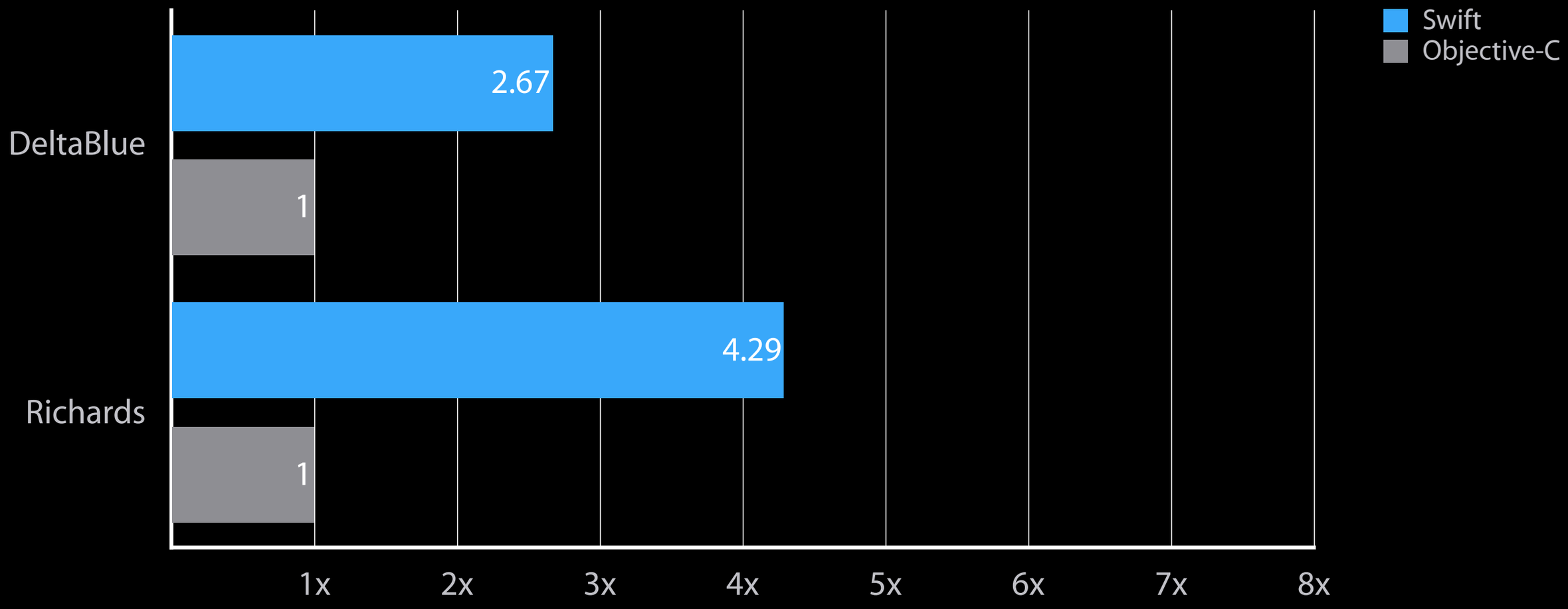
Understand Swift Performance

@唐巧_boy

小猿搜题产品技术负责人

Swift vs. Objective-C

Program speed (higher is better)



Swift vs Objective-C

	Created	Developed
Swift	2014	3 years
Objective-C	1980s	over 30 years

Understand the implementation
to understand performance

Performance optimizations

- Compiler optimization（ Whole Module Optimization ）
- Memory optimization（ Value Type & Immutable & Copy-On-Write ）
- Method dispatch optimization（ No Message Send ）
- Reference count optimization
- The internal implementation of protocol types

Agenda

- Performance optimizations:
  - ~~Compiler optimization（Whole Module Optimization）~~
  - ~~Memory optimization（Value Type & Immutable & Copy-On-Write）~~
  - ~~Method dispatch optimization（No Message Send）~~
  - Reference count optimization
  - The internal implementation of protocol types
- Demo

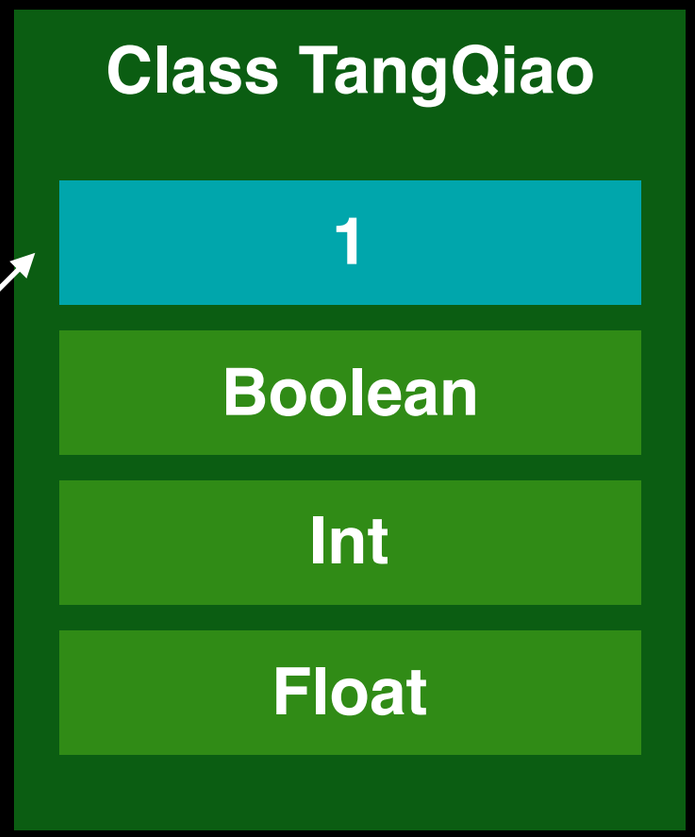
Reference count optimization

```
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}

func exam(person: TangQiao?) {
    // ...
}

var x: TangQiao? = TangQiao()
var y  = x

exam(person: y)

y = nil
x = nil
```

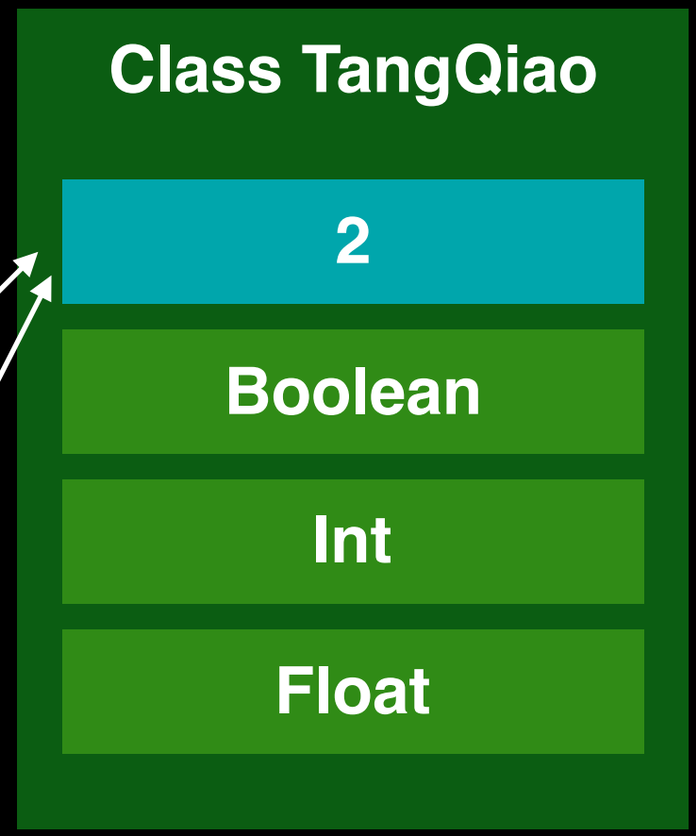
```swift
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}

func exam(person: TangQiao?) {
    // ...
}
```

```swift
var x: TangQiao? = TangQiao()
var y  = x

exam(person: y)

y = nil
x = nil
```

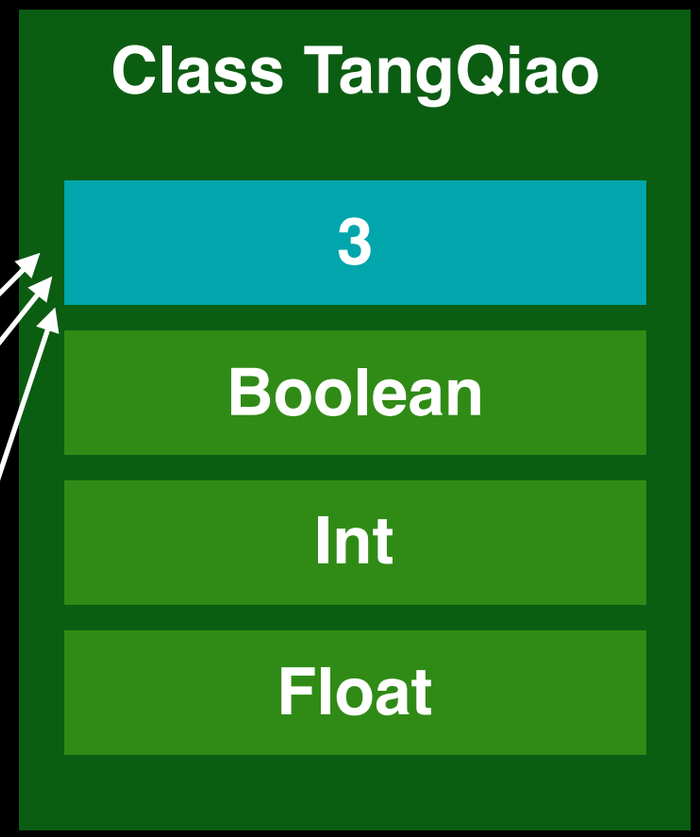
```swift
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}

func exam(person: TangQiao?) {
    // ...
}

var x: TangQiao? = TangQiao()
var y  = x

exam(person: y)

y = nil
x = nil
```



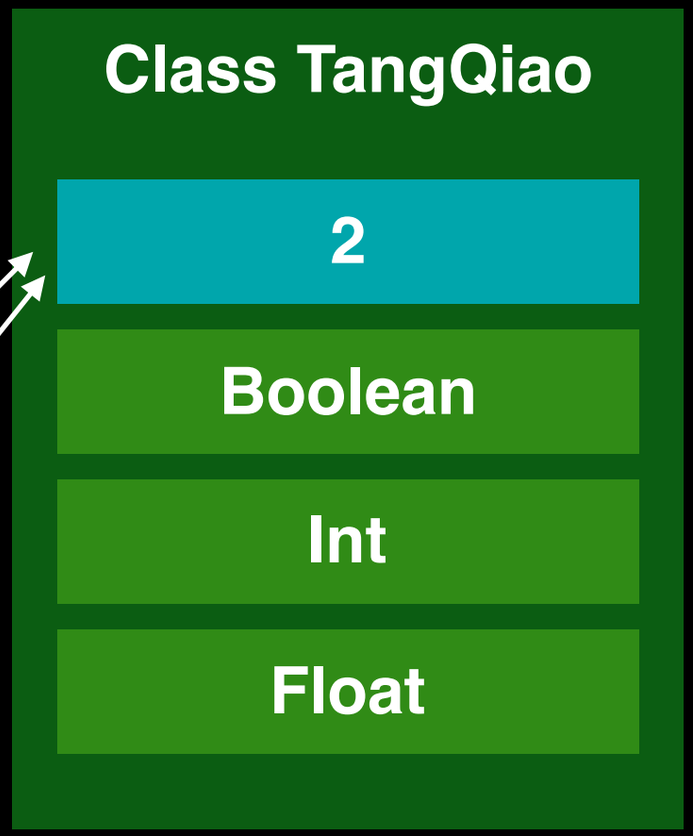
```swift
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}

func exam(person: TangQiao?) {
    // ...
}

var x: TangQiao? = TangQiao()
var y  = x

exam(person: y)

y = nil
x = nil
```



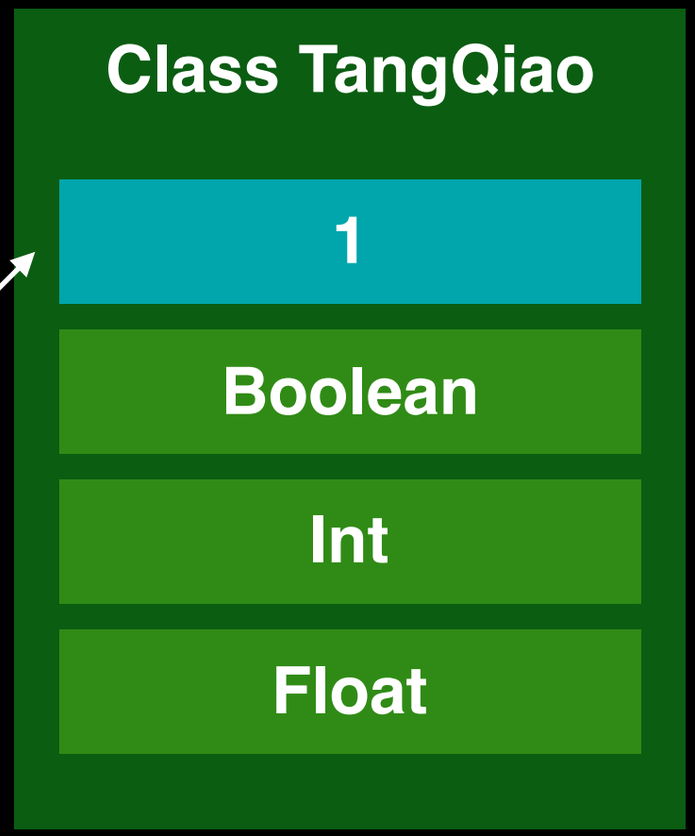
```
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}

func exam(person: TangQiao?) {
    // ...
}

var x: TangQiao? = TangQiao()
var y  = x

exam(person: y)

y = nil
x = nil
```



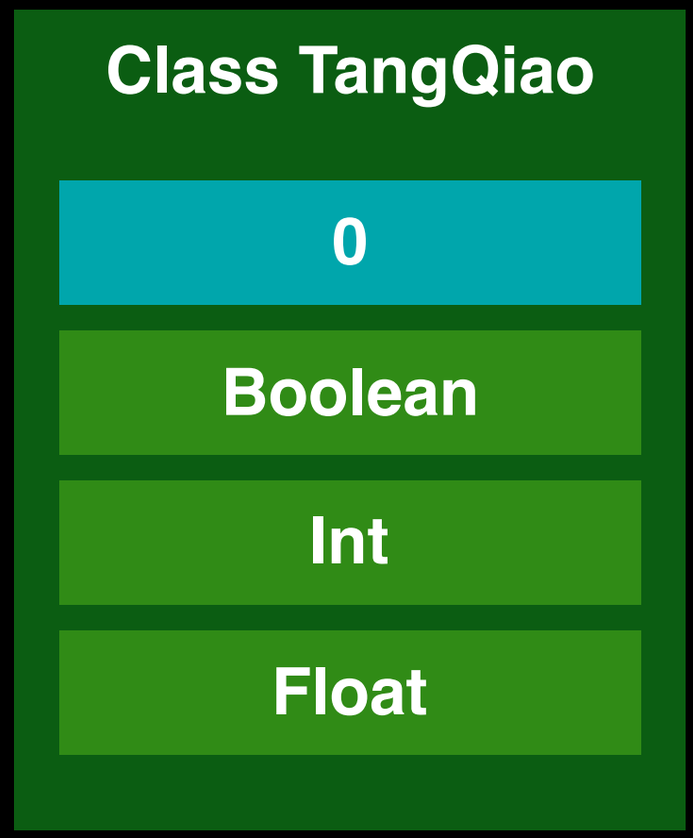
```swift
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}

func exam(person: TangQiao?) {
    // ...
}

var x: TangQiao? = TangQiao()
var y  = x

exam(person: y)

y = nil
x = nil
```



```swift
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}

func exam(person: TangQiao?) {
    // ...
}

var x: TangQiao? = TangQiao()
var y  = x

exam(person: y)

y = nil
x = nil
```



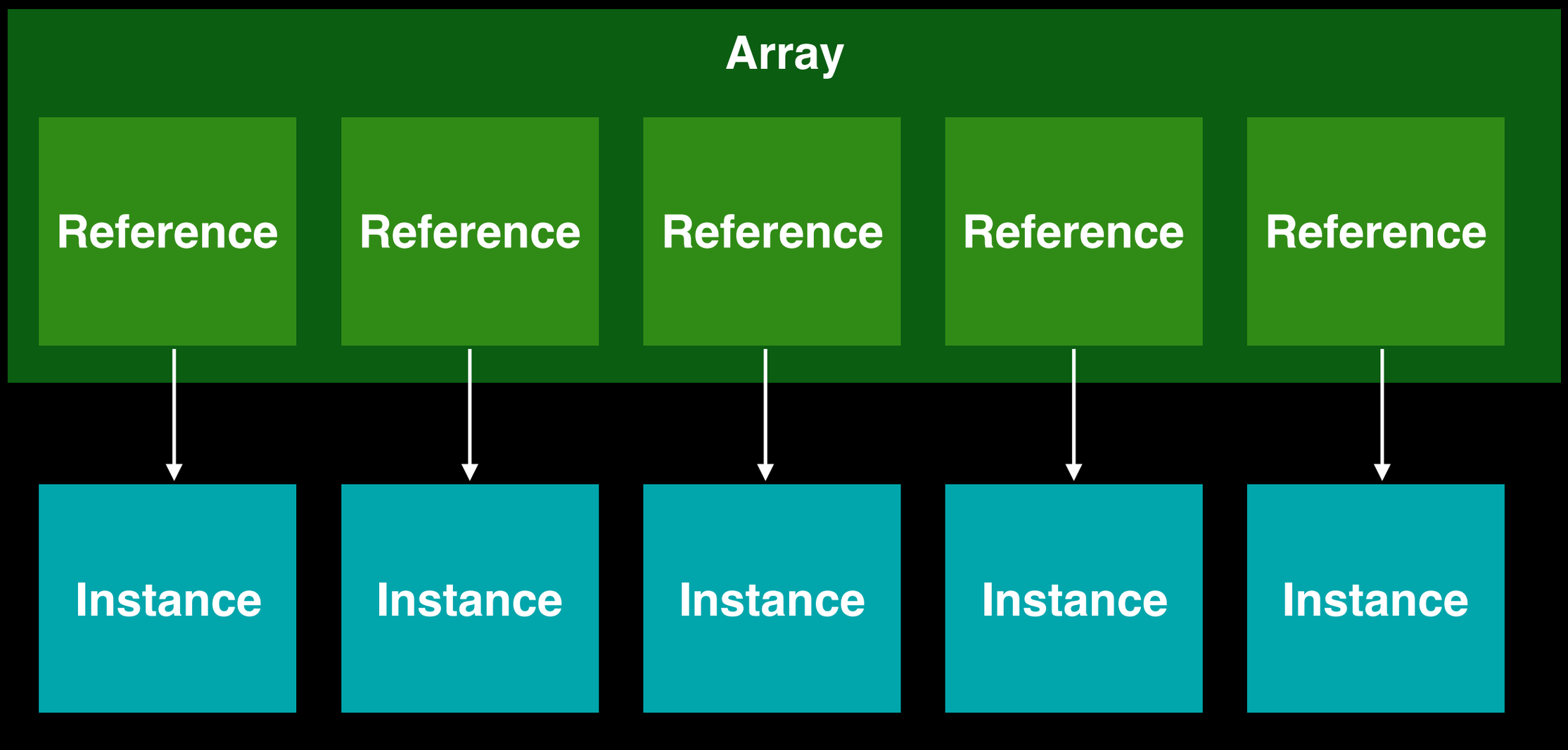
```swift
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}

func exam(person: TangQiao?) {
    // ...
}

var x: TangQiao? = TangQiao()
var y  = x

exam(person: y)

y = nil
x = nil
```



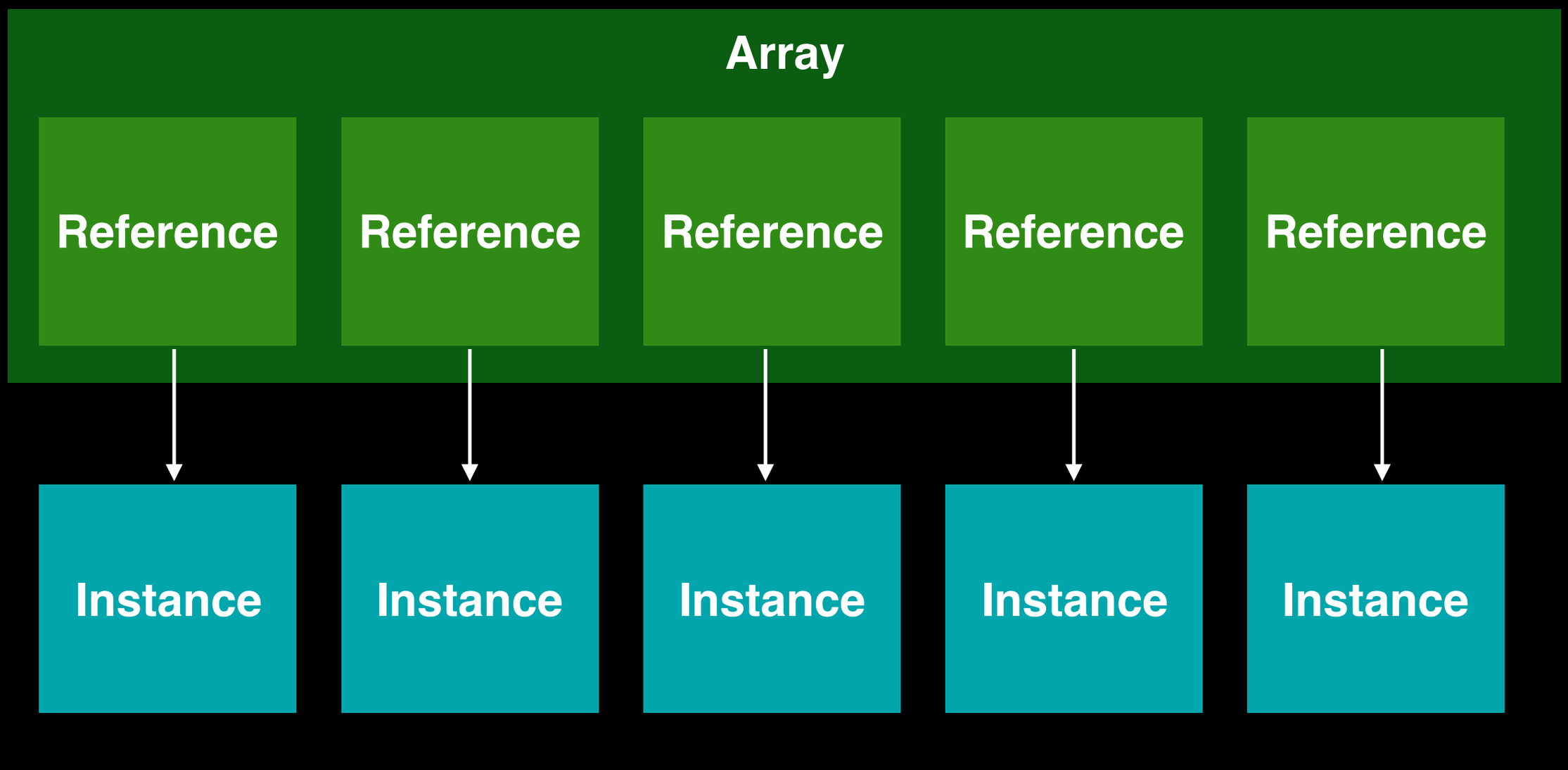
```swift
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}

func exam(person: TangQiao?) {
    // ...
}

var x: TangQiao? = TangQiao()
var y  = x

exam(person: y)

y = nil
x = nil
```

Reference count costs a lot of CPU sometimes

```
var array: [TangQiao] = []

for t in array {
    // increase RC
    // decrease RC
}
```

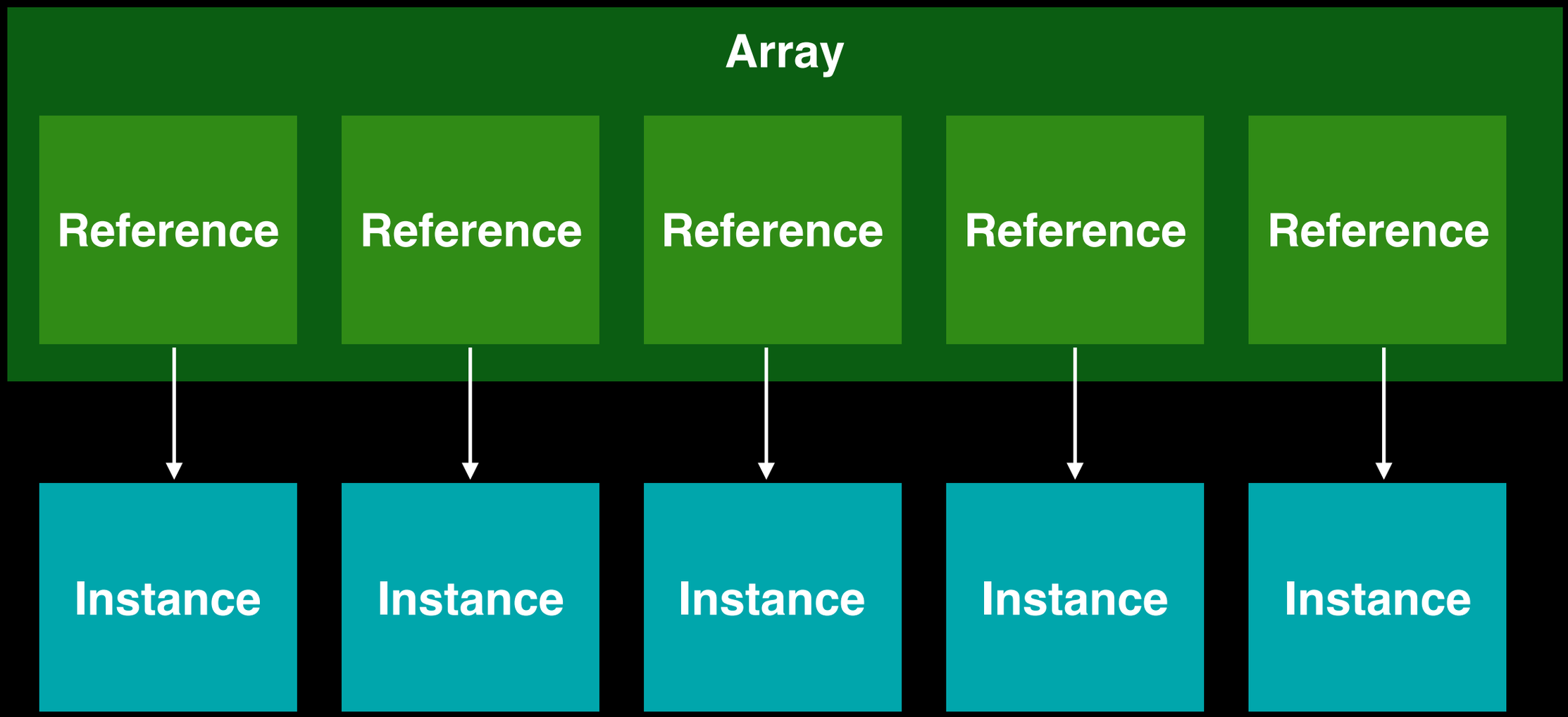


```
var array: [TangQiao] = []

for t in array {
    // increase RC
    // decrease RC
}
```



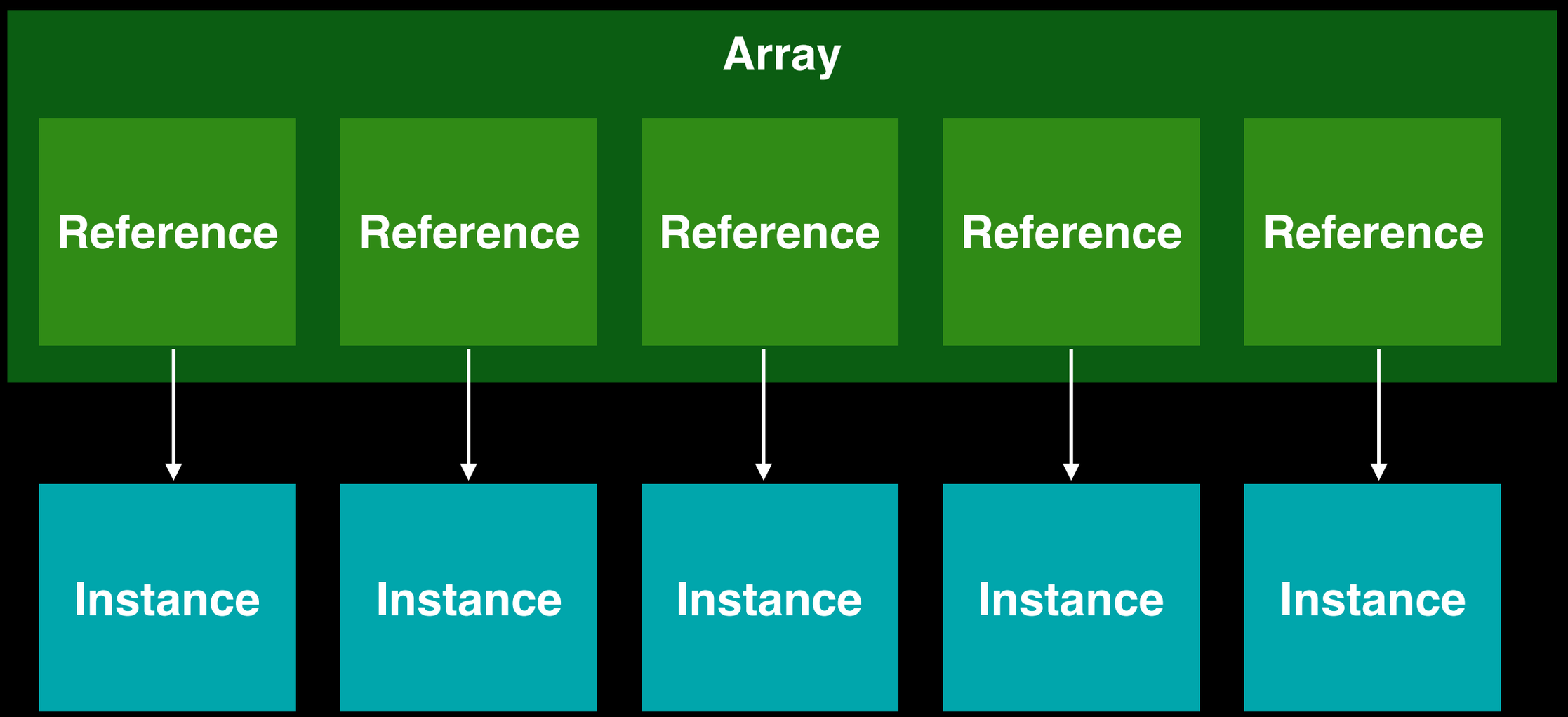
Swift has value type:
struct

```
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}
```



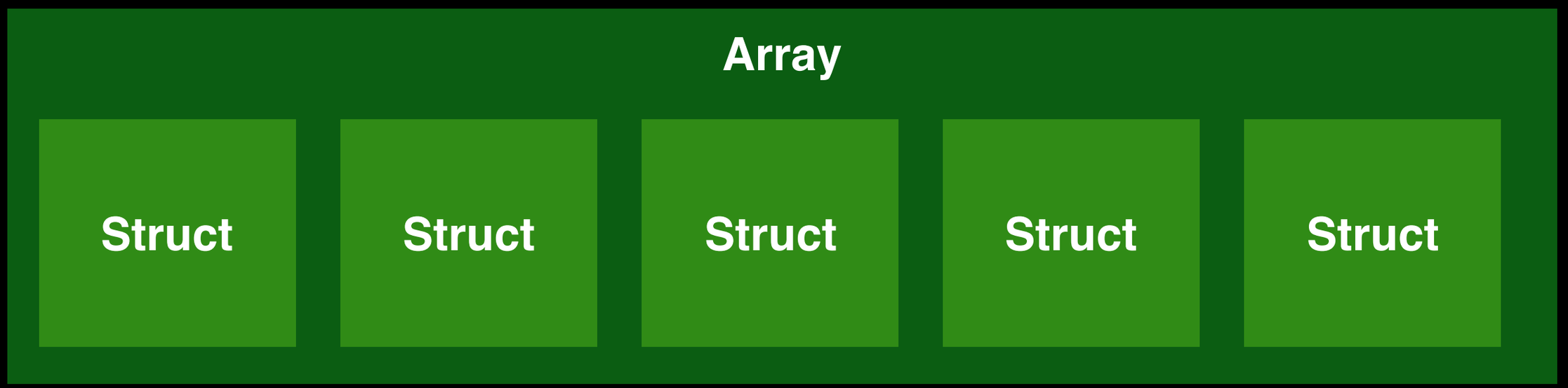
```swift
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}
```

```swift
struct TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}
```

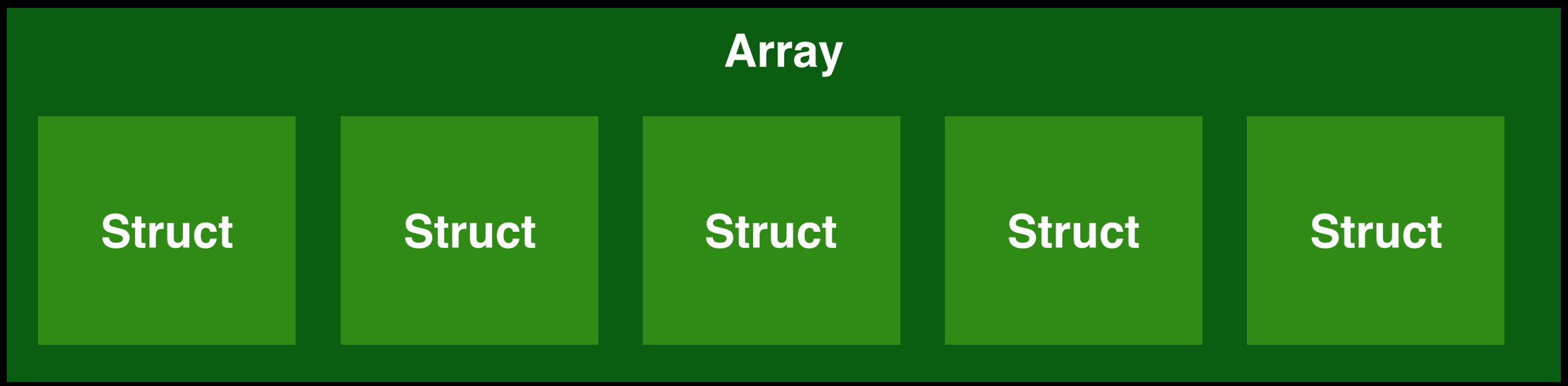


```swift
class TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}
```

```swift
struct TangQiao {
    var alive = true
    var age = 32
    var height = 1.76
}
```



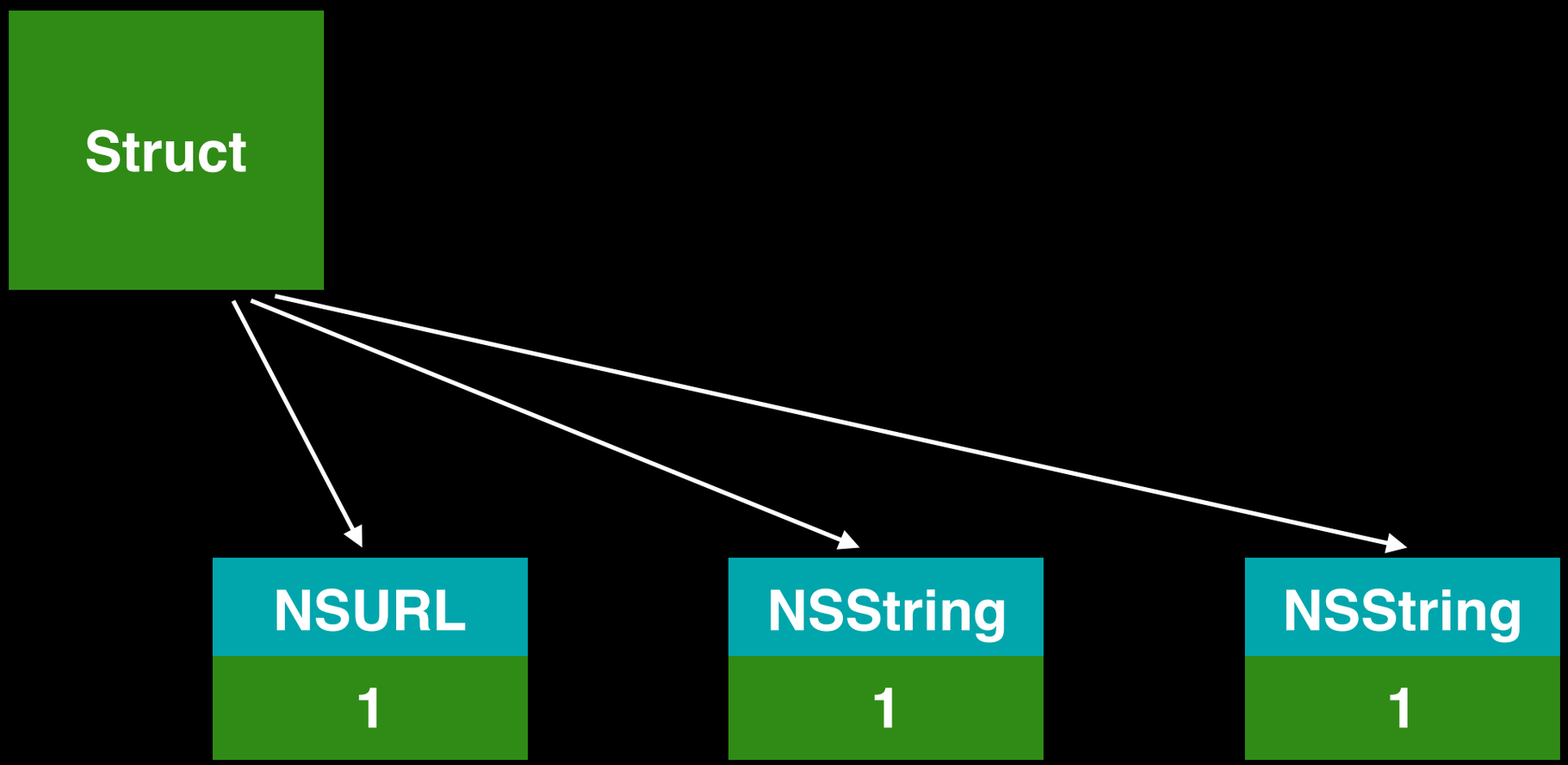
```
var array: [TangQiao] = []

for t in array {
    // increase RC
    // decrease RC
}
```



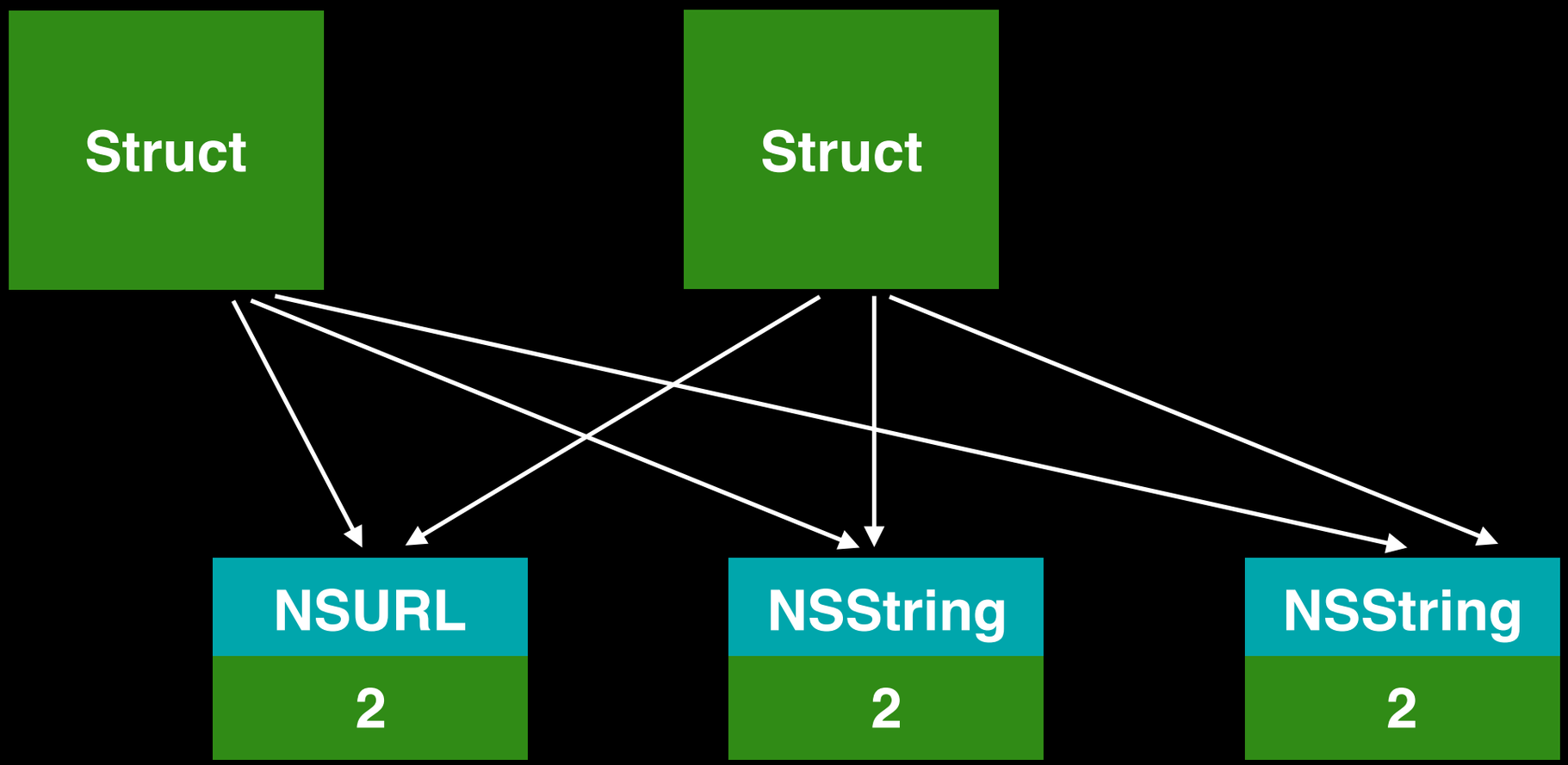
Struct world is not perfect

```
struct TangQiao {
    var website = NSURL(string: "http://blog.devtang.com")
    var name = NSString(string: "tangqiaoboy")
    var addr = NSString(string: "address")
}
var x = TangQiao()
var y = x
```

```
struct TangQiao {
    var website = NSURL(string: "http://blog.devtang.com")
    var name = NSString(string: "tangqiaoboy")
    var addr = NSString(string: "address")
}
var x = TangQiao()
var y = x
```



```swift
struct TangQiao {
    var website = NSURL(string: "http://blog.devtang.com")
    var name = NSString(string: "tangqiaoboy")
    var addr = NSString(string: "address")
}
var x = TangQiao()
var y = x
```



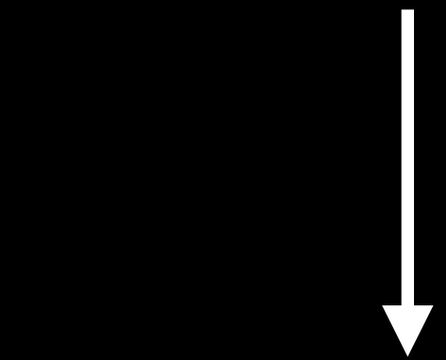
```
struct TangQiao {
    var website = NSURL(string: "http://blog.devtang.com")
    var name = NSString(string: "tangqiaoboy")
    var addr = NSString(string: "address")
}
var x = TangQiao()
var y = x
```

Struct assignment will cause lots of RC operations, if it contains many class member variables.

We can use a wrapper class to solve this issue.

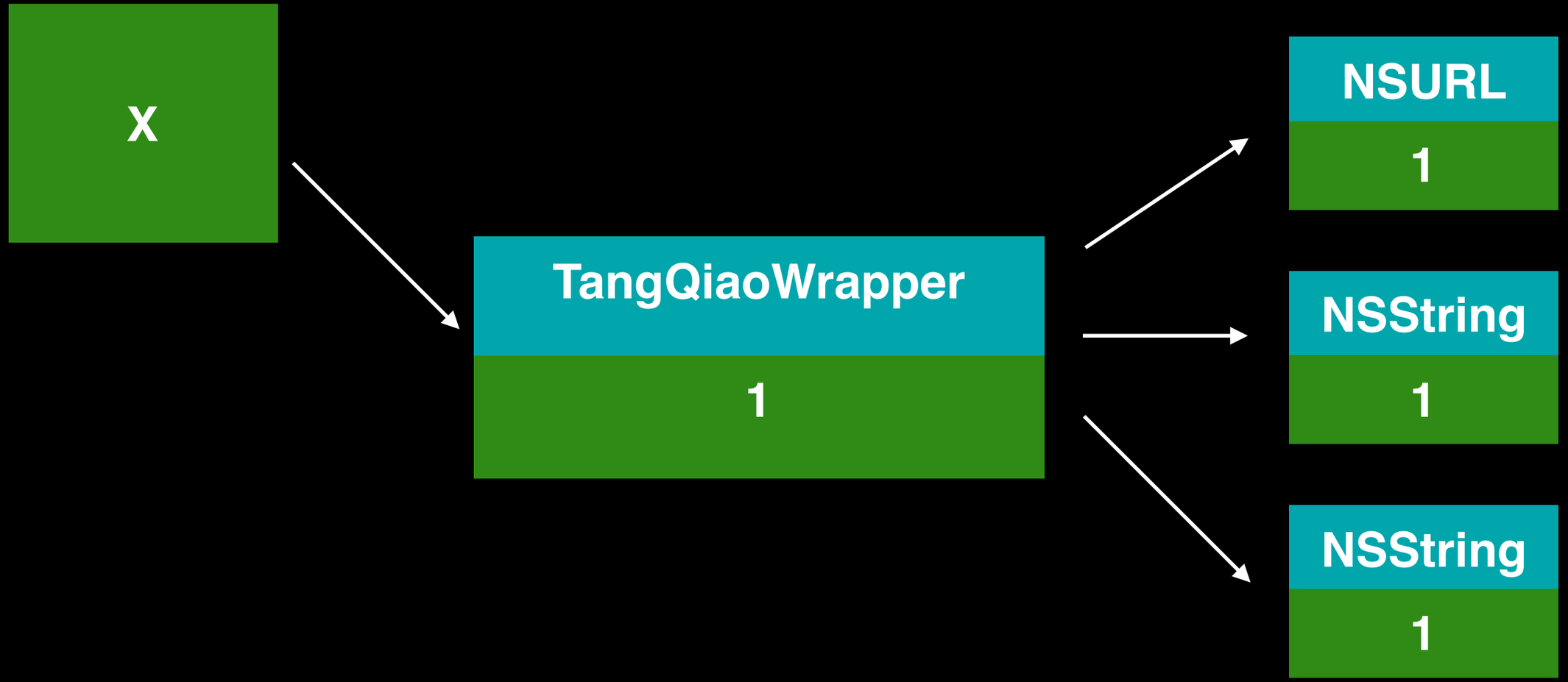
```
struct TangQiao {
    var website = NSURL(string: "http://blog.devtang.com")
    var name = NSString(string: "tangqiaoboy")
    var addr = NSString(string: "address")
}
var x = TangQiao()
var y = x
```

```swift
struct TangQiao {
    var website = NSURL(string: "http://blog.devtang.com")
    var name = NSString(string: "tangqiaoboy")
    var addr = NSString(string: "address")
}
var x = TangQiao()
var y = x
```

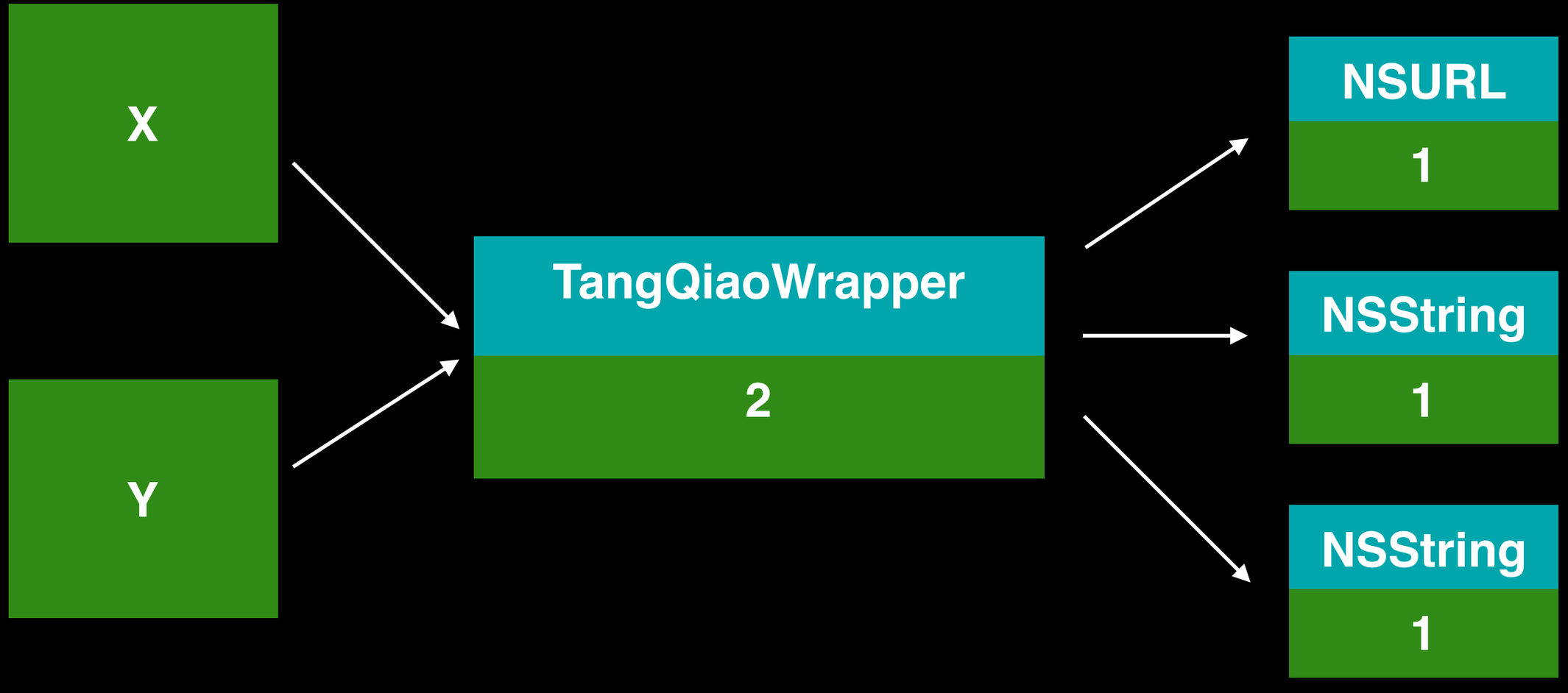


```swift
struct TangQiao {
    var member: TangQiaoWrapper = TangQiaoWrapper()
}

class TangQiaoWrapper {
    var website = NSURL(string: "http://blog.devtang.com")
    var name = NSString(string: "tangqiaoboy")
    var addr = NSString(string: "address")
}
var x = TangQiao()
var y = x
```

```
struct TangQiao {
    var member: TangQiaoWrapper = TangQiaoWrapper()
}

class TangQiaoWrapper {
    var website = NSURL(string: "http://blog.devtang.com")
    var name = NSString(string: "tangqiaoboy")
    var addr = NSString(string: "address")
}
var x = TangQiao()
var y = x
```



```
struct TangQiao {
    var member: TangQiaoWrapper = TangQiaoWrapper()
}

class TangQiaoWrapper {
    var website = NSURL(string: "http://blog.devtang.com")
    var name = NSString(string: "tangqiaoboy")
    var addr = NSString(string: "address")
}
var x = TangQiao()
var y = x
```



Conclusion

Using a wrapper class to hold class members if your struct has many class member variables.

The internal implementation of struct with protocol

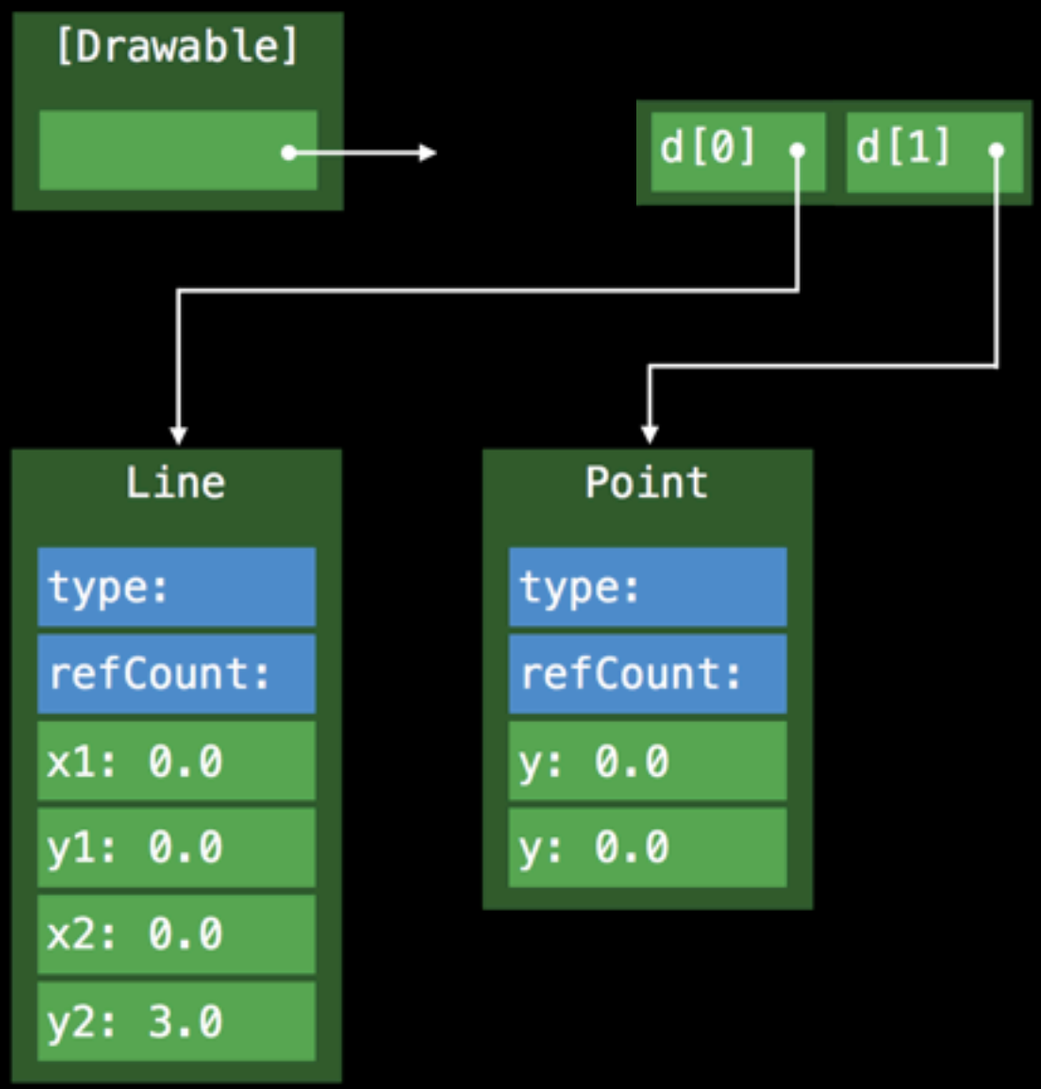
Protocol Witness Table

&

Value Witness Table

An array of class instances

```
class Drawable { func draw() {} }

class Point : Drawable {
   var x, y: Double
   override func draw() { … }
}
class Line : Drawable {
   var x1, y1, x2, y2: Double
   override func draw() { … }
}

var drawables: [Drawable]
for d in drawables {
   d.draw()
}
```

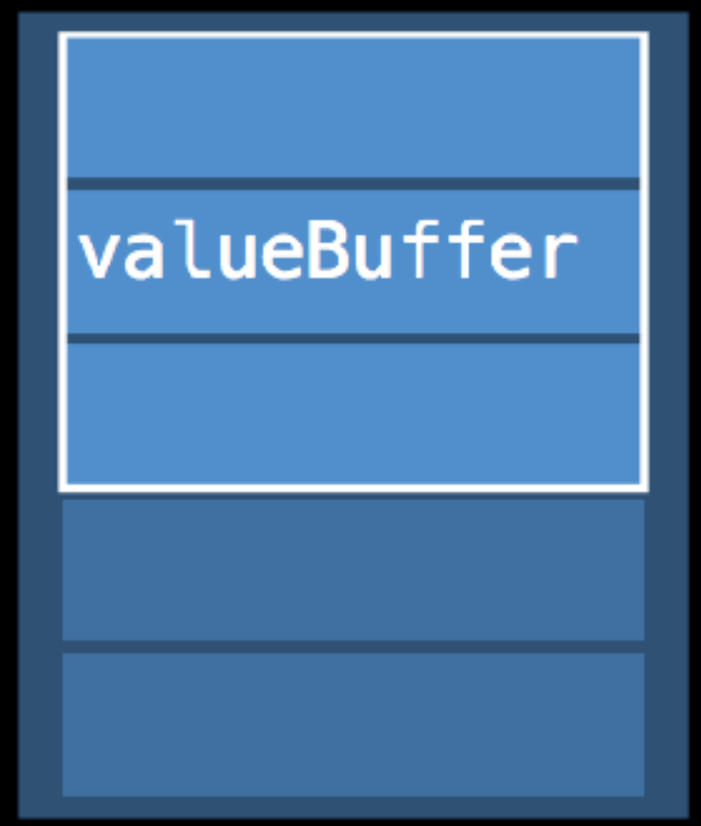


An array of struct instances

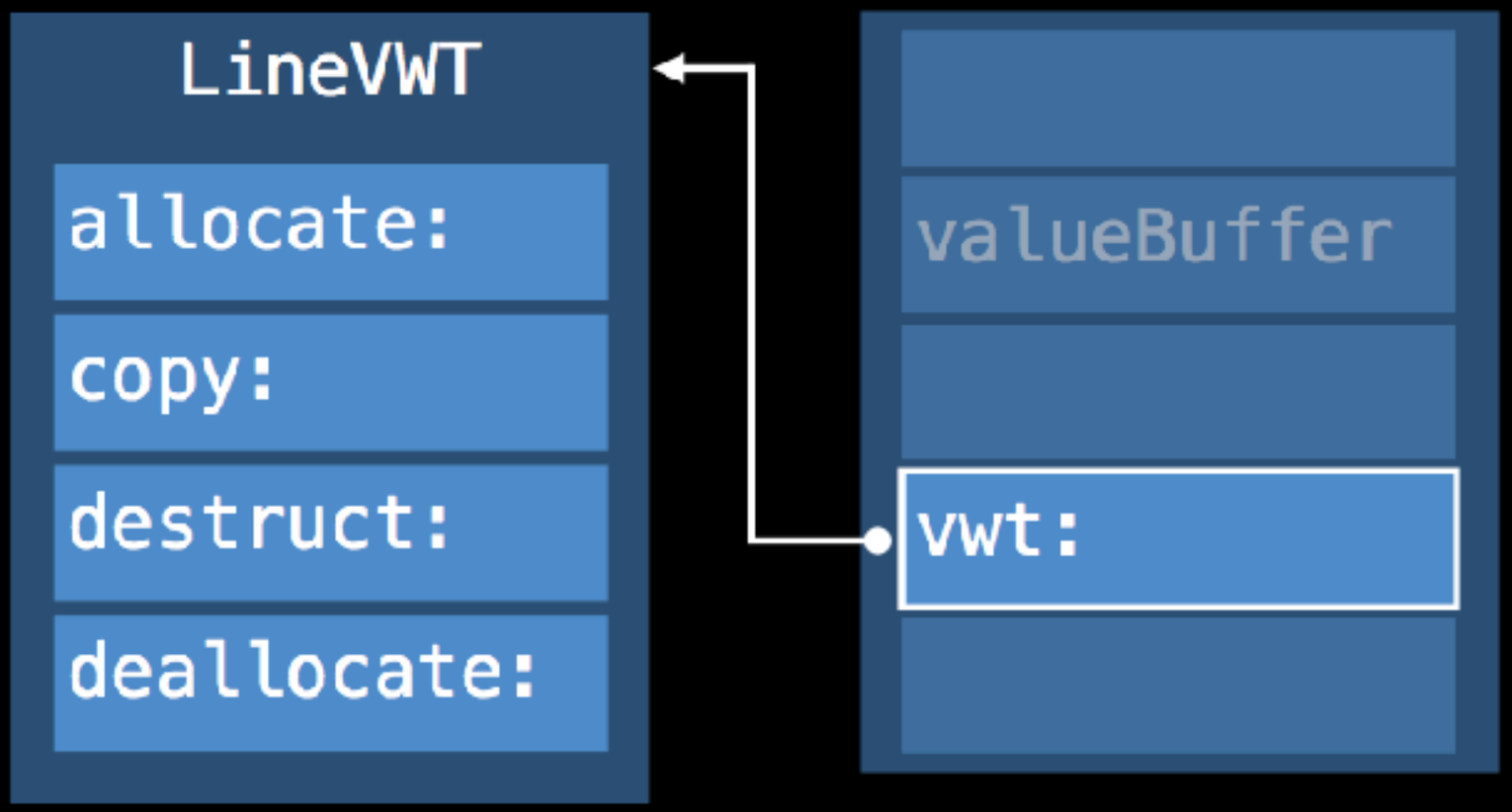
```
protocol Drawable { func draw() }

struct Point : Drawable {
    var x, y: Double
    func draw() { … }
}
struct Line : Drawable {
    var x1, y1, x2, y2: Double
    func draw() { … }
}

var drawables: [Drawable]
for d in drawables {
    d.draw()
}
```

The Existential Container

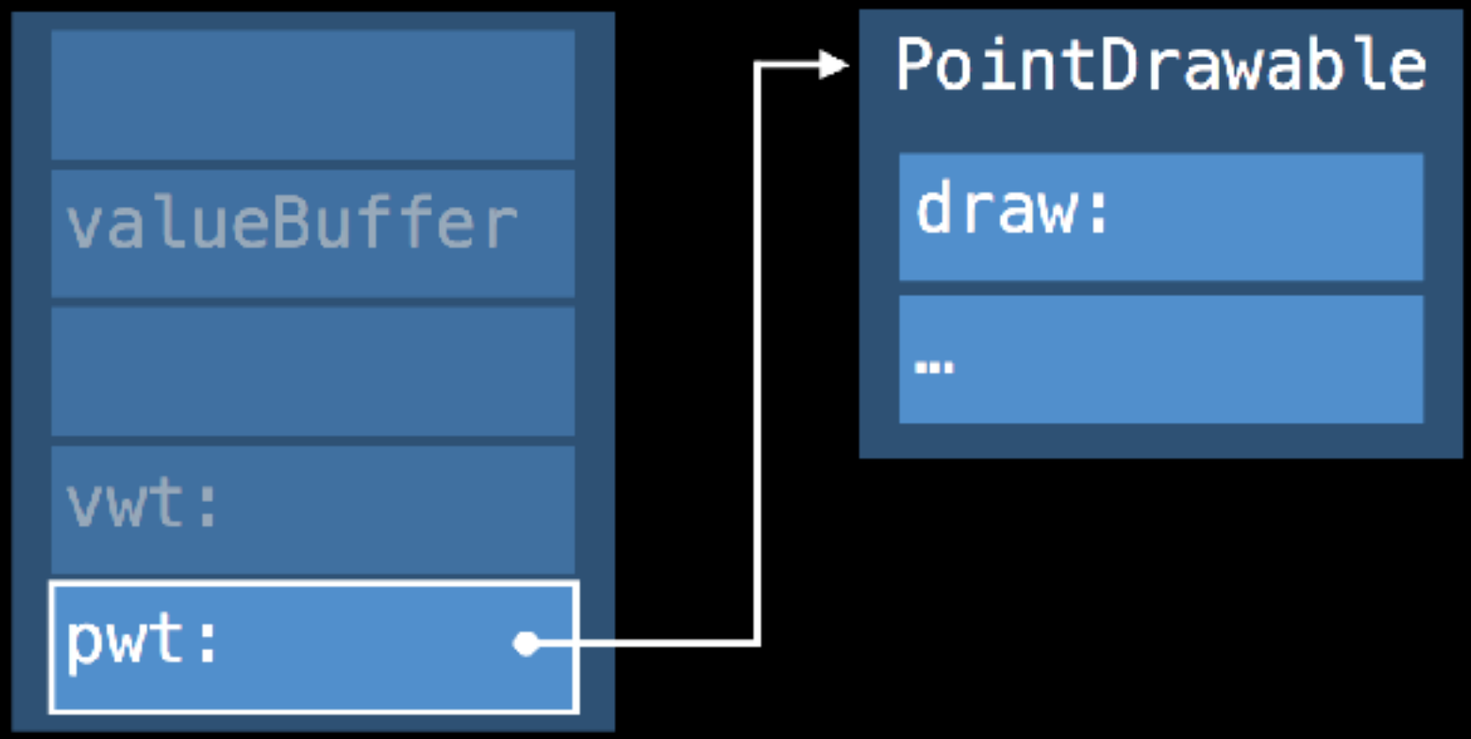
- 5 words (40 bytes in 64-bit CPU)
- inline value buffer: 3 words

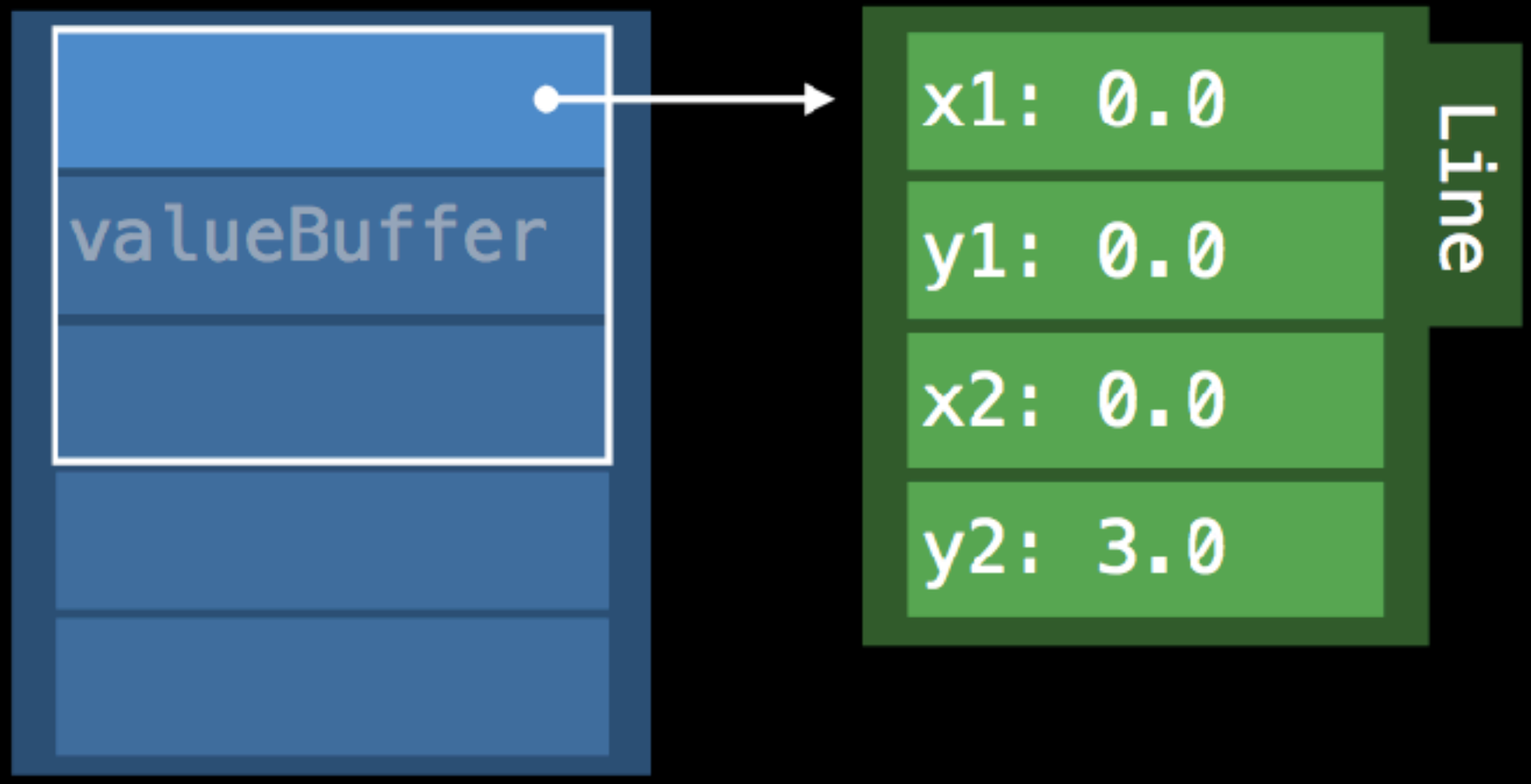


Value Witness Table



Protocol Witness Table





What if my struct is bigger than 3 words?

```
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
   local.draw()
}
let val : Drawable = Point()
drawACopy(val)
```

```
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
   local.draw()
}
let val : Drawable = Point()
drawACopy(val)
```

```
// Generated code
struct ExistContDrawable {
   var valueBuffer: (Int, Int, Int)
   var vwt: ValueWitnessTable
   var pwt: DrawableProtocolWitnessTable
}
```

```
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
    local.draw()
}
let val : Drawable = Point()
drawACopy(val)
```

```
// Generated code
func drawACopy(val: ExistContDrawable) {
```

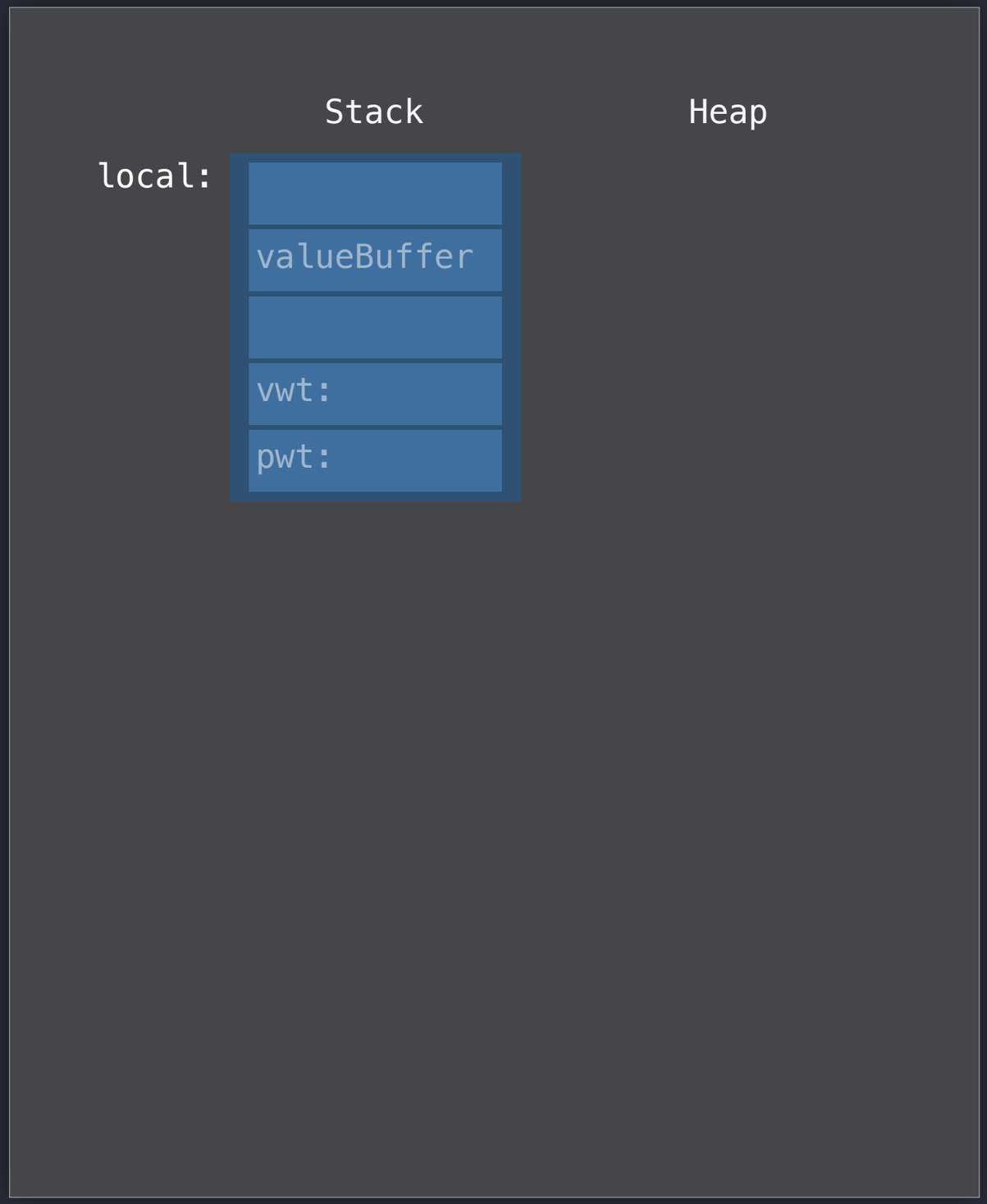
```
// Protocol Types
// The Existential in action
func drawACopy(local : Drawable) {
    local.draw()
}
let val : Drawable = Point()
drawACopy(val)
```

```
let local = val
```

```
// Generated code
func drawACopy(val: ExistContDrawable) {
```

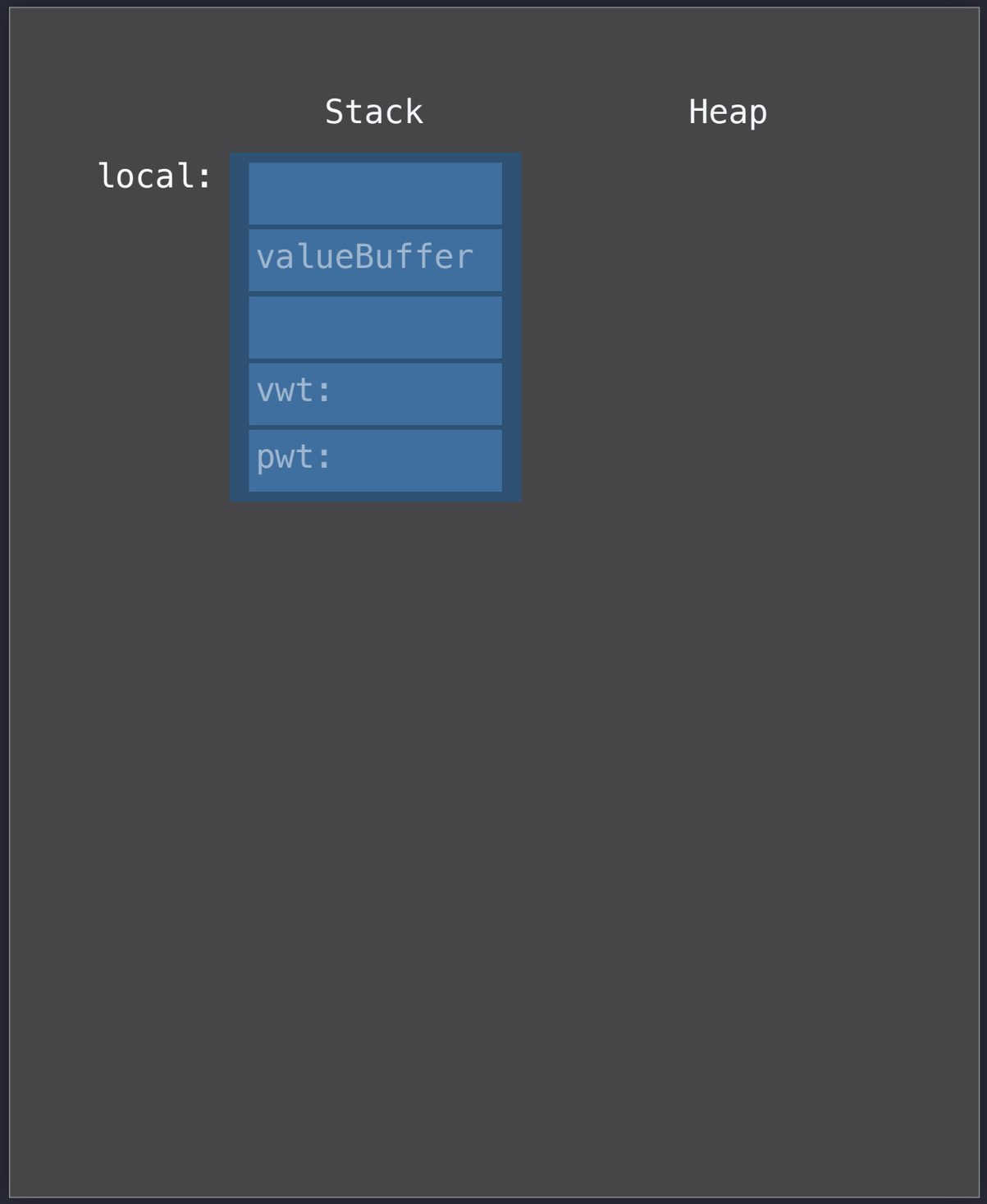
```
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
   local.draw()
}
let val : Drawable = Point()
drawACopy(val)
```

```
// Generated code
func drawACopy(val: ExistContDrawable) {
   var local = ExistContDrawable()
```



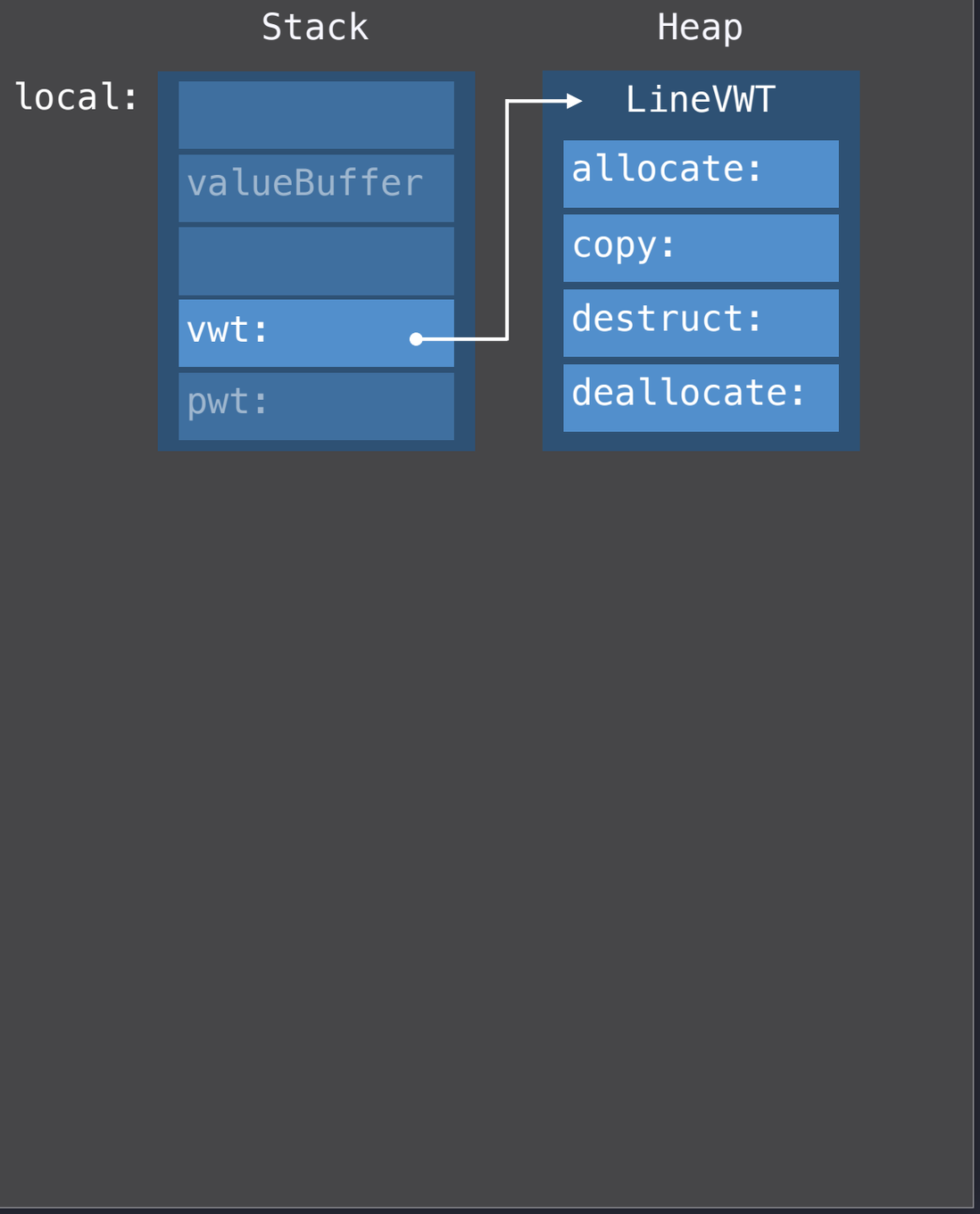
```
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
   local.draw()
}
let val : Drawable = Point()
drawACopy(val)
```

```
// Generated code
func drawACopy(val: ExistContDrawable) {
   var local = ExistContDrawable()
   let vwt = val.vwt
   let pwt = val.pwt
   local.type = type
   local.pwt = pwt
```



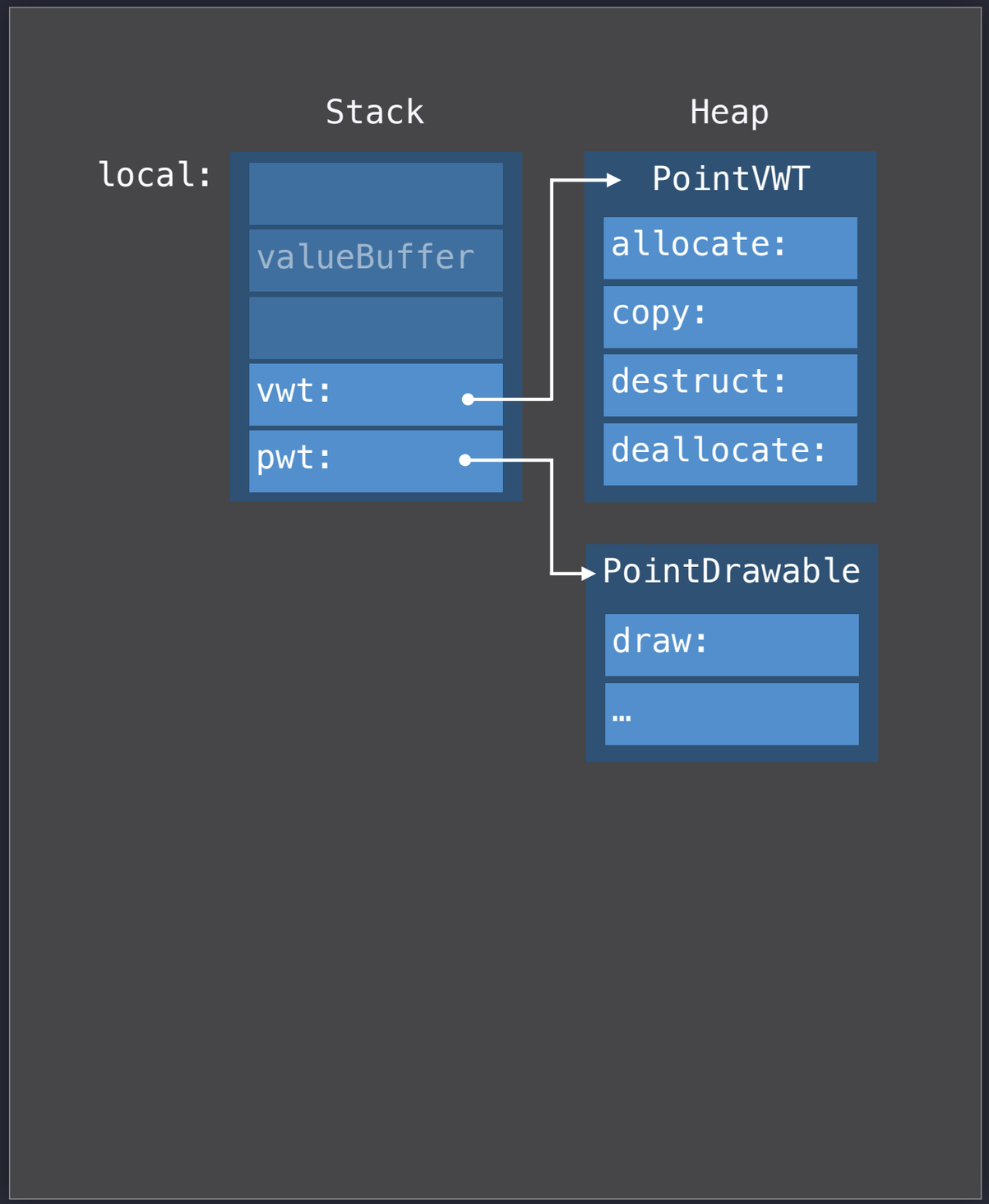
```
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
   local.draw()
}
let val : Drawable = Point()
drawACopy(val)
```

```
// Generated code
func drawACopy(val: ExistContDrawable) {
   var local = ExistContDrawable()
   let vwt = val.vwt
   let pwt = val.pwt
   local.type = type
   local.pwt = pwt
```



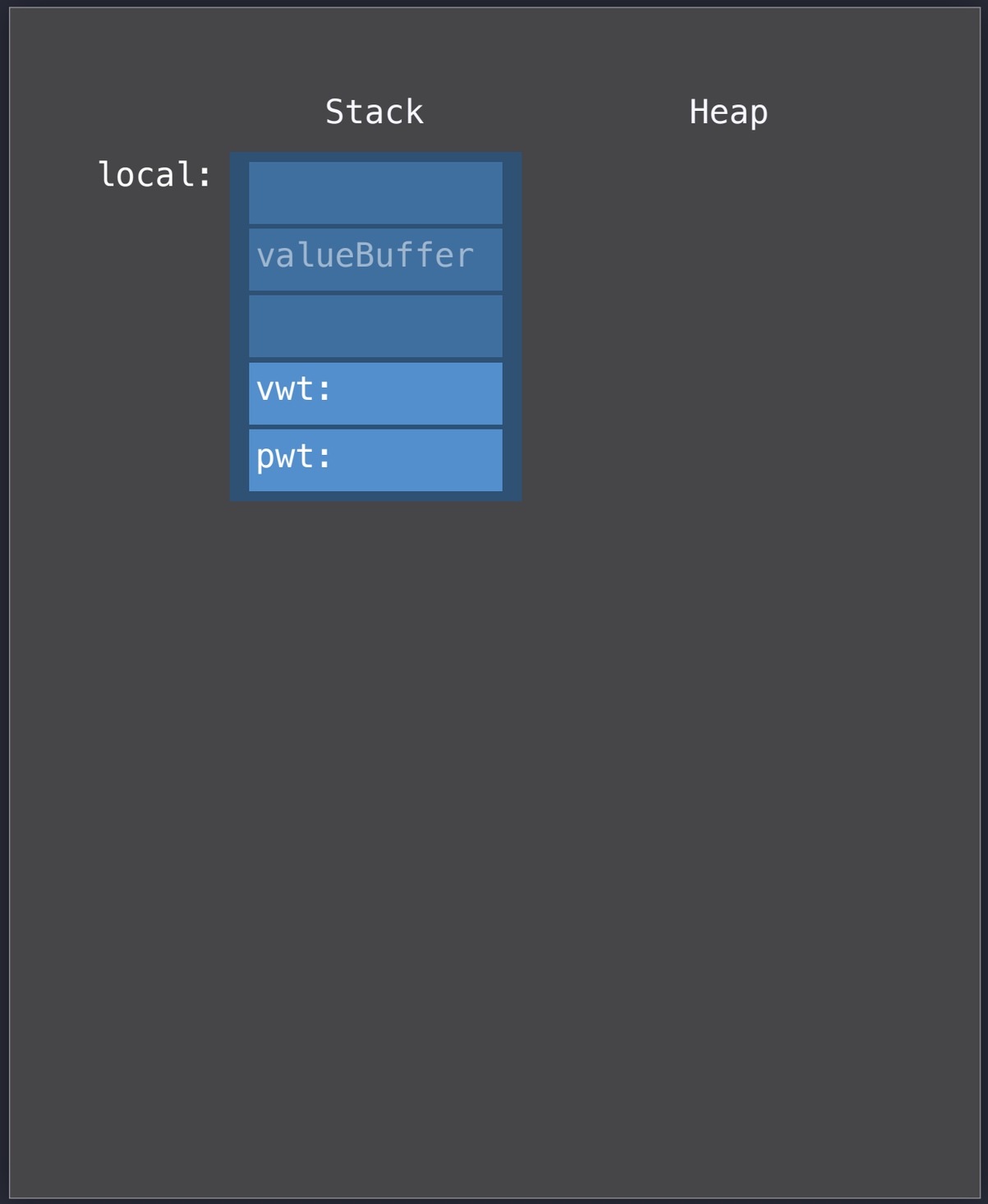
```
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
   local.draw()
}
let val : Drawable = Point()
drawACopy(val)
```

```
// Generated code
func drawACopy(val: ExistContDrawable) {
   var local = ExistContDrawable()
   let vwt = val.vwt
   let pwt = val.pwt
   local.type = type
   local.pwt = pwt
```



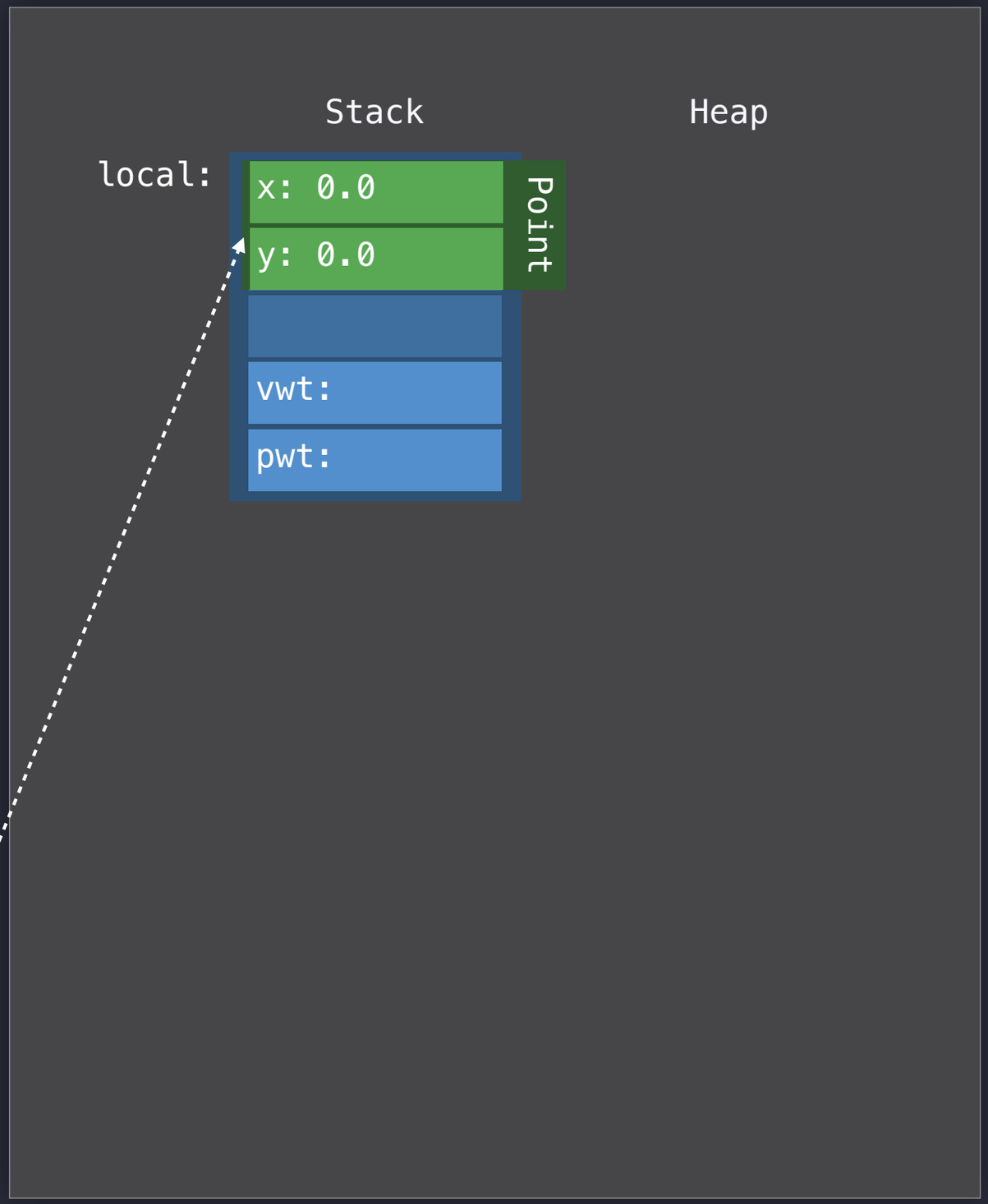
```swift
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
    local.draw()
}
let val : Drawable = Point()
drawACopy(val)
```

```swift
// Generated code
func drawACopy(val: ExistContDrawable) {
    var local = ExistContDrawable()
    let vwt = val.vwt
    let pwt = val.pwt
    local.type = type
    local.pwt = pwt
    vwt.allocateBufferAndCopyValue(&local, val)
```



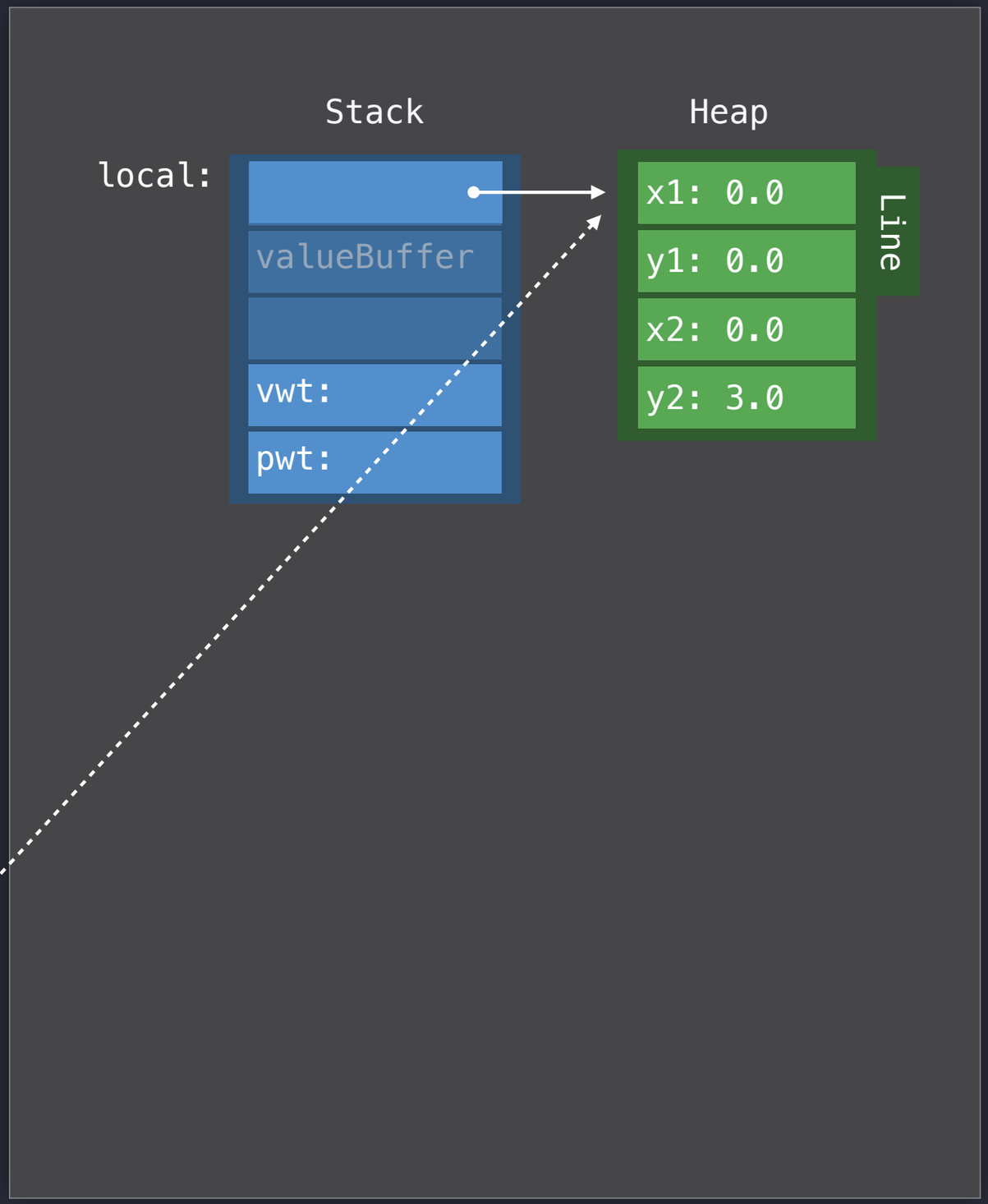
```swift
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
    local.draw()
}
let val : Drawable = Point()
drawACopy(val)
```

```swift
// Generated code
func drawACopy(val: ExistContDrawable) {
    var local = ExistContDrawable()
    let vwt = val.vwt
    let pwt = val.pwt
    local.type = type
    local.pwt = pwt
    vwt.allocateBufferAndCopyValue(&local, val)
```



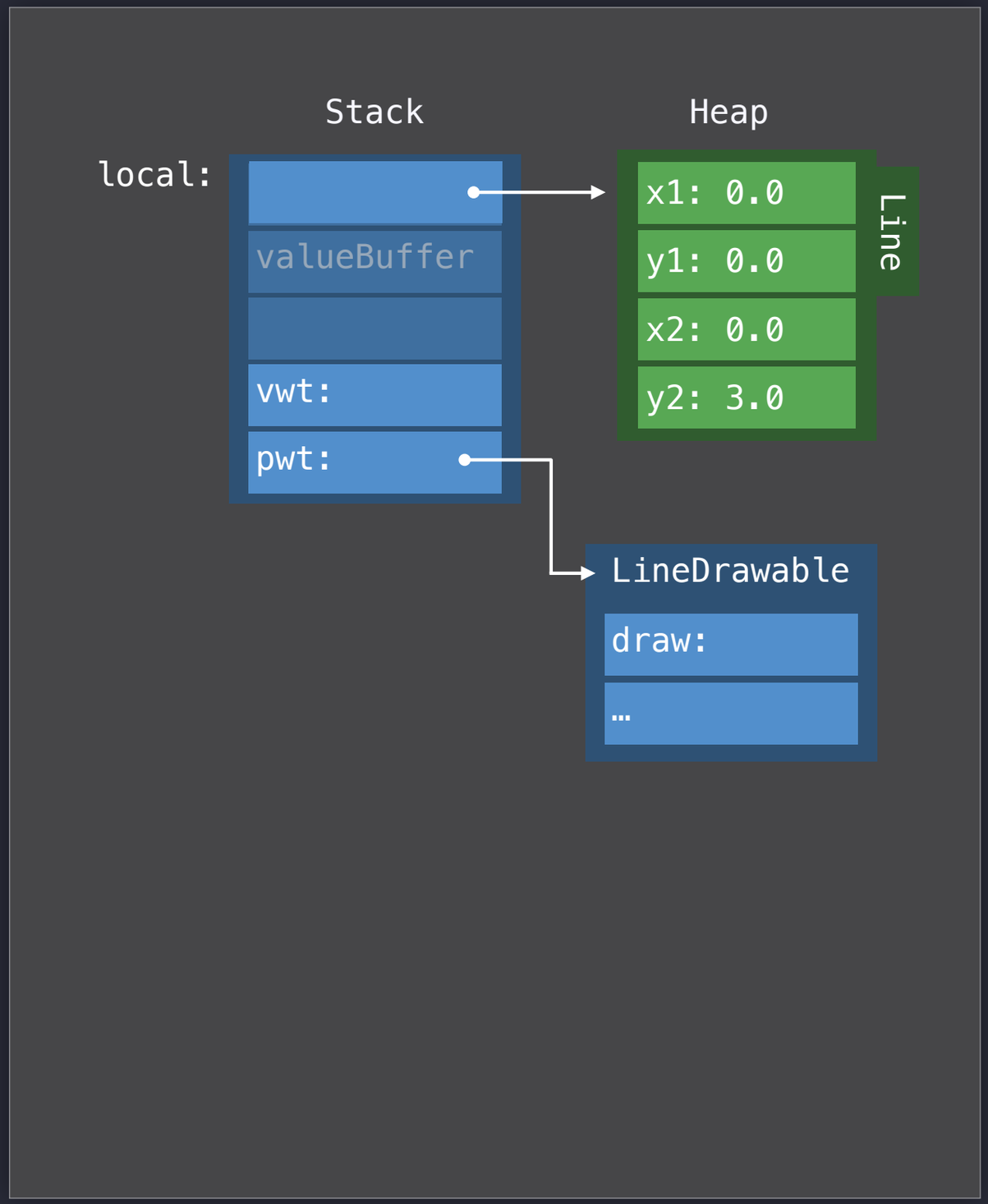
```
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
    local.draw()
}
let val : Drawable = Line()
drawACopy(val)
```

```
// Generated code
func drawACopy(val: ExistContDrawable) {
    var local = ExistContDrawable()
    let vwt = val.vwt
    let pwt = val.pwt
    local.type = type
    local.pwt = pwt
    vwt.allocateBufferAndCopyValue(&local, val)
```



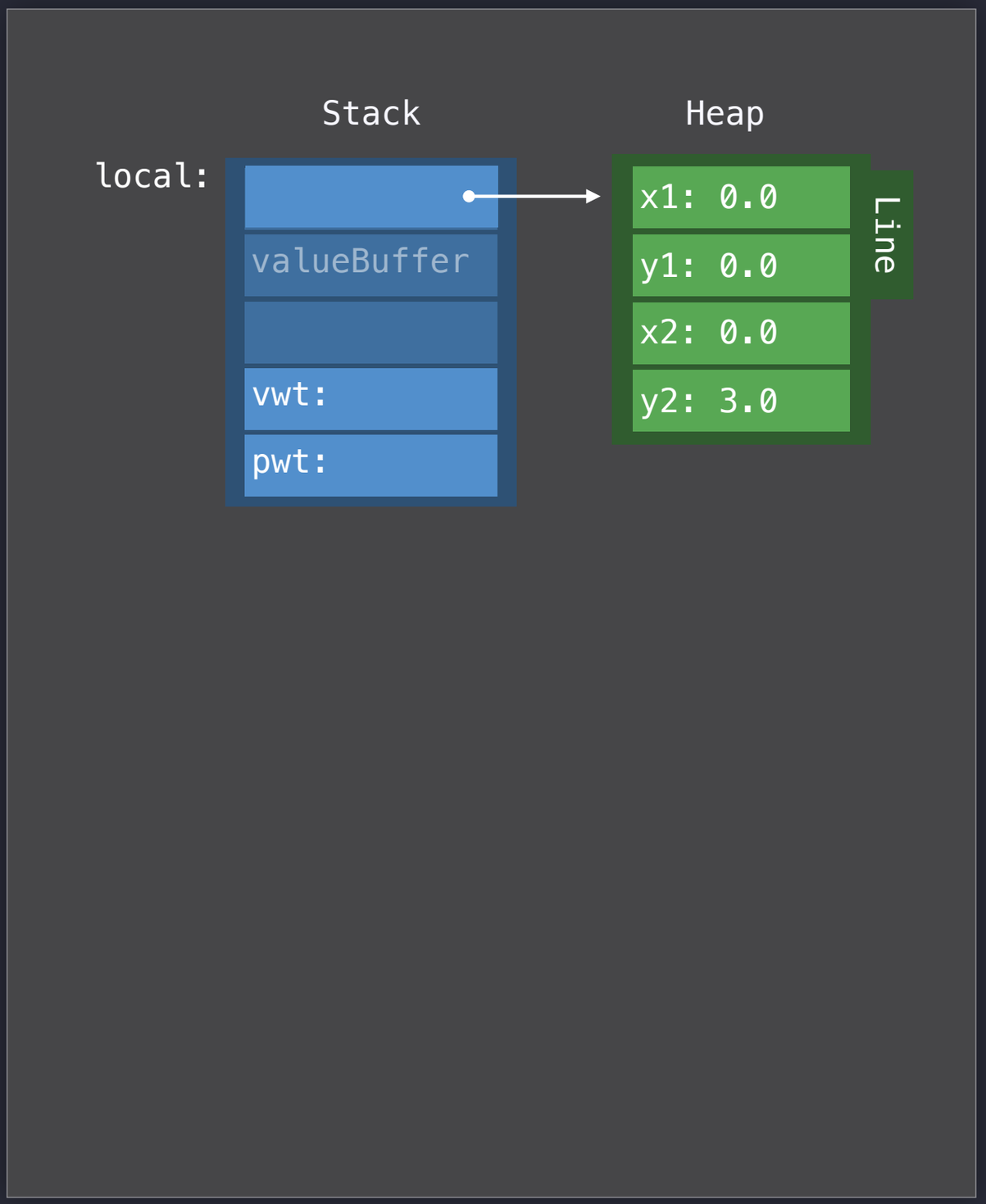
```
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
    local.draw()
}
let val : Drawable = Line()
drawACopy(val)
```

```
// Generated code
func drawACopy(val: ExistContDrawable) {
    var local = ExistContDrawable()
    let vwt = val.vwt
    let pwt = val.pwt
    local.type = type
    local.pwt = pwt
    vwt.allocateBufferAndCopyValue(&local, val)
    pwt.draw(vwt.projectBuffer(&local))
```



```
// Protocol Types
// The Existential Container in action
func drawACopy(local : Drawable) {
   local.draw()
}
let val : Drawable = Line()
drawACopy(val)
```

```
// Generated code
func drawACopy(val: ExistContDrawable) {
   var local = ExistContDrawable()
   let vwt = val.vwt
   let pwt = val.pwt
   local.type = type
   local.pwt = pwt
   vwt.allocateBufferAndCopyValue(&local, val)
   pwt.draw(vwt.projectBuffer(&local))
   vwt.destructAndDeallocateBuffer(temp)
}
```



LLDB Commands

LLDB

- (lldb) breakpoint set -a 0x100001c1a
- (lldb) continue

LLDB

- (lldb) di -s 0x1eb8 -c 20

LLDB

- (lldb) x -s8 -c5 -fx $rdi
    - $rdi will be the first argument when calling method

LLDB

- re r rdi rsi rdx rcx rax
  - read register usage information
  - rcx

```swift
protocol Drawable {
    func draw()
}
```

```swift
struct Point: Drawable {
    var x: Int
    var y: Int
    init() {
        x = 1
        y = 2
    }
    func draw() {
        print("Point draw")
    }
}

struct Line: Drawable {
    var x1: Int
    var y1: Int
    var x2: Int
    var y2: Int
    init() {
        x1 = 5
        y1 = 6
        x2 = 7
        y2 = 8
    }

    func draw() {
        print("Line draw")
    }
}
```

```
protocol Drawable {
    func draw()
}
```

```
struct Point: Drawable {
    var x: Int
    var y: Int
    init() {
        x = 1
        y = 2
    }
    func draw() {
        print("Point draw")
    }
}

struct Line: Drawable {
    var x1: Int
    var y1: Int
    var x2: Int
    var y2: Int
    init() {
        x1 = 5
        y1 = 6
        x2 = 7
        y2 = 8
    }

    func draw() {
        print("Line draw")
    }
}
```

```
let a: Drawable = Point()
a.draw()

let b: Drawable = Line()
b.draw()

let arr: [Drawable] = [a , b]
outputArray(arr)
```

Demo

Review

- Value type has better performance.
    - We need to pay attention, if value type has lots of reference members.
- Existential container can be used for value type + protocol oriented programming.
- LLDB can help us to understand the implementation detail of existential container.

Thanks