при поддержке
{а}вижн

Лекция 20

Перевод и адаптация материалов Джона ДеНиро (John DeNero). Используется с разрешения автора.

Множества

Множества

Множества

Ещё один контейнерный тип в Python

Множества

Ещё один контейнерный тип в Python

- Множества задаются в фигурных скобках

Множества

Ещё один контейнерный тип в Python

- Множества задаются в фигурных скобках
- Повторяющиеся элементы удаляются на этапе создания

Множества

Ещё один контейнерный тип в Python

- Множества задаются в фигурных скобках
- Повторяющиеся элементы удаляются на этапе создания
- Элементы множества неупорядочены

Множества

Ещё один контейнерный тип в Python

- Множества задаются в фигурных скобках
- Повторяющиеся элементы удаляются на этапе создания
- Элементы множества неупорядочены

```
>>> s = {'один', 'два', 'три', 'четыре', 'четыре'}
```

Множества

Ещё один контейнерный тип в Python

- Множества задаются в фигурных скобках
- Повторяющиеся элементы удаляются на этапе создания
- Элементы множества неупорядочены

```
>>> s = {'один', 'два', 'три', 'четыре', 'четыре'}
>>> s
{'четыре', 'один', 'три', 'два'}
```

Множества

Ещё один контейнерный тип в Python

- Множества задаются в фигурных скобках
- Повторяющиеся элементы удаляются на этапе создания
- Элементы множества неупорядочены

```
>>> s = {'один', 'два', 'три', 'четыре', 'четыре'}
>>> s
{'четыре', 'один', 'три', 'два'}
>>> 'три' in s
True
```

Множества

Ещё один контейнерный тип в Python

- Множества задаются в фигурных скобках
- Повторяющиеся элементы удаляются на этапе создания
- Элементы множества неупорядочены

```
>>> s = {'один', 'два', 'три', 'четыре', 'четыре'}
>>> s
{'четыре', 'один', 'три', 'два'}
>>> 'три' in s
True
>>> len(s)
4
```

Множества

Ещё один контейнерный тип в Python

- Множества задаются в фигурных скобках
- Повторяющиеся элементы удаляются на этапе создания
- Элементы множества неупорядочены

```
>>> s = {'один', 'два', 'три', 'четыре', 'четыре'}
>>> s
{'четыре', 'один', 'три', 'два'}
>>> 'три' in s
True
>>> len(s)
4
>>> s.union({'один', 'пять'})
{'пять', 'три', 'четыре', 'один', 'два'}
```

Множества

Ещё один контейнерный тип в Python

- Множества задаются в фигурных скобках
- Повторяющиеся элементы удаляются на этапе создания
- Элементы множества неупорядочены

```
>>> s = {'один', 'два', 'три', 'четыре', 'четыре'}
>>> s
{'четыре', 'один', 'три', 'два'}
>>> 'три' in s
True
>>> len(s)
4
>>> s.union({'один', 'пять'})
{'пять', 'три', 'четыре', 'один', 'два'}
>>> s.intersection({'шесть', 'пять', 'четыре', 'три'})
{'четыре', 'три'}
```

Операции с множествами

Операции с множествами

Что можно делать с множествами:

Операции с множествами

Что можно делать с множествами:

- **Проверка принадлежности:** Является ли значение элементом множества?

Операции с множествами

Что можно делать с множествами:

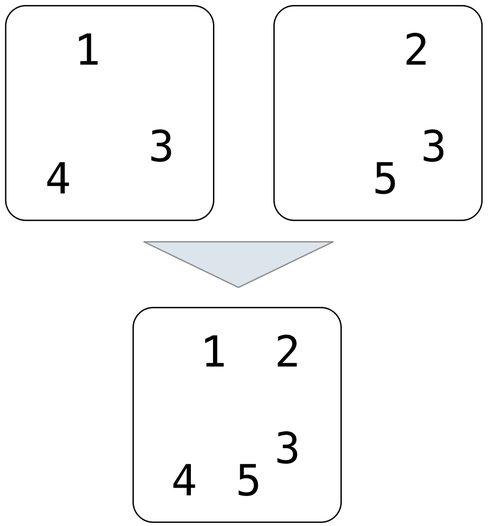
- **Проверка принадлежности:** Является ли значение элементом множества?
- **Объединение:** Возвращает множество элементов находящихся в set1 или в set2

Операции с множествами

Что можно делать с множествами:

- **Проверка принадлежности:** Является ли значение элементом множества?
- **Объединение:** Возвращает множество элементов находящихся в set1 или в set2

Объединение

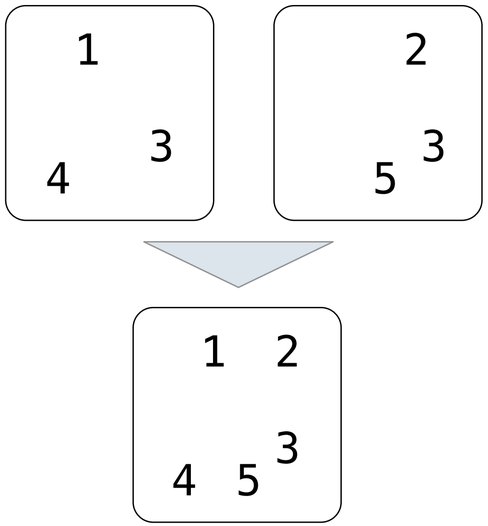


Операции с множествами

Что можно делать с множествами:

- **Проверка принадлежности:** Является ли значение элементом множества?
- **Объединение:** Возвращает множество элементов находящихся в set1 или в set2
- **Пересечение:** Возвращает множество элементов находящихся и в set1, и в set2

Объединение

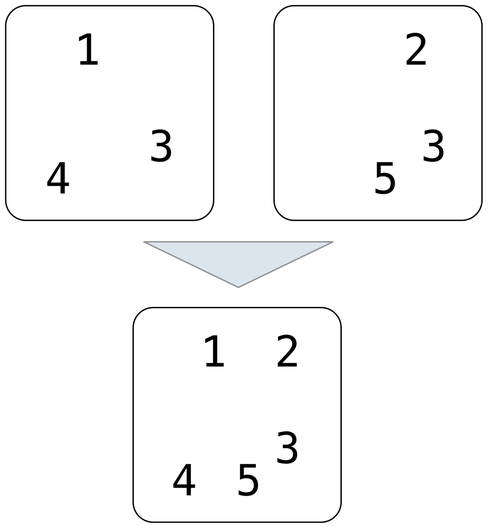


Операции с множествами

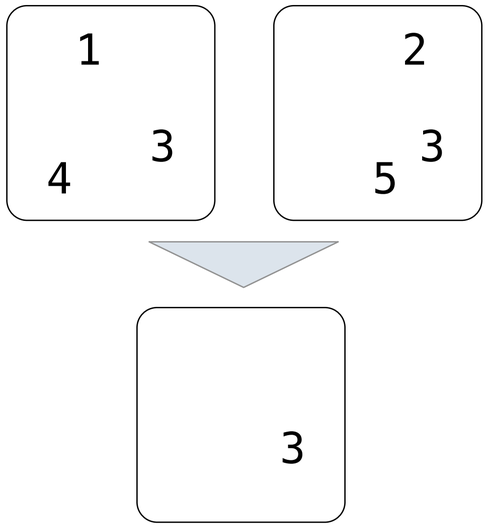
Что можно делать с множествами:

- **Проверка принадлежности:** Является ли значение элементом множества?
- **Объединение:** Возвращает множество элементов находящихся в set1 или в set2
- **Пересечение:** Возвращает множество элементов находящихся и в set1, и в set2

Объединение



Пересечение

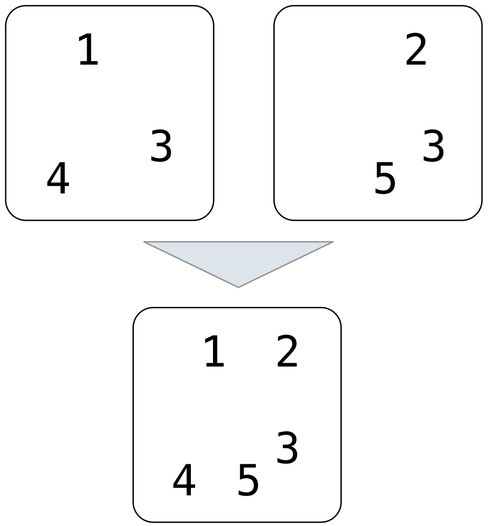


Операции с множествами

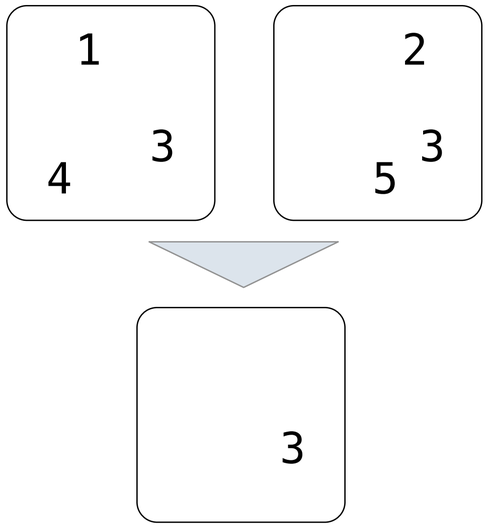
Что можно делать с множествами:

- **Проверка принадлежности:** Является ли значение элементом множества?
- **Объединение:** Возвращает множество элементов находящихся в set1 или в set2
- **Пересечение:** Возвращает множество элементов находящихся и в set1, и в set2
- **Присоединение (добавление):** Возвращает множество элементов из s и значение v

Объединение



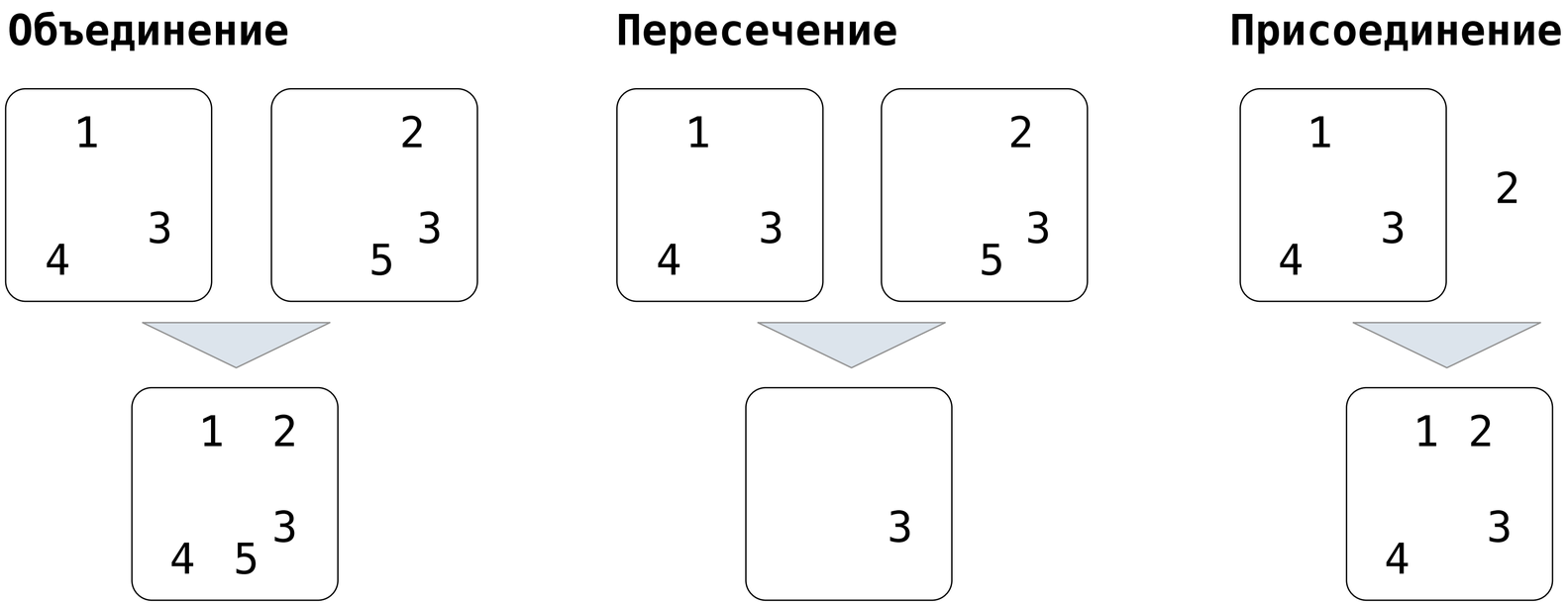
Пересечение



Операции с множествами

Что можно делать с множествами:

- **Проверка принадлежности:** Является ли значение элементом множества?
- **Объединение:** Возвращает множество элементов находящихся в set1 или в set2
- **Пересечение:** Возвращает множество элементов находящихся и в set1, и в set2
- **Присоединение (добавление):** Возвращает множество элементов из s и значение v



Множества на связных списках

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

```
def empty(s):
    return s is Link.empty
```

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

```
def empty(s):
    return s is Link.empty


def contains(s, v):
    """Проверяет, что множество s содержит значение v.

    >>> s = Link(1, Link(2, Link(3)))
    >>> set_contains(s, 2)
    True
    """
```

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

```
def empty(s):
    return s is Link.empty


def contains(s, v):
    """Проверяет, что множество s содержит значение v.

    >>> s = Link(1, Link(2, Link(3)))
    >>> set_contains(s, 2)
    True
    """
    if empty(s):
        return False
```

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

```
def empty(s):
    return s is Link.empty


def contains(s, v):
    """Проверяет, что множество s содержит значение v.

    >>> s = Link(1, Link(2, Link(3)))
    >>> set_contains(s, 2)
    True
    """
    if empty(s):
        return False
    elif s.first == v:
        return True
```

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

```
def empty(s):
    return s is Link.empty


def contains(s, v):
    """Проверяет, что множество s содержит значение v.

    >>> s = Link(1, Link(2, Link(3)))
    >>> set_contains(s, 2)
    True
    """
    if empty(s):
        return False
    elif s.first == v:
        return True
    else:
        return contains(s.rest, v)
```

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

Временной порядок роста

```
def empty(s):
    return s is Link.empty


def contains(s, v):
    """Проверяет, что множество s содержит значение v.

    >>> s = Link(1, Link(2, Link(3)))
    >>> set_contains(s, 2)
    True
    """
    if empty(s):
        return False
    elif s.first == v:
        return True
    else:
        return contains(s.rest, v)
```

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

Временной порядок роста

```
def empty(s):
    return s is Link.empty
```

$\Theta(1)$

```
def contains(s, v):
    """Проверяет, что множество s содержит значение v.

    >>> s = Link(1, Link(2, Link(3)))
    >>> set_contains(s, 2)
    True
    """
    if empty(s):
        return False
    elif s.first == v:
        return True
    else:
        return contains(s.rest, v)
```

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

Временной порядок роста

```
def empty(s):
    return s is Link.empty
```

$\Theta(1)$

```
def contains(s, v):
    """Проверяет, что множество s содержит значение v.

    >>> s = Link(1, Link(2, Link(3)))
    >>> set_contains(s, 2)
    True
    """
    if empty(s):
        return False
    elif s.first == v:
        return True
    else:
        return contains(s.rest, v)
```

Время зависит от того, где v находится в s

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

Временной порядок роста

```
def empty(s):
    return s is Link.empty
```

$\Theta(1)$

```
def contains(s, v):
    """Проверяет, что множество s содержит значение v.

    >>> s = Link(1, Link(2, Link(3)))
    >>> set_contains(s, 2)
    True
    """
    if empty(s):
        return False
    elif s.first == v:
        return True
    else:
        return contains(s.rest, v)
```

Время зависит от того, где v находится в s

$\Theta(n)$

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

```
def empty(s):
    return s is Link.empty


def contains(s, v):
    """Проверяет, что множество s содержит значение v.

    >>> s = Link(1, Link(2, Link(3)))
    >>> set_contains(s, 2)
    True
    """
    if empty(s):
        return False
    elif s.first == v:
        return True
    else:
        return contains(s.rest, v)
```

Временной порядок роста

$\Theta(1)$

Время зависит от того, где v находится в s

$\Theta(n)$

Худший случай: v не находится в s

или

Средний случай: находится в равнораспределённом случайном месте

Множества как неупорядоченные последовательности

Предложение 1: Множество представлено связным списком уникальных элементов.

Временной порядок роста

```
def empty(s):
    return s is Link.empty
```

$\Theta(1)$

```
def contains(s, v):
    """Проверяет, что множество s содержит значение v.

    >>> s = Link(1, Link(2, Link(3)))
    >>> set_contains(s, 2)
    True
    """
    if empty(s):
        return False
    elif s.first == v:
        return True
    else:
        return contains(s.rest, v)
```

Время зависит от того, где v находится в s

$\Theta(n)$

Худший случай: v не находится в s

или

Средний случай: находится в равнораспределённом случайном месте

(Пример)

Множества как неупорядоченные последовательности

Множества как неупорядоченные последовательности

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)
```

Множества как неупорядоченные последовательности

Временной порядок роста

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)
```

Множества как неупорядоченные последовательности

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)
```

Временной порядок роста

$$\Theta(n)$$

Множества как неупорядоченные последовательности

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)
```

Временной порядок роста

$\Theta(n)$

Размер множества

Множества как неупорядоченные последовательности

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)


def intersect(s, t):
    if s is Link.empty:
        return Link.empty
    rest = ____________________
    if contains(t, s.first):
        return ____________________
    else:
        return rest
```

Временной порядок роста

$\Theta(n)$

Размер множества

Множества как неупорядоченные последовательности

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)


def intersect(s, t):
    if s is Link.empty:
        return Link.empty
    rest = intersect(s.rest, t)
    if contains(t, s.first):
        return ____________________
    else:
        return rest
```

Временной порядок роста

$\Theta(n)$

Размер множества

Множества как неупорядоченные последовательности

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)


def intersect(s, t):
    if s is Link.empty:
        return Link.empty
    rest = intersect(s.rest, t)
    if contains(t, s.first):
        return Link(s.first, rest)
    else:
        return rest
```

Временной порядок роста

$\Theta(n)$

Размер множества

Множества как неупорядоченные последовательности

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)
```

```
def intersect(s, t):
    if s is Link.empty:
        return Link.empty
    rest = intersect(s.rest, t)
    if contains(t, s.first):
        return Link(s.first, rest)
    else:
        return rest
```

Временной порядок роста

$\Theta(n)$

Размер множества

$\Theta(n^2)$

Множества как неупорядоченные последовательности

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)


def intersect(s, t):
    if s is Link.empty:
        return Link.empty
    rest = intersect(s.rest, t)
    if contains(t, s.first):
        return Link(s.first, rest)
    else:
        return rest
```

Временной порядок роста

$\Theta(n)$

Размер множества

$\Theta(n^2)$

Если множества одного размера

Множества как неупорядоченные последовательности

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)


def intersect(s, t):
    if s is Link.empty:
        return Link.empty
    rest = intersect(s.rest, t)
    if contains(t, s.first):
        return Link(s.first, rest)
    else:
        return rest
```

Временной порядок роста

$\Theta(n)$

Размер множества

$\Theta(n^2)$

Если множества одного размера

(Пример)

Множества как упорядоченные последовательности

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Части программы, которые...	Множества — это...	Используя...

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Части программы, которые...	Множества — это...	Используя...
используют множества для хранения значений		

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Части программы, которые...	Множества — это...	Используя...
используют множества для хранения значений	неупорядоченные коллекции	

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Части программы, которые...	Множества — это...	Используя...
используют множества для хранения значений	неупорядоченные коллекции	`empty`, `set_contains`, `adjoin_set`, `intersect_set`, `union_set`

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Части программы, которые...	Множества — это...	Используя...
используют множества для хранения значений	неупорядоченные коллекции	`empty`, `set_contains`, `adjoin_set`, `intersect_set`, `union_set`
Реализуют операции над множествами		

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Части программы, которые...	Множества — это...	Используя...
используют множества для хранения значений	неупорядоченные коллекции	`empty`, `set_contains`, `adjoin_set`, `intersect_set`, `union_set`
Реализуют операции над множествами	Упорядоченные связные списки	

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Части программы, которые...	Множества — это...	Используя...
используют множества для хранения значений	неупорядоченные коллекции	`empty`, `set_contains`, `adjoin_set`, `intersect_set`, `union_set`
Реализуют операции над множествами	Упорядоченные связные списки	`first`, `rest`, `<`, `>`, `==`

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Части программы, которые...	Множества — это...	Используя...
используют множества для хранения значений	неупорядоченные коллекции	`empty`, `set_contains`, `adjoin_set`, `intersect_set`, `union_set`
Реализуют операции над множествами	Упорядоченные связные списки	`first`, `rest`, `<`, `>`, `==`

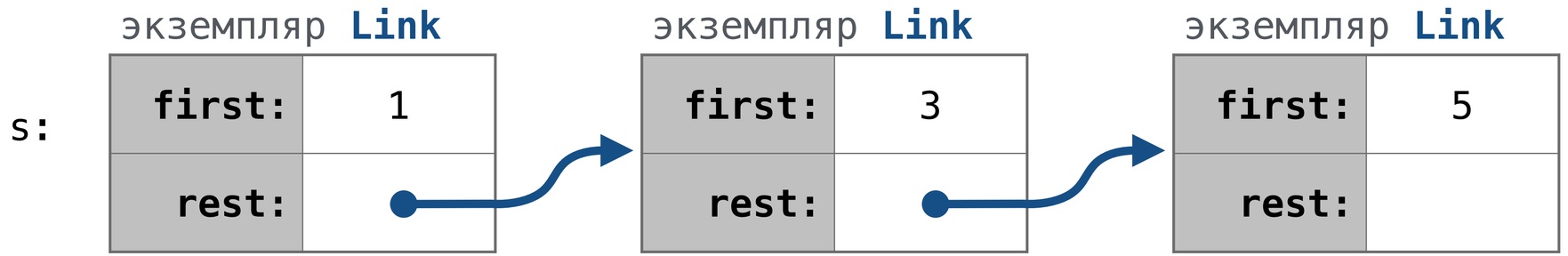
Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Части программы, которые...	Множества — это...	Используя...
используют множества для хранения значений	неупорядоченные коллекции	`empty`, `set_contains`, `adjoin_set`, `intersect_set`, `union_set`
Реализуют операции над множествами	Упорядоченные связные списки	`first`, `rest`, `<`, `>`, `==`

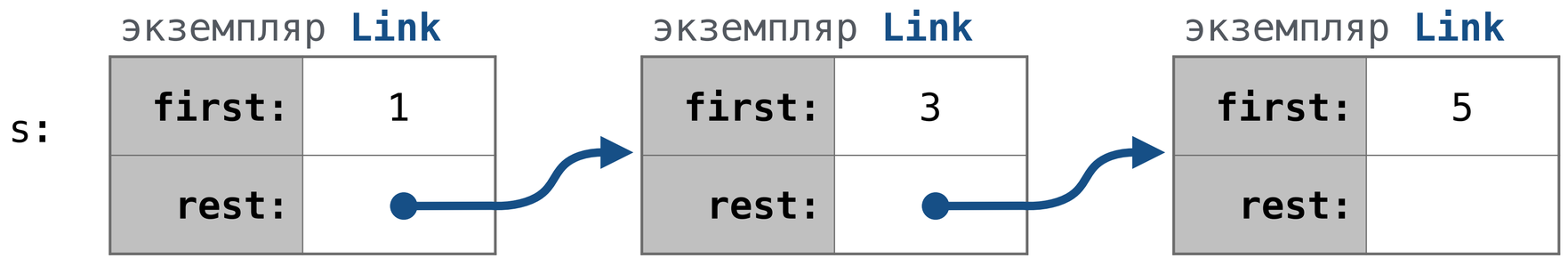
Различные части программы могут делать различные предположения о данных

Поиск в упорядоченном связном списке



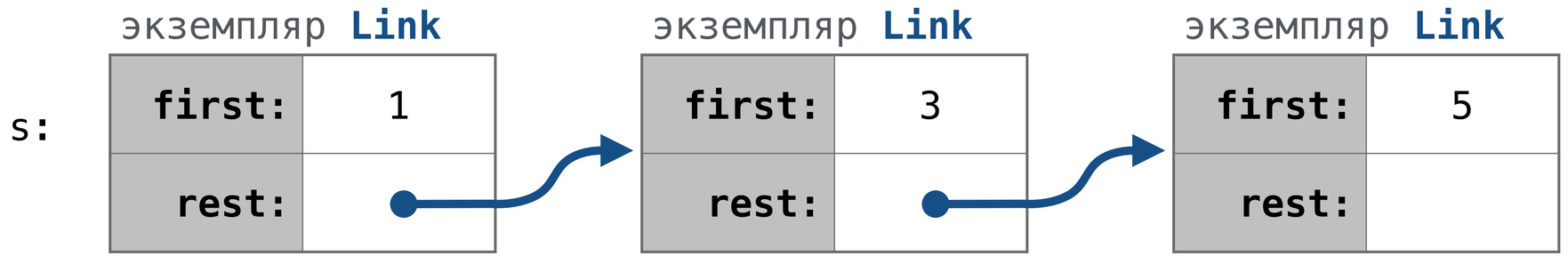
Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
```



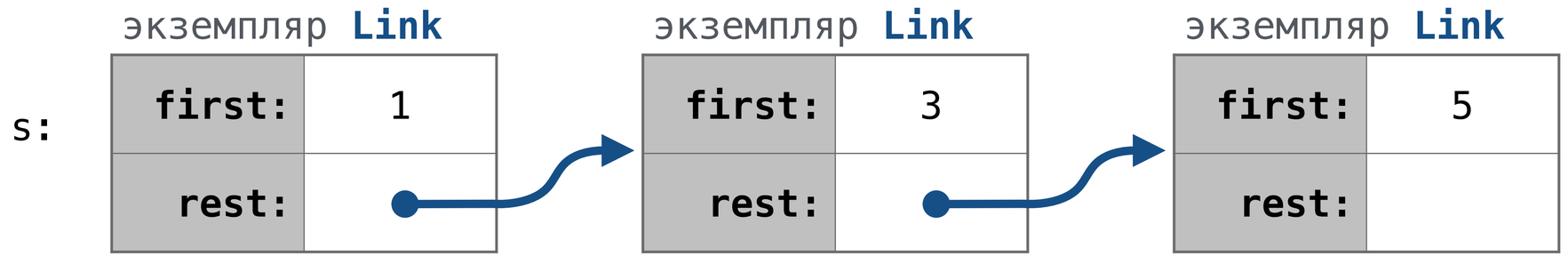
Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
```



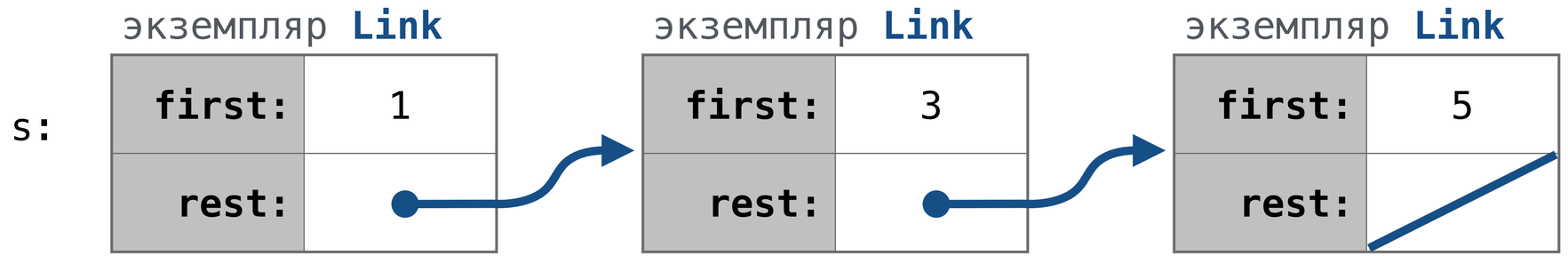
Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
```



Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```



Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

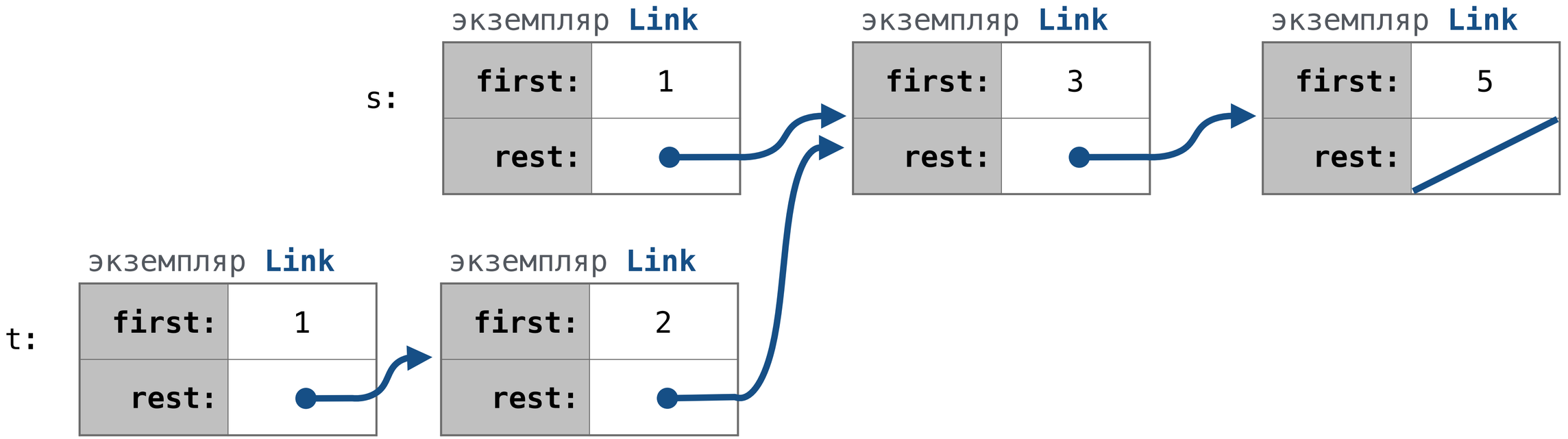
t:

Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

Поиск в упорядоченном связном списке

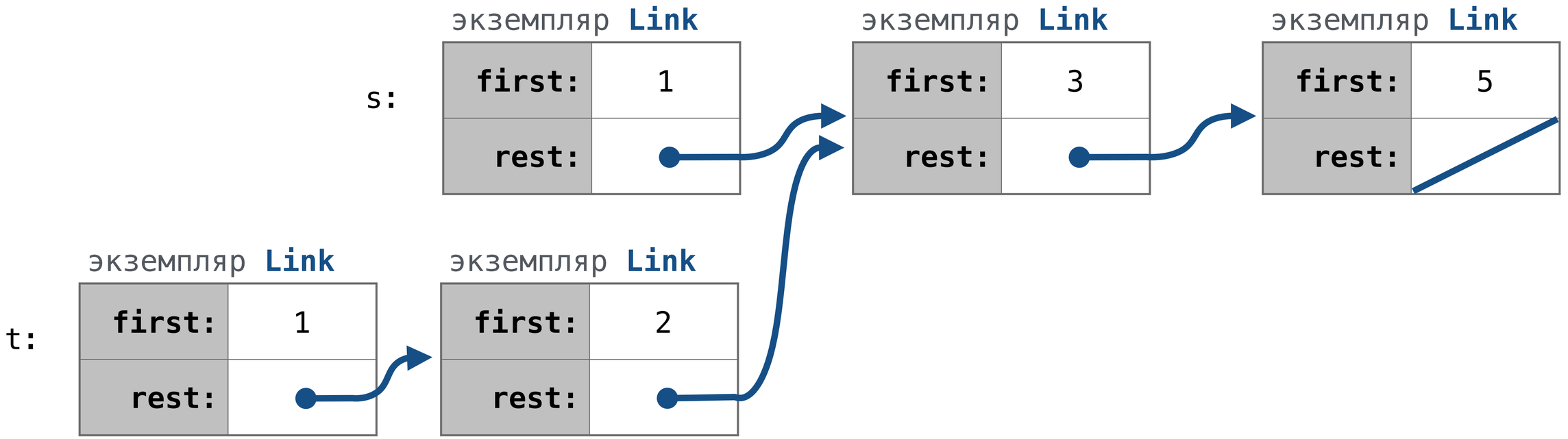
```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```



Поиск в упорядоченном связном списке

Временной порядок роста

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

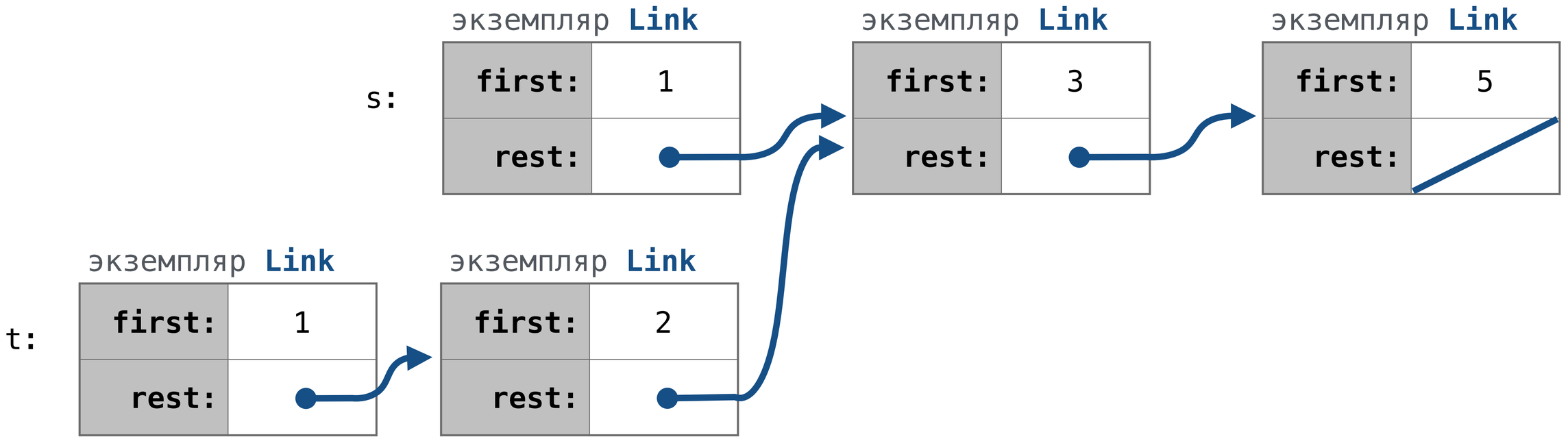


Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

Временной порядок роста

$$\Theta(n)$$

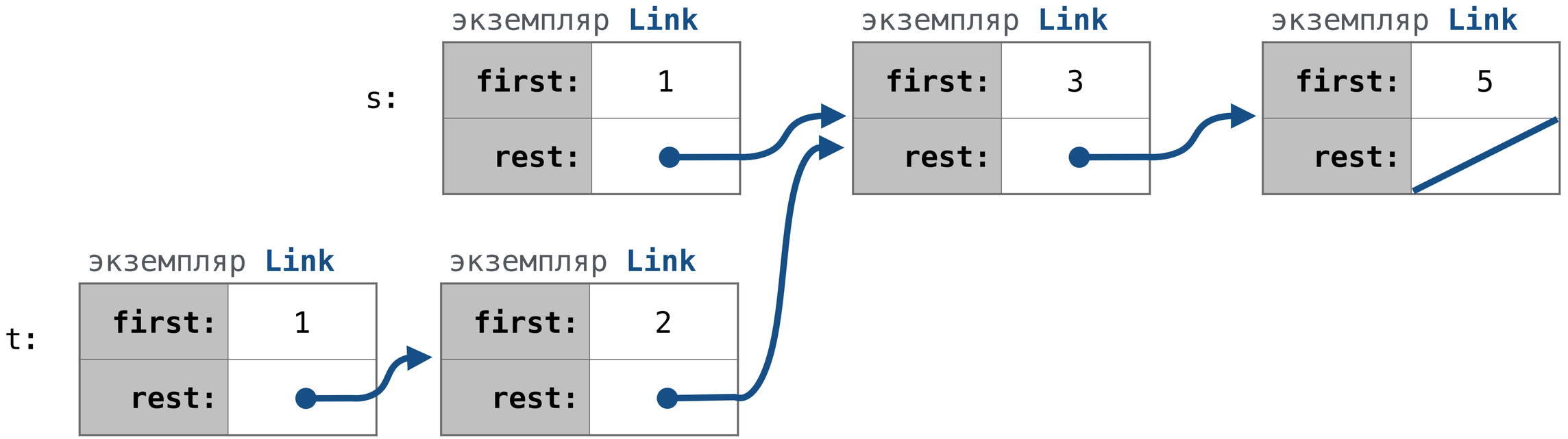


Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

Действие **Временной порядок роста**

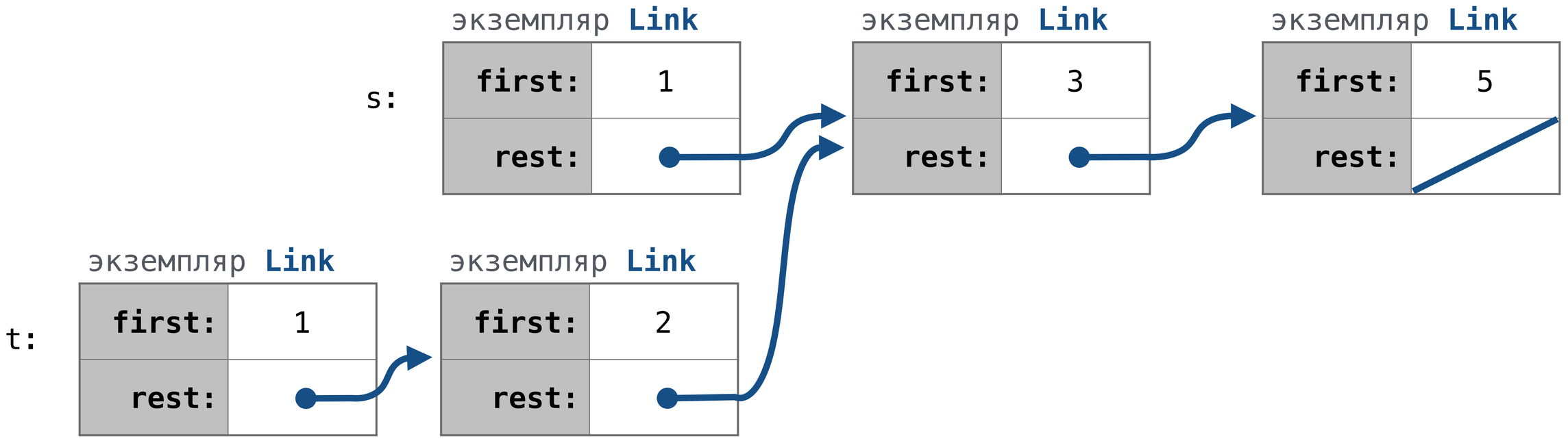
$$\Theta(n)$$



Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

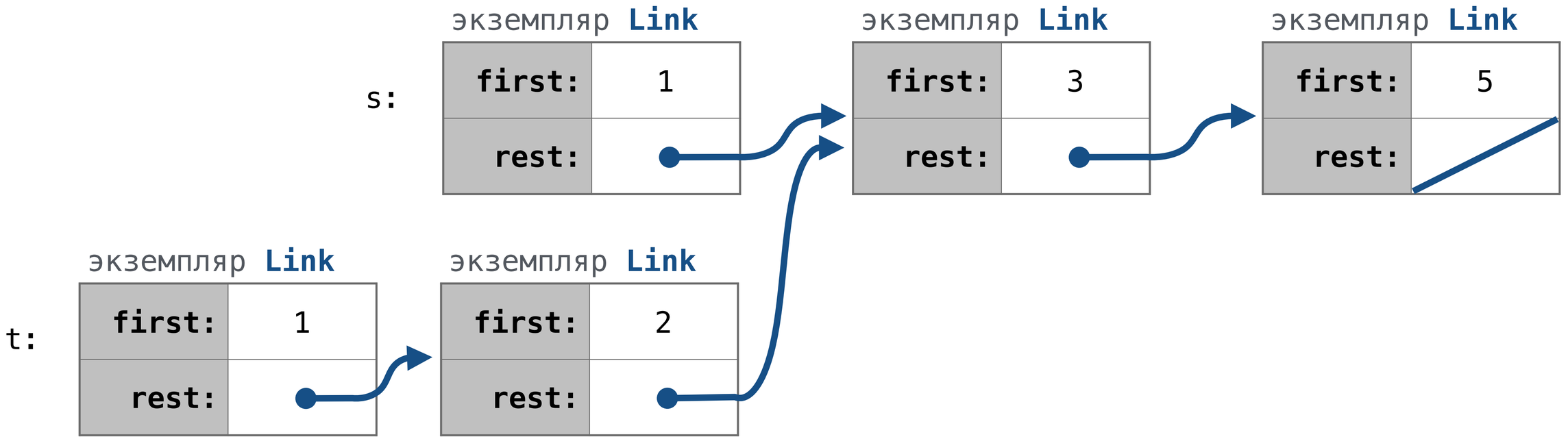
Действие	Временной порядок роста
contains	$\Theta(n)$



Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

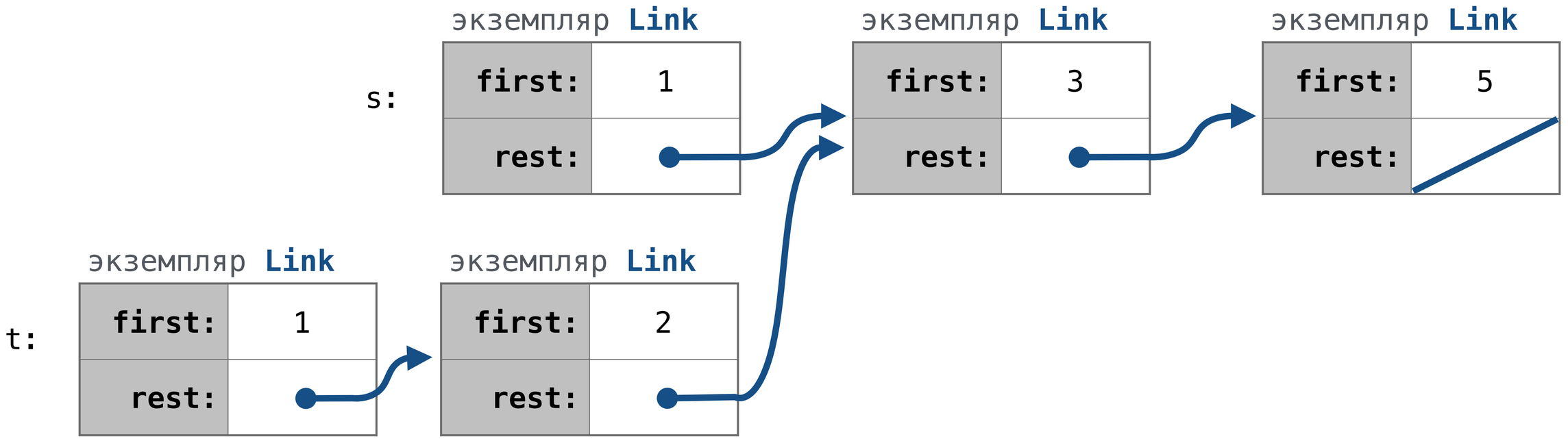
Действие	Временной порядок роста
contains	$\Theta(n)$
adjoin	



Поиск в упорядоченном связном списке

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

Действие	Временной порядок роста
contains	$\Theta(n)$
adjoin	$\Theta(n)$



Операции над множествами

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

```
def intersect(s, t):
```

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

```
def intersect(s, t):
    if empty(s) or empty(t):
        return Link.empty
```

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

```
def intersect(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
```

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

```
def intersect(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
```

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

```
def intersect(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
```

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

```
def intersect(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
```

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

```
def intersect(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
        elif e2 < e1:
            return intersect(s, t.rest)
```

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

```
def intersect(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
        elif e2 < e1:
            return intersect(s, t.rest)
```

Порядок роста?

Множества как упорядоченные последовательности

Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

```
def intersect(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
        elif e2 < e1:
            return intersect(s, t.rest)
```

Порядок роста? Если s и t размера n, то $\Theta(n)$

Множества как упорядоченные последовательности

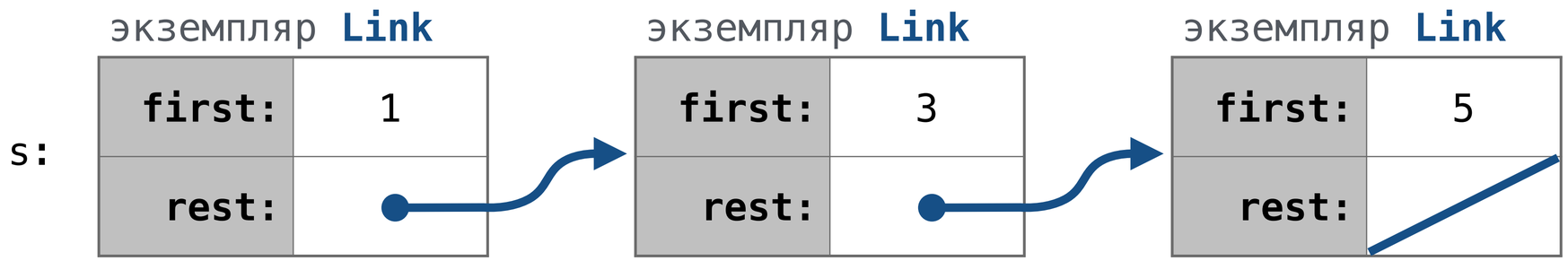
Предложение 2: Множество представлено связным списком уникальных элементов, **упорядоченных от меньшего к большему.**

```
def intersect(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
        elif e2 < e1:
            return intersect(s, t.rest)
```

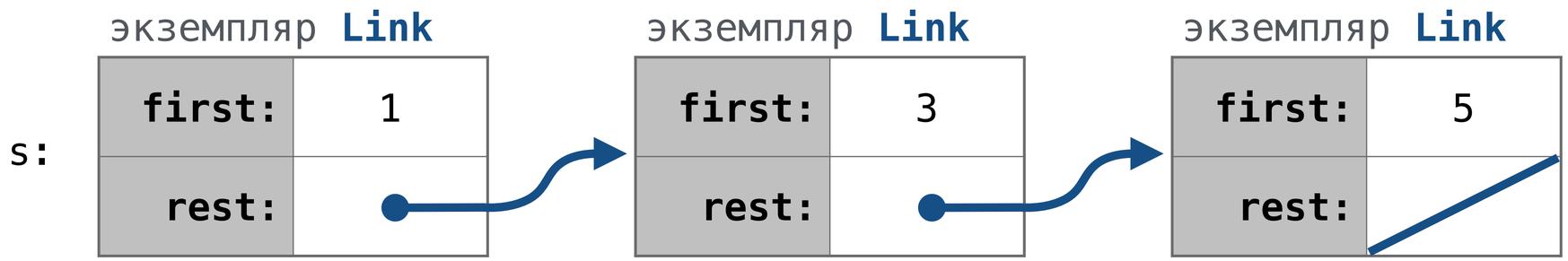
Порядок роста? Если s и t размера n, то $\Theta(n)$ (Пример)

Изменение множества

Добавление в упорядоченный список

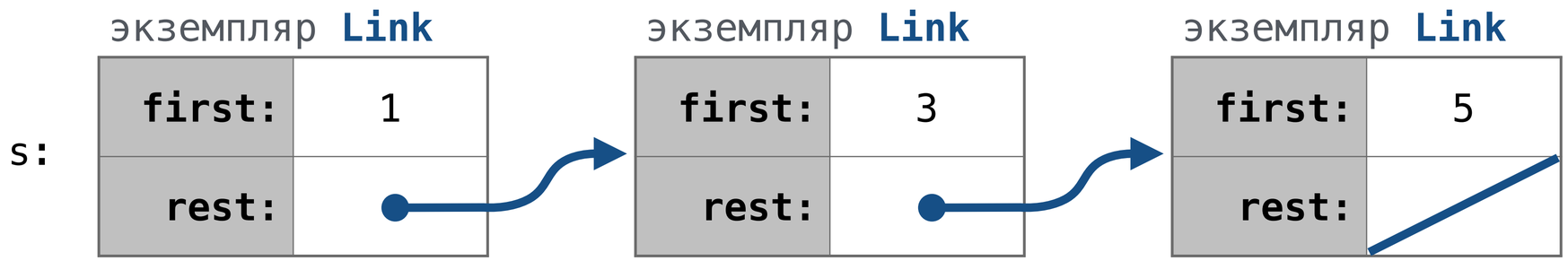


Добавление в упорядоченный список



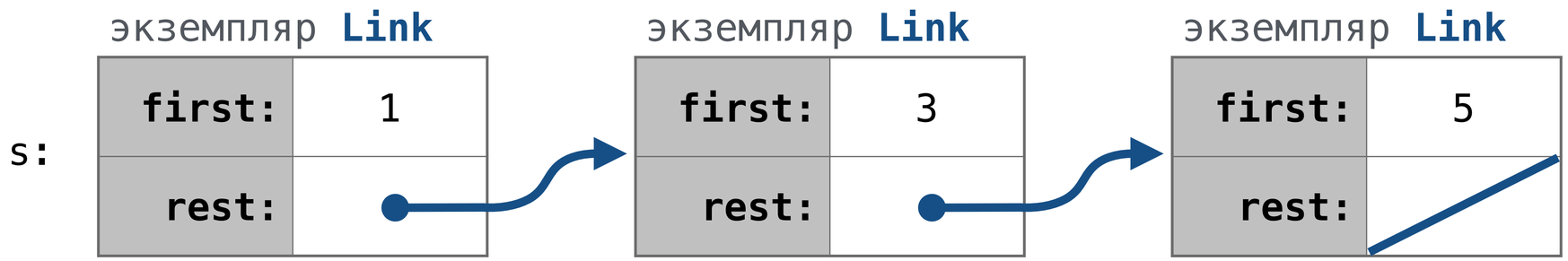

```
add(s, 0)
```

Добавление в упорядоченный список

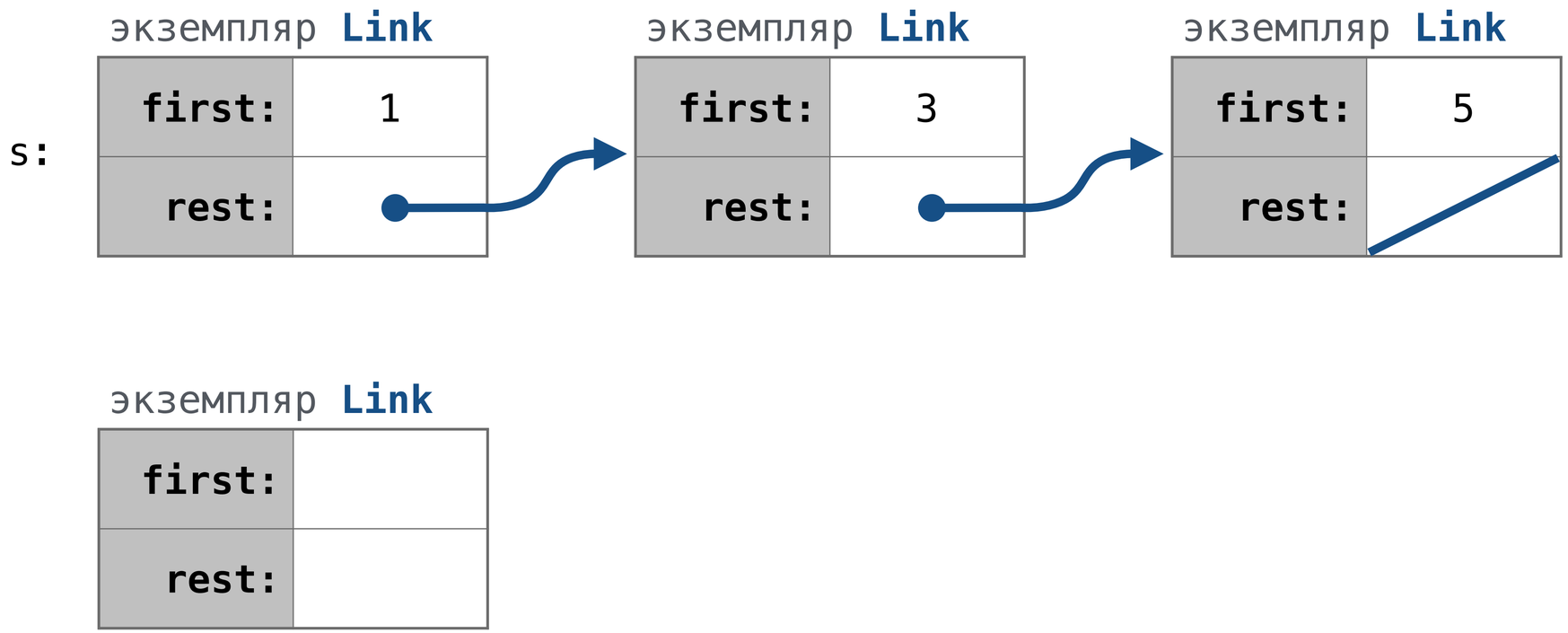



```
add(s, 0)   Попробуем использовать исходный объект
```

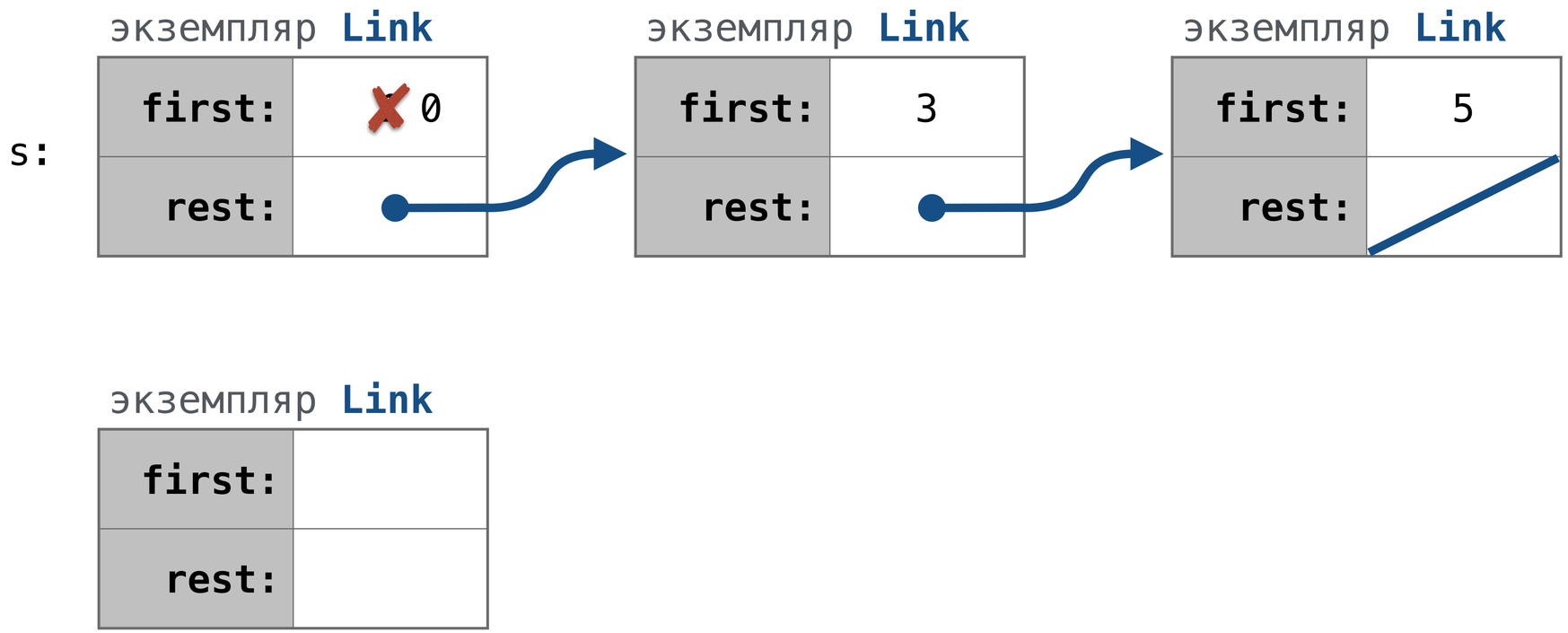
Добавление в упорядоченный список



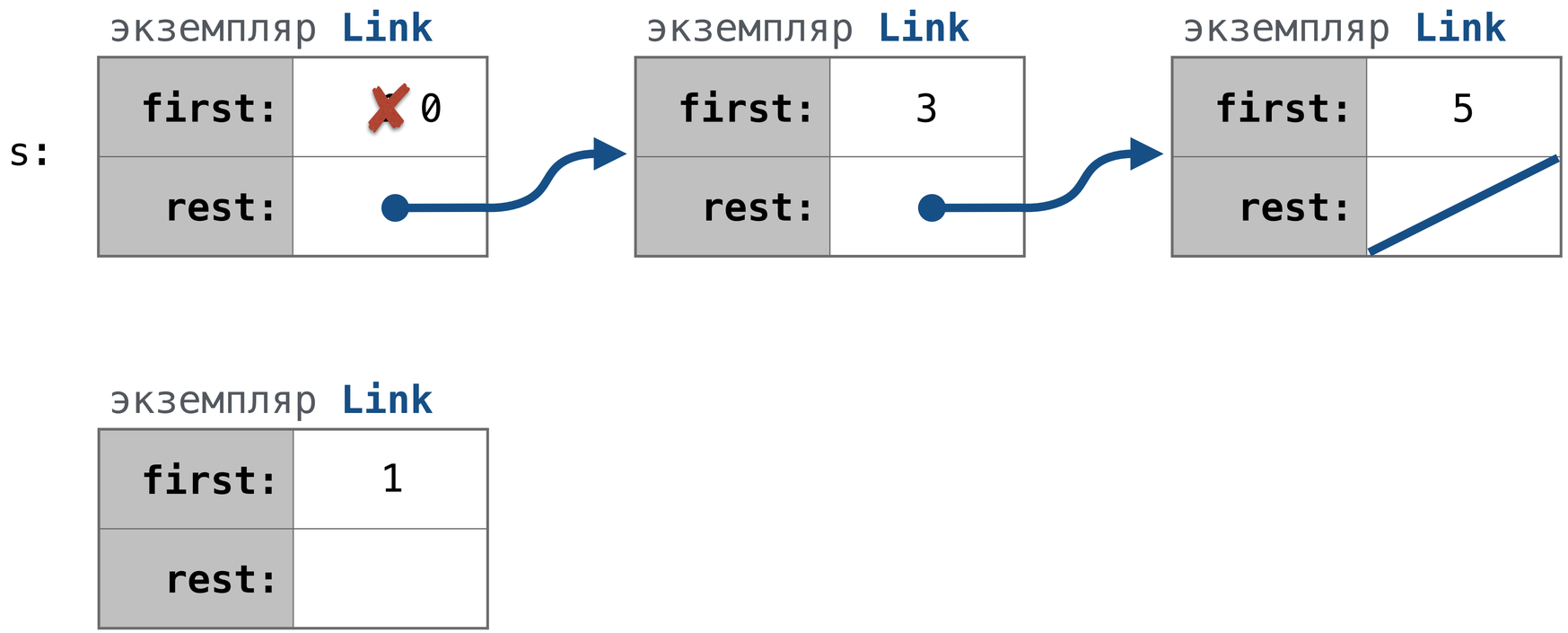
Добавление в упорядоченный список



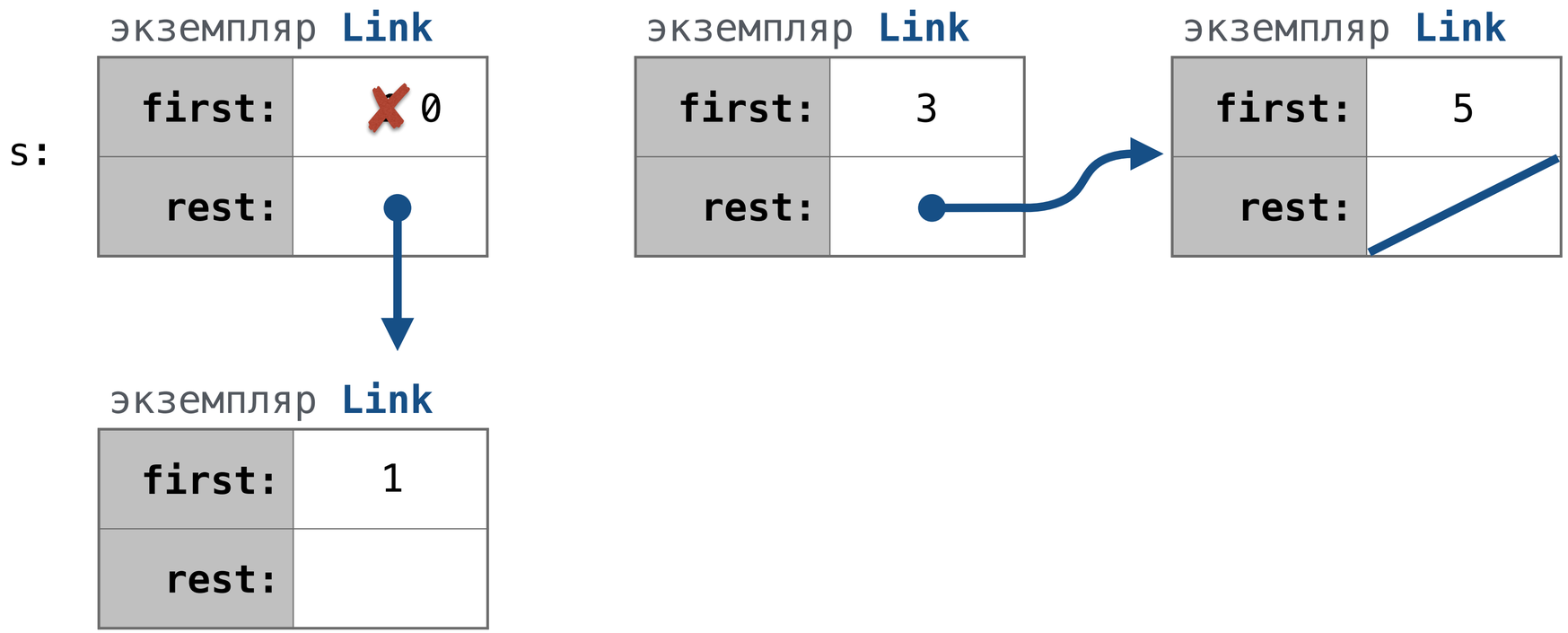
Добавление в упорядоченный список



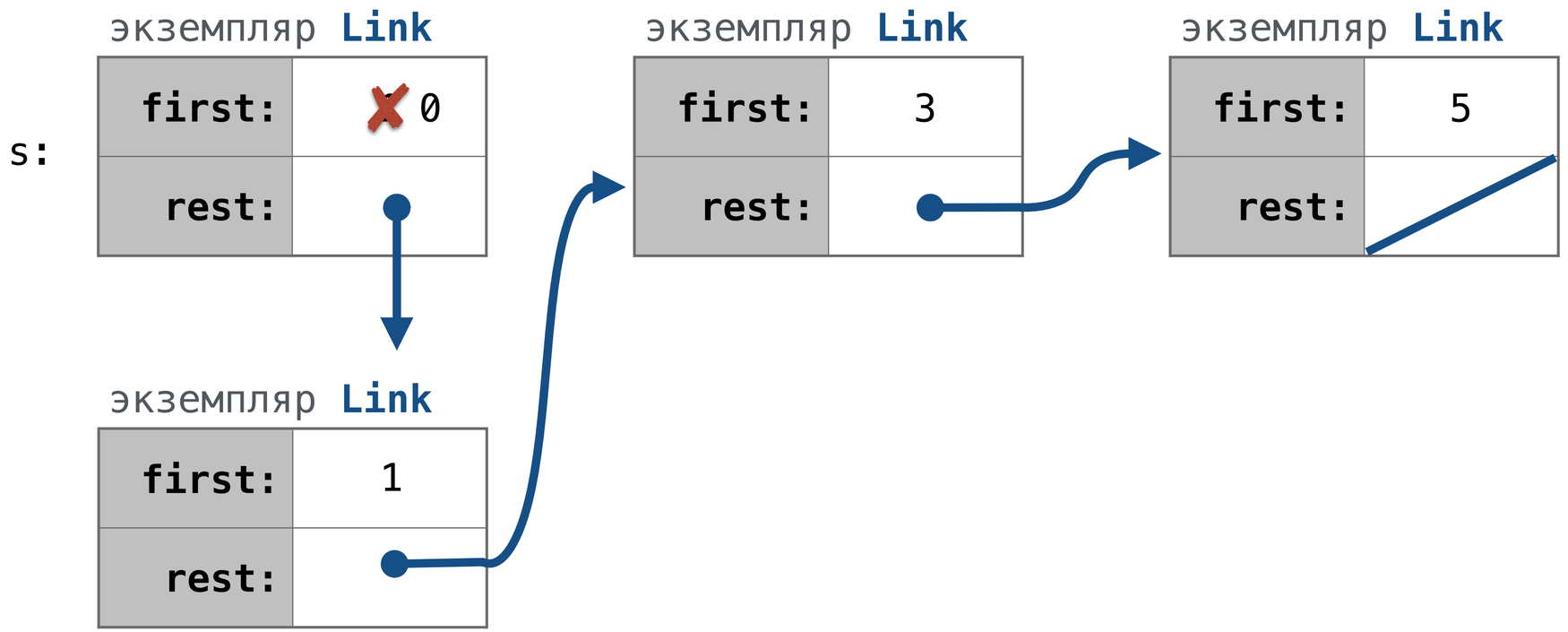
Добавление в упорядоченный список



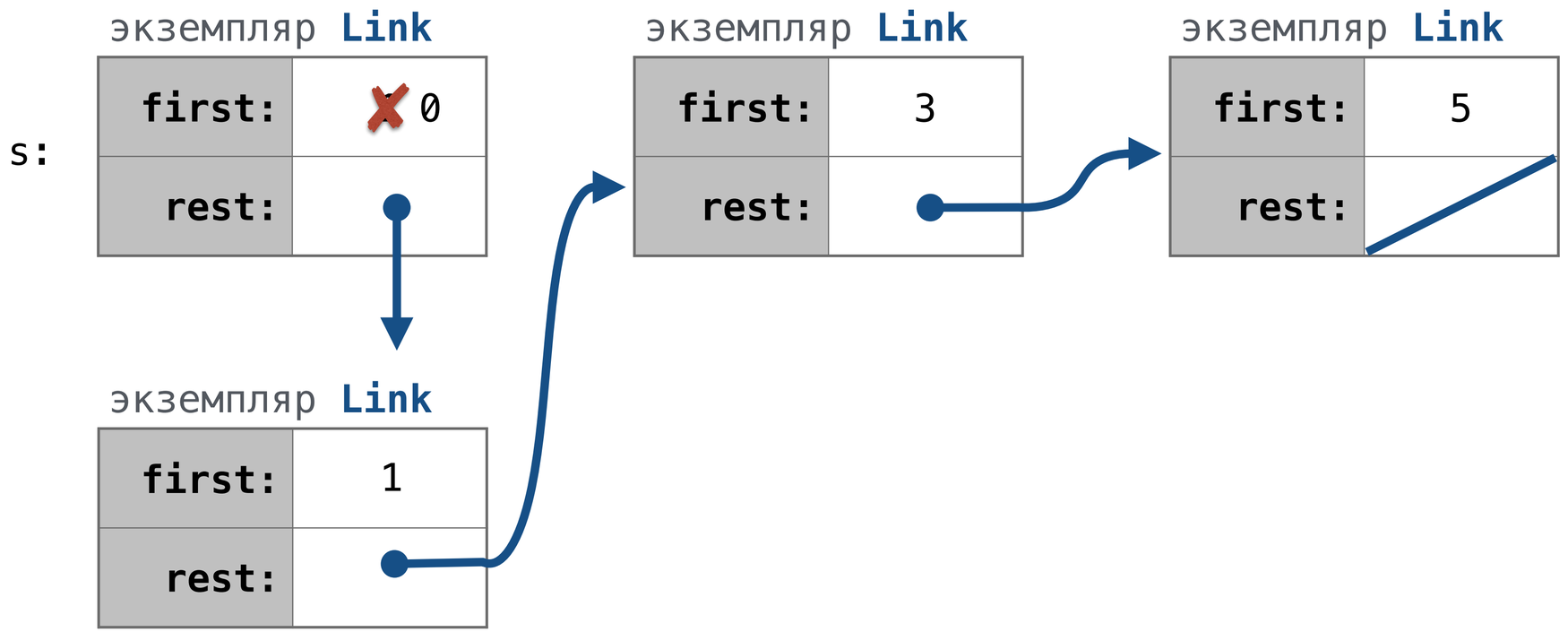
Добавление в упорядоченный список



Добавление в упорядоченный список

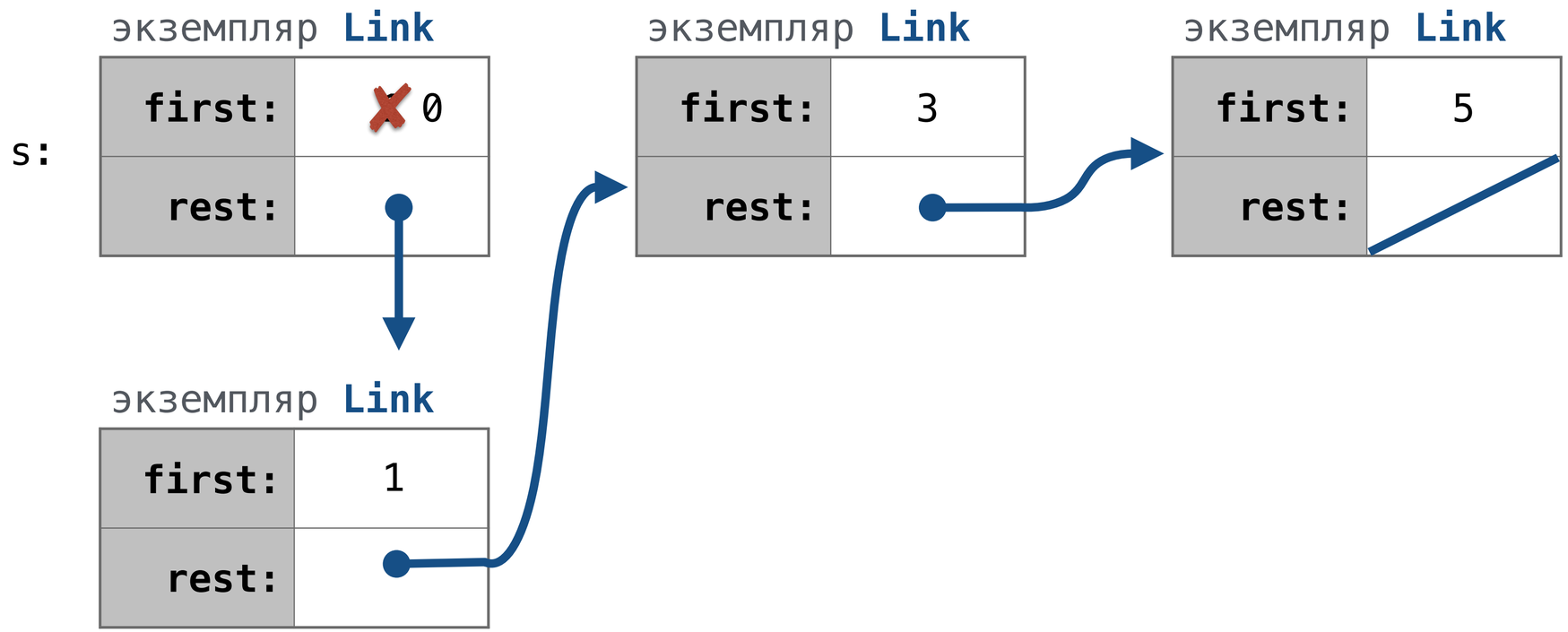


Добавление в упорядоченный список



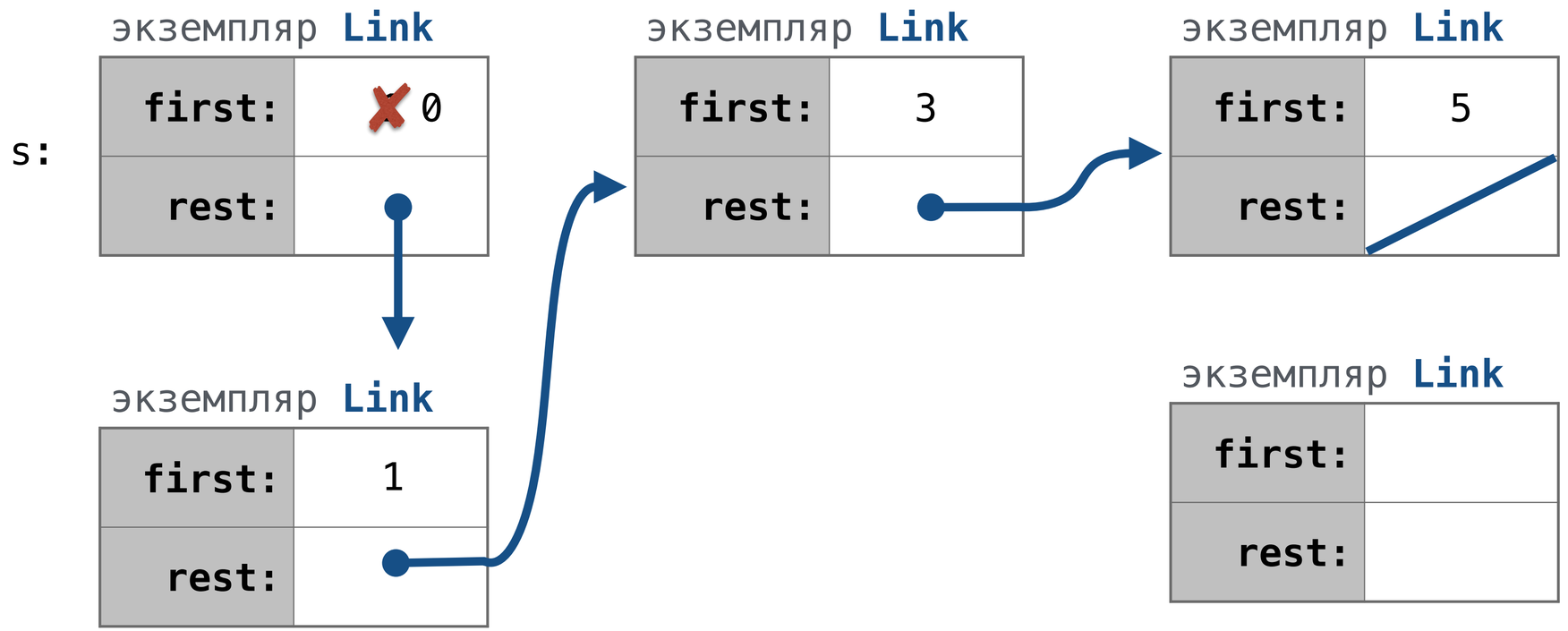

```
add(s, 4)
```

Добавление в упорядоченный список



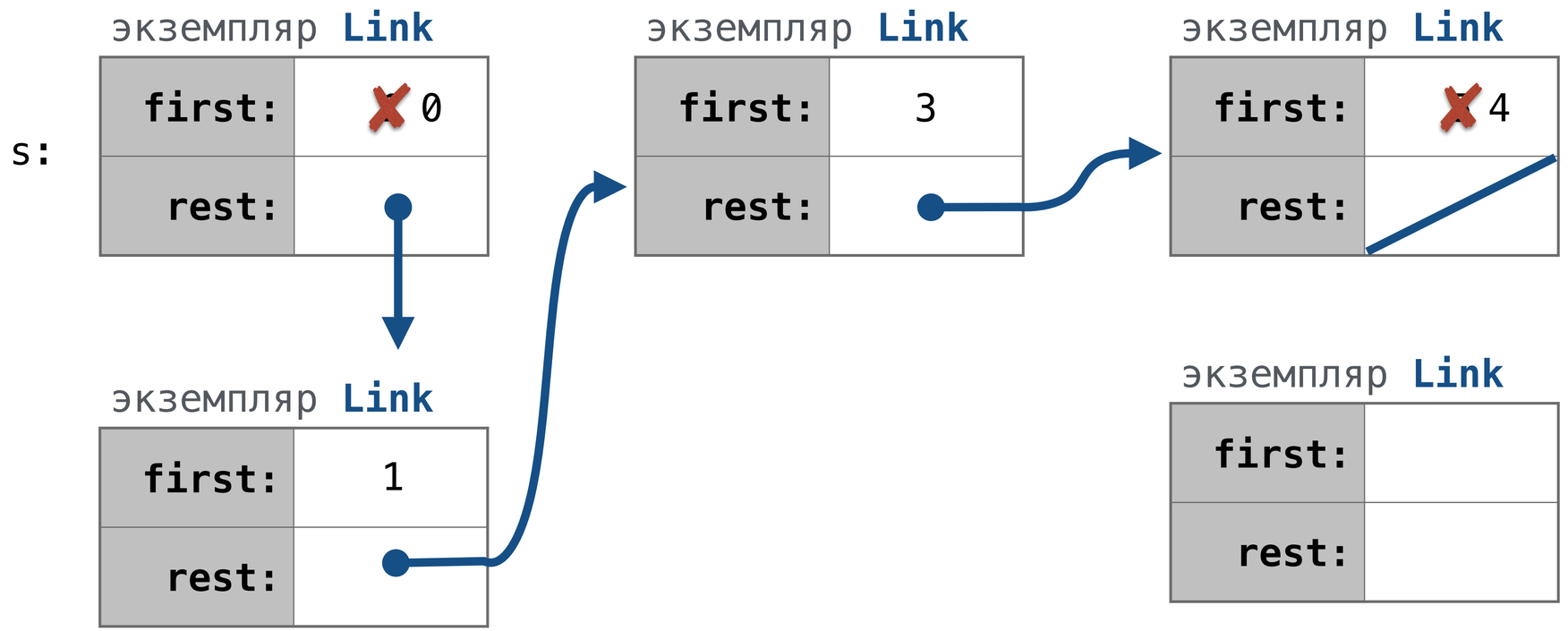

```
add(s, 4)
```

Добавление в упорядоченный список



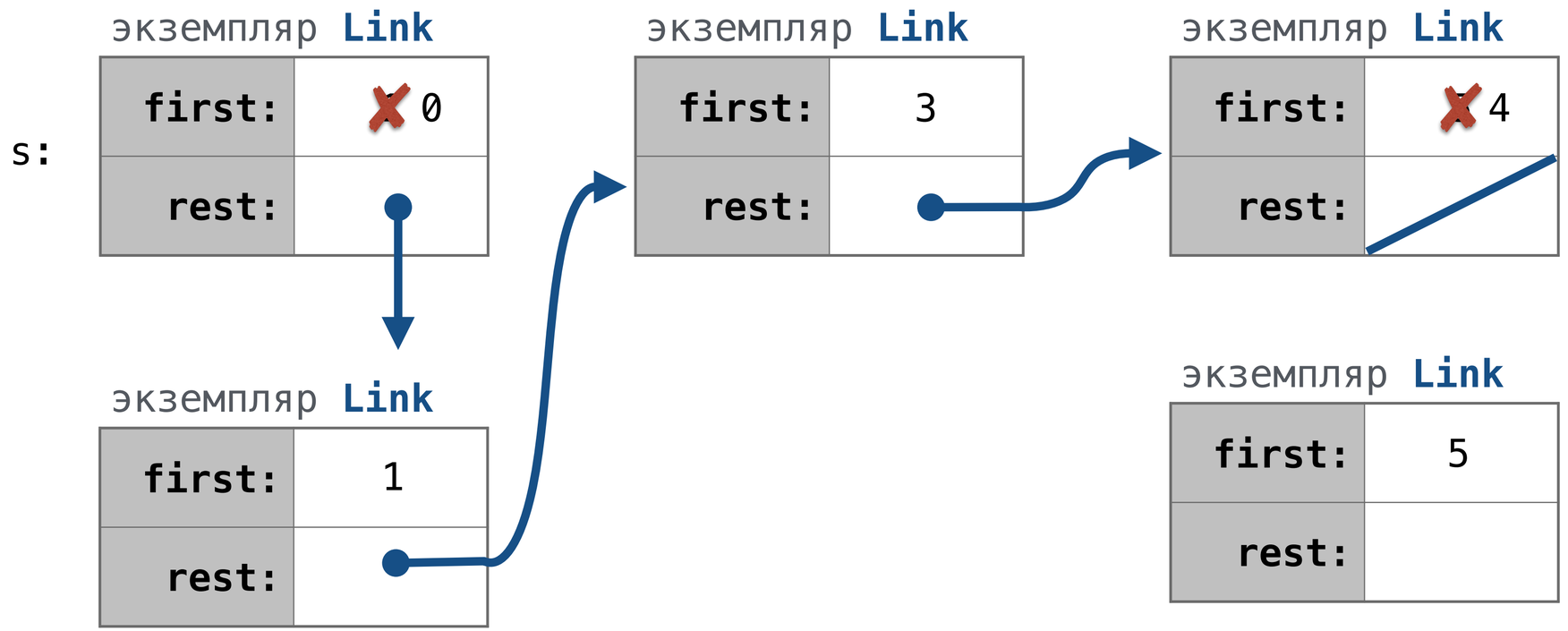

```
add(s, 4)
```

Добавление в упорядоченный список



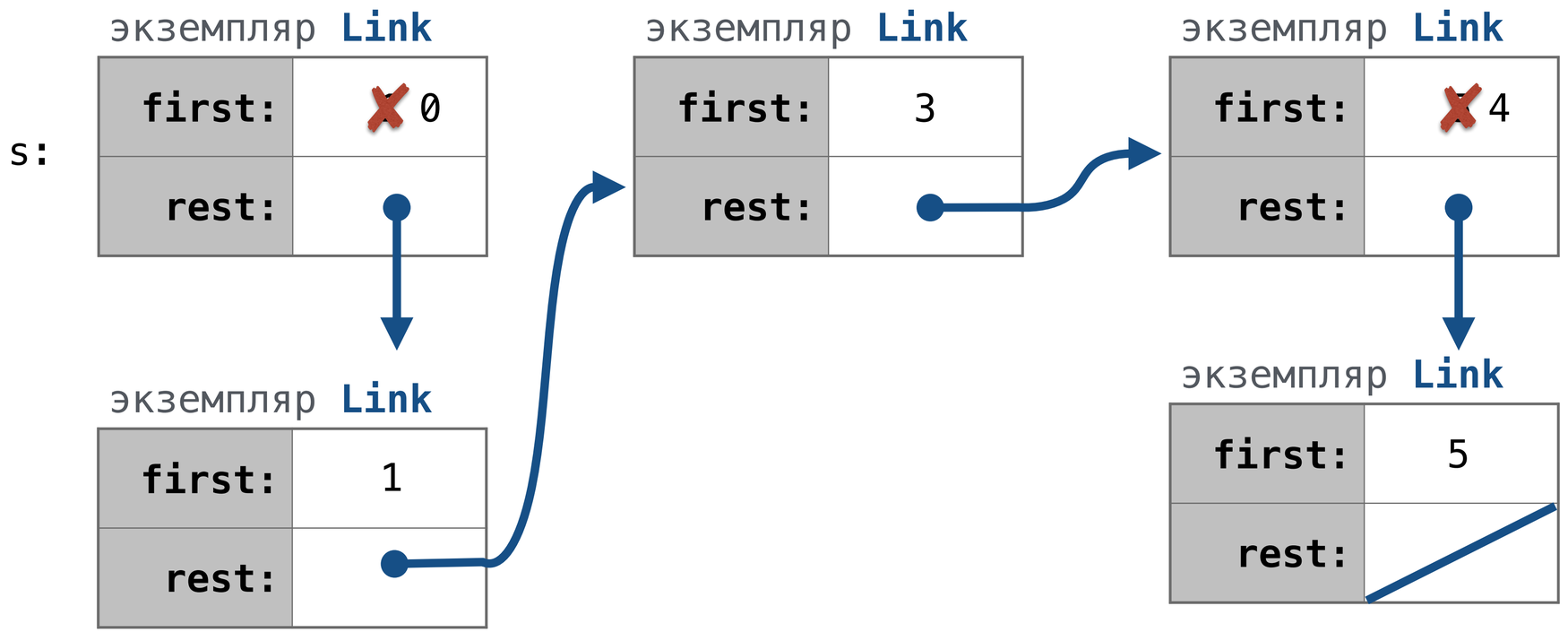

```
add(s, 4)
```

Добавление в упорядоченный список



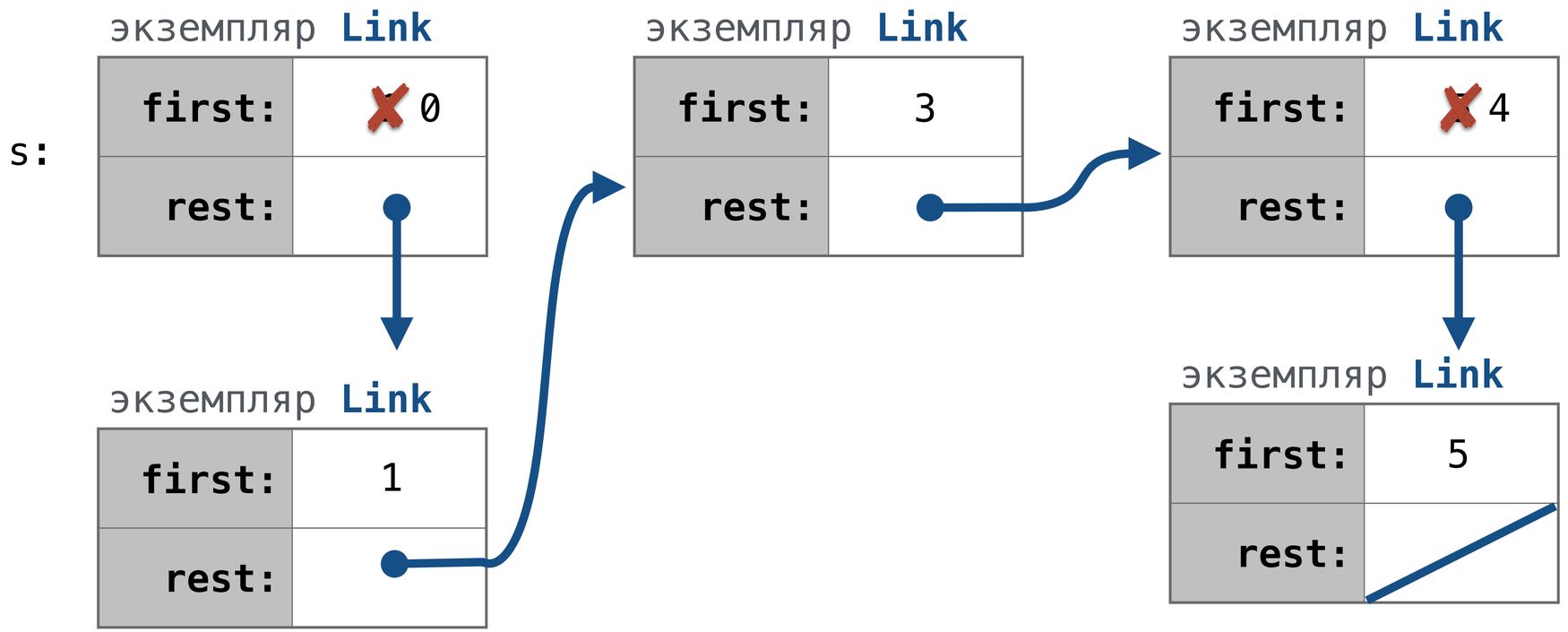

```
add(s, 4)
```

Добавление в упорядоченный список



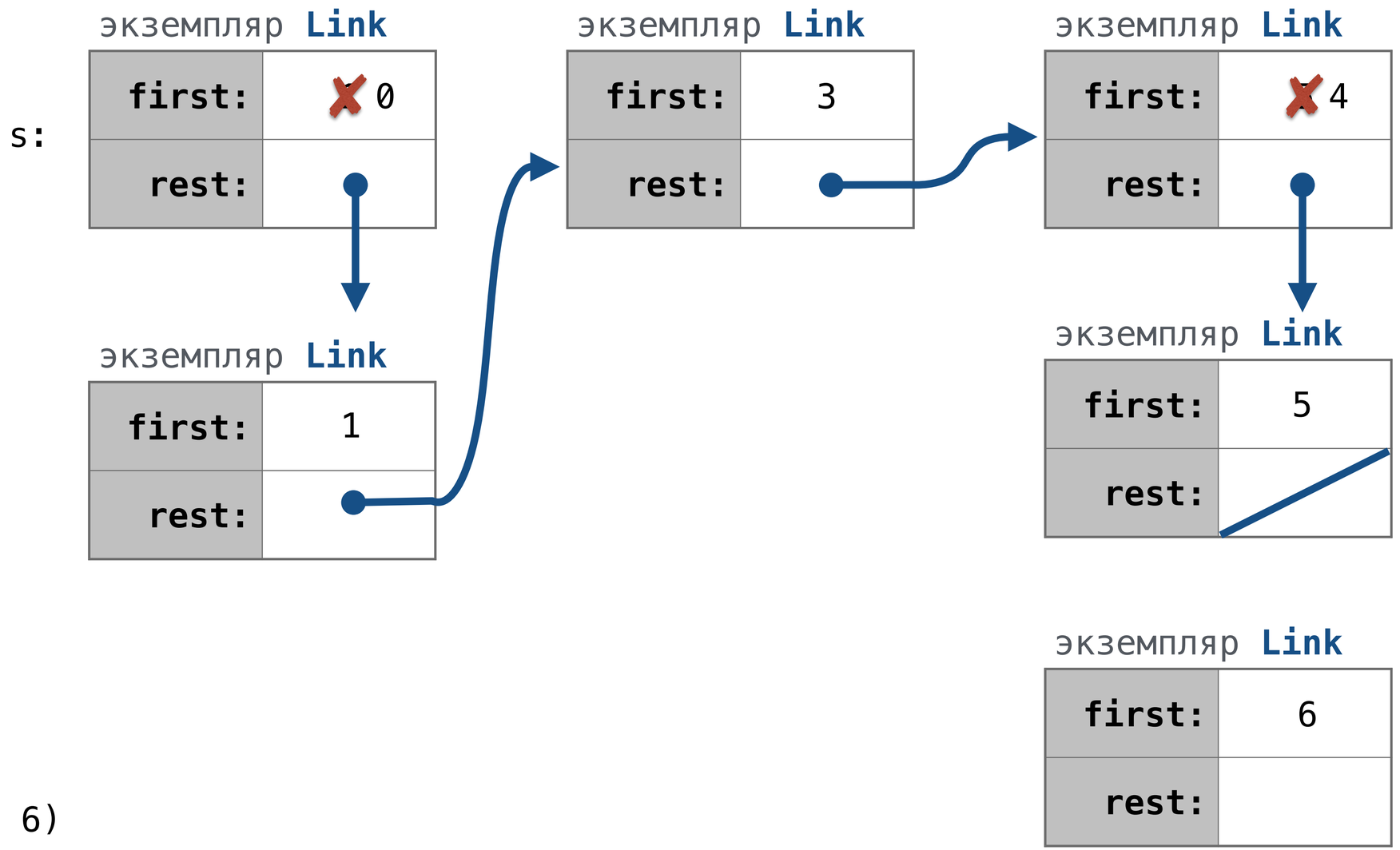

```
add(s, 4)
```

Добавление в упорядоченный список



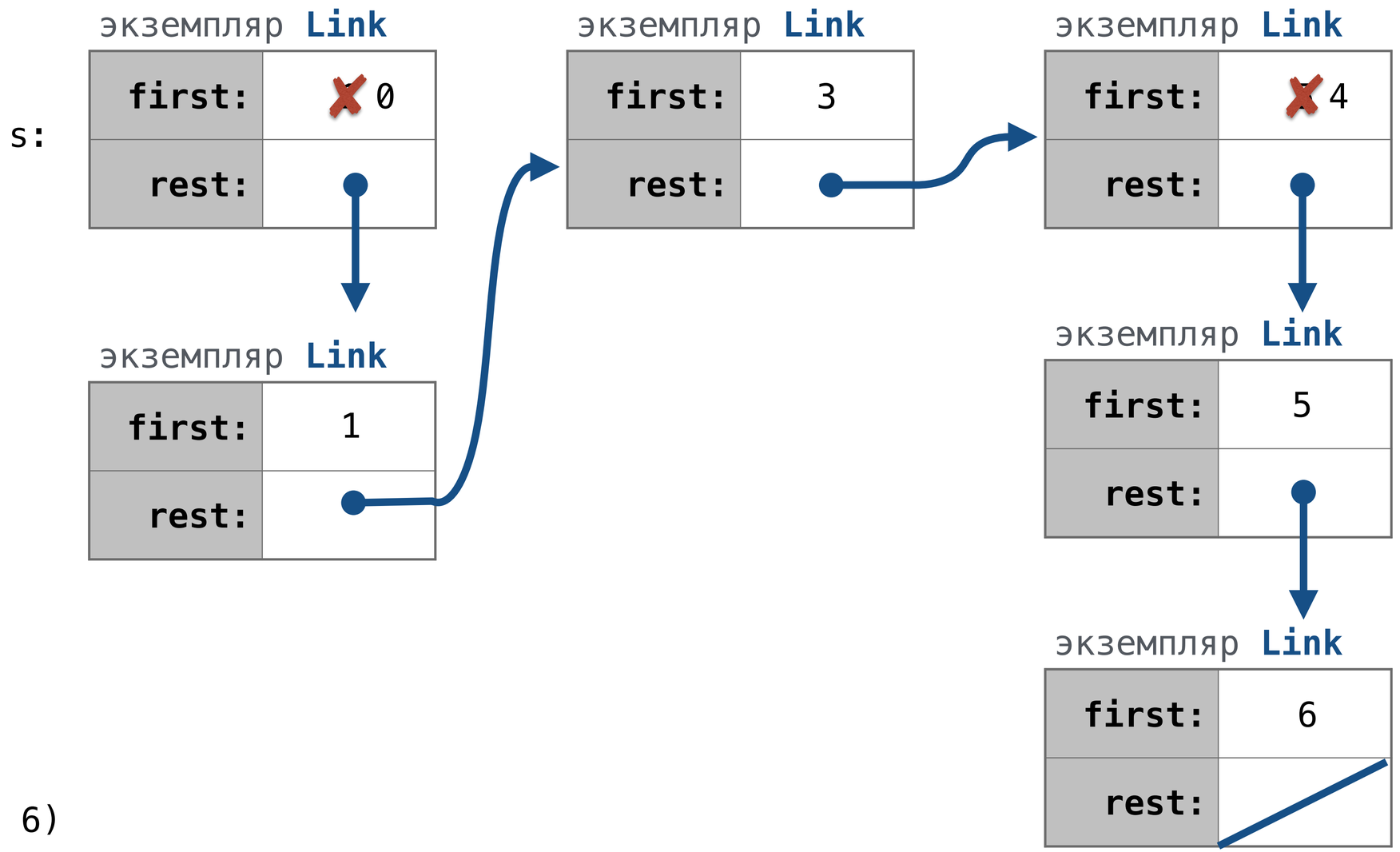

```
add(s, 6)
```

Добавление в упорядоченный список



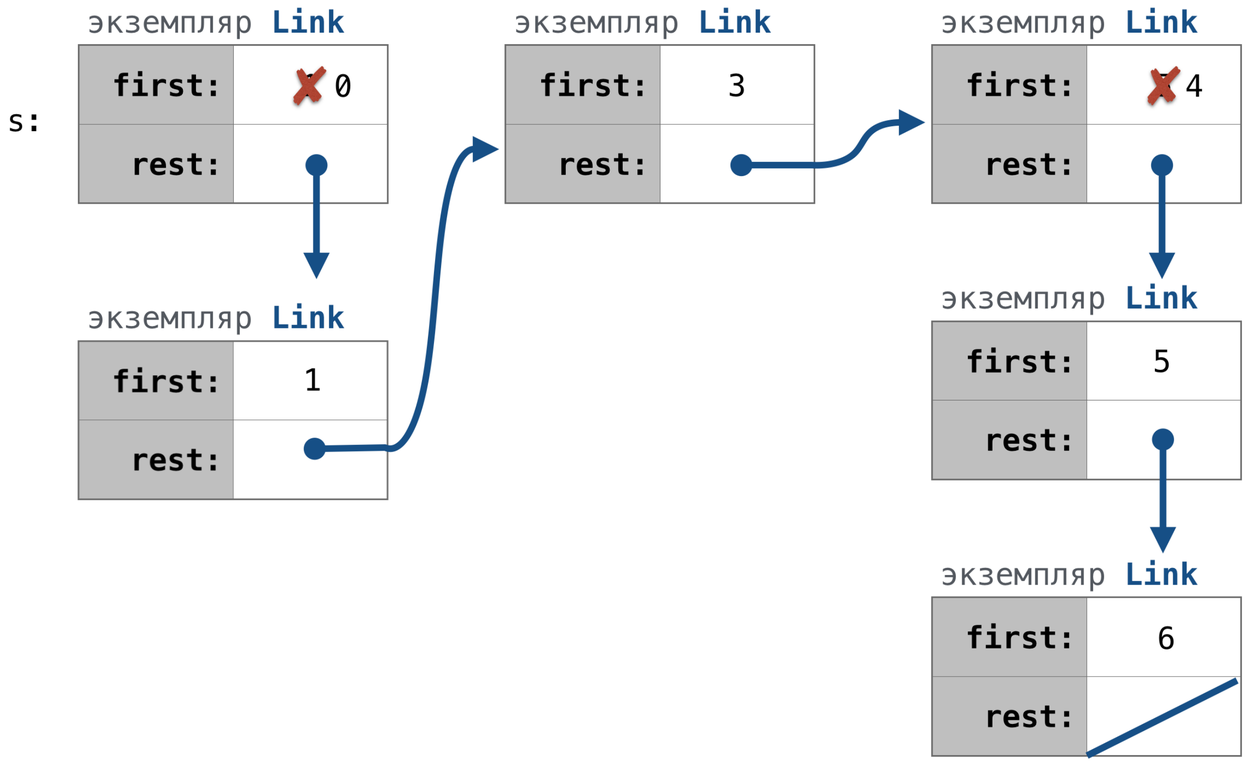

```
add(s, 6)
```

Добавление в упорядоченный список



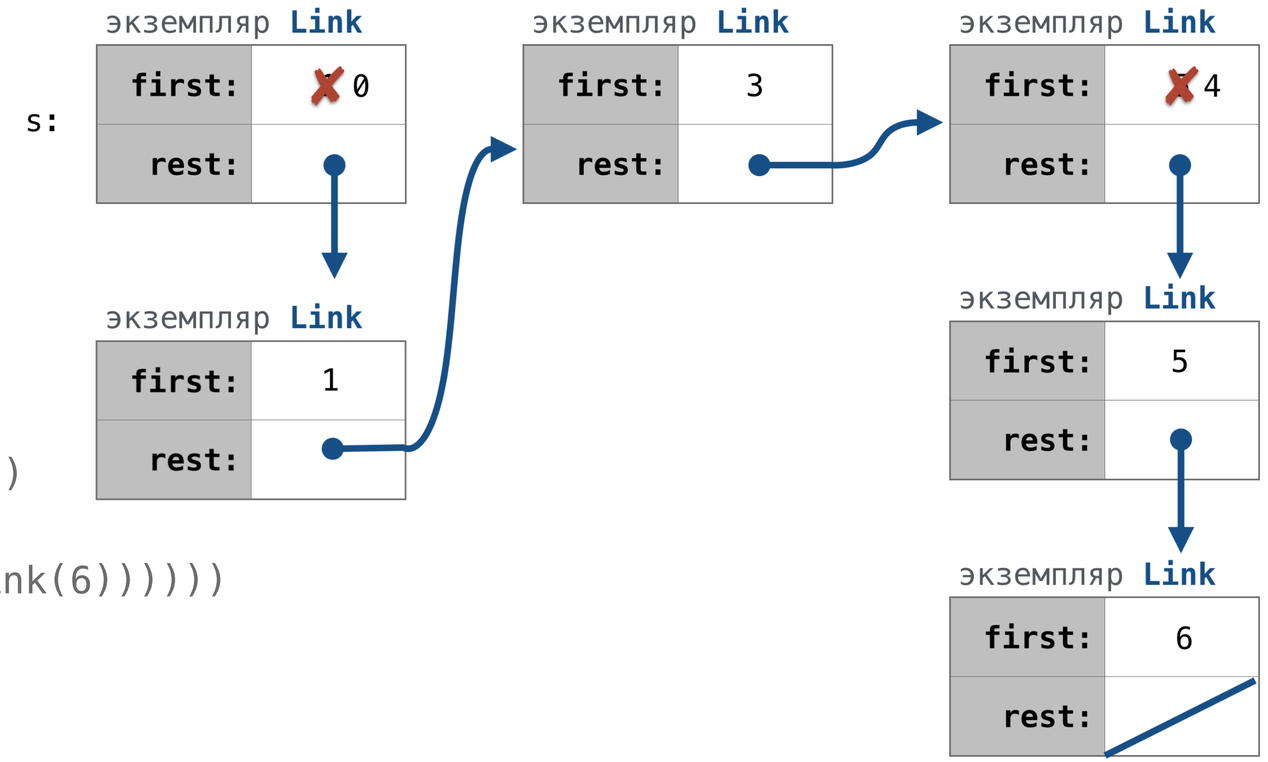

```
add(s, 6)
```

Добавление в множество на упорядоченных списках



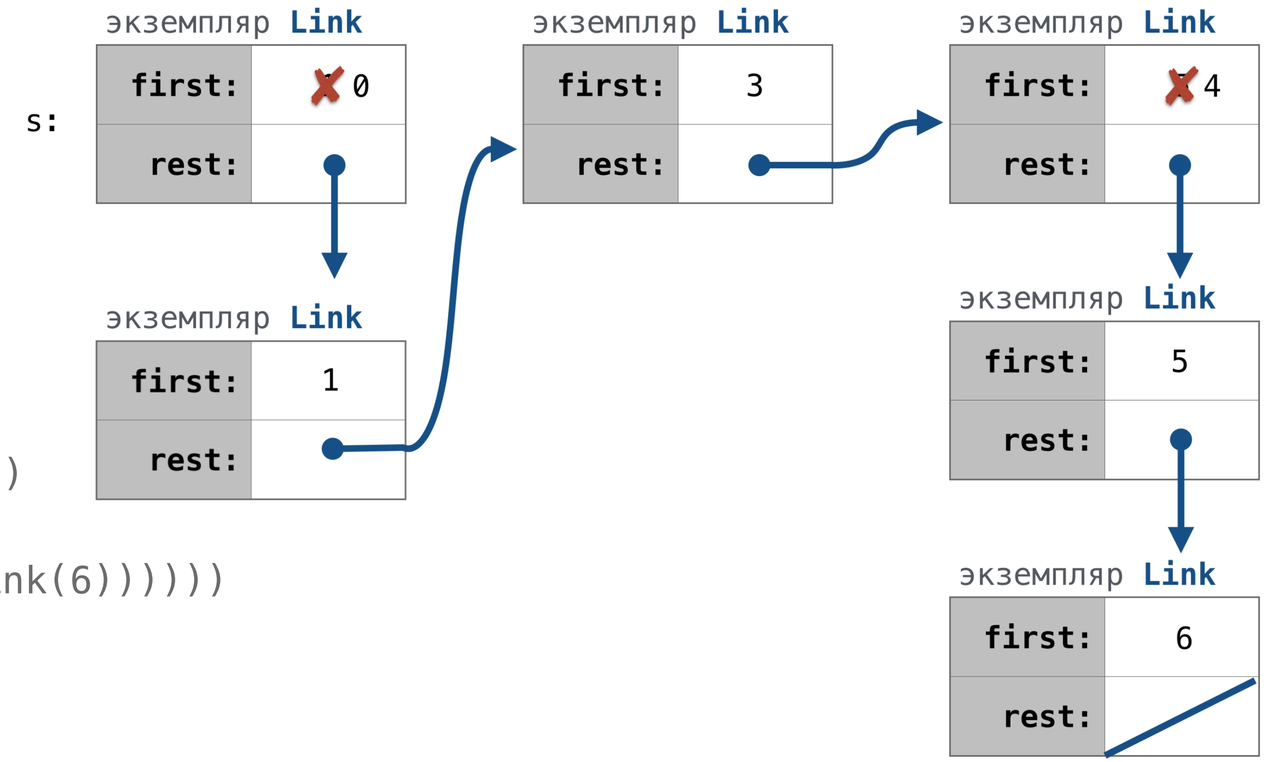
Добавление в множество на упорядоченных списках

```
def add(s, v):
    """Добавляет v во множество s.

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))
    """
    if empty(s):
        return Link(v)
    if s.first > v:
        s.first, s.rest = _____________________, _____________________
    elif s.first < v and empty(s.rest):
        s.rest = ____________________________________________________
    elif s.first < v:
        ____________________________________________________
    return s
```



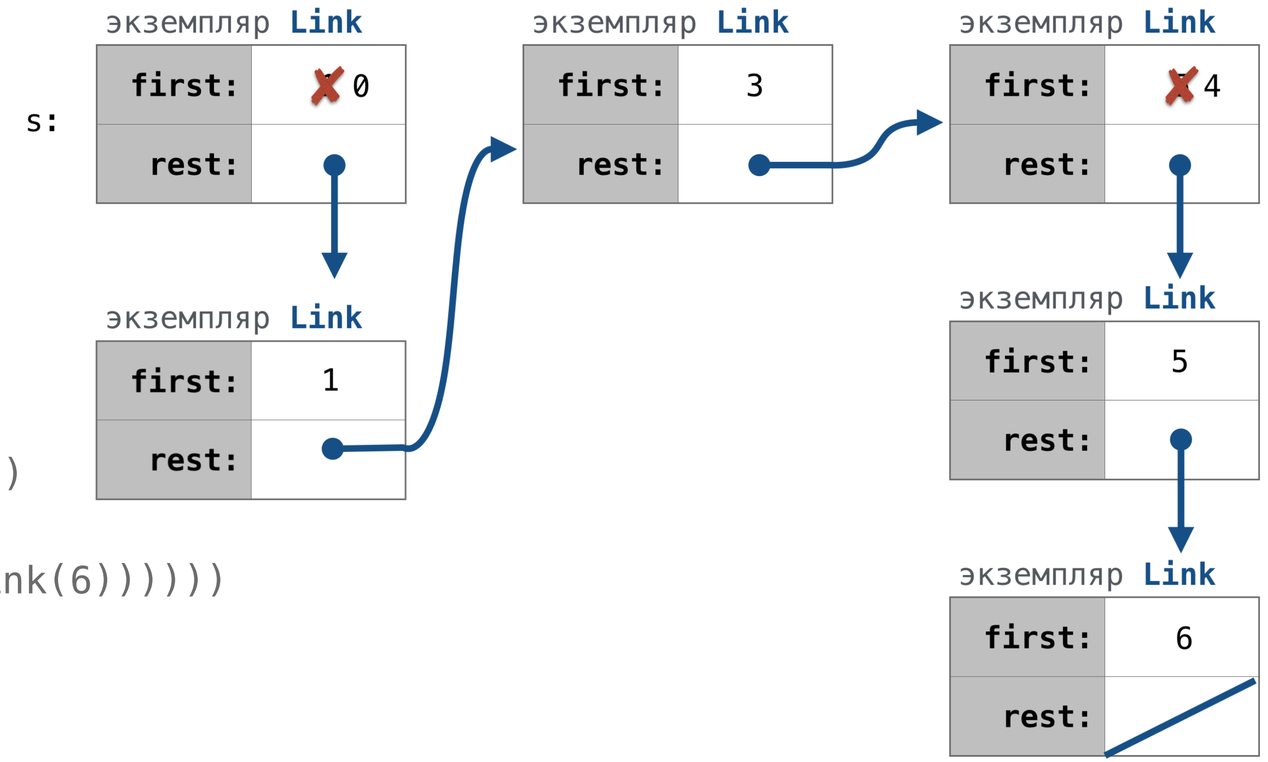
Добавление в множество на упорядоченных списках

```
def add(s, v):
    """Добавляет v во множество s.

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))
    """
    if empty(s):
        return Link(v)
    if s.first > v:
        s.first, s.rest = _____v_____, _____________________
    elif s.first < v and empty(s.rest):
        s.rest = _____________________________________________
    elif s.first < v:

        _____________________________________________
    return s
```



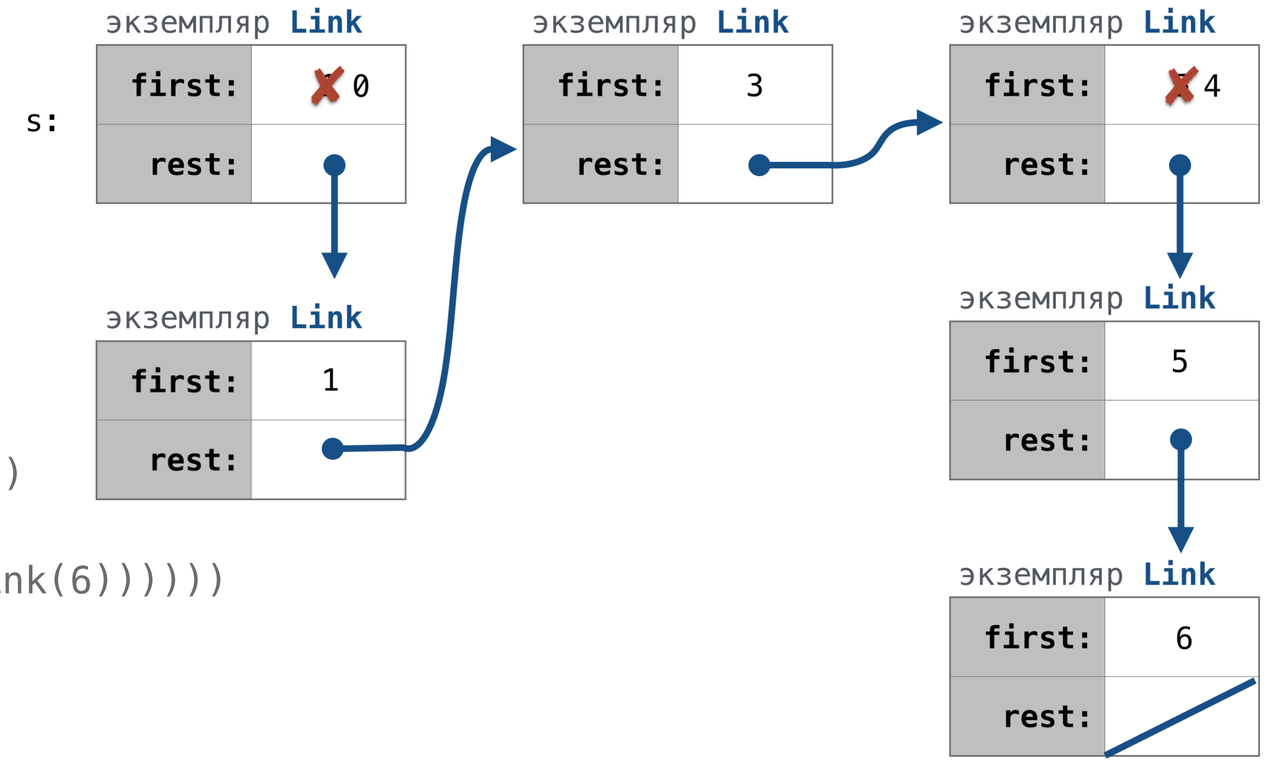
Добавление в множество на упорядоченных списках

```
def add(s, v):
    """Добавляет v во множество s.

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))
    """
    if empty(s):
        return Link(v)
    if s.first > v:
        s.first, s.rest = _____v_____, _Link(s.first, s.rest)_
    elif s.first < v and empty(s.rest):
        s.rest = ____________________________________________
    elif s.first < v:
        ____________________________________________
    return s
```



Добавление в множество на упорядоченных списках

```
def add(s, v):
    """Добавляет v во множество s.

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))
    """
    if empty(s):
        return Link(v)
    if s.first > v:
        s.first, s.rest = ____v____, Link(s.first, s.rest)
    elif s.first < v and empty(s.rest):
        s.rest = ____________Link(v, s.rest)____________
    elif s.first < v:
        ____________________________________________
    return s
```



Добавление в множество на упорядоченных списках

```
def add(s, v):
    """Добавляет v во множество s.

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))
    """
    if empty(s):
        return Link(v)
    if s.first > v:
        s.first, s.rest = ____v____, Link(s.first, s.rest)
    elif s.first < v and empty(s.rest):
        s.rest = ____Link(v, s.rest)____
    elif s.first < v:
        ____add(s.rest, v)____
    return s
```