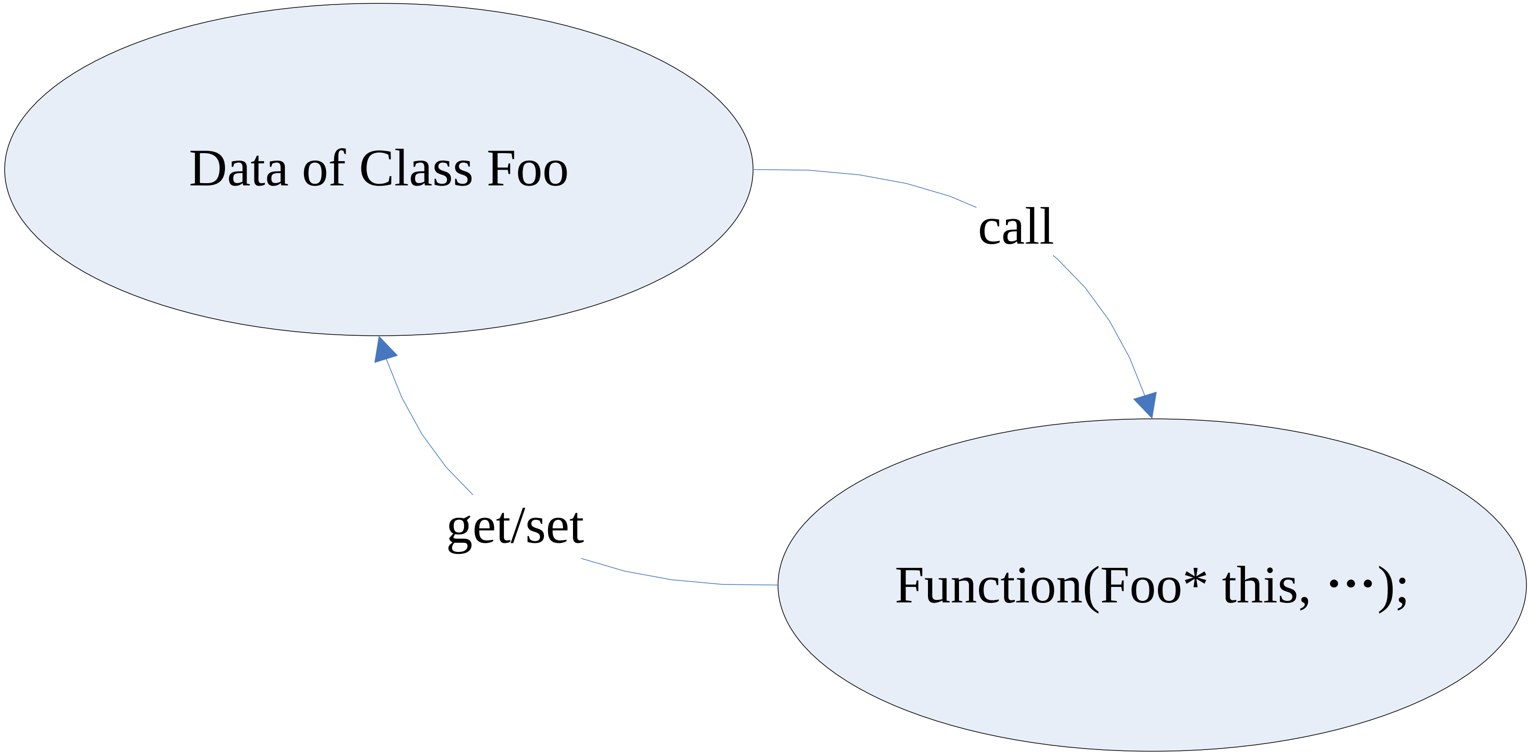
C++ Object Model

shifan@freecity.cn

Object = Data + Algorithm



Data of Class Foo

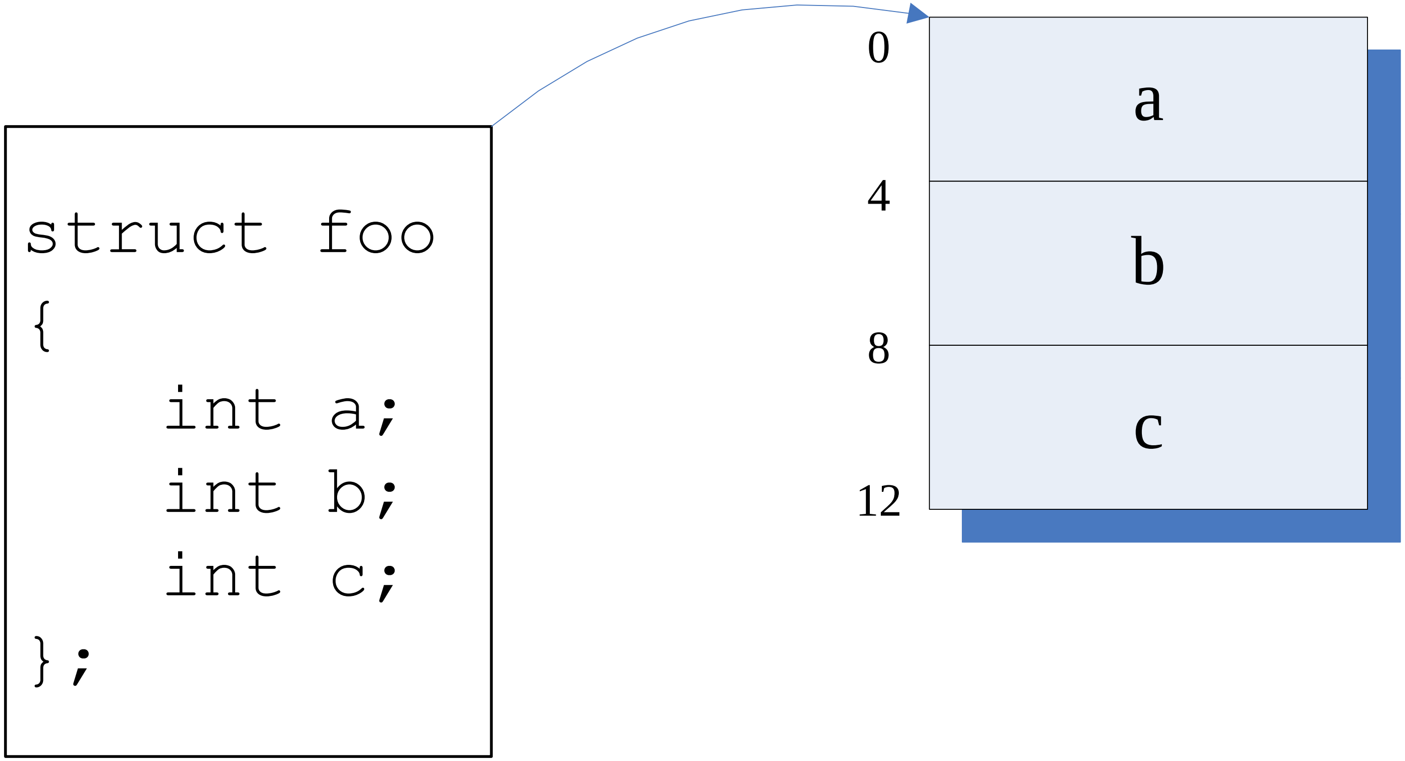
call

get/set

Function(Foo* this, …);

Data Layout

- Plain object:

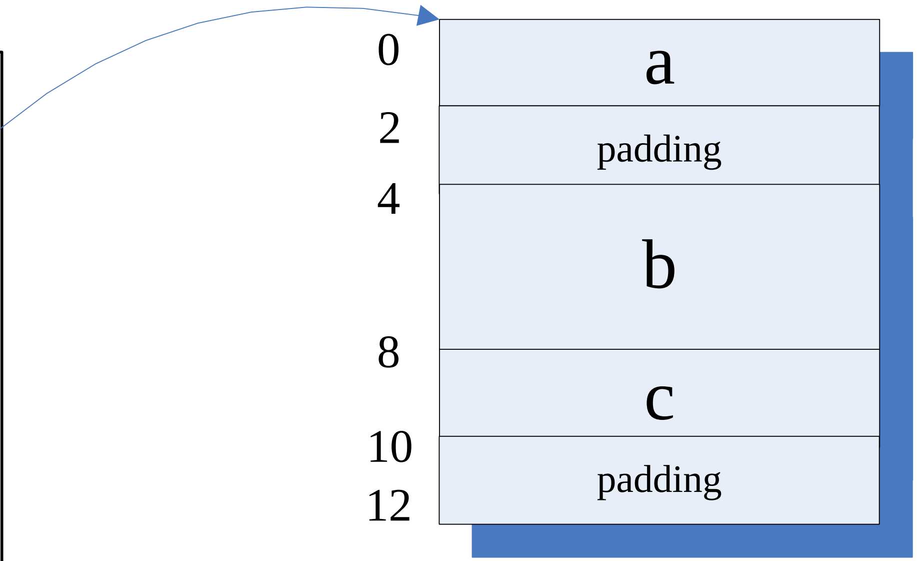


```
struct foo
{
    int a;
    int b;
    int c;
};
```

Data Layout

- Alignment:

```
struct foo
{
    short a;
    int b;
    short c;
};
```



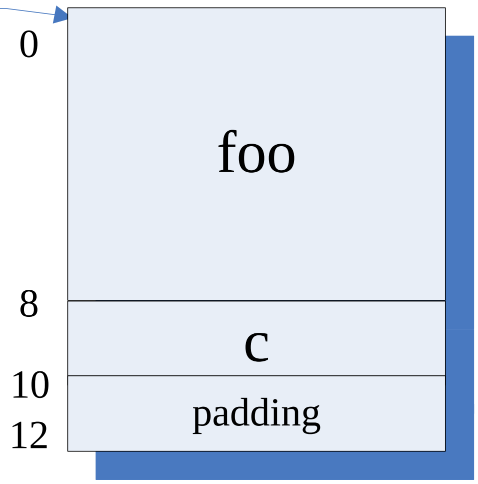
0 a
2 padding
4 b
8 c
10 padding
12

4-bit alignment

Data Layout

- Inheritance:

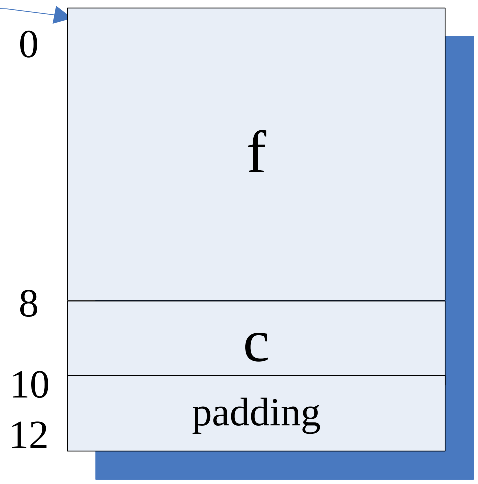
```
struct foo
{
    short a;
    int b;
};
Struct bar : foo
{
    short c;
};
```



Data Layout

- Object in object:

```
struct foo
{
    short a;
    int b;
};
struct bar
{
    foo f;
    short c;
};
```



the same as inheritance

Data Layout

- Static Storage: somewhere else
  - Static members
  - Functions
  - Vtable
- Metaclass

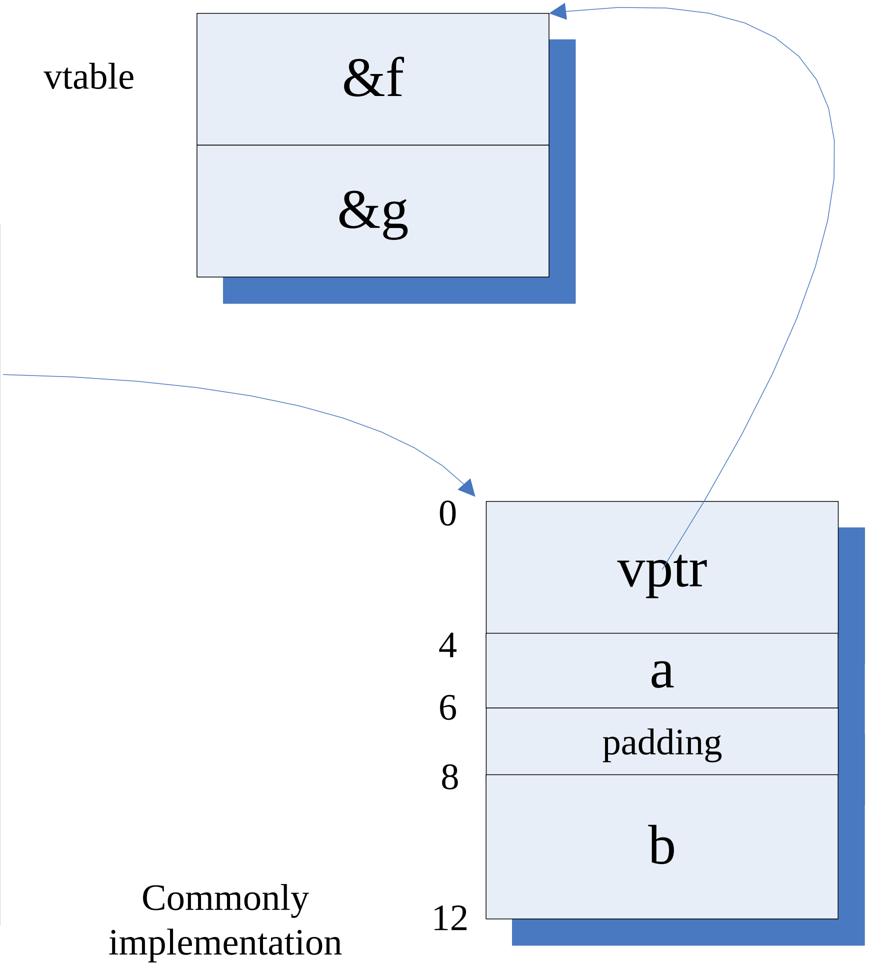
Virtual Binding

- Virtual binding:
  - A pointer or reference to an object calls virtual function
- Static binding
  - An object calls function
  - A pointer or reference to an object or the object itself calls any non-virtual function

Data Layout

- Almost portable virtual table

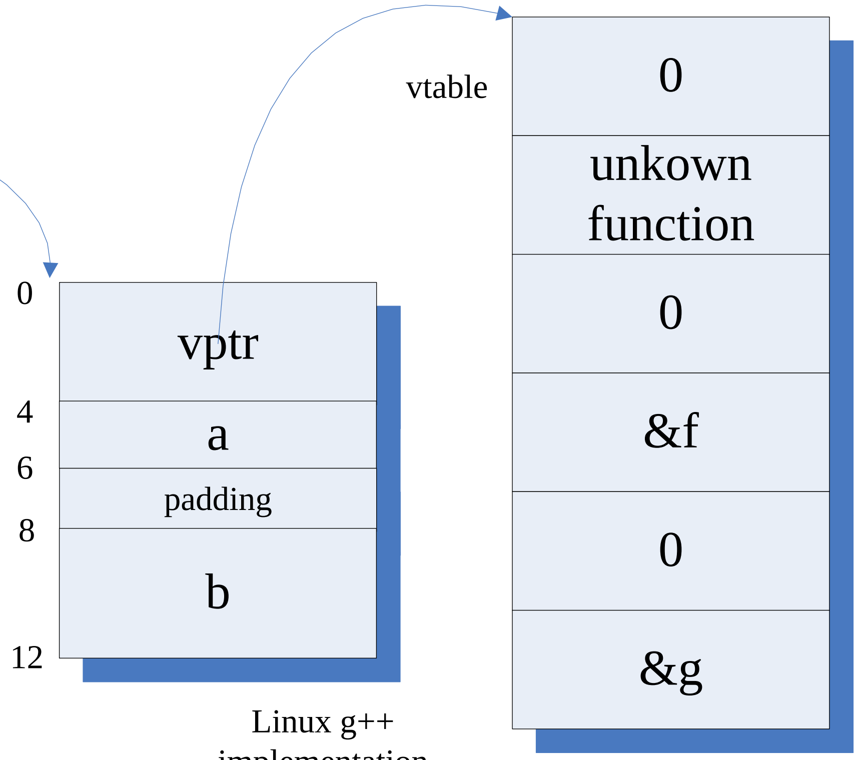
```
struct foo
{
    short a;
    int b;
    virtual void f(){}
    virtual void g(){}
};
```



Data Layout

- Linux g++ virtual table (From Imperfect C++)

```
struct foo
{
    short a;
    int b;
    virtual void f(){}
    virtual void g(){}
};
```

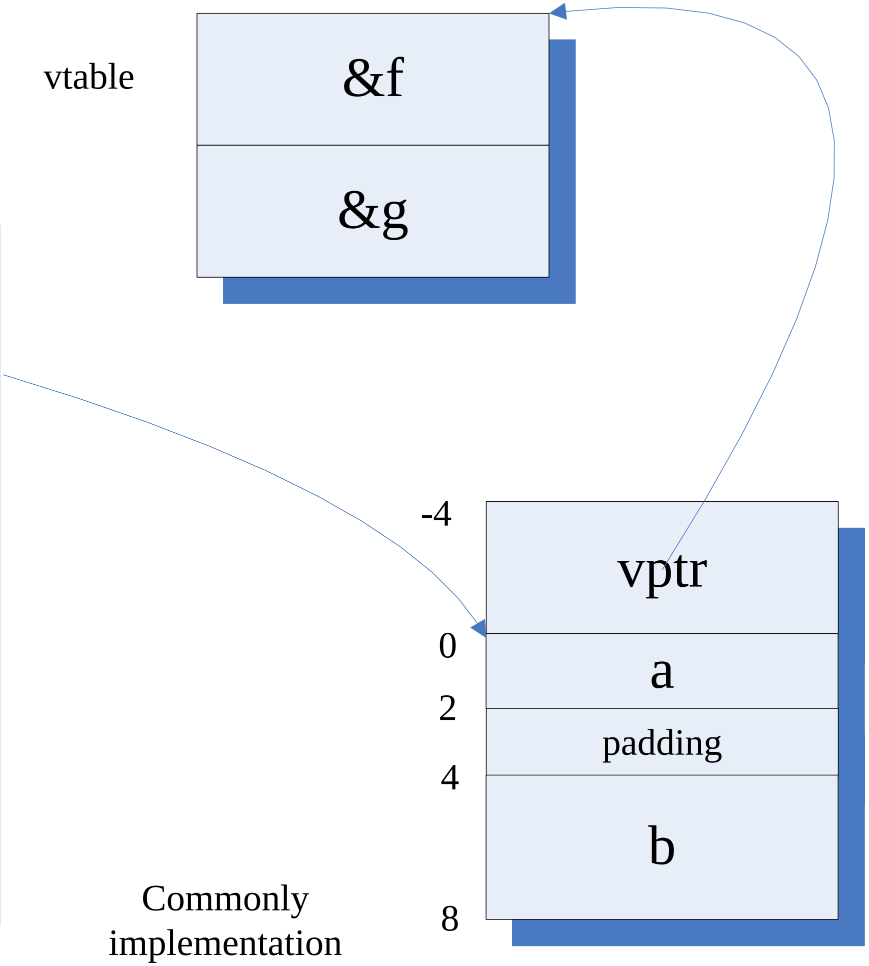


Linux g++ implementation

Data Layout

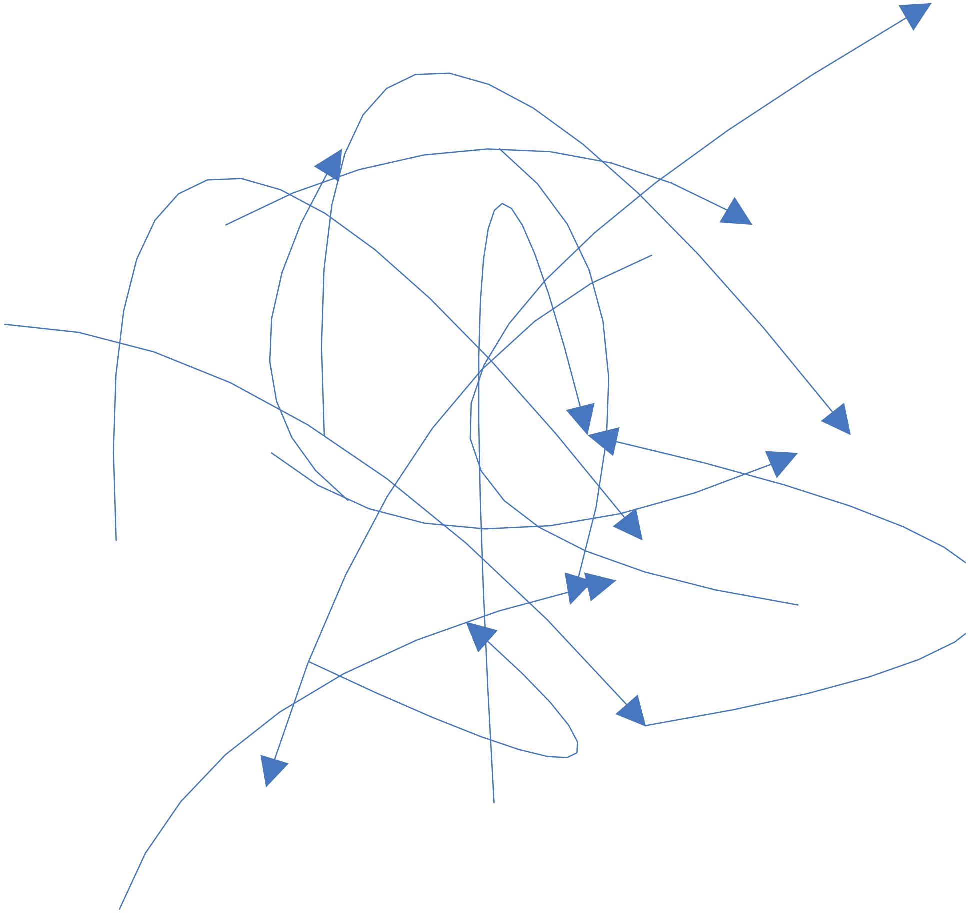
- Another possible layout

```
struct foo
{
    short a;
    int b;
    virtual void f(){}
    virtual void g(){}
};
```



Data Layout

- Multiple virtual inheritance
  - Chaotic evil

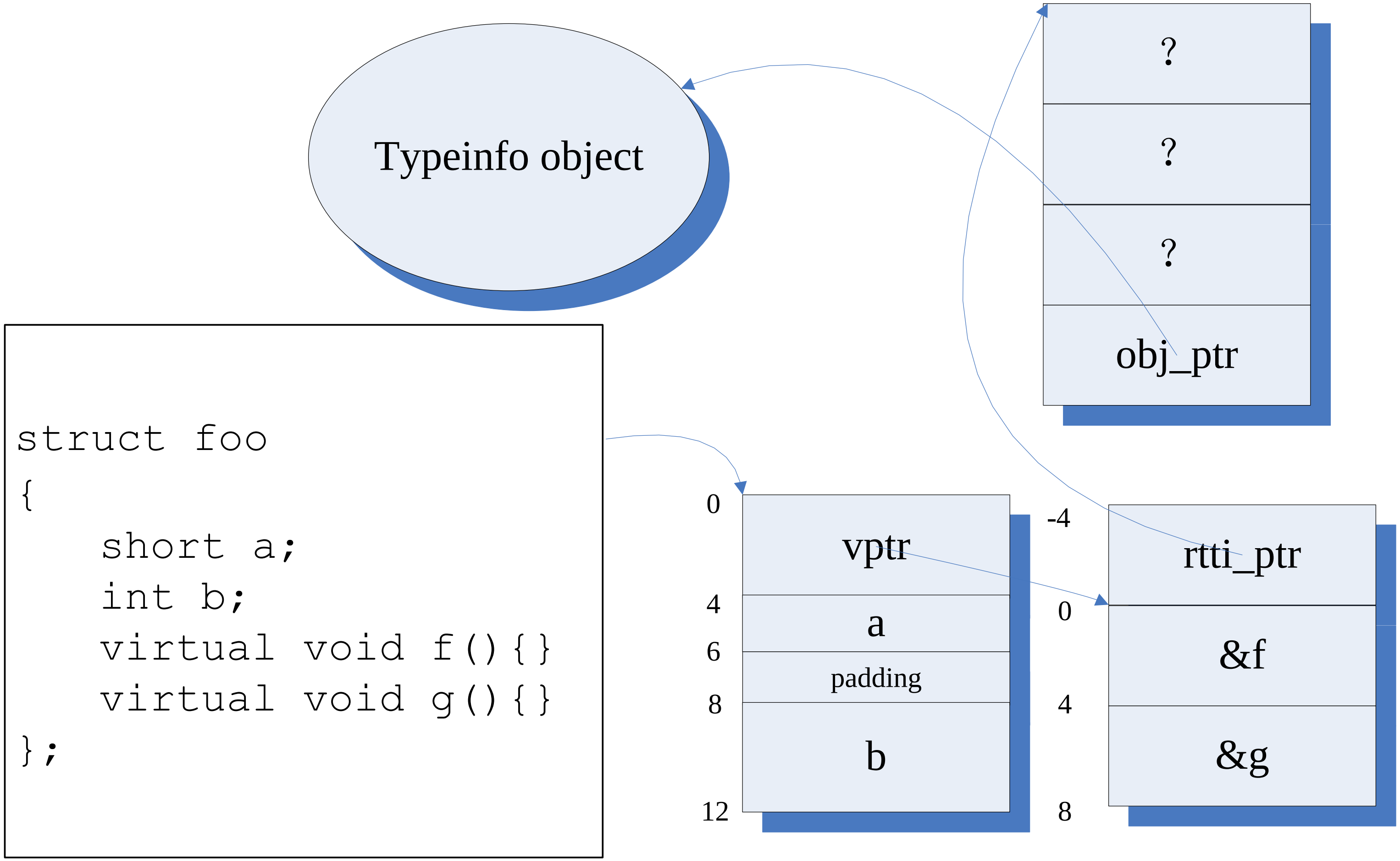


Accessibility

- public/protected/private
- Anti-gentlemen's not anti-villain
- Methods to penetrate private/protected protection
  - #define private public
  - Redeclare class
  - Raw pointer access
  - Template specialization
  - Inheritance

Type info

- MSVC implementation

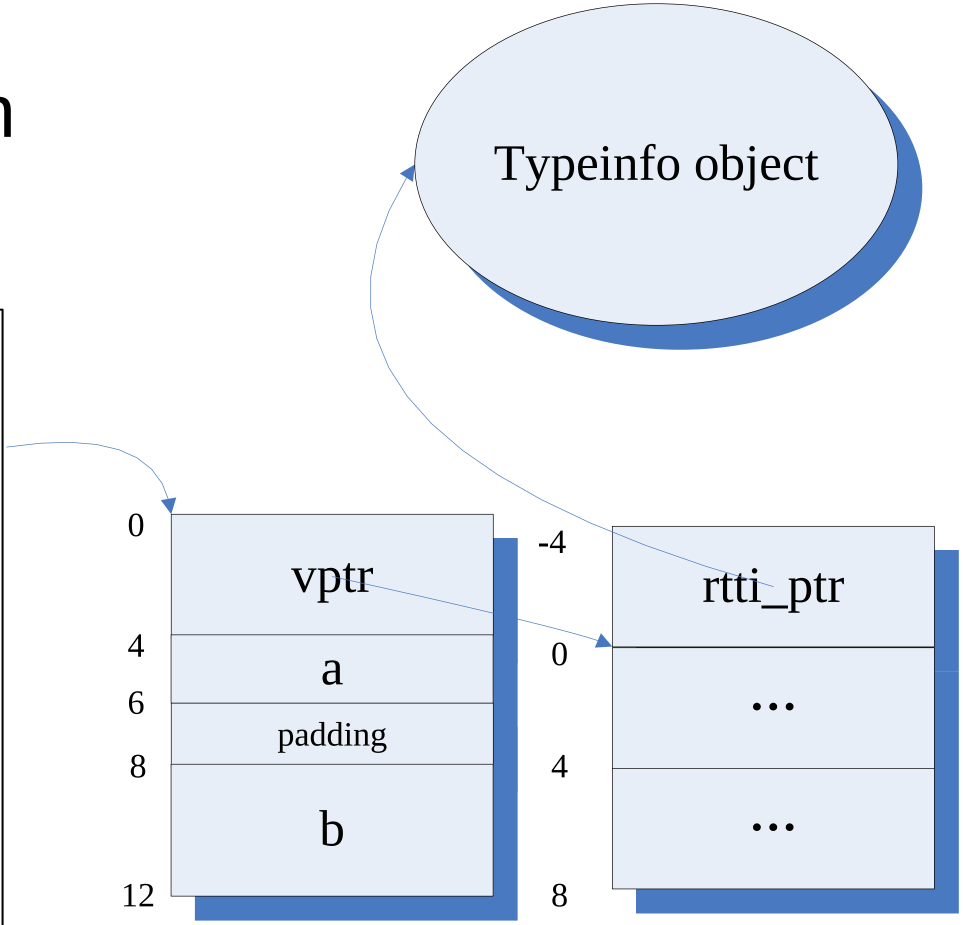


```
struct foo
{
    short a;
    int b;
    virtual void f(){}
    virtual void g(){}
};
```

Type info

- g++ implementation

```
struct foo
{
    short a;
    int b;
    virtual void f(){}
    virtual void g(){}
};
```



Type info

- Typeinfo lookup: typeid(class) or typeid(obj)
- Implementation (MSVC 2003)
  - typeid(class) or typeid(obj) in which obj is not a reference: statically table lookup
  - typeid(obj) in which obj is a reference:
    - All types except class/struct: statically table lookup
    - Class/struct without virtual function: statically table lookup
    - Class/struct with virtual function: find type_info object through vptr

Construct Order

- Construct virtual base class(es)
- Construct base class(es)
- Construct vptr(s)
- Construct objects not in initialization list
- Construct objects in initialization list
- Call constructor

Construct Order

- vptr is replaced again and again down the hierarchy tree
- Virtual function lose its virtuousness before the construction complete reguardless of static or dynamic binding