Vererbung: Grundprinzip von OO

- Übernahme von Eigenschaften aus einer Klasse
- Erweiterung / Modifikation

Beispiel: ein Stack mit Buchführung

```
class CountedStack : public Stack // IST EIN STACK
{
   int min, max, n, sum; // zusätzliche Attribute
public:
   CountedStack(int dim = 100);
   void push (int i); // redefined !
   int minimum(); // neu
   int maximum(); // neu
   double mean(); // neu
   double actual_mean();// neu
// pop, empty, full aus der Basisklasse !
};
```

```
CountedStack::CountedStack(int dim):Stack(dim),n(0),sum(0){}

void CountedStack::push(int i) {
   sum+=i;
   if (!n++) { min = max = i; }
   else { min = (i<min) ? i : min; max = (i>max) ? i : max; }
   Stack::push(i); // use base functionality NOT push(i)
}

double CountedStack::actual_mean() {
   if (top) { int s=0;
       for (int i=0; i<top; i++) s += data[i];
       return double(s)/top; // direct access to base members
   } else std::exit(-4);
}
```

Ist ein (nutzerdefinierter) Copy-Konstruktor erforderlich ?

Nein, weil der implizite Copy-K. die Copy-K.en aller Basisklassen ruft und für die Erweiterung `CountedStack` shallow copy ausreichend ist:

```
// implizit bereitgestellt:
CountedStack::CountedStack(const CountedStack& other)
  :
  Stack(other) {/* real copy */}
```

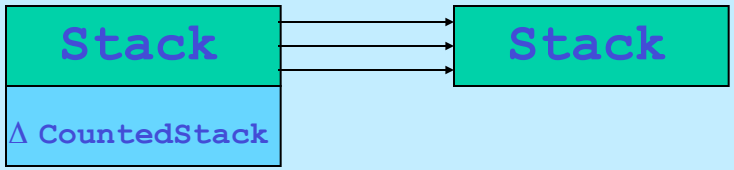
Der (nutzerdefinierte) `Stack`-Copy-K. erwartet allerdings eine
`const Stack& ????`

- Jedes **CountedStack** - Objekt **IST EIN** **Stack**-Objekt

```
CountedStack cs; ... cs.pop();
void foo (Stack&); ... foo (cs);
```

- von der Ableitung zur Basisklasse ist implizit eine Projektion definiert

```
void bar (Stack); ... bar(cs); // slicing
```



- nur bei `public` Vererbung gilt die **IST EIN** Relation

non-`public` Vererbung

```
class Deriv1 : private Base { .... };
```

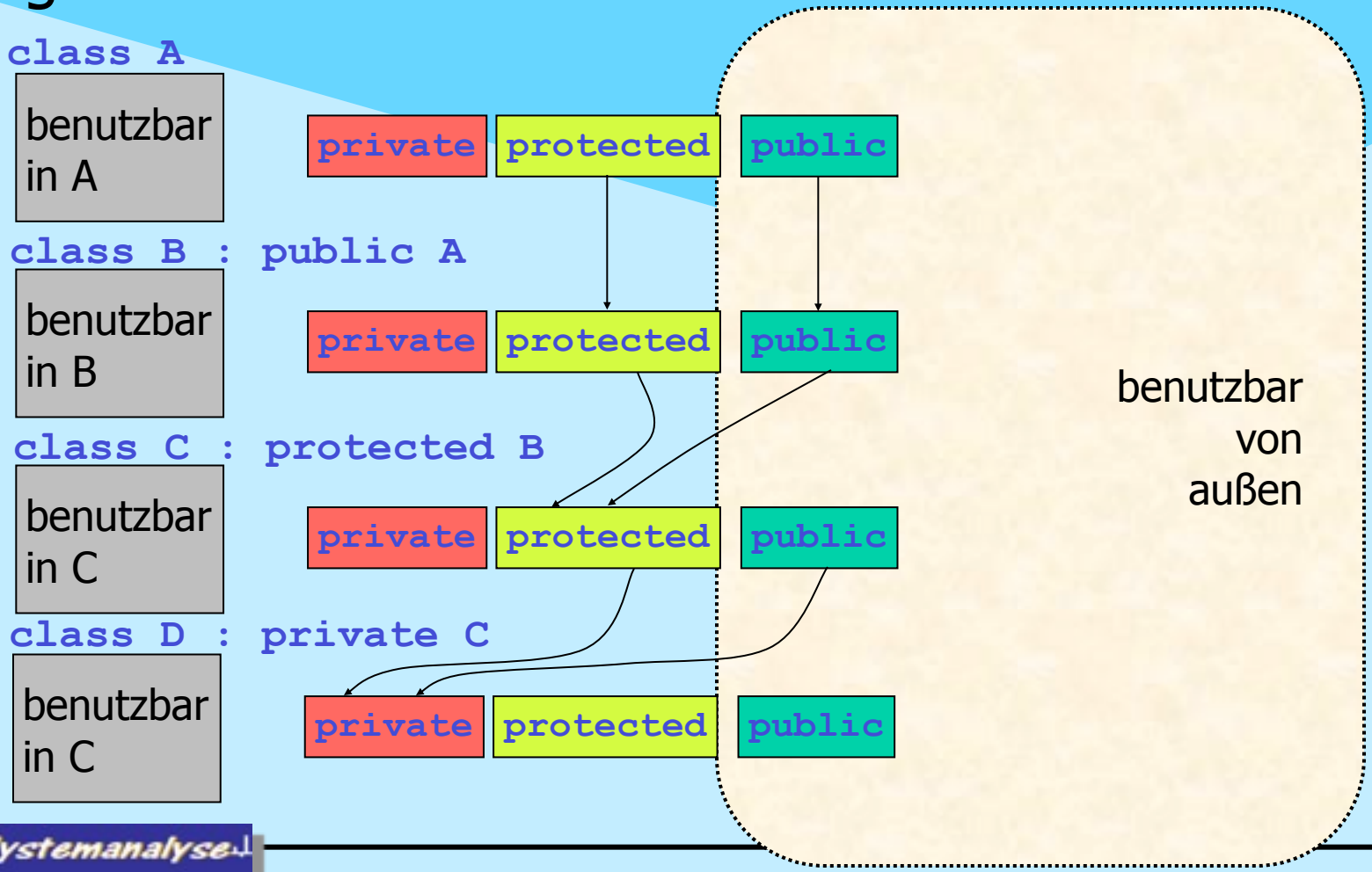
`Deriv1` IST nirgends EIN `Base` == die Vererbung ist ein (nicht erkennbares) Implementationsdetail

```
class Deriv2 : protected Base { .... };
```

`Deriv2` IST nur in Ableitungen von `Deriv2` EIN `Base` == die Vererbung ist nur Ableitungen `Deriv2` von bekannt

das Layout von Objekten abgeleiteter Klassen wird von der Art der Vererbung **NICHT** beeinflusst !

Zugriffsrechte in C++



`struct` ist implizit `public`, `class` ist implizit `private`

Depricated:

`struct` erbt implizit `public`, `class` erbt implizit `private`

Beim lookup von Funktionsnamen erfolgt
overload resolution **VOR** access check !

```
class X {
        foo(int);
public:
        foo(int, int = 0);
};

int main(){  X x;
             x.foo(1); //call of overloaded `foo(int)' is ambiguous
}
```