BTH001
Object Oriented Programming
Lesson 07
Deep and shallow copying

Shallow copying

- When a pointer variable is assigned the value of another pointer variable:
  - both variables will contain the same address
  - they point to the same ”thing”, for example
    - an object
    - an array

Assume the class Circle containing

Constructor: Circle(int radius = 0);

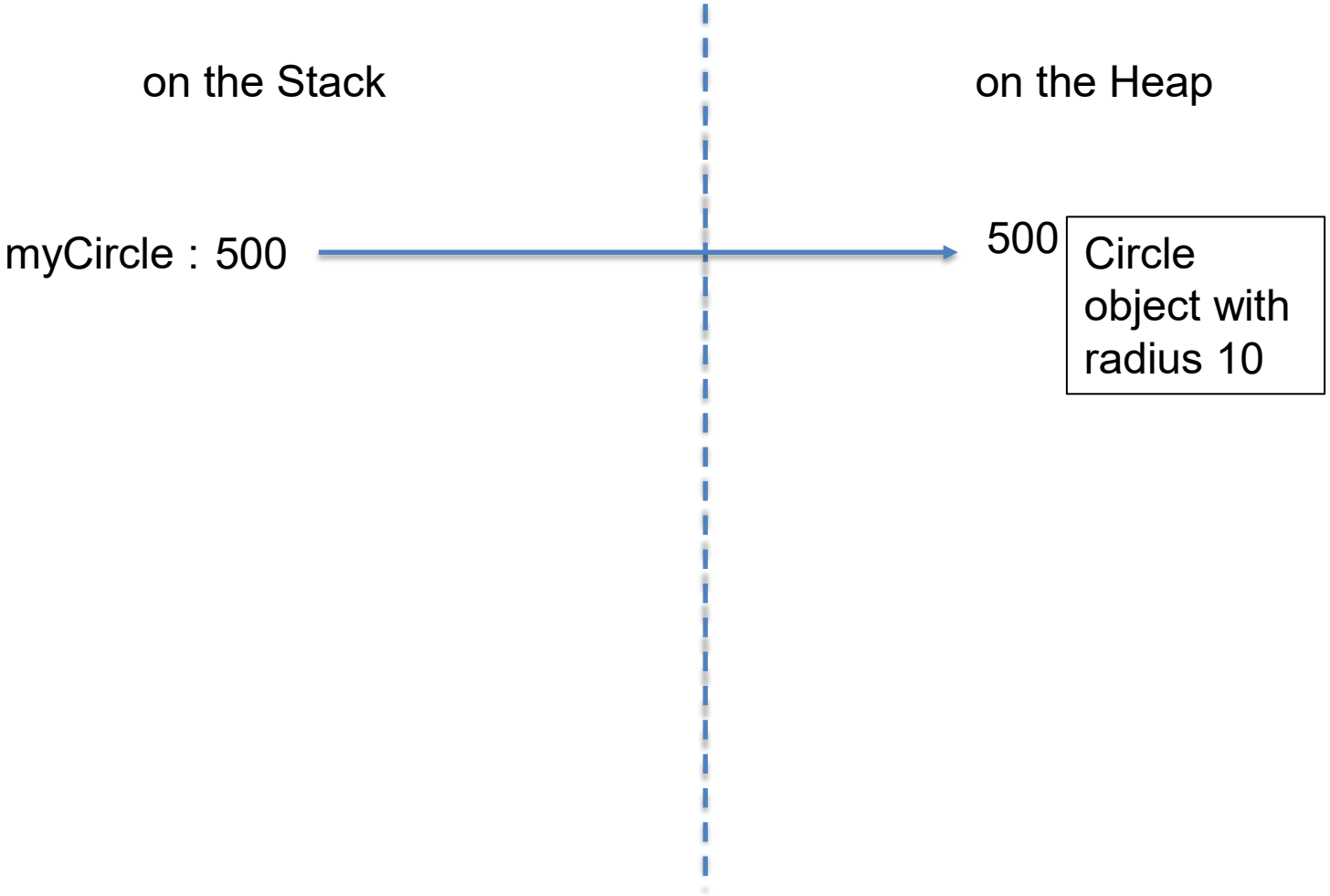
Member functions:

void setRadius(int radius);

int getRadius() const;

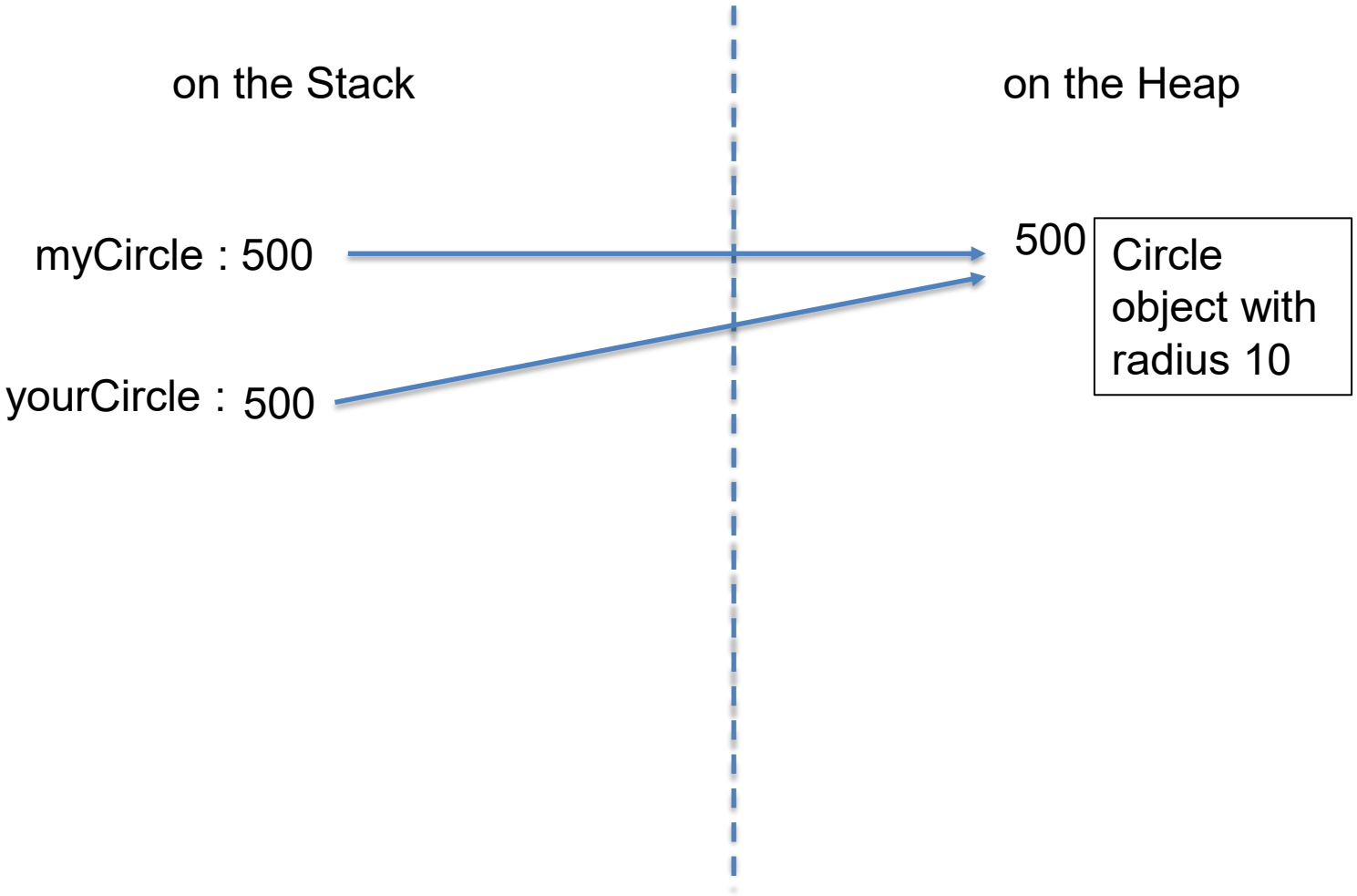
Example 1: Shallow copying

```
Circle *myCircle = new Circle(10);
```



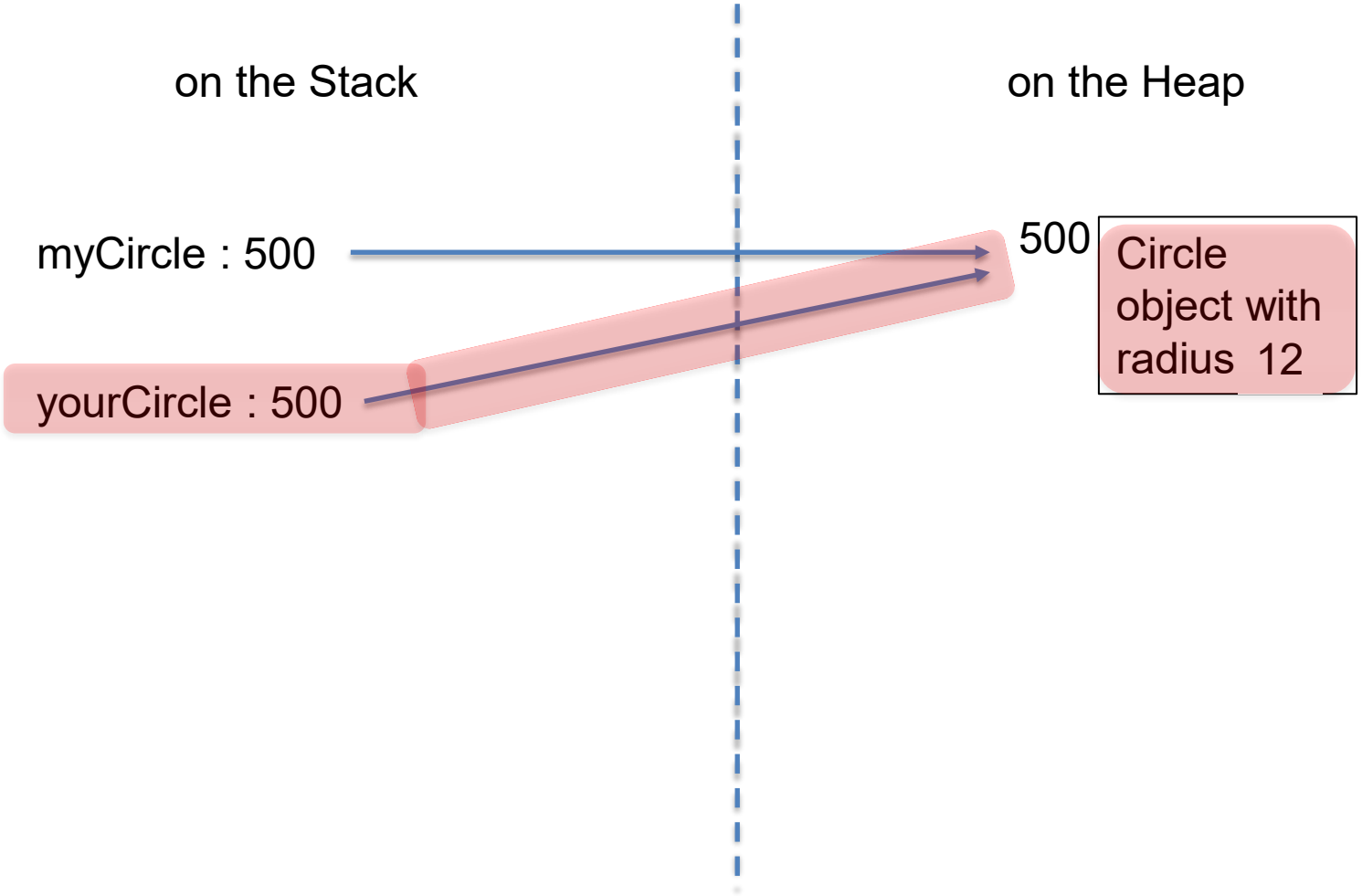
Example 1: Shallow copying (cont)

```
Circle *yourCircle = myCircle;
```



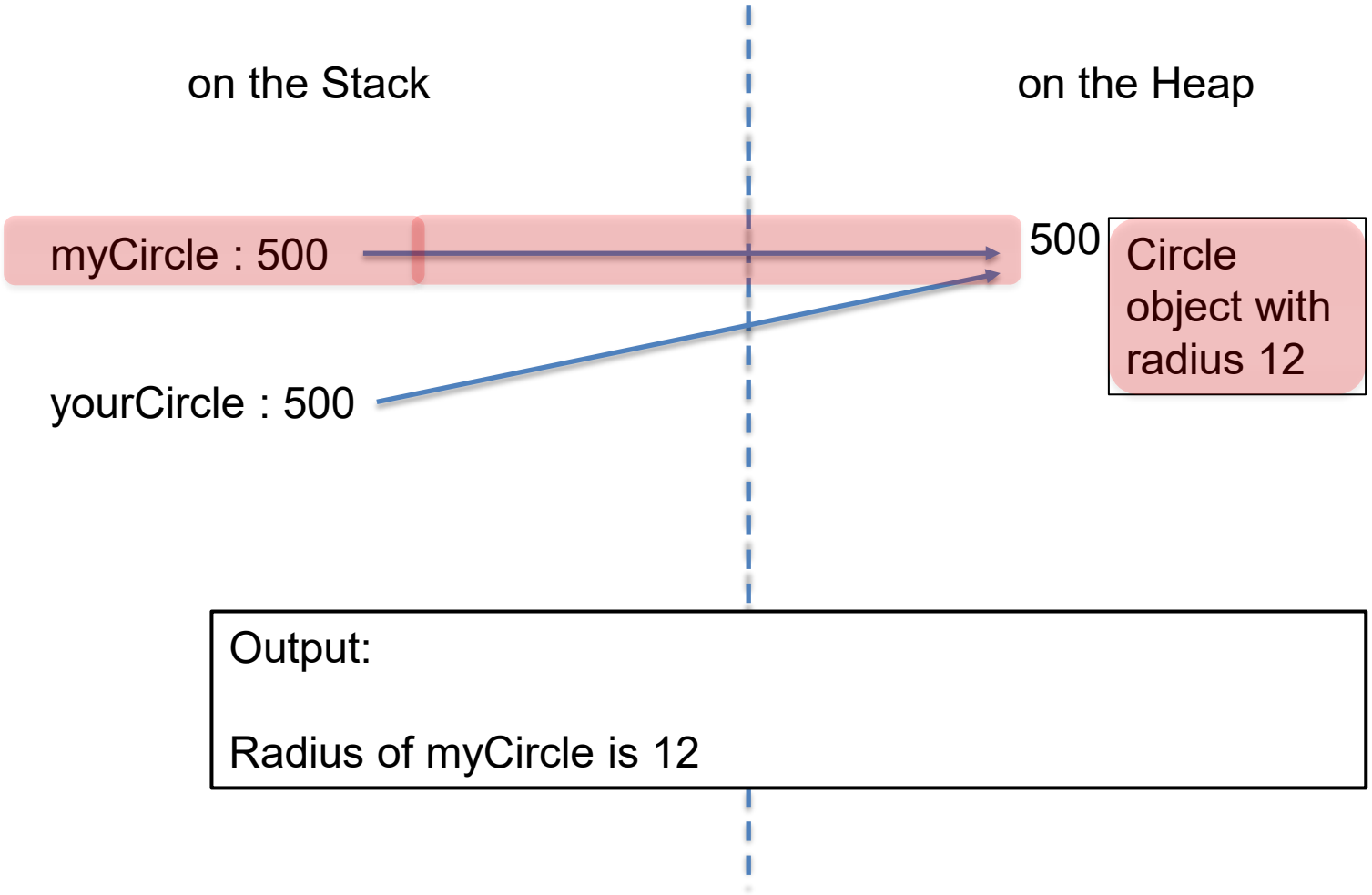
Example 1: Shallow copying (cont)

```
yourCircle->setRadius(12);
```



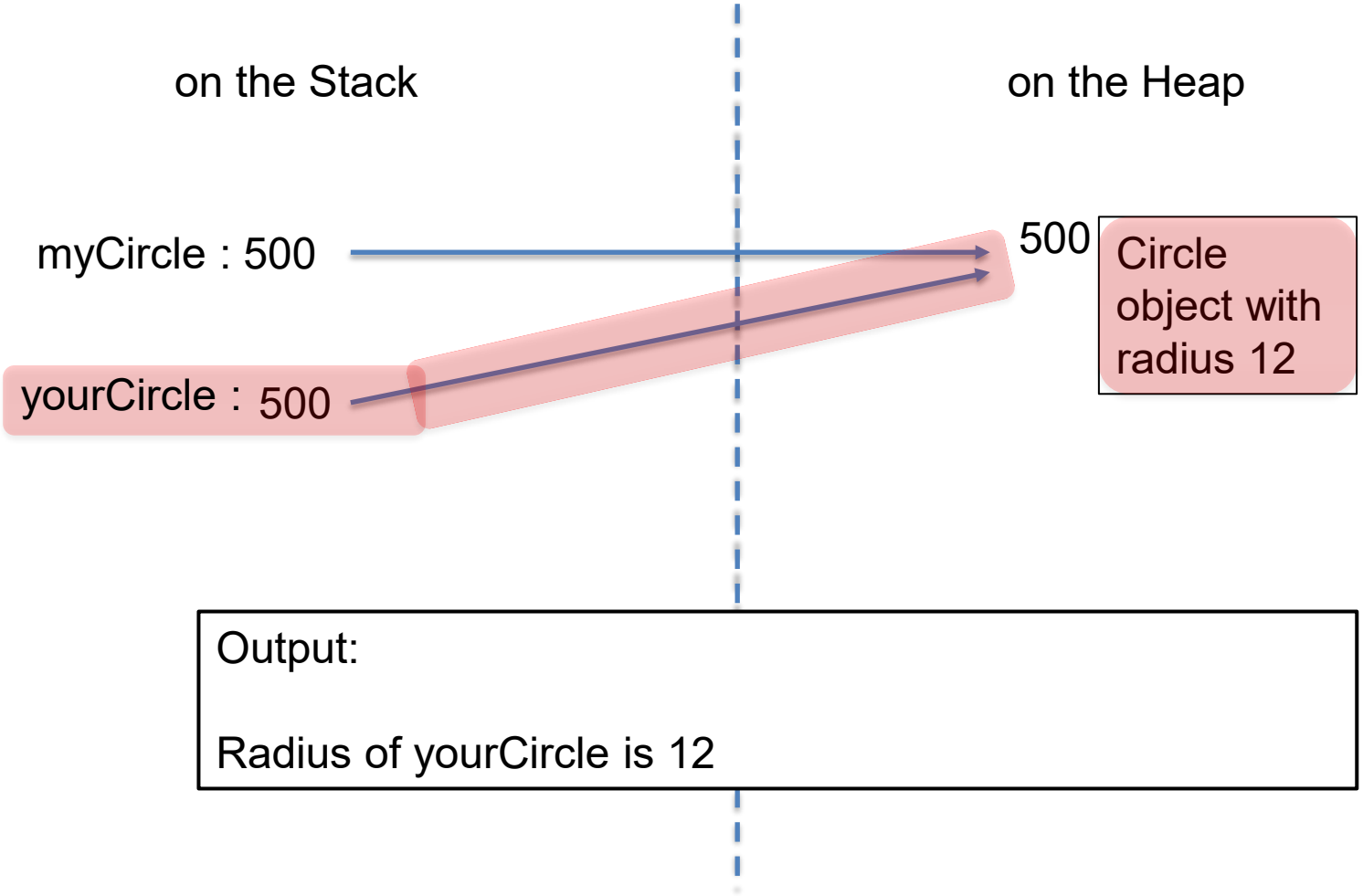
Example 1: Shallow copying (cont)

```
cout<<"Radius of myCircle is "
        <<myCircle->getRadius()<<endl;
```



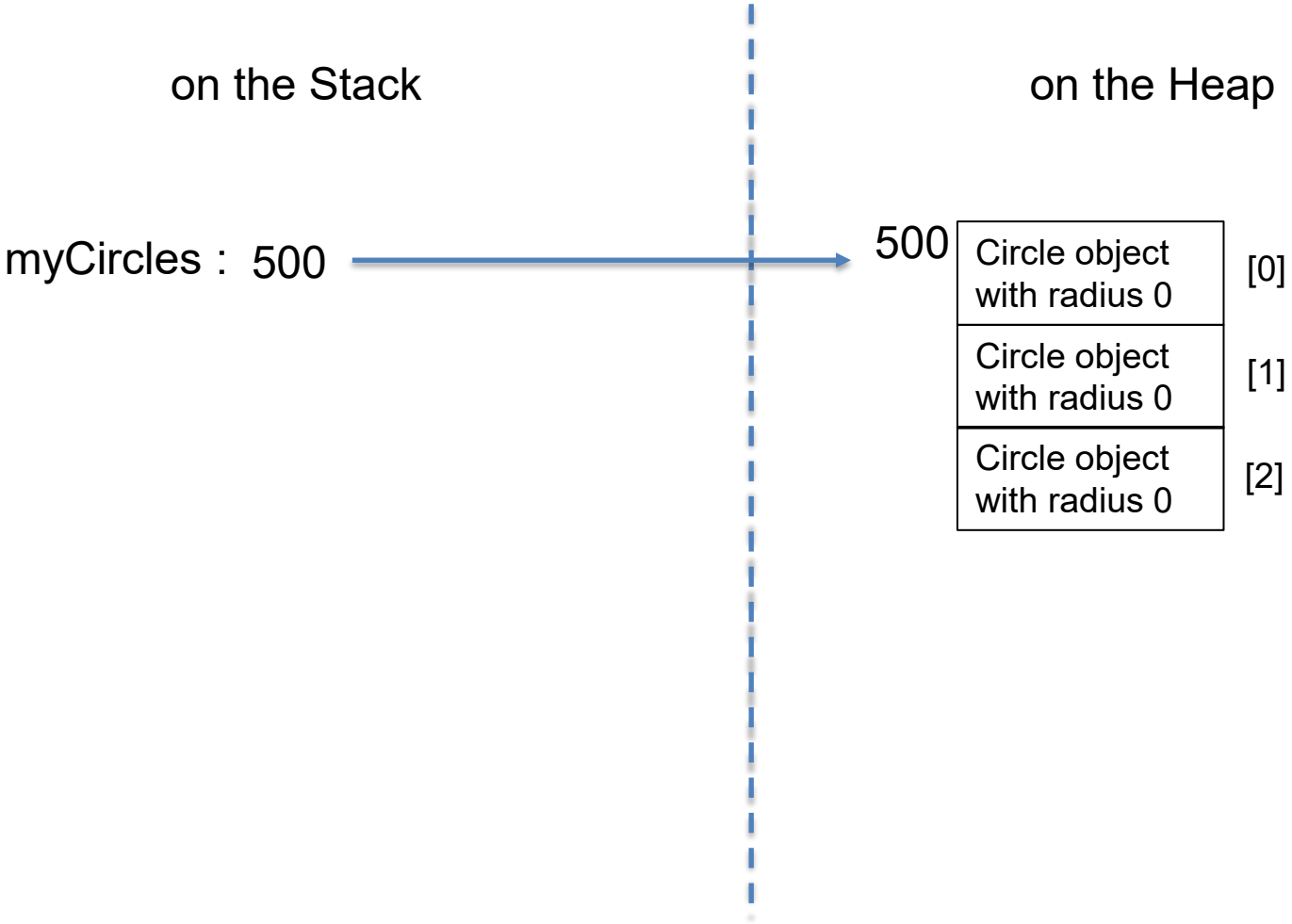
Example 1: Shallow copying (cont)

```
cout<<"Radius of yourCircle is "
    <<yourCircle->getRadius()<<endl;
```



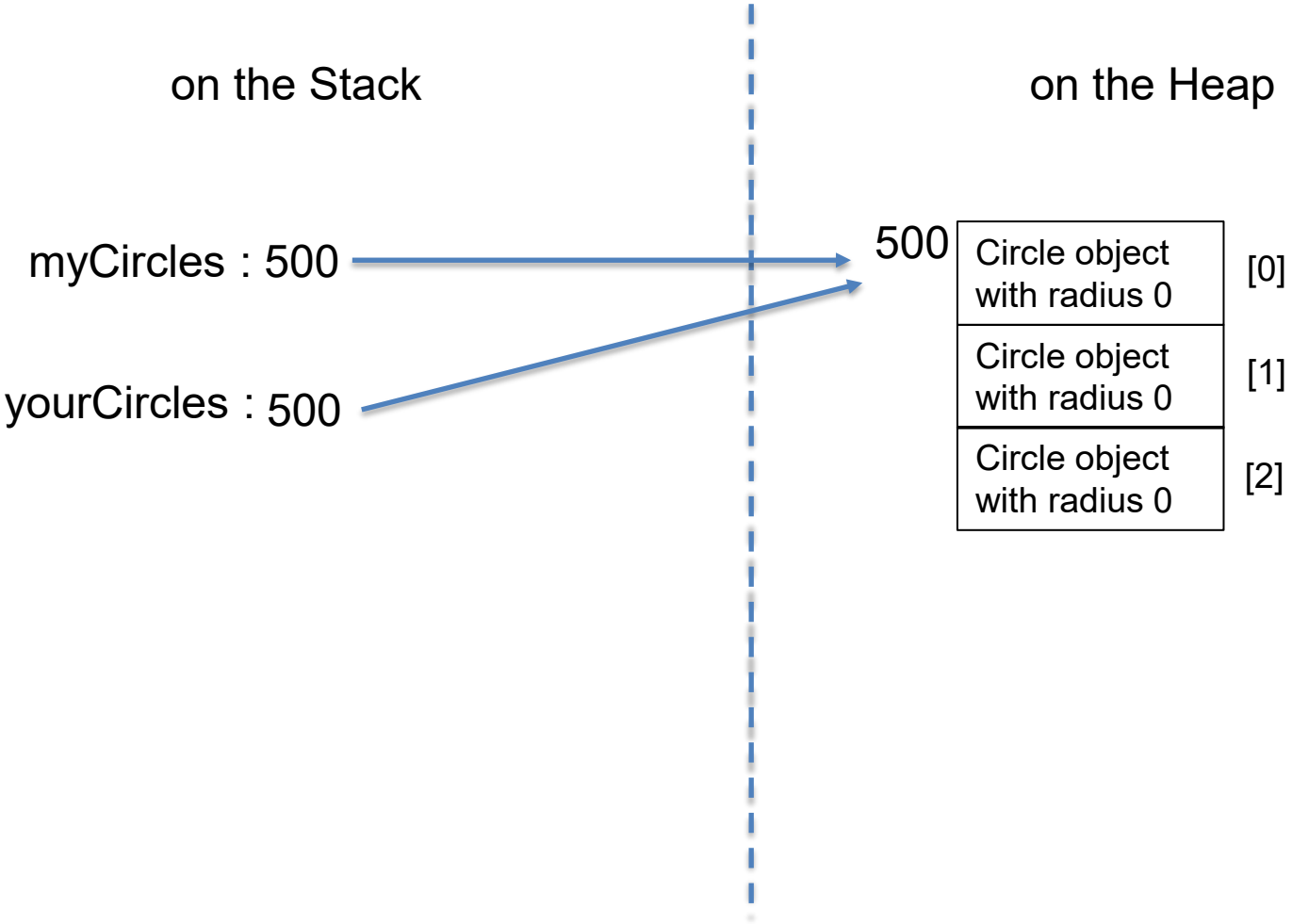
Example 2: Shallow copying

```
Circle *myCircles = new Circle[3];
```



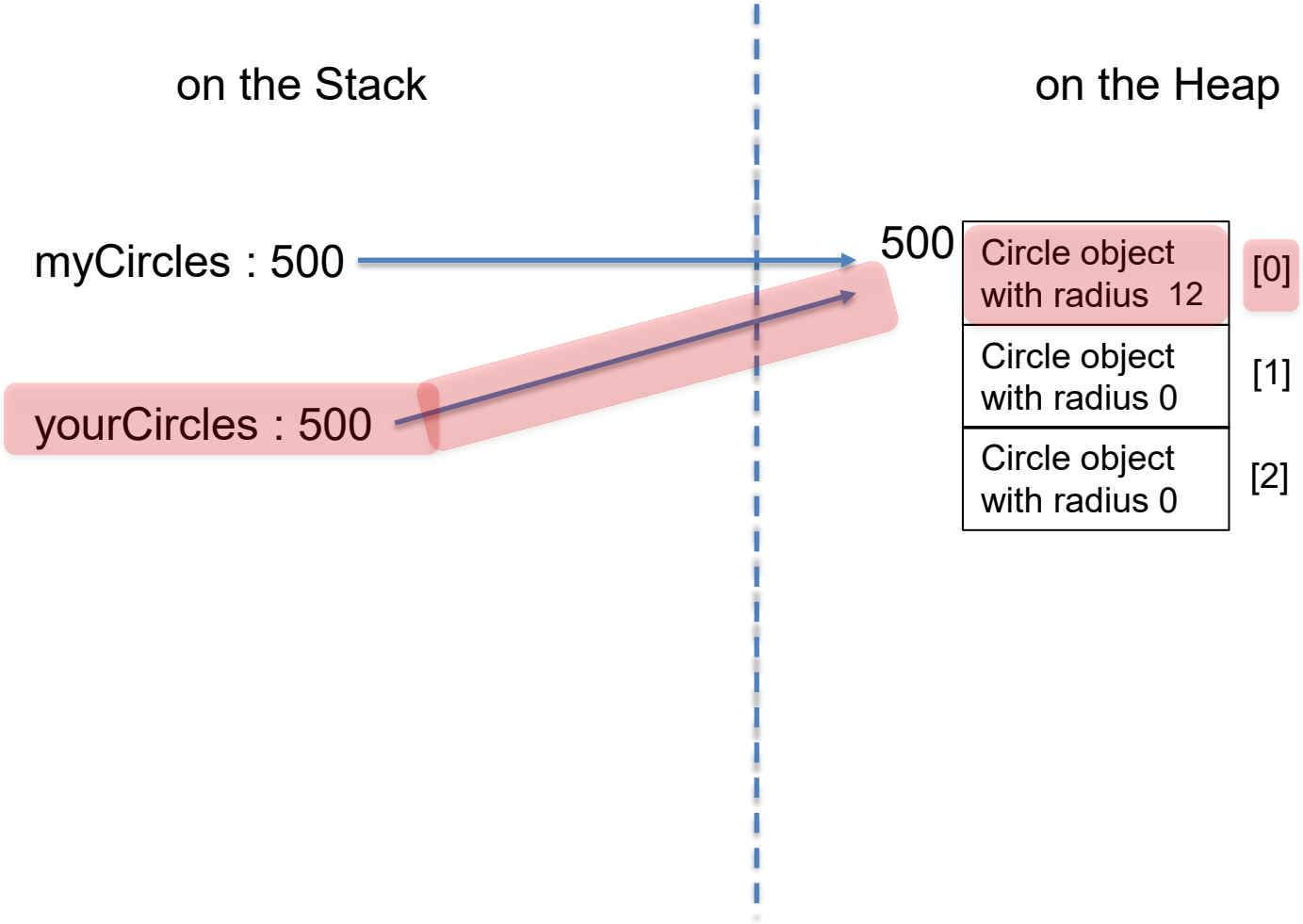
Example 2: Shallow copying (cont)

```
Circle *yourCircles = myCircles;
```



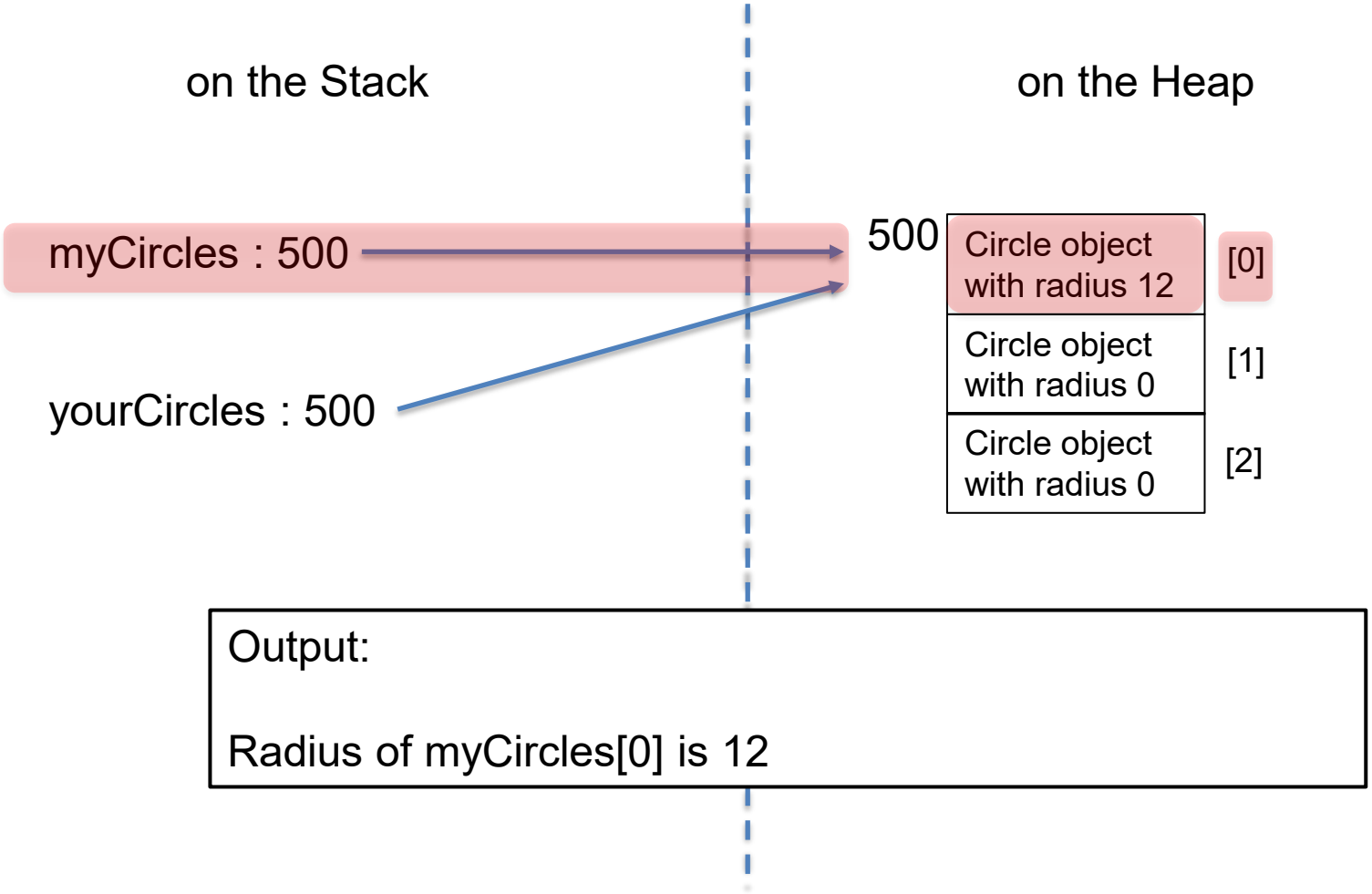
Example 2: Shallow copying (cont)

`yourCircles[0].setRadius(12);`



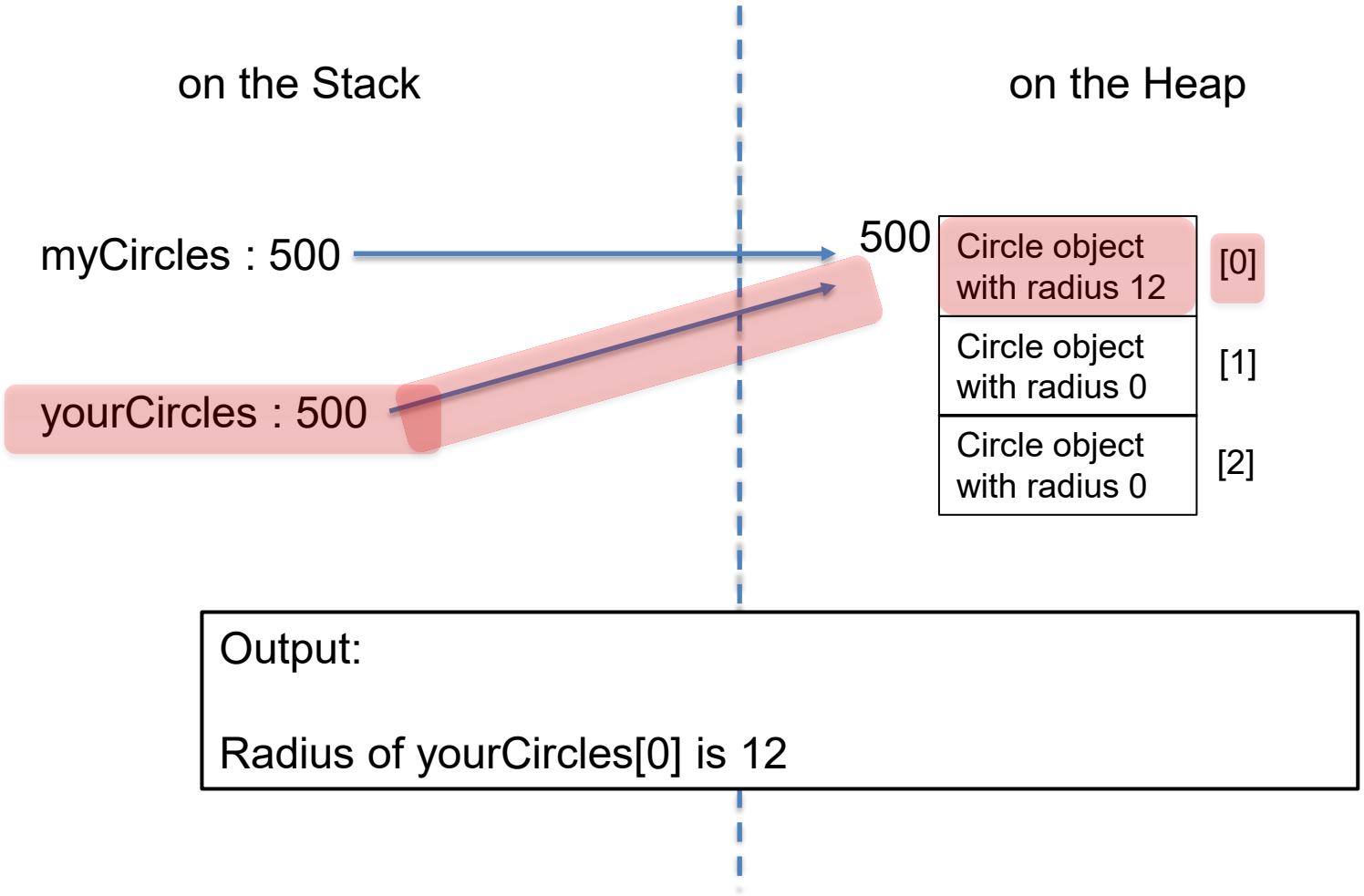
Example 2: Shallow copying (cont)

```
cout<<"Radius of myCircles[0] is "
        <<myCircles[0].getRadius()<<endl;
```



Example 2: Shallow copying (cont)

```
cout<<"Radius of yourCircles[0] is "
    <<yourCircles[0].getRadius()<<endl;
```

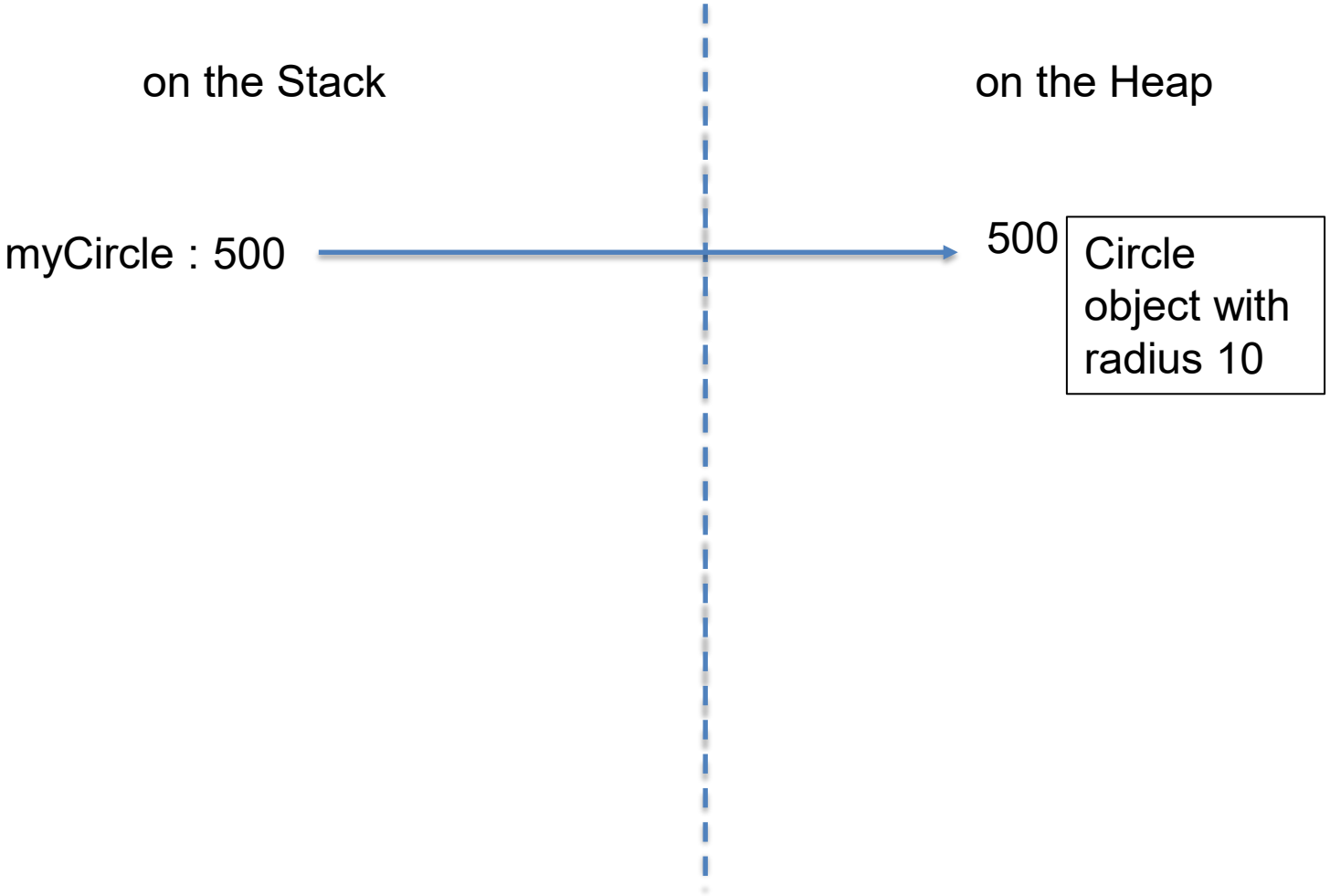


Deep copying

- When we want a pointer variable to point to an identical ”thing” as another pointer variable:
  - the variables must have different addresses
  - they point to different ”things” but the content of those things will be identical, for example
    - two identical objects
    - two identical arrays

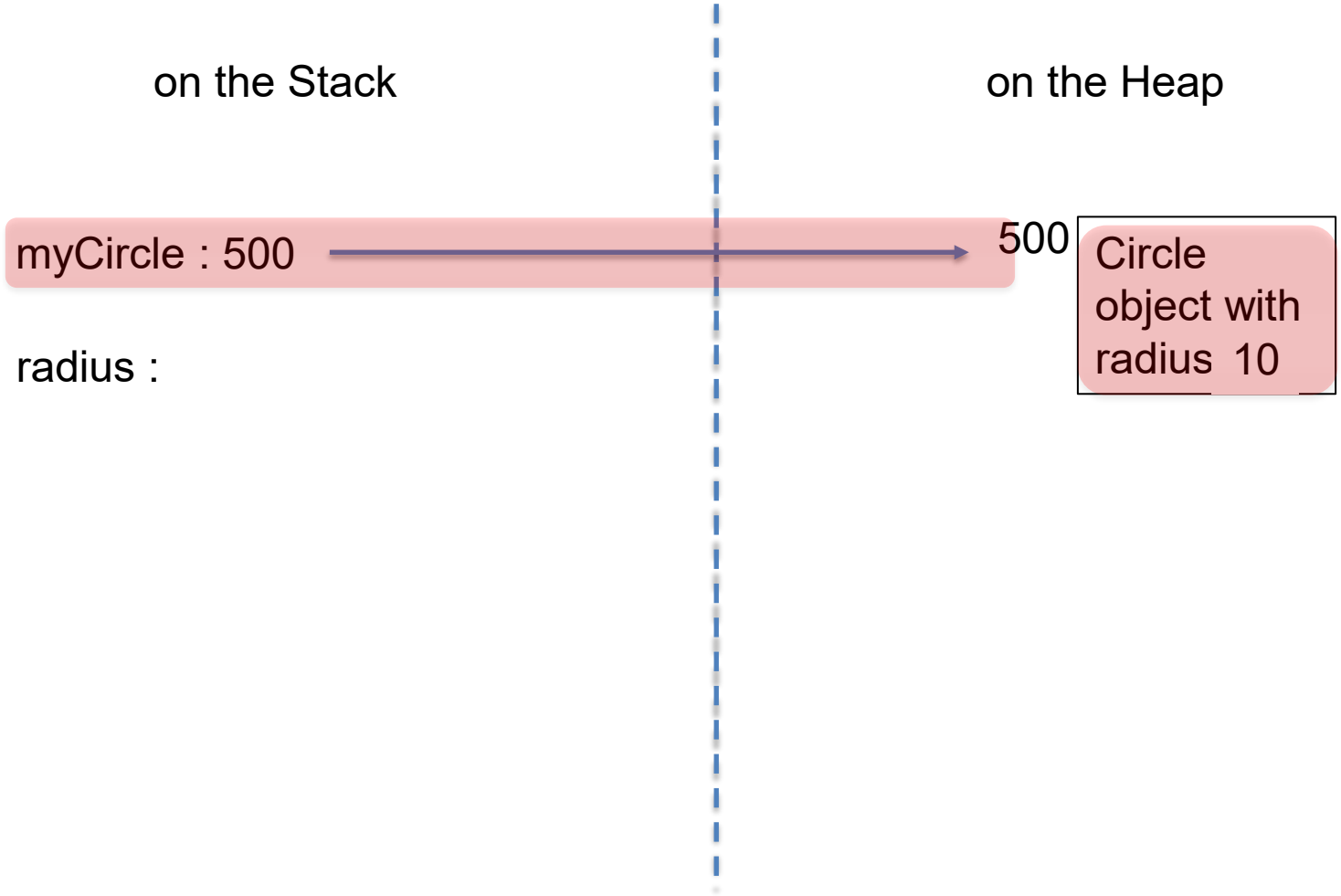
Example 1: Deep copying

```
Circle *myCircle = new Circle(10);
```



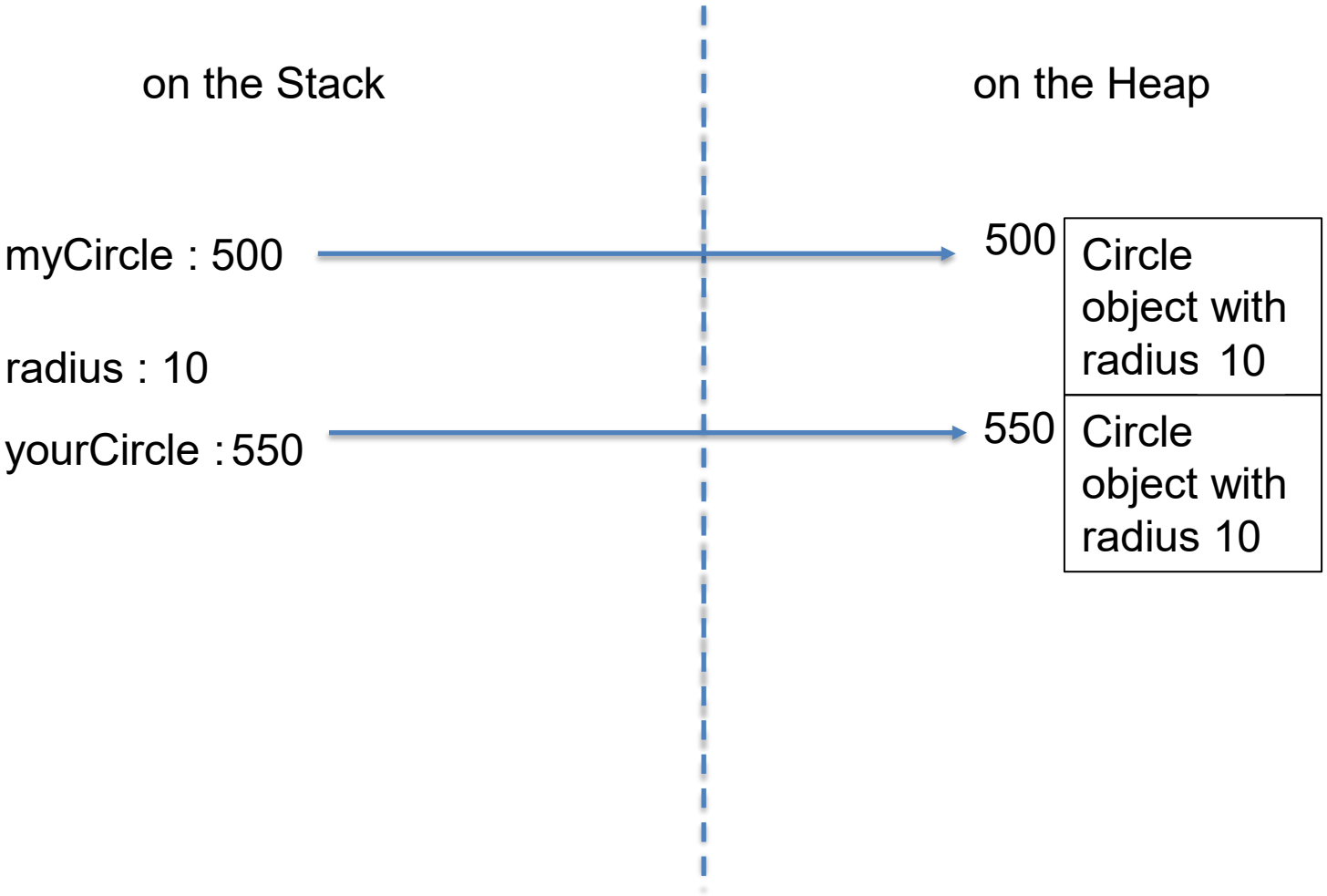
Example 1: Deep copying (cont)

```
int radius = myCircle->getRadius();
```



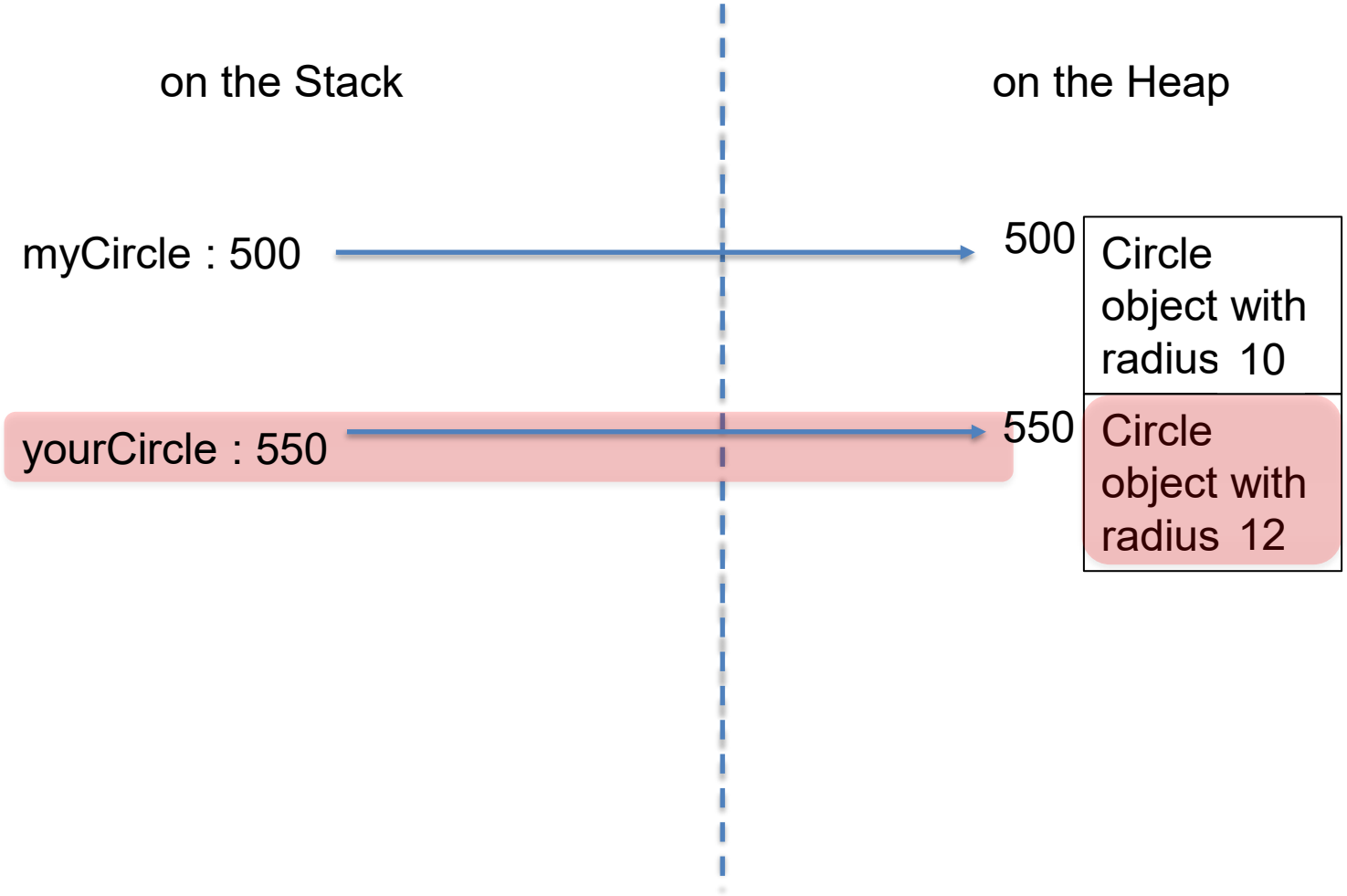
Example 1: Deep copying (cont)

```
int radius = myCircle->getRadius();
Circle *yourCircle = new Circle(radius);
```



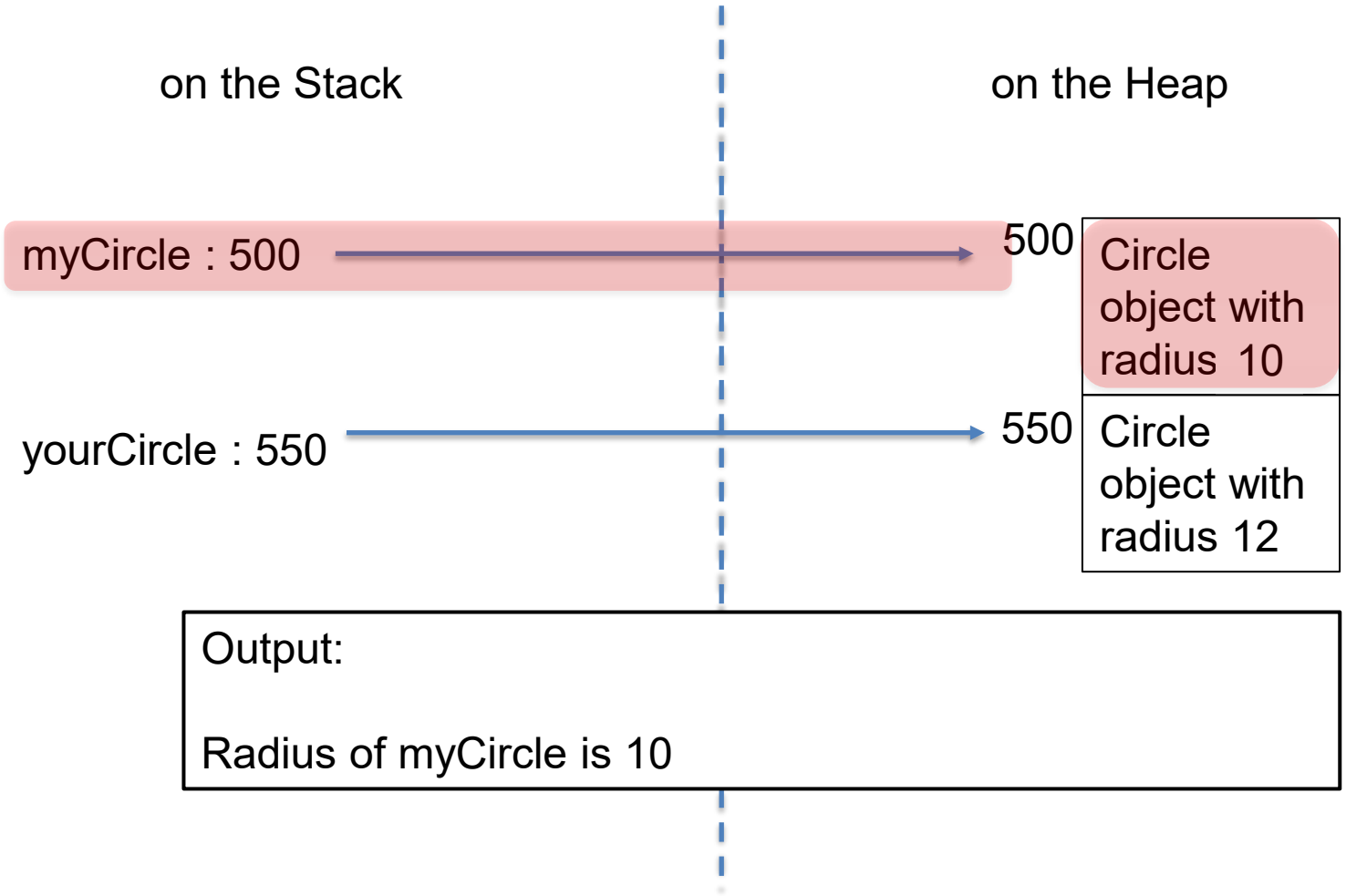
Example 1: Deep copying (cont)

```
yourCircle->setRadius(12);
```



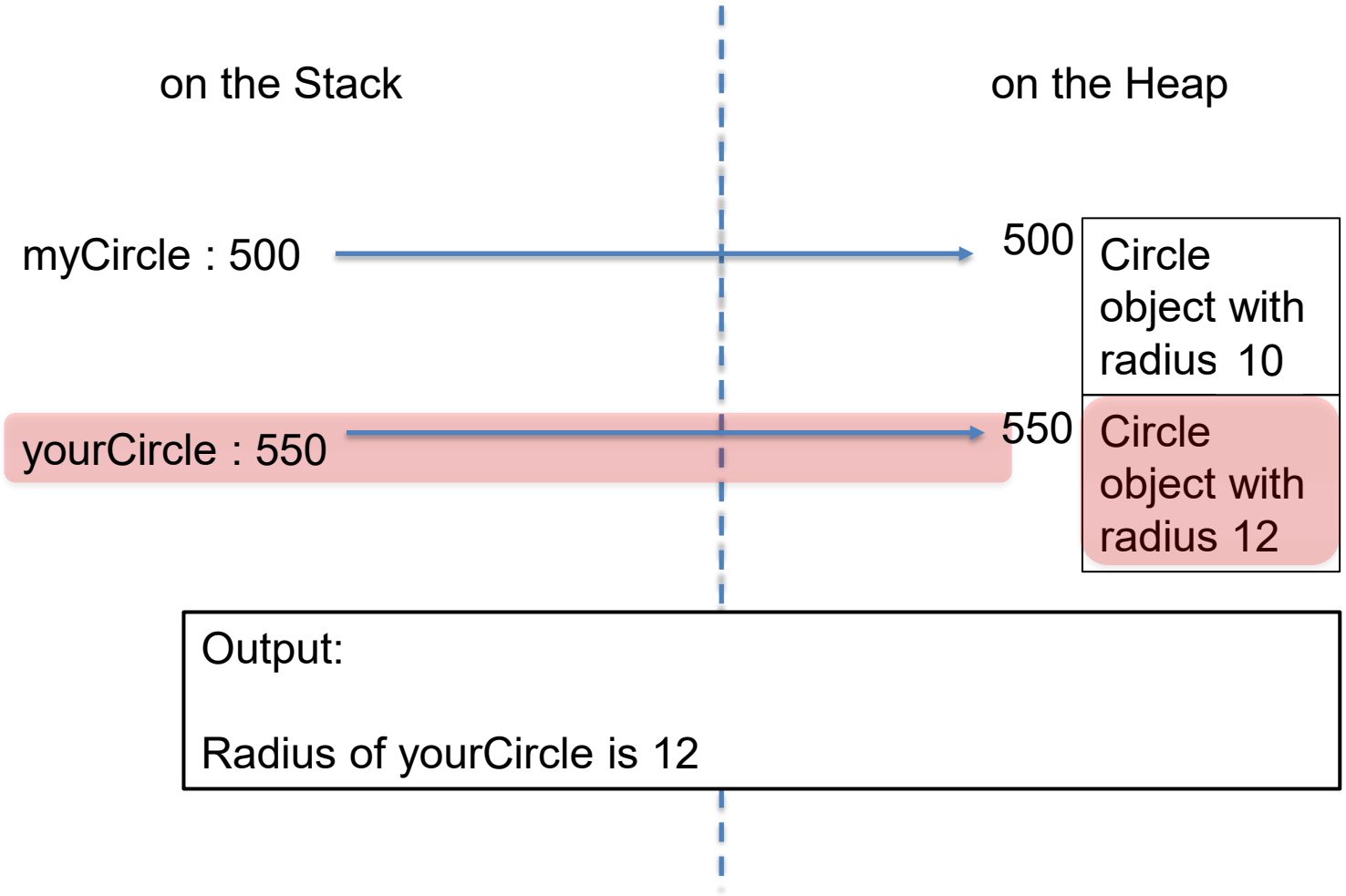
Example 1: Deep copying (cont)

```
cout<<"Radius of myCircle is "
    <<myCircle->getRadius()<<endl;
```



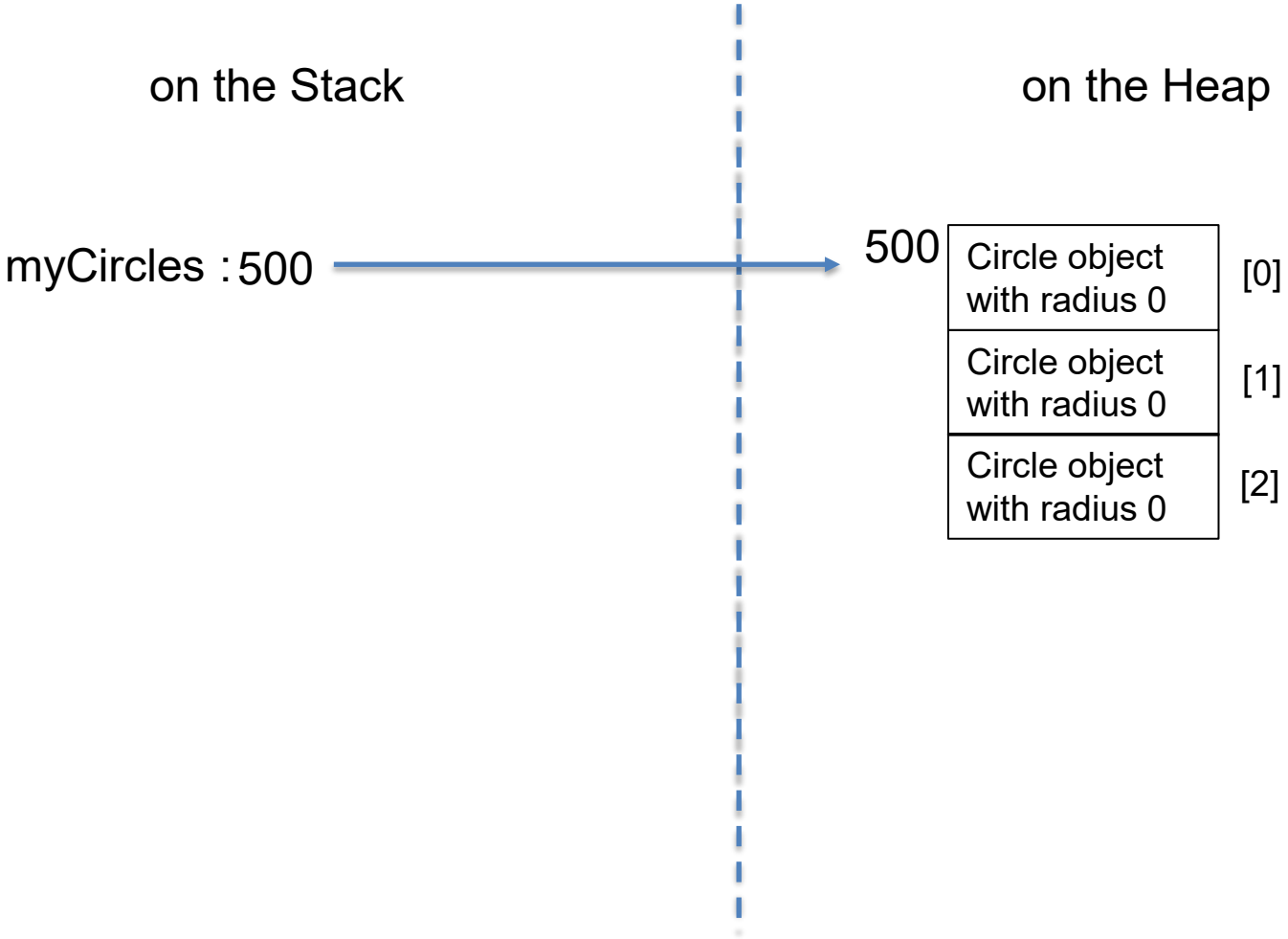
Example 1: Deep copying (cont)

```
cout<<"Radius of yourCircle is "
    <<yourCircle->getRadius()<<endl;
```



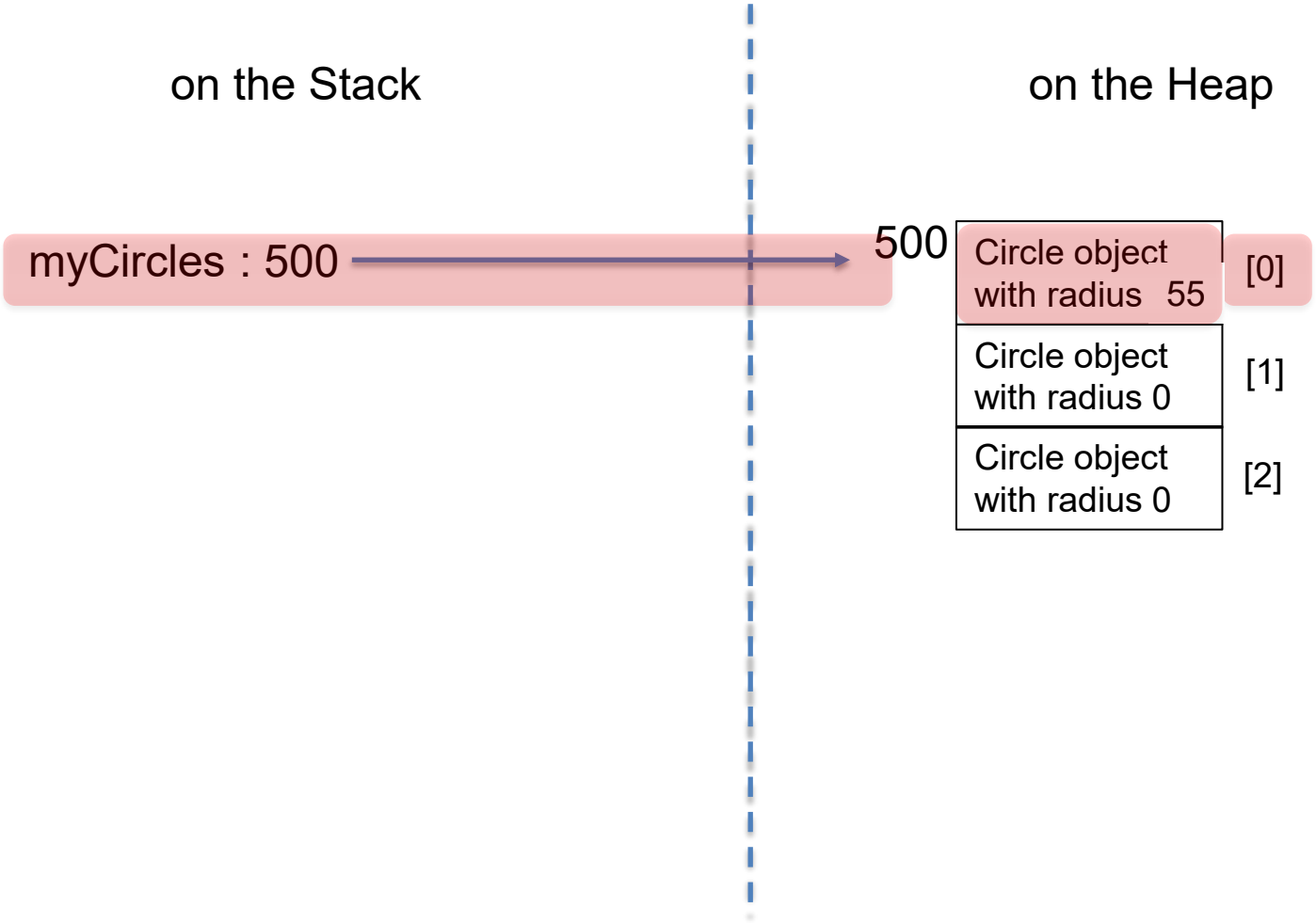
Example 2: Deep copying

```
Circle *myCircles = new Circle[3];
```



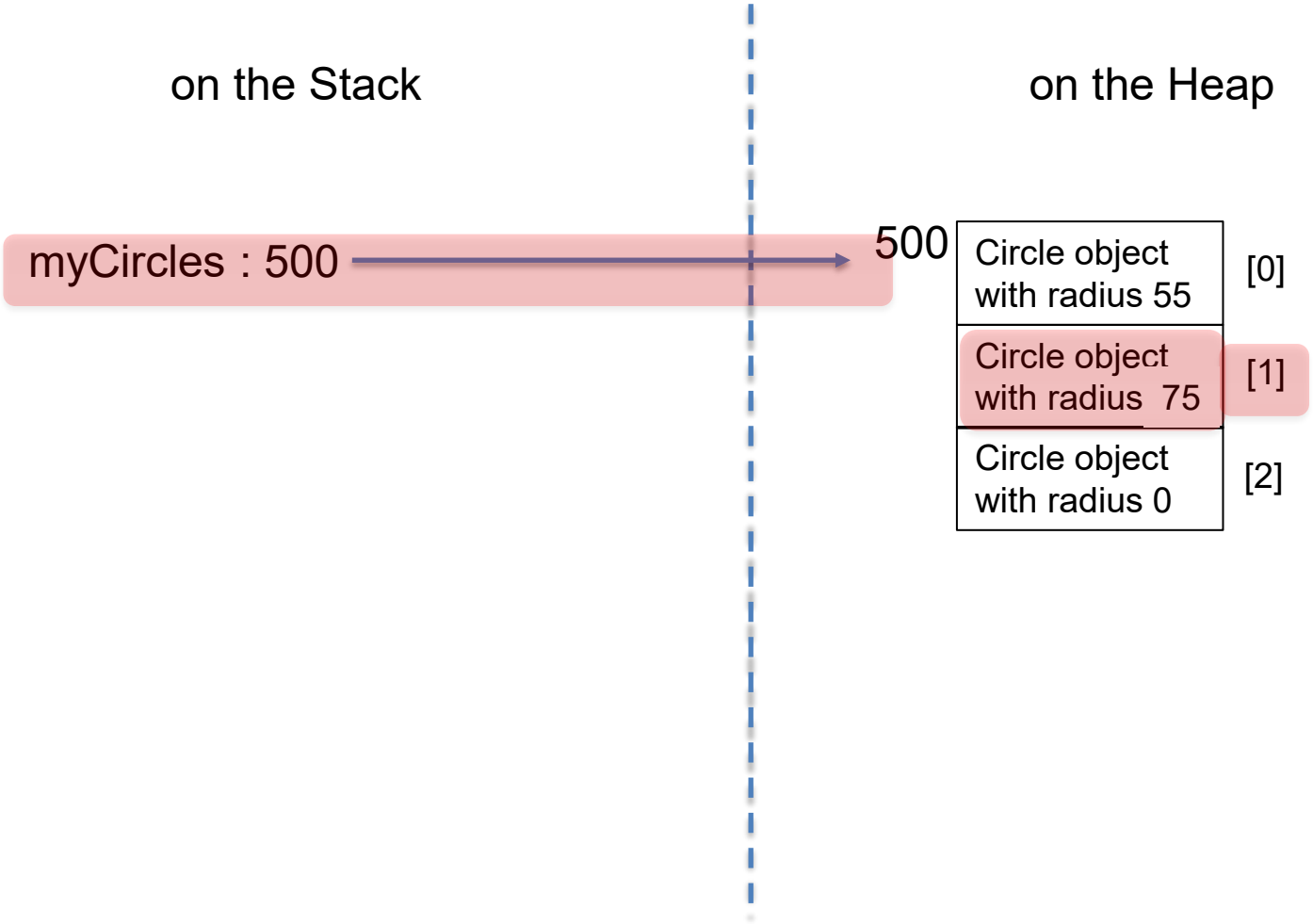
Example 2: Deep copying (cont)

`myCircle[0].setRadius(55);`



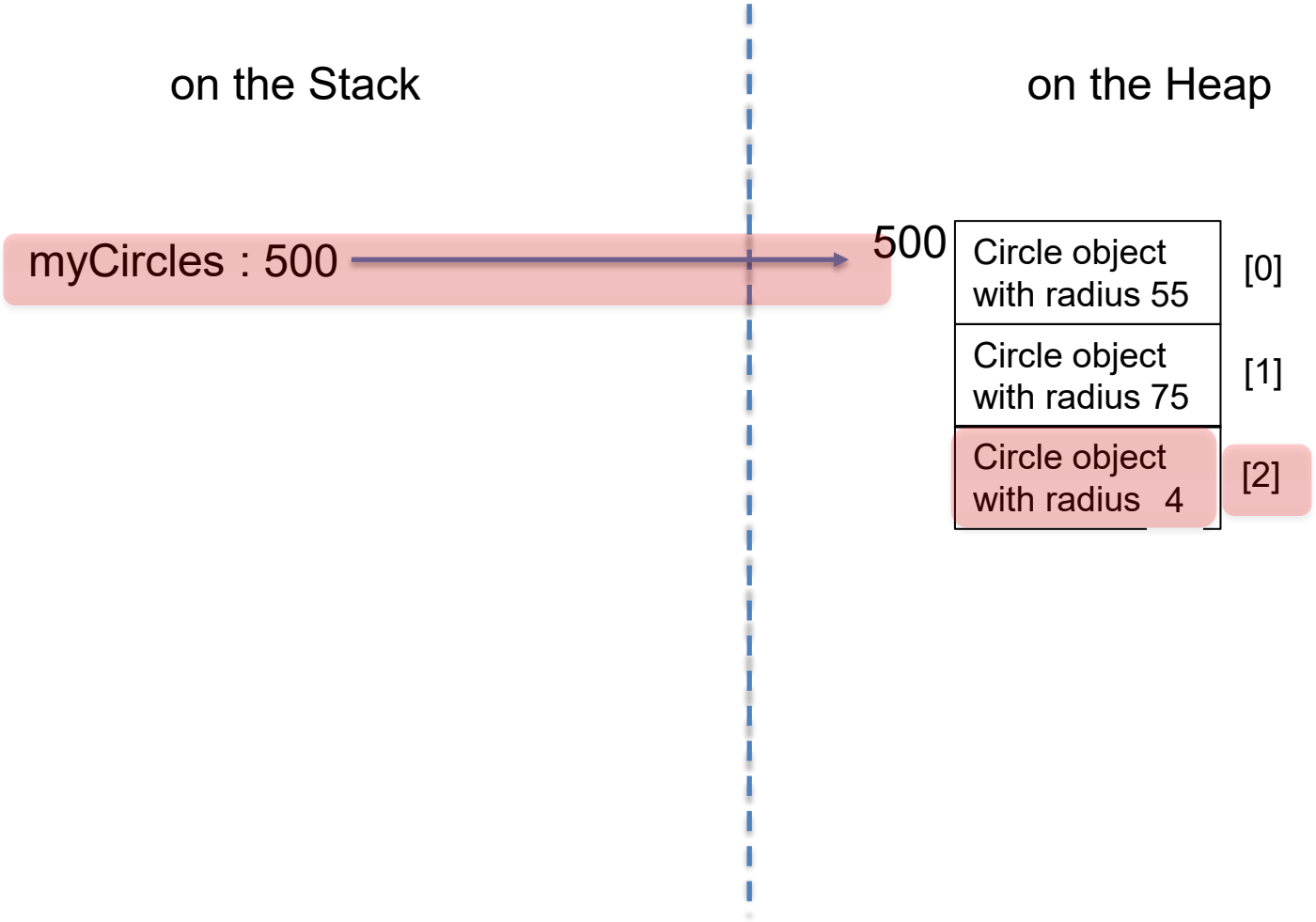
Example 2: Deep copying (cont)

`myCircle[1].setRadius(75);`



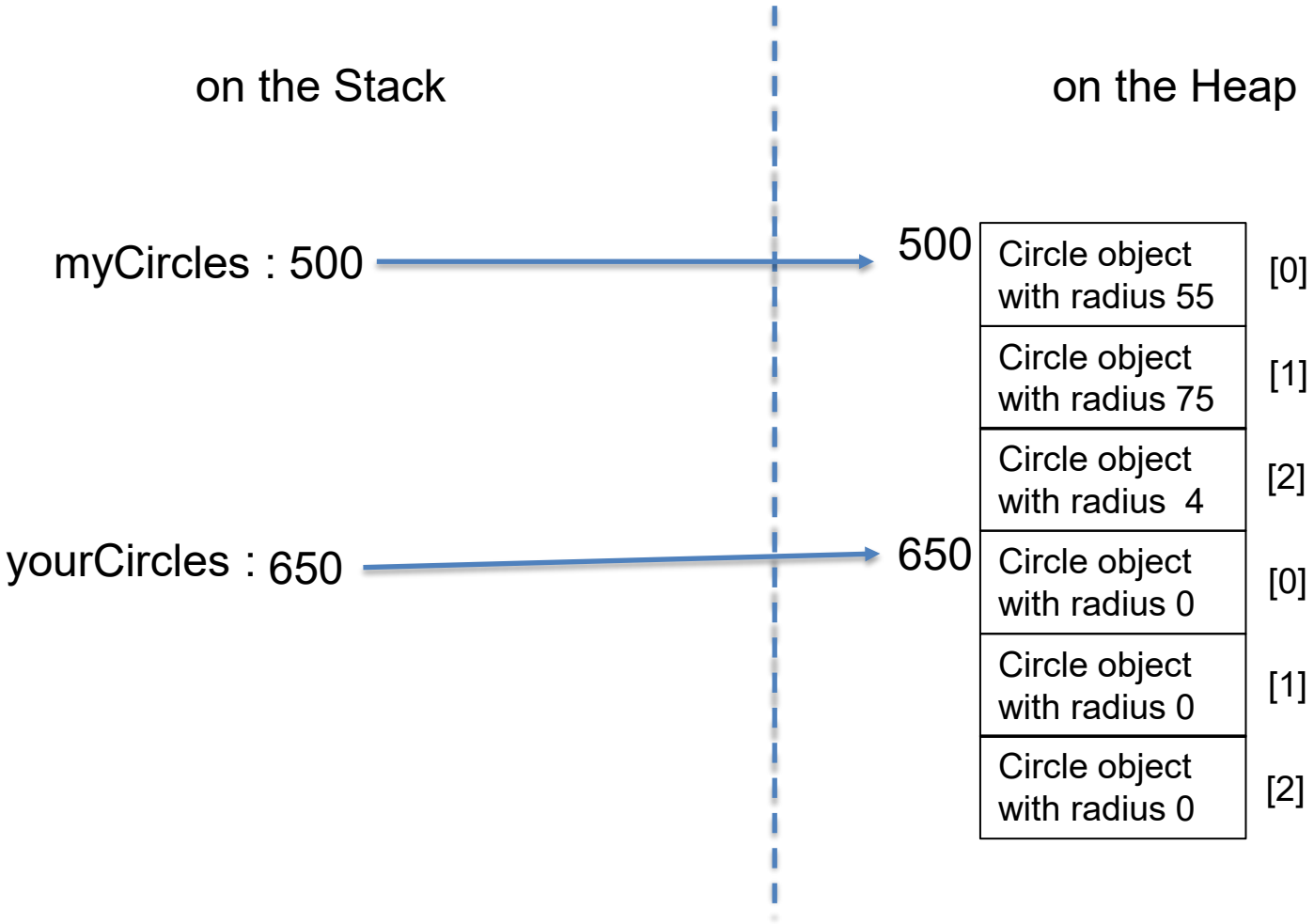
Example 2: Deep copying (cont)

`myCircle[2].setRadius(4);`



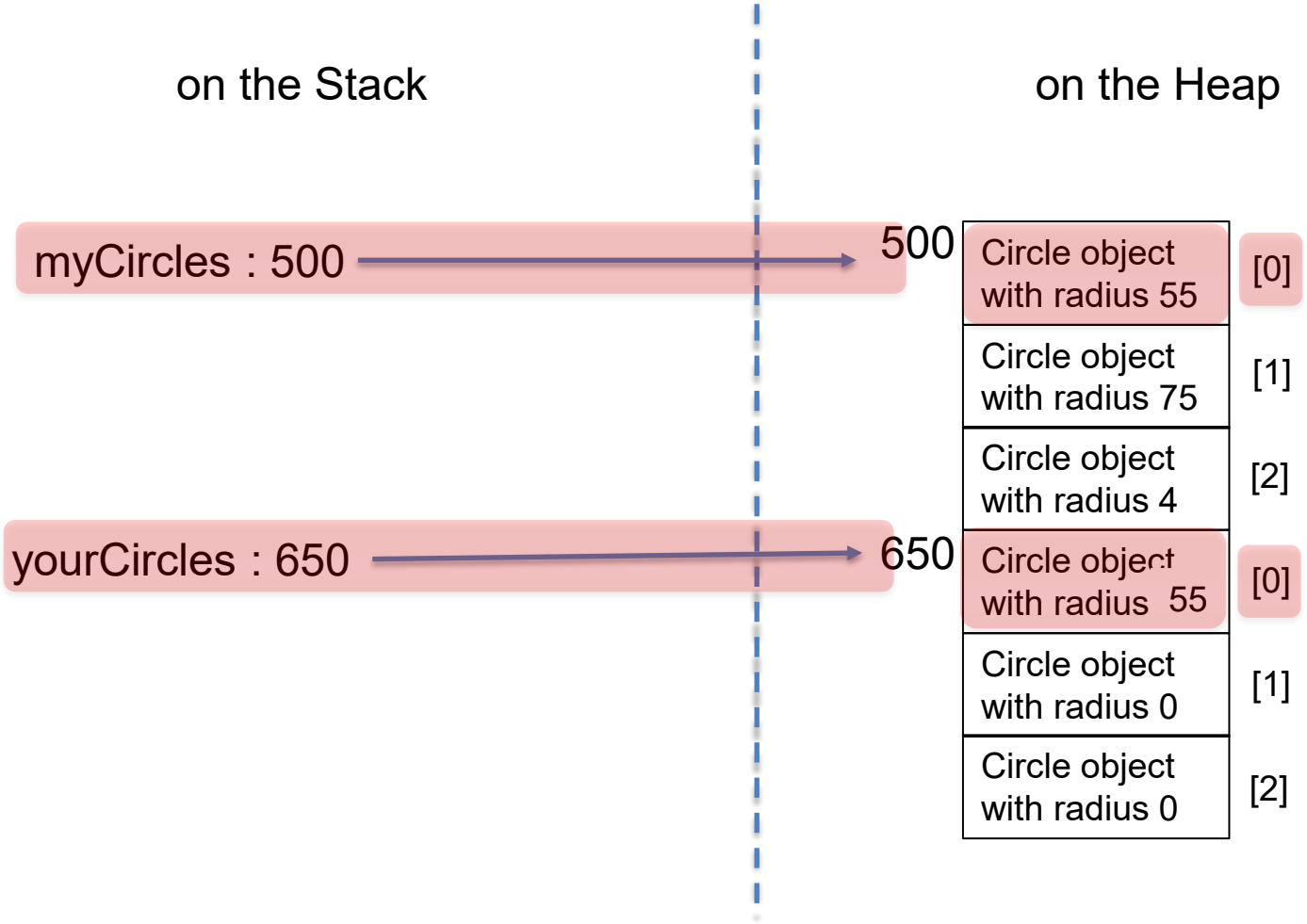
Example 2: Deep copying (cont)

```
Circle *yourCircles = new Circle[3];
```



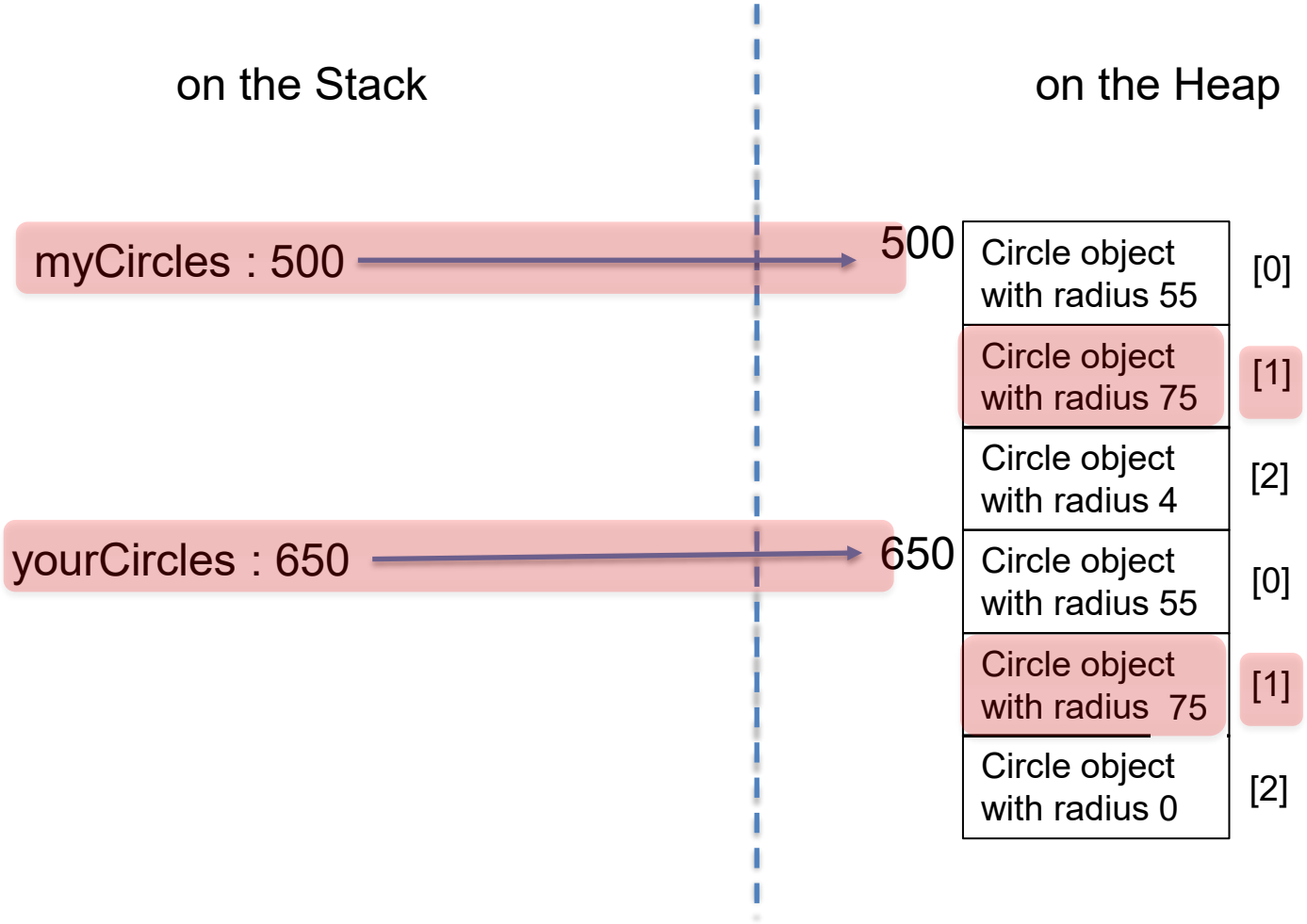
Example 2: Deep copying (cont)

```
for (int i=0; i<3; i++)
    yourCircles[i] = myCircles[i];
```



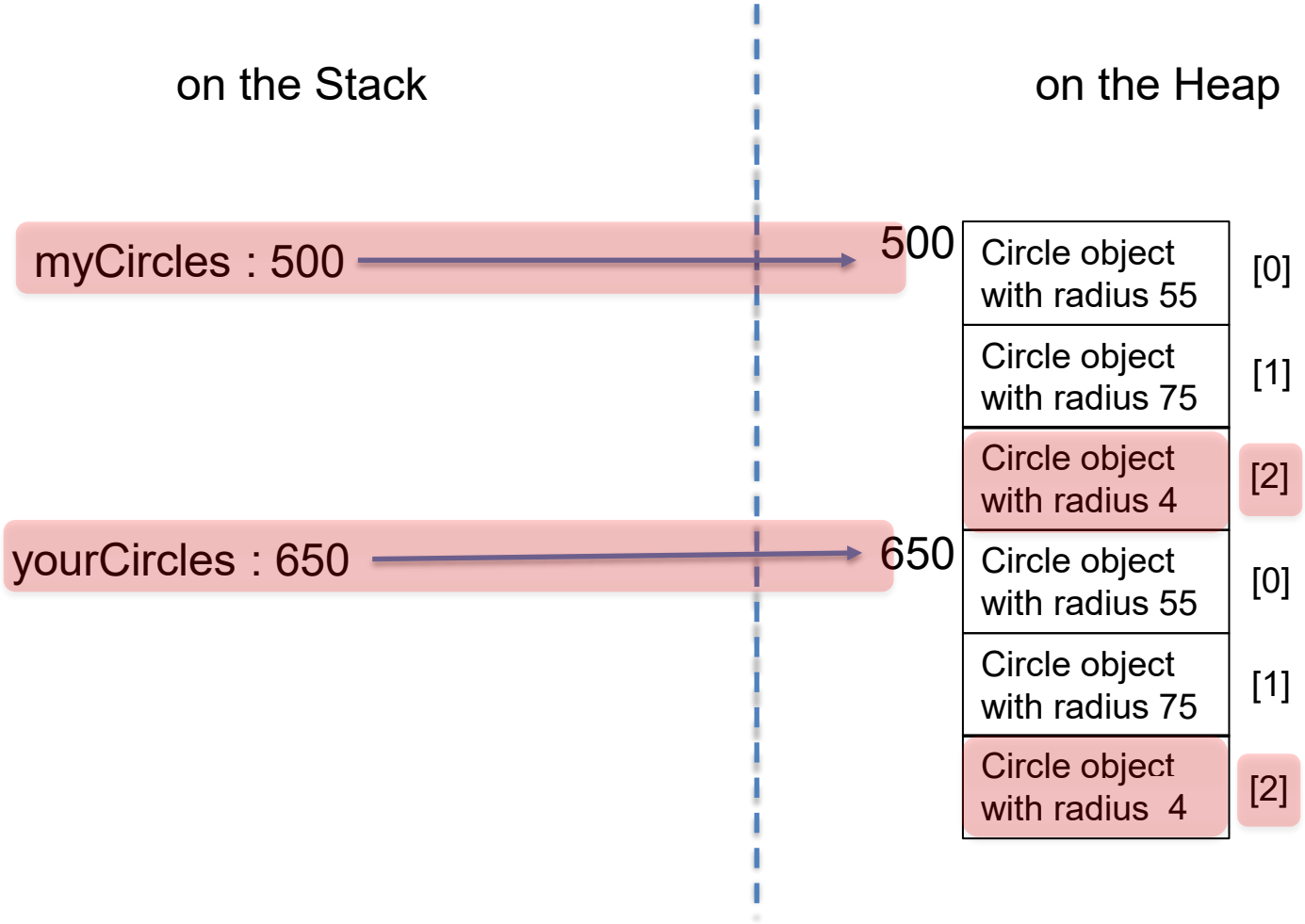
Example 2: Deep copying (cont)

```
for (int i=0; i<3; i++)
    yourCircles[i] = myCircles[i];
```



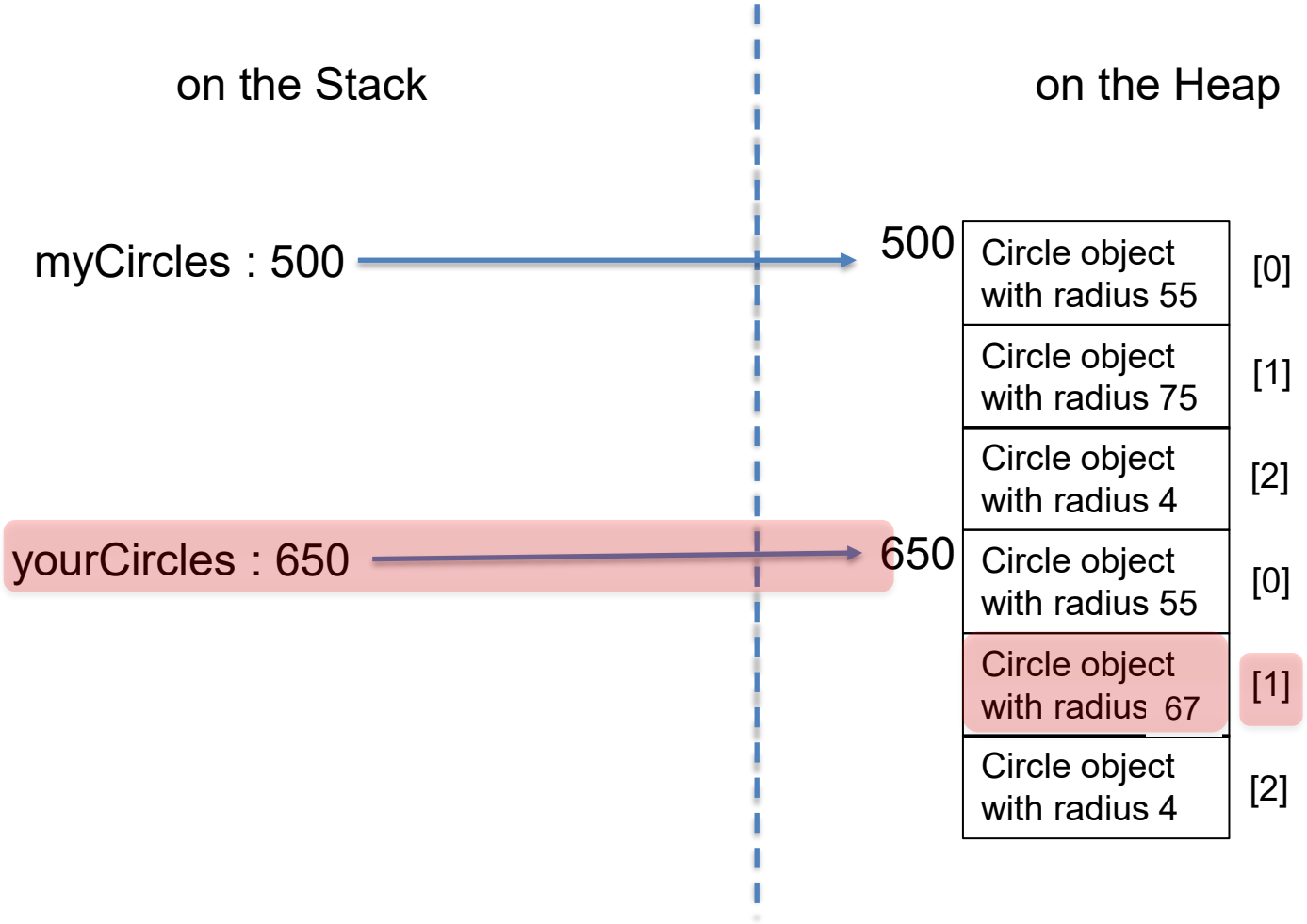
Example 2: Deep copying (cont)

```
for (int i=0; i<3; i++)
    yourCircles[i] = myCircles[i];
```



Example 2: Deep copying (cont)

```
yourCircles[1].setRadius(67);
```



Classes

- If no constructor is defined in a class it will get an automatically generated **default constructor**
- Furthermore a class will automatically get a generated
  - **Copy constructor**
  - **Assignment operator**
  - **Destructor**

Classes continued

If a class has member variables that are pointers it is necessary to replace the automatically generated

- **Default constructor**
- **Copy constructor**
- **Assignment operator**
- **Destructor**

Copy constructor

- Each class has a copy constructor
- Its purpose is to create an identical copy of another object
- Syntax:

  ClassName(const ClassName &parameterName)
- Automatically generated if not defined
  - Uses memberwise copying
  - If a member variable is a pointer this will result in shallow copying

Copy constructor continued

- If the class has at least one member variable which is a pointer it is necessary to:
  - replace the automatically generated copy construcor by implementing it
    - Use memberwise (copy-by-value) copying of member variables that are not pointers
    - Use **deep copying** for member variables that are **pointers**

Call of copy constructor

Classtype variableOfObject = variableOfAnotherObject ;

Ex..

- Circle c1(10);
- Circle c2 = c1; // call of copy constructor

Classtype variableOfObject(variableOfAnotherObject);

- Circle c1(10);
- Circle c2(c1); // call of copy constructor

And also when

- we use call-by-value

Assignment operator

- Each class has an assignment operator
- Its purpose is to create an identical copy of another object
- Syntax:

  void operator=(const ClassName &parameterName)

  (*return type could also be ClassName or ClassName&*)
- Automatically generated if not defined
  - Uses memberwise copying
  - If a member variable is a pointer this will result in shallow copying

Assignment operator continued

- If the class has at least one member variable which is a pointer it is necessary to:
  - replace the automatically generated assignment operator by implementing it
    - If the object is not assigned to itself
      - Deallocate dynamically allocated memory
      - Use memberwise (copy-by-value) copying of member variables that are not pointers
      - Use **deep copying** for member variables that are **pointers**

Call of assignment operator

variableOfObject = variableOfObject;

Circle c1(12);

Circle c2(10);

…

c2 = c1; // call of assignment operator

Destructor

- If the class has at least one member variable which is a pointer it is necessary to:
  - Deallocate the memory that has been allocated for the pointers by the object