Data Structures and Object Oriented Programming

Lecture 20

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Inheritance

Copy Constructor & Assignment Operator

Copy Constructor

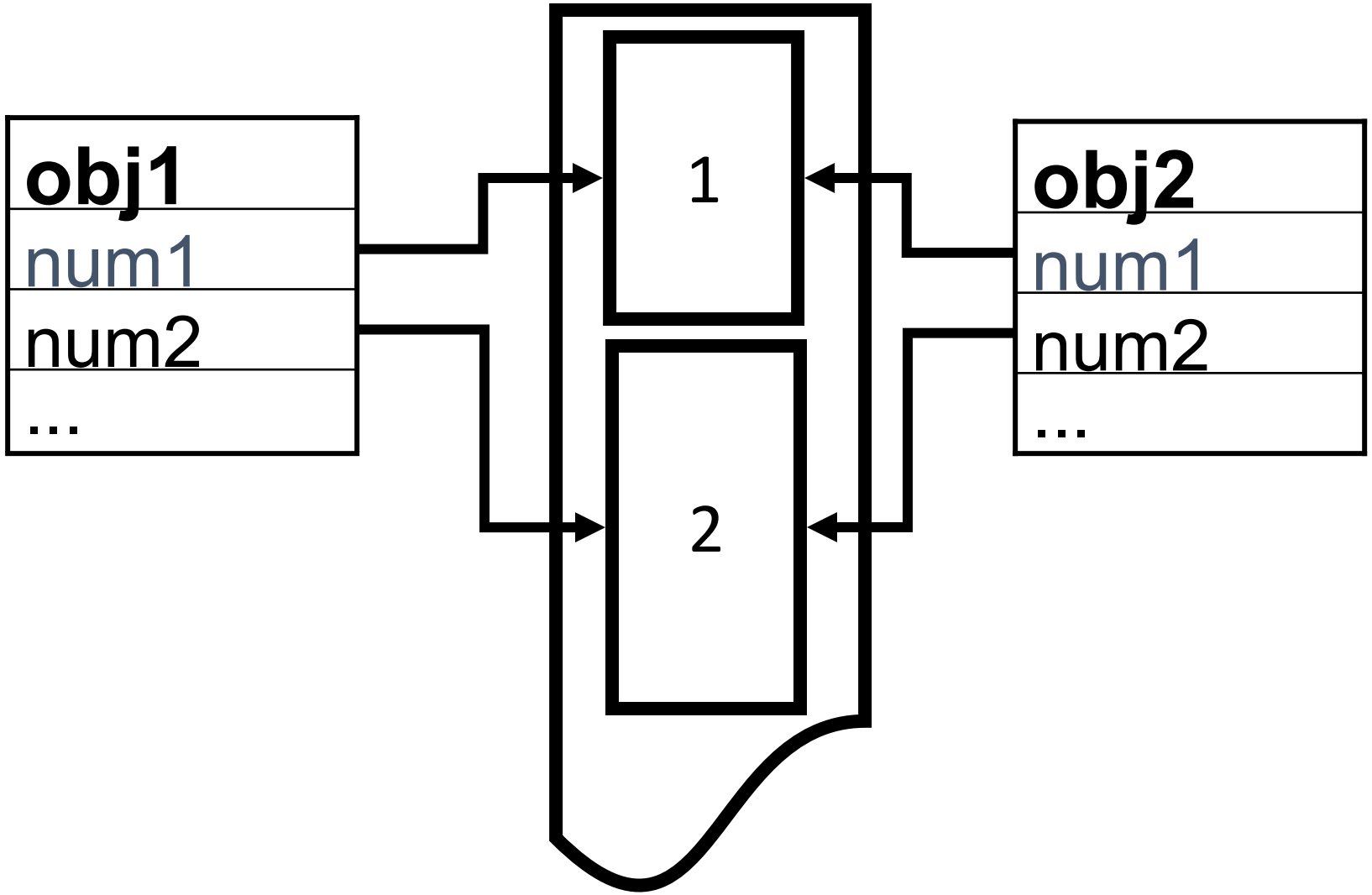
- Compiler generates copy constructor for base and derived classes, if needed
- Derived class Copy constructor is invoked which in turn calls the Copy constructor of the base class
- The base part is copied first and then the derived part

Example

```
class parent{
    int* num1;
public:
    parent(int a=0)
    {
        num1 = new int;
        *num1 = a;
    }
    int get_num()
    {
        return *num1;
    }
};
```

```
class child: public parent {
    int* num2;
public:
    child(int a=0, int b=0):parent(a)
    {
    num2 = new int;
    *num2 = b;
    }
    void print() {
    cout << get_num() << endl;
    cout << *num2 << endl;
    }
};

int main()
{
    child obj1(1, 2);
    child obj2 = obj1;
    obj1.print();
    obj2.print()
}
```

Shallow Copy

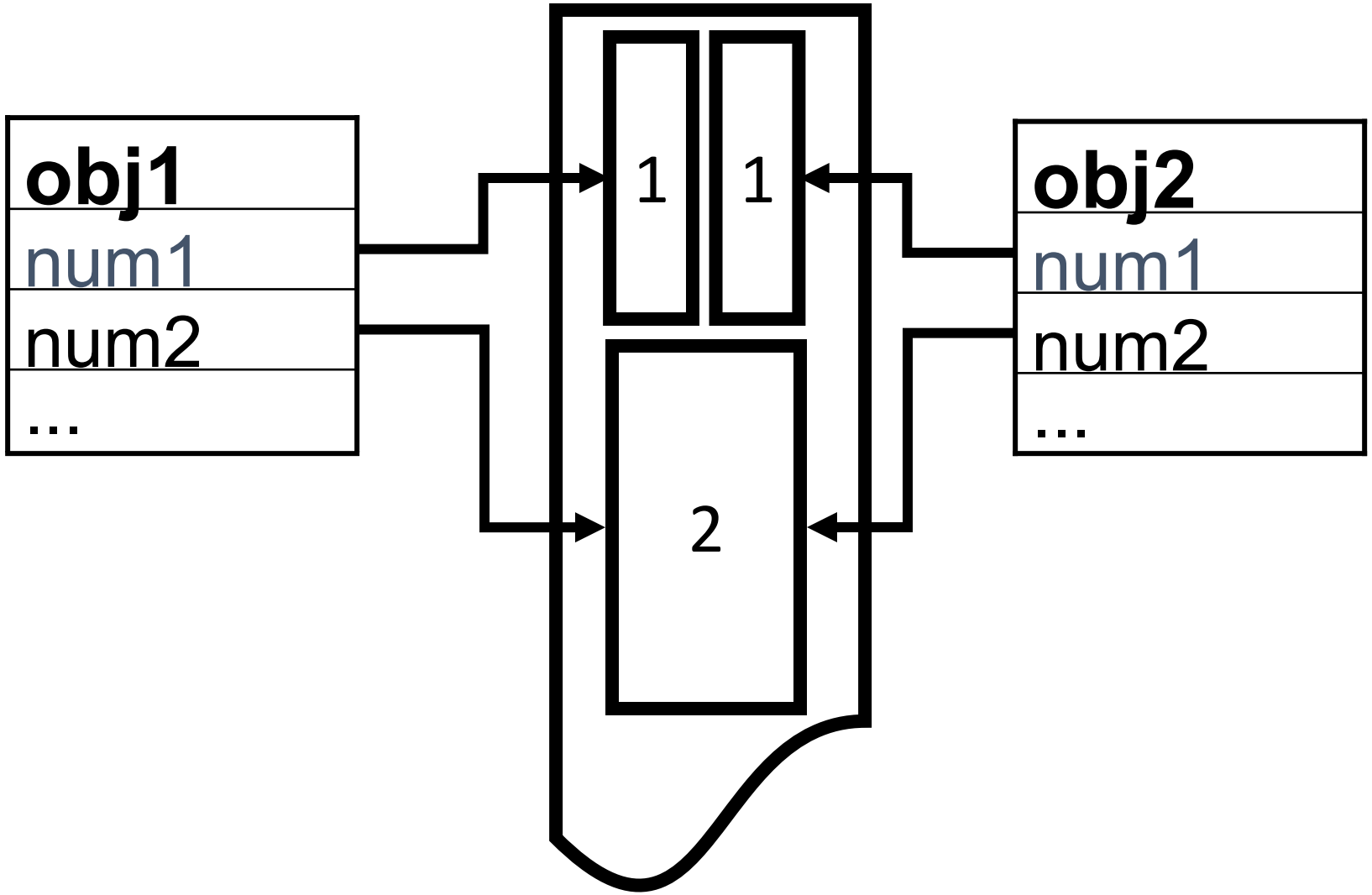


Example

```
class parent{
    int* num1;
public:
    parent(int a=0)
    {
    num1 = new int;
    *num1 = a;
    }
    int get_num() {
        return *num1;
    }
    parent(const parent& a)  // deepcopy
    {
        num1 = new int;
        *num1 = *a.num1;
    }

};
```

```
class child: public parent {
    int* num2;
public:
    child(int a=0, int b=0):parent(a)
    {
    num2 = new int;
    *num2 = b;
    }
    void print() {
    cout << get_num() << endl;
    cout << *num2 << endl;
    }
};

int main()
{
    child obj1(1, 2);
    child obj2 = obj1;
    obj1.print();
    obj2.print()
}
```

Deep and Shallow Copy



Copy Constructor

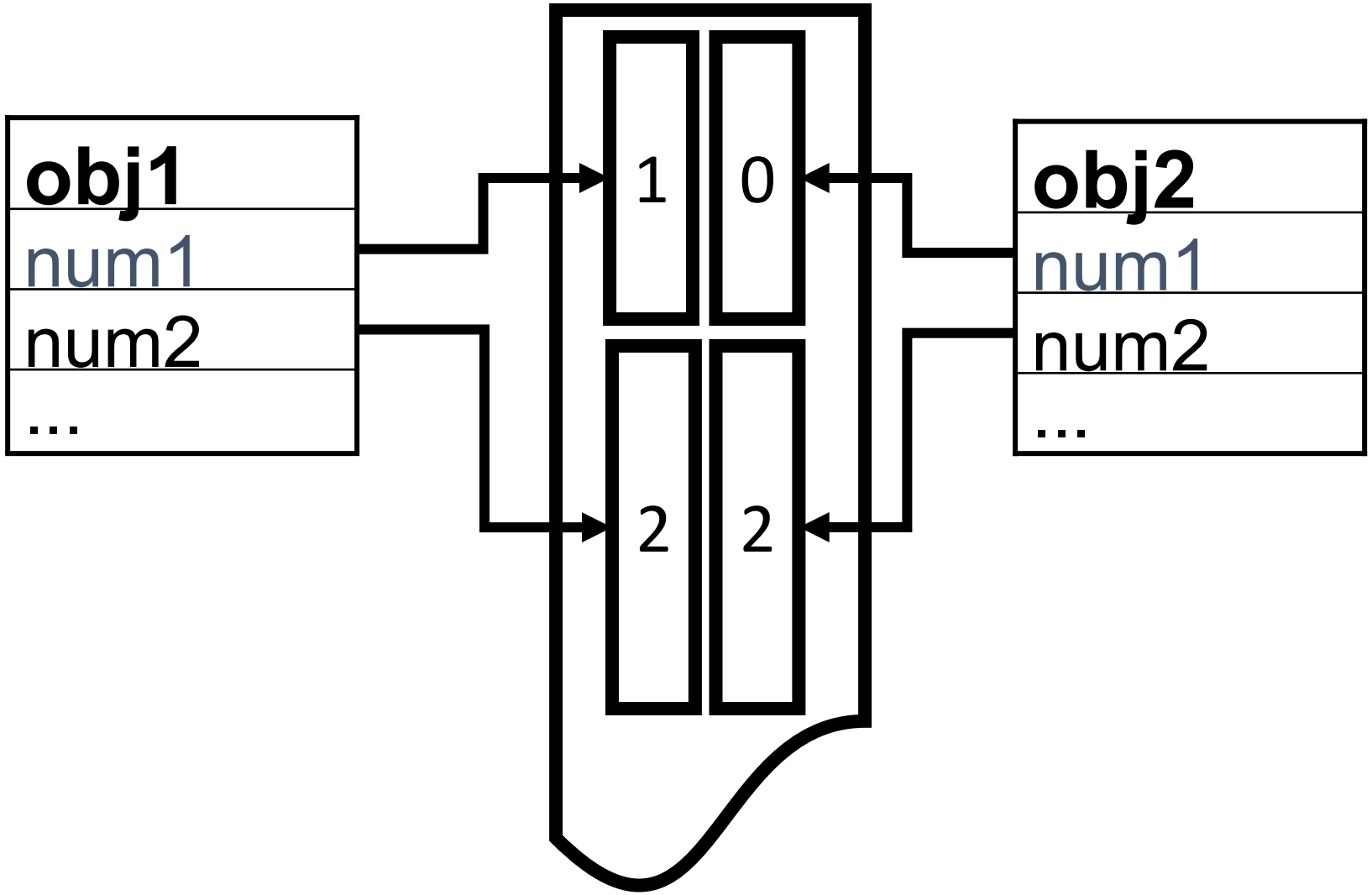
- Compiler generates copy constructor for derived class
- Calls the copy constructor of the base class (**deep copy**)
- Then performs the **shallow copy** of the derived class's data members

Example

```
class parent{
    int* num1;
public:
    parent(int a=0)
    {
    num1 = new int;
    *num1 = a;
    }
    parent(const parent& a)  // deepcopy
    {
        num1 = new int;
        *num1 = *a.num1;
    }
    int get_num() {
    return *num1;
    }
};
```

```
class child: public parent {
    int* num2;
public:
    child(int a=0, int b=0):parent(a){
        num2 = new int;
        *num2 = b; }
    void print() {
        cout << get_num() << endl;
        cout << *num2 << endl;
    }
    child(const child& a){     // deepcopy
        num2 = new int;
        *num2 = *a.num2;}
};

int main()
{
    child obj1(1, 2);
    child obj2 = obj1;
    obj1.print();
    obj2.print()
}
```

Deep Copy



Copy Constructor

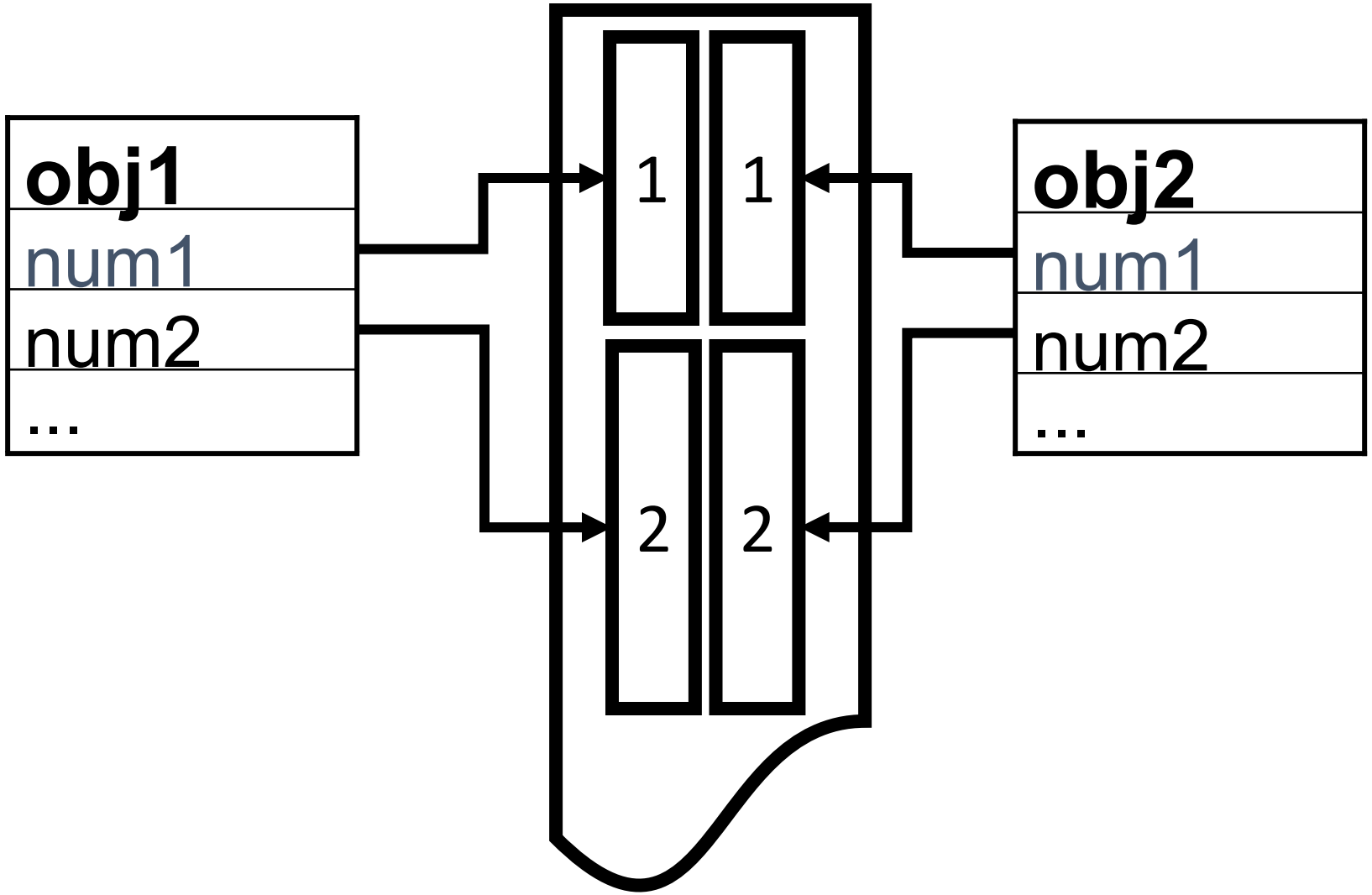
- Programmer must explicitly call the base class copy constructor from the copy constructor of derived class

Example

```
class parent{
    int* num1;
public:
    parent(int a=0)
    {
    num1 = new int;
    *num1 = a;
    }
    parent(const parent& a)  // deepcopy
    {
        num1 = new int;
        *num1 = *a.num1;
    }
    int get_num() {
    return *num1;
    }
};
```

```
class child: public parent {
    int* num2;
public:
    child(int a=0, int b=0):parent(a){
        num2 = new int;
        *num2 = b; }
    void print() {
        cout << get_num() << endl;
        cout << *num2 << endl;
    }
    child(const child& a):parent(a) // deepcopy
        num2 = new int;
        *num2 = *a.num2;}
};

int main()
{
    child obj1(1, 2);
    child obj2 = obj1;
    obj1.print();
    obj2.print()
}
```

Deep Copy



Assignment Operator

- Compiler generates copy assignment operator for base and derived classes, if needed
- Derived class copy assignment operator is invoked which in turn calls the assignment operator of the base class
- The base part is assigned first and then the derived part

Assignment Operator

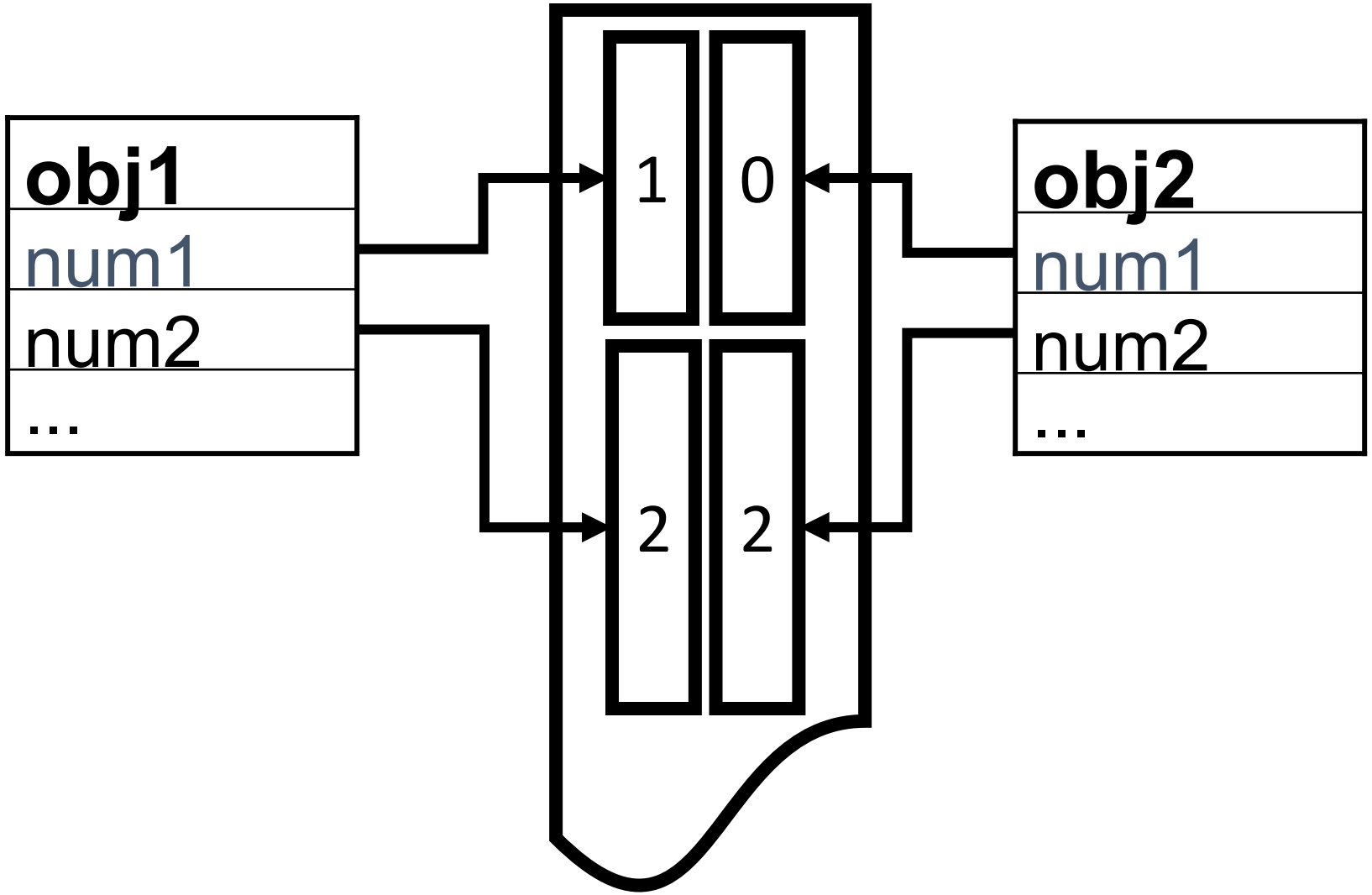
- Programmer has to call operator of base class, if he is writing assignment operator of derived class

Example

```
class parent{
    int* num1;
public:
    parent(int a=0)
    {
        num1 = new int;
        *num1 = a;
    }
    parent& operator = (const parent& a)
    {
        num1 = new int;
        *num1 = *a.num1;
        return *this;
    }

    int get_num() {
        return *num1;
    }
};
```

```
class child: public parent {
    int* num2;
public:
    child(int a=0, int b=0):parent(a){
        num2 = new int;
        *num2 = b;}
    child& operator = (const child& a){
        num2 = new int;
        *num2 = *a.num2;
        return *this;}
    void print() {
        cout << get_num() << endl;
        cout << *num2 << endl;
    }
};      int main()
        {
            child obj1(1, 2);
            child obj2;
            obj2 = obj1;
            obj1.print();
            obj2.print()
        }
```

Deep Copy

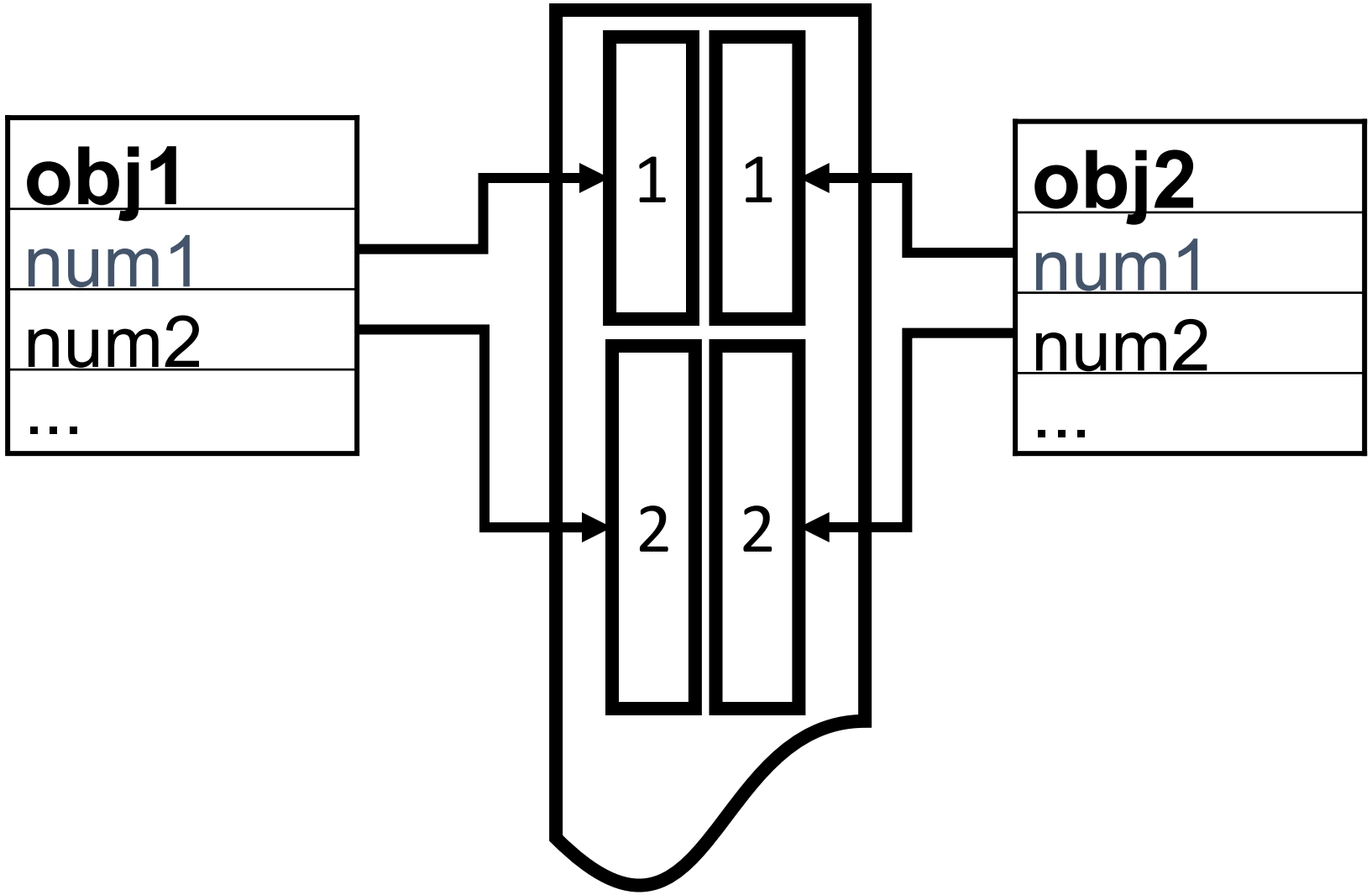


Example

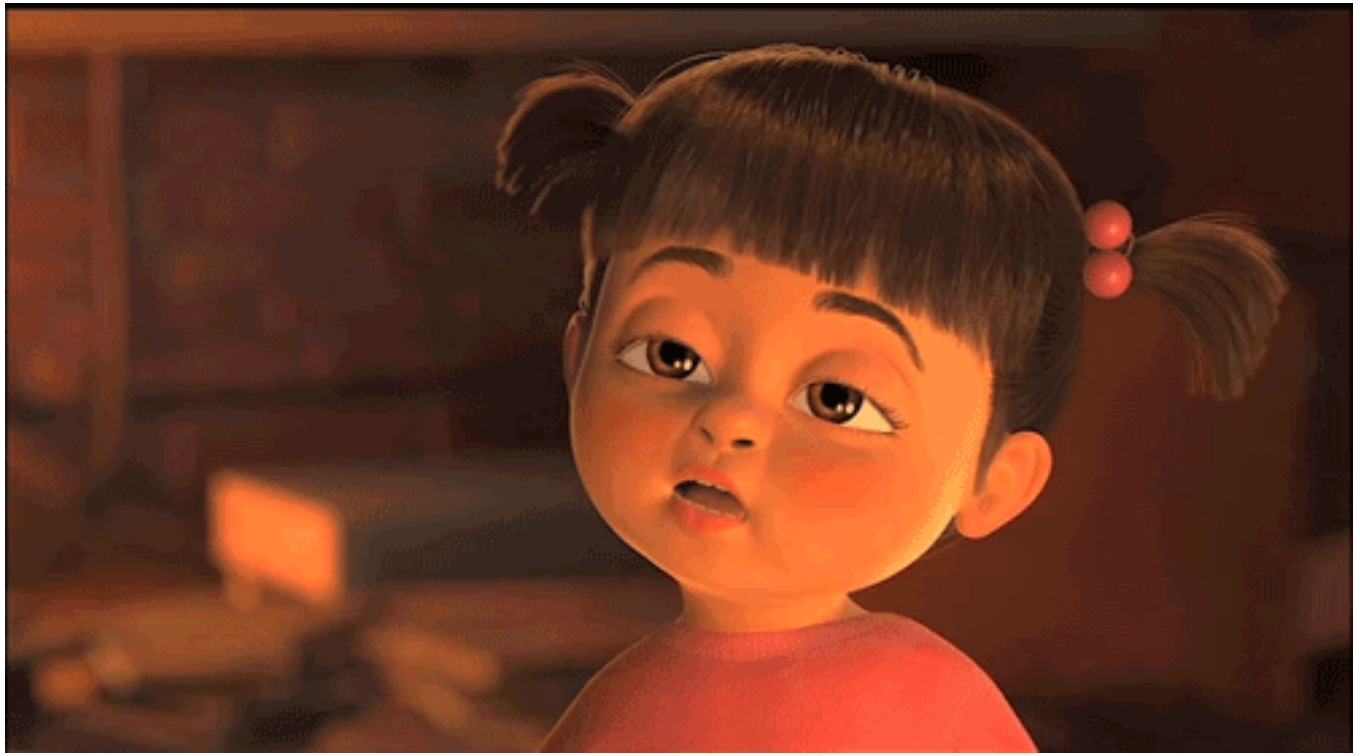
```
class parent{
    int* num1;
public:
    parent(int a=0)
    {
        num1 = new int;
        *num1 = a;
    }
    parent& operator = (const parent& a)
    {
        num1 = new int;
        *num1 = *a.num1;
        return *this;
    }

    int get_num() {
        return *num1;
    }
};
```

```
class child: public parent {
    int* num2;
public:
    child(int a=0, int b=0):parent(a){
        num2 = new int;
        *num2 = b;}
    child& operator = (const child& a){
        parent::operator = (a);
        num2 = new int;
        *num2 = *a.num2;
        return *this;}
    void print() {
        cout << get_num() << endl;
        cout << *num2 << endl;
    }
};
 int main()
 {   child obj1(1, 2);
     child obj2;
     child obj2 = obj1;
     obj1.print();
     obj2.print()   }
```

Deep Copy



Thanks a lot



If you are taking a Nap, **wake up**.....Lecture Over