The Diamond Problem and Virtual Base Classes

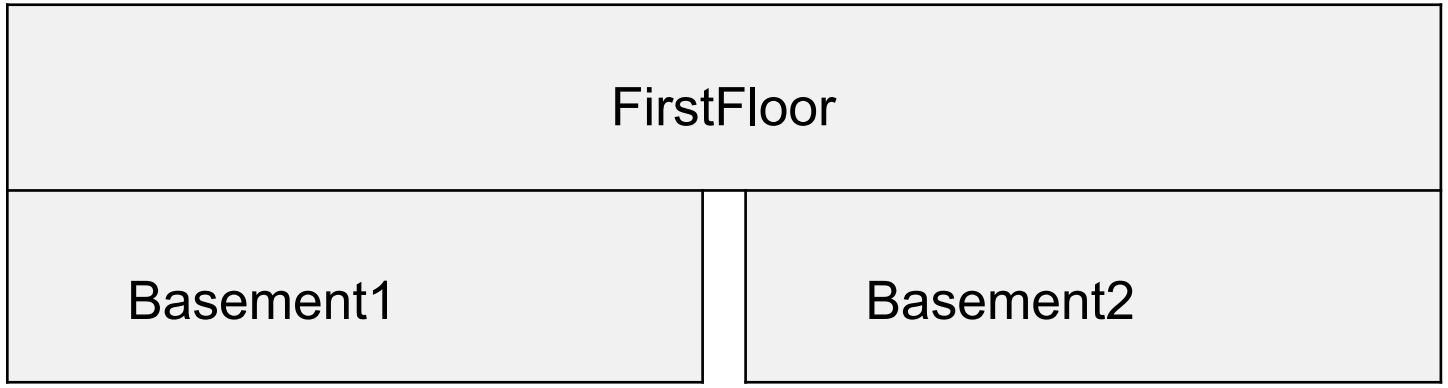
CSE 205 - Week 11 Class 1

Lec Raiyan Rahman
Dept of CSE, MIST
raiyan@cse.mist.ac.bd
CC: Lec Saidul Hoque Anik

Let’s do a quick recap of Multiple Inheritance

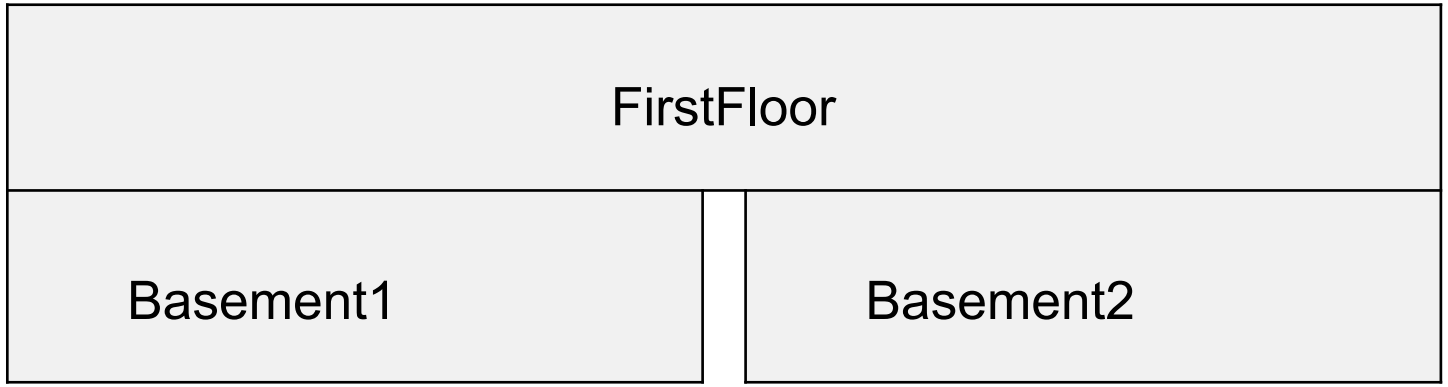
Hierarchical Inheritance/Multiple Inheritance

Derived Class with multiple base classes



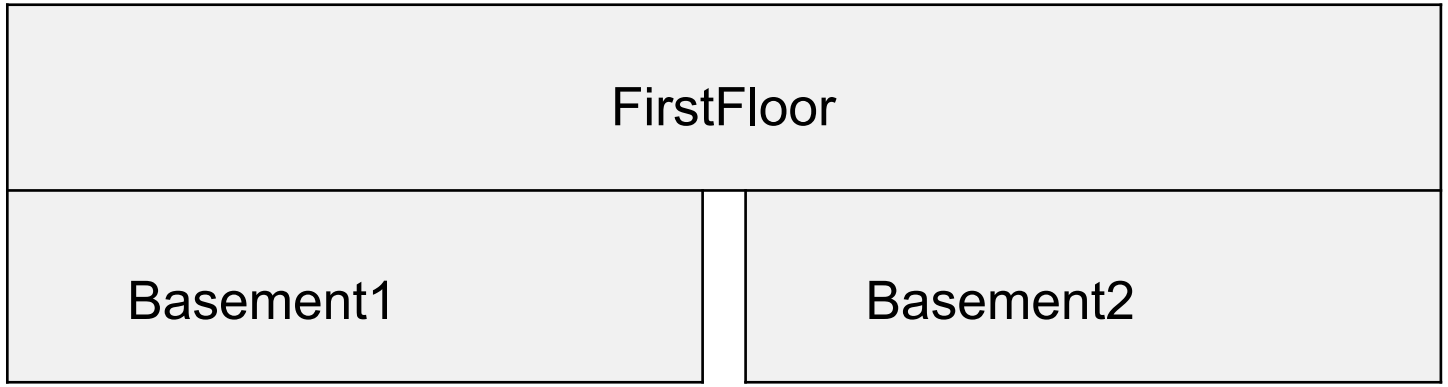
Hierarchical Inheritance

What will be the order of constructor?



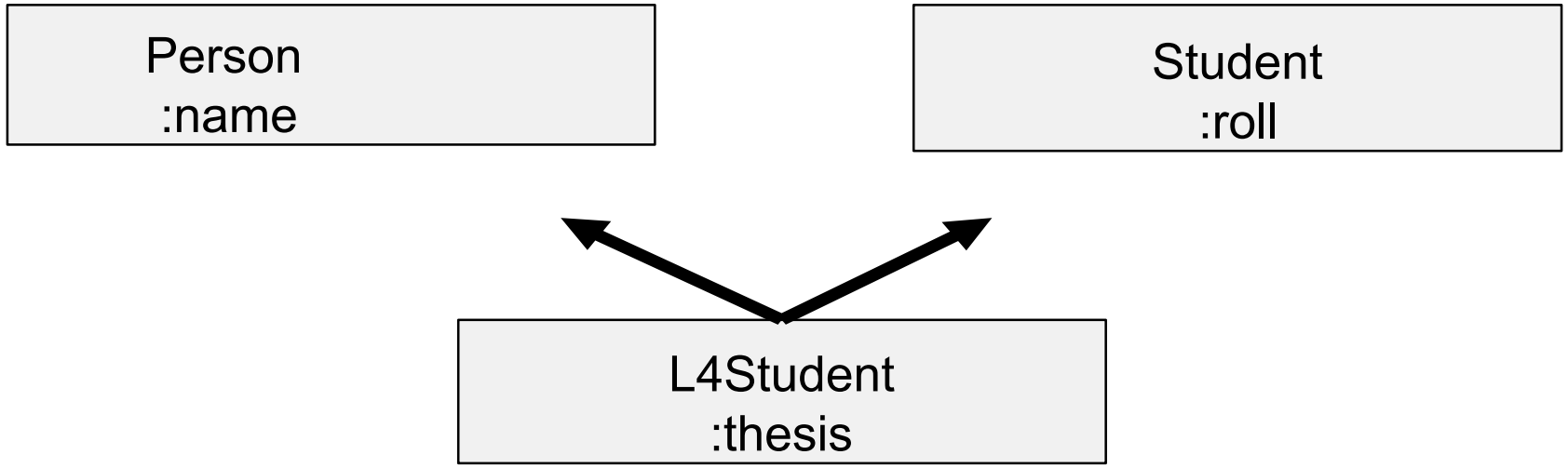
Hierarchical Inheritance

What will be the order of destructor?



Hierarchical Inheritance

Derived Class with multiple base classes



Hierarchical Inheritance

Derived Class with multiple base classes

```
class Person
{
    char name[100];
public:
    Person(char * n)
    {
        strcpy(name, n);
    }
    void printName()
    {
        cout << name << endl;
    }
};
```

```
class Student
{
    int roll;
public:
    Student(int r)
    {
        roll = r;
    }
    void printRoll()
    {
        cout << roll << endl;
    }
};
```

Hierarchical Inheritance

Derived Class with multiple base classes

```
class L4Student : public Person, public Student
{
    int thesis;
public:
    L4Student(char * name, int roll, int th): Person(name), Student(roll)
    {
        thesis = th;
    }
    void printMark()
    {
        cout << thesis << endl;
    }
};
```

Hierarchical Inheritance

Derived Class with multiple base classes

This will be called first

```
class L4Student : public Person, public Student
{
    int thesis;
public:
    L4Student(char * name, int roll, int th): Person(name), Student(roll)
    {
        thesis = th;
    }
    void printMark()
    {
        cout << thesis << endl;
    }
};
```

Hierarchical Inheritance

Derived Class with multiple base classes

```
class L4Student : public Person, public Student
{
    int thesis;
public:
    L4Student(char * name, int roll, int th): Person(name), Student(roll)
    {
        thesis = th;
    }
    void printMark()
    {
        cout << thesis << endl;
    }
};
```

This will be called first

This will be called after first

Hierarchical Inheritance

Derived Class with multiple base classes

```
class L4Student : public Person, public Student
{
    int thesis;
public:
    L4Student(char * name, int roll, int th): Person(name), Student(roll)
    {
        thesis = th;
    }
    void printMark()
    {
        cout << thesis << endl;
    }
};
```

This will be called first

This will be called after first

This will be called at last

Hierarchical Inheritance

What will be the order of destructor?

```
class L4Student : public Person, public Student
{
    int thesis;
public:
    L4Student(char * name, int roll, int th): Person(name), Student(roll)
    {
        thesis = th;
    }
    void printMark()
    {
        cout << thesis << endl;
    }
};
```

This will be called first

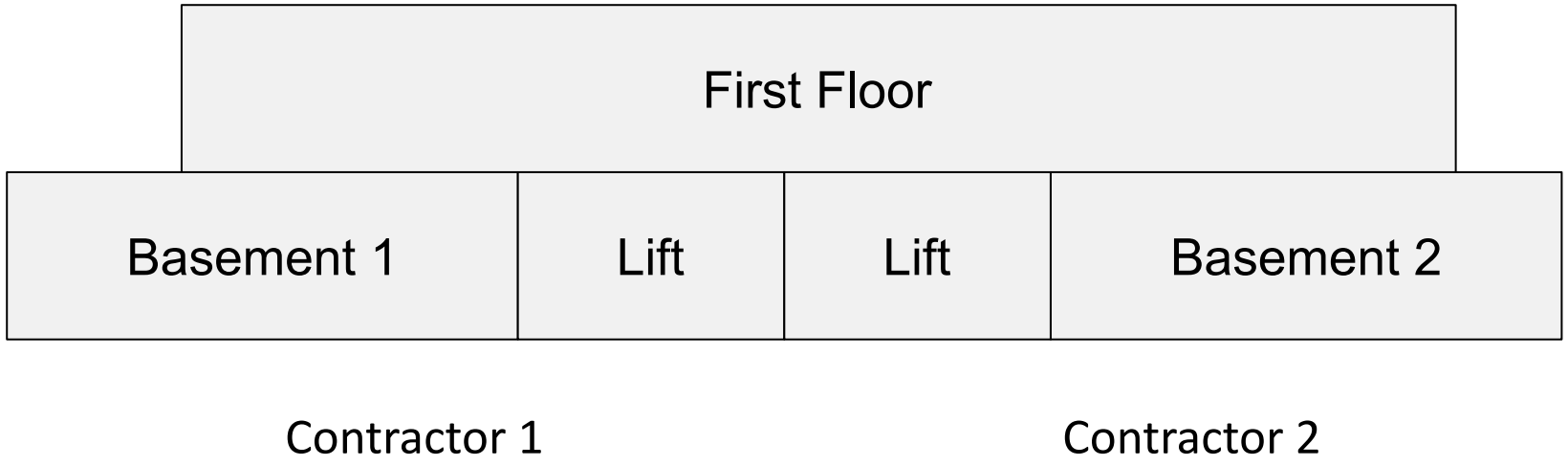
This will be called after first

This will be called at last

Now, Let’s consider this scenario in Multiple Inheritance

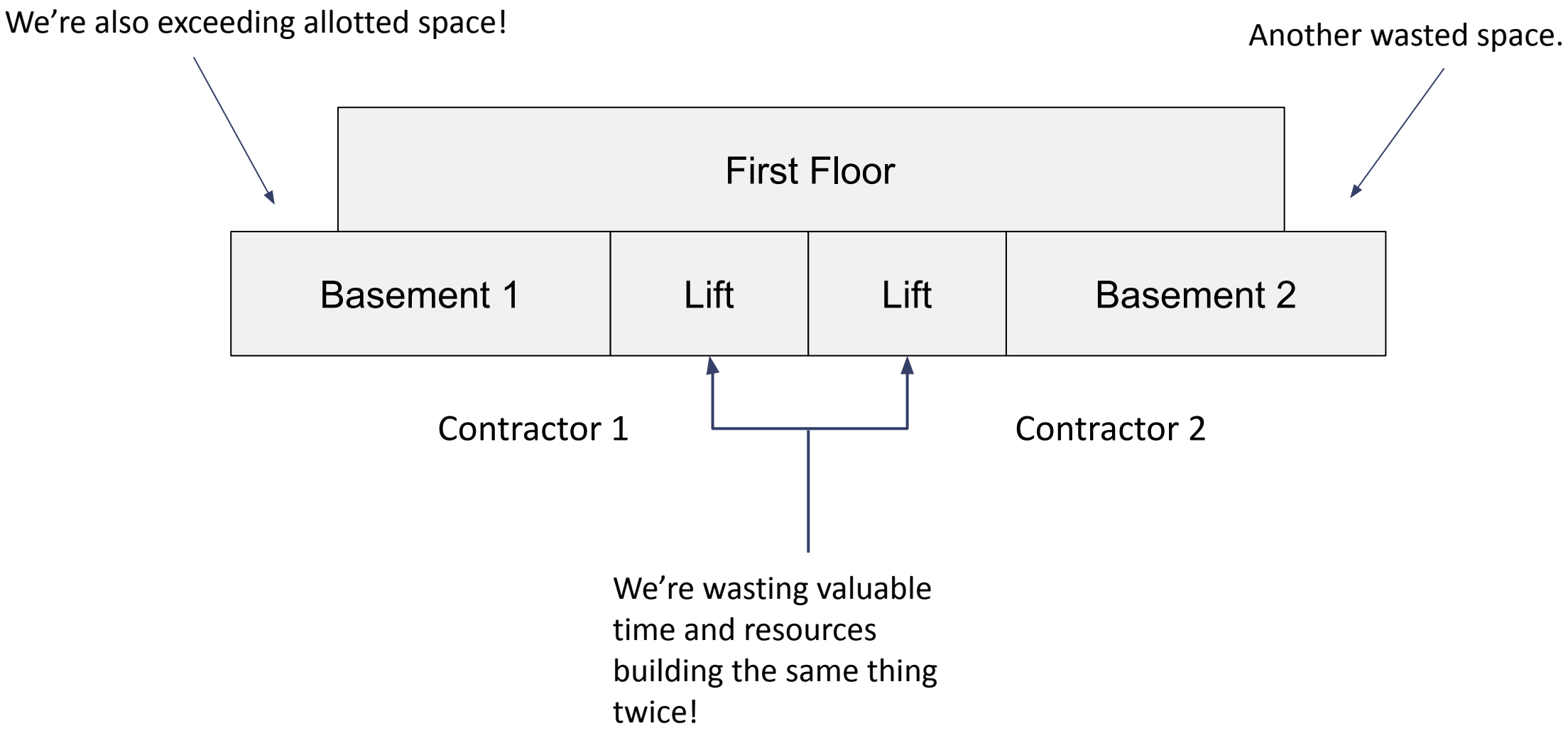
Hierarchical Inheritance/Multiple Inheritance - What if?

Derived Class with multiple base classes with common attributes in Base classes



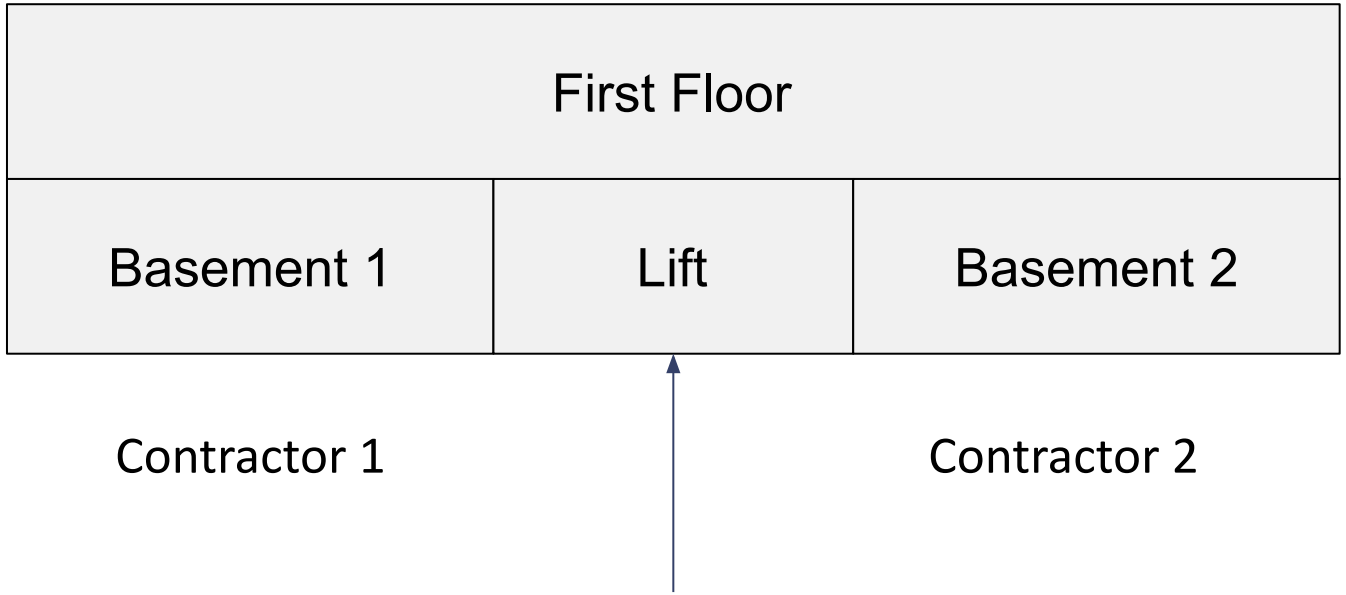
Hierarchical Inheritance/Multiple Inheritance - What if?

What are the obvious drawbacks of this scenario?



Hierarchical Inheritance/Multiple Inheritance - **Solution**

Derived Class with multiple base classes with common attributes in Base classes



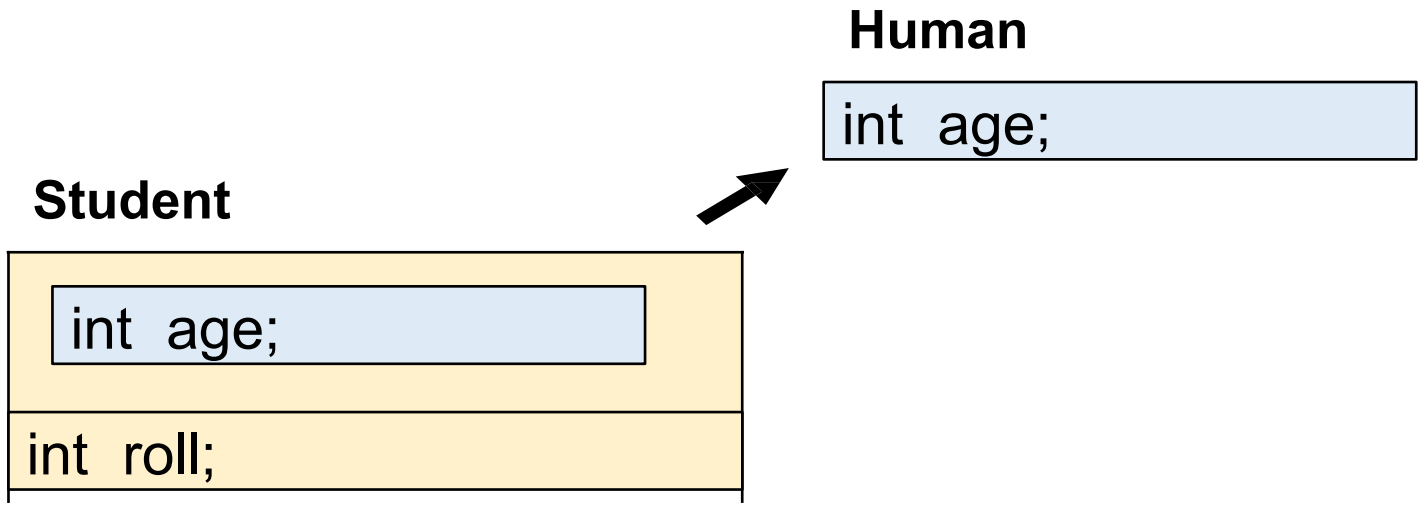
Who builds the common part then?
We'll soon see.

The Diamond Problem

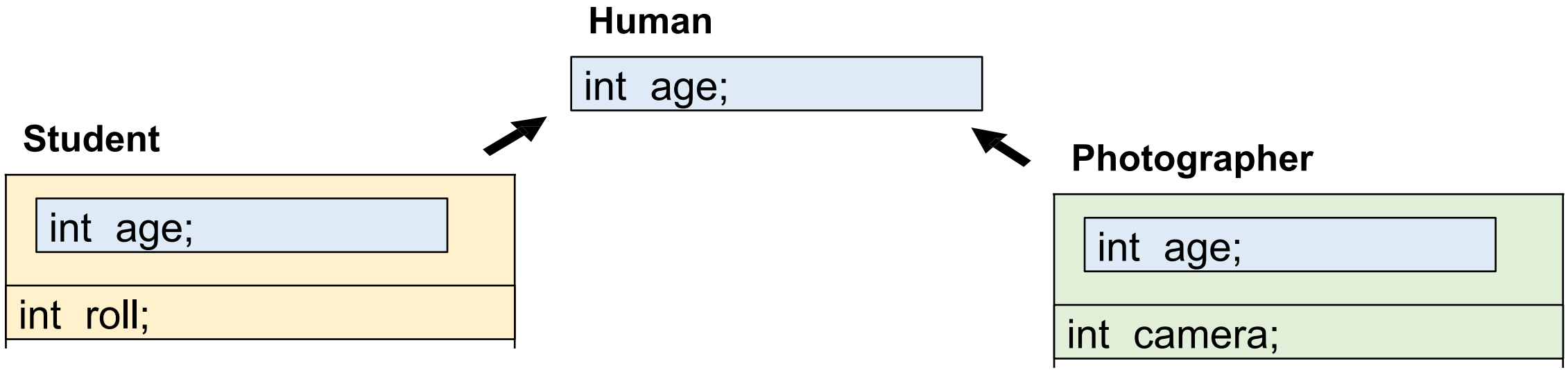
Human

```
int  age;
```

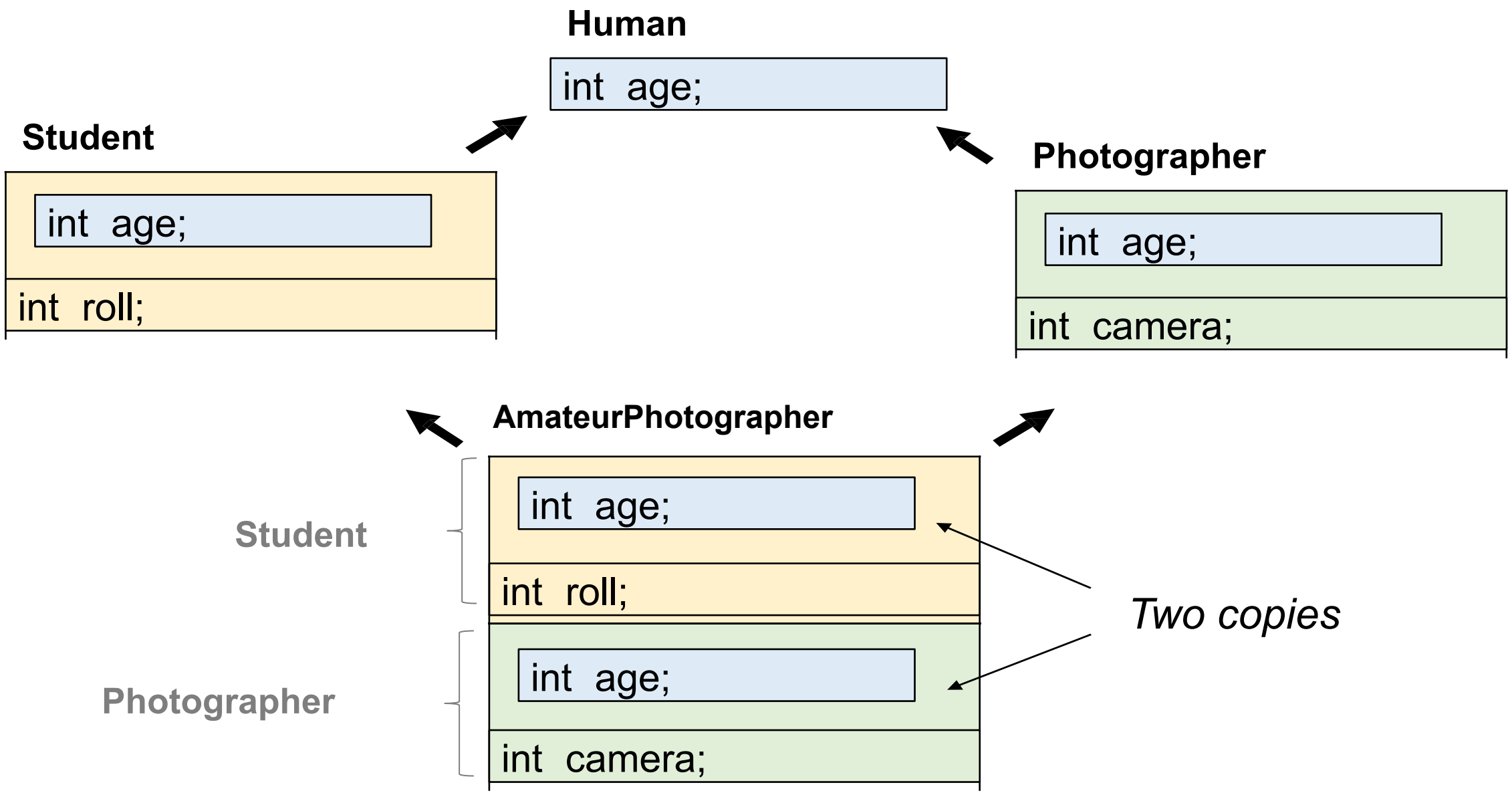
The Diamond Problem



The Diamond Problem



The Diamond Problem



The Diamond Problem

The diamond problem is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C.

If there is a variable (age variable) in A that B and C have assigned different values, then which version of the variable does D inherit?

that of B, or that of C?

If there is a method in A that B and C have overridden, and D does not override it, then which version of the method does D inherit: that of B, or that of C?

Source:https://en.wikipedia.org/wiki/Multiple_inheritance#The_diamond_problem

The Diamond Problem

```
class Human
{
public:
    int age;
};

class Student : public Human
{
public:
    int roll;
};

class Photographer : public Human
{
public:
    int camera;
};
```

The Diamond Problem

```
class AmaturePhotographer : public Student, public Photographer
{
public:
    void printDetails()
    {
        cout << "Age is: " << age <<endl;
        cout << "Roll is: " << roll <<endl;
        cout << "Number of camera: " << camera << endl;
    }

};
```

Which age is used?

The Diamond Problem

```
class AmaturePhotographer : public Student, public Photographer
{
public:
    void printDetails()
    {
        cout << "Age is: " << age <<endl;
        cout << "Roll is: " << roll <<endl;
        cout << "Number of camera: " << camera << endl;
    }

};
```

Which age is used?

```
int main()
{
    AmaturePhotographer person1;
    person1.age = 20;
    person1.roll = 50;
    person1.camera = 2;
    person1.printDetails();

}
```

Which age is used?

The Diamond Problem

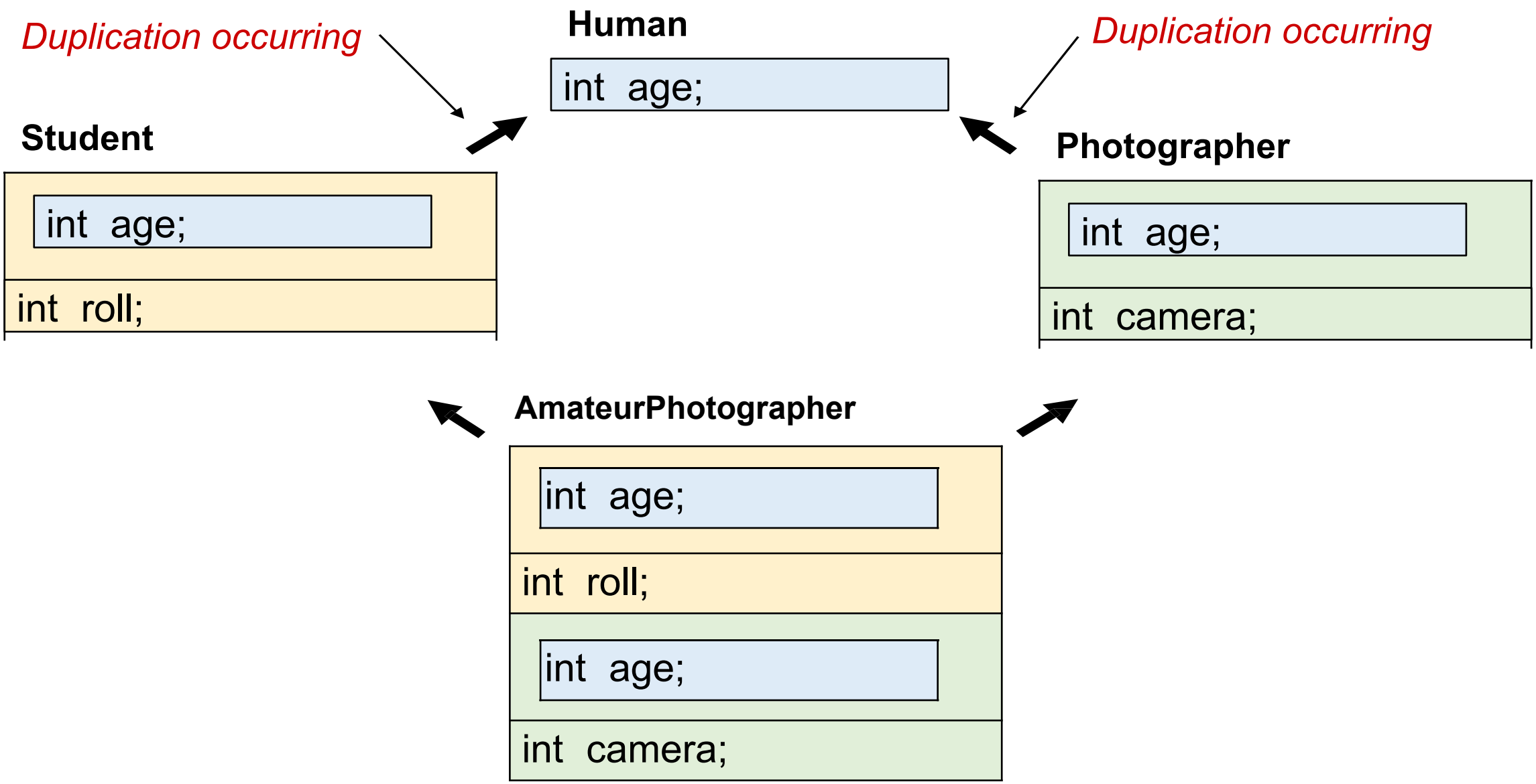
We can always use scope resolution operator

```
class AmaturePhotographer : public Student, public Photographer
{
public:
    void printDetails()
    {
        cout << "Age is: " << Student::age <<endl;
        cout << "Roll is: " << roll <<endl;
        cout << "Number of camera: " << camera << endl;
    }

};
```

```
int main()
{
    AmaturePhotographer person1;
    person1.Photographer::age = 20;
    person1.roll = 50;
    person1.camera = 2;
    person1.printDetails();

}
```

What if wrong scope is used?

We'll have to prevent duplication



Virtual Inheritance

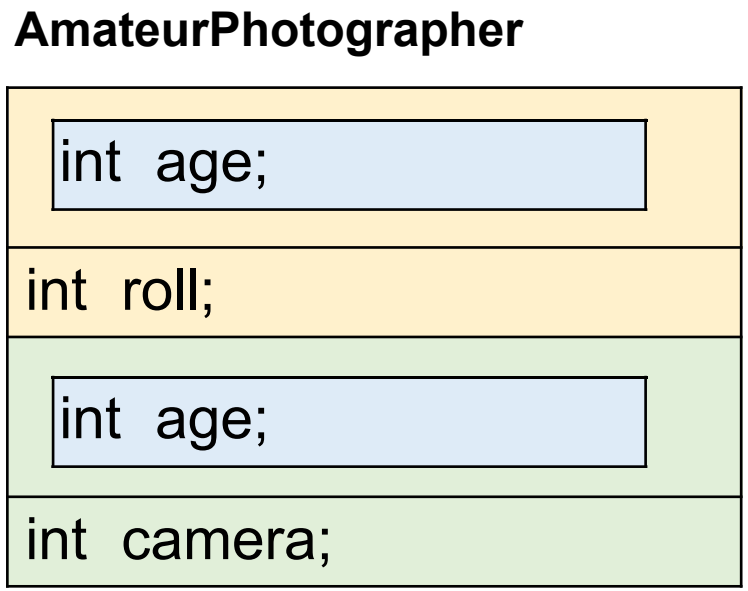
```
class Student : virtual public Human
{
public:
    int roll;
};

class Photographer : virtual public Human
{
public:
    int camera;
};
```

This is to ensure that always a single copy is created

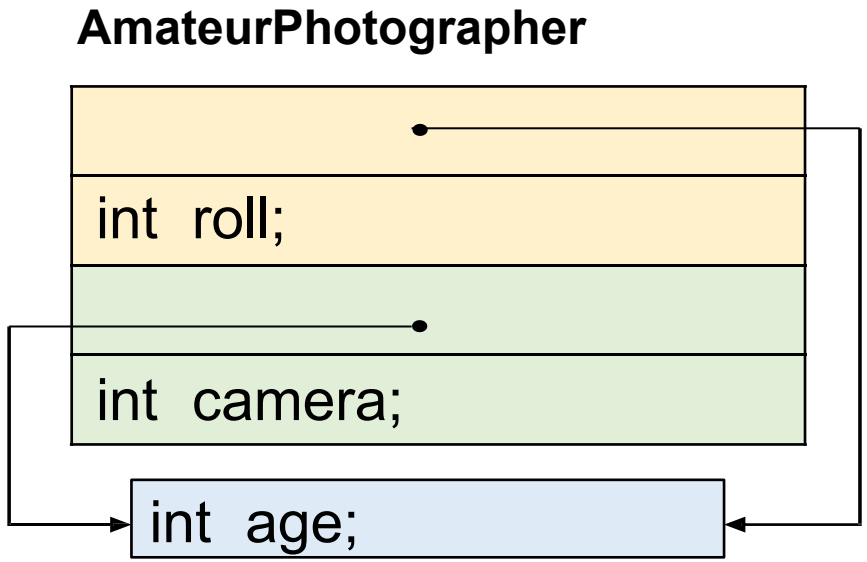
Virtual Inheritance

```
class Student : virtual public Human
{
public:
    int roll;
};

class Photographer : virtual public Human
{
public:
    int camera;
};
```

This is now **virtual base class**



Normal Inheritance



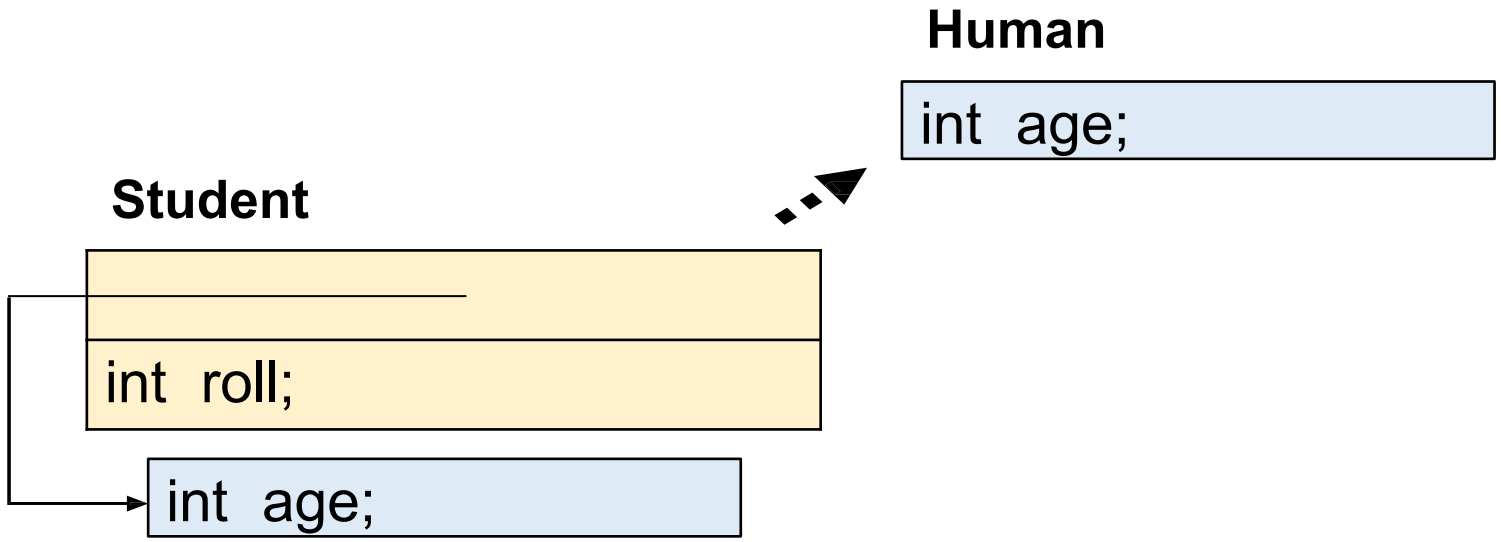
Virtual Inheritance

Virtual Inheritance

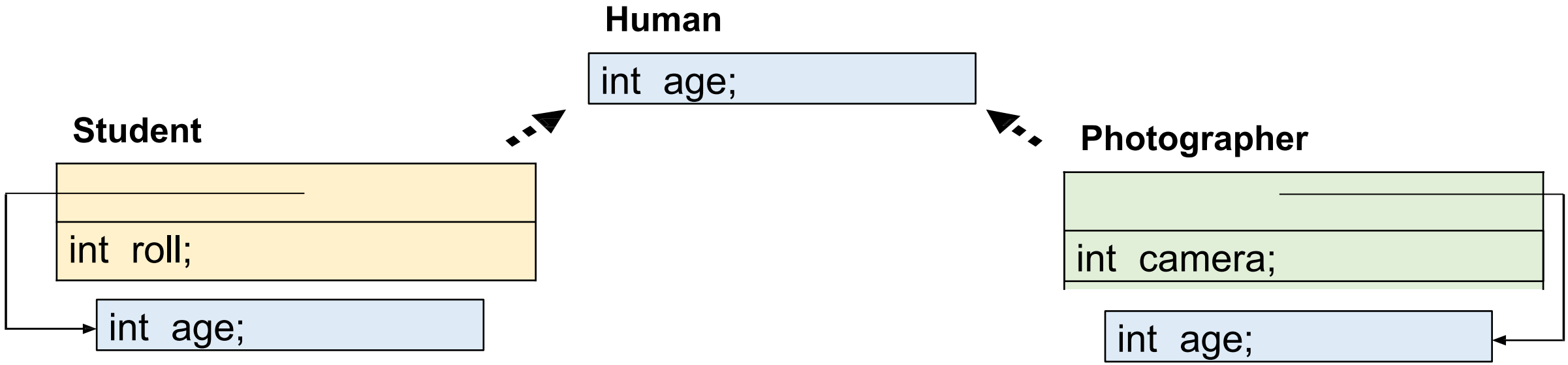
Human

```
int  age;
```

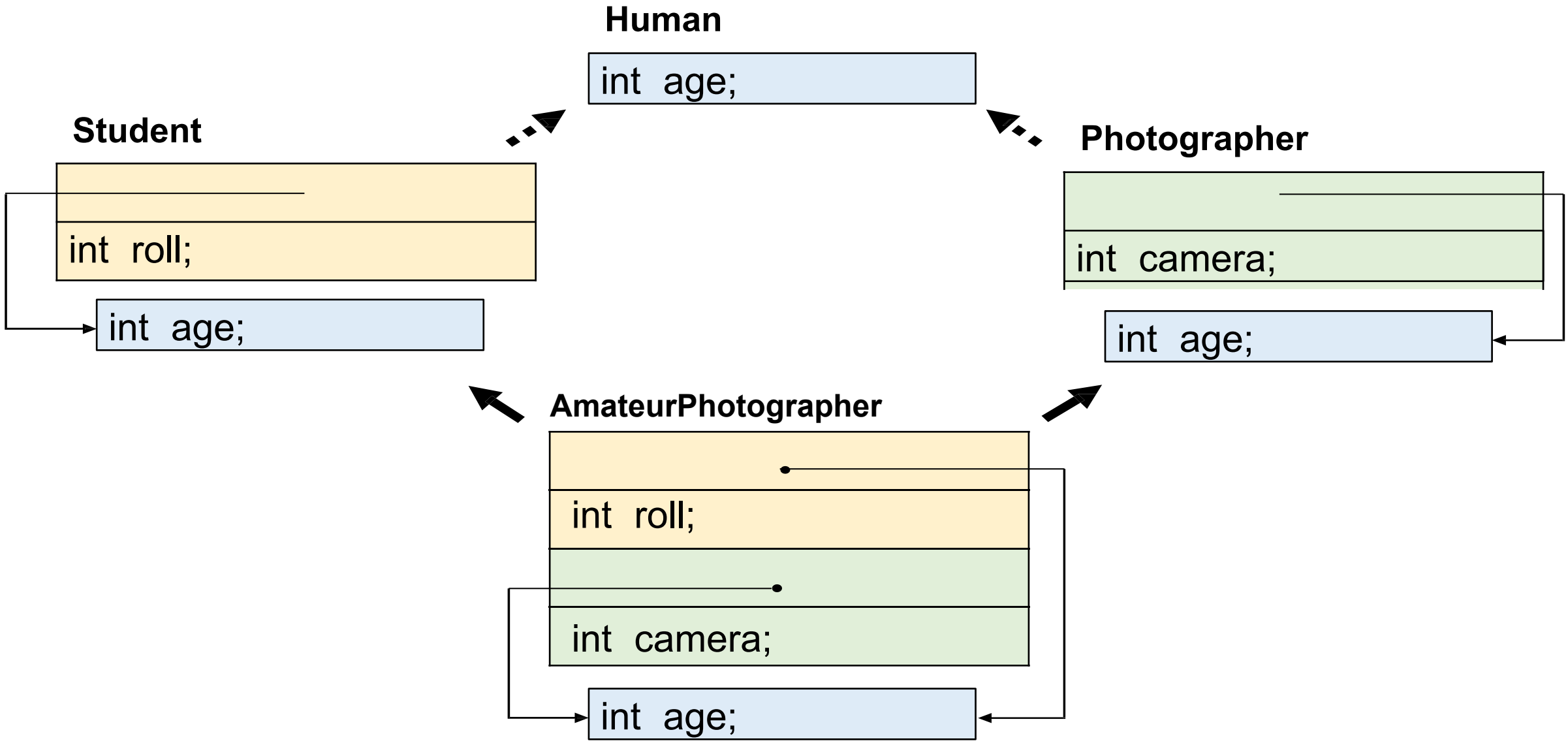
Virtual Inheritance



Virtual Inheritance



Virtual Inheritance



Constructor in Virtual Inheritance

```
class Human
{
public:
    int age;
    Human(int a)
    {
        cout << "Human is created" <<endl;
        age = a;
    }

};
```

Constructor in Virtual Inheritance

```
class Photographer : virtual public Human
{
public:
    int camera;
    Photographer(int a, int c): Human(a)
    {
        cout << "Photographer is created" <<endl;
        camera = c;
    }

};
```

Constructor in Virtual Inheritance

```
class AmateurPhotographer : public Student, public Photographer
{
public:
    AmateurPhotographer(int a, int r, int c) :  Student(a, r),
                                                Photographer(a, c),
                                                Human(a)
    {
        cout << "AmateurPhotographer is created" <<endl;
    }
};

int main()
{
    //Student s1;
    //Photographer p1;
    AmateurPhotographer ap1(1, 2, 3);
}
```

Virtual Inheritance - Syntax

So, we need to do two things while implementing virtual inheritance to overcome Diamond Problem:

1. Inherit the common base class (Human) as virtual in the first level (while writing the class definition for Student and Photographer).

2. While writing the constructor for the **most derived** class (the class at the bottom of the inheritance sequence, i.e. AmateurPhotographer), pass variables to the virtual base class (i.e. Human) as if it was a direct base class of AmateurPhotographer.

Constructor in Virtual Inheritance

A few things to note

One, Virtual base classes are always created **<u>before</u>** non-virtual base classes

```
class AmateurPhotographer : public Student, public Photographer
{
```

Constructor in Virtual Inheritance

A few things to note

One, Virtual base classes are always created **<u>before</u>** non-virtual base classes

```
class AmateurPhotographer : public Student, public Photographer
{
```

Ctor Seq: Human > Student > Photographer

Constructor in Virtual Inheritance

Even if we had a standalone unrelated class (for example, a new class “Employee”) to the Virtual Base class (Human), the virtual base class will be called first.

Then, the rest of the classes will be called as per the inheritance sequence.

```
class Employee
{
public:
    Employee()
    {
        cout <<"Employee created"<<endl;
    }
};

class AmateurPhotographer : public Employee, public Student,
                            public Photographer
{
public:
    AmateurPhotographer(int a, int r, int c) :  Student(a, r),
                                                Photographer(a, c),
                                                Human(a)
```

Constructor in Virtual Inheritance

A few things to note

One, Virtual base classes are always created **<u>before</u>** non-virtual base classes

```
class Employee
{
public:
    Employee()
    {
        cout <<"Employee created"<<endl;
    }
};

class AmateurPhotographer : public Employee, public Student,
                            public Photographer
{
public:
    AmateurPhotographer(int a, int r, int c) :  Student(a, r),
                                                Photographer(a, c),
                                                Human(a)
```

Ctor Seq: Human > **Employee** > Student > Photographer

Constructor in Virtual Inheritance

A few things to note

Two, the *most* derived class is responsible for constructing the virtual base class

```
int roll;
Student(int a, int r) : Human(a)
{
    cout << "Student is created" <<endl;

int camera;
Photographer(int a, int c) : Human(a)

public:
    AmateurPhotographer(int a, int r, int c) :  Student(a, r),
                                                Photographer(a, c),
                                                Human(a)

    //Photographer p1;
    AmateurPhotographer ap1(1, 2, 3);
}
```

This constructor is called

Constructor in Virtual Inheritance

A few things to note

Two, the *most* derived class is responsible for constructing the virtual base class

```
int roll;
Student(int a, int r) : Human(a)
{
    cout << "Student is created" <<endl;

int camera;
Photographer(int a, int c) : Human(a)

public:
    AmateurPhotographer(int a, int r, int c) :  Student(a, r),
                                                Photographer(a, c),
                                                Human(a)
{

    AmateurPhotographer ap1(1, 2, 3);
}
```

These calls are ignored

Constructor in Virtual Inheritance

A few things to note

Three, a virtual base class is always considered a direct base of its most derived class.

This is why the most derived class is responsible for its construction.

```
public:
    AmateurPhotographer(int a, int r, int c) :  Student(a, r),
                                                Photographer(a, c),
                                                Human(a)
```

References

- https://docs.microsoft.com/en-us/cpp/cpp/multiple-base-classes
- http://www.learncpp.com/cpp-tutorial/128-virtual-base-classes/

Thank you