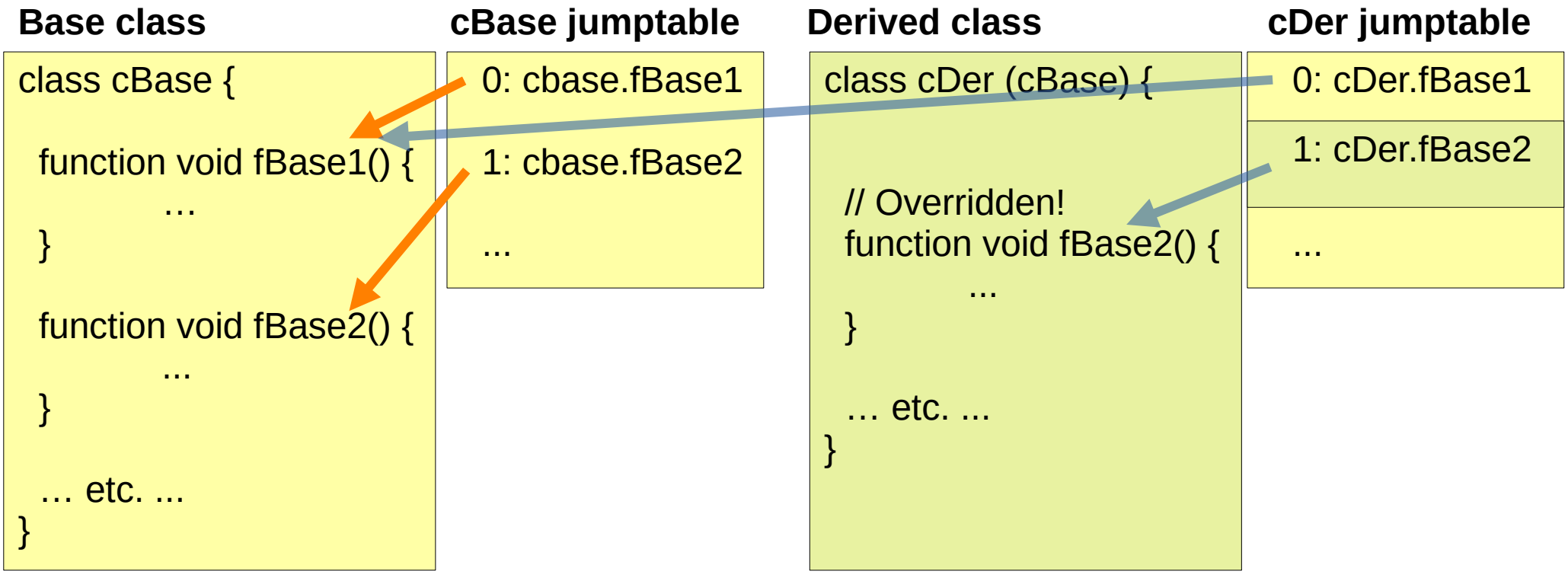
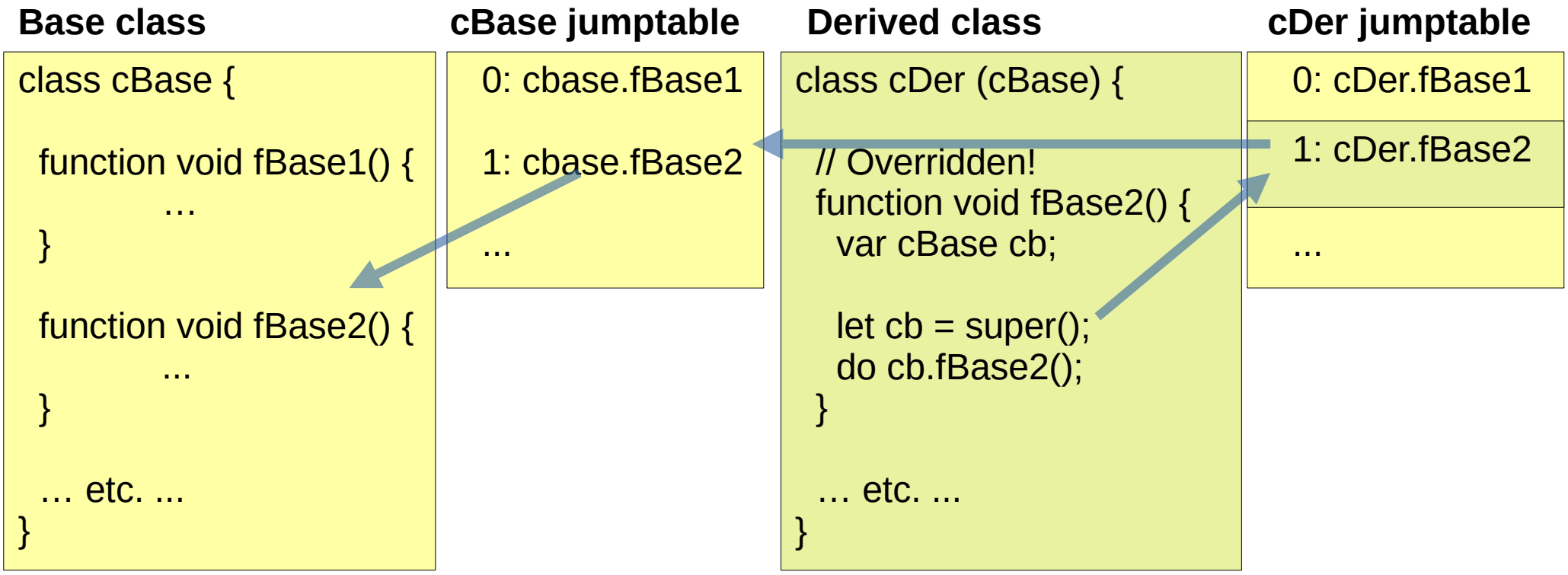
Class Function Calling

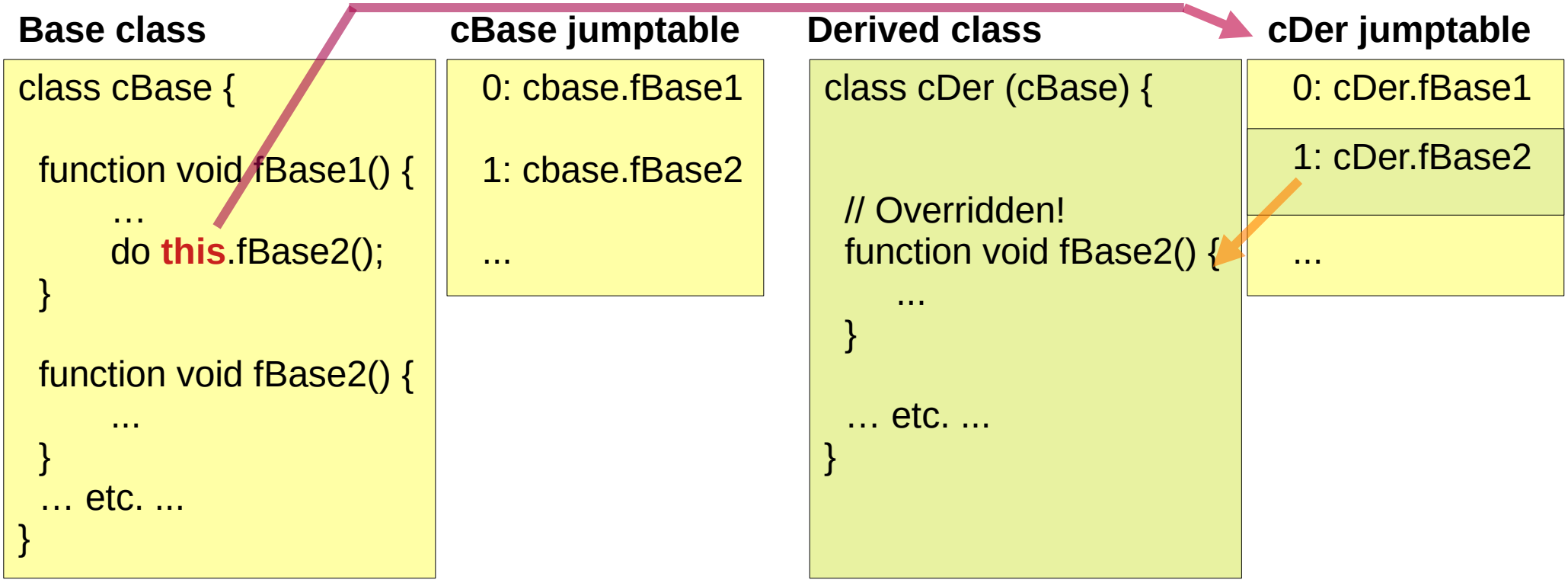


Base Class Function Calling



We need a link between a class & its jumptable!

Derived Class Function Calling



We need a link between ‚this‘ (the object instance) & its jumptable!

Instance Workings

Instance call

```
var cDer cd;

let cd = cDer.new();     // Create a derived class instance
do cDer.fBase1();        // As fBase1 is NOT overridden, calls cBase.fBase1()
                         // but ‚this' still pointing to ‚cDer', the derived class!
                         // If part of the instance data (‚this'→) is a class jumptable ref. …

function void fBase1() {
  do this.fBase2();      // Requires jumptable usage
}
```

Jumptable call

```
...
call this→jumptable[<n>]
...
```

Class & Instance Data

Base class data

RefBase: Null
<bfunction#0>
<bfunction#1>
…
<bfunction#n>

<base_static_var#0>
…
<base_static_var#n>

Base class instance data

RefCls: → BaseClass
<base_field_var#0>
<base_field_var#1>
…
<base_field_var#n>

Derived class data

RefBase: → base
<bfunction#0>
<dfunction#1>
…
<bfunction#n>
<dfunction#n+1>
...

<base_static_var#0>
…
<base_static_var#n>

<der_static_var#0>
…
<der_static_var#n>

Derived class instance data

RefCls: → derivedClass
<base_field_var#0>
<base_field_var#1>
…
<base_field_var#n>
<der_field_var#0>
…
<der_field_var#n>

Lib functions (called on init.):
- CreateClsJumpTable(cls)
  generated by compiler

Called dynamically:
- this = CreateInstance(cls)
- CallInstance(this, <n>)
- super(this)

Design Decisions

1. Field variables of a class are considered private, that is, in a derived class the base class field variables are still **NOT** accessible (provide setter/getter methods if nec. !).

2. For performance reasons, **ONLY** methods will be accessed via jumptables – otherwise each & every function call will have to be indirect via a jumptable!

Drawback: (Static) function hiding/overriding/re-routing will not be possible.

Alternative A: Provide an attribute to indicate an overrideable (static) function (accessed via a jumptable)? But this will require a 4th function type (additional to ‚function', ‚method' & ‚constructor') …
And who decides, when to use the attribute & how does the user know thereof?
→ A road not taken …

Alternative B: If functions could live **outside** of classes (but in the same source), this may be used as the missing distinction! This would change the character of the language quite massively.
→ Maybe in a later language revision …

3. All methods of a base class will be listed in a derived class jumptable.