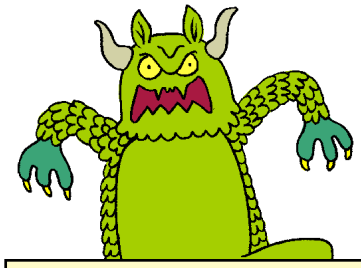
MULTIPLE INHERITANCE

Overrided Functions

```
class Car
{
    public:
        void vroom()
        {
            cout << "Car::vroom\n";
        }
};

class Geo : public Car
{
  public:
      void vroom()
        {
            cout << "Geo::vroom\n";
        }
};
```

- So far we've been saying that overrided functions "hide" their base class versions
- What would this code fragment output?

```
Geo prizm;
prizm.vroom();
```



Overrided Functions

```
class Car
{
    public:
        void vroom()
        {
            cout << "Car::vroom\n";
        }
};

class Geo : public Car
{
  public:
      void vroom()
        {
            cout << "Geo::vroom\n";
            base::stuff();
        }
};
```

- "Hidden" doesn't mean "gone", though!
- Sometimes you might want to call the base class version of a function...
- You can do that using the scope resolution operator (::)

What does this print now?

```
Geo prizm;
prizm.vroom();
```

Some Weird Syntax...

```
class Car
{
    public:
        void vroom()
        {
            cout << "Car::vroom\n";
        }
};

class Geo : public Car
{
  public:
      void vroom()
       {
           cout << "Geo::vroom\n";
       }
};
```

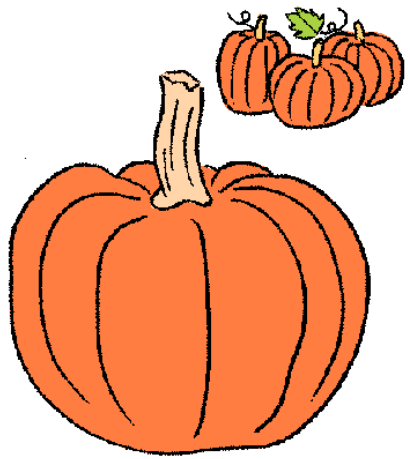
- You can even do this from *outside* a class
- Say you want to call the base class version of **vroom**() from the main function:

```
int main()
{
  Geo prizm;
  prizm.base::vroom();
}
```

```
void vroom()
{
    cout << "Global Vroom!!\n";
}

class Car
{
    public:
        void vroom()
        {
            cout << "Car::vroom\n";
        }
};

class Geo : public Car
{
  public:
      void vroom()
        {
            cout << "Geo::vroom\n";
            Global vroom()?
        }
};
```

Question

- What if we add *another* vroom() function - a global one?
- Could we call that from Geo::vroom()?

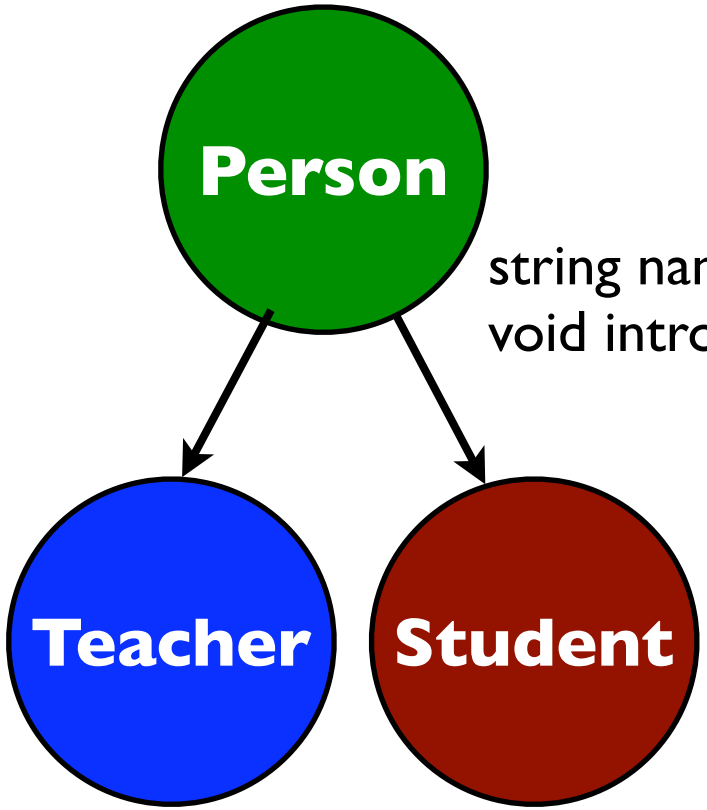


```
void vroom()
{
    cout << "Global Vroom!!\n";
}

class Car
{
    public:
        void vroom()
        {
            cout << "Car::vroom\n";
        }
};

class Geo : public Car
{
  public:
      void vroom()
        {
            cout << "Geo::vroom\n";
            ::vroom();
        }
};
```

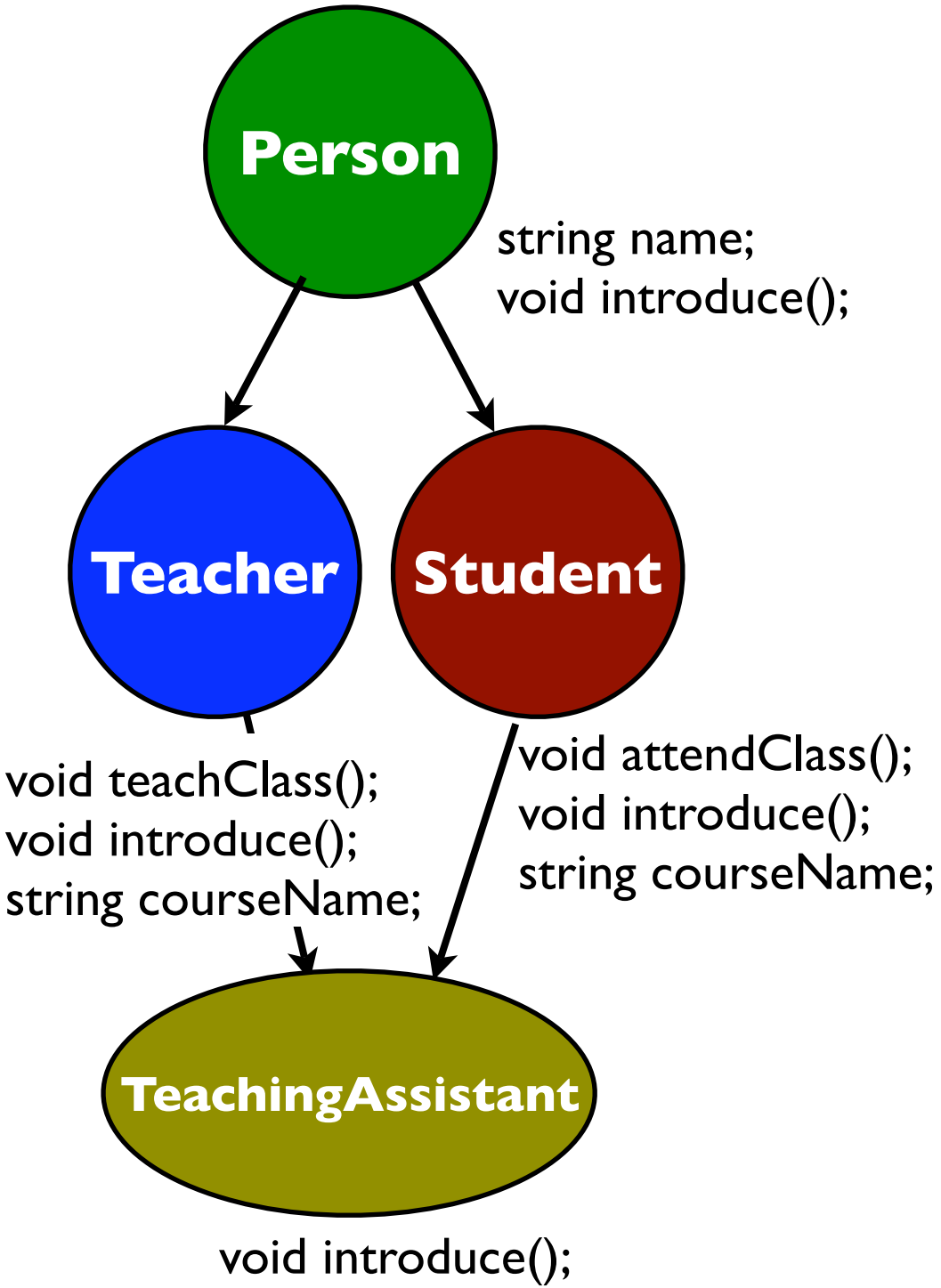
Question

- When used on its own, :: means "access the global scope, not the local scope"
- So, to call the global vroom() function, we use the :: operator to call the containing scope

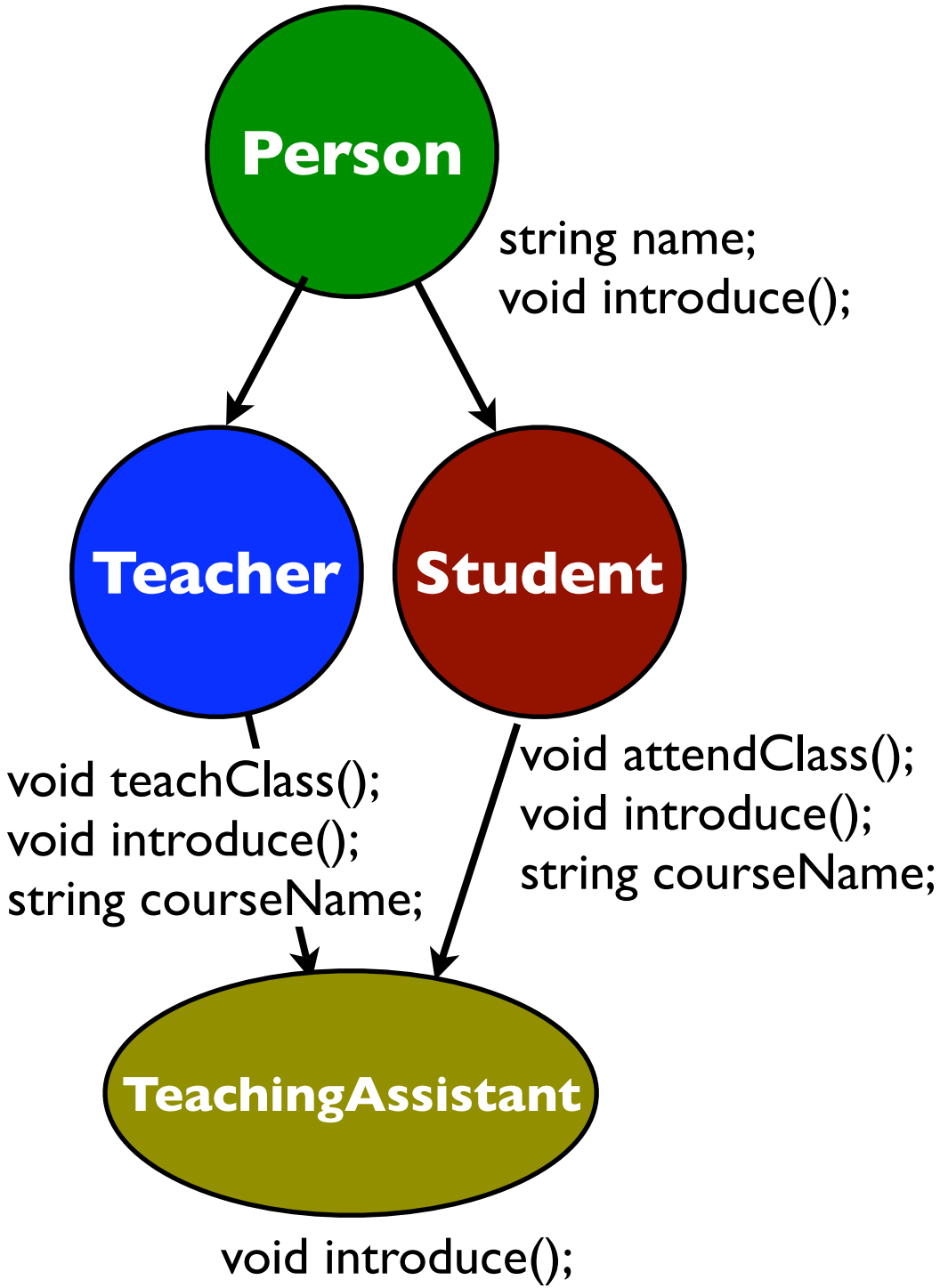
Multiple Inheritance



- Sometimes inheriting from a single class isn't enough!
- Say we've got the simple class hierarchy to the left:
- What do we do when we want to define a **TeachingAssistant** class?
  - A TeachingAssistant both teaches *and* attends classes
  - No one base class is enough!



- We have to make **TeachingAssistant** inherit from *both* Teacher and Student!
- So: our new TA class will inherit *all* the stuff from both base classes!
- How would we write an introduce method that explains what course the TA teaches, *and* what course he/she studies?



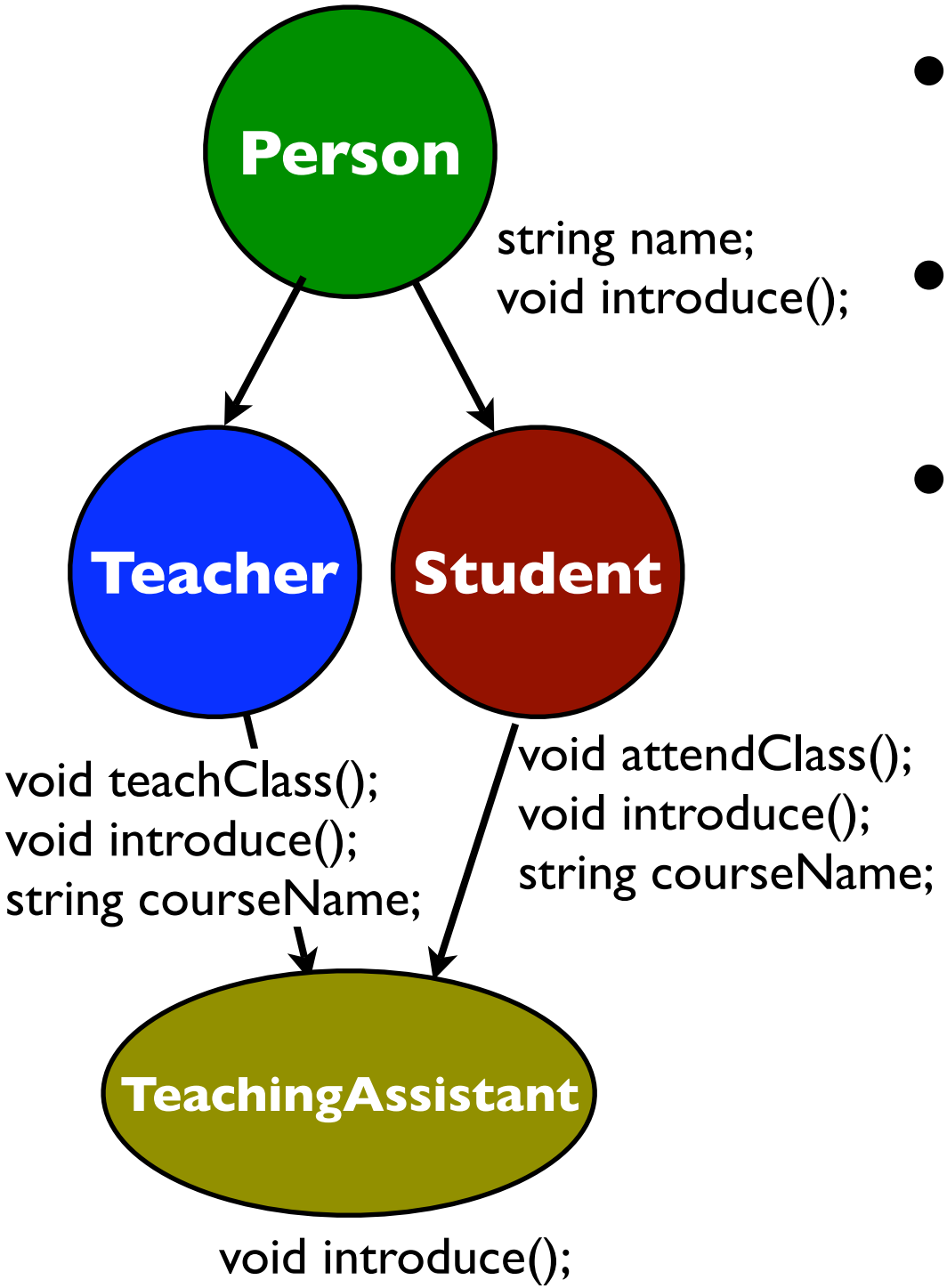
- How many courseName variables are there in TeachingAssistant?
- How do we print out the right version at the right time?

```
void TA::introduce()
{
    cout << "I teach: ";
    cout <<  (?)
    cout << "I study: ";
    cout <<  (?)
}
```

Multiple Inheritance

```
class Teacher : public Person
{   // declaration mostly omitted
  public:
    Teacher( string name );
};

class Student : public Person
{   // declaration mostly omitted
 public:
    Student( string name );
};

class TA :
      public Teacher, public Student
{
  public:
    TA() :
      Student(name), Teacher(name)
    {}
};
```

- Doing this is pretty simple:
- Just add to the list of classes your class inherits from
- You may need to add to the constructor init list too!



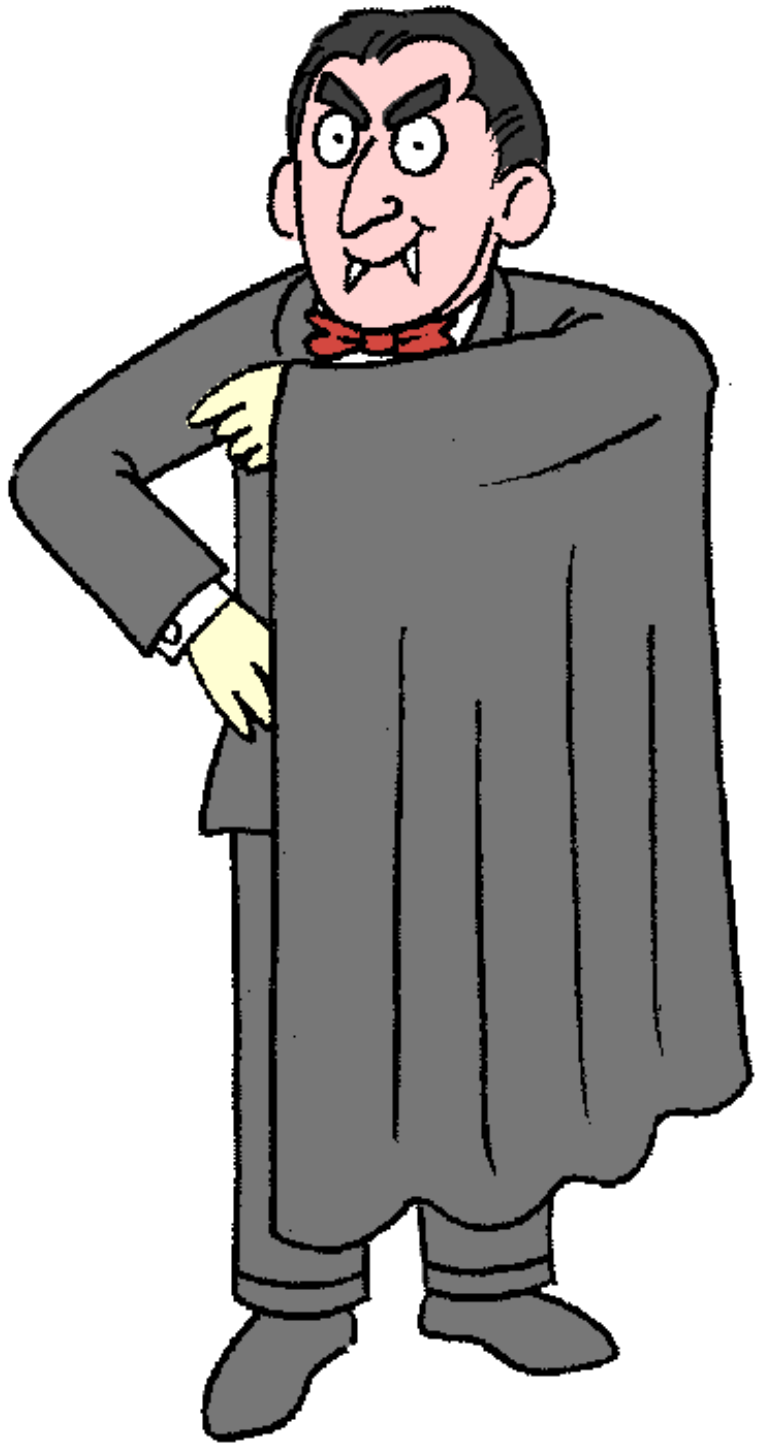
- One problem you may have noticed:
- How many copies of **name** does TeachingAssistant have?
- Which one do we use? Does it matter?

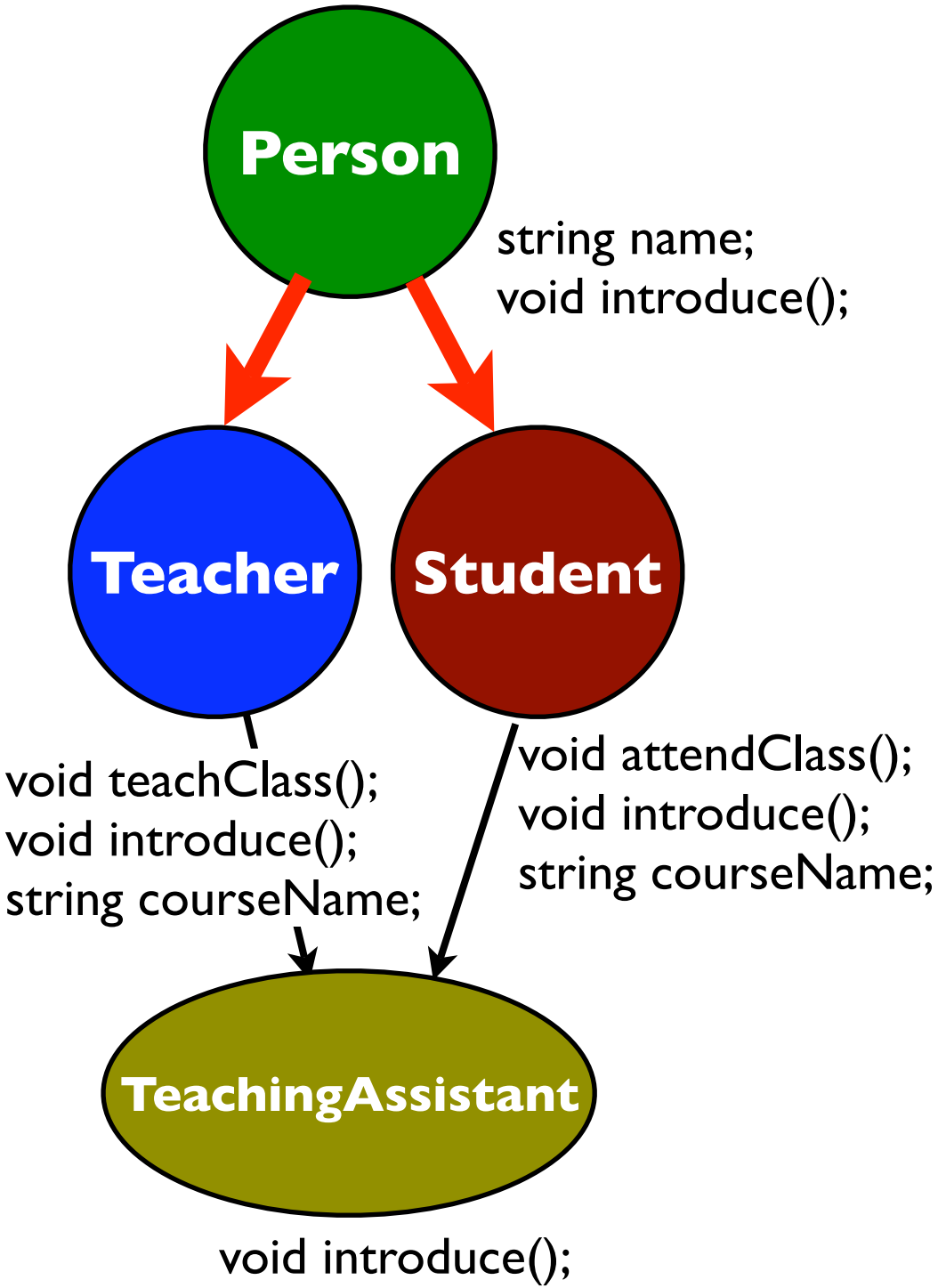
```
void TA::introduce()
{
    cout << "My name is:";
    cout <<  (?)
    cout << "I teach: ";
    cout <<  (?)
    cout << "I study: ";
    cout <<  (?)
}
```

- **TeachingAssistant** is derived from both **Student** and **Teacher**
- Both Student and Teacher inherited a **name** attribute from **Person**
- Therefore, TeachingAssistant has **two** copies of **name**!
- This might be OK but it might not: could each copy of name have a different value?

Virtual Inheritance

- The way to solve this: **virtual inheritance**
- If you inherit "virtually" from a base class, you tell the compiler:
  - there must be one instance of that base class if someone inherits from the current class
- This is weird, and ugly, but it solves the problem neatly





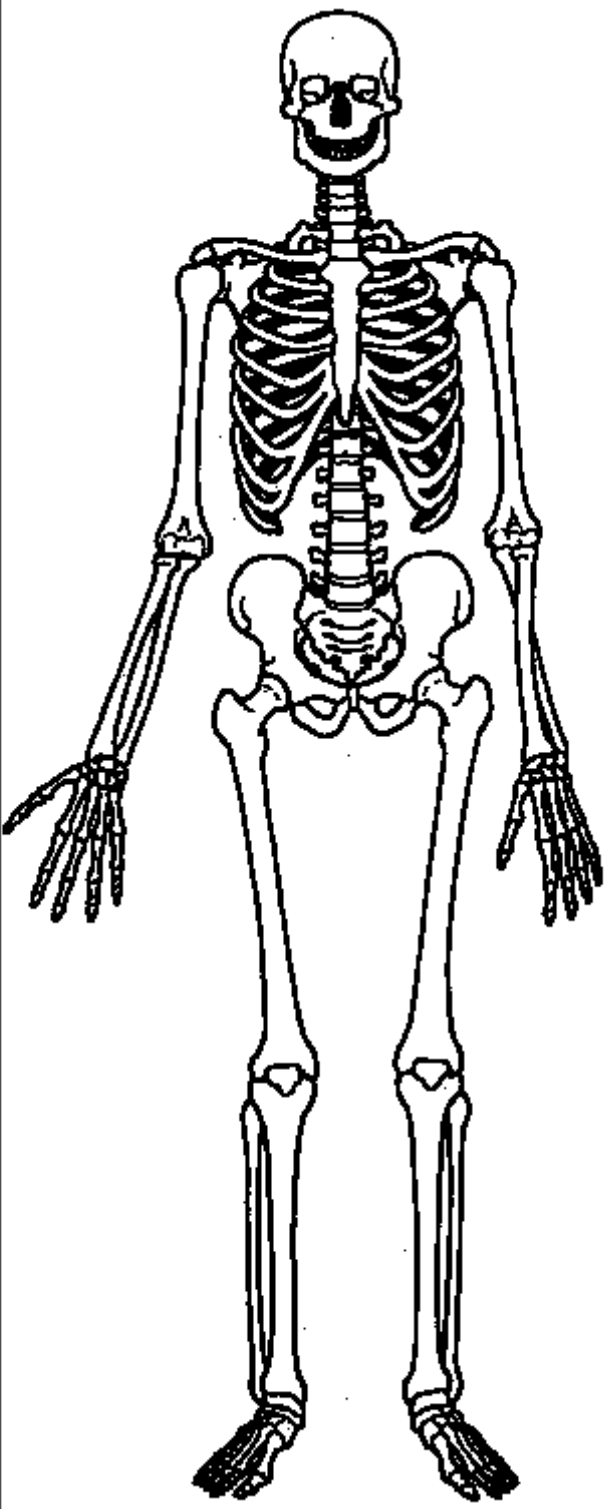
how this works:

- Before we had **two** copies of name in TeachingAssistant
- Now, Teacher and Student are inheriting *virtually* from Person (red arrows)
- So there will be only *one* copy of Person in any class inherited from Teacher and Student
- ... aka TeachingAssistant, only has a single copy of Person - (therefore, name)

```
// declarations mostly omitted...
class Person
{
    string name;
};

class Teacher : virtual public Person
{
  public:
    Teacher( string name );
};

class Student : virtual public Person
{
 public:
    Student( string name );
};

class TA :
      public Teacher, public Student
{
  public:
    TA() :
      Student(name), Teacher(name)
    {}
};
```

Virtual Inheritance

- To inherit virtually, just stick the keyword **virtual** right before the **public**
- This has nothing to do with virtual functions!
- Why do *both* Student and Teacher use virtual inheritance? Is this necessary?



Multiple Inheritance

- Many people disagree on the usefulness of Multiple Inheritance
  - Most newer languages don't support MI at all, or only a small subset of it
- If you find yourself needing to use MI a lot, consider redesigning your classes so you don't!
- Not used nearly as widely as regular inheritance