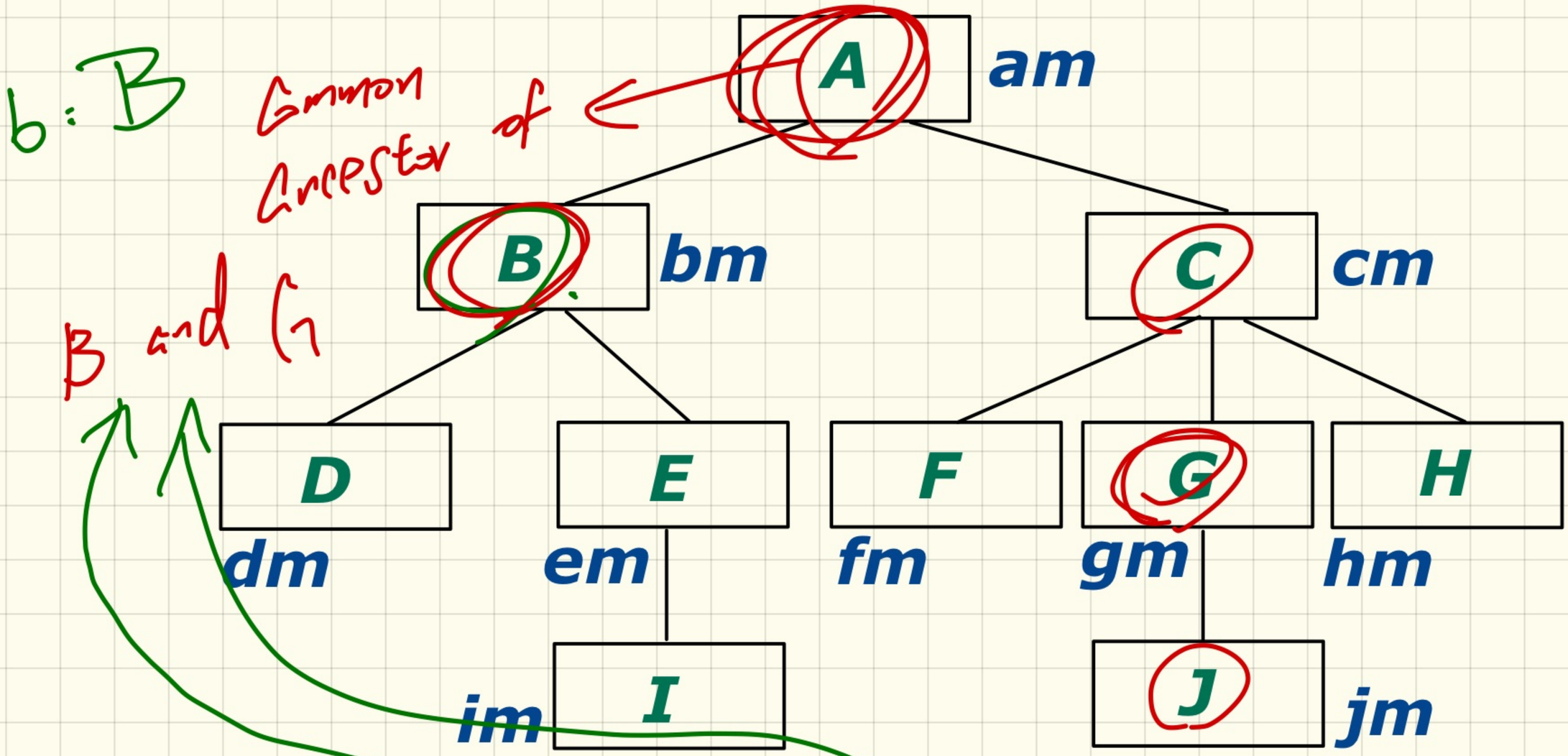
Lecture 11

Thursday October 10

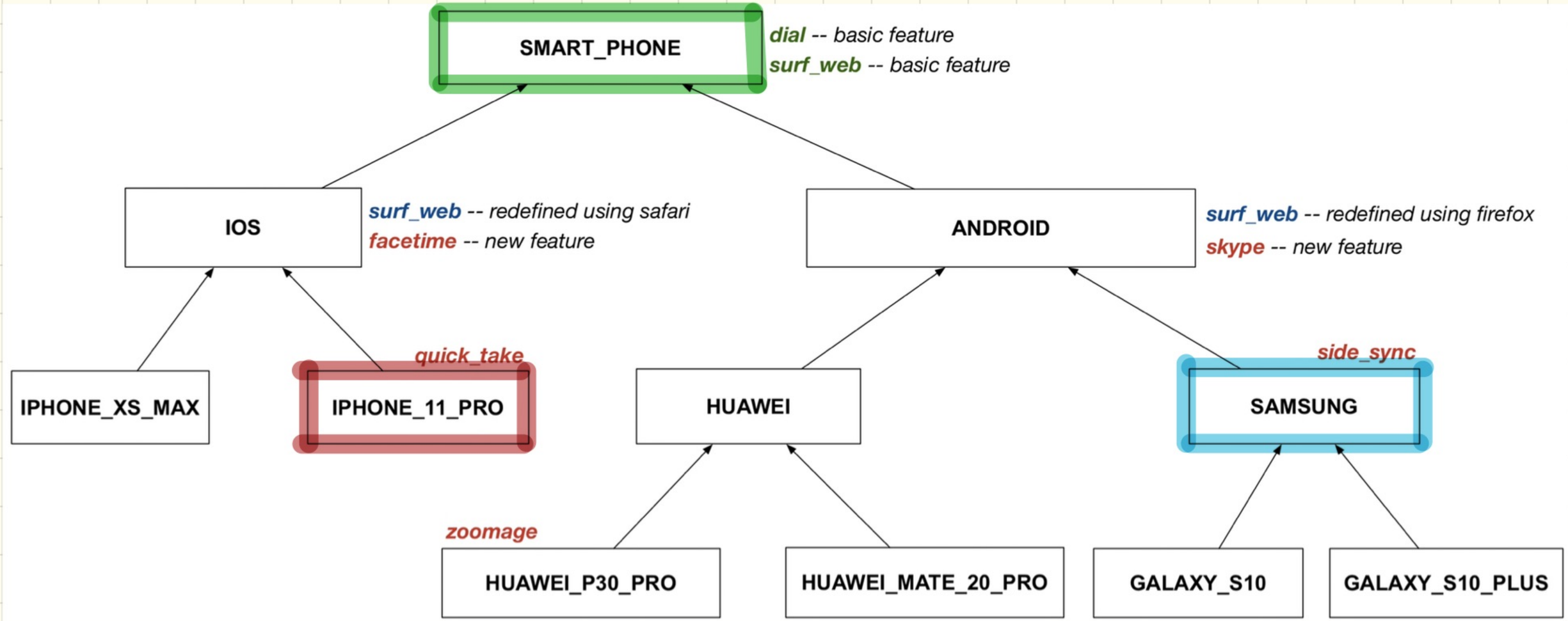
Inheritance Forms a Type Hierarchy (1)



	ancestors	expectations	descendants
B	B, A	bm	B D E I
G	G C A	gm cm am	
J	J G C A	jm gm cm am	

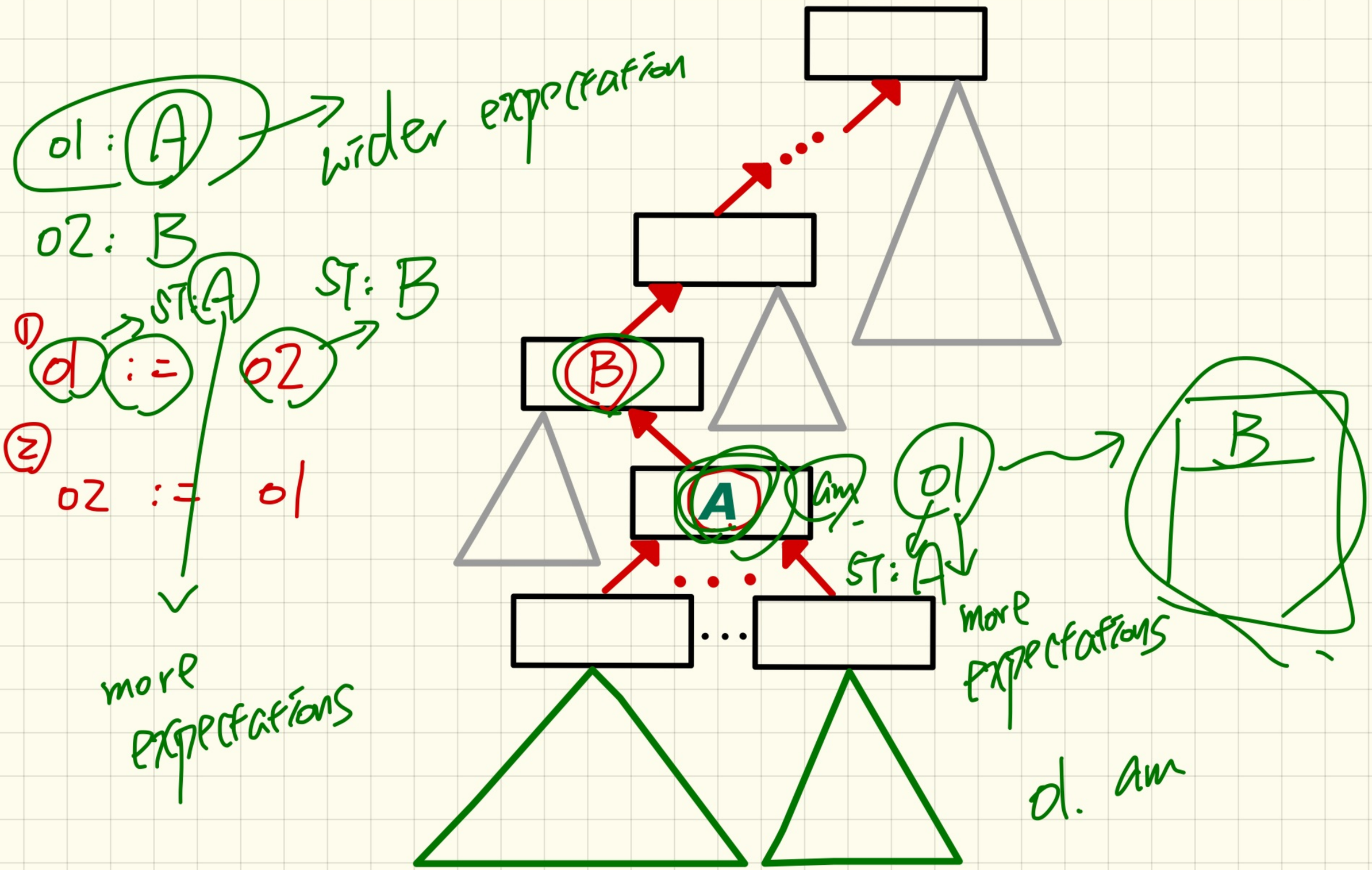
∴ G is an ancestor of J
E(G) ⊆ E(J)

Inheritance Forms a Type Hierarchy (2)



	ancestors	expectations	descendants

Ancestors, Expectations, Descendants, and Code Reuse



o1 : B

Create {B} o1. make

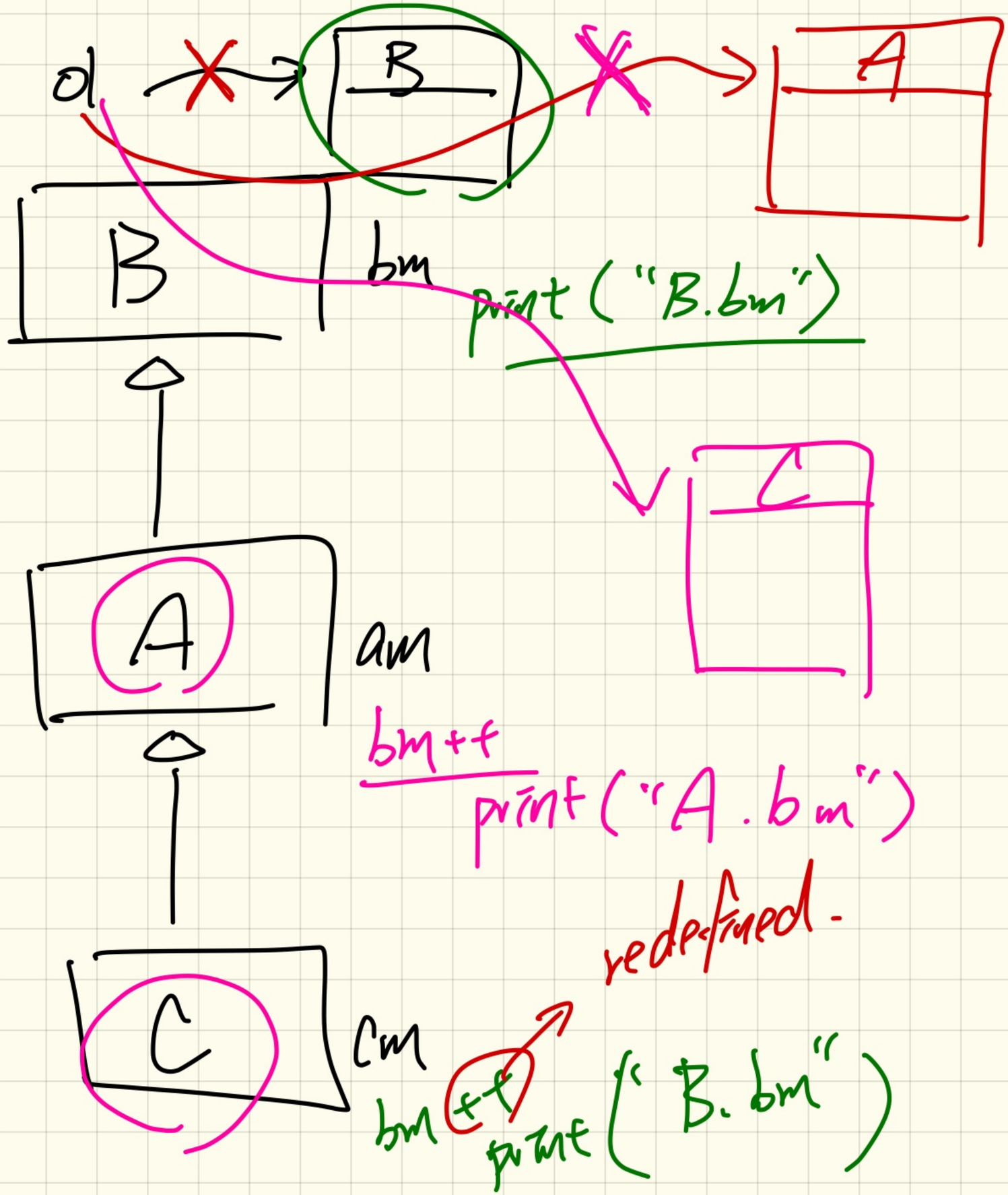
o1. bm → B. bm

Create {A} o1. make

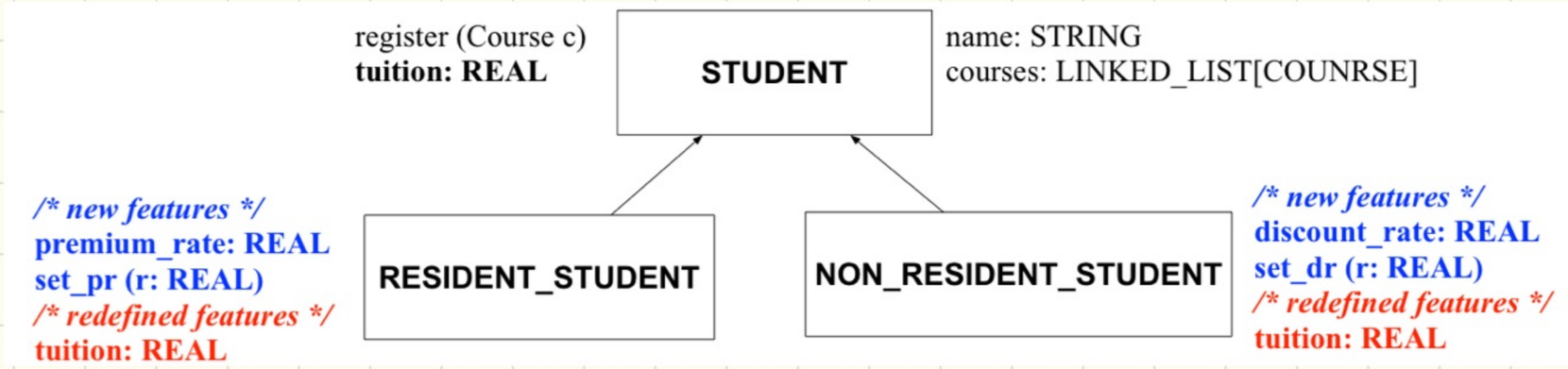
o1. bm → A. bm

Create {C} o1. make

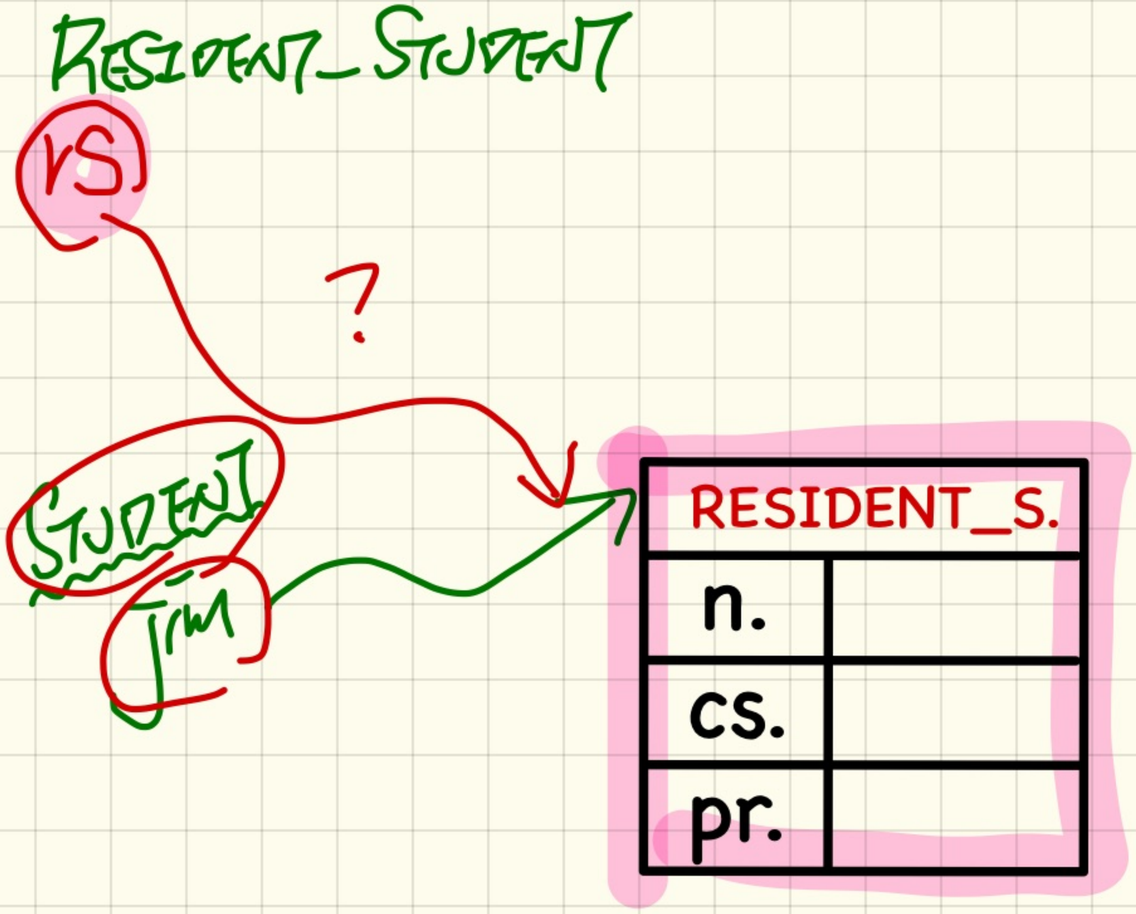
o1. bm → A. bm

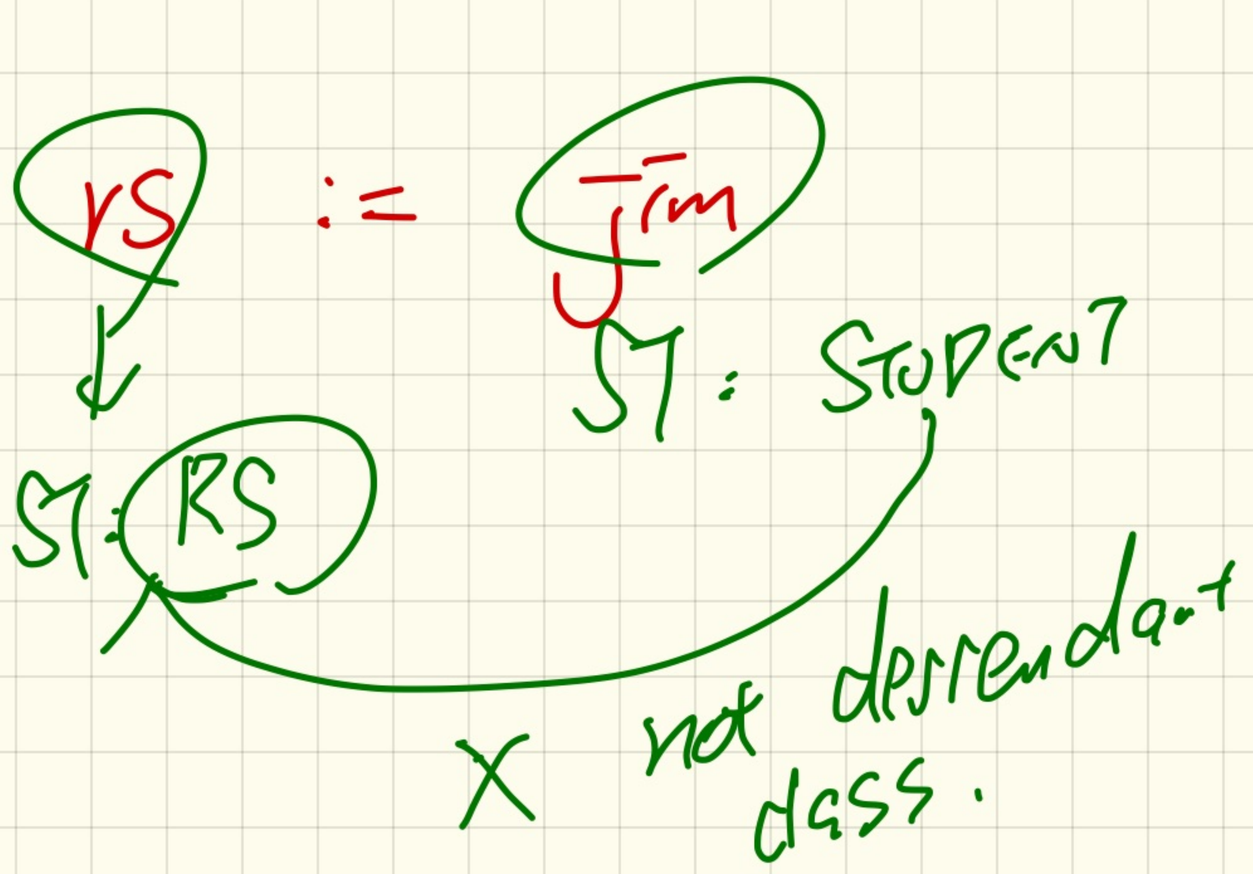


Type Cast: Motivation

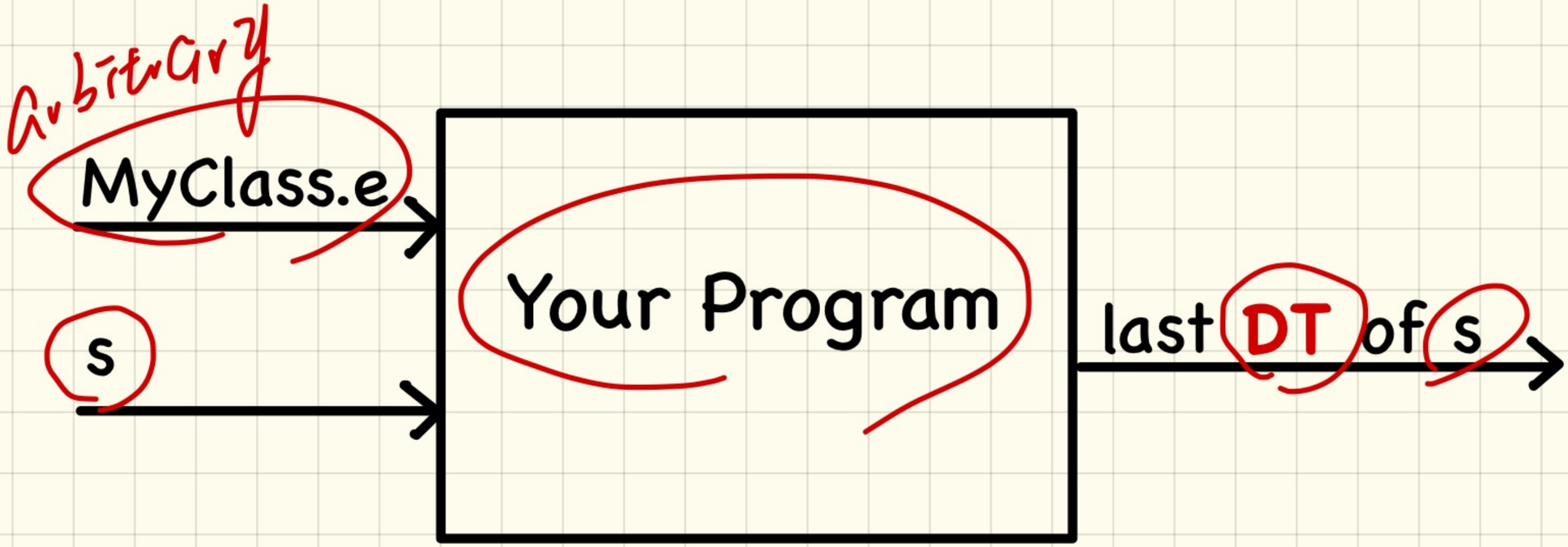



```
1  local jim: STUDENT; rs: RESIDENT_STUDENT
2  do create {RESIDENT_STUDENT} jim.make ("J. Davis")
3     rs := jim
4     rs.setPremiumRate(1.5)
```

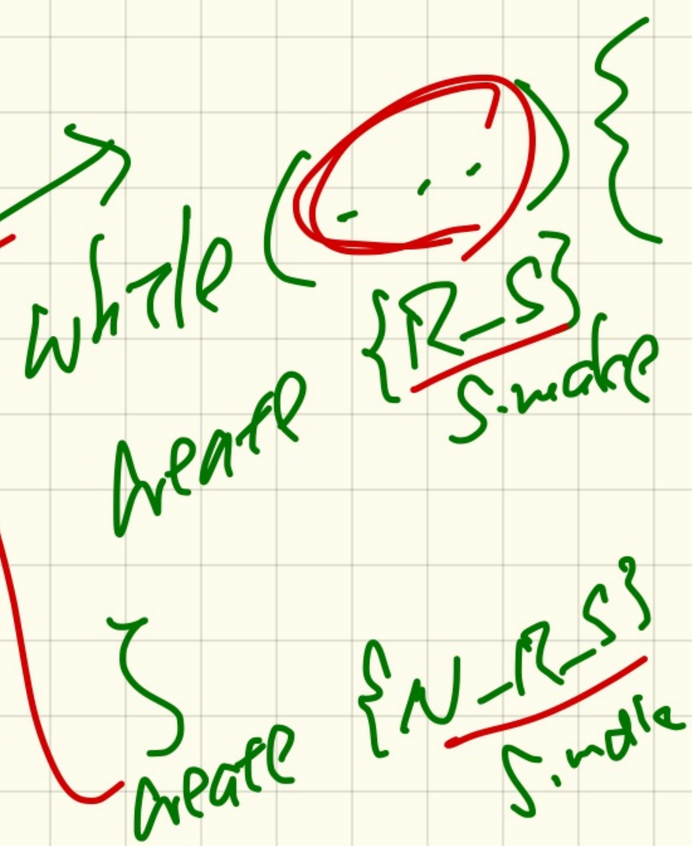




Inferring the DT of a Variable is Undecidable




```
class MyClass
    make
        local
            s: STUDENT
        do
            create {RESIDENT_STUDENT} s.make
        end
end
```



Type Cast: Syntax

```
1  check attached {RESIDENT_STUDENT} jim as rs_jim then
2    rs := rs_jim
3    rs.set_pr (1.5)
4  end
```

rs := rs_jim

RESIDENT_STUDENT
rs_jim

STUDENT
jim

RESIDENT_S.	
n.	"J. Davies"
cs.	
pr.	1.5

1. no new object created

2. ST of jim was not modified

RS

rs

rs := rs_jim

ST: RS

ST: RS

Boolean Exp.

check attached {RS} from as rs_from

and

attached {NRS} alan as nrs_alan

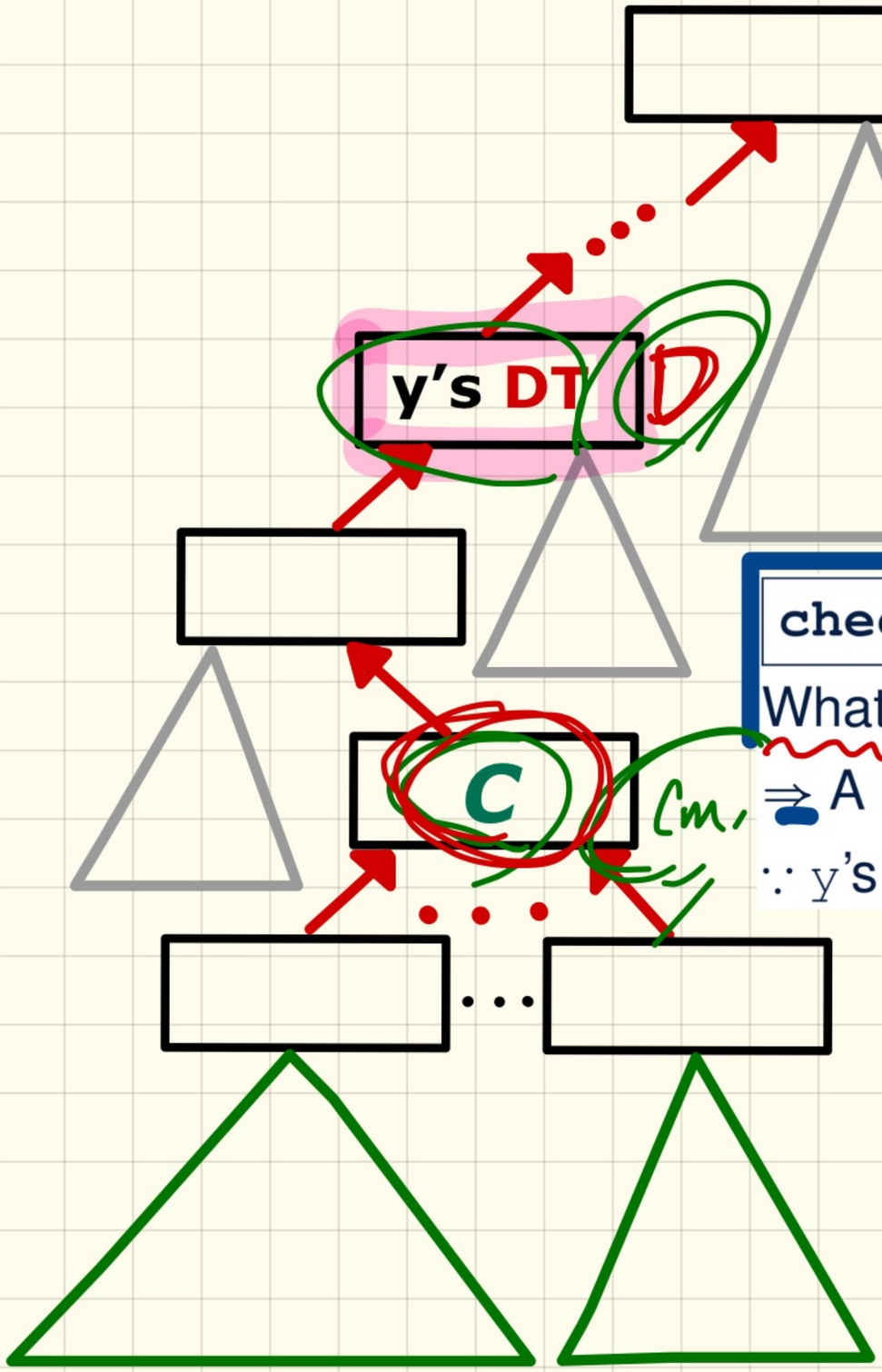
then

rs_from : RS

nrs_alan : NRS

end

Ancestors, Expectations, Descendants, and Code Reuse



create {D} y.make

Assume we allowed this cast

⇒ ST of c_y : C

c_y . Cm ⇔ c_y .

```
check attached {C} y then ... end
```

always compiles

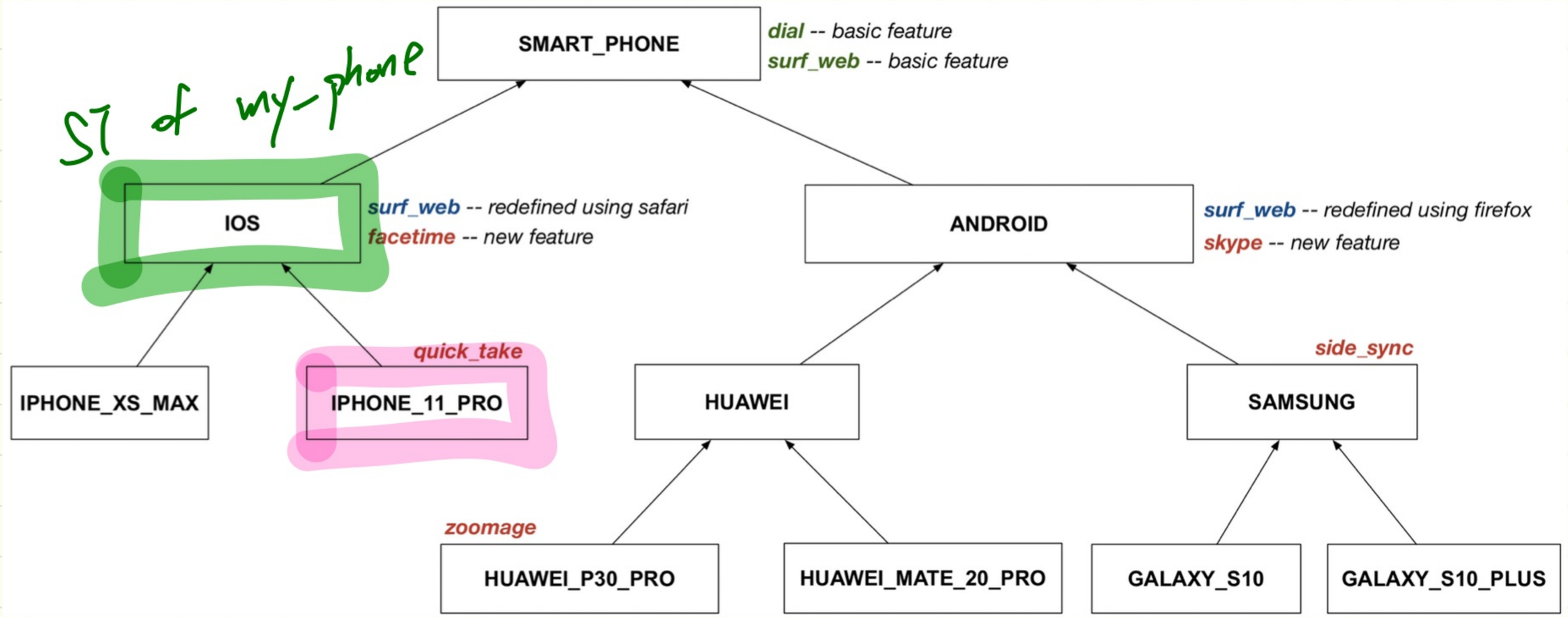
What if C is not an **ancestor** of y's ***DT***?

⇒ A ***runtime*** assertion violation occurs!

∵ y's ***DT*** cannot fulfill the expectation of C.

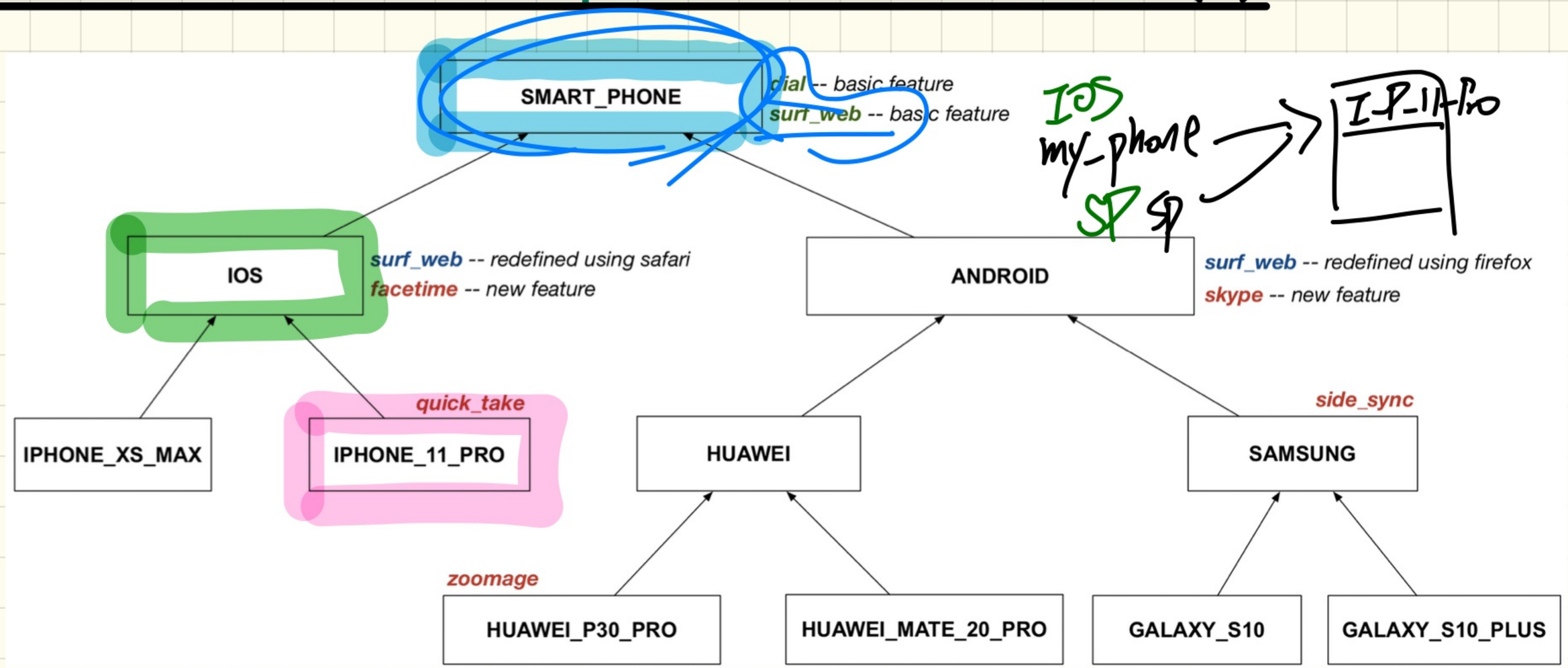
for a cast to be successful, "C" must be an ancestor of y's DT.

Violation-Free Cast: Upwards or Downwards (1)



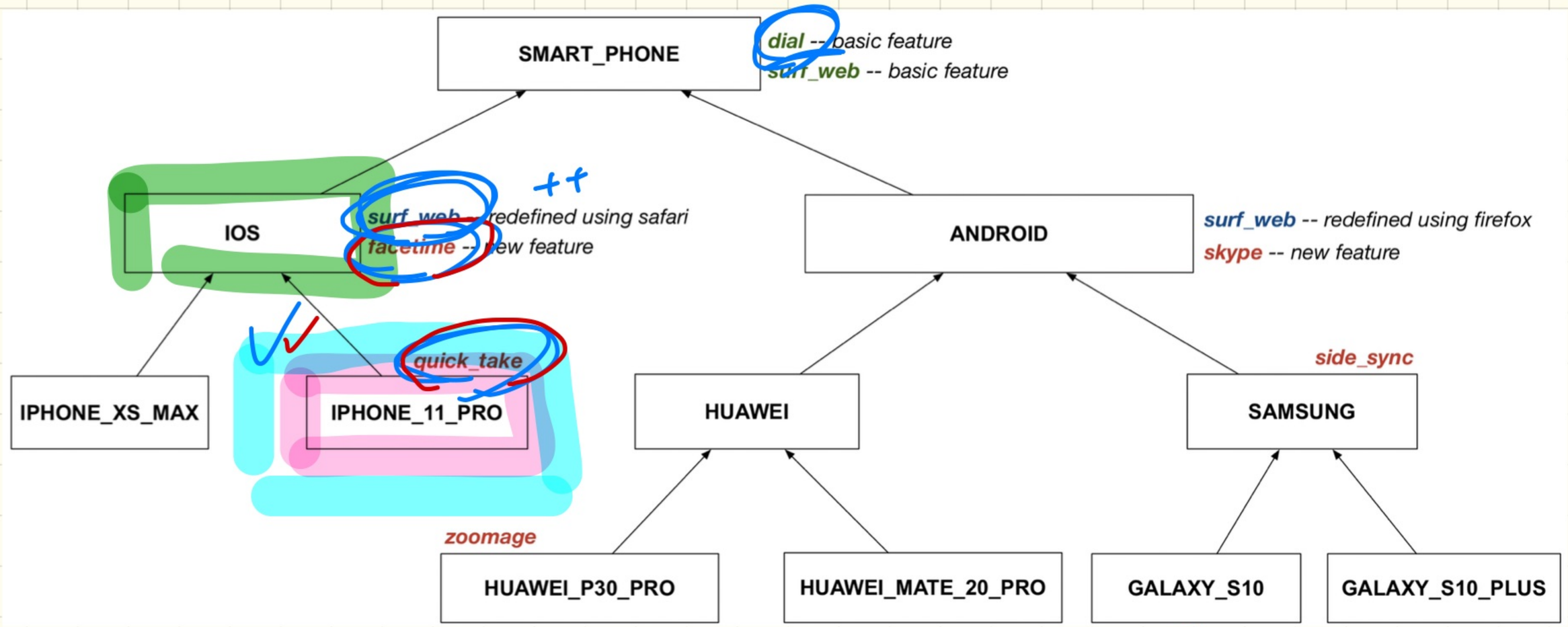

```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime ●  quick_take, skype, side_sync, zoomage ●
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web ●  facetime, quick_take, skype, side_sync, zoomage ●
end
check attached {IPHONE_11_PRO} my_phone as ip11_pro then
  -- can now call features defined in IPHONE_11_PRO on ip11_pro
  -- dial, surf_web, facetime, quick_take ●  skype, side_sync, zoomage ●
end
```

Violation-Free Cast: Upwards or Downwards (2)

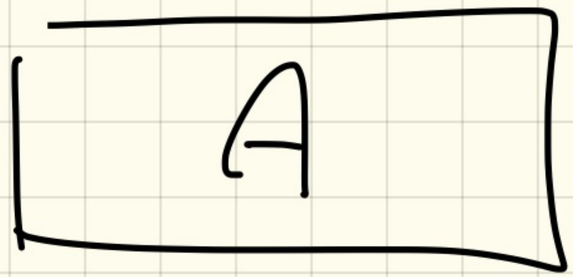



```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime, quick_take, skype, side_sync, zoomage
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web, facetime, quick_take, skype, side_sync, zoomage
end
check attached {IPHONE_11_PRO} my_phone as ip11_pro then
  -- can now call features defined in IPHONE_11_PRO on ip11_pro
  -- dial, surf_web, facetime, quick_take, skype, side_sync, zoomage
end
```

Violation-Free Cast: Upwards or Downwards (3)

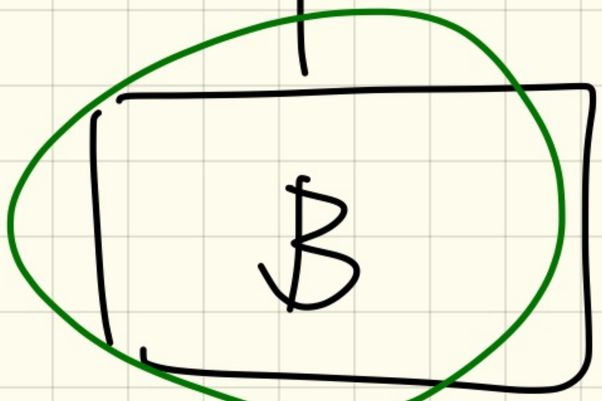



```
my_phone: IOS
create {IPHONE_11_PRO} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web, facetime, quick_take, skype, side_sync, zoomage
end
check attached {IPHONE_11_PRO} my_phone as ip11_pro then
  -- can now call features defined in IPHONE_11_PRO on ip11_pro
  -- dial, surf_web, facetime, quick_take, skype, side_sync, zoomage
end
```



am

print ("A.am")



am + f

bm

print ("B.am")

o : B

Create {B} o.make

o.am → "B.am"

check {A} o as a_o then

ST: A

a_o.am → "B.am"

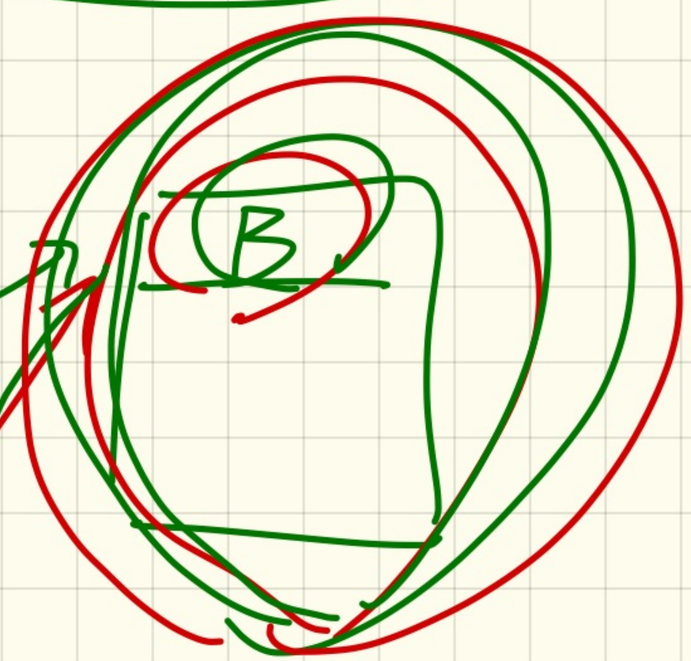
end

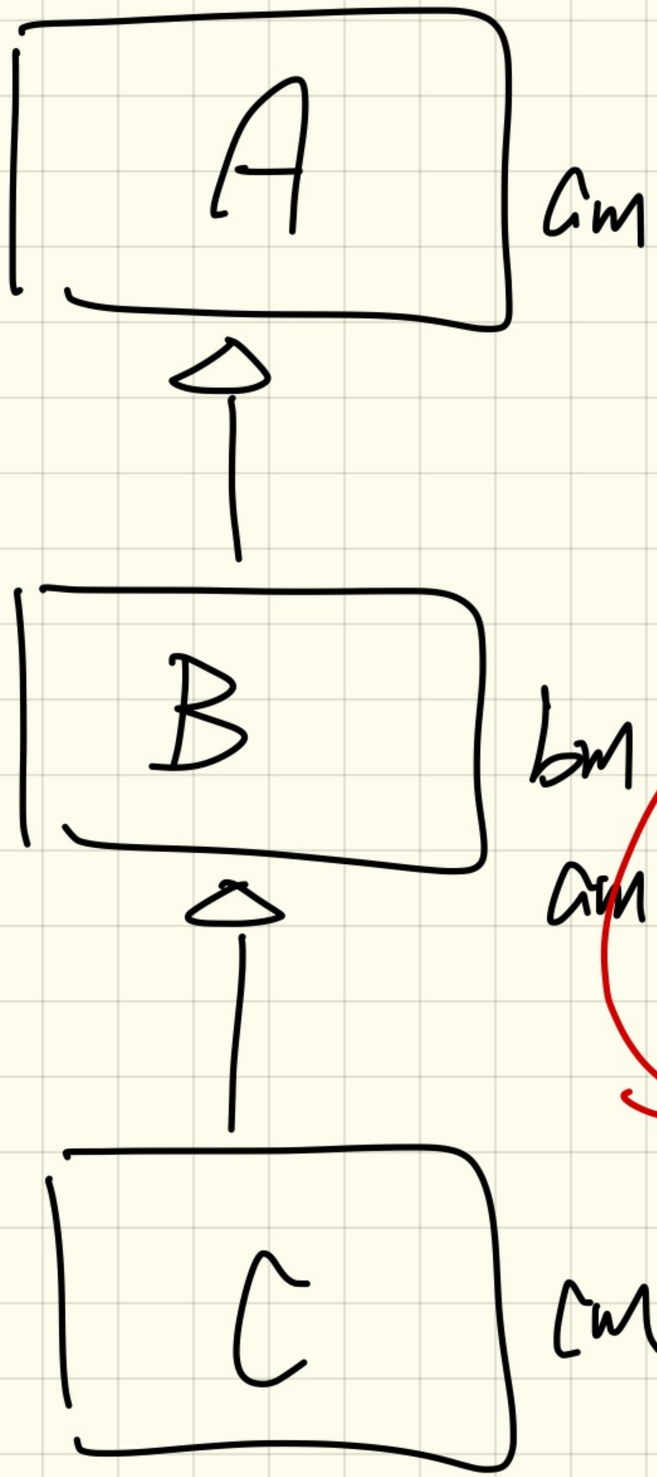
ST: B

o

a_o

ST: A





b: ?A {C}

check attached b as C

end

whether this cast succeeds or surface depends on if C is an ancestor of b's DT.

Does a line compile?

SI

Version of feature called?

DI.