```
class Person{

Date *dob;

public:

void setDob(Date dob){
this->dob=dob;
}

Date getDob(){
return dob;
}

}
```

```
class Date{

int day;
int month;
int year;

public:
void setDay(int day);
int getDay();
}
```

```
main(){
Person p;
Date d;
p.setDob(d);

}
```

accept()

name

marks

gender

email

batch

rolllno

```
display(){
name
marks
gender
email
batch
rolllno
}
```

rollno, name

rollno -> marks

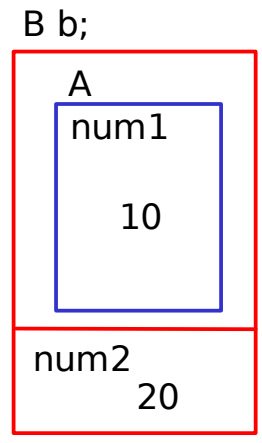
is-a relationship

inheritance

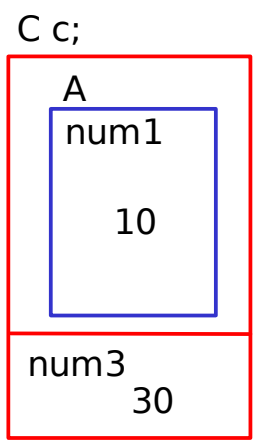
1. Single
2. Multilevel
3. Multiple
4. Hirerachical
5. Hybid Inheritance

```
class Parent // Base
{

}

class Child :public Parent // Derived
{

}
```

```
class A{
int num1 = 10;
}

class B:public A{
int num2 = 20;
}

class C : public A{
int num3 = 30;
}

class D : public B, public C{
int num4 = 40;
}
```

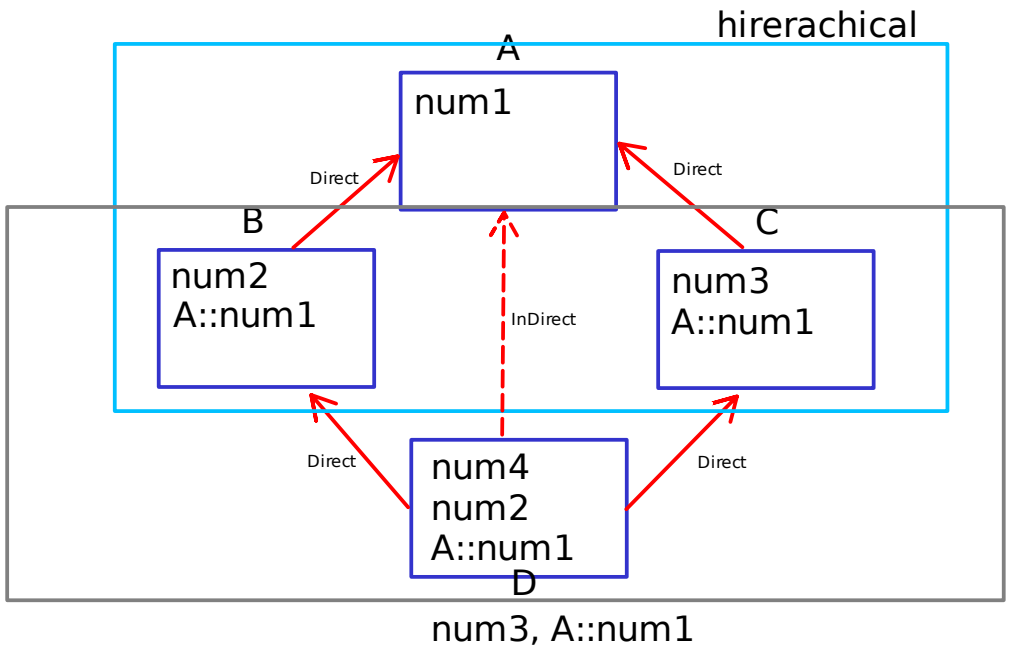


data member of B class
+
object of A class

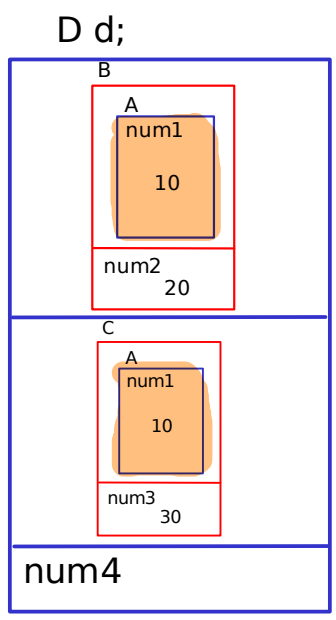


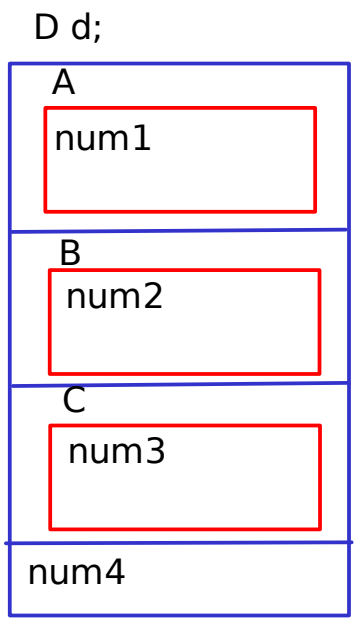
data member of C class
+
Object of A class

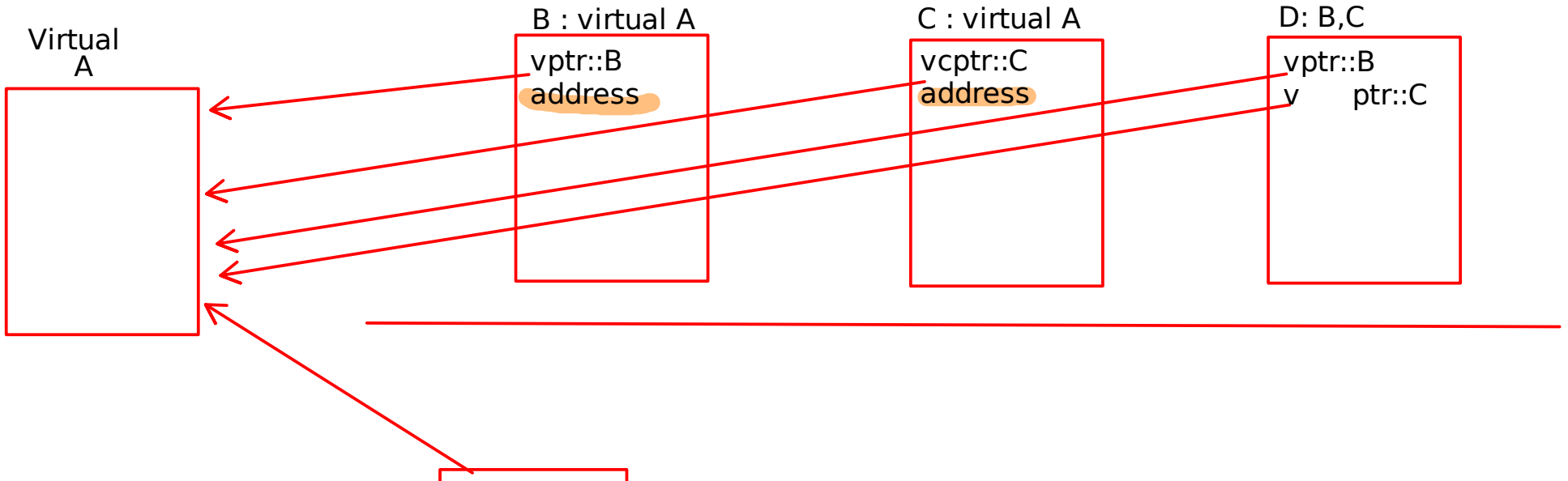
hirerachical

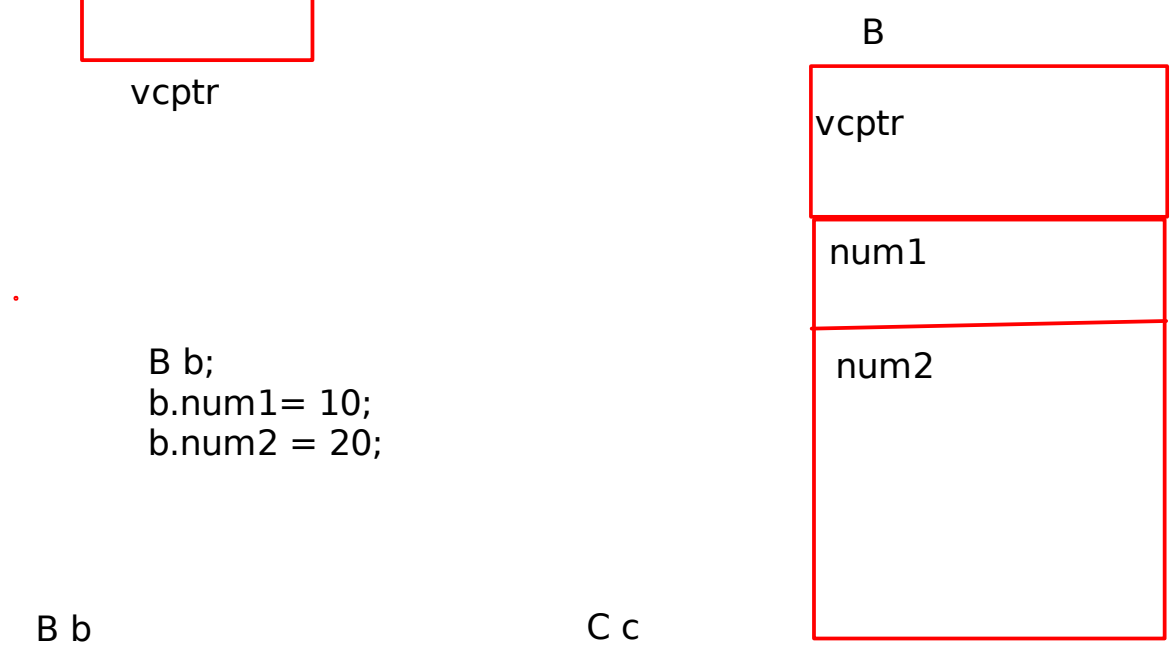


multiple





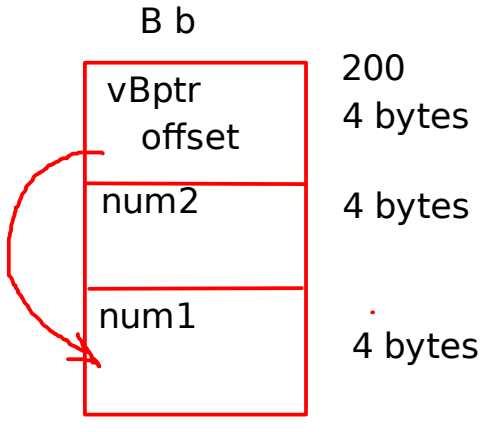


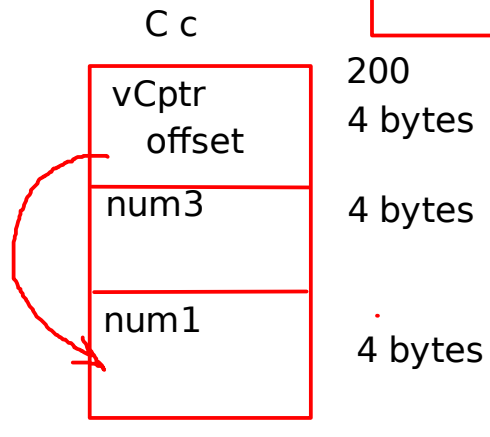


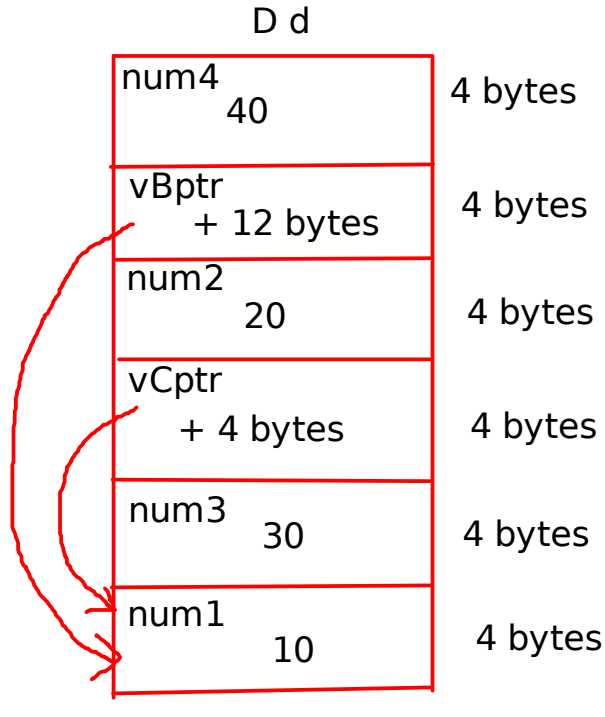

```
class Employee{
Date doj;
Car *cptr;
```

```
B b;
b.num1= 10;
b.num2 = 20;
```

```
A a;
*vptr
}
```







Upcasting
C *c = new D();

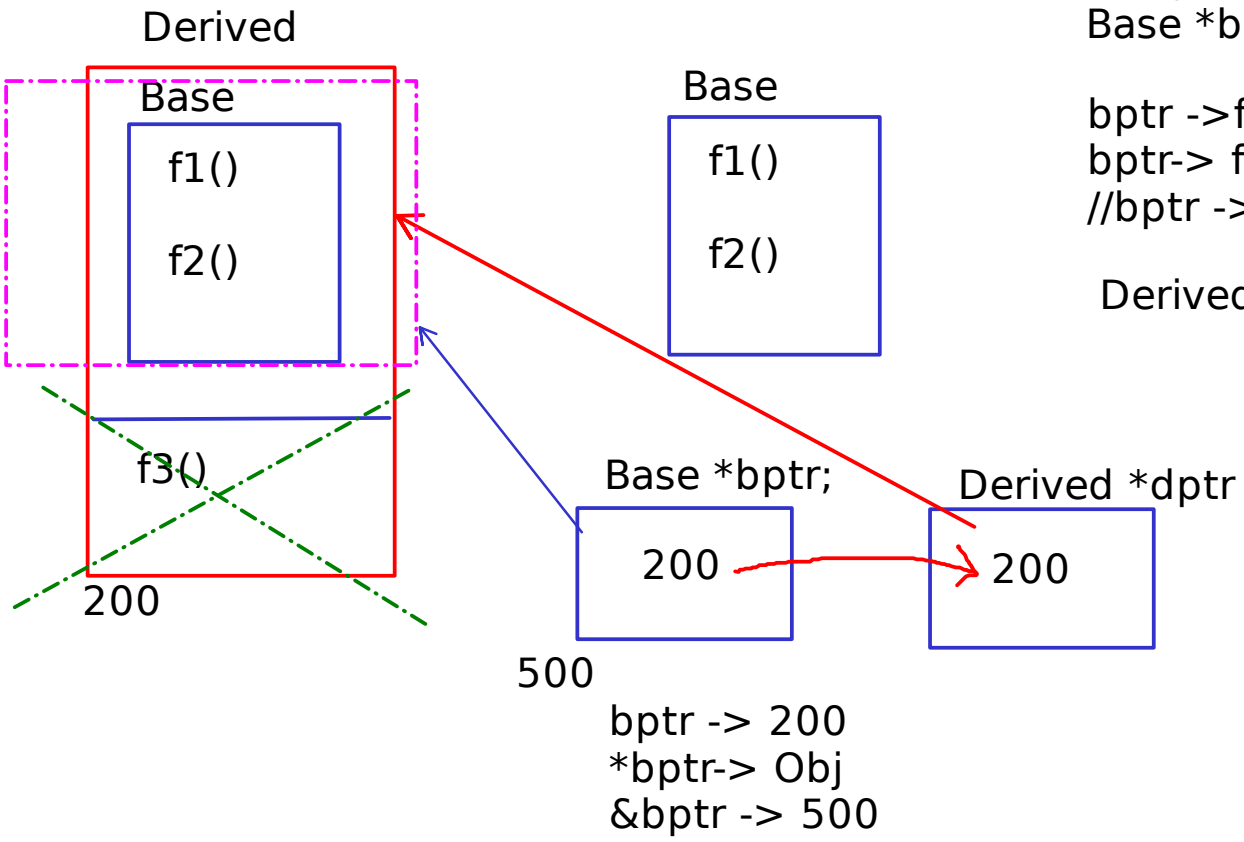
B *b = new D();

Virtual Base class is
managed by Compiler

```
class Base{

void f1(){
Base::f1()
}

void f2(){
Base::f2()
}
}
```

```
class Derived{

void f1(){
Base::f1()
}

void f2(){
Base::f2()
}

void f3(){
Derived::f3()
}
}
```

```
Base *bptr =new Base();
bptr->f1();
bptr->f2();

Derived *dptr = new Derived();
dptr->f1();
dptr->f2();
dptr->f3();
```



bptr -> 200
*bptr-> Obj
&bptr -> 500

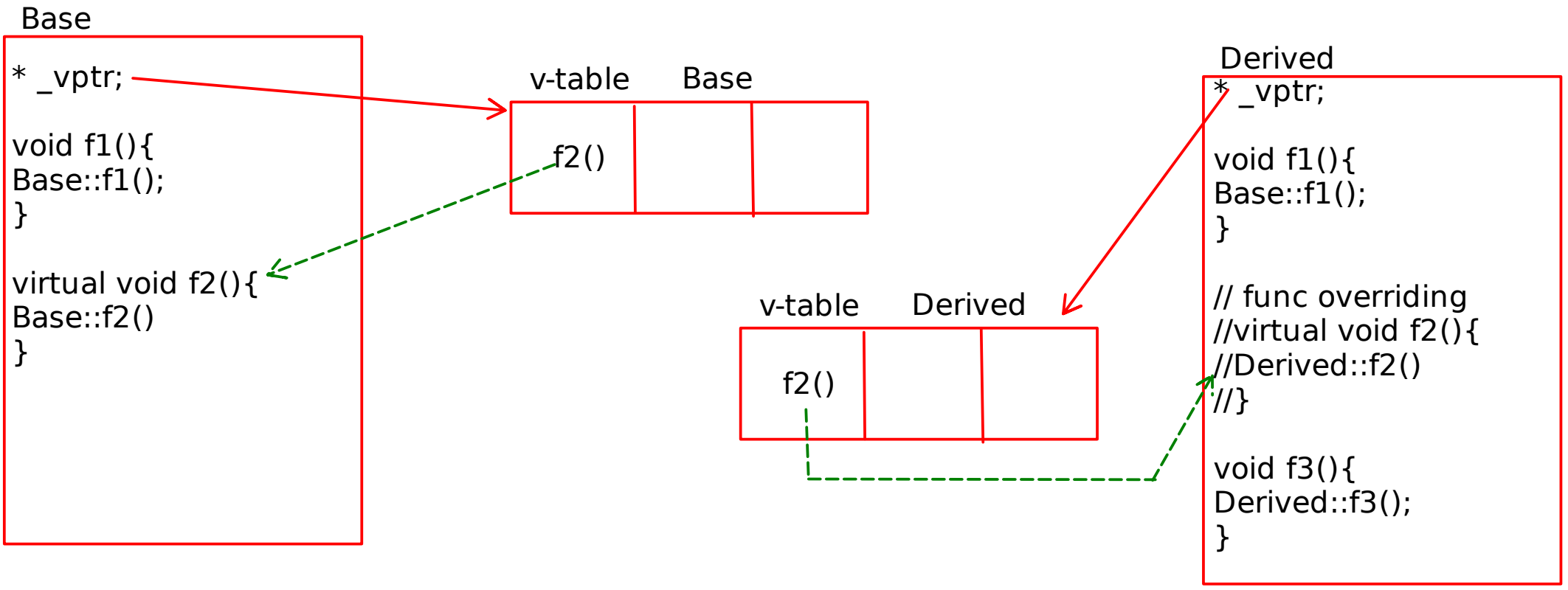
```
// object slicing
Base *bptr = new Derived();// upcasting

bptr ->f1();
bptr-> f2();
//bptr ->f3(); //NOT OK object slicing

 Derived  *dptr =(Derived *) bptr; // Downcasting

                dptr->f3();
                dptr->f1();
                dptr-> f2();
```

```
Base * b=new Derived();
```

```
class Base{

void f1(){
Base::f1()
}

void f2(){
Base::f2()
}
}
```

```
class Derived{

void f1(){
Base::f1()
}

void f2(){
Base::f2()
}

void f2(){
Derived::f2()
}

void f3(){
Derived::f3()
}
}
```

```
Base *bptr = new Derived(); // upcasting

 bptr->f1();
 bptr->f2(); // Base::f2()

Derived * dptr = (Derived *) bptr; // downcasting;

 dptr->f1();//OK
 dptr->f2(); // Derived::f2()
 dptr->f3();// OK
```




```
Base *bptr = new Base();
bptr->f1();
bptr->f2(); // Base::f2();
```

```
Base *bptr = new Derived();
bptr->f1();
bptr->f2(); // Derived::f2();
```