Einführung in die Programmierung
Introduction to Programming

Prof. Dr. Bertrand Meyer

Exercise Session 10

Today

- Multiple inheritance

Inheritance is never the only way

Given the classes

- TRAIN_CAR, RESTAURANT

how would you implement a DINER?

- You could have an attribute in TRAIN_CAR

  train_service: SERVICE

- Then have RESTAURANT inherit from SERVICE
- This is flexible if the kind of service may change to a type that is unrelated to TRAIN_CAR
- Changes in TRAIN_CAR do not affect SERVICE easily

Examples of multiple inheritance

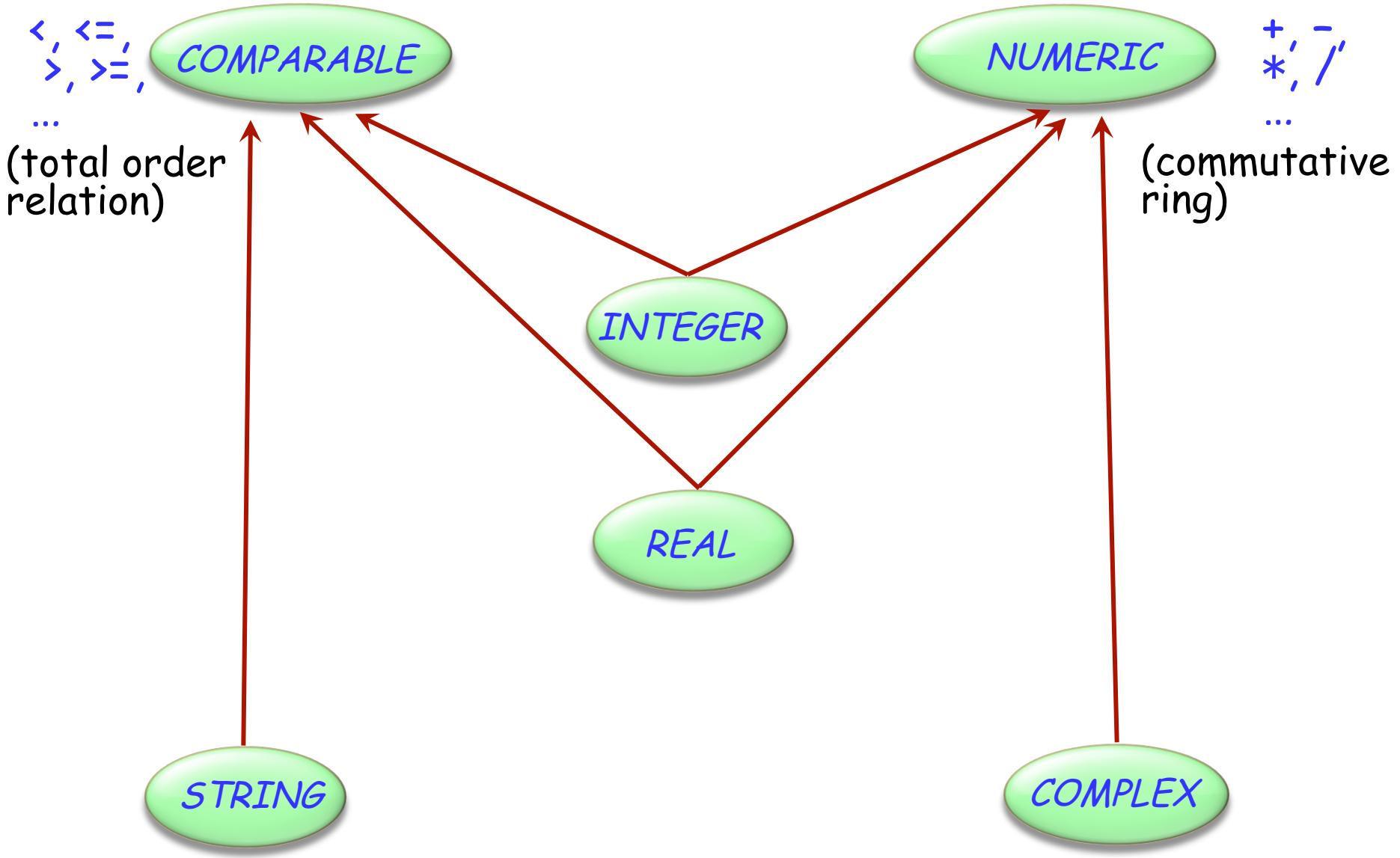
Combining separate abstractions:

- Restaurant, train car
- Calculator, watch

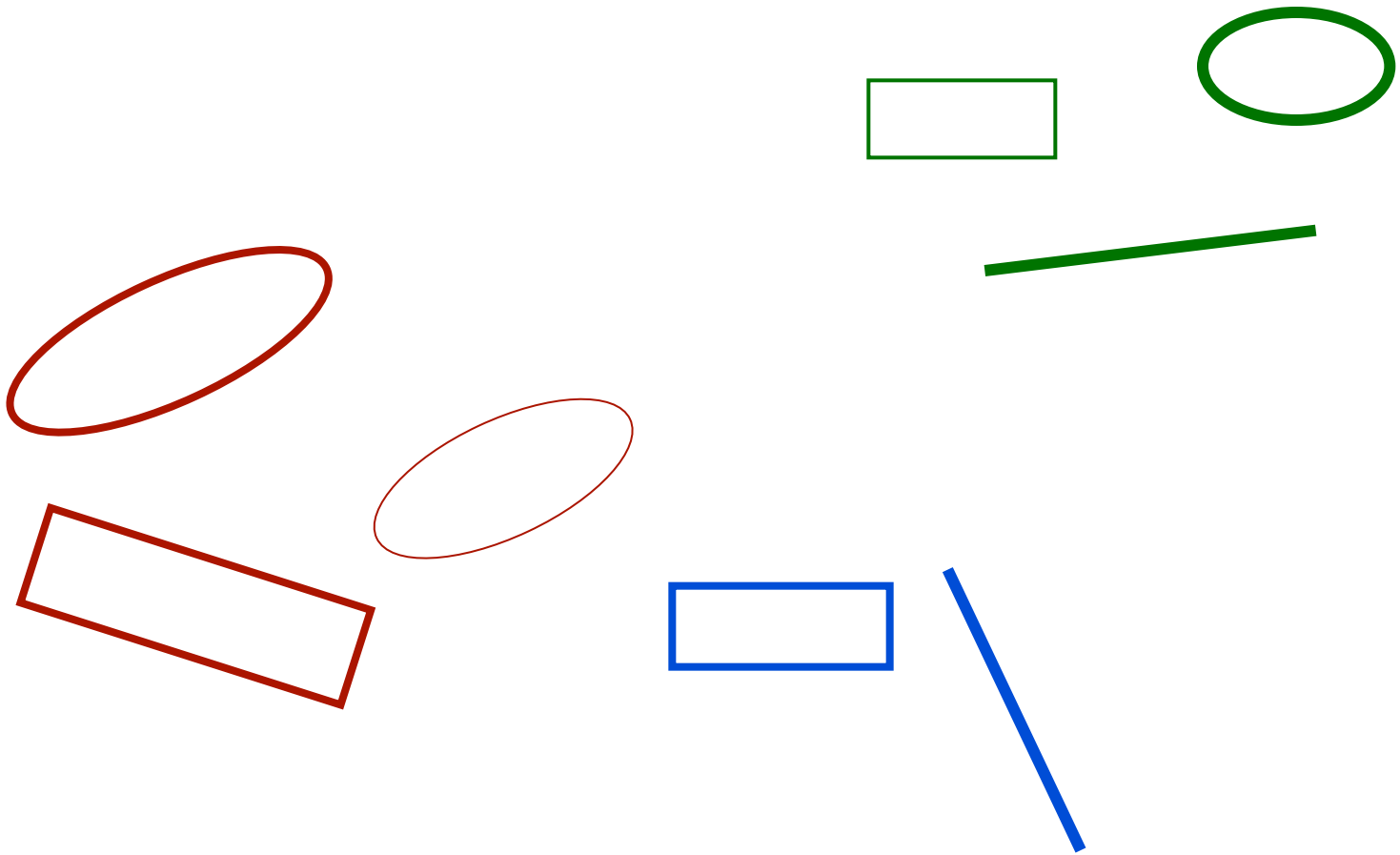
- Other examples?

- Teacher, student
- Home, vehicle

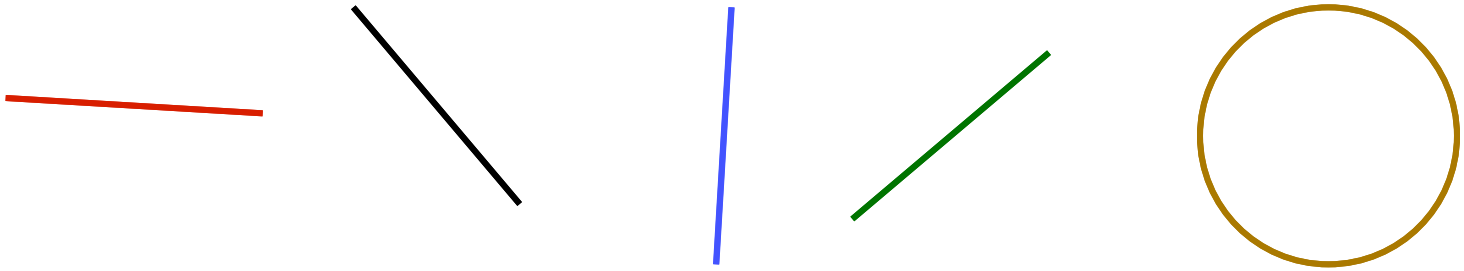
Multiple inheritance: Combining abstractions



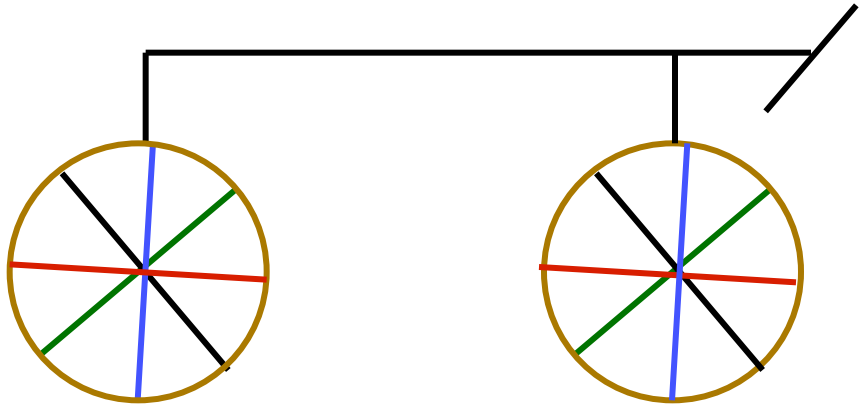
Composite figures



Multiple inheritance: Composite figures

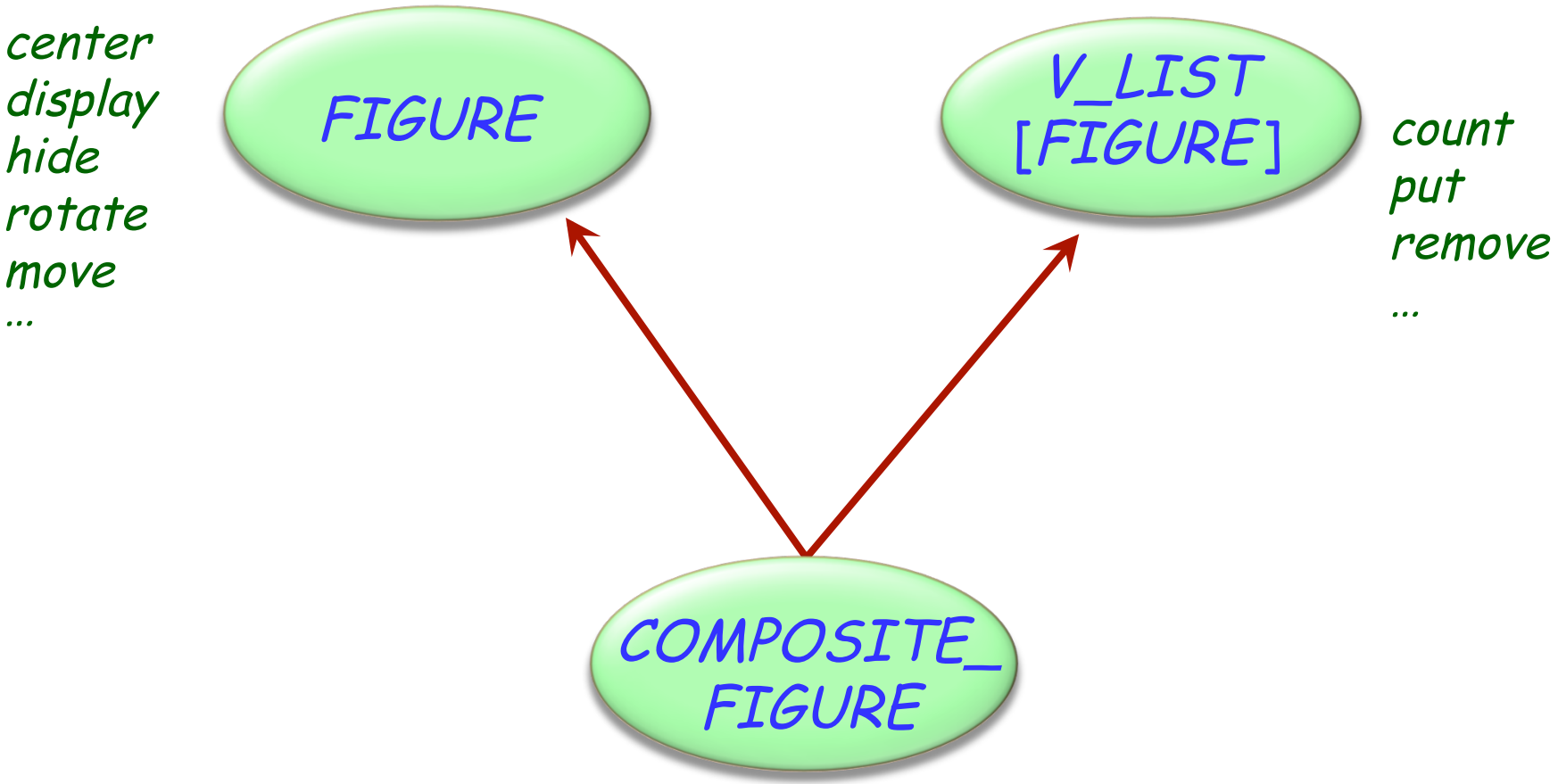


Simple figures

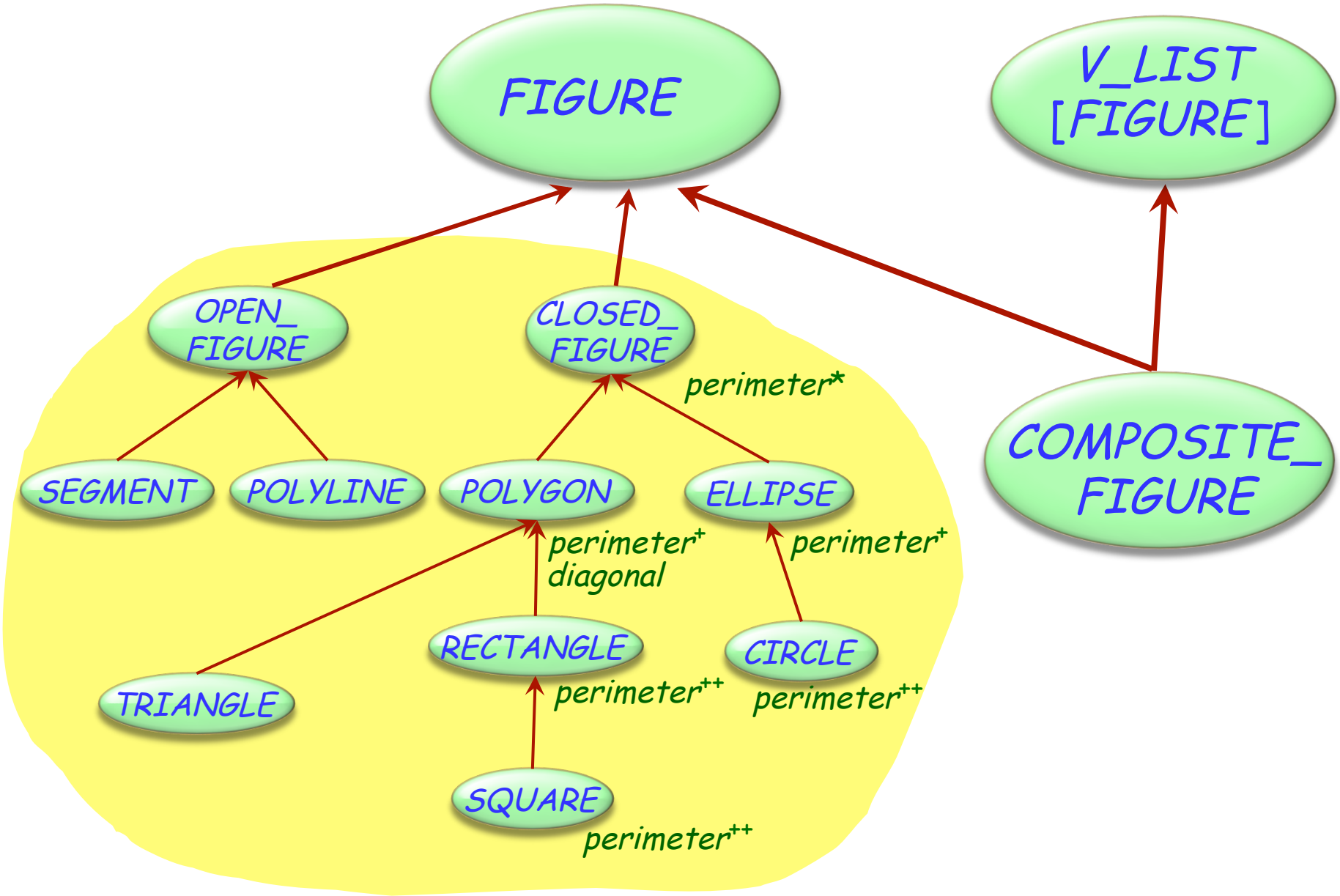


A composite figure

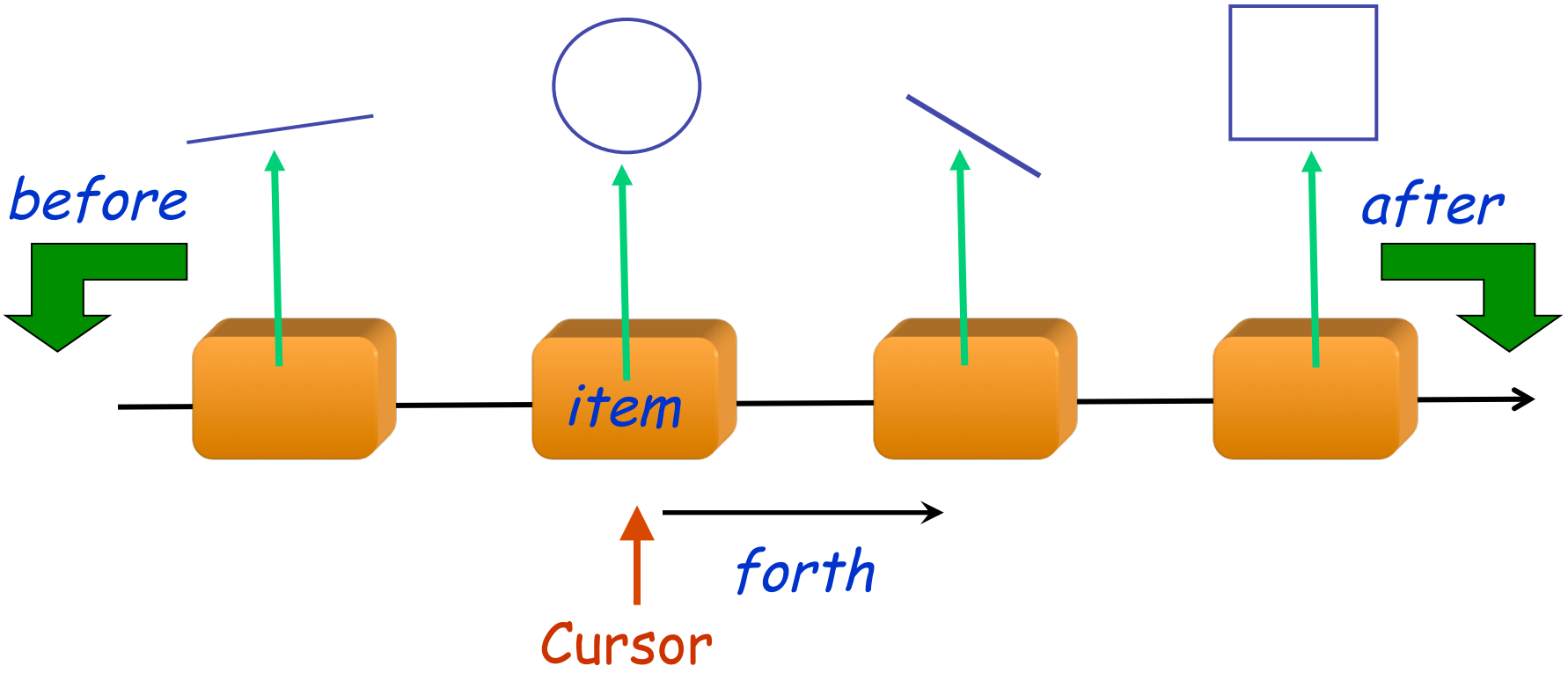
Defining the notion of composite figure



In the overall structure



A composite figure as a list

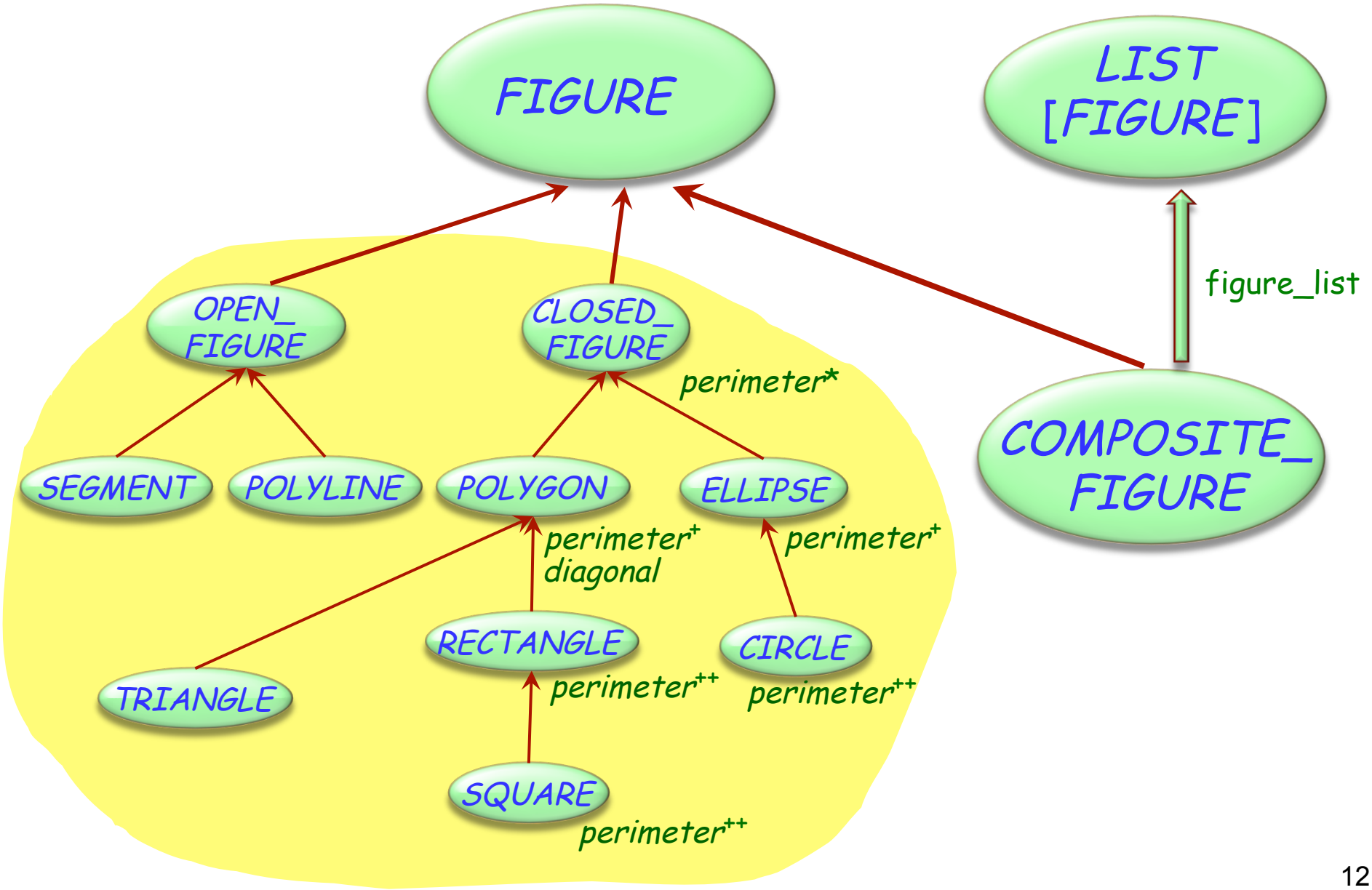


Composite figures

```
class COMPOSITE_FIGURE inherit
        FIGURE
        V_LIST [FIGURE]
feature
        display
                -- Display each constituent figure in turn.
        do
                from start until after loop
                        item.display
                        forth
                end
        end
        ... Similarly for move, rotate etc. ...
end
```

Requires dynamic binding

An alternative solution: the composite pattern



The Java-C# solution

No multiple inheritance for classes

"Interfaces": specification only (but no contracts)

- Similar to completely deferred classes (with no effective feature)

A class may inherit from:

- At most one class
- Any number of interfaces

Lessons from this example

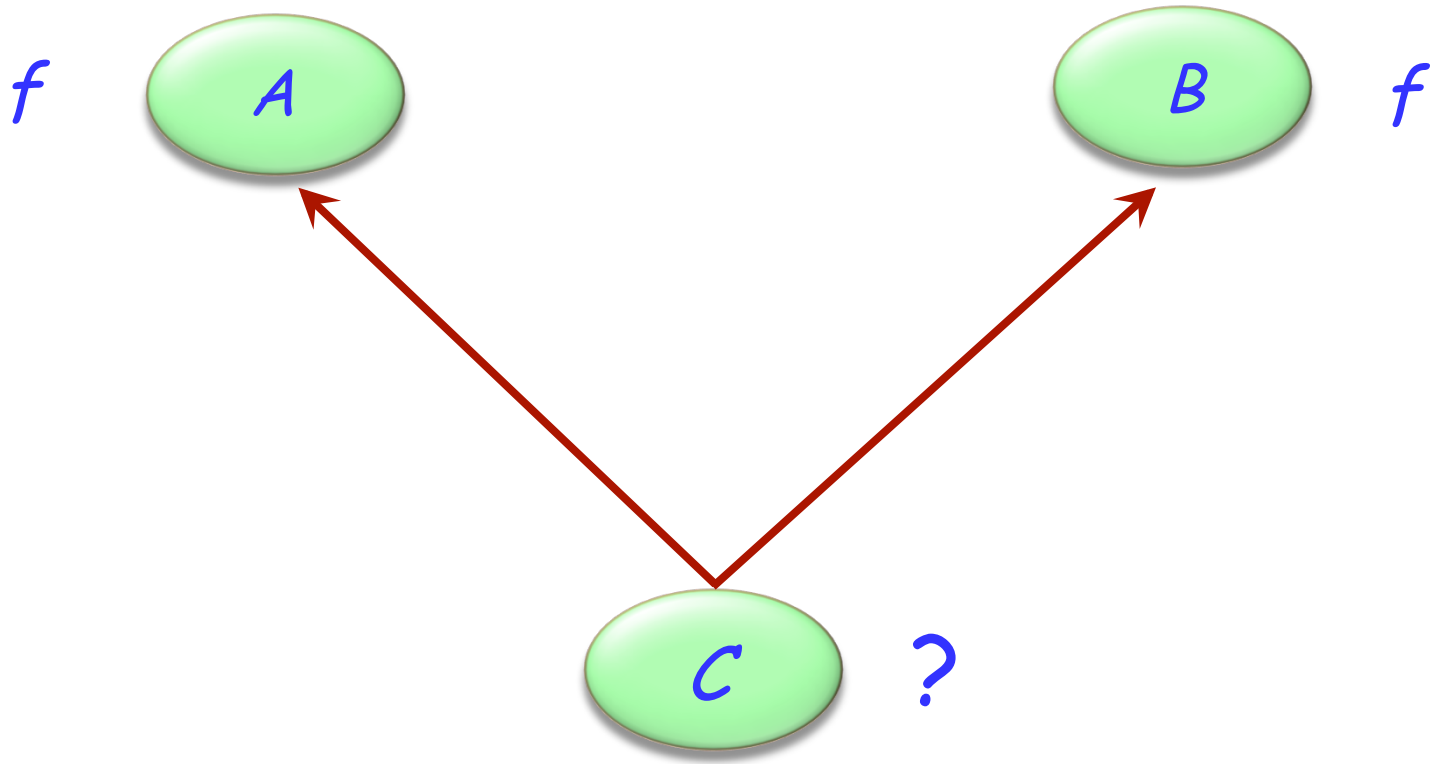
Typical example of *program with holes*

We need the full spectrum from fully abstract (fully deferred) to fully implemented classes

Multiple inheritance is there to help us combine abstractions

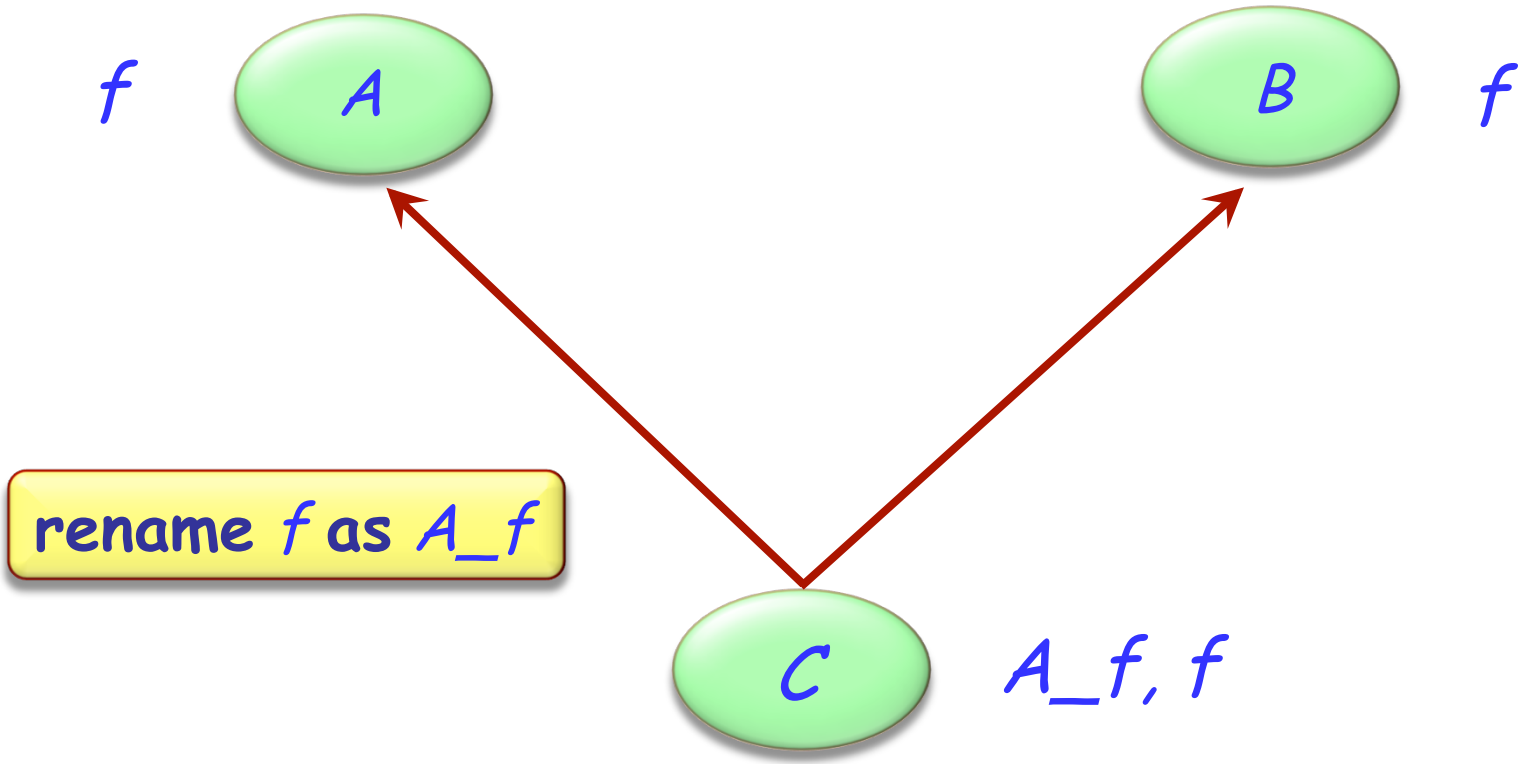
Multiple inheritance: Name clashes

Hands-On



Resolving name clashes

Hands-On

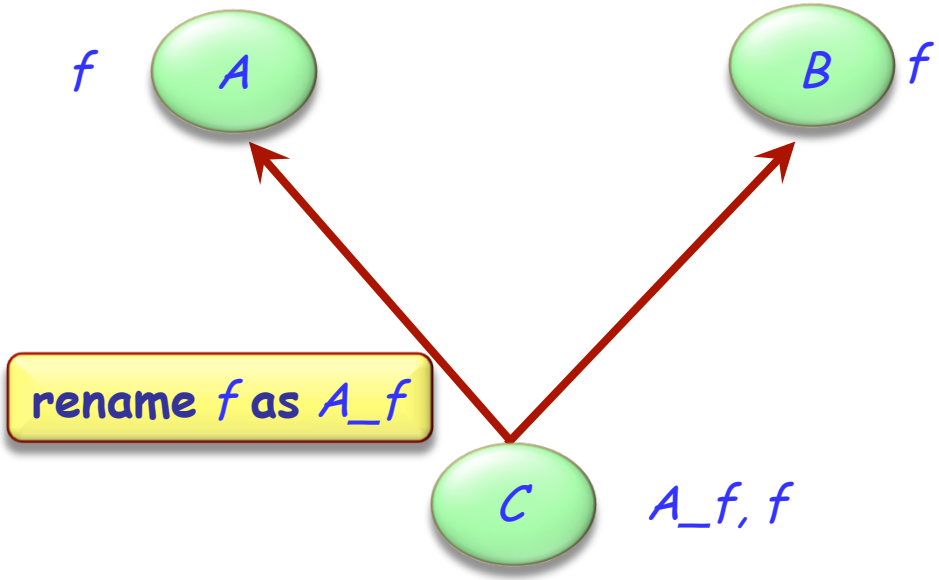


Consequences of renaming

Hands-On

Valid or invalid?

a1: A
b1: B
c1: C
...



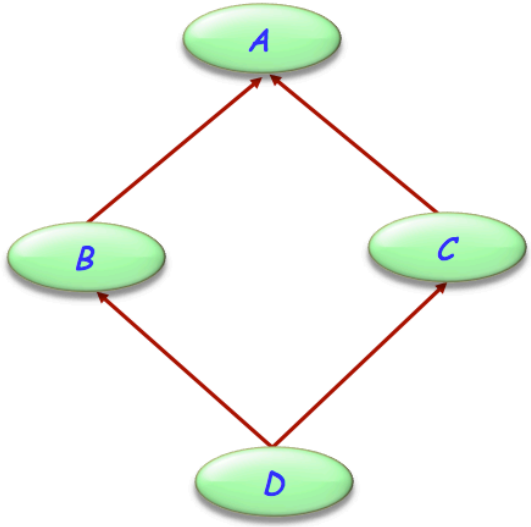
c1.f	Valid
a1.A_f	Invalid
c1.A_f	Valid
b1.f	Valid
b1.A_f	Invalid

Are all name clashes bad?

A name clash must be removed unless it is:

- Under repeated inheritance (i.e. not a real clash)

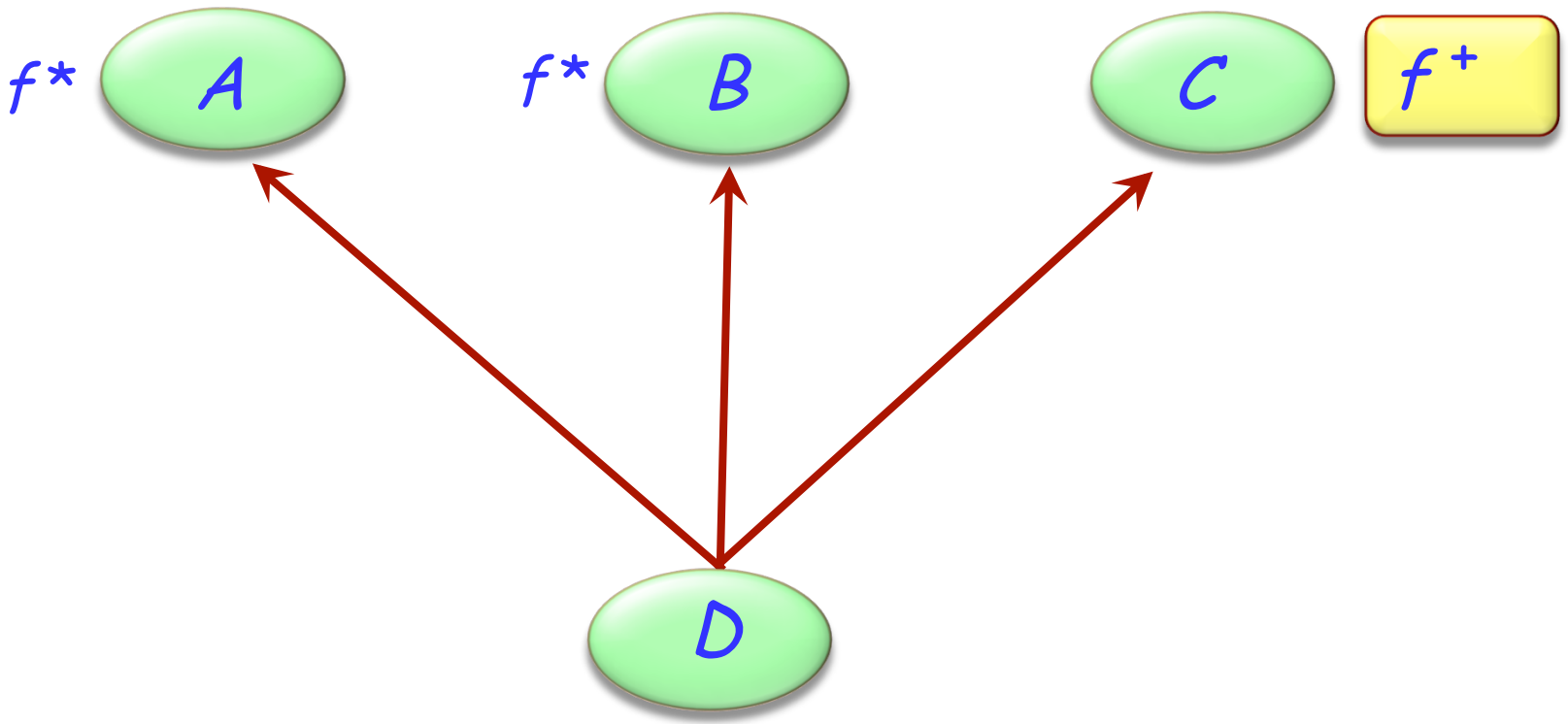
indirect repeated inheritance



direct repeated inheritance

- Between features of which at most one is effective (i.e. others are deferred)

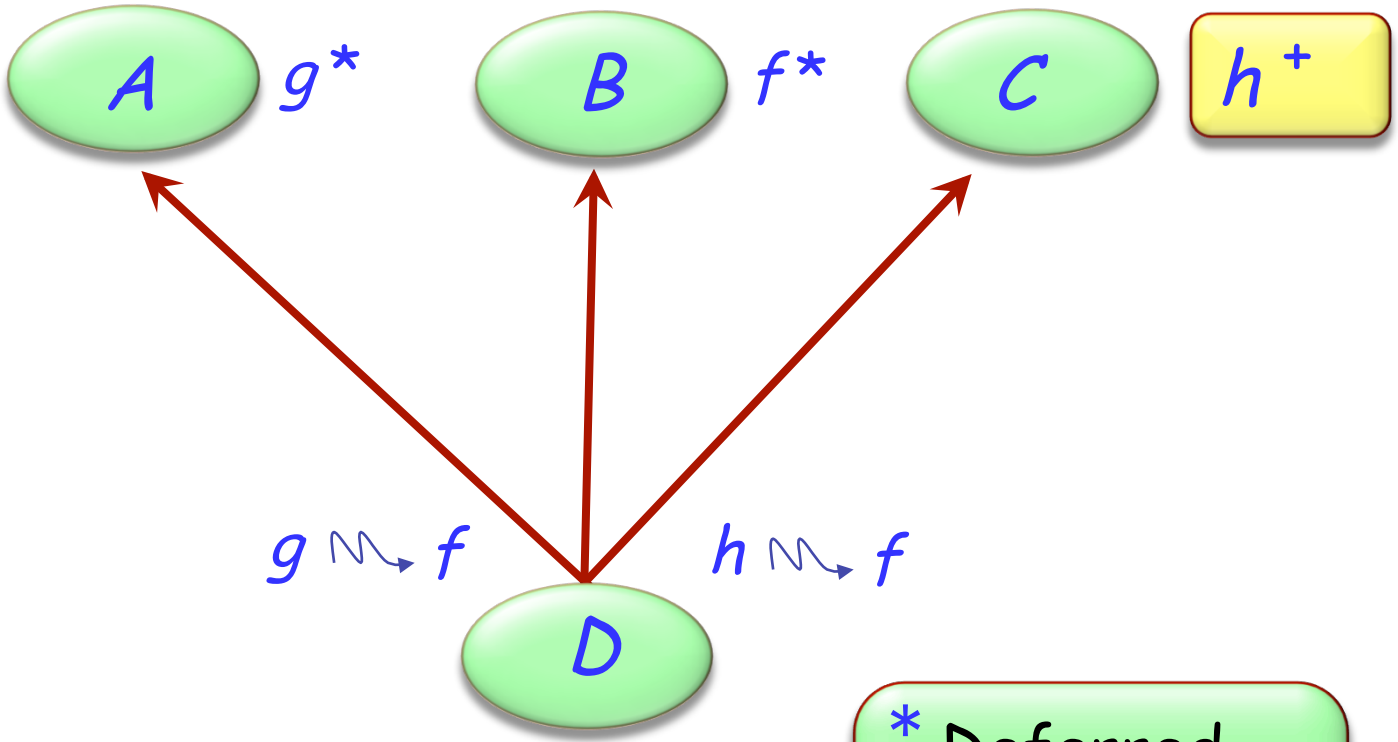
Feature merging



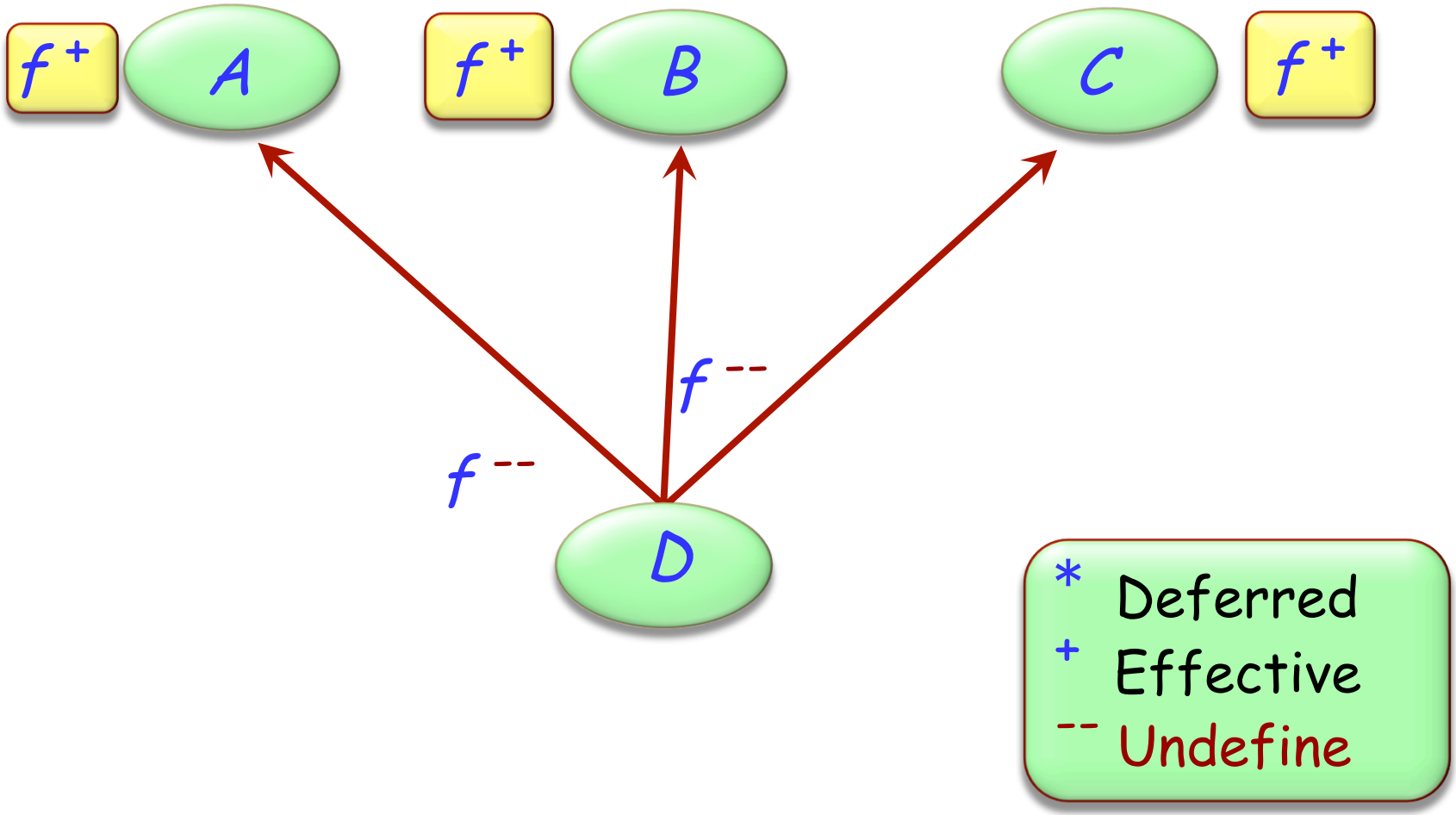
* Deferred
\+ Effective

Feature merging: with different names

```
class
   D
inherit
   A
         rename
            g as f
         end
   B
   C
         rename
            h as f
         end
feature
   ...
end
```



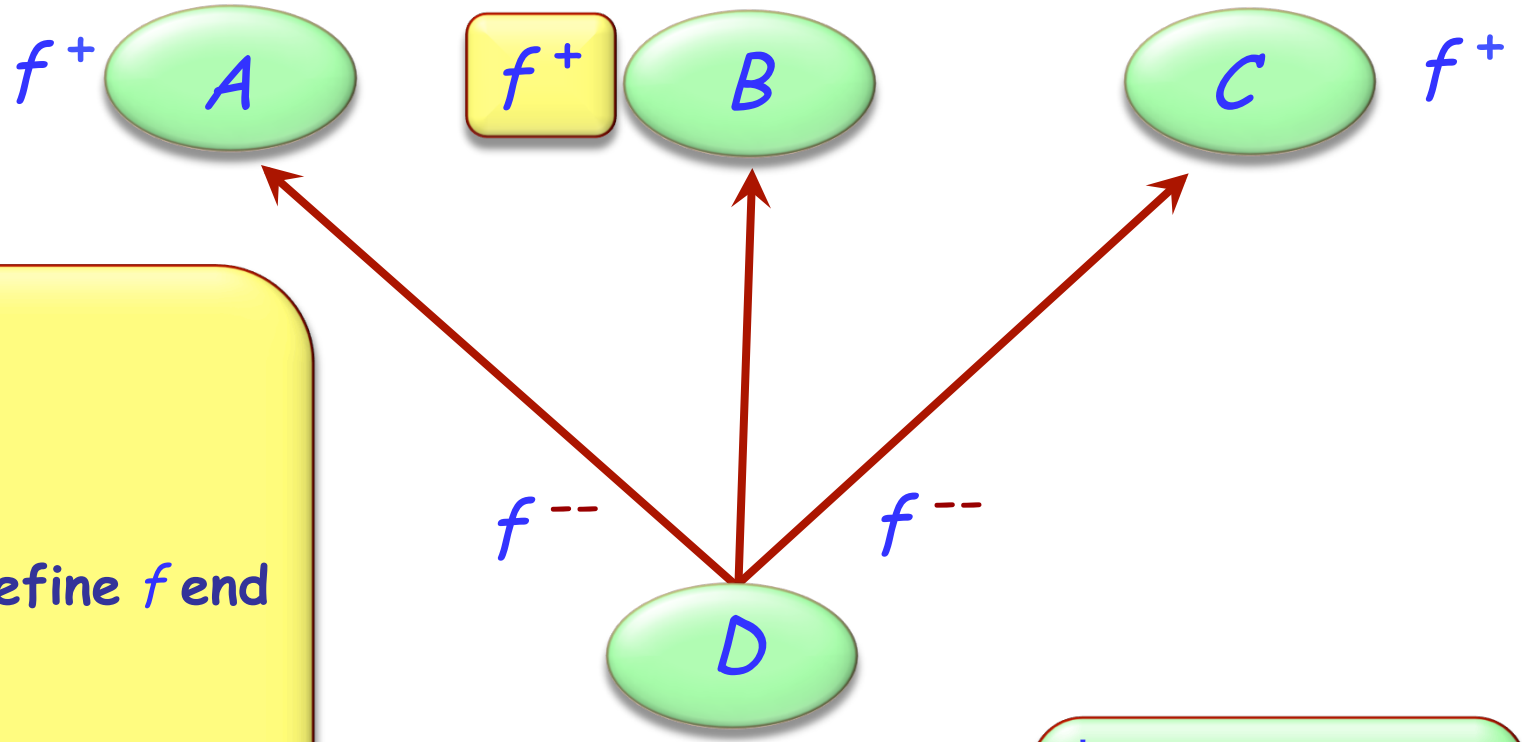
Feature merging: effective features



Undefinition

```
deferred class
    T
inherit
    S
        undefine v end
feature
        ...
end
```

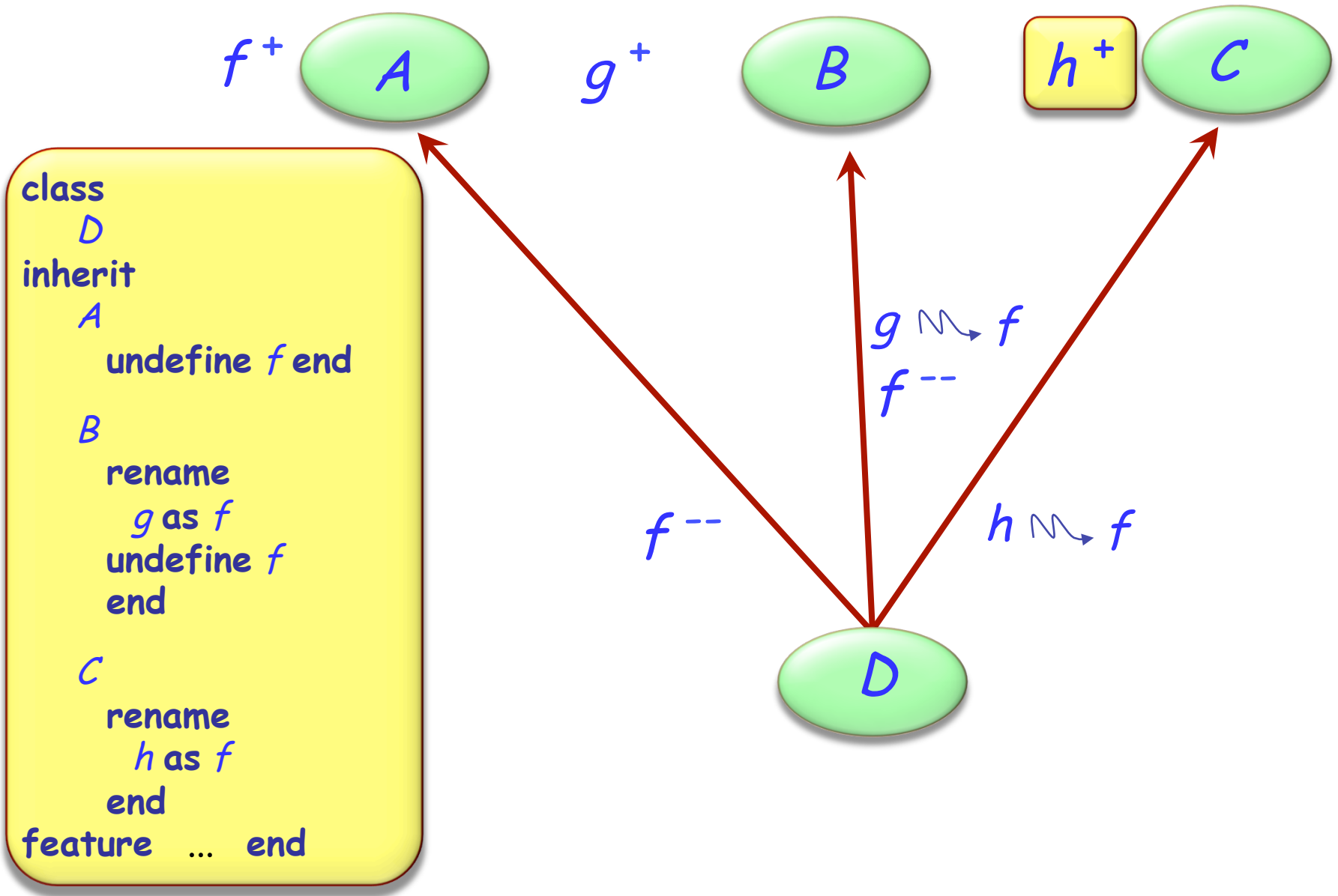
Merging through undefinition




```
class
    D
inherit
    A
        undefine f end
    B
    C
        undefine f end
feature
        ...
end
```

* Deferred
\+ Effective
-- Undefine

Merging effective features with different names



Acceptable name clashes

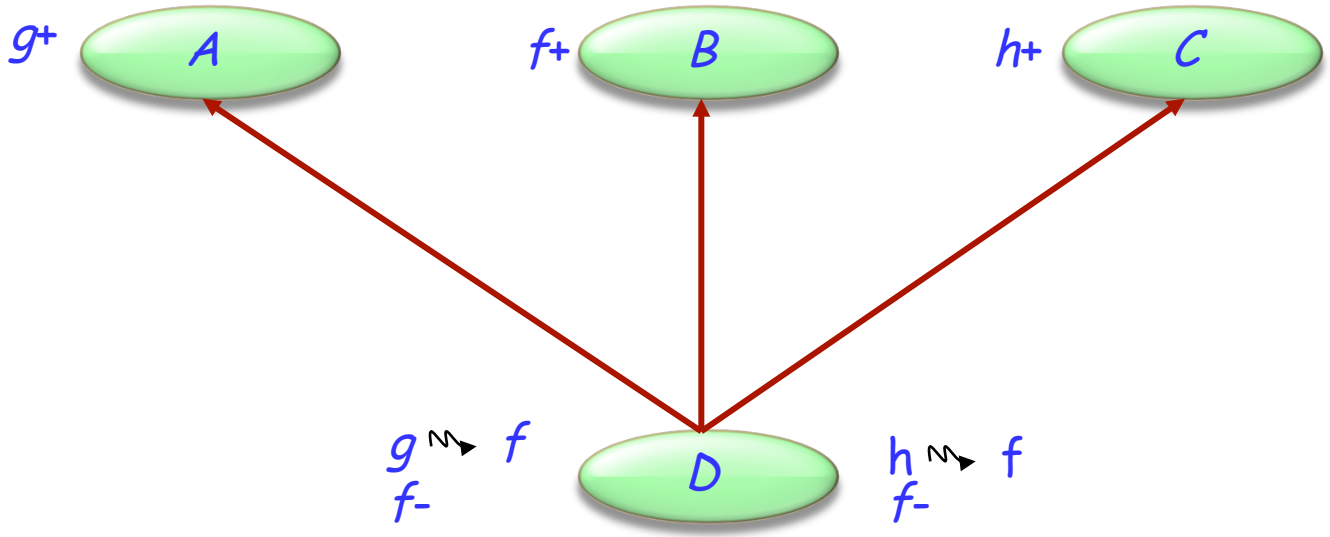
If inherited features have all the same names, there is no harmful name clash if:

- They all have compatible signatures
- At most one of them is effective

Semantics of such a case:

- Merge all features into one
- If there is an effective feature, it imposes its implementation

Feature merging: effective features



a1: A
a1.g

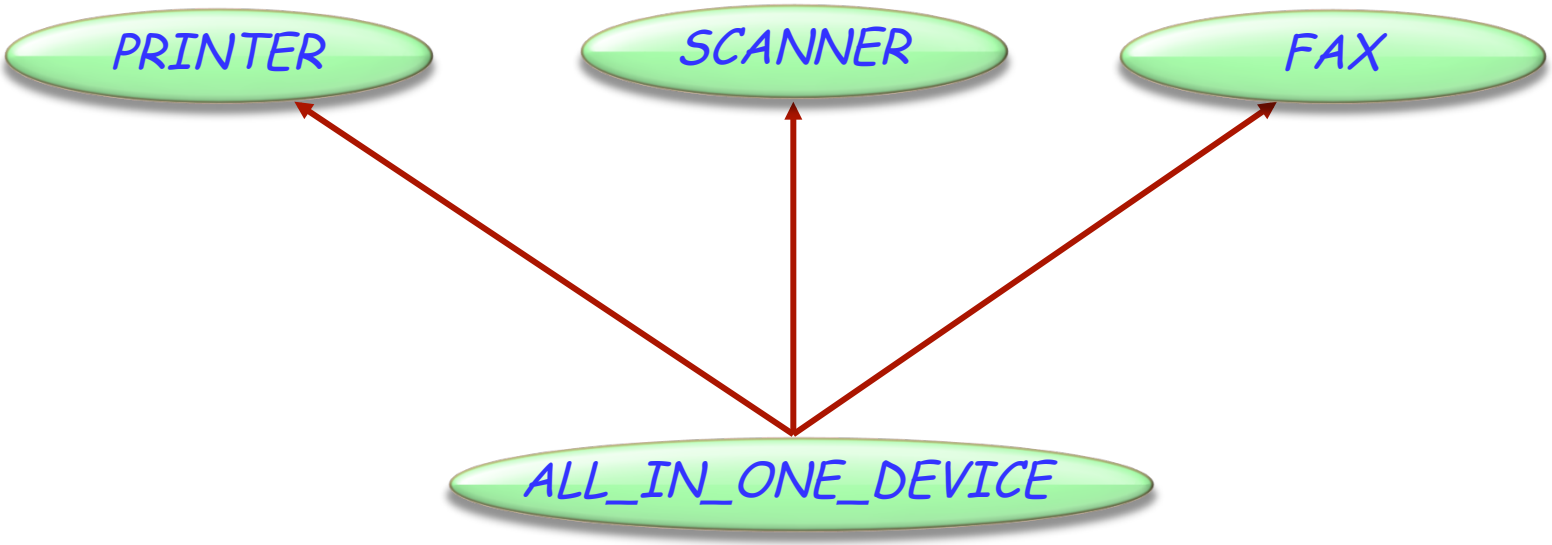
b1: B
b1.f

c1: C
c1.h

d1: D
d1.f

Exercise: All-in-one-device

Hands-On



Exercise: All-in-one-device

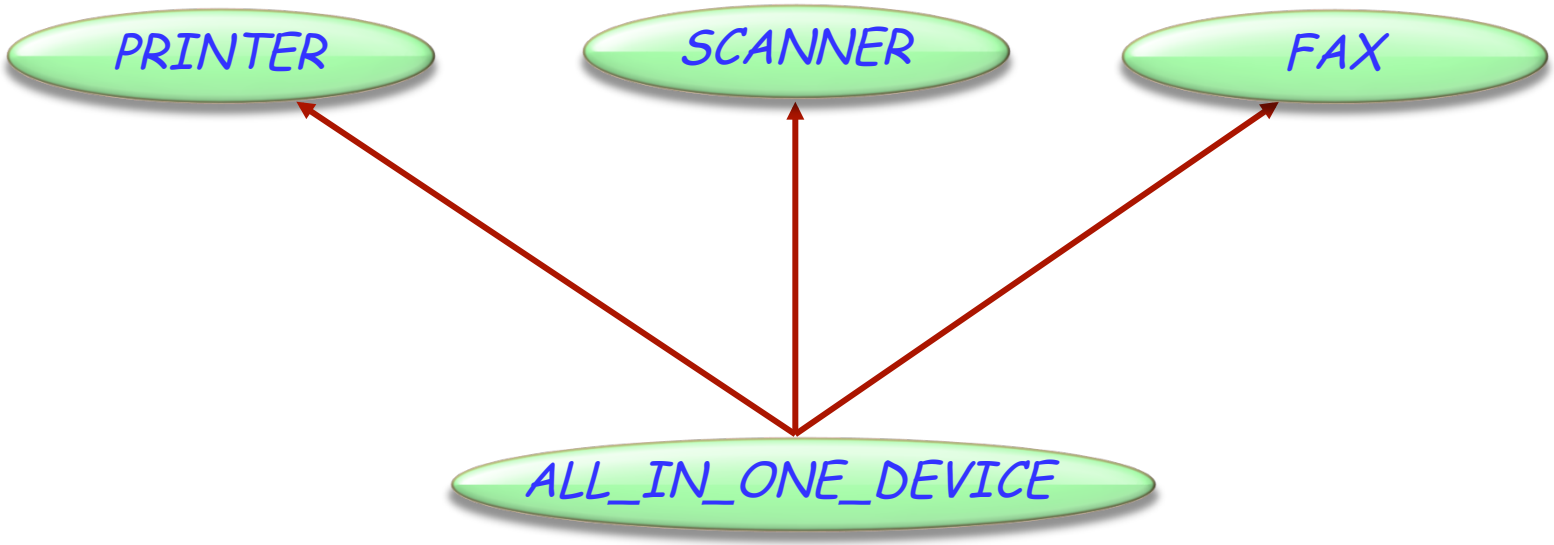
Hands-On

```
class PRINTER
feature
    print_page  -- Print a page.
        do
            print ("Printer prints a page...")
        end

    switch_on  -- Switch from 'off' to 'on'
        do
            print ("Printer switched on...")
        end

end

class FAX
feature
    send  -- Send a page over the phone net.
        do
            print ("Fax sends a page...")
        end

    start  -- Switch from 'off' to 'on'
        do
            print ("Fax switched on...")
        end

end
```

```
class SCANNER

feature
    scan_page  -- Scan a page.
        do
            print ("Scanner scans a page...")
        end

    switch_on  -- Switch from 'off' to 'on'
        do
            print ("Scanner switched on...")
        end

    send  -- Send data to PC.
        do
            print ("Scanner sends data...")
        end

end
```

Exercise: All-in-one-device

Hands-On




```
class
    ALL_IN_ONE_DEVICE

inherit
    ...

end
```

How to resolve the name clashes?

- switch_on
- send

Exercise: All-in-one-device


```
class ALL_IN_ONE_DEVICE

inherit
    PRINTER
        rename
            switch_on as start
        undefine
            start
        end

    SCANNER
        rename
            switch_on as start,
            send as send_data
        end

    FAX
        rename
            send as send_message
        undefine
            start
        end

feature  ...  end
```

Valid or invalid?

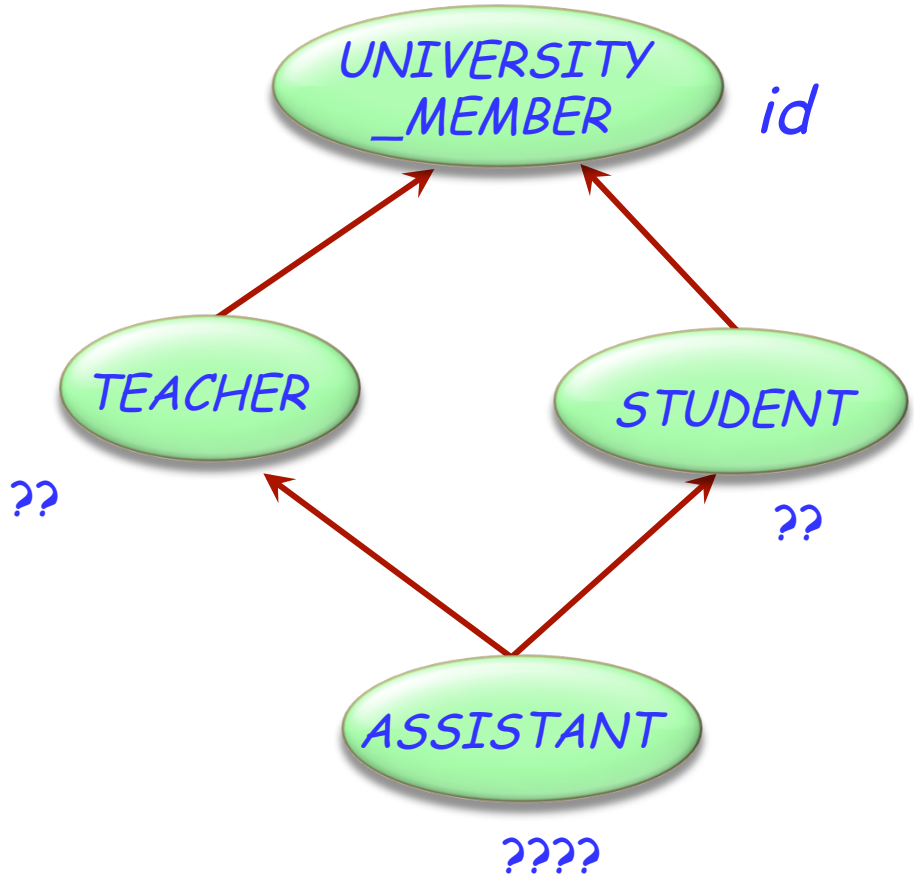
Hands-On

```
class ALL_IN_ONE_DEVICE

inherit
    PRINTER
        rename
            switch_on as start
        undefine
            start
        end

    SCANNER
        rename
            switch_on as start,
            send as send_data
        end

    FAX
        rename
            send as send_message
        undefine
            start
        end

feature    ...    end
```

s: SCANNER
f: FAX
a: ALL_IN_ONE_DEVICE

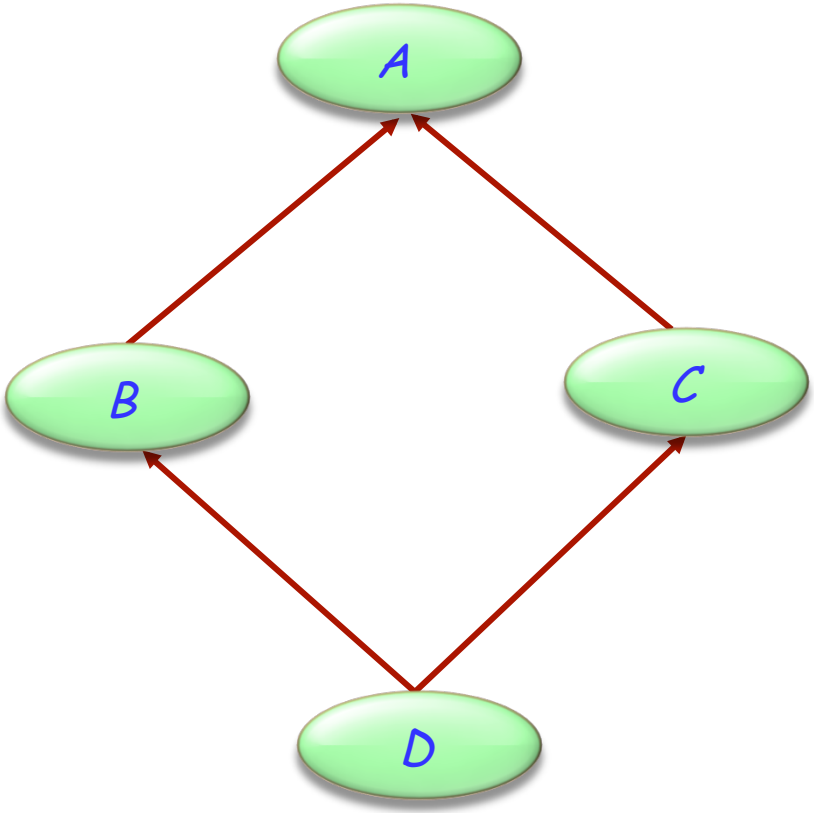
- a.switch_on — Invalid
- a.print_page — Valid
- f.send_message — Invalid
- s.switch_on — Valid
- f.send — Valid
- a.send — Invalid

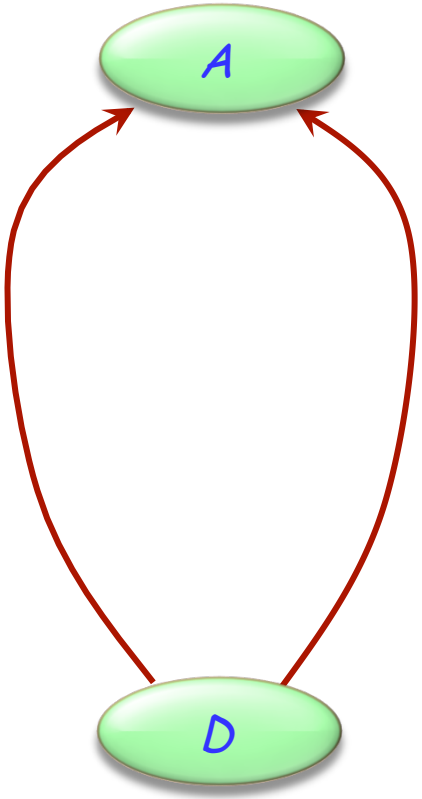
A special case of multiple inheritance



This is a case of repeated inheritance

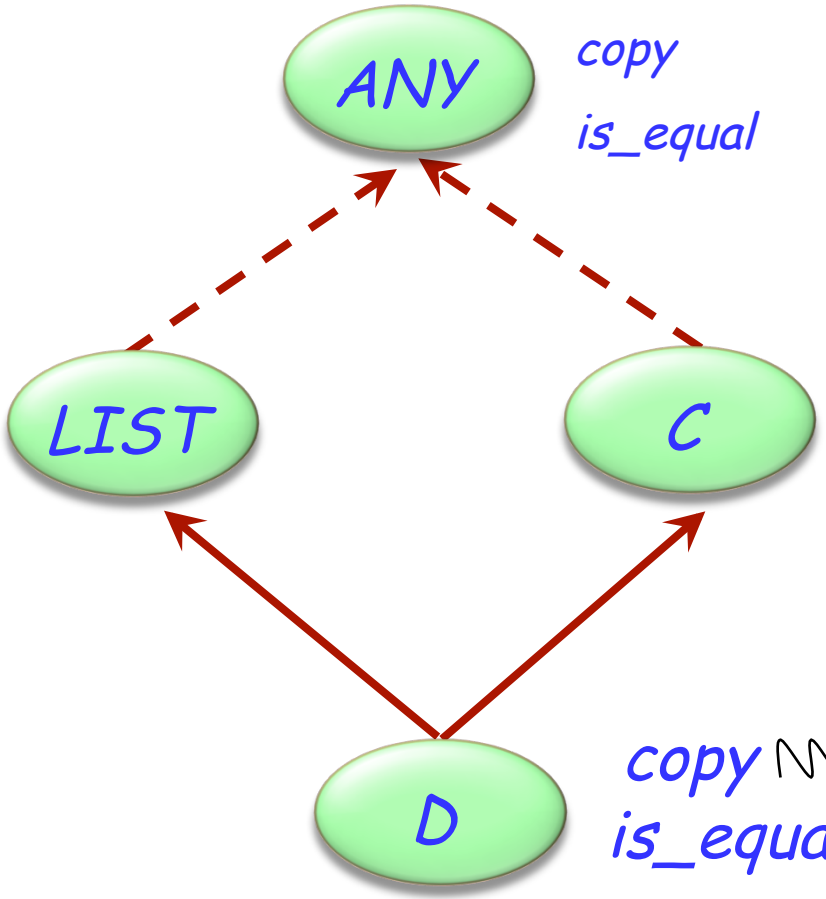
Indirect and direct repeated inheritance



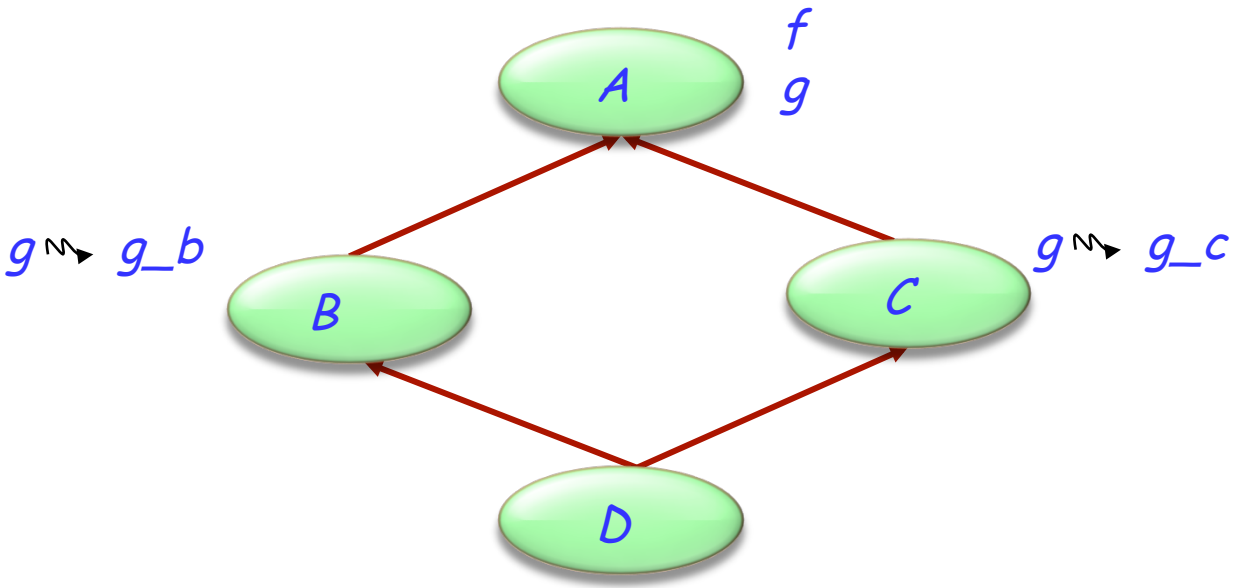


Multiple is also repeated inheritance

A typical case:



Sharing and replication



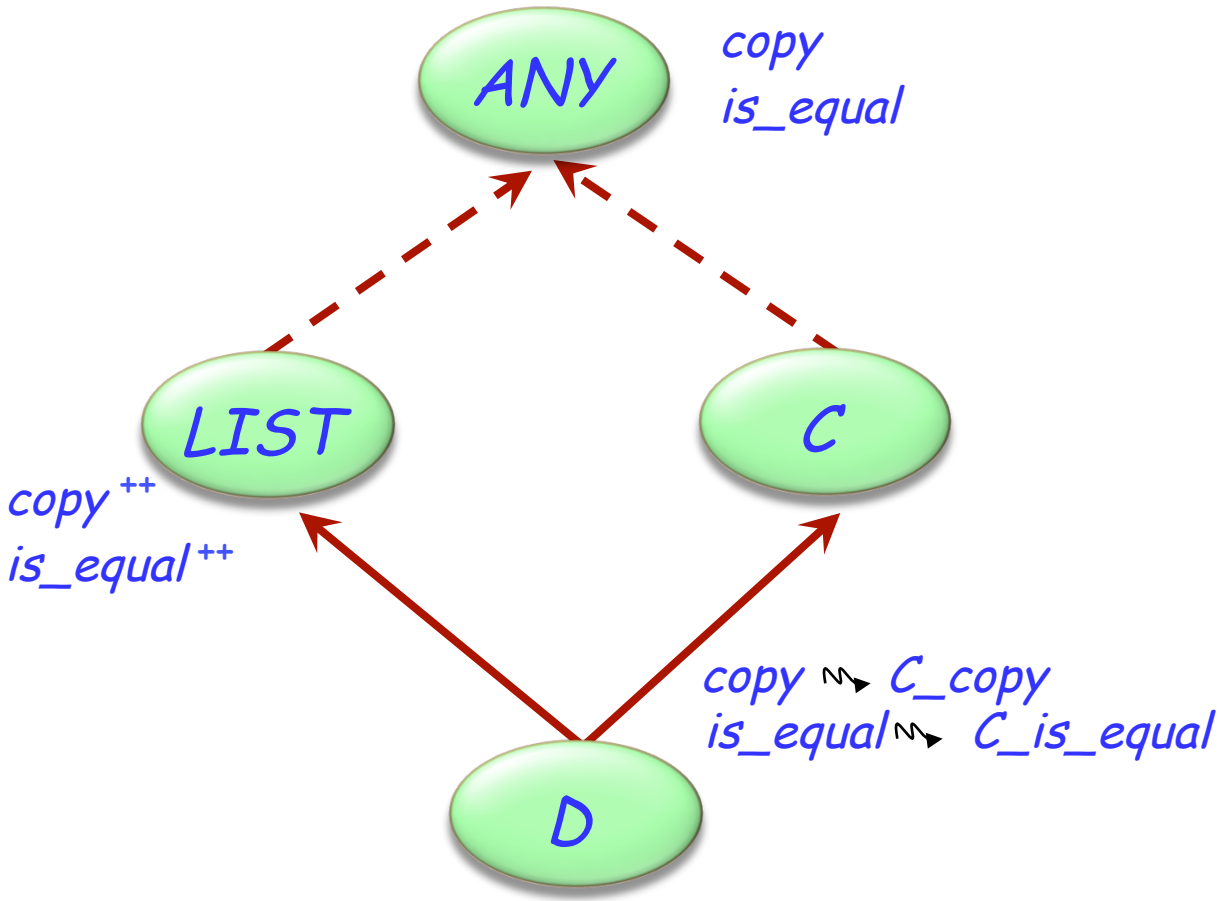
Features such as *f*, not renamed along any of the inheritance paths, will be shared.

Features such as *g*, inherited under different names, will be replicated.

The need for select

A potential ambiguity arises because of polymorphism and dynamic binding:

```
a1: ANY
d1: D

...

a1 := d1

a1.copy (...)
```



Removing the ambiguity

```
class
        D
inherit
        V_LIST [T]
                select
                        copy,
                        is_equal
                end

        C
                rename
                        copy as C_copy,
                        is_equal as C_is_equal,
                                ...
                end
```

When is a name clash acceptable?

(Between *n* features of a class, all with the same name, immediate or inherited.)

- They must all have compatible signatures.
- If more than one is effective, they must all come from a common ancestor feature under repeated inheritance.