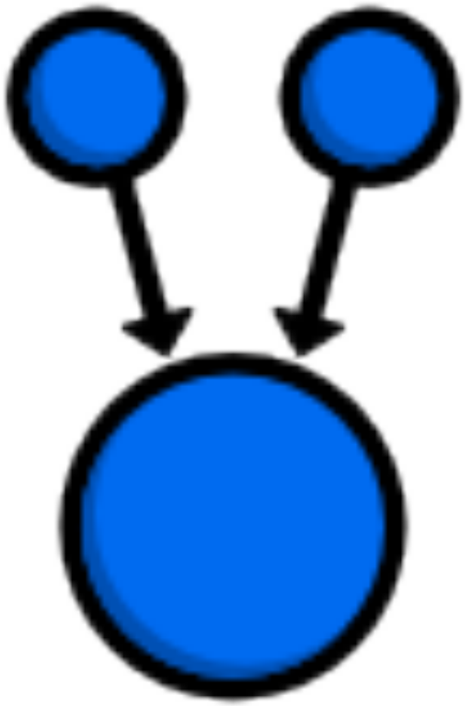
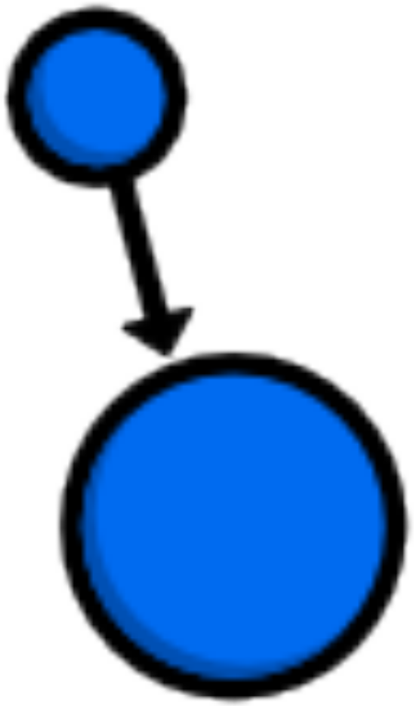
Gerenciamento de Memória

Contagem de referências

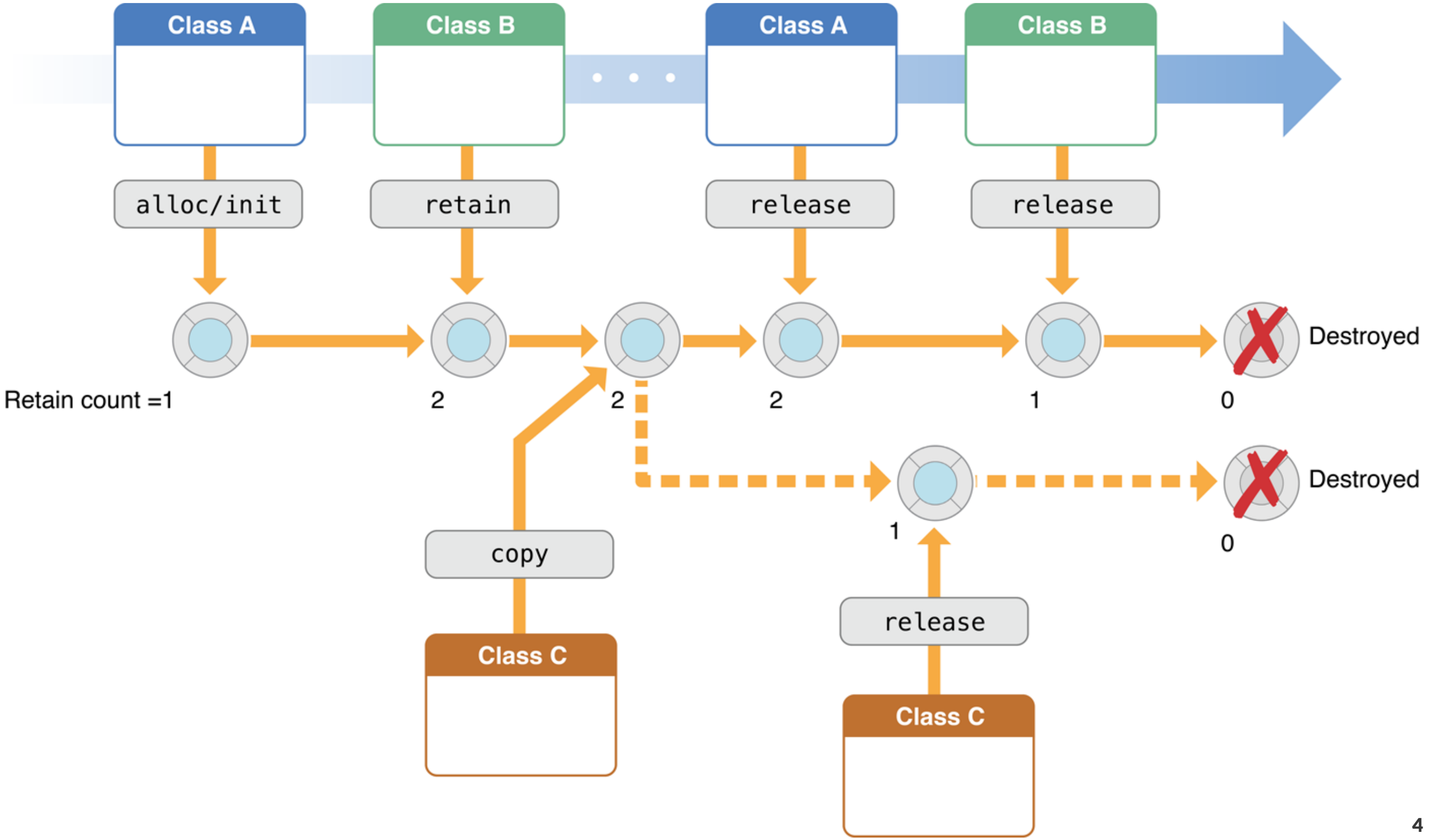


Reference
Count: 2



Reference
Count: 1

Reference
Count: 0


Class A
Class B
Class A
Class B
alloc/init
retain
release
release
Retain count =1
2
2
2
1
0
Destroyed
copy
Class C
1
0
Destroyed
release
Class C

http://whentouseretaincount.com

Never

There's pretty much never a good reason to use `-retainCount`. Here's a short and mildly abusive explanation why.

What about when I need to *x*?

No.

But so-and-so said it was a good idea in this case?

Still no.

I used it in some code already and it worked fine!

Did it, by any chance, look like this?

```
while ([a retainCount] > 0) {
 [a release];
}
```

Good luck with that. Please let me know which app so I can avoid buying it.

Okay, okay. I get the idea, but why?

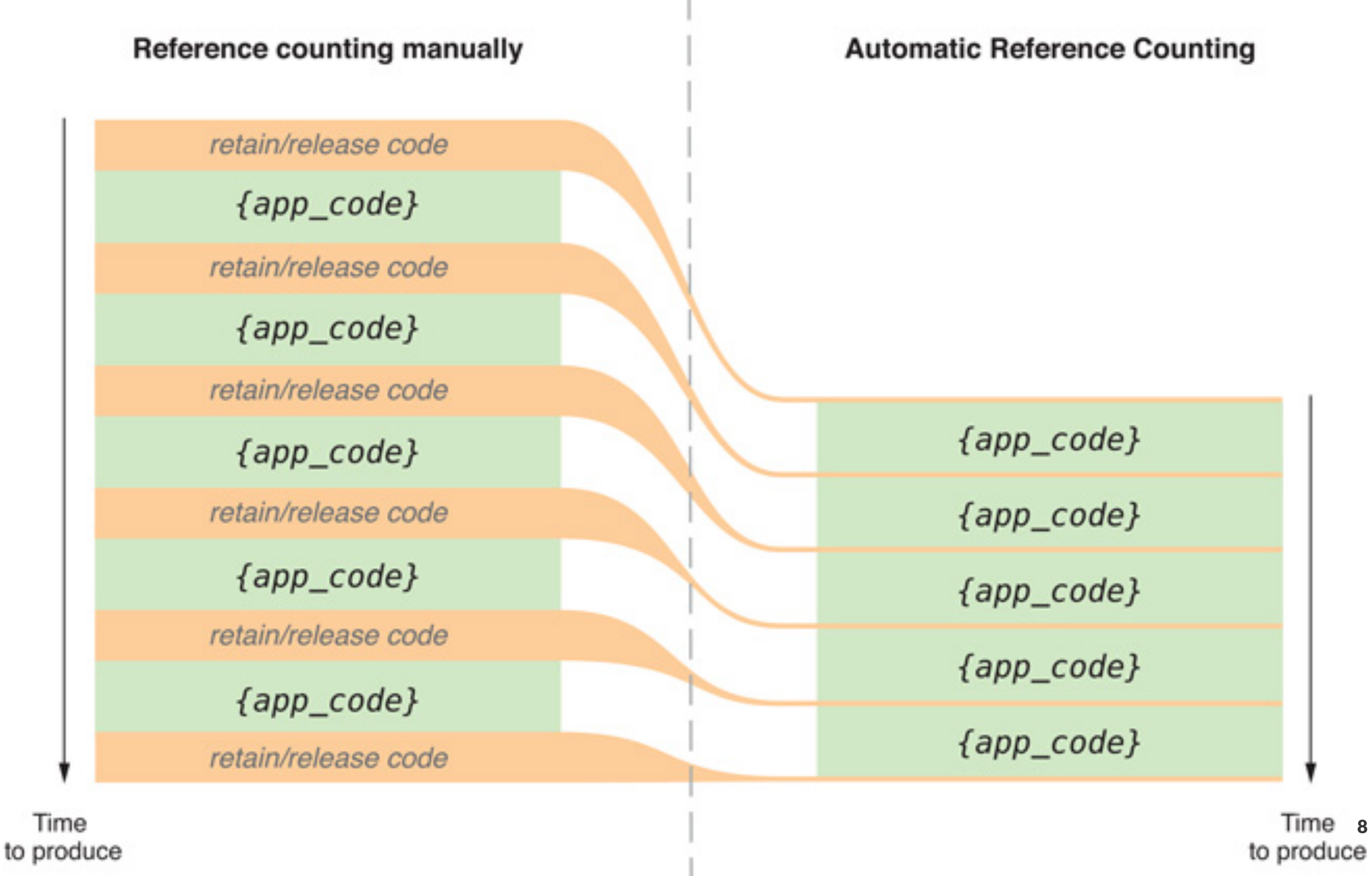
Basically it doesn't say what you think it does. Or rather, it does but it's never accurate in any non-trivial case.

For example:

- You'd think that `[NSNumber numberWithInt:1]` would have a retain count of 1. It doesn't. It's 2.
- You'd think that `@"Foo"` would have a retain count of 1. It doesn't. It's 1152921504606846975.
- You'd think that `[NSString stringWithString:@"Foo"]` would have a retain count of 1. It doesn't. Again, it's 1152921504606846975.

(H/T to Dave DeLong for these.)


ARC


Reference counting manually
Automatic Reference Counting
retain/release code
{app_code}
retain/release code
{app_code}
retain/release code
{app_code}
retain/release code
{app_code}
retain/release code
{app_code}
retain/release code
{app_code}
{app_code}
{app_code}
{app_code}
{app_code}
{app_code}
Time to produce
Time to produce

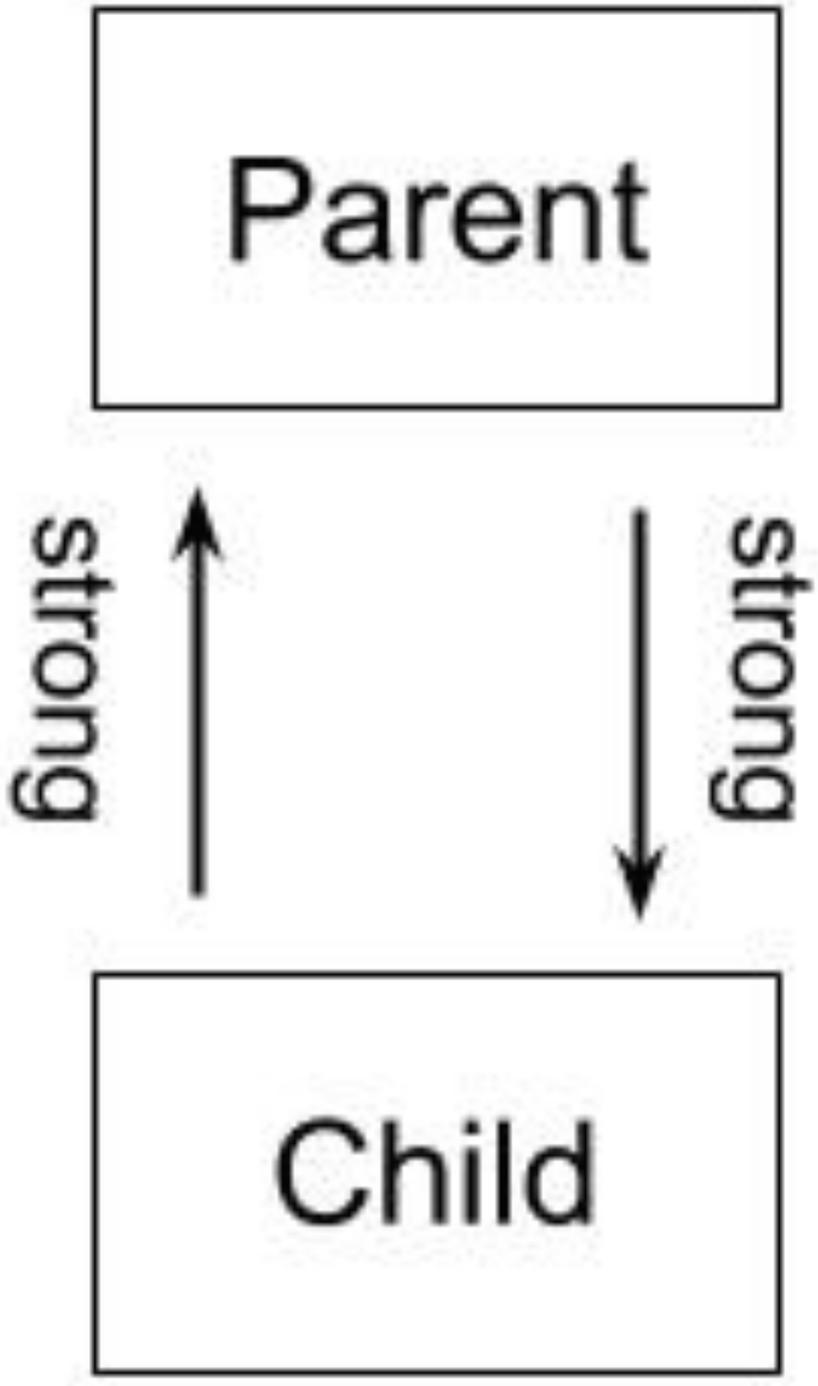
ARC

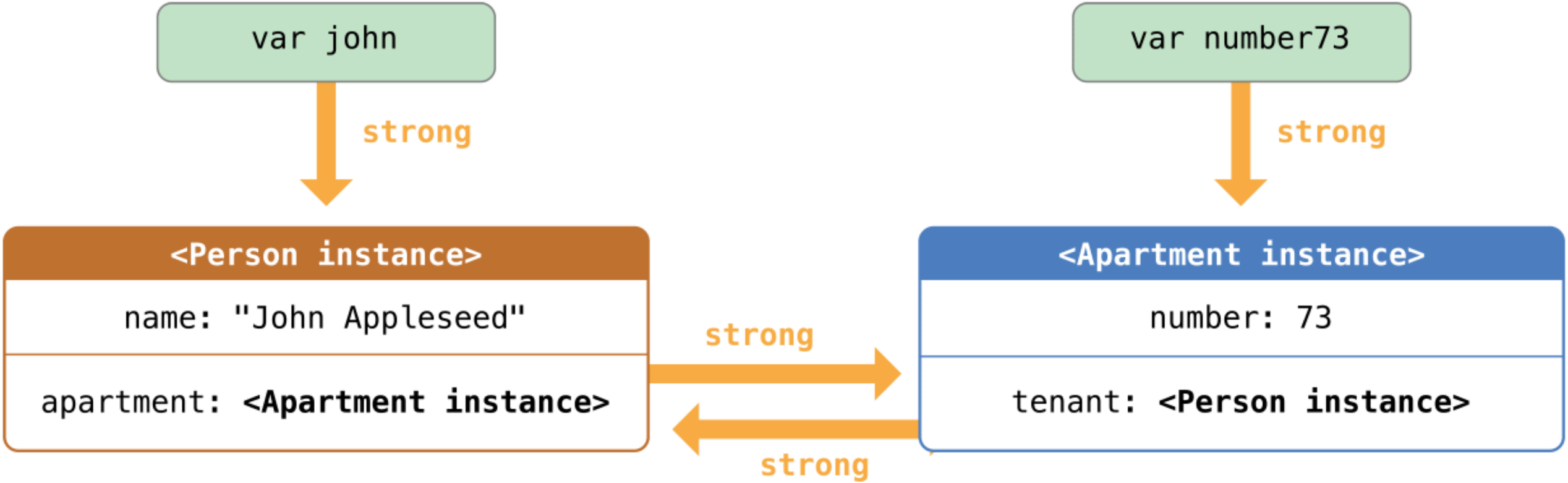
- Em tempo de compilação
- Única opção em Swift

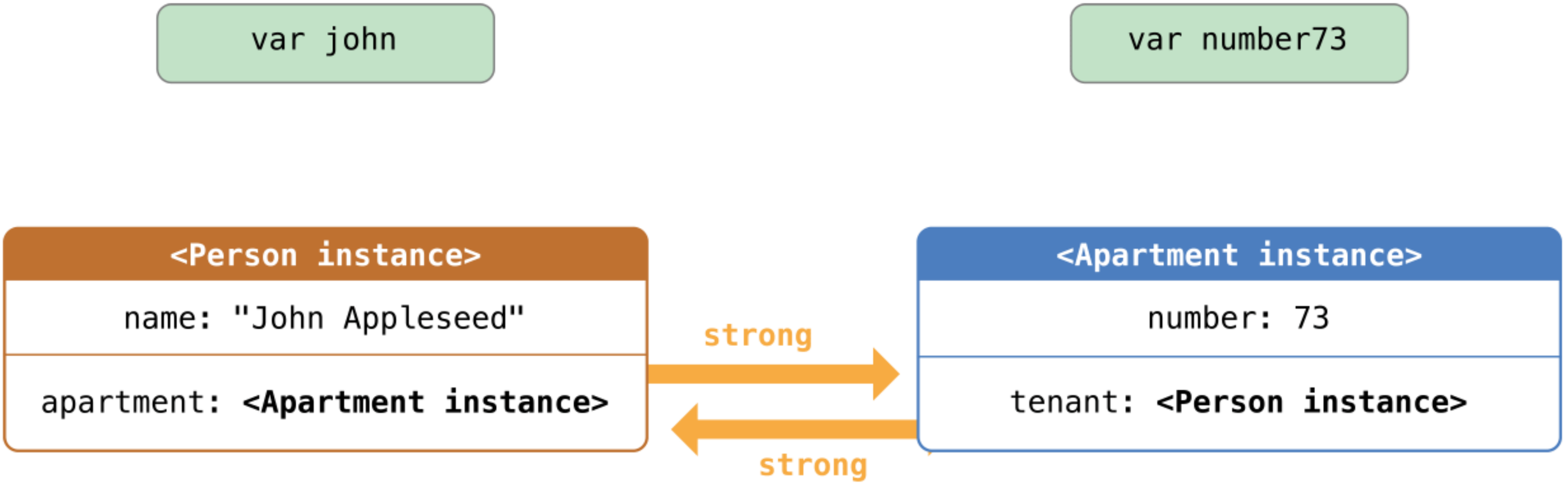
Reference counting only applies to instances of classes. Structures and enumerations are value types, not reference types, and are not stored and passed by reference.

Mesmo com ARC, podem acontecer **memory leaks**!

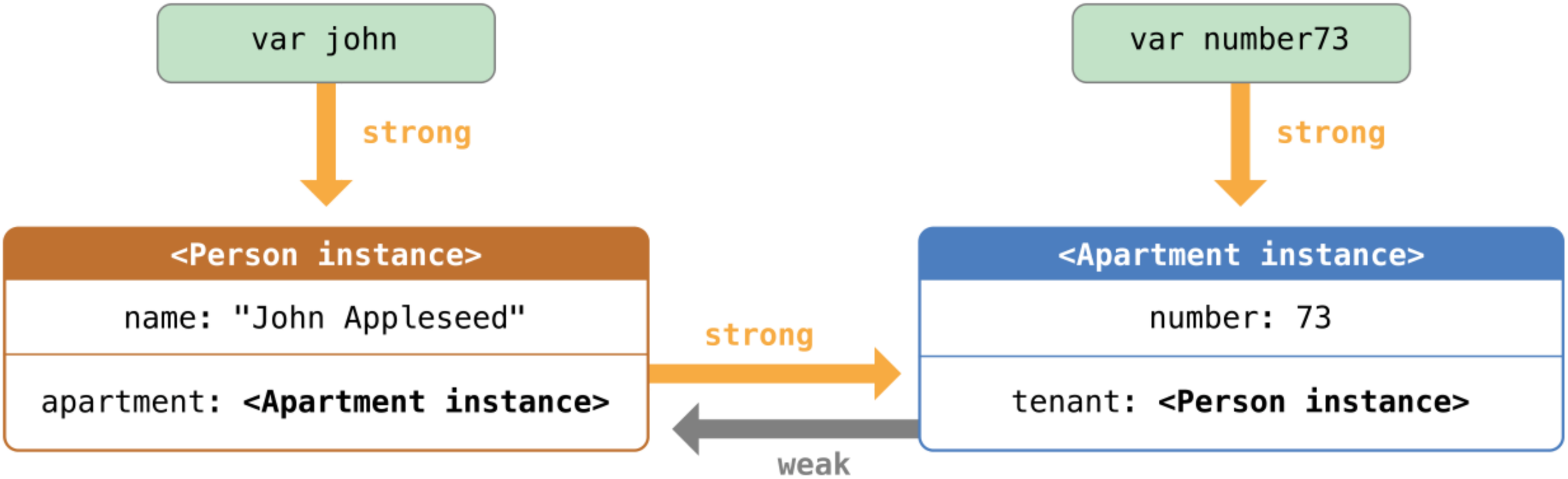
Referência cíclica

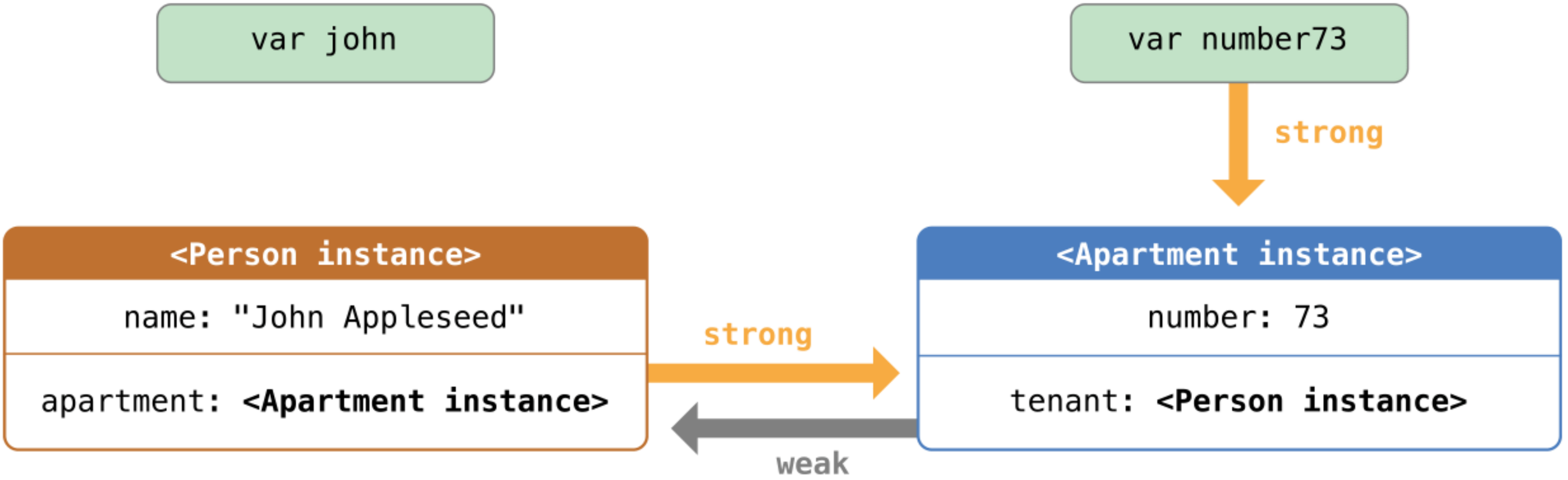

Parent
strong
strong
Child

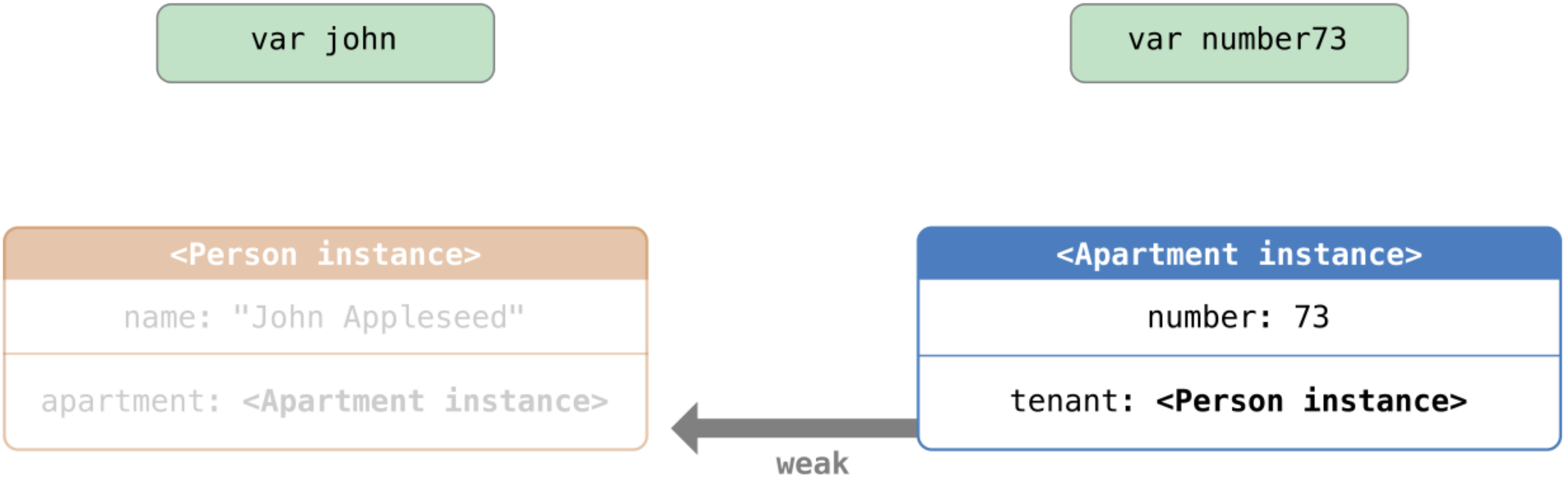

var john
strong
var number73
strong
<Person instance>
name: "John Appleseed"
apartment: <Apartment instance>
strong
strong
<Apartment instance>
number: 73
tenant: <Person instance>


var john
var number73
<Person instance>
name: "John Appleseed"
apartment: <Apartment instance>
strong
strong
<Apartment instance>
number: 73
tenant: <Person instance>

weak


var john
strong
var number73
strong
<Person instance>
name: "John Appleseed"
apartment: <Apartment instance>
strong
weak
<Apartment instance>
number: 73
tenant: <Person instance>


var john
var number73
strong
<Person instance>
name: "John Appleseed"
apartment: <Apartment instance>
strong
weak
<Apartment instance>
number: 73
tenant: <Person instance>


var john
var number73
<Person instance>
name: "John Appleseed"
apartment: <Apartment instance>
<Apartment instance>
number: 73
tenant: <Person instance>
weak

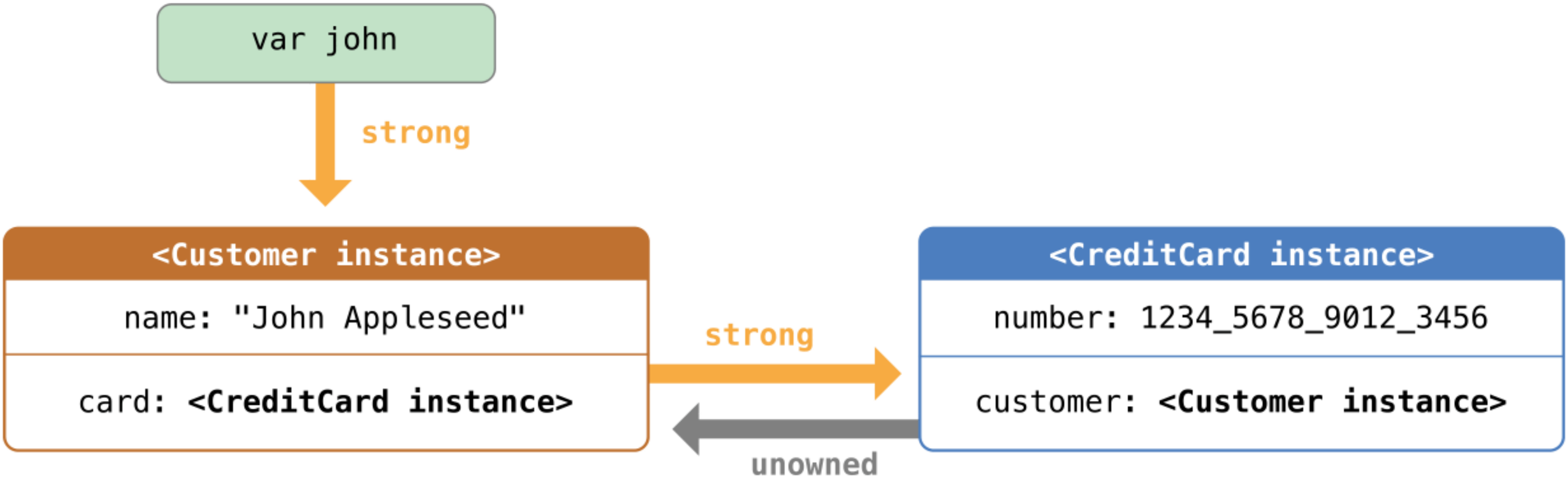
Variáveis `weak` têm que ser `Optional`, pois são setadas para `nil` quando o objeto é desalocado

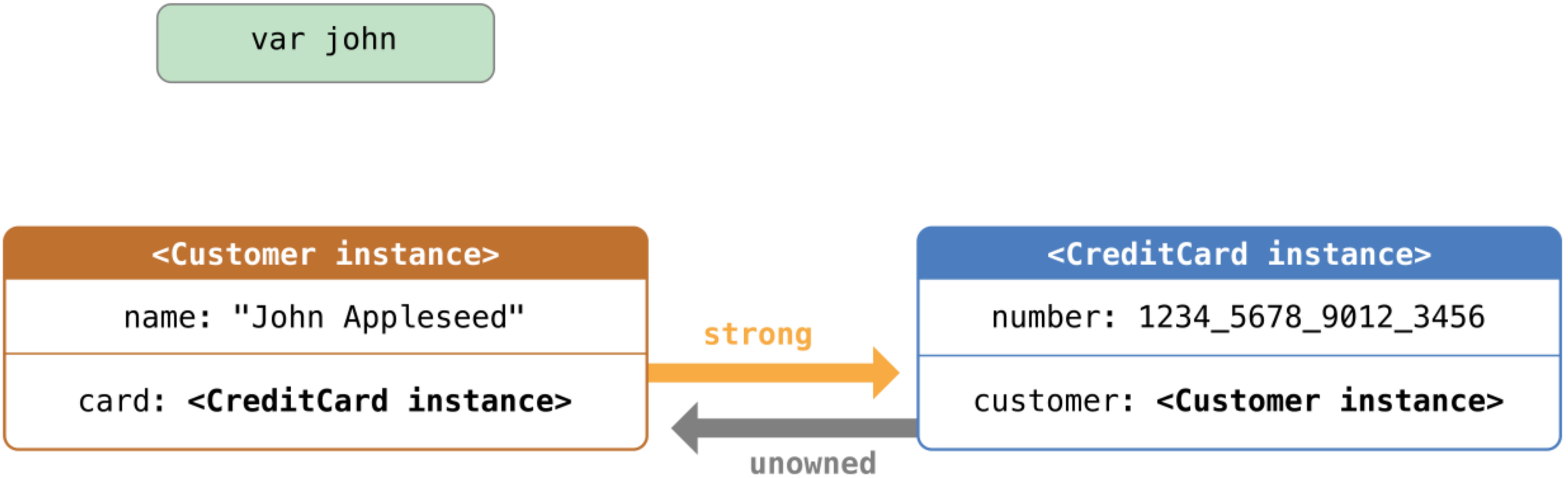
E se tivermos certeza que a referência sempre terá um valor?

unowned

`unowned`

- Sempre tem valor (nunca é `Optional`)
- Crash se acessar depois da instância ter sido desalocada


var john
strong
<Customer instance>
name: "John Appleseed"
card: <CreditCard instance>
strong
unowned
<CreditCard instance>
number: 1234_5678_9012_3456
customer: <Customer instance>


var john
<Customer instance>
name: "John Appleseed"
card: <CreditCard instance>
strong
unowned
<CreditCard instance>
number: 1234_5678_9012_3456
customer: <Customer instance>

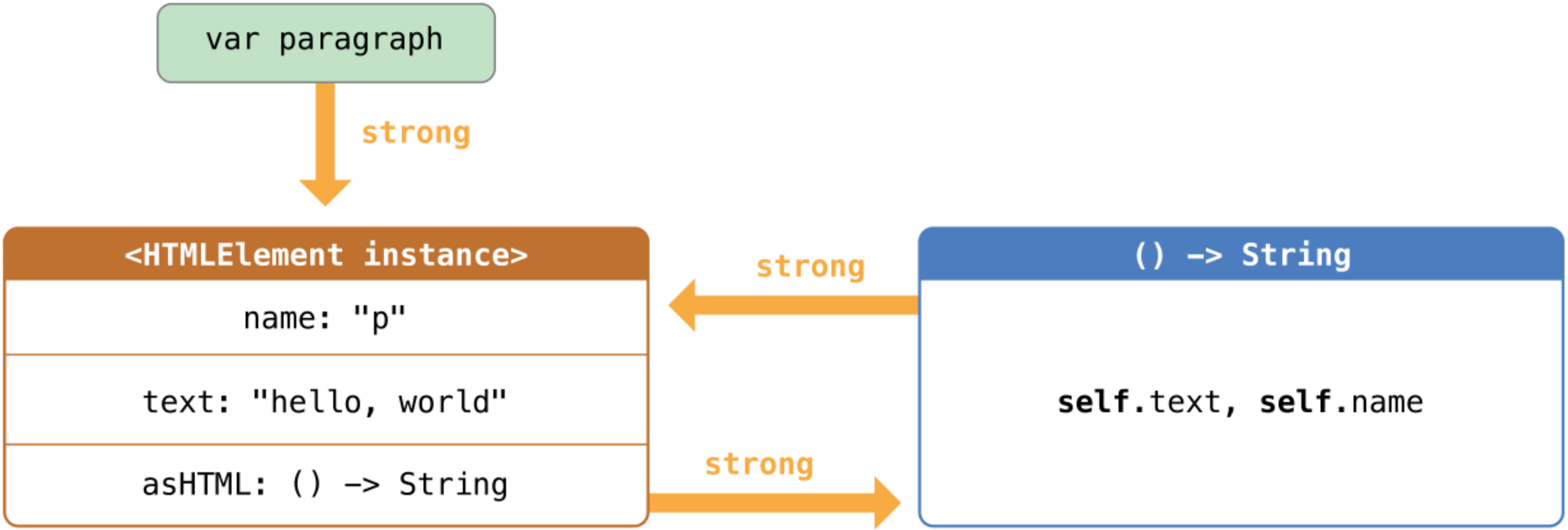
Closures podem capturar referências

Closures são *reference types*

```
class HTMLElement {

    let name: String
    let text: String?

    lazy var asHTML: () -> String = {
        if let text = self.text {
            return "<\(self.name)>\(text)</\(self.name)>"
        } else {
            return "<\(self.name) />"
        }
    }
}
```

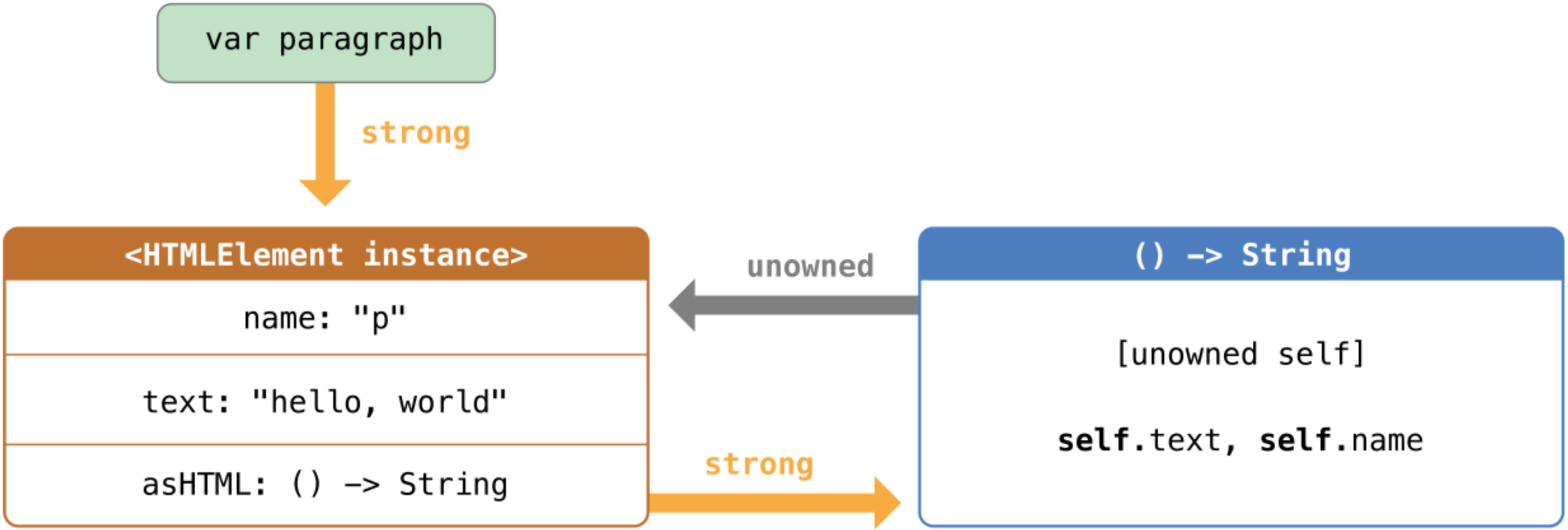

var paragraph
strong
<HTMLElement instance>
name: "p"
text: "hello, world"
asHTML: () -> String
strong
strong
() -> String
self.text, self.name

```
lazy var asHTML: () -> String = { [unowned self] in

    if let text = self.text {
        return "<\(self.name)>\(text)</\(self.name)>"
    } else {
        return "<\(self.name) />"
    }
}
```

Não existe chance de **`self`** ser desalocado antes do closure.

Por isso foi usado **unowned**.


var paragraph
strong
<HTMLElement instance>
name: "p"
text: "hello, world"
asHTML: () -> String
unowned
() -> String
[unowned self]
self.text, self.name
strong

Exercício

Verificar se não tem nenhum retain cycle no app

```
deinit {
    println("\(self.dynamicType) deinit")
}
```

Lembre-se que apenas **classes** podem ser retidas.
`enums` e `structs` são sempre copiados.