FABERNOVEL

Down the memory rabbit hole 🕵🏿‍♂️

Value type

```
var x: Int = 0

var y: Int = x

y += 1

print(x,y)            // 0 1
```

Value type

```
var x: Int = 0

var y: Int = x

y += 1

print(x,y)             // 0 1
```

Reference type

```
var x: NSMutableArray = []

var y: NSMutableArray = x

y.add(1)

print(x,y)         // [1] [1]
```

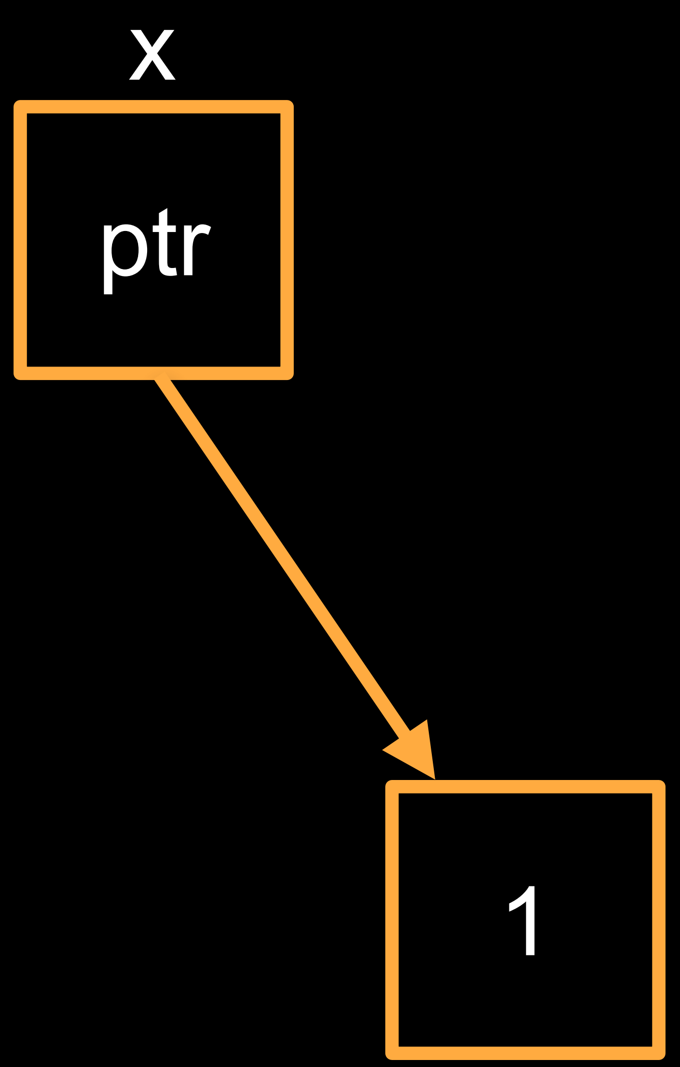
Value type

```swift
var x: Int = 0

var y: Int = x

y += 1

print(x,y)              // 0 1
```

```swift
var x: [Int] = [1]

var y: [Int] = x

y.append(2)

print(x,y)          // [1] [1, 2]
```

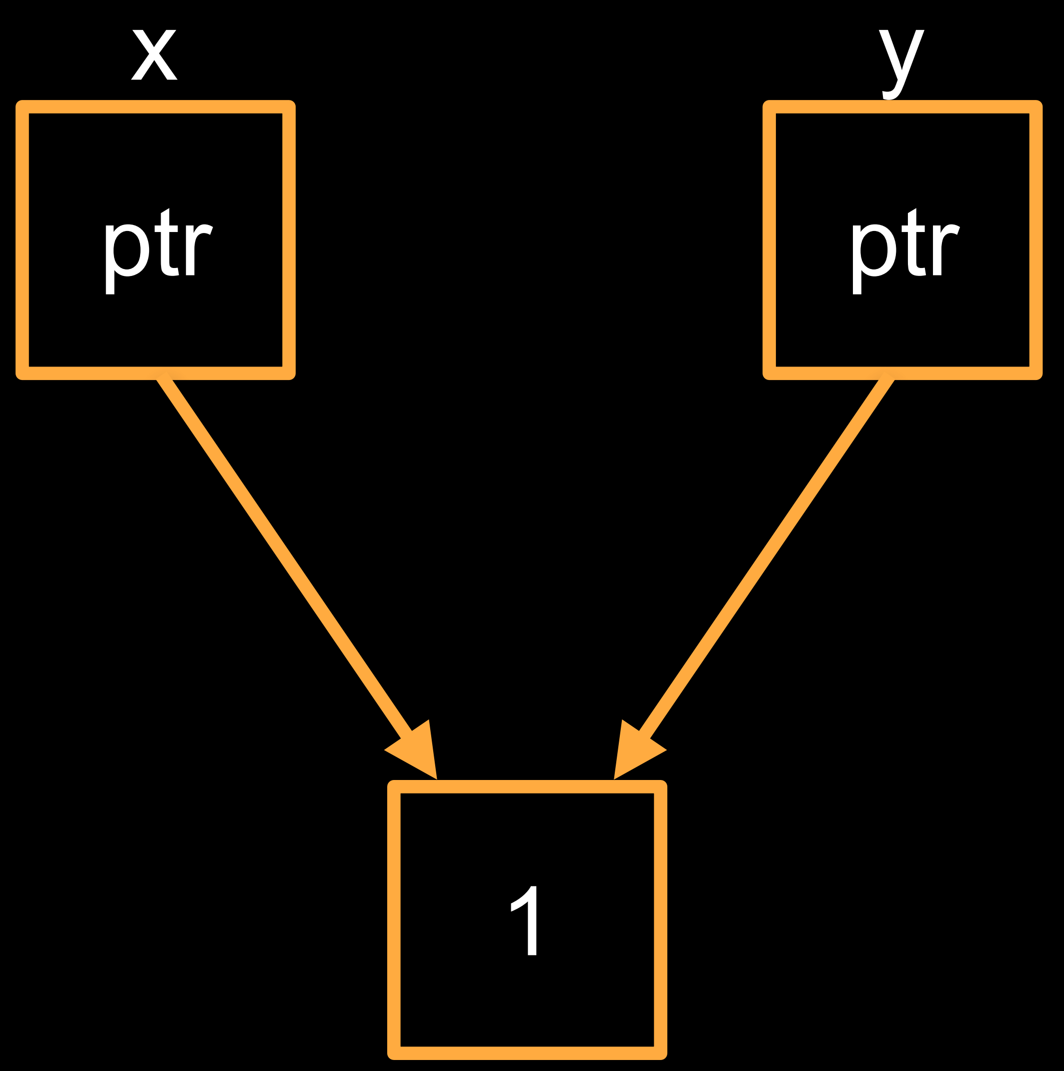
Value type

```swift
var x: [Int] = [1]

var y: [Int] = x

y.append(2)
```



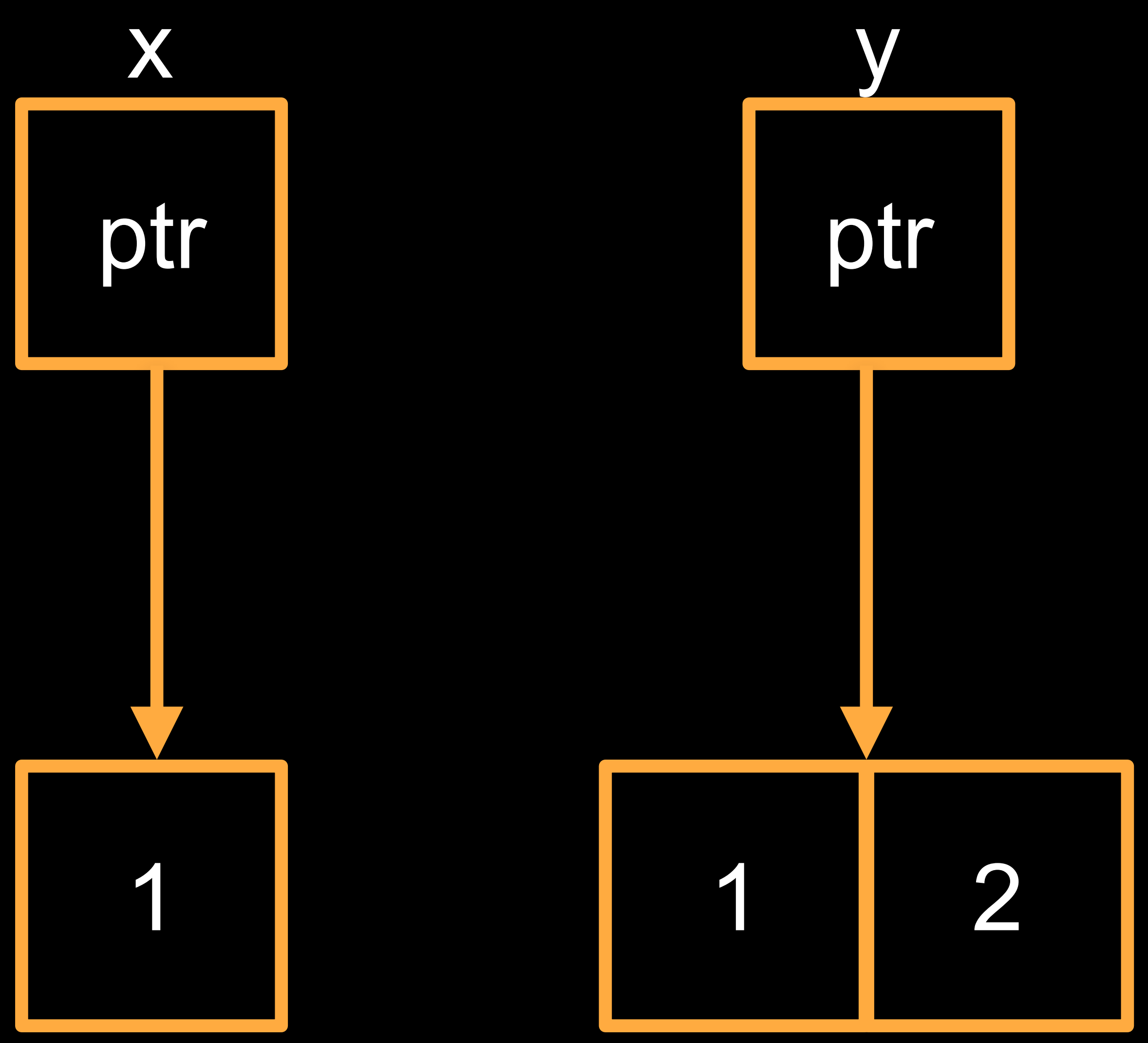
Value type

```
var x: [Int] = [1]

var y: [Int] = x

y.append(2)
```



Value type

```
var x: [Int] = [1]

var y: [Int] = x

y.append(2)
```



Take aways

In swift struct and enums are value types while classes are reference types

CoW can provide value semantic to a type that requires a class implementation

Most Swift collections rely on copy on write

Performances

```
public struct Vector {
    var x: Int
    var y: Int
    var z: Int
}
```

⏱ **500 ns**

```
public func vectorFunc(_ arg: Vector) -> Vector {
    return arg
}
```

Performances

```
public struct ComplexStruct {
    var id: Int
    var topics: [String]
    var headline: String
    var isHighlighted: Bool?
    var title: String
    var picturesUrl: [String]
    var keywords: [String]
    var publicationDateTime: String
    var likesCount: Int?
    var readingTime: Int
    var isLiked: Bool?
    var isNew: Bool
}

public func frequentStruct(_ arg: ComplexStruct) -> ComplexStruct {
   return arg
}
```

⏱️ 603 ns

Performances

```
public class ComplexClass {
    var id: Int
    var topics: [String]
    var headline: String
    var isHighlighted: Bool?
    var title: String
    var picturesUrl: [String]
    var keywords: [String]
    var publicationDateTime: String
    var likesCount: Int?
    var readingTime: Int
    var isLiked: Bool?
    var isNew: Bool
}

public func frequentClass(_ arg: ComplexClass) -> ComplexClass {
   return arg
}
```

⏱ **518 ns**

Performances

```
public struct CowComplexStruct {
    var id: Int
    var topics: [String]
    var headline: String
    var isHighlighted: Bool?
    var title: String
    var picturesUrl: [String]
    var keywords: [String]
    var publicationDateTime: String
    var likesCount: Int?
    var readingTime: Int
    var isLiked: Bool?
    var isNew: Bool
}

public func cowComplexStruct(_ arg: CowComplexStruct) -> CowComplexStruct {
   return arg
}
```

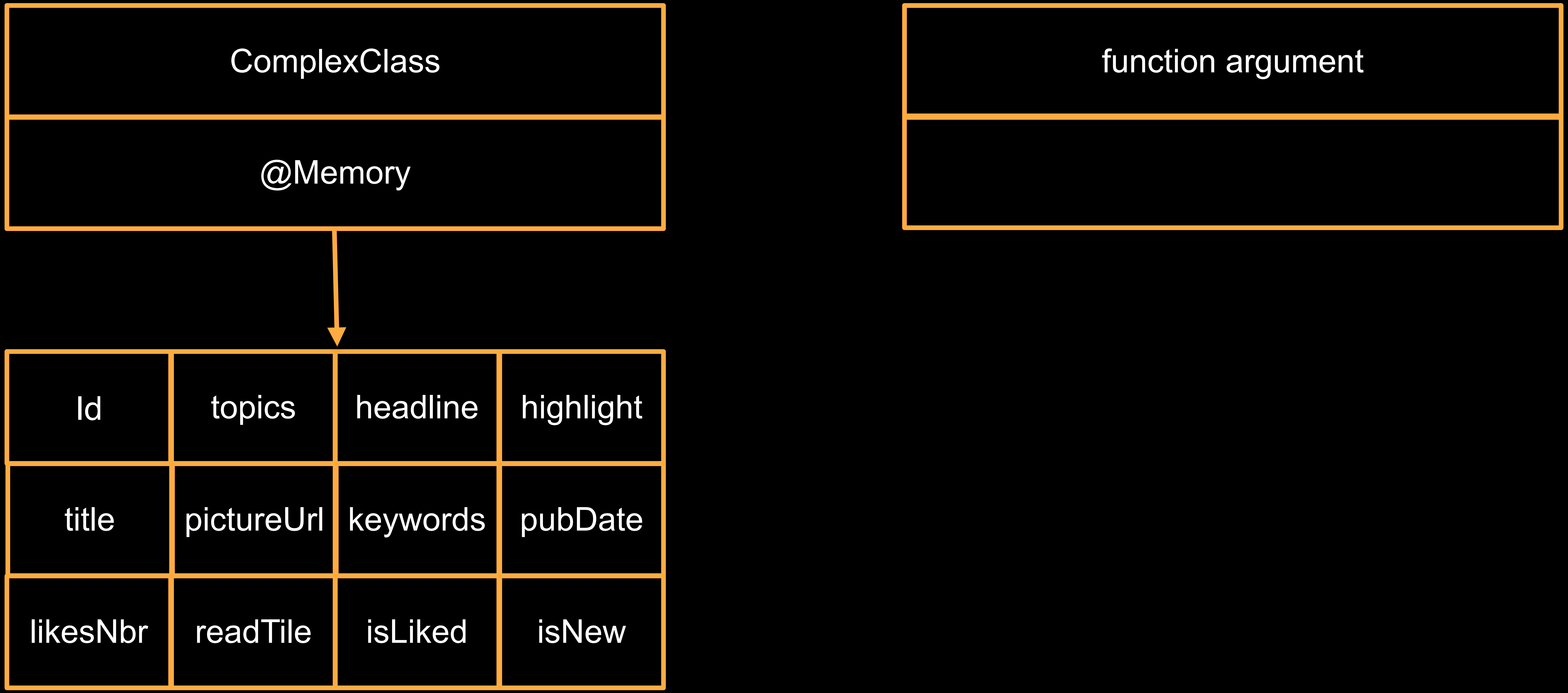
⏱ **519 ns**

Performances

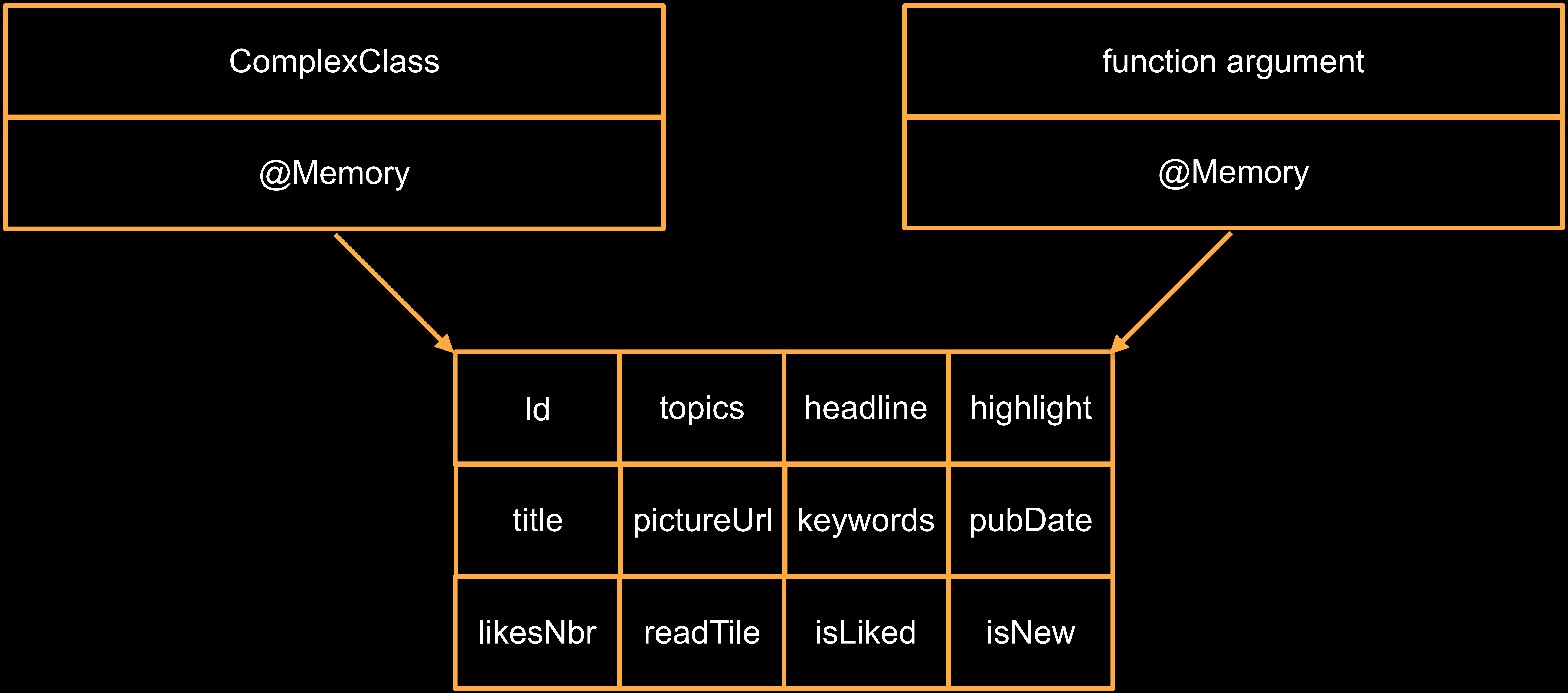
ComplexStruct			
Id	topics	headline	highlight
title	pictureUrl	keywords	pubDate
likesNbr	readTile	isLiked	isNew

function argument			
id	topics	headline	highlight
title	pictureUrl	keywords	pubDate
likesNbr	readTile	isLiked	isNew

Performances



Performances



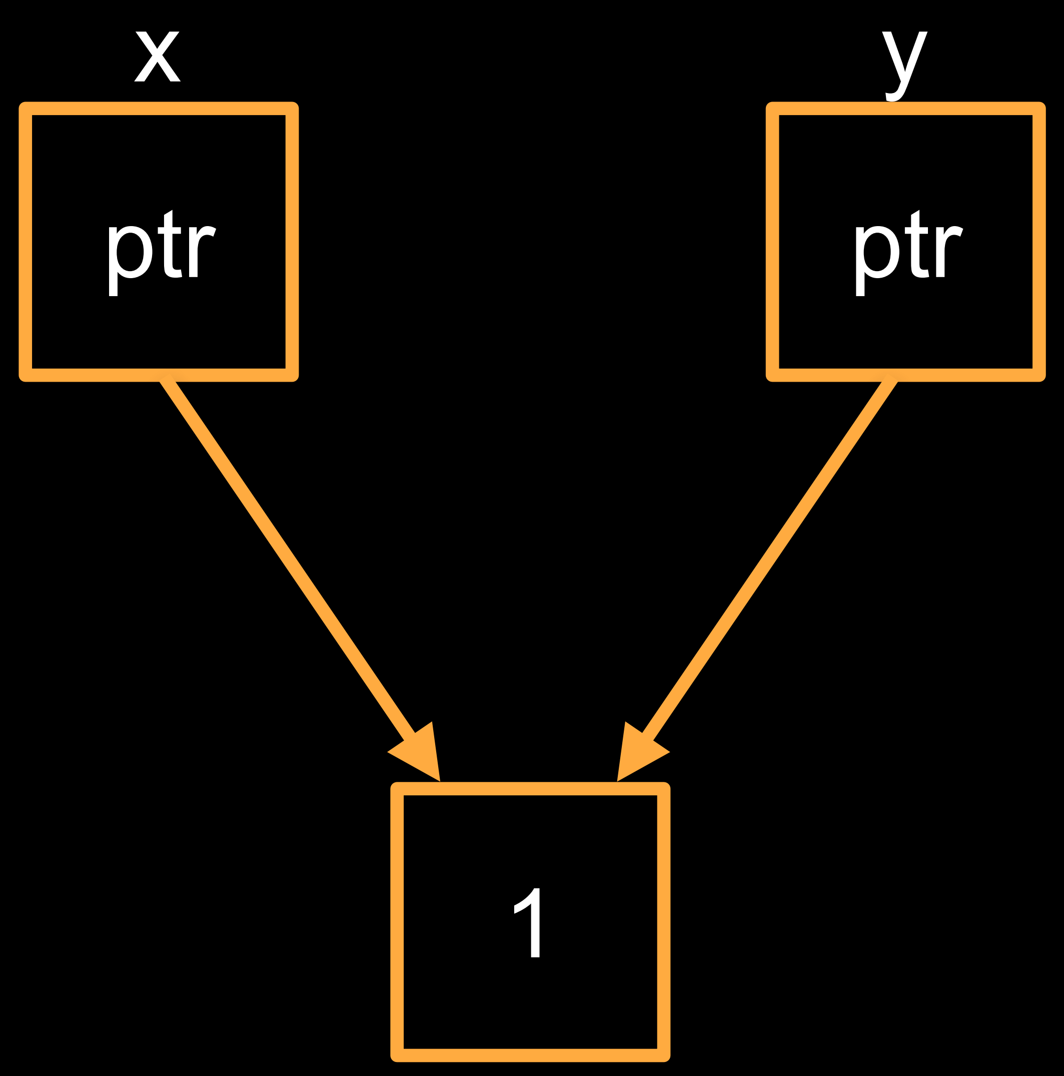
Performances

Int	Vector	Class	CowStruct	Struct
493	**500**	**518**	**519**	**603**

Copy on write

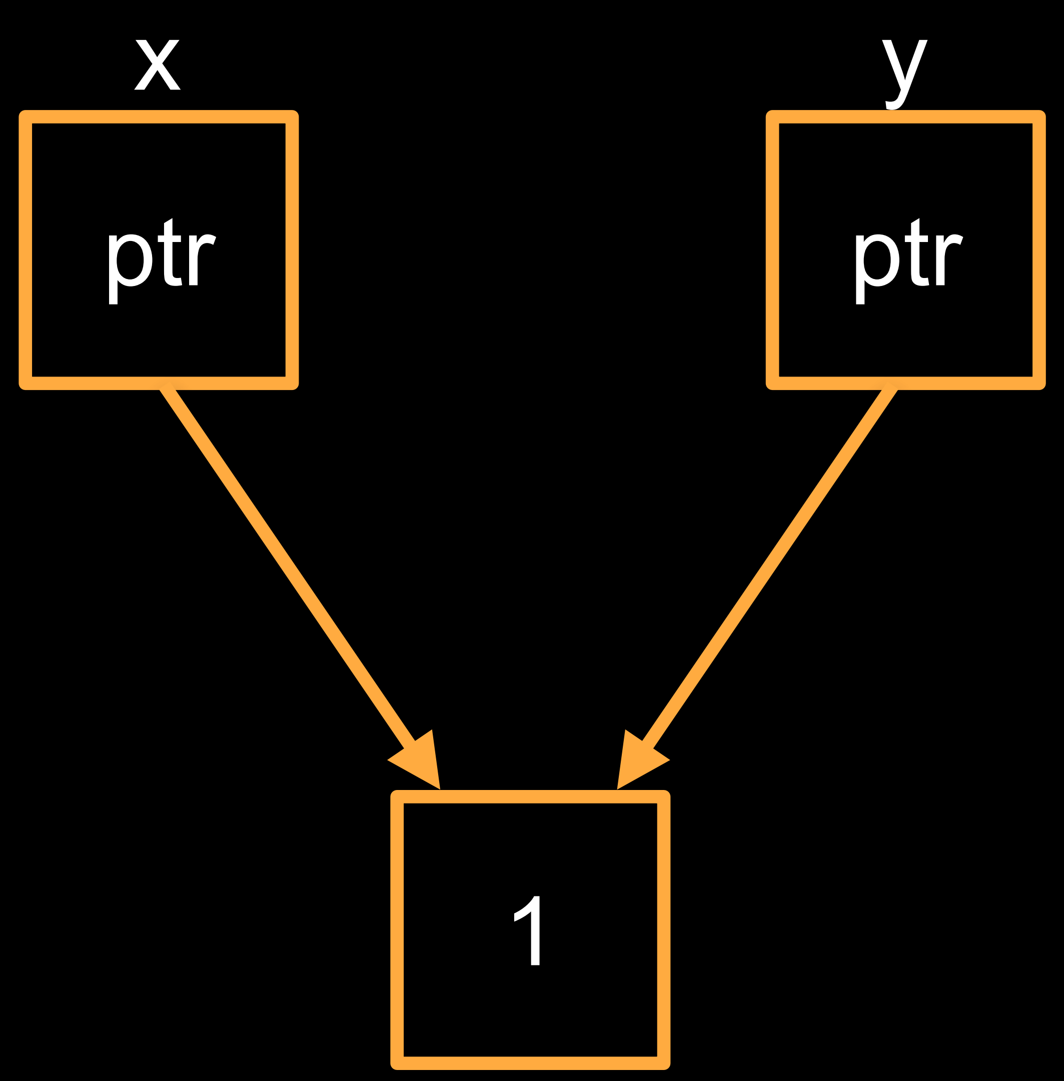
```swift
var x: [Int] = [1]
var y: [Int] = x
```

```swift
struct CowVector3D {
        var x: Double
        var y: Double
        var z: Double
}
```



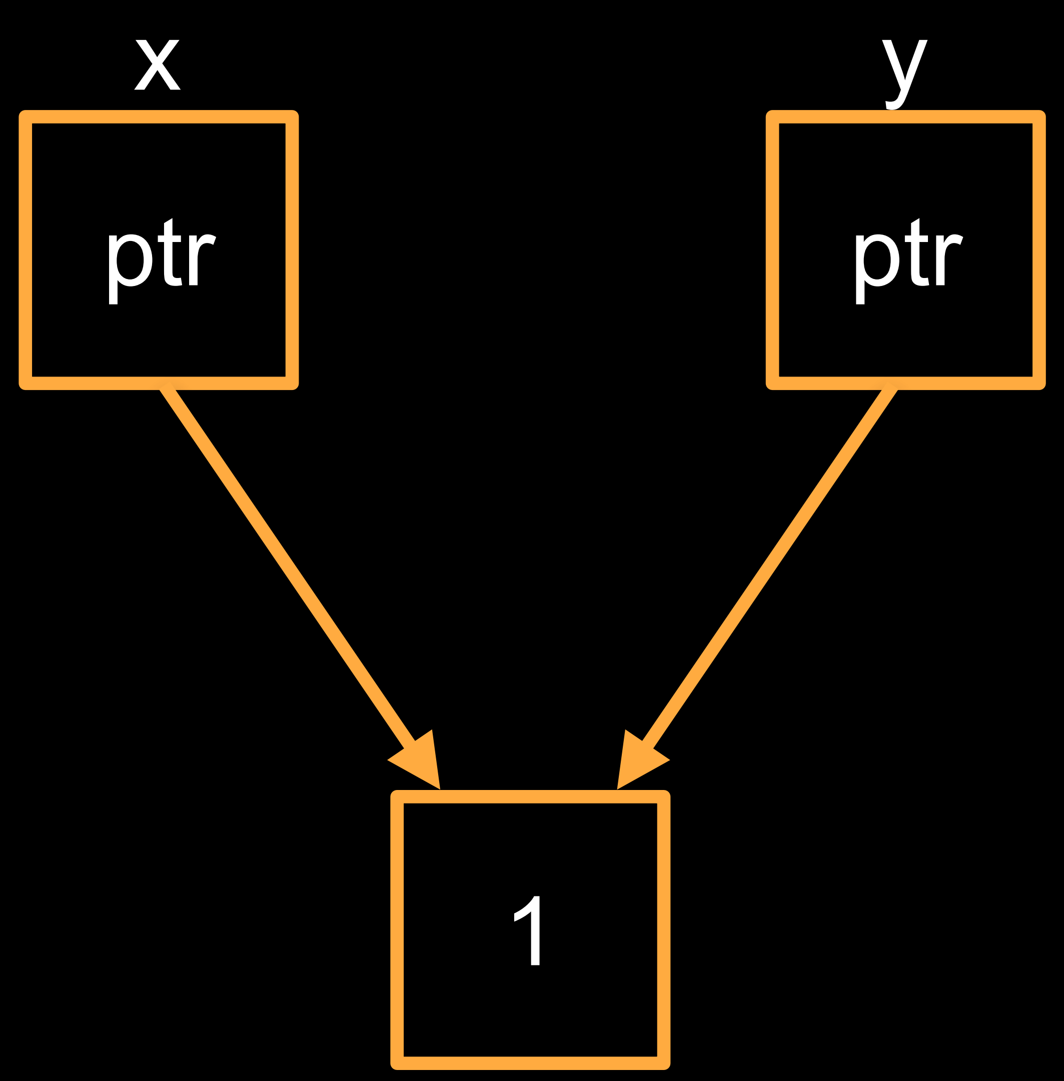
Copy on write

```
extension CowVector3D {
    class Storage {
        init(x: Double,
             y: Double,
             z: Double) {
            self.x = x
            self.y = y
            self.z = z
            print("initWithProperties")
        }

        init(_ toCopy: Storage) {
            x = toCopy.x
            y = toCopy.y
            z = toCopy.z
            print("initWithCopy")
        }

        var x, y, z: Double
    }
}
```



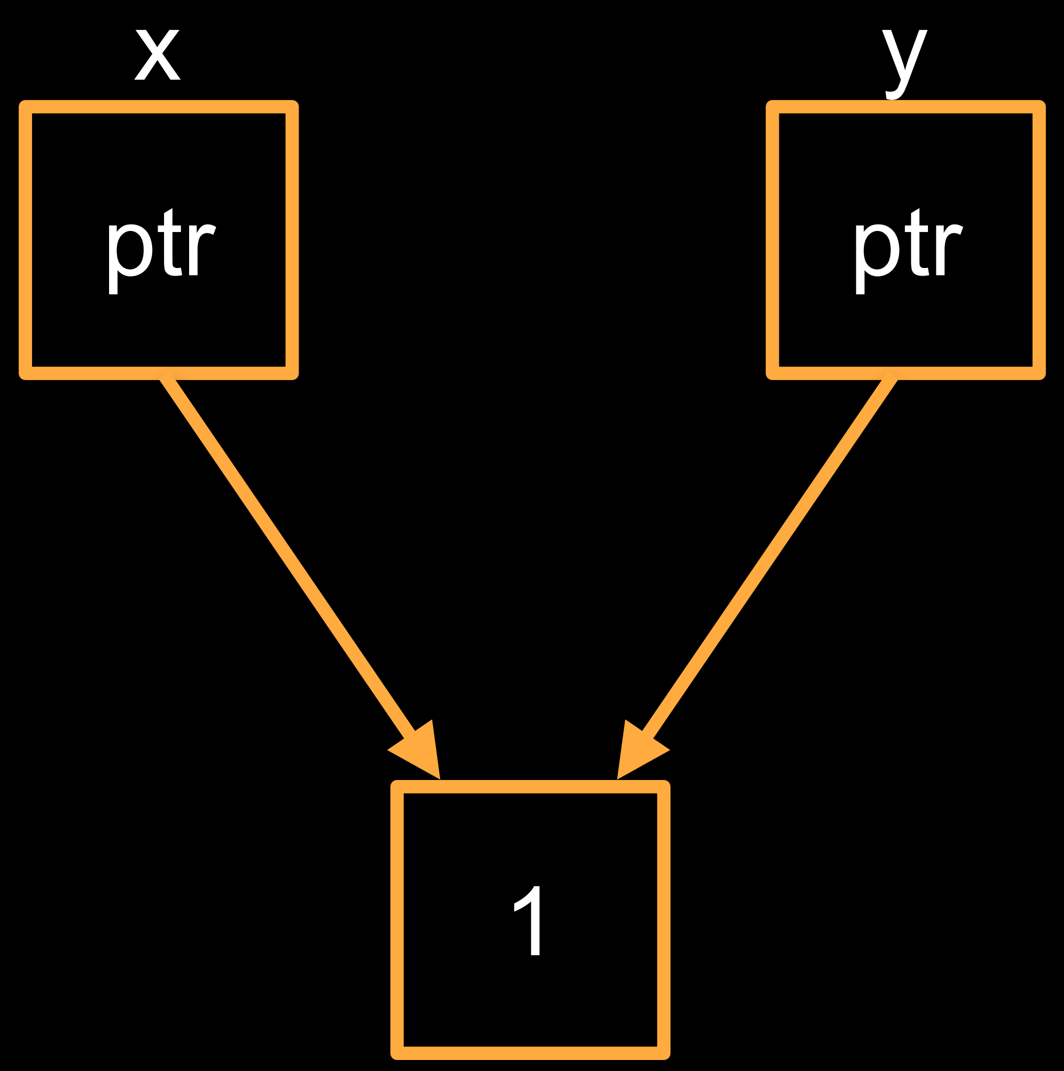
Copy on write

```
struct CowVector3D {
        var x: Double
        var y: Double
        var z: Double
}
```



Copy on write

```swift
struct CowVector3D {
    private var storage: Storage

    init(x: Double, y: Double, z: Double) {
        storage = Storage(x: x, y: y, z: z)
        // prints initWithProperties
    }

    var x: Double {
        get { storage.x }
        set {

        }
    }

    ...
}
```



Copy on write

```swift
struct CowVector3D {
    private var storage: Storage

    init(x: Double, y: Double, z: Double) {
        storage = Storage(x: x, y: y, z: z)
        // prints initWithProperties
    }

    var x: Double {
        get { storage.x }
        set {
            storage = Storage(self.storage)
            // prints initWithCopy
            storage.x = newValue
        }
    }

    ...
}
```

Copy on write

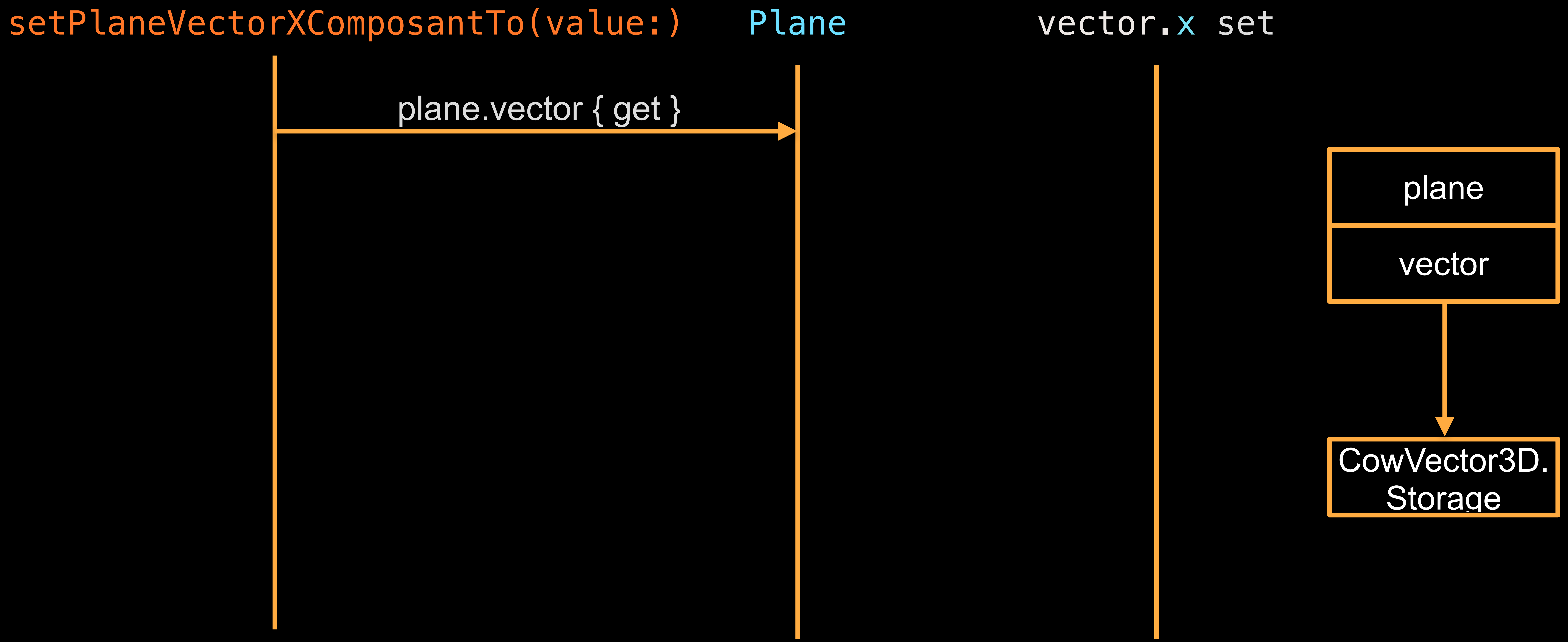
```swift
struct CowVector3D {
    private var storage: Storage

    init(x: Double, y: Double, z: Double) {
        storage = Storage(x: x, y: y, z: z)
        // prints initWithProperties
    }

    var x: Double {
        get { storage.x }
        set {
            storage = Storage(self.storage)
            // prints initWithCopy
            storage.x = newValue
        }
    }

    ...
}

var vector = CowVector3D(x: 0, y: 0, z: 0)
// prints initWithProperties
vector.x += 1
// prints initWithCopy
```

Copy on write

```swift
struct CowVector3D {
    private var storage: Storage

    init(x: Double, y: Double, z: Double) {
        storage = Storage(x: x, y: y, z: z)
        // prints initWithProperties
    }

    var x: Double {
        get { storage.x }
        set {
            if isKnownUniquelyReferenced(&storage) {
                storage.x = newValue
            } else {
                storage = Storage(self.storage)
                // prints initWithCopy
                storage.x = newValue
            }
        }
    }

    ...
}
```

```swift
class Plane {
    private var orthogonalVector: CowVector3D

    var vector: CowVector3D {
        get {
            print("get")
            return orthogonalVector
        }
        set {
            print("set")
            orthogonalVector = newValue
        }
    }
}


func setPlaneVectorXComposantTo(value: Double) {
    plane.vector.x = value
}


// prints initWithProperties
let plane = Plane(vector: CowVector3D(x: 0, y: 0, z: 0))
setPlaneVectorXComposantTo(value: 1)
// prints get initWithCopy set
```

```swift
func setPlaneVectorXComposantTo(value: Double) {
    plane.vector.x = value
}
```



```swift
func setPlaneVectorXComposantTo(value: Double) {
    plane.vector.x = value
}
```

setPlaneVectorXComposantTo(value:)
Plane
vector.x set
plane.vector { get }
Vector
plane
vector
copy
CowVector3D.
Storage

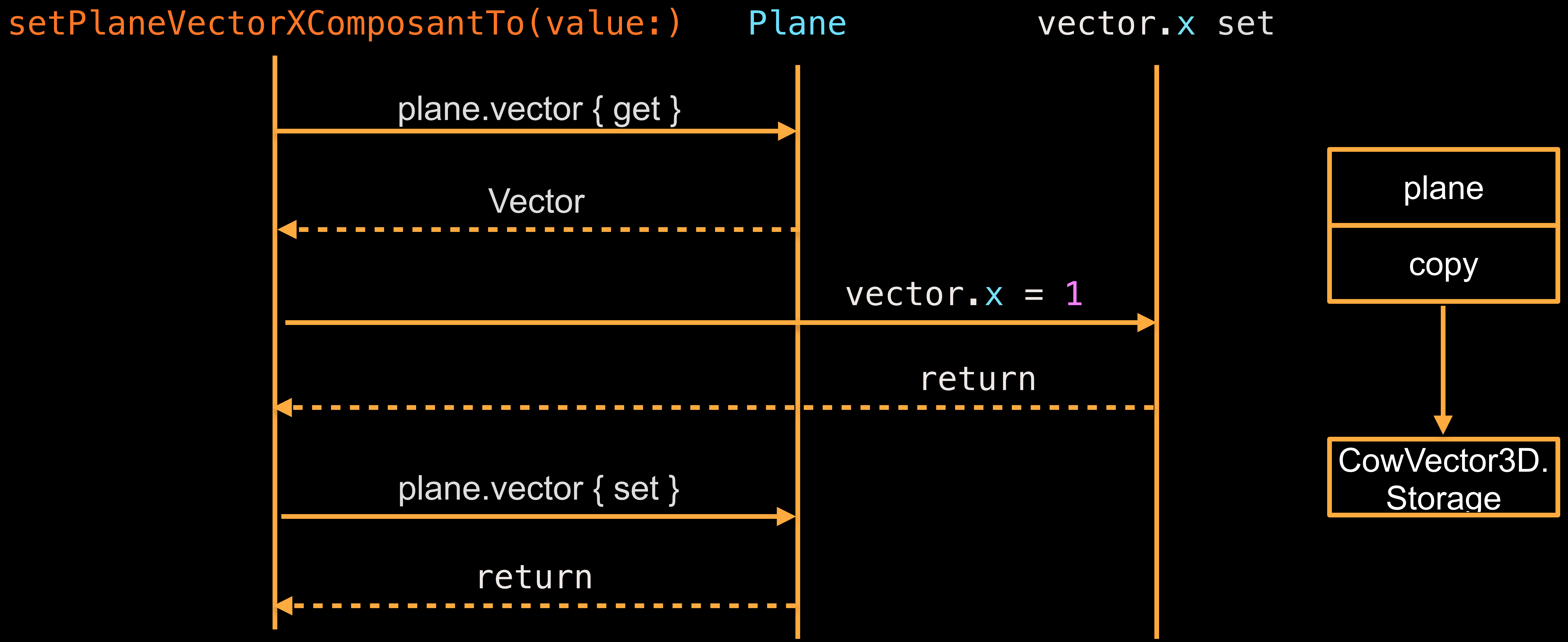
```swift
func setPlaneVectorXComposantTo(value: Double) {
    plane.vector.x = value
}
```

setPlaneVectorXComposantTo(value:)
Plane
vector.x set
plane.vector { get }
Vector
vector.x = 1
return
plane
vector
copy
CowVector3D. Storage
CowVector3D. Storage

```swift
func setPlaneVectorXComposantTo(value: Double) {
    plane.vector.x = value
}
```

setPlaneVectorXComposantTo(value:)
Plane
vector.x set
plane.vector { get }
Vector
vector.x = 1
return
plane.vector { set }
plane
copy
CowVector3D.Storage
CowVector3D.Storage

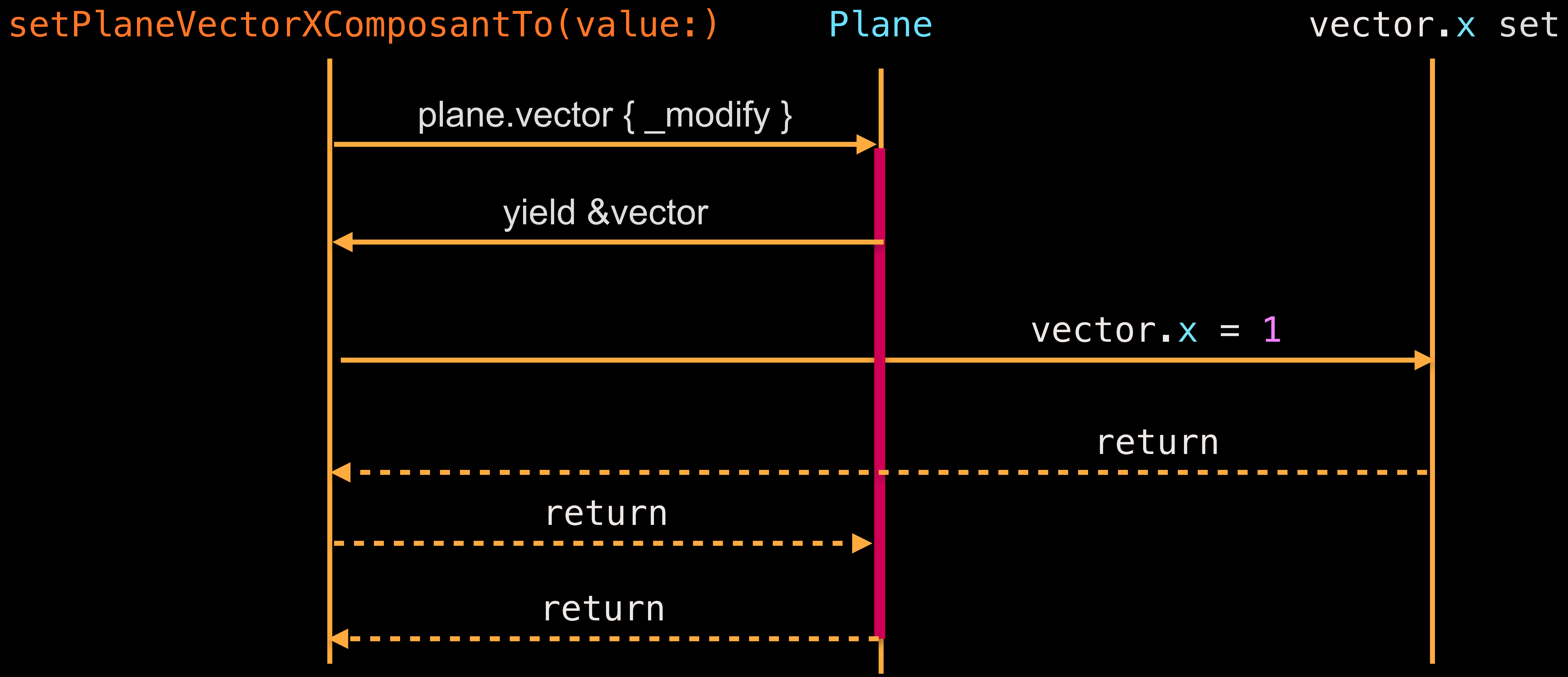
```swift
func setPlaneVectorXComposantTo(value: Double) {
    plane.vector.x = value
}
```



```swift
class Plane {
    private var orthogonalVector: CowVector3D

    var vector: CowVector3D {
        get {
            print("get")
            return orthogonalVector
        }
        set {
            print("set")
            orthogonalVector = newValue
        }
    }

    init(vector: CowVector3D) {
        orthogonalVector = vector
    }
}
```

```swift
class Plane {
    private var orthogonalVector: CowVector3D

    var vector: CowVector3D {
        get {
            print("get")
            return orthogonalVector
        }
        set {
            print("set")
            orthogonalVector = newValue
        }
        _modify {
            print("modify")
            yield &orthogonalVector
        }
    }

    init(vector: CowVector3D) {
        orthogonalVector = vector
    }
}
```

```swift
func setPlaneVectorXComposantTo(value: Double) {
    plane.vector.x = value
}
```



```swift
@propertyWrapper
struct SomeWrapper<T> {
    private var underlyingValue: T

    init(underlyingValue: T) {
        self.underlyingValue = underlyingValue
    }

    var wrappedValue: T {
        get {
            // Some code to make my wrapper interesting
            underlyingValue
        }
        set {
            // Some code to make my wrapper interesting
            underlyingValue = newValue
        }
    }
}
```

```swift
enum MyEnum {

    case a([Int])
    case b([Int])

    var array: [Int] {
        get {
            switch self {
            case let .a(array):
                return array
            case let .b(array):
                return array
            }
        }
        set {
            switch self {
            case .a:
                self = .a(newValue)
            case .b:
                self = .b(newValue)
            }
        }
    }
}
```

```swift
extension Array {
    var first: Element? {
        get { isEmpty ? nil : self[0] }
        _modify {
            var tmp: Optional<Element>
            if isEmpty {
                tmp = nil
                yield &tmp
                if let newValue = tmp {
                    self.append(newValue)
                }
            } else {
                tmp = self[0]
                yield &tmp
                if let newValue = tmp {
                    self[0] = newValue
                } else {
                    self.removeFirst()
                }
            }
        }
    }
}
```

Take aways

Computed variables might reduce performances issues if _modify is not implemented

Chosing between reference and value type is about semantics not performance

Benchmark when implementing to copy on write

Swift forum thread
https://forums.swift.org/t/modify-accessors/31872

Ben Cohen - Fast Safe Mutable State :
https://www.youtube.com/watch?v=BXJIIQ-B4-E

Johannes Weiss - High-performance system in Swift :
https://www.youtube.com/watch?v=iLDldae64xE

Cory Benfield - High-performance system in Swift :
https://www.youtube.com/watch?v=WCUj581Dpec

Thank you to Johannes Weiss for helping me with the benchmark part

Denis Poifol

FABERNOVEL

THANK YOU