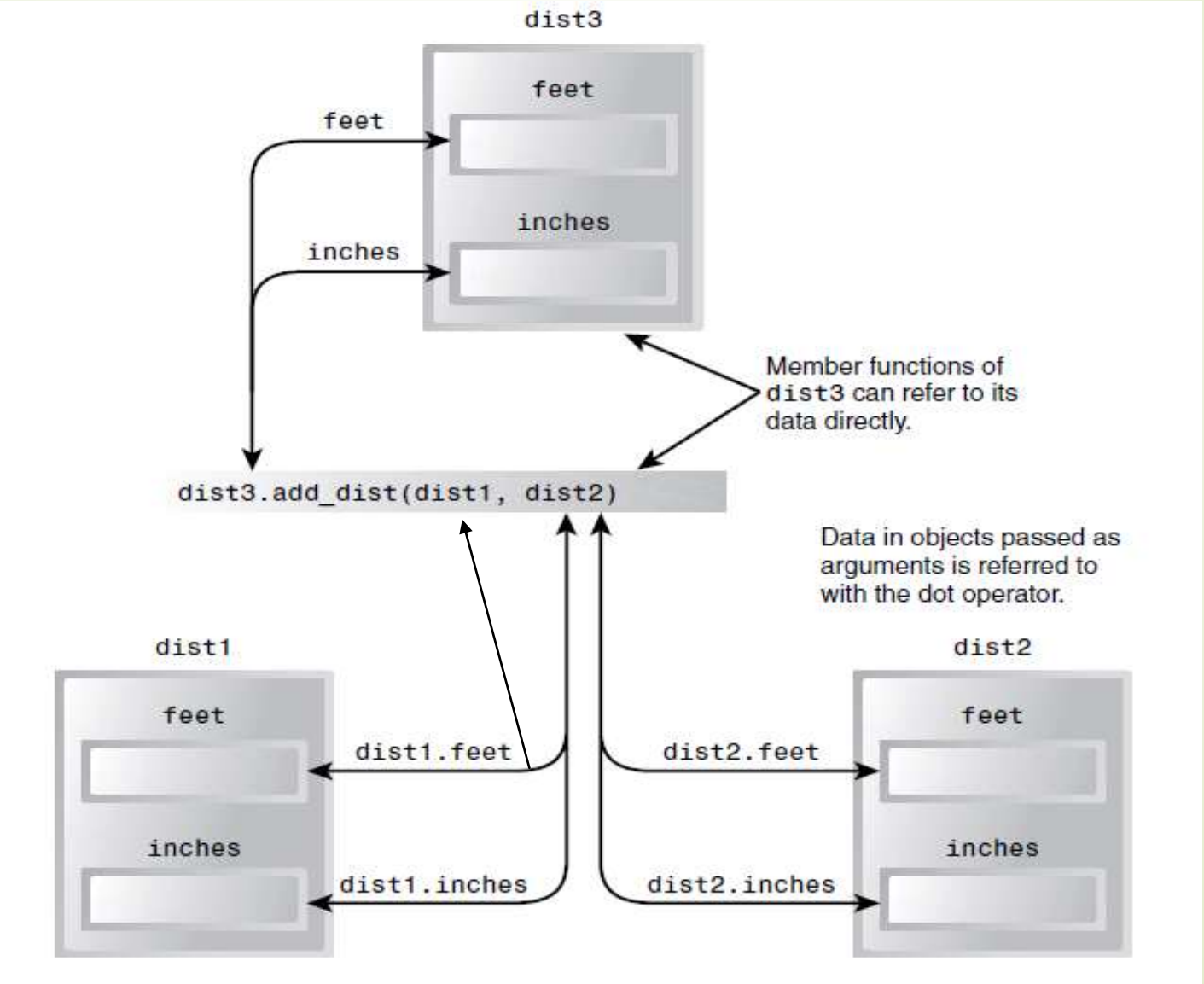
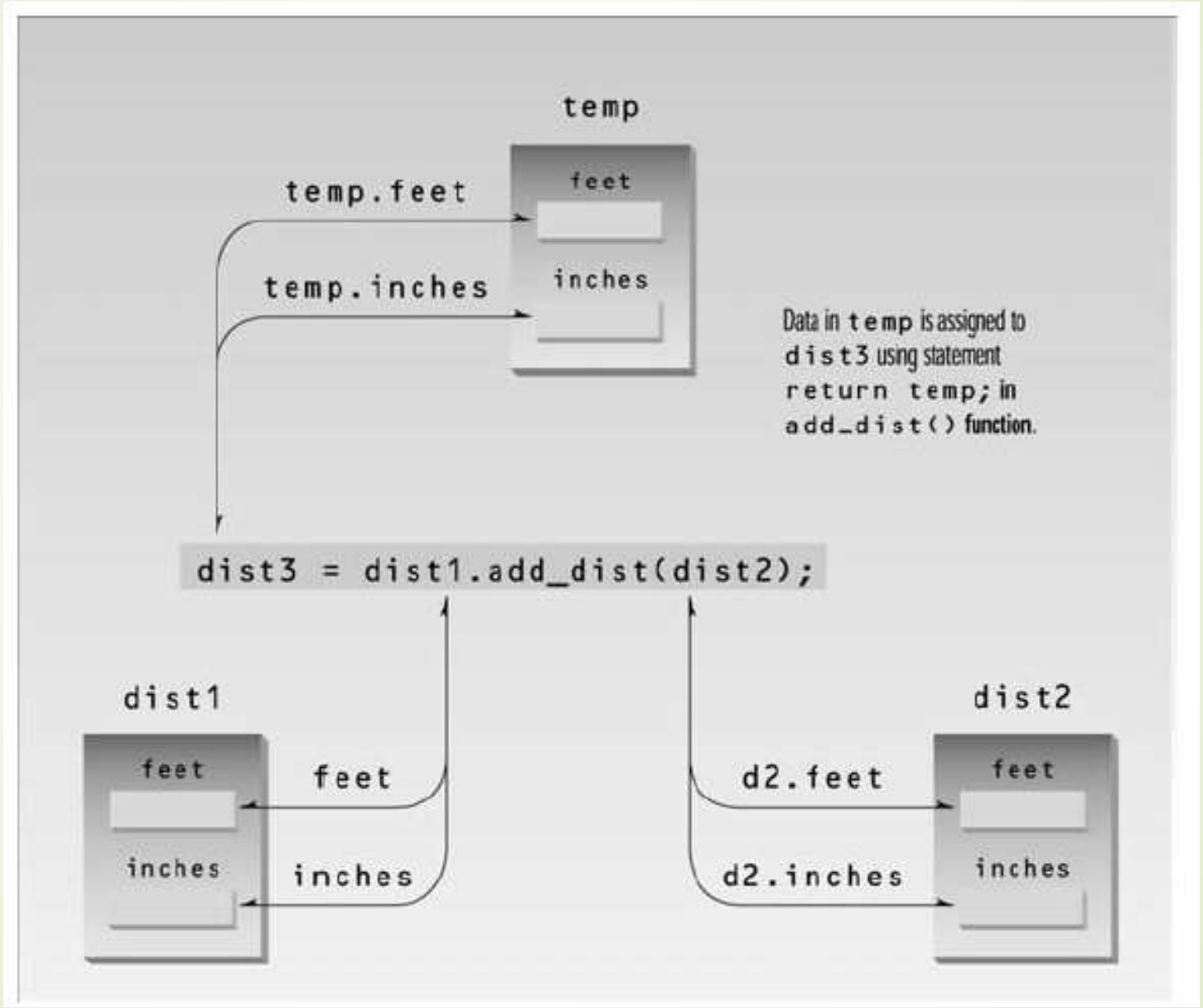
Objects as function arguments

- Pass objects as parameters in a function

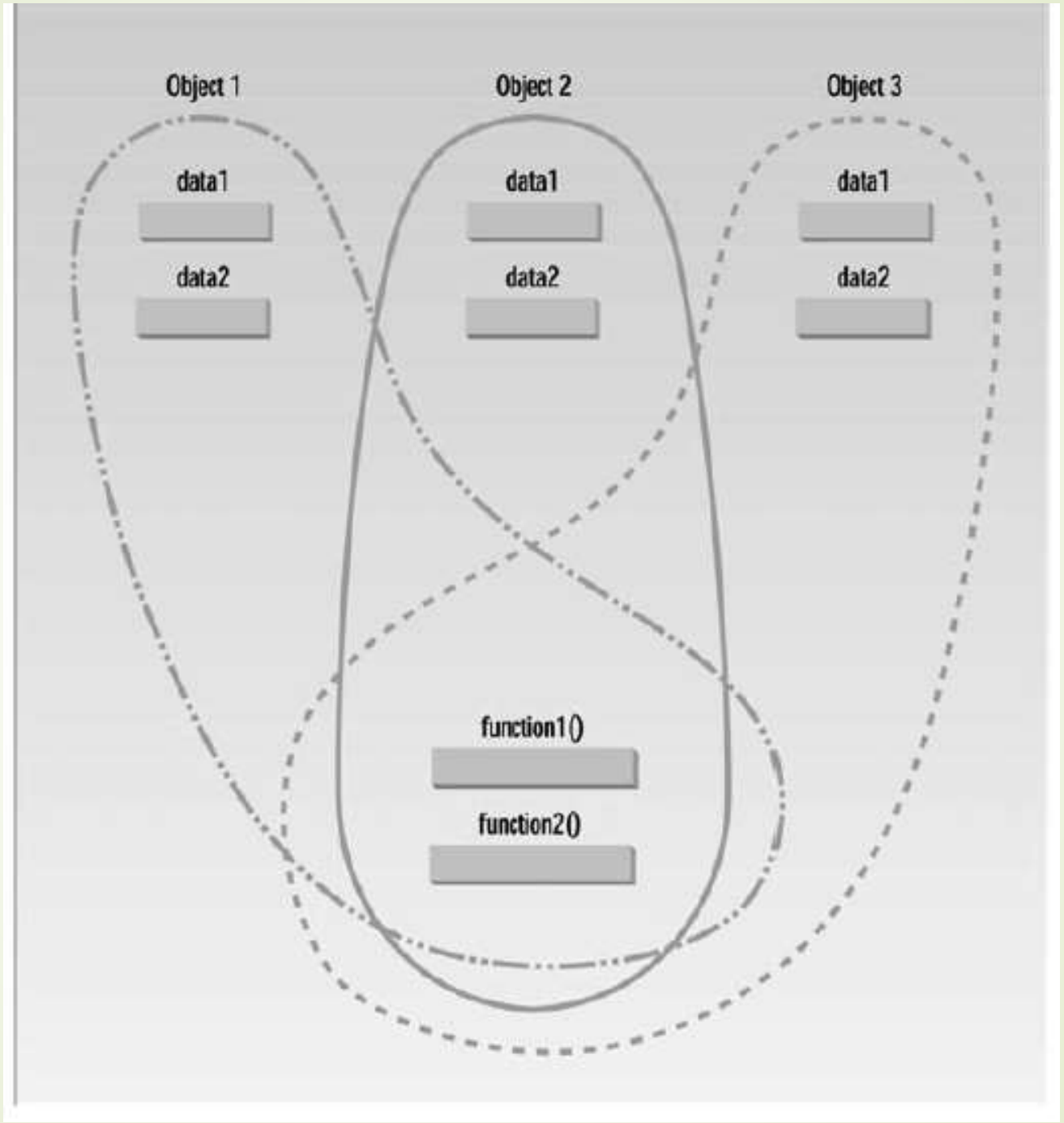


Returning objects from the function



Classes, objects and memory

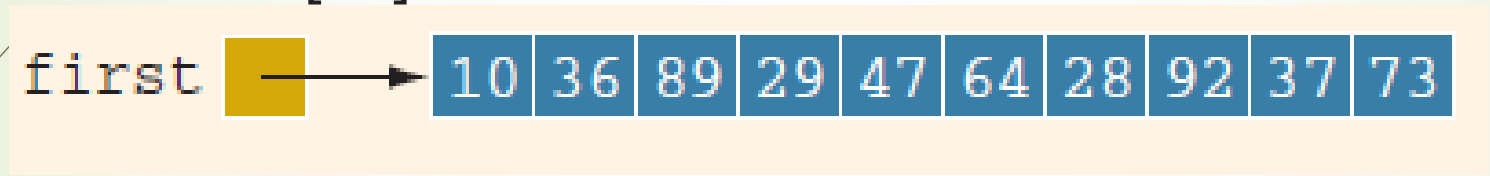
- We've probably given you the impression that each object created from a class contains separate copies of that class's data and member functions. This is a good first approximation, since it Objects and Classes emphasizes that objects are complete, self-contained entities, designed using the class definition.
- It's true that each object has its own separate data items. On the other hand, contrary to what you may have been led to believe, all the objects in a given class use the same member functions.
- The member functions are created and placed in memory only once.
- In most situations you don't need to know that there is only one member function for an entire class. It's simpler to visualize each object as containing both its own data and its own member functions.


Object 1
Object 2
Object 3
data1
data2
data1
data2
data1
data2
function1()
function2()

Shallow versus Deep Copy and Pointers

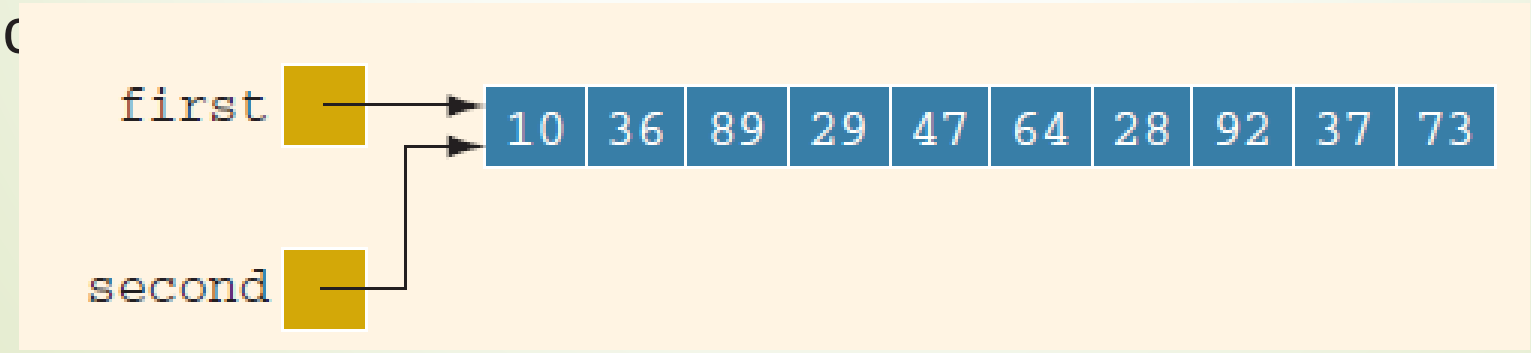
Consider the following statements:

```
int *first;
int *second;
first = new int[10];
```



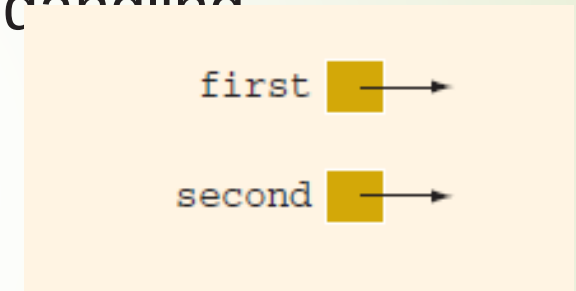
Next, consider the following statement:

```
sec
```



Shallow versus Deep Copy and Pointers

delete [] second;
After this statement executes, the array pointed to by second is deleted. first (as well as second) are now dangling pointers.



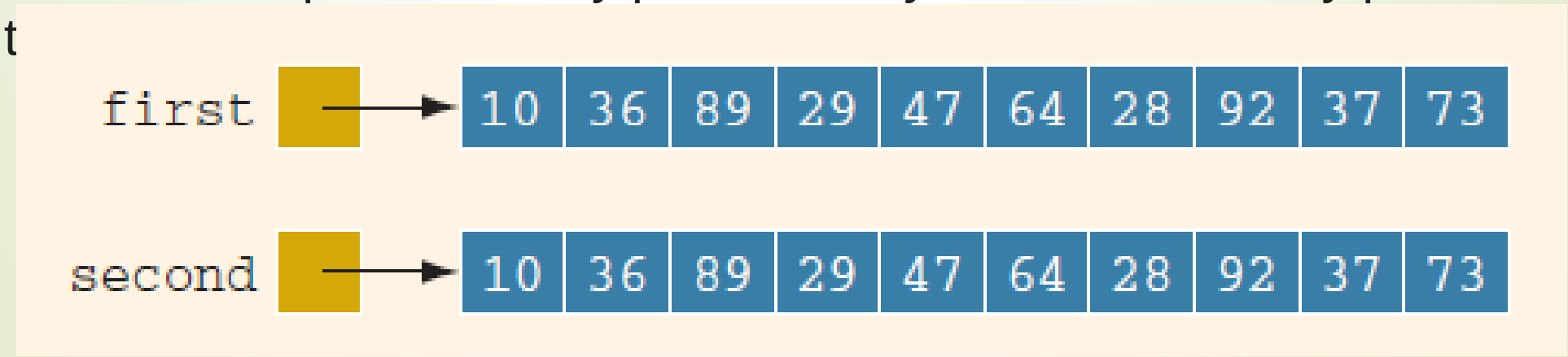
This case is an example of a shallow copy. More formally, in a shallow copy, two or more pointers of the same type point to the same memory; that is, they point to the same data.

Shallow versus Deep Copy and Pointers

we have the following statements:

```
second = new int[10];
for (int j = 0; j < 10; j++)
    second[j] = first[j];
```

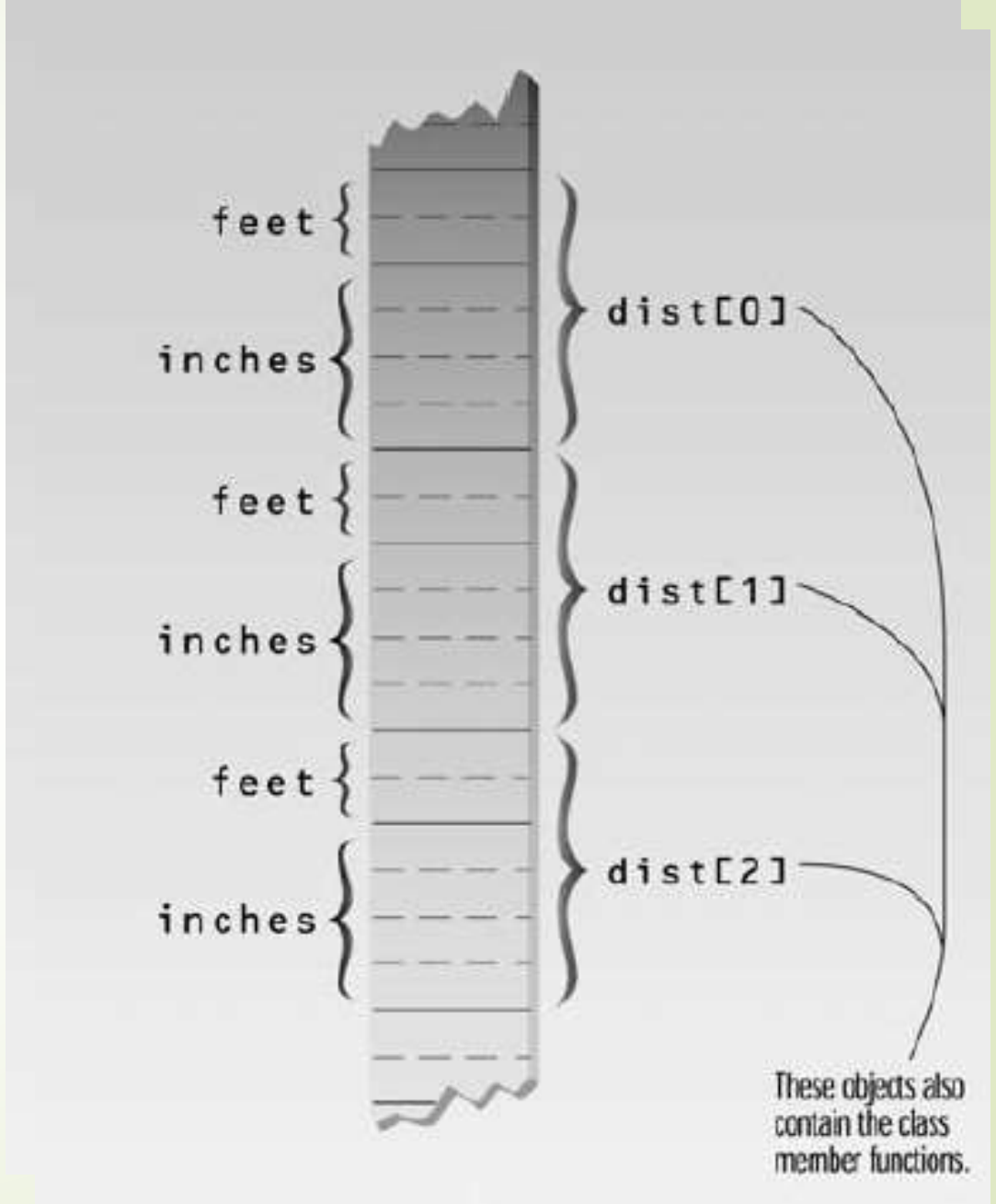
The first statement creates an array of 10 components of type int, and the base address of the array is stored in second. The second statement copies the array pointed to by first into the array pointed t



Classes and arrays

Arrays of Objects

We've seen how an object can contain an array. We can also reverse that situation and create an array of objects.



Classes and dynamic objects

```
#include <iostream>
using namespace std;

class Box {
Public:
   int y;
    public:
       Box() {
          cout << "Constructor called!" <<endl;
       }

};
int main() {
   Box* myBoxArray = new Box;
```