Динамический массив
Списки
Деки

Филипп Воронов

Филипп Воронов

Teamlead, Поиск Mail.ru

Аккаунты в соц.сетях

@Филипп Воронов

План занятия

1. Динамический массив
2. Списки
3. Деки

Динамический массив

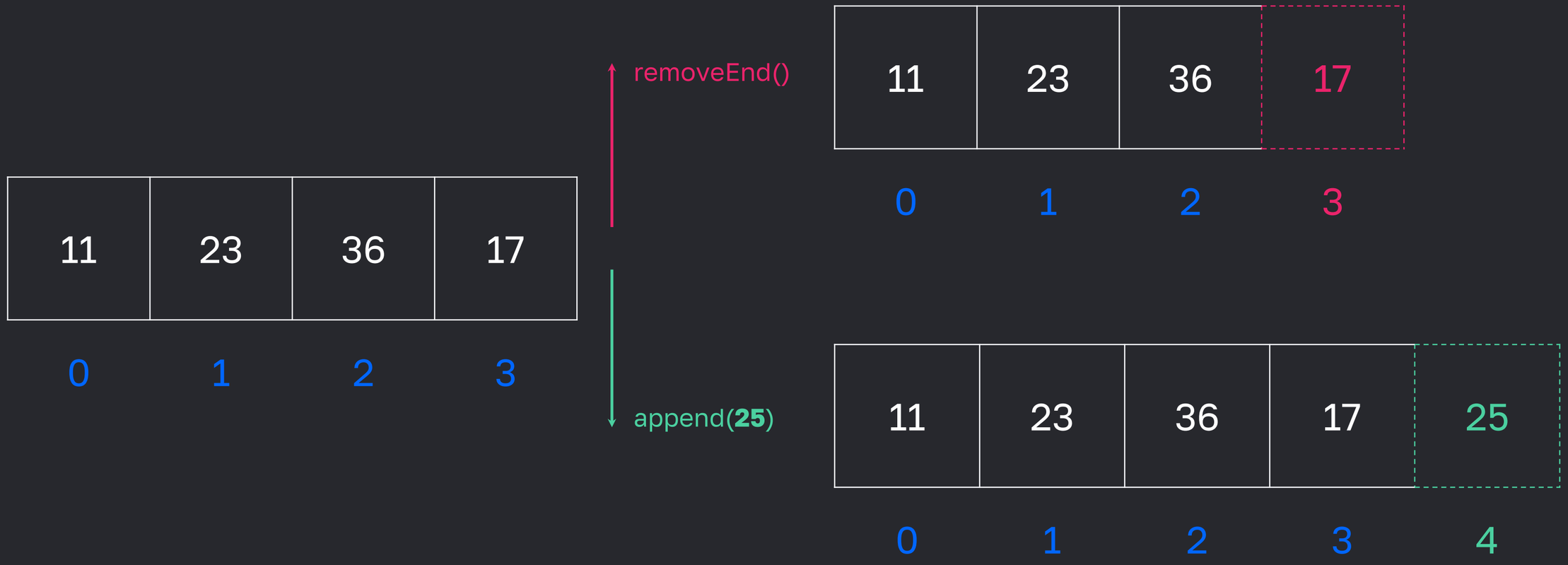
Особенности стандартного массива

Асимптотика доступа к ячейке константная.
Получение по индексу занимает O(1).

Менять размер массива нельзя.

Динамический массив

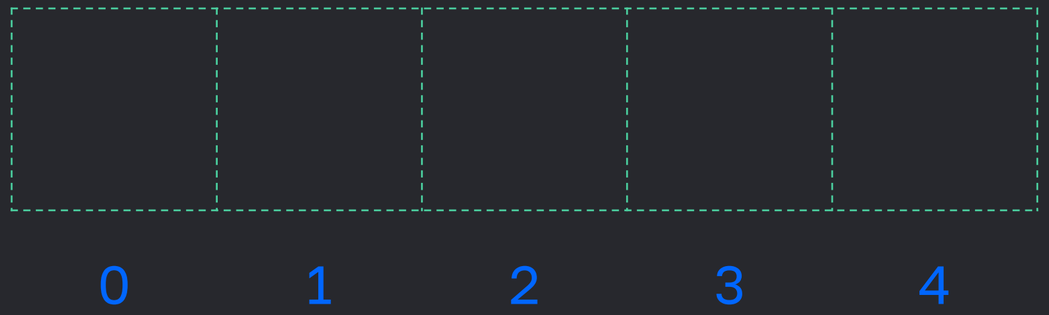
Нам бы хотелось иметь такие новые операции:



Наивная реализация динамического массива

Что делает append()?

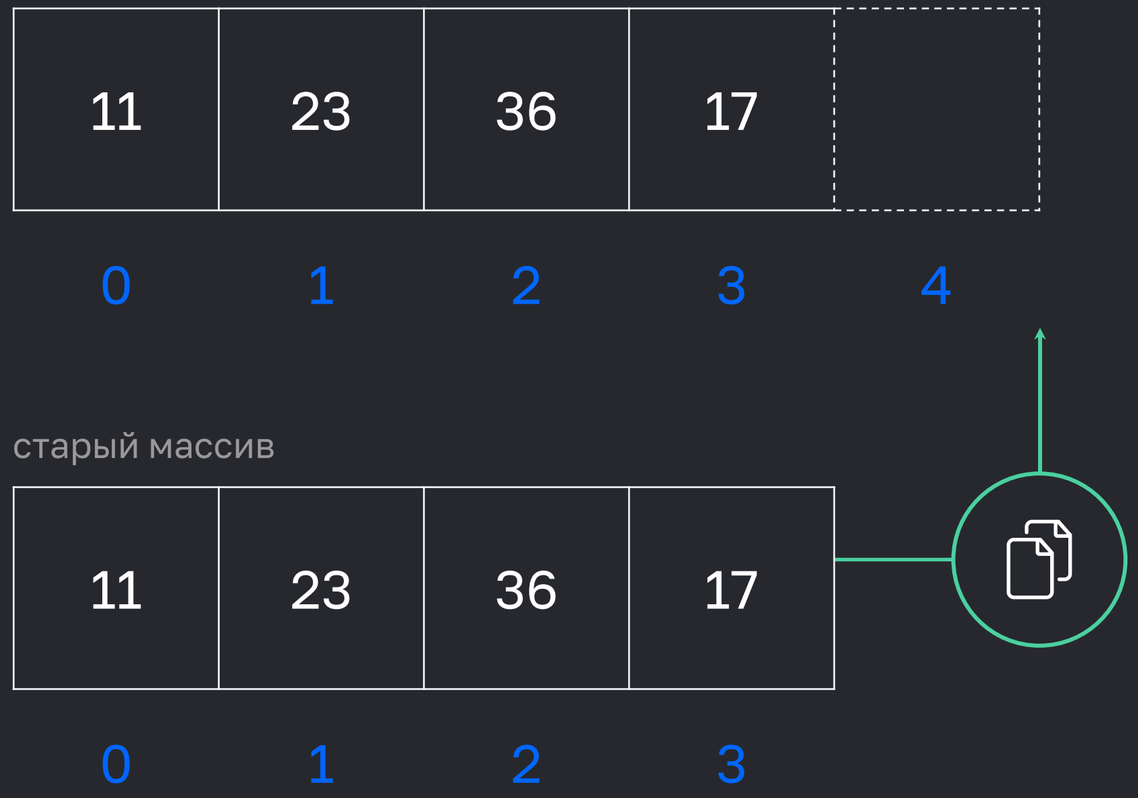
- Создаётся новый пустой массив



Наивная реализация динамического массива

Что делает append()?

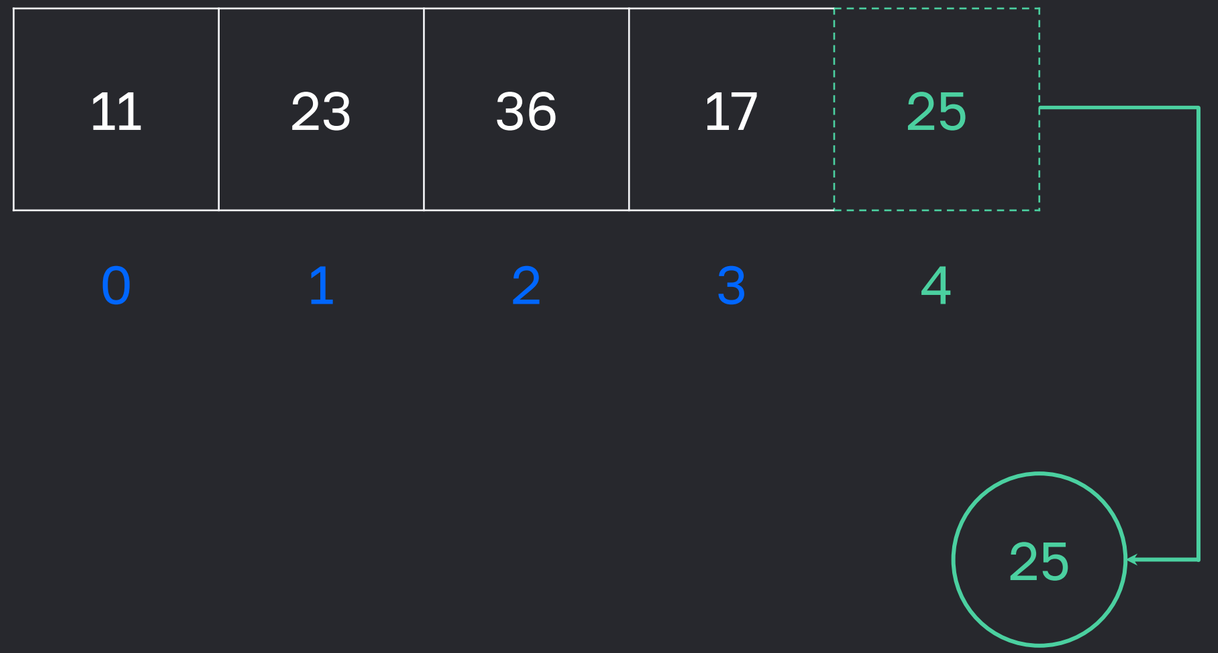
- Создаётся новый пустой массив
- Старый массив копируется в новый



Наивная реализация динамического массива

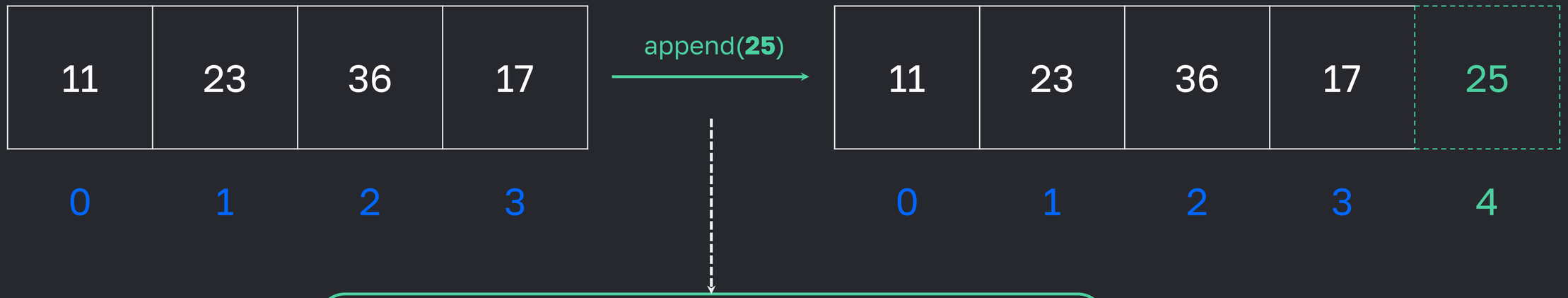
Что делает append()?

- Создаётся новый пустой массив
- Старый массив копируется в новый
- Новый элемент добавляется в конец



Наивная реализация динамического массива

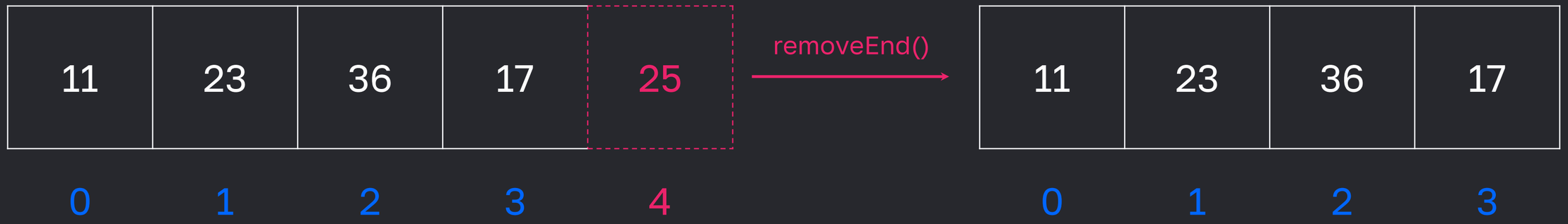
Наивный подход при добавлении нового элемента:



- Создаётся новый пустой массив
- Старый массив копируется в новый
- Новый элемент добавляется в конец

Наивная реализация динамического массива

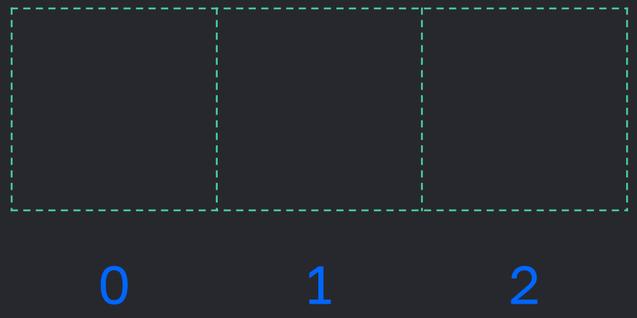
Наивный подход при удалении элемента:

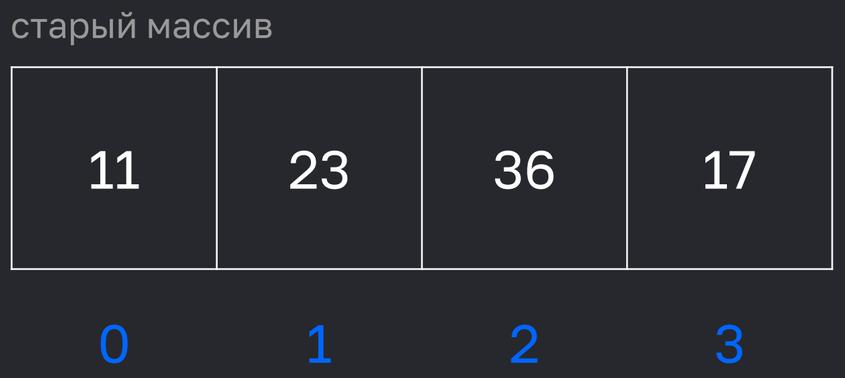


Наивная реализация динамического массива

Что делает removeEnd()?

- Создаётся новый пустой массив

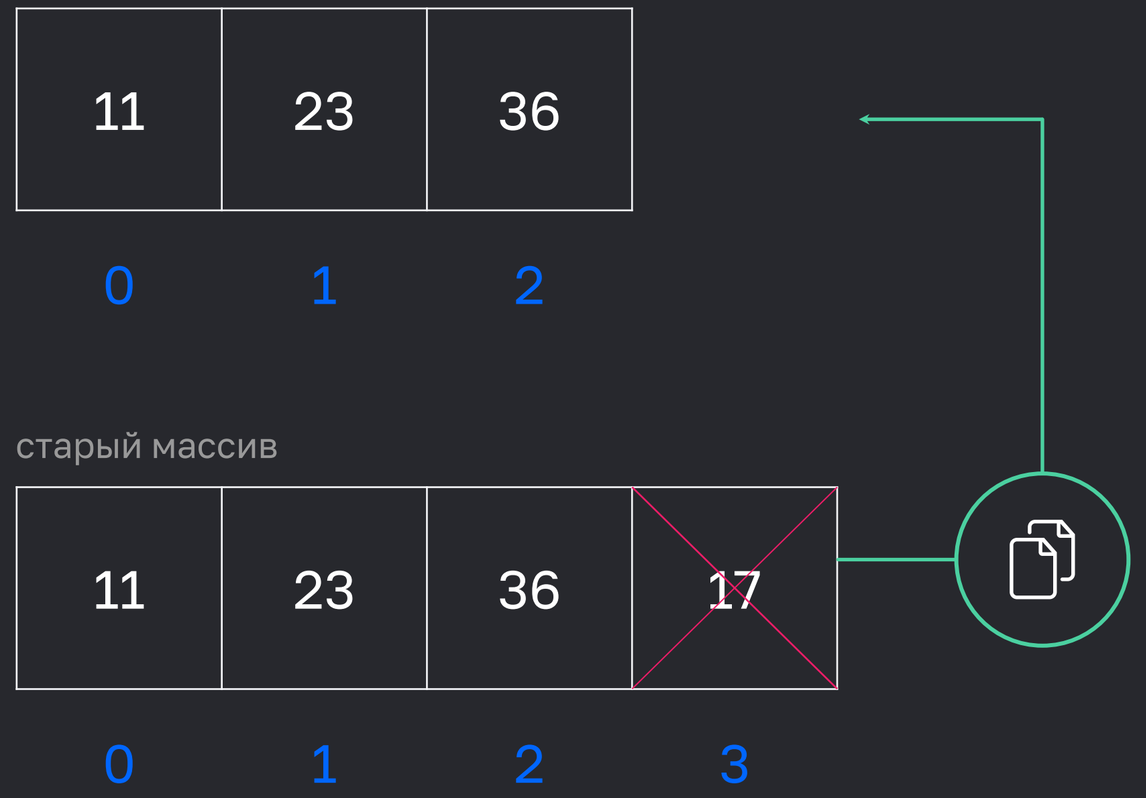




Наивная реализация динамического массива

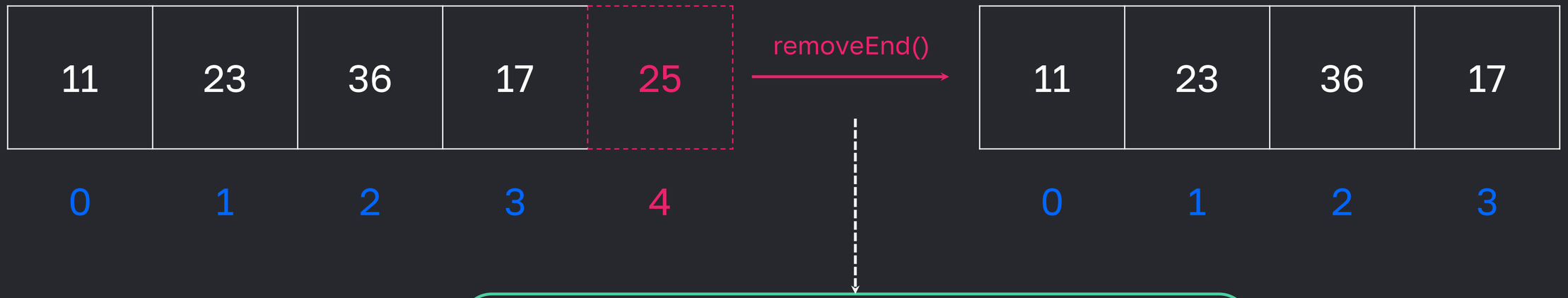
Что делает removeEnd()

- Создаётся новый пустой массив
- Старый массив копируется в новый (кроме последнего элемента)



Наивная реализация динамического массива

Наивный подход при удалении элемента:



- Создаётся новый пустой массив (его нельзя изменять)
- Старый массив копируется в новый (кроме последнего элемента)

Наивная реализация динамического массива

```
1  DynamicArray {
2   data: []
3
4   операция [i]:
5    return data[i]
6
7   append(e):
8    new_data = [длина(data)+1 нулей]
9    скопируем data в new_data
10   data = new_data
11   data[конец] = e
12
13  removeEnd():
14   new_data = [длина(data)-1 нулей]
15   скопируем data без посл. эл. в new_data
16   data = new_data
17 }
```

DynamicArray — название нового типа массива с операциями.
data — внутренний массив, внутри которого нужно всё хранить

Наивная реализация динамического массива

Алгосложность [i]. O(1) времени

```
1   DynamicArray {
2    data: []
3
4    операция [i]:
5     return data[i]
6
7    append(e):
8     new_data = [длина(data)+1 нулей]
9     скопируем data в new_data
10    data = new_data
11    data[конец] = e
12
13   removeEnd():
14    new_data = [длина(data)-1 нулей]
15    скопируем data без посл. эл. в new_data
16    data = new_data
17  }
```

Для получения доступа к элементу по индексу мы просто обращаемся к ячейке обычного внутреннего массива под таким же индексом.

Наивная реализация динамического массива

Алгосложность append. O(n) времени

```
1   DynamicArray {
2    data: []
3
4    операция [i]:
5     return data[i]
6
7    append(e):
8     new_data = [длина(data)+1 нулей]
9     скопируем data в new_data
10    data = new_data
11    data[конец] = e
12
13   removeEnd():
14    new_data = [длина(data)-1 нулей]
15    скопируем data без посл. эл. в new_data
16    data = new_data
17  }
```

Создание нового массива и копирование в него элементов из старого занимает O(n) времени (здесь и далее: n = длина массива data).

Наивная реализация динамического массива

Алгосложность removeEnd. O(n) времени

```
1   DynamicArray {
2    data: []
3
4    операция [i]:
5     return data[i]
6
7    append(e):
8     new_data = [длина(data)+1 нулей]
9     скопируем data в new_data
10    data = new_data
11    data[конец] = e
12
13   removeEnd():
14    new_data = [длина(data)-1 нулей]
15    скопируем data без посл. эл. в new_data
16    data = new_data
17  }
```

Создание нового массива и копирование в него элементов из старого занимает O(n) времени.

Наивная реализация динамического массива

```
1   DynamicArray {
2    data: []
3
4    операция [i]:
5     return data[i]
6
7    append(e):
8     new_data = [длина(data)+1 нулей]
9     скопируем data в new_data
10    data = new_data
11    data[конец] = e
12
13   removeEnd():
14    new_data = [длина(data)-1 нулей]
15    скопируем data без посл. эл. в new_data
16    data = new_data
17  }
```

И то, и то за O(n).

А можно ли быстрее?

Динамический массив с добавлением в конец

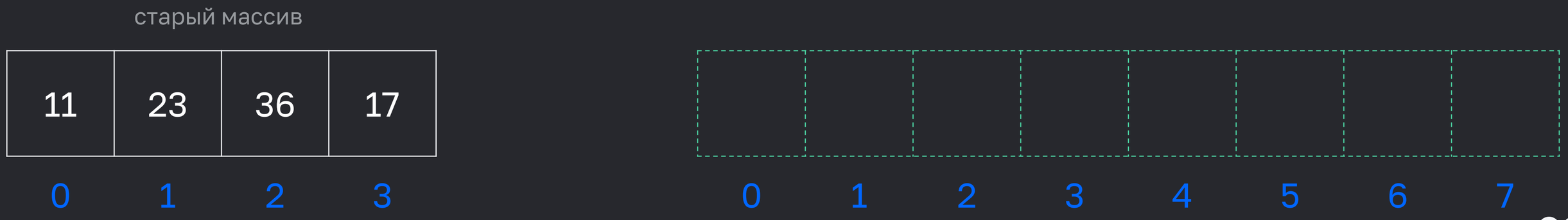
Улучшенный подход при добавлении нового элемента

Рассмотрим массив:

Динамический массив с добавлением в конец

Улучшенный подход добавления нового элемента

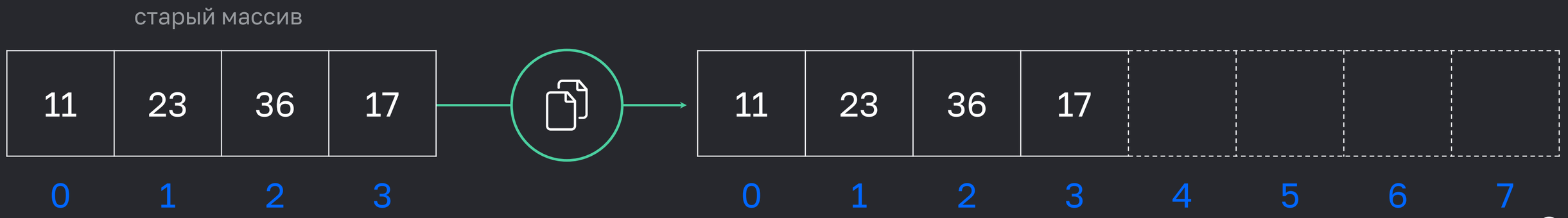
- создаём массив в 2 раза больше (про запас)



Динамический массив с добавлением в конец

Улучшенный подход добавления нового элемента.

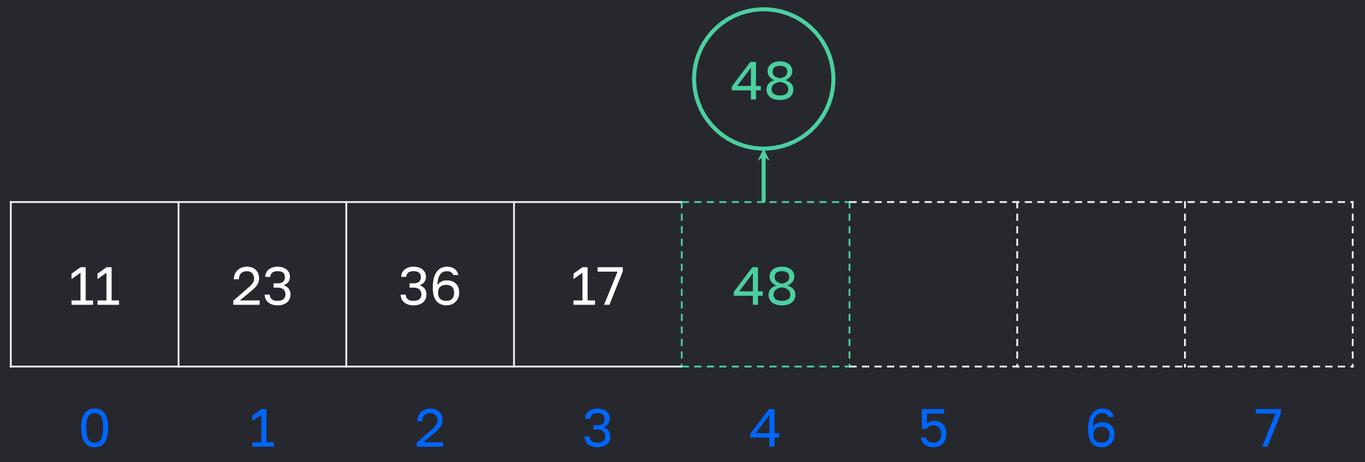
- создаём массив в 2 раза больше (про запас)
- старый массив копируется в новый



Динамический массив с добавлением в конец

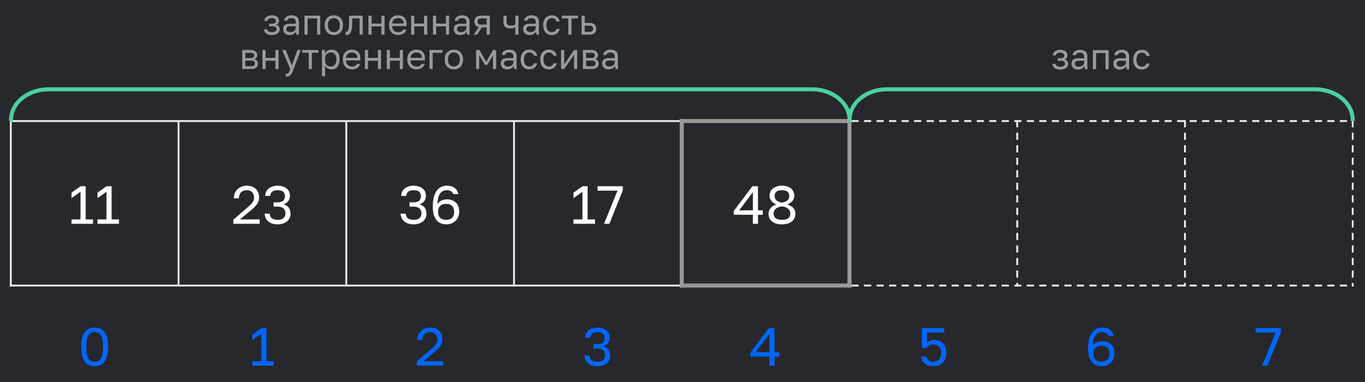
Улучшенный подход добавления нового элемента.

- создаём массив в 2 раза больше (про запас)
- старый массив копируется в новый
- добавляем элемент в первую свободную ячейку из запаса



Динамический массив с добавлением в конец

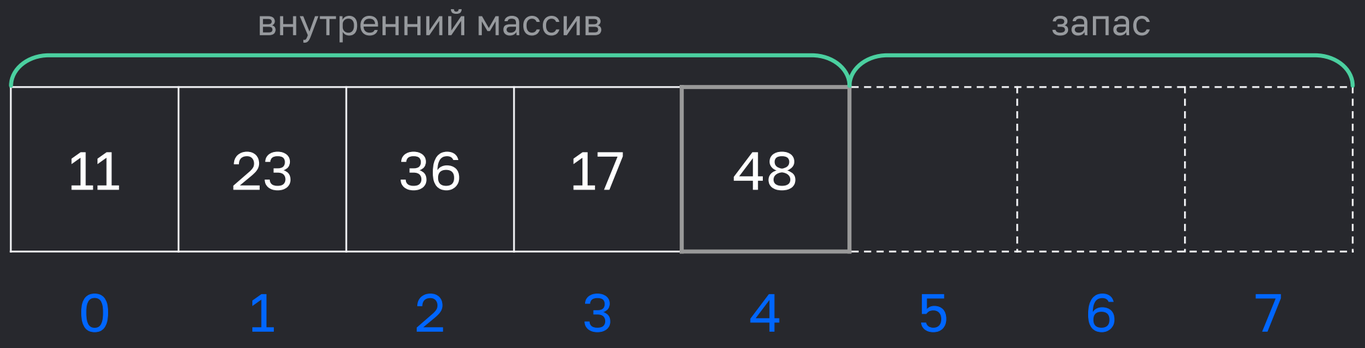
Наш динамический массив должен вести себя так, будто его длина всего 5, когда как его внутренний массив имеет длину 8 (3 элемента про запас).



Динамический массив с добавлением в конец

Улучшенный подход при добавлении нового элемента

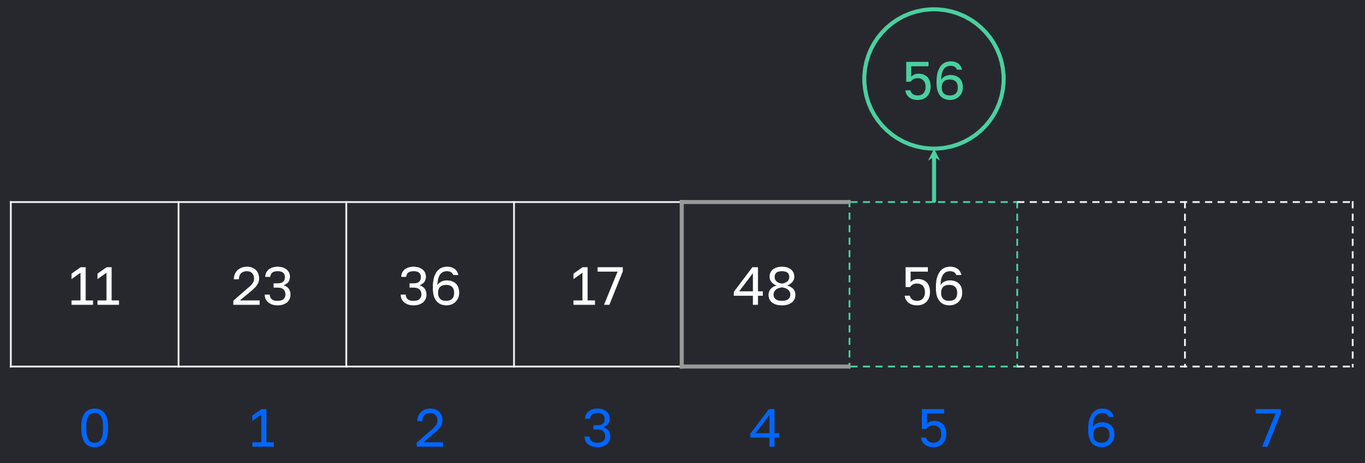
Рассмотрим массив с запасом



Динамический массив с добавлением в конец

Улучшенный подход при добавлении нового элемента

- добавляем в первую свободную ячейку из запаса (теперь копировать не надо)



Динамический массив с добавлением в конец

Улучшенный подход при добавлении нового элемента

```
1   DynamicArray {
2    data: [0]
3    last: -1
4
5    операция [i]:
6     if i <= last
7      return data[i]
8     else
9      ошибка, выход за границы дин. массива
10
11   append(e):
12    if last+1 = длина(data)
13     new_data = [2*длина(data) нулей]
14     скопируем data в new_data
15    data = new_data
16    data[last+1] = e
17    last += 1
18  }
```

Теперь нам надо хранить индекс последнего заполненного элемента в массиве

Динамический массив с добавлением в конец

Улучшенный подход при добавлении нового элемента

```
1   DynamicArray {
2    data: [0]
3    last: -1
4
5    операция [i]:
6     if i <= last
7      return data[i]
8     else
9      ошибка, выход за границы дин. массива
10
11   append(e):
12    if last+1 = длина(data)
13     new_data = [2*длина(data) нулей]
14     скопируем data в new_data
15    data = new_data
16    data[last+1] = e
17    last += 1
18  }
```

Ничего не изменилось, только теперь надо проверять,
не вышли ли мы за границы заполненной части массива

Динамический массив с добавлением в конец

Улучшенный подход при добавлении нового элемента

```
1   DynamicArray {
2    data: [0]
3    last: -1
4
5    операция [i]:
6     if i <= last
7      return data[i]
8     else
9      ошибка, выход за границы дин. массива
10
11   append(e):
12    if last+1 = длина(data)
13     new_data = [2*длина(data) нулей]
14     скопируем data в new_data
15    data = new_data
16    data[last+1] = e
17    last += 1
18  }
```

Ничего не изменилось, только теперь надо проверять,
не вышли ли мы за границы заполненной части массива

Динамический массив с добавлением в конец

? Как вы считаете, есть ли отличия в расходе памяти при добавлении нового элемента, используя наивный и улучшенный подходы?

Динамический массив с добавлением в конец

? Как вы считаете, есть ли отличия в расходе памяти при добавлении нового элемента, используя наивный и улучшенный подходы?

Хоть память и осталась O(n), теперь у нас O(n) «лишних» ячеек, выделенных «про запас».
По памяти стало хуже, но терпимо.

Динамический массив с добавлением в конец

Ключевое отличие: в среднем операция добавления стала O(1), хоть и в худшем случае O(n).

Такая асимптотика называется амортизационной, а процедура её подсчёта — амортизационным анализом.

Использование армотизационной константности у динамического массива

Динамический массив. Задача

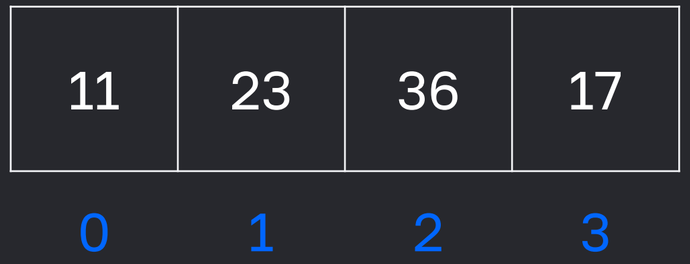
Рассмотрим задачу: нужно создать пустой массив
и последовательно добавить в него n чисел.
Какова асимптотика такой программы?

Разберем наивный подход добавления элемента.

Динамический массив. Задача

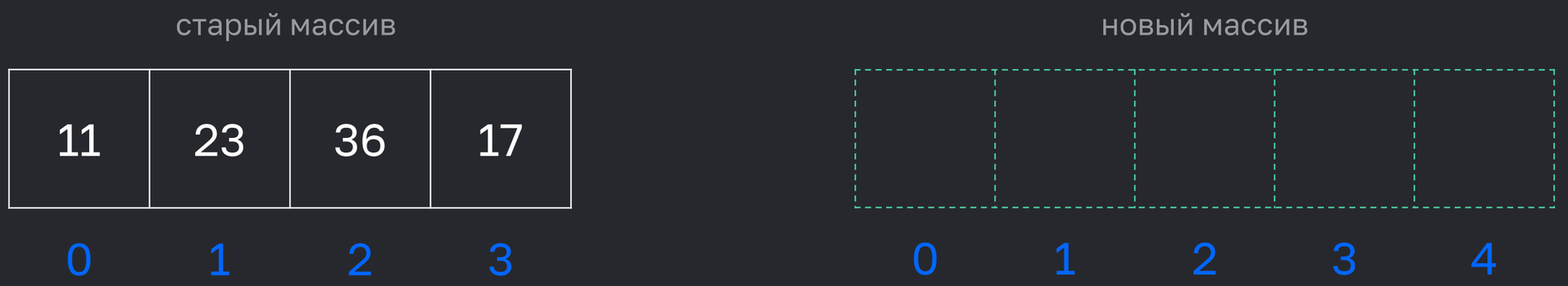
Предположим, что мы уже добавили 4 элемента
и хотим добавить 5-й элемент к старому массиву.

старый массив



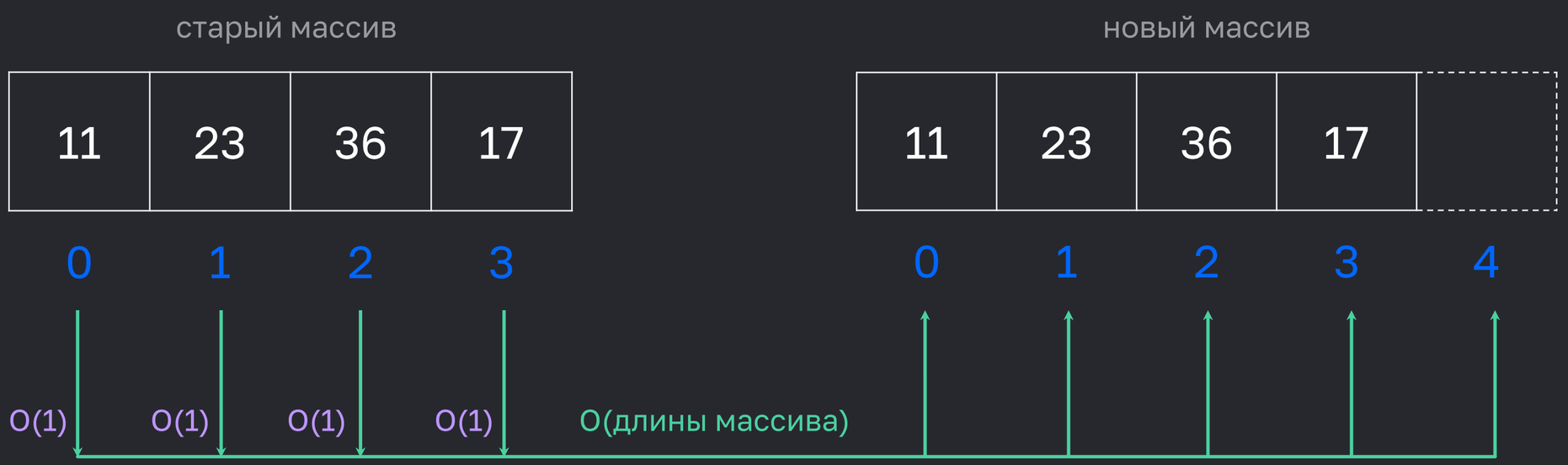
Динамический массив. Задача

1. создается новый массив



Динамический массив. Задача

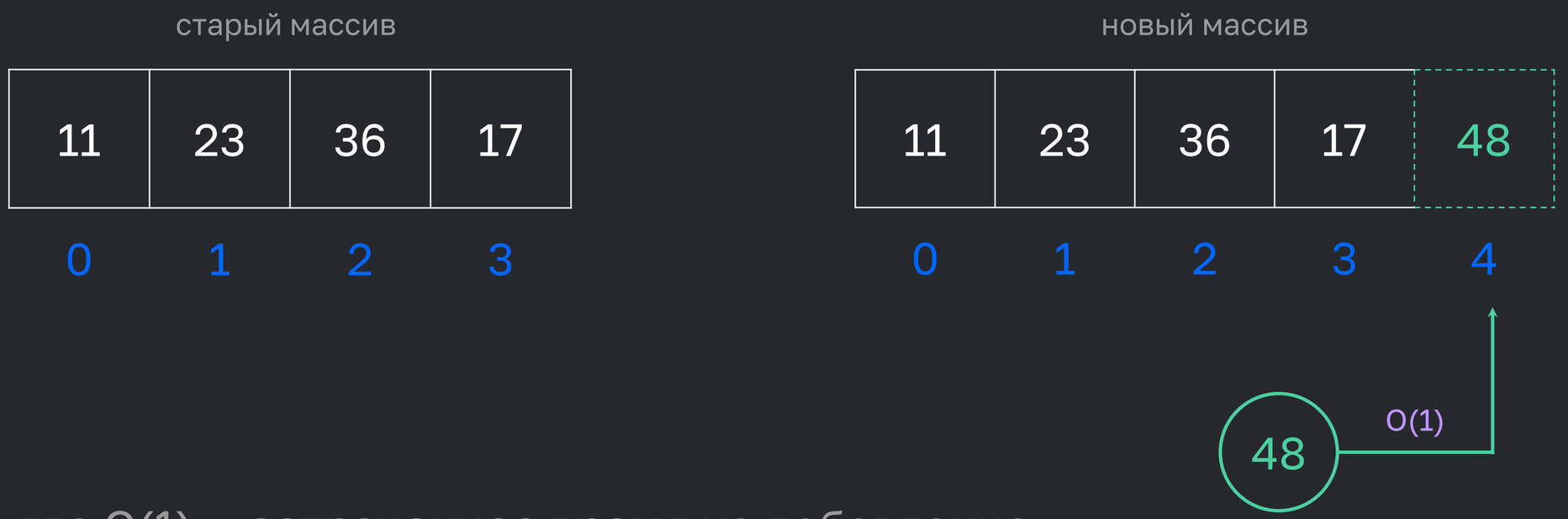
2 копируется старый массив в новый



O(длина массива) — **затраченное время**
на перенос старого массива в новый

Динамический массив. Задача

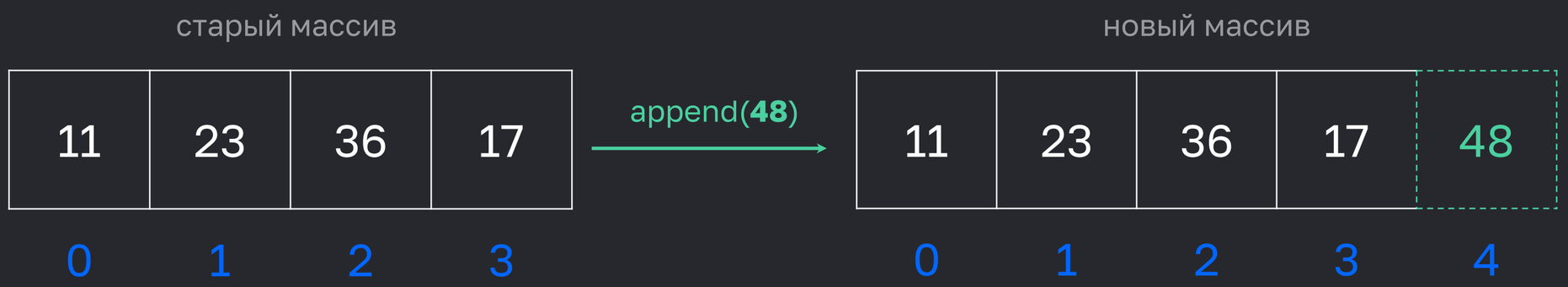
3 новый элемент добавляется в конец массива



где O(1) — **затраченное время** на добавление нового элемента в конец массива

Динамический массив. Задача

Вставка в конец наивного динамического массива
всегда занимает O(длина массива) времени.



Динамический массив. Задача

Рассмотрим задачу: нужно создать пустой массив
и последовательно добавить в него n чисел.
Какова асимптотика такой программы?

Ответ: при наивном подходе — последовательная
вставка n элементов в конец займет $O(n^2)$ времени.

Динамический массив. Задача

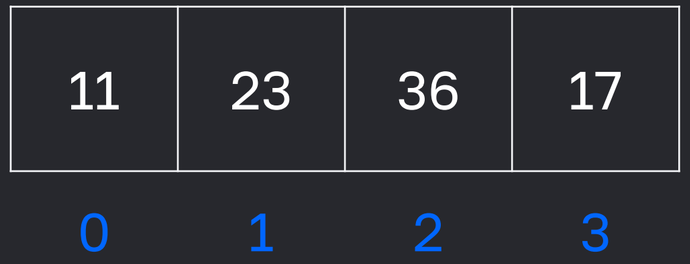
Рассмотрим задачу: нужно создать пустой массив
и последовательно добавить в него n чисел.
Какова асимптотика такой программы?

Разберем улучшенный подход добавления элемента.

Динамический массив. Задача

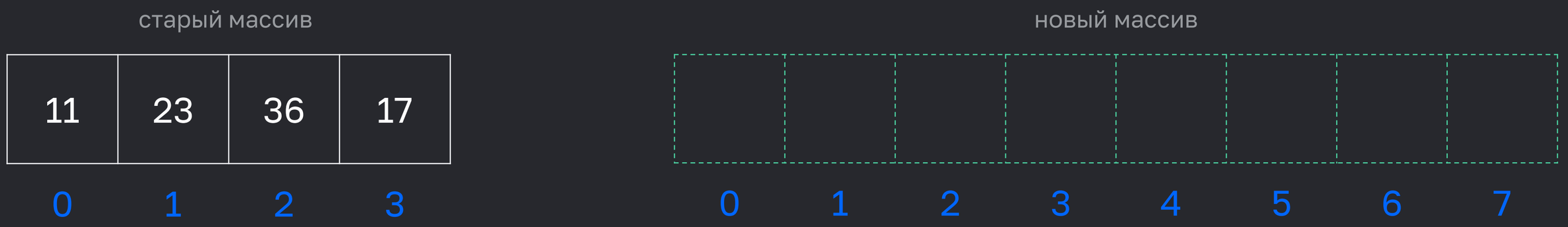
Также предположим, что мы уже добавили 4 элемента
и хотим добавить 5-й элемент к старому массиву.

старый массив



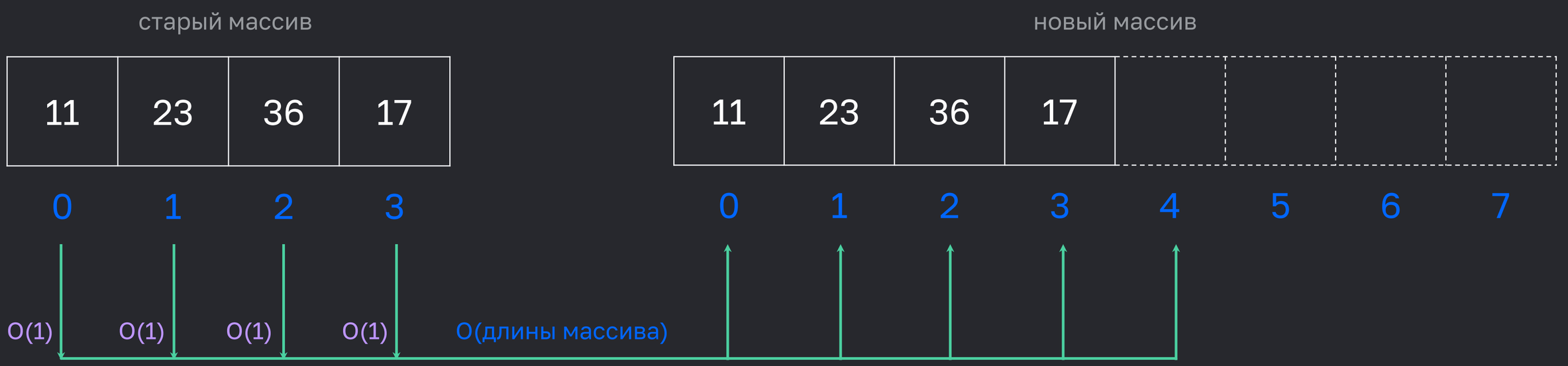
Динамический массив. Задача

1. создаем новый массив в 2 раза больше



Динамический массив. Задача

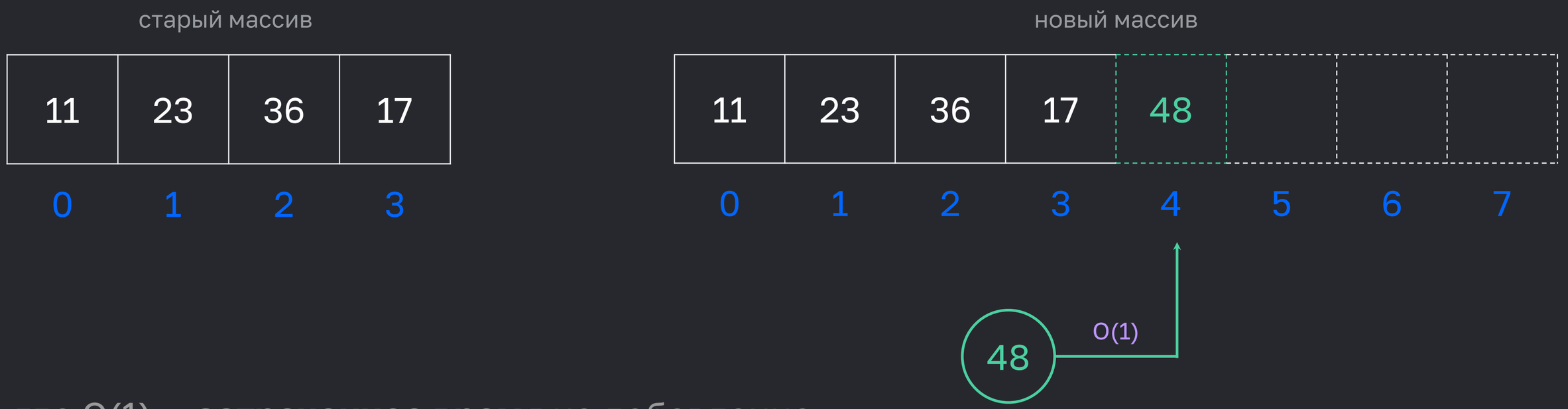
2 копируется старый массив в новый



O(длина массива) — **затраченное время**
на перенос старого массива в новый

Динамический массив. Задача

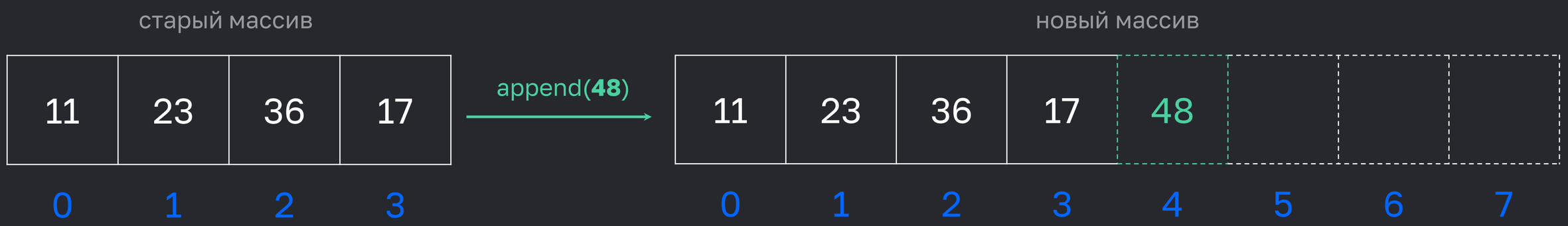
3 новый элемент добавляется в свободную ячейку



где O(1) — **затраченное время** на добавление нового элемента в свободную ячейку

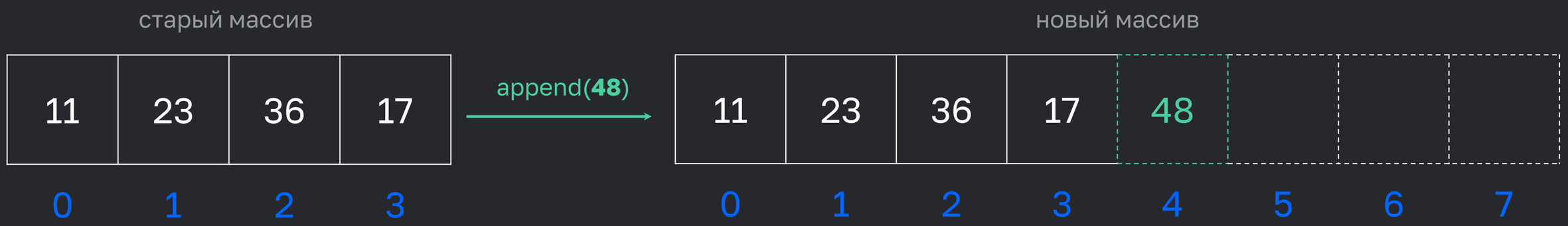
Динамический массив. Задача

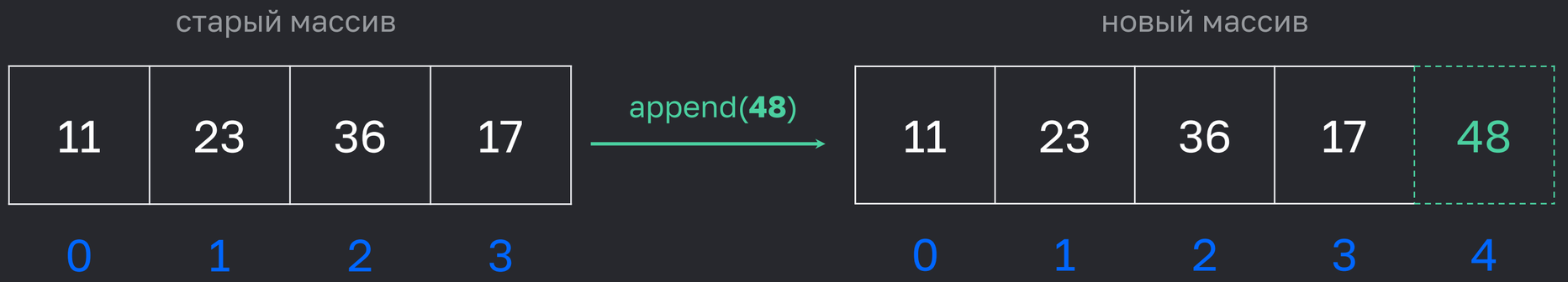
Вставка в конец улучшенного динамического массива,
не имеющего запаса, занимает O(длина массива) времени.



Динамический массив. Задача

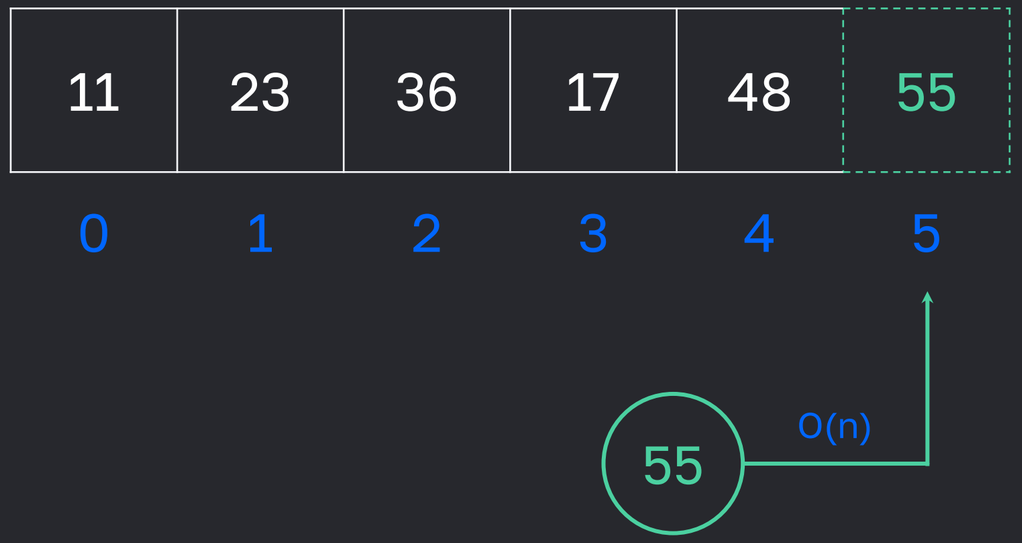
Тогда в чем отличия двух подходов в нашей задаче,
если оба заняли по времени O(длина массива)?





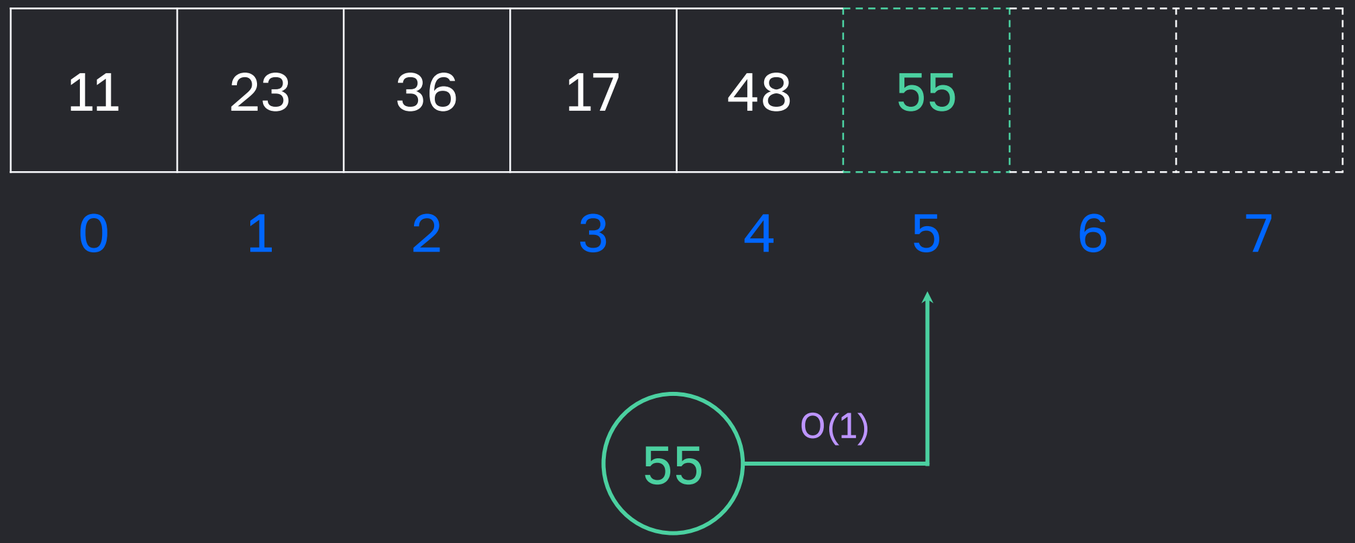
Динамический массив. Задача

При добавлении 6-го элемента, используя наивный подход, всегда затрачивается O(длина массива) времени.



Динамический массив. Задача

При добавлении 6-го элемента, используя улучшенный подход, затрачивается O(1) времени.



Динамический массив. Задача

Подход	Затраченное время при добавлении нового элемента										
	Длина массива (n)										
	1	2	3	4	5	6	7	8	9	10	11
наивный	O(1)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n	O(n)
улучшенный	O(1)	O(n)	O(n)	O(1)	O(n)	O(1)	O(1)	O(1)	O(n)	O(1)	O(1)

n — длина массива

O(1) — новый элемент добавляется в свободную ячейку из запаса

Динамический массив. Задача

Подход	Затраченное время при добавлении нового элемента										
	Длина массива (n)										
	1	2	3	4	5	6	7	8	9	10	11
наивный	O(1)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n	O(n)
улучшенный	O(1)	O(n)	O(n)	O(1)	O(n)	O(1)	O(1)	O(1)	O(n)	O(1)	O(1)

Вставка в конец улучшенного динамического массива иногда бывает за O(длина массива), но в среднем она за O(1).

Динамический массив. Задача

Рассмотрим задачу: нужно создать пустой массив
и последовательно добавить в него n чисел.
Какова асимптотика такой программы?

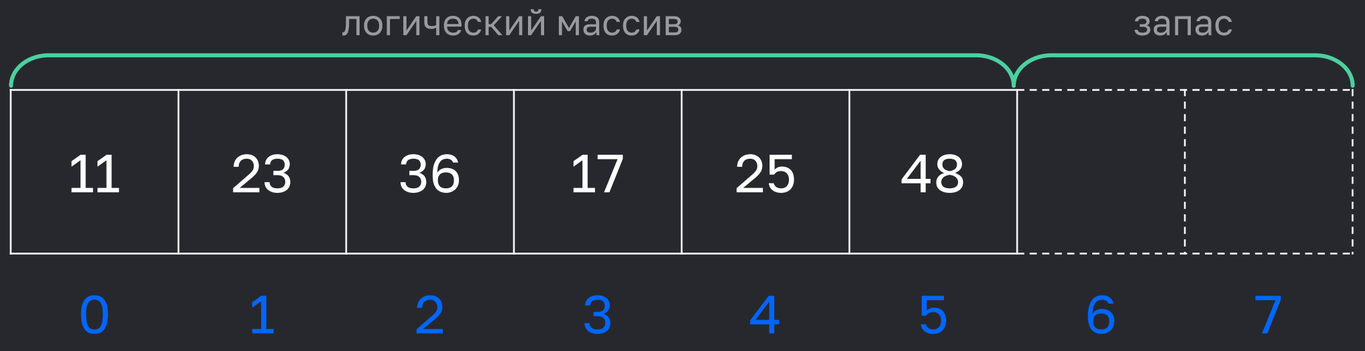
Ответ: при улучшенном подходе — последовательная вставка
n элементов в конец займет O(n) времени, поскольку новый
массив будет создаваться не на каждом шаге, а только
на длинах, равных степеням двойки.

Динамический массив с удалением с конца

Динамический массив с удалением с конца

Улучшенный подход при удалении элемента.

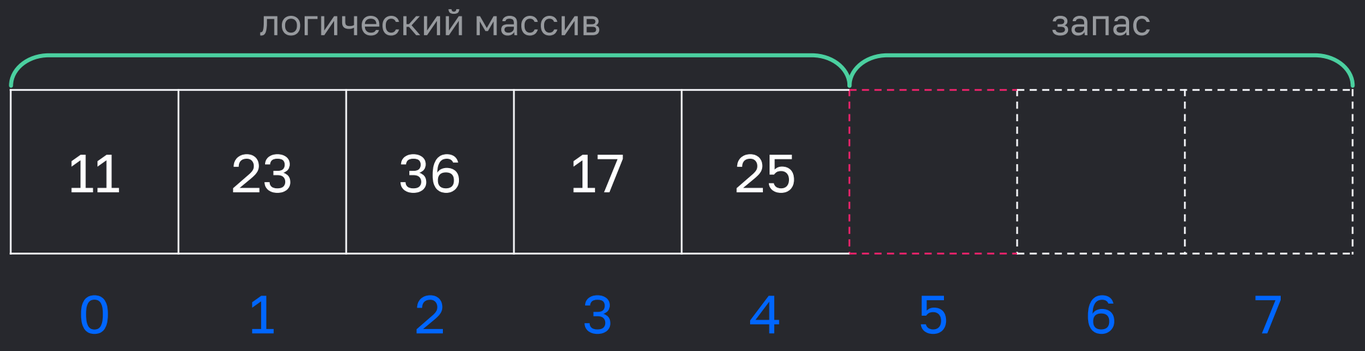
Рассмотрим массив, пока запас меньше половины.



Динамический массив с удалением с конца

Улучшенный подход при удалении элемента.

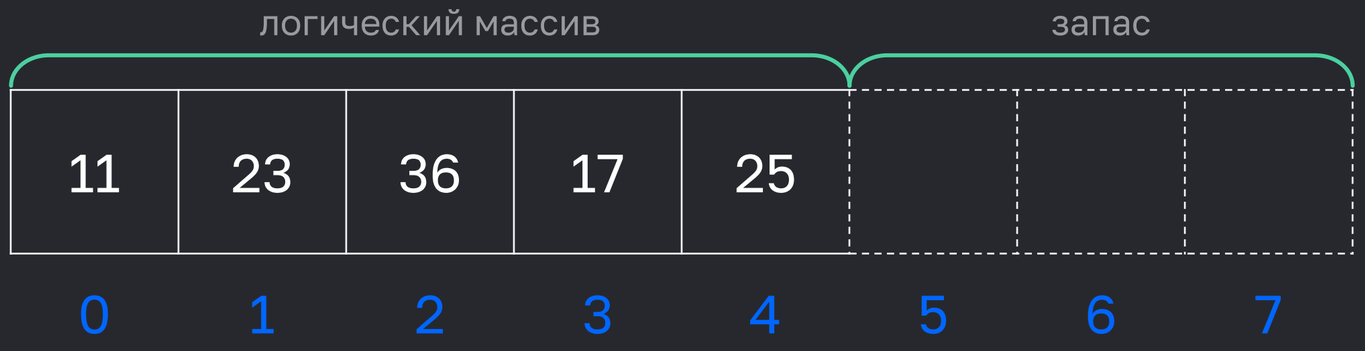
- считаем последнюю ячейку частью запаса



Динамический массив с удалением с конца

Улучшенный подход при удалении элемента.

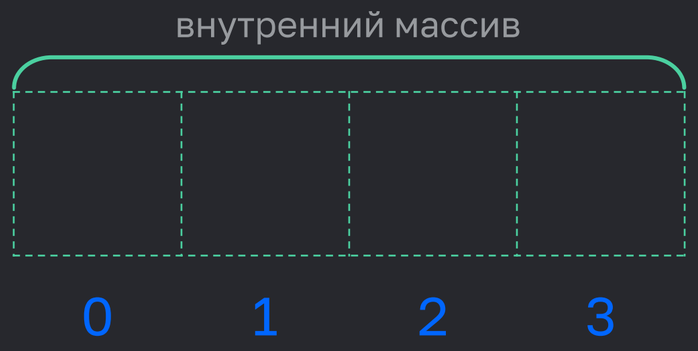
Рассмотрим массив, в котором после удаления запас станет половиной.



Динамический массив с удалением с конца

Улучшенный подход при удалении элемента.

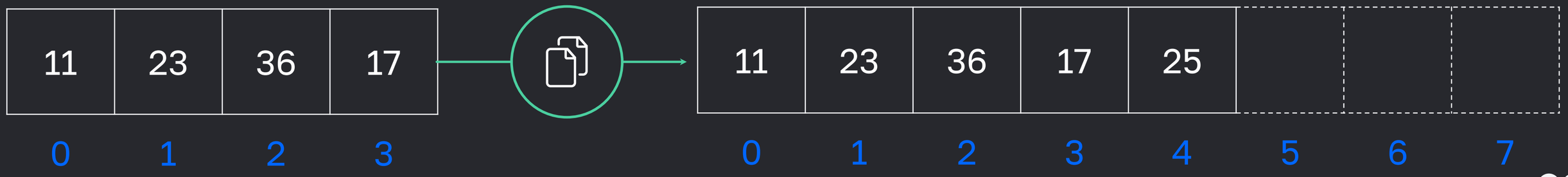
- создаём новый массив вдвое меньше



Динамический массив с удалением с конца

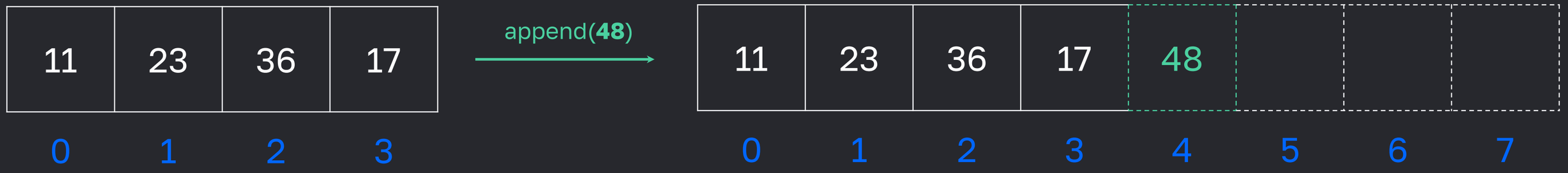
Улучшенный подход при удалении элемента.

- создаём новый массив вдвое меньше
- копируем в него всё без удаляемого элемента



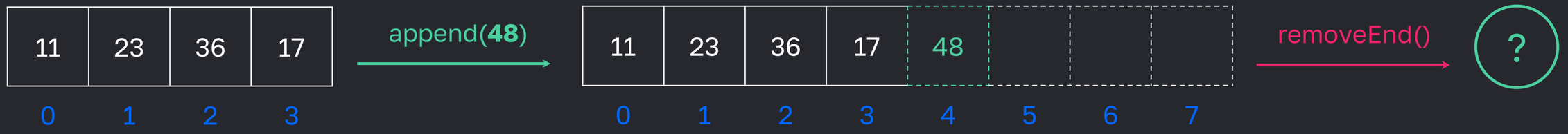
Динамический массив: проблема удаления

Посмотрим на динамический массив сразу после операции append, при которой внутренний массив расширился в два раза.



Динамический массив: проблема удаления

Что произойдет с таким массивом, если пользователь вызовет removeEnd()?




```
DynamicArray {
  ...

  removeEnd():
    last -= 1
    if last < длина(data) / 2
      new_data = [длина(data) / 2 нулей]
      скопируем data в new_data
      data = new_data
}
```

Динамический массив: проблема удаления

Результатом вызова removeEnd будет:

Динамический массив: проблема удаления

Что теперь будет, если на получившемся массиве вызвать append(48)?

Динамический массив: проблема удаления

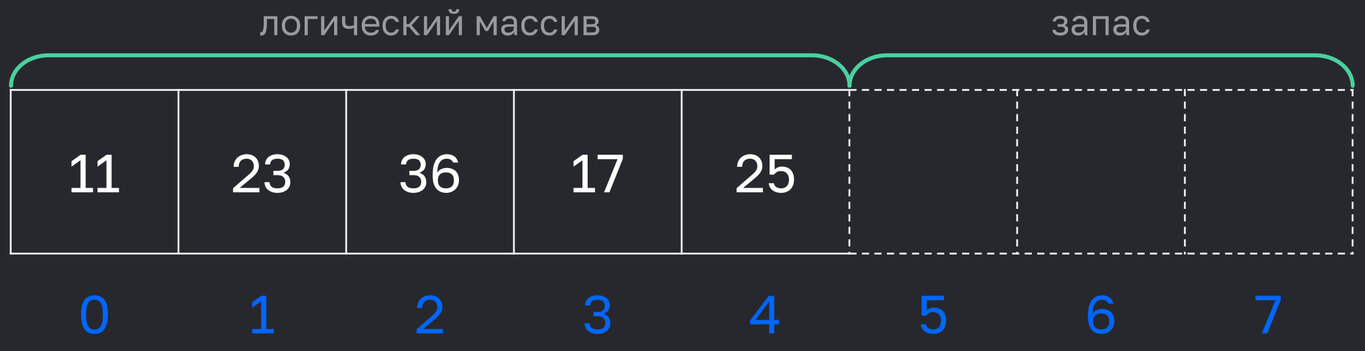
Почему такой подход считается плохим?

Потому что повторяя чередование append и removeEnd, мы будем попадать каждый раз на линейное расширение или уменьшение внутреннего массива, что превратит нашу амортизацированную (среднюю) асимптотику append и removeEnd из O(1) в O(n).

Динамический массив с удалением с конца

Улучшенный подход при удалении элемента.

Рассмотрим массив, в котором после удаления запас станет половиной.



Динамический массив

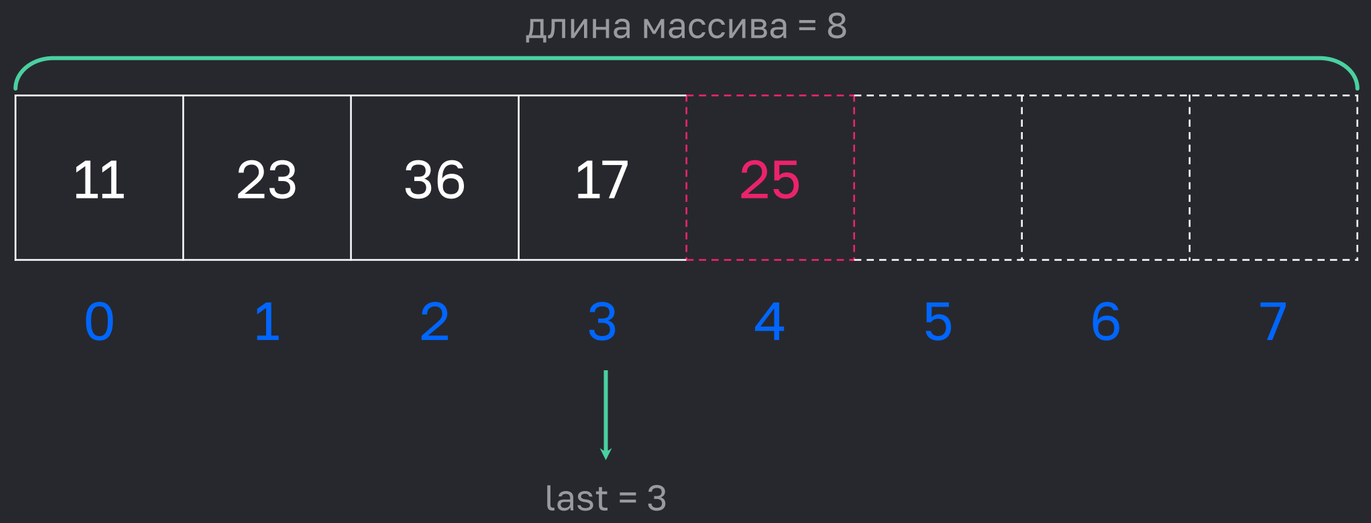
Решение этой проблемы: уменьшать массив
не тогда, когда он заполнен на половину,

а когда заполнен меньше чем на треть

Динамический массив

Если последний индекс физического массива (за исключением удаляемого) был меньше, чем длина массива / 2, то мы уменьшали этот массив в 2 раза.

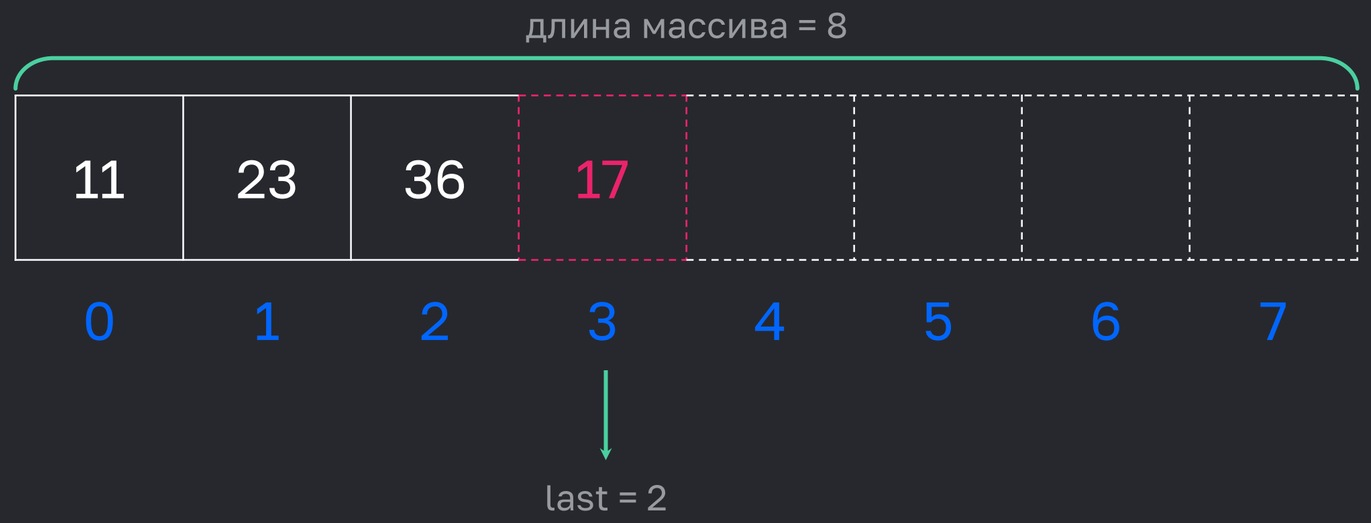
```
1   DynamicArray {
2    ...
3
4    removeEnd():
5     last -= 1
6     if last < длина(data) / 2
7       new_data = [длина(data) / 2 нулей]
8       скопируем data в new_data
9       data = new_data
10  }
```



Динамический массив

Теперь, если последний индекс физического массива (кроме удаляемого) будет меньше, чем длина массива / 3, то тогда мы уменьшим этот массив

```
1   DynamicArray {
2     ...
3
4     removeEnd():
5       last -= 1
6       if last < длина(data) / 3
7         new_data = [длина(data) / 3 нулей]
8         скопируем data в new_data
9         data = new_data
10  }
```



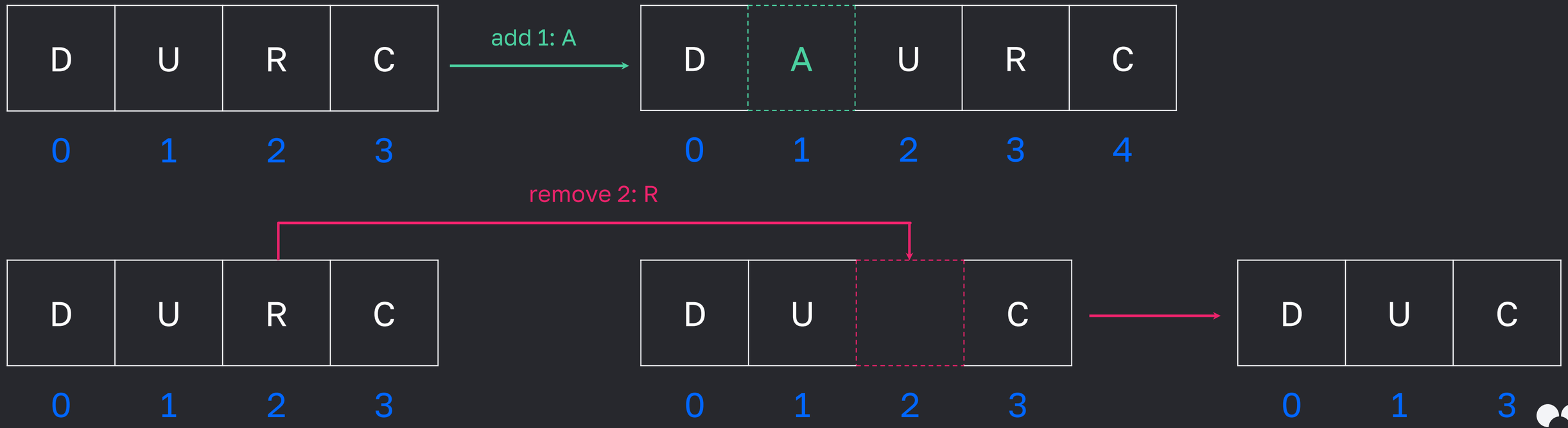
Динамический массив

Почему такой подход лучше?

Теперь в худшем случае между расширением и уменьшением внутреннего массива пройдёт (½ - ⅓)n операций, что позволит нам считать обе операции амортизировано (в среднем) O(1).

Динамический массив

Вставка и удаление произвольной позиции всё равно
будет линейной из-за необходимости сдвинуть
все элементы справа от этой позиции



Динамический массив. Итоги

Мы придумали структуру данных, которая для обычных операций над массивами работает так же быстро, как и обычный массив, но дополнительно к ним умеет добавлять элементы в конец и удалять с конца в среднем за O(1), а не O(длина массива).

Списки

Списки

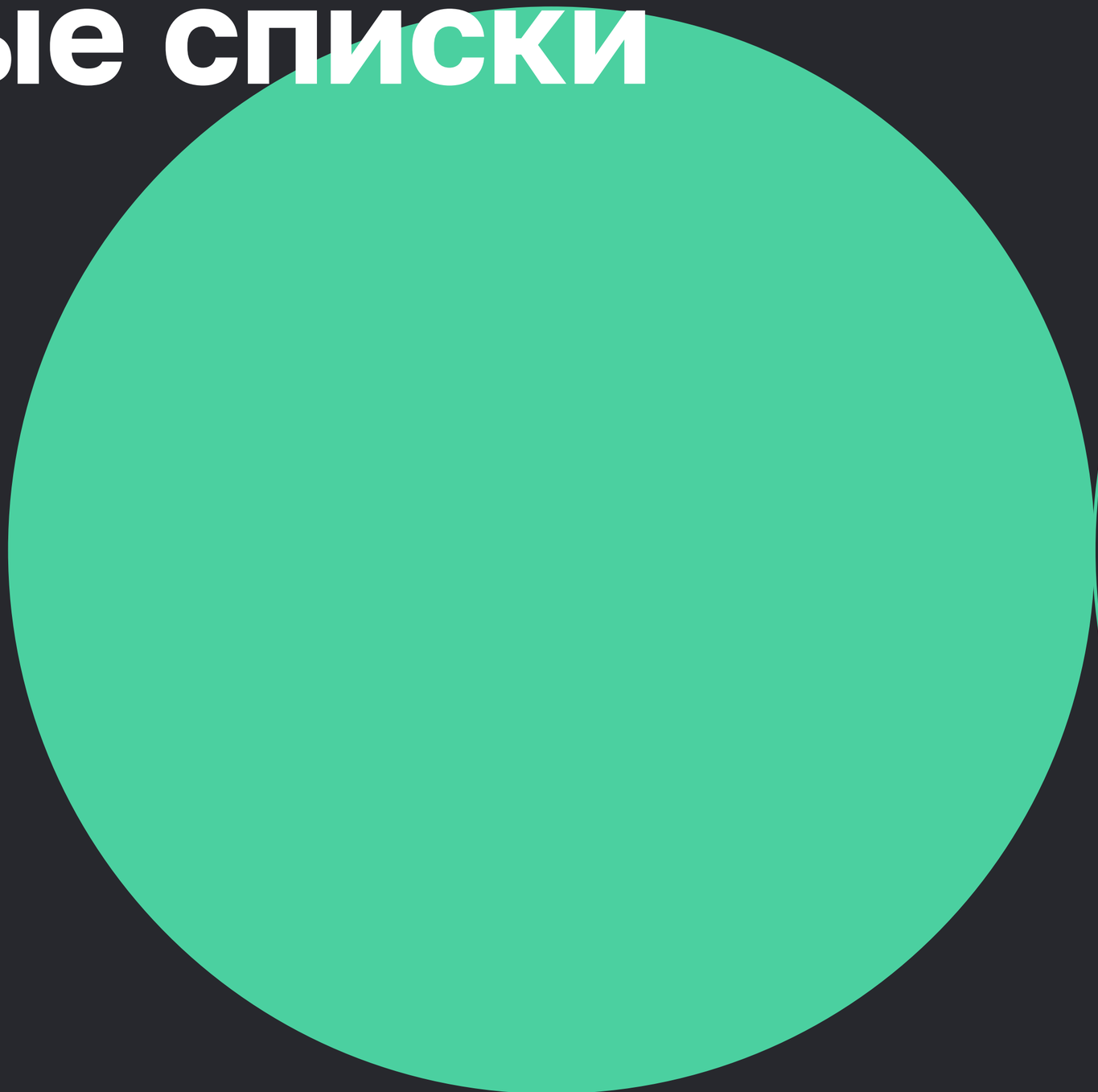
Что такое списки?

Это семейство структур данных, которые умеют делать как минимум следующие операции:

Операция	Значение
add	добавляет элементы в список
remove	удаляет элементы из списка
get	достает элемент по номеру из списка
set	меняет элемент по номеру из списка на новый

Списки. Пример

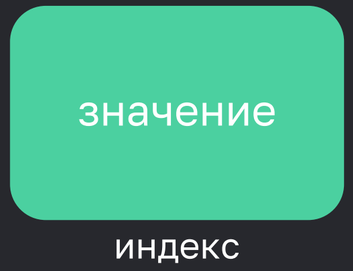
Связные списки



Связный список. Узел

Давайте придумаем такую обертку
над элементом списка, в которой будет:

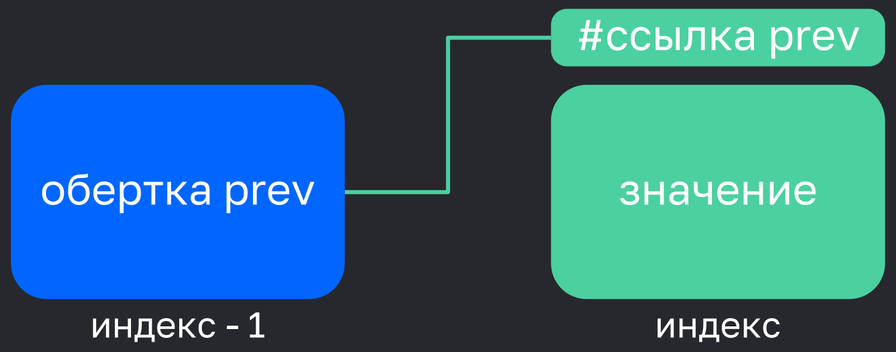
- значение элемента



Связный список. Узел

Давайте придумаем такую обертку
над элементом списка, в которой будет:

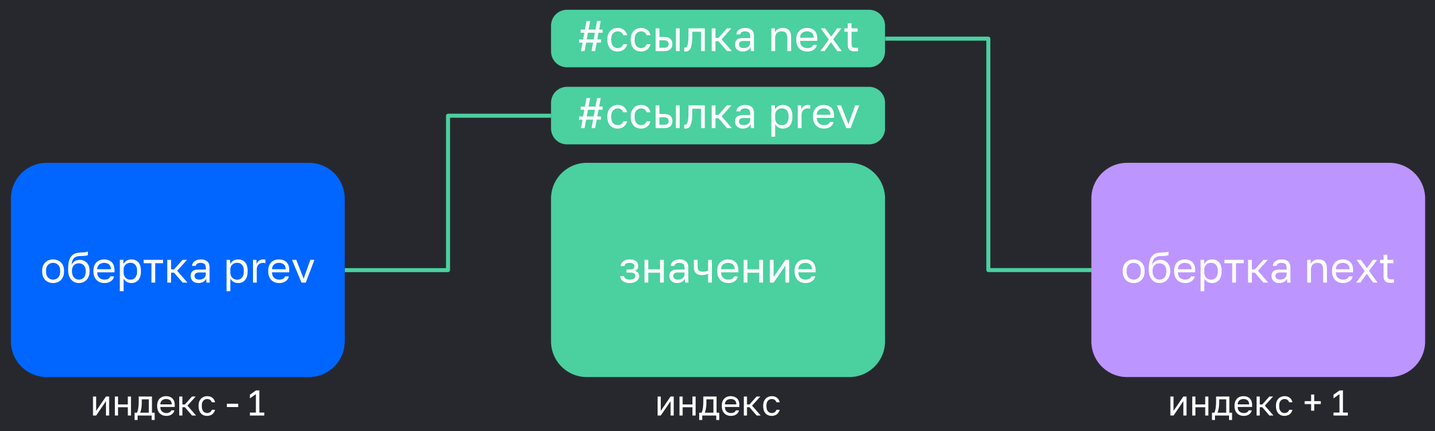
- значение элемента
- ссылка на обертку над предыдущим элементом списка



Связный список. Узел

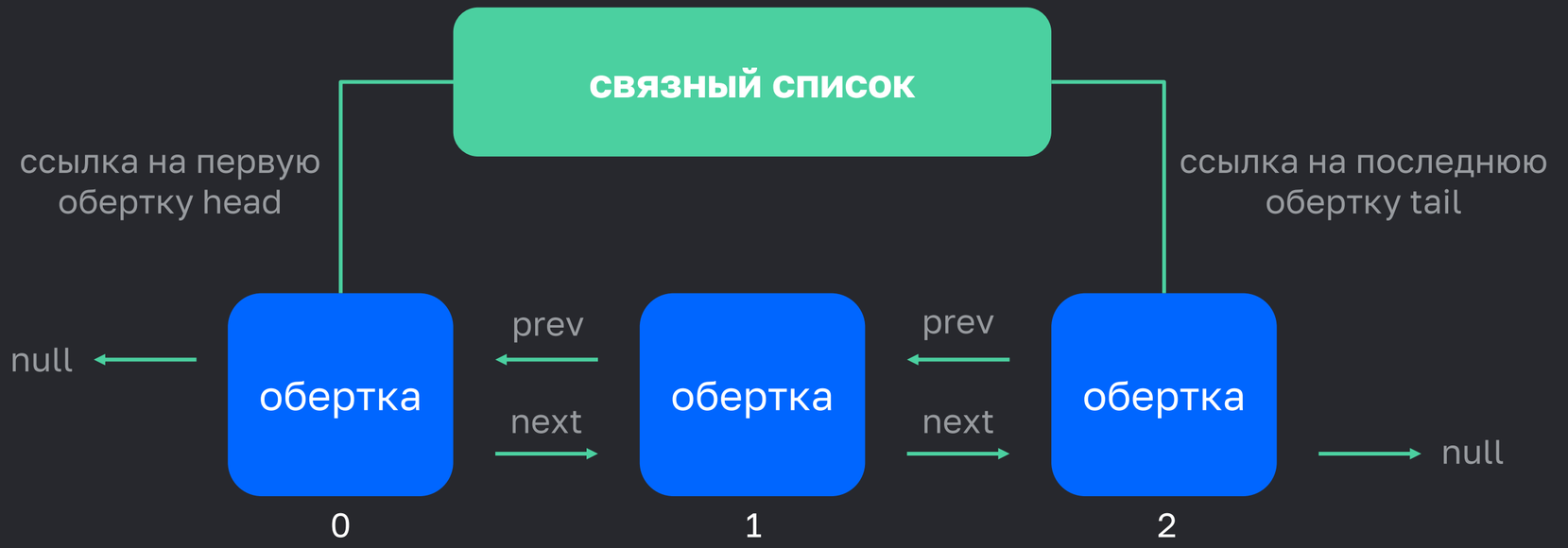
Давайте придумаем такую обертку
над элементом списка, в которой будет:

- значение элемента
- ссылка на обертку над предыдущим элементом списка
- ссылка на обертку над следующим элементом списка



Связный список. Узел

Сам связный список будет хранить просто ссылку на обертку над первым и ссылку на обертку над последним элементами списка и с помощью них реализовывать все операции.



Связный список. Узел

Сам связный список будет хранить просто ссылку на обертку над первым и ссылку на обертку над последним элементами списка и с помощью них реализовывать все операции.

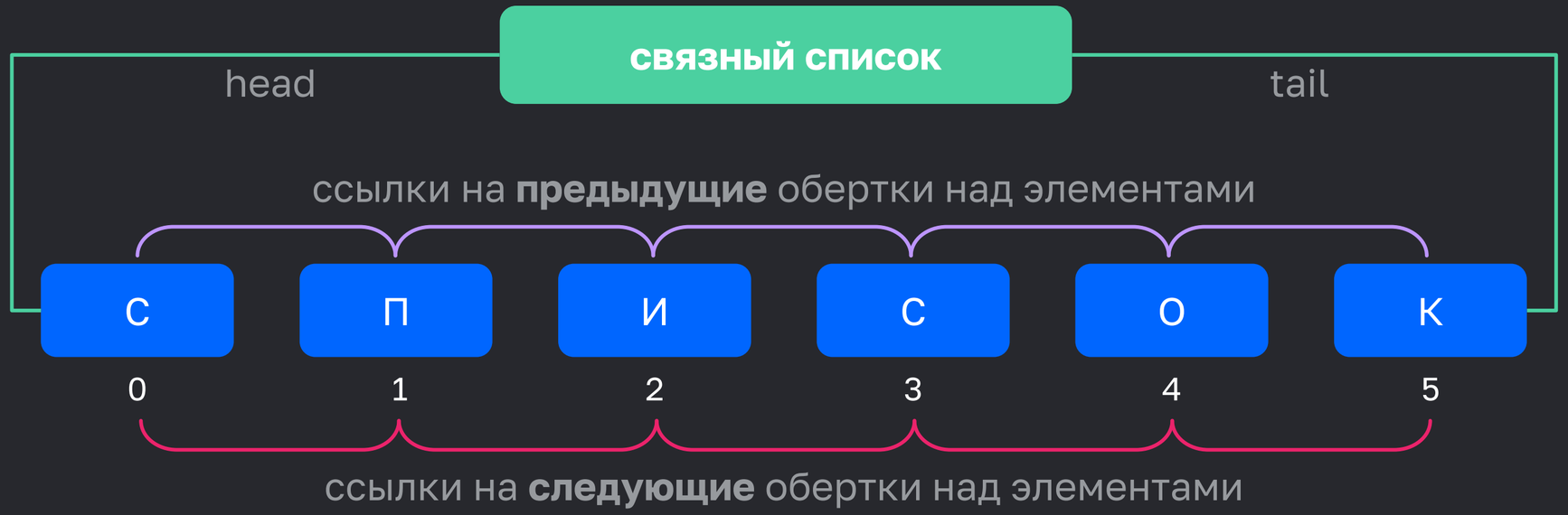
```
Node {
  e: элемент,
  prev: ссылка не предыдущий или пусто,
  next: ссылка на следующий или пусто
}

LinkedList {
  head: ссылка на первый или пусто,
  tail: ссылка на последний или пусто,
  size: размер
}
```

Связный список: get / set

Связный список: get / set

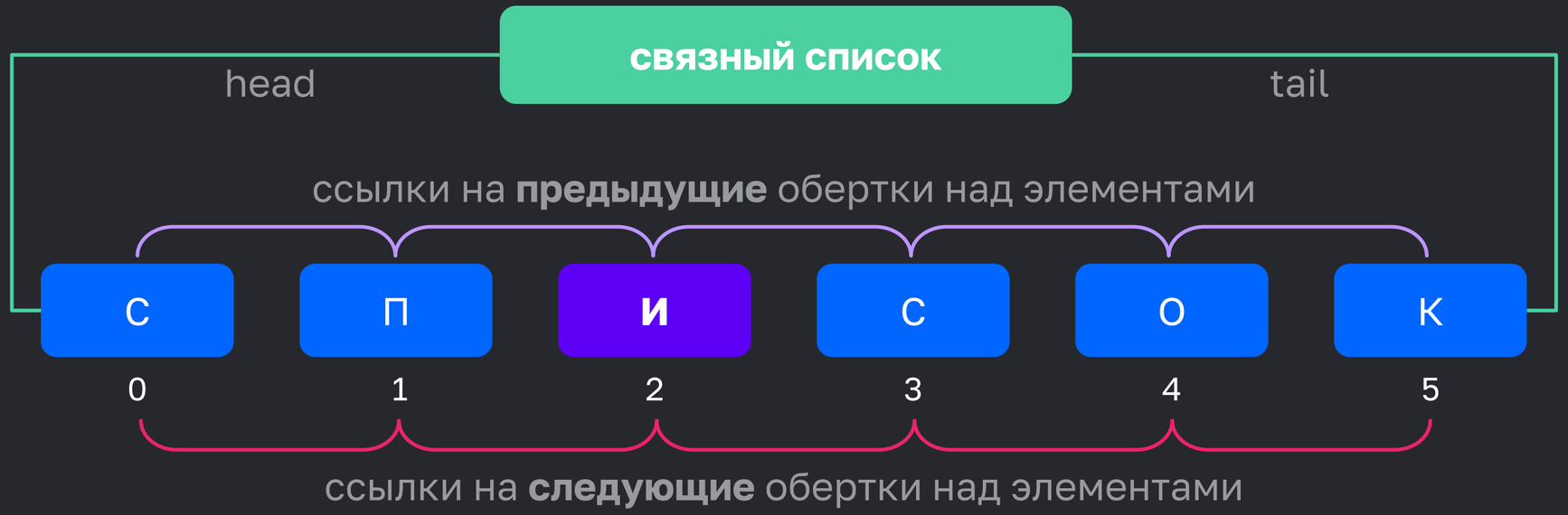
В отличие от динамического массива, доступ по индексу (get/set) к произвольному элементу по индексу не O(1), а O(n).

К примеру, у нас есть связный список:



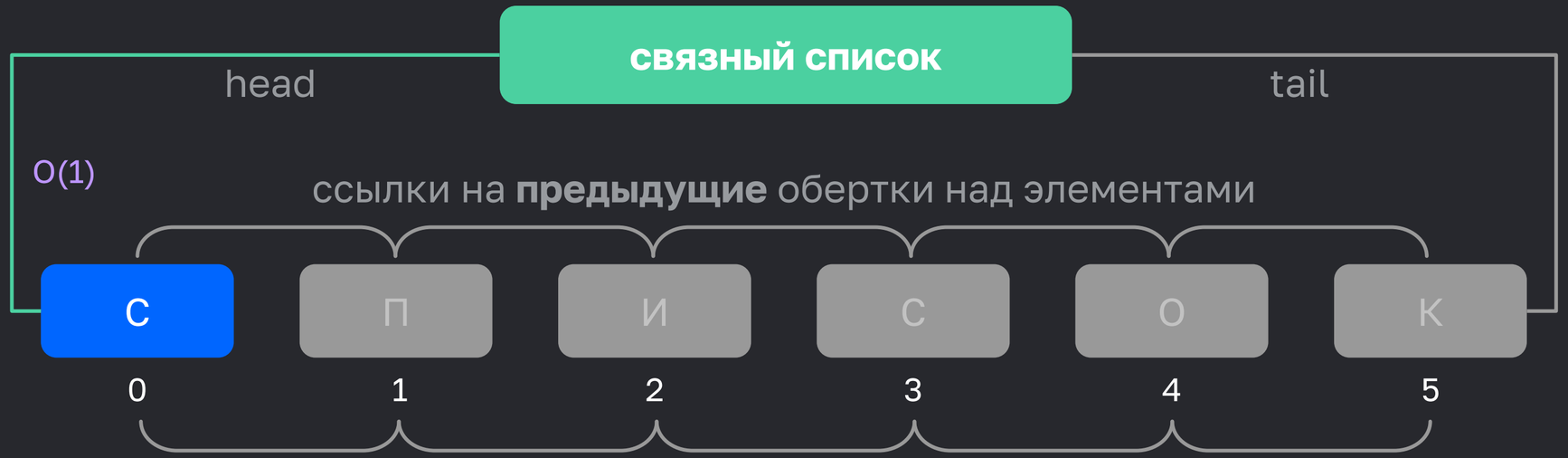
Связный список: get / set

Для доступа к элементу по индексу 2,
нам надо сделать O(индекс) шагов.



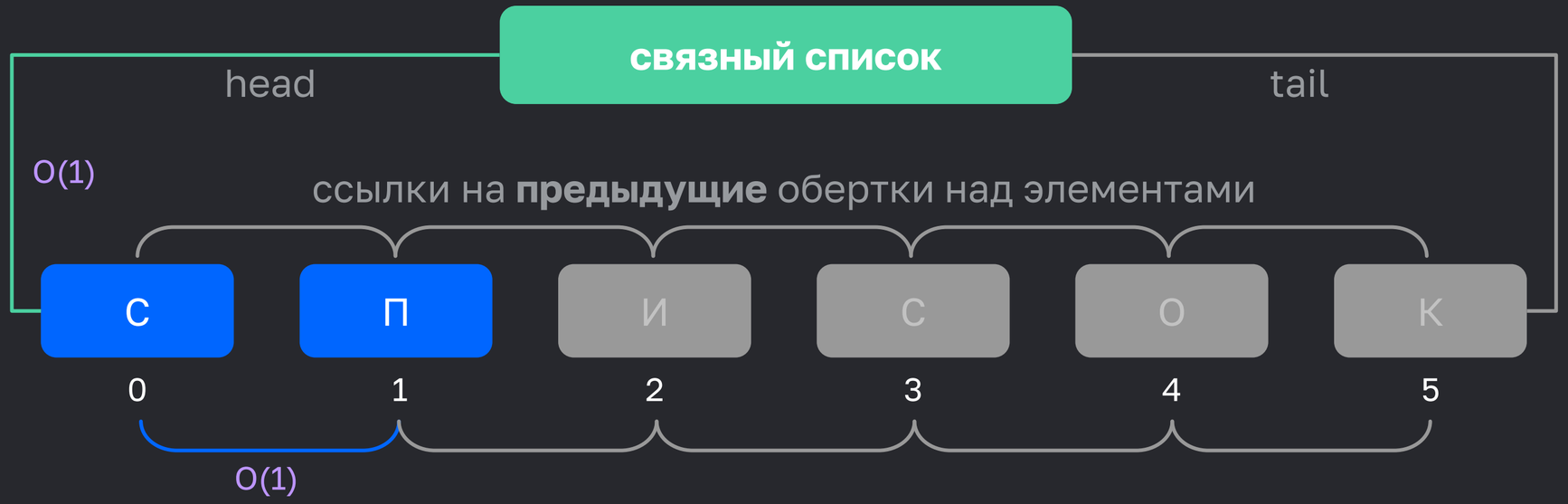
Связный список: get / set

Для доступа к элементу по индексу 2,
нам надо сделать O(индекс) шагов.



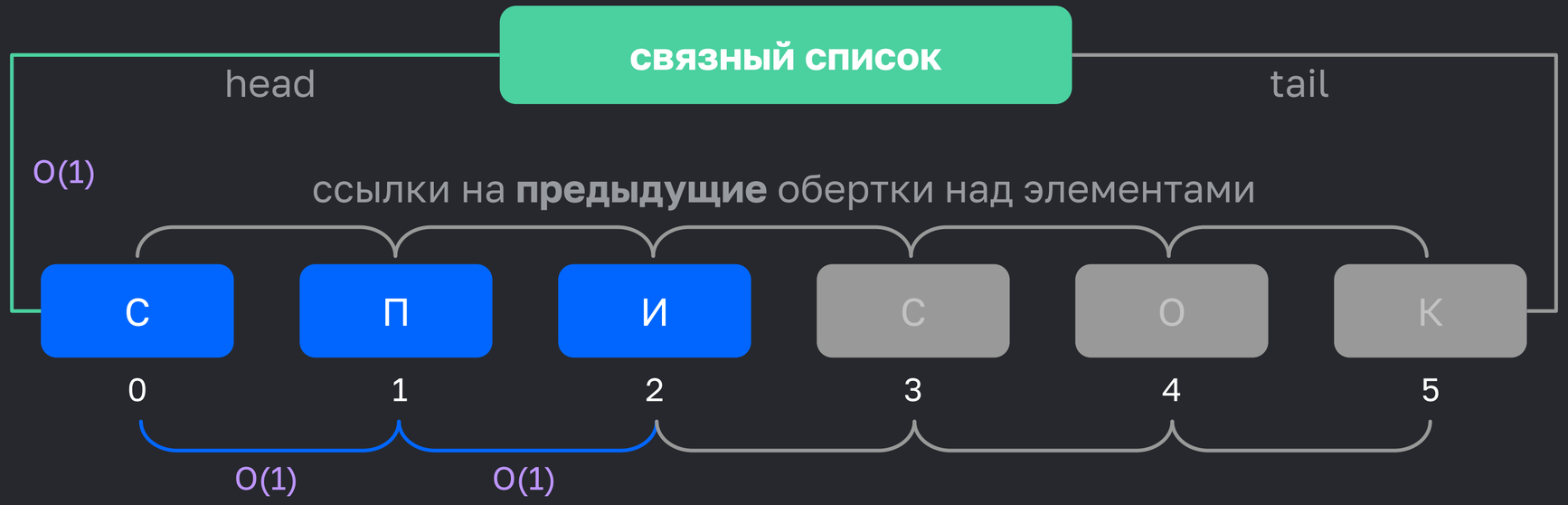
Связный список: get / set

Для доступа к элементу по индексу 2,
нам надо сделать O(индекс) шагов.



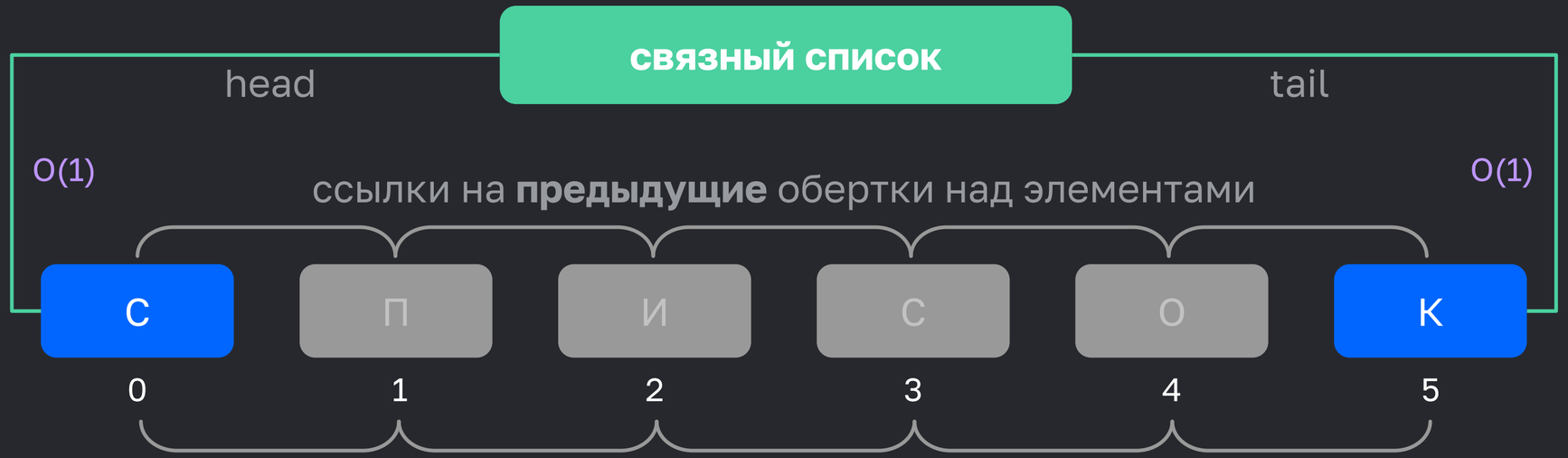
Связный список: get / set

Для доступа к элементу по индексу 2,
нам надо сделать O(индекс) шагов.



Связный список: get / set

Если необходимо обратиться (get/set) к первому или последнему элементам, то это займет O(1), поскольку ссылки на них уже есть.



Связный список: get / set.

Разбор задачи

Связный список: get / set. Задача

Задача: Поменять на всех чётных
индексах списка элементы на 0.

Если мы просто вызовем set для каждого четного
индекса, то получим $O(n^2)$, поскольку придется
для каждого индекса идти с самого начала списка.

Связный список: get / set. Задача

Задача: Поменять на всех чётных индексах списка элементы на 0.

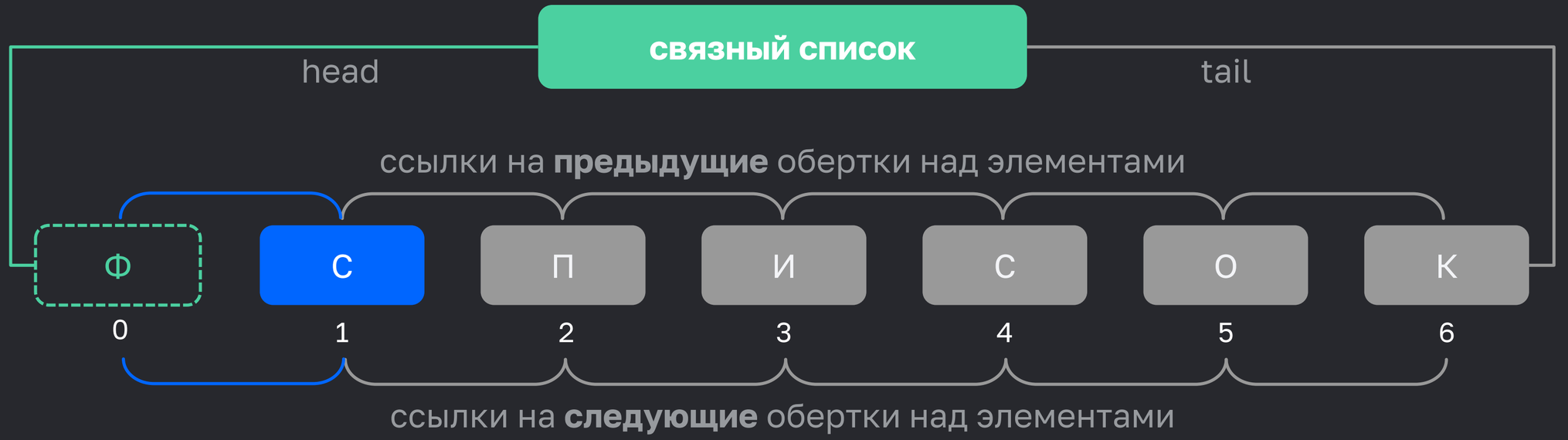
Если мы просто вызовем set для каждого четного индекса, то получим $O(n^2)$, поскольку придется для каждого индекса идти с самого начала списка.

Но можем сделать хитрее: пройтись один раз по списку и менять все элементы прямо в этом цикле, ведь если мы уже находимся в нужном узле, то get/set для него $O(1)$. В таком случае выйдет за $O(n)$.

Связный список: add и remove

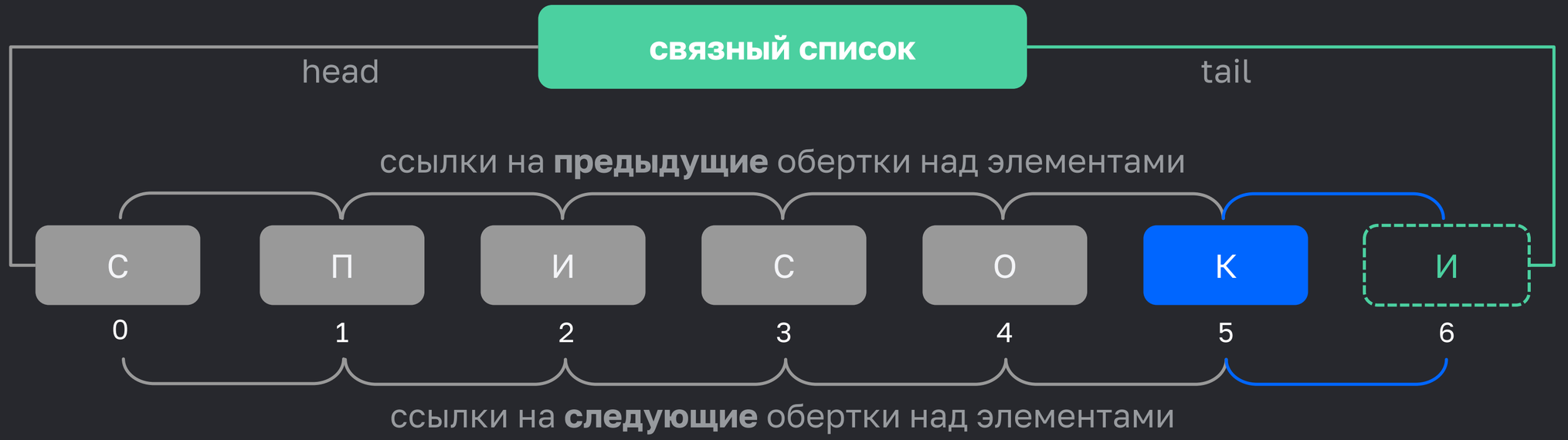
Связный список: add

При добавлении в начало достаточно просто аккуратно изменить ссылку у первого элемента и обновить ссылку у списка. Делается это за O(1).



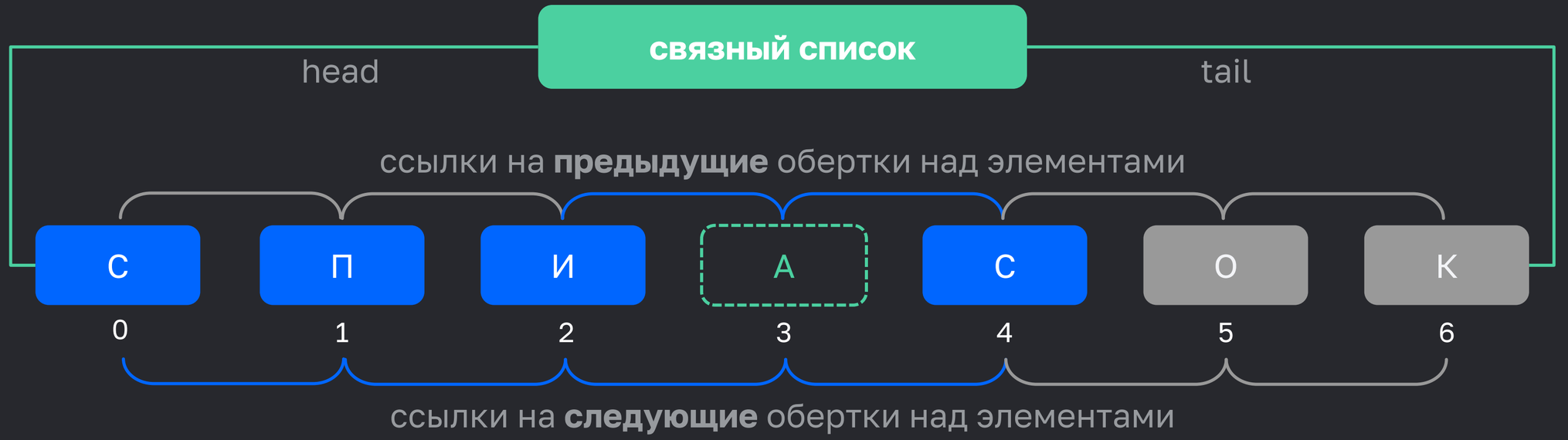
Связный список: add

При добавлении в конец достаточно просто аккуратно изменить ссылку у последнего элемента и обновить ссылку у списка. Делается это за O(1).



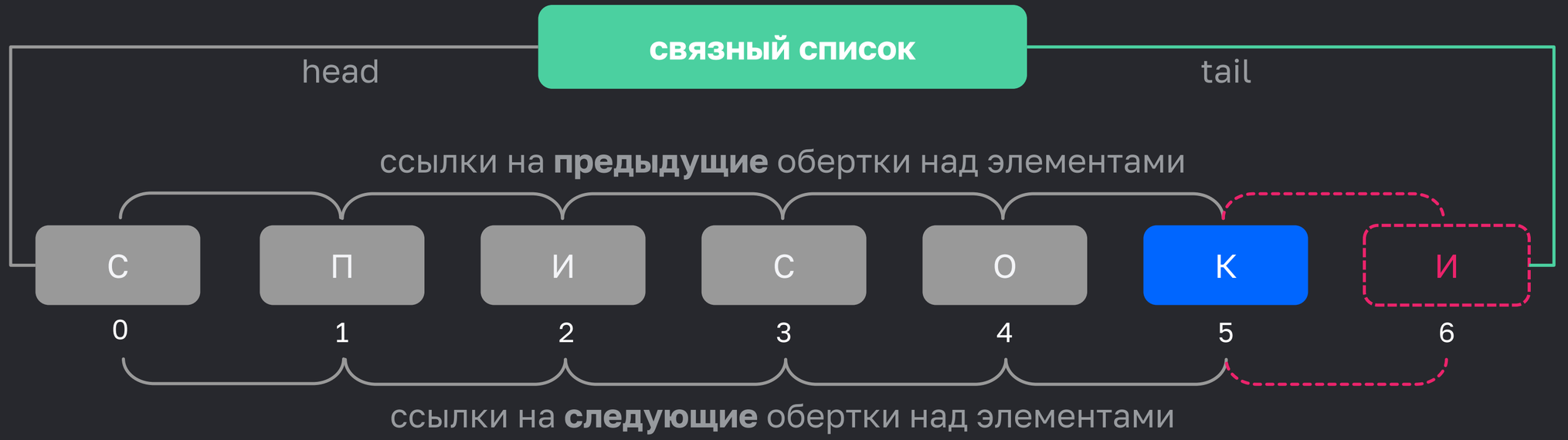
Связный список: add

При добавлении в произвольное место основной вклад вносит время, затрачиваемое на то чтобы дойти до нужного элемента: O(n).
Затем аккуратно перекидываются ссылки, суммарно за O(1).



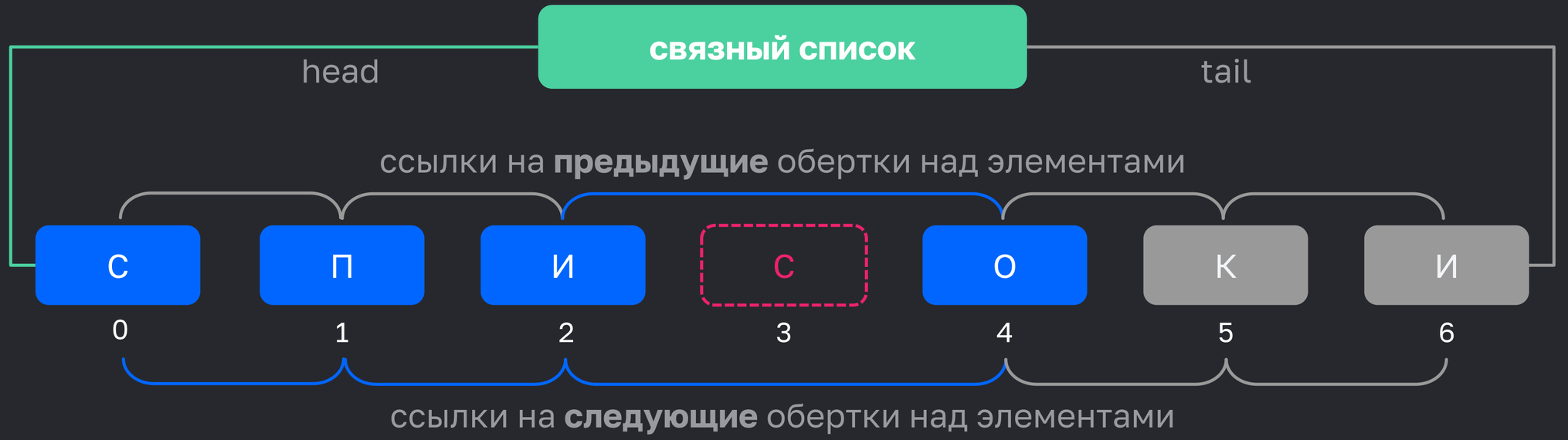
Связный список: remove

При удалении с конца достаточно просто изменить ссылки у предпоследнего элемента и обновить ссылки списка. Делается это за O(1).



Связный список: remove

При удалении в произвольном месте основной вклад
в асимптотику — время, чтобы дойти до нужного места за O(n).
Само удаление — это перекидывание ссылок за O(1).



Динамический массив VS связный список

Динамический массив VS связный список

Рассмотрим итоги в таблице.

Операция	Динамический массив	Связный список
добавление в конец	в среднем O(1)	O(1)

Динамический массив VS связный список

Рассмотрим итоги в таблице.

Операция	Динамический массив	Связный список
добавление в конец	в среднем O(1)	O(1)
добавление в начало	O(n)	O(1)

Динамический массив VS связный список

Рассмотрим итоги в таблице.

Операция	Динамический массив	Связный список
добавление в конец	в среднем O(1)	O(1)
добавление в начало	O(n)	O(1)
добавление в произвольное место	O(n)	O(1) если дошли до элемента, иначе O(n)

Динамический массив VS связный список

Рассмотрим итоги в таблице.

Операция	Динамический массив	Связный список
добавление в конец	в среднем O(1)	O(1)
добавление в начало	O(n)	O(1)
добавление в произвольное место	O(n)	O(1) если дошли до элемента, иначе O(n)
удаление с конца	в среднем O(1)	O(1)

Динамический массив VS связный список

Рассмотрим итоги в таблице.

Операция	Динамический массив	Связный список
добавление в конец	в среднем O(1)	O(1)
добавление в начало	O(n)	O(1)
добавление в произвольное место	O(n)	O(1) если дошли до элемента, иначе O(n)
удаление с конца	в среднем O(1)	O(1)
удаление с начала	O(n)	O(1)

Динамический массив VS связный список

Рассмотрим итоги в таблице.

Операция	Динамический массив	Связный список
добавление в конец	в среднем O(1)	O(1)
добавление в начало	O(n)	O(1)
добавление в произвольное место	O(n)	O(1) если дошли до элемента, иначе O(n)
удаление с конца	в среднем O(1)	O(1)
удаление с начала	O(n)	O(1)
удаление с произвольного места	O(n)	O(1) если дошли до элемента, иначе O(n)

Динамический массив VS связный список

Рассмотрим итоги в таблице.

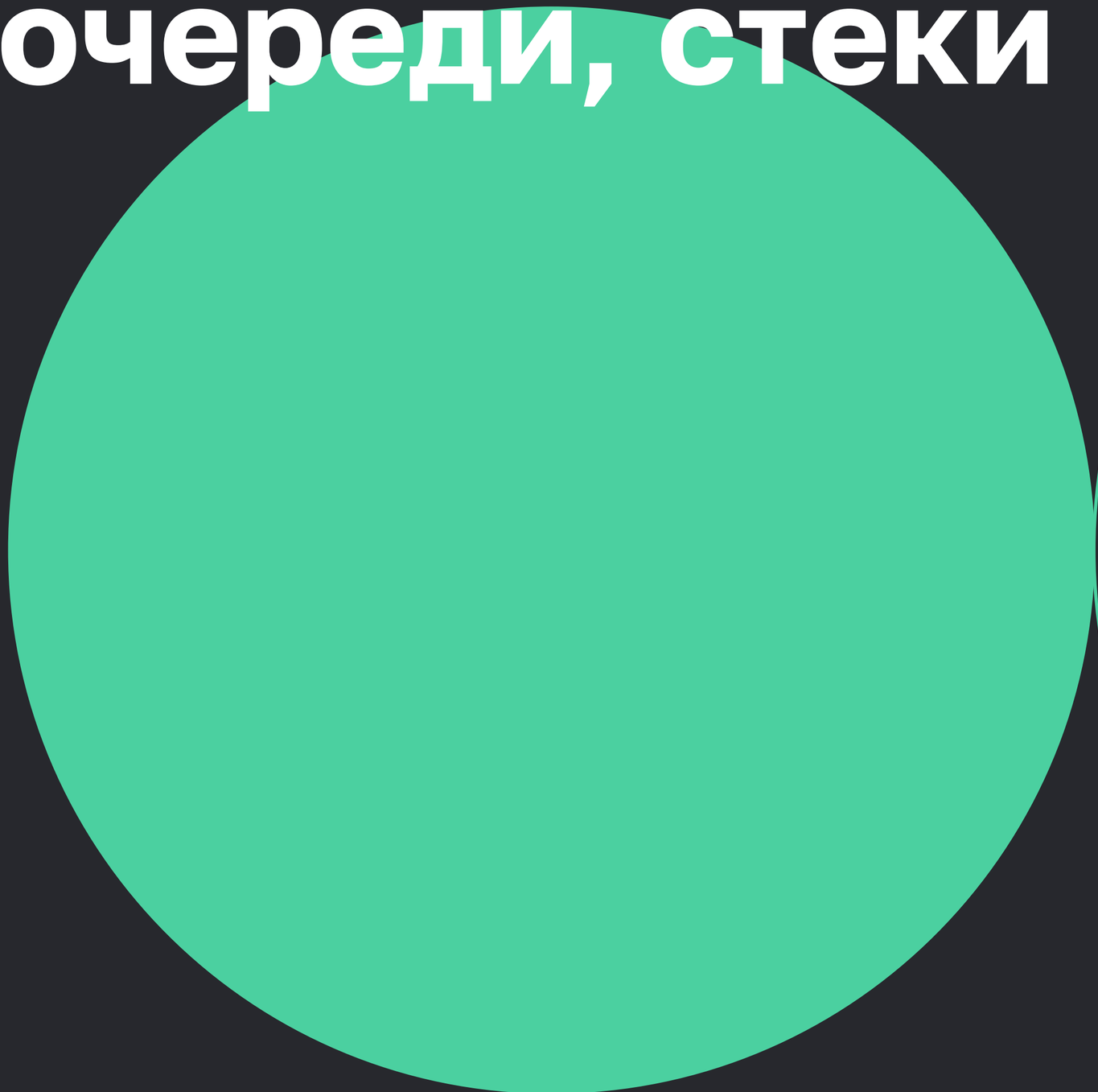
Операция	Динамический массив	Связный список
добавление в конец	в среднем O(1)	O(1)
добавление в начало	O(n)	O(1)
добавление в произвольное место	O(n)	O(1) если дошли до элемента, иначе O(n)
удаление с конца	в среднем O(1)	O(1)
удаление с начала	O(n)	O(1)
удаление с произвольного места	O(n)	O(1) если дошли до элемента, иначе O(n)
доступ по индексу	O(1)	O(n)

Динамический массив VS связный список

Немного про O(n). Оно говорит нам лишь о характере роста времени при увеличении размера входных данных. Поэтому нельзя сказать, какой из двух разных алгоритмов, работающих за O(n), быстрее.

Часто там, где и у динамического массива и у связного списка стоит O(n), динамический массив работает быстрее, потому что работа с массивами обычно быстрее, чем переходы по ссылкам в произвольные места памяти.

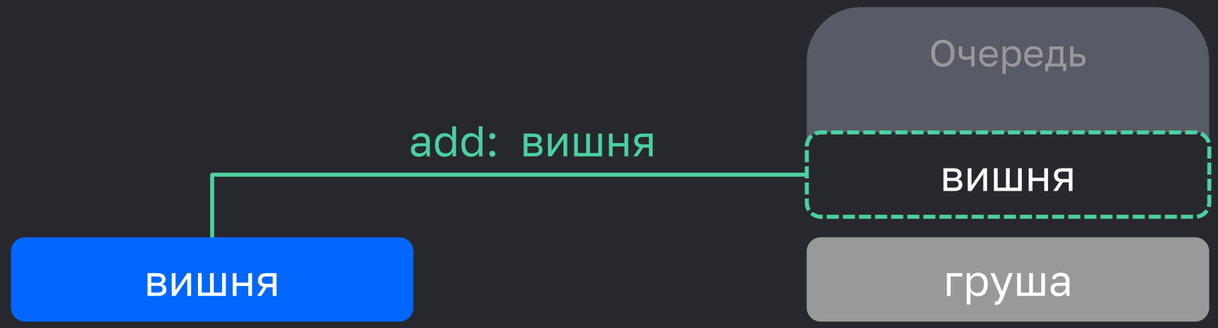
ДЕКИ, очереди, стеки



Очереди

Что такое очереди? Это семейство структур данных, которые умеют делать как минимум следующие операции:

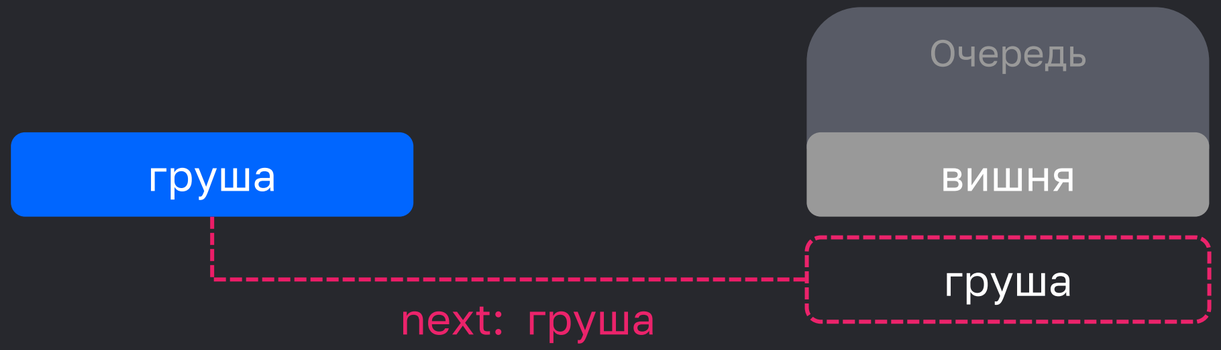
Операция	Значение
add	добавляет элемент в очередь
next	вынимает элемент из очереди



Очереди

Что такое очереди? Это семейство структур данных, которые умеют делать как минимум следующие операции:

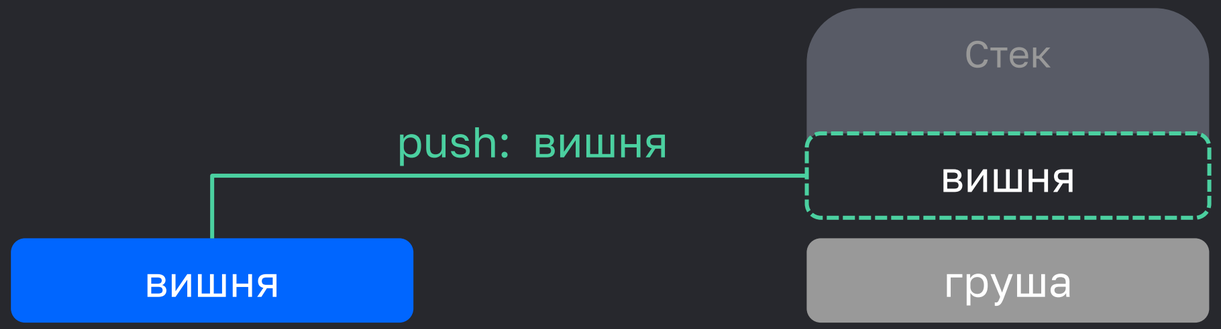
Операция	Значение
add	добавляет элемент в очередь
next	вынимает элемент из очереди



Стеки

Что такое стеки («стопки»)? Это семейство структур данных, которые умеют делать как минимум следующие операции:

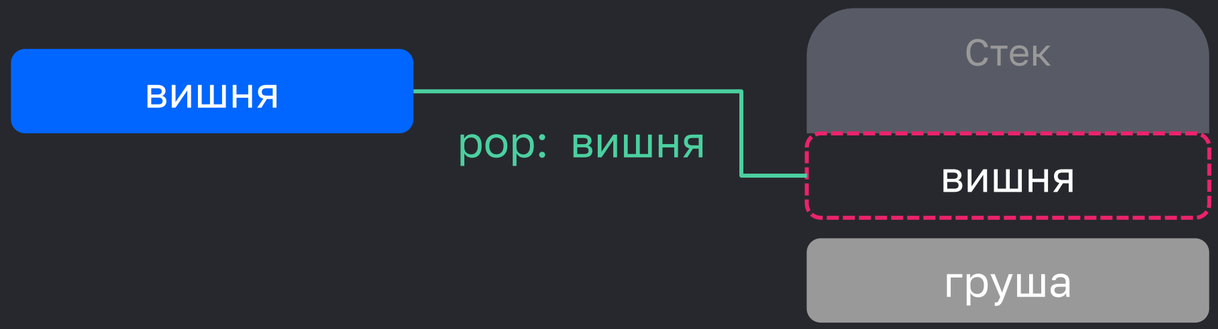
Операция	Значение
push	добавляет элемент в стек
pop	снимает элемент со стека



Стеки

Что такое стеки («стопки»)? Это семейство структур данных, которые умеют делать как минимум следующие операции:

Операция	Значение
push	добавляет элемент в стек
pop	снимает элемент со стека



Деки

Что такое деки? Дек (анг. Double-Ended QUEUE - очередь с двумя концами) — это семейство структур данных, в которые умеют вставать и с которых умеют уходить с обоих концов. Очередь и стек можно реализовать через дек:

Операция стека	Операция дека
push	addTail
pop	nextTail

Операция очереди	Операция дека
add	addHead
next	nextTail

Деки на связном списке

Деки можно реализовать на связном списке:

Операция дека	Операция связного списка	Асимптотика
addHead	вставка в начало	O(1)
nextHead	удаление с начала	
addTail	вставка в конец	
nextTail	удаление с конца	

Просто заменить в данной реализации связный список на динамический массив без ухудшения асимптотики не получится: addHead и nextHead превратятся в O(n).