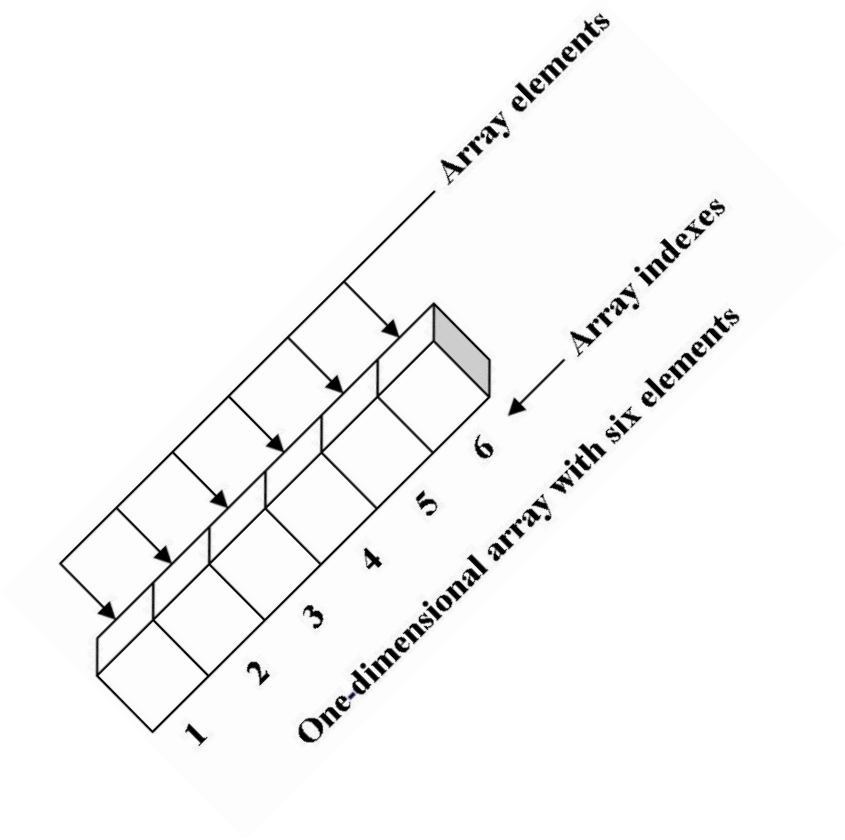


Java Arrays

Computer Science OOD
Boston University

Christine Papadakis

Back to Our Simple Java Program

```
import java.util.*;

public class Play {
   public static void main( String [][] args ) {
      Scanner scan = new Scanner( System.in );
      int num1 = scan.nextInt();
      int num2 = scan.nextInt();
      int num3 = scan.nextInt();

      // Allow user to see which number was entered









   }
}
```

Back to Our Simple Java Program

```
import java.util.*;

public class Play {
   public static void main( String [][] args ) {
      Scanner scan = new Scanner( System.in );
      int num1 = scan.nextInt();
      int num2 = scan.nextInt();
      int num3 = scan.nextInt();

      System.out.println("Show which number ?" );
      int number = scan.nextInt();

                        .
                        .
                        .
                        .
                        .


   }
}
```

Our Simple Java Program

```
import java.util.*;

public class Play {
   public static void main( String [][] args ) {
      Scanner scan = new Scanner( System.in );
      int num1 = scan.nextInt();
      int num2 = scan.nextInt();
      int num3 = scan.nextInt();

      System.out.println("Show which number ?" );
      int number = scan.nextInt();

      System.out.printl("Number entered is: ");
      if (number == 1 )
          System.out.println(num1);
      else if (number == 2 )
         System.out.println(num2);
      else
         System.out.println(num3);
   }
}
```

Our Simple Java Program

```
import java.util.*;

public class Play {
   public static void main( String [][] args ) {
      Scanner scan = new Scanner( System.in );
      int num1 = scan.nextInt();
      int num2 = scan.nextInt();
      int num3 = scan.nextInt();

      System.out.println("Show which
      int number = scan.nextInt();

      System.out.printl("Number entered is: ");
      if (number == 1 )
          System.out.println(num1);
      else if (number == 2 )
         System.out.println(num2);
      else
         System.out.println(num3);
   }
}
```

The problem is that each memory cell is mapped to a specific variable name.

Our Simple Java Program with an array

```
import java.util.*;

public class Play {
   public static void main( String [][] args ) {
      Scanner scan = new Scanner( System.in );
      int [] nums = int[3];

      nums[0] = scan.nextInt();
      nums[1] = scan.nextInt();
      nums[2] = scan.nextInt();

      System.out.println("Show which number ?" );
      int number = scan.nextInt();

      System.out.printl("Number entered is ");
      nums[number-1];

   }
}
```

Can access any one of the three integers entered through the *array* variable **nums** by using an offset.

Arrays

An Array is a *fixed* data structure.

It is a *container* that holds
a fixed number of elements of the same data type.

The length of an array is established when
the array is declared and
the physical size of the array
cannot be altered during run time.

Arrays

Specifically:

An array variable references
a *contiguous* memory allocation of
some fixed number of elements
of the same data type.

Why is this important?

This is why arrays are more efficient and
require less overhead than Python lists!

Contiguous allocation means that all the elements
of the array are stored in consecutive memory cells.

The fact that all the elements are of the *exact data type* means
that each element is allocated the same number of bytes.

Therefore if we know the address location of the first element,
we can create an offset from that starting address and
directly access every element.

Array Variables

- We use a variable to represent the array as a whole.

- Example of declaring an array *variable*:

```
int[] temps;
```

  - the `[]` indicates that variable temps represents an array
  - the `int` indicates that the elements will be of type `int`
  - note that the array itself has not been allocated, only the *variable that will reference* the array has been declared.

Array Variables

- We use a variable to represent the array as a whole.

- Example of declaring an array *variable*:

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

  - the `[]` indicates that variable temps represents an array
  - the `int` indicates that the elements will be of type `int`
  - creates an array in memory, *initialized with the specified elements*, and assigns the memory location (of the first element) to variable temps.

Array Variables

- We use a variable to represent the array as a whole.

- Example of declaring an array *variable*:

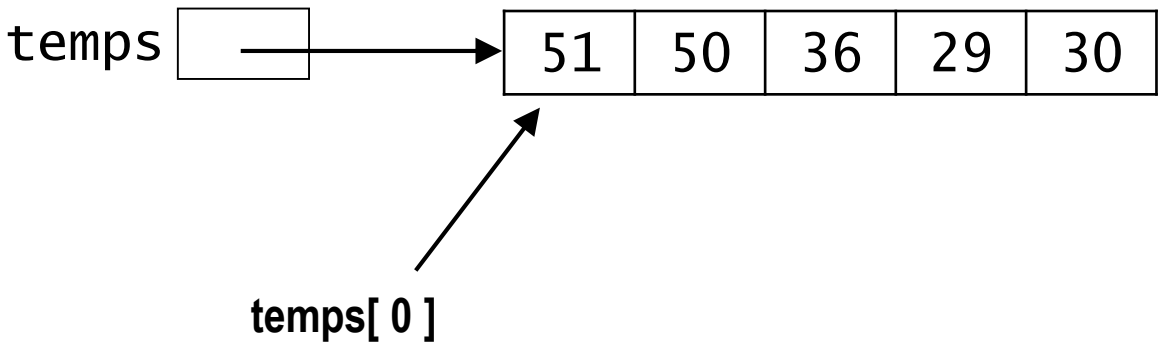
```
int[] temps = new int[5];
```

  - the `[]` indicates that variable temps represents an array
  - the `int` indicates that the elements will be of type `int`
  - creates an array of five elements (initialized to default value) in memory and assigns the memory location (of the first element) to variable temps.

Arrays and References

- An array variable does *not* store the array itself.
- It stores a *reference* to the array.
  - the memory address of the array

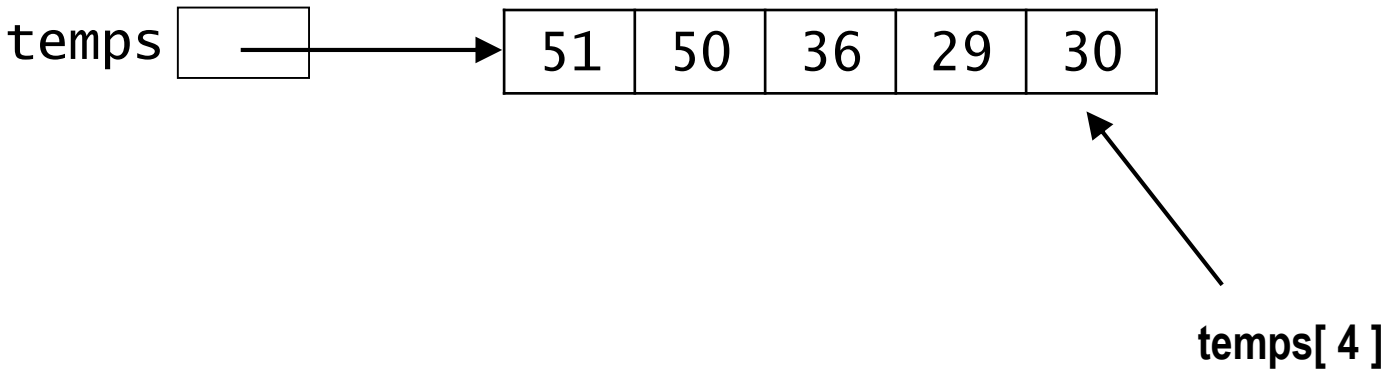
```
int[] temps = {51, 50, 36, 29, 30};
```



Arrays and References

- An array variable does *not* store the array itself.
- It stores a *reference* to the array.
  - the memory address of the array

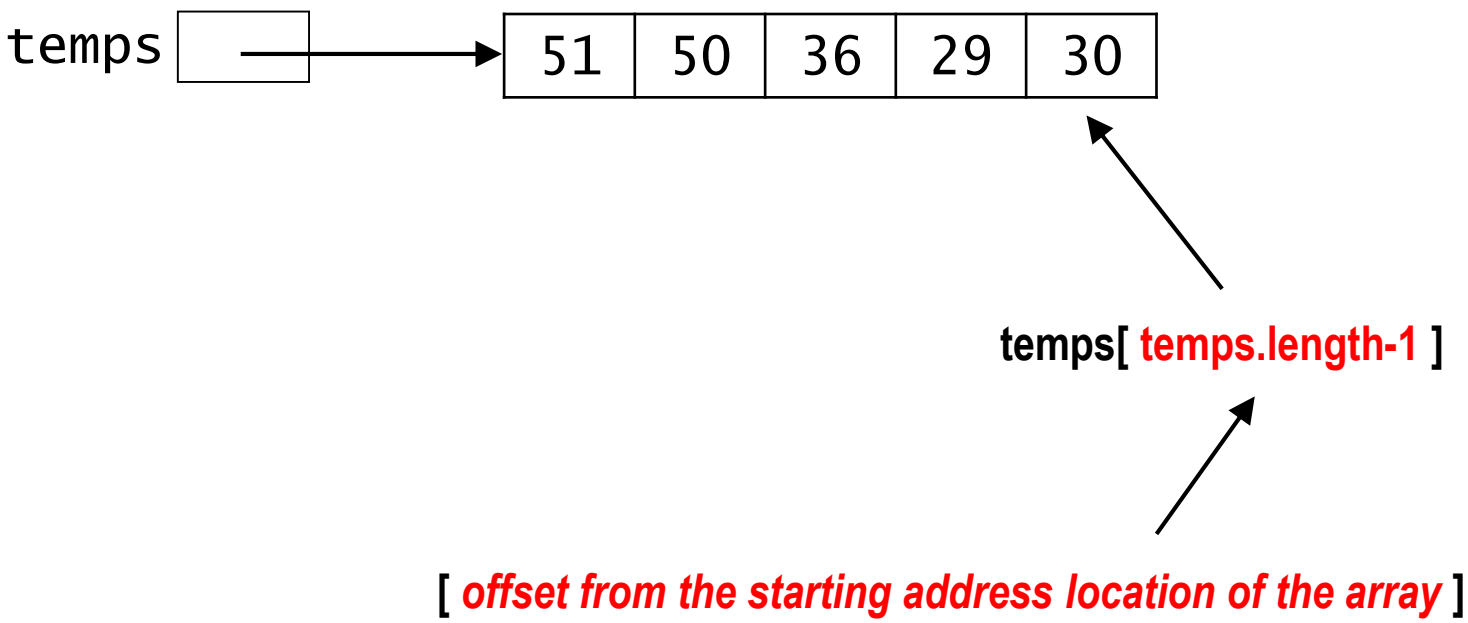
```
int[] temps = {51, 50, 36, 29, 30};
```



Arrays and References

- An array variable does *not* store the array itself.
- It stores a *reference* to the array.
  - the memory address of the array

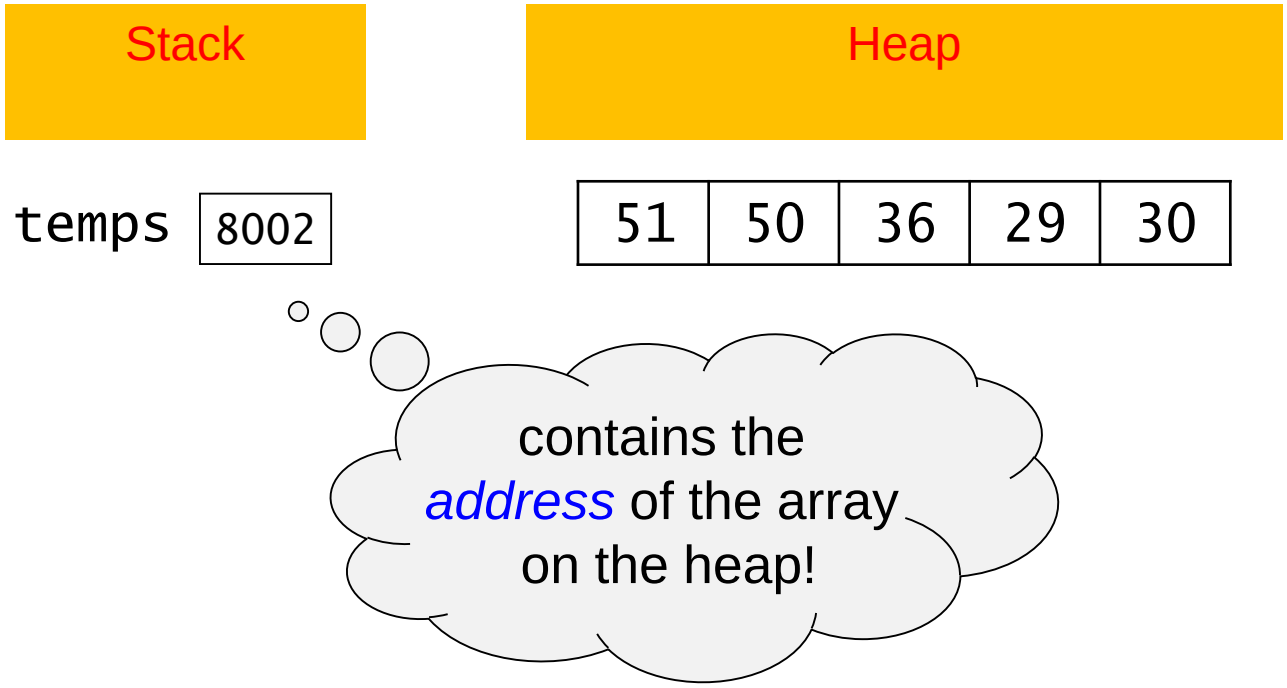
```
int[] temps = {51, 50, 36, 29, 30};
```



Arrays and References

- An array variable does *not* store the array itself.
- It stores a *reference* to the array.
  - the memory address of the array

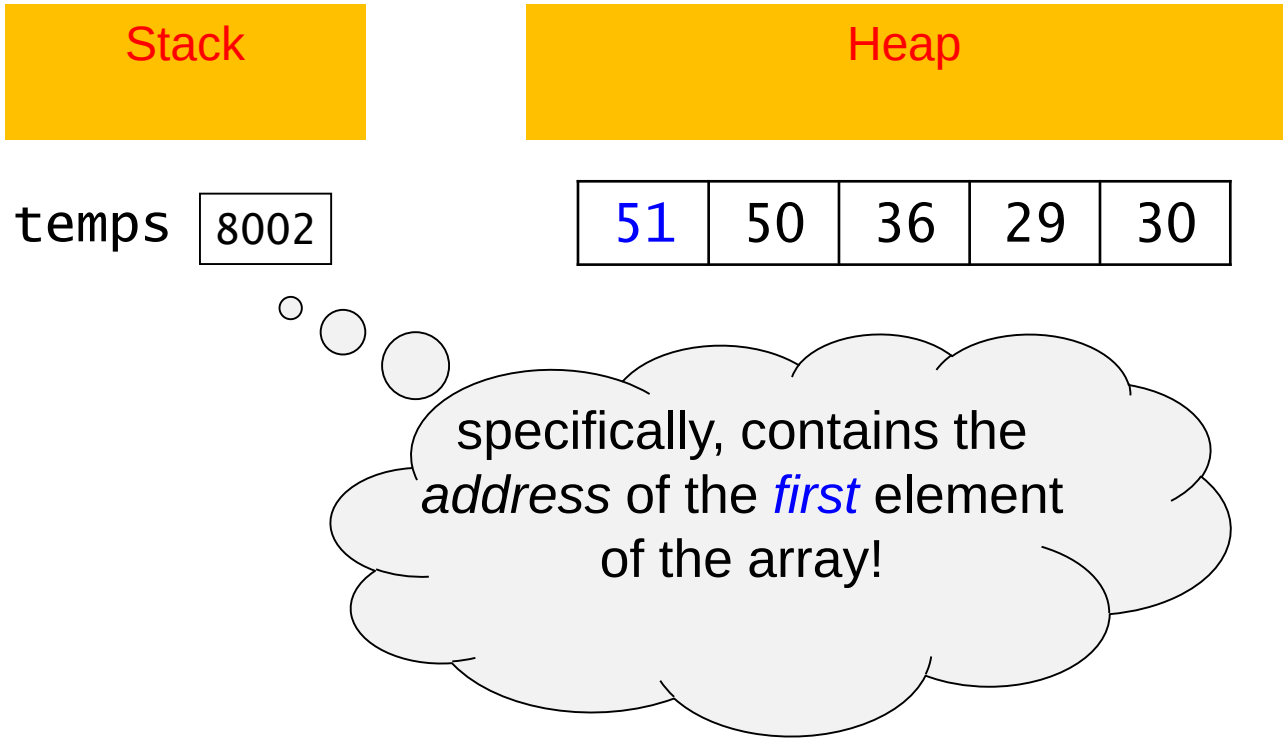
```
int[] temps = {51, 50, 36, 29, 30};
```



Arrays and References

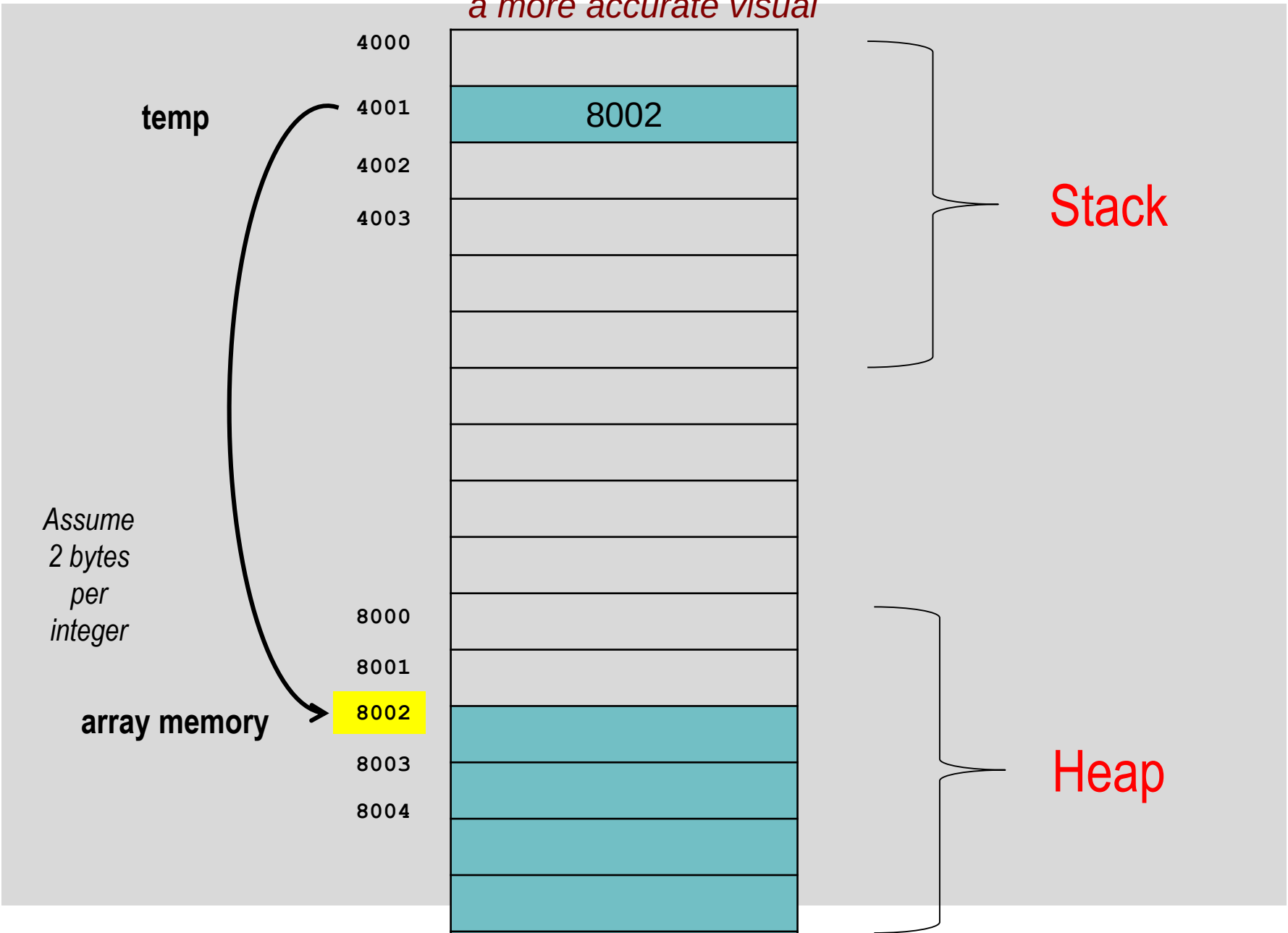
- An array variable does *not* store the array itself.
- It stores a *reference* to the array.
  - the memory address of the array

```
int[] temps = {51, 50, 36, 29, 30};
```



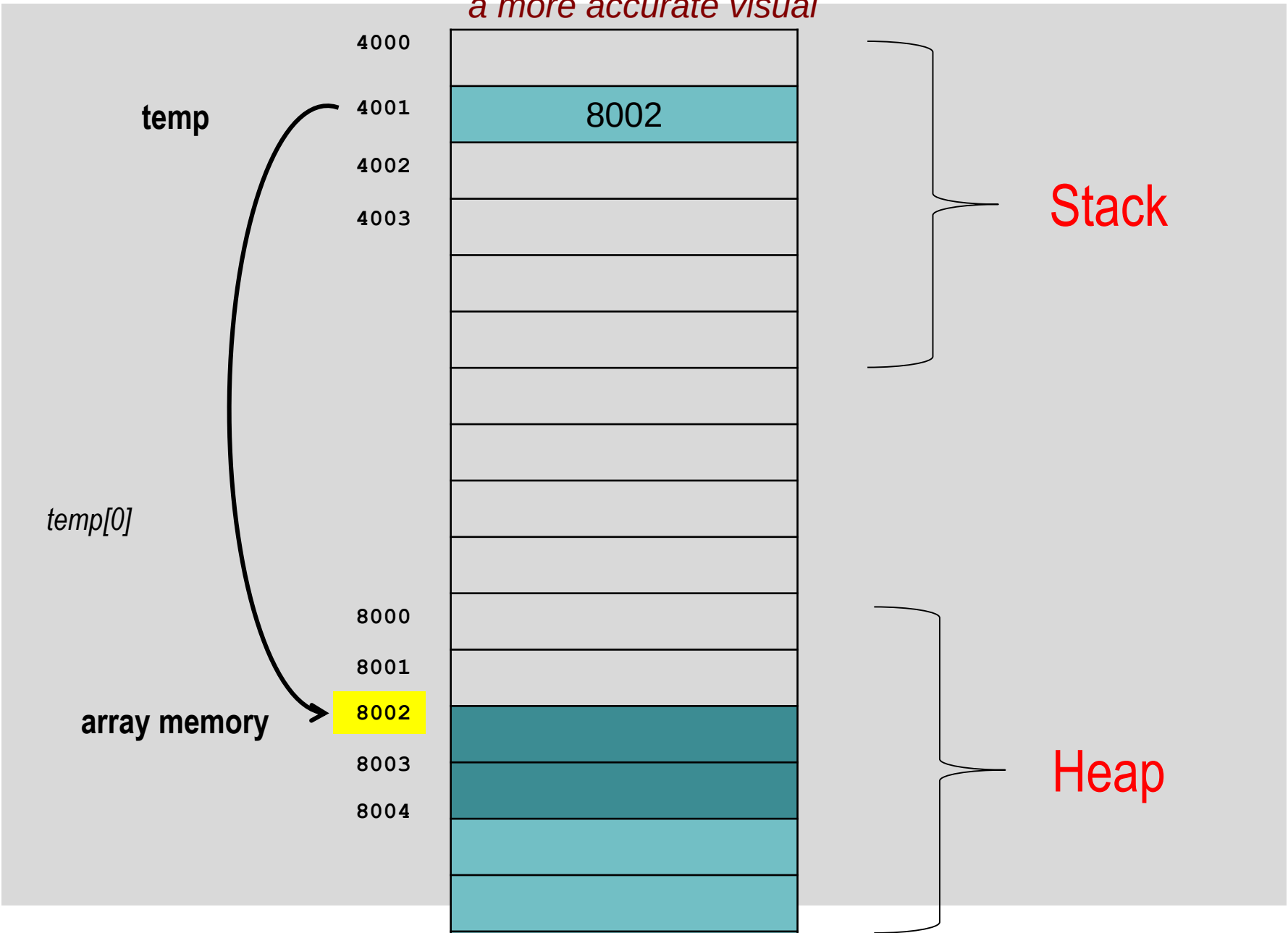
Arrays and References:

a more accurate visual



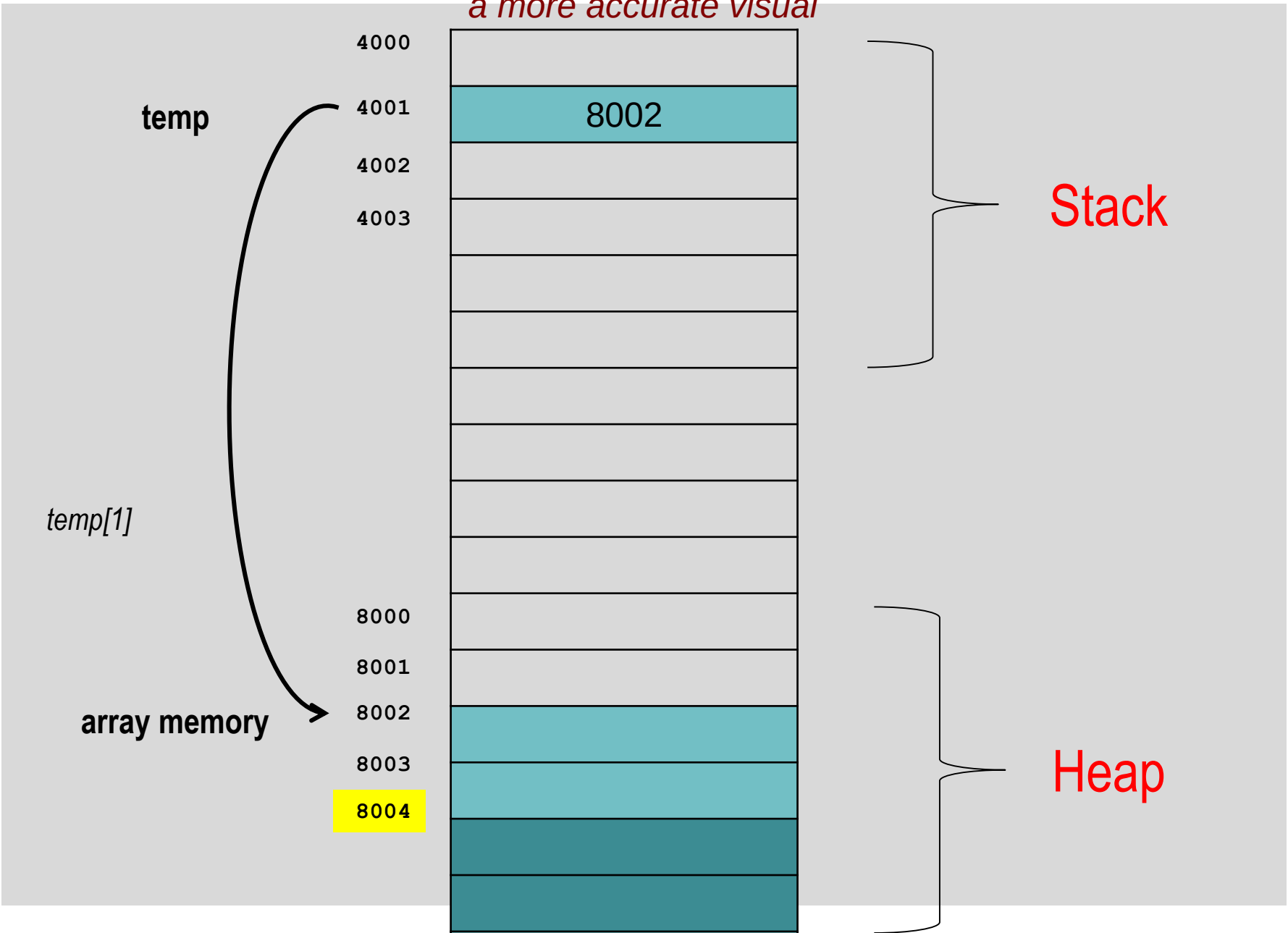
Arrays and References:

a more accurate visual



Arrays and References:

a more accurate visual



Arrays and References

- An array variable does *not* store the array itself.

- It stores a *reference* to the array.
  - the memory address of the array

```
int[] temps = {51, 50, 36, 29, 30};
```

- If we print an array variable, we print the contents or value of the variable, which is the memory address of the array!

```
System.out.println(temps);
```

output:

```
[I@1e1fd124
```

Arrays and References

- An array variable does *not* store the array itself.

- It stores a *reference* to the array.
  - the memory address of the array

```
int[] temps = {51, 50, 36, 29, 30};
```

- To print the contents of the array, we must first invoke a *static* method of the Array class that returns a string representation of the specified array:

```
System.println( Arrays.toString(temps) );
```

Processing a Sequence Using a Loop

Index-based:

```
for (int i = 0; i < array.length; i++) {
    do something with array[i]
}
```

where *array* is the array variable

Element-based:

```
for (int val: array) {
    do something with val
}
```

where *array* is the array variable

- Index-based is more flexible:
  - you can use it to *change* the element with index `i`
  - you can keep track of where you saw a given value

Processing a Sequence Using a Loop

Index-based:

```
for (int i = 0; i <  array
     do something with
}
```

ray variable

. The loop iterates over the elements of the array, so variable type depends on the array data type.

Element-based:

```
for (int val: array) {
     do something with val
}
```

where *array* is the array variable

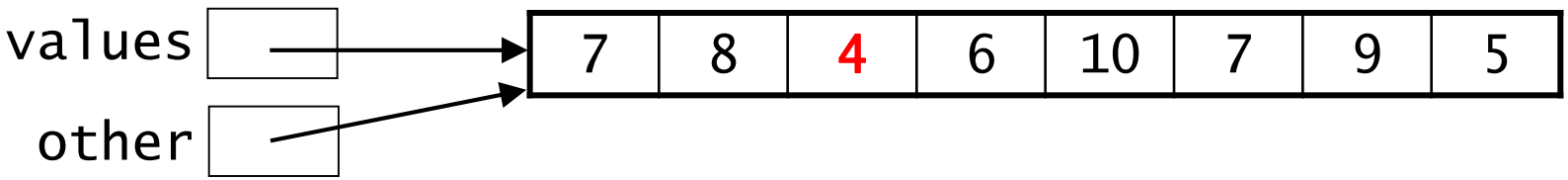
- Index-based is more flexible:
  - you can use it to *change* the element with index `i`
  - you can keep track of where you saw a given value

Array Assignment

Copying a Reference Variable

- What does this do?

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = values;
```



- Given the lines of code above, what will the lines below print?

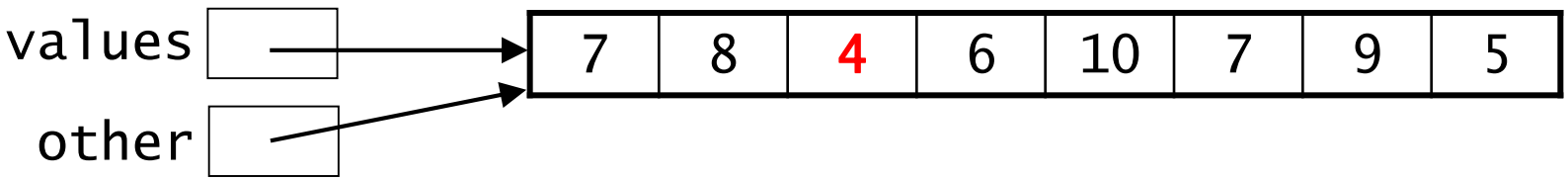
```
other[2] = 4;
System.out.println(values[2] + " " + other[2]);
                   4         + " " +    4
                              "4 4"
```

Array Assignment

Copying a Reference Variable

- What does this do?

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = values;
```



- Given the lines of code above, what will the lines below print?

```
other[2] = 4;
System.out.println(values[2] + " " + other[2]);
```

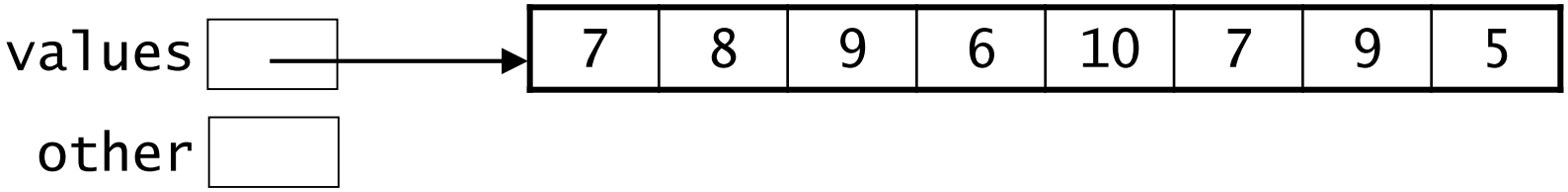
Shallow Copy

Copying an Array

- To actually create a **deep** copy of an array, we can:
  - create a new array of the same length as the first
  - traverse the arrays and copy the individual elements

- Example:

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = new int[values.length];
for (int i = 0; i < values.length; i++) {
    other[i] = values[i];
}
```

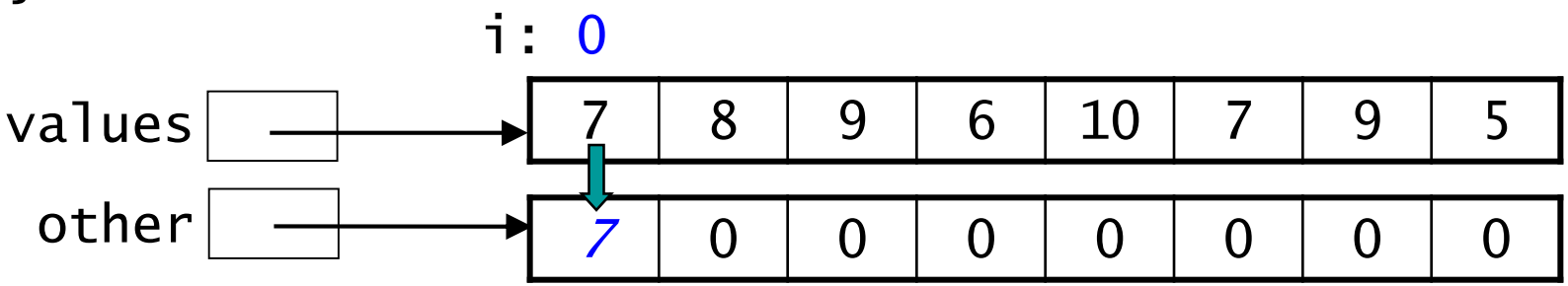


Copying an Array

- To actually create a **deep** copy of an array, we can:
  - create a new array of the same length as the first
  - traverse the arrays and copy the individual elements

- Example:

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = new int[values.length];
for (int i = 0; i < values.length; i++) {
    other[i] = values[i];
}
```

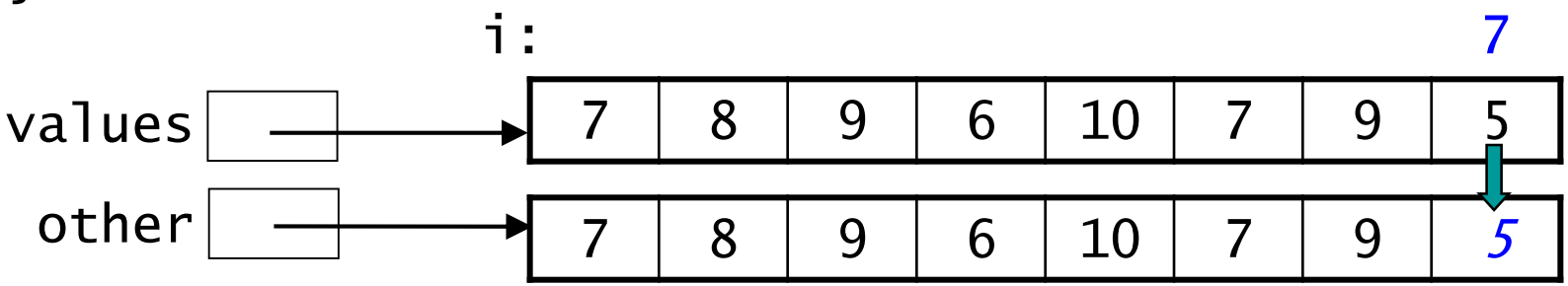


Copying an Array

- To actually create a **deep** copy of an array, we can:
  - create a new array of the same length as the first
  - traverse the arrays and copy the individual elements

- Example:

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = new int[values.length];
for (int i = 0; i < values.length; i++) {
    other[i] = values[i];
}
```

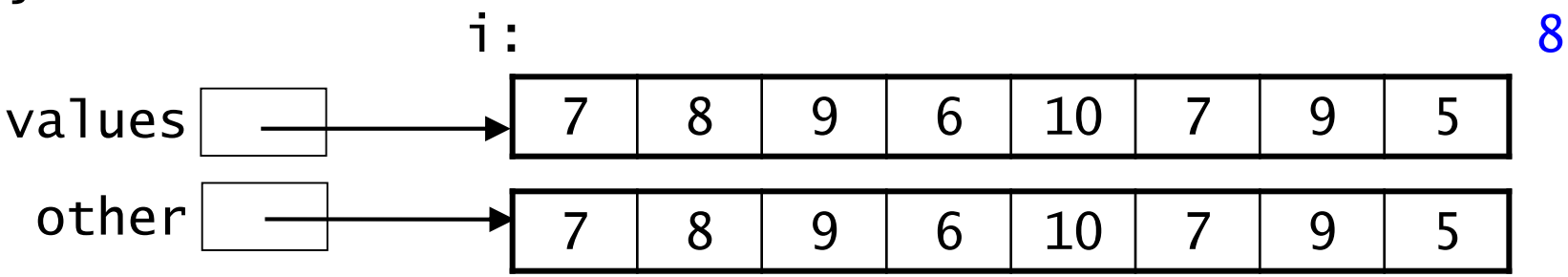


Copying an Array

- To actually create a **deep** copy of an array, we can:
  - create a new array of the same length as the first
  - traverse the arrays and copy the individual elements

- Example:

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = new int[values.length];
for (int i = 0; i < values.length; i++) {
    other[i] = values[i];
}
```

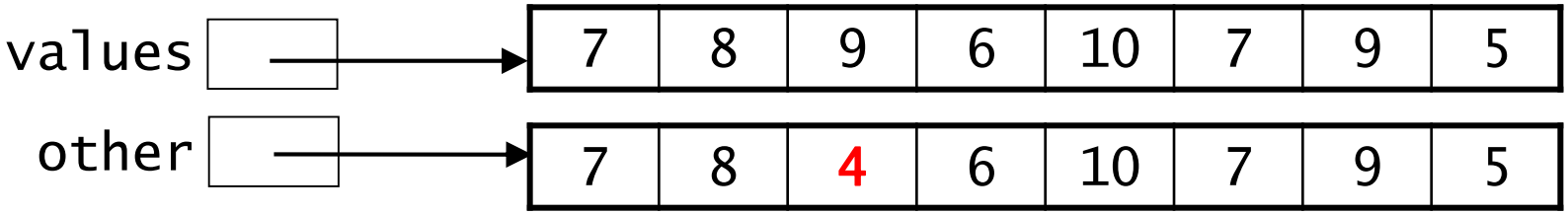


Copying an Array

- To actually create a **deep** copy of an array, we can:
  - create a new array of the same length as the first
  - traverse the arrays and copy the individual elements

- Example:

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = new int[values.length];
for (int i = 0; i < values.length; i++) {
    other[i] = values[i];
}
```



- What do the following lines print now?

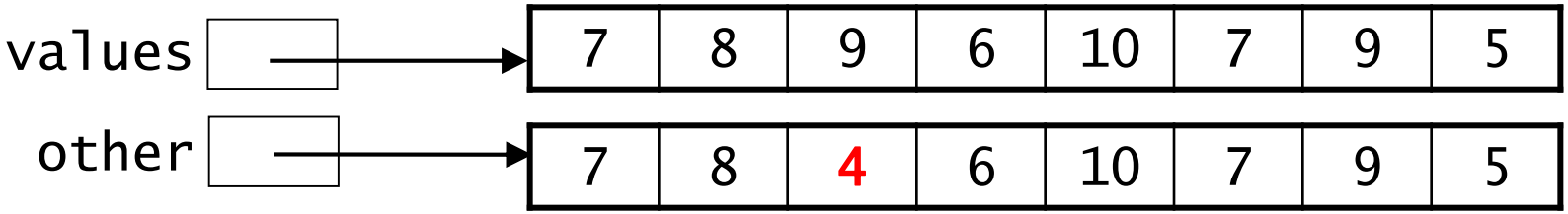
```
other[2] = 4;
System.out.println(values[2] + " " + other[2]);
```

Copying an Array

- To actually create a **deep** copy of an array, we can:
  - create a new array of the same length as the first
  - traverse the arrays and copy the individual elements

- Example:

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = new int[values.length];
for (int i = 0; i < values.length; i++) {
    other[i] = values[i];
}
```



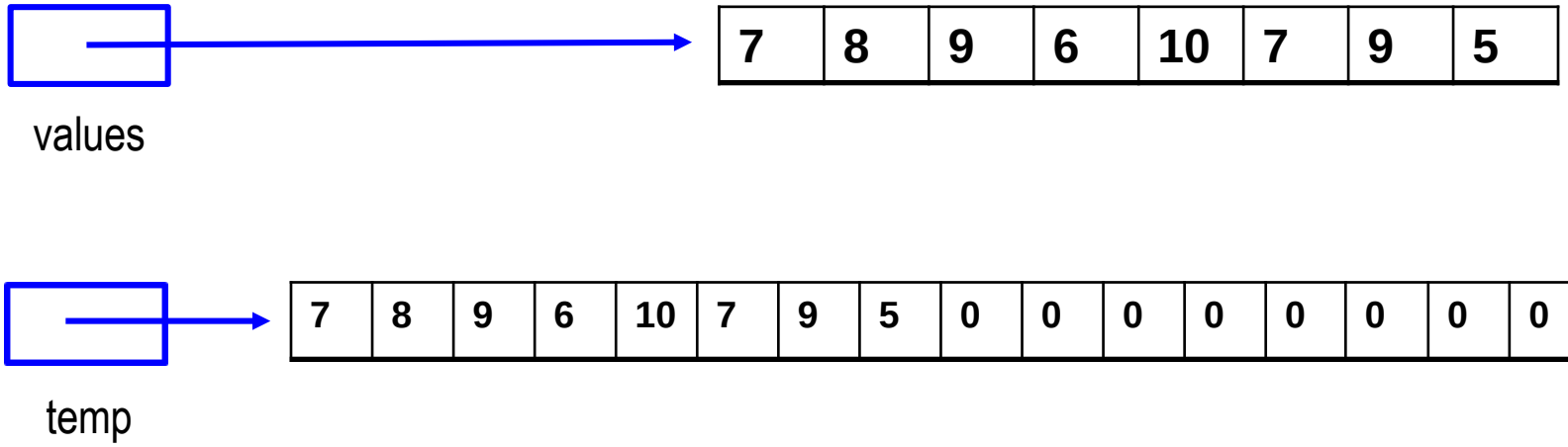
- What do the following lines print now?

```
other[2] = 4;
System.out.println(9 + " " + 4);
```

Example: Memory layout

Growing an Array

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
...
int[] temp = new int[16];
for (int i = 0; i < values.length; i++) {
     temp[i] = values[i];
}
```

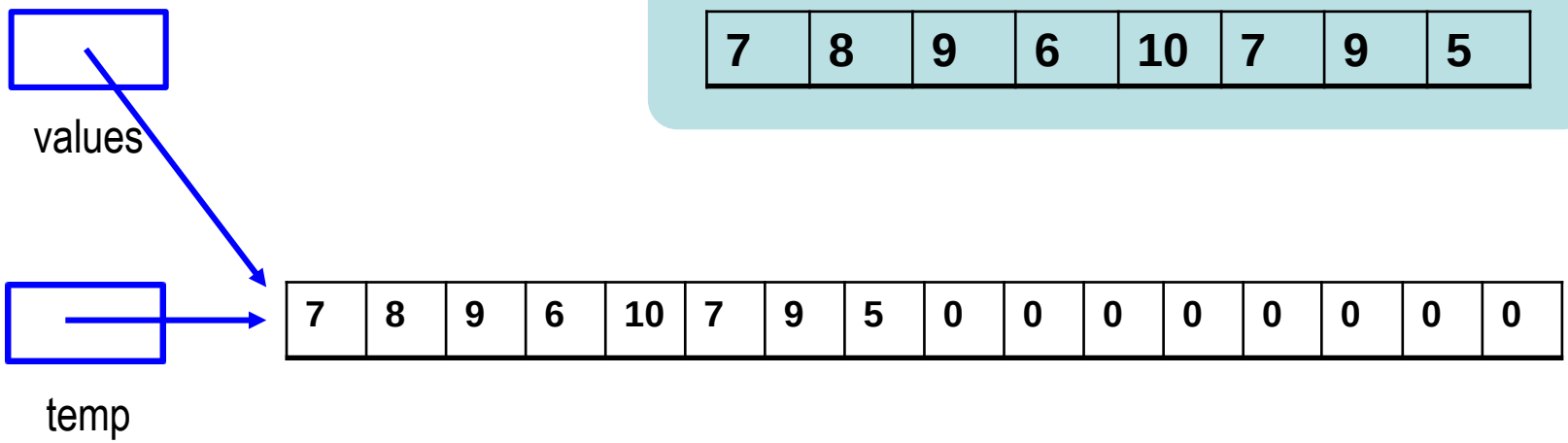


Example: Memory Layout

Growing an Array

```
int[] values = {7, 8, 9, 6, 10
...
int[] temp = new int[16];
for (int i = 0; i < values.length
     temp[i] = values[i];
}
```

This memory remains "in limbo" waiting for the garbage collector to free it!




```
// re-assign the temporary array
values = temp
```

A 2-D Array

- A 2-D array, *an array of arrays*
- To declare a 2-D array we must specify both the `row` and the `column` dimension as:

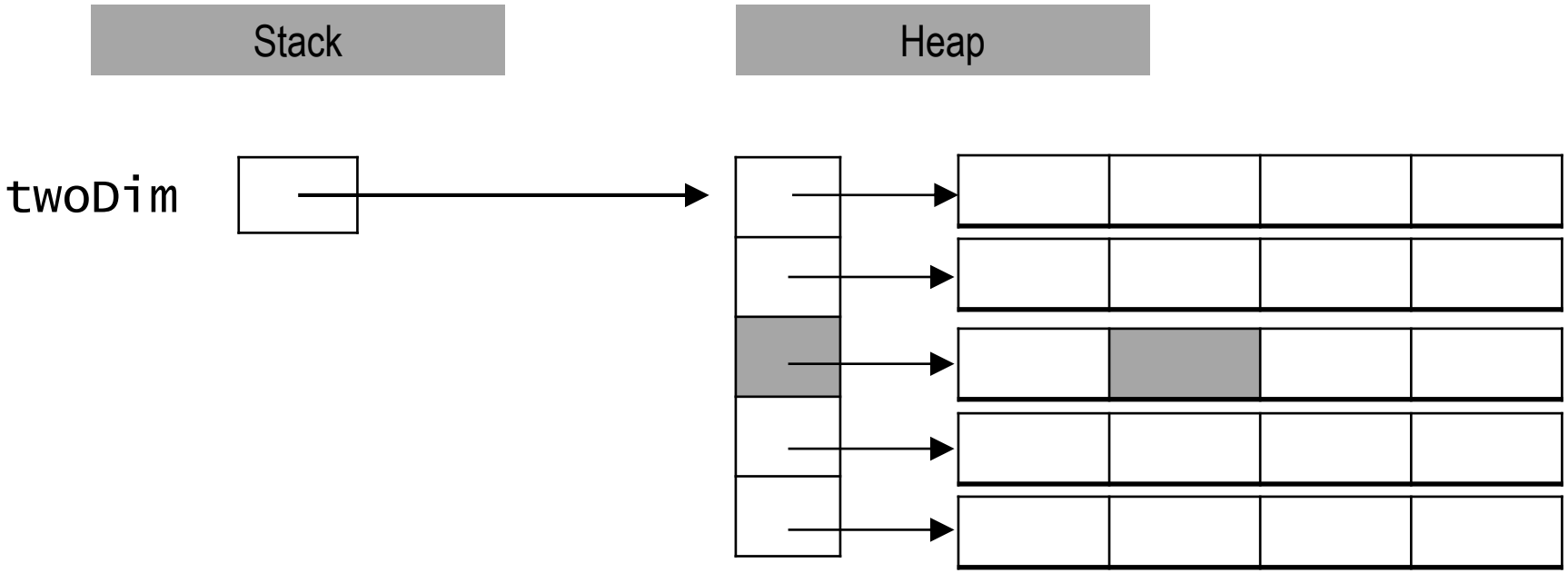
```
int[][] twoDim = new int [5][4];
```

- This statement is equivalent to:

```
int[][]twoDim = new int[5][];
twoDim[0] = new int[4];
twoDIm[1] = new int[4];
twoDim[2] = new int[4];
```

```
int[][]twoDim;
twoDim = new int[5][4];
```

A 2-D Array




```
twoDim[2][1] = 12;
```

Maintaining a Game Board

- For a ***simple*** Tic-Tac-Toe board, we *could* use a 2-D array to keep track of the state of the board:

```
char[][] board = new char[3][3];
```

- Alternatively, we could create *and* initialize it as follows:

```
char[][] board = {{' ', ' ', ' '},
                  {' ', ' ', ' '},
                  {' ', ' ', ' '}};
```

- If a player puts an X in the middle square, we could record this fact by making the following assignment:

```
board[1][1] = 'X';
```

Processing All of the Elements in a 2-D Array

- To perform some operation on all of the elements in a 2-D array, we typically use a nested loop.
  - example: finding the maximum value in a 2-D array.

```
public static int maxValue(int[][] arr) {
    int max = arr[0][0];
    for (int r = 0; r < arr.length; r++) {
        for (int c = 0; c < arr[r].length; c++) {
            if (arr[r][c] > max) {
                max = arr[r][c];
            }
        }
    }

    return max;
}
```

Sources

Thanks to contributing slides from:

- David Sullivan, PhD