Linux Binary Exploitation

C++ Exploitation

angelboy@chroot.org

Outline

- Name Mangling
- Virtual function table
  - Vtable Hijacking
- Vector & String
- New & delete
- Copy constructor & assignment operator

Outline

- Name Mangling
- Virtual function table
  - Vtable Hijacking
- Vector & String
- New & delete
- Copy constructor & assignment operator

Name Mangling

- C++ 為了 Overloading 時，可以讓 compiler 和 linker 可以辨別出相同 function 名稱，參數不同的 function 引用的機制
- 使 programer 在不同的 namespace 底下可以有著多個相同名稱的 function

Name Mangling

- 而在 compiler 和 linker 處理 symbol 時，就會使用該機制讓每個 function 名對應到一個修飾過後的名稱

- C++ 中全域變數和靜態變數也有相同的機制

```
angelboy@ubuntu:~$ c++filt _ZNSo3putEc
std::basic_ostream<char, std::char_traits<char> >::put(char)
```

Name Mangling

- 在 gdb 中可以使用下列指令讓 function 好看一點
  - *set print asm-demangle on*

Outline

- Name Mangling
- Virtual function table
- Vector & String
- New & delete
- Copy constructor & assignment operator

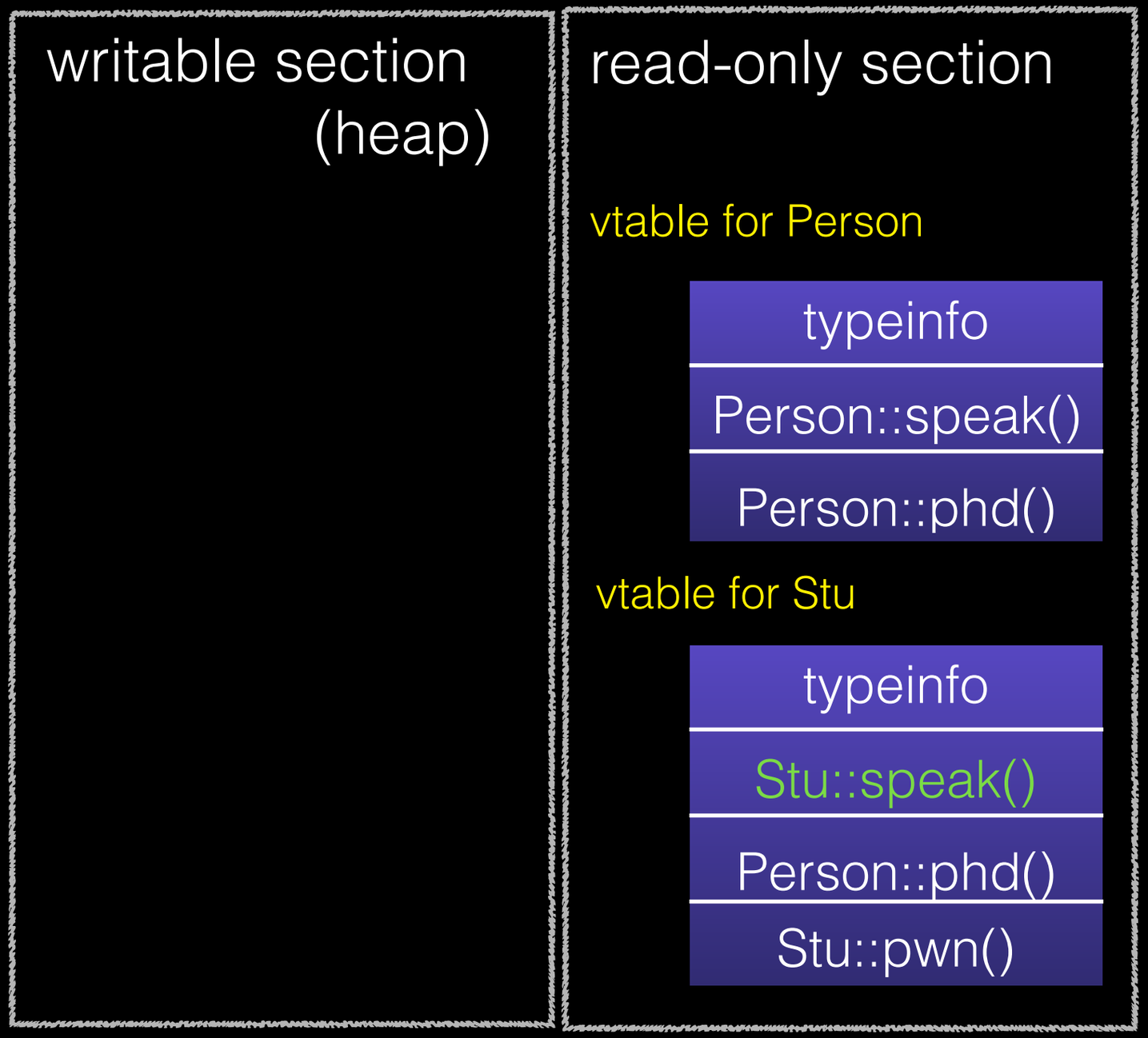
Virtual function table

- Virtual function is a key mechanism to support polymorphism in C++
- For each class with virtual functions, depending on the class inheritance hierarchy, the compiler will create one or more associated virtual function table

Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

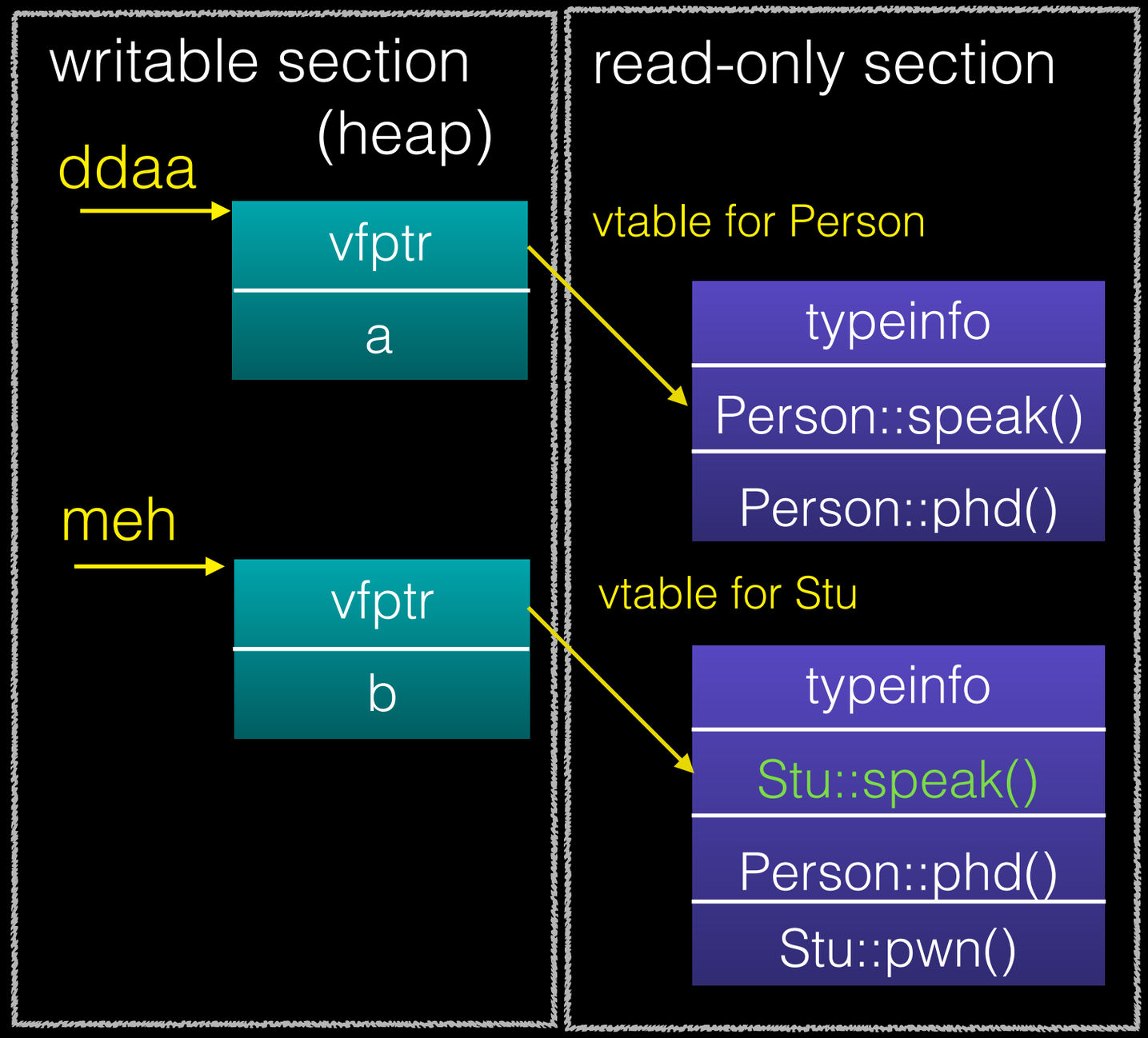
compiler generates
the table for all class



Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

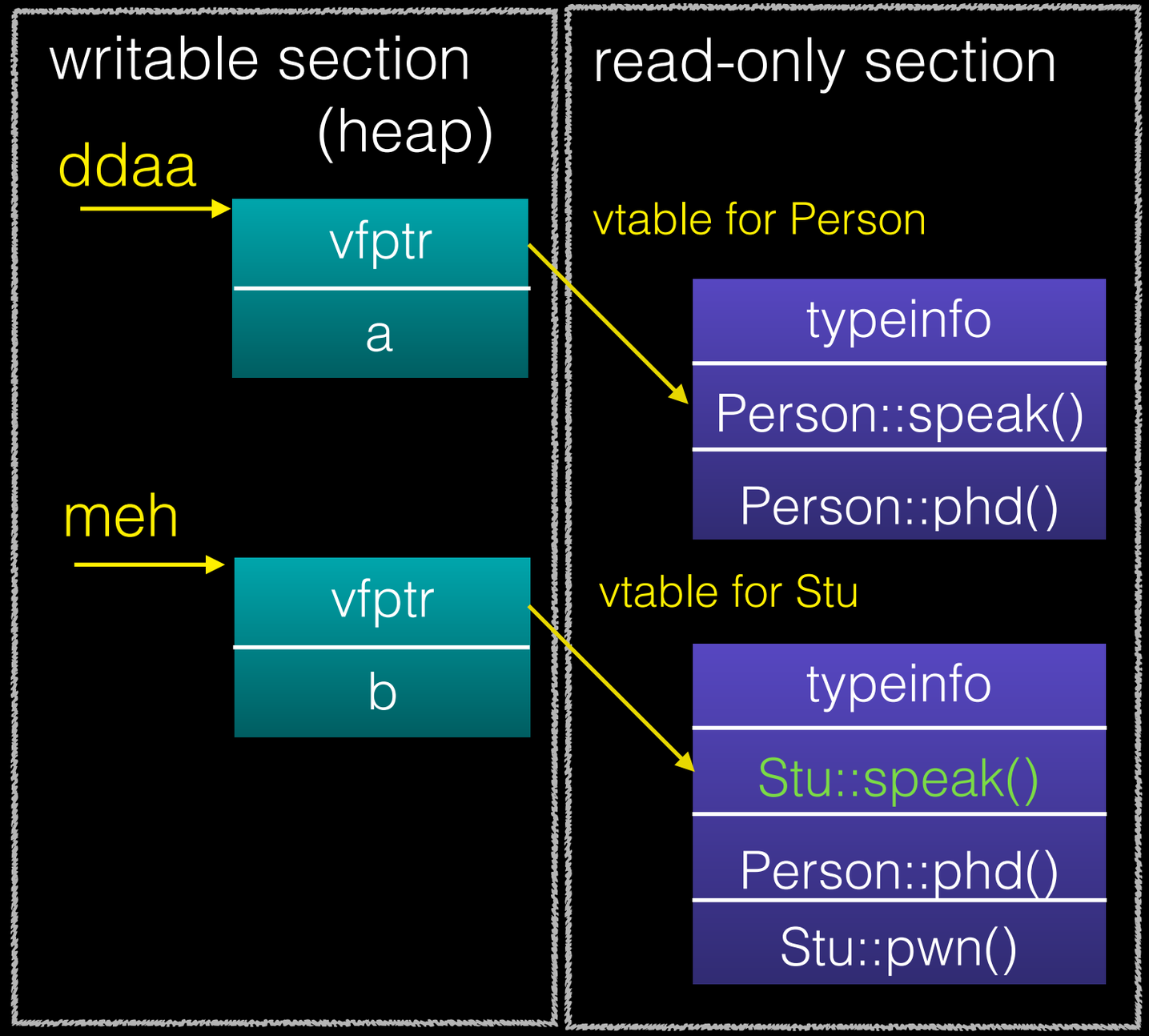
new a Person and a Stu object



Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

ddaa->speak()

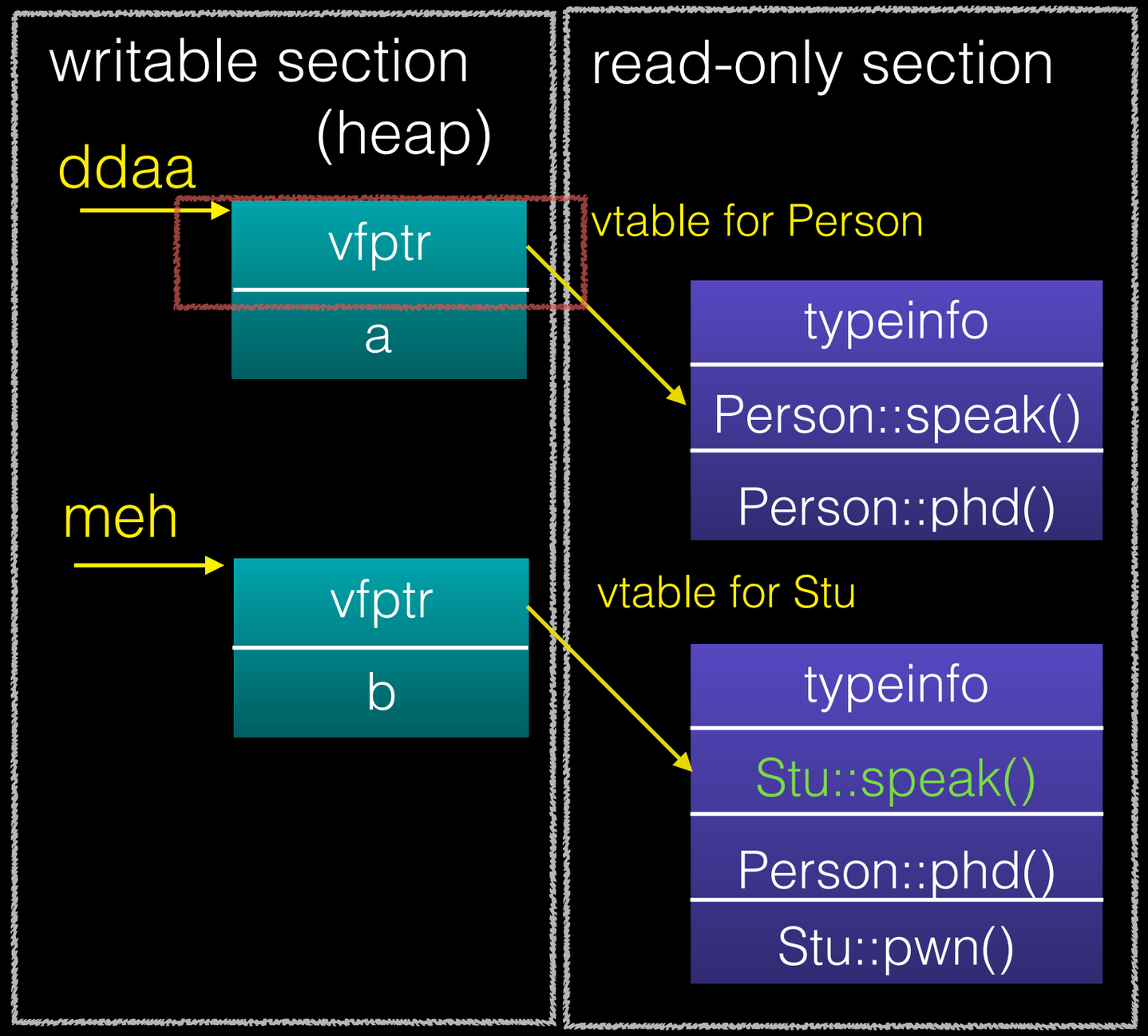


Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

vfptr == *ddaa

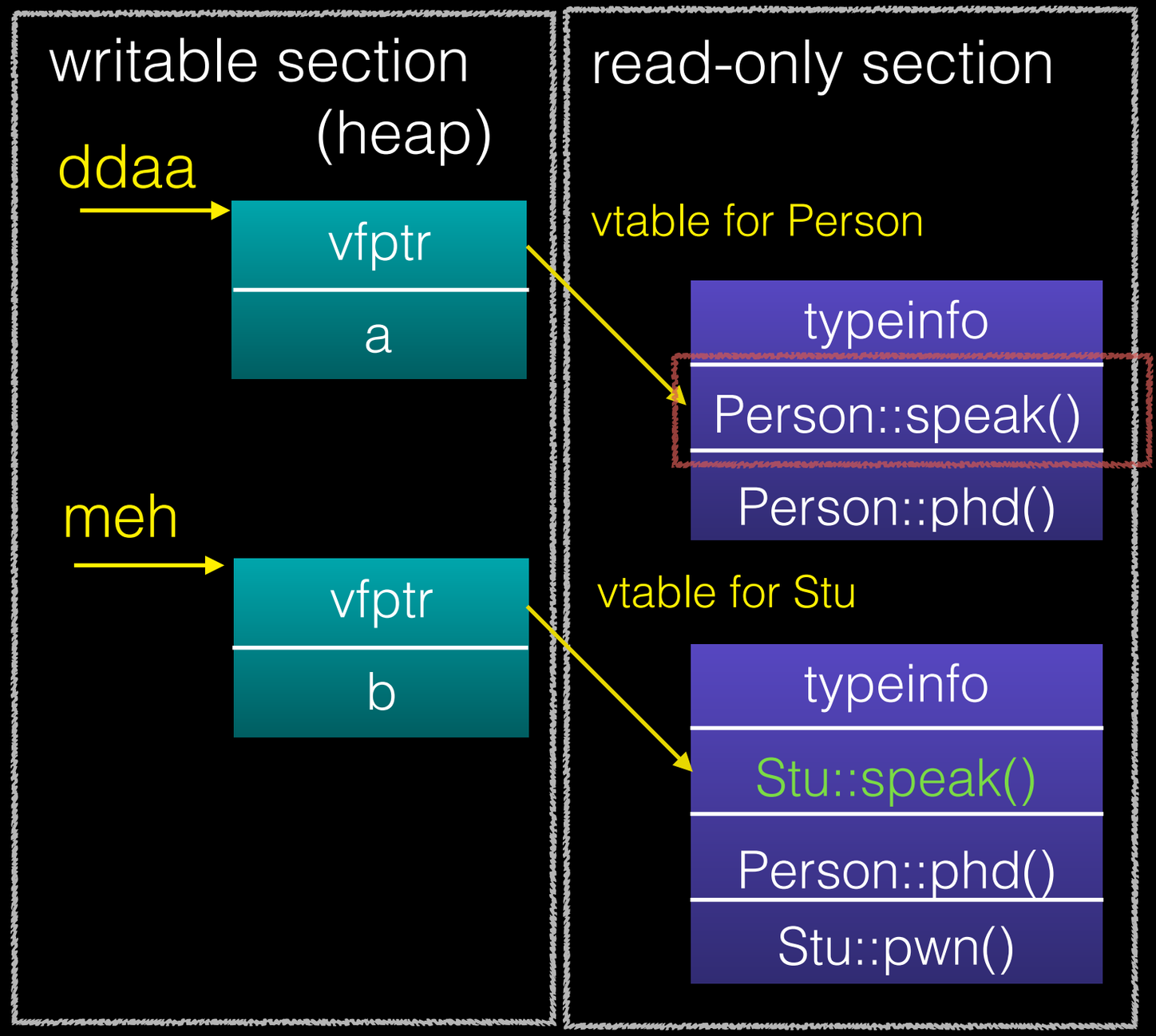
取 vfptr



Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

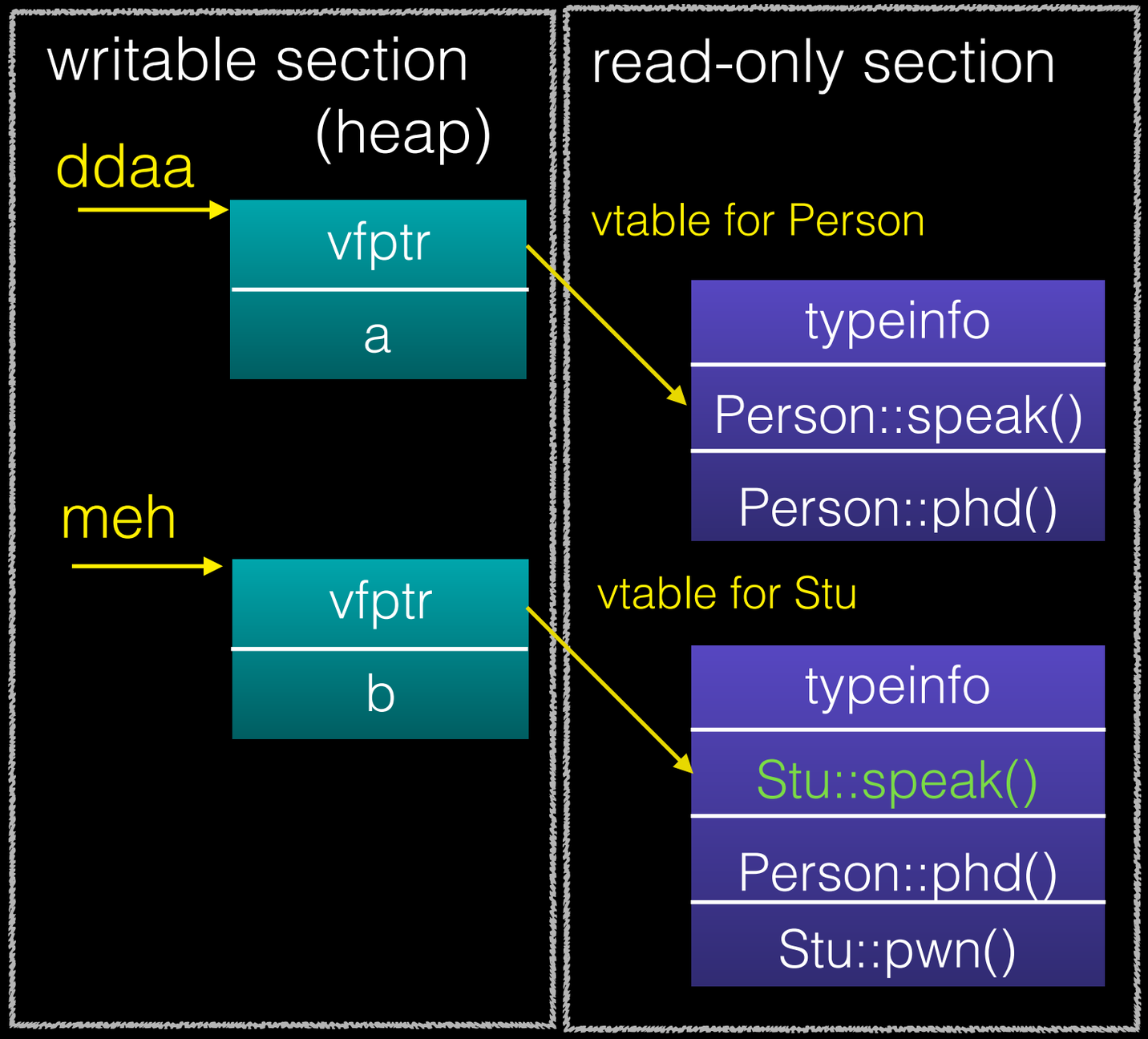
call *vfptr
(Person::speak(ddaa))



Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

meh->speak()

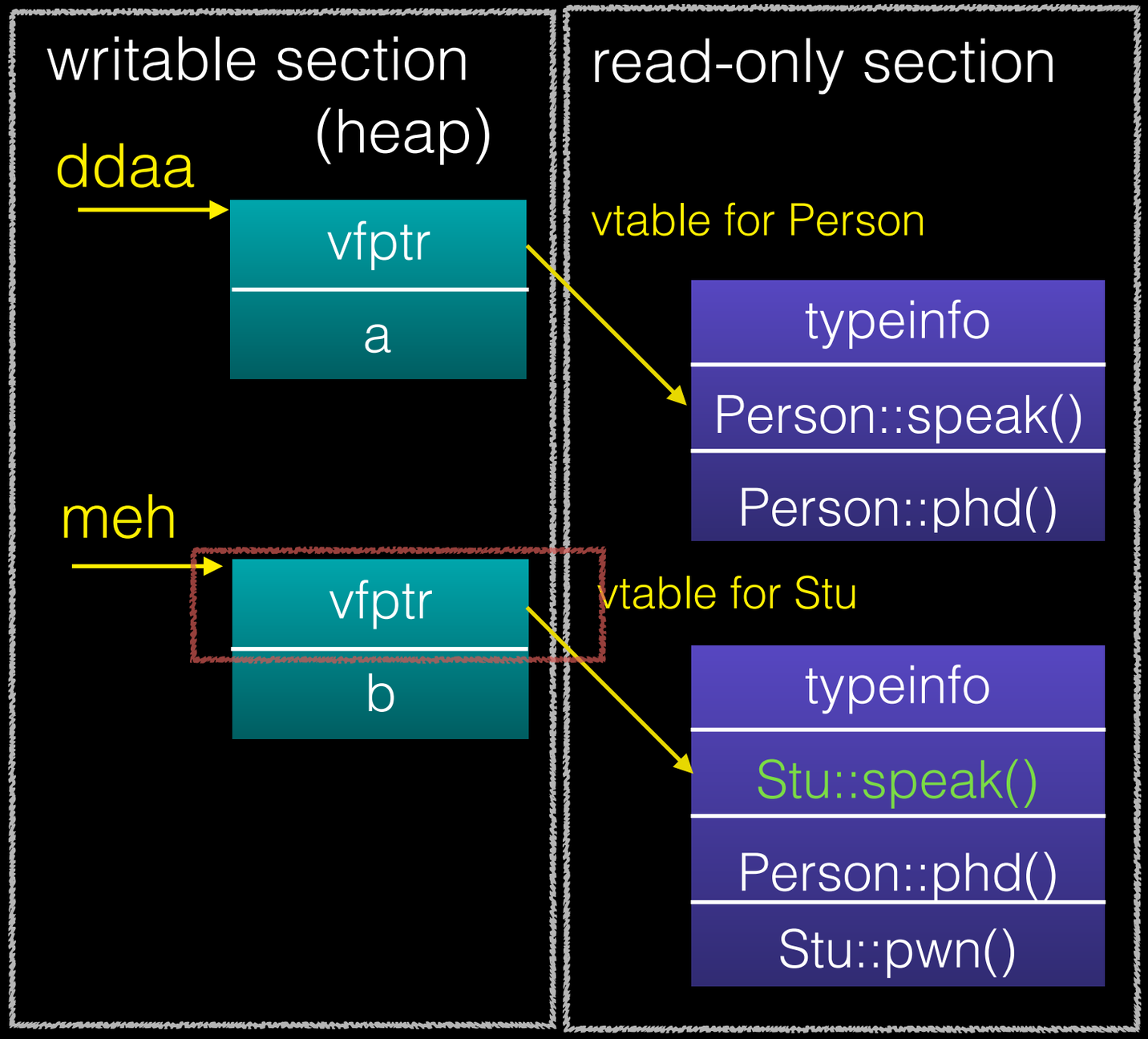


Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

vfptr == *meh

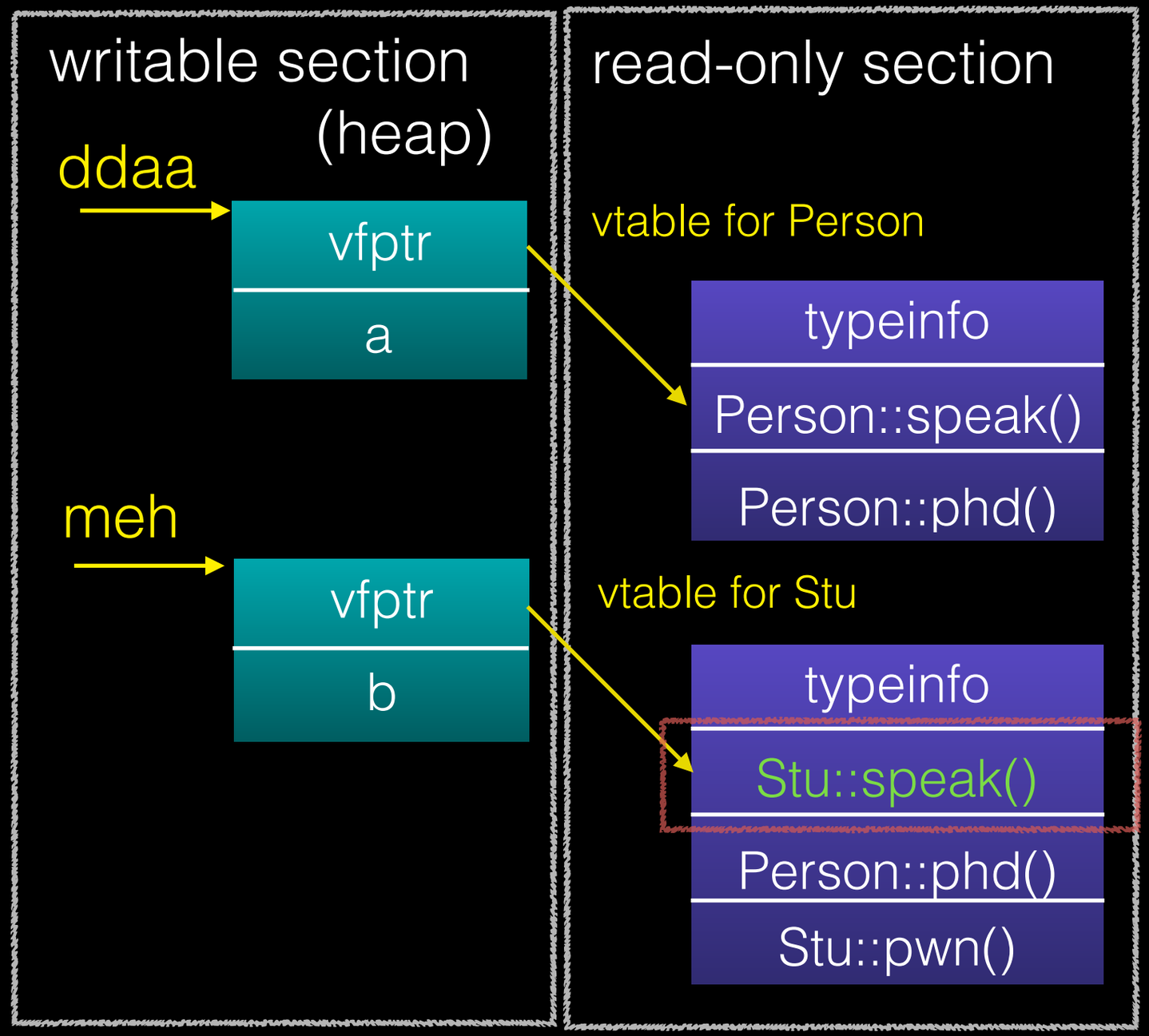
取 vfptr



Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

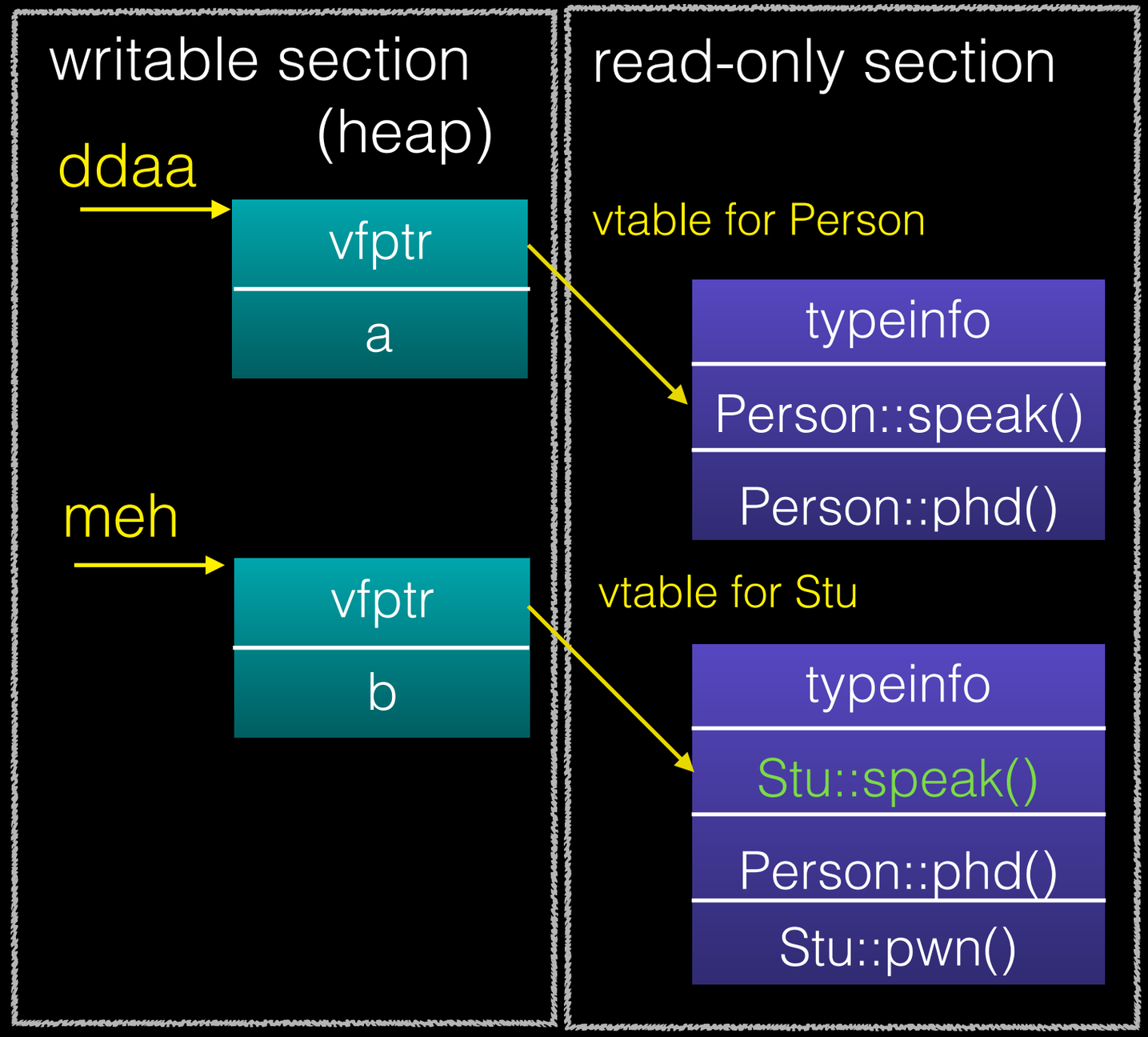
call *vfptr (Stu::speak(meh))



Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0;
}
```

meh->pwn()

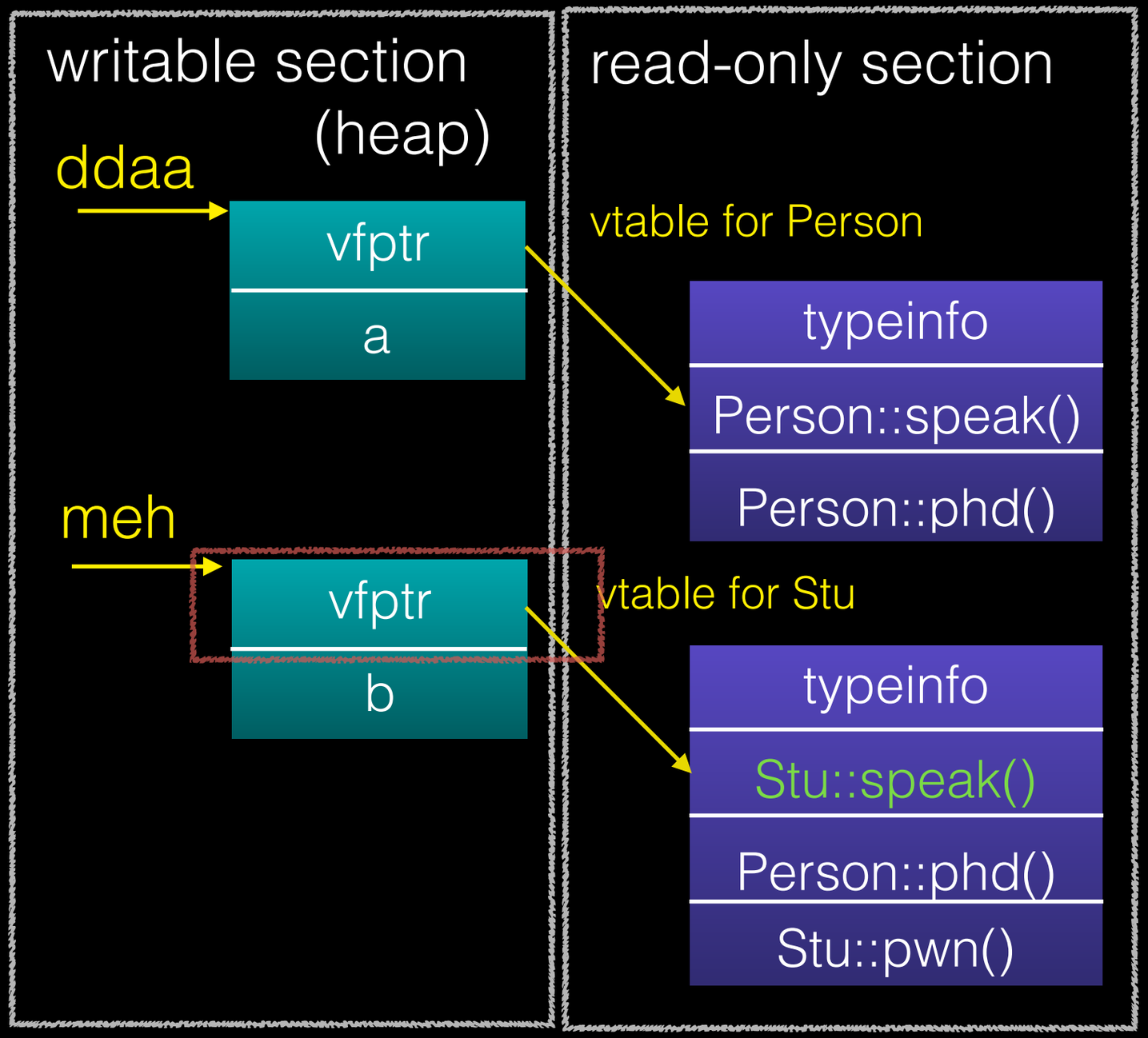


Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0;
}
```

vfptr == *meh

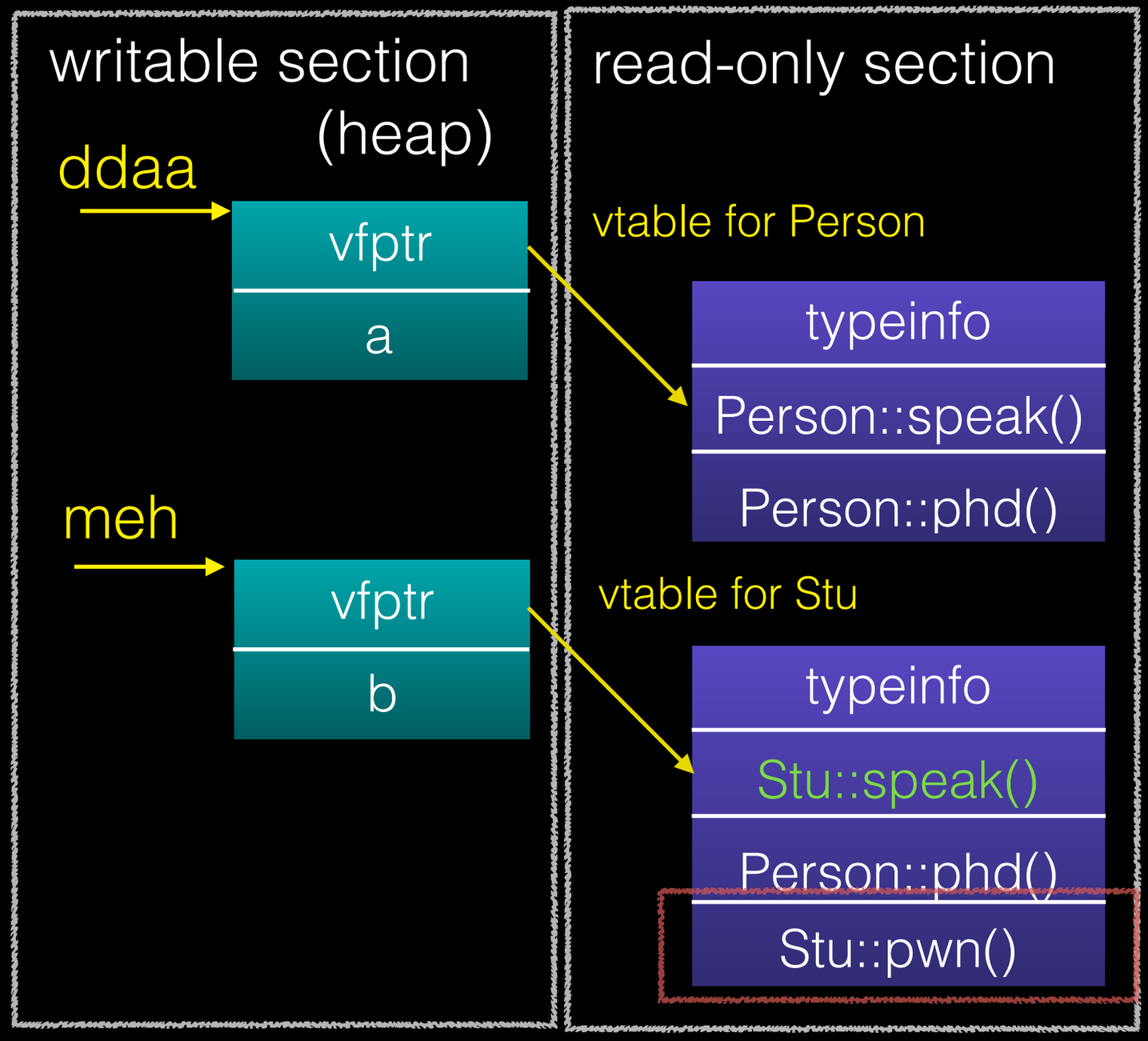
取 vfptr



Virtual function table

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0;
}
```

call *(vfptr+0x10)
(Stu::pwn(meh))

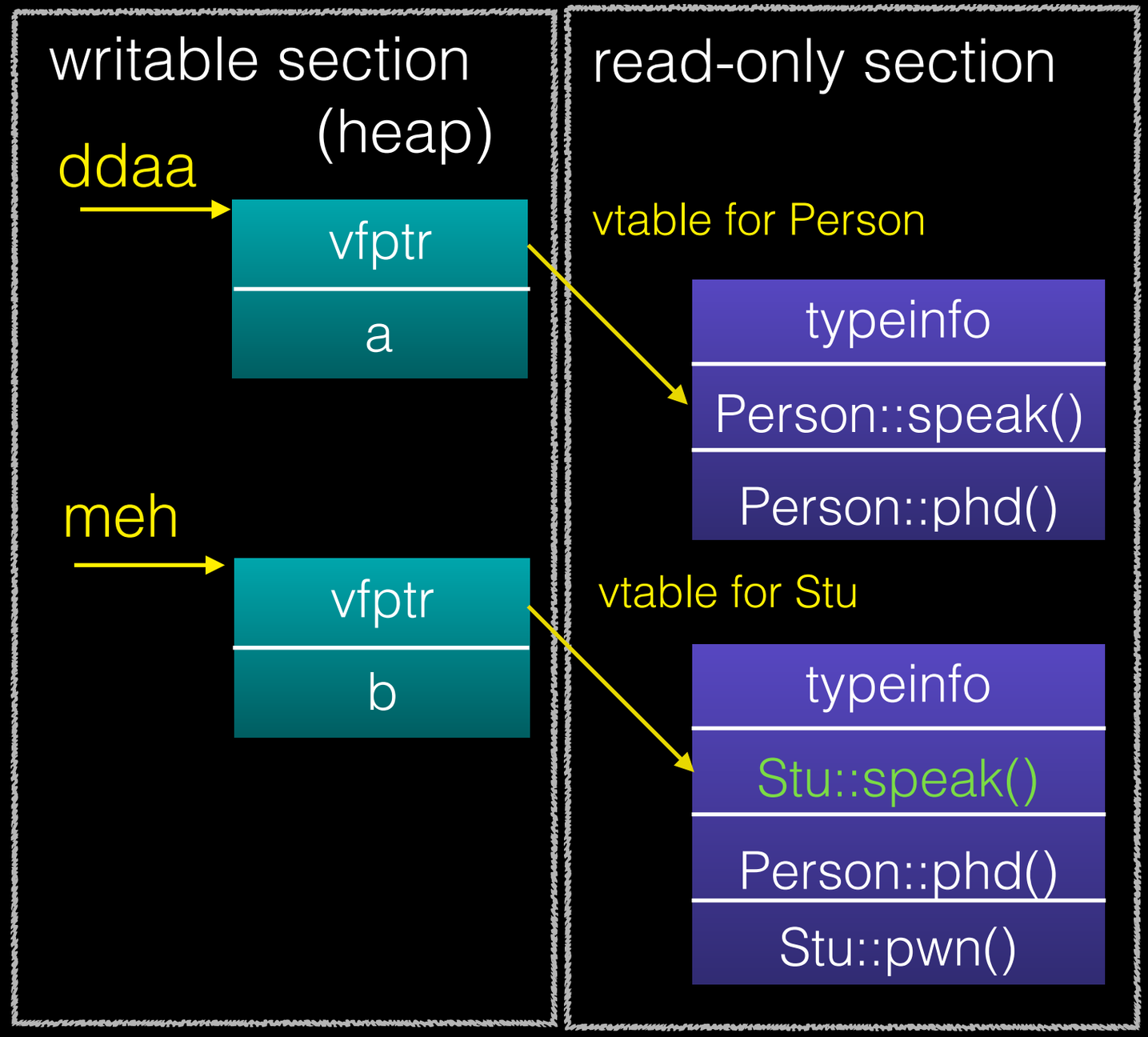


Vtable Hijacking

- Need other vulnerabilities
  - Use-after-free, Heap overflow ….
- Force the table and Hijack the vfptr
  - Because the vfptr is writable

Vtable Hijacking

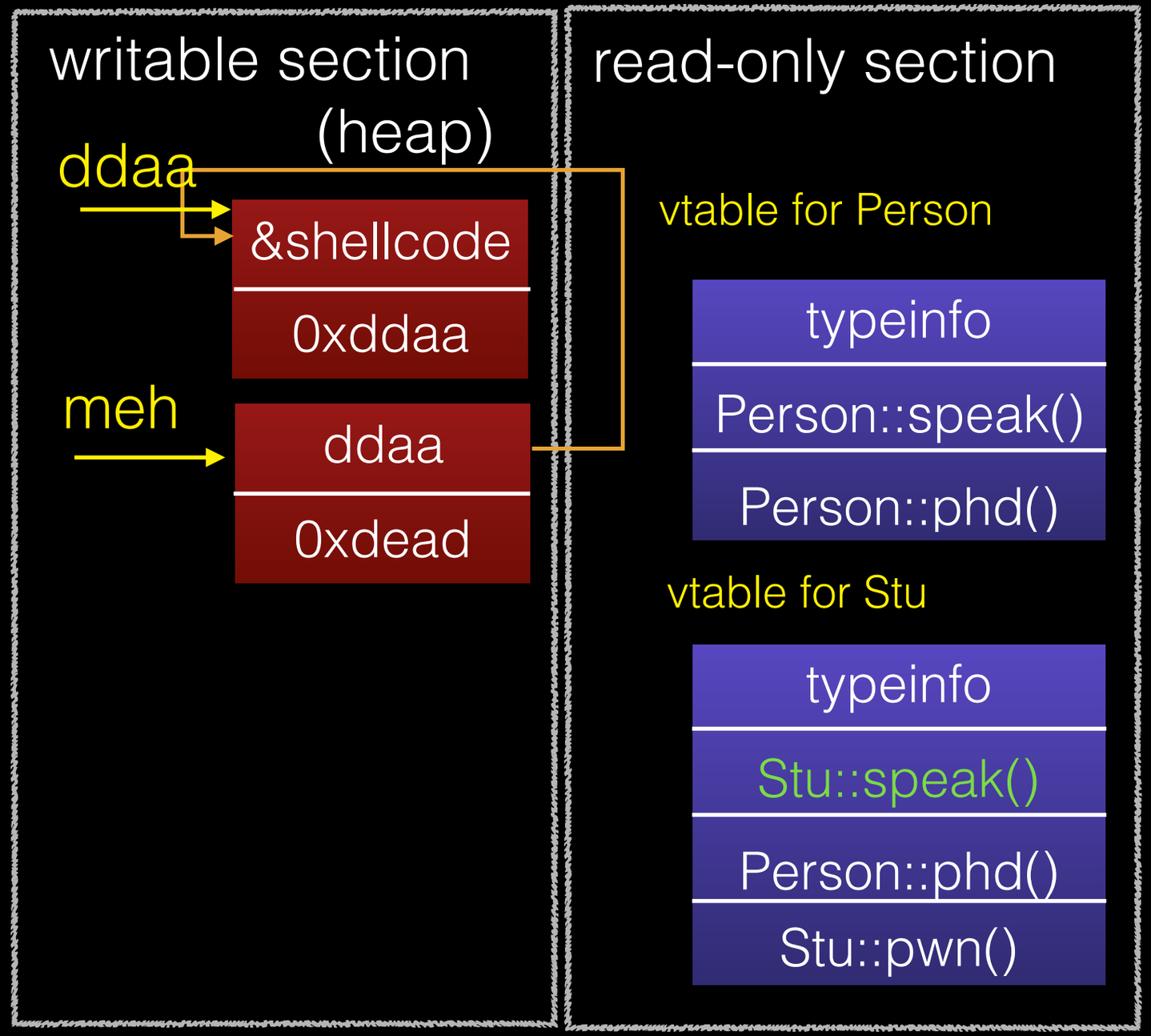
```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```



Vtable Hijacking

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

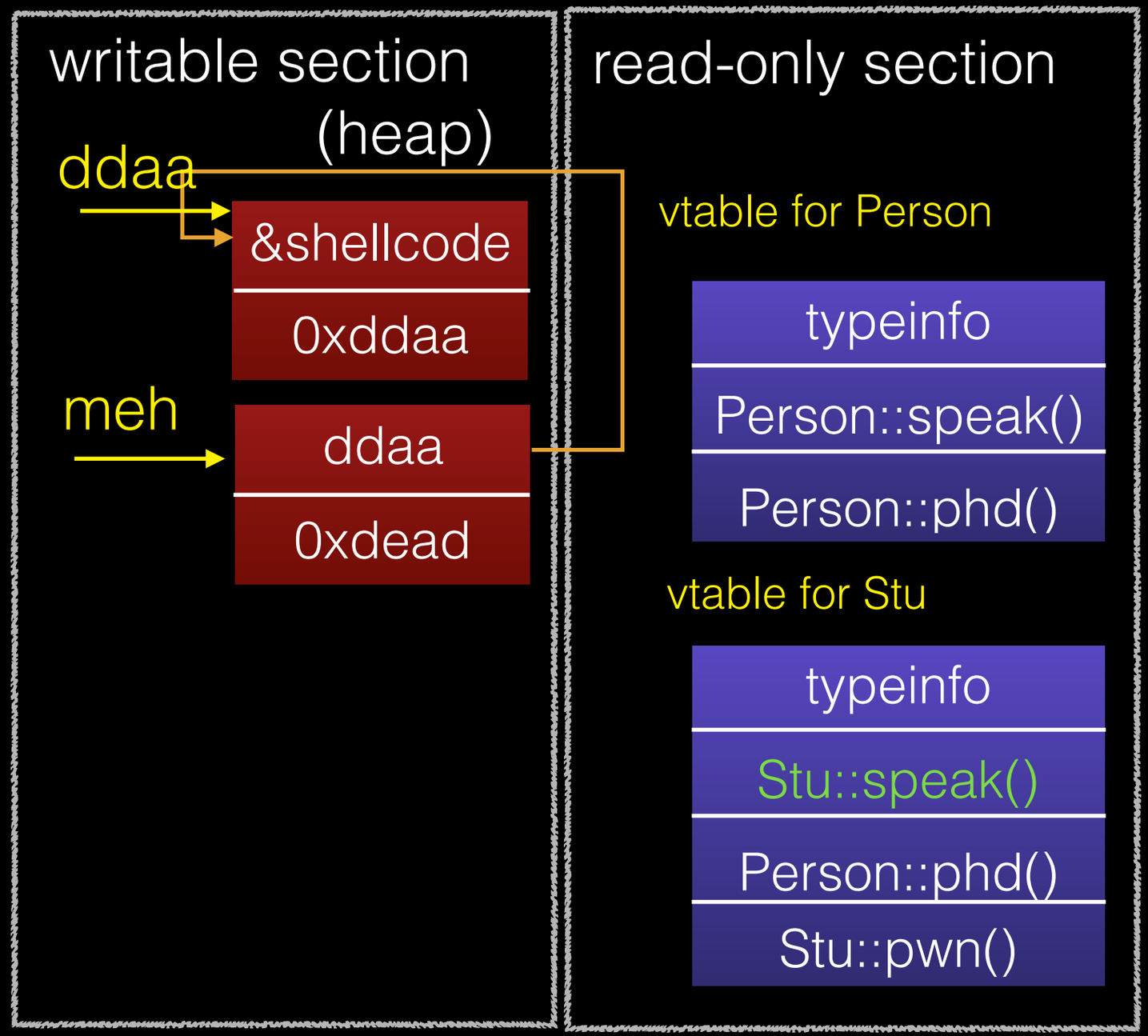
heap overflow
forces a vtabe in ddaa
and hijack the vfptr of meh



Vtable Hijacking

```
1  #include <iostream>
2  using namespace std;
3
4  class Person {
5      public :
6          virtual void speak(){;}
7          virtual void phd(){;}
8      private :
9          int a ;
10
11 };
12
13 class Stu :public Person {
14     public :
15         virtual void speak(){;}
16         virtual void pwn(){;}
17     private :
18         int b;
19 };
20
21
22
23 int main(void){
24     Person *ddaa = new Person();
25     Stu *meh = new Stu();
26     ddaa->speak();
27     meh->speak();
28     meh->pwn();
29     return 0 ;
30 }
31
```

meh->speak()

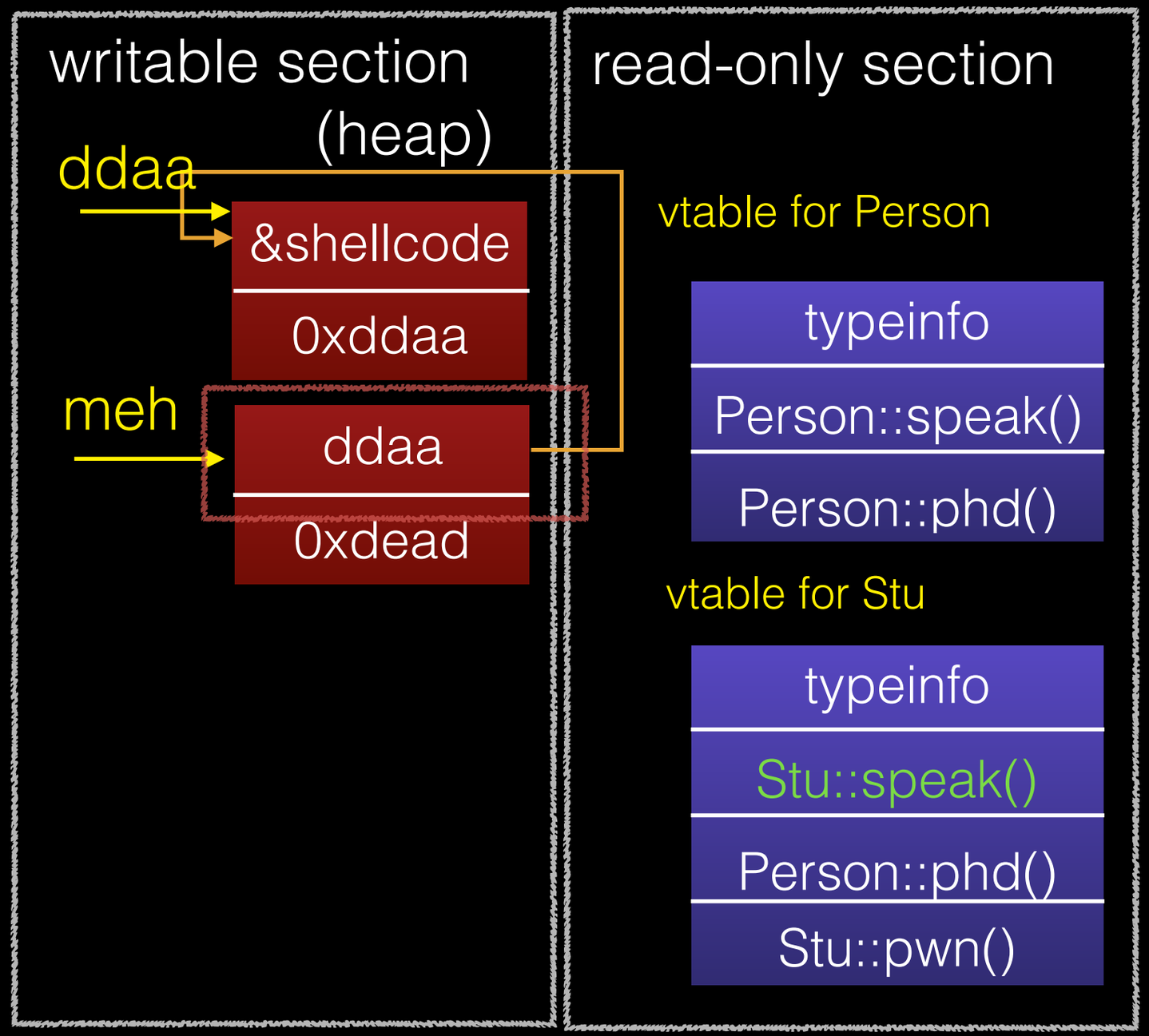


Vtable Hijacking

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

vfptr = *meh

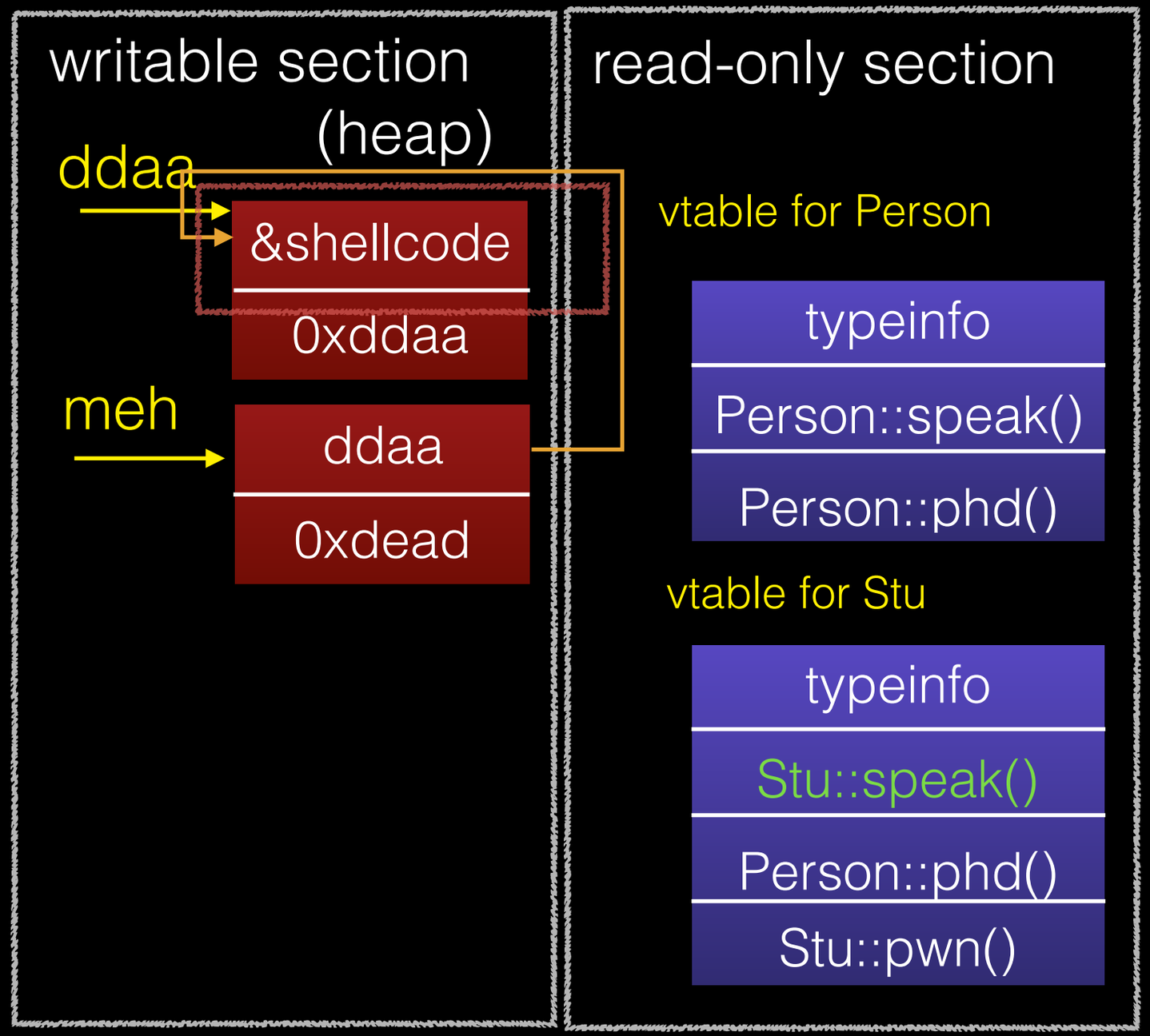
取 vfptr



Vtable Hijacking

```cpp
#include <iostream>
using namespace std;

class Person {
    public :
        virtual void speak(){;}
        virtual void phd(){;}
    private :
        int a ;

};

class Stu :public Person {
    public :
        virtual void speak(){;}
        virtual void pwn(){;}
    private :
        int b;
};


int main(void){
    Person *ddaa = new Person();
    Stu *meh = new Stu();
    ddaa->speak();
    meh->speak();
    meh->pwn();
    return 0 ;
}
```

call *(vfptr)
(call shellcode(meh))



Vtable Hijacking

```
1  #include <iostream>
2  using
3
4  class
5      pu
6
7
8      pr
9
10
11 };
12
13 class
14     pu
15
16
17     pr
18
19 };
20
21
22
23 int ma
24     Pe
25     St
26     do
27     me
28     me
29     re
30 }
31
```

writable section

read-only section

PWN !!

但通常會有 DEP/NX 保護，可能要跳 libc 中的 one-gadget
或是其他可利用的地方

Person::phd()

Stu::pwn()

call *(vfptr)
(call shellcode(meh))

Outline

- Name Mangling
- Virtual function table
- Vector & String
- New & delete
- Copy constructor & assignment operator

Vector & String

- Vector
  - A dynamic array
  - 分配在 heap 段
  - 比一般 c 中的陣列更有彈性，當空間不夠大時會重新兩倍大的的小來放置新的 vector ，再把原本的空間還給系統

Vector & String

- Vector
  - member
    - _M_start : vector 起始位置
      - vector::begin()
    - _M_finish : vector 結尾位置
      - vector::end()
    - _M_end_of_storage ：容器最後位置
      - if _M_finish == _M_end_of_storage in push_back
        - It will alloca a new space for the vector
        - 以這個來判斷空間是否足夠放元素

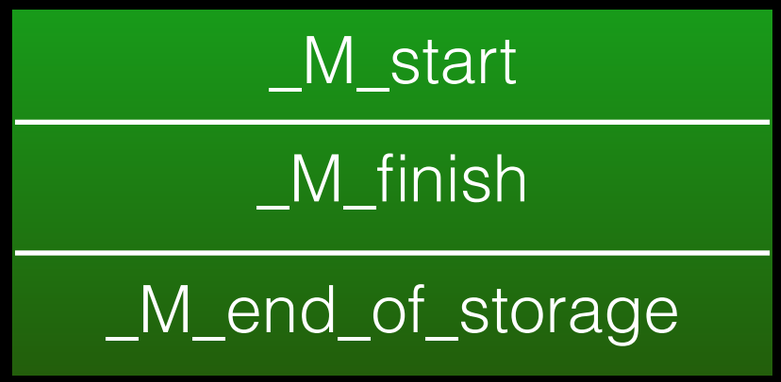
Vector & String

- Vector
  - member function
    - push_back : 在 vector 最後加入新元素
    - pop_back : 移除 vector 最後一個元素
    - insert ：插入元素到 vector 第 n 個位置
    - erase ：移除 vector 中第 n 個元素
    - ……

Vector & String

- Vector layout

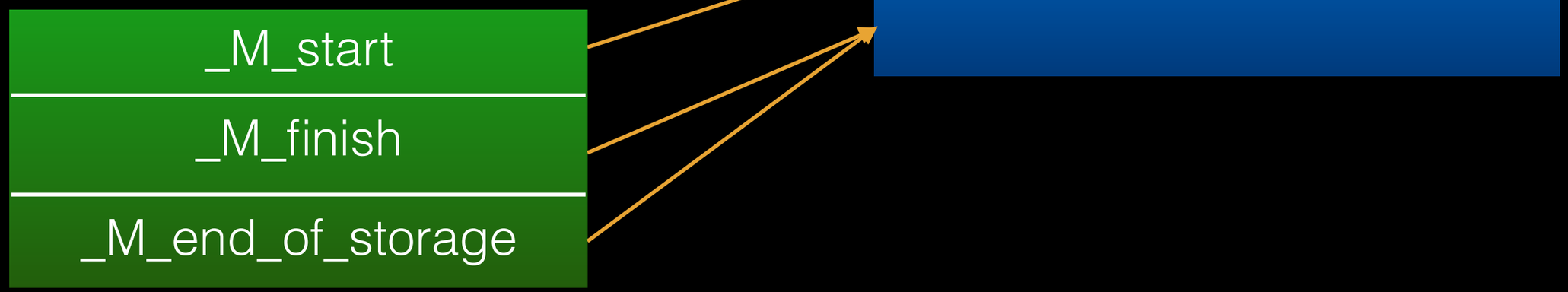
vector <string> vec



Vector & String

- Vector layout

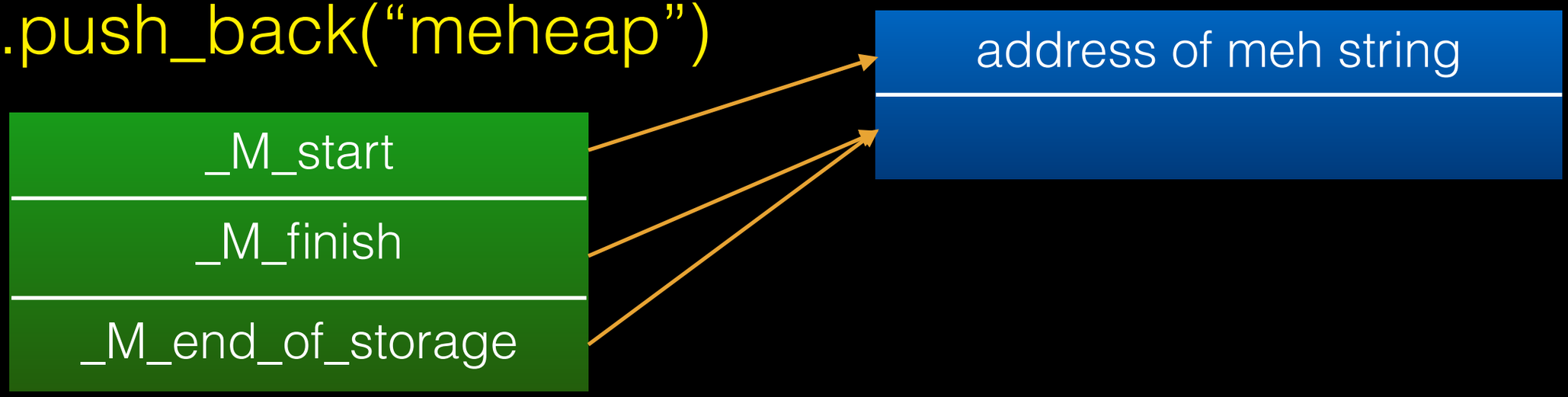
vec.push_back(“meh”)



Vector & String

- Vector layout

vec.push_back(“meheap”)



因為 _M_finish == _ M_end_of_storage

所以會先從新 new 一塊新的 vector

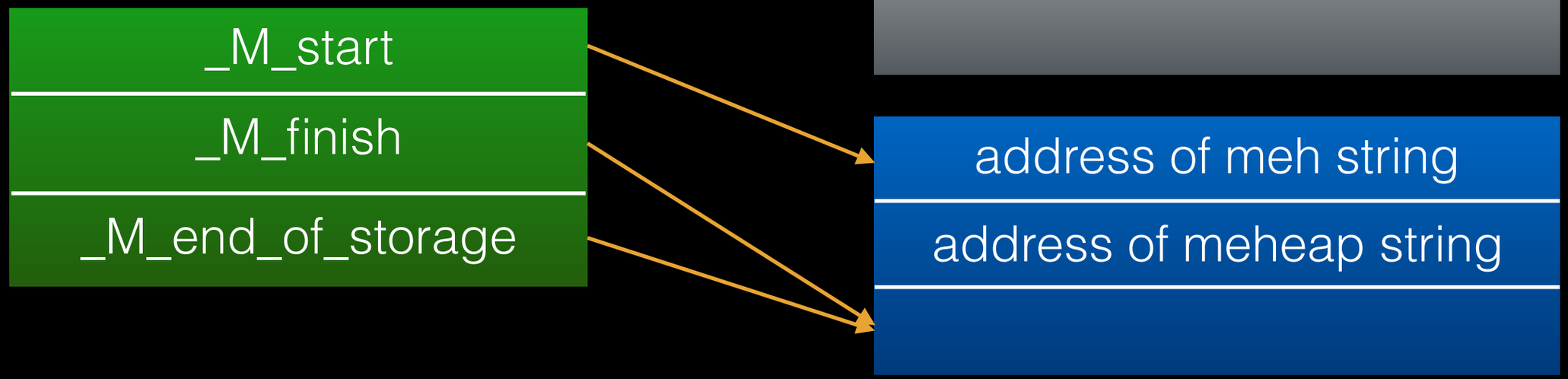
並把舊的值複製過去

再將藍色那塊 delete 掉

Vector & String

- Vector layout

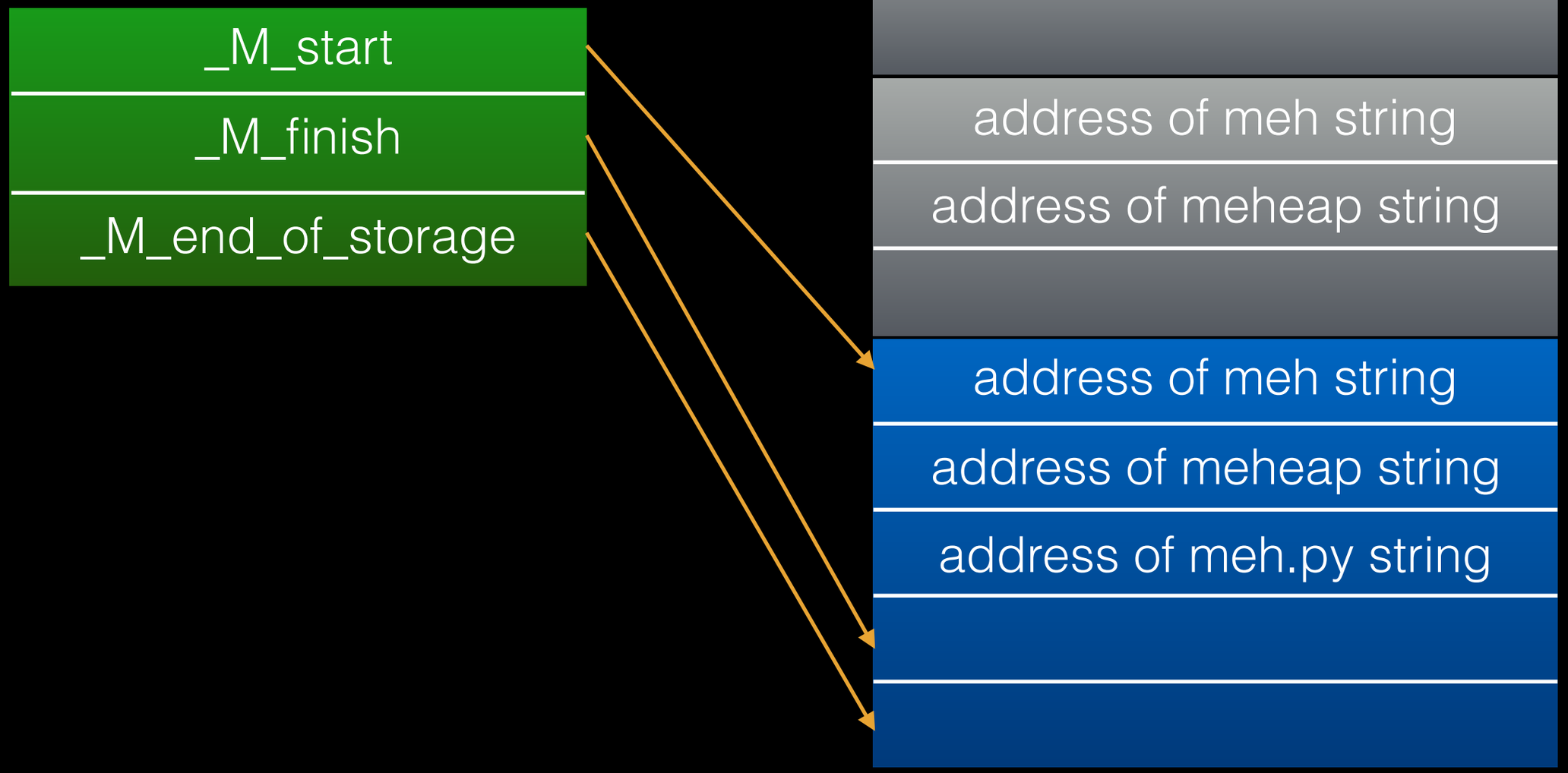
vec.push_back(“meheap”)



Vector & String

- Vector layout

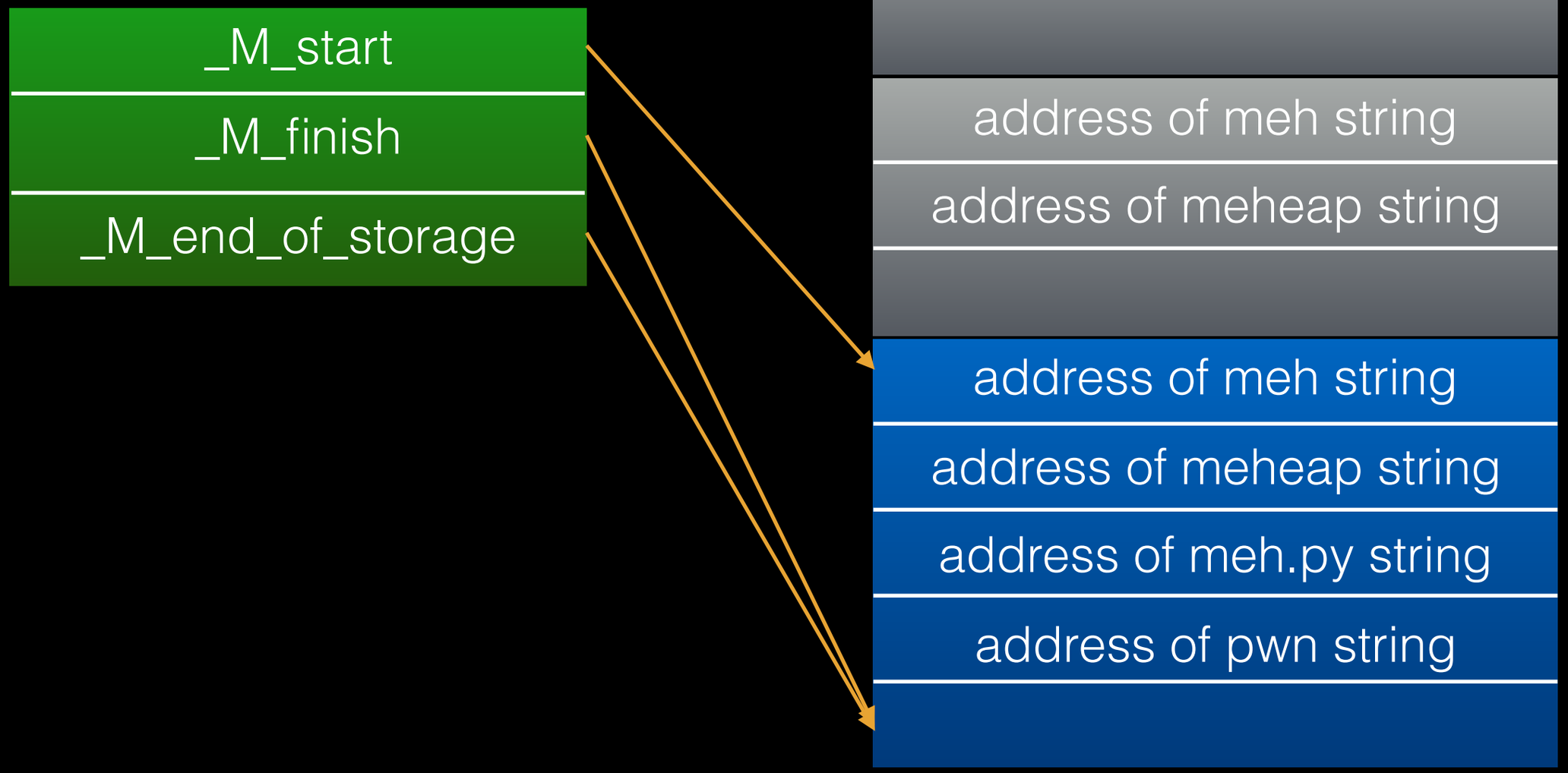
vec.push_back(“meh.py”)



Vector & String

- Vector layout

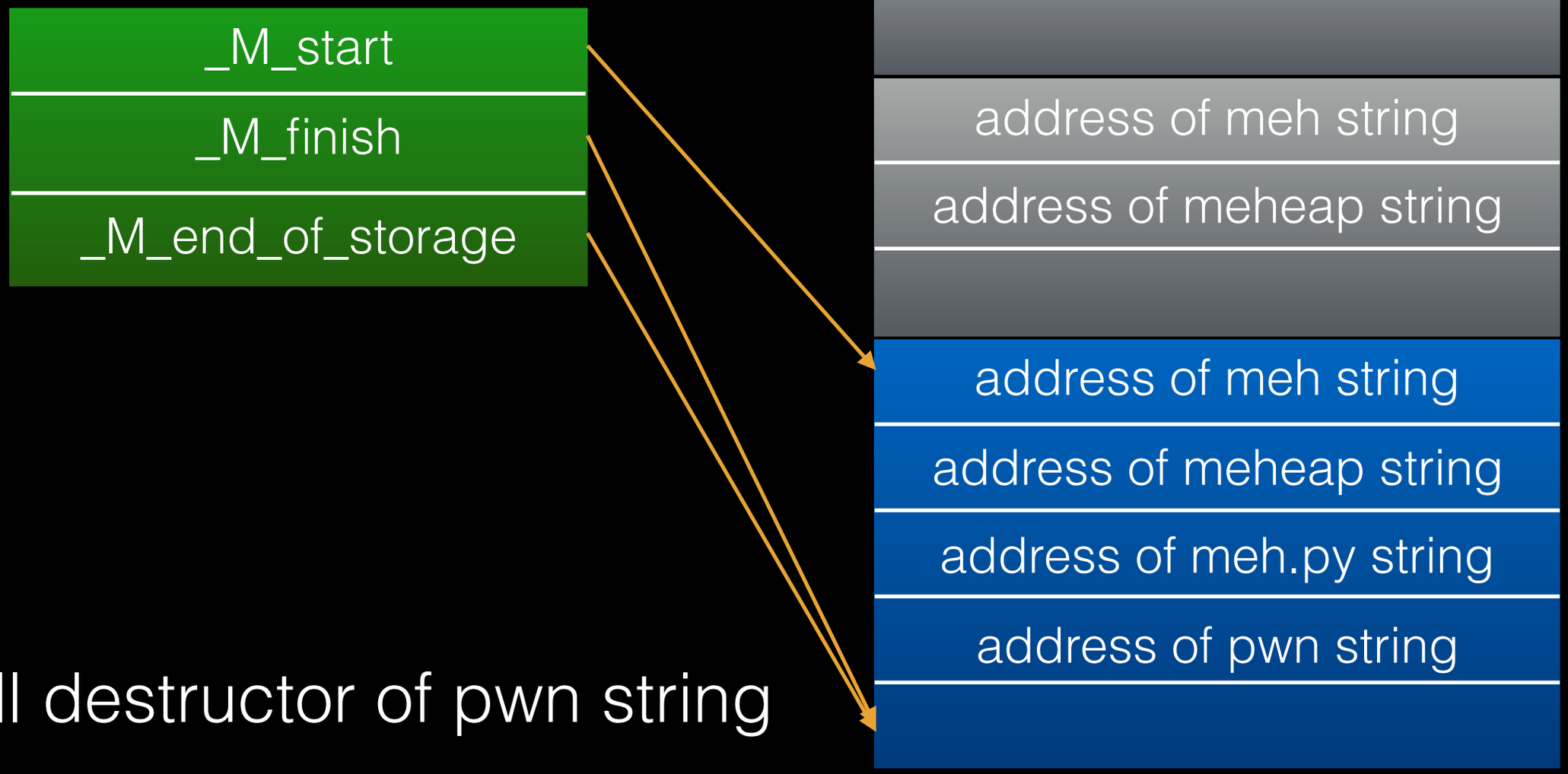
vec.push_back(“pwn”)



Vector & String

- Vector layout

vec.pop_back()

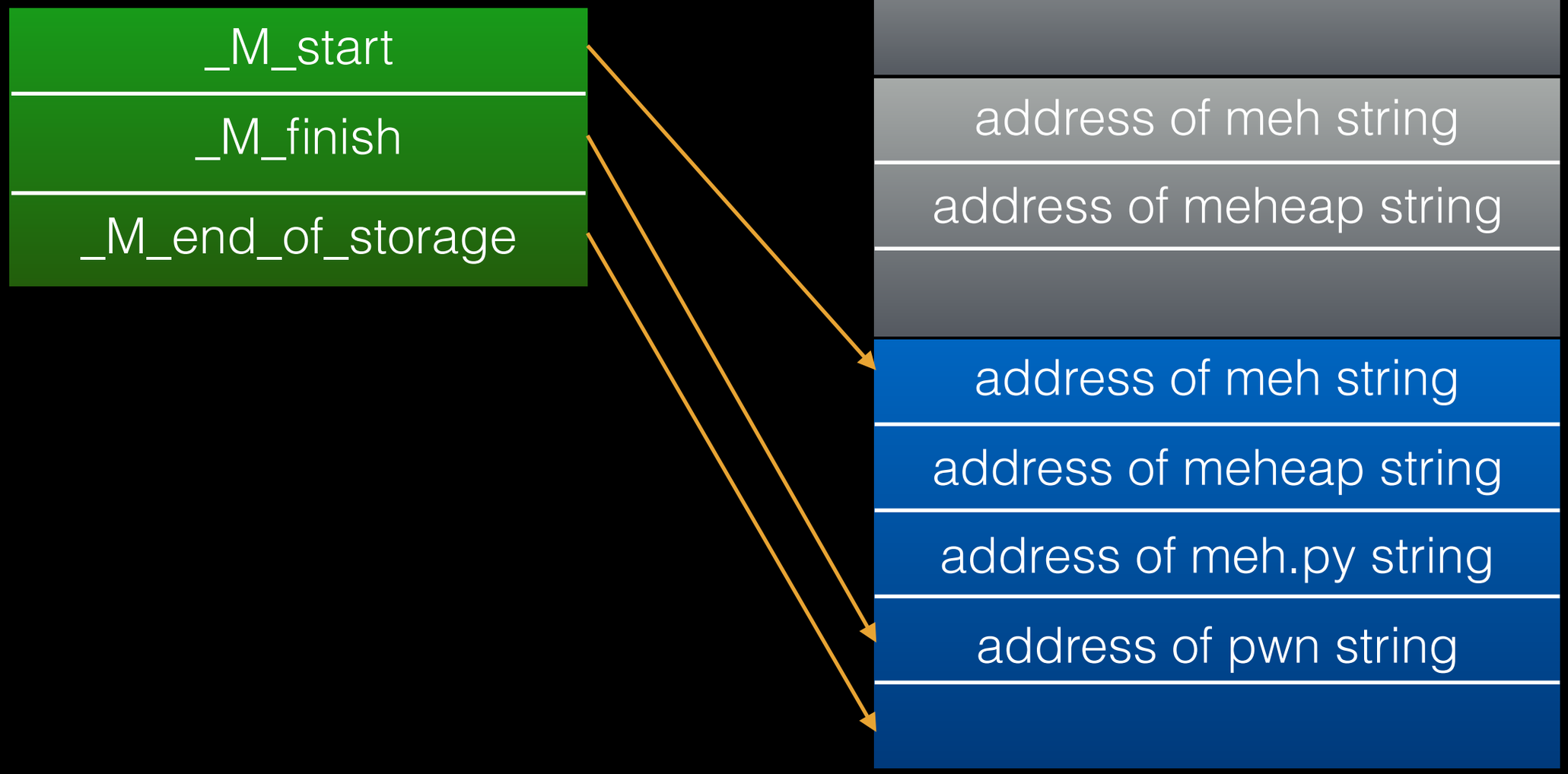


call destructor of pwn string

Vector & String

- Vector layout

  vec.pop_back()



Vector & String

- String
  - a dynamic char array
  - 比起以往的字串陣列更加安全，全部動態配置記憶體空間，減少一般 buffer overflow 的發生
  - 在給定 input 時，會不斷重新分配空間給 user 直到結束後，就會回傳適當的大小給 user
  - 有許多種實作方式，這邊介紹最常見的一種
    - g++ < 5

Vector & String

- String
  - member
    - size ：字串的長度
    - Capacity : 該 string 空間的容量
    - reference count : 引用計數
      - 只要有其他元素引用該字串就會增加
      - 如果其他元素不引用了，也會減少
        - 當 reference == 0 時就會，將空間 delete 掉
    - value : 存放字串內容

Vector & String

- String
  - member function
    - length() : string 大小
    - capacity() : 目前 string 空間容量
    - c_str() : Get C string equivalent
    - ……

Vector & String

- String layout

  string str

  cin >> str

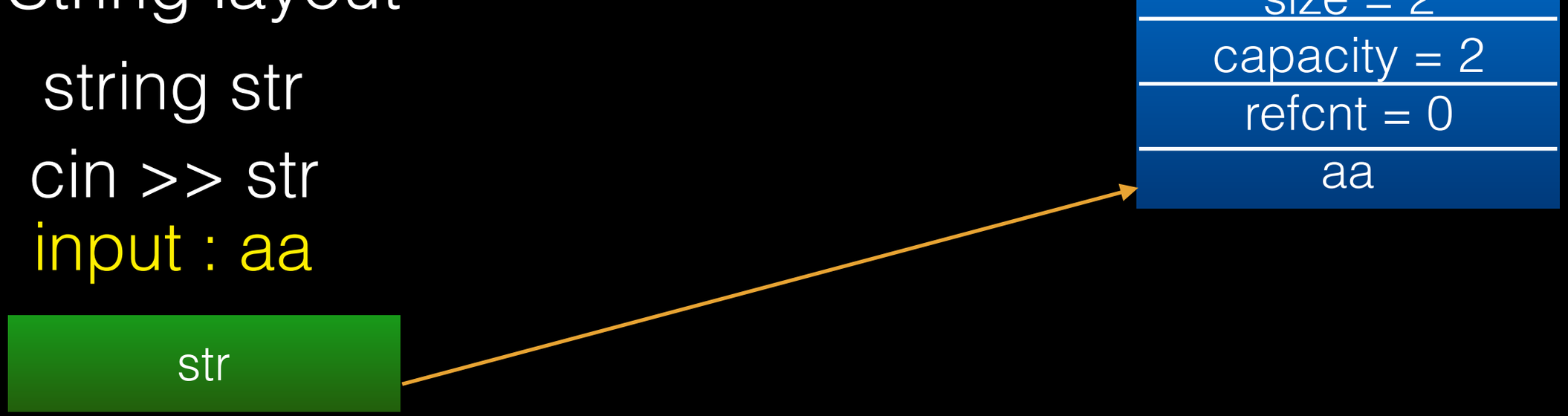
Vector & String

- String layout

  string str

  cin >> str

  input : aa



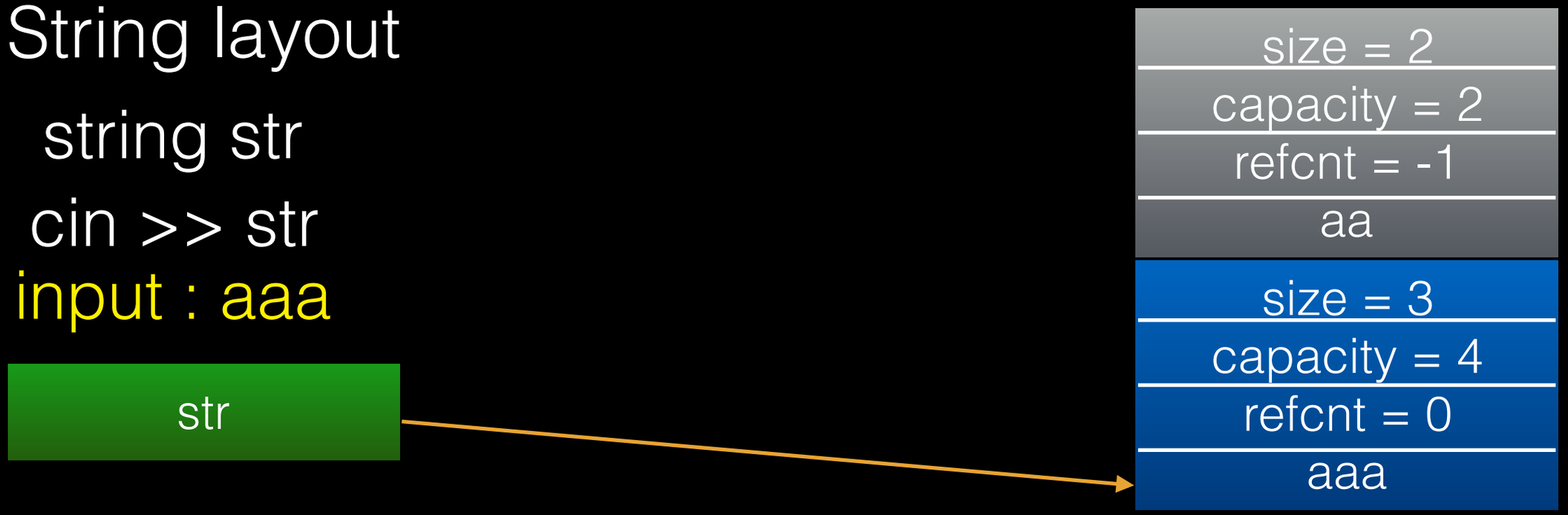
Vector & String

- String layout

  string str

  cin >> str

  input : aaa



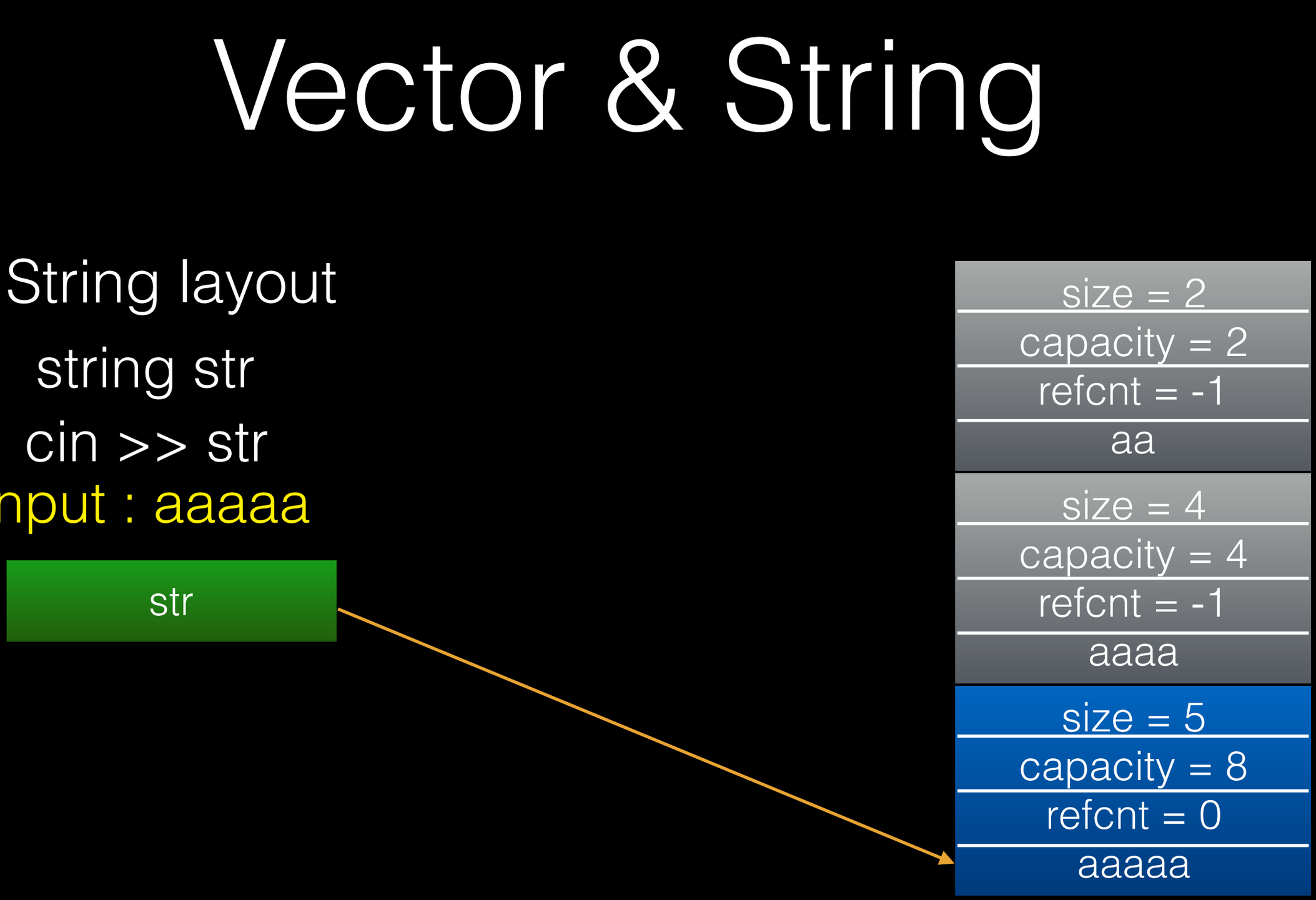
Vector & String

- String layout

  string str

  cin >> str

input : aaaaa



Vector & String

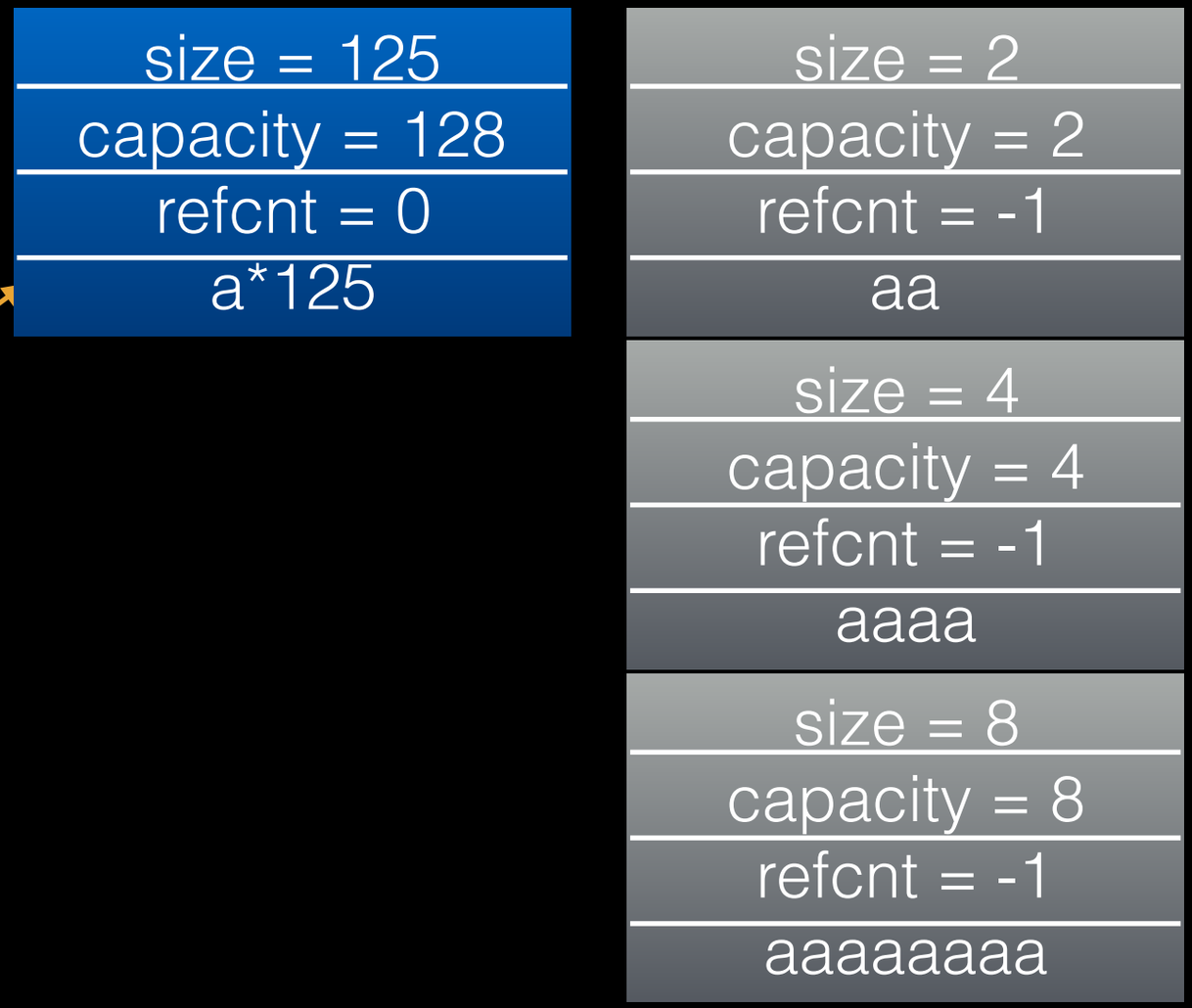
- String layout

string str

cin >> str

input : a*125

str

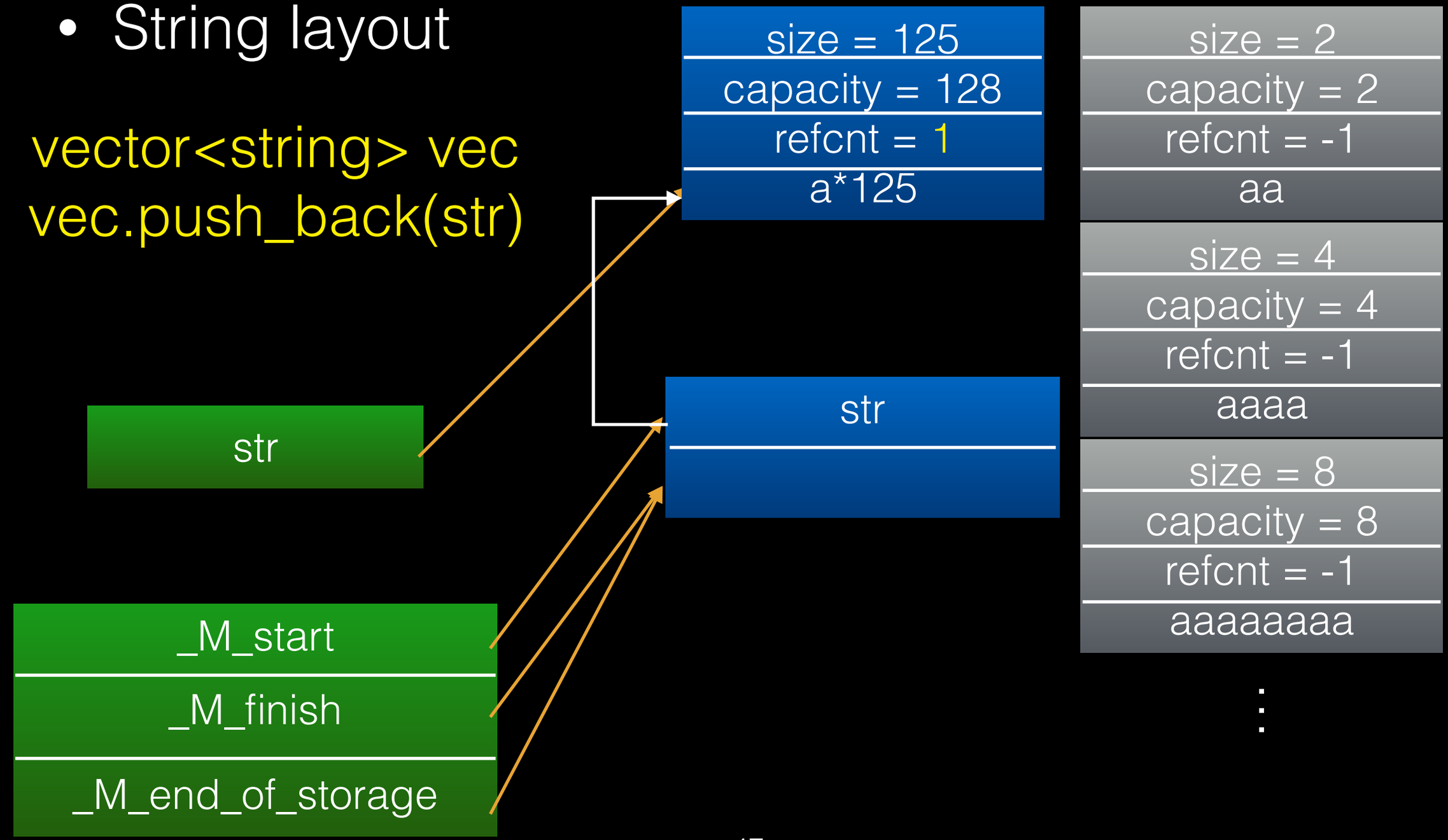


依此類推 capacity 會
不斷以二的指數倍增長
直到 input 結束

Vector & String

- String layout

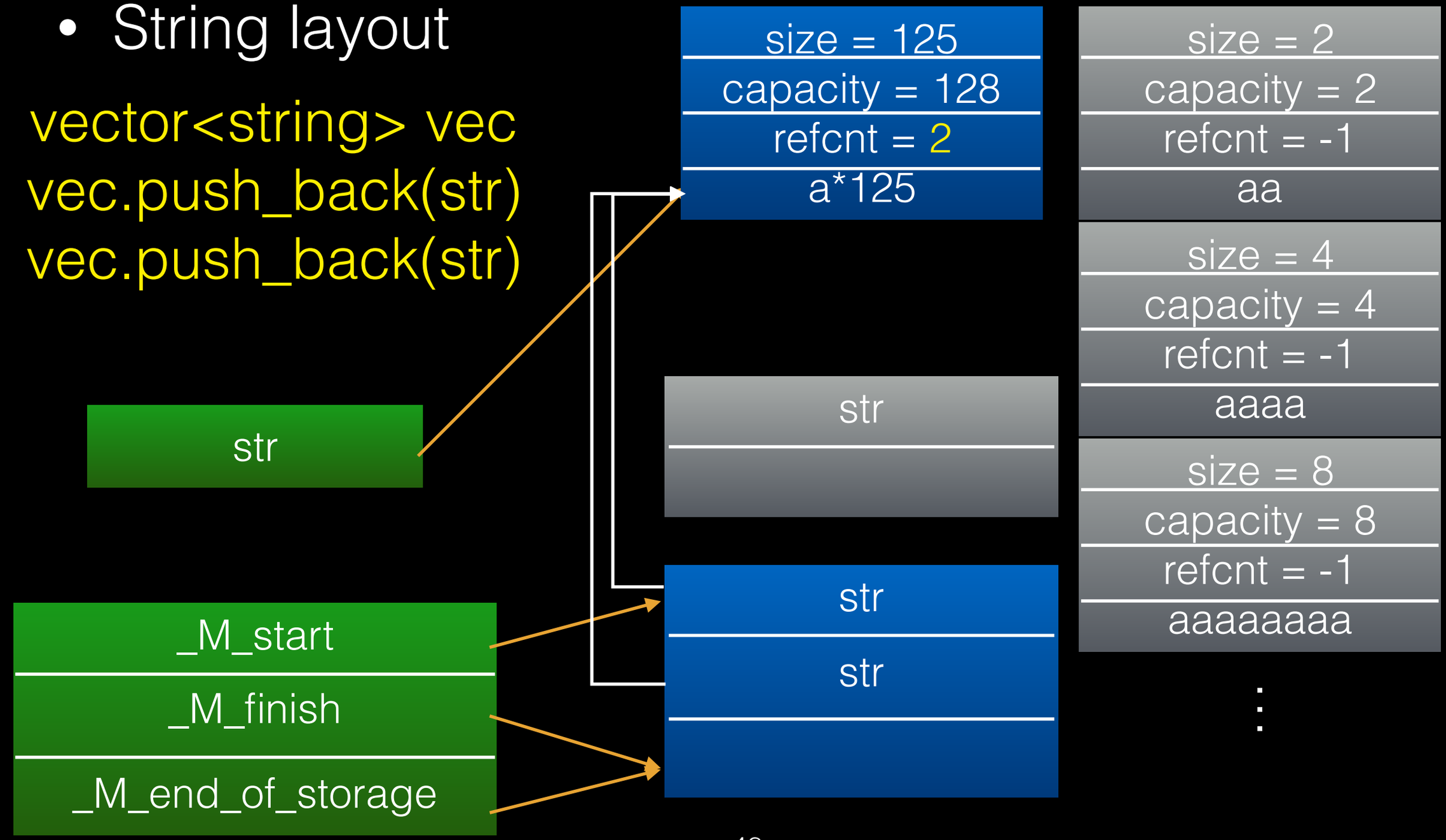
vector<string> vec
vec.push_back(str)



Vector & String

- String layout

vector<string> vec
vec.push_back(str)
vec.push_back(str)

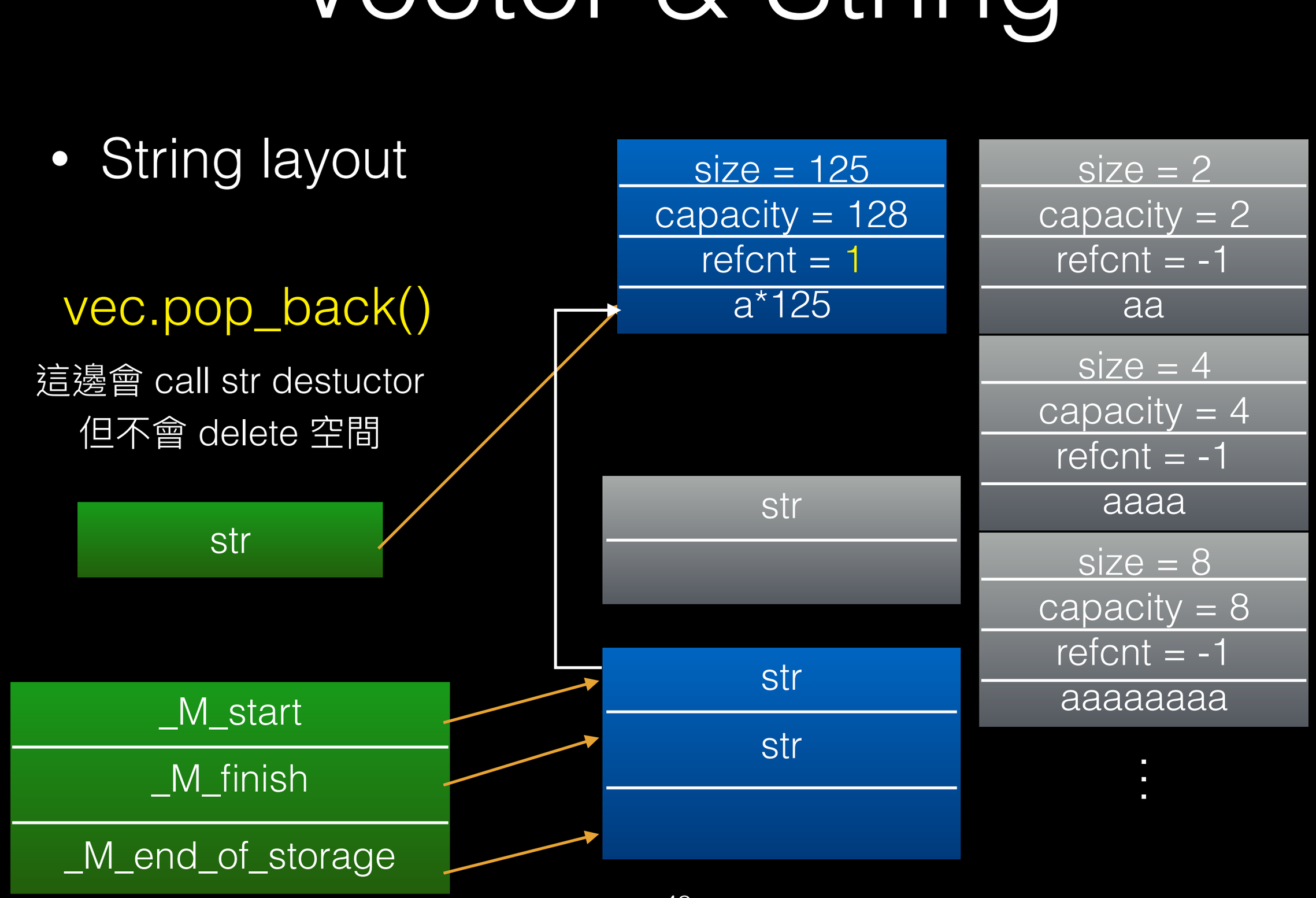


Vector & String

- String layout

vec.pop_back()

這邊會 call str destuctor
但不會 delete 空間

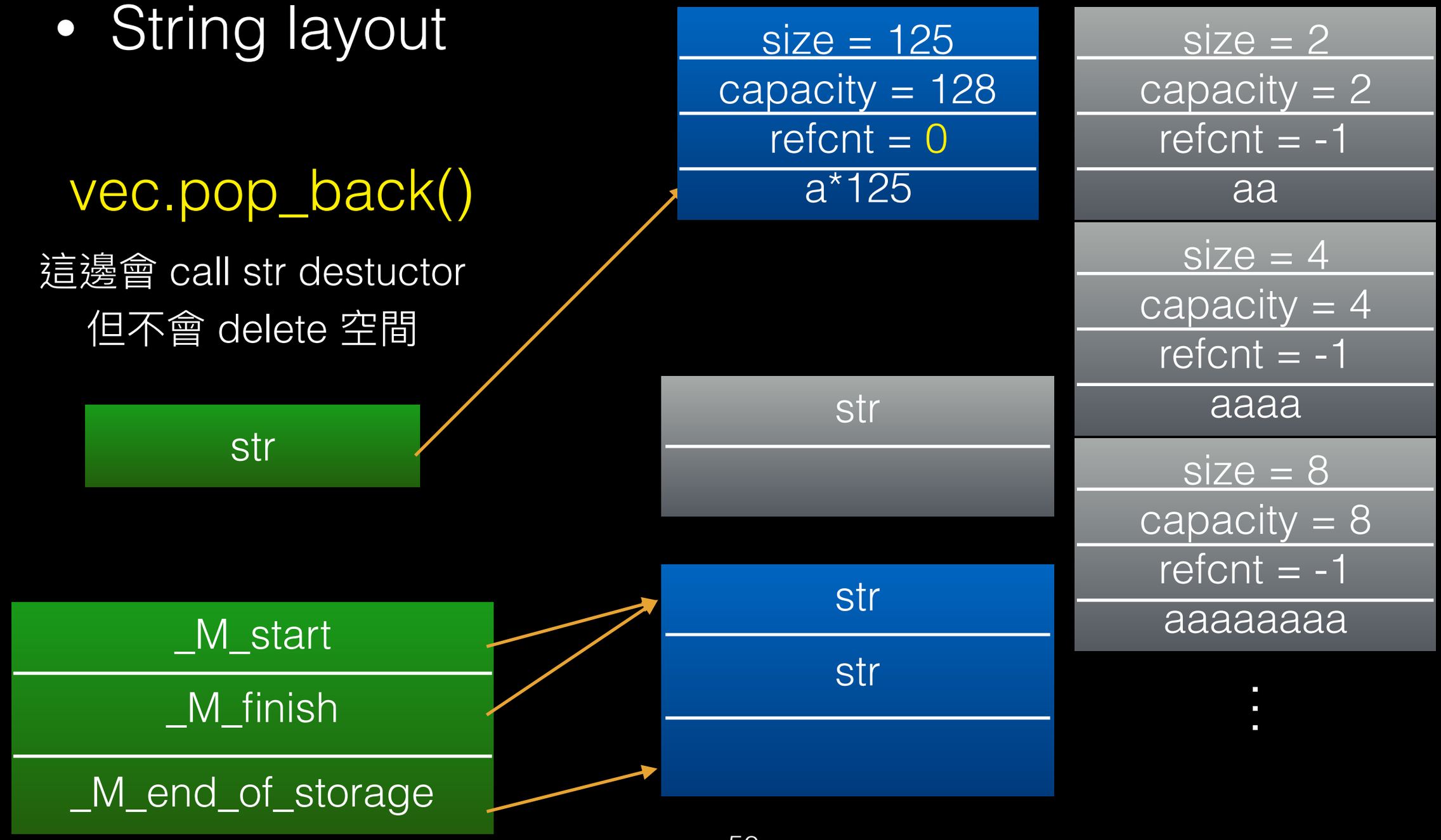


Vector & String

- String layout

vec.pop_back()

這邊會 call str destuctor
但不會 delete 空間



Vector & String

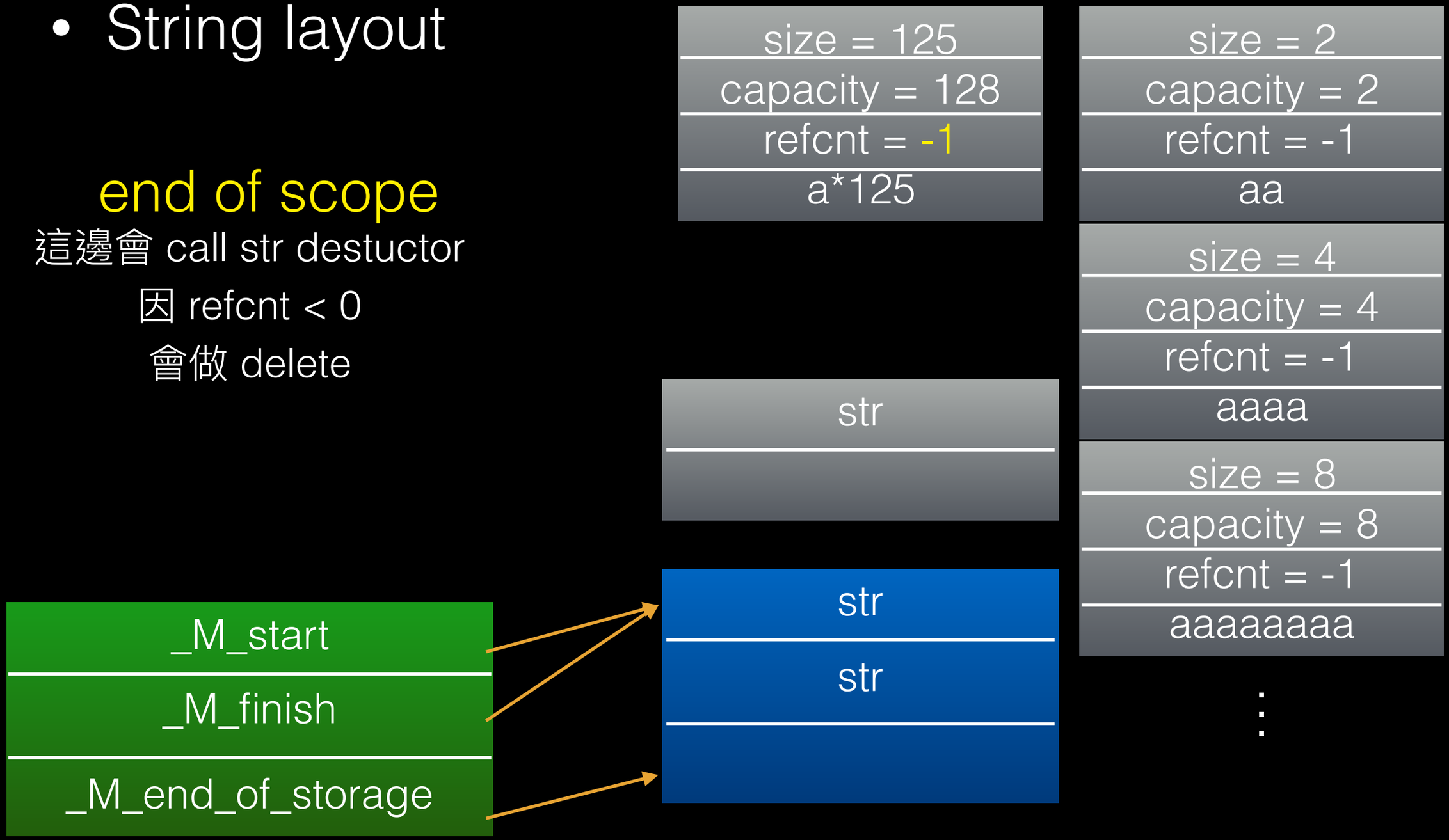
- String layout

end of scope

這邊會 call str destuctor

因 refcnt < 0

會做 delete



Vector & String

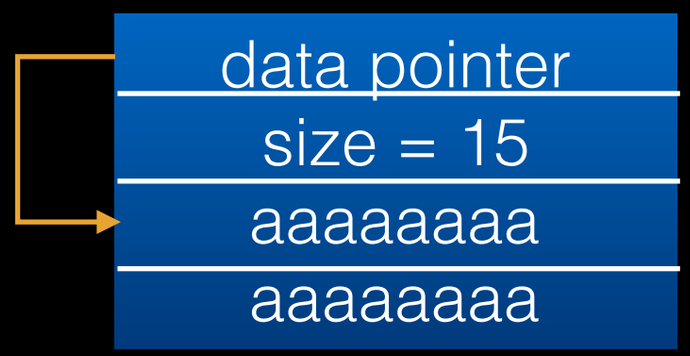
- String
  - g++ > 5 之後取消了 Copy-on-Write 機制
- 所以少掉了 reference count 這個欄位
- 在 data length
  - <= 15 時會用 local buffer
  - > 15 時則會在 heap allocate 空間給 data 使用

Vector & String

- String
  - data pointer : 指向 data 位置
  - size : 分配出去 string 的長度
  - union
    - local buffer : 在 size <= 15 時會直接把這欄位拿來存 data
    - allocated capacity : size > 15 時會拿來紀錄 capcity

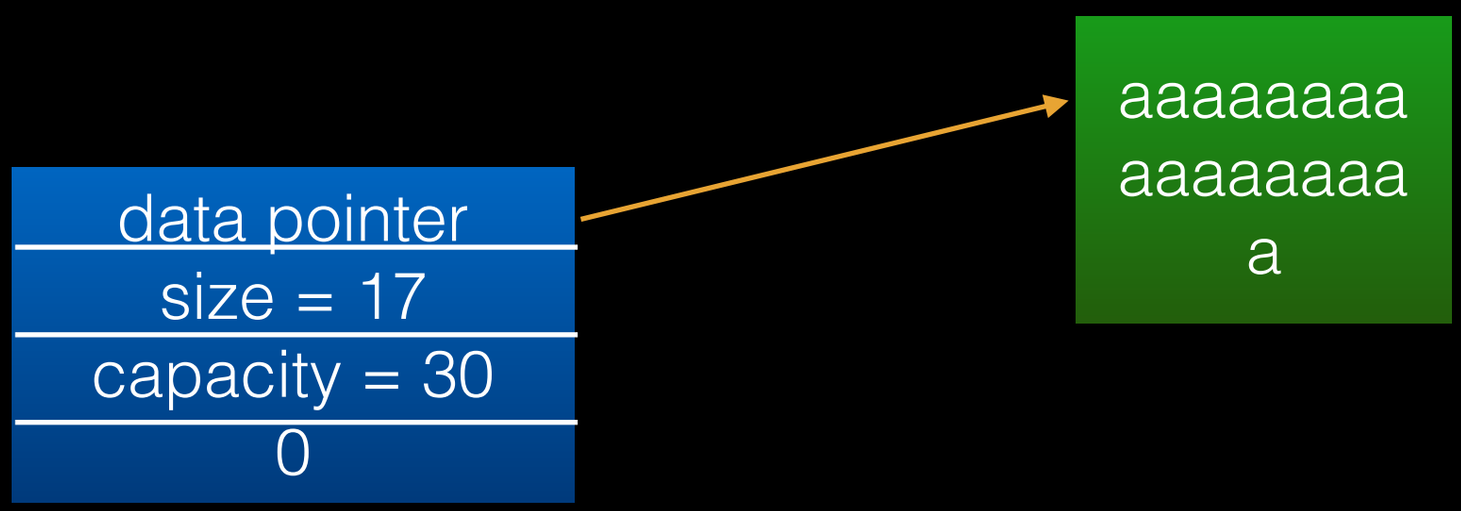
Vector & String

- String
  - size < 15



Vector & String

- String
  - size > 15



Outline

- Name Mangling
- Virtual function table
- Vector & String
- New & delete
- Copy constructor & assignment operator

New & Delete

- 在 c++ 預設的情況下 ，new/delete 的最底層實作依舊是靠 malloc/free 去處理記憶體管理
  - 在 c++ 中，記憶體配置池稱為 free store ，但預設情況下 free store 位置是在 heap
- 不過事實上 new/delete 是可以 overloading 的，也就是自行去做記憶體管理，另外最大差別就是new/delete 實際上會去 call constructor/destructor 而 malloc/free 只做單純的記憶體配置
- 因此盡量不要讓 malloc/free 與 new/delete 混用，不然可能會出現一些不可預期的狀況

New & Delete

- new 大致上流程
  - operator new
    - 與 malloc 類似，單純配置記憶體空間，但配置失敗會進入 exception
      而 malloc 則是返回 null ，有點像在 malloc 上一層 wrapper
  - constructor
- delete 大致上流程
  - destructor
  - operator delete
    - 與 free 類似，釋放一塊記憶體空間，有點像是在 free 上一層 wrapper

New & Delete

- 因此 new/delete 及 operator new / operator delete 也應該成對配對

- 記憶體函式配對

配置函式	解除函式
new	delete
new []	delete []
operator new	operator delete
operator new[]	operator delete[]
malloc	free

Outline

- Name Mangling
- Virtual function table
- Vector & String
- New & delete
- Copy constructor & assignment operator

What’s wrong in this code

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4 #include <string.h>
5 using namespace std;
6
7 class Stu{
8     public :
9         Stu():name(NULL),id(0){}
10        Stu(string str,int stuid){
11            name = new char[str.length()+1];
12            strcpy(name,str.c_str());
13            id = stuid;
14        }
15        void putinfo(){
16            cout << id << ":" <<  name << endl ;
17        }
18
19        ~Stu(){
20            delete[] name ;
21
22        }
23    private :
24        int id ;
25        char *name ;
26 };
27
28
29 int main(void){
30     vector<Stu> stulist ;
31     Stu student = Stu("orange",1337);
32     stulist.push_back(student);
33     stulist[0].putinfo();
34     return 0 ;
35 }
36
~
```

What’s wrong in this code

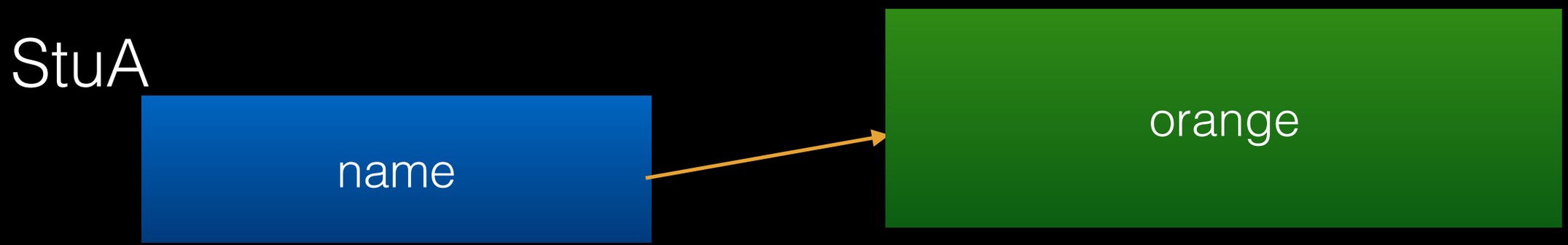
```
angelboy@ubuntu:~$ ./a.out
1337:orange
*** Error in `./a.out': double free or corruption (fasttop): 0x0000000001214040 ***
Aborted (core dumped)
```

Copy constructor & assignment operator

- shallow copy
  - 只做單純 pointer (value) 的複製，複製完後內容與原本的相同
- deep copy
  - 會在配置更多記憶體空間，包含 pointer 所指向的內容也都一併複製

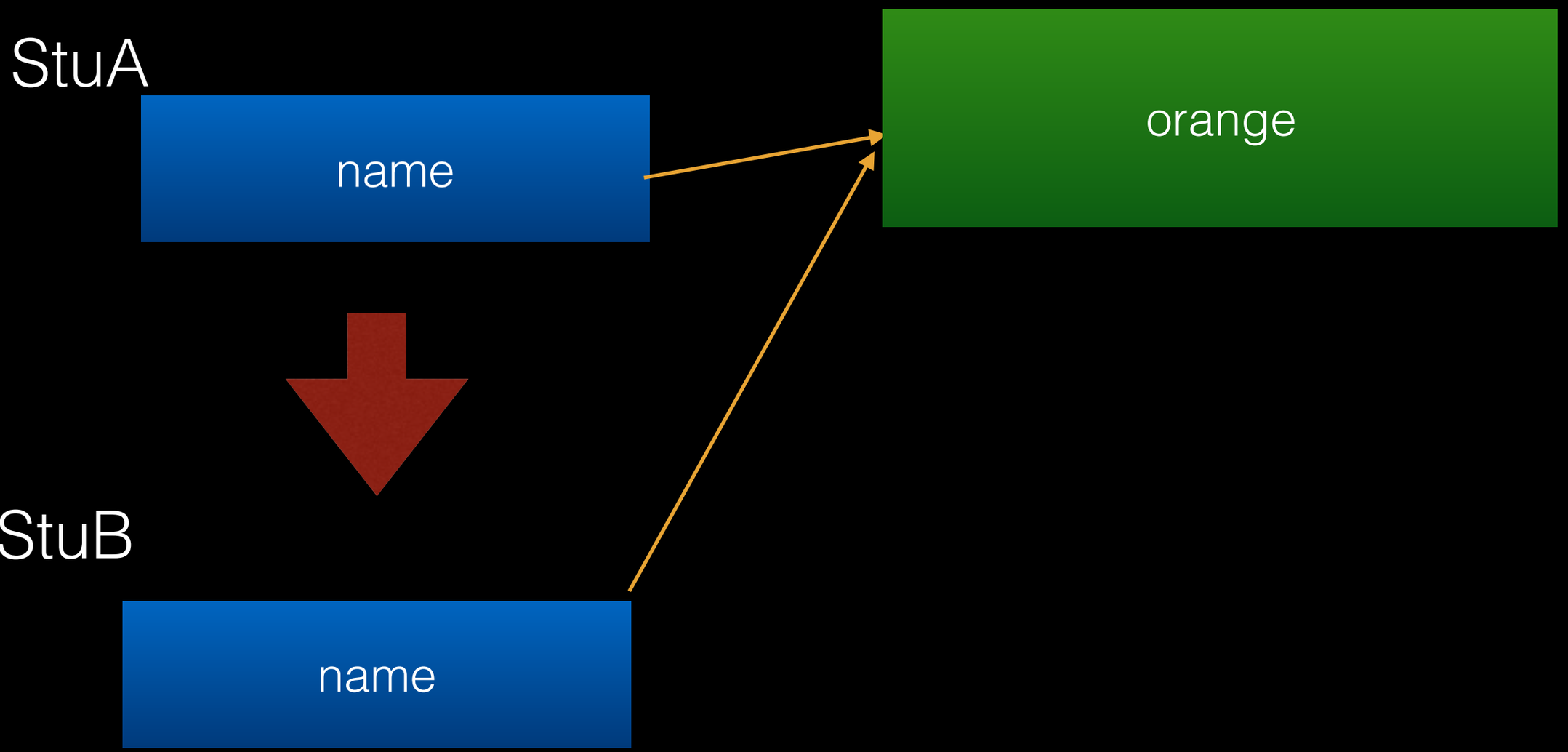
Copy constructor & assignment operator

- shallow copy



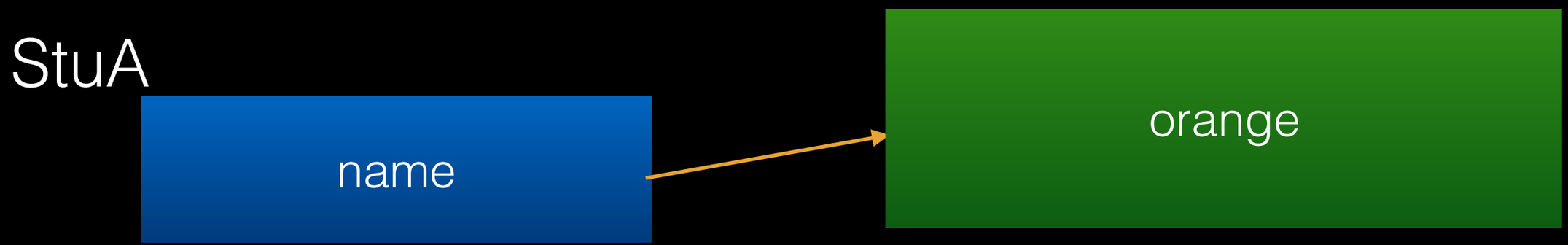
Copy constructor & assignment operator

- shallow copy



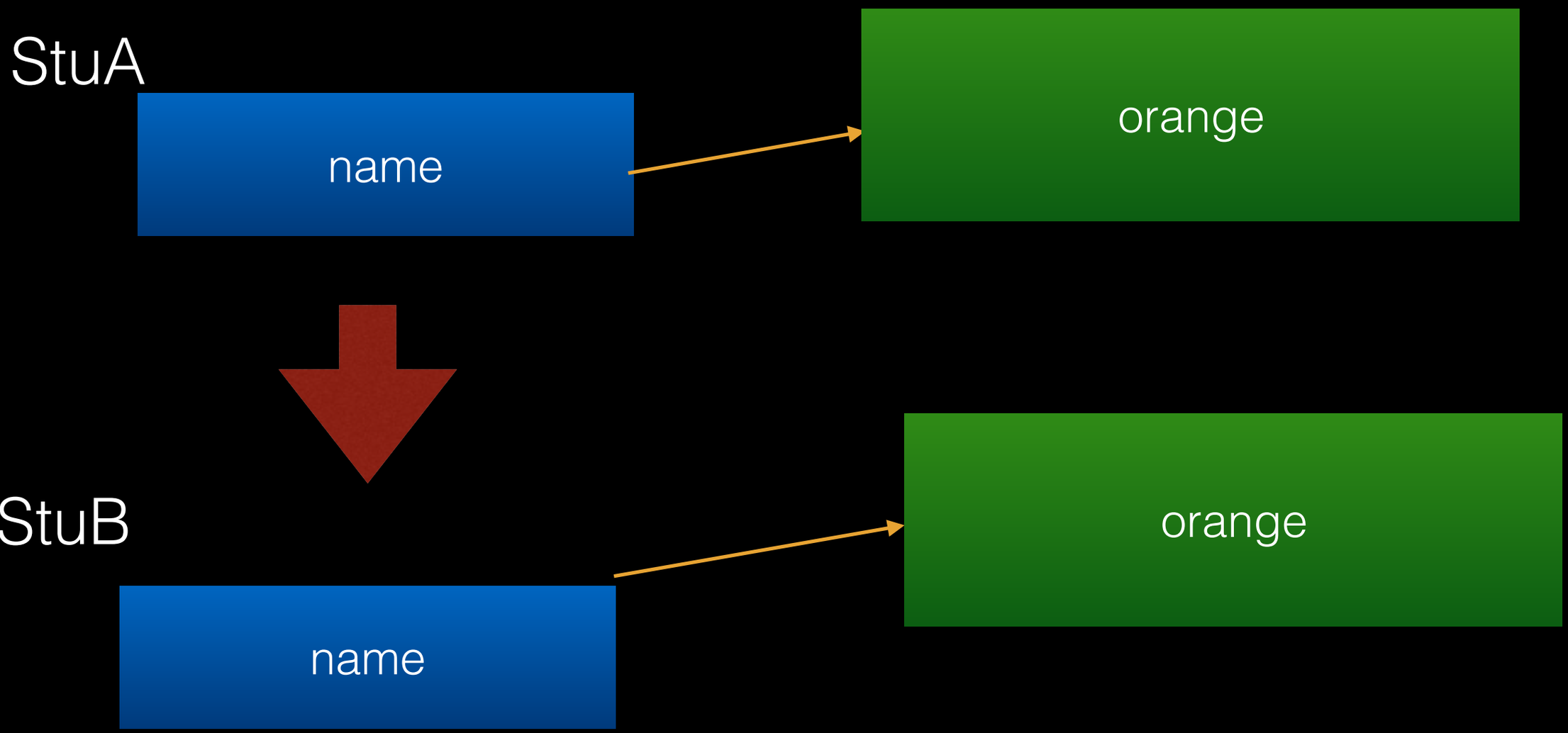
Copy constructor & assignment operator

- deep copy



Copy constructor & assignment operator

- deep copy



Copy constructor & assignment operator

- Copy constructor
  - c++ 在進行複製 object 會使用 copy constructor
    - 通常 class 的 member 有指標時，就需要自行去實作
- 若未特別定義則會使用 default copy constructor
  - 只做 shallow copy

Copy constructor & assignment operator

- Assignment operator
  - c++ 在進行 “=“ 這個 operator 時 object 會使用 assignment operator 這個 function 去 assign object 的值
    - 通常 class 的 member 有指標時，就需要自行去實作
- 若未特別定義則會使用 default assignment operator
  - 只做 shallow copy

Copy constructor & assignment operator

- 何時會使用 copy constructor
  - func(Stu stu)
  - return stu
  - vector 等 STL 容器
  - ....

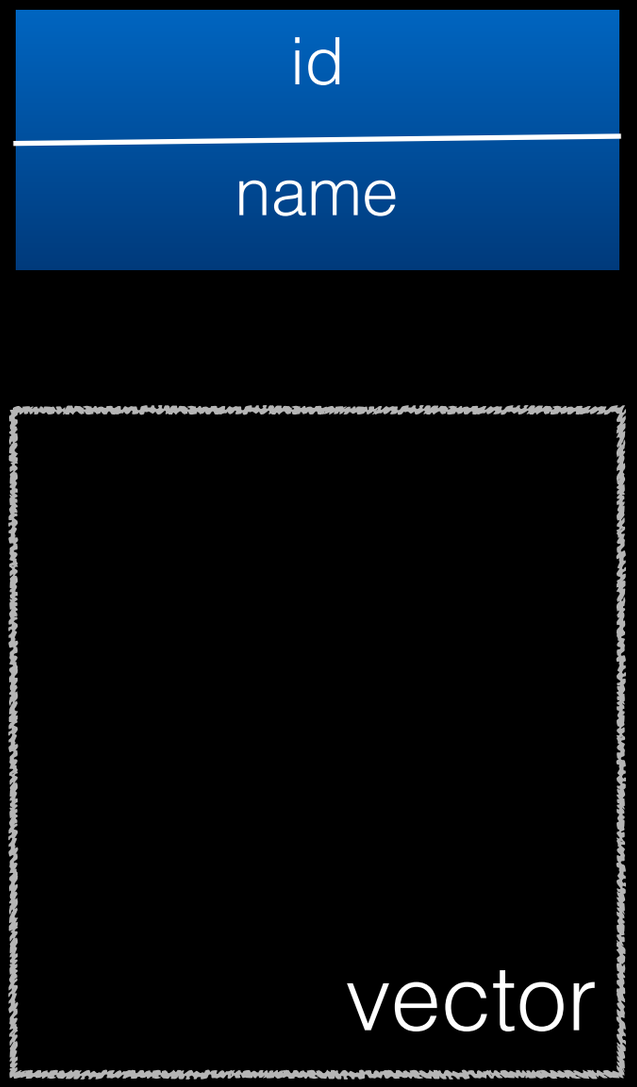
Copy constructor & assignment operator

- 何時會使用 assignment operator
  - stuA = stuB
  - vector 等 STL 容器
    - e.g. vector.erase()
  - …

Copy constructor & assignment operator

call constructor

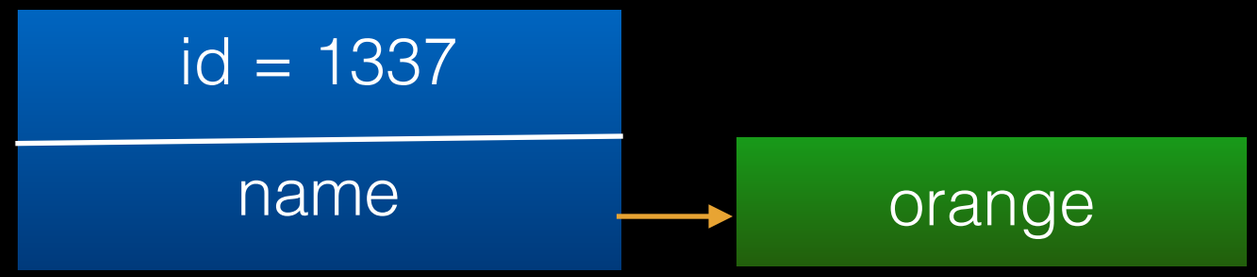
```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4 #include <string.h>
5 using namespace std;
6
7 class Stu{
8     public :
9         Stu():name(NULL),id(0){}
10        Stu(string str,int stuid){
11            name = new char[str.length()+1];
12            strcpy(name,str.c_str());
13            id = stuid;
14        }
15        void putinfo(){
16            cout << id << ":" <<  name << endl ;
17        }
18
19        ~Stu(){
20            delete[] name ;
21
22        }
23    private :
24        int id ;
25        char *name ;
26 };
27
28
29 int main(void){
30    vector<Stu> stulist ;
31    Stu student = Stu("orange",1337);
32    stulist.push_back(student);
33    stulist[0].putinfo();
34    return 0 ;
35 }
36
~
```

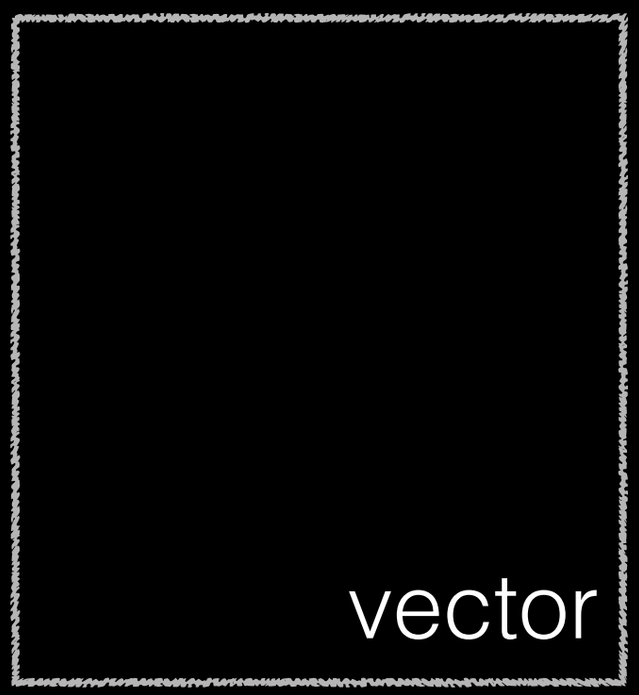


Copy constructor & assignment operator

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4 #include <string.h>
5 using namespace std;
6
7 class Stu{
8     public :
9         Stu():name(NULL),id(0){}
10        Stu(string str,int stuid){
11            name = new char[str.length()+1];
12            strcpy(name,str.c_str());
13            id = stuid;
14        }
15        void putinfo(){
16            cout << id << ":" <<  name << endl ;
17        }
18
19        ~Stu(){
20            delete[] name ;
21
22        }
23    private :
24        int id ;
25        char *name ;
26 };
27
28
29 int main(void){
30    vector<Stu> stulist ;
31    Stu student = Stu("orange",1337);
32    stulist.push_back(student);
33    stulist[0].putinfo();
34    return 0 ;
35 }
36
~
```

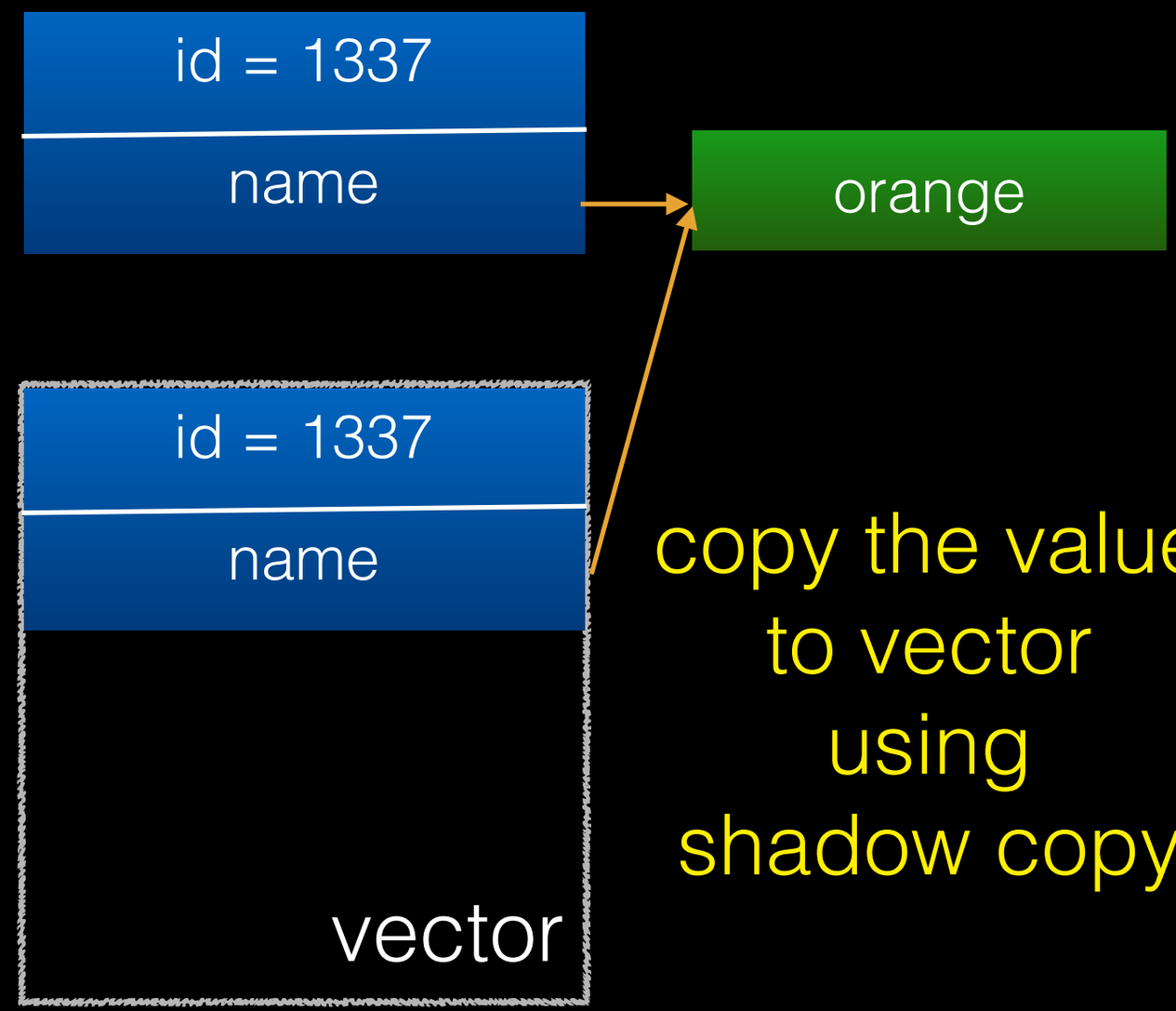
new char [str.length() + 1]





Copy constructor & assignment operator

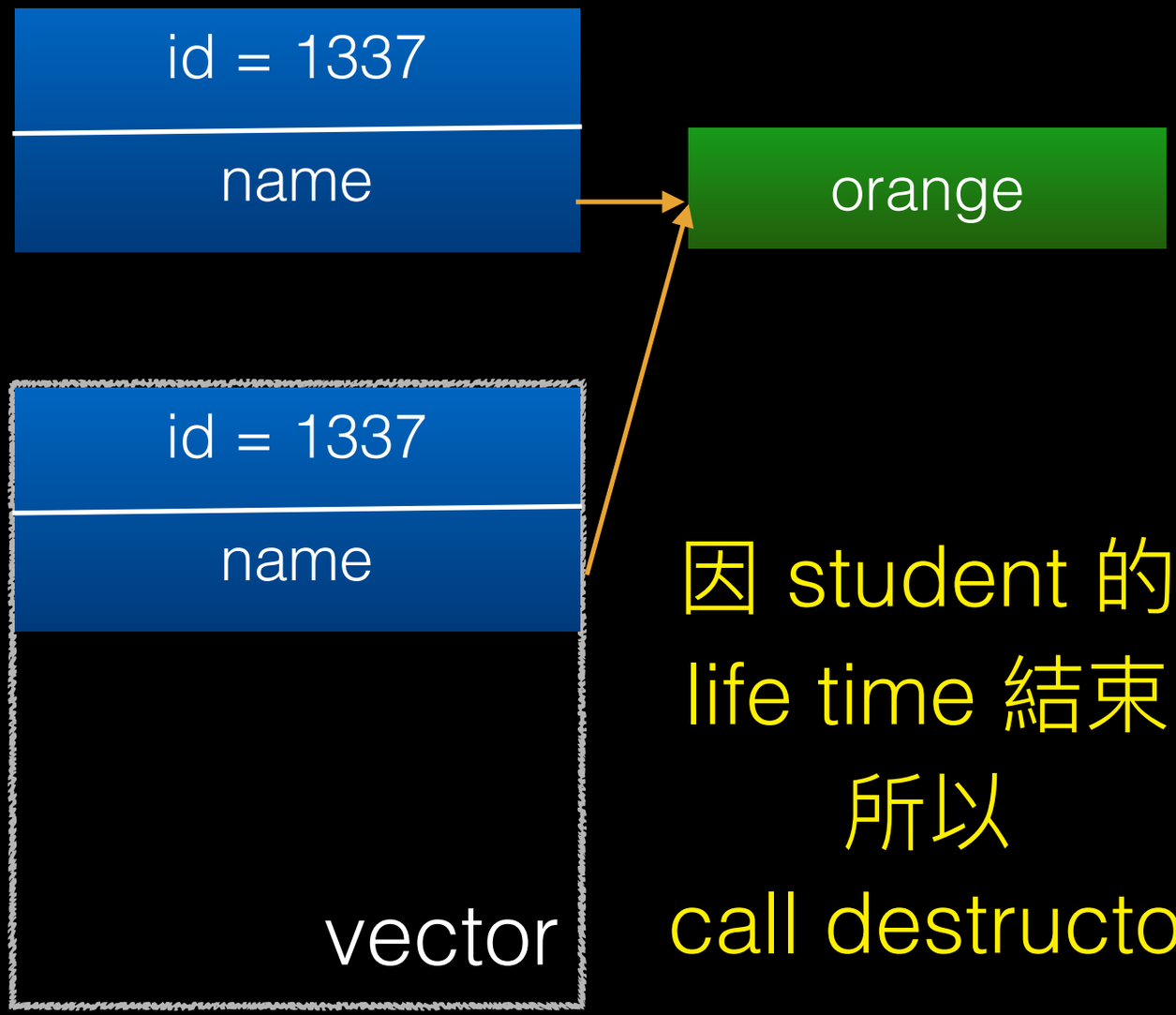
```cpp
#include <iostream>
#include <string>
#include <vector>
#include <string.h>
using namespace std;

class Stu{
    public :
        Stu():name(NULL),id(0){}
        Stu(string str,int stuid){
            name = new char[str.length()+1];
            strcpy(name,str.c_str());
            id = stuid;
        }
        void putinfo(){
            cout << id << ":" <<  name << endl ;
        }

        ~Stu(){
            delete[] name ;

        }
    private :
        int id ;
        char *name ;
};


int main(void){
    vector<Stu> stulist ;
    Stu student = Stu("orange",1337);
    stulist.push_back(student);
    stulist[0].putinfo();
    return 0 ;
}

```



Copy constructor & assignment operator

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <string.h>
using namespace std;

class Stu{
    public :
        Stu():name(NULL),id(0){}
        Stu(string str,int stuid){
            name = new char[str.length()+1];
            strcpy(name,str.c_str());
            id = stuid;
        }
        void putinfo(){
            cout << id << ":" <<  name << endl ;
        }

        ~Stu(){
            delete[] name ;

        }
    private :
        int id ;
        char *name ;
};


int main(void){
    vector<Stu> stulist ;
    Stu student = Stu("orange",1337);
    stulist.push_back(student);
    stulist[0].putinfo();
    return 0 ;
}
```

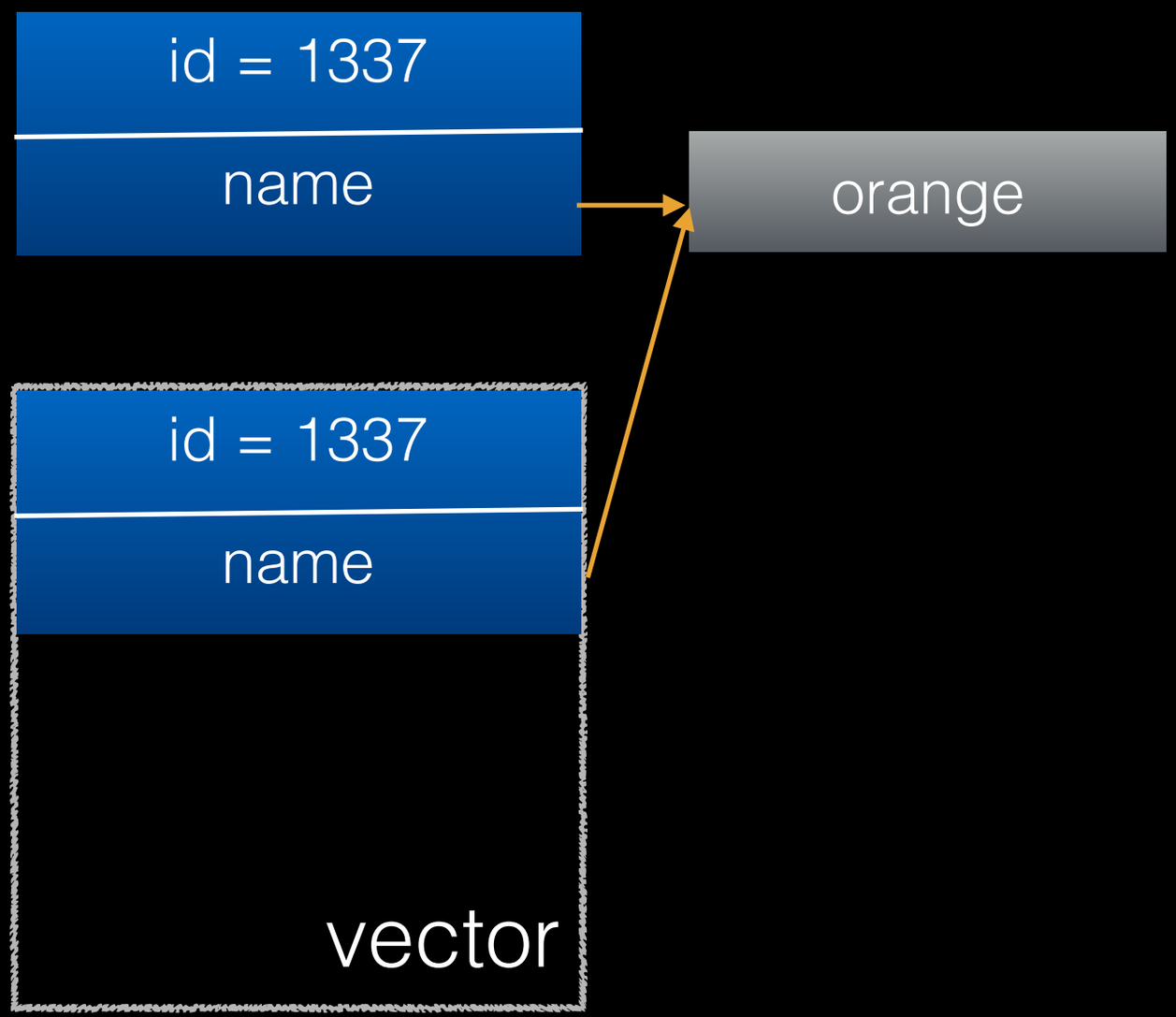
~Stu()



因 student 的 life time 結束
所以
call destructor

Copy constructor & assignment operator

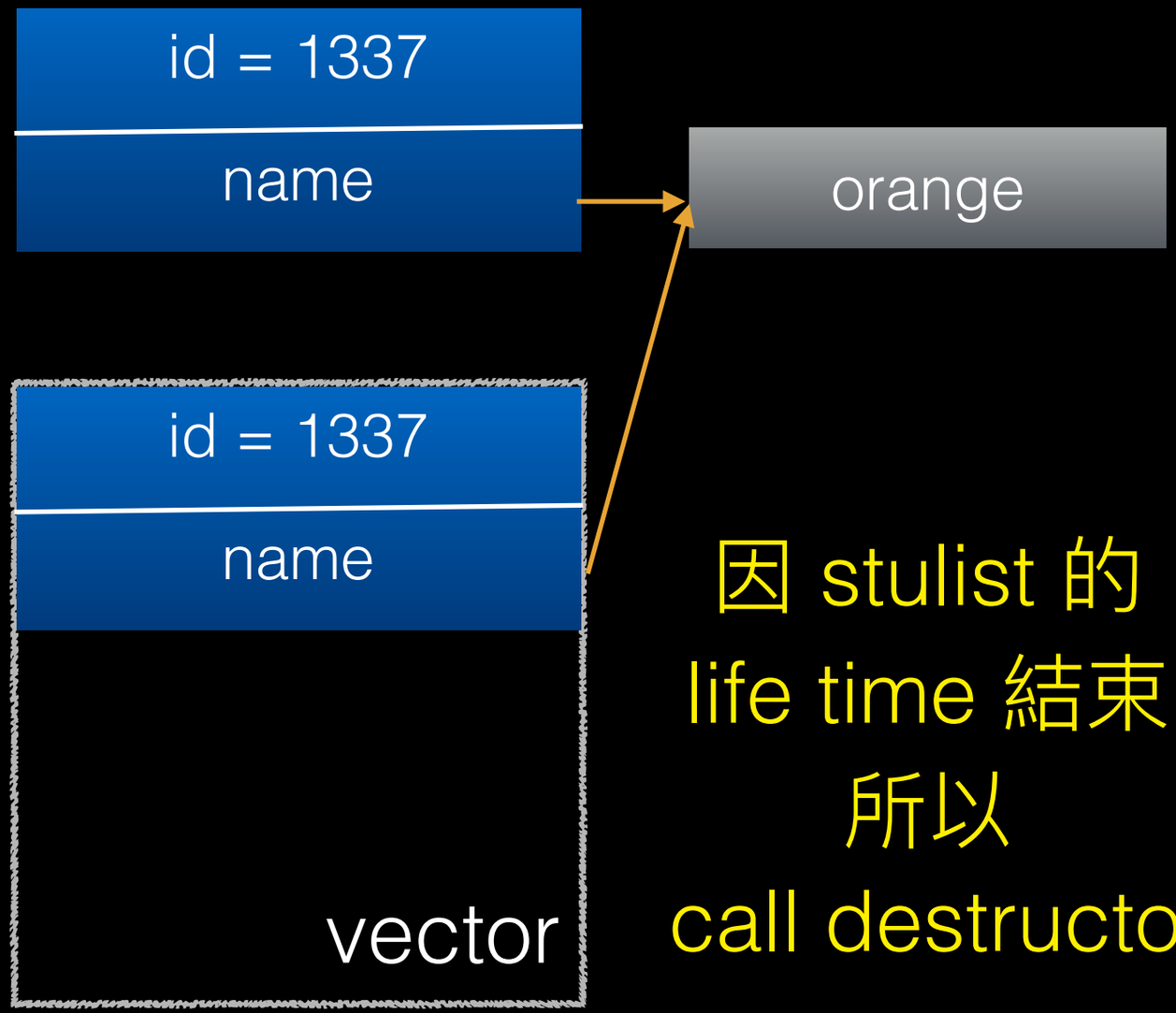
```cpp
#include <iostream>
#include <string>
#include <vector>
#include <string.h>
using namespace std;

class Stu{
    public :
        Stu():name(NULL),id(0){}
        Stu(string str,int stuid){
            name = new char[str.length()+1];
            strcpy(name,str.c_str());
            id = stuid;
        }
        void putinfo(){
            cout << id << ":" <<  name << endl ;
        }

        ~Stu(){
            delete[] name ;

        }
    private :
        int id ;
        char *name ;
};


int main(void){
    vector<Stu> stulist ;
    Stu student = Stu("orange",1337);
    stulist.push_back(student);
    stulist[0].putinfo();
    return 0 ;
}

```



Copy constructor & assignment operator

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <string.h>
using namespace std;

class Stu{
    public :
        Stu():name(NULL),id(0){}
        Stu(string str,int stuid){
            name = new char[str.length()+1];
            strcpy(name,str.c_str());
            id = stuid;
        }
        void putinfo(){
            cout << id << ":" <<  name << endl ;
        }

        ~Stu(){
            delete[] name ;

        }
    private :
        int id ;
        char *name ;
};


int main(void){
    vector<Stu> stulist ;
    Stu student = Stu("orange",1337);
    stulist.push_back(student);
    stulist[0].putinfo();
    return 0 ;
}
```

~vector()

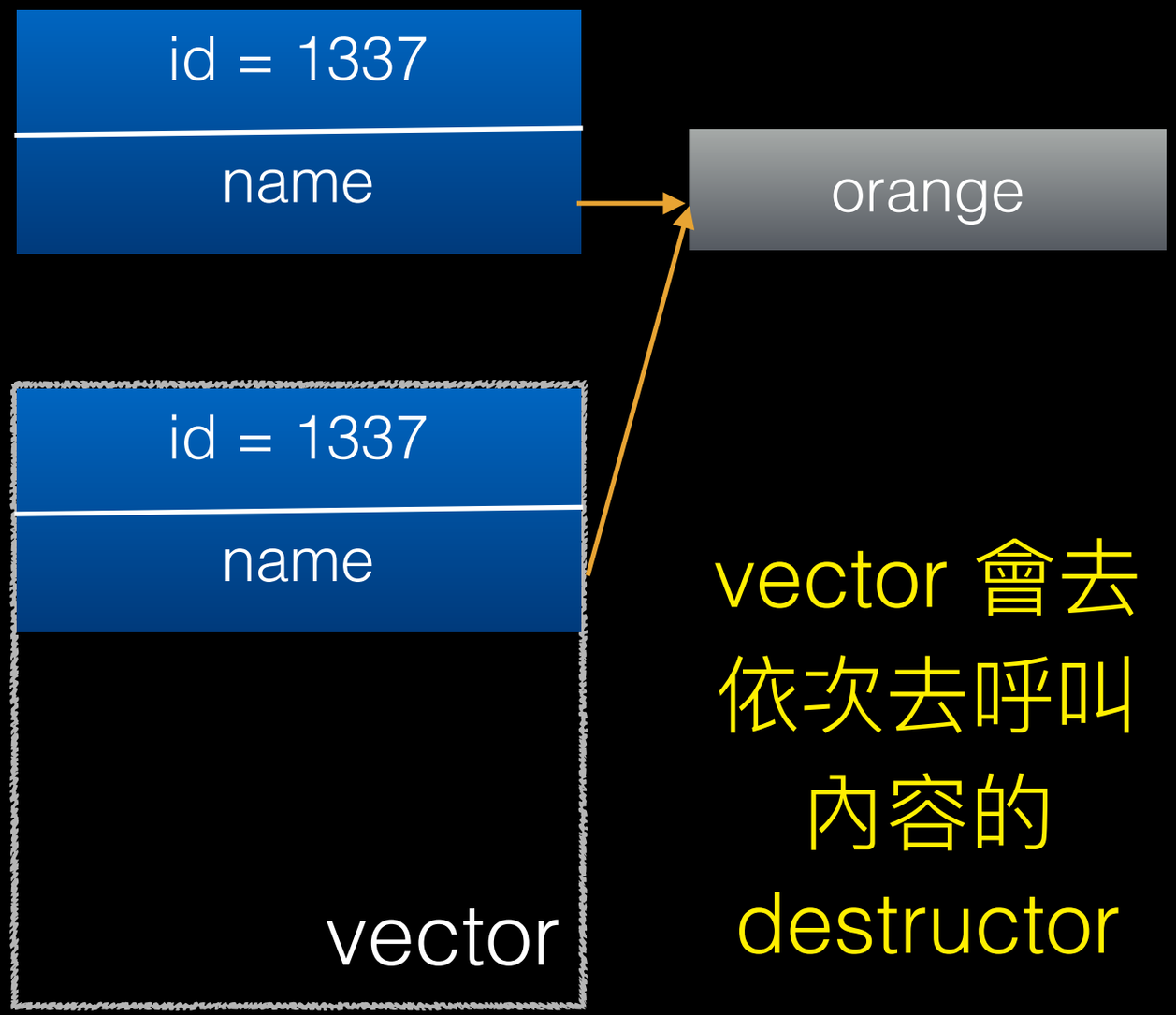


因 stulist 的 life time 結束 所以 call destructor

Copy constructor & assignment operator

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <string.h>
using namespace std;

class Stu{
    public :
        Stu():name(NULL),id(0){}
        Stu(string str,int stuid){
            name = new char[str.length()+1];
            strcpy(name,str.c_str());
            id = stuid;
        }
        void putinfo(){
            cout << id << ":" <<  name << endl ;
        }

        ~Stu(){
            delete[] name ;

        }
    private :
        int id ;
        char *name ;
};


int main(void){
    vector<Stu> stulist ;
    Stu student = Stu("orange",1337);
    stulist.push_back(student);
    stulist[0].putinfo();
    return 0 ;
}
```

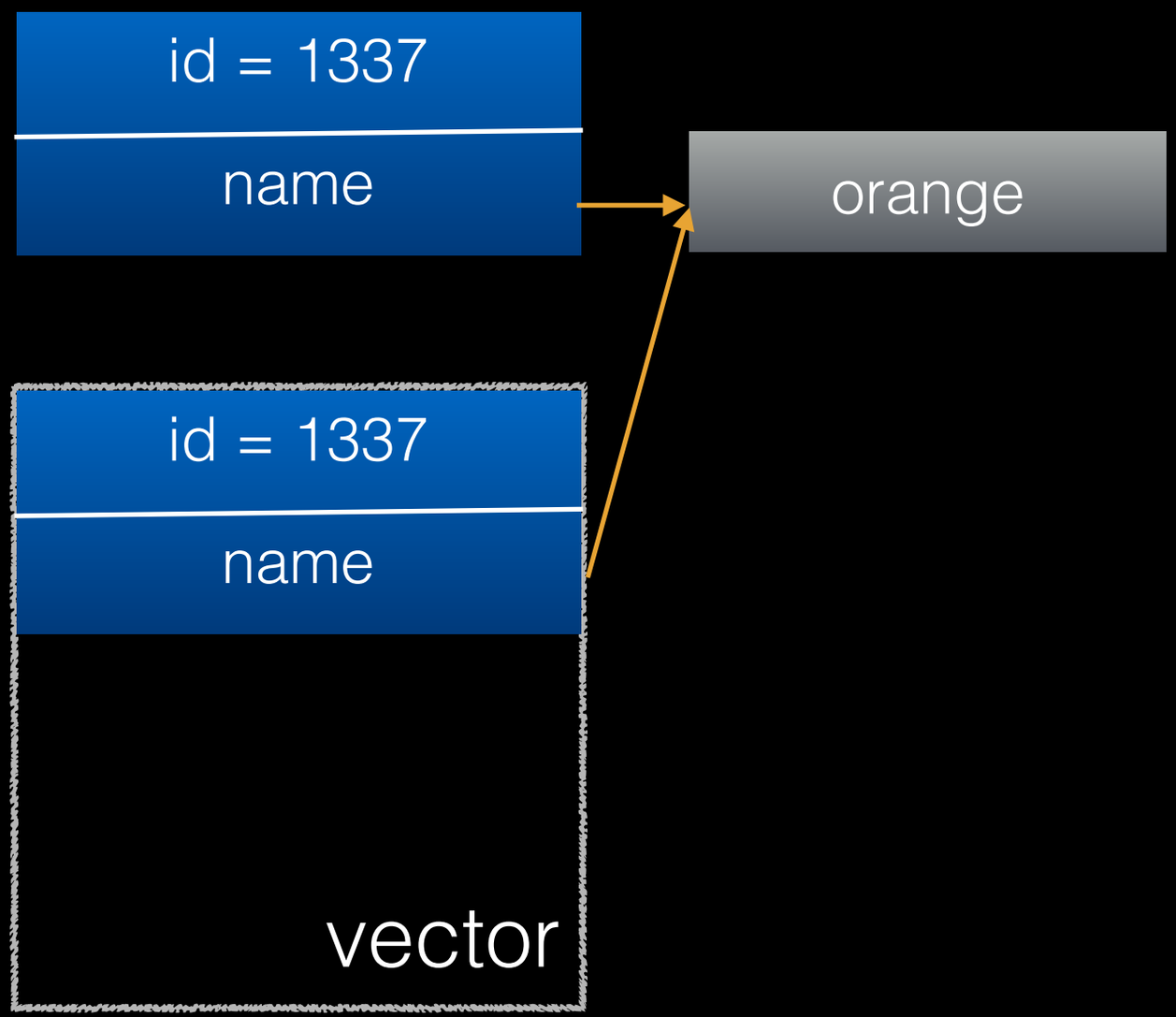
~Stu()



vector 會去依次去呼叫內容的 destructor

Copy constructor & assignment operator

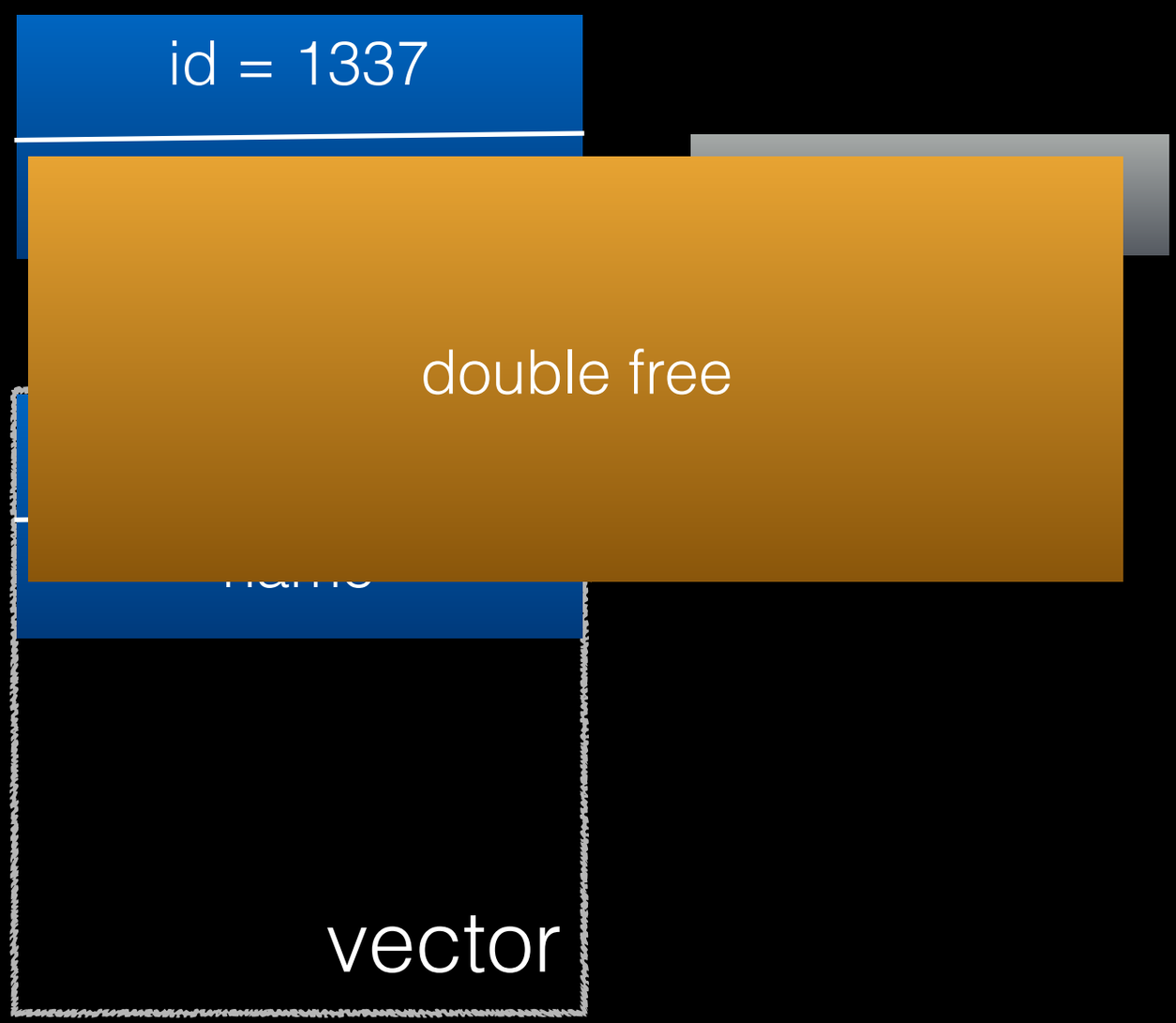
```cpp
#include <iostream>
#include <string>
#include <vector>
#include <string.h>
using namespace std;

class Stu{
    public :
        Stu():name(NULL),id(0){}
        Stu(string str,int stuid){
            name = new char[str.length()+1];
            strcpy(name,str.c_str());
            id = stuid;
        }
        void putinfo(){
            cout << id << ":" <<  name << endl ;
        }

        ~Stu(){
            delete[] name ;

        }
    private :
        int id ;
        char *name ;
};


int main(void){
    vector<Stu> stulist ;
    Stu student = Stu("orange",1337);
    stulist.push_back(student);
    stulist[0].putinfo();
    return 0 ;
}

```



Copy constructor & assignment operator

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4 #include <string.h>
5 using namespace std;
6
7 class Stu{
8     public :
9         Stu():name(NULL),id(0){}
10        Stu(string str,int stuid){
11            name = new char[str.length()+1];
12            strcpy(name,str.c_str());
13            id = stuid;
14        }
15        void putinfo(){
16            cout << id << ":" <<  name << endl ;
17        }
18
19        ~Stu(){
20            delete[] name ;
21
22        }
23    private :
24        int id ;
25        char *name ;
26 };
27
28
29 int main(void){
30     vector<Stu> stulist ;
31     Stu student = Stu("orange",1337);
32     stulist.push_back(student);
33     stulist[0].putinfo();
34     return 0 ;
35 }
36
~
```

delete [] name



Copy constructor & assignment operator

- 總結
  - 基本上 c++ 在只要做任何複製的動作時通常都會去使用 copy constructor 或者是 assignment operator
  - 所以基本上物件只要有 pointer 都盡量養成習慣去定義 copy constructor 跟 assignment opeator

Lab 15

- zoo

Q & A