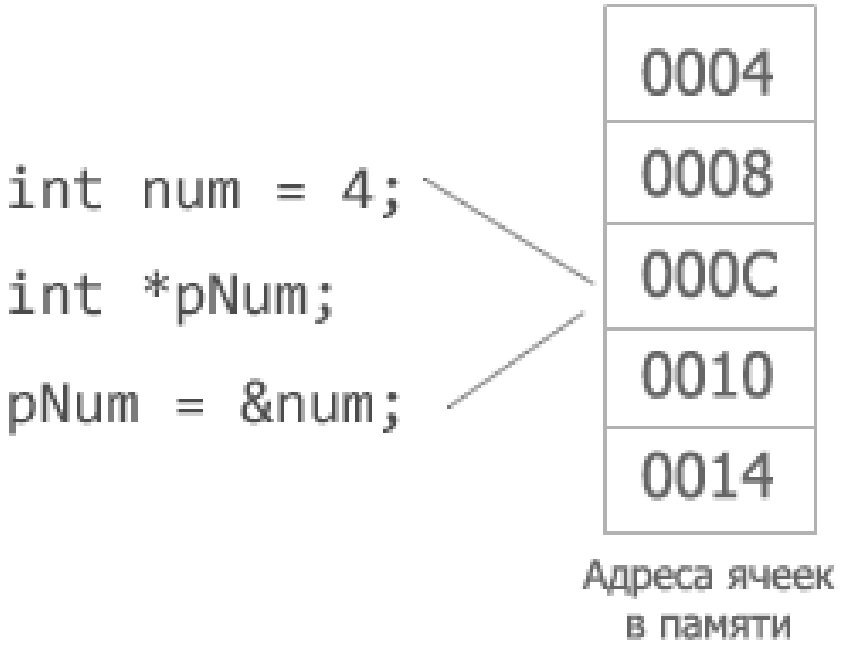
Динамическая память

Указатели:

- **& – возвращает адрес**
- *** – получение значения по адресу**




```
int  *p;
int  k=5;
int  *t;

p=&k;
t=p;
```

Динамическая память

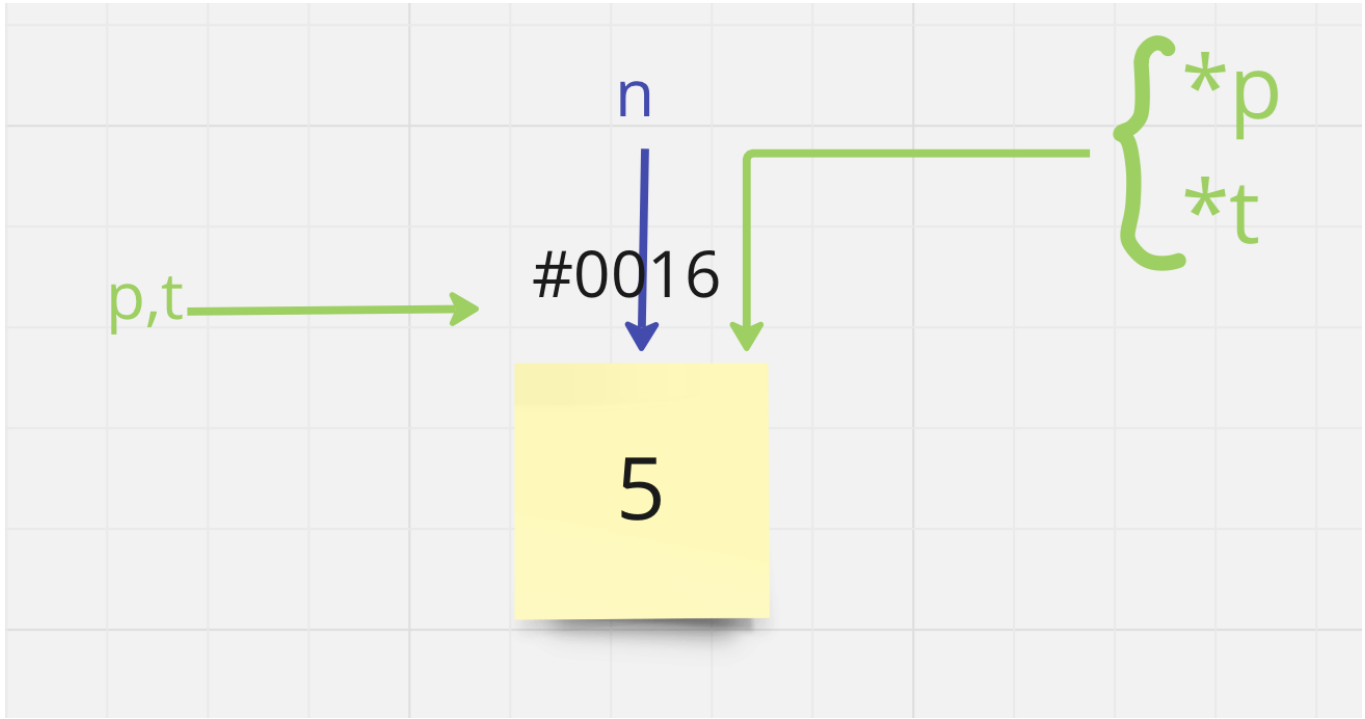
Указатели:

- **& – возвращает адрес**
- *** – получение значения по адресу**

Динамическая память

Указатели:

- **& – возвращает адрес**
- *** – получение значения по адресу**

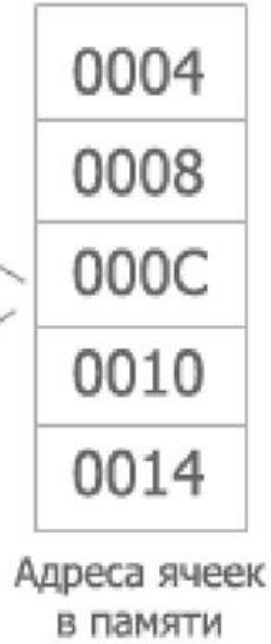


Динамическая память

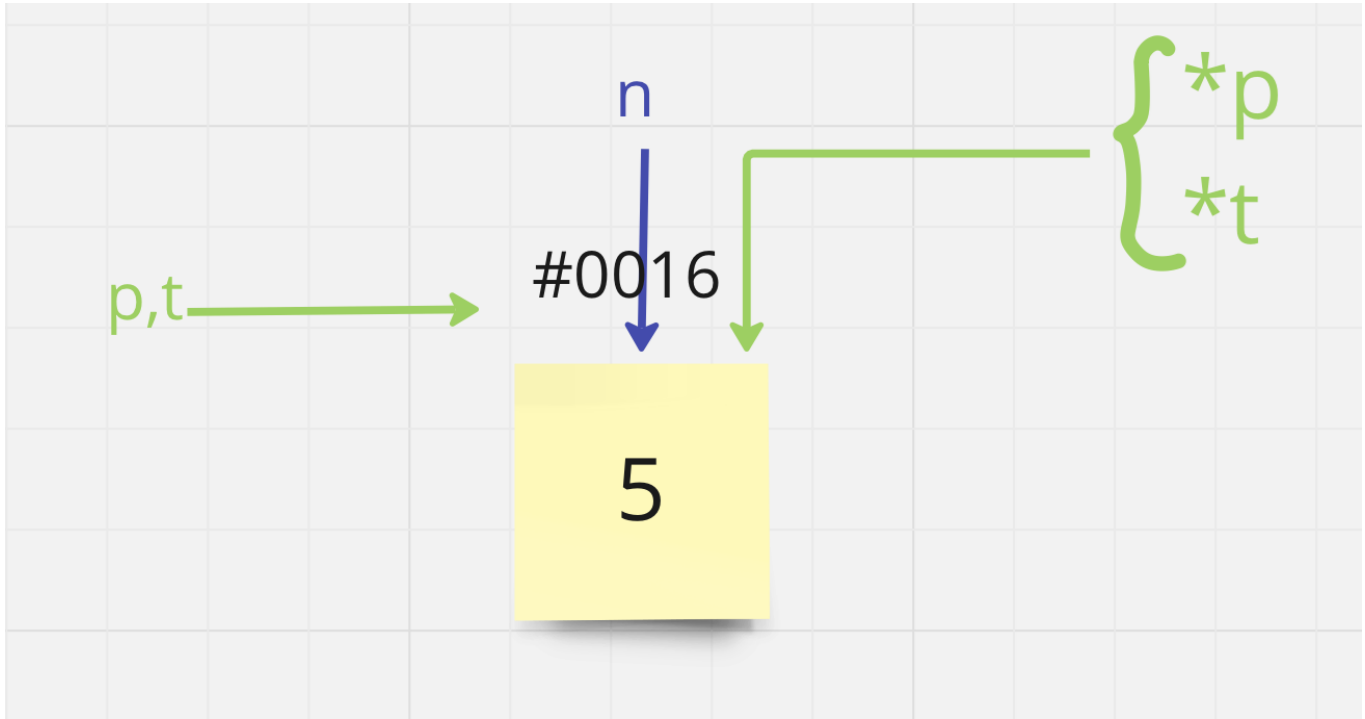
Указатели:

- **& – возвращает адрес**
- *** – получение значения по адресу**

```
int num = 4;
int *pNum;
pNum = &num;
```




```
int  *p;
int  k=5;
int  *t;

p=&k;
t=p;
```



МНОГО Динамической памяти

```
int  *p;
int  *k;
int  *t;
int  *d;


*p=5;
*k=12;
*t=88;
*d=99;
```

Много указателей!

Как работать с ними как с единым целым?

МНОГО Динамической памяти

```
int  *p;
int  *k;
int  *t;
int  *d;


*p=5;
*k=12;
*t=88;
*d=99;
```

Много указателей!

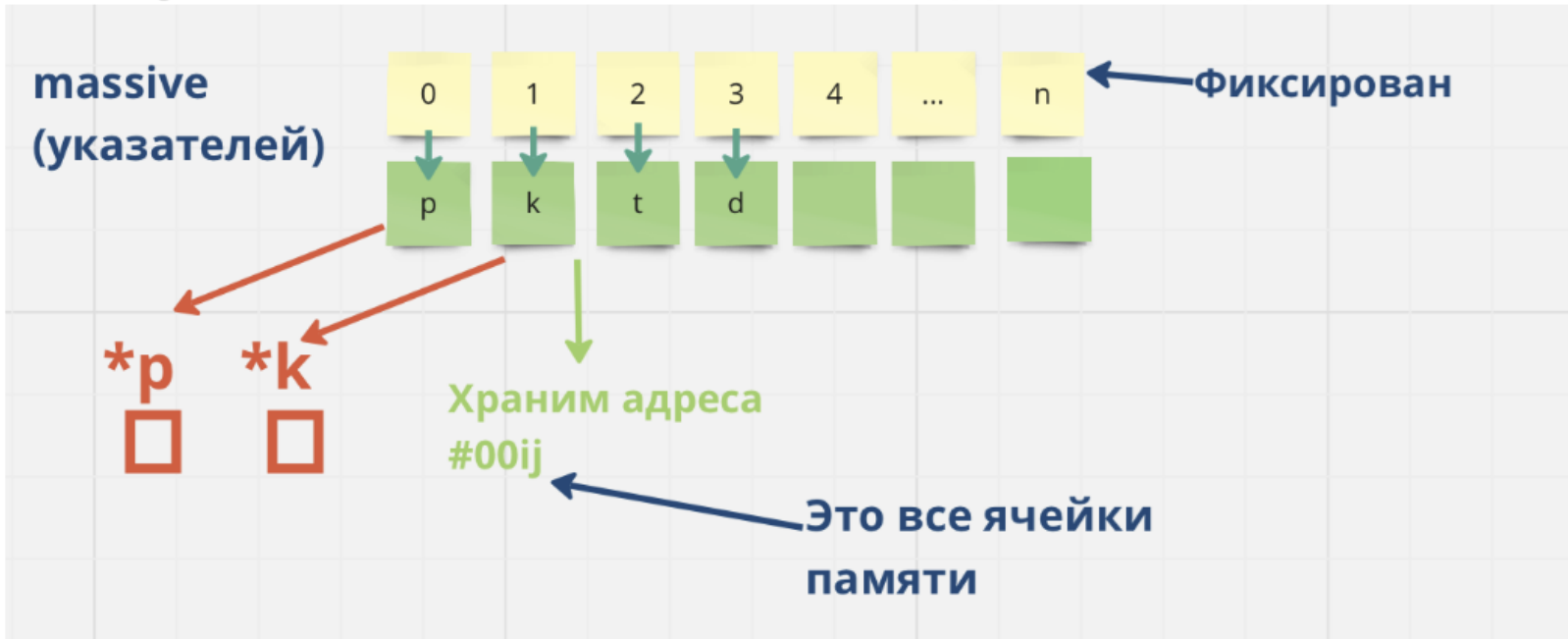
Как работать с ними как с единым целым?

МНОГО Динамической памяти

```
int  *p;
int  *k;
int  *t;
int  *d;
//Выделили память
*p=5;
*k=12;
*t=88;
*d=99;
```

Много указателей!

Как работать с ними как с единым целым?



Не нужно будет создавать кучу переменных,
все будет теперь в massive[i]

вместо *p, *t, *k, *d

МНОГО Динамической памяти

```
int  *p;
int  *k;
int  *t;
int  *d;
????
```

1) НЕ известно сколько именно указателей понадобиться!

Динамическая - динамическая память!

Создавать по необходимости!

2) Объединять как, как рассмотреть как единую структуру?

нужно связать ячейки памяти между собой

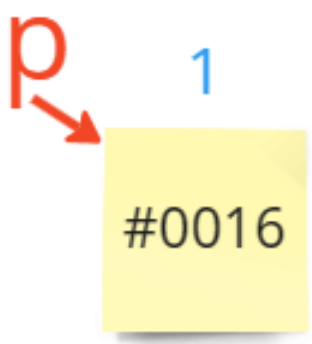
МНОГО Динамической памяти

```
int  *p;
int  *k;
int  *t;
int  *d;
????
```

1) НЕ известно сколько именно указателей понадобиться!

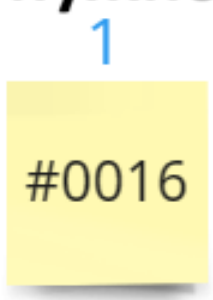
Динамическая - динамическая память!

Создавать по необходимости!



2) Объединять как, как рассмотреть как единую структуру?

нужно связать ячейки памяти между собой



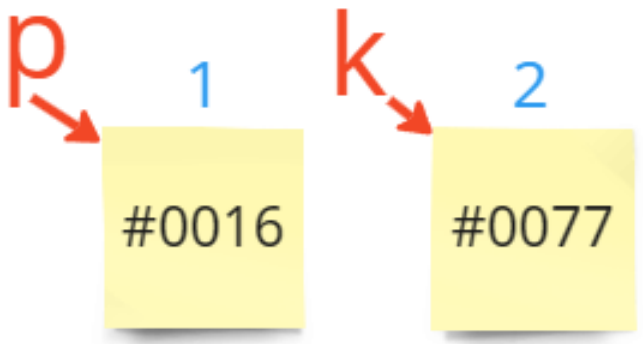
МНОГО Динамической памяти

```
int  *p;
int  *k;
int  *t;
int  *d;
????
```

1) НЕ известно сколько именно указателей понадобиться!

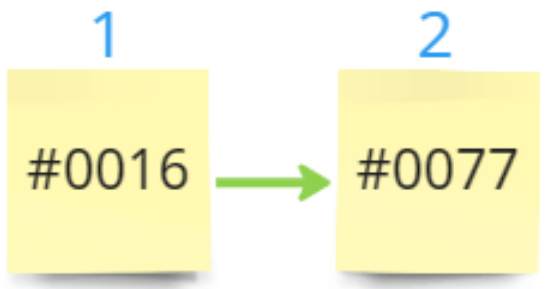
Динамическая - динамическая память!

Создавать по необходимости!



2) Объединять как, как рассмотреть как единую структуру?

нужно связать ячейки памяти между собой



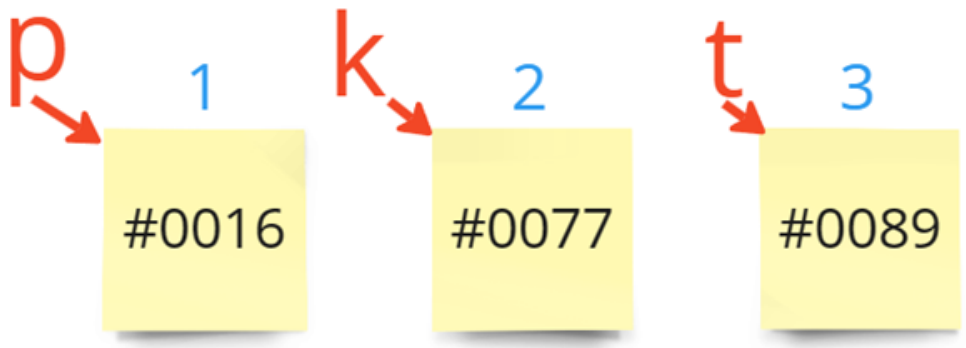
МНОГО Динамической памяти

```
int  *p;
int  *k;
int  *t;
int  *d;
????
```

1) НЕ известно сколько именно указателей понадобиться!

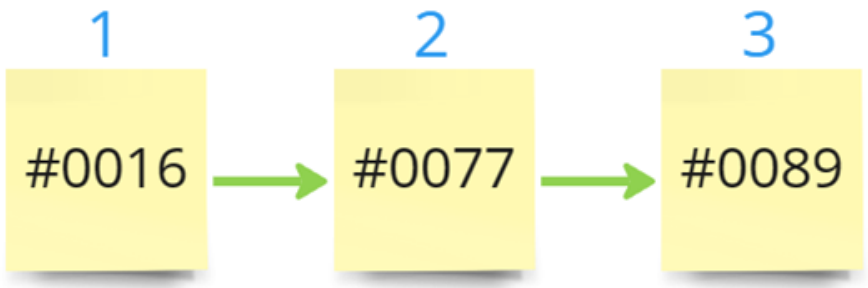
Динамическая - динамическая память!

Создавать по необходимости!



2) Объединять как, как рассмотреть как единую структуру?

нужно связать ячейки памяти между собой



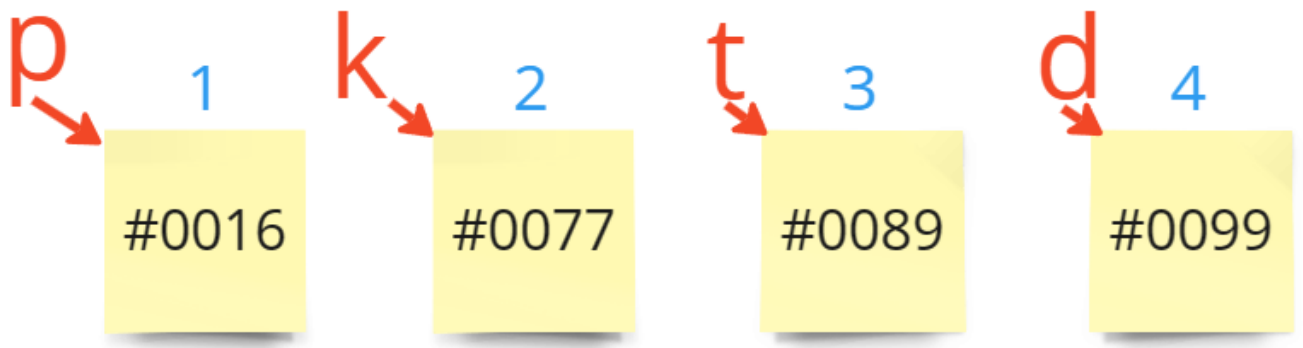
МНОГО Динамической памяти

```
int  *p;
int  *k;
int  *t;
int  *d;
????
```

1) НЕ известно сколько именно указателей понадобиться!

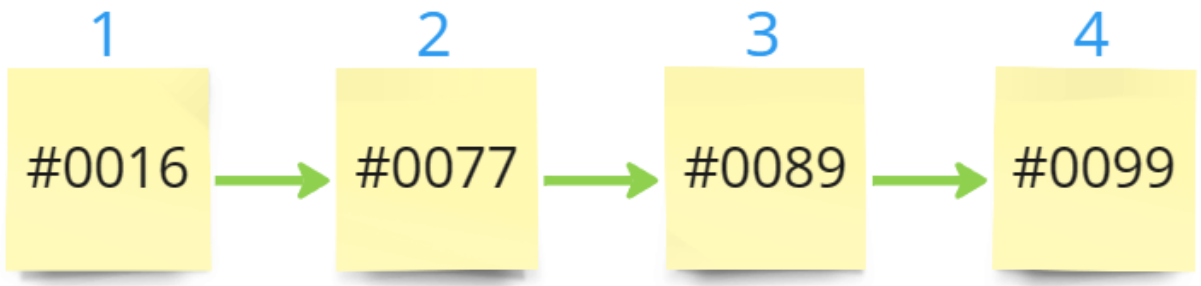
Динамическая - динамическая память!

Создавать по необходимости!



2) Объединять как, как рассмотреть как единую структуру?

нужно связать ячейки памяти между собой



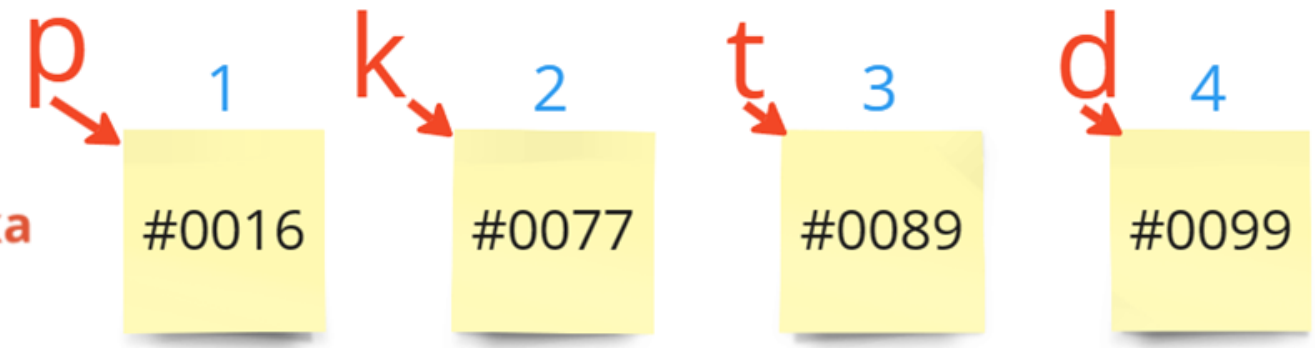
МНОГО Динамической памяти

```
int  *p;
int  *k;
int  *t;
int  *d;
????
```

1) НЕ известно сколько именно указателей понадобиться!

Динамическая - динамическая память!

Создавать по необходимости!

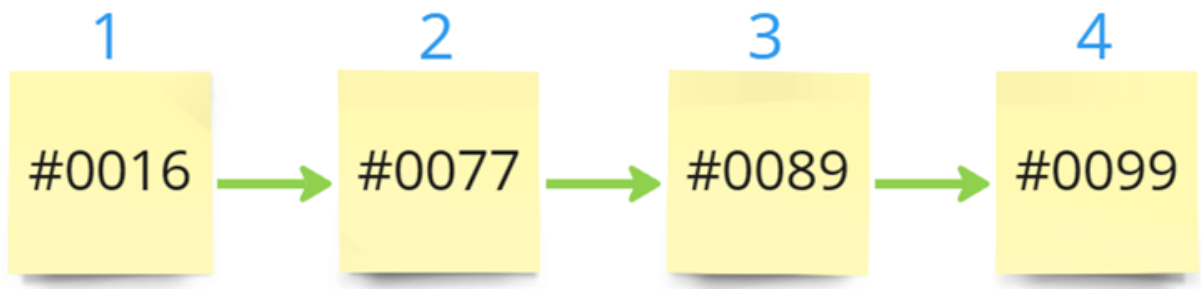


Может 151 ячейка нужна будет

151 переменная не вариант

2) Объединять как, как рассмотреть как единую структуру?

нужно связать ячейки памяти между собой

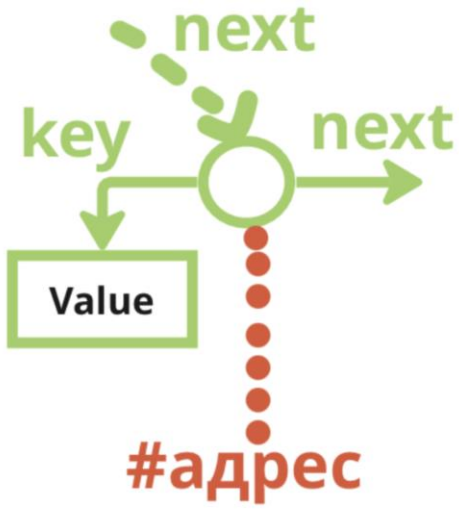


Динамическая память

Динамическая память

Связываем свободные ячейки памяти, как вагоны
в один поезд
Локомотив - это начало всего списка этих ячеек

Динамическая память



Связываем свободные ячейки памяти, как вагоны в один поезд

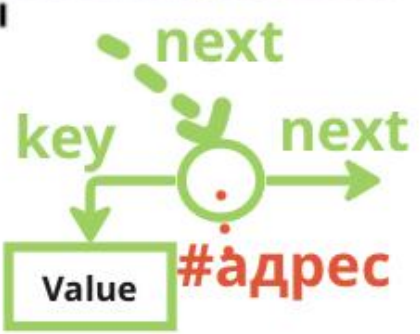
Локомотив - это начало всего списка этих ячеек

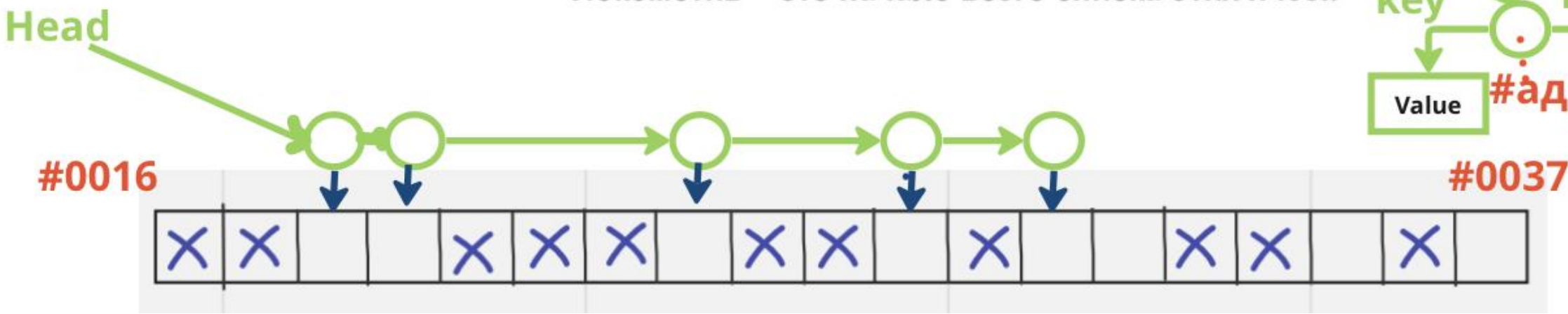
Динамическая память

важен локомотив

Связываем свободные ячейки памяти, как вагоны в один поезд

Локомотив - это начало всего списка этих ячеек



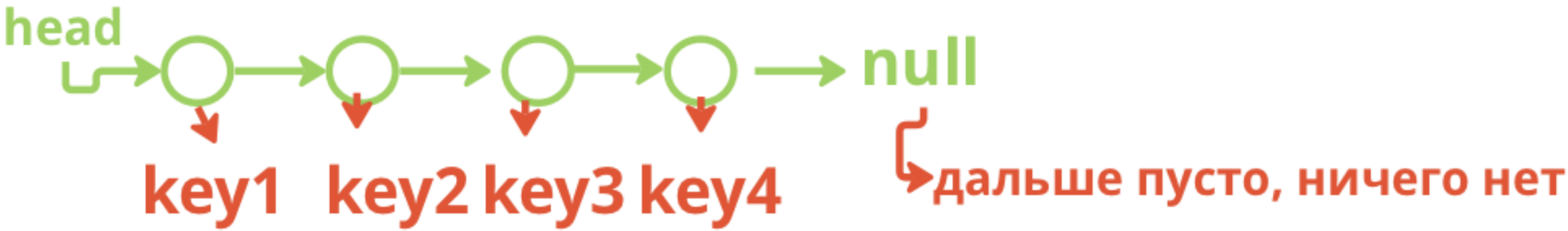


1) НЕ храним длину

2) Как определить последнюю ячейку

1) НЕ храним длину

2) Как определить последнюю ячейку (null / nil)



1) НЕ храним длину

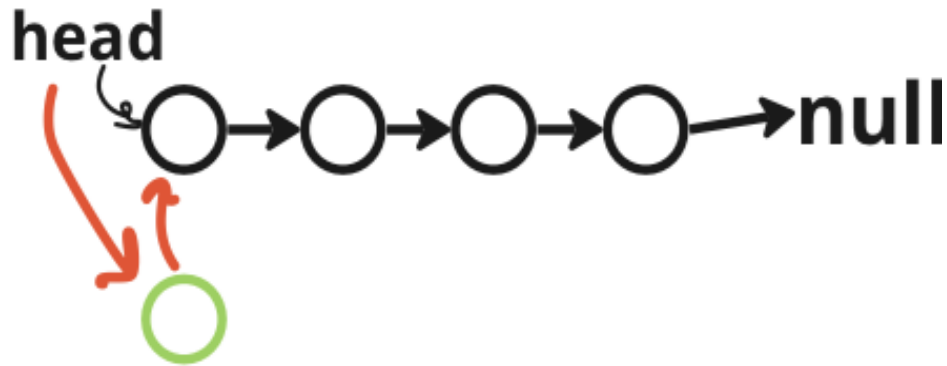
2) Как определить последнюю ячейку

3) Как привязать еще одну ячейку

1) НЕ храним длину

2) Как определить последнюю ячейку

3) Как привязать еще одну ячейку



1) НЕ храним длину

2) Как определить последнюю ячейку

3) Как привязать еще одну ячейку

4) Как удалять ячейку

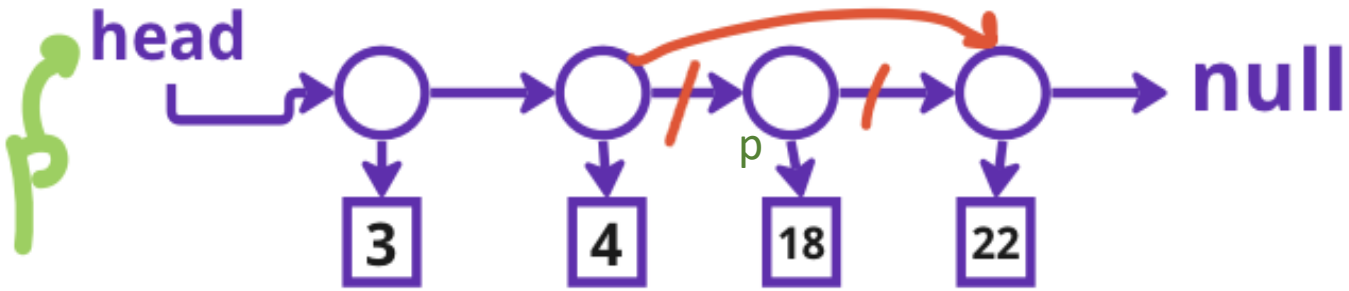
1) НЕ храним длину

2) Как определить последнюю ячейку

3) Как привязать еще одну ячейку

4) Как удалять ячейку

Delete 18



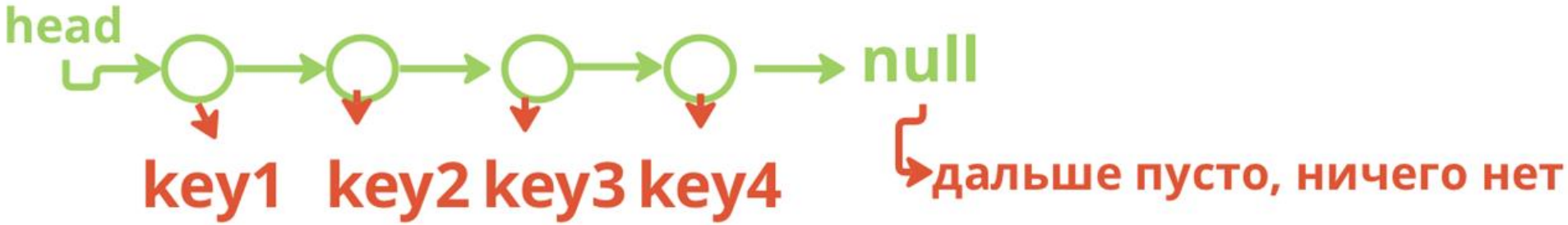
Освободить

Как убрать вагон из поезда

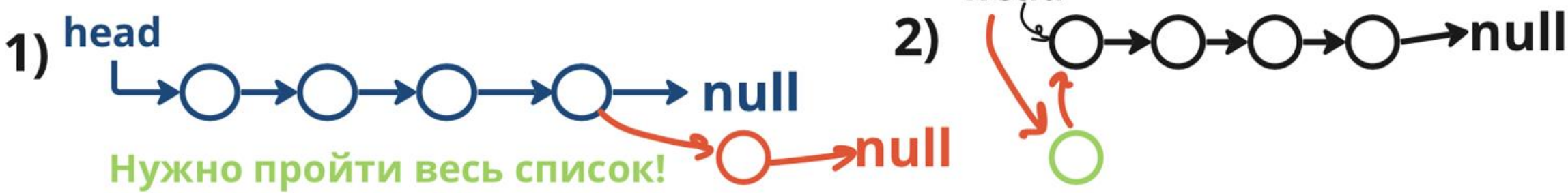
с помощью p-> key == 18

1) НЕ храним длину

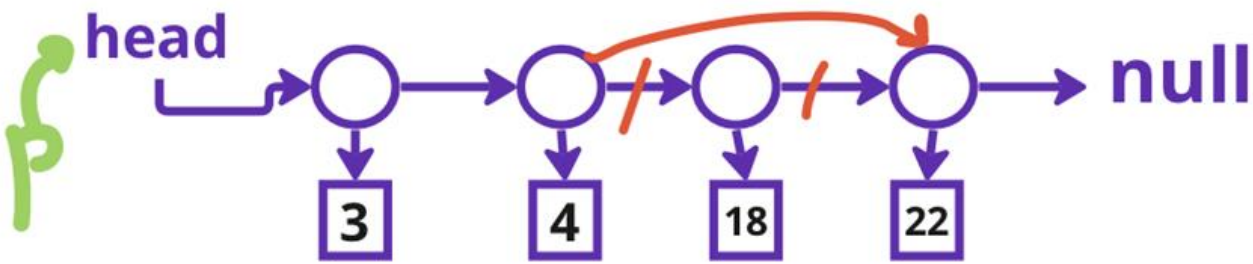
2) Как определить последнюю ячейку (null / nil)



3) Как привязать еще одну ячейку



4) Как удалять ячейку

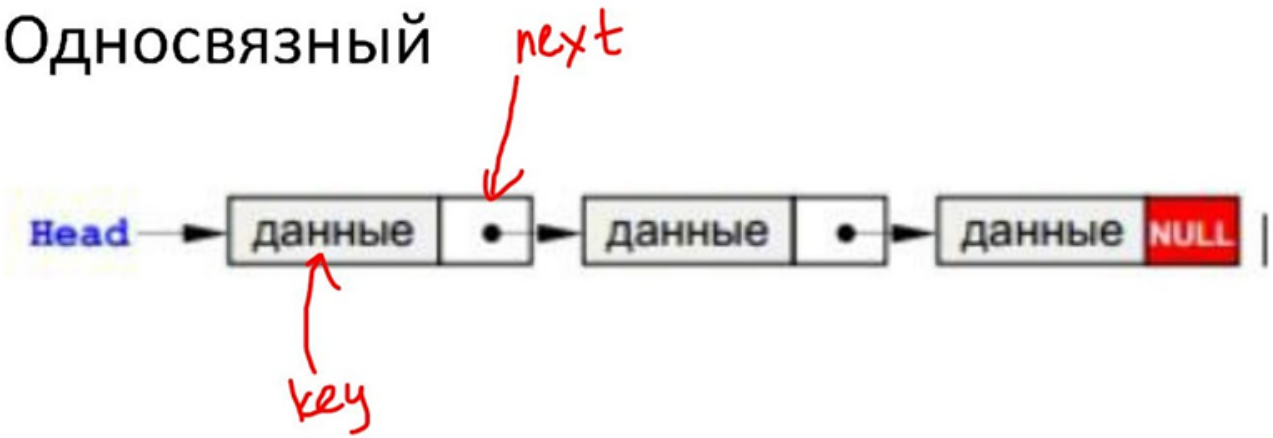


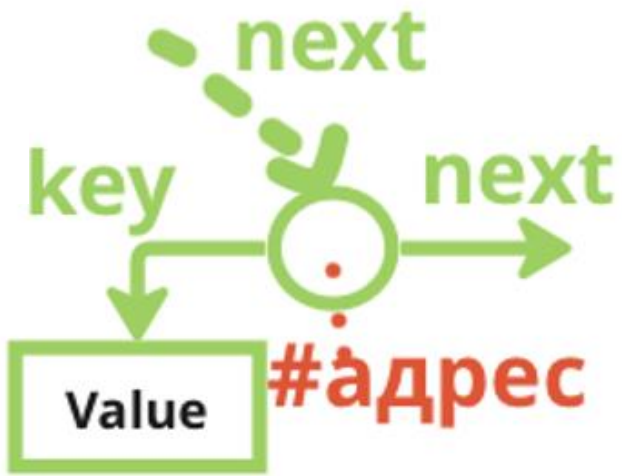
Как убрать вагон из поезда

Динамические связанные списки

```
struct Node
{
    int  key;   // значение
    struct Node *next;     //  адрес следующей ячейки памяти  - указатель на следующий элемент
};

Struct  Node * Head;
```

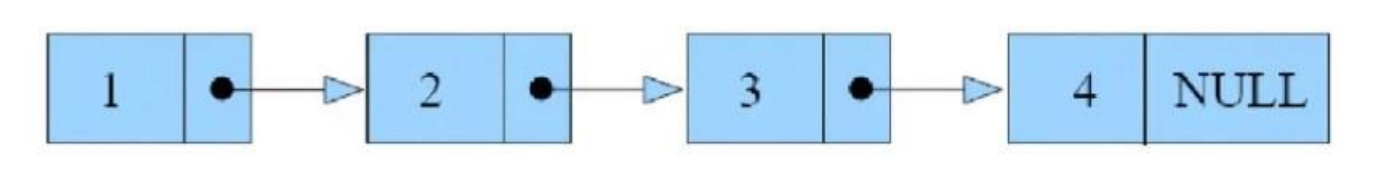
Односвязный





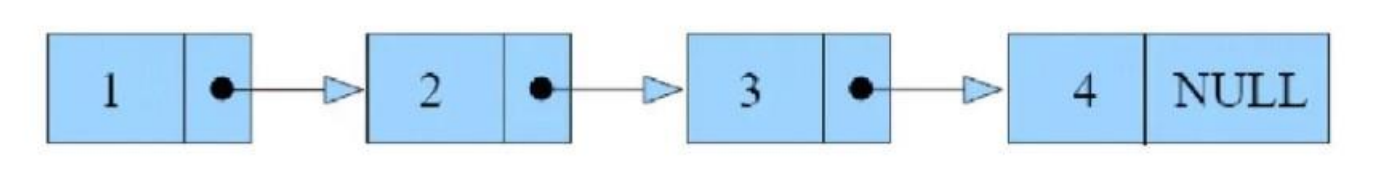
Односвязный список

```
struct Node
 {
     int  key;    // значение
     struct Node *next;      //  адрес следующей ячейки памяти  - указатель на следующий элемент
 };

Struct  Node * Head;
```



Односвязный список

```
struct Node
 {
     int  key;    // значение
     struct Node *next;      //  адрес следующей ячейки памяти  - указатель на следующий элемент
 };

Struct  Node * Head;
```



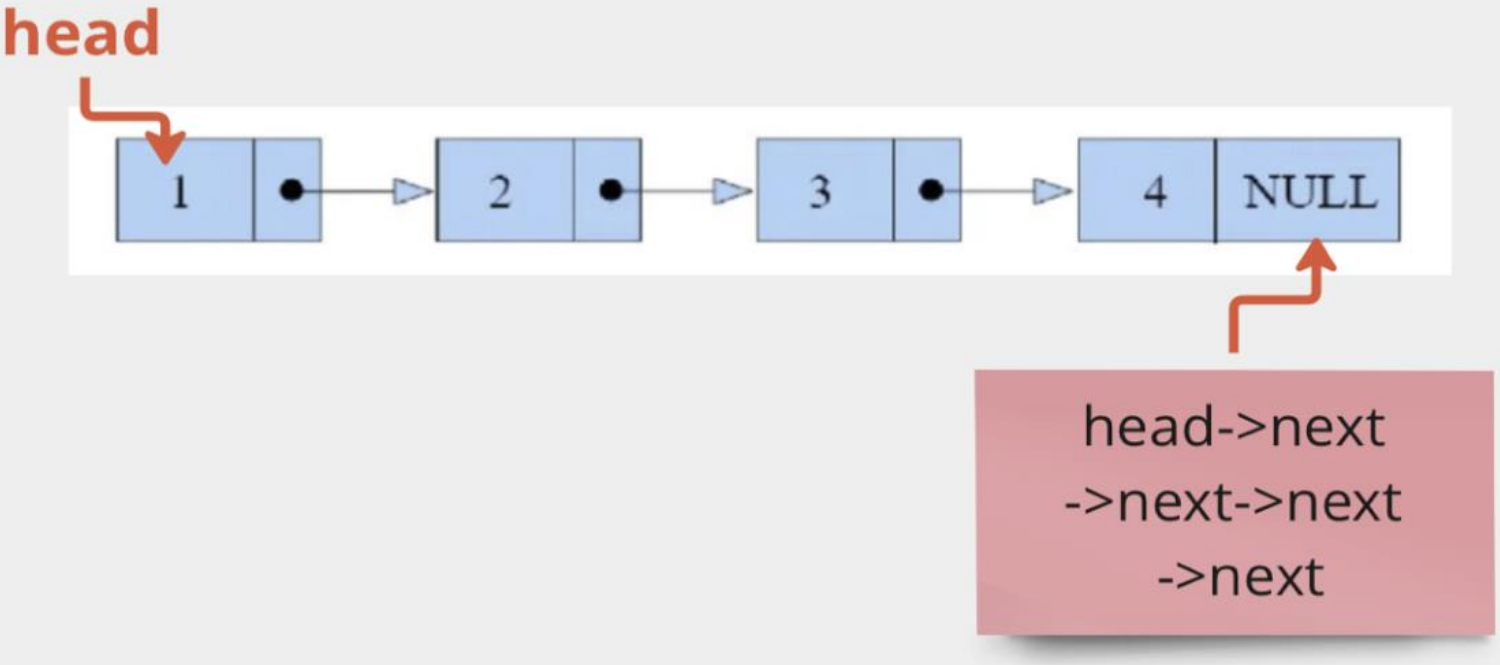
Head.next ?

Односвязный список

```
struct Node
{
    int  key;    // значение
    struct Node *next;      //  адрес следующей ячейки памяти  - указатель на следующий элемент
};

Struct  Node * Head;
```

Head.next ?

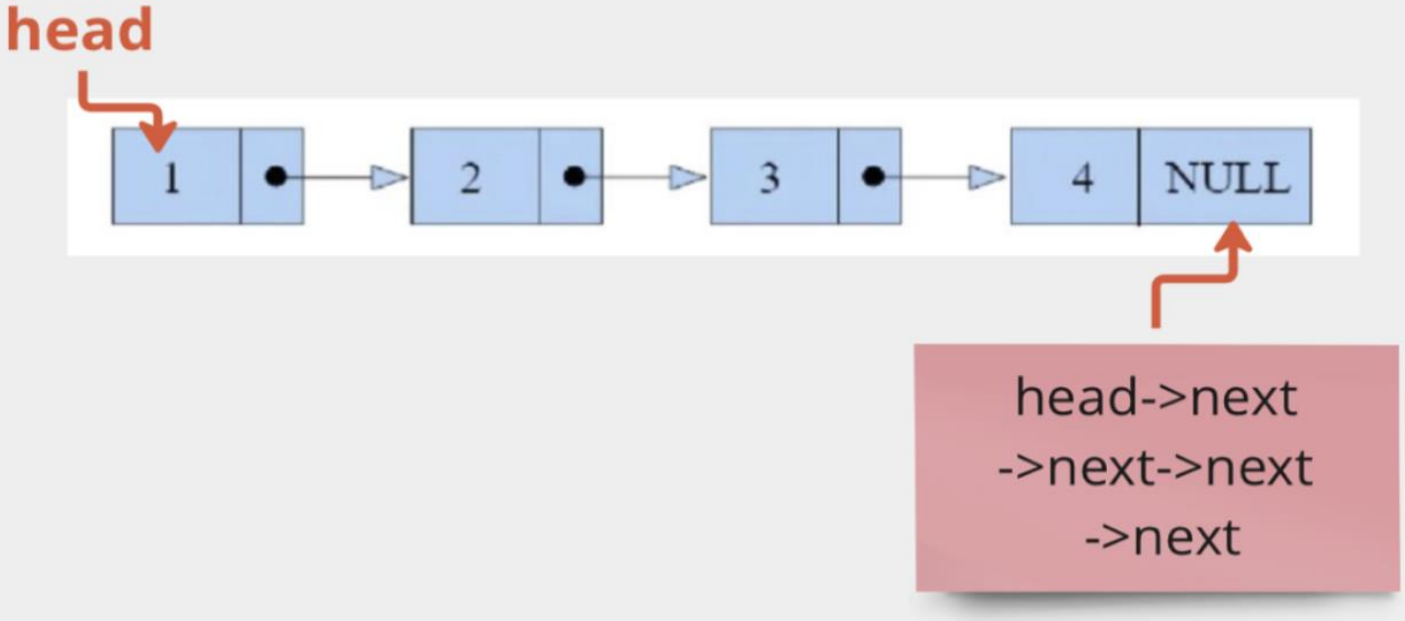
адрес/указатель



Односвязный список

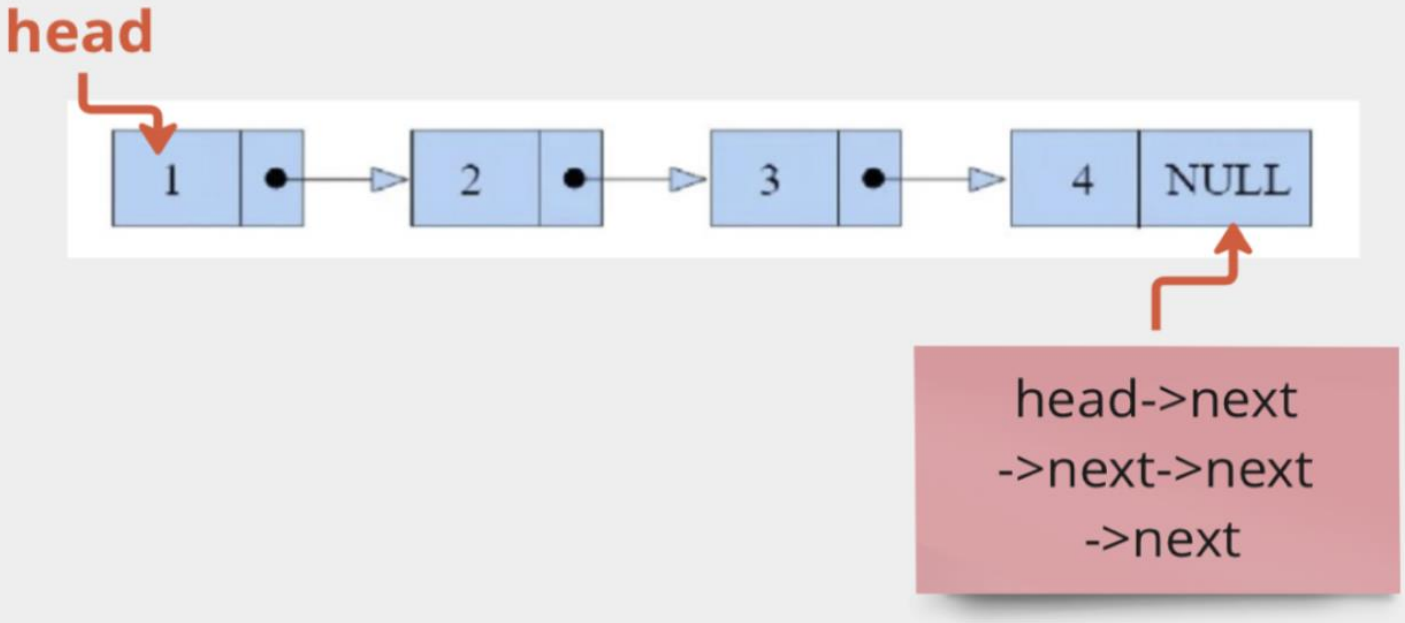
```
struct Node
{
    int  key;    // значение
    struct Node *next;      //  адрес следующей ячейки памяти  - указатель на следующий элемент
};

Struct  Node * Head;
```

Head.key ?



Односвязный список

```
struct Node
{
    int  key;    // значение
    struct Node *next;      //  адрес следующей ячейки памяти  - указатель на следующий элемент
};

Struct  Node * Head;
```

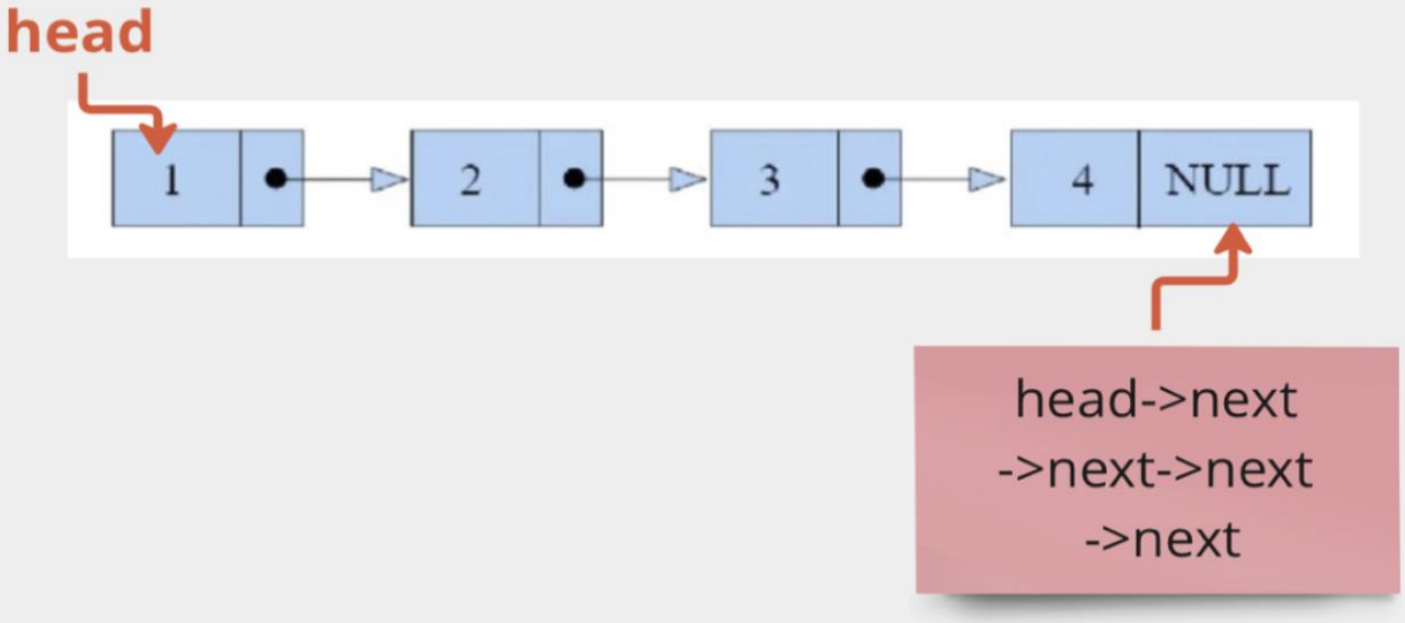


Head.key ? = 1

Односвязный список

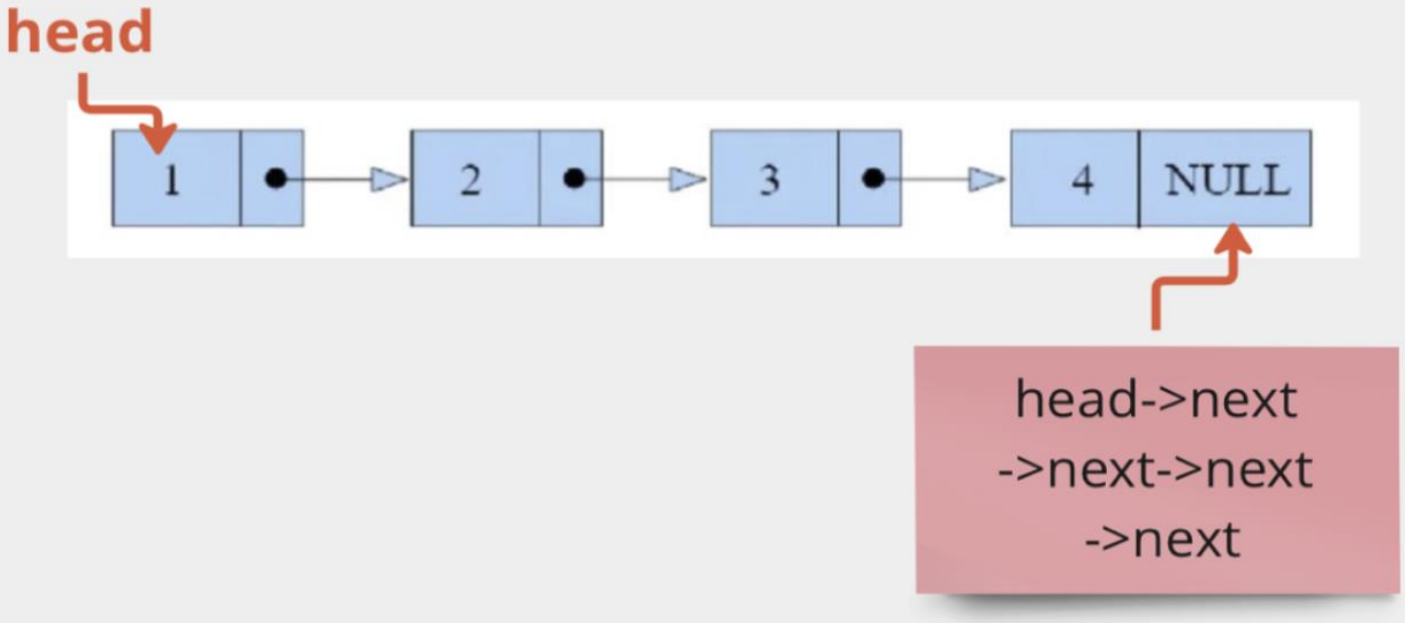
```
struct Node
{
    int  key;    // значение
    struct Node *next;      //  адрес следующей ячейки памяти  - указатель на следующий элемент
};

Struct  Node * Head;
```

Head.next.next?



Односвязный список

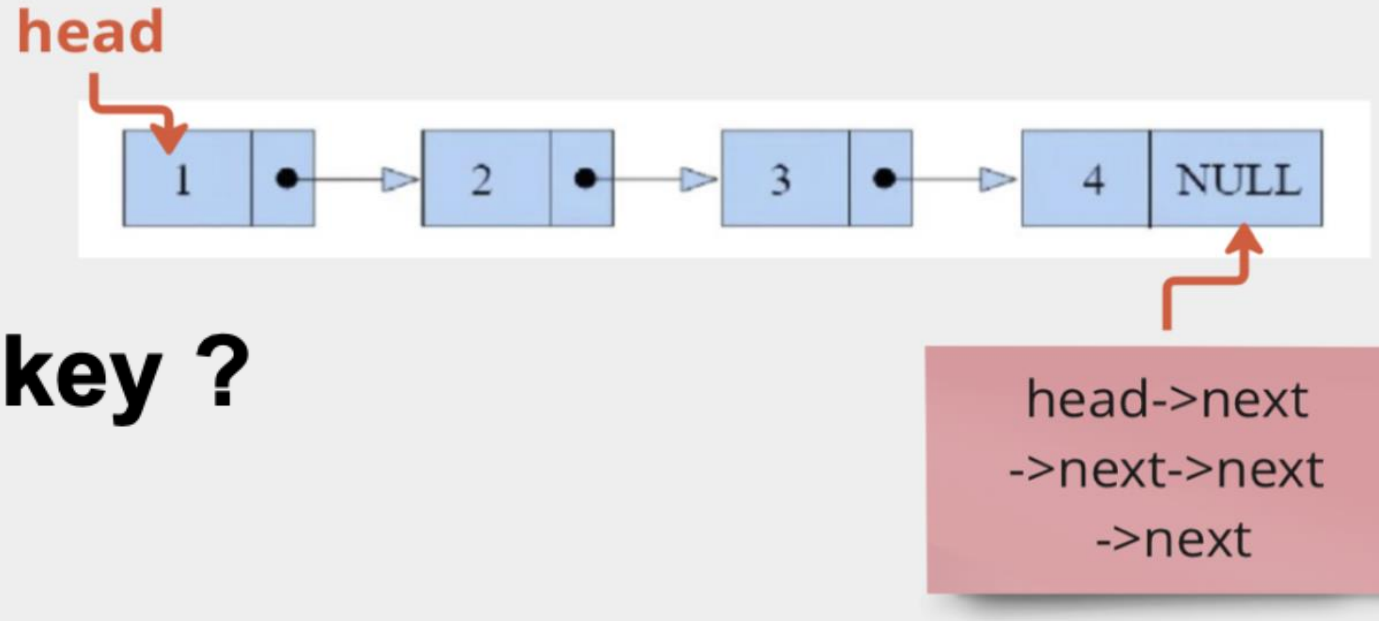
```
struct Node
{
    int  key;    // значение
    struct Node *next;      //  адрес следующей ячейки памяти  - указатель на следующий элемент
};

Struct  Node * Head;
```



Head.next.next?

Односвязный список

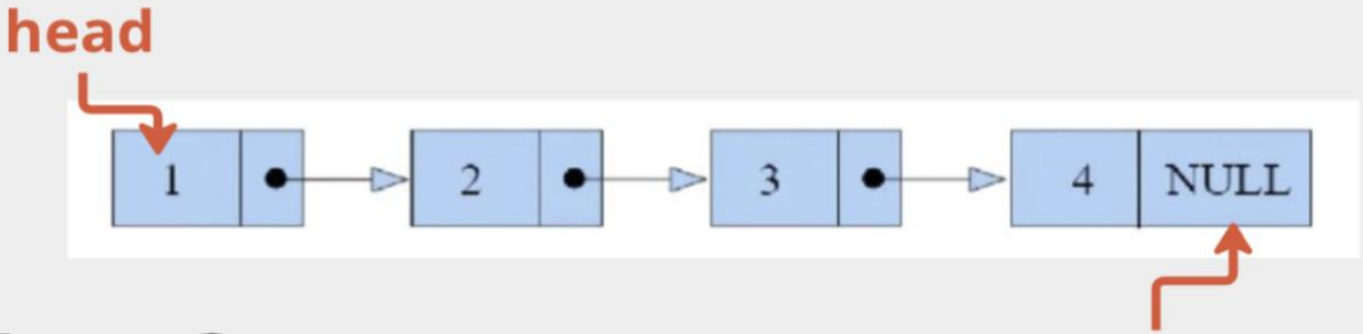
```
struct Node
{
    int  key;    // значение
    struct Node *next;      //  адрес следующей ячейки памяти  - указатель на следующий элемент
};

Struct  Node * Head;
```



Head.next.next.key ?

Односвязный список

```
struct Node
{
    int  key;    // значение
    struct Node *next;      //  адрес следующей ячейки памяти  - указатель на следующий элемент
};

Struct  Node * Head;
```



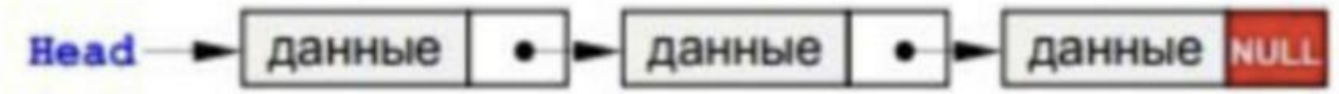
Head.next.next.key ? =3

head->next
->next->next
->next

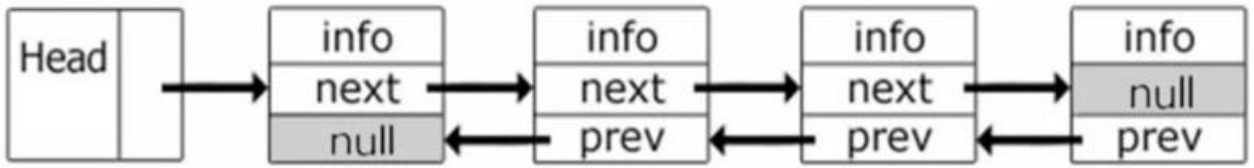
```
struct Node
{
    int  key;   // значение
    struct Node *next;     //  адрес следующей ячейки памяти  - указатель на следующий элемент
    struct Node *prev;     //  адрес следующей предыдущей ячейки памяти  - указатель на предыдущий элемент
};

Struct  Node * Head, *Tail;
```

Целая структура / объект / 50 полей!

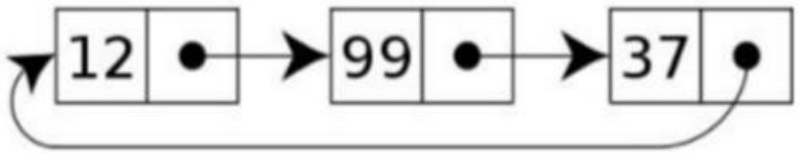
- Односвязный



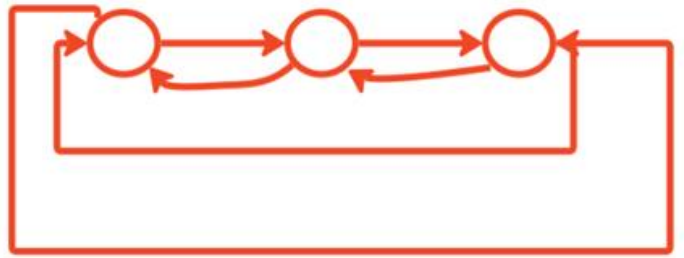
- Двусвязный



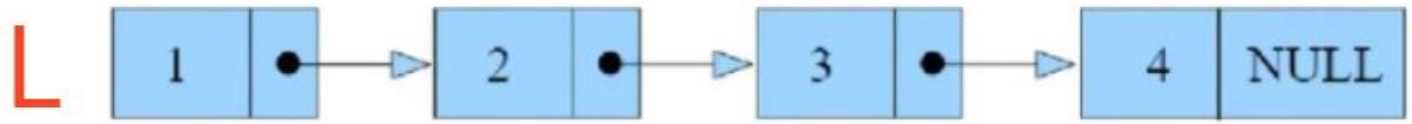
- Циклический



Двусвязный + цикличный

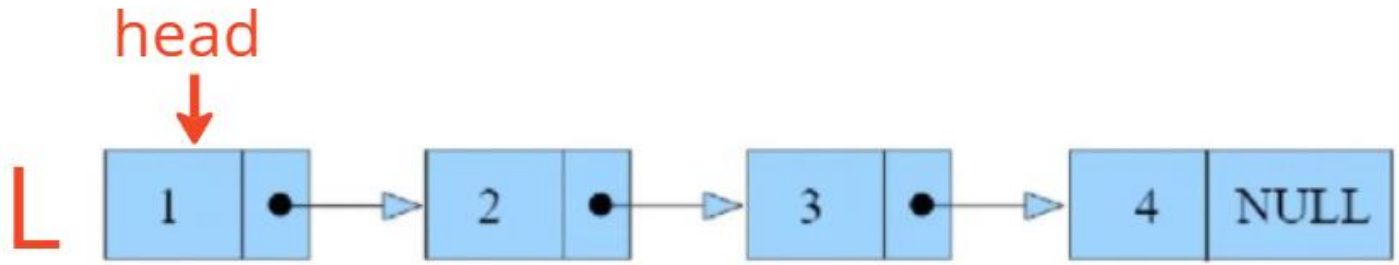


ПОИСК



ПОИСК

List-Search(L, 3)



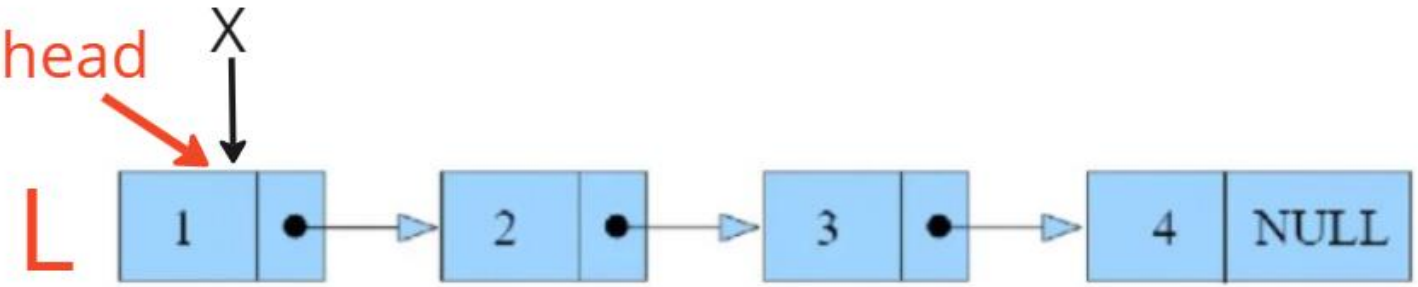
LIST-SEARCH(L, k)

1 $x = L.head$
2 **while** $x \neq$ NIL и $x.key \neq k$
3 $\quad x = x.next$
4 **return** x

ПОИСК

List-Search(L, 3)

```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL и x.key ≠ k
3        x = x.next
4  return x
```

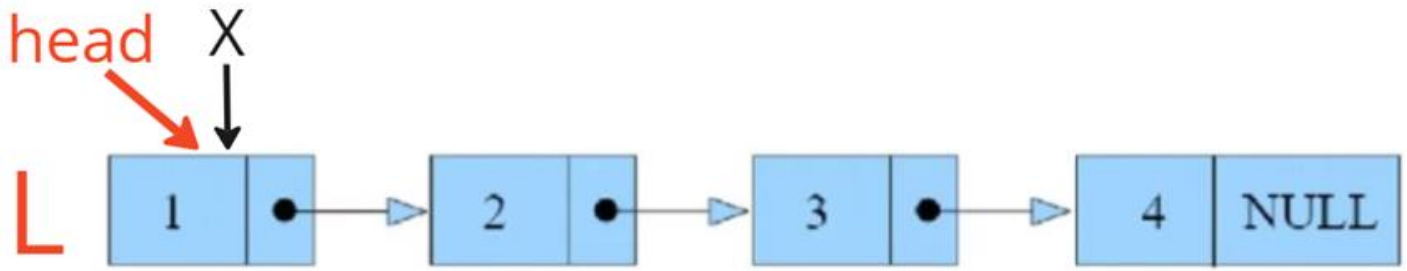


Скопируем указатель на голову, чтобы её не потерять

x = L.head

ПОИСК

List-Search(L, 3)




```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL и x.key ≠ k
3        x = x.next
4  return x
```

Скопируем указатель на голову, чтобы её не потерять

x = L.head

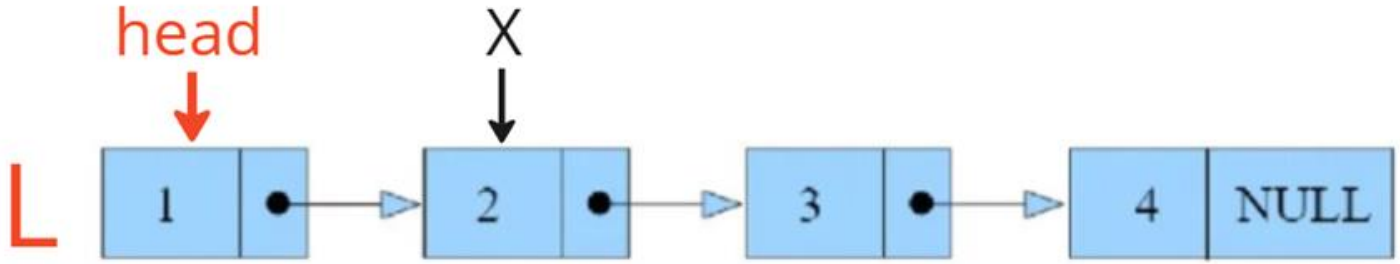
Будем идти по списку пока не встретим Null или искомый элемент

x == NULL
or
x.key == k

ПОИСК

List-Search(L, 3)

```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL и x.key ≠ k
3        x = x.next
4  return x
```



Скопируем указатель на голову, чтобы её не потерять

x = L.head

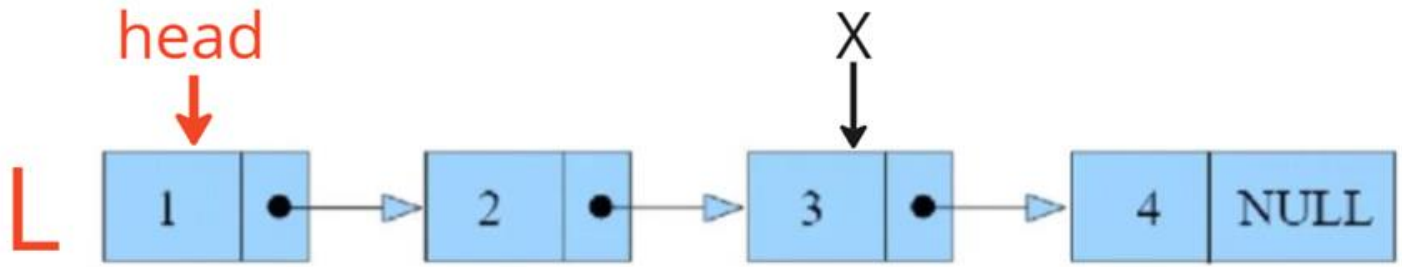
Будем идти по списку пока не встретим Null или искомый элемент

x == NULL
or
x.key == k

ПОИСК

List-Search(L, 3)

```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL и x.key ≠ k
3        x = x.next
4  return x
```



Скопируем указатель на голову, чтобы её не потерять

x = L.head

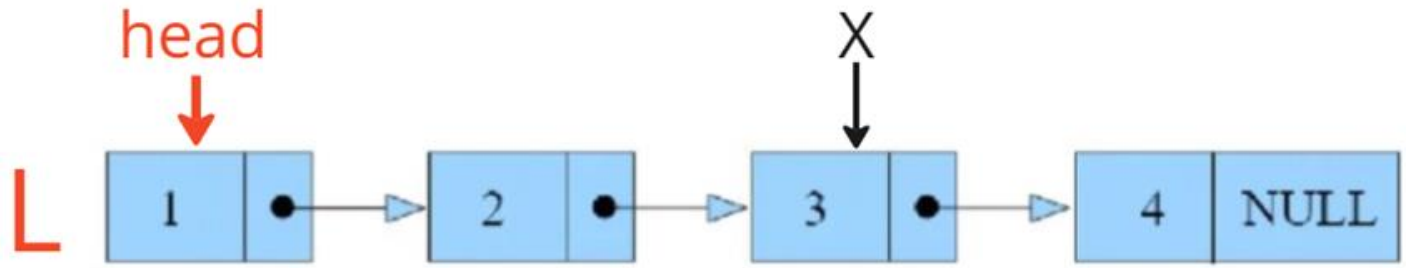
Будем идти по списку пока не встретим Null или искомый элемент

x == NULL
or
x.key == k

ПОИСК

List-Search(L, 3)

```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL и x.key ≠ k
3        x = x.next
4  return x
```



Скопируем указатель на голову, чтобы её не потерять

x = L.head

Будем идти по списку пока не встретим Null или искомый элемент

x == NULL
or
x.key == k

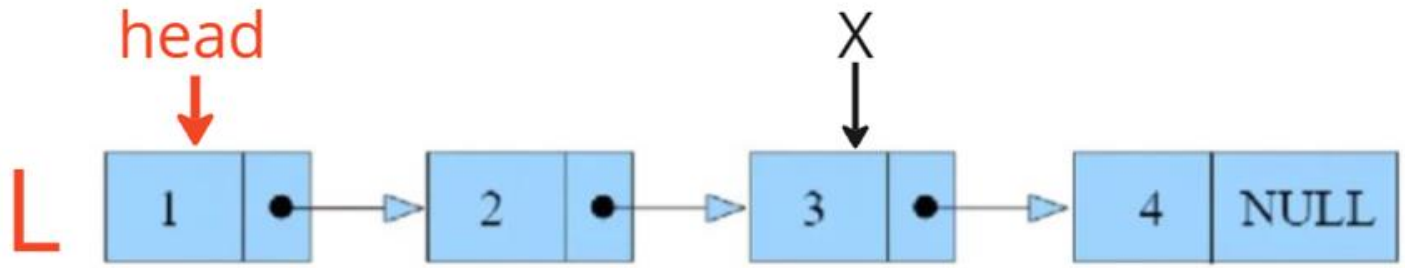
Нашли!

ПОИСК

List-Search(L, 3)

```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL и x.key ≠ k
3        x = x.next
4  return x
```

Список длины n => search(head, key) ~O(n) т.к. весь список нужно пройти



Скопируем указатель на голову, чтобы её не потерять

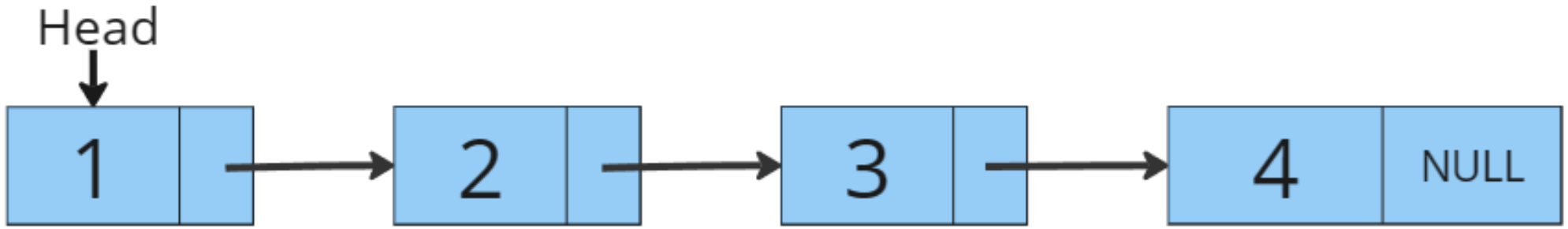
x = L.head

Будем идти по списку пока не встретим Null или искомый элемент

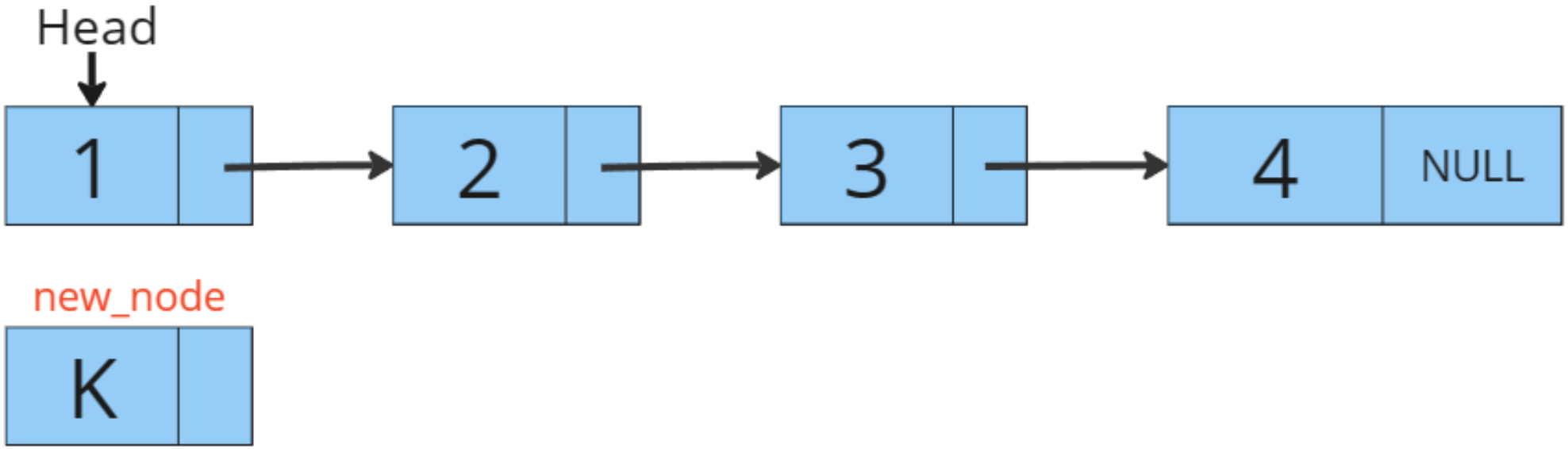
x == NULL
or
x.key == k

Нашли!

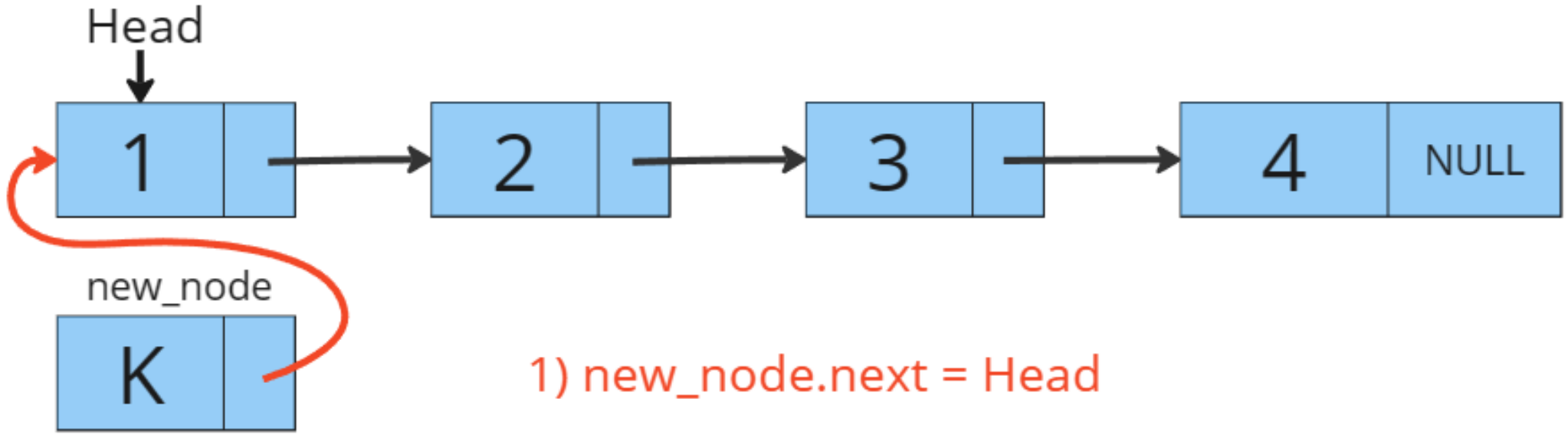
Добавить новый элемент в список



Добавить новый элемент в список

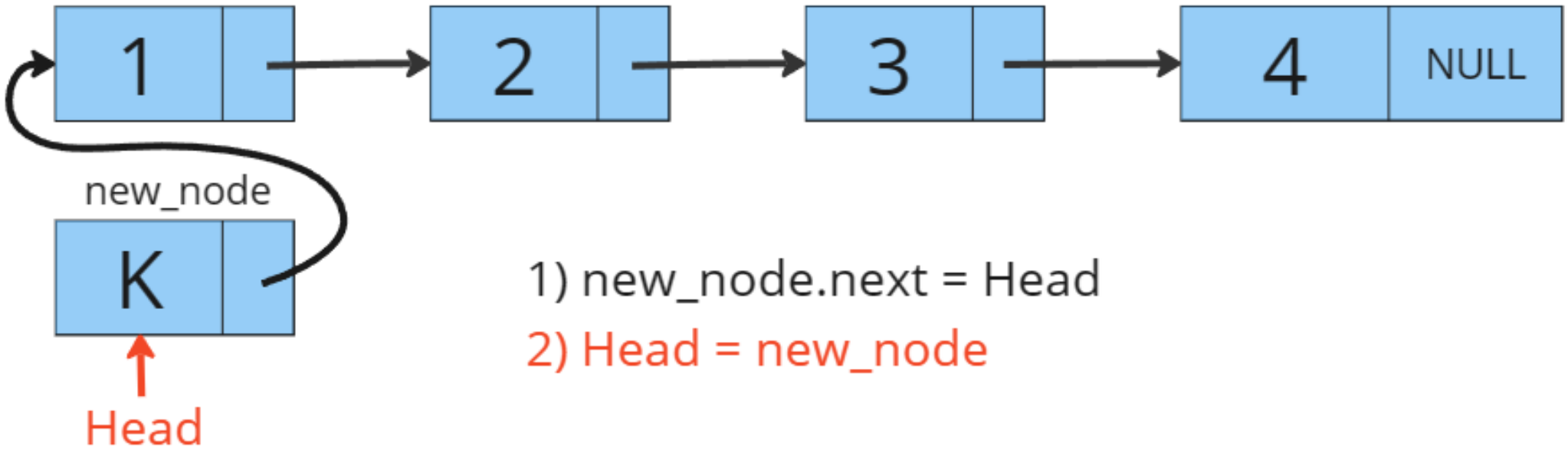


Добавить новый элемент в список



1) new_node.next = Head

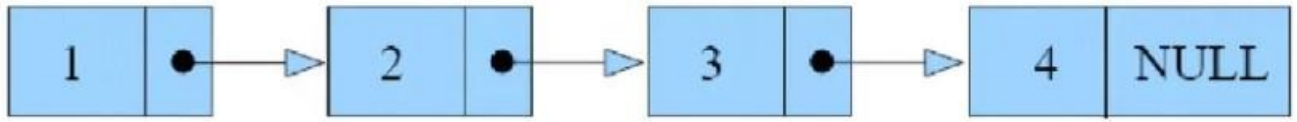
Добавить новый элемент в список



1) new_node.next = Head

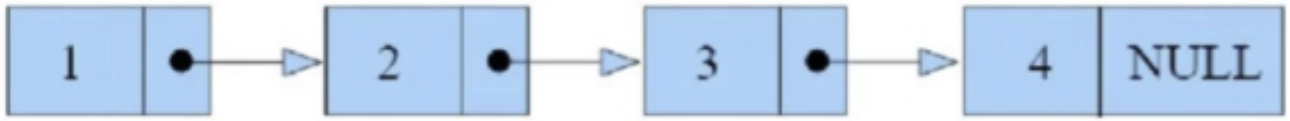
2) Head = new_node

Добавить новый элемент в заданное место в списке



Добавить новый элемент в заданное место в списке Head

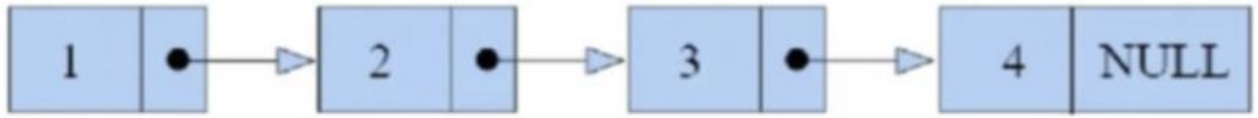
p = head



Добавить новый элемент в заданное место в списке

Head

p = head

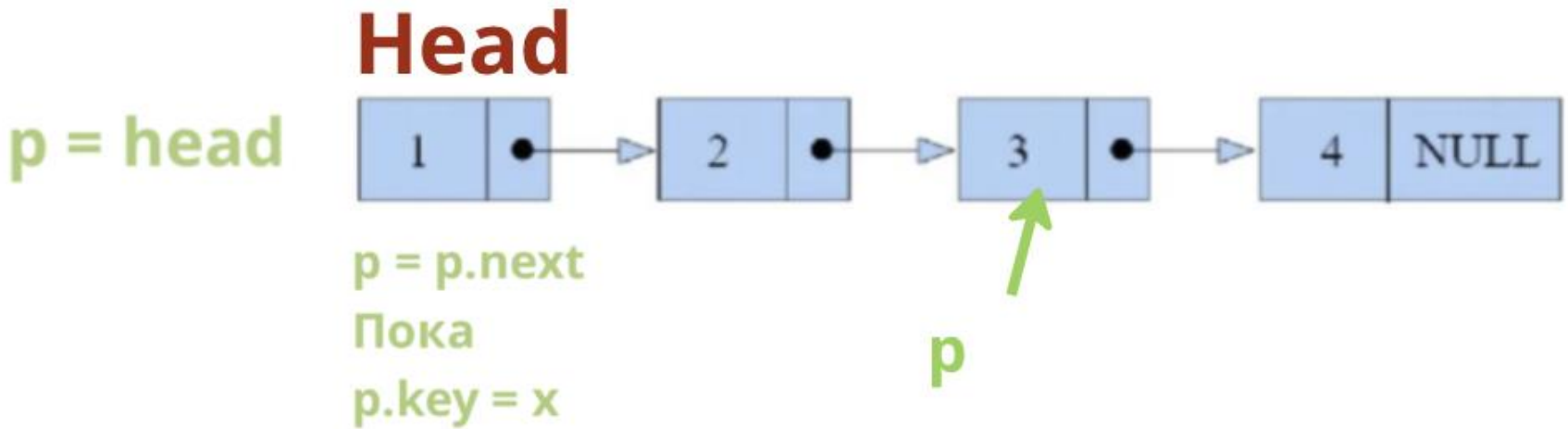


p = p.next

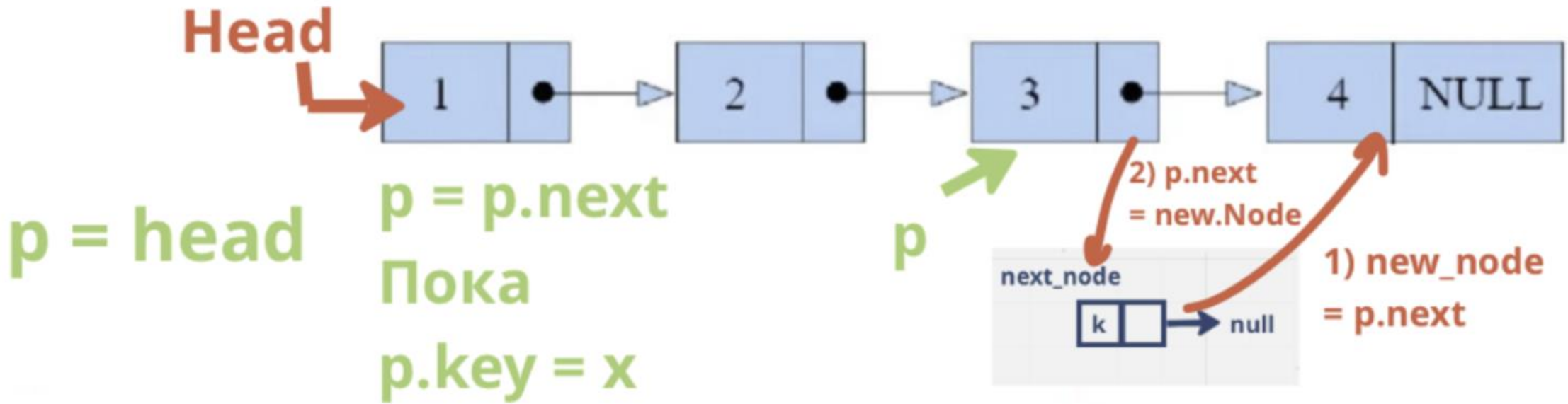
Пока

p.key = x

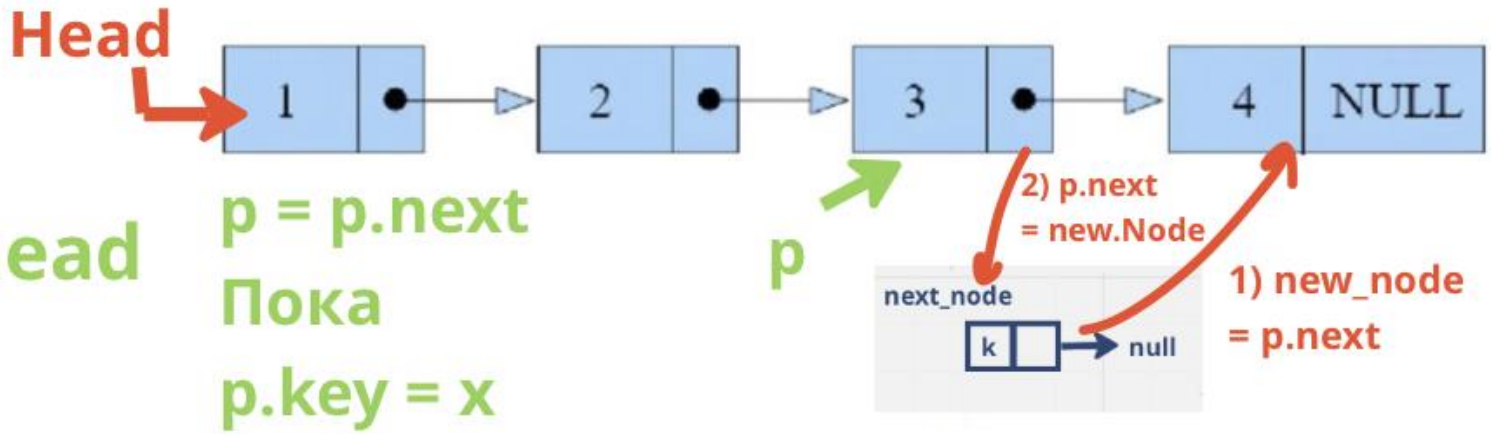
Добавить новый элемент в заданное место в списке



Добавить новый элемент в заданное место в списке

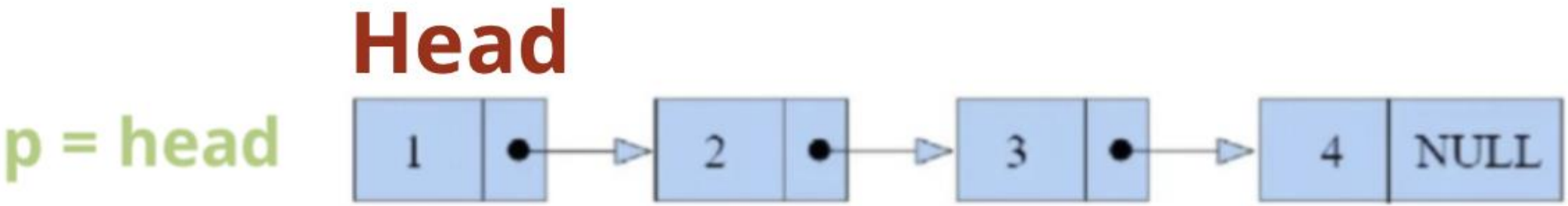


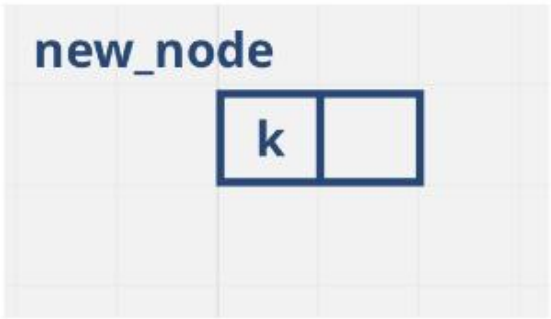
Добавить новый элемент в заданное место в списке



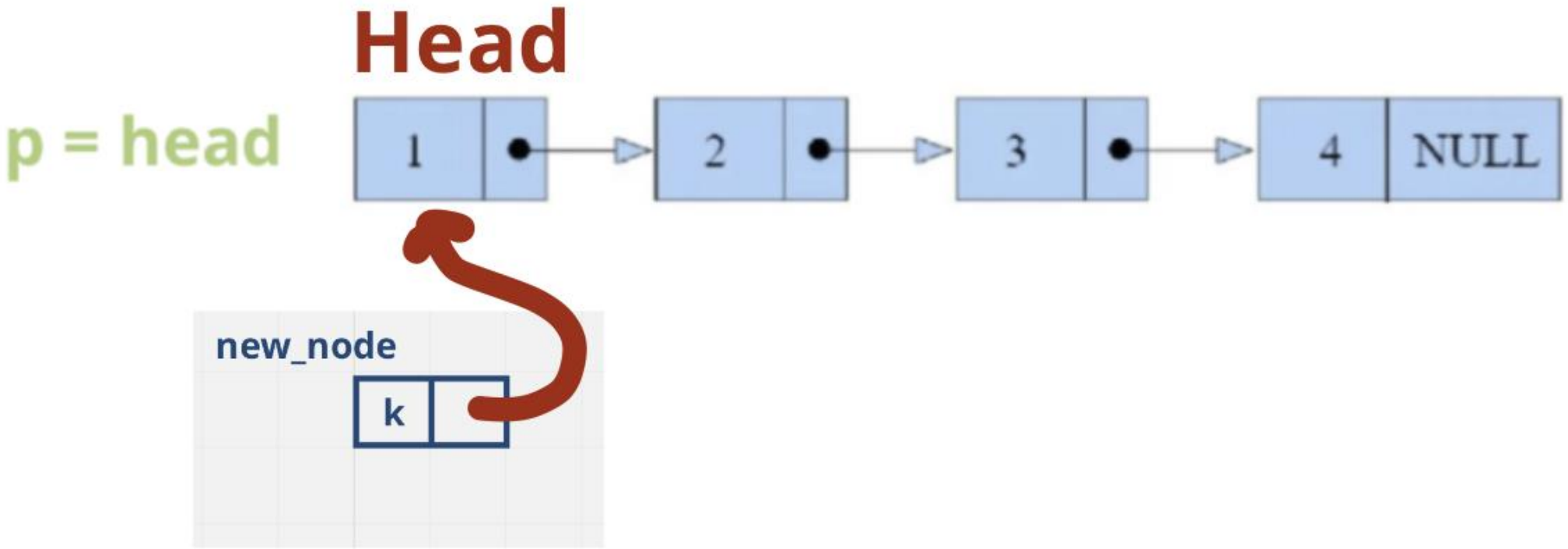
В конец
Пока p.next != null
=> p.next = New_node

Добавление в голову

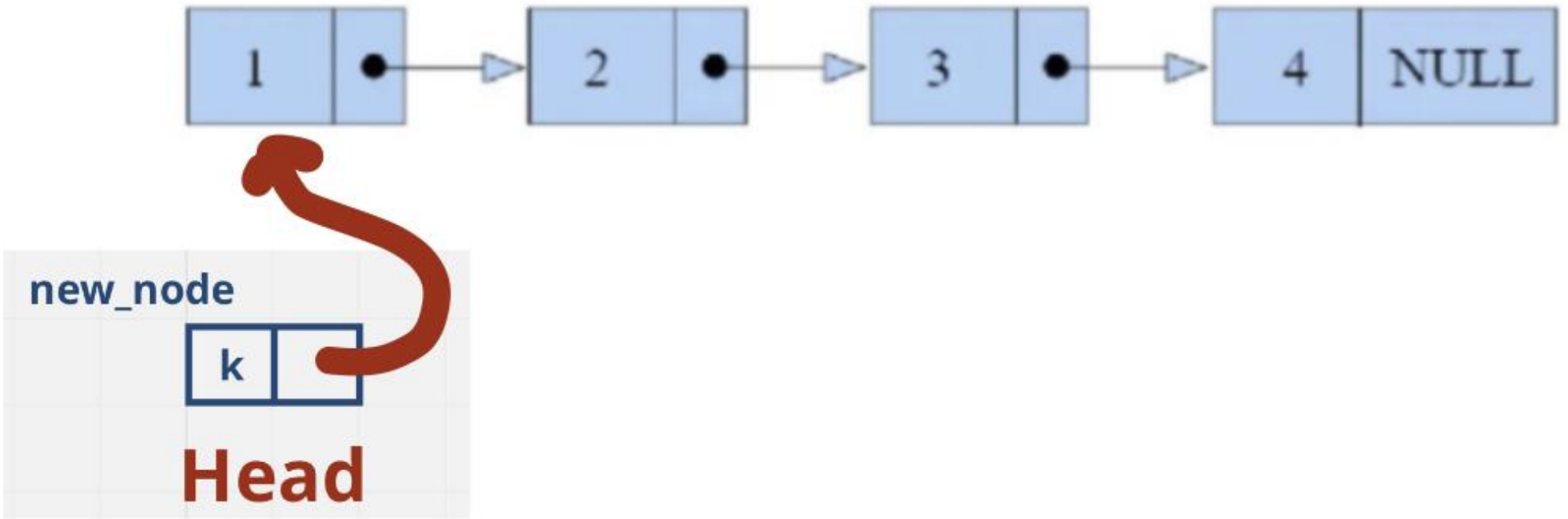




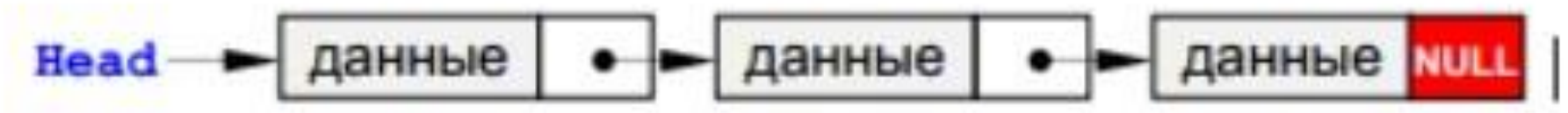
Добавление в голову



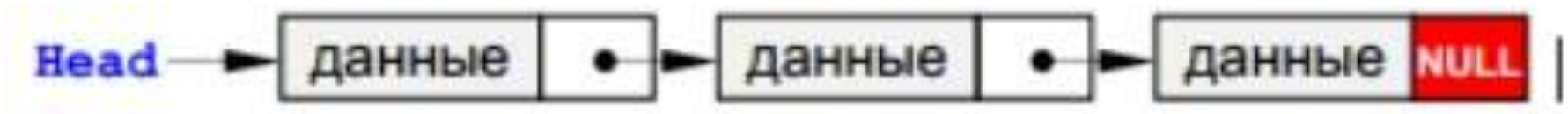
Добавление в голову



Удаление элемента из списка

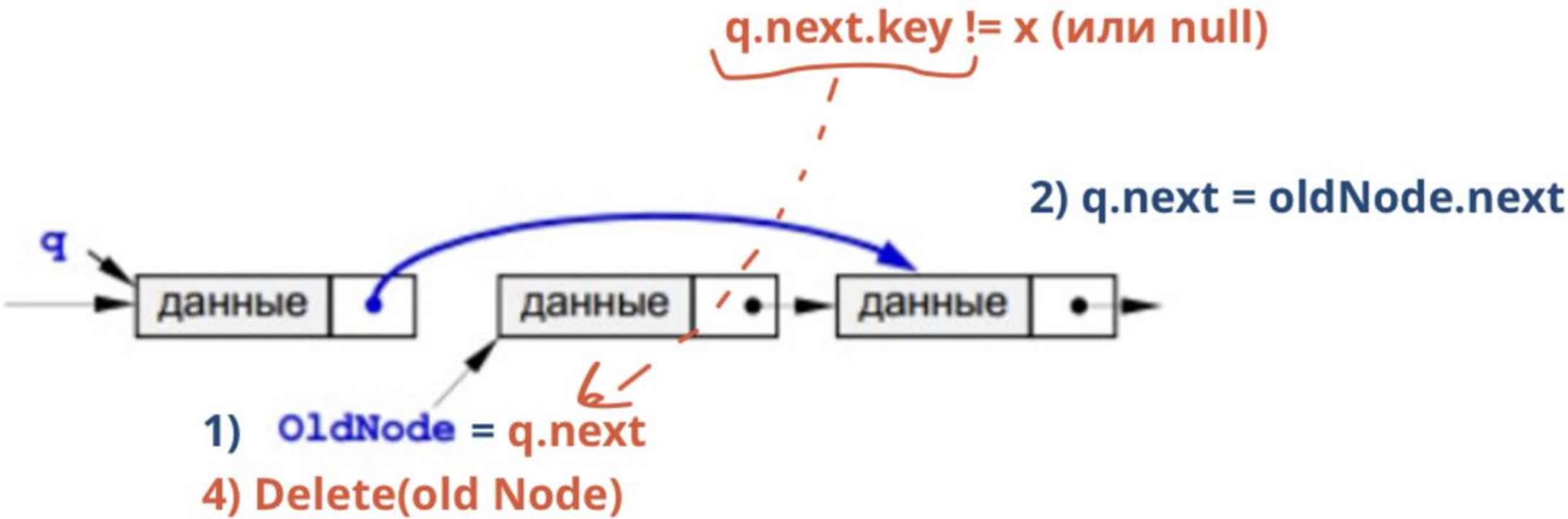


Удаление элемента из списка



Удаление элемента из списка

Удаление элемента из списка



Удаление элемента из списка

3) oldNode.next = null

а если удаляем голову?
а если удаляем последний элемент?

Удаление элемента из списка

Пока

q.next.key != x (или null)

2) q.next = oldNode.next

1) OldNode = q.next

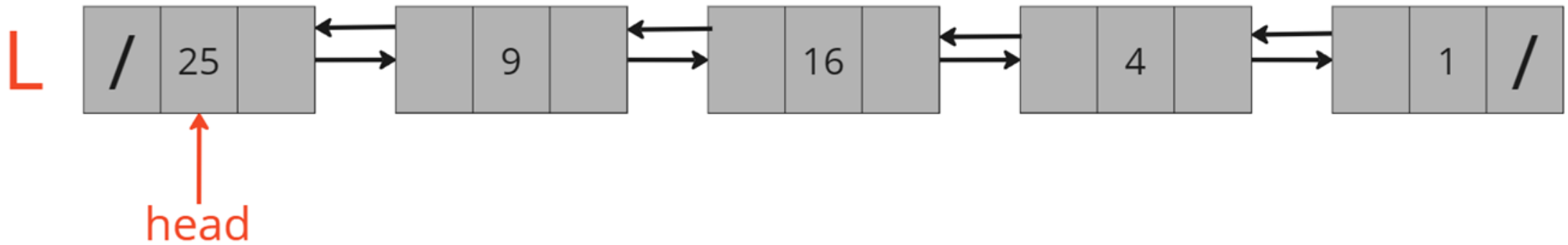
4) Delete(old Node)

3) oldNode.next = null

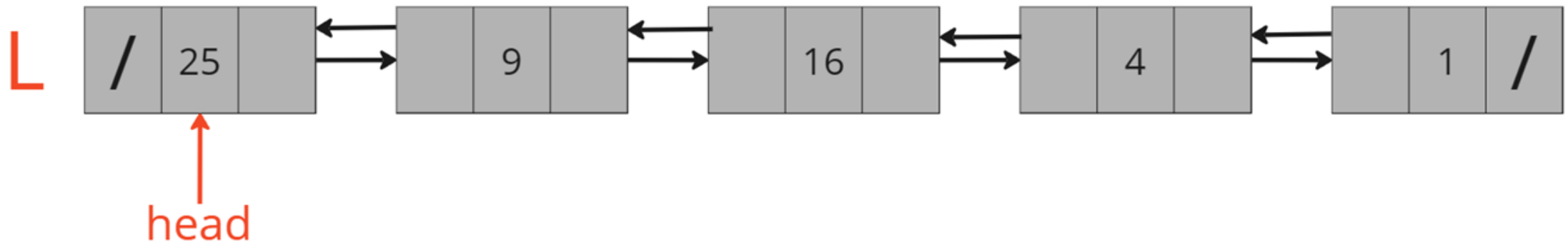
а если удаляем голову?

а если удаляем последний элемент?

ДВУСВЯЗНЫЙ СПИСОК: ДОБАВЛЕНИЕ

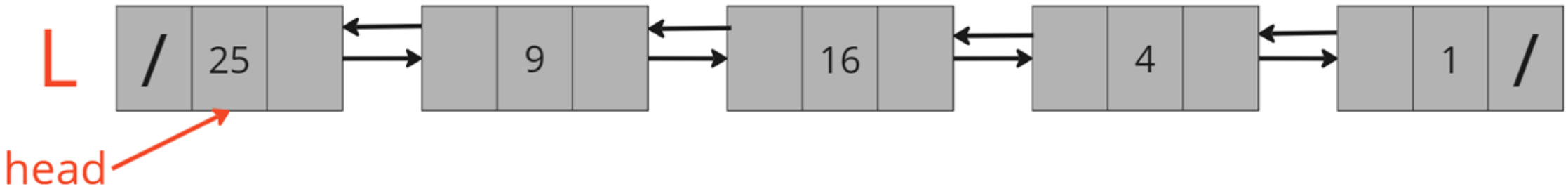


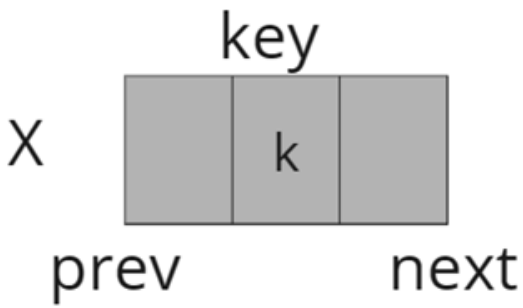
ДВУСВЯЗНЫЙ СПИСОК: ДОБАВЛЕНИЕ




```
List-Insert(L,x)
1    x.next = L.head
2    if L.head != NULL
3        L.head.prev = x
4    L.head = x
5    x.prev = NULL
```

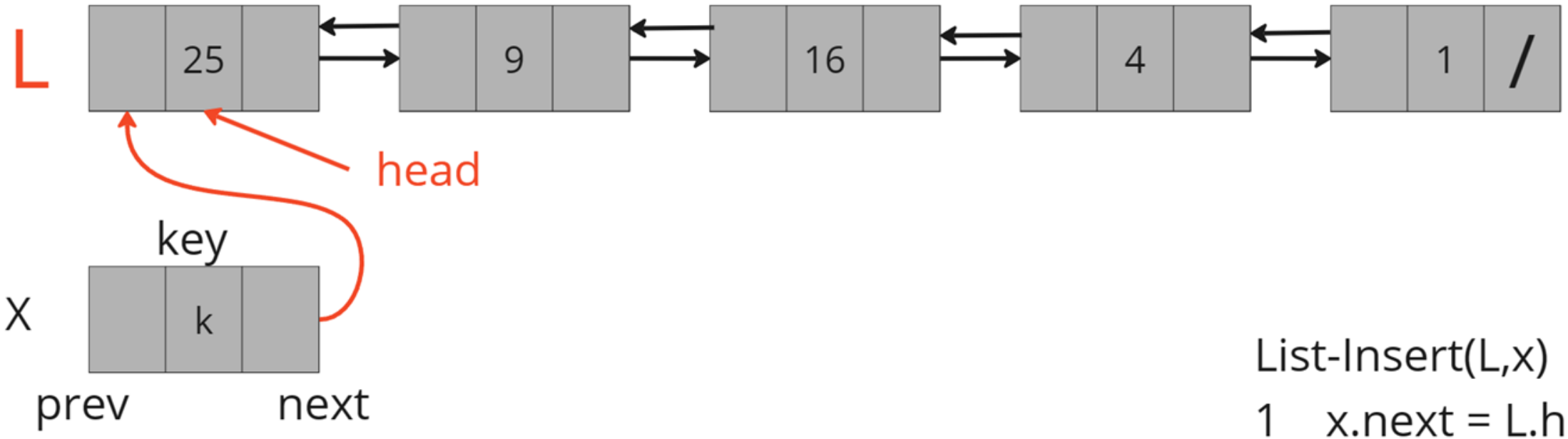
ДВУСВЯЗНЫЙ СПИСОК: ДОБАВЛЕНИЕ





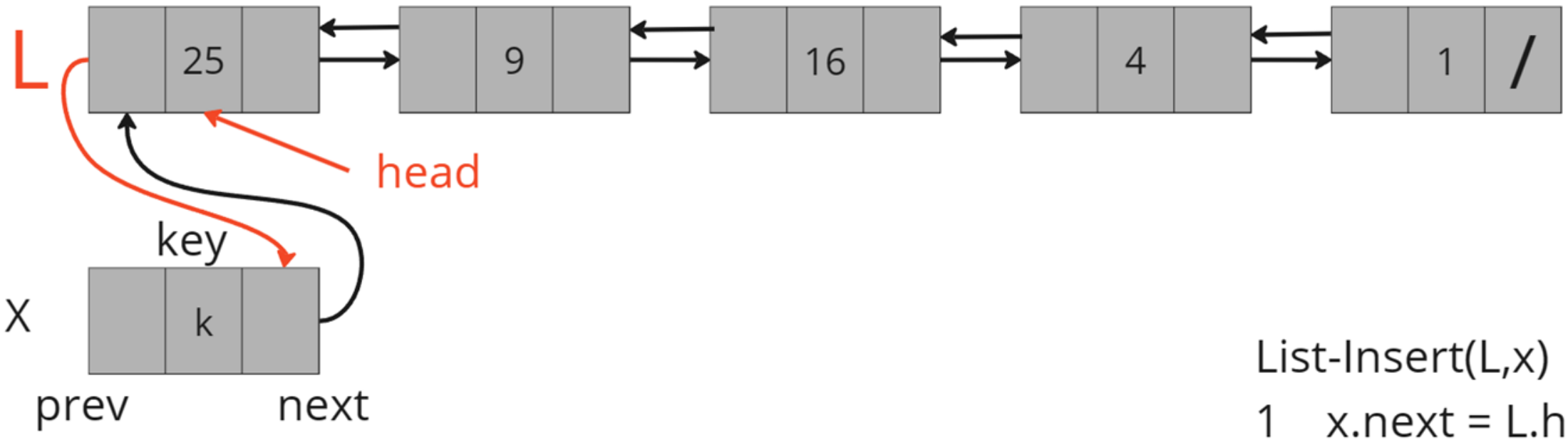

```
List-Insert(L,x)
1    x.next = L.head
2    if L.head != NULL
3        L.head.prev = x
4    L.head = x
5    x.prev = NULL
```

ДВУСВЯЗНЫЙ СПИСОК: ДОБАВЛЕНИЕ



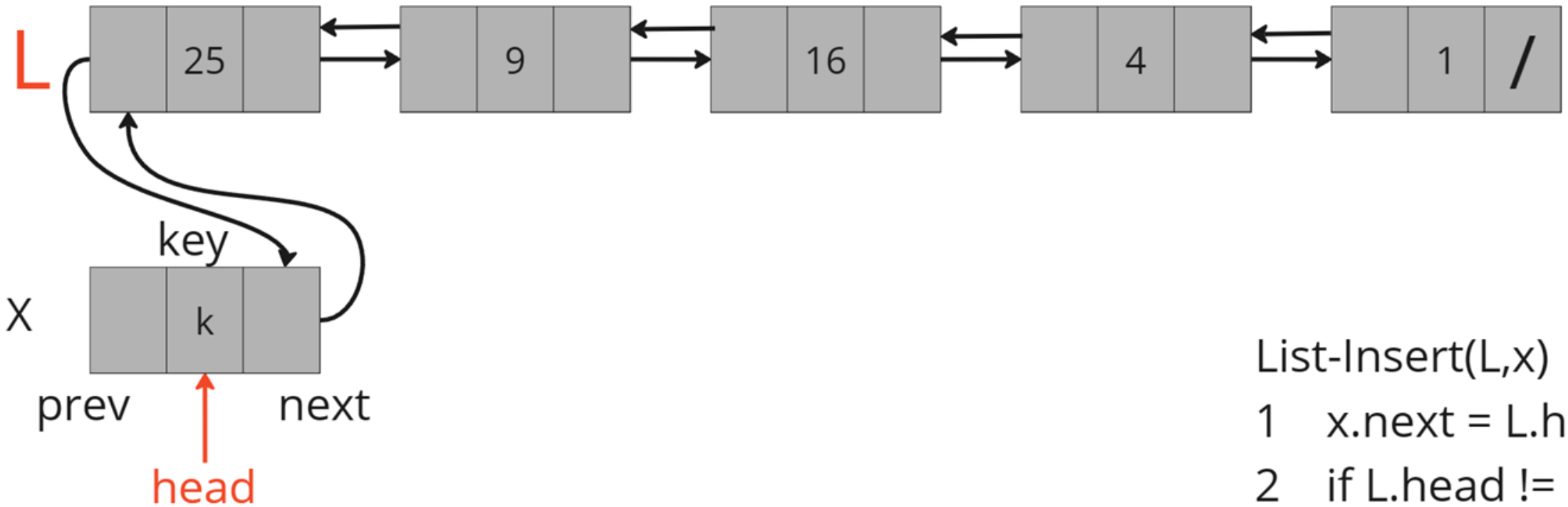

```
List-Insert(L,x)
1    x.next = L.head
2    if L.head != NULL
3        L.head.prev = x
4    L.head = x
5    x.prev = NULL
```

ДВУСВЯЗНЫЙ СПИСОК: ДОБАВЛЕНИЕ



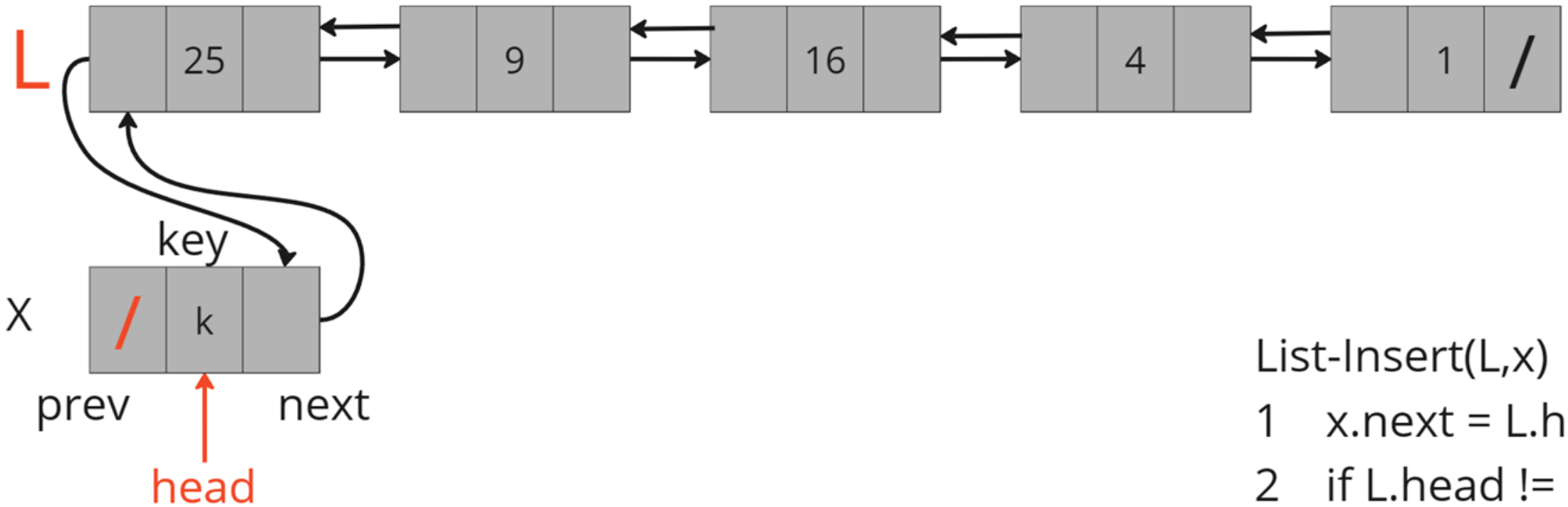

```
List-Insert(L,x)
1    x.next = L.head
2    if L.head != NULL
3        L.head.prev = x
4    L.head = x
5    x.prev = NULL
```

ДВУСВЯЗНЫЙ СПИСОК: ДОБАВЛЕНИЕ



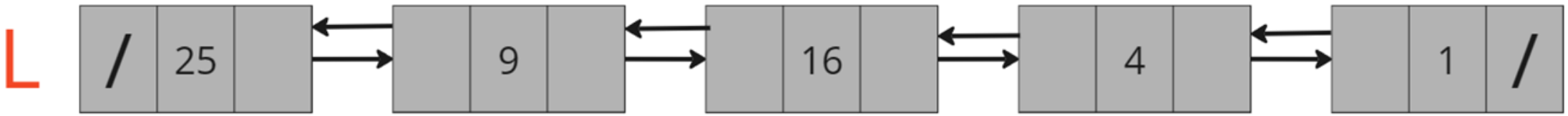

```
List-Insert(L,x)
1    x.next = L.head
2    if L.head != NULL
3        L.head.prev = x
4    L.head = x
5    x.prev = NULL
```

ДВУСВЯЗНЫЙ СПИСОК: ДОБАВЛЕНИЕ



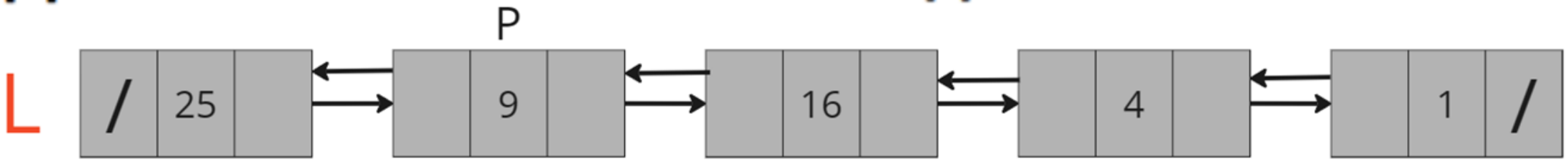

```
List-Insert(L,x)
1    x.next = L.head
2    if L.head != NULL
3        L.head.prev = x
4    L.head = x
5    x.prev = NULL
```

ДВУСВЯЗНЫЙ СПИСОК: в заданное место

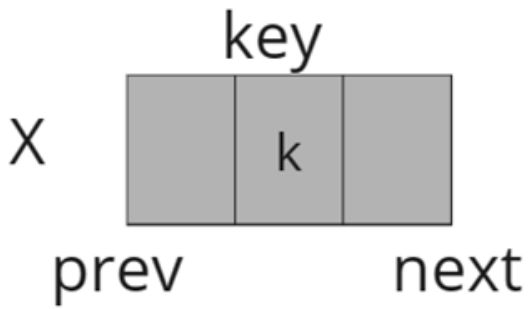



```
List-Insert(L, p, x)
1    if p.next == NULL
2        x.next = NULL
3        p.next = x
4        x.prev = p
5        return
6    if p == L.head
7        x.prev = NULL
8        L.head.prev = x
9        x.next = L.head
10       L.head = x
11       return
12   p.next.prev = x
13   x.next = p.next
14   p.next = x
15   x.prev = p
```

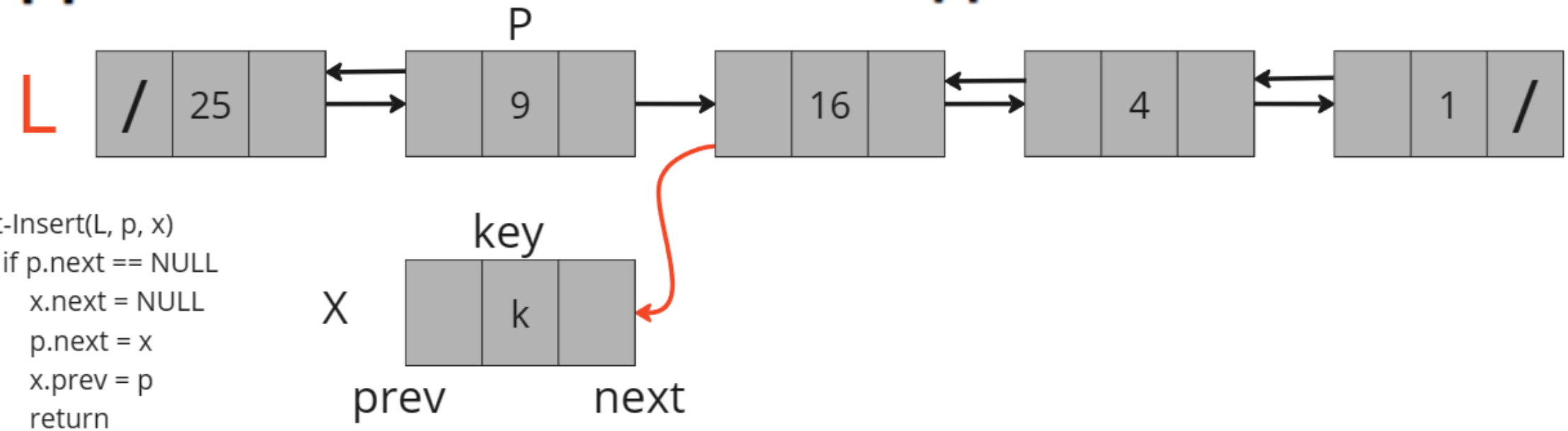
ДВУСВЯЗНЫЙ СПИСОК: в заданное место




```
List-Insert(L, p, x)
1    if p.next == NULL
2        x.next = NULL
3        p.next = x
4        x.prev = p
5        return
6    if p == L.head
7        x.prev = NULL
8        L.head.prev = x
9        x.next = L.head
10       L.head = x
11       return
12  p.next.prev = x
13  x.next = p.next
14  p.next = x
15  x.prev = p
```

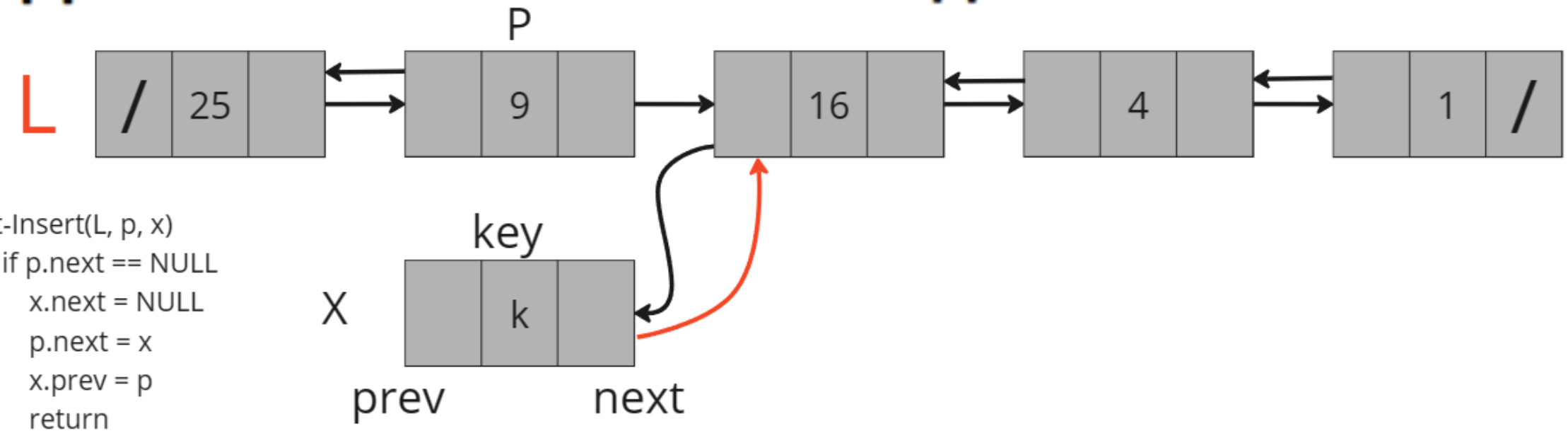


ДВУСВЯЗНЫЙ СПИСОК: в заданное место



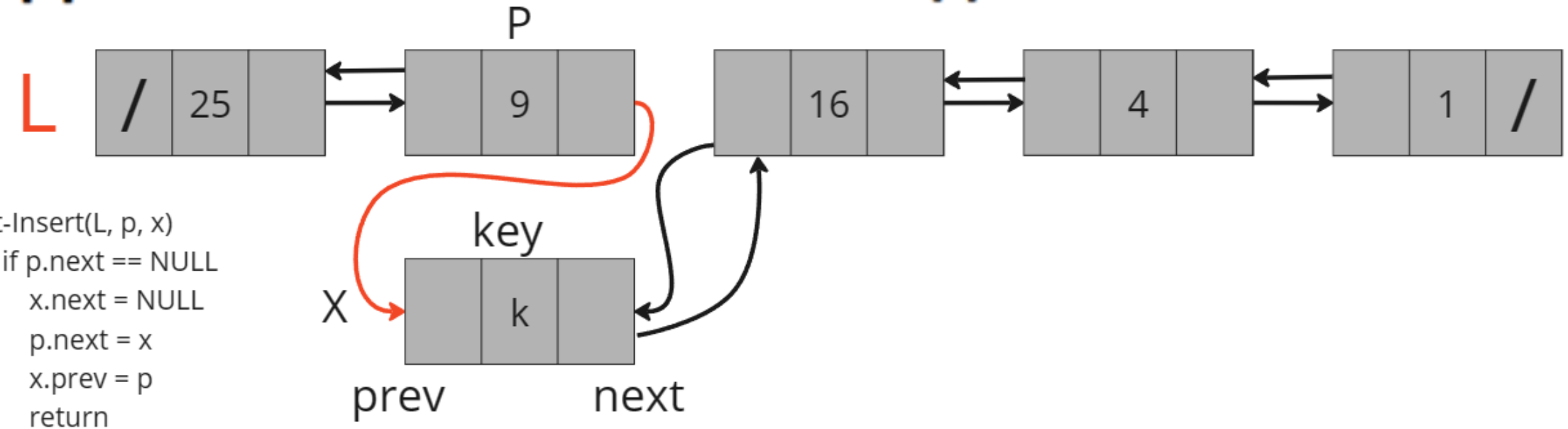

```
List-Insert(L, p, x)
1   if p.next == NULL
2       x.next = NULL
3       p.next = x
4       x.prev = p
5       return
6   if p == L.head
7       x.prev = NULL
8       L.head.prev = x
9       x.next = L.head
10      L.head = x
11      return
12  p.next.prev = x
13  x.next = p.next
14  p.next = x
15  x.prev = p
```

ДВУСВЯЗНЫЙ СПИСОК: в заданное место



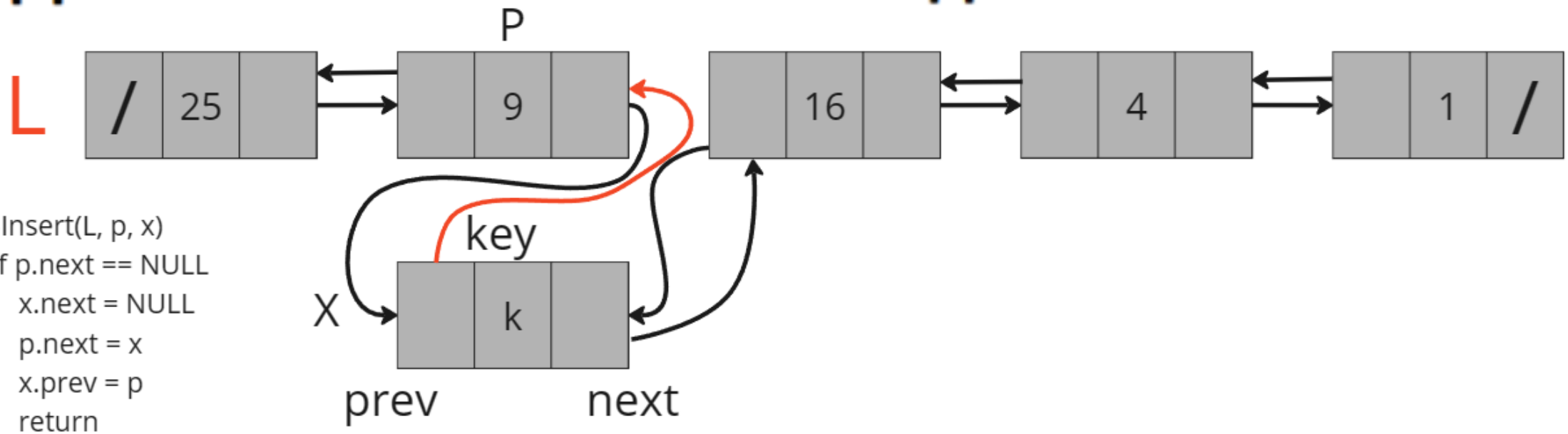

```
List-Insert(L, p, x)
1   if p.next == NULL
2       x.next = NULL
3       p.next = x
4       x.prev = p
5       return
6   if p == L.head
7       x.prev = NULL
8       L.head.prev = x
9       x.next = L.head
10      L.head = x
11      return
12  p.next.prev = x
13  x.next = p.next
14  p.next = x
15  x.prev = p
```

ДВУСВЯЗНЫЙ СПИСОК: в заданное место



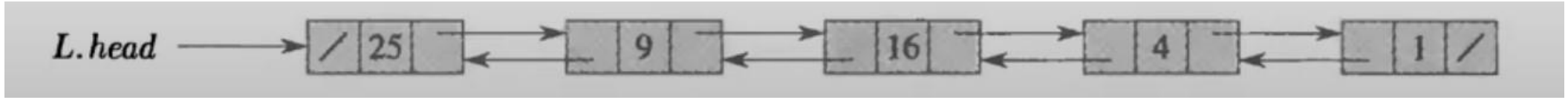

```
List-Insert(L, p, x)
1   if p.next == NULL
2       x.next = NULL
3       p.next = x
4       x.prev = p
5       return
6   if p == L.head
7       x.prev = NULL
8       L.head.prev = x
9       x.next = L.head
10      L.head = x
11      return
12  p.next.prev = x
13  x.next = p.next
14  p.next = x
15  x.prev = p
```

ДВУСВЯЗНЫЙ СПИСОК: в заданное место




```
List-Insert(L, p, x)
1   if p.next == NULL
2       x.next = NULL
3       p.next = x
4       x.prev = p
5       return
6   if p == L.head
7       x.prev = NULL
8       L.head.prev = x
9       x.next = L.head
10      L.head = x
11      return
12  p.next.prev = x
13  x.next = p.next
14  p.next = x
15  x.prev = p
```

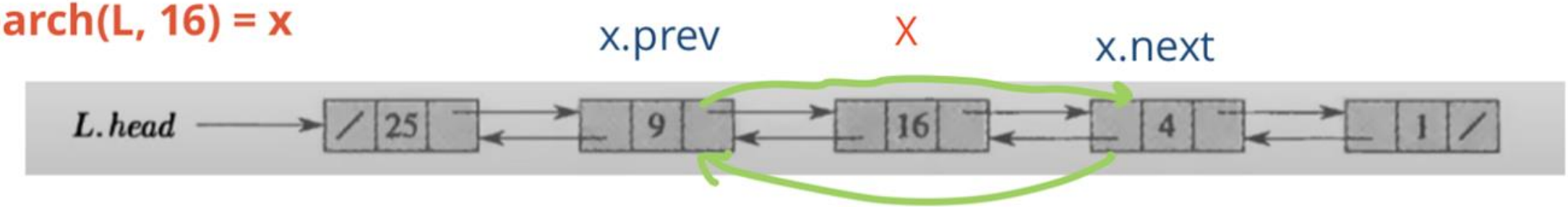
ДВУСВЯЗНЫЙ СПИСОК: УДАЛЕНИЕ



ДВУСВЯЗНЫЙ СПИСОК: УДАЛЕНИЕ

Удалим (16)

search(L, 16) = x



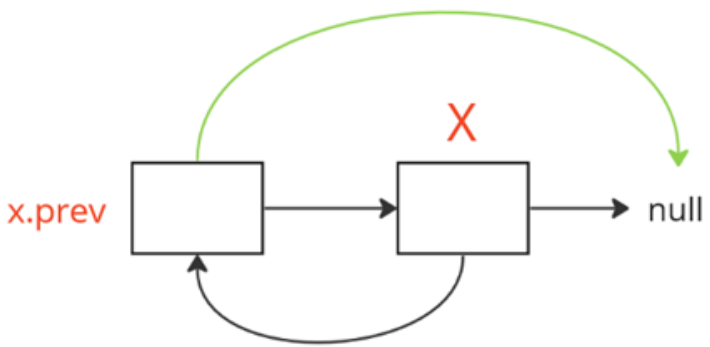
1) x.prev.next = x.next

2) x.next.prev = x.prev

ДВУСВЯЗНЫЙ СПИСОК: УДАЛЕНИЕ

а если удаляем голову? **(x.prev == null) => L.head = x.next**

а если удаляем последний элемент? **(x.next == null) => x.prev.next = null**



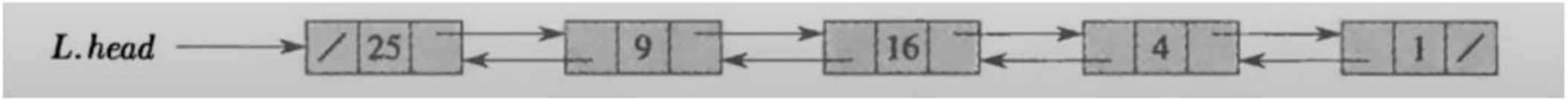
ДВУСВЯЗНЫЙ СПИСОК:

```
LIST-DELETE(L, x)
1  if x.prev ≠ NIL
2      x.prev.next = x.next
3  else L.head = x.next
4  if x.next ≠ NIL
5      x.next.prev = x.prev
```

Search(L, K) ~ O(n)

удаляем голову

O(n) + O(1) => ~O(n)



ДВУСВЯЗНЫЙ СПИСОК:

```
LIST-DELETE(L, x)
1  if x.prev ≠ NIL
2      x.prev.next = x.next
3  else L.head = x.next
4  if x.next ≠ NIL
5      x.next.prev = x.prev
```

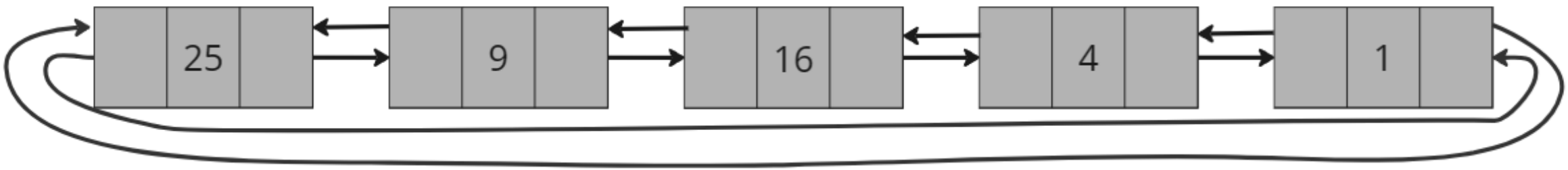
Search(L, K) ~ O(n)

удаляем голову

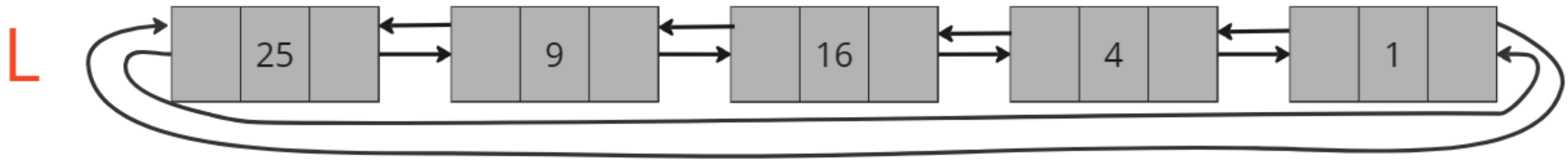
O(n) + O(1) => ~O(n)

Циклический список: примеры

L

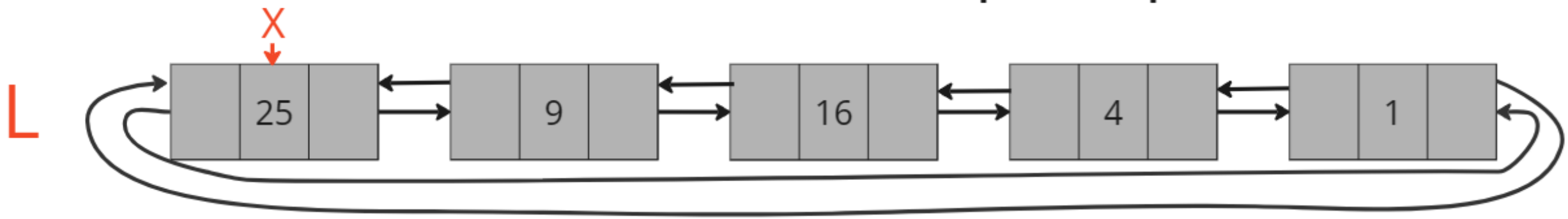


Циклический список: примеры



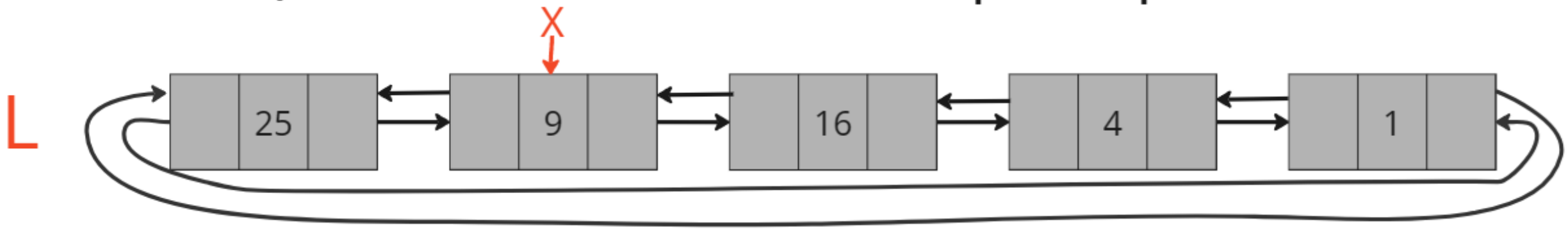
delete(L, search(L, 9))

Циклический список: примеры



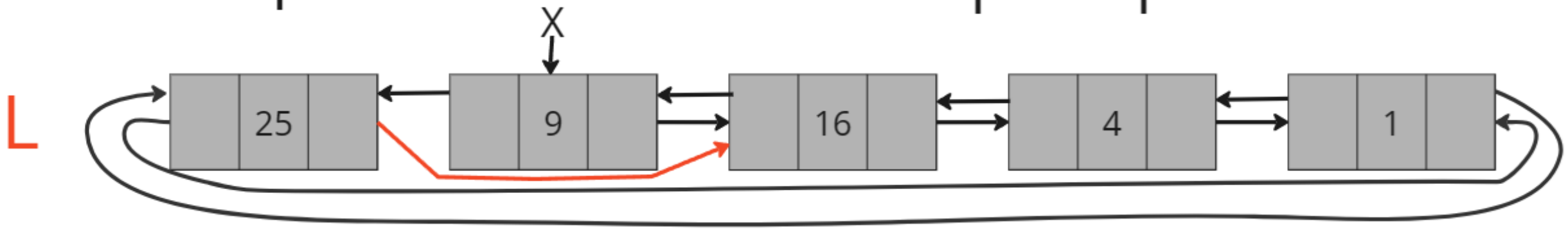
delete(L, search(L, 9))

Циклический список: примеры



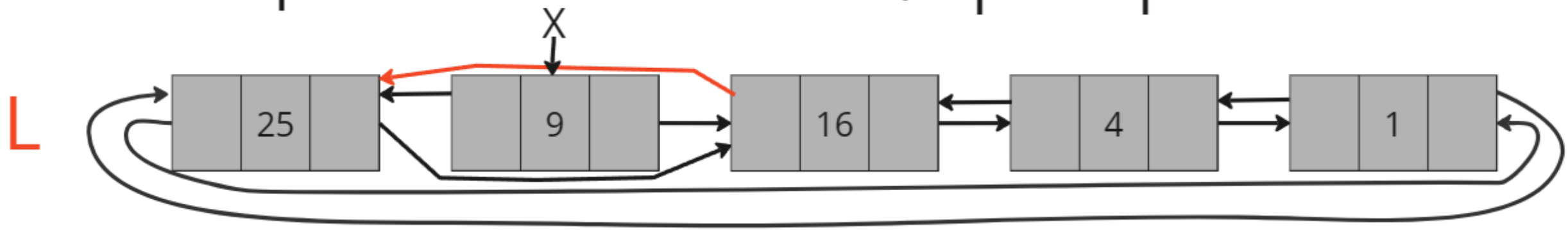
delete(L, search(L, 9))

Циклический список: примеры



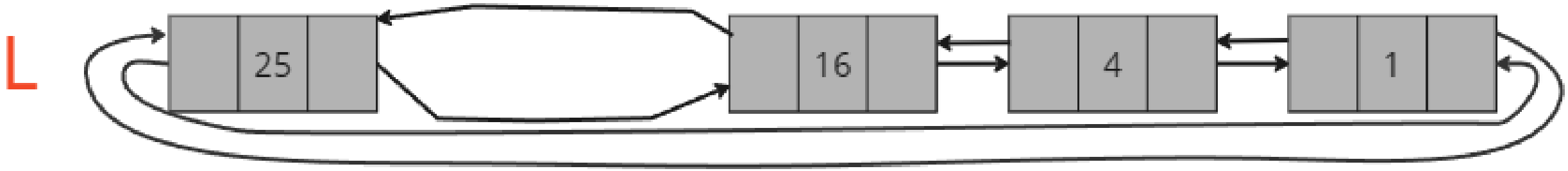
delete(L, search(L, 9))

Циклический список: примеры



delete(L, search(L, 9))

Циклический список: примеры



delete(L, search(L, 9))