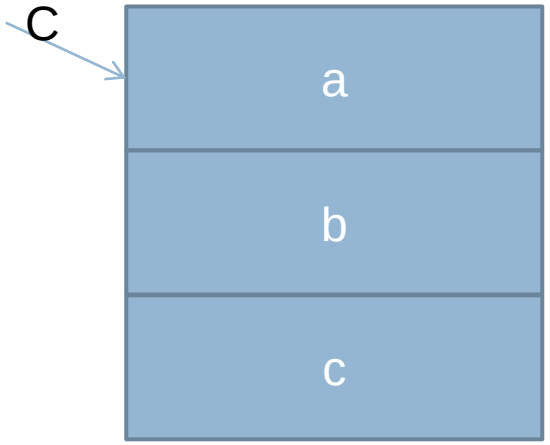
VIACNÁSOBNÉ DEDENIE

Príklad

```
class A {
   int a;
};

class B {
   int b;
};

class C : public A, public B {
   int c;
};
```

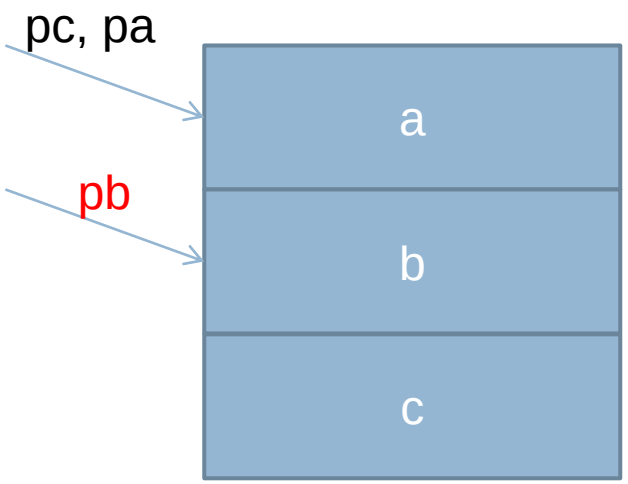


Dôsledok 1

- Smerník na rozšírenú triedu už nie je súčasne smerníkom na základnú triedu. (Smerník na typ C nemôže byť súčasne považovaný aj za smerník na A aj na B).
- Zmena typu smerníka už nie je len syntaktický cukor (t.j. zmena interpretácie čísla v rámci kompilátora), ale vyžaduje vykonanie aritmetických operácií v run-time (generovanie kódu).

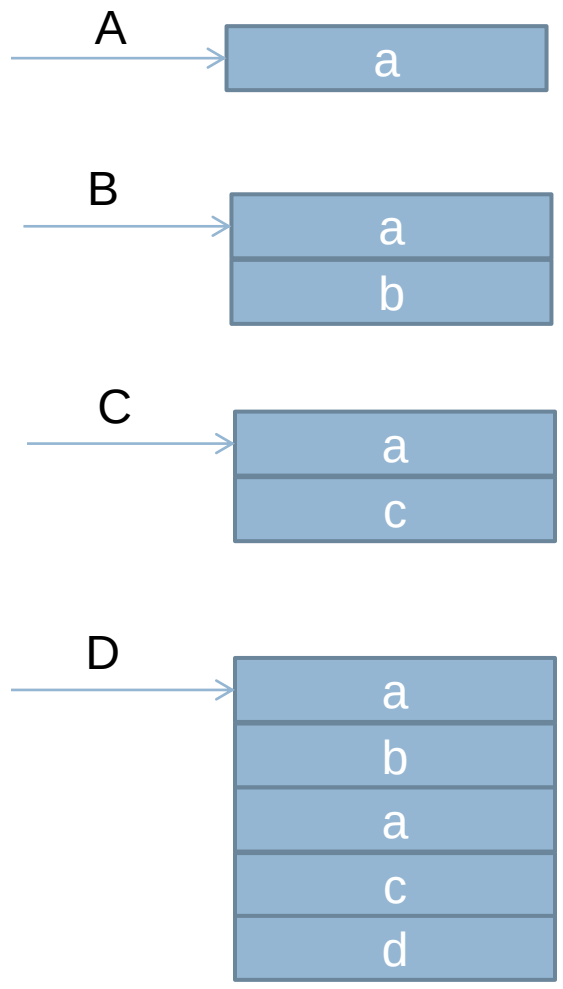
Príklad

```
class A {
  int a;
};

class B {
  int b;
};

class C : public A, public B {
  int c;
};

int main() {
  A *pa;  B *pb;
  C c, *pc;
  pc = &c;
  pa = pc;
  pb = pc;
}
```



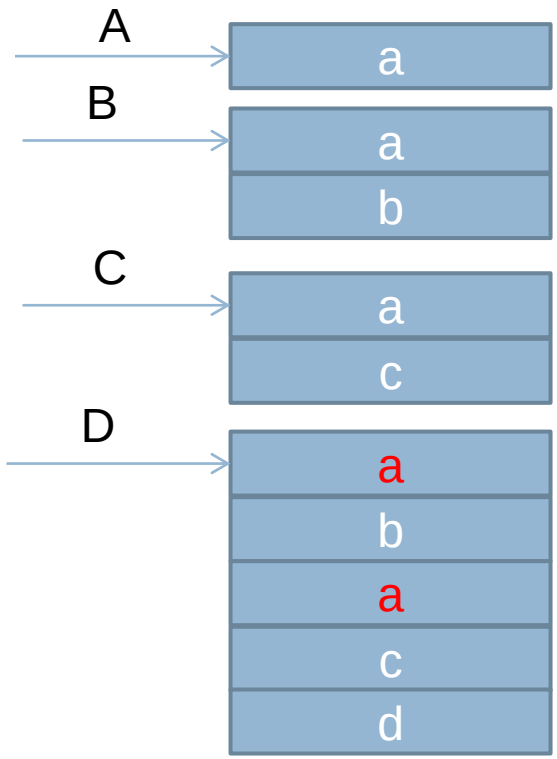
Diamantový problém

```
class A {
  int a;
};

class B : public A {
  int b;
};

class C : public A {
  int c;
};

class D : public B, public C {
  int d;
};
```



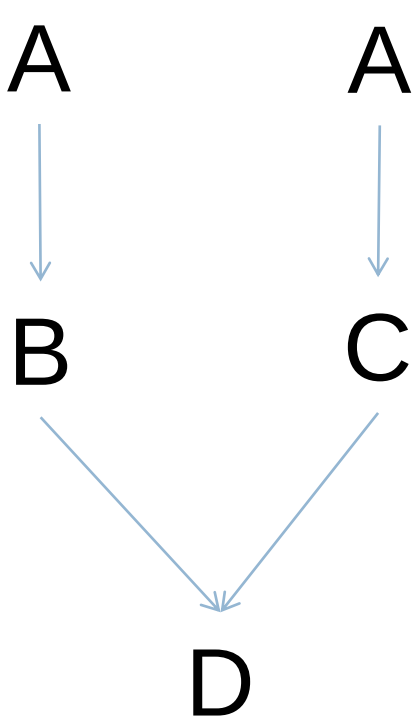
Diamantový problém

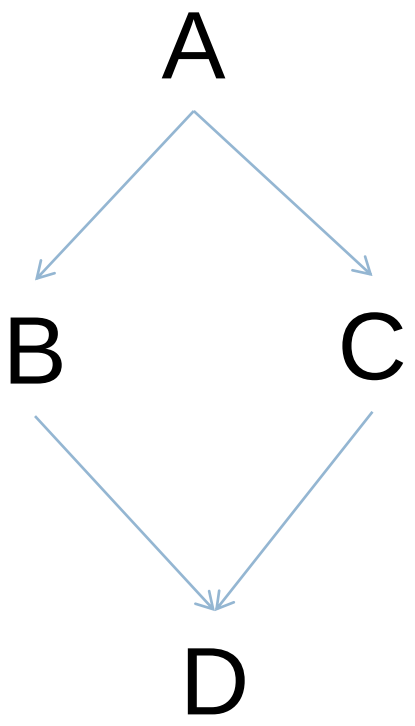
```
class A {
public:    int a;
};
class B : public A {
public:    int b;
};
class C : public A {
public:    int c;
};
class D : public B, public C {
public:    int d;
};

int main() {
  D d;
  d.b = 1;     // ok
  d.c = 2;     // ok
  d.a = 3;     // chyba!
}
```




```
t.c: In function ‘int main()’:
t.c:19:5: error: request for member ‘a’ is ambiguous
t.c:3:15: error: candidates are: int A::a
t.c:3:15: error:                 int A::a
```

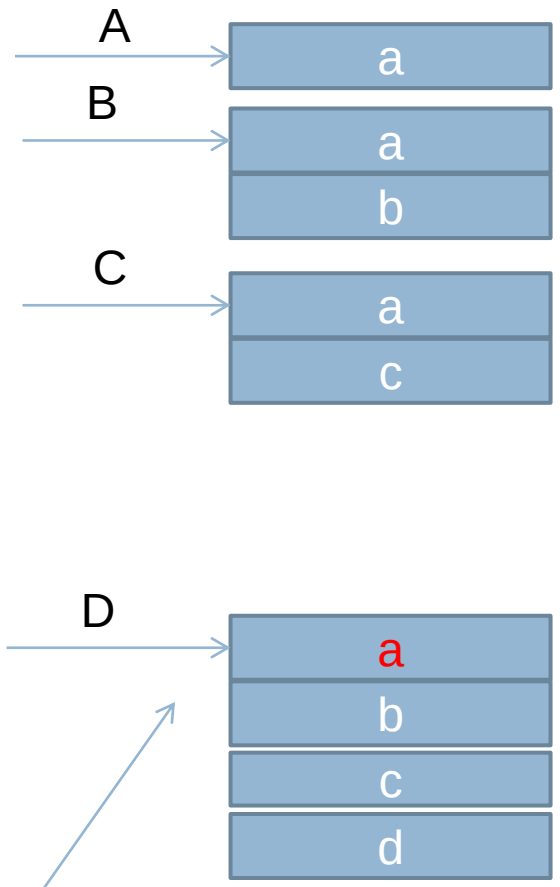
Chceme skôr diamantový graf dedenia





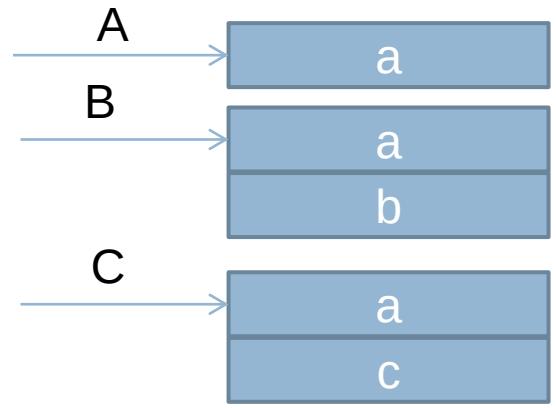
Riešenie C++: virtuálne dedenie (odvodenie)

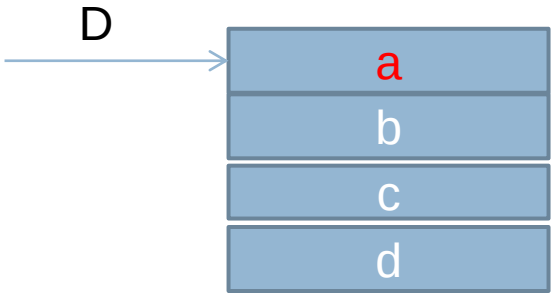
```
class A {
public:    int a;
};
class B : public virtual A {
public:    int b;
};
class C : public virtual A {
public:    int c;
};
class D : public B, public C {
public:    int d;
};
int main() {
  D d;
  d.b = 1;     // ok
  d.c = 2;     // ok
  d.a = 3;     // ok
}
```



virtuálne dedenie (odvodenie): problém

```
class A {
public:    int a;
};
class B : public virtual A {
public:    int b;
};
class C : public virtual A {
public:    int c;
};
class D : public B, public C {
public:    int d;
};
int main() {
  D d;
  d.b = 1;        // ok
  d.c = 2;        // ok
  d.a = 3;        // ok
  C *pc = &d;     // ???
}
```

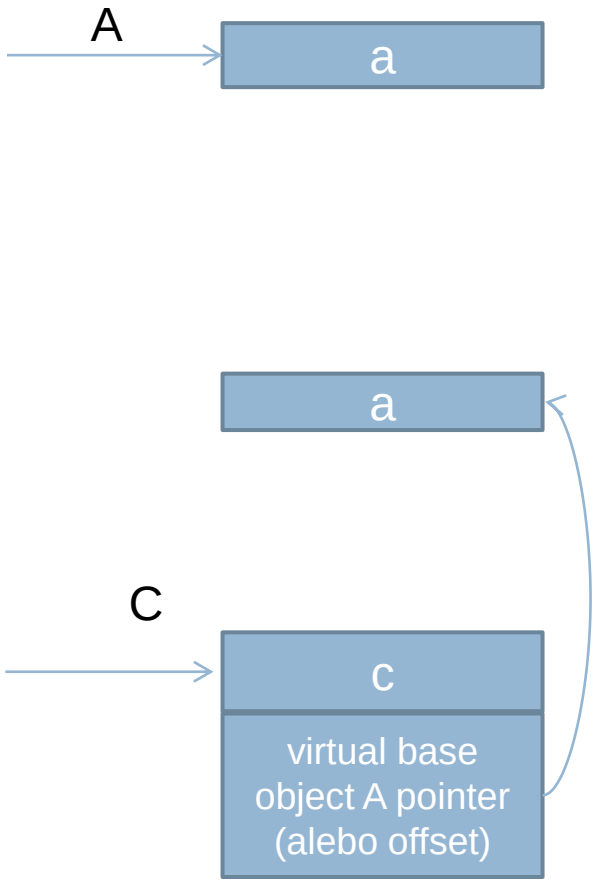




Kde je v D uložená trieda C?
Čo bude smerník na C?

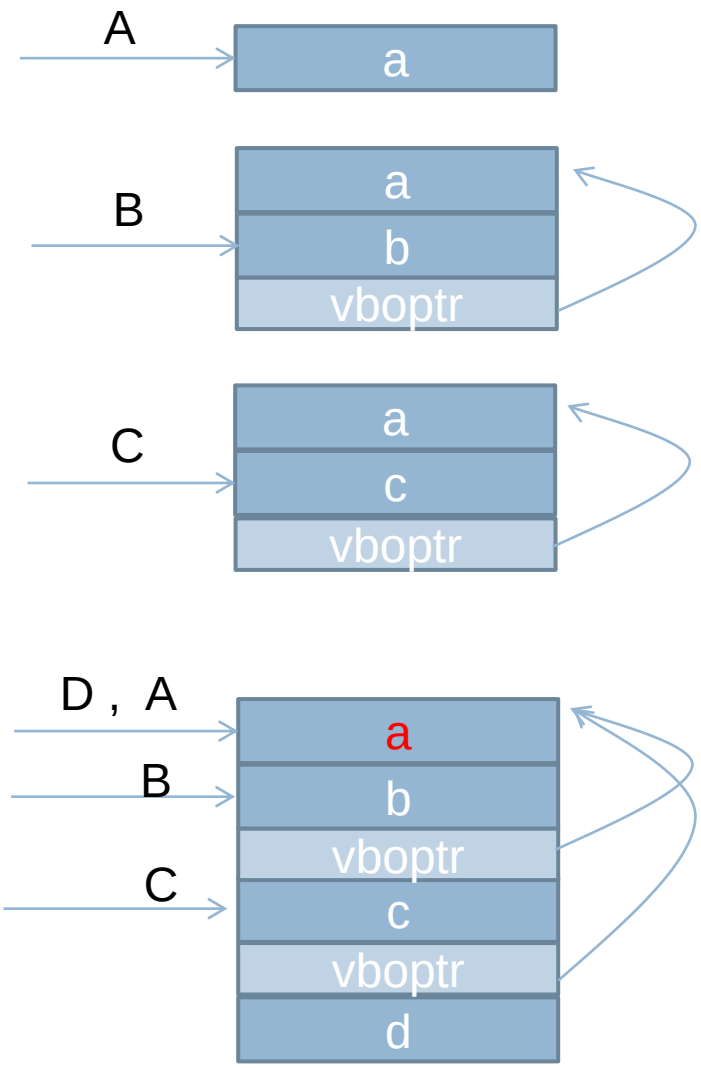
Virtuálne dedenie: ako na to

```
class A {
public:    int a;
};

class C : public virtual A {
public:    int c;
};
```



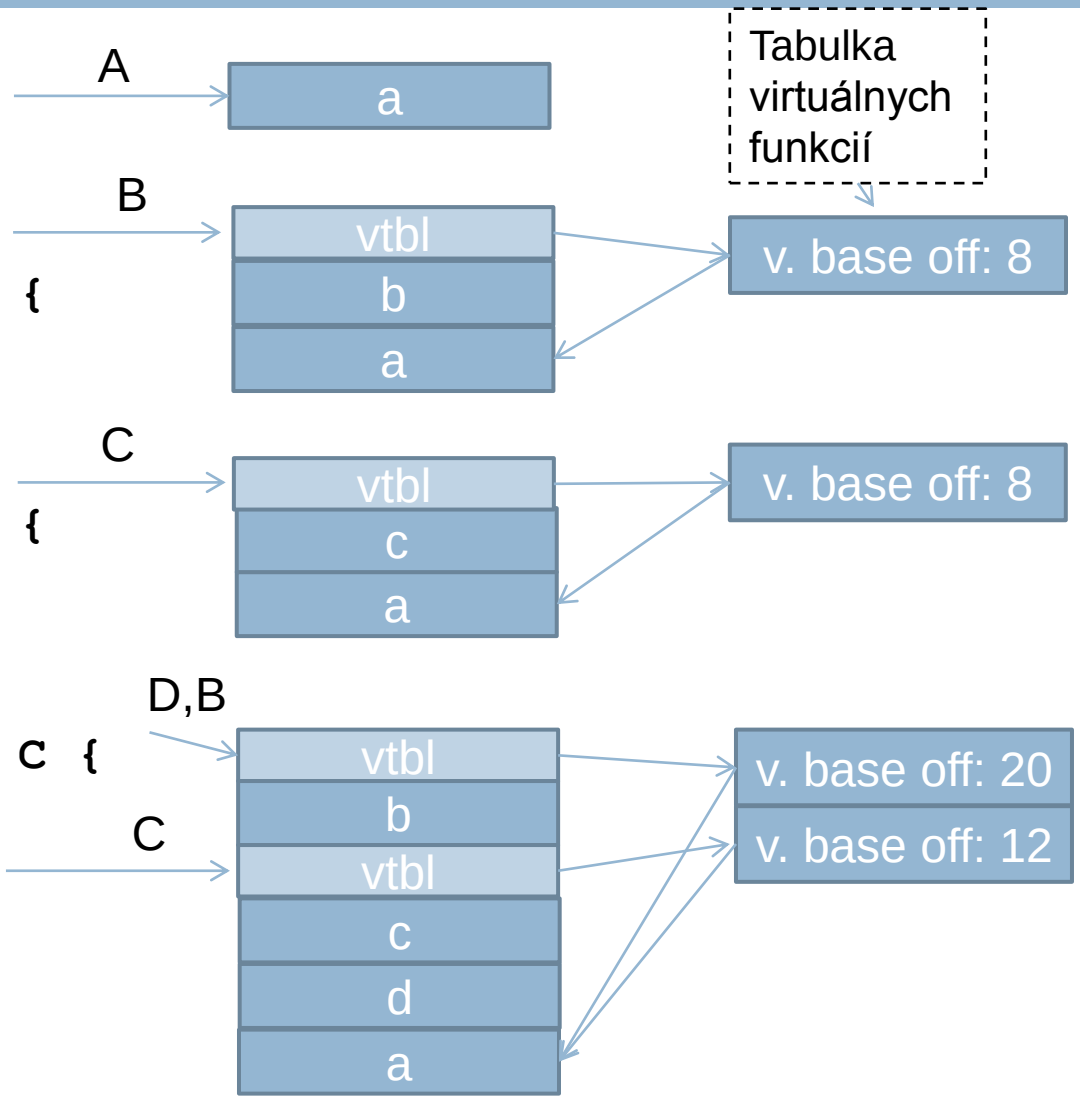
Virtuálne dedenie: ako na to

```
class A {
public:    int a;
};

class B : public virtual A {
public:    int b;
};

class C : public virtual A {
public:    int c;
};

class D : public B, public C {
public:    int d;
};
```



Virtuálne dedenie: skutočnosť

```
class A {
public:    int a;
};

class B : public virtual A {
public:    int b;
};

class C : public virtual A {
public:    int c;
};

class D : public B, public C {
public:    int d;
};
```



Java a C# riešenie

- Nepovoliť viacnásobné dedenie na triedach
- Zaviesť akýsi okýptený tvar tried bez možnosti definovať dáta a nazvať ho "*interface*" (*interface* navyše nemôže definovať metódu s telom, len signatúry metód sú povolené)
- Povoliť formu viacnásobného dedenia (cez kľúčové slovíčko "`implements`") iba na *interface*-och.

Príklad Java interface

```
interface Ordered {
  boolean isLessThan(Ordered x);
}

class Sortings {
  static void quickSort(Ordered[] a)  {
     ...
     if (a[i].isLessThan(a[j])) ...
     ...
  }
}

class MyButton extends Button  implements Ordered {
   boolean isLessThan(Ordered x) { ... }
   ...
      MyButton[] a = ...;
      Sorting.quickSort(a);
   ...
}
```

Príklad C++ viacnásobné dedenie

```
class Ordered {
  virtual bool isLessThan(Ordered *x) = 0;
}

class Sortings {
  static void quickSort(Ordered *a)   {
      ...
      if (a[i].isLessThan(&a[j])) ...
      ...
  }
}

class MyButton : public Button, public Ordered {
   virtual bool isLessThan(Ordered *x) { ... }
   ...
      MyButton *a = ...;
      Sorting::quickSort(a);
   ...
}
```

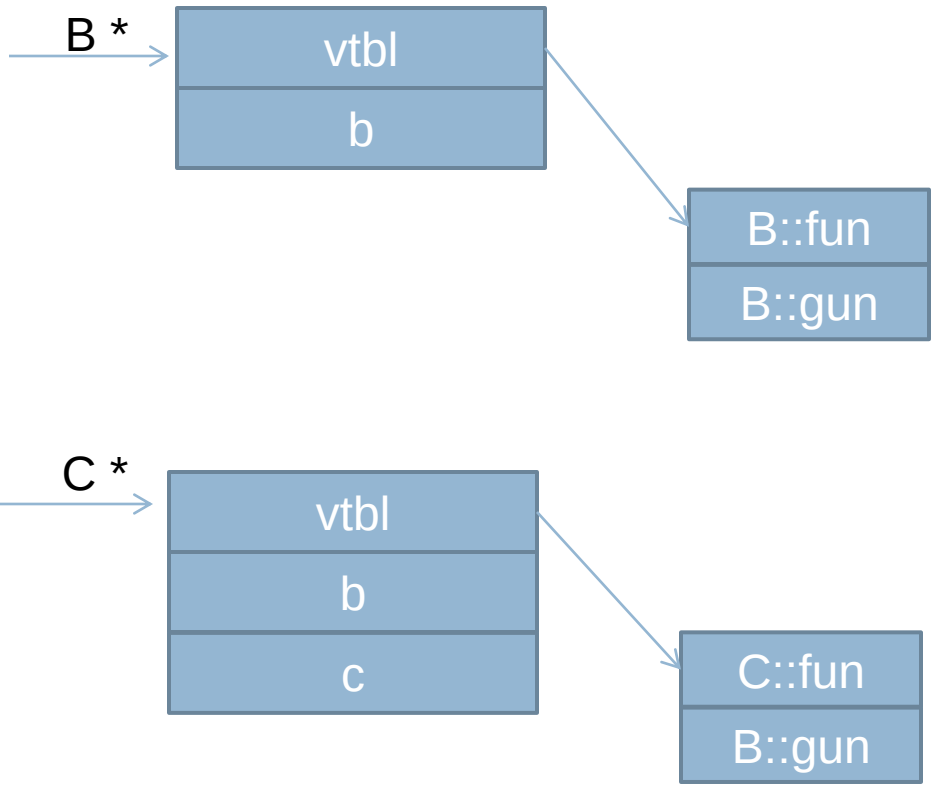
Virtuálne metódy

```
class B {
public:  int b;
  virtual void fun(){printf("B::fun()");}
  virtual void gun(){fun();}
};

class C : public B {
public:  int c;
  virtual void fun(){printf("C::fun()");}
};

int main() {
  C c;      B *pb = &c;
  pb->gun();
}
```

Zatiaľ jednoduché dedenie, program by mal vypísať: C::fun()

Reprezentácia v pamäti (jednoduché dedenie)

```
class B {
public:  int b;
  virtual void fun() {...}
  virtual void gun(){...}
};

class C : public B {
public:  int c;
  virtual void fun(){...}
};
```



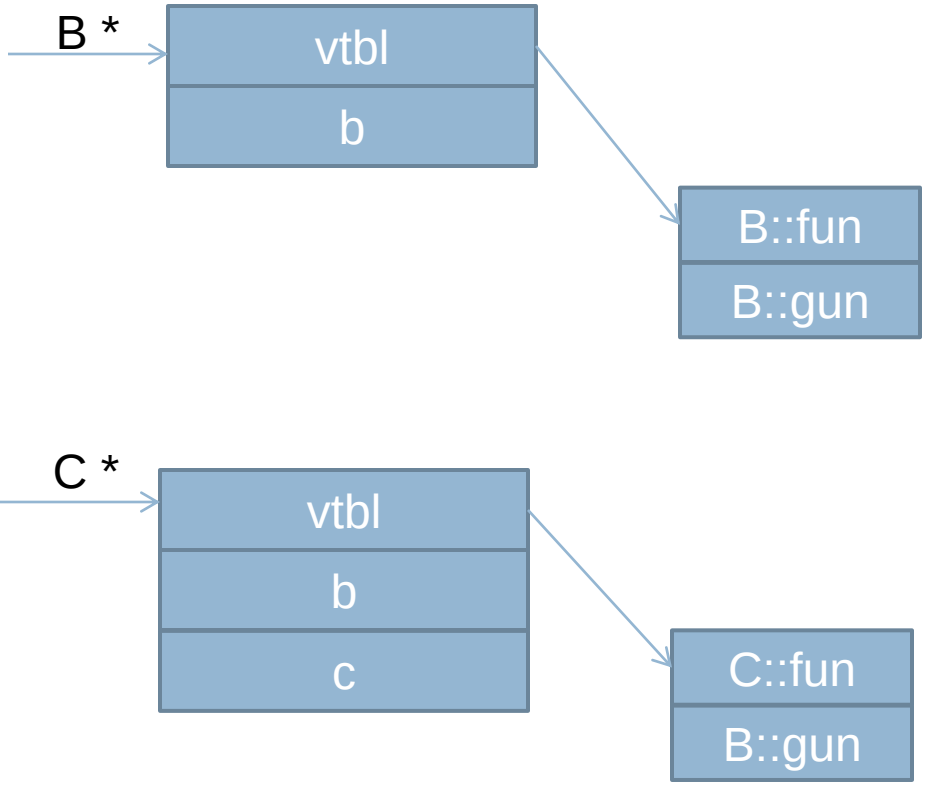
Viacnásobné dedenie a virtuálne metódy

```
class A {
public:  int a;
};

class B {
public:  int b;
  virtual void fun(){printf("B::fun()");}
  virtual void gun(){fun();}
};

class C : public A, public B {
public:  int c;
  virtual void fun(){printf("C::fun()");}
};

int main() {
  C c;      B *pb = &c;
  pb->gun();
}
```

Len sme pridali triedu, ktorá nič nepredefinovala program by mal stále vypísať:
C::fun()

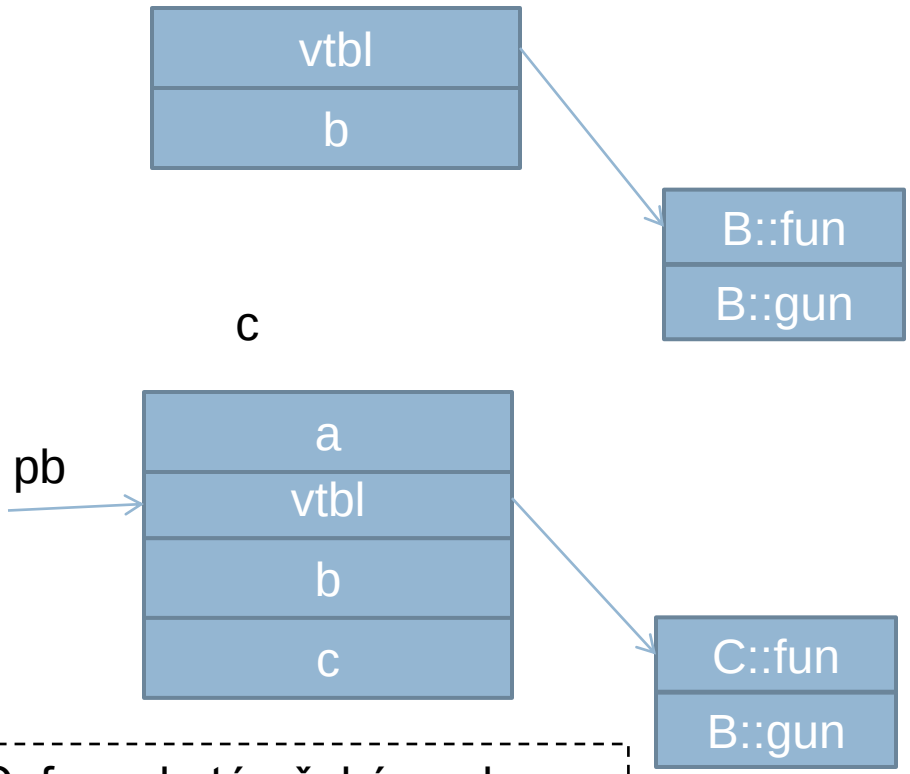
Reprezentácia v pamäti (jednoduché dedenie)

```
class B {
public:  int b;
  virtual void fun() {...}
  virtual void gun(){...}
};

class C : public B {
public:  int c;
  virtual void fun(){...}
};
```



Reprezentácia v pamäti

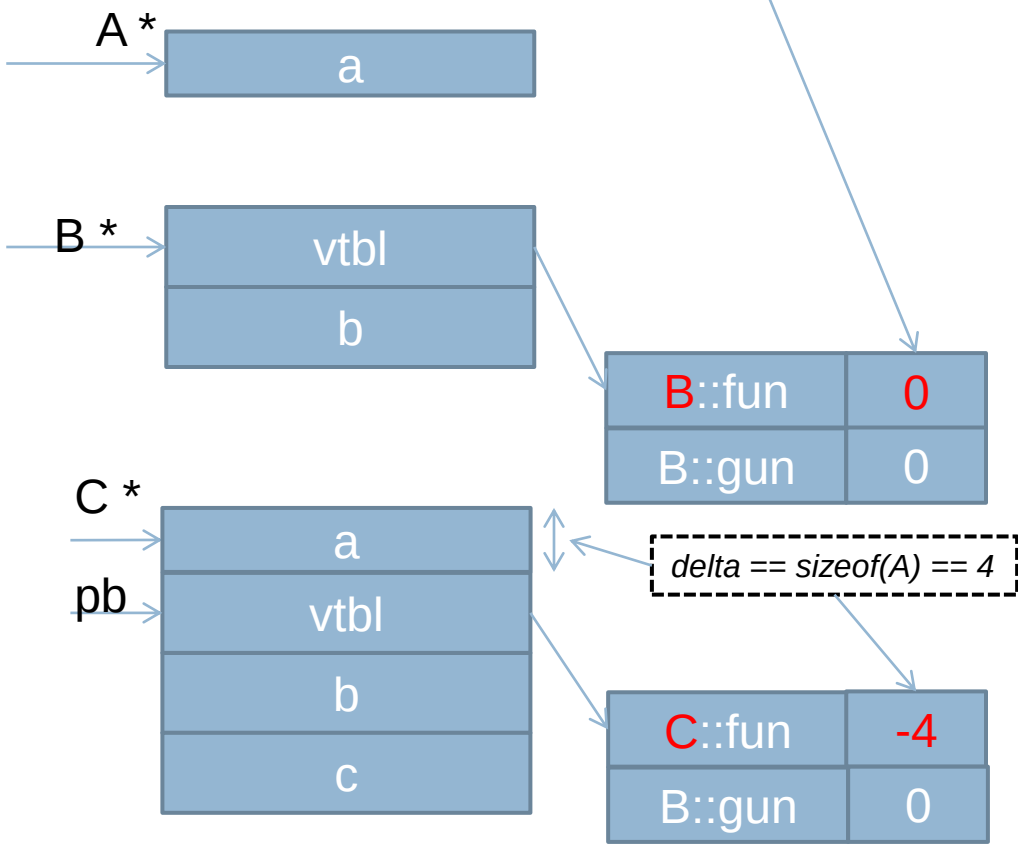
```
class A {
public:  int a;
};

class B {
public:  int b;
  virtual void fun(){...}
  virtual void gun(){...}
};

class C : public A, public B {
public:  int c;
  virtual void fun(){...}
};

int main() {
  C c;
  B *pb = &c;
  pb->fun();
}
```



Vyvolá C::fun, ale tá očakáva ako implicitný parameter smerník na C nie na B. Ako mám vedieť kedy ho zmeniť a ako?

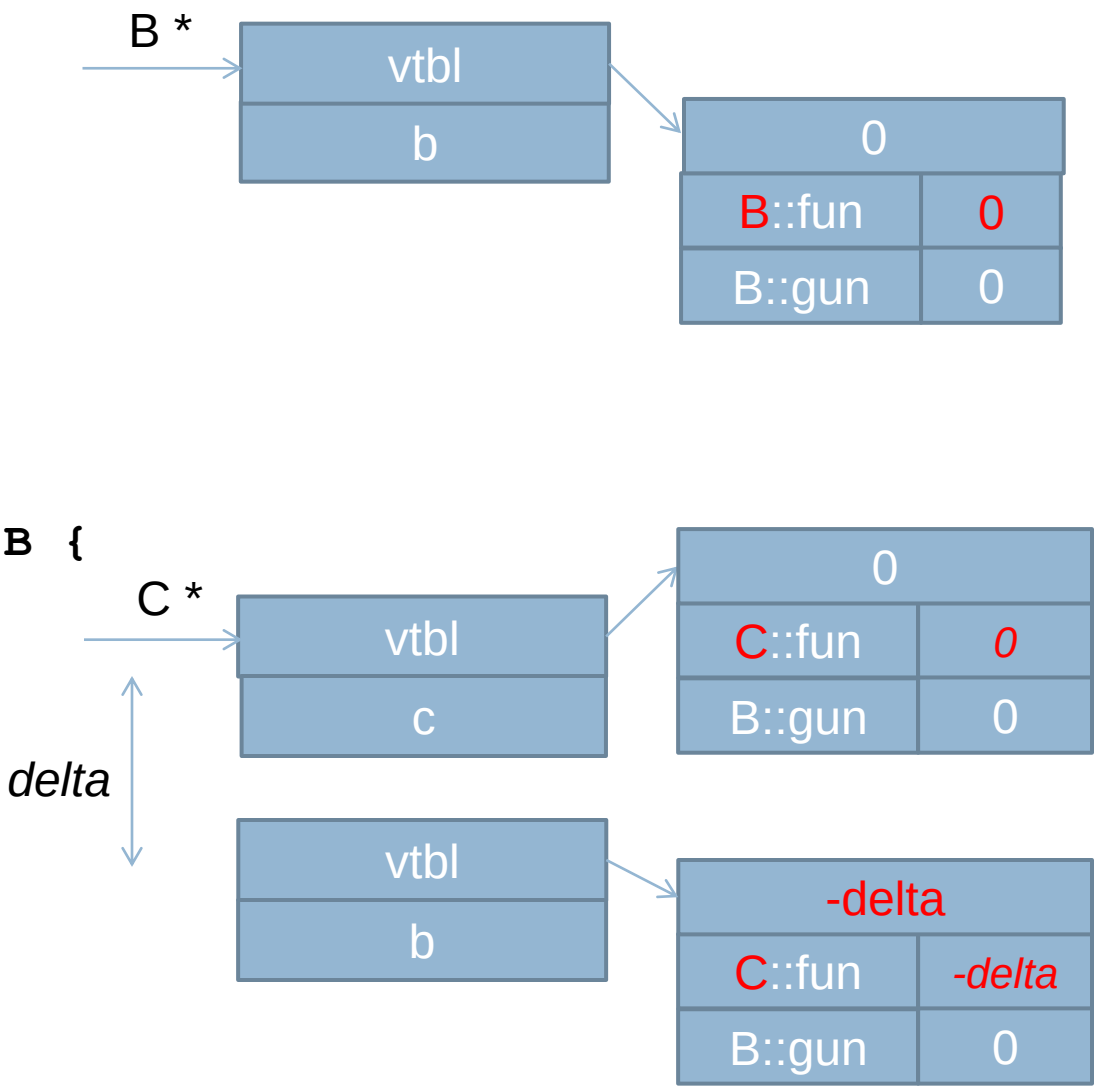
Reprezentácia v pamäti

```
class A {
public:  int a;
};

class B {
public:  int b;
  virtual void fun(){...}
  virtual void gun(){...}
};

class C : public A, public B {
public:  int c;
  virtual void fun(){...}
};
int main() {
  C c;
  B *pb = &c;
  pb->fun();
}
```



Virtuálne dedenie a virtuálne metódy

```
class B {
public:  int b;
  virtual void fun(){...}
  virtual void gun(){...}
};

class C : public virtual B {
public:  int c;
  virtual void fun(){}
};
```



Záver

- Vyvolanie virtuálnej metódy vyžaduje dodatočný prístup do pamäte (a eventuálne korekciu smerníka)
- Prístup k premennej a vyvolanie virtuálnej metódy cez virtuálne dedenie stojí minimálne jeden dodatočný prístup do pamäte. Toto pravdepodobne platí aj pre vyvolanie metódy z interfacu v Jave a C#.

"Dominácia" mien

```
class A {
public: int x;
};

class B : public virtual A {
public: int x;
};

class C : public virtual A {
};

class D : public C, public B {
  void fun() {
       x ++;
  }
};
```

? chyba
? alebo A::x ++
? alebo B::x ++

"Dominácia" mien

```
class A {
public: int x;
};

class B : public virtual A {
public: int x;
};

class C : public virtual A {
};

class D : public C, public B {
  void fun() {
      x ++;          // == B::x ++
  }
};
```

Ak trieda B virtuálne rozširuje triedu A, tak každé meno B::x "dominuje" menu A::x. Prípadná nejednoznačnosť sa rieši v prospech dominujúceho mena.

B::x "dominuje" nad A::x

Virtual base class and casting

```
class A {
  int a;
};

class B : public virtual A {
  ...
};

void fun(A *pa) {
  B *pb;
  pb  = (B *) pa;
}
```

Chyba! Podľa štandardu C++
nemožno konvertovať pointer
z triedy získanej cez virtuálne dedenie
(Reprezentácia v pamäti vo všeobecnosti
neobsahuje back pointer z A na B)

Ale

```
class A {
  int a;
  virtual ~A() {}
};

class B : public virtual A {
  ...
};

void fun(A *pa) {
  B *pb;
  pb  = dynamic_cast<B *>(pa);
}
```

Base and members initializers syntax

```
class X : public A, public B {
  int xx;
  X(int a, int b) : A(a), B(b), xx(0) {}
};
```