Chapter 7

An array is a sequence of values.

An array is a fixed-length sequence of values.

An array is a fixed-length sequence of values, all of the same type.

You can declare, create, access, and update arrays.

Arrays are declared with a base type and [] to denote "array of."

An array variable holds a reference to an array.

(Therefore, you must create the array with operator new.)

You can access or modify the elements in an array, one at a time.

(Like a nerd, the array starts counting at 0.)

Array syntax

Declaring and creating

```
int [] ia = new int[10];
```

Accessing

```
int x = ia[0];
```

Updating

```
ia[0] = x * y;
```

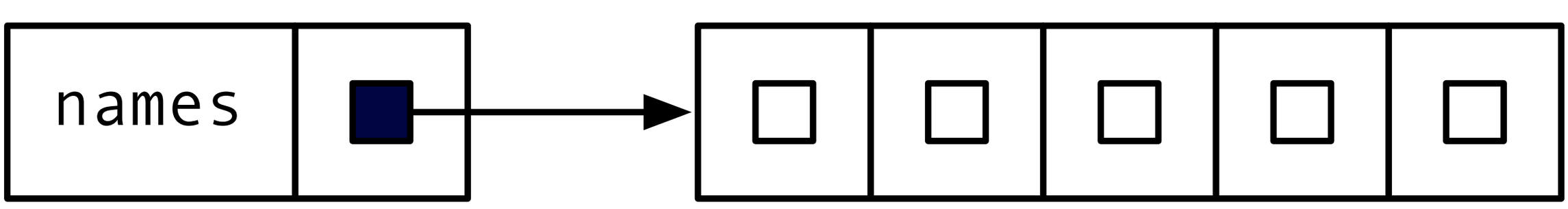

11

Declaring and creating arrays

```
String[] names = new String[5];
```

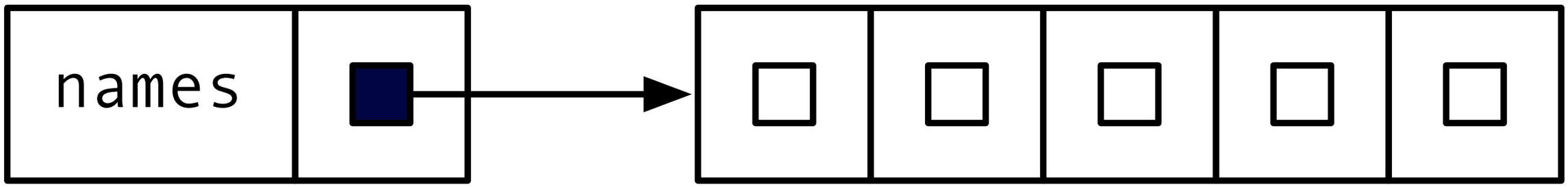
Declaring and creating arrays

```
String[] names = new String[5];
```



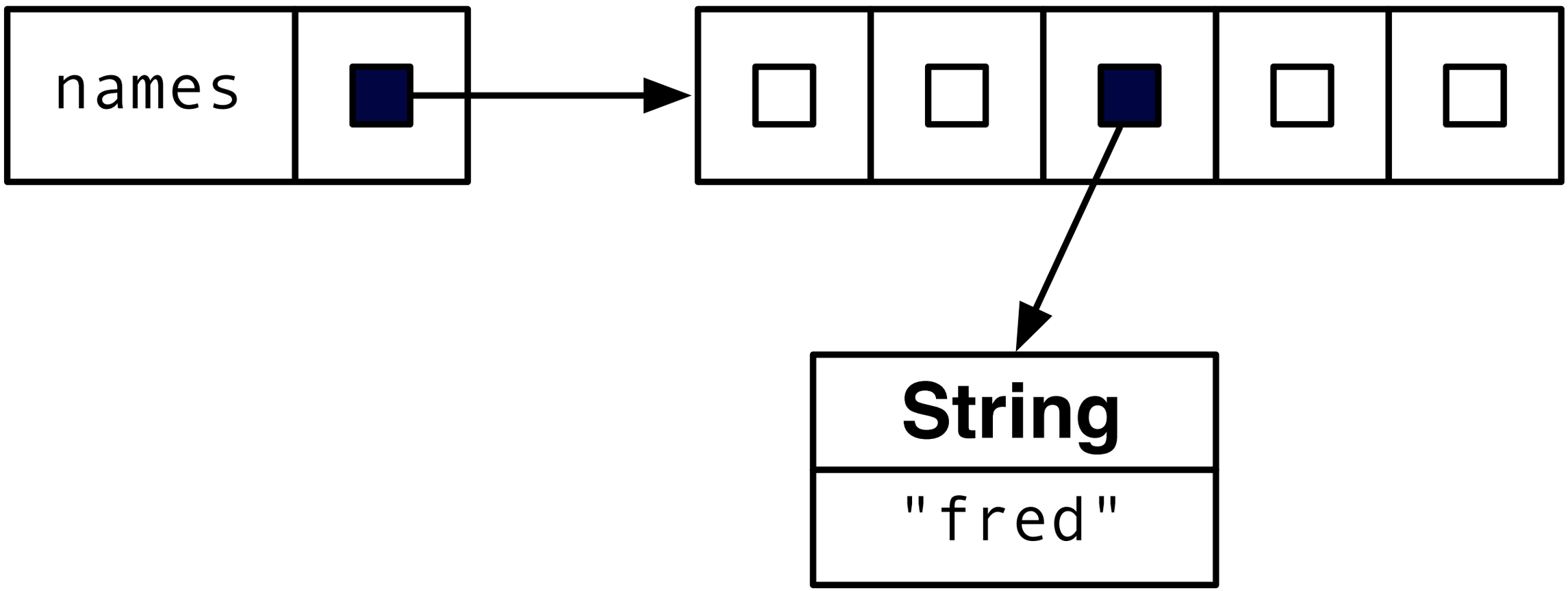
Modifying array elements

```
names[2] = "fred";
```



Modifying array elements

```
names[2] = "fred";
```

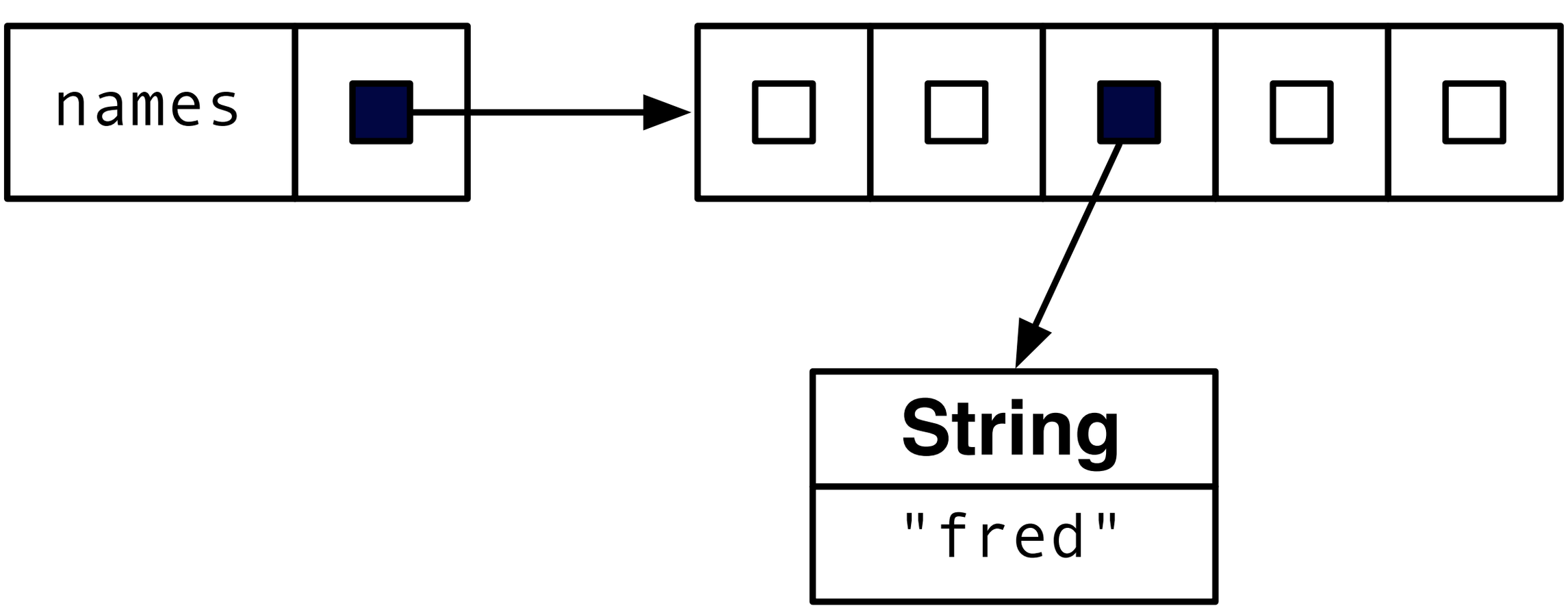


Accessing array elements

```
String fred = names[2];
```

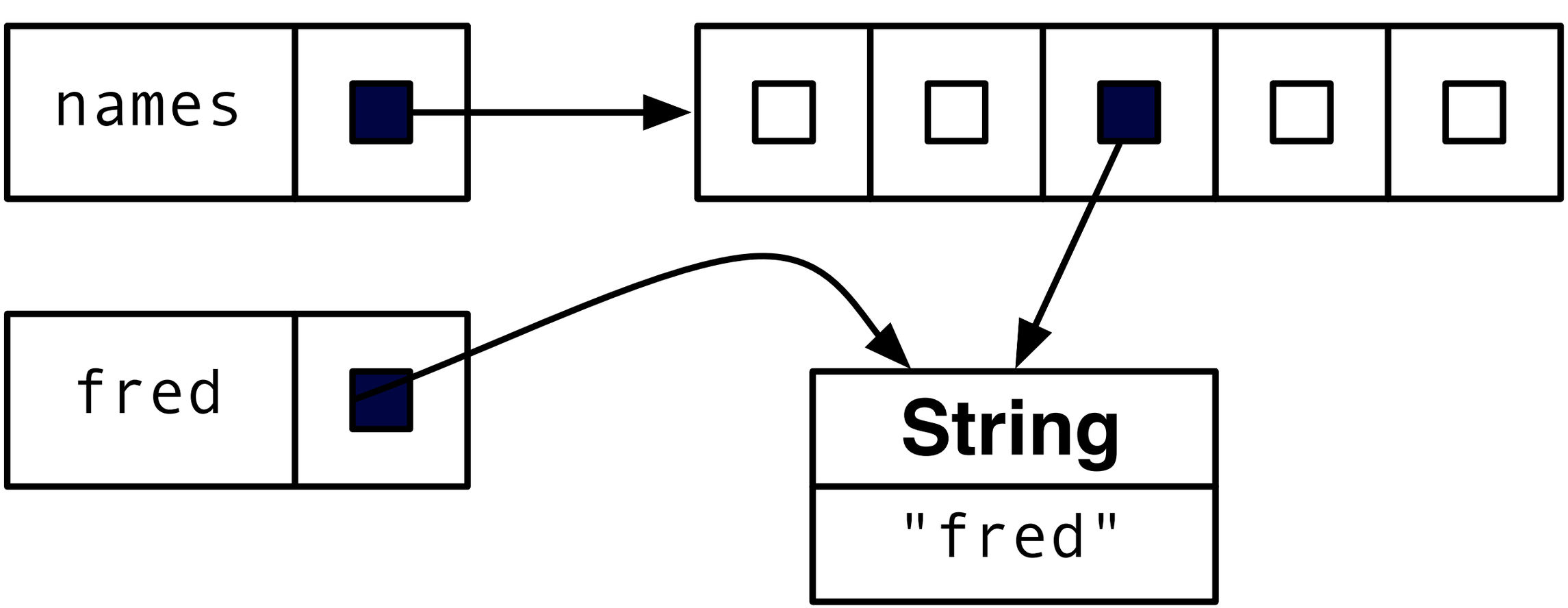
Accessing array elements

```
String fred = names[2];
```



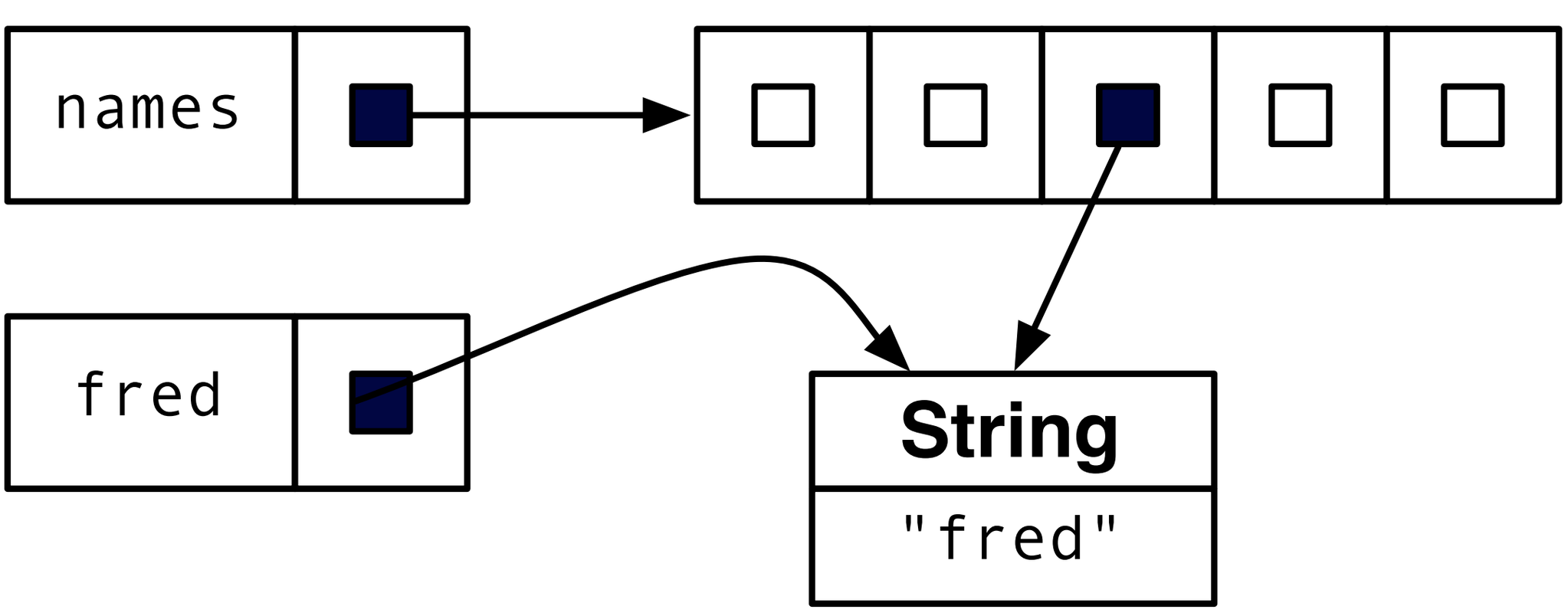
Accessing array elements

```
String fred = names[2];
```



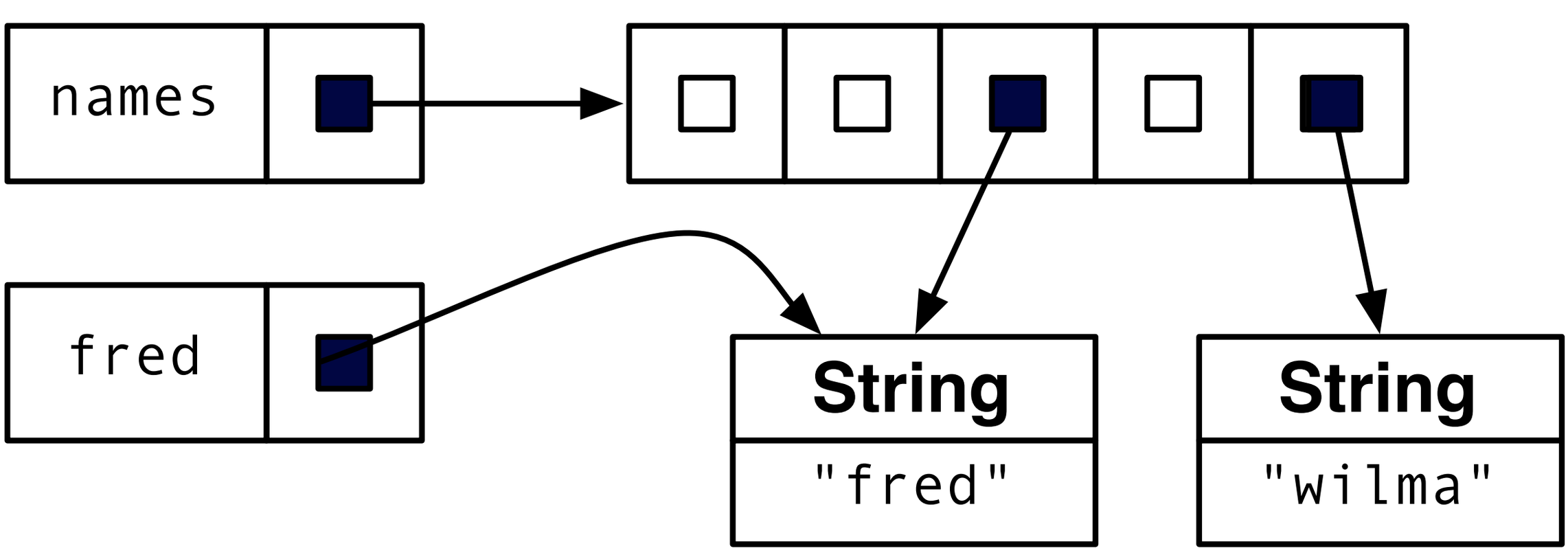
Metadata and .length

```
names[names.length - 1] = "wilma";
```



Metadata and .length

```
names[names.length - 1] = "wilma";
```



What values does a newly-allocated array contain?

You can also ask an array how many elements it has.

Array syntax

Element count

```
ia.length
```

Iterating through an array

```
for (int i = 0; i < ia.length; i++) {
   System.out.println(ia[i]);
}
```


18

Note that the last valid index for some array *a* is (a.length - 1).

(It's because we start counting at zero.)

Exercise

```
/** Returns the sum of all even numbers
    between lo and hi (inclusive).
    @param lo the low end of the range
    @param hi the high end of the range
    @return as above
 */
public static int sumEvens(int lo, int hi) {

}
```


20

```
public static int sumEvens(int lo, int hi) {
    int total = 0;
    int start = (lo % 2 == 0) ? lo : lo + 1;
    for (int i = start; i <= hi; i+=2) {
        total += i;
    }
    return total;
}
```

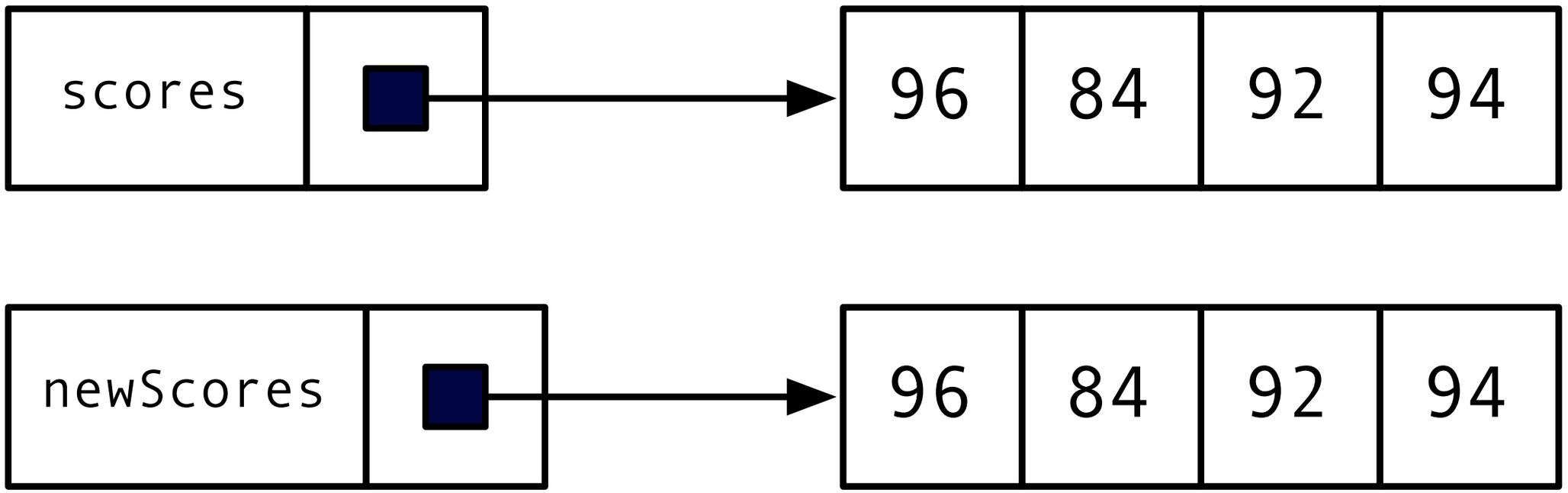
You can copy arrays with a for loop as well

```
int[] newScores = new int[scores.length];

for (int i = 0; i < scores.length; i++) {
  newScores[i] = scores[i];
}
```


23

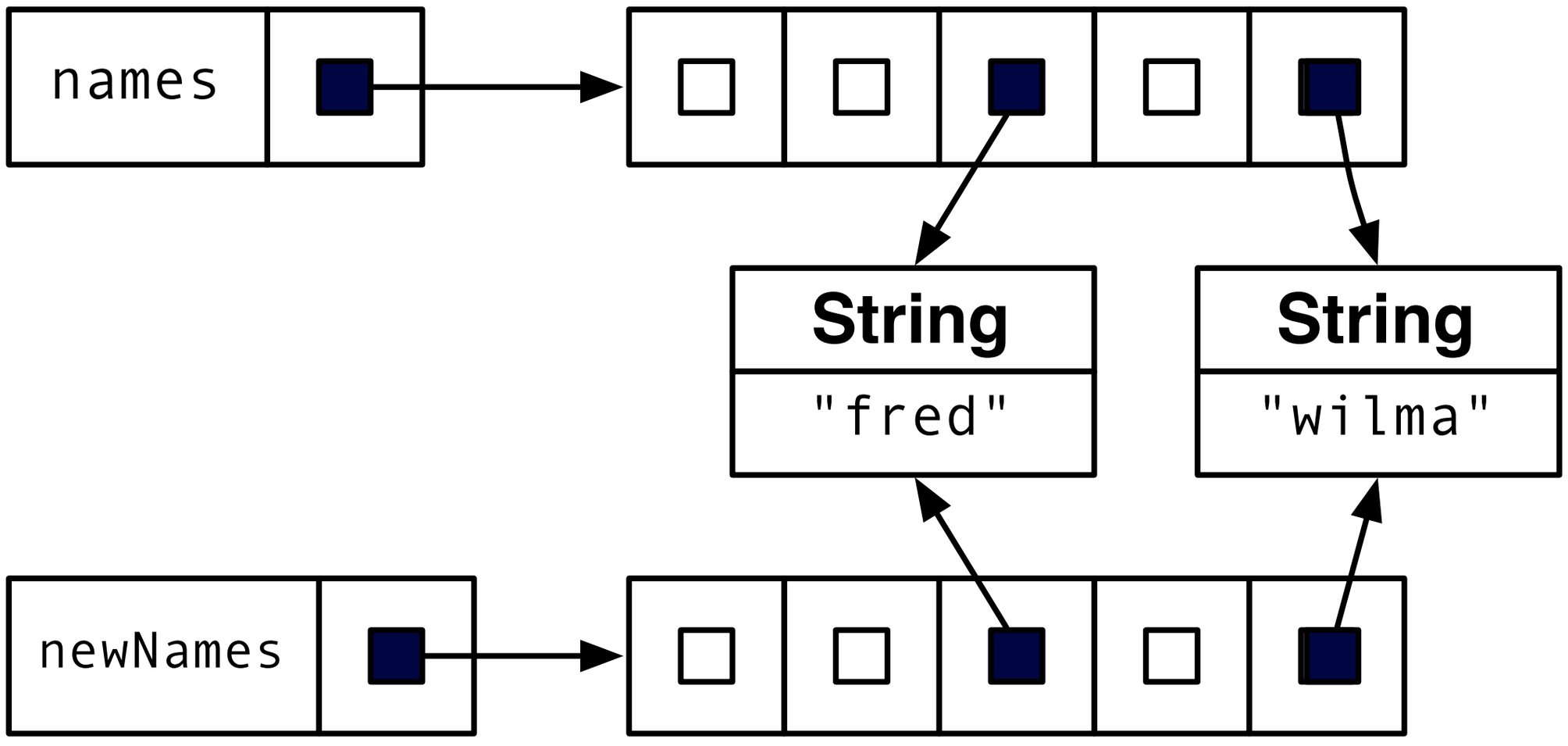
A *shallow copy* copies the values in the array, but does not create new objects for array element referents.

```
int[] newScores = new int[scores.length];

for (int i = 0; i < scores.length; i++) {
  newScores[i] = scores[i];
}
```



25

```
String[] newNames =
    new String[names.length];

for (int i = 0; i < names.length; i++) {
  newNames[i] = names[i];
}
```



26

Exercise

`reverse` takes an array `arr` and returns a newly-allocated array with the elements of `arr` in reverse order.

`reverseInPlace` takes an array `arr` and modifies it by reversing the order of its elements.

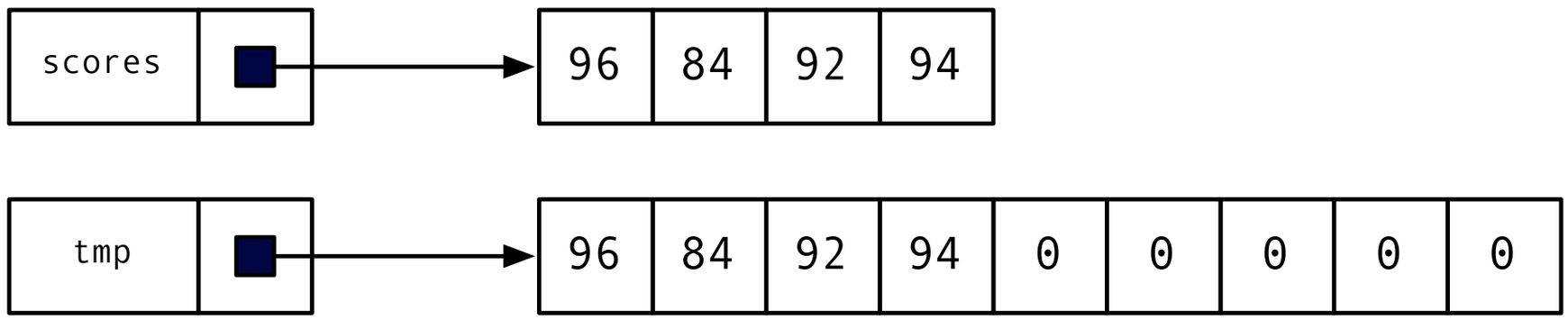
```
int[] reverse(int[] arr)

void reverseInPlace(int[] arr)
```


27

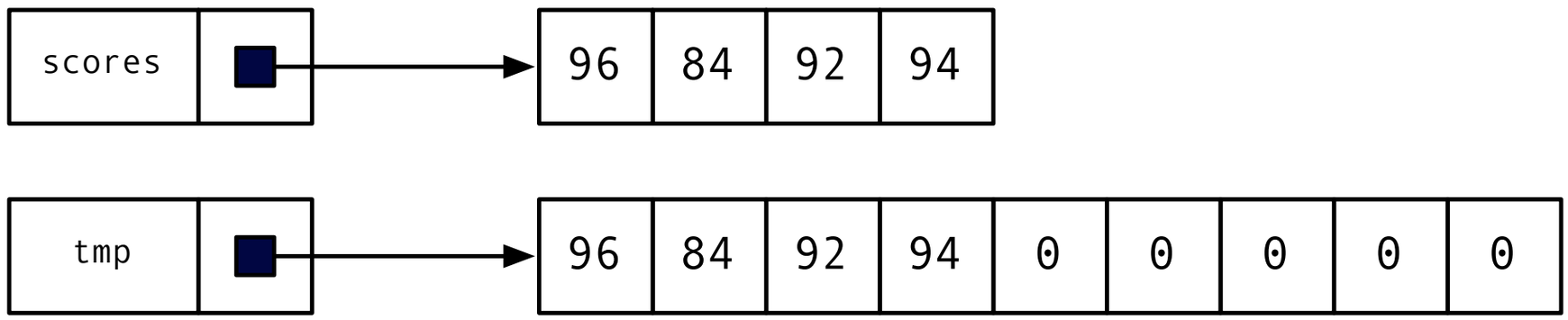
Recall that an array is a *fixed-size* array of values.

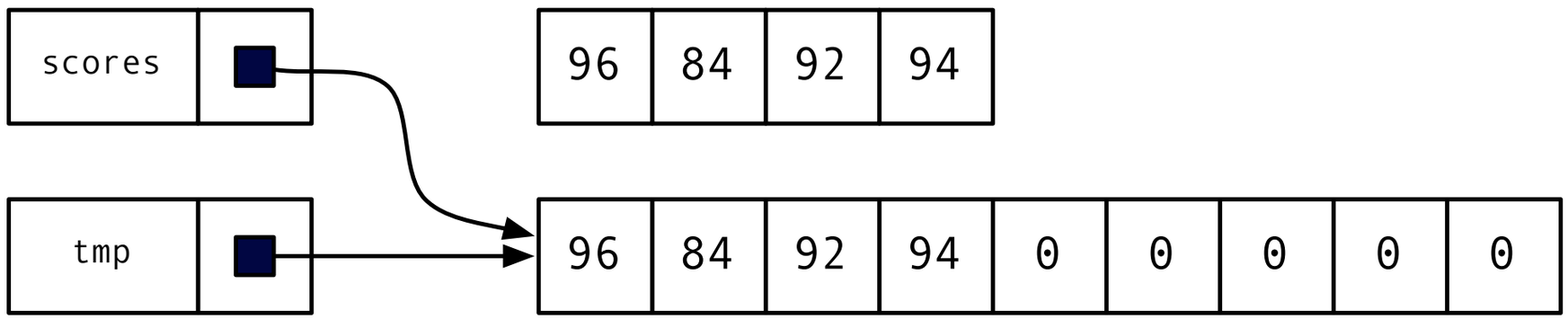
Sometimes we need more space than we thought!

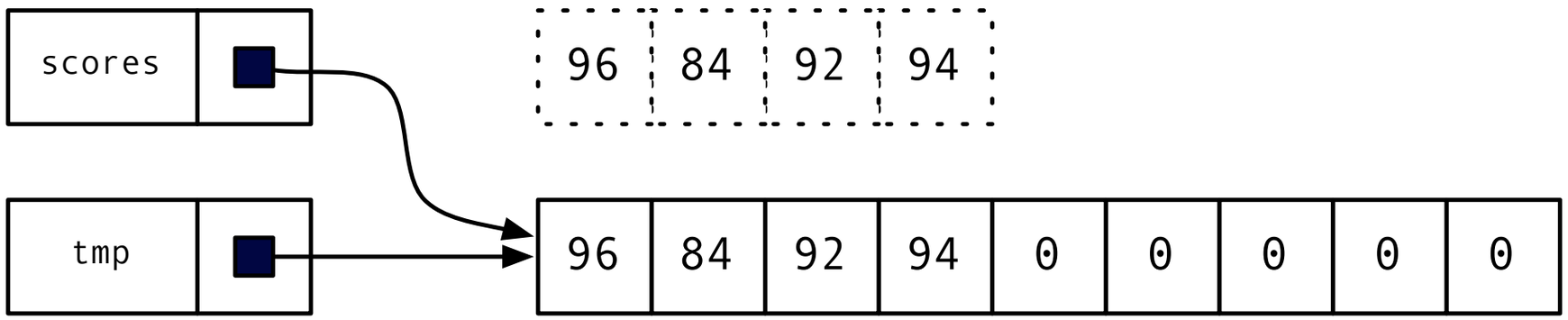
To resize an array, create a new array and copy the elements from the old one.



Then, give a reference to the new array to the old variable.







How would we write the code to do this?

We've seen several kinds of *literals* -- `int` literals, `String` literals, etc.

42

2.0f

We've seen several kinds of *literals* -- `int` literals, `String` literals, etc.

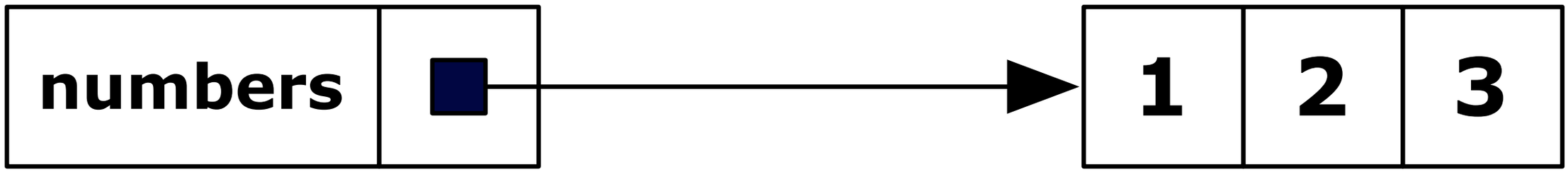
`true`

`"Hello"`

Java also lets us declare *array literals*.

To declare an array literal, write a sequence of values in curly brackets.

```
int[] numbers = {1, 2, 3};
```



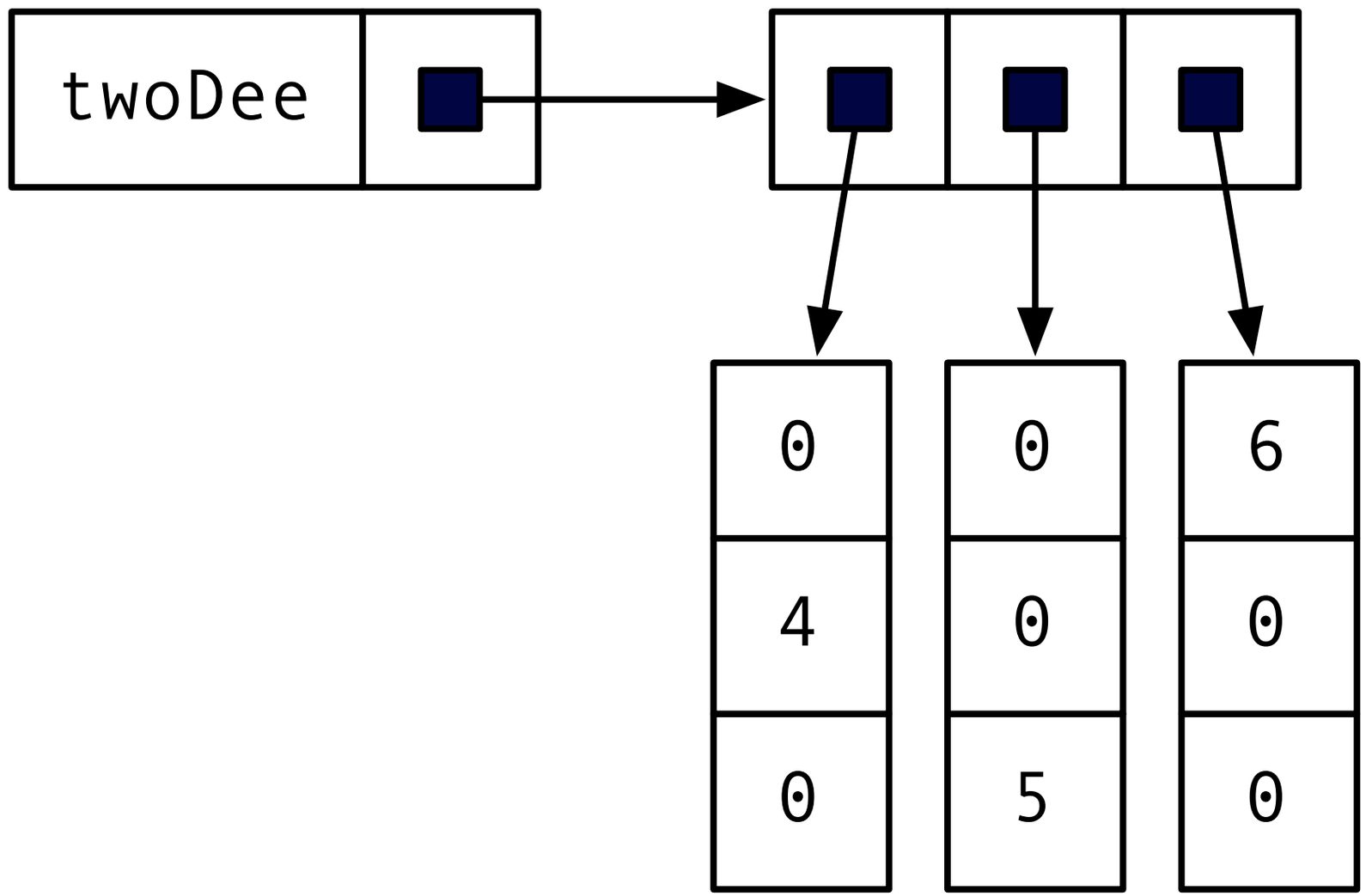
(This syntax only works in initializations -- not everywhere you'd use an array reference!)

Arrays may contain other arrays.

The type of an array is *basetype*[].

We add additional [] to indicate arrays of arrays!

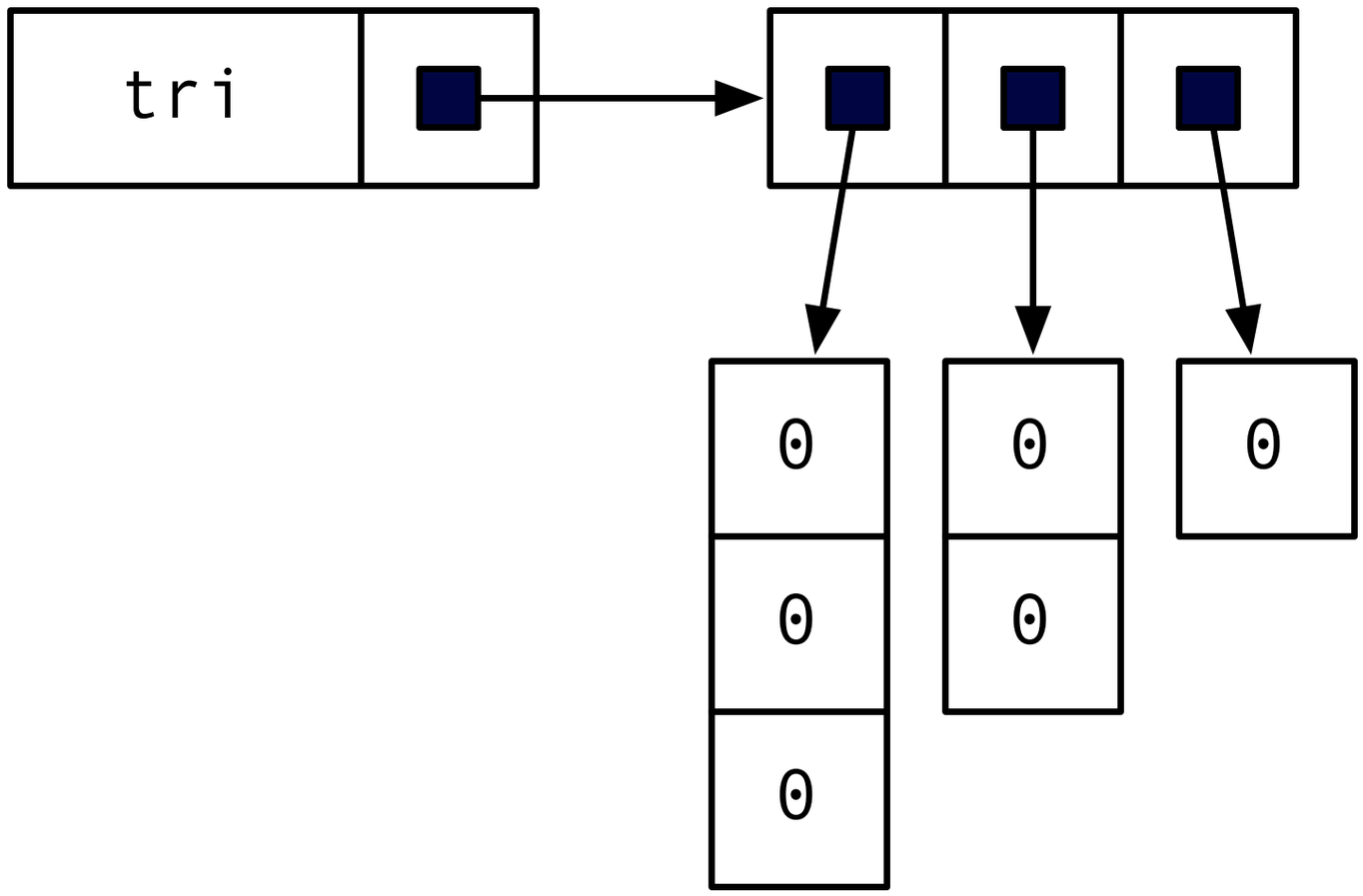
```
int[][] twoDee = new int[3][3];
twoDee[0][1] = 4;
twoDee[1][2] = 5;
twoDee[2][0] = 6;
```



How would we iterate through such an array?

Every sub-array need not have the same length!

```
int[][] tri = new int[3][];
int i;
for (i = 0; i < tri.length; i++) {
   tri[i] = new int[3 - i];
}
```



How would we iterate through such an array?

How to iterate through every element in a jagged array (setting each to 37):

```
// assume int[][] tri

for (int i = 0; i < tri.length; i++)
    for (int j = 0; j < tri[i].length; j++)
        tri[i][j] = 37;
```

Review: how do we write array literals?

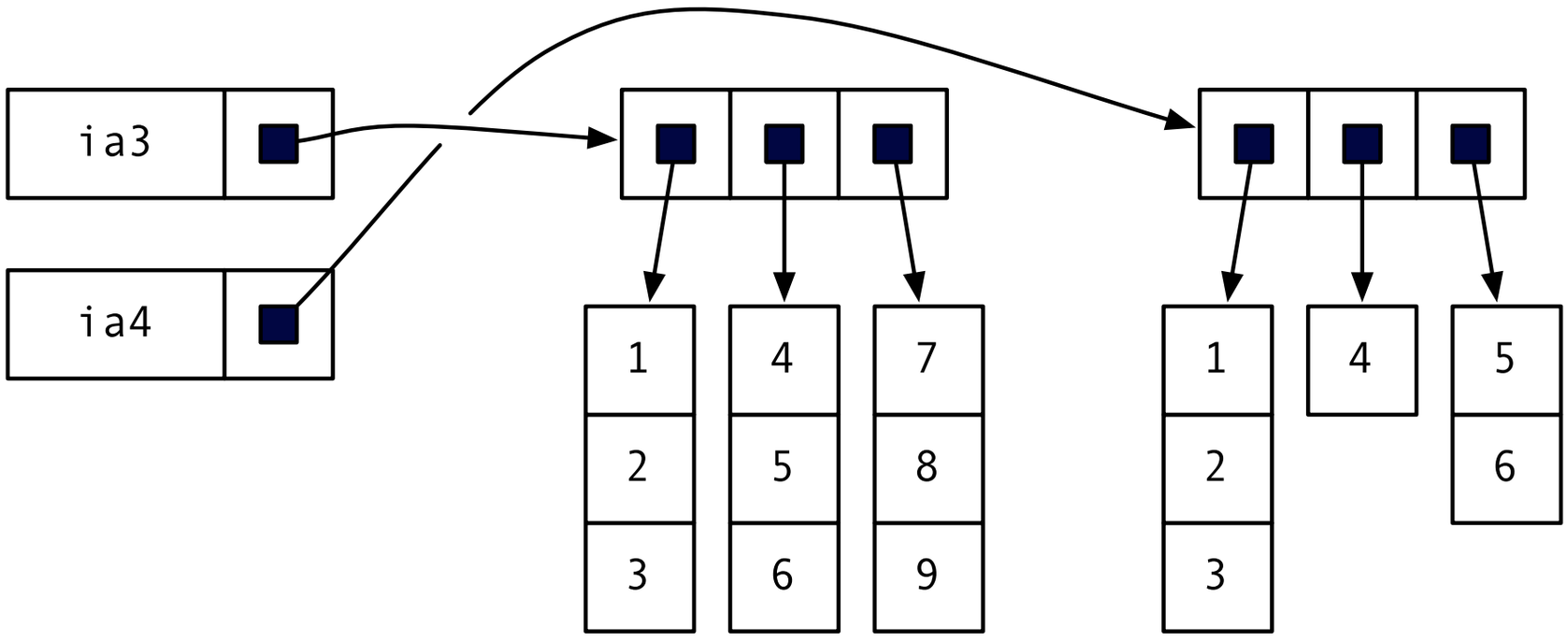
How do you suppose we write multi-dimensional array literals?

How to write multi-dimensional array literals

```
int[][] ia3 = {{1, 2, 3},
               {4, 5, 6},
               {7, 8, 9}};

int[][] ia4 = {{1, 2, 3}, {4}, {5, 6}};
```

How to write multi-dimensional array literals

```
int[][] ia3 = {{1, 2, 3},
               {4, 5, 6},
               {7, 8, 9}};

int[][] ia4 = {{1, 2, 3}, {4}, {5, 6}};
```



2D puzzle

- Write a method that visits each element of a 2D array with only one `for` loop, assigning some value to each

50

Let's say we want to write this code without a nested `for` loop.

```
public static void fillArray1(int[][] arg) {
    int k = 0;

    for (int i = 0; i < arg.length; i++) {
        for (int j = 0; j < arg[i].length; j++) {
            arg[i][j] = k++;
        }
    }
}
```

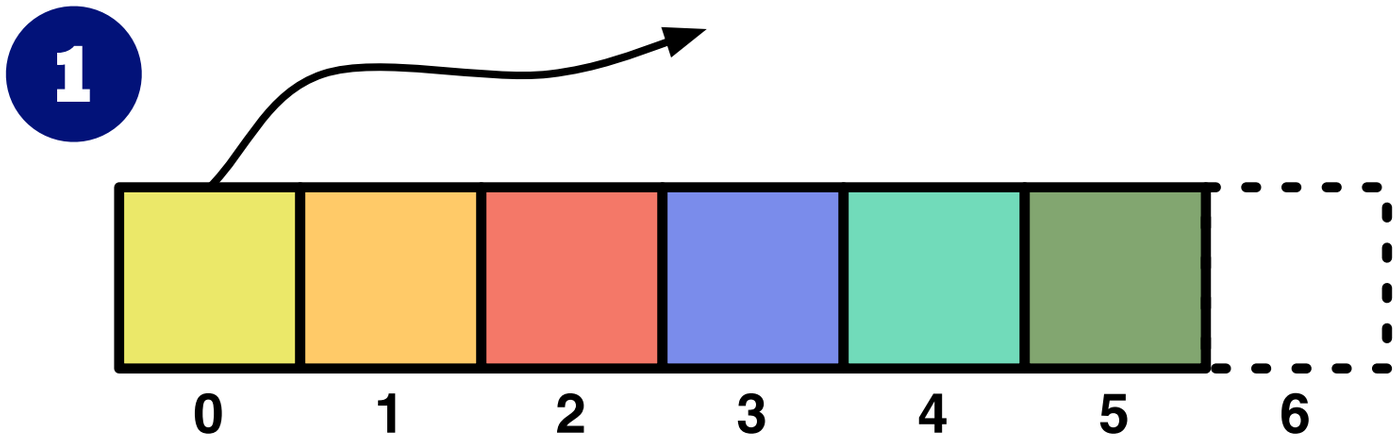
```
public static void fillArray(int[][] arg) {
    final int X = arg.length;
    final int Y = arg[0].length;

    for (int k = 0; k < X * Y; k++) {
        int x = k / Y;
        int y = k % Y;
        arg[x][y] = k;
    }
}
```

```
public static void fillArray(int[][] arg) {
    final int X = arg.length;
    final int Y = arg[0].length;

    for (int k = 0; k < X * Y; k++) {
        int x = k / Y;
        int y = k % Y;
        arg[x][y] = k;
    }
}
```

(What assumptions does this code make about `arg`?)

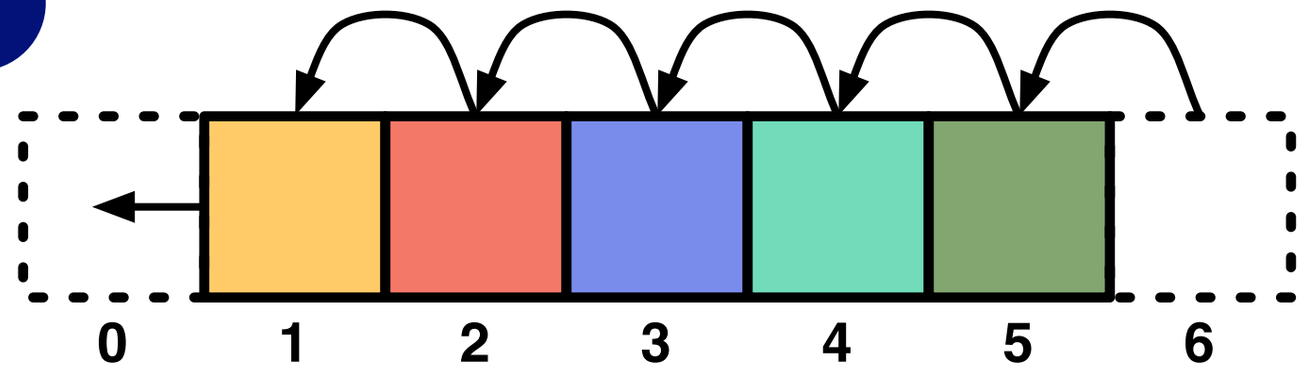
Some things are a little tricky with arrays, like inserting and removing elements.

What does your program need to do to remove the first element from an array?



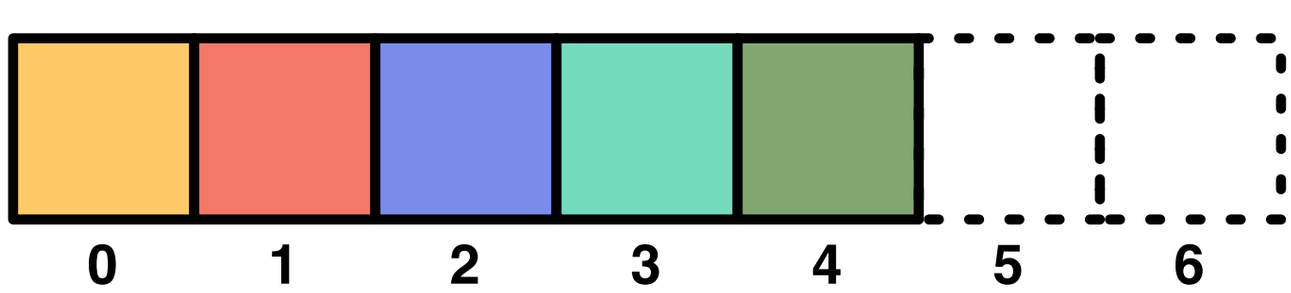
55

2



55

3



55

How would you write code to remove an element from an array?

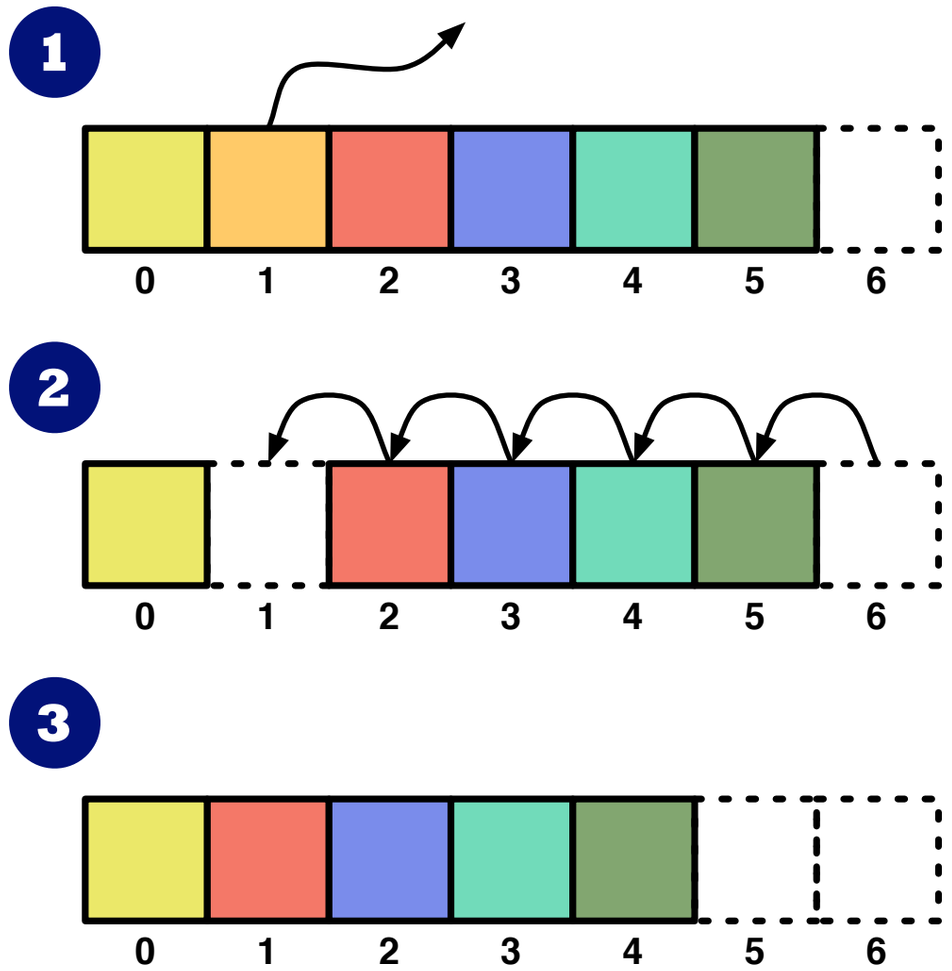
(What about inserting an element?)

Removing elements

- In general, to remove element `n`, shift every element to the right of `n` to its left

- Be sure to add an additional `null` at the end!

- What would this code look like?

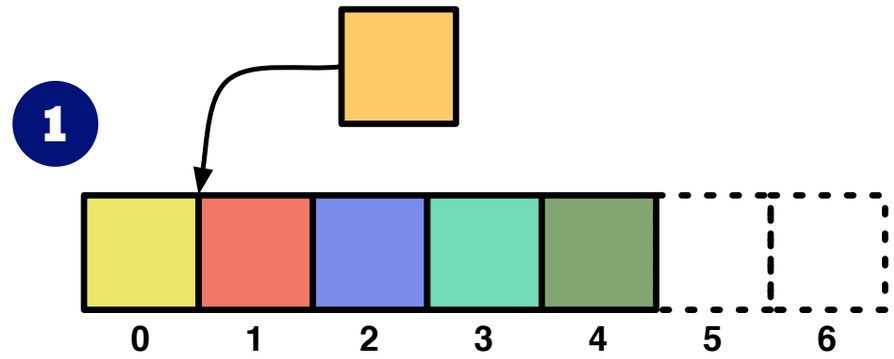


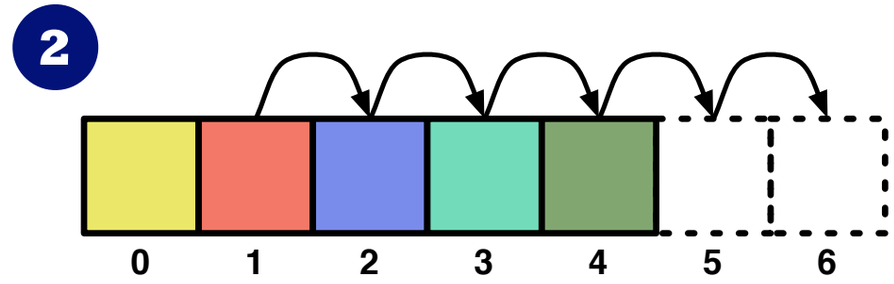
57

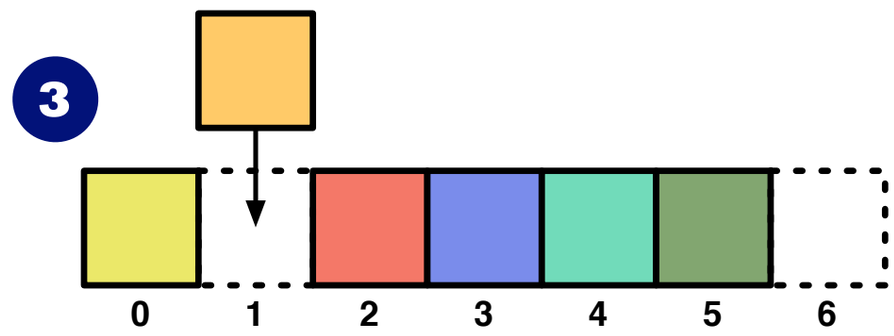
```
public static String remove(int elt, String[] arr) {
    String ret = arr[elt];
    for (int i = elt + 1; i < arr.length; i++) {
        arr[i-1] = arr[i];
    }
    arr[arr.length] = null;
    return ret;
}
```

Inserting elements

- To insert an element before element n, shift every element from n on to the right







- You may have to resize the array if there's not enough space!

- **What would this code look like?**

59

Partially-filled arrays

- Sometimes, not every element of an array will contain a useful value
  - "extra space" at end
  - "holes" in middle
- How do we deal with these situations?

60

Metadata

- Metadata is "data about data"
- "data" is what's in your array
- "metadata" is information about your array
  - e.g. `length` field
- We can use metadata to handle partially-filled arrays

61

Contiguous arrays

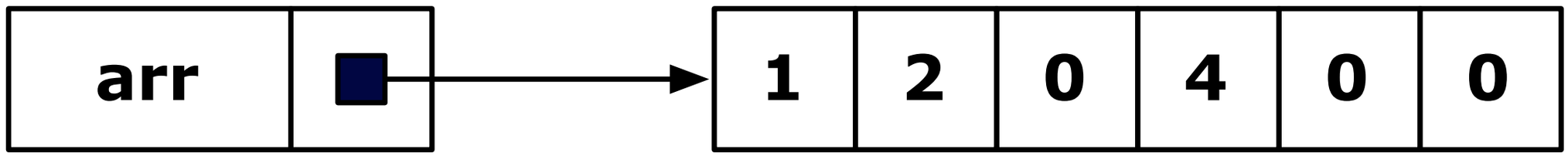
- Like a deck of cards in a box: you remove an element, but there are no “holes” -- you just aren’t using the whole box
- **How do we keep track of such an array?**

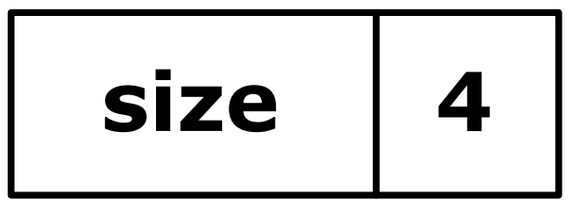
62

Idea

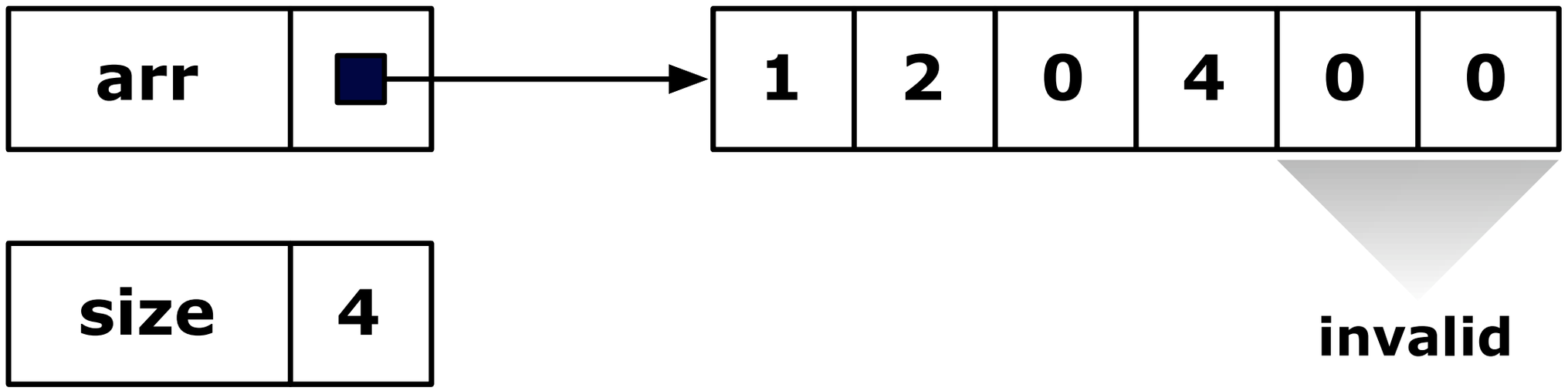
- Maintain a "size" variable along with the array
  - or, better yet, an instance field
- Everything from `arr[size]` to `arr[arr.length]` is not valid
- Must update size when inserting, removing, or adding elements!

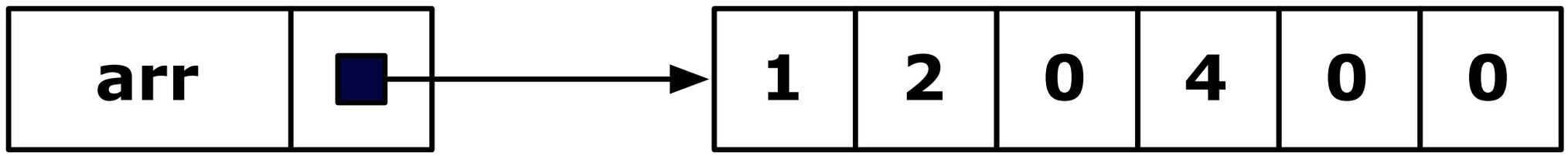
63

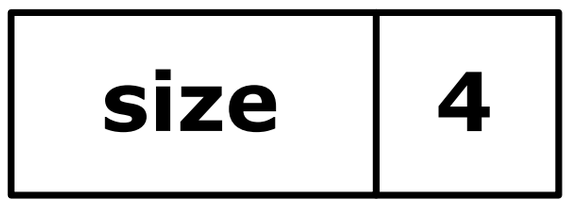




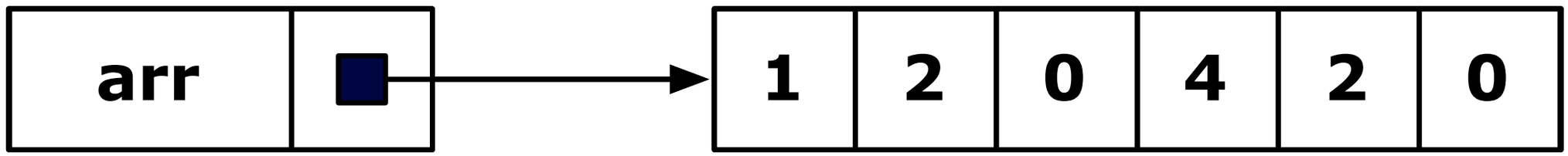
64

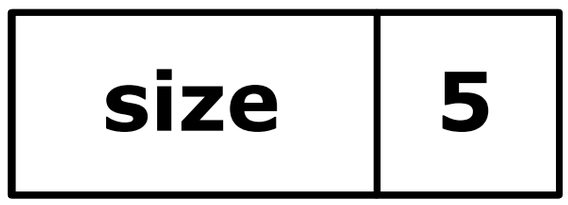






64



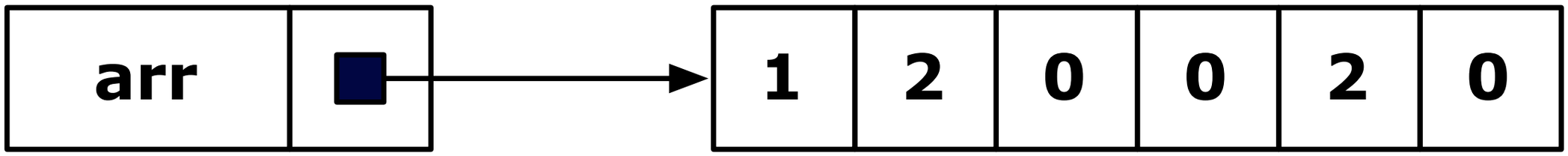


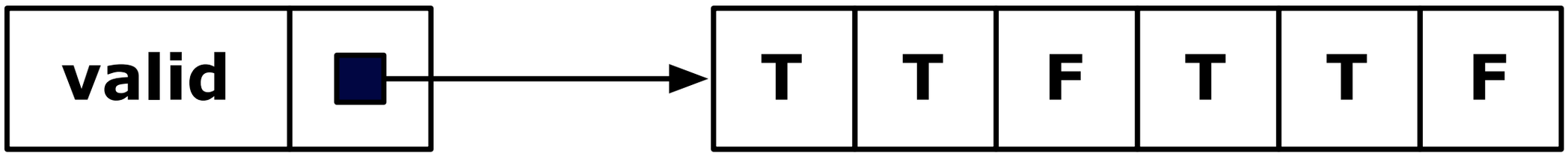
Of course, it would be best to package these data up in a class!

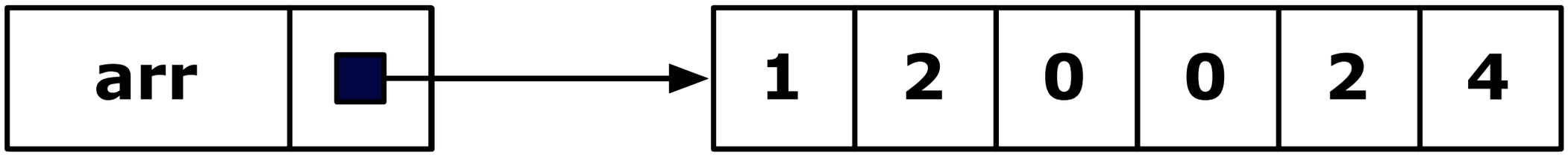
Sparse arrays

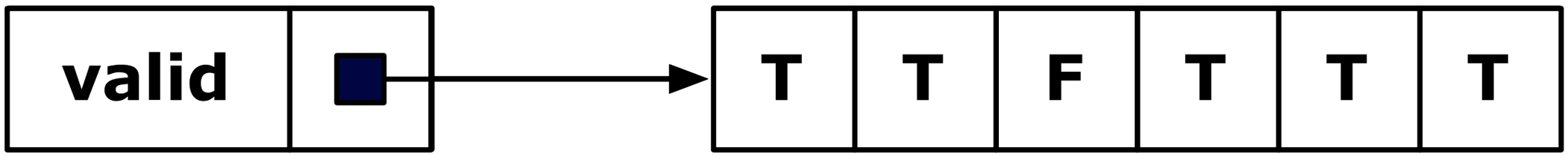
- Like a parking lot -- when you remove something, it creates a hole
- **What problems do we have to solve?**
- **How should we solve them?**

66









67