват всички <дефиниция> и след това се оценява <тяло>
  - Първо се създава среда  $E_1$ , в която формалните параметри се свързват с оценките на фактическите
  - След това се създава среда  $E_2$ , която разширява  $E_1$ , за вложените дефиниции
  - В средата  $E_2$  се записват всички символи от вложени дефиниции **без стойности**

## Вложени дефиниции

- (define (<функция> {<параметър>}) {<дефиниция>}) <тяло>)
- При извикване на <функция> първо се оценяват всички <дефиниция> и след това се оценява <тяло>
  - Първо се създава среда  $E_1$ , в която формалните параметри се свързват с оценките на фактическите
  - След това се създава среда  $E_2$ , която разширява  $E_1$ , за вложените дефиниции
  - В средата  $E_2$  се записват всички символи от вложени дефиниции **без стойности**
  - Всички вложени дефиниции се **оценяват** в  $E_2$

## Вложени дефиниции

- (define (<функция> {<параметър>}) {<дефиниция>}) <тяло>)
- При извикване на <функция> първо се оценяват всички <дефиниция> и след това се оценява <тяло>
  - Първо се създава среда  $E_1$ , в която формалните параметри се свързват с оценките на фактическите
  - След това се създава среда  $E_2$ , която разширява  $E_1$ , за вложените дефиниции
  - В средата  $E_2$  се записват всички символи от вложени дефиниции **без стойности**
  - Всички вложени дефиниции се **оценяват** в  $E_2$
  - Накрая получените оценки се **свързват** със съответните символи в  $E_2$

## Вложени дефиниции

- (define (<функция> {<параметър>}) {<дефиниция>}) <тяло>)
- При извикване на <функция> първо се оценяват всички <дефиниция> и след това се оценява <тяло>
  - Първо се създава среда  $E_1$ , в която формалните параметри се свързват с оценките на фактическите
  - След това се създава среда  $E_2$ , която разширява  $E_1$ , за вложените дефиниции
  - В средата  $E_2$  се записват всички символи от вложени дефиниции **без стойности**
  - Всички вложени дефиниции се **оценяват** в  $E_2$
  - Накрая получените оценки се **свързват** със съответните символи в  $E_2$
- Пример:

```
(define (dist x1 y1 x2 y2)
  (define dx (- x2 x1))
  (define dy (- y2 y1))
  (define (sq x) (* x x))
  (sqrt (+ (sq dx) (sq dy))))
```

# Оценка на вложени функции

{E} (dist 2 5 -1 9)

E	
dist :	Параметри : x1 y1 x2 y2
	Тяло : ...
	Среда : E

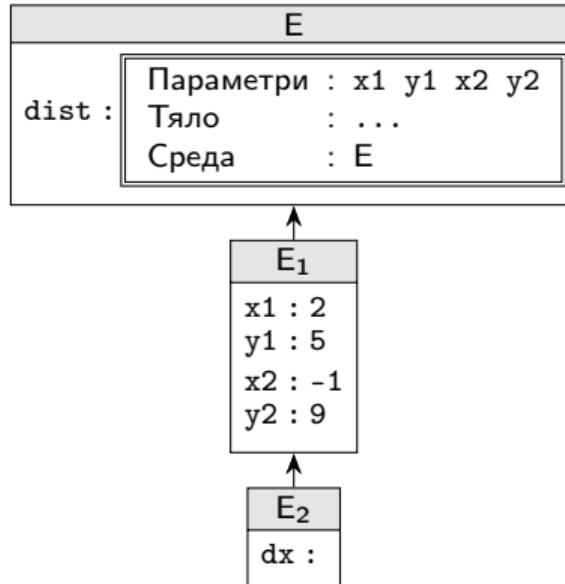
# Оценка на вложени функции

{E} (dist 2 5 -1 9)



# Оценка на вложени функции

```
{E}      (dist 2 5 -1 9)
         ↓
{E2}    (define dx (- x2 x1))
```



# Оценка на вложени функции

```
{E}      (dist 2 5 -1 9)
         ↓
{E2}   (define dx (- x2 x1))
{E2}   (define dy (- y2 y1))
```



# Оценка на вложени функции

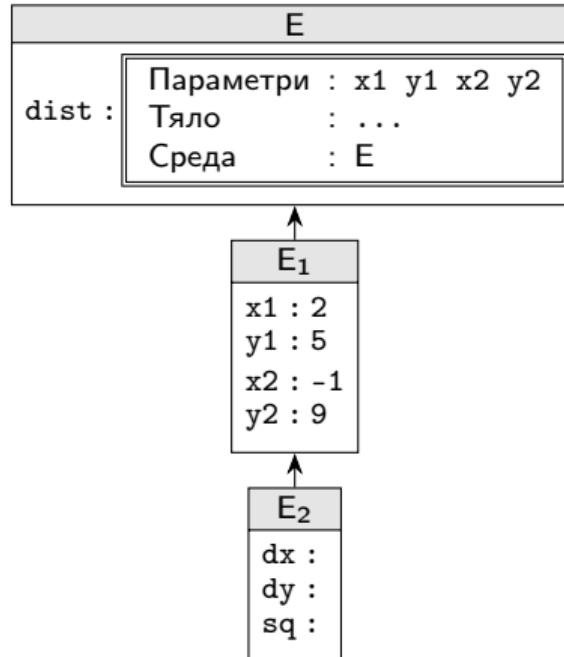
{E} (dist 2 5 -1 9)



{E<sub>2</sub>} (define dx (- x2 x1))

{E<sub>2</sub>} (define dy (- y2 y1))

{E<sub>2</sub>} (define (sq x) (\* x x))

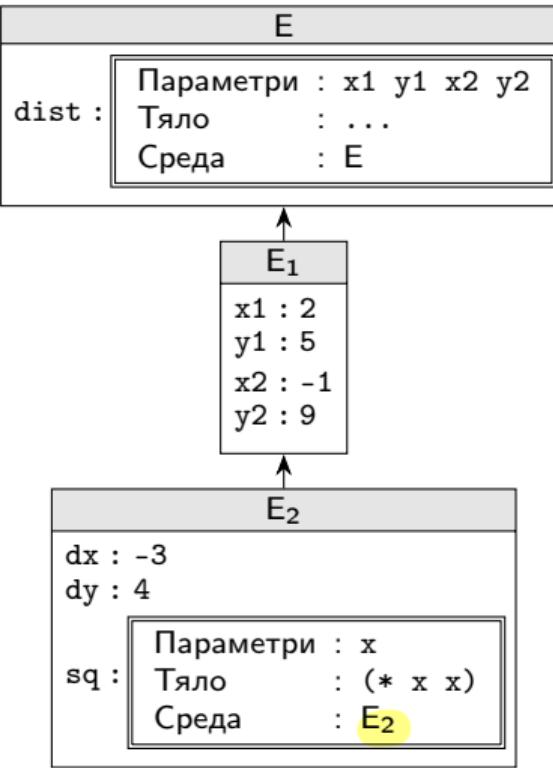


# Оценка на вложени функции

```

{E}      (dist 2 5 -1 9)
         ↓
{E2}   (define dx (- x2 x1))
{E2}   (define dy (- y2 y1))
{E2}   (define (sq x) (* x x))
{E2}   (sqrt (+ (sq dx) (sq dy)))

```



# Оценка на вложени функции

{E} (dist 2 5 -1 9)



{E<sub>2</sub>} (define dx (- x2 x1))

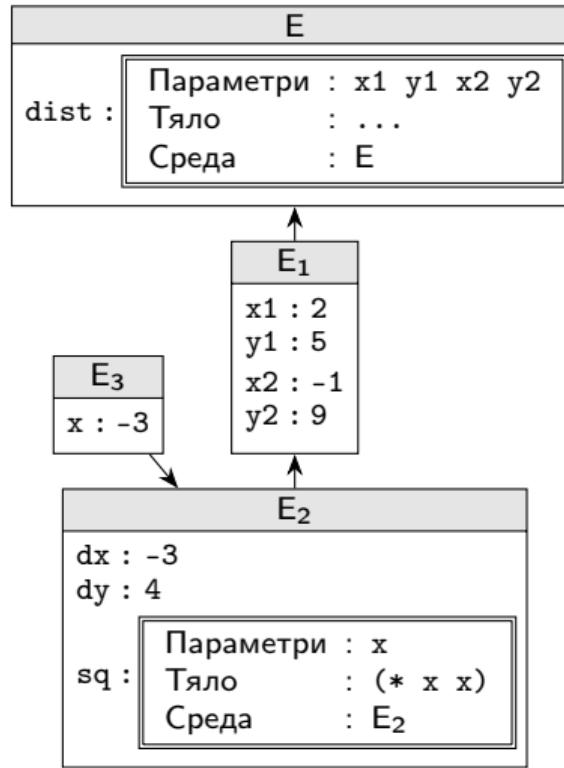
{E<sub>2</sub>} (define dy (- y2 y1))

{E<sub>2</sub>} (define (sq x) (\* x x))

{E<sub>2</sub>} (sqrt (+ (sq dx) (sq dy)))



{E<sub>3</sub>} (sqrt (+ (\* x x) (sq dy)))



# Оценка на вложени функции

{E} (dist 2 5 -1 9)



{E<sub>2</sub>} (define dx (- x2 x1))

{E<sub>2</sub>} (define dy (- y2 y1))

{E<sub>2</sub>} (define (sq x) (\* x x))

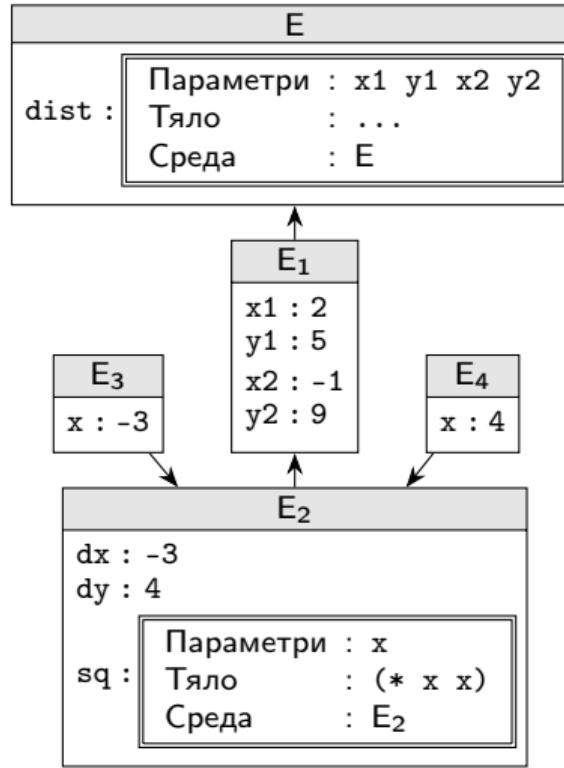
{E<sub>2</sub>} (sqrt (+ (sq dx) (sq dy)))



{E<sub>3</sub>} (sqrt (+ (\* x x) (sq dy)))



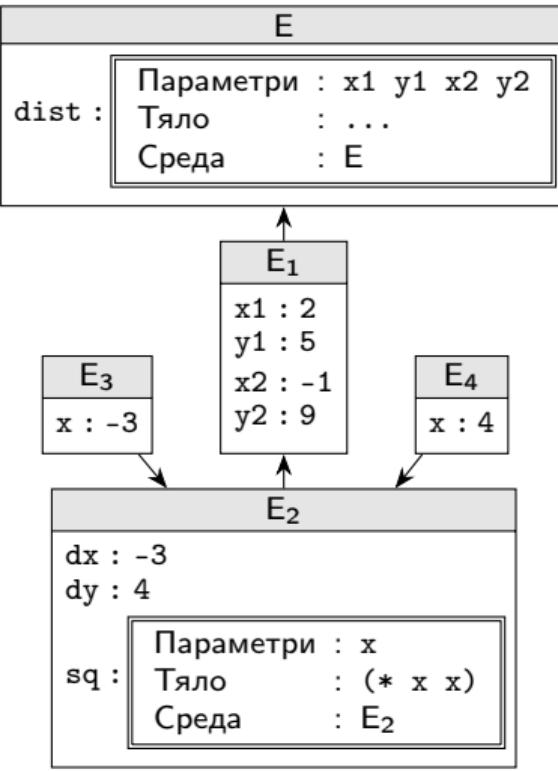
{E<sub>4</sub>} (sqrt (+ 9 (\* x x)))



# Оценка на вложени функции

```

{E}      (dist 2 5 -1 9)
         ↓
{E2}   (define dx (- x2 x1))
{E2}   (define dy (- y2 y1))
{E2}   (define (sq x) (* x x))
{E2}   (sqrt (+ (sq dx) (sq dy)))
         ↓
{E3}   (sqrt (+ (* x x) (sq dy)))
         ↓
{E4}   (sqrt (+ 9 (* x x)))
         ↓
{E2}   (sqrt (+ 9 16))
  
```

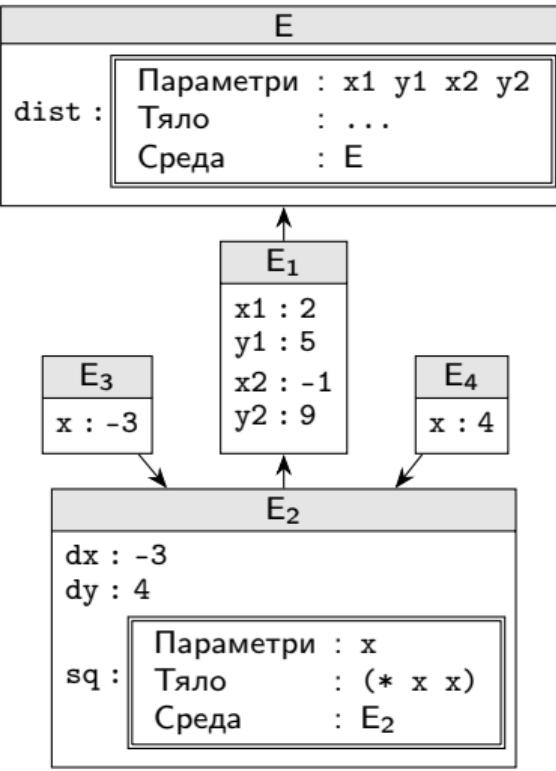


# Оценка на вложени функции

```

{E}      (dist 2 5 -1 9)
         ↓
{E2}   (define dx (- x2 x1))
{E2}   (define dy (- y2 y1))
{E2}   (define (sq x) (* x x))
{E2}   (sqrt (+ (sq dx) (sq dy)))
         ↓
{E3}   (sqrt (+ (* x x) (sq dy)))
         ↓
{E4}   (sqrt (+ 9 (* x x)))
         ↓
{E2}   (sqrt (+ 9 16))
         ↓
{E2}   (sqrt 25)

```



# Оценка на вложени функции

{E}      (dist 2 5 -1 9)



{E<sub>2</sub>}    (define dx (- x2 x1))

{E<sub>2</sub>}    (define dy (- y2 y1))

{E<sub>2</sub>}    (define (sq x) (\* x x))

{E<sub>2</sub>}    (sqrt (+ (sq dx) (sq dy)))



{E<sub>3</sub>}    (sqrt (+ (\* x x) (sq dy)))



{E<sub>4</sub>}    (sqrt (+ 9 (\* x x)))



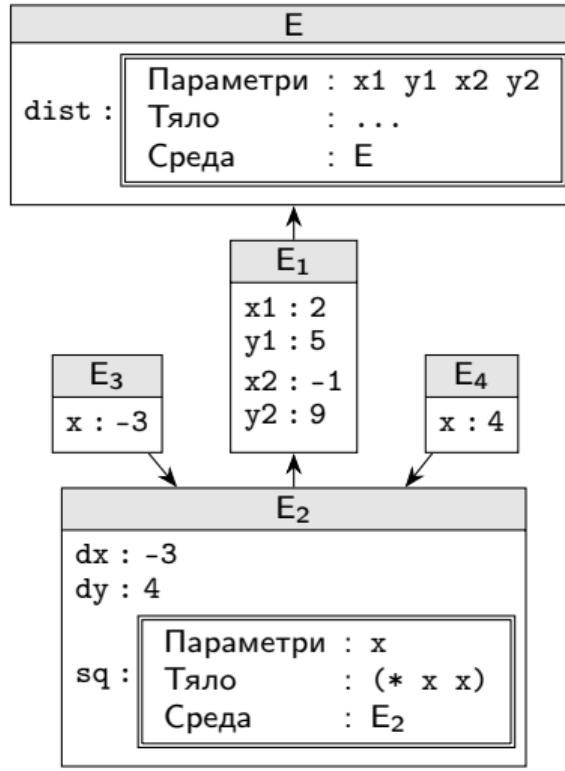
{E<sub>2</sub>}    (sqrt (+ 9 16))



{E<sub>2</sub>}    (sqrt 25)



{E<sub>2</sub>}    5



## Вложена помощна итеративна функция

При итеративни функция е удобно помощната функция да е вложена.

```
(define (for n r i)
  (if (<= i n)
      (for n (* r i) (+ i 1))
      r))
```

```
(define (fact n)
  (for n 1 1))
```

## Вложена помощна итеративна функция

При итеративни функции е удобно помощната функция да е вложена.

```
(define (fact n)
  (define (for r i)
    (if (<= i n)
        (for (* r i) (+ i 1))
        r))
  (for 1 1))
```

## Вложена помощна итеративна функция

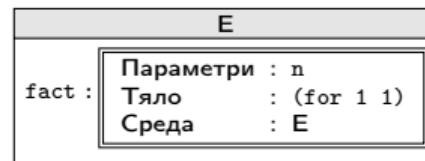
При итеративни функция е удобно помощната функция да е вложена.

```
(define (fact n)
  (define (for r i)
    (if (<= i n)
        (for (* r i) (+ i 1))
        r))
  (for 1 1))
```

Вложените дефиниции „виждат“ символите на обхващащите им дефиниции.

# Оценка на итеративен факториел с вложена функция

{E} (fact 4)

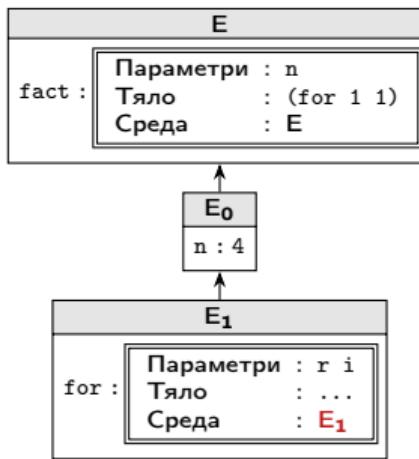


# Оценка на итеративен факториел с вложена функция



# Оценка на итеративен факториел с вложена функция

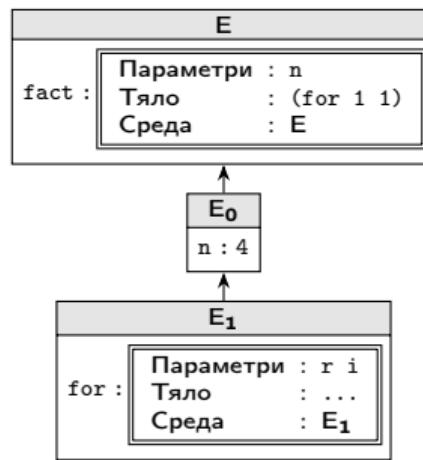
{E}      (fact 4)  
           ↓  
 {E<sub>1</sub>}    (define (for r i) ...)



# Оценка на итеративен факториел с вложена функция

```

{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
  
```



# Оценка на итеративен факториел с вложена функция

```

{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
         ↓
{E2}   (if (<= i n) (for (* r i) (+ i 1)) r)
  
```



# Оценка на итеративен факториел с вложена функция

```

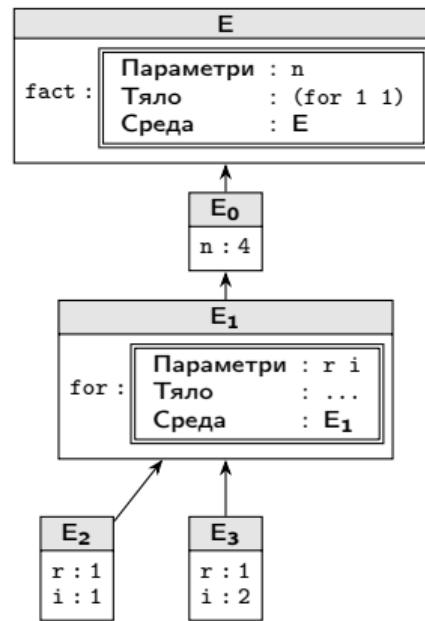
{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
         ↓
{E2}   (for 1 2)
  
```



# Оценка на итеративен факториел с вложена функция

```

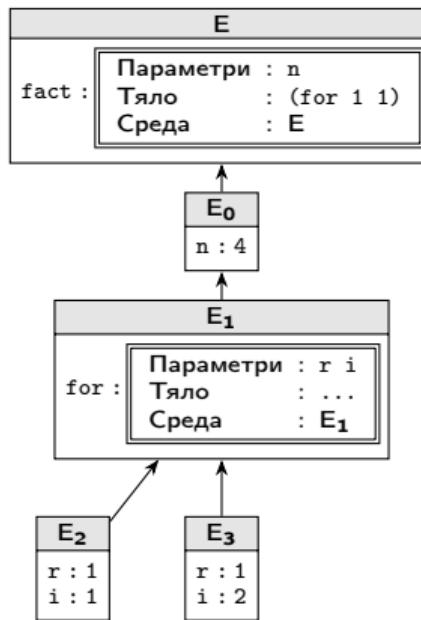
{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
         ↓
{E2}   (for 1 2)
         ↓
{E3}   (if (<= i n) (for (* r i) (+ i 1)) r)
  
```



# Оценка на итеративен факториел с вложена функция

```

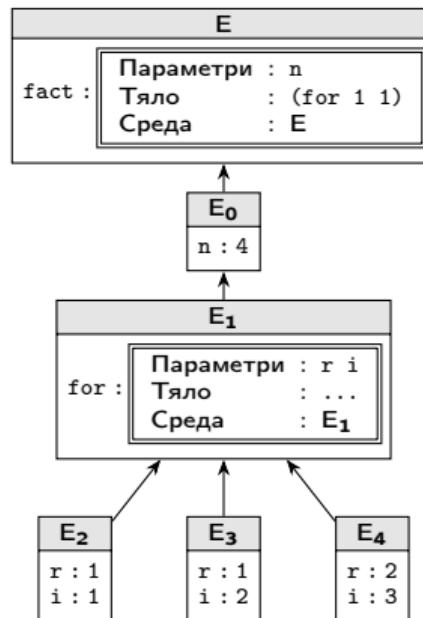
{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
         ↓
{E2}   (for 1 2)
         ↓
{E3}   (for 2 3)
  
```



# Оценка на итеративен факториел с вложена функция

```

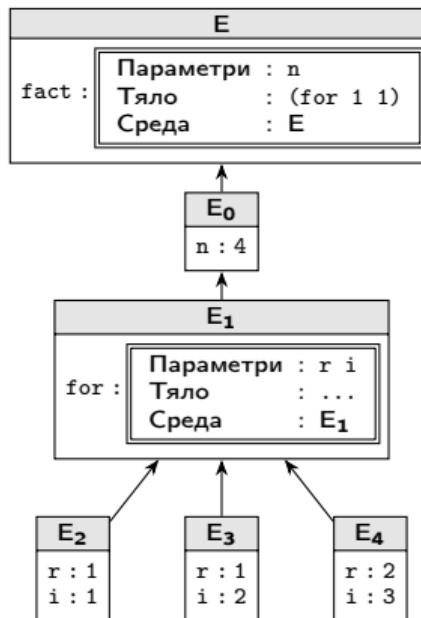
{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
         ↓
{E2}   (for 1 2)
         ↓
{E3}   (for 2 3)
         ↓
{E4}   (if (<= i n) (for (* r i) (+ i 1)) r)
  
```



# Оценка на итеративен факториел с вложена функция

```

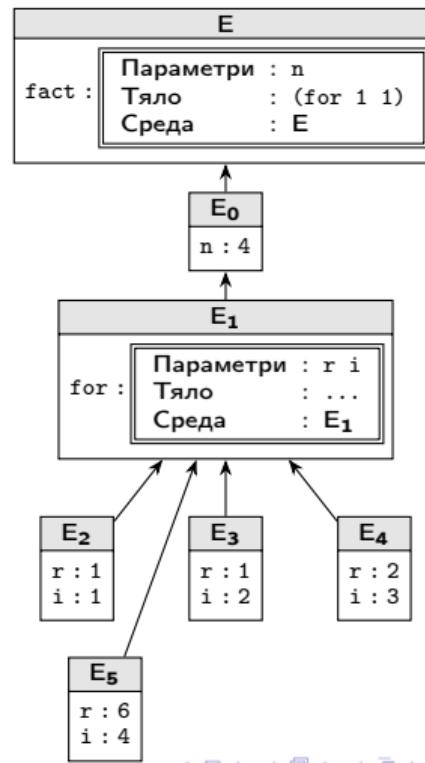
{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
         ↓
{E2}   (for 1 2)
         ↓
{E3}   (for 2 3)
         ↓
{E4}   (for 6 4)
  
```



# Оценка на итеративен факториел с вложена функция

```

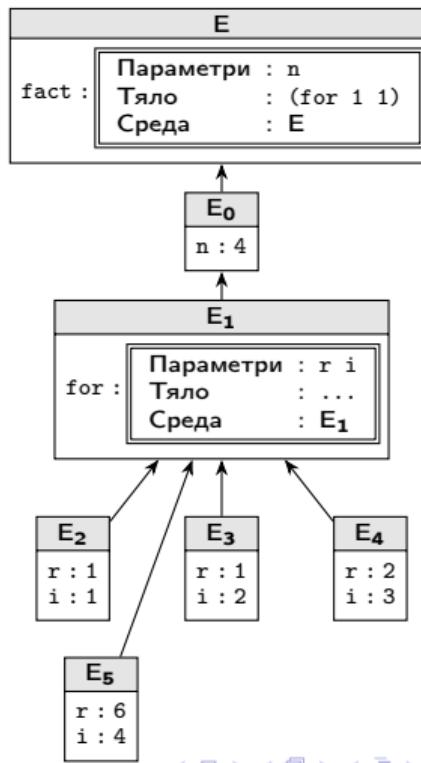
{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
         ↓
{E2}   (for 1 2)
         ↓
{E3}   (for 2 3)
         ↓
{E4}   (for 6 4)
         ↓
{E5}   (if (<= i n) (for (* r i) (+ i 1)) r)
  
```



# Оценка на итеративен факториел с вложена функция

```

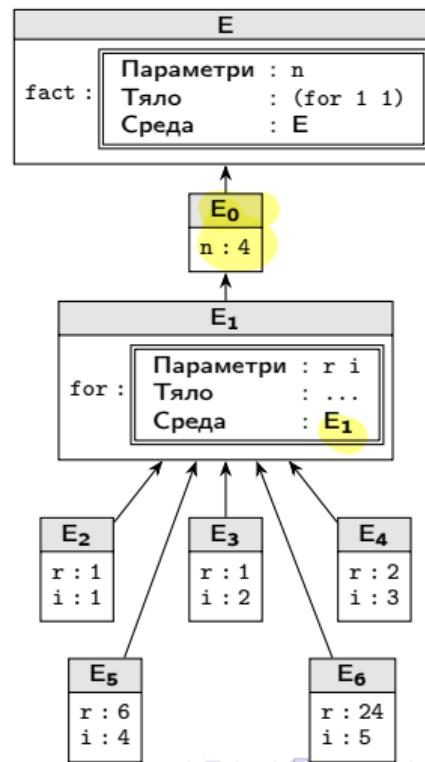
{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
         ↓
{E2}   (for 1 2)
         ↓
{E3}   (for 2 3)
         ↓
{E4}   (for 6 4)
         ↓
{E5}   (for 24 5)
  
```



# Оценка на итеративен факториел с вложена функция

```

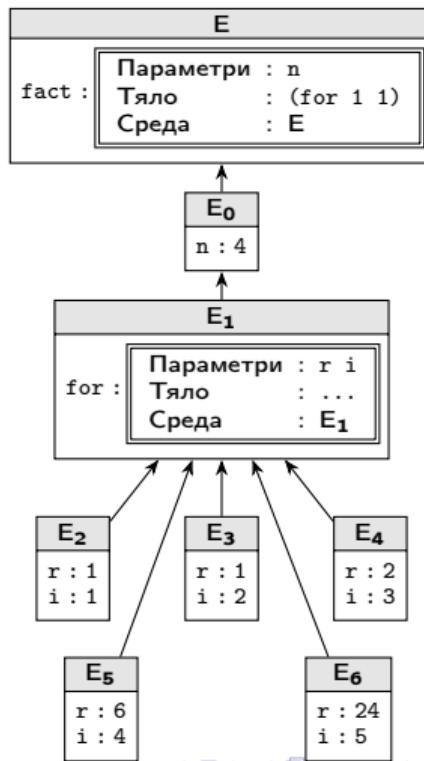
{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
         ↓
{E2}   (for 1 2)
         ↓
{E3}   (for 2 3)
         ↓
{E4}   (for 6 4)
         ↓
{E5}   (for 24 5)
         ↓
{E6}   (if (<= i n) (for (* r i) (+ i 1)) r)
  
```



# Оценка на итеративен факториел с вложена функция

```

{E}      (fact 4)
         ↓
{E1}   (define (for r i)...)
         ↓
{E1}   (for 1 1)
         ↓
{E2}   (for 1 2)
         ↓
{E3}   (for 2 3)
         ↓
{E4}   (for 6 4)
         ↓
{E5}   (for 24 5)
         ↓
{E6}   24
  
```



## Специална форма let

- (let ({(<символ> <израз>)}) <тяло>)

## Специална форма let

- (let ({(<символ> <израз>)}) <тяло>)
- (let ((<символ<sub>1</sub>> <израз<sub>1</sub>>)  
<символ<sub>2</sub>> <израз<sub>2</sub>>)  
...  
(<символ<sub>n</sub>> <израз<sub>n</sub>>))  
<тяло>)

## Специална форма let

- $(\text{let } (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let } ((\text{_1})$   
 $\quad (\text{_2})$   
 $\quad \dots$   
 $\quad (\text{_n}))$   
 $\quad \text{})$
- При оценка на let в среда E:

## Специална форма let

- $(\text{let } (\{(\langle \text{символ} \rangle \langle \text{израз} \rangle)\}) \langle \text{тяло} \rangle)$
- $(\text{let } ((\langle \text{символ}_1 \rangle \langle \text{израз}_1 \rangle)$   
 $\quad (\langle \text{символ}_2 \rangle \langle \text{израз}_2 \rangle)$   
 $\quad \dots$   
 $\quad (\langle \text{символ}_n \rangle \langle \text{израз}_n \rangle))$   
 $\quad \langle \text{тяло} \rangle)$
- При оценка на let в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E

## Специална форма let

- $(\text{let } (\{(\langle \text{символ} \rangle \langle \text{израз} \rangle)\}) \langle \text{тяло} \rangle)$
- $(\text{let } ((\langle \text{символ}_1 \rangle \langle \text{израз}_1 \rangle)$   
 $\quad (\langle \text{символ}_2 \rangle \langle \text{израз}_2 \rangle)$   
 $\quad \dots$   
 $\quad (\langle \text{символ}_n \rangle \langle \text{израз}_n \rangle))$   
 $\quad \langle \text{тяло} \rangle)$
- При оценка на let в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на  $\langle \text{израз}_1 \rangle$  в E се свързва със  $\langle \text{символ}_1 \rangle$  в  $E_1$

## Специална форма let

- $(\text{let } (\{(\langle \text{символ} \rangle \langle \text{израз} \rangle)\}) \langle \text{тяло} \rangle)$
- $(\text{let } ((\langle \text{символ}_1 \rangle \langle \text{израз}_1 \rangle)$   
 $\quad (\langle \text{символ}_2 \rangle \langle \text{израз}_2 \rangle)$   
 $\quad \dots$   
 $\quad (\langle \text{символ}_n \rangle \langle \text{израз}_n \rangle))$   
 $\quad \langle \text{тяло} \rangle)$
- При оценка на let в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на  $\langle \text{израз}_1 \rangle$  в E се свързва със  $\langle \text{символ}_1 \rangle$  в  $E_1$
  - Оценката на  $\langle \text{израз}_2 \rangle$  в E се свързва със  $\langle \text{символ}_2 \rangle$  в  $E_1$

## Специална форма let

- (let ({(<символ> <израз>)}) <тяло>)
- (let ((<символ<sub>1</sub>> <израз<sub>1</sub>>)  
<символ<sub>2</sub>> <израз<sub>2</sub>>)  
...  
(<символ<sub>n</sub>> <израз<sub>n</sub>>))  
<тяло>)
- При оценка на let в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на <израз<sub>1</sub>> в E се свързва със <символ<sub>1</sub>> в  $E_1$
  - Оценката на <израз<sub>2</sub>> в E се свързва със <символ<sub>2</sub>> в  $E_1$
  - ...

## Специална форма let

- $(\text{let } (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let } ((\text{_1})$   
 $\quad (\text{_2})$   
 $\quad \dots$   
 $\quad (\text{_n}))$   
 $\quad \text{})$
- При оценка на let в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на  $\text{_1}$  в E се свързва със  $\text{ в  $E_1$$
  - Оценката на  $\text{_2}$  в E се свързва със  $\text{ в  $E_1$$
  - $\dots$
  - Оценката на  $\text{_n}$  в E се свързва със  $\text{ в  $E_1$$

## Специална форма let

- (let ({(<символ> <израз>)}) <тяло>)
- (let ((<символ<sub>1</sub>> <израз<sub>1</sub>>)  
<символ<sub>2</sub>> <израз<sub>2</sub>>)  
...  
(<символ<sub>n</sub>> <израз<sub>n</sub>>))  
<тяло>)
- При оценка на let в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на <израз<sub>1</sub>> в E се свързва със <символ<sub>1</sub>> в  $E_1$
  - Оценката на <израз<sub>2</sub>> в E се свързва със <символ<sub>2</sub>> в  $E_1$
  - ...
  - Оценката на <израз<sub>n</sub>> в E се свързва със <символ<sub>n</sub>> в  $E_1$
  - Връща се оценката на <тяло> в средата  $E_1$

## Специална форма let

- (let ({(<символ> <израз>)}) <тяло>)
- (let ((<символ<sub>1</sub>> <израз<sub>1</sub>>)  
<символ<sub>2</sub>> <израз<sub>2</sub>>)  
...  
(<символ<sub>n</sub>> <израз<sub>n</sub>>))  
<тяло>)
- При оценка на let в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на <израз<sub>1</sub>> в E се свързва със <символ<sub>1</sub>> в  $E_1$
  - Оценката на <израз<sub>2</sub>> в E се свързва със <символ<sub>2</sub>> в  $E_1$
  - ...
  - Оценката на <израз<sub>n</sub>> в E се свързва със <символ<sub>n</sub>> в  $E_1$
  - Връща се оценката на <тяло> в средата  $E_1$
- **let няма странични ефекти върху средата!**

## Пример за let

```
(define (dist x1 y1 x2 y2)
  (let ((dx (- x2 x1))
        (dy (- y2 y1)))
    (sqrt (+ (sq dx) (sq dy)))))
```

## Пример за let

```
(define (dist x1 y1 x2 y2)
  (let ((dx (- x2 x1))
        (dy (- y2 y1)))
    (sqrt (+ (sq dx) (sq dy)))))

(define (area x1 y1 x2 y2 x3 y3)
  (let ((a (dist x1 y1 x2 y2))
        (b (dist x2 y2 x3 y3))
        (c (dist x3 y3 x1 y1)))
    (p (/ (+ a b c) 2)))
  (sqrt (* p (- p a) (- p b) (- p c)))))
```

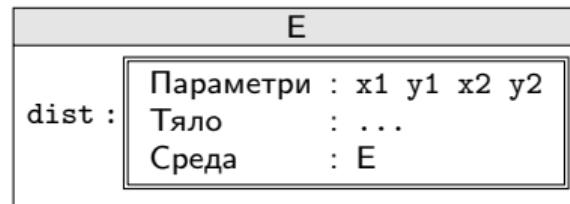
## Пример за let

```
(define (dist x1 y1 x2 y2)
  (let ((dx (- x2 x1))
        (dy (- y2 y1)))
    (sqrt (+ (sq dx) (sq dy)))))
```

```
(define (area x1 y1 x2 y2 x3 y3)
  (let ((a (dist x1 y1 x2 y2))
        (b (dist x2 y2 x3 y3))
        (c (dist x3 y3 x1 y1))
        (p (/ (+ a b c) 2)))
    (sqrt (* p (- p a) (- p b) (- p c))))))
```

# Оценка на let

{E} (dist 2 5 -1 9)



# Оценка на let

```
{E}      (dist 2 5 -1 9)
        ↓
(let ((dx (- x2 x1))
      (dy (- y2 y1)))
  (sqrt (+ (sq dx) (sq dy))))
```

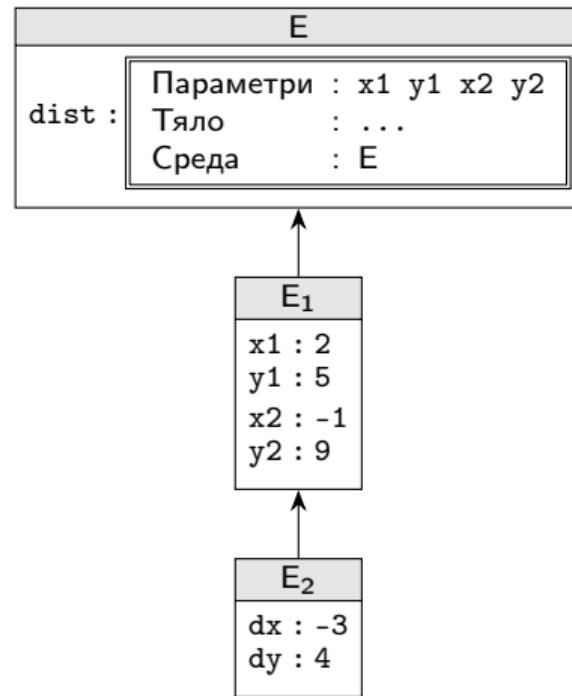
{E<sub>1</sub>}



# Оценка на let

```

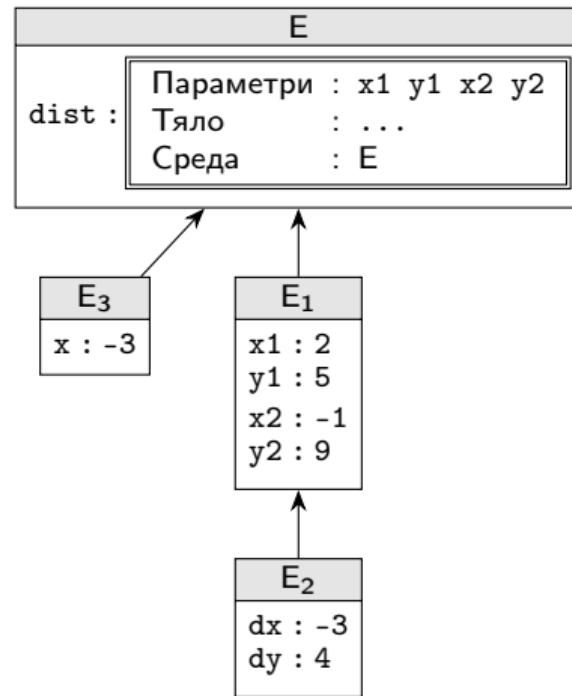
{E}      (dist 2 5 -1 9)
          ↓
(let ((dx (- x2 x1))
      (dy (- y2 y1)))
  (sqrt (+ (sq dx) (sq dy))))
          ↓
{E2}   (sqrt (+ (sq dx) (sq dy)))
  
```



# Оценка на let

```

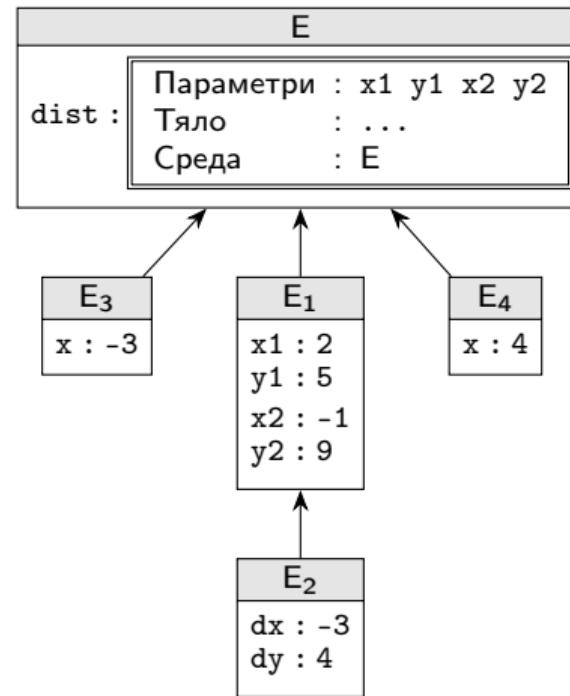
{E}      (dist 2 5 -1 9)
          ↓
(let ((dx (- x2 x1))
      (dy (- y2 y1)))
  (sqrt (+ (sq dx) (sq dy))))
          ↓
{E2}      (sqrt (+ (sq dx) (sq dy)))
          ↓
{E3}      (sqrt (+ (* x x) (sq dy)))
  
```



# Оценка на let

```

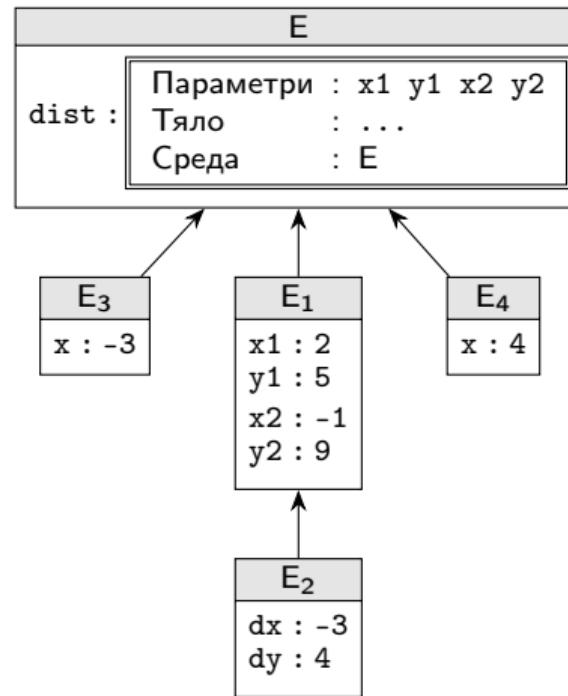
{E}      (dist 2 5 -1 9)
          ↓
(let ((dx (- x2 x1))
      (dy (- y2 y1)))
  (sqrt (+ (sq dx) (sq dy))))
          ↓
{E2}      (sqrt (+ (sq dx) (sq dy)))
          ↓
{E3}      (sqrt (+ (* x x) (sq dy)))
          ↓
{E4}      (sqrt (+ 9 (* x x)))
  
```



# Оценка на let

```

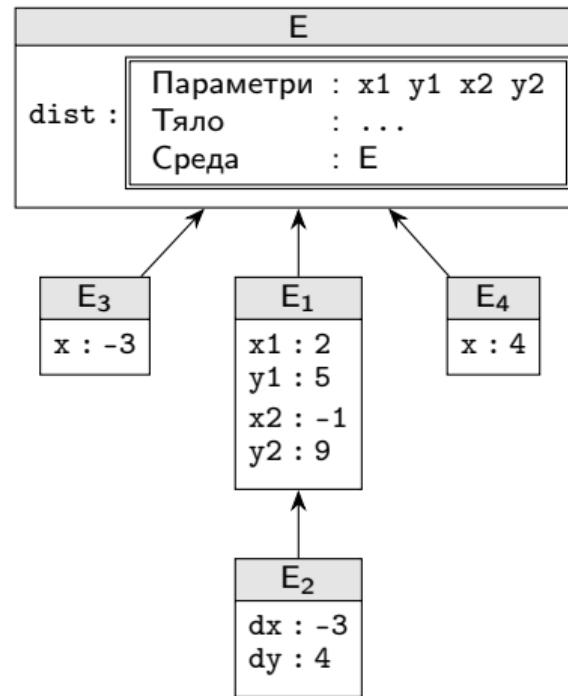
{E}      (dist 2 5 -1 9)
          ↓
{E1}    (let ((dx (- x2 x1))
           (dy (- y2 y1)))
        (sqrt (+ (sq dx) (sq dy))))
          ↓
{E2}    (sqrt (+ (sq dx) (sq dy)))
          ↓
{E3}    (sqrt (+ (* x x) (sq dy)))
          ↓
{E4}    (sqrt (+ 9 (* x x)))
          ↓
{E2}    (sqrt (+ 9 16))
  
```



# Оценка на let

```

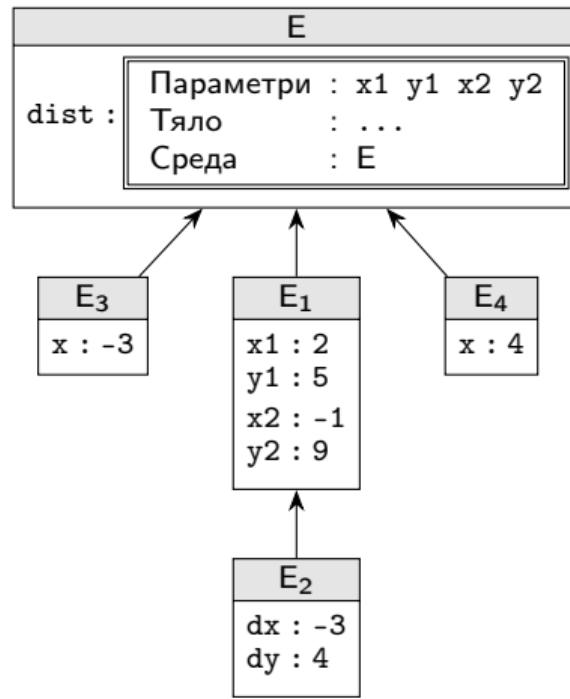
{E}      (dist 2 5 -1 9)
         ↓
{E1}    (let ((dx (- x2 x1))
           (dy (- y2 y1)))
        (sqrt (+ (sq dx) (sq dy))))
         ↓
{E2}    (sqrt (+ (sq dx) (sq dy)))
         ↓
{E3}    (sqrt (+ (* x x) (sq dy)))
         ↓
{E4}    (sqrt (+ 9 (* x x)))
         ↓
{E2}    (sqrt (+ 9 16))
         ↓
{E2}    (sqrt 25)
  
```



# Оценка на let

```

{E}      (dist 2 5 -1 9)
         ↓
{E1}    (let ((dx (- x2 x1))
           (dy (- y2 y1)))
        (sqrt (+ (sq dx) (sq dy))))
         ↓
{E2}    (sqrt (+ (sq dx) (sq dy)))
         ↓
{E3}    (sqrt (+ (* x x) (sq dy)))
         ↓
{E4}    (sqrt (+ 9 (* x x)))
         ↓
{E2}    (sqrt (+ 9 16))
         ↓
{E2}    (sqrt 25)
         ↓
{E2}    5
  
```



## Специална форма let\*

- (let\* ({(<символ> <израз>)}) <тяло>)

## Специална форма let\*

- $(\text{let*} (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let*} ((\text{}$

## Специална форма let\*

- $(\text{let*} (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let*} ((\text{_1})$   
 $\quad (\text{_2})$   
 $\quad \dots$   
 $\quad (\text{_n}))$   
 $\text{ })$
- При оценка на let\* в среда E:

## Специална форма let\*

- $(\text{let*} (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let*} ((\text{}$
- При оценка на let\* в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E

## Специална форма let\*

- $(\text{let*} (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let*} ((\text{_1})$   
 $\quad (\text{_2})$   
 $\quad \dots$   
 $\quad (\text{_n}))$   
 $\text{ })$
- При оценка на let\* в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на  $\text{_1}$  в E се свързва със  $\text{ в  $E_1$$

## Специална форма let\*

- $(\text{let*} (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let*} ((\text{_1})$   
 $\quad (\text{_2})$   
 $\quad \dots$   
 $\quad (\text{_n}))$   
 $\text{ })$
- При оценка на let\* в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на  $\text{_1}$  в E се свързва със  $\text{ в  $E_1$$
  - Създава се нова среда  $E_2$  разширение на текущата среда  $E_1$

## Специална форма let\*

- $(\text{let*} (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let*} ((\text{_1})$   
 $\quad (\text{_2})$   
 $\quad \dots$   
 $\quad (\text{_n}))$   
 $\text{ })$
- При оценка на let\* в среда  $E$ :
  - Създава се нова среда  $E_1$  разширение на текущата среда  $E$
  - Оценката на  $\text{_1}$  в  $E$  се свързва със  $\text{ в  $E_1$$
  - Създава се нова среда  $E_2$  разширение на текущата среда  $E_1$
  - Оценката на  $\text{_2}$  в  $E_1$  се свързва със  $\text{ в  $E_2$$

## Специална форма let\*

- $(\text{let*} (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let*} ((\text{_1})$   
 $\quad (\text{_2})$   
 $\quad \dots$   
 $\quad (\text{_n}))$   
 $\text{ })$
- При оценка на let\* в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на  $\text{_1}$  в E се свързва със  $\text{ в  $E_1$$
  - Създава се нова среда  $E_2$  разширение на текущата среда  $E_1$
  - Оценката на  $\text{_2}$  в  $E_1$  се свързва със  $\text{ в  $E_2$$
  - ...

## Специална форма let\*

- $(\text{let*} (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let*} ((\text{)$   
 $\quad (\text{  
 $\quad \dots$   
 $\quad (\text{  
 $\text{ })$$$
- При оценка на let\* в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на  $\text{ в E се свързва със  $\text{ в  $E_1$$$
  - Създава се нова среда  $E_2$  разширение на текущата среда  $E_1$
  - Оценката на  $\text{ в  $E_1$  се свързва със  $\text{ в  $E_2$$$
  - ...
  - Създава се нова среда  $E_n$  разширение на текущата среда  $E_{n-1}$

## Специална форма let\*

- $(\text{let*} (\{(\langle \text{символ} \rangle \langle \text{израз} \rangle)\}) \langle \text{тяло} \rangle)$
- $(\text{let*} ((\langle \text{символ}_1 \rangle \langle \text{израз}_1 \rangle) (\langle \text{символ}_2 \rangle \langle \text{израз}_2 \rangle) \dots (\langle \text{символ}_n \rangle \langle \text{израз}_n \rangle)) \langle \text{тяло} \rangle)$
- При оценка на let\* в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на  $\langle \text{израз}_1 \rangle$  в E се свързва със  $\langle \text{символ}_1 \rangle$  в  $E_1$
  - Създава се нова среда  $E_2$  разширение на текущата среда  $E_1$
  - Оценката на  $\langle \text{израз}_2 \rangle$  в  $E_1$  се свързва със  $\langle \text{символ}_2 \rangle$  в  $E_2$
  - ...
  - Създава се нова среда  $E_n$  разширение на текущата среда  $E_{n-1}$
  - Оценката на  $\langle \text{израз}_n \rangle$  в  $E_{n-1}$  се свързва със  $\langle \text{символ}_n \rangle$  в  $E_n$

## Специална форма let\*

- $(\text{let*} (\{(\text{} \text{ })\}) \text{ })$
- $(\text{let*} ((\text{_1})$   
 $\quad (\text{_2})$   
 $\quad \dots$   
 $\quad (\text{_n}))$   
 $\text{ })$
- При оценка на let\* в среда E:
  - Създава се нова среда  $E_1$  разширение на текущата среда E
  - Оценката на  $\text{_1}$  в E се свързва със  $\text{ в  $E_1$$
  - Създава се нова среда  $E_2$  разширение на текущата среда  $E_1$
  - Оценката на  $\text{_2}$  в  $E_1$  се свързва със  $\text{ в  $E_2$$
  - ...
  - Създава се нова среда  $E_n$  разширение на текущата среда  $E_{n-1}$
  - Оценката на  $\text{_n}$  в  $E_{n-1}$  се свързва със  $\text{_n}$  в  $E_n$
  - Връща се оценката на  $\text{  }$  в средата  $E_n$

## Пример за let\*

```
(define (area x1 y1 x2 y2 x3 y3)
  (let* ((a (dist x1 y1 x2 y2))
         (b (dist x2 y2 x3 y3))
         (c (dist x3 y3 x1 y1))
         (p (/ (+ a b c) 2)))
```

(let ((a 3))  
 (let ((b 4))  
 (let ((c 5)) (+ a b c))))  
  
(let\* ((a 3) (b 4) (c 5)) (+ a b c))

## Пример за let\*

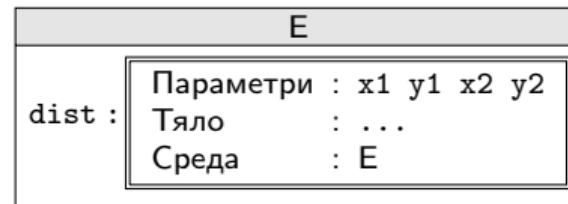
```
(define (area x1 y1 x2 y2 x3 y3)
  (let* ((a (dist x1 y1 x2 y2))
         (b (dist x2 y2 x3 y3))
         (c (dist x3 y3 x1 y1)))
    (p (/ (+ a b c) 2)))
```

Редът има значение!

```
(define (area x1 y1 x2 y2 x3 y3)
  (let* ((p (/ (+ a b c) 2))
         (a (dist x1 y1 x2 y2))
         (b (dist x2 y2 x3 y3))
         (c (dist x3 y3 x1 y1)))
    (sqrt (* p (- p a) (- p b) (- p c)))))
```

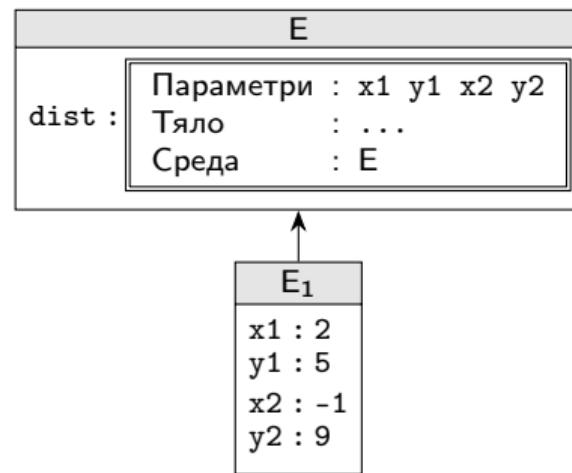
# Оценка на let\*

{E} (dist 2 5 -1 9)



# Оценка на let\*

```
{E}      (dist 2 5 -1 9)
        ↓
(let* ((dx (- x2 x1))
       (dy (- y2 y1)))
{E1}      (sqrt (+ (sq dx) (sq dy))))
```

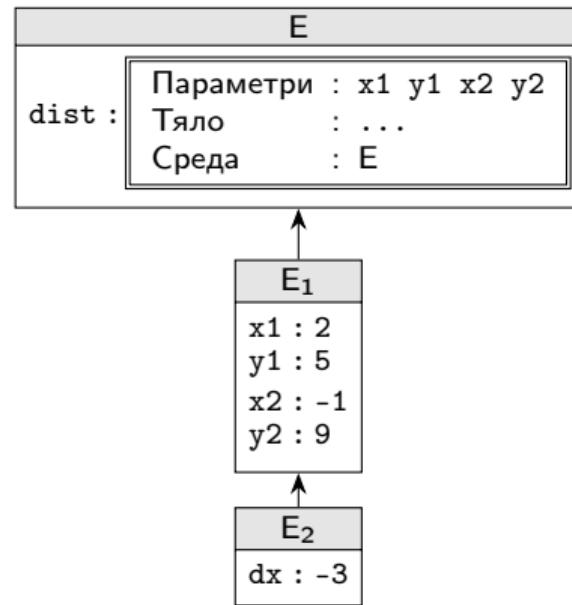


# Оценка на let\*

```

{E}      (dist 2 5 -1 9)
         ↓
(let* ((dx (- x2 x1))
       (dy (- y2 y1)))
{E1}      (sqrt (+ (sq dx) (sq dy))))

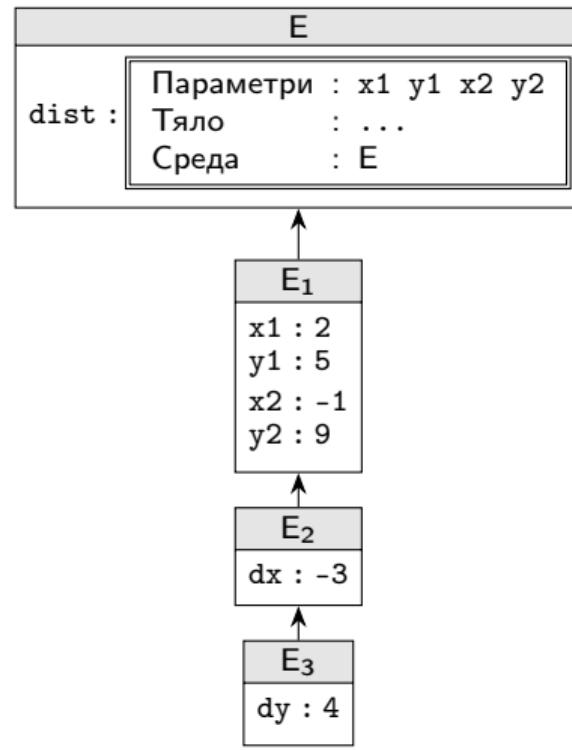
```



# Оценка на let\*

```

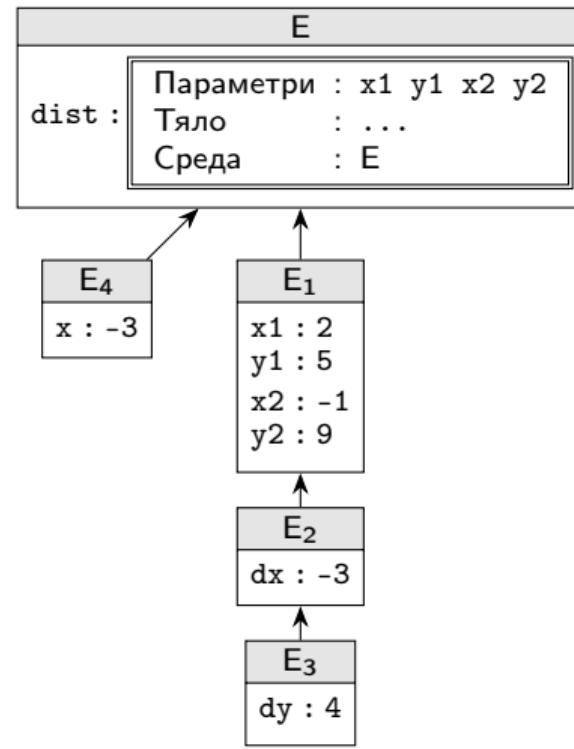
{E}      (dist 2 5 -1 9)
         ↓
(let* ((dx (- x2 x1))
       (dy (- y2 y1)))
  (sqrt (+ (sq dx) (sq dy))))
         ↓
{E3}   (sqrt (+ (sq dx) (sq dy)))
  
```



# Оценка на let\*

```

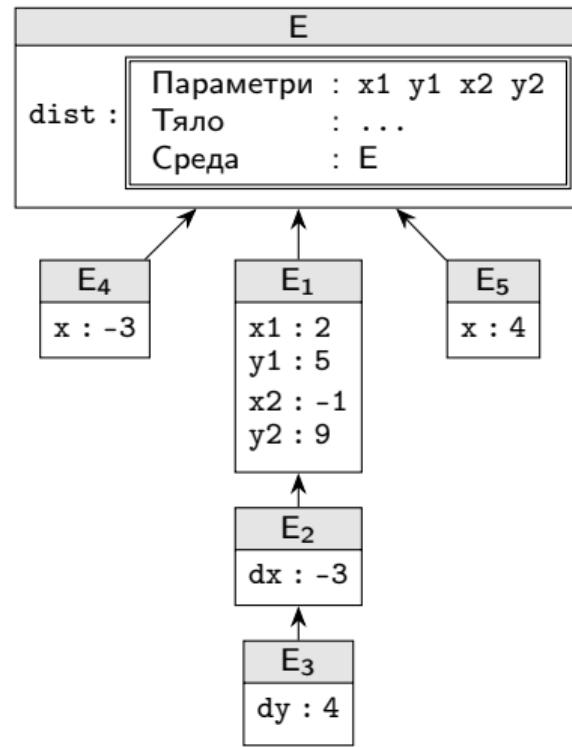
{E}      (dist 2 5 -1 9)
          ↓
{E1}  (let* ((dx (- x2 x1))
              (dy (- y2 y1)))
          (sqrt (+ (sq dx) (sq dy))))
          ↓
{E3}  (sqrt (+ (sq dx) (sq dy)))
          ↓
{E4}  (sqrt (+ (* x x) (sq dy)))
    
```



# Оценка на let\*

```

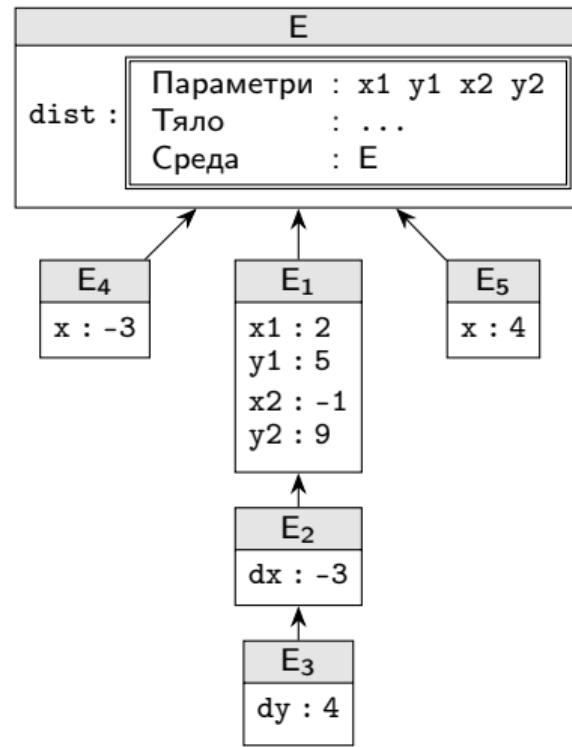
{E}      (dist 2 5 -1 9)
         ↓
{E1}  (let* ((dx (- x2 x1))
              (dy (- y2 y1)))
          (sqrt (+ (sq dx) (sq dy))))
         ↓
{E3}  (sqrt (+ (sq dx) (sq dy)))
         ↓
{E4}  (sqrt (+ (* x x) (sq dy)))
         ↓
{E5}  (sqrt (+ 9 (* x x)))
  
```



# Оценка на let\*

```

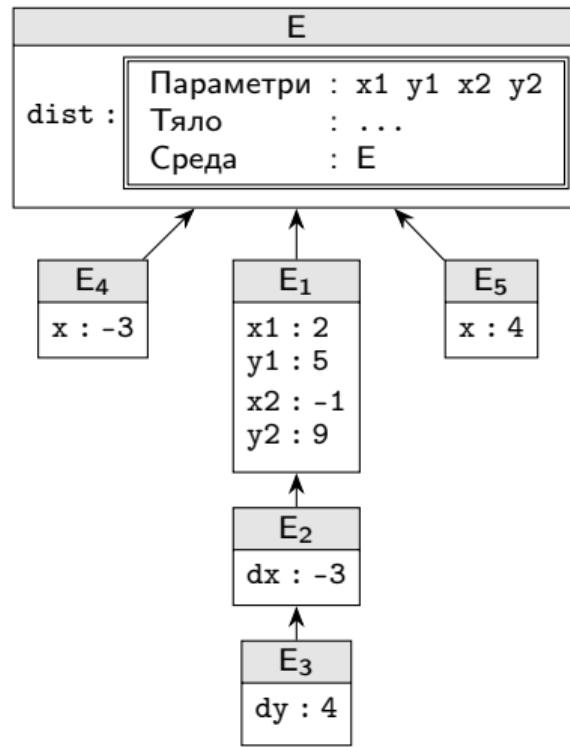
{E}      (dist 2 5 -1 9)
         ↓
{E1}   (let* ((dx (- x2 x1))
                  (dy (- y2 y1)))
            (sqrt (+ (sq dx) (sq dy))))
         ↓
{E3}   (sqrt (+ (sq dx) (sq dy)))
         ↓
{E4}   (sqrt (+ (* x x) (sq dy)))
         ↓
{E5}   (sqrt (+ 9 (* x x)))
         ↓
{E3}   (sqrt (+ 9 16))
  
```



# Оценка на let\*

```

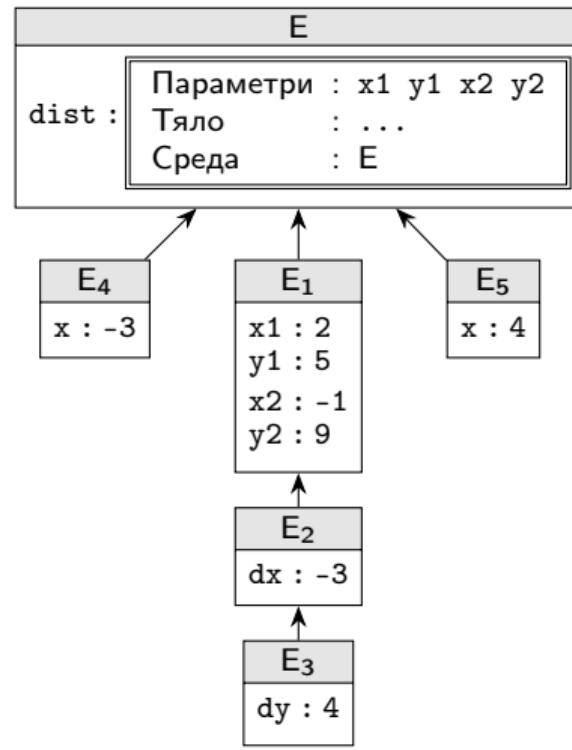
{E}      (dist 2 5 -1 9)
         ↓
{E1}    (let* ((dx (- x2 x1))
             (dy (- y2 y1)))
        (sqrt (+ (sq dx) (sq dy))))
         ↓
{E3}    (sqrt (+ (sq dx) (sq dy)))
         ↓
{E4}    (sqrt (+ (* x x) (sq dy)))
         ↓
{E5}    (sqrt (+ 9 (* x x)))
         ↓
{E3}    (sqrt (+ 9 16))
         ↓
{E3}    (sqrt 25)
  
```



# Оценка на let\*

```

{E}      (dist 2 5 -1 9)
         ↓
{E1}   (let* ((dx (- x2 x1))
                  (dy (- y2 y1)))
            (sqrt (+ (sq dx) (sq dy))))
         ↓
{E3}   (sqrt (+ (sq dx) (sq dy)))
         ↓
{E4}   (sqrt (+ (* x x) (sq dy)))
         ↓
{E5}   (sqrt (+ 9 (* x x)))
         ↓
{E3}   (sqrt (+ 9 16))
         ↓
{E3}   (sqrt 25)
         ↓
{E3}   5
  
```



# Степенуване

Функцията  $x^n$  може да се дефинира по следния начин:

$$x^n = \begin{cases} 1, & \text{ако } n = 0, \\ \frac{1}{x^{-n}}, & \text{ако } n < 0, \\ x \cdot x^{n-1}, & \text{ако } n > 0. \end{cases}$$

# Степенуване

Функцията  $x^n$  може да се дефинира по следния начин:

$$x^n = \begin{cases} 1, & \text{ако } n = 0, \\ \frac{1}{x^{-n}}, & \text{ако } n < 0, \\ x \cdot x^{n-1}, & \text{ако } n > 0. \end{cases}$$

```
(define (pow x n)
  (cond ((= n 0) 1)
        ((< n 0) (/ 1 (pow x (- n)))))
        (else (* x (pow x (- n 1))))))
```

# Оценка на степенуване

(row 2 6)

# Оценка на степенуване

$$\begin{array}{c} (\text{pow } 2 \ 6) \\ \downarrow \\ (* \ 2 \ (\text{pow } 2 \ 5)) \end{array}$$

# Оценка на степенуване

```
(pow 2 6)
  ↓
(* 2 (pow 2 5))
  ↓
(* 2 (* 2 (pow 2 4)))
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
(* 2 (pow 2 5))
  ↓
(* 2 (* 2 (pow 2 4)))
  ↓
(* 2 (* 2 (* 2 (pow 2 3)))))
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
(* 2 (pow 2 5))
  ↓
(* 2 (* 2 (pow 2 4)))
  ↓
(* 2 (* 2 (* 2 (pow 2 3)))))
  ↓
(* 2 (* 2 (* 2 (* 2 (pow 2 2))))))
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
(* 2 (pow 2 5))
  ↓
(* 2 (* 2 (pow 2 4)))
  ↓
(* 2 (* 2 (* 2 (pow 2 3))))
  ↓
(* 2 (* 2 (* 2 (* 2 (pow 2 2)))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 1))))))
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
(* 2 (pow 2 5))
  ↓
(* 2 (* 2 (pow 2 4)))
  ↓
(* 2 (* 2 (* 2 (pow 2 3)))))
  ↓
(* 2 (* 2 (* 2 (* 2 (pow 2 2))))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 1)))))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 0)))))))
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
(* 2 (pow 2 5))
  ↓
(* 2 (* 2 (pow 2 4)))
  ↓
(* 2 (* 2 (* 2 (pow 2 3))))
  ↓
(* 2 (* 2 (* 2 (* 2 (pow 2 2)))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 1))))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 0)))))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 1)))))))
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
(* 2 (pow 2 5))
  ↓
(* 2 (* 2 (pow 2 4)))
  ↓
(* 2 (* 2 (* 2 (pow 2 3))))
  ↓
(* 2 (* 2 (* 2 (* 2 (pow 2 2)))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 1))))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 0)))))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 1)))))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (* 2 2))))))
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
(* 2 (pow 2 5))
  ↓
(* 2 (* 2 (pow 2 4)))
  ↓
(* 2 (* 2 (* 2 (pow 2 3))))
  ↓
(* 2 (* 2 (* 2 (* 2 (pow 2 2)))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 1))))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 0)))))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 1)))))))
  ↓
(* 2 (* 2 (* 2 (* 2 (* 2 (* 2 2))))))
  ↓
(* 2 (* 2 (* 2 (* 2 4))))
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
  (* 2 (pow 2 5))
  ↓
  (* 2 (* 2 (pow 2 4)))
  ↓
  (* 2 (* 2 (* 2 (pow 2 3))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (pow 2 2)))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 1))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 0)))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 1)))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 2)))))
  ↓
  (* 2 (* 2 (* 2 (* 2 4))))
  ↓
  (* 2 (* 2 (* 2 8)))
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
  (* 2 (pow 2 5))
  ↓
  (* 2 (* 2 (pow 2 4)))
  ↓
  (* 2 (* 2 (* 2 (pow 2 3))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (pow 2 2)))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 1))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 0)))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 1)))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 2)))))
  ↓
  (* 2 (* 2 (* 2 (* 2 4))))
  ↓
  (* 2 (* 2 (* 2 8)))
  ↓
  (* 2 (* 2 16))
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
  (* 2 (pow 2 5))
  ↓
  (* 2 (* 2 (pow 2 4)))
  ↓
  (* 2 (* 2 (* 2 (pow 2 3))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (pow 2 2)))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 1))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 0)))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 1)))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 2)))))
  ↓
  (* 2 (* 2 (* 2 (* 2 4))))
  ↓
  (* 2 (* 2 (* 2 8)))
  ↓
  (* 2 (* 2 16))
  ↓
  (* 2 32)
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
  (* 2 (pow 2 5))
  ↓
  (* 2 (* 2 (pow 2 4)))
  ↓
  (* 2 (* 2 (* 2 (pow 2 3))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (pow 2 2)))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 1))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 0)))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 1)))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 2)))))
  ↓
  (* 2 (* 2 (* 2 (* 2 4))))
  ↓
  (* 2 (* 2 (* 2 8)))
  ↓
  (* 2 (* 2 16))
  ↓
  (* 2 32)
  ↓
  64
```

# Оценка на степенуване

```
(pow 2 6)
  ↓
  (* 2 (pow 2 5))
  ↓
  (* 2 (* 2 (pow 2 4)))
  ↓
  (* 2 (* 2 (* 2 (pow 2 3))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (pow 2 2)))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 1))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (pow 2 0)))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 (* 2 1)))))))
  ↓
  (* 2 (* 2 (* 2 (* 2 (* 2 2)))))
  ↓
  (* 2 (* 2 (* 2 (* 2 4))))
  ↓
  (* 2 (* 2 (* 2 8)))
  ↓
  (* 2 (* 2 16))
  ↓
  (* 2 32)
  ↓
  64
```

Линеен рекурсивен процес

## Бързо степенуване

Алтернативна дефиниция на  $x^n$ :

$$x^n = \begin{cases} 1, & \text{ако } n = 0, \\ \frac{1}{x^{-n}}, & \text{ако } n < 0, \\ (x^{\frac{n}{2}})^2, & \text{ако } n > 0, n - \text{четно,} \\ x \cdot x^{n-1}, & \text{ако } n > 0, n - \text{нечетно.} \end{cases}$$

## Бързо степенуване

Алтернативна дефиниция на  $x^n$ :

$$x^n = \begin{cases} 1, & \text{ако } n = 0, \\ \frac{1}{x^{-n}}, & \text{ако } n < 0, \\ (x^{\frac{n}{2}})^2, & \text{ако } n > 0, n \text{ — четно,} \\ x \cdot x^{n-1}, & \text{ако } n > 0, n \text{ — нечетно.} \end{cases}$$

```
(define (qpow x n)
  (define (sqr x) (* x x))
  (cond ((= n 0) 1)
        ((< n 0) (/ 1 (qpow x (- n)))))
        ((even? n) (sqr (qpow x (quotient n 2)))))
        (else (* x (qpow x (- n 1))))))
```

# Оценка на бързо степенуване

(qpow 2 6)

# Оценка на бързо степенуване

$$\begin{array}{c} (\text{qpow } 2 \ 6) \\ \downarrow \\ (\text{sqr } (\text{qpow } 2 \ 3)) \end{array}$$

# Оценка на бързо степенуване

```
(qpow 2 6)
  ↓
(sqr (qpow 2 3))
  ↓
(sqr (* 2 (qpow 2 2)))
```

# Оценка на бързо степенуване

```
(qpow 2 6)
  ↓
(sqr (qpow 2 3))
  ↓
(sqr (* 2 (qpow 2 2)))
  ↓
(sqr (* 2 (sqr (qpow 2 1)))))
```

# Оценка на бързо степенуване

```
(qpow 2 6)
  ↓
(sqr (qpow 2 3))
  ↓
(sqr (* 2 (qpow 2 2)))
  ↓
(sqr (* 2 (sqr (qpow 2 1)))))
  ↓
(sqr (* 2 (sqr (* 2 (qpow 2 0))))))
```

# Оценка на бързо степенуване

```
(qpow 2 6)
  ↓
(sqr (qpow 2 3))
  ↓
(sqr (* 2 (qpow 2 2)))
  ↓
(sqr (* 2 (sqr (qpow 2 1)))))
  ↓
(sqr (* 2 (sqr (* 2 (qpow 2 0))))))
  ↓
(sqr (* 2 (sqr (* 2 1))))
```

# Оценка на бързо степенуване

```
(qpow 2 6)
  ↓
(sqr (qpow 2 3))
  ↓
(sqr (* 2 (qpow 2 2)))
  ↓
(sqr (* 2 (sqr (qpow 2 1)))))
  ↓
(sqr (* 2 (sqr (* 2 (qpow 2 0))))))
  ↓
(sqr (* 2 (sqr (* 2 1))))
  ↓
(sqr (* 2 (sqr 2)))
```

# Оценка на бързо степенуване

```
(qpow 2 6)
  ↓
(sqr (qpow 2 3))
  ↓
(sqr (* 2 (qpow 2 2)))
  ↓
(sqr (* 2 (sqr (qpow 2 1)))))
  ↓
(sqr (* 2 (sqr (* 2 (qpow 2 0))))))
  ↓
(sqr (* 2 (sqr (* 2 1))))
  ↓
(sqr (* 2 (sqr 2)))
  ↓
(sqr (* 2 4))
```

# Оценка на бързо степенуване

```
(qpow 2 6)
  ↓
(sqr (qpow 2 3))
  ↓
(sqr (* 2 (qpow 2 2)))
  ↓
(sqr (* 2 (sqr (qpow 2 1)))))
  ↓
(sqr (* 2 (sqr (* 2 (qpow 2 0))))))
  ↓
(sqr (* 2 (sqr (* 2 1))))
  ↓
(sqr (* 2 (sqr 2)))
  ↓
(sqr (* 2 4))
  ↓
(sqr 8)
```

# Оценка на бързо степенуване

```
(qpow 2 6)
  ↓
(sqr (qpow 2 3))
  ↓
(sqr (* 2 (qpow 2 2)))
  ↓
(sqr (* 2 (sqr (qpow 2 1)))))
  ↓
(sqr (* 2 (sqr (* 2 (qpow 2 0))))))
  ↓
(sqr (* 2 (sqr (* 2 1))))
  ↓
(sqr (* 2 (sqr 2)))
  ↓
(sqr (* 2 4))
  ↓
(sqr 8)
  ↓
64
```

# Оценка на бързо степенуване

$\mathcal{O}(\log n)$

$\downarrow$

$\mathcal{O}(\log \log n)$

```
(qpow 2 6)
↓
(sqr (qpow 2 3))
↓
(sqr (* 2 (qpow 2 2)))
↓
(sqr (* 2 (sqr (qpow 2 1)))))
↓
(sqr (* 2 (sqr (* 2 (qpow 2 0))))))
↓
(sqr (* 2 (sqr (* 2 1))))
↓
(sqr (* 2 (sqr 2)))
↓
(sqr (* 2 4))
↓
(sqr 8)
↓
64
```

Логаритмичен рекурсивен процес

# Числа на Фиbonачи

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

# Числа на Фиbonачи

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

$$f_n = \begin{cases} 0, & \text{за } n = 0, \\ 1, & \text{за } n = 1, \\ f_{n-1} + f_{n-2}, & \text{за } n \geq 2. \end{cases}$$

# Числа на Фиbonачи

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

$$f_n = \begin{cases} 0, & \text{за } n = 0, \\ 1, & \text{за } n = 1, \\ f_{n-1} + f_{n-2}, & \text{за } n \geq 2. \end{cases}$$

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1)) (fib (- n 2))))))
```

# Числа на Фиbonачи

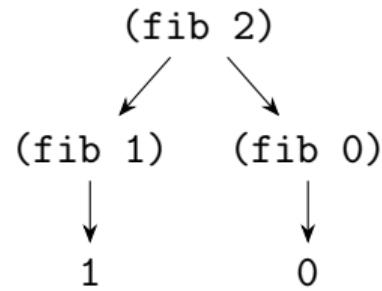
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

$$f_n = \begin{cases} 0, & \text{за } n = 0, \\ 1, & \text{за } n = 1, \\ f_{n-1} + f_{n-2}, & \text{за } n \geq 2. \end{cases}$$

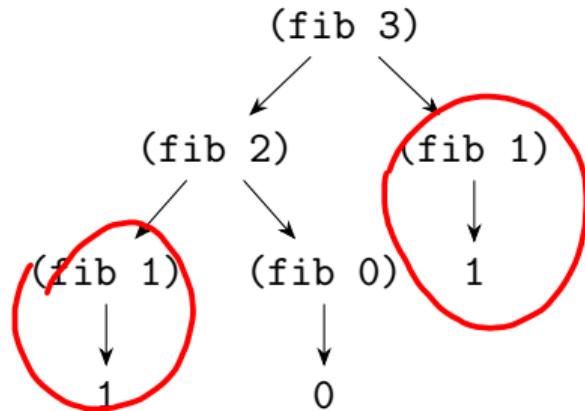
```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1)) (fib (- n 2))))))
```

$$f_{40} = ?$$

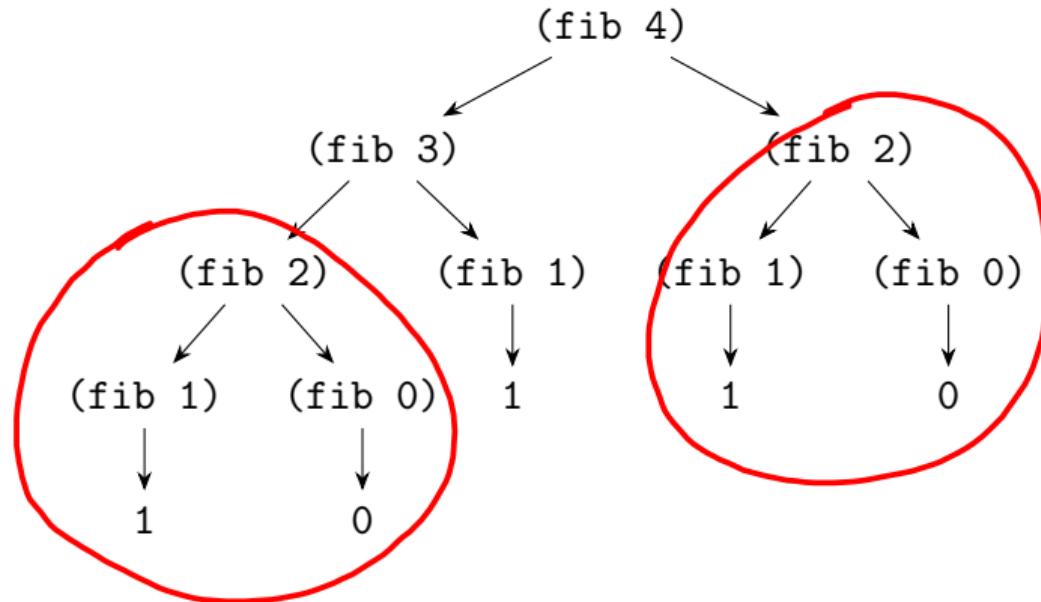
# Дървовидна рекурсия



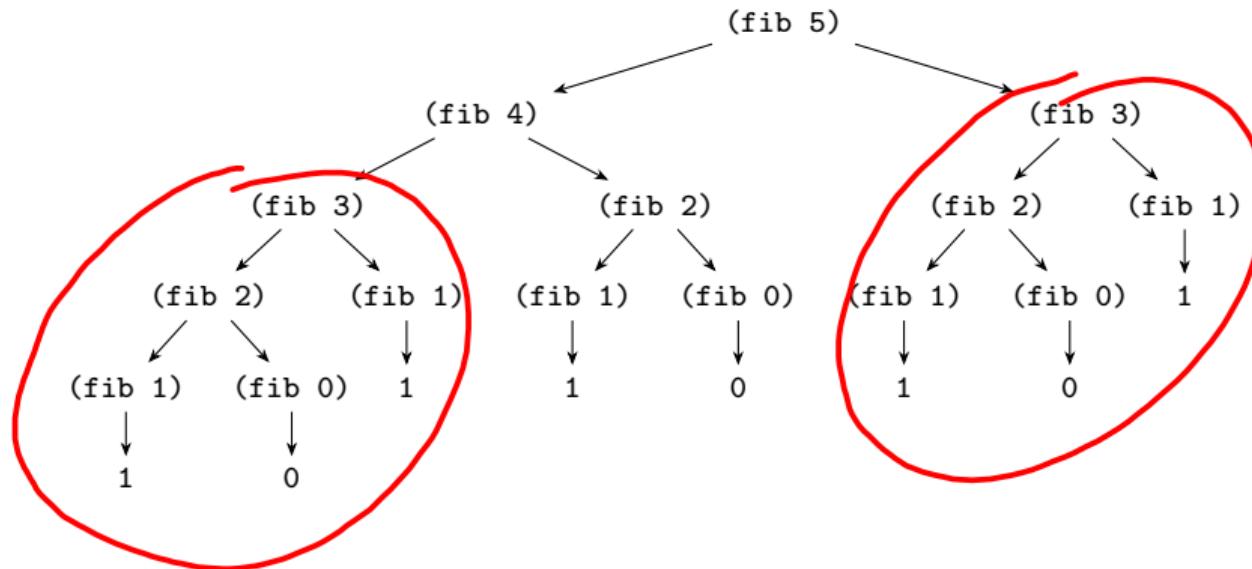
## Дърводидна рекурсия



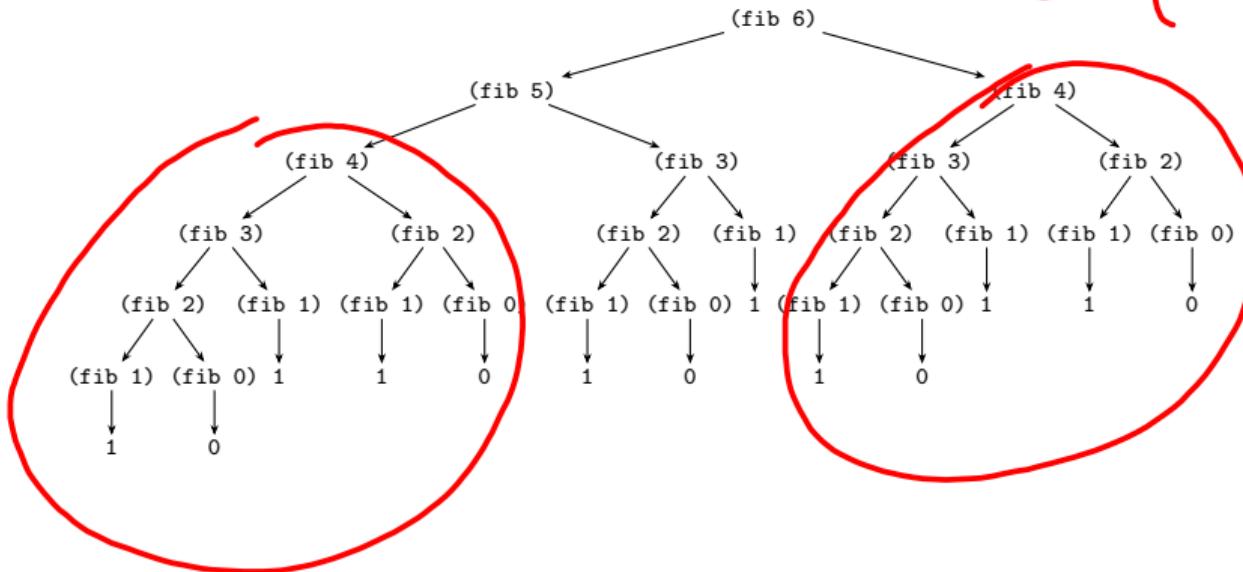
# Дърводидна рекурсия



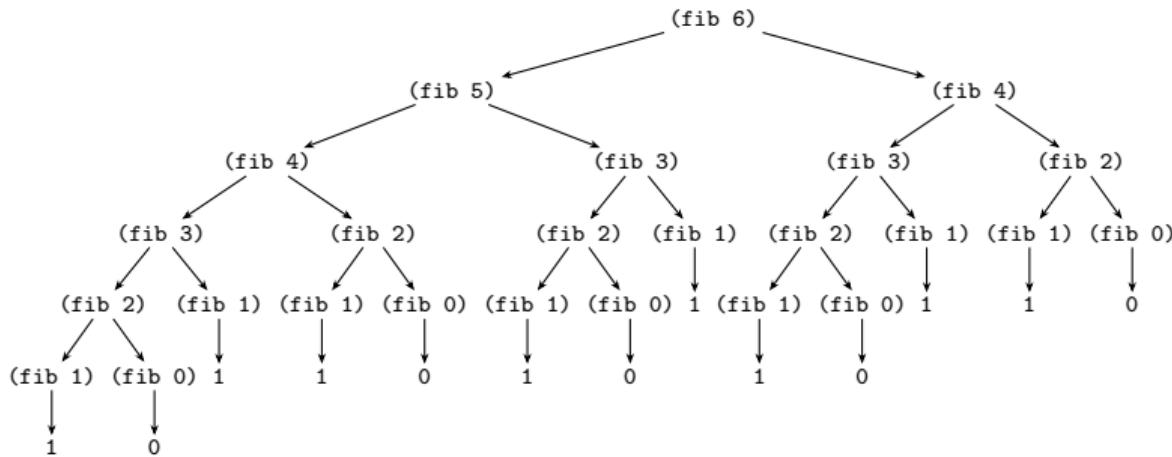
# Дърводидна рекурсия



## Дърводидна рекурсия

 $O(\varphi^h)$ 

# Дърводидна рекурсия



Дърводиден рекурсивен процес

# Как да оптимизираме?

## Решение №1: мемоизация

Да помним вече пресметнатите стойности, вместо да ги смятаме пак.

# Как да оптимизираме?

## Решение №1: мемоизация

Да помним вече пресметнатите стойности, вместо да ги смятаме пак.

За ефективна реализация обикновено са нужни странични ефекти.

# Как да оптимизираме?

## Решение №1: мемоизация

Да помним вече пресметнатите стойности, вместо да ги смятаме пак.

За ефективна реализация обикновено са нужни странични ефекти.

## Решение №2: динамично програмиране

Строим последователно всички числа на Фибоначи в нарастващ ред.

# Как да оптимизираме?

## Решение №1: мемоизация

Да помним вече пресметнатите стойности, вместо да ги смятаме пак.

За ефективна реализация обикновено са нужни странични ефекти.

## Решение №2: динамично програмиране

Строим последователно всички числа на Фибоначи в нарастващ ред.

Нужно е да помним само последните две числа!

# Как да оптимизираме?

## Решение №1: мемоизация

Да помним вече пресметнатите стойности, вместо да ги смятаме пак.  
За ефективна реализация обикновено са нужни странични ефекти.

## Решение №2: динамично програмиране

Строим последователно всички числа на Фибоначи в нарастващ ред.

Нужно е да помним само последните две числа!

```
(define (fib n)
  (define (iter i fi fi-1)
    (if (= i n) fi
        (iter (+ i 1) (+ fi fi-1) fi)))
  (if (= n 0) 0
      (iter 1 1 0)))
```

$$\overset{\cdot}{i+1} - \overset{\cdot}{1} = \overset{\cdot}{i}$$

# Итеративно генериране на числата на Фибоначи

(fib 7)

# Итеративно генериране на числата на Фибоначи

```
(fib 7)
  ↓
(itер 1 1 0)
```

# Итеративно генериране на числата на Фибоначи

```
(fib 7)
  ↓
(itер 1 1 0)
  ↓
(itер 2 1 1)
```

# Итеративно генериране на числата на Фибоначи

```
(fib 7)
  ↓
(itер 1 1 0)
  ↓
(itер 2 1 1)
  ↓
(itер 3 2 1)
```

# Итеративно генериране на числата на Фибоначи

```
(fib 7)
  ↓
(itер 1 1 0)
  ↓
(itер 2 1 1)
  ↓
(itер 3 2 1)
  ↓
(itер 4 3 2)
```

# Итеративно генериране на числата на Фибоначи

```
(fib 7)
  ↓
 (iter 1 1 0)
  ↓
 (iter 2 1 1)
  ↓
 (iter 3 2 1)
  ↓
 (iter 4 3 2)
  ↓
 (iter 5 5 3)
```

# Итеративно генериране на числата на Фибоначи

```
(fib 7)
  ↓
 (iter 1 1 0)
  ↓
 (iter 2 1 1)
  ↓
 (iter 3 2 1)
  ↓
 (iter 4 3 2)
  ↓
 (iter 5 5 3)
  ↓
 (iter 6 8 5)
```

# Итеративно генериране на числата на Фибоначи

```
(fib 7)
  ↓
 (iter 1 1 0)
  ↓
 (iter 2 1 1)
  ↓
 (iter 3 2 1)
  ↓
 (iter 4 3 2)
  ↓
 (iter 5 5 3)
  ↓
 (iter 6 8 5)
  ↓
 (iter 7 13 8)
```

# Итеративно генериране на числата на Фибоначи

```
(fib 7)
  ↓
 (iter 1 1 0)
  ↓
 (iter 2 1 1)
  ↓
 (iter 3 2 1)
  ↓
 (iter 4 3 2)
  ↓
 (iter 5 5 3)
  ↓
 (iter 6 8 5)
  ↓
 (iter 7 13 8)
  ↓
 13
```