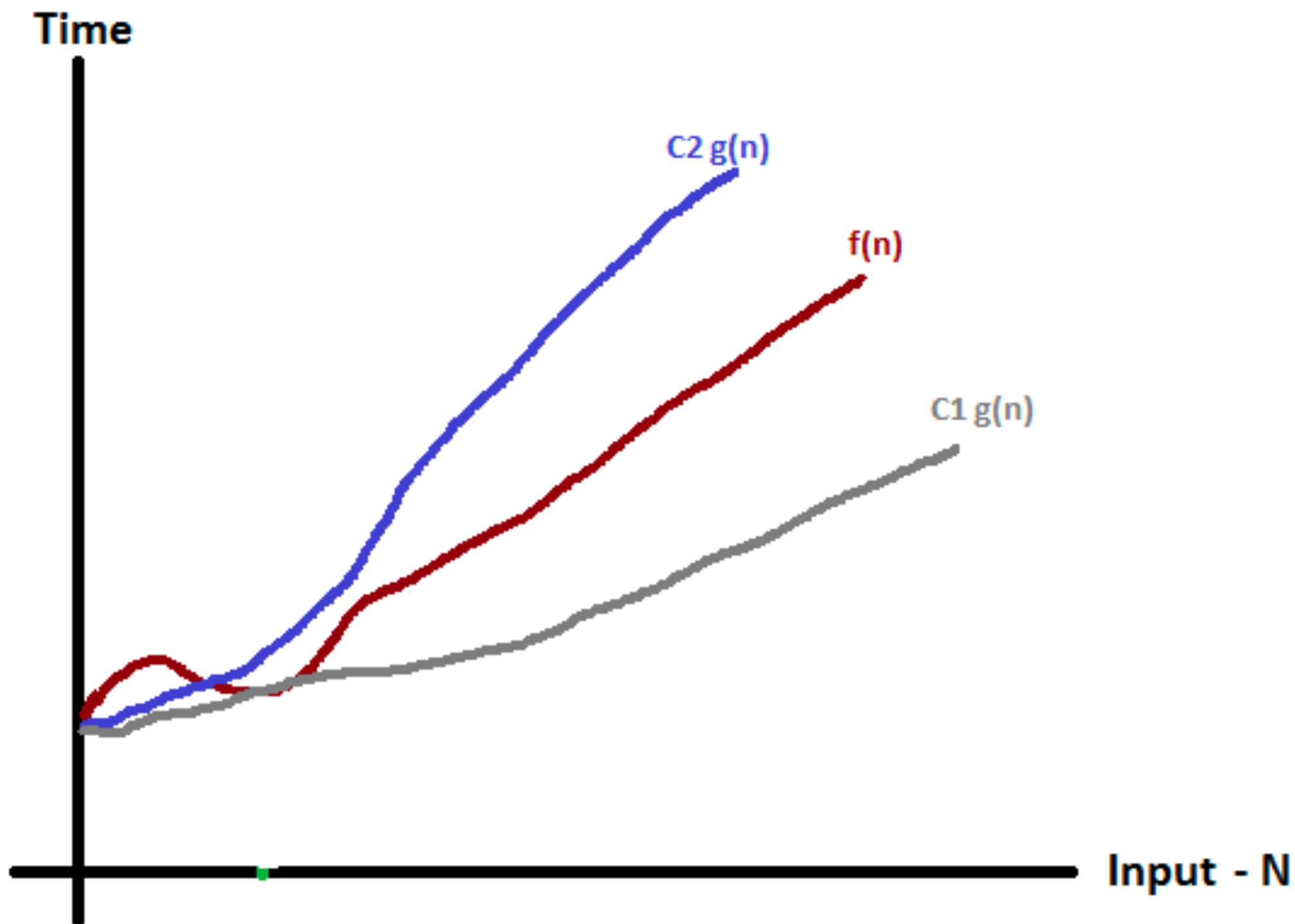


О-оценка

- Как оценивать время работы алгоритма с точки зрения оптимиста?
- С точки зрения пессимиста?
- С точки зрения реалиста?
- Какой(ие) подход(ы) нам интересны?

Theta-оценка



O-оценка

```
s = 0
for i in range(N):
    s += i
```

O-оценка

```
s = 0
for i in range(N):
    for j in range(N):
        s += i
```

O-оценка

```
s = 0
for i in range(N):
    for j in range(i):
        s += i
```

O-оценка

```
s = 0
for i in range(N):
    for j in range(i):
        if j % 5 == 0:
            s += j
```

O-оценка

```
s = 0
for i in range(N):
    if i == 5:
        break
```

O-оценка

```
s = 0
for i in range(N):
    for j in range(i):
        if j % 5 == 0:
            break
```


O-оценка

```
s = N  
while s > 0:  
    s -= 1
```

O-оценка

```
s = N
while s > 0:
    s = s // 2
```

O-оценка

```
for i in range(N):  
    s = N  
    while s > 0:  
        s = s // 2
```

O-оценка

```
for i in range(N):  
    s = i  
    while s > 0:  
        s = s // 2
```

O-оценка

```
flag = False
for i in range(N):
    s = i
    while s > 0 and not flag:
        s = s // 2
        if s == 4:
            flag = True
```

O-оценка

```
for i in range(N):  
    flag = False  
    s = i  
    while s > 0 and not flag:  
        s = s // 2  
        if s == 4:  
            flag = True
```

O-оценка

```
flag = False
for i in range(N):
    s = i
    while s > 0 and not flag:
        s = s // 2
        if s == 4:
            flag = True

    if s ** 2 + 5 > 6:
        flag = False
```

O-оценка

```
s = 0
for i in range(N):
    for j in range(i):
        for k in range(j):
            s += 1
```


O-оценка

```
for i in range(N):  
    k = 0  
    for j in range(N):  
        if j % 7 == 0:  
            k += 1  
    while k > 0:  
        k -= 1
```

Что можно почитать:

С. Скиена. **Алгоритмы. Руководство по разработке. Глава 2: Анализ алгоритмов**
Л. Лакман Макдауэлл. **Карьера программиста. Часть 6: O-большое**

O-оценка

```
s = 0
k = 2
for i in range(N):
    s += 1
    if i == k:
        k = k * 2
        for j in range(i):
            s += 1
```

O-оценка

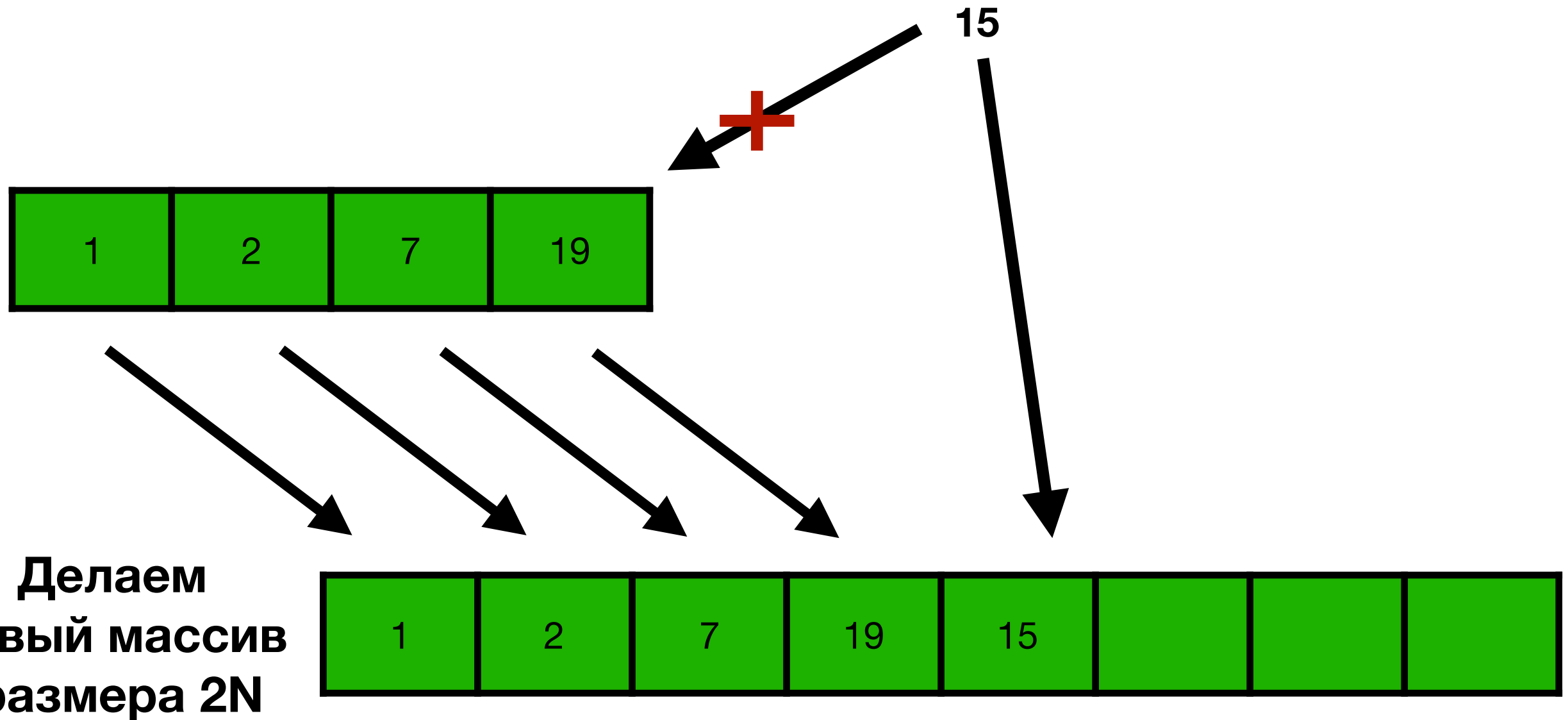
```
A - массив размера N
k = 0
for i in range(N):
    while k > 0 and A[k] != 5:
        if random() > 0.5:
            k -= 1
        else:
            k = max(0, k - 5)
    if A[k] == 5:
        k += 1
```

Амортизационный анализ

Амортизационный анализ (англ. *amortized analysis*) — метод подсчёта времени, требуемого для выполнения последовательности операций над структурой данных. При этом время усредняется по всем выполняемым операциям, и анализируется средняя производительность операций в худшем случае.

Динамический массив

Добавляем число



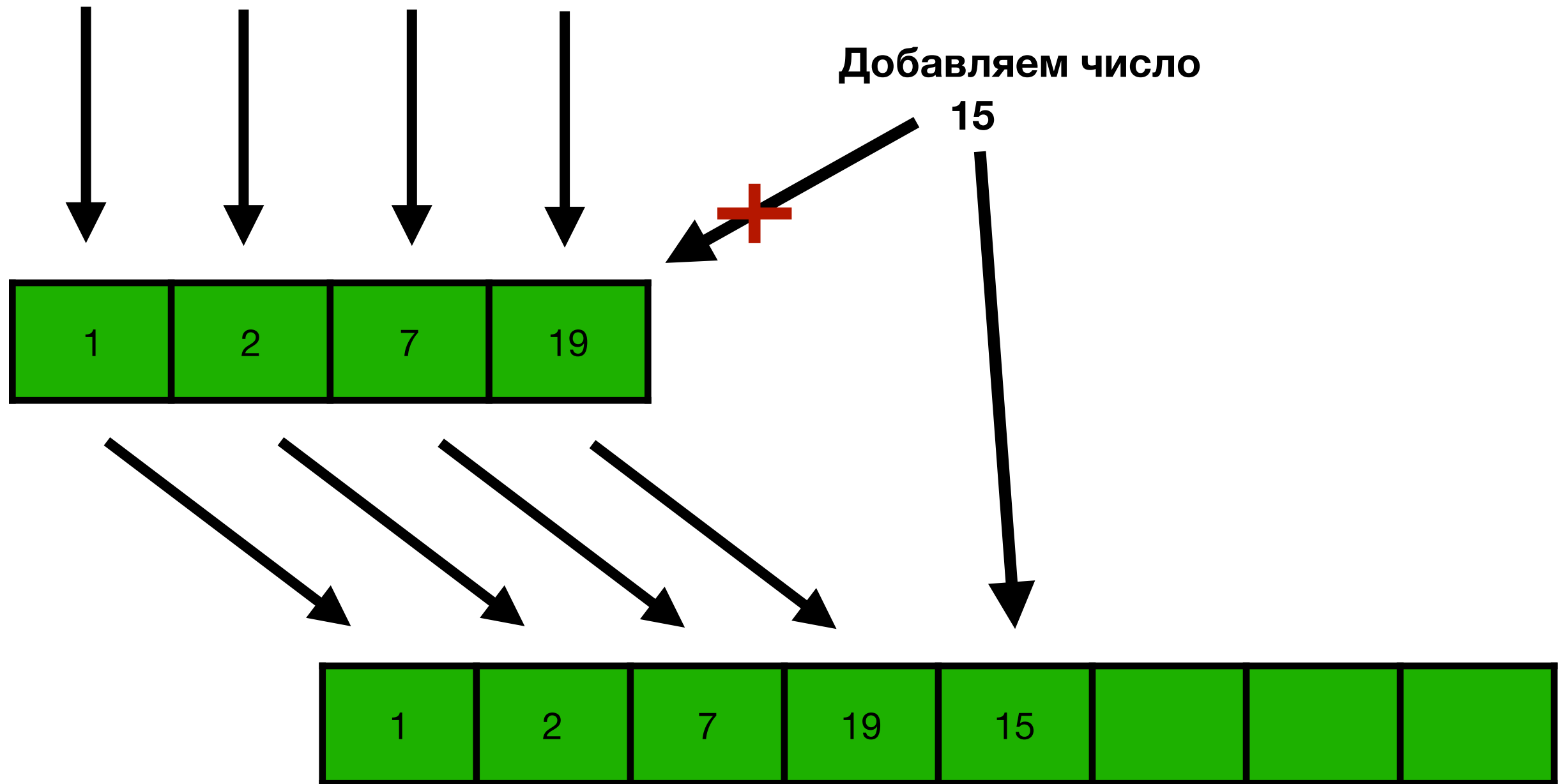
Надо скопировать N элементов - работает за $O(N)$

Сколько будет стоить добавить N элементов в массив в худшем случае?

Казалось бы, $O(N) * N = O(N^2)$.

Но это неверно

Но прежде чем добавить $N+1$ число, нам надо было добавить N элементов,
они добавлялись за $O(1)$



Надо скопировать N элементов - работает за $O(N)$

Подсчитаем, сколько в среднем на каждый элемент

$$\frac{O(N) + O(1) * N}{N + 1} \approx \frac{O(N) + O(1) * N}{N} = O(1) + O(1) = O(1)$$

При этом мы не можем попасть в худший случай $O(N)$, не пройдя случаи, когда мы добавляли за $O(1)$.

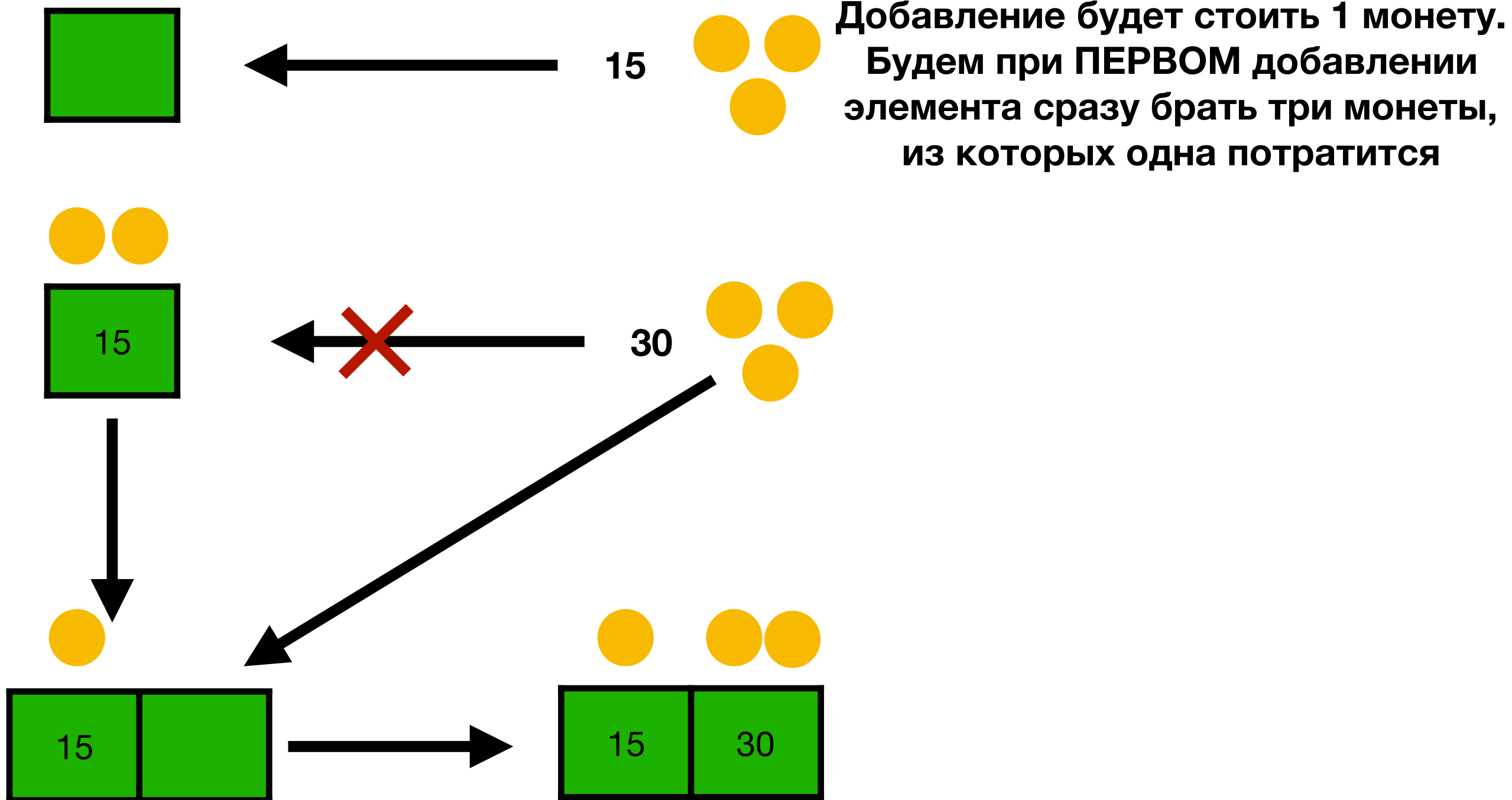
Потому добавить N элементов в массив будет стоить $O(1)$ на каждый элемент или $O(N)$ в целом.

Метод предоплаты (бухгалтерского учета)

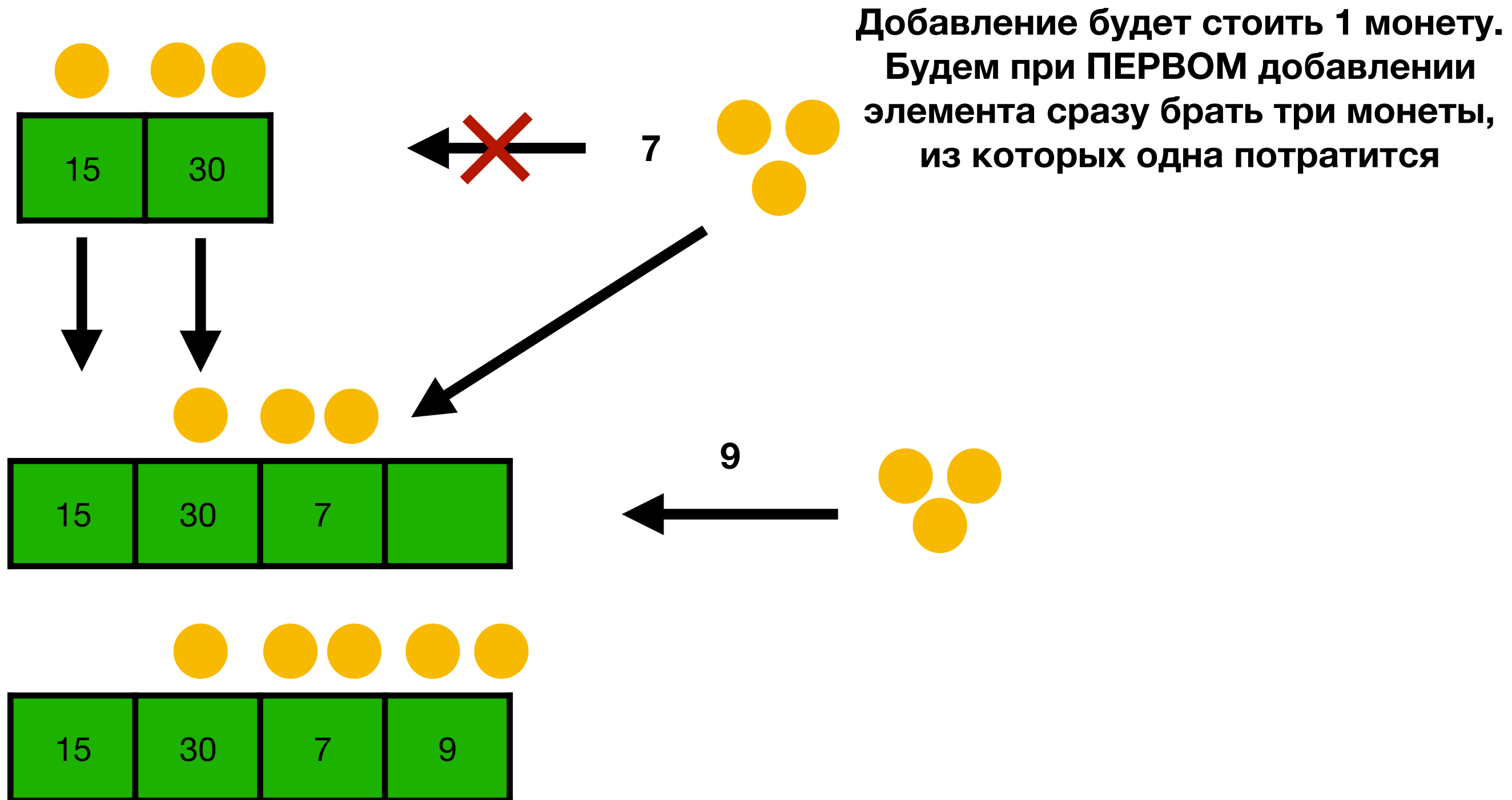
**Пусть каждая операция стоит сколько-то момент (платим жадному гному)
Мы не хотим в какой-то момент остаться без возможности заплатить.
Потому в какие-то моменты времени мы откладываем монеты, которые
заплатим в дальнейшем.**

Динамический массив

Сначала имеем массив размера 1, с 1 пустым местом

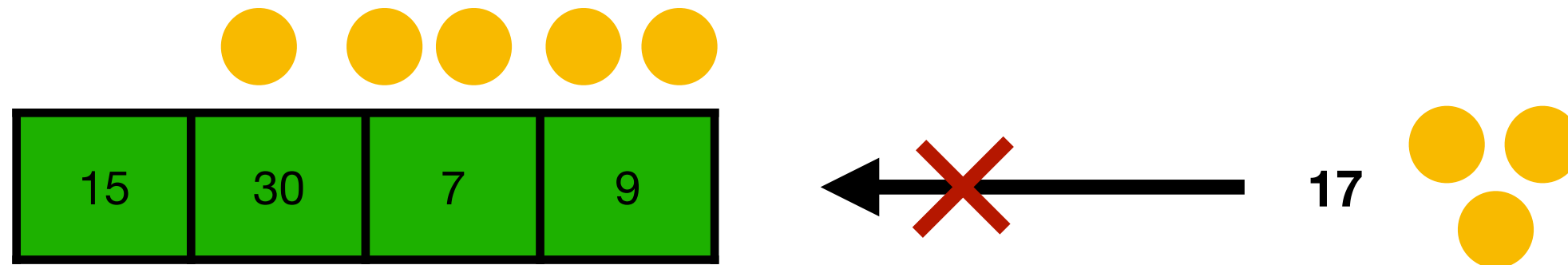


Динамический массив



Динамический массив

Сначала имеем массив размера 1, с 1 пустым местом

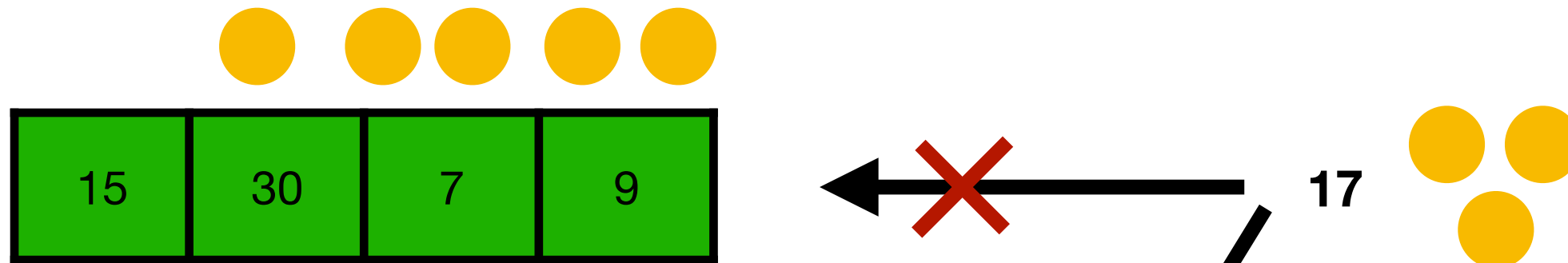


У 15 нет монеток

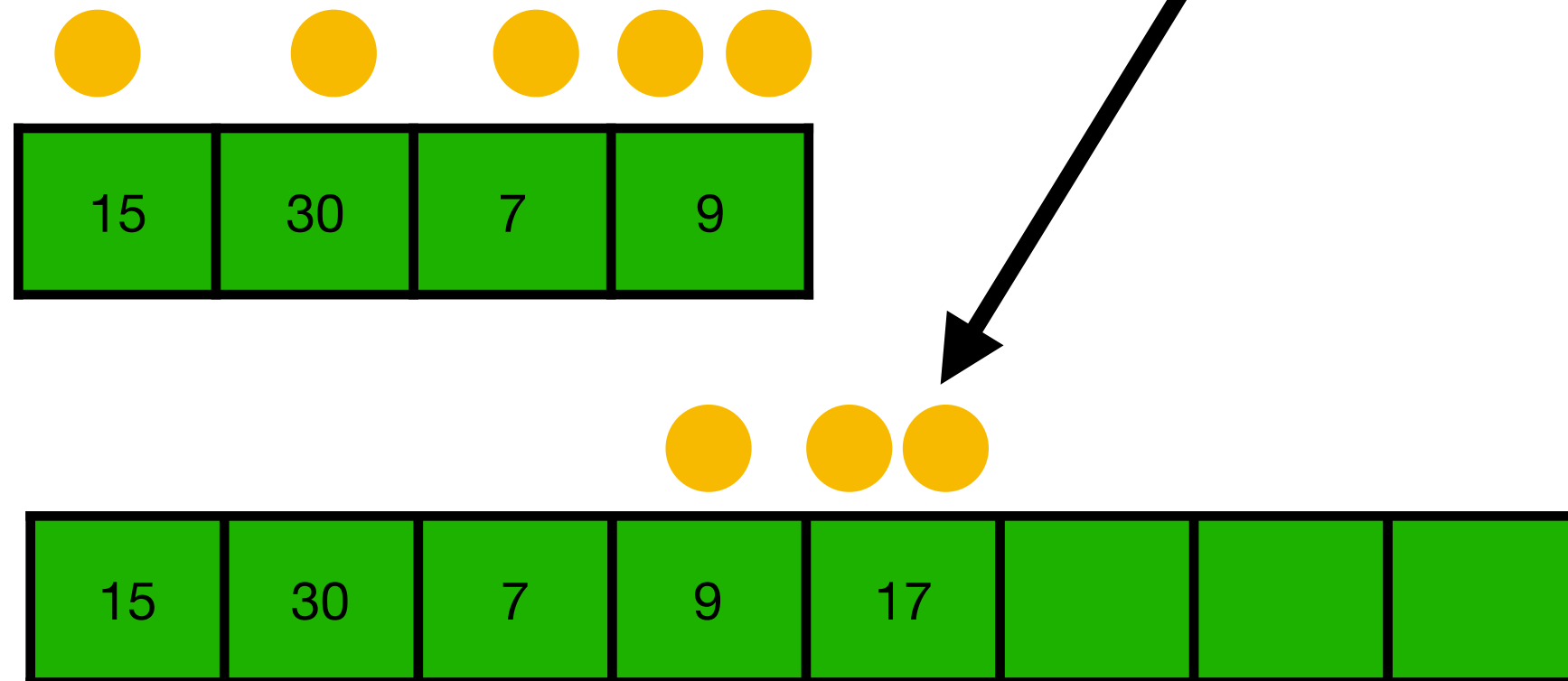


Динамический массив

Сначала имеем массив размера 1, с 1 пустым местом

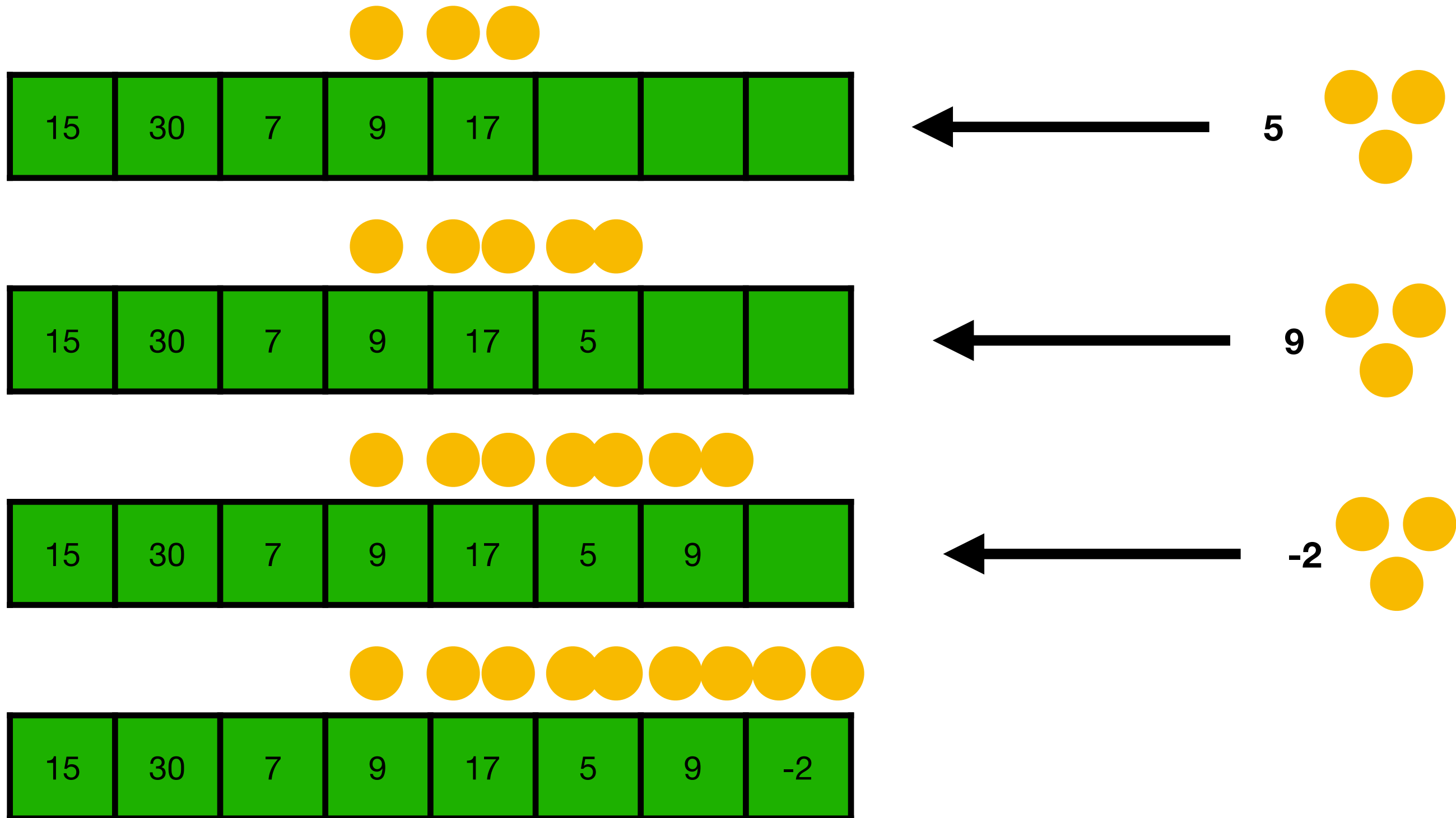


У 15 нет монеток, но с ним может
поделиться один из элементов
второй части массива



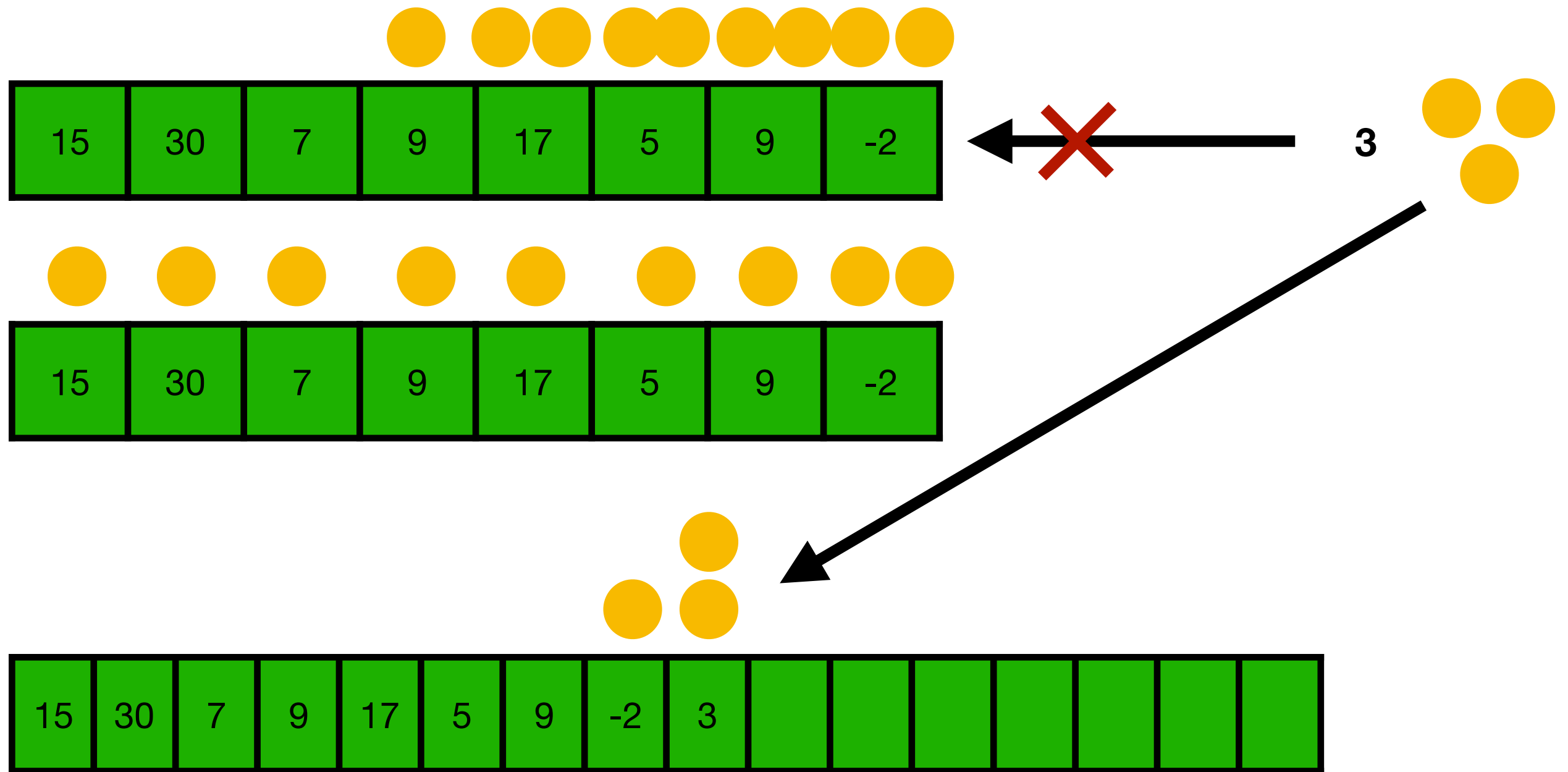
Динамический массив

Сначала имеем массив размера 1, с 1 пустым местом



Динамический массив

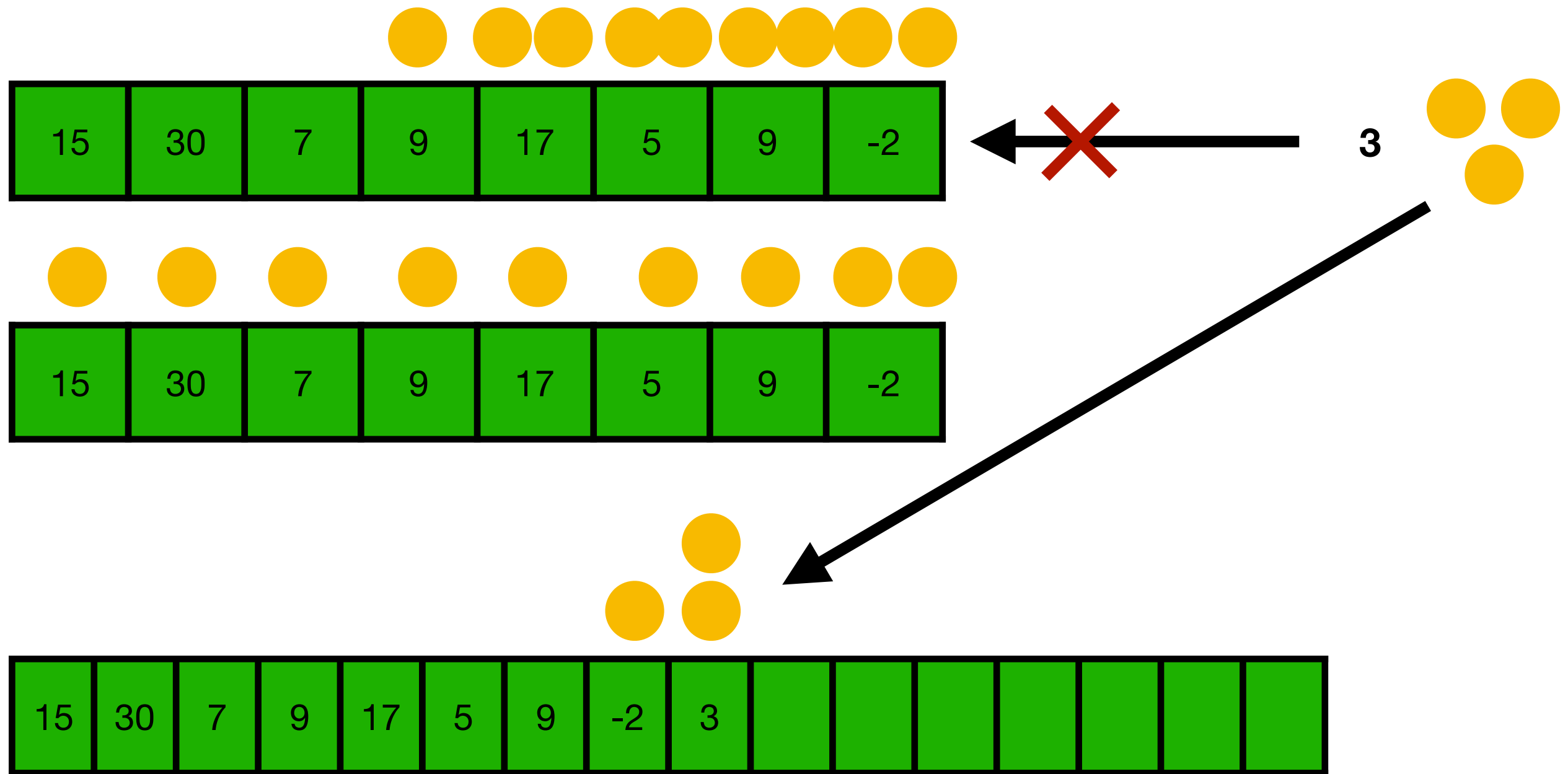
Сначала имеем массив размера 1, с 1 пустым местом



Всегда ли нам будет чем заплатить за перенос элементов?

Динамический массив

Сначала имеем массив размера 1, с 1 пустым местом



Всегда, так как у нас есть не больше $N/2$ без монеток и не меньше $N/2$ элементов с двумя монетками

Динамический массив

Таким образом, если теперь мы переведем монетки в элементарные операции, то получается, что если мы оцениваем сложность каждого из N добавлений в динамический массив как $3 O(1)$ операции, то наша оценка является правильной верхней оценкой.

Тогда суммарно N последовательных операций добавлен займут $O(3 * N) = O(N)$ времени.

Говорим, что добавление в динамический массив занимает **амортизационно** $O(1)$

Что можно прочесть

- Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн -
Алгоритмы. Построение и анализ. Глава
Амортизационный анализ. Разбирается 3 способа
подсчета амортизационной стоимости.

O-оценка

```
s = 0
k = 2
for i in range(N):
    s += 1
    if i == k:
        k = k * 2
        for j in range(i):
            s += 1
```

O-оценка

```
A - массив размера N
k = 0
for i in range(N):
    while k > 0 and A[k] != 5:
        if random() > 0.5:
            k -= 1
        else:
            k = max(0, k - 5)
    if A[k] == 5:
        k += 1
```

БИТОВЫЕ ОПЕРАЦИИ

Все данные в компьютере хранятся в бинарном виде - набор 0 и 1.

Например, 7 хранится виде 111.

8 хранится в виде 1000.

В каком виде хранится число 17?

БИТОВЫЕ ОПЕРАЦИИ

Все данные в компьютере хранятся в бинарном виде - набор 0 и 1.

Например, 7 может храниться в одном байте в виде 00000111.

8 - в виде 00001000.

В каком виде хранится число 17?

На бинарных данных можно определить набор операций:

Отрицание

Если в бинарной записи числа до этого стоял 0, то ставим 1, иначе 0

$$\neg(00001000) = 11110111$$

Побитовое И

Если в обоих аргументах на соответствующих позициях
1 - 1, иначе - 0

$$\begin{array}{l} \text{\&} \begin{array}{l} 00000111 \\ 00001010 \end{array} = 00000010 \end{array}$$

Побитовое ИЛИ

Если хотя бы в одном аргументе на соответствующих позициях 1
- 1, иначе - 0

$$\begin{array}{l} 00000111 \\ 00001010 \end{array} \text{ ИЛИ } = 00001111$$

Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)

Если ровно в одном из аргументов на соответствующих позициях 1 - 1, иначе - 0

$$\begin{array}{l} \wedge \quad 00000111 \\ \quad 00001010 \end{array} = 00001101$$

Сдвиг влево

00000111 << 1 = 00001110

00000111 << 3 = 00111000

00000111 << 7 = 10000000

Сдвиг вправо

00001011 >> 1 = 00000101

00001011 >> 3 = 00000001

00000111 >> 7 = 00000000

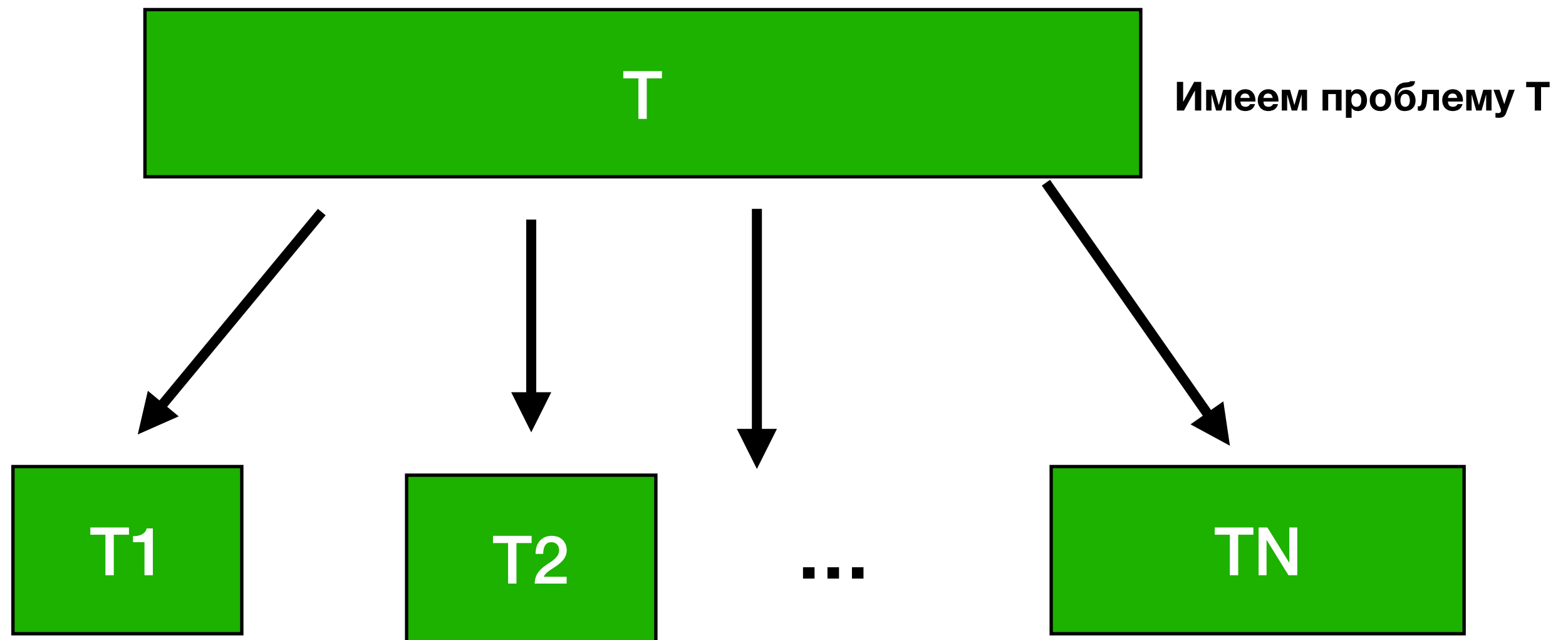
**Как подсчитать число
ненулевых битов в числе?**

**Как узнать чему равен 4
бит в числе?**

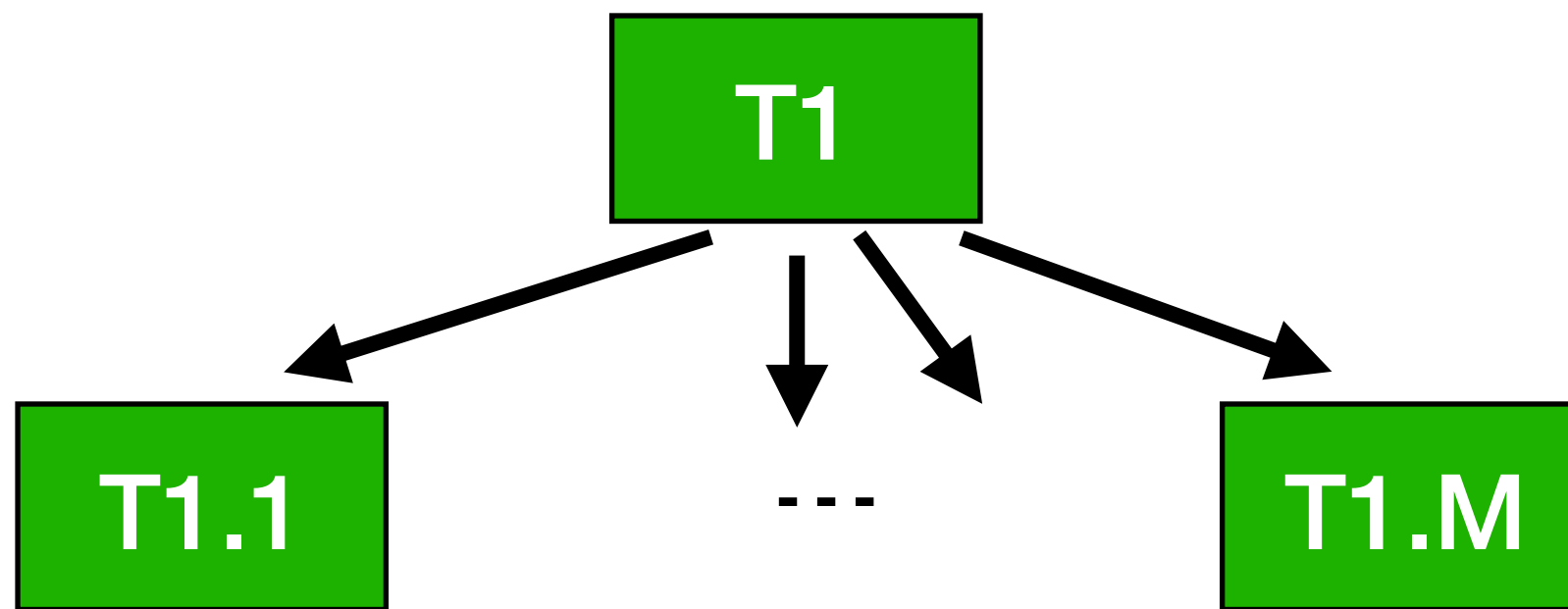
**Как “установить” 4й бит в
числе в значение 1**

**Как “установить” 4й бит в
числе в значение 0**

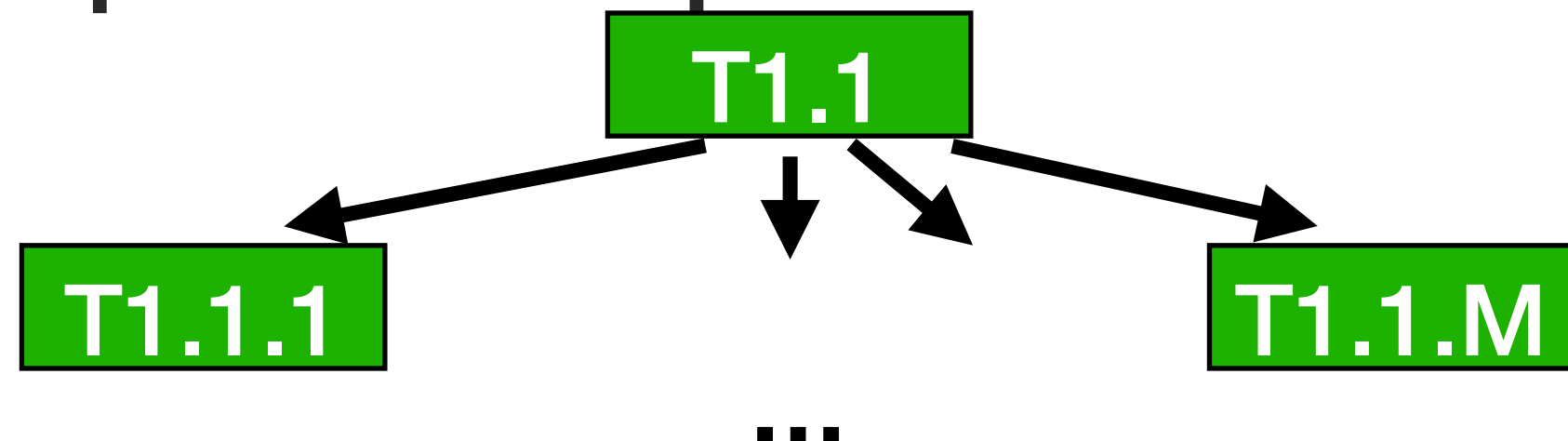
Стратегия “разделяй и властвуй”



Divide - разделим
на несколько проблем поменьше

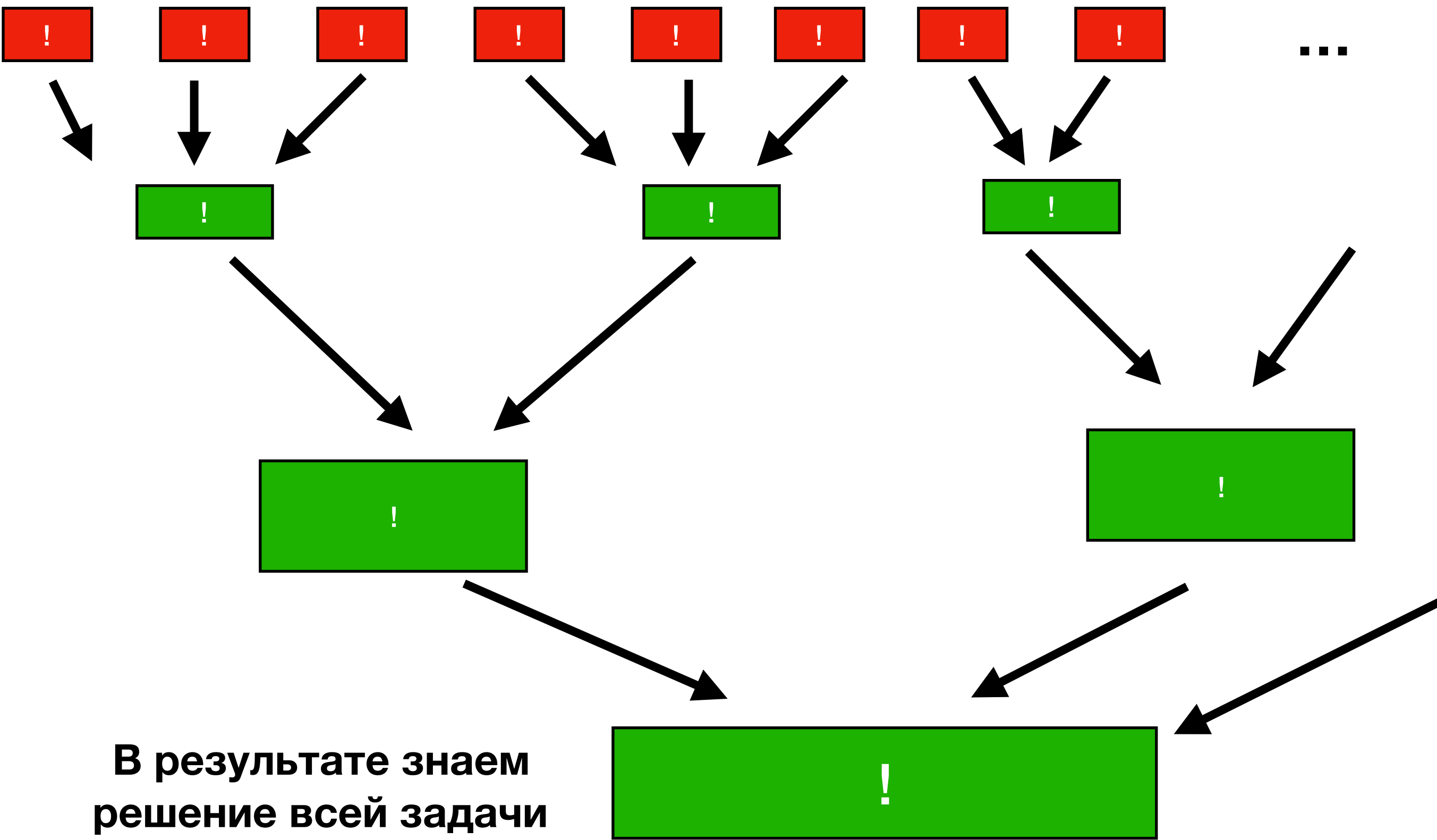


Conquer - будем разбивать получившиеся проблемы на подчасти пока не получим части настолько малые, что для них мы имеем тривиальное решение



Эти проблемы знаем как легко решать

Combine - объединяя ответы на маленькие подзадачи, получаем ответы для больших



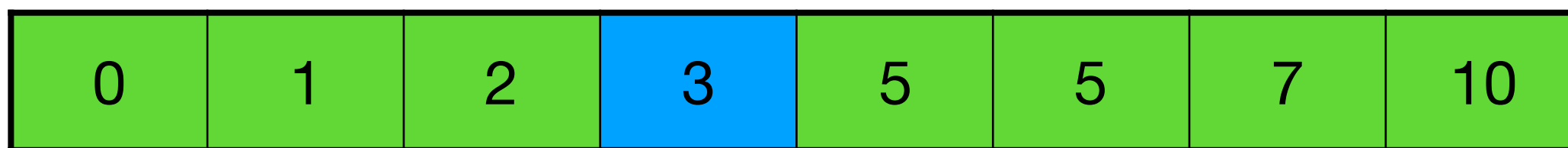


Какой алгоритм из уже изученных в курсе использует эту стратегию?

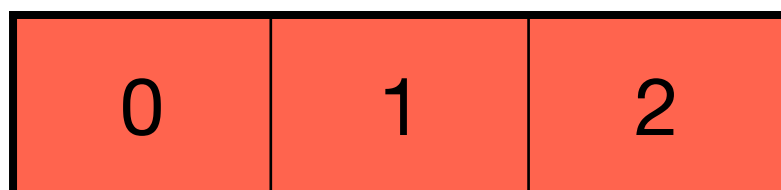


Бинарный поиск

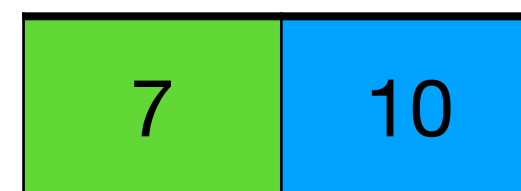
Есть ли 7 в массиве?



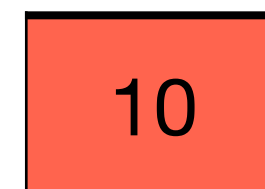
$3 < 7 ?$



$5 < 7 ?$



$7 < 10 ?$



Тривиальное решение:
Так как все элементы здесь меньше
либо равны 3, то 7 среди них нет



Аналогично



Массив длины 1, можем
легко проверить, если ли
искомый элемент просто
сравнив

Аналогично

Бинарный поиск

