Developer Tools

#WWDC15

Optimizing Swift Performance

Session 409

Nadav Rotem Manager, Swift Performance Team
Michael Gottesman Engineer, Swift Performance Team
Joe Grzywacz Engineer, Performance Tools

Agenda

Swift 2.0 performance update

Understanding Swift performance

Using Instruments to analyze the performance of Swift programs

Swift is a Flexible, Safe Programming Language with ARC

Swift is a Flexible, Safe Programming Language with ARC

Flexible	Safe	ARC
function signature specializations	overflow checks removal	ARC optimizer
global variable optimizations	bounds checks elimination	copy forwarding
lock-less metadata caches	obj-c bridge optimizations	heap to stack
generics specializations	checked casting optimizations	code motion
class hierarchy analysis		alias analysis
closure optimizations		reference counting analysis
SSA optimizations		copy-on-write optimizations
call graph analysis		
loop optimizations		
devirtualization		
function inliner		

Swift is a Flexible, Safe Programming Language with ARC

Flexible	Safe	ARC
function signature specializations	overflow checks removal	ARC optimizer
global variable optimizations	bounds checks elimination	copy forwarding
lock-less metadata caches	obj-c bridge optimizations	heap to stack
generics specializations	checked casting optimizations	code motion
class hierarchy analysis		alias analysis
closure optimizations		reference counting analysis
SSA optimizations		copy-on-write optimizations
call graph analysis		
loop optimizations		
devirtualization		
function inliner		

Array Bounds Checks Optimizations

Swift ensures that array access happen in bounds

Swift can lift checks out of loops

O(n) checks become O(1)

```
for i in 0..<n {
    A[i] ^= 13
}
```

Array Bounds Checks Optimizations

Swift ensures that array access happen in bounds

Swift can lift checks out of loops

O(n) checks become O(1)

```
for i in 0..<n {
precondition (i < length)
    A[i] ^= 13
}
```

Array Bounds Checks Optimizations

Swift ensures that array access happen in bounds

Swift can lift checks out of loops

O(n) checks become O(1)

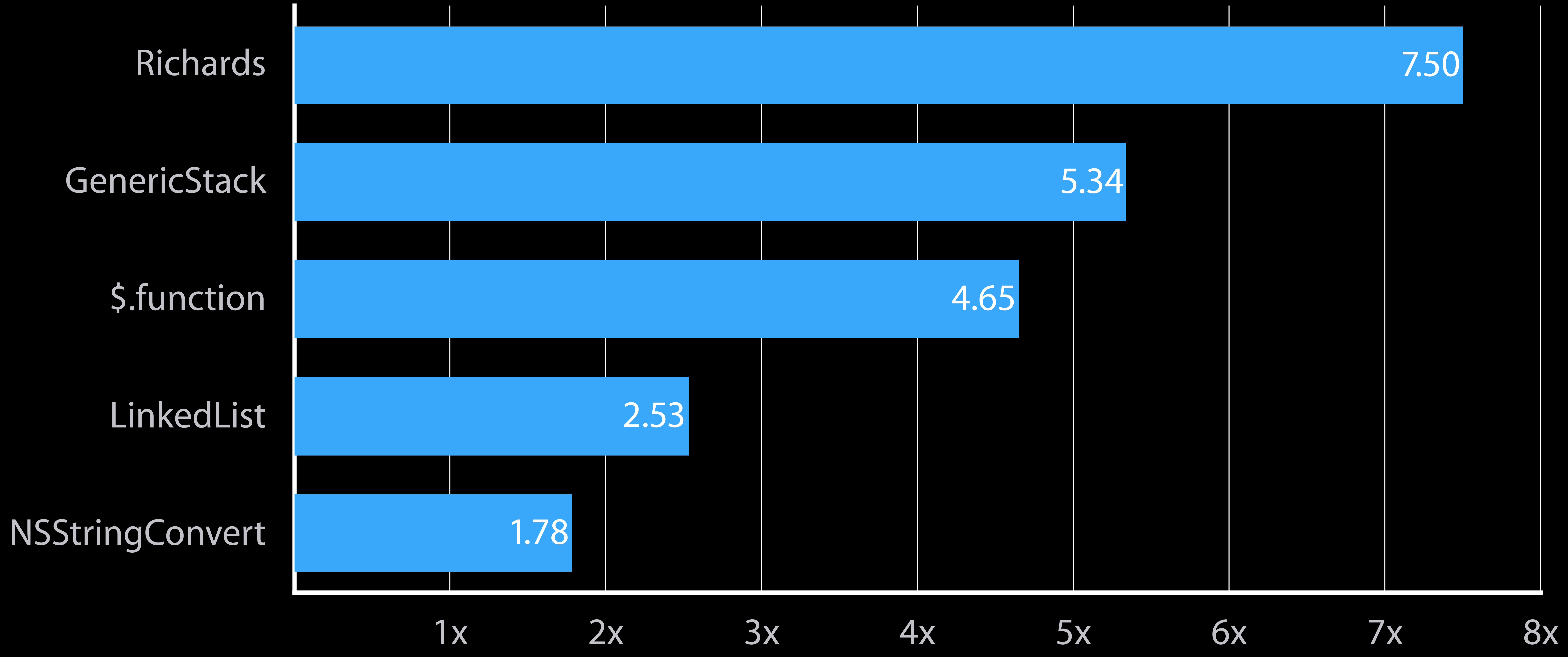
```
precondition (n ≤ length)
for i in 0..<n {

    A[i] ^= 13
}
```

Swift is a Flexible, Safe Programming Language with ARC

Flexible	Safe	ARC
function signature specializations	overflow checks removal	ARC optimizer
global variable optimizations	bounds checks elimination	copy forwarding
lock-less metadata caches	obj-c bridge optimizations	heap to stack
generics specializations	checked casting optimizations	code motion
class hierarchy analysis		alias analysis
closure optimizations		reference counting analysis
SSA optimizations		copy-on-write optimizations
call graph analysis		
loop optimizations		
devirtualization		
function inliner		

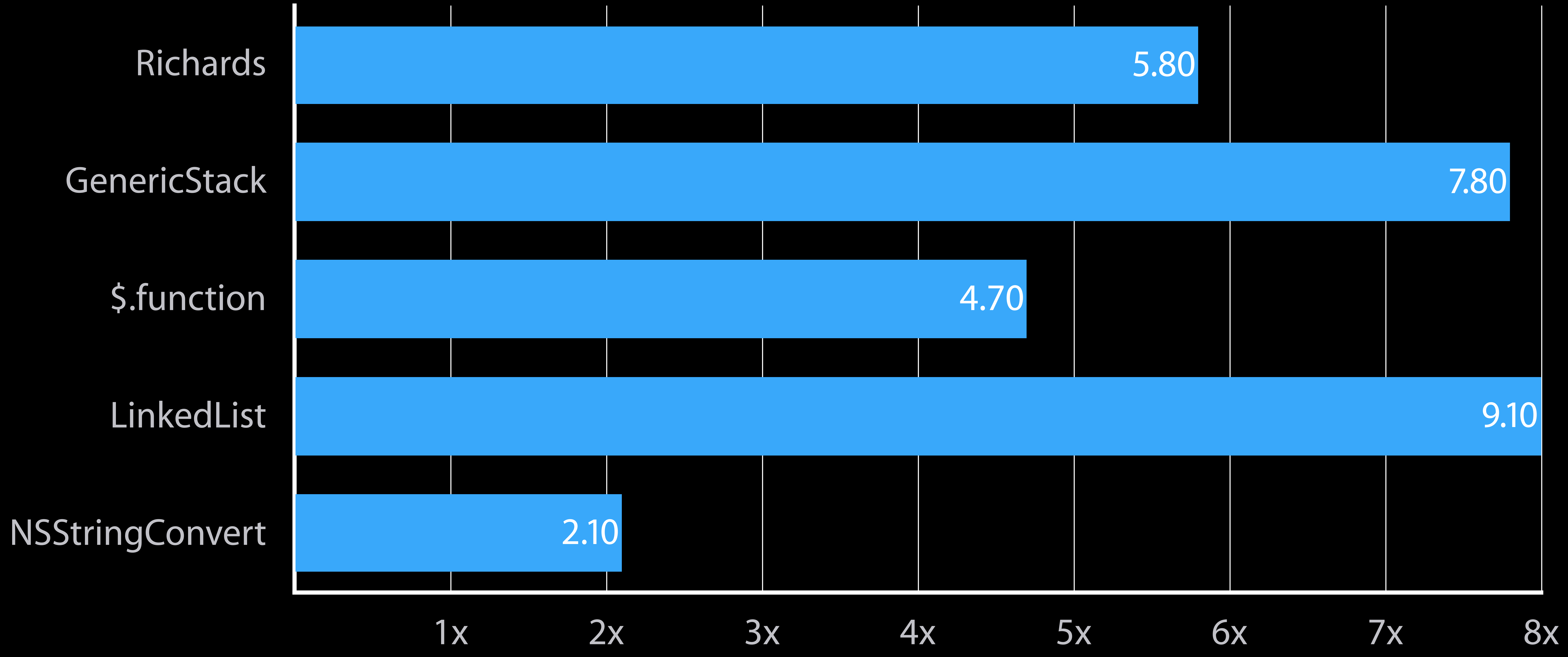
Performance Improvements Since 1.0

Optimized programs (higher is better)



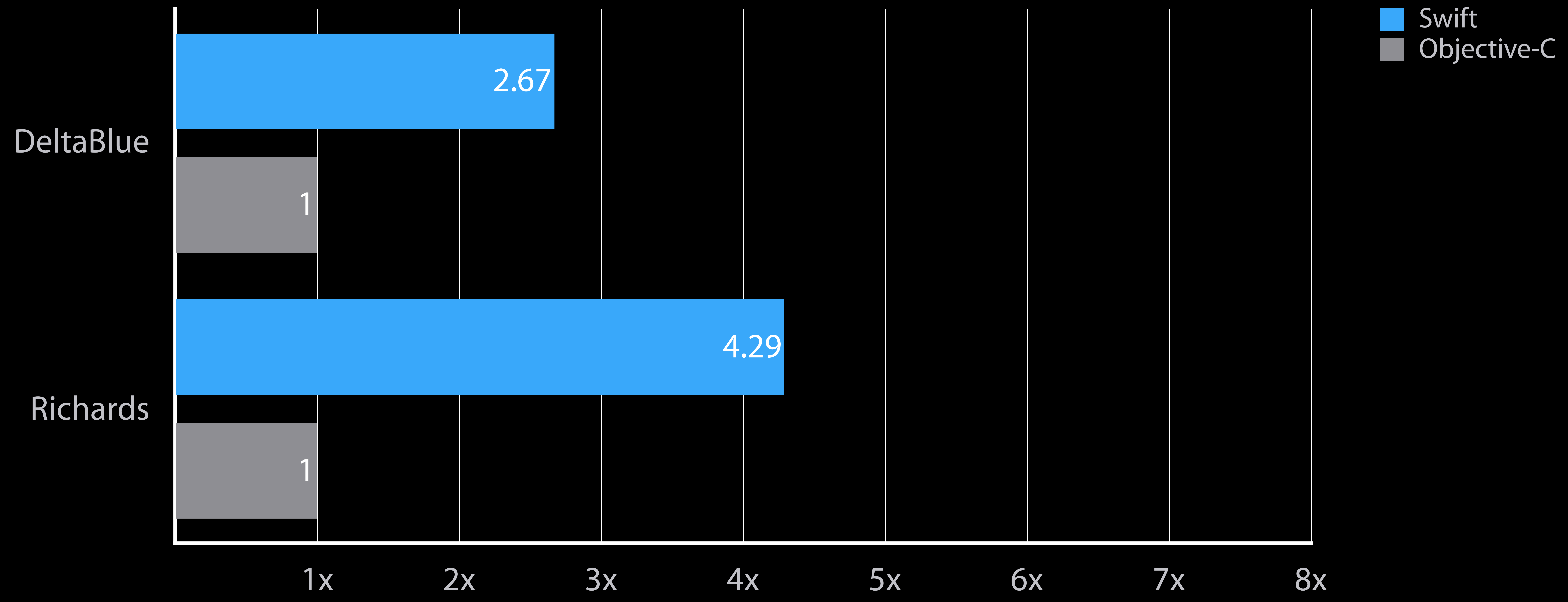
Performance Improvements Since 1.0

Unoptimized programs (higher is better)



Swift vs. Objective-C

Program speed (higher is better)

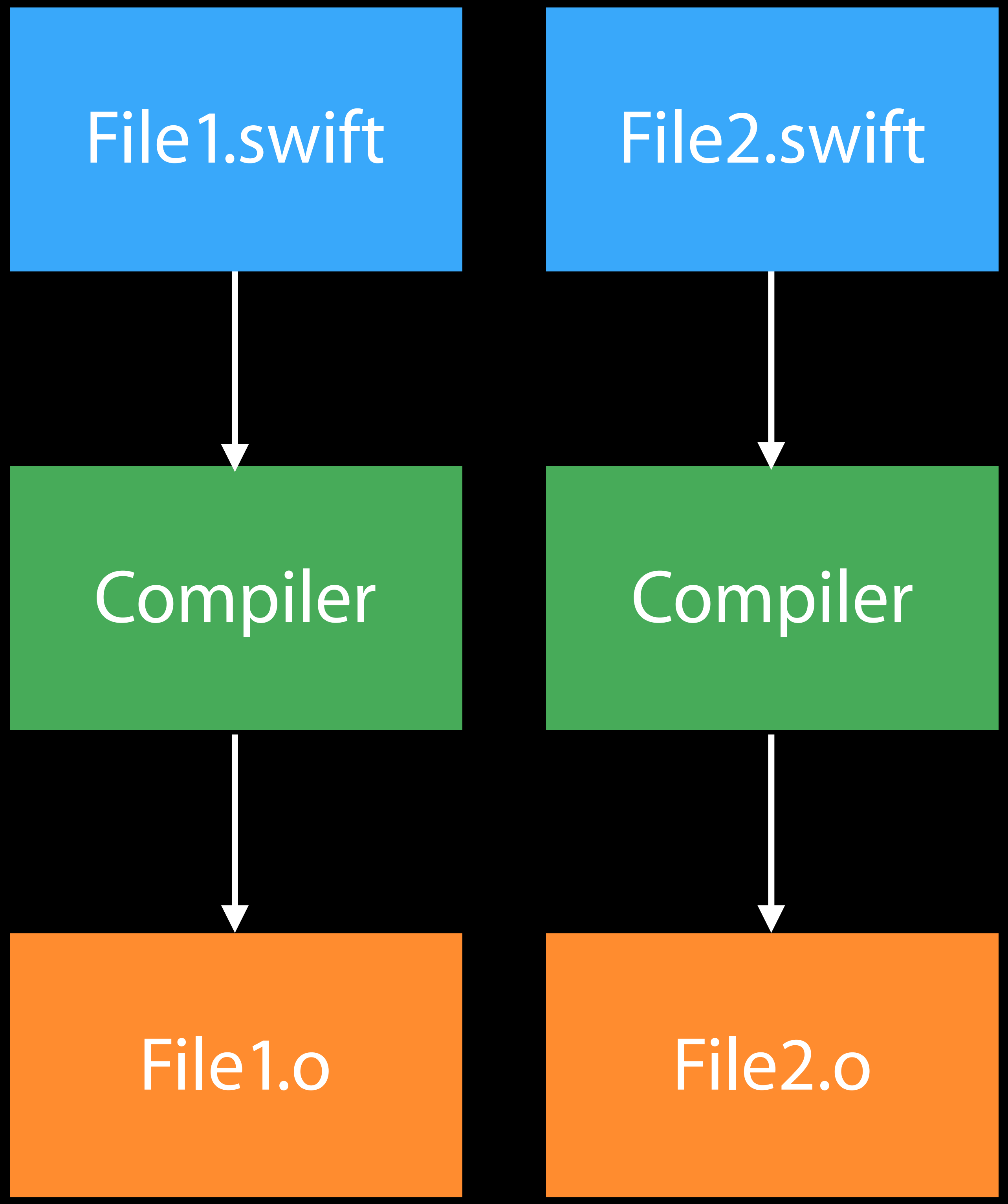


Swift Compilation

Xcode compiles files independently, in parallel

Re-compile only files that need to be updated

Optimizer is limited to scope of one file



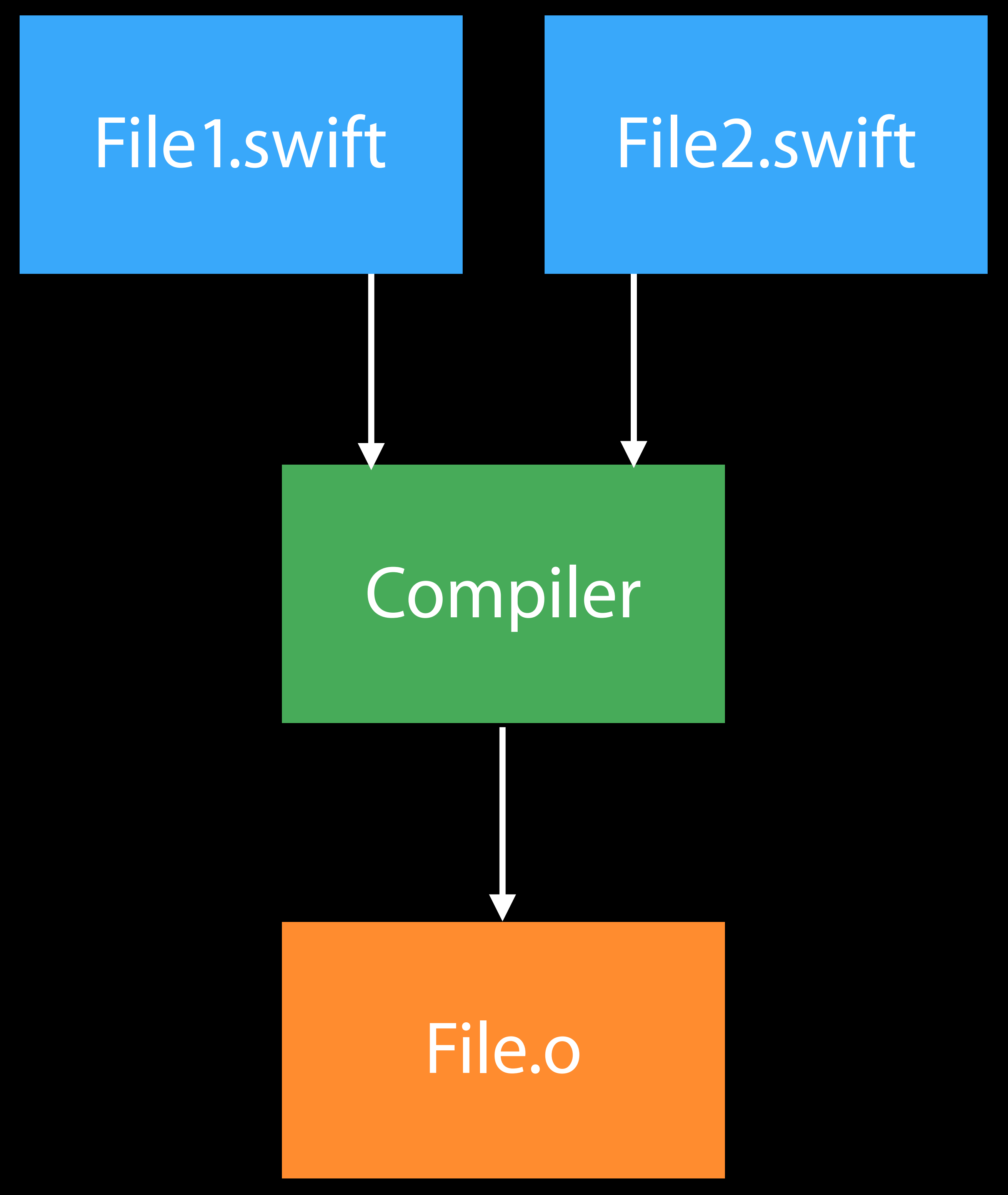
Whole Module Optimizations

Compilation is not limited to the scope of one file

Analyzing the whole module allows better optimizations

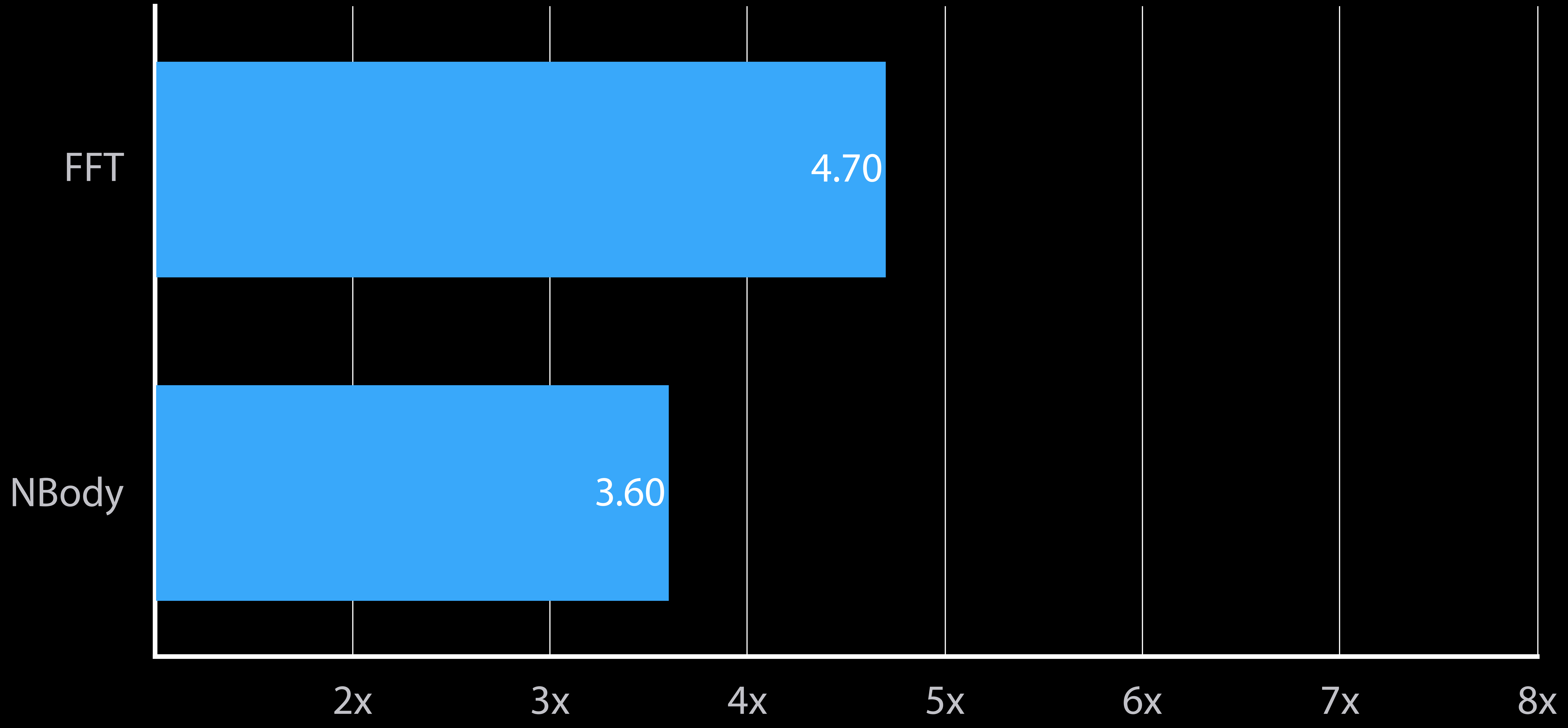
Whole Module Optimization greatly improved in Swift 2.0

- Better optimizations
- Parallel code generation

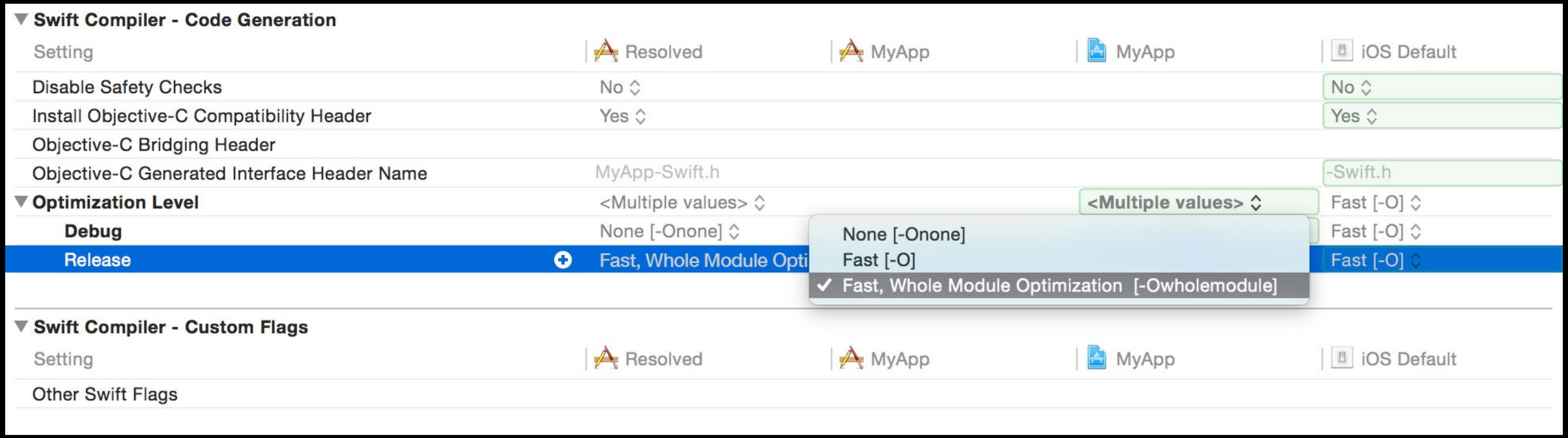


Performance Improvements Due to WMO

Swift 2 vs Swift 2 + WMO (higher is better)



New Optimization Level Configurations



Swift Compiler - Code Generation

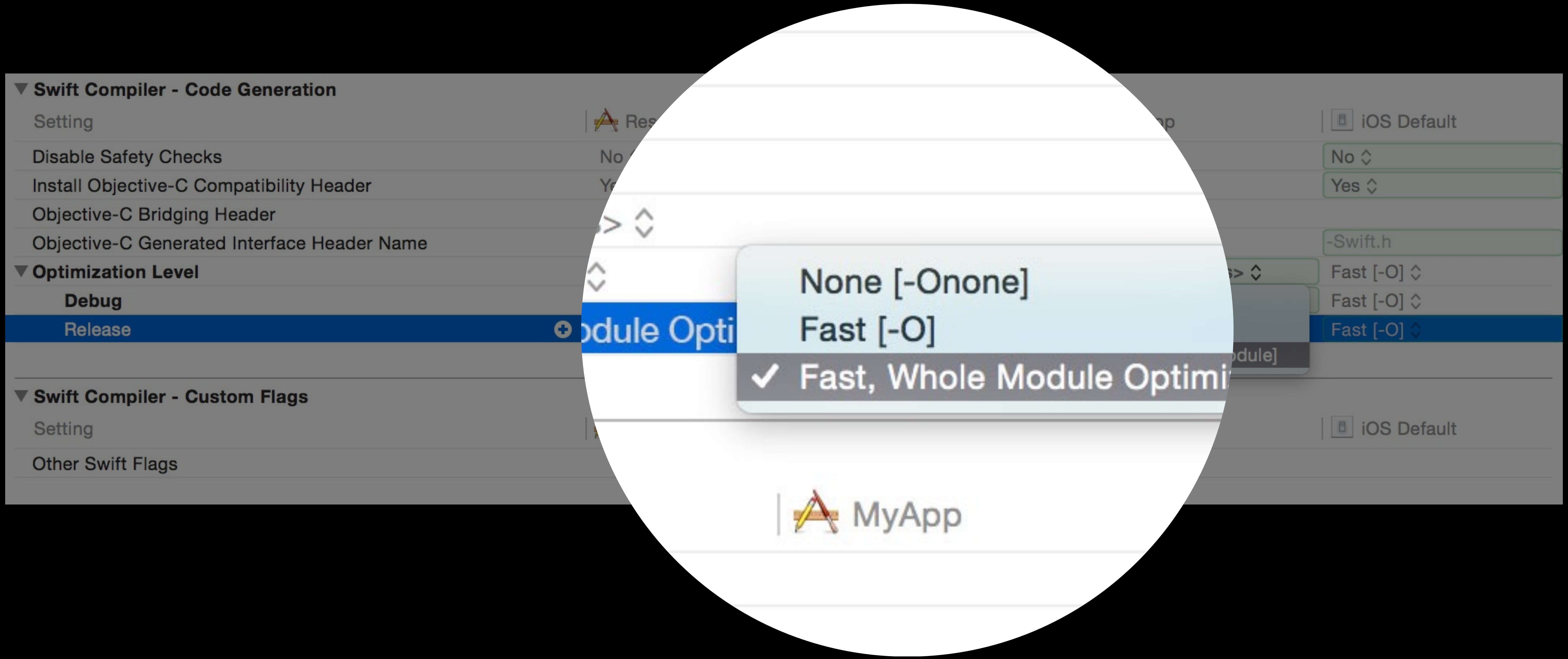
Setting	Resolved	MyApp	MyApp	iOS Default
Disable Safety Checks	No			No
Install Objective-C Compatibility Header	Yes			Yes
Objective-C Bridging Header				
Objective-C Generated Interface Header Name	MyApp-Swift.h			-Swift.h
Optimization Level	<Multiple values>		**<Multiple values>**	Fast [-O]
Debug	None [-Onone]			Fast [-O]
Release	Fast, Whole Module Opti			Fast [-O]

None [-Onone]
Fast [-O]
✓ Fast, Whole Module Optimization [-Owholemodule]

Swift Compiler - Custom Flags

Setting	Resolved	MyApp	MyApp	iOS Default
Other Swift Flags				

New Optimization Level Configurations



▼ Swift Compiler - Code Generation

Setting	Res		iOS Default
Disable Safety Checks	No		No ◇
Install Objective-C Compatibility Header	Y		Yes ◇
Objective-C Bridging Header	> ◇		
Objective-C Generated Interface Header Name			-Swift.h
▼ Optimization Level	◇	> ◇	Fast [-O] ◇
Debug			Fast [-O] ◇
Release	odule Opti		Fast [-O] ◇

None [-Onone]
Fast [-O]
✓ Fast, Whole Module Optimi

odule]

▼ Swift Compiler - Custom Flags

Setting		iOS Default
Other Swift Flags		

MyApp

Writing High Performance Swift Code

Michael Gottesman Engineer, Swift Performance Team

Overview

Reference Counting

Generics

Dynamic Dispatch

Overview

Reference Counting

Generics

Dynamic Dispatch

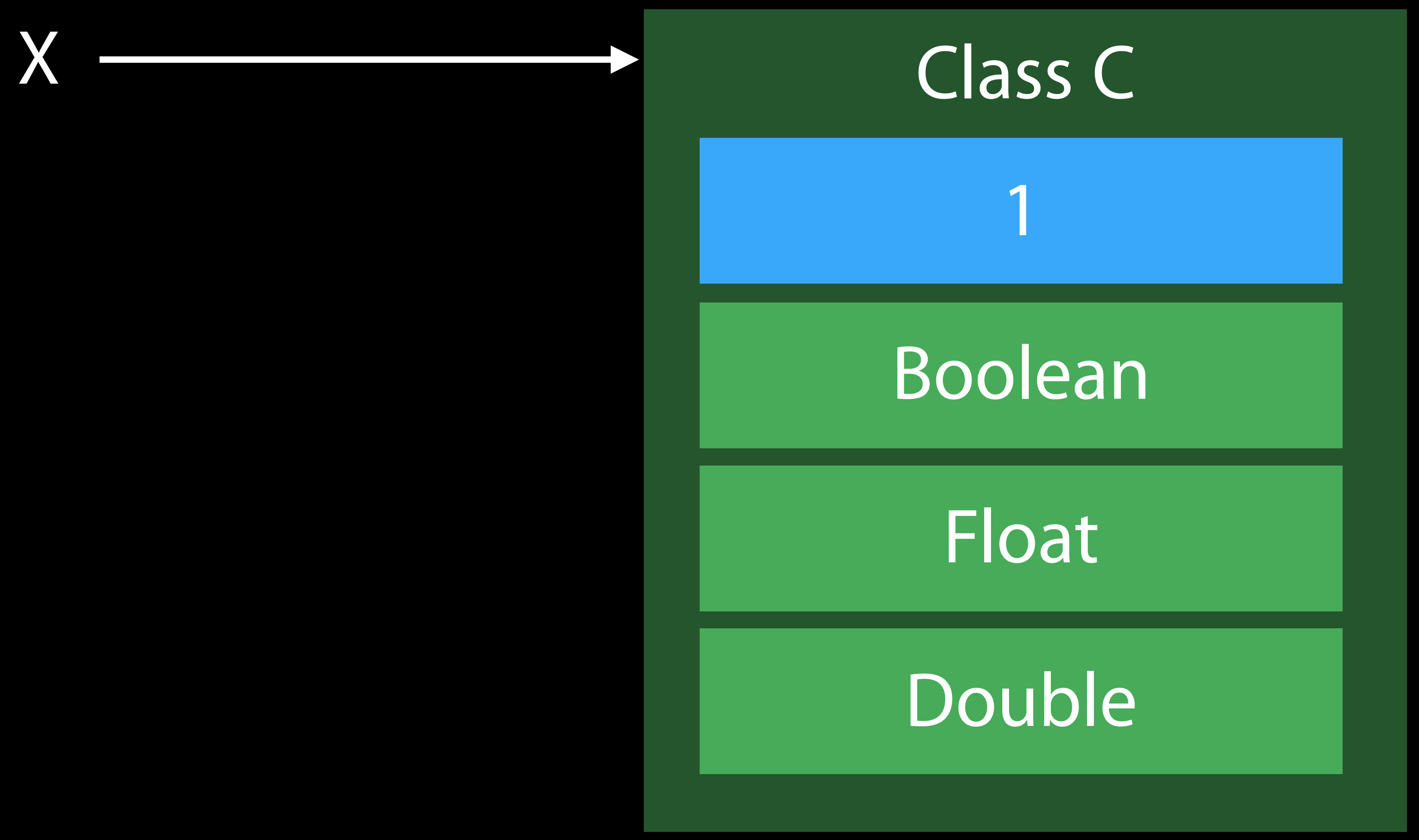
How Reference Counting Works

How Reference Counting Works

```
class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
```

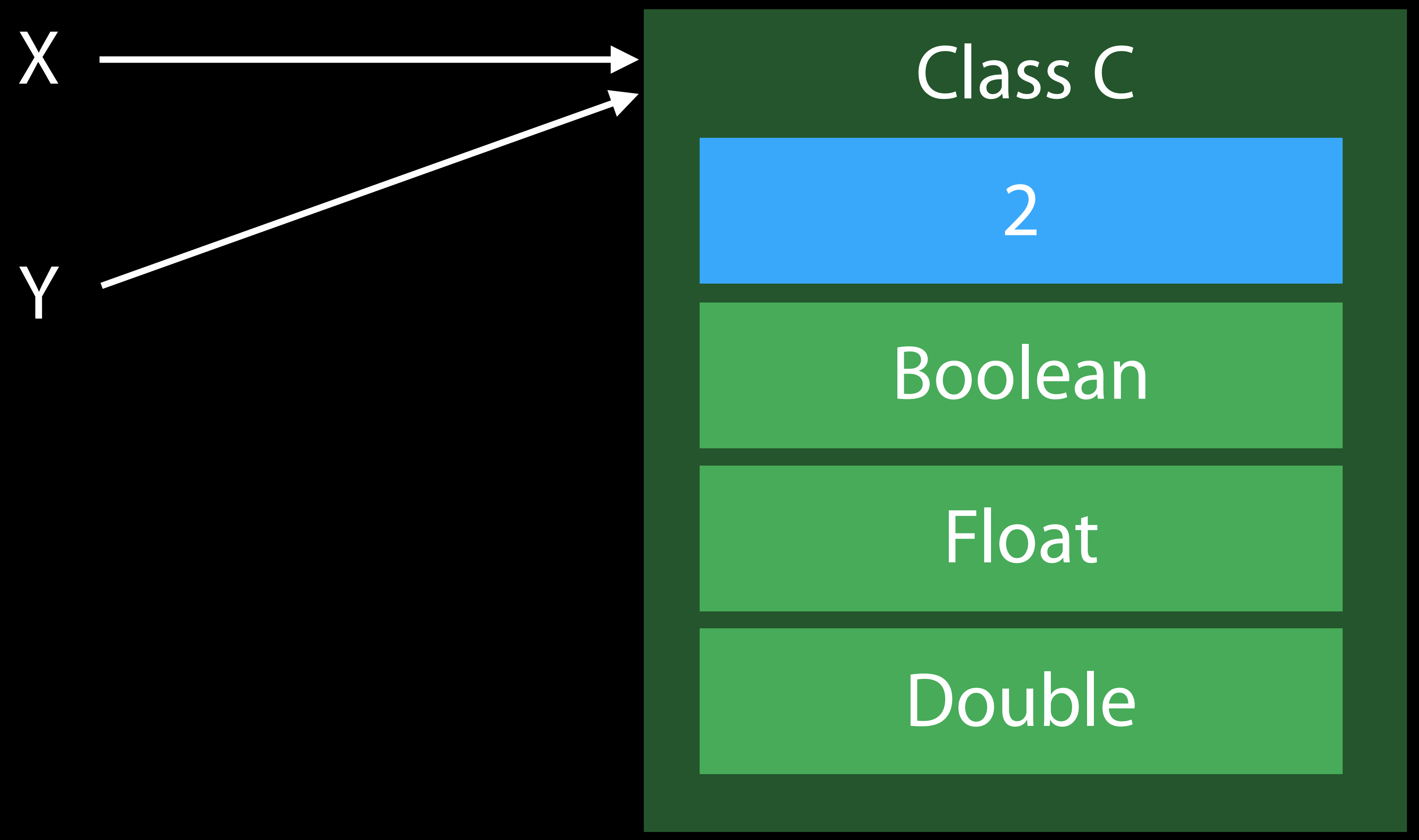
How Reference Counting Works

```
class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
```



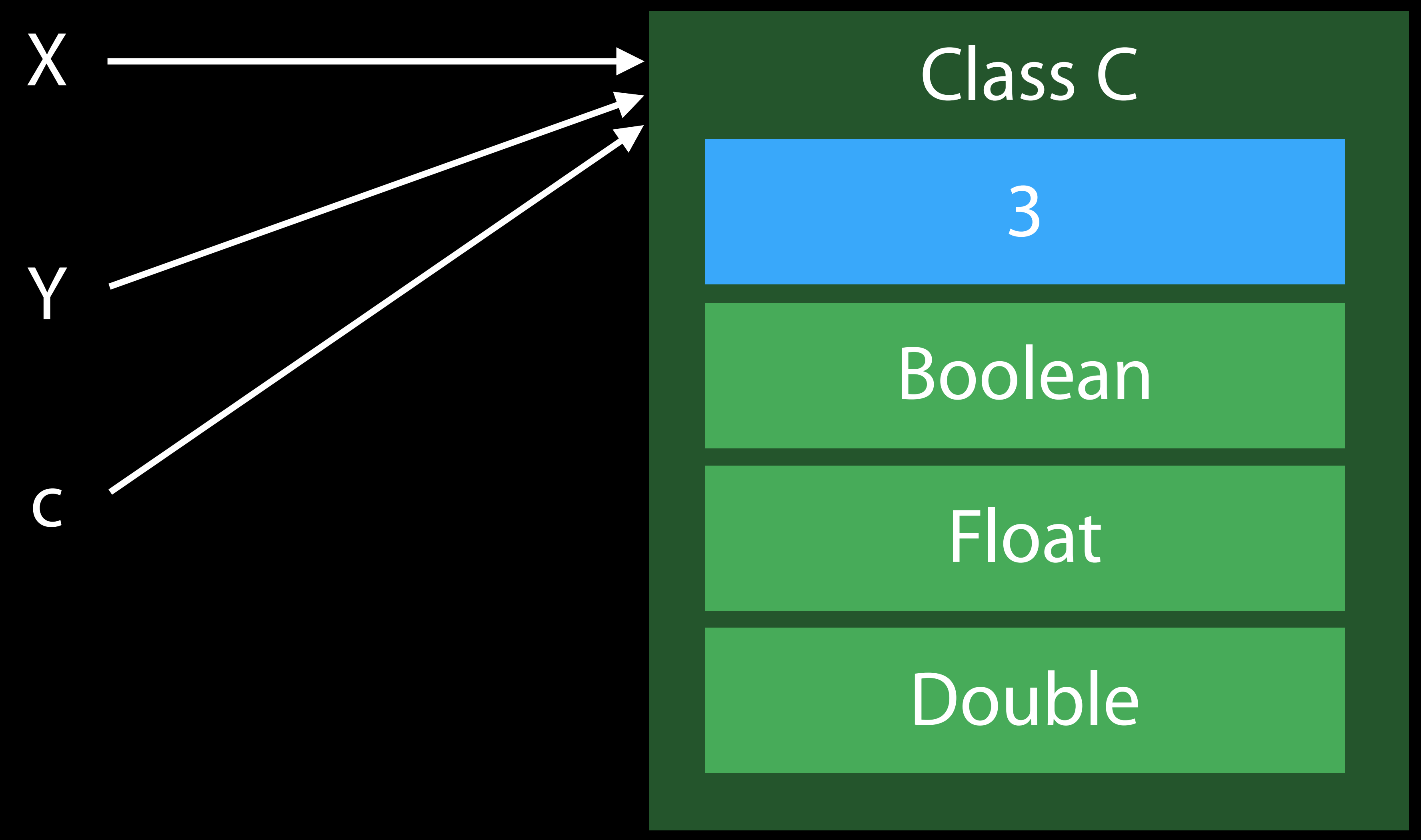
How Reference Counting Works

```
class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
```



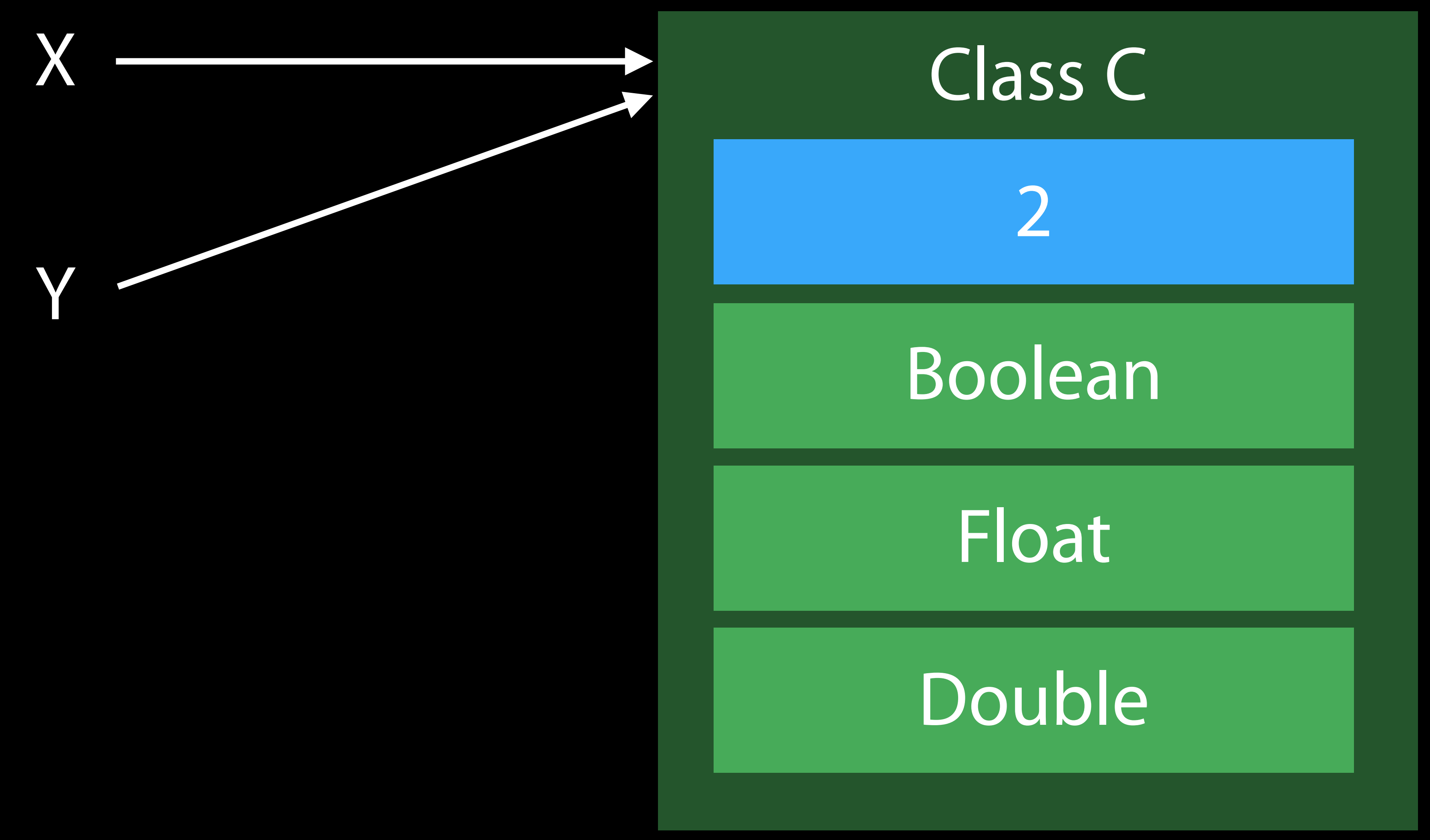
How Reference Counting Works

```
class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
```



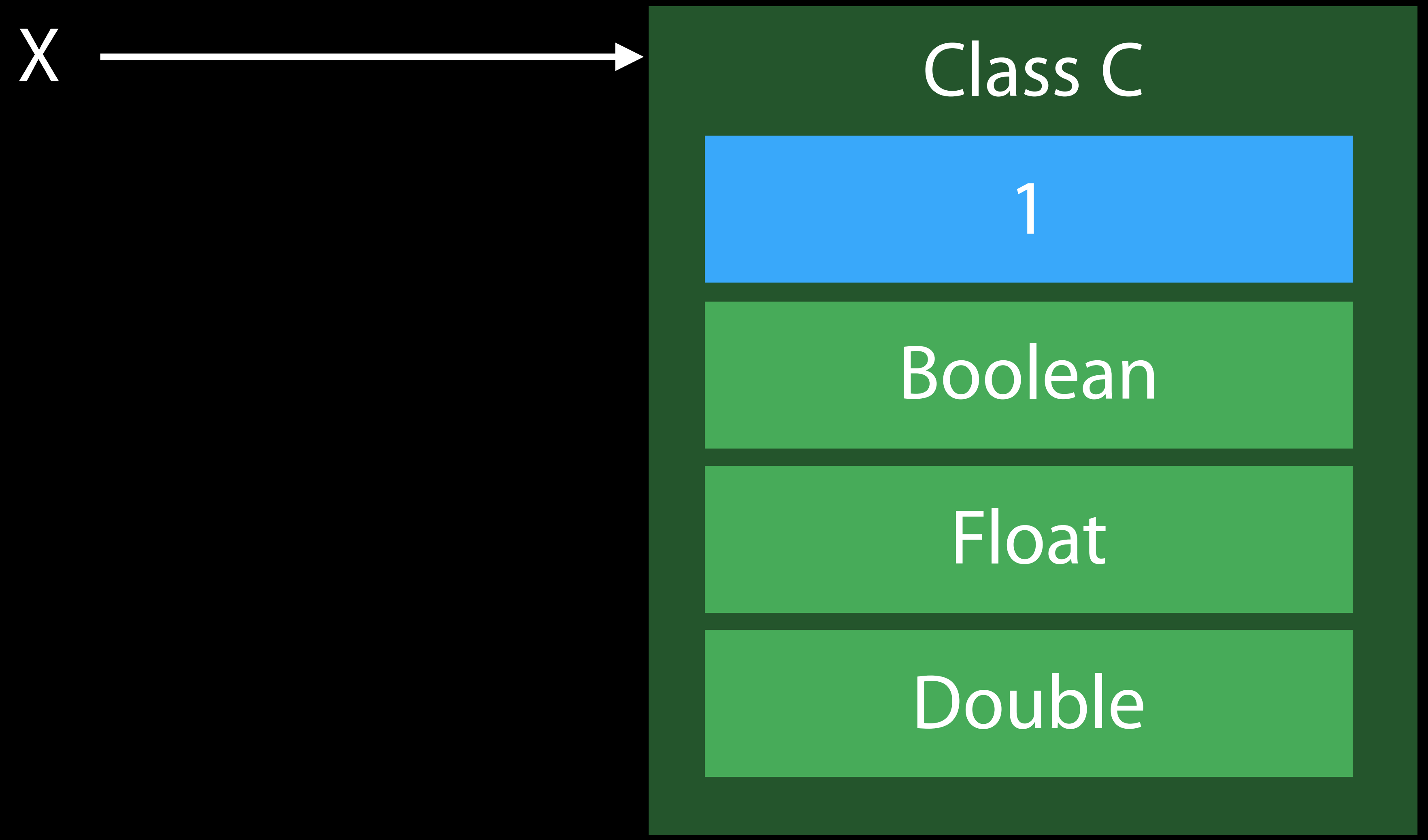
How Reference Counting Works

```
class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
```



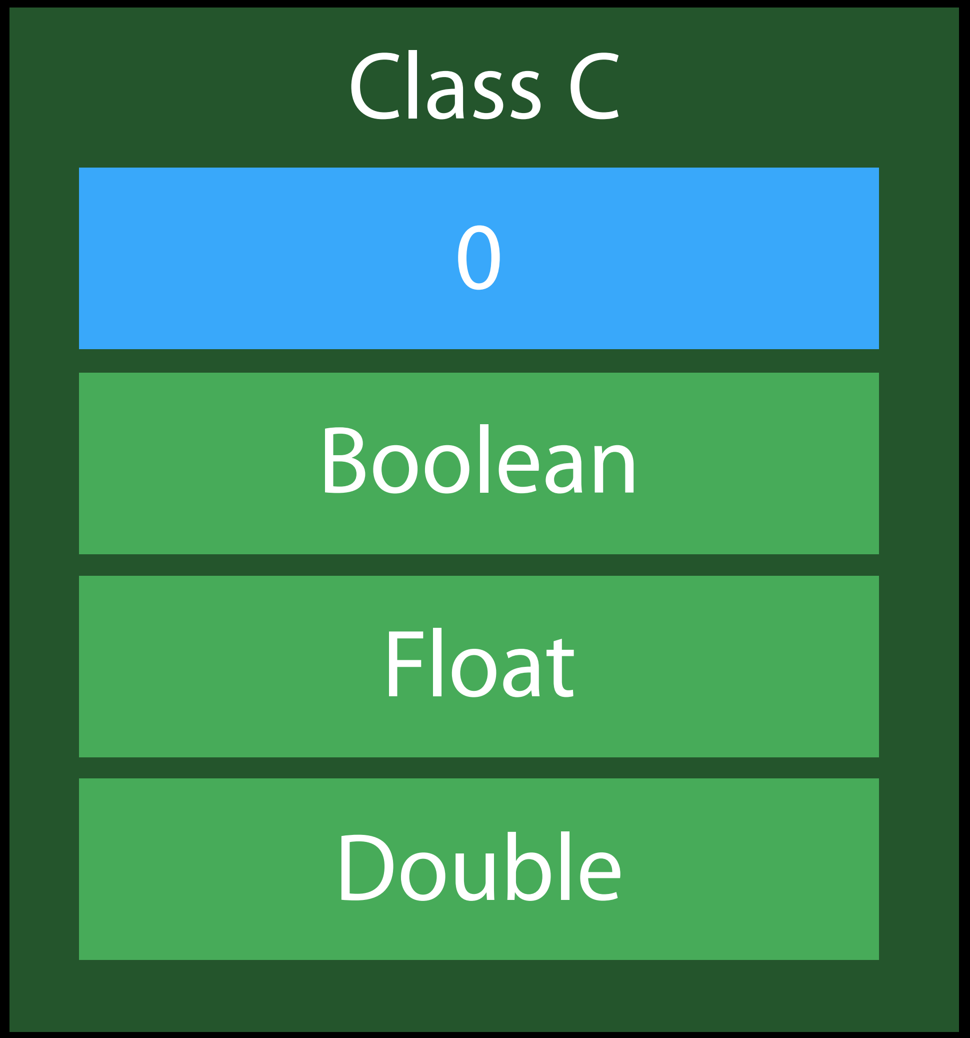
How Reference Counting Works

```
class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
```



How Reference Counting Works

```
class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
```



How Reference Counting Works

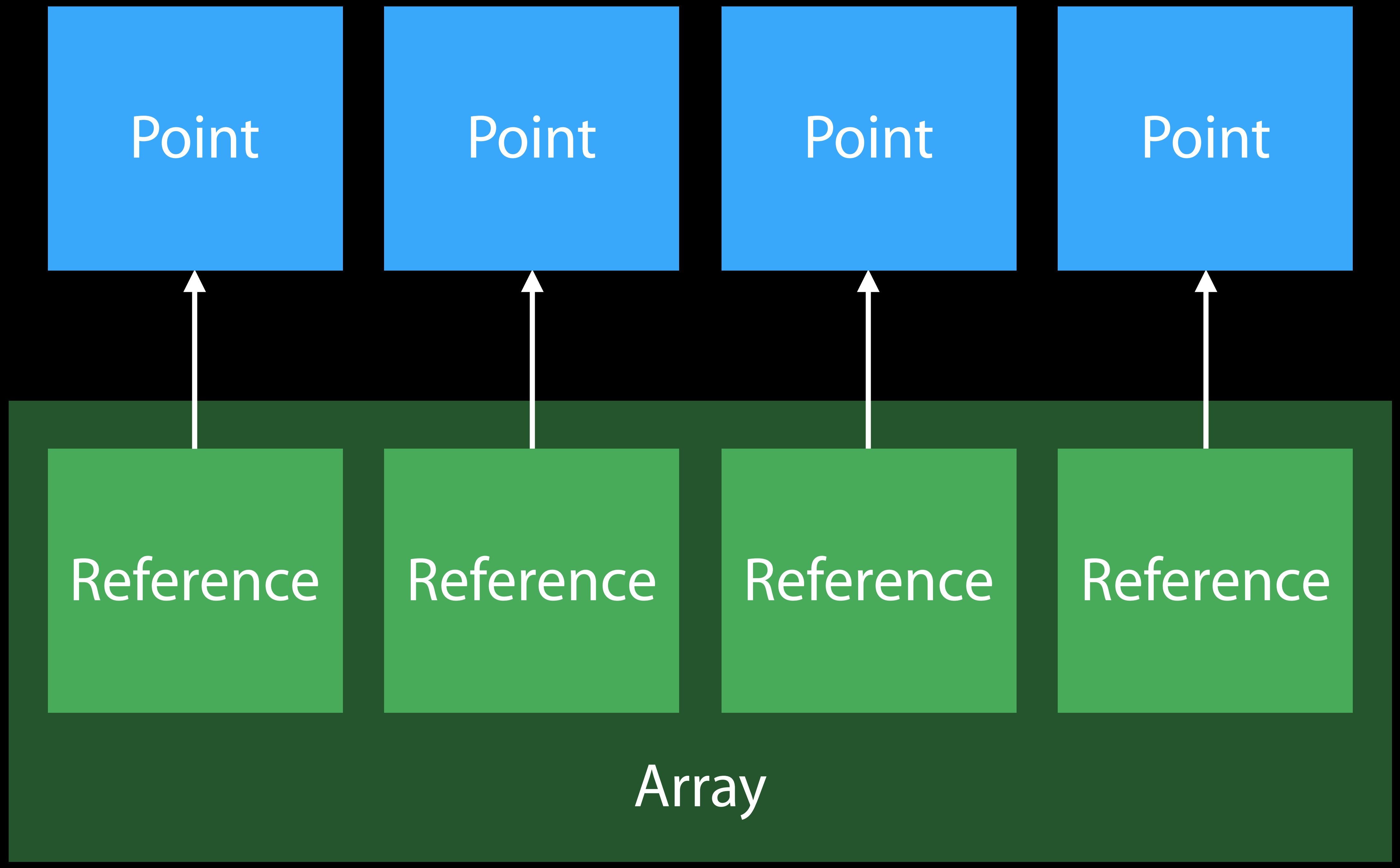
```
class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
```

Classes That Do Not Contain References

```
class Point {
    var x, y: Float
}
```

Classes That Do Not Contain References

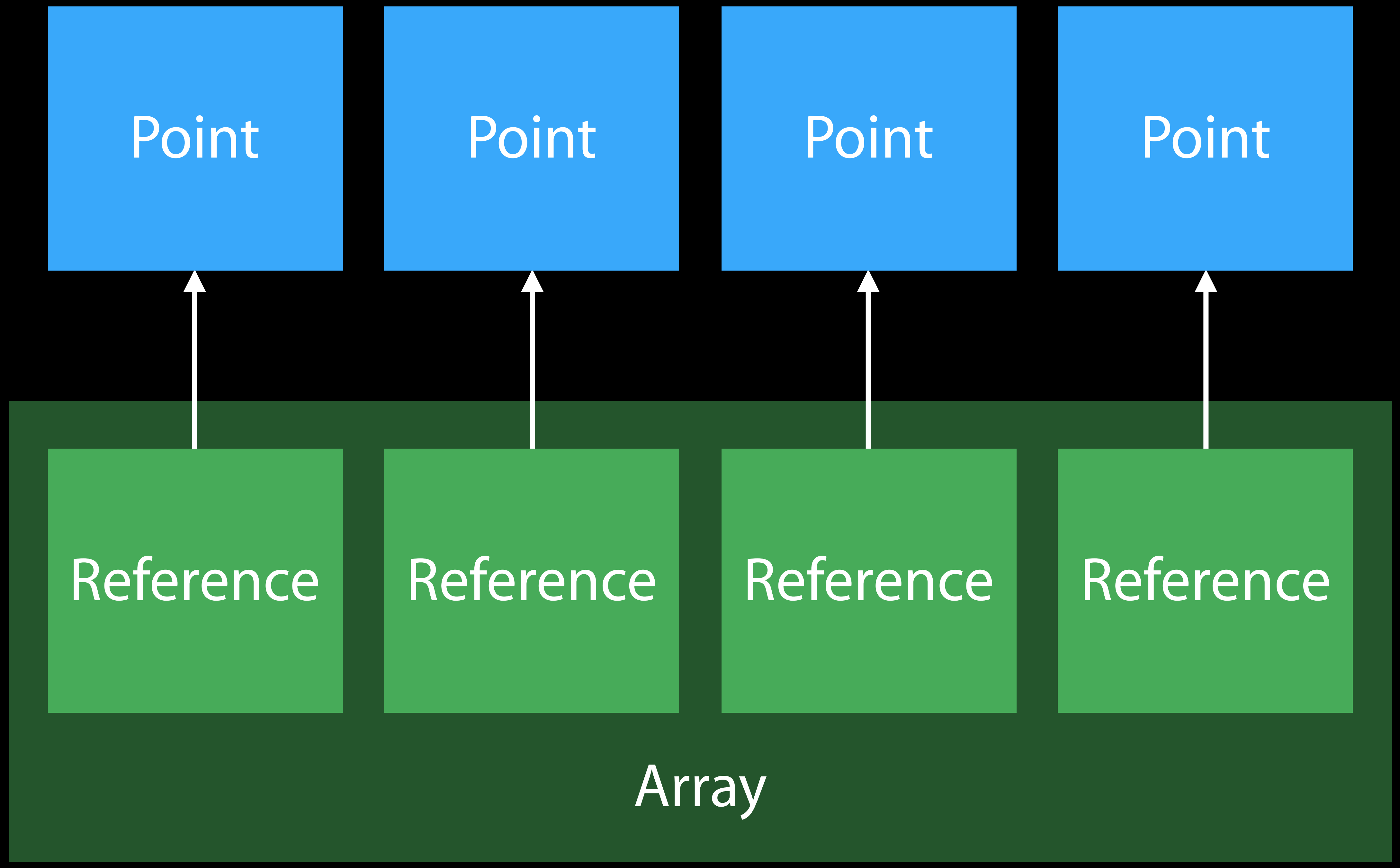
```
class Point {
    var x, y: Float
}
```



Classes That Do Not Contain References

```
class Point {
    var x, y: Float
}
```

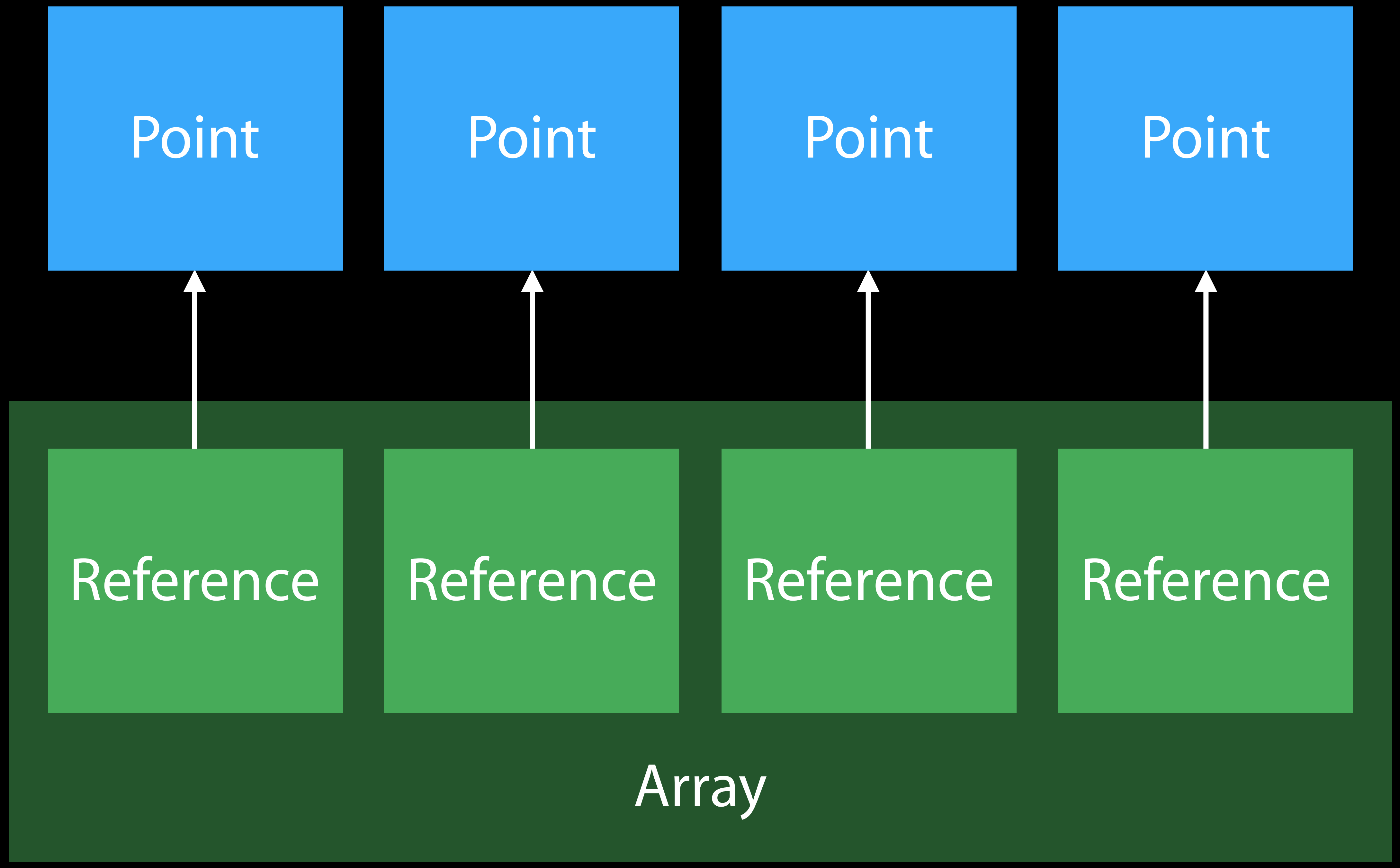
```
var array: [Point] = ...
for p in array {
    ...
}
```



Classes That Do Not Contain References

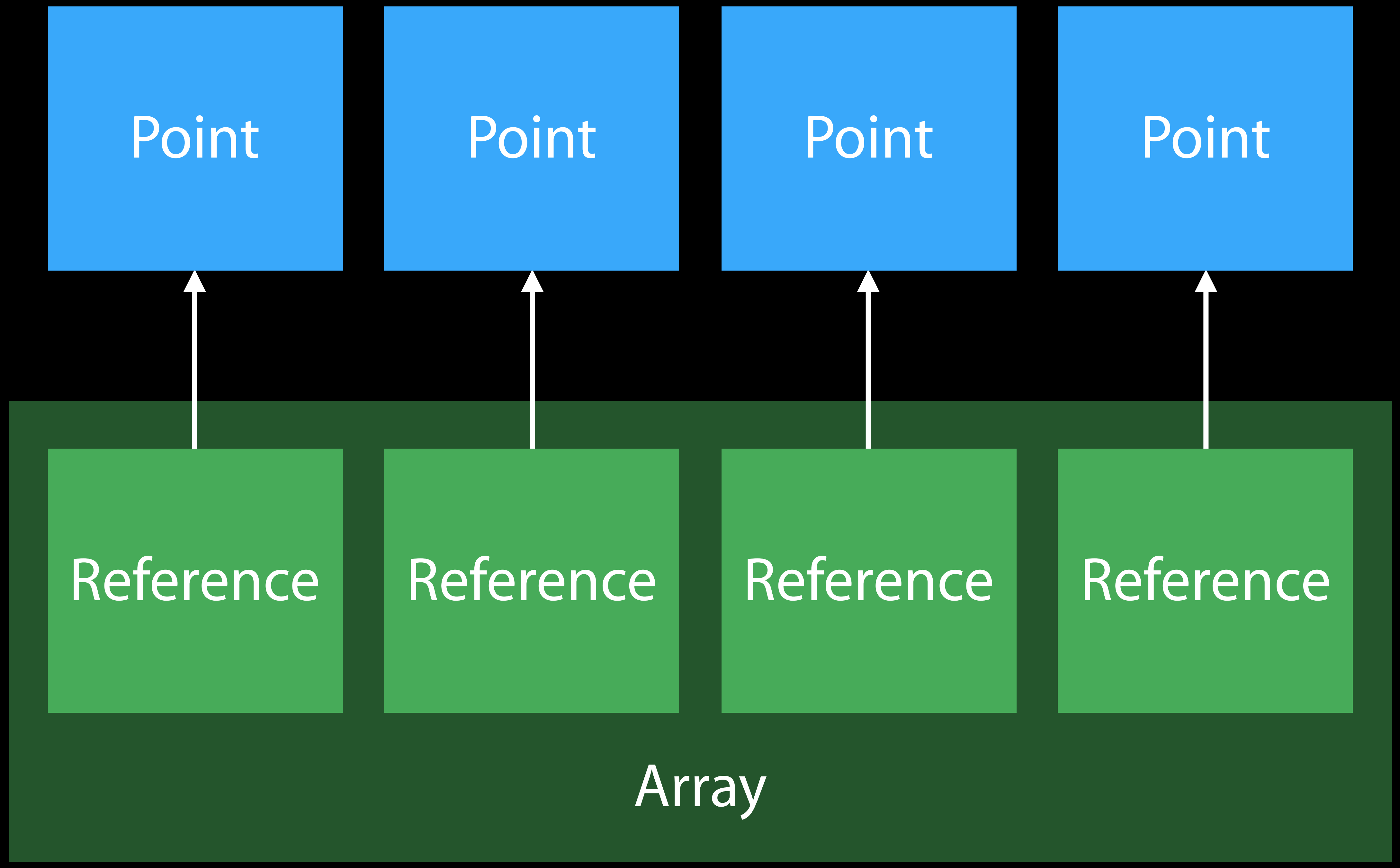
```
class Point {
    var x, y: Float
}
```

```
var array: [Point] = ...
for p in array {
   increment
   ...
}
```



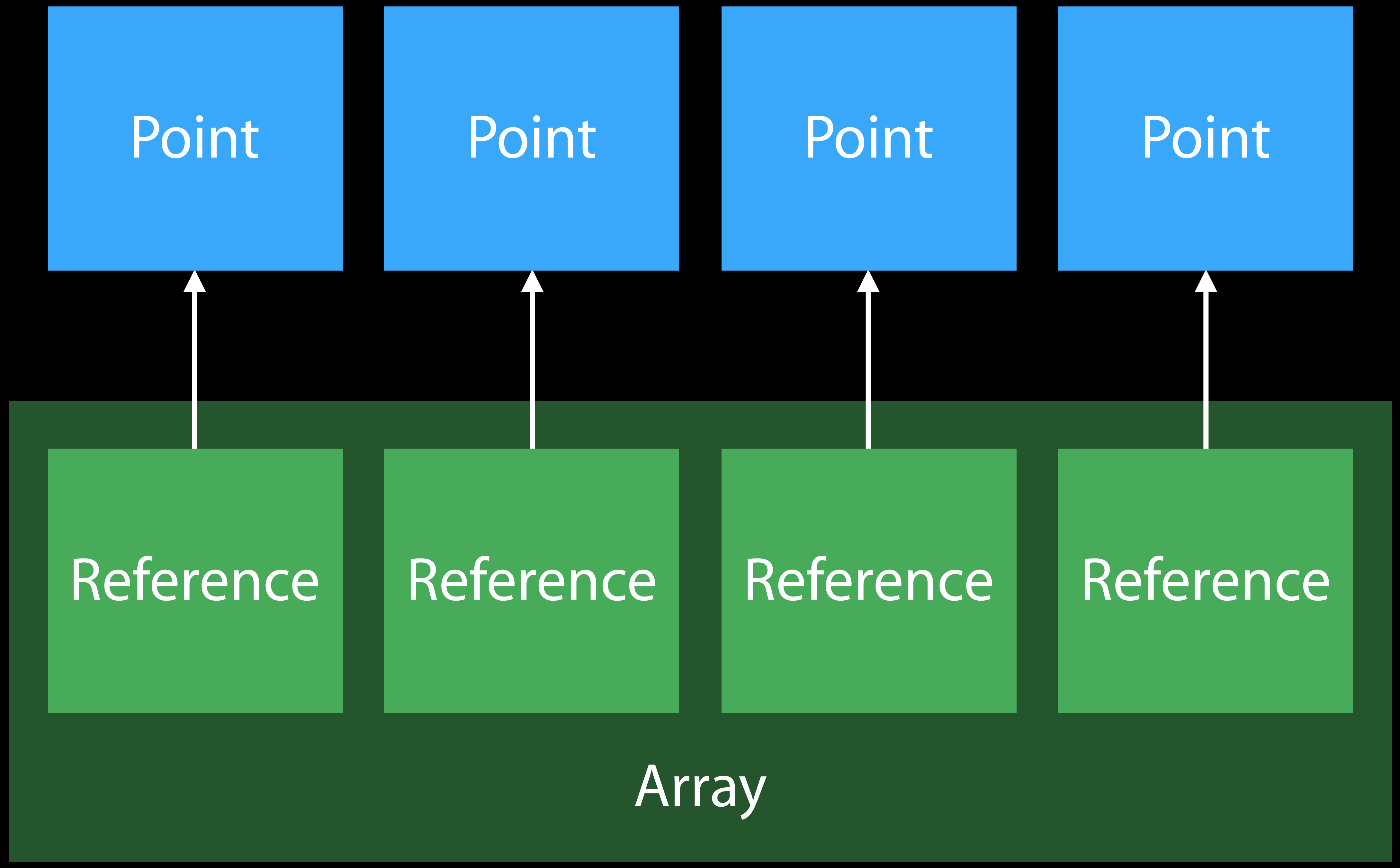
Classes That Do Not Contain References

```
class Point {
    var x, y: Float
}


var array: [Point] = ...
for p in array {
   increment
   ...
   decrement
}
```



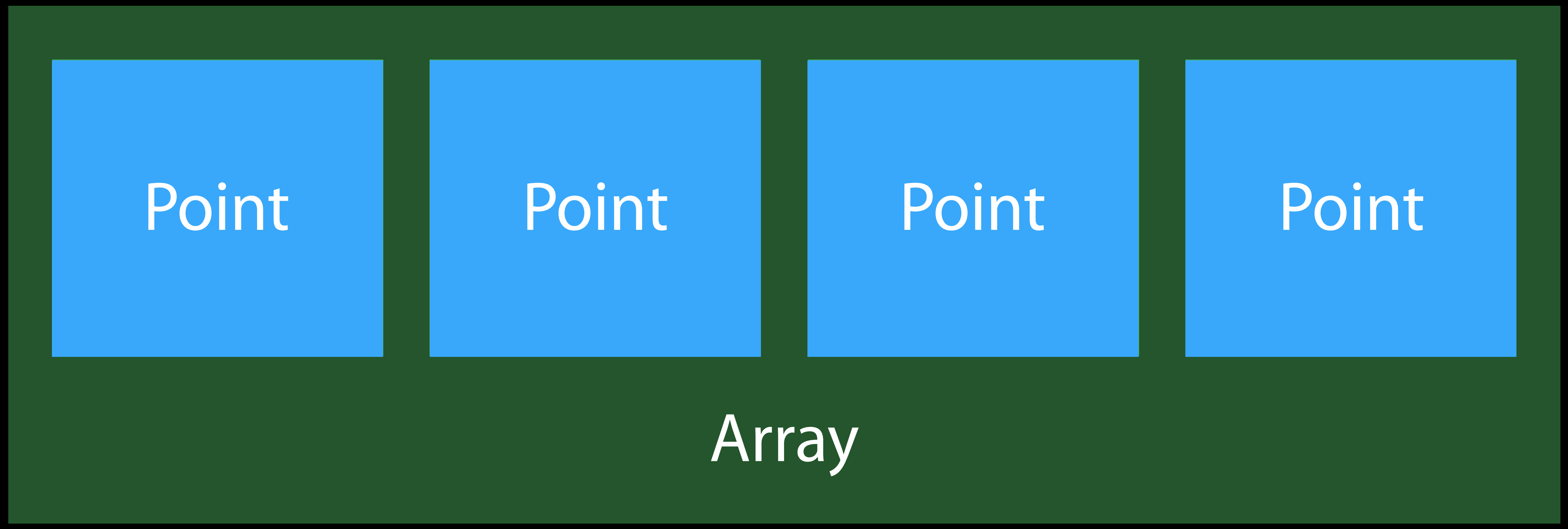
Structs That Do Not Contain References

```
struct Point {
    var x, y: Float
}


var array: [Point] = ...
for p in array {
   increment
   ...
   decrement
}
```



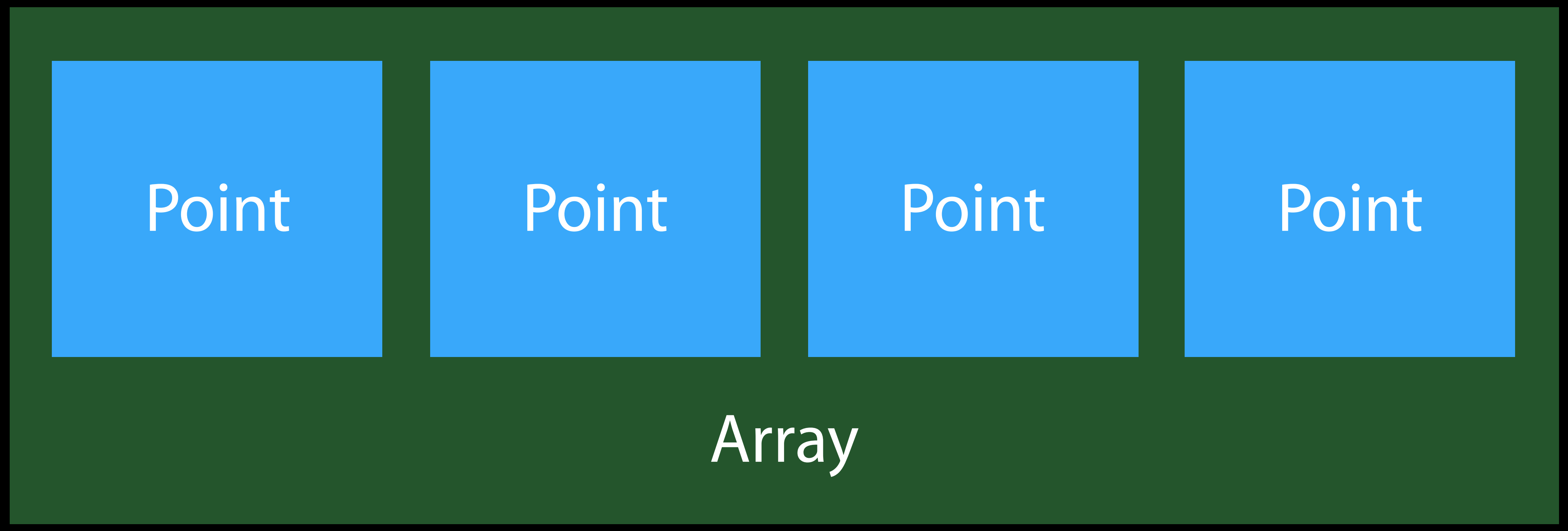
Structs That Do Not Contain References

```
struct Point {
    var x, y: Float
}

var array: [Point] = ...
for p in array {
   increment
   ...
   decrement
}
```



Structs That Do Not Contain References

```
struct Point {
    var x, y: Float
}

var array: [Point] = ...
for p in array {
   ...
}
```



All reference counting operations eliminated

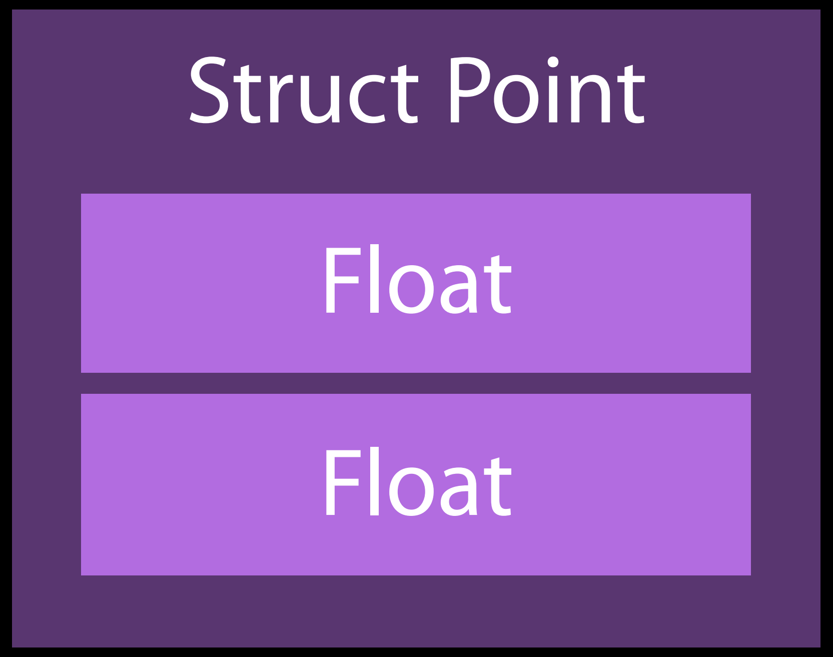
Structs Containing a Reference

Structs Containing a Reference

A Struct requires reference counting if its properties require reference counting

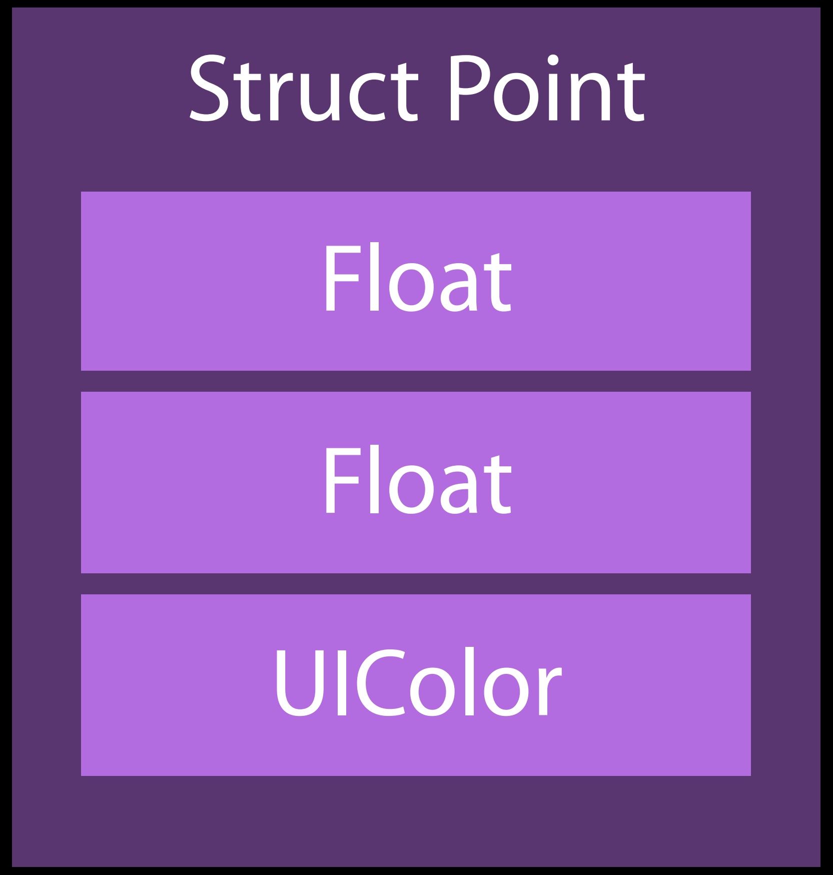
Structs Containing a Reference

A Struct requires reference counting if its properties require reference counting



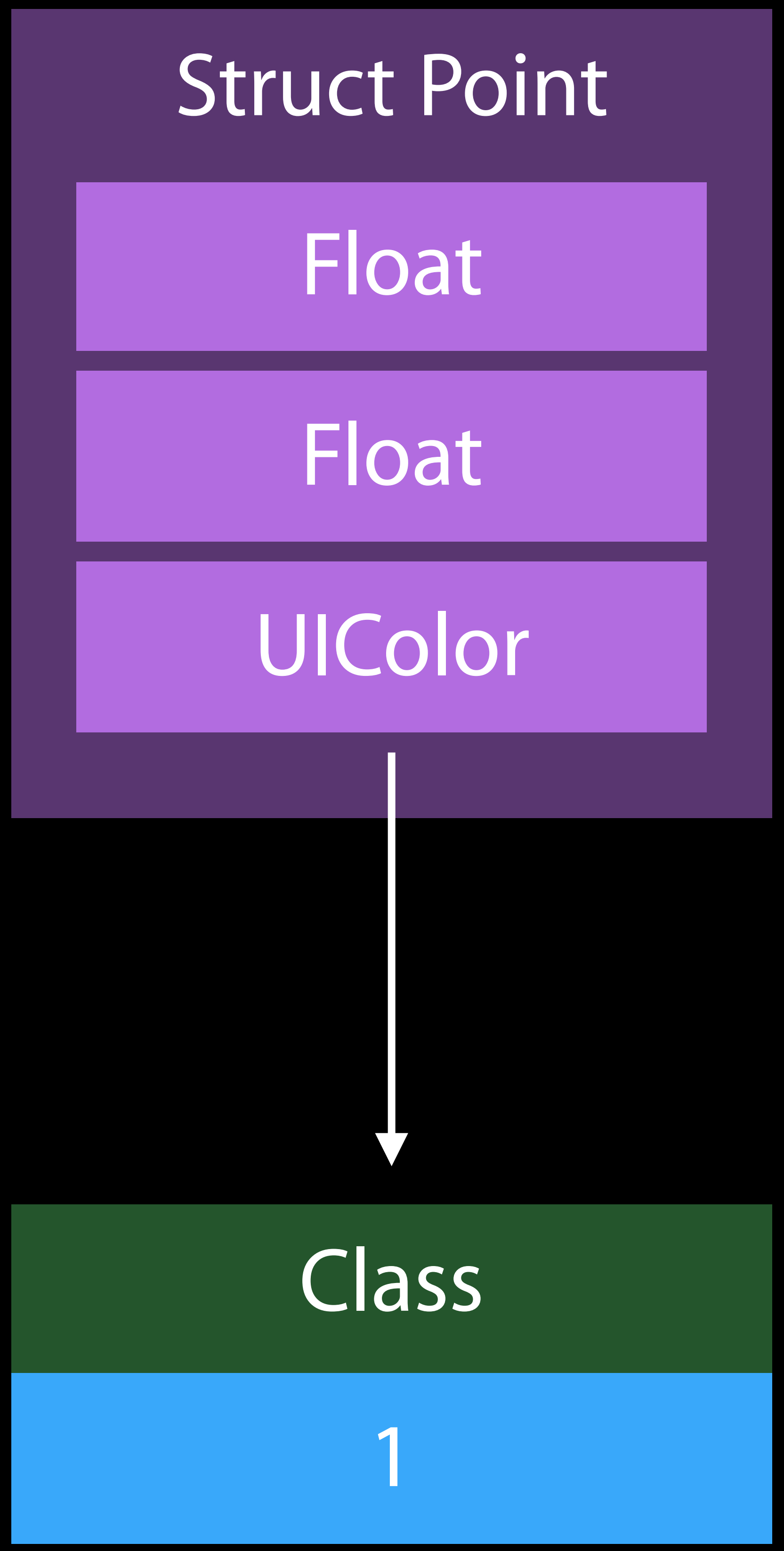
Structs Containing a Reference

A Struct requires reference counting if its properties require reference counting



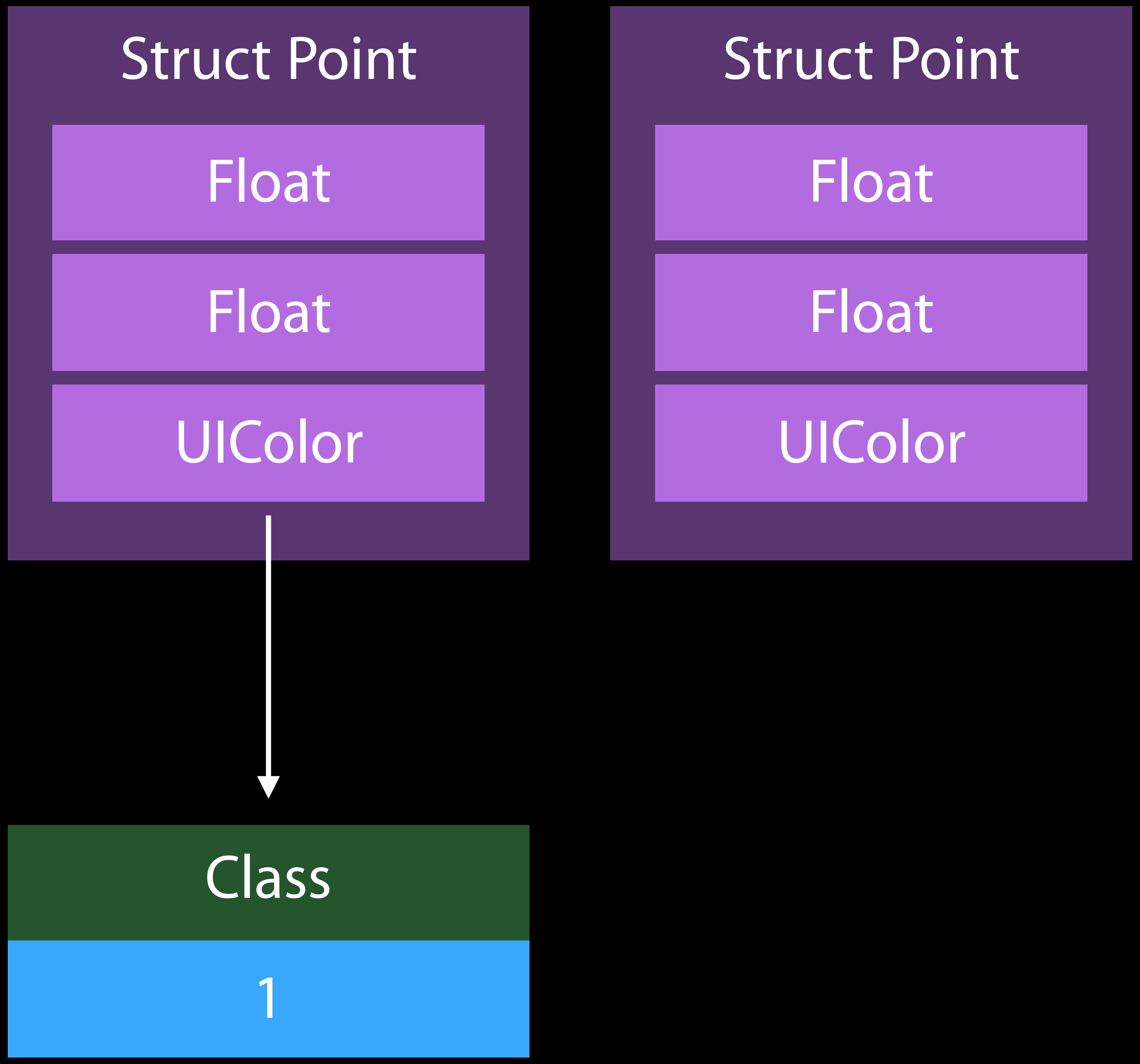
Structs Containing a Reference

A Struct requires reference counting if its properties require reference counting



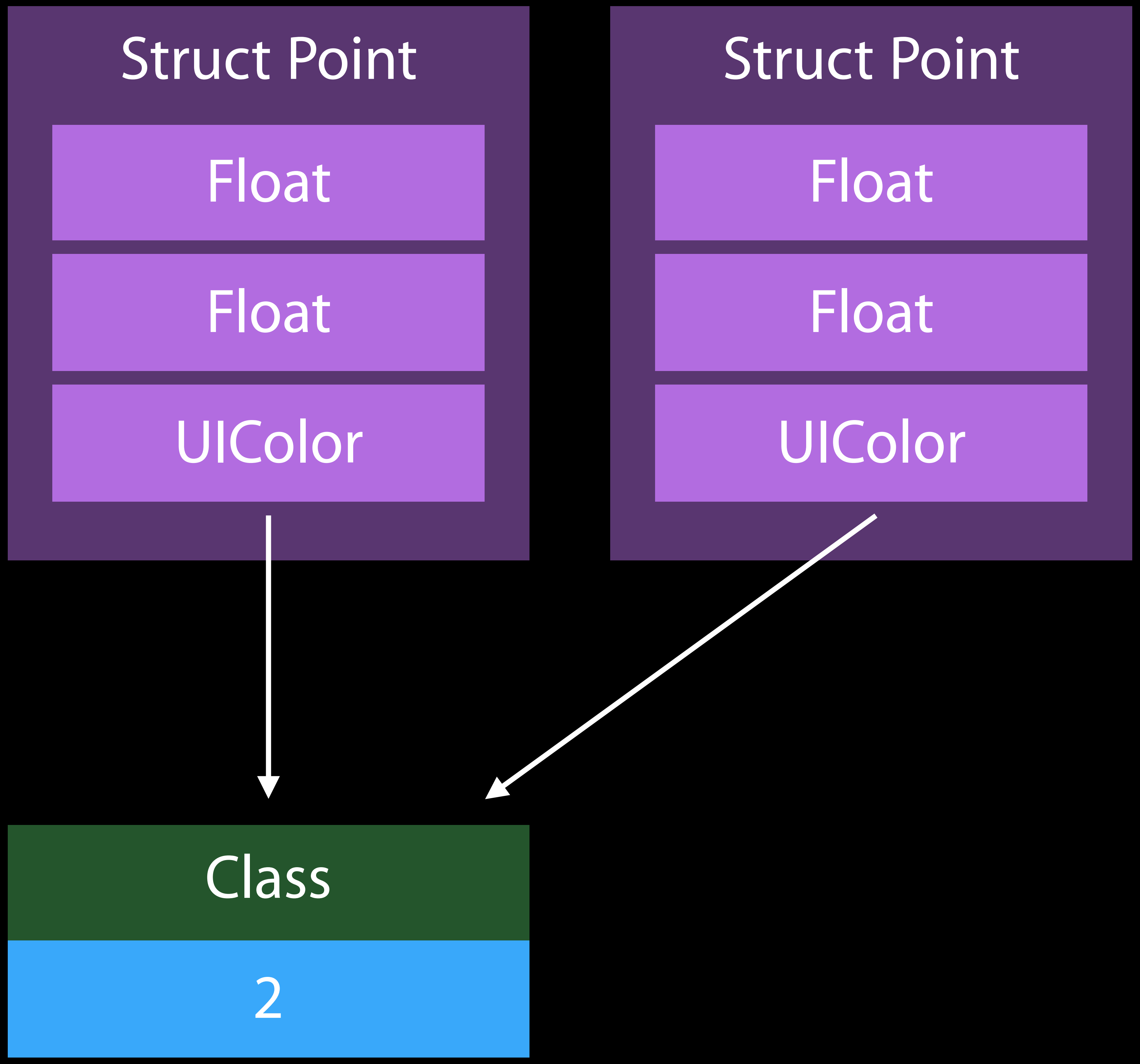
Structs Containing a Reference

A Struct requires reference counting if its properties require reference counting

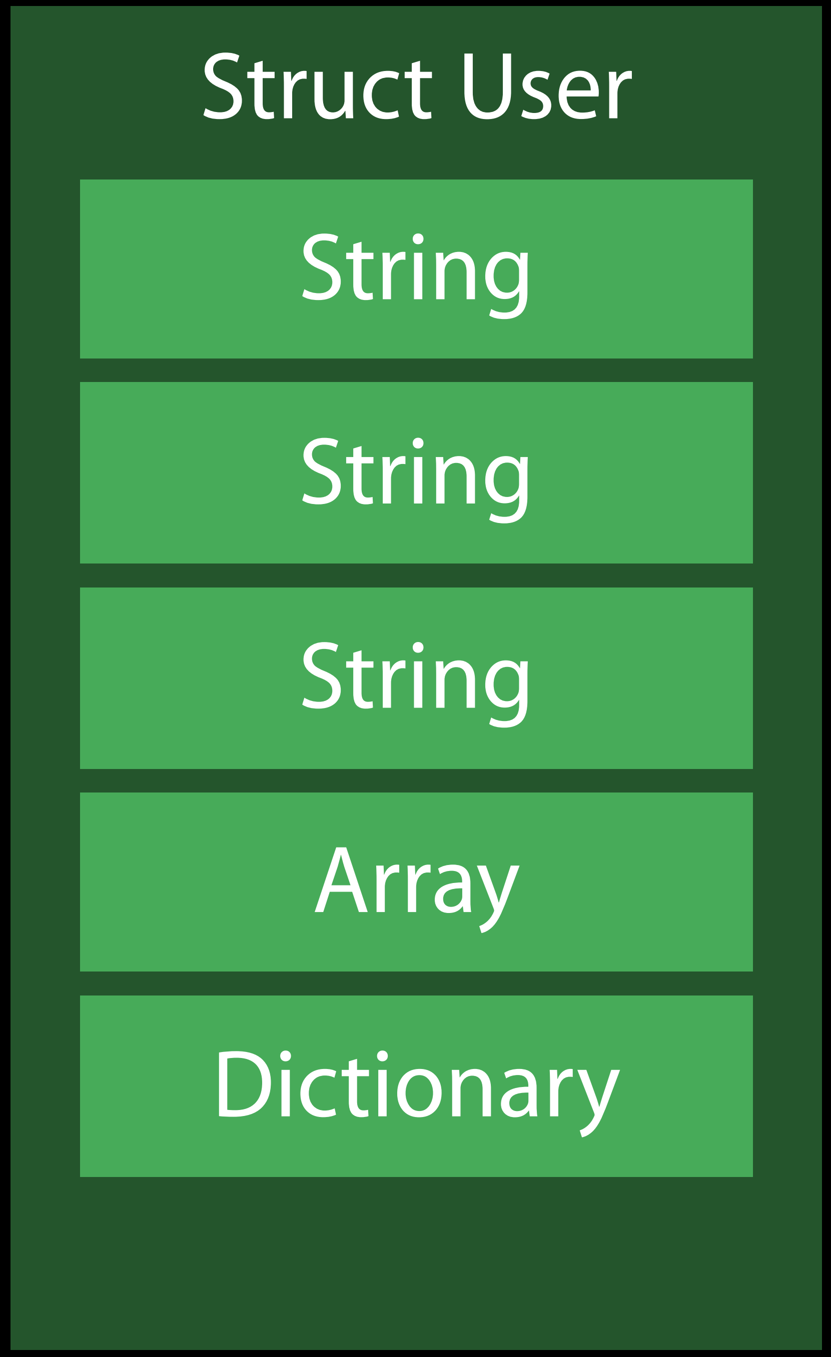


Structs Containing a Reference

A Struct requires reference counting if its properties require reference counting



Structs Containing Many References



Struct User

String

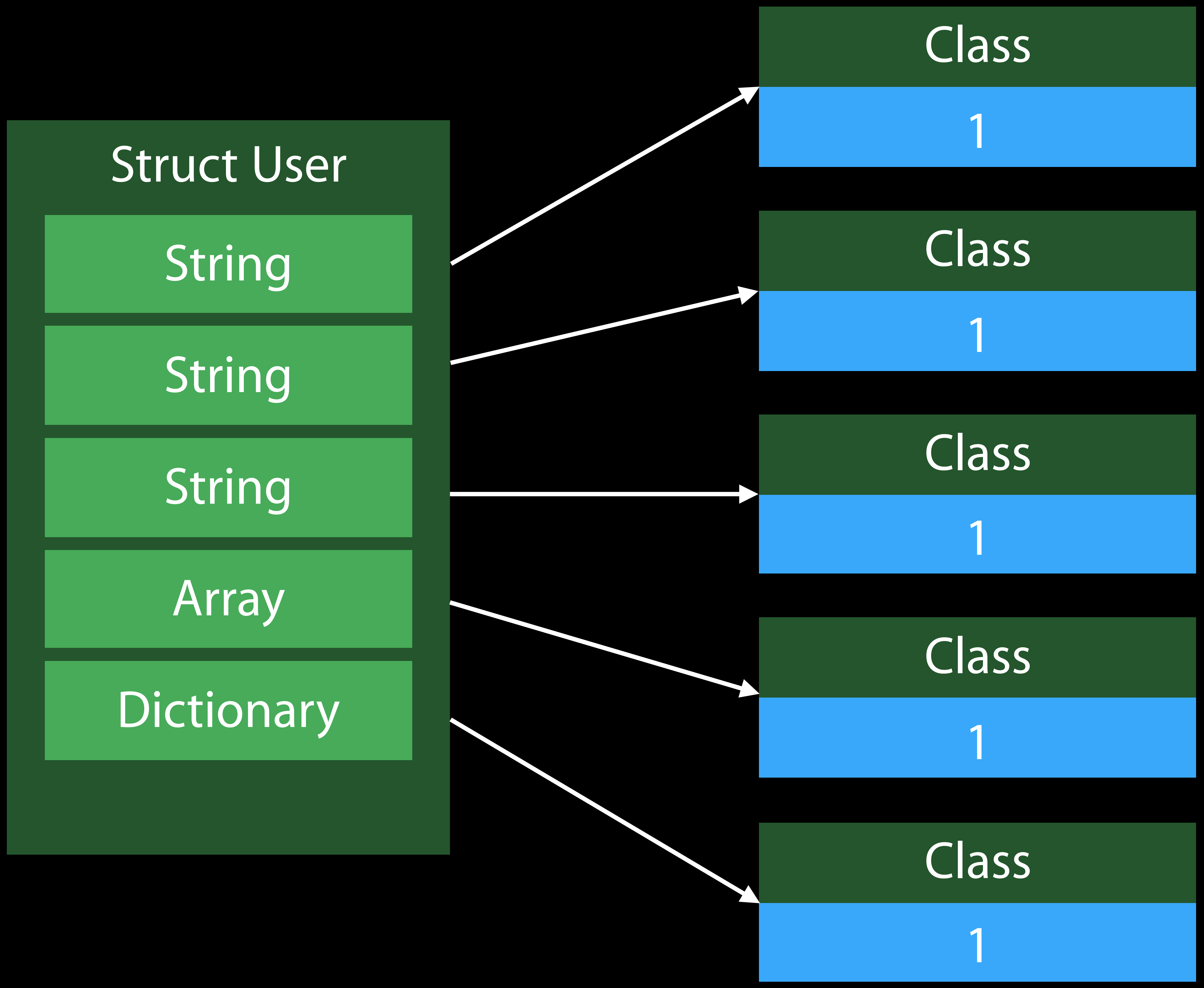
String

String

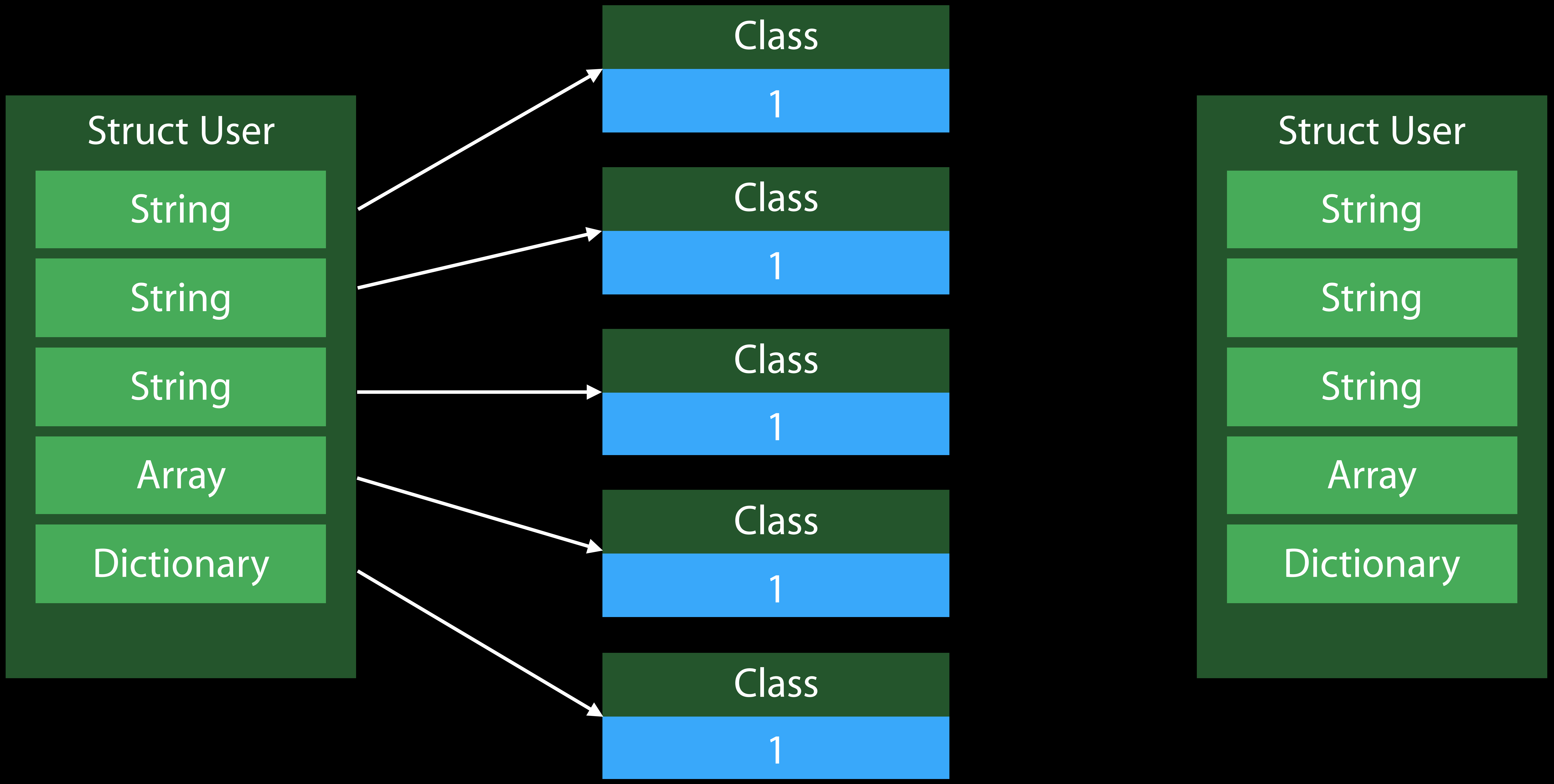
Array

Dictionary

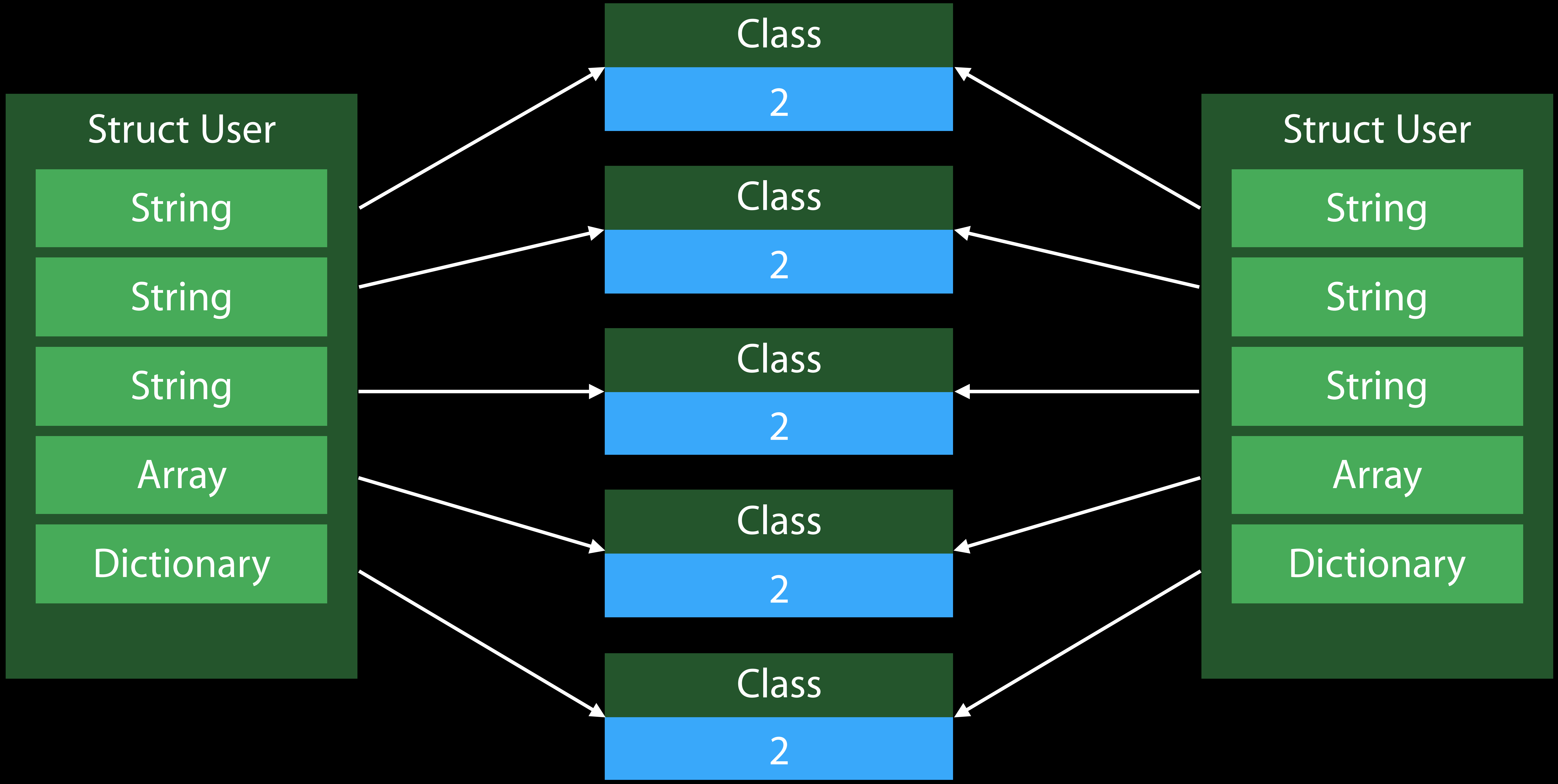
Structs Containing Many References



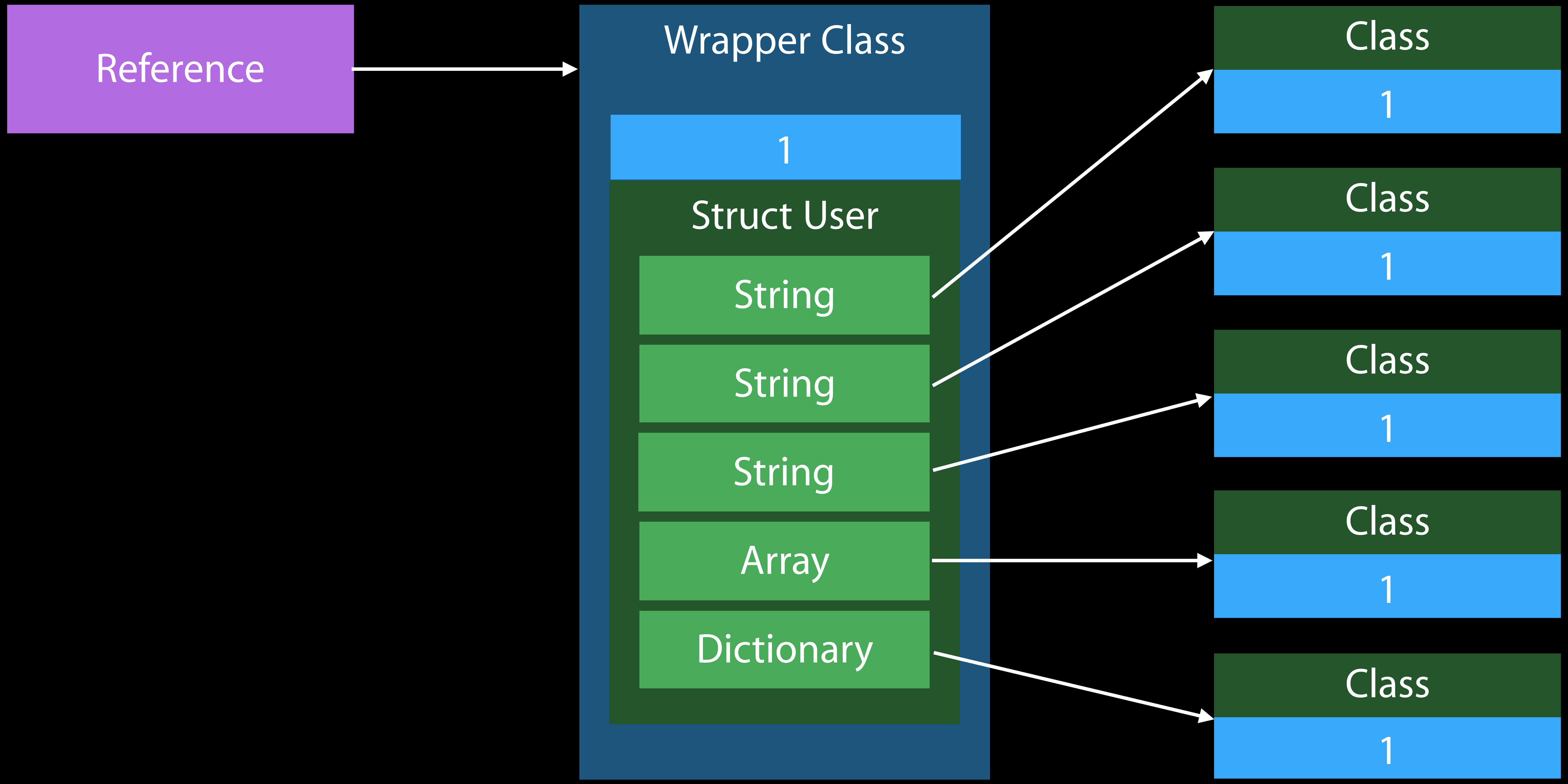
Structs Containing Many References



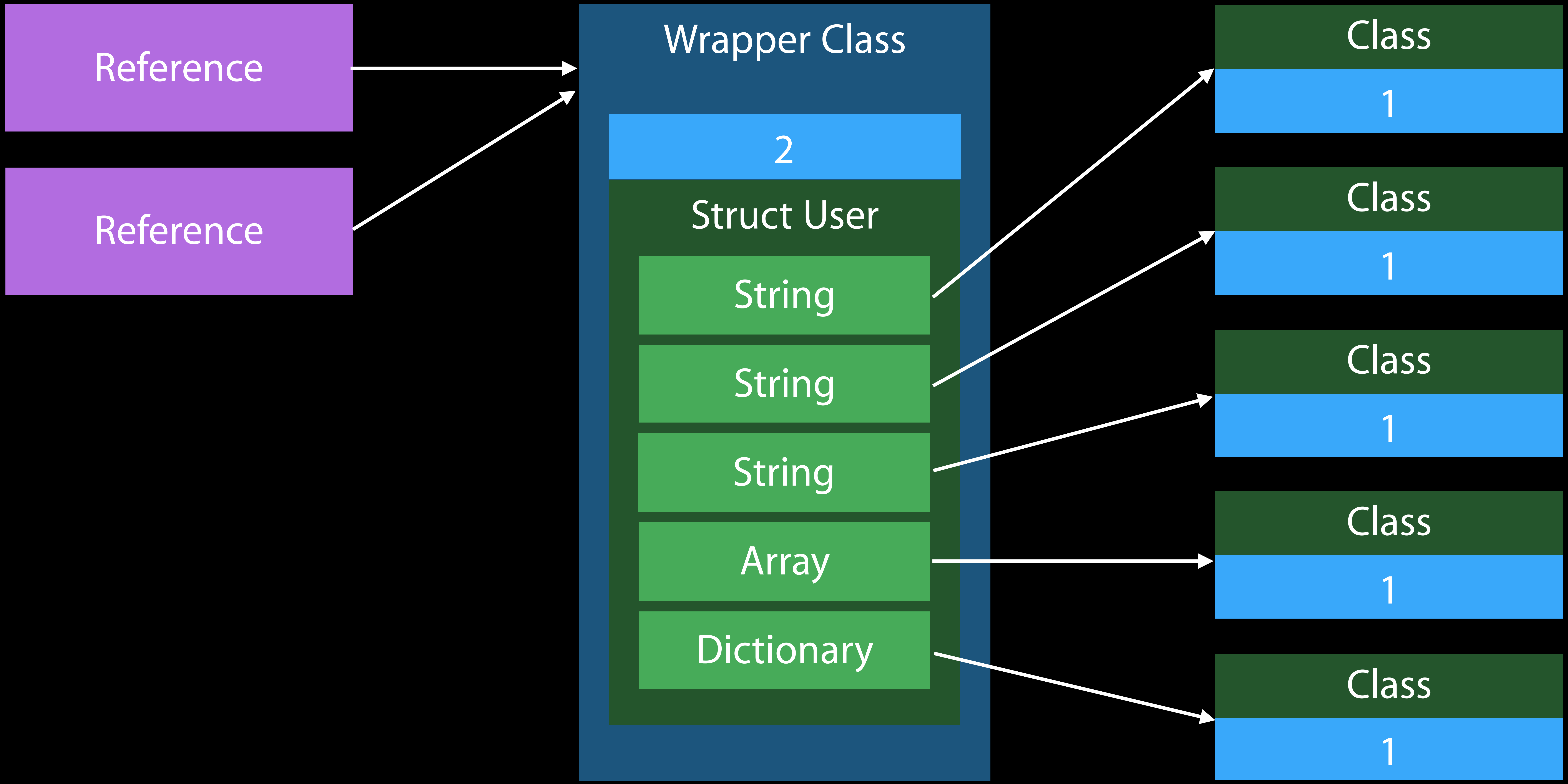
Structs Containing Many References



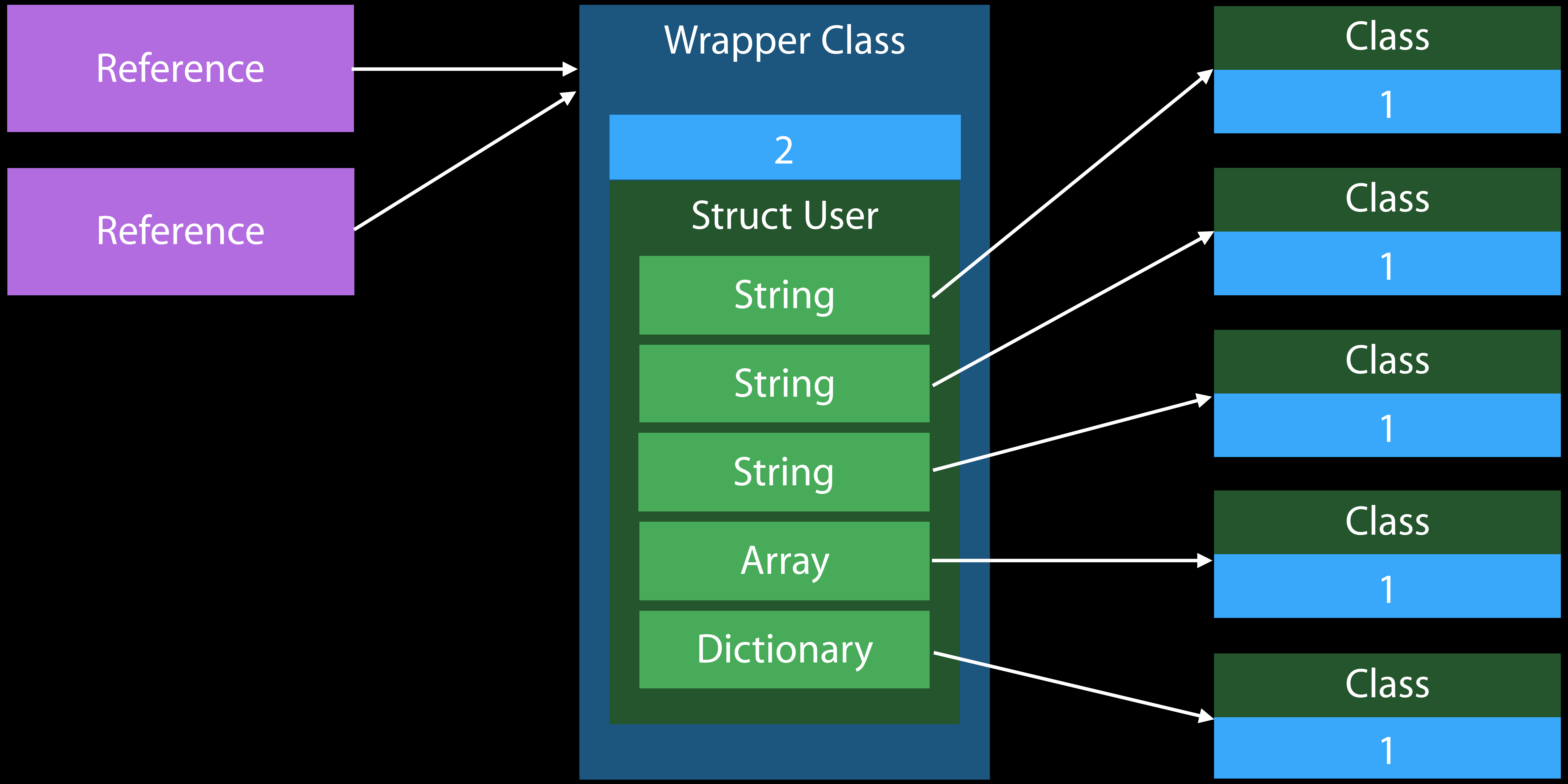
Use a Wrapper Class



Use a Wrapper Class



Use a Wrapper Class



Building Better Apps with Value Types in Swift

Mission

Friday 2:30PM

Overview

Reference Counting

Generics

Dynamic Dispatch

Overview

Reference Counting

Generics

Dynamic Dispatch

How Generics Work

```
func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

How Generics Work

```swift
func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

```swift
func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

How Generics Work

```swift
func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

```swift
func min<T : Comparable>(x: T, y: T, FTable: FunctionTable) -> T {
    let xCopy = FTable.copy(x)
    let yCopy = FTable.copy(y)
    let m = FTable.lessThan(yCopy, xCopy) ? y : x
    FTable.release(x)
    FTable.release(y)
    return m
}
```

How Generics Work

```
func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

```
func min<T : Comparable>(x: T, y: T, FTable: FunctionTable) -> T {
    let xCopy = FTable.copy(x)
    let yCopy = FTable.copy(y)
    let m = FTable.lessThan(yCopy, xCopy) ? y : x
    FTable.release(x)
    FTable.release(y)
    return m
}
```

How Generics Work

```
func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

```
func min<T : Comparable>(x: T, y: T, FTable: FunctionTable) -> T {
    let xCopy = FTable.copy(x)
    let yCopy = FTable.copy(y)
    let m = FTable.lessThan(yCopy, xCopy) ? y : x
    FTable.release(x)
    FTable.release(y)
    return m
}
```

Generic Specialization

```
func foo()  {
   let x: Int = ...
   let y: Int = ...
   let r = min(x, y)
   ...
}
```

Generic Specialization

```
func foo()  {
   let x: Int = ...
   let y: Int = ...
   let r = min(x, y)
   ...
}
```

Generic Specialization

```
func foo()  {
   let x: Int = ...
   let y: Int = ...
   let r = min(x, y)
   ...
}
```

```
func min<T : Comparable>(x: T, y: T, FTable: FunctionTable) -> T {
    let xCopy = FTable.copy(x)
    let yCopy = FTable.copy(y)
    let m = FTable.lessThan(yCopy, xCopy) ? y : x
    FTable.release(x)
    FTable.release(y)
    return m
}
```

Generic Specialization

```
func foo()  {
   let x: Int = ...
   let y: Int = ...
   let r = min(x, y)
   ...
}
```

```
func min<Int>(x: Int, y: Int, FTable: FunctionTable) -> Int {
    let xCopy = FTable.copy(x)
    let yCopy = FTable.copy(y)
    let m = FTable.lessThan(yCopy, xCopy) ? y : x
    FTable.release(x)
    FTable.release(y)
    return m
}
```

Generic Specialization

```
func foo()  {
   let x: Int = ...
   let y: Int = ...
   let r = min(x, y)
   ...
}
```

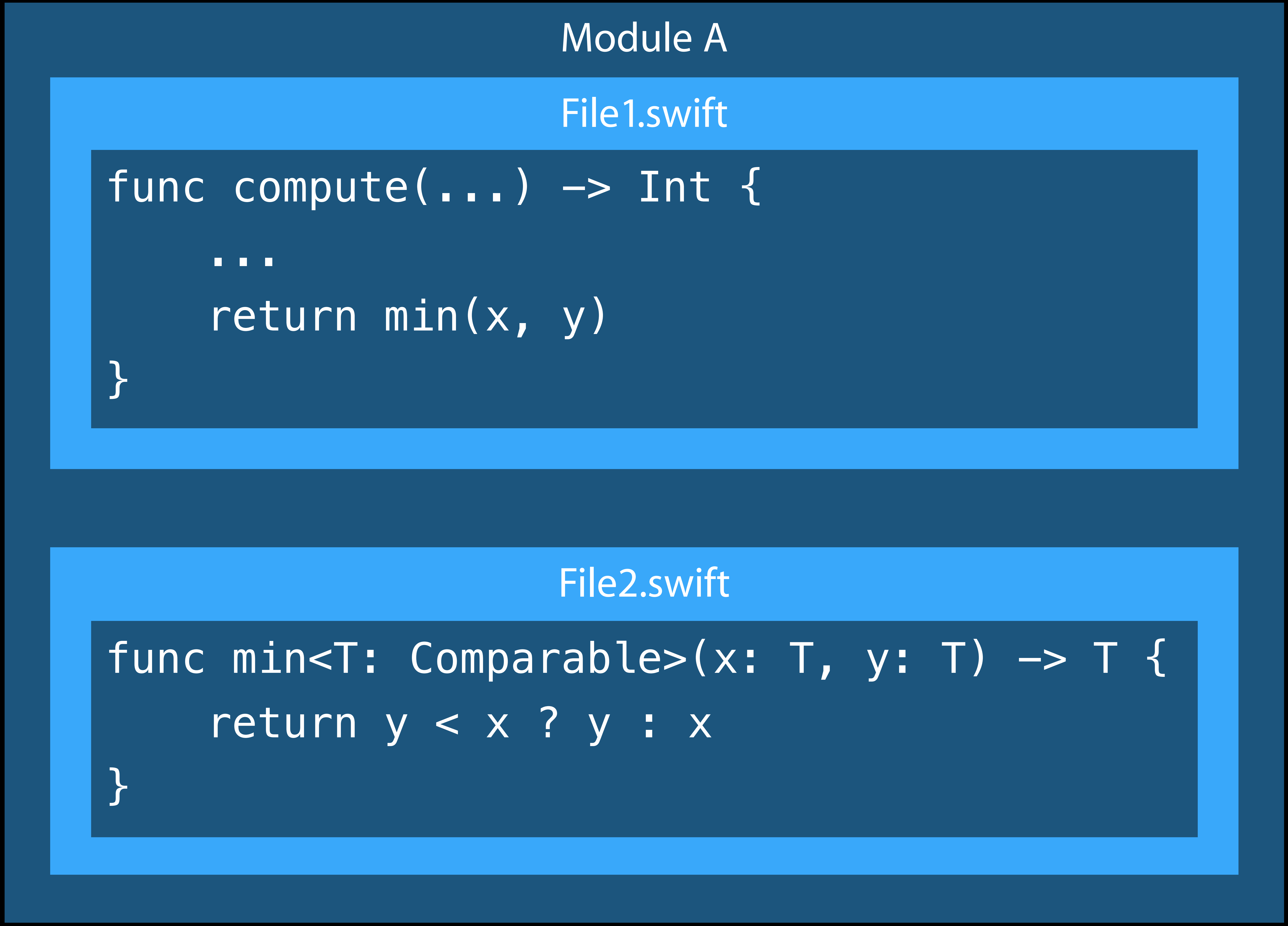
```
func min<Int>(x: Int, y: Int) -> Int {
    return y < x ? y : x
}
```

Generic Specialization

```
func foo()  {
   let x: Int = ...
   let y: Int = ...
   let r = min<Int>(x, y)
   ...
}
```

```
func min<Int>(x: Int, y: Int) -> Int {
    return y < x ? y : x
}
```

Specialization is Limited by Visibility



Module A

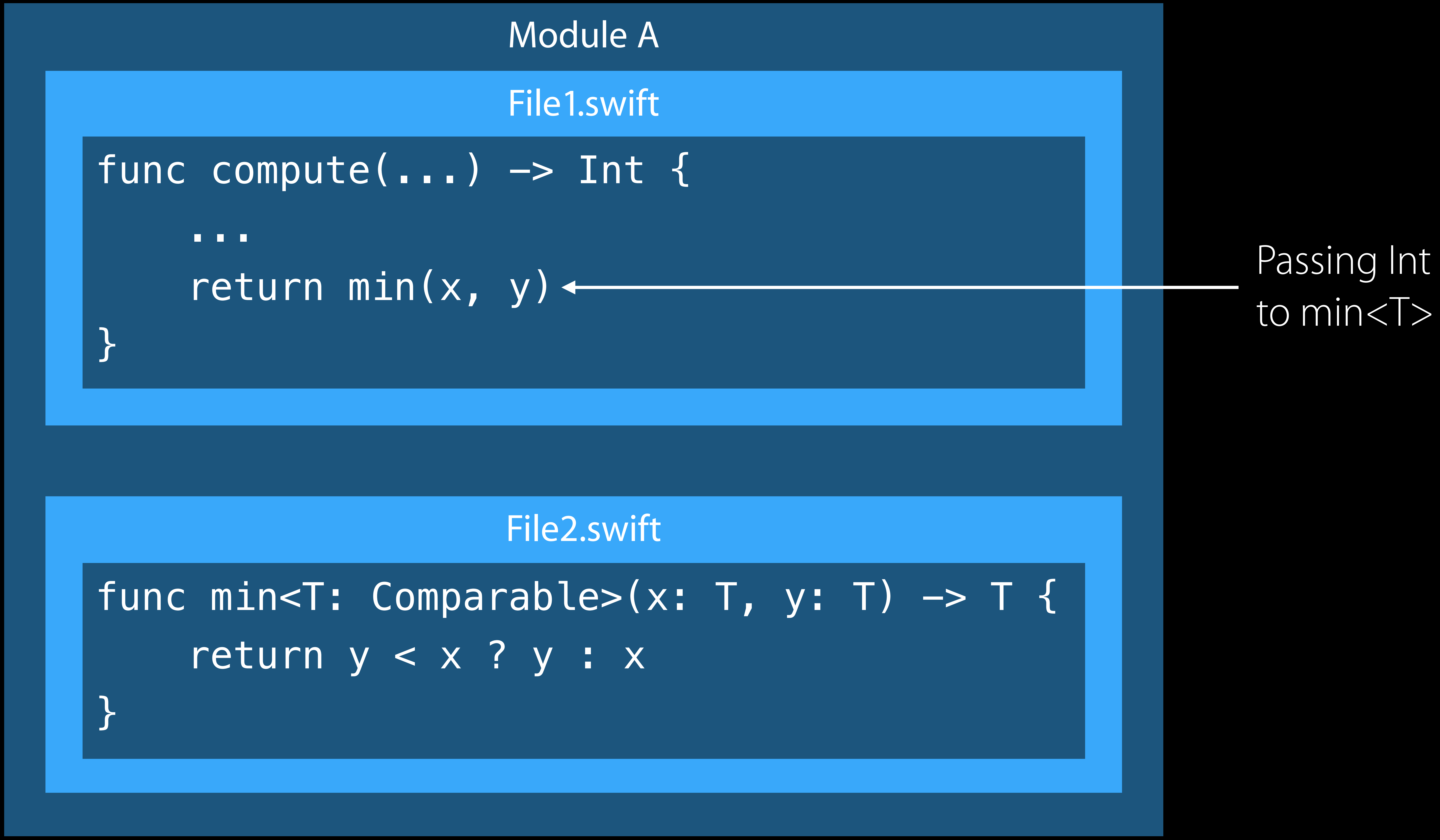
File1.swift

```
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

File2.swift

```
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

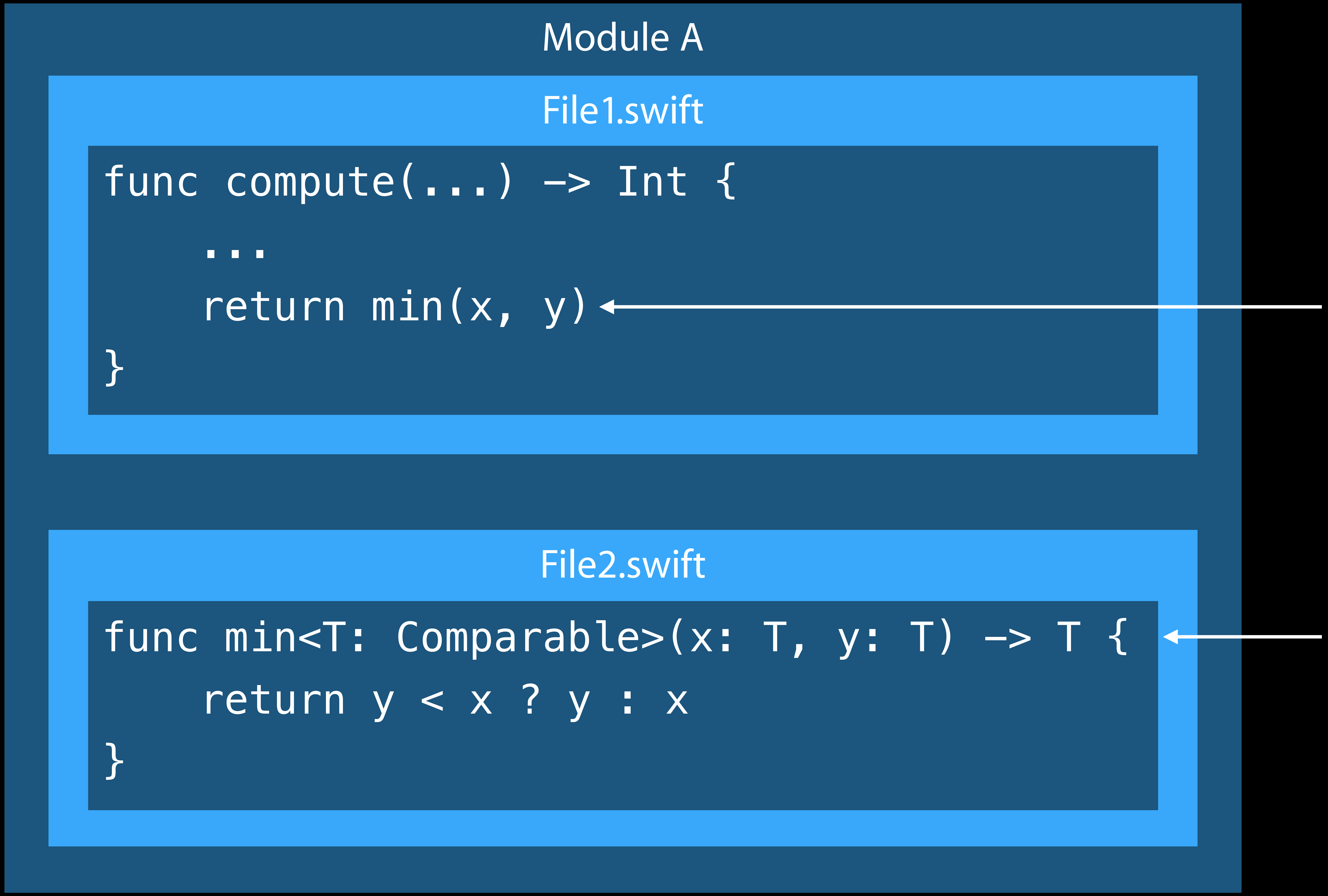
Specialization is Limited by Visibility



```
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

```
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

Specialization is Limited by Visibility



Module A

File1.swift

```
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

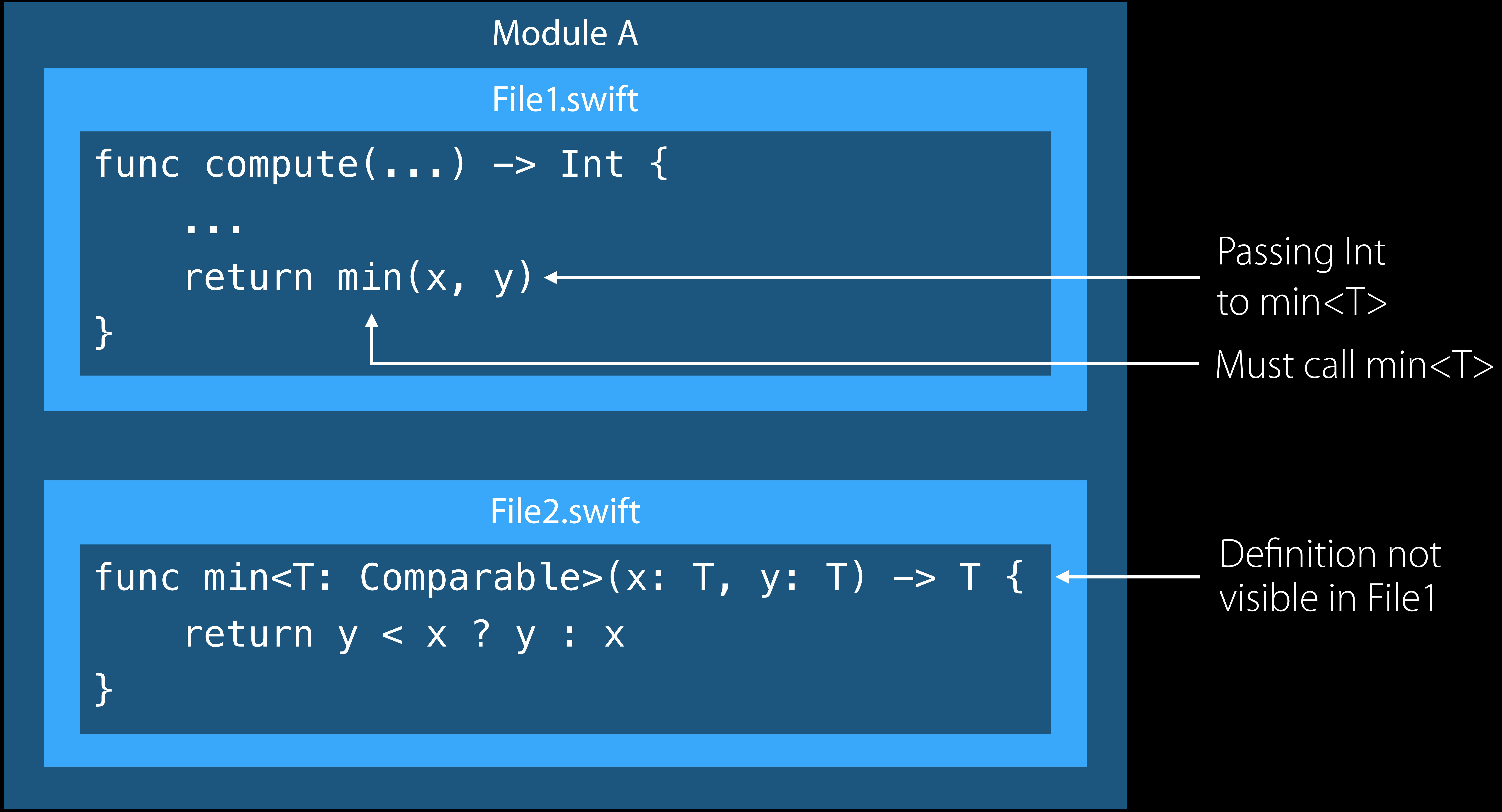
File2.swift

```
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

Passing Int
to min<T>

Definition not
visible in File1

Specialization is Limited by Visibility



Module A

File1.swift

```
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

Passing Int
to min<T>

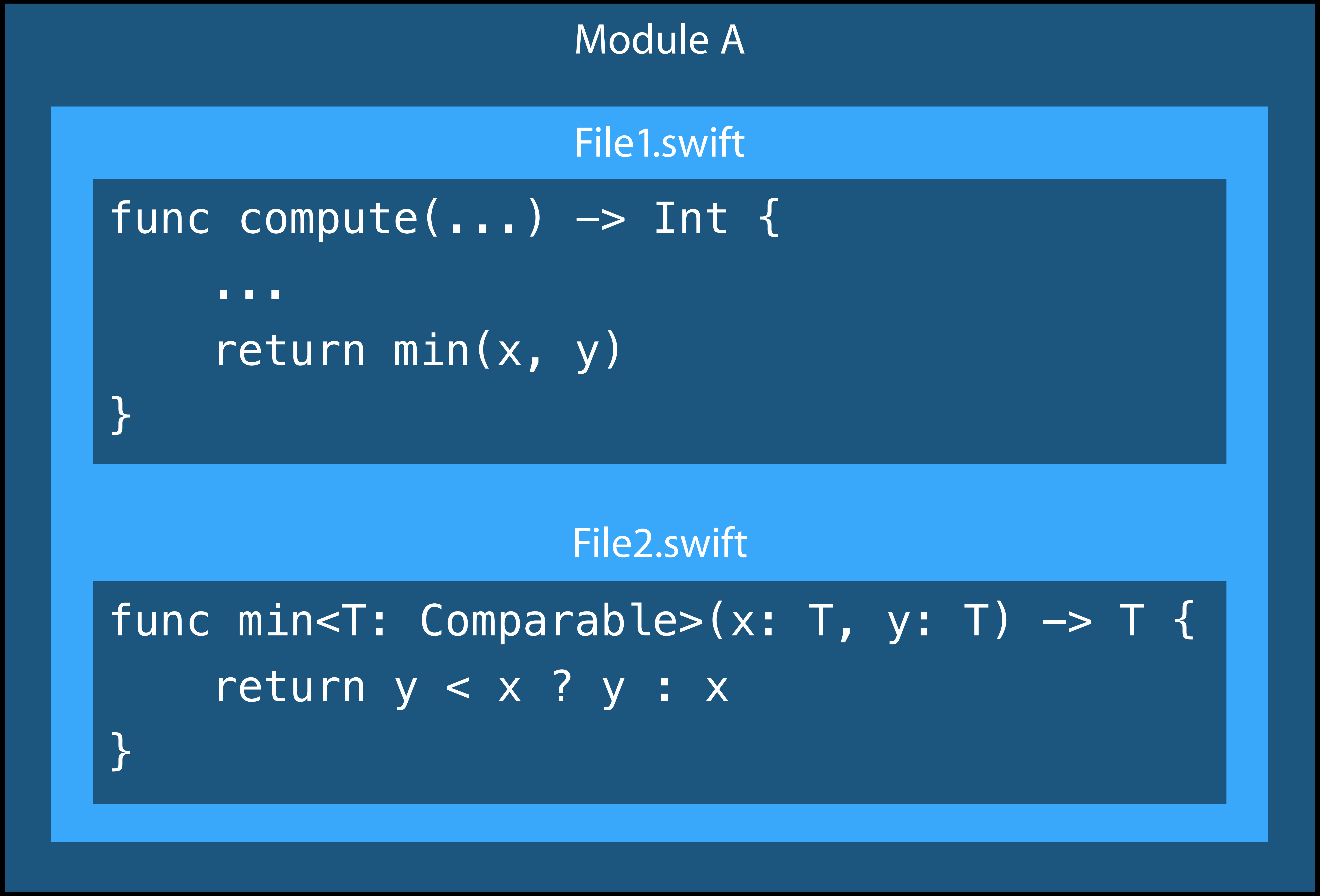
Must call min<T>

File2.swift

```
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

Definition not
visible in File1

Whole Module Optimization



Module A

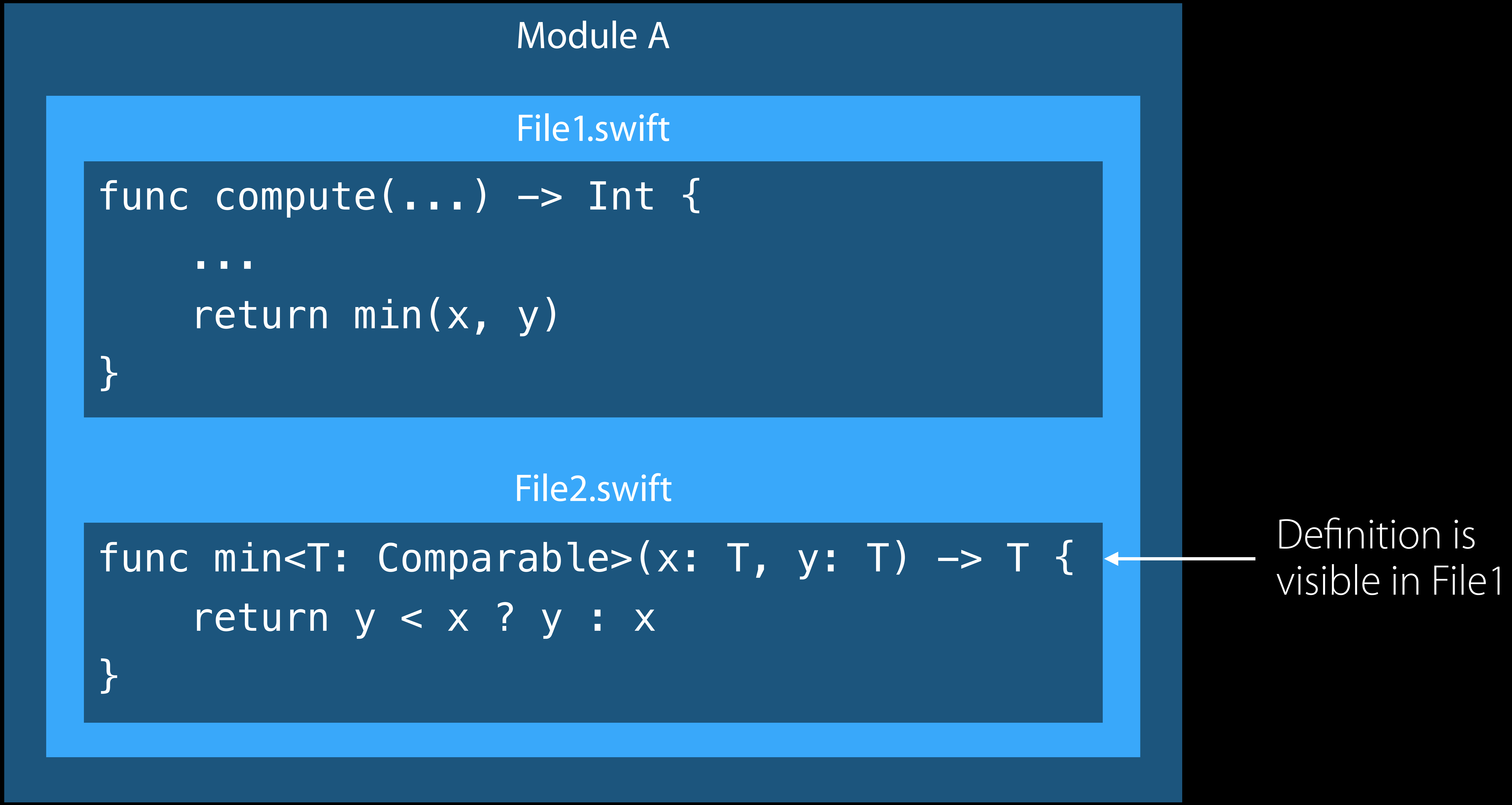
File1.swift

```
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

File2.swift

```
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

Whole Module Optimization



Module A

File1.swift

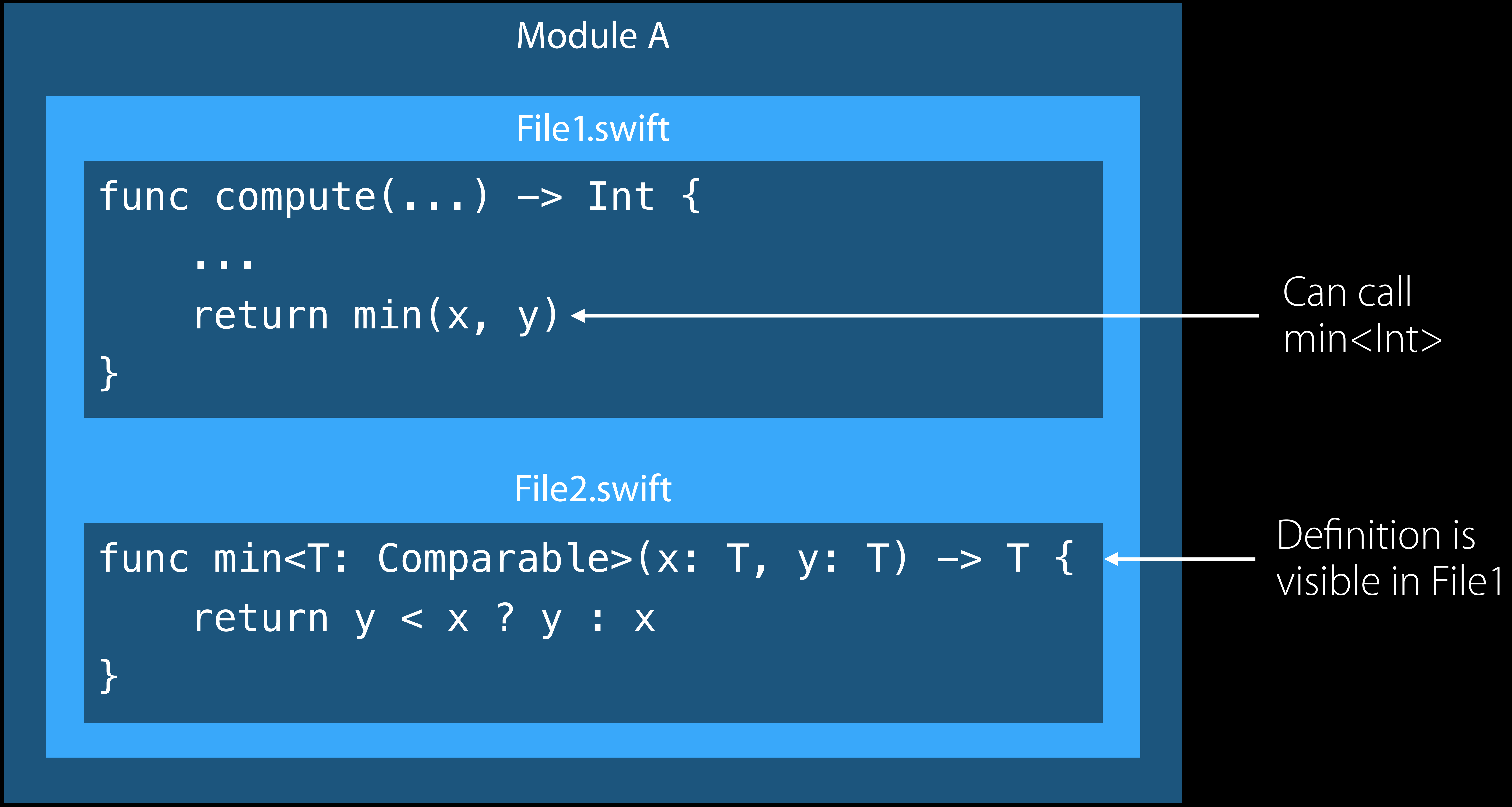
```
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

File2.swift

```
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

Definition is
visible in File1

Whole Module Optimization



Module A

File1.swift

```
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

Can call
min<Int>

File2.swift

```
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

Definition is
visible in File1

Overview

Reference Counting

Generics

Dynamic Dispatch

Overview

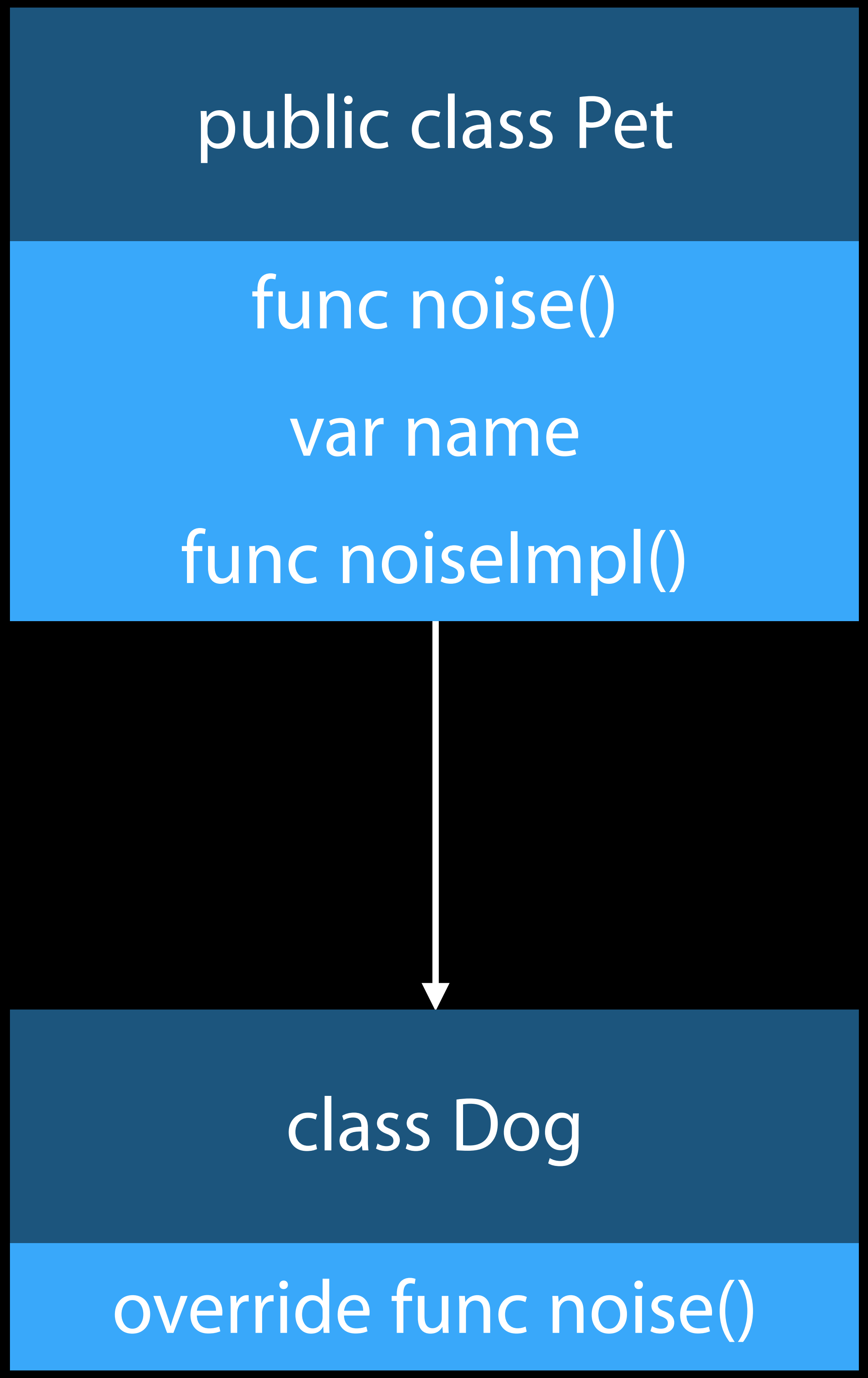
Reference Counting

Generics

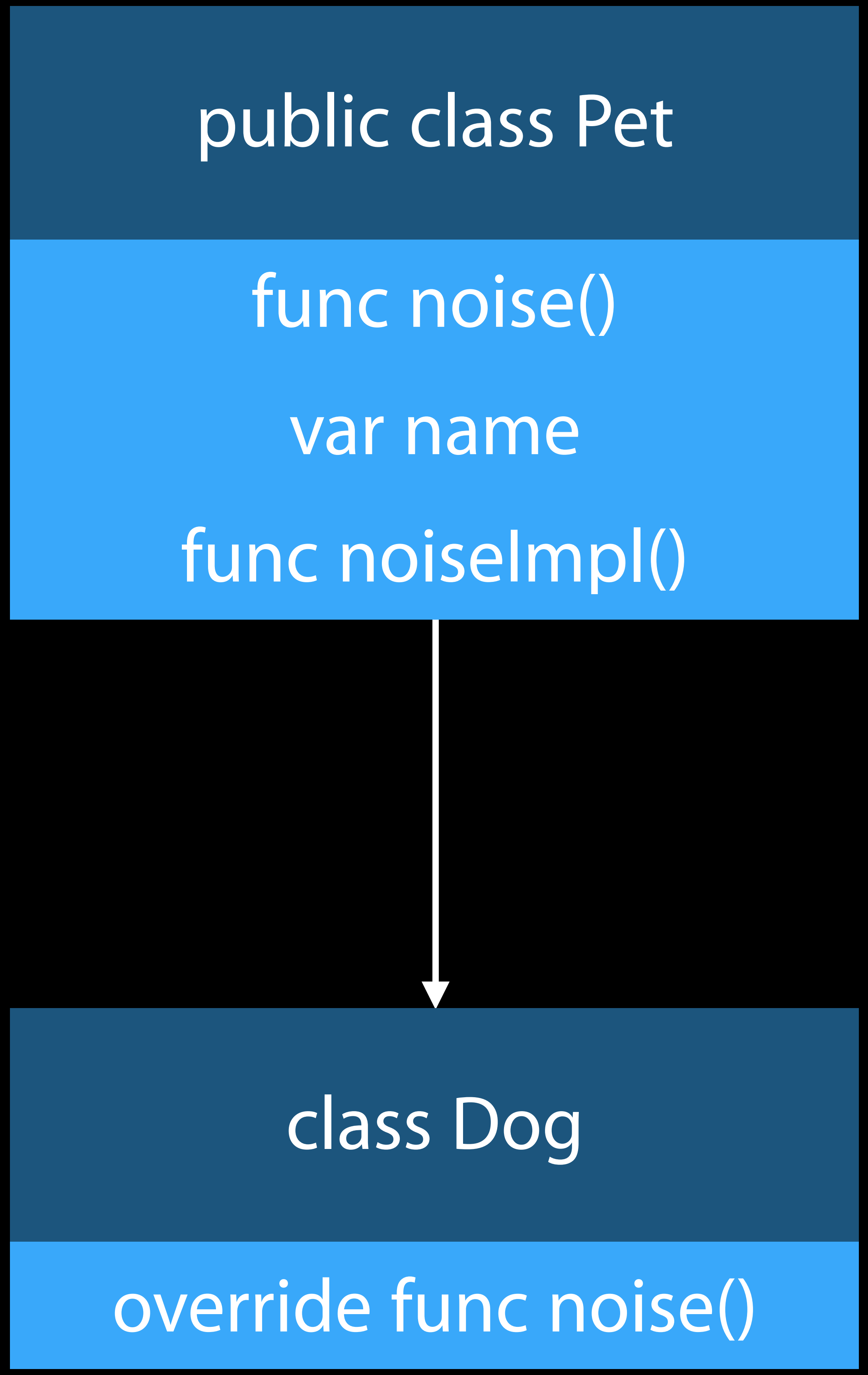
Dynamic Dispatch

Dynamic Dispatch

Dynamic Dispatch



Dynamic Dispatch



public class Pet

func noise()

var name

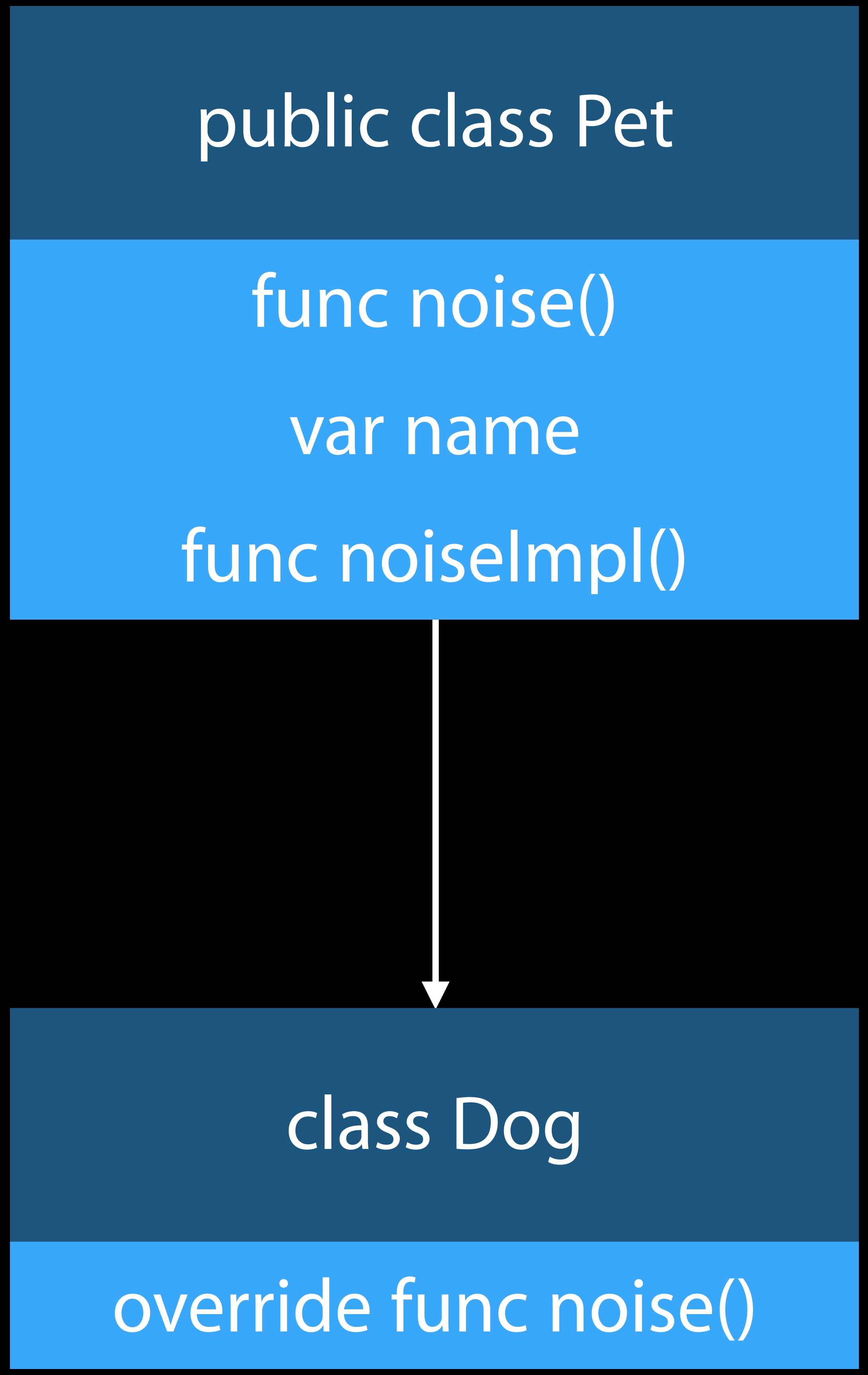
func noiseImpl()

class Dog

override func noise()

```
func makeNoise(p: Pet) {
    print("My name is \(p.name)")
    p.noise()
}
```

Dynamic Dispatch



public class Pet

func noise()

var name

func noiseImpl()

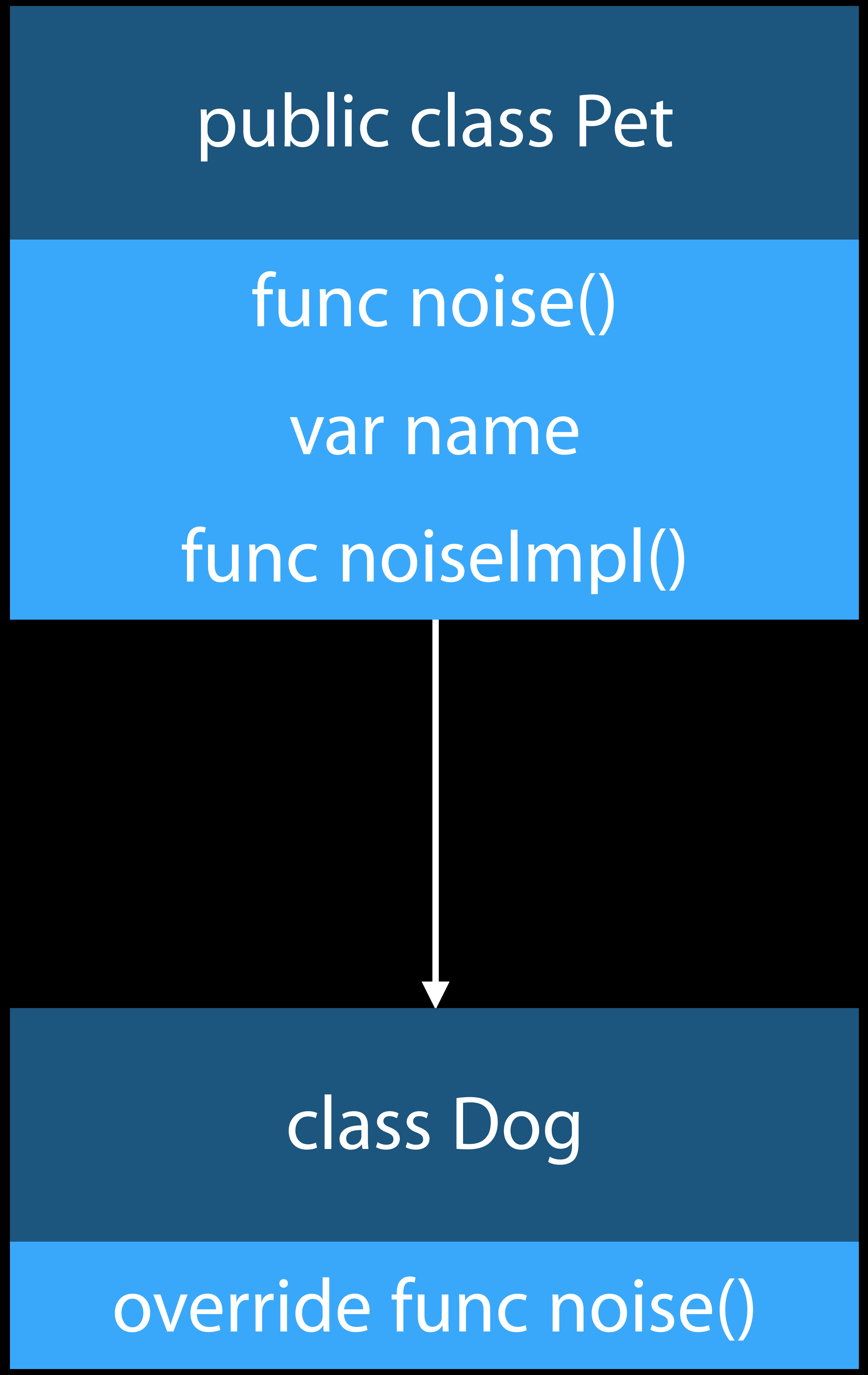
class Dog

override func noise()

```
func makeNoise(p: Pet) {
    print("My name is \(p.name)")
    p.noise()
}
```

```
func makeNoise(p: Pet) {
    print("My name is \(p.name)")
    p.noise()
}
```

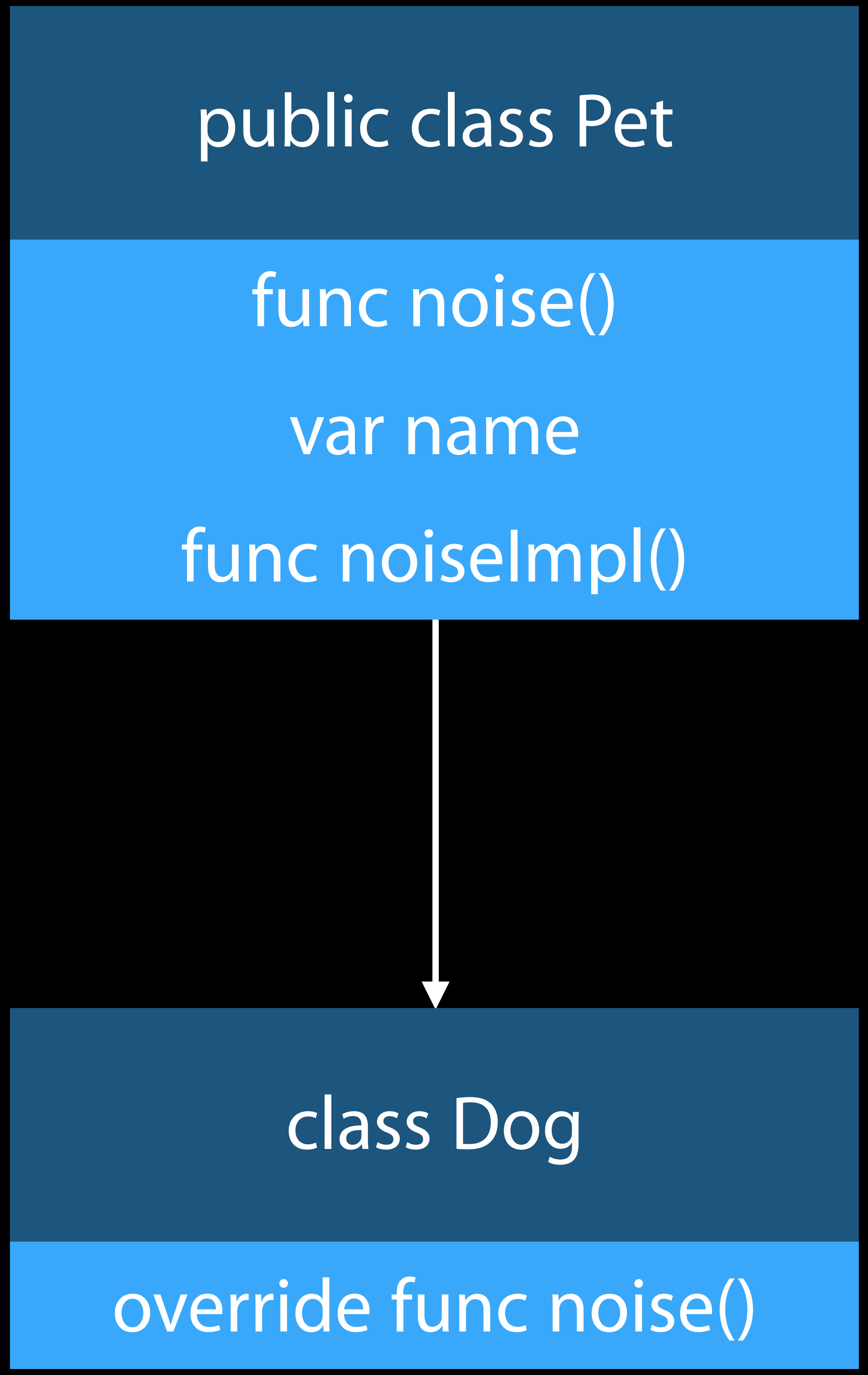
Dynamic Dispatch



```
func makeNoise(p: Pet) {
    print("My name is \(p.name)")
    p.noise()
}
```

```
func makeNoise(p: Pet) {



}
```

Dynamic Dispatch



public class Pet

func noise()

var name

func noiseImpl()

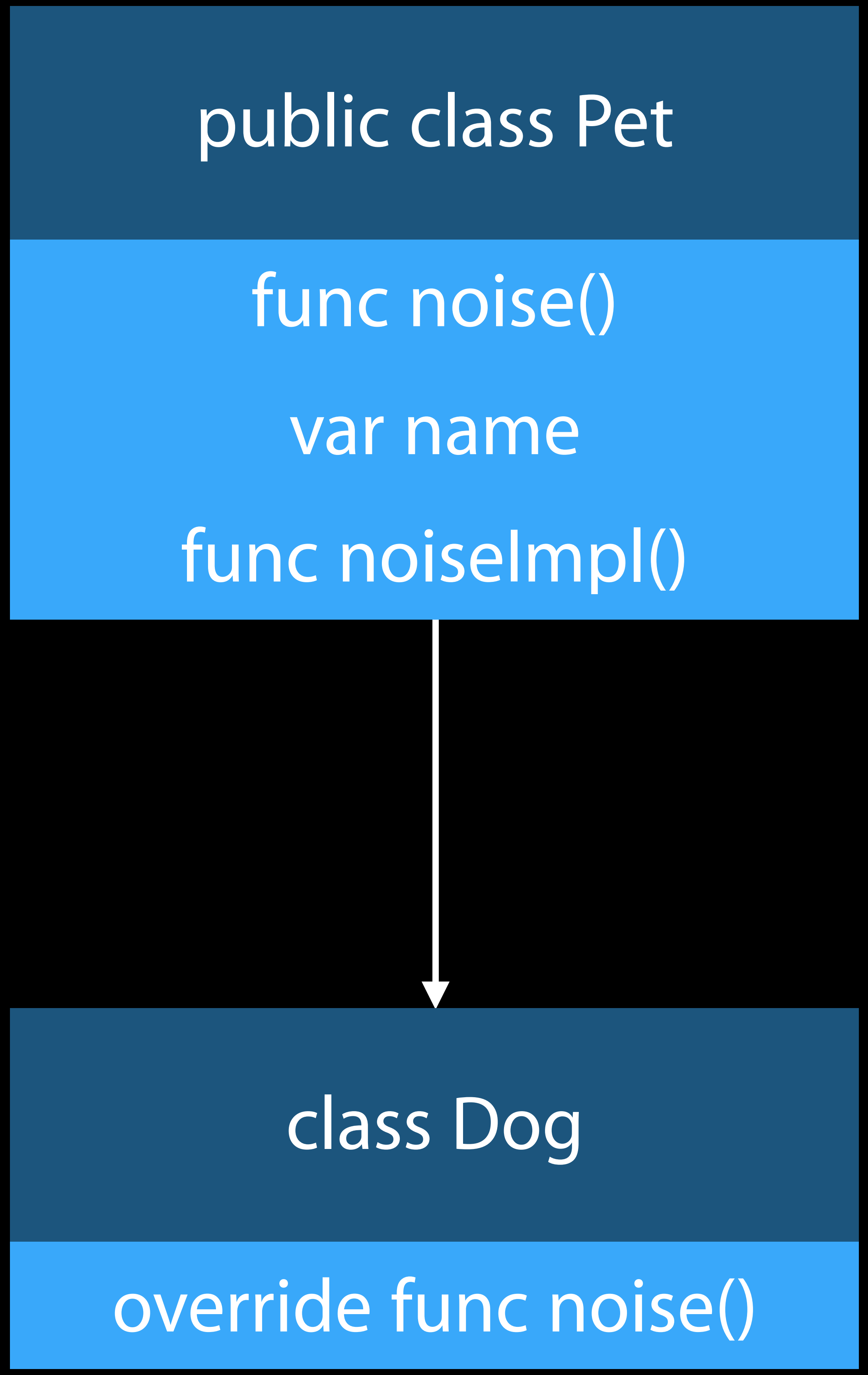
class Dog

override func noise()

```
func makeNoise(p: Pet) {
    print("My name is \(p.name)")
    p.noise()
}
```

```
func makeNoise(p: Pet) {
    let nameGetter = Pet.nameGetter(p)
    print("My name is \(nameGetter(p))")
    let noiseMethod = Pet.noiseMethod(p)
    noiseMethod(p)
}
```

Dynamic Dispatch



```
func makeNoise(p: Pet) {
    print("My name is \(p.name)")
    p.noise()
}
```

```
func makeNoise(p: Pet) {
    let nameGetter = Pet.nameGetter(p)
    print("My name is \(nameGetter(p))")
    let noiseMethod = Pet.noiseMethod(p)
    noiseMethod(p)
}
```

Can only emit direct calls if it is known that the method is not overridden

Communicate API Constraints

Communicate API Constraints

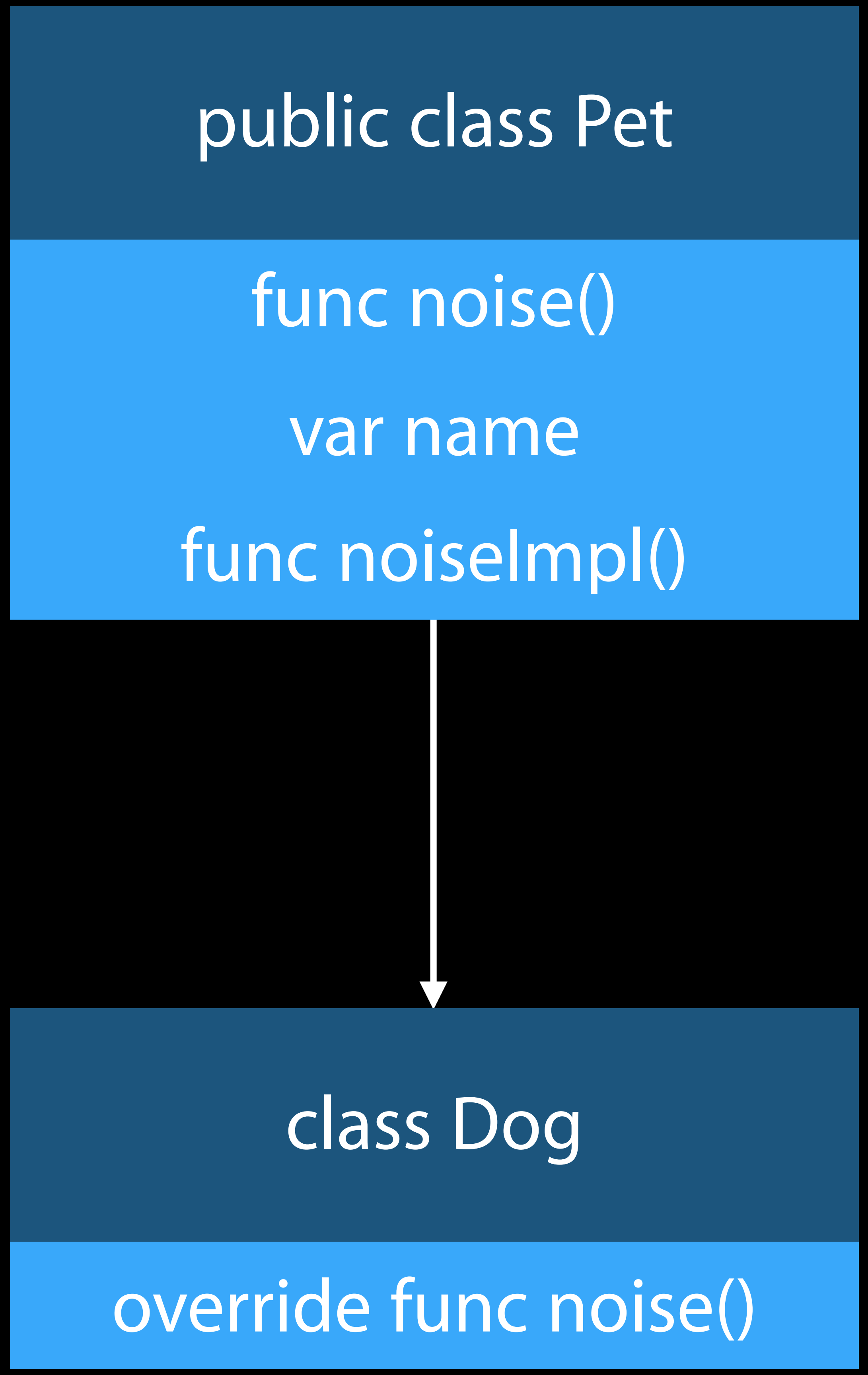
Inheritance

Communicate API Constraints

Inheritance

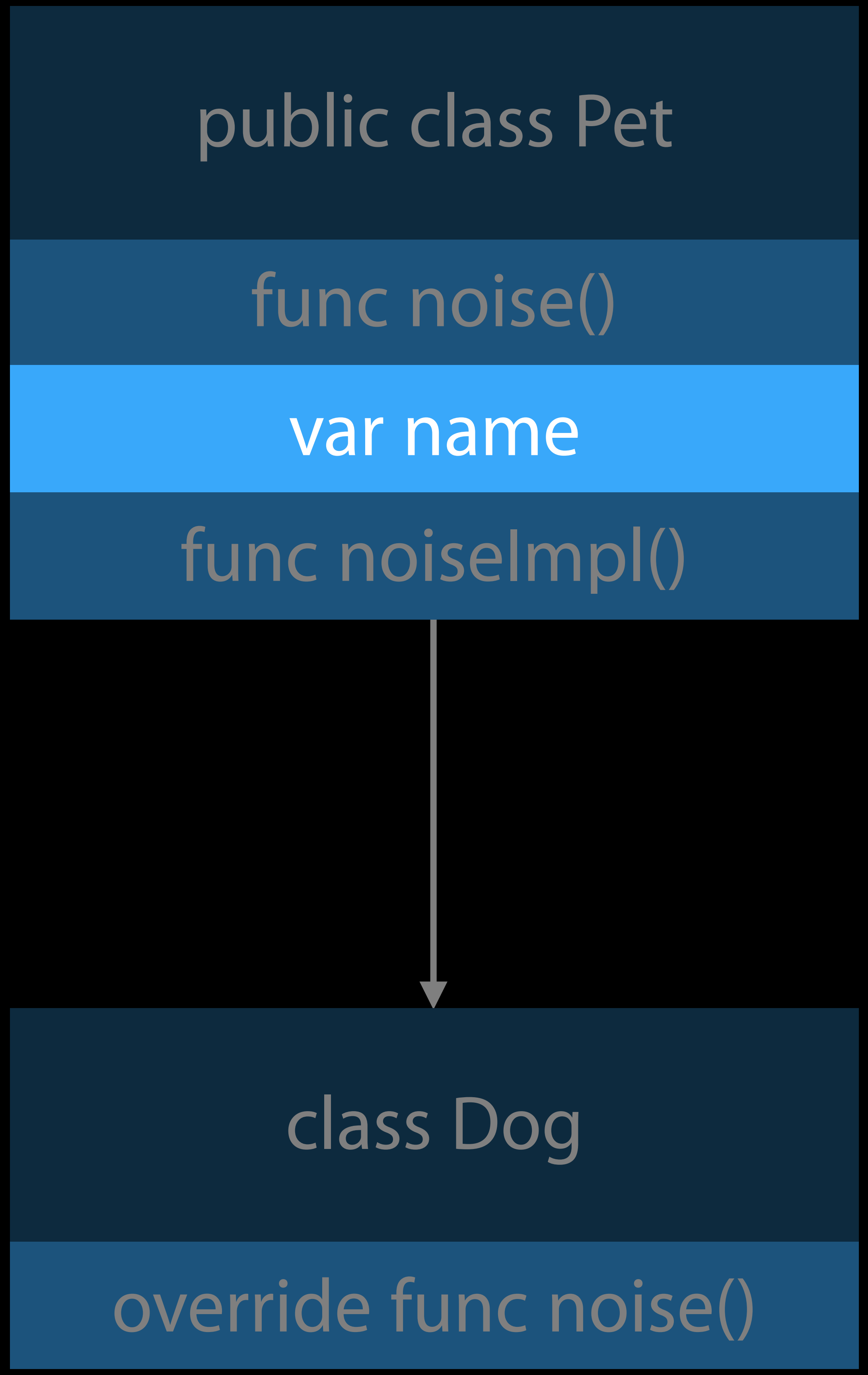
Access Control

Inheritance



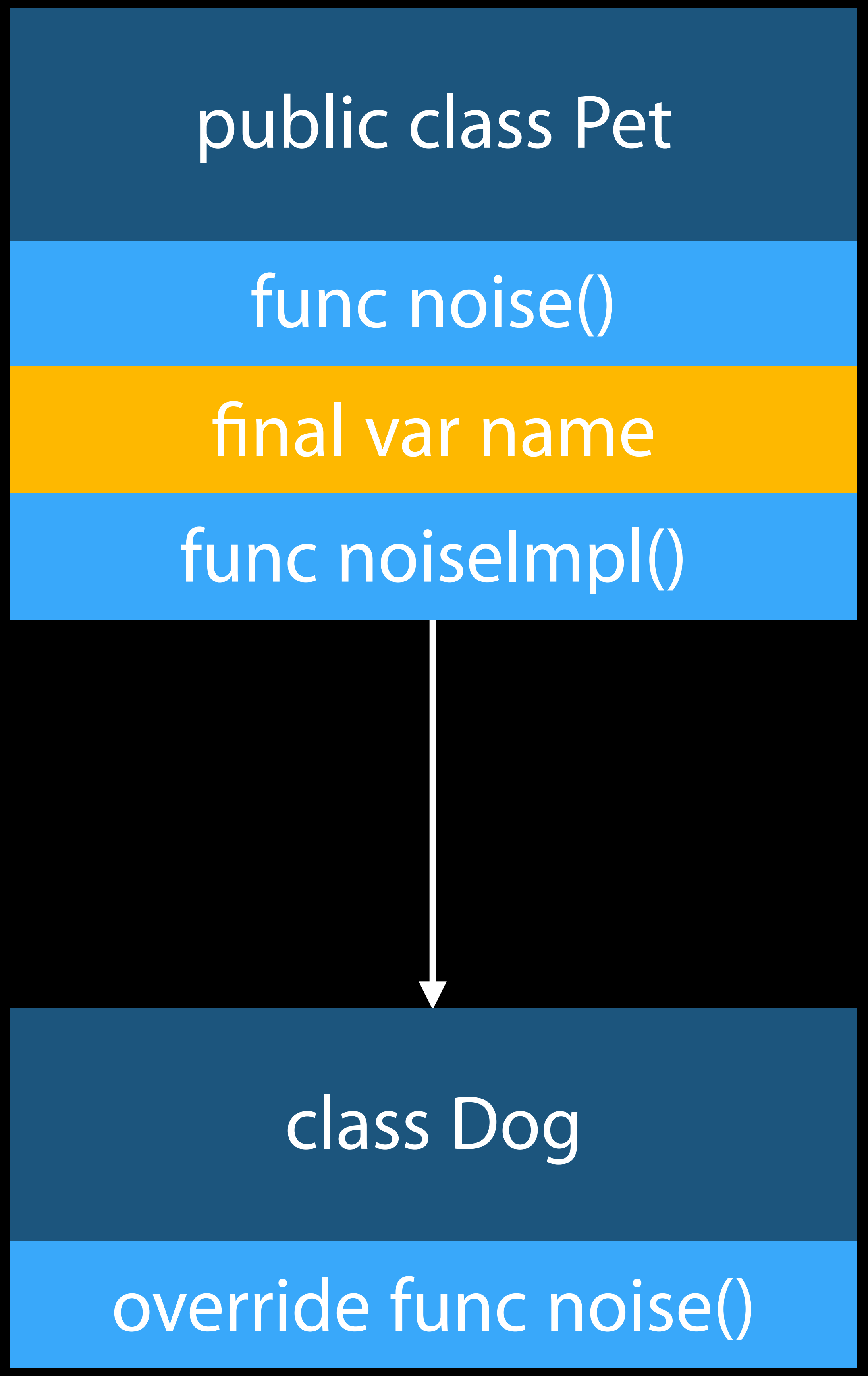
```
func makeNoise(p: Pet) {
    let nameGetter = Pet.getNameGetter(p)
    print("My name is \(nameGetter(p))")
    let noiseMethod = Pet.getNoiseMethod(p)
    noiseMethod(p)
}
```

Inheritance



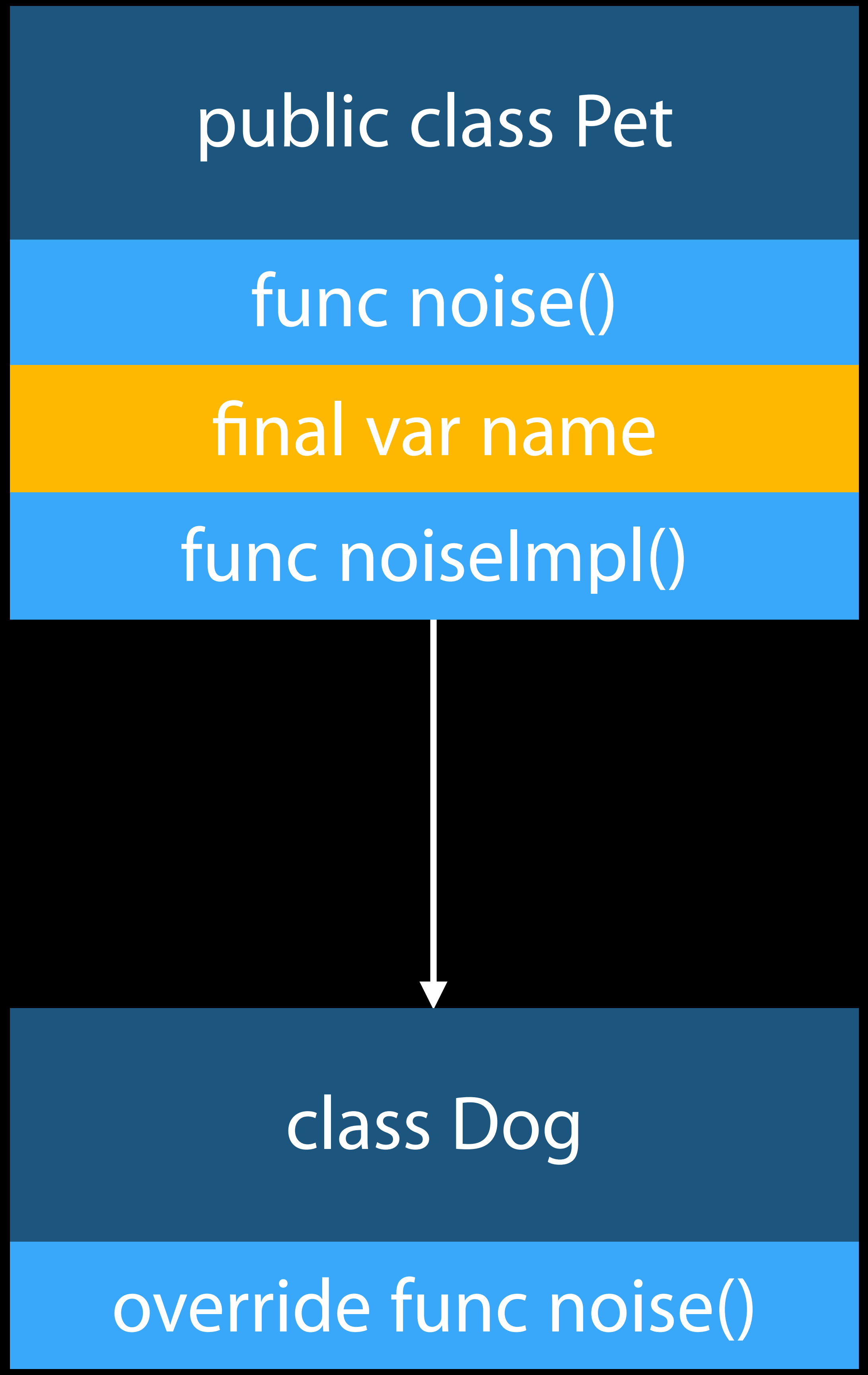
```
func makeNoise(p: Pet) {
    let nameGetter = Pet.getNameGetter(p)
    print("My name is \(nameGetter(p))")
    let noiseMethod = Pet.getNoiseMethod(p)
    noiseMethod(p)
}
```

Inheritance



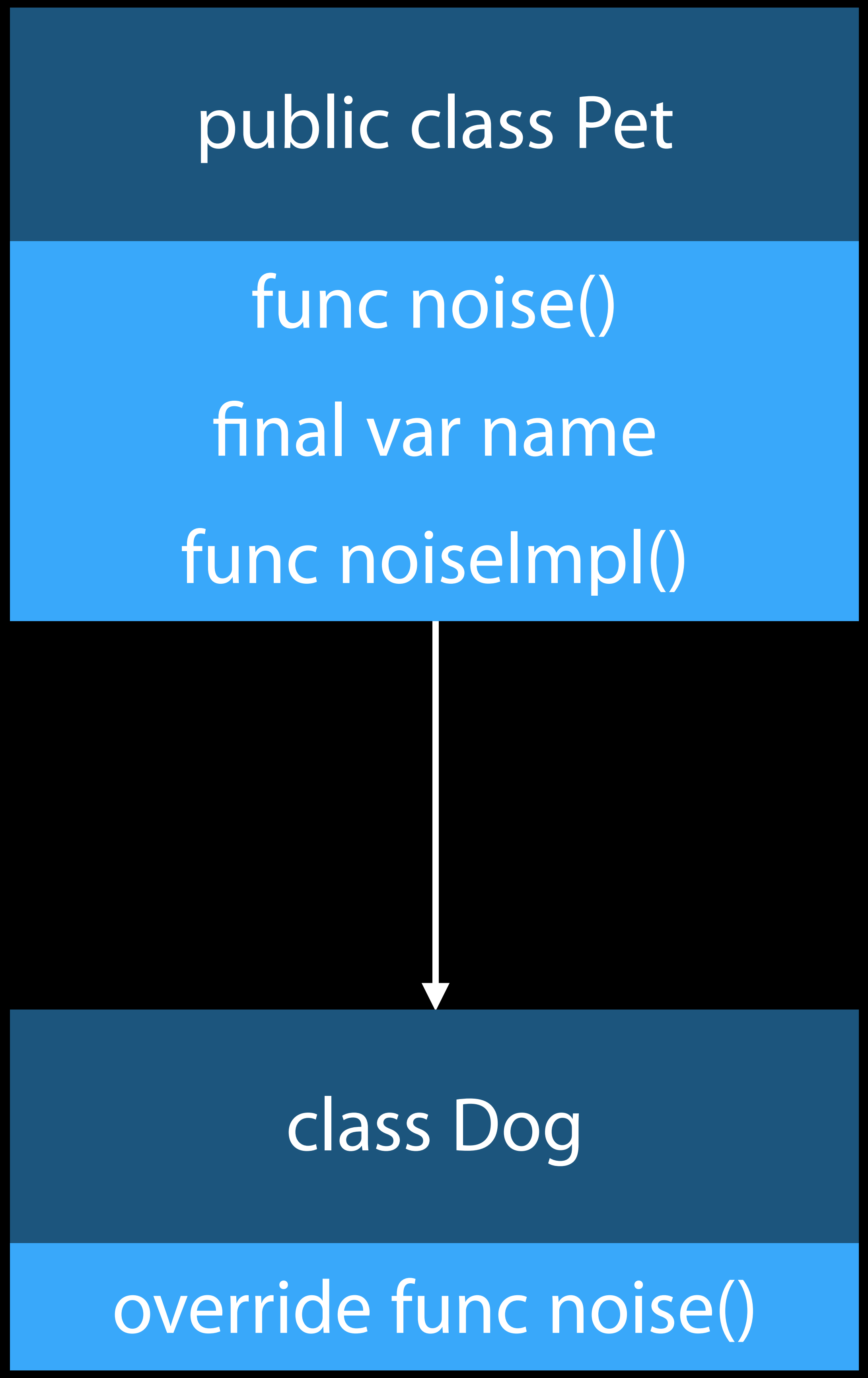
```
func makeNoise(p: Pet) {
    let nameGetter = Pet.getNameGetter(p)
    print("My name is \(nameGetter(p))")
    let noiseMethod = Pet.getNoiseMethod(p)
    noiseMethod(p)
}
```

Inheritance



```
func makeNoise(p: Pet) {
    print("My name is \(p.name)")
    let noiseMethod = Pet.getNoiseMethod(p)
    noiseMethod(p)
}
```

Access Control



public class Pet

func noise()

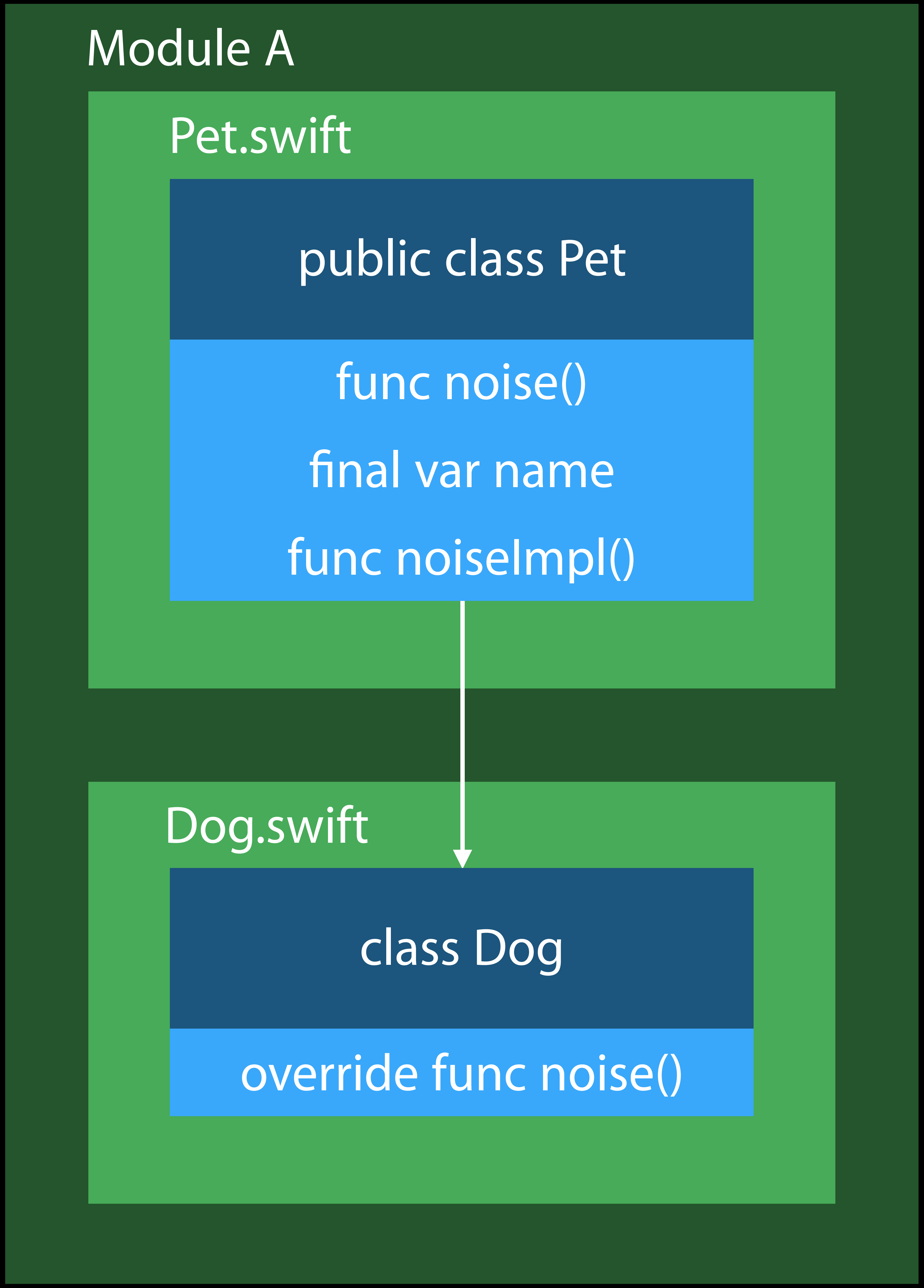
final var name

func noiseImpl()

class Dog

override func noise()

Access Control



Module A

Pet.swift

public class Pet

func noise()

final var name

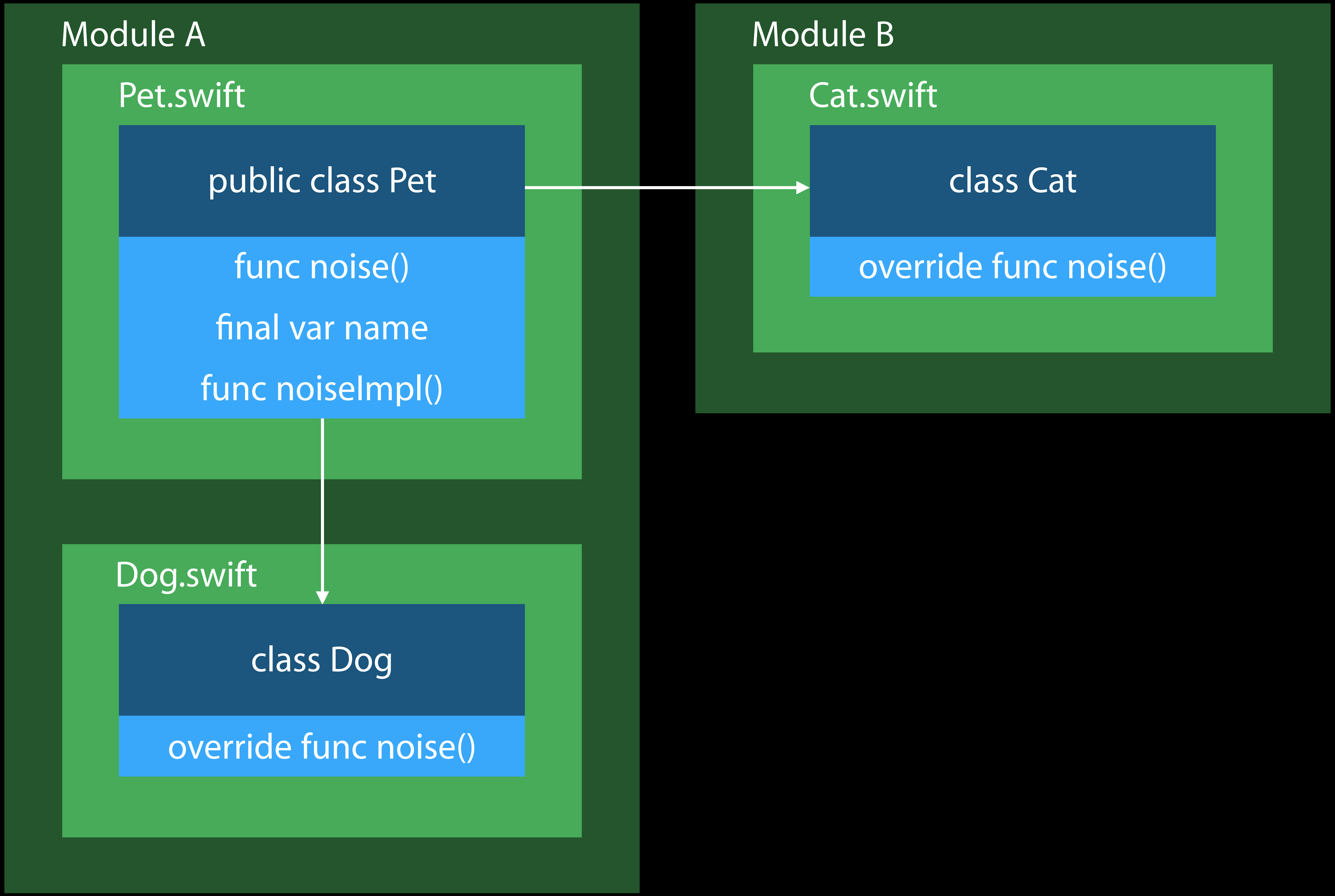
func noiseImpl()

Dog.swift

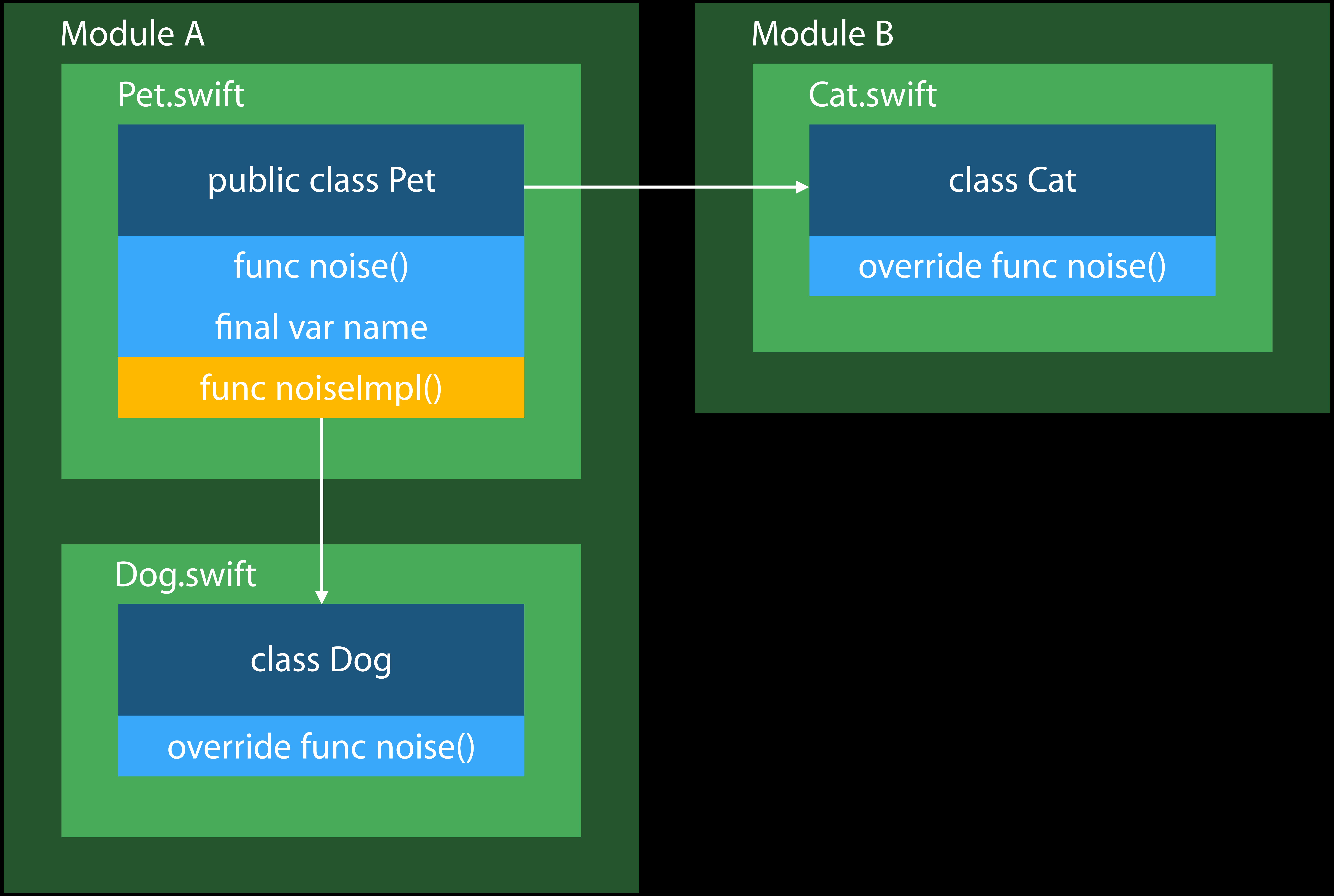
class Dog

override func noise()

Access Control



Access Control



Module A

Pet.swift

public class Pet

func noise()

final var name

func noiseImpl()

Dog.swift

class Dog

override func noise()

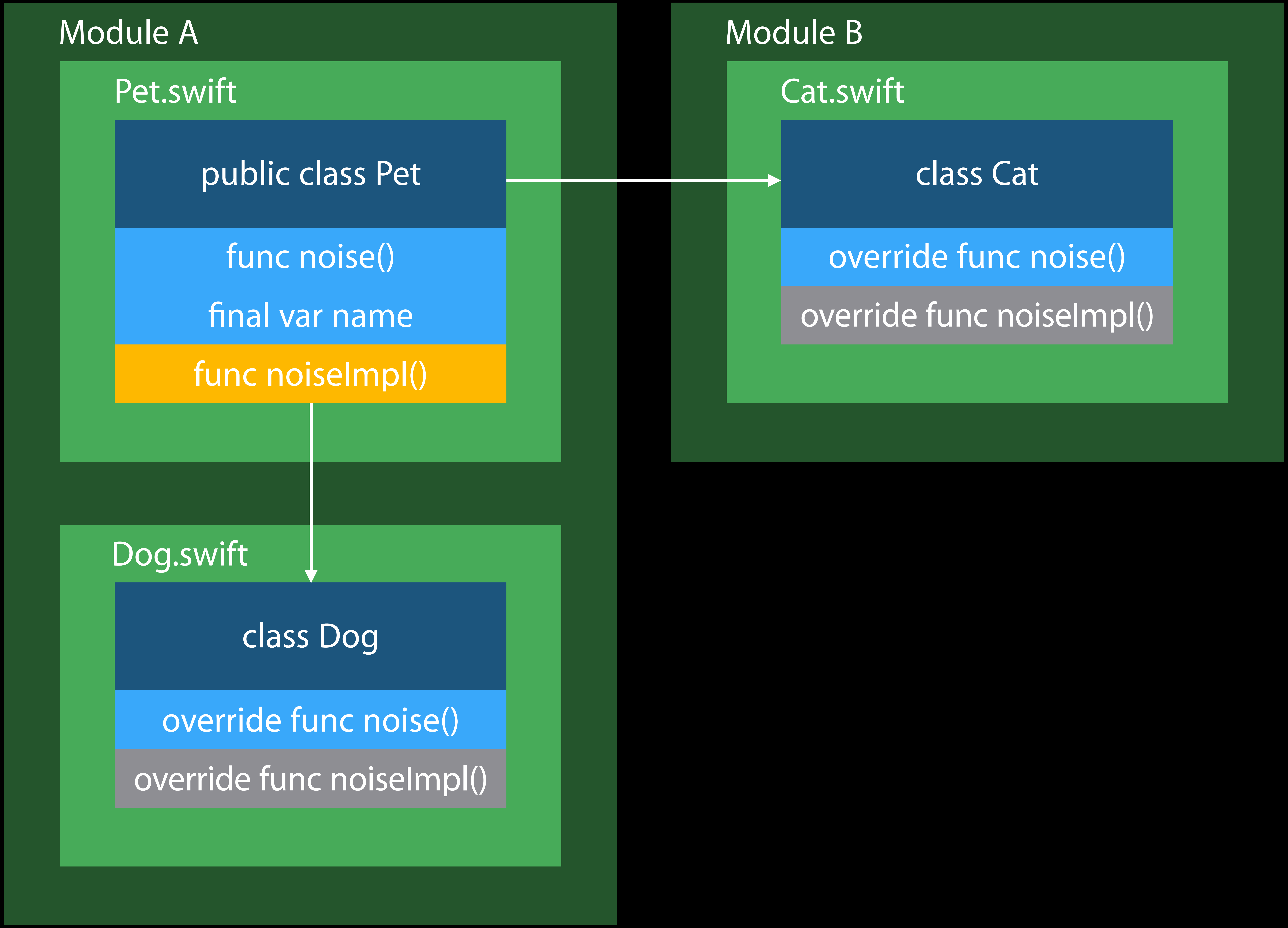
Module B

Cat.swift

class Cat

override func noise()

Access Control



Module A

Pet.swift

public class Pet

func noise()

final var name

func noiseImpl()

Dog.swift

class Dog

override func noise()

override func noiseImpl()

Module B

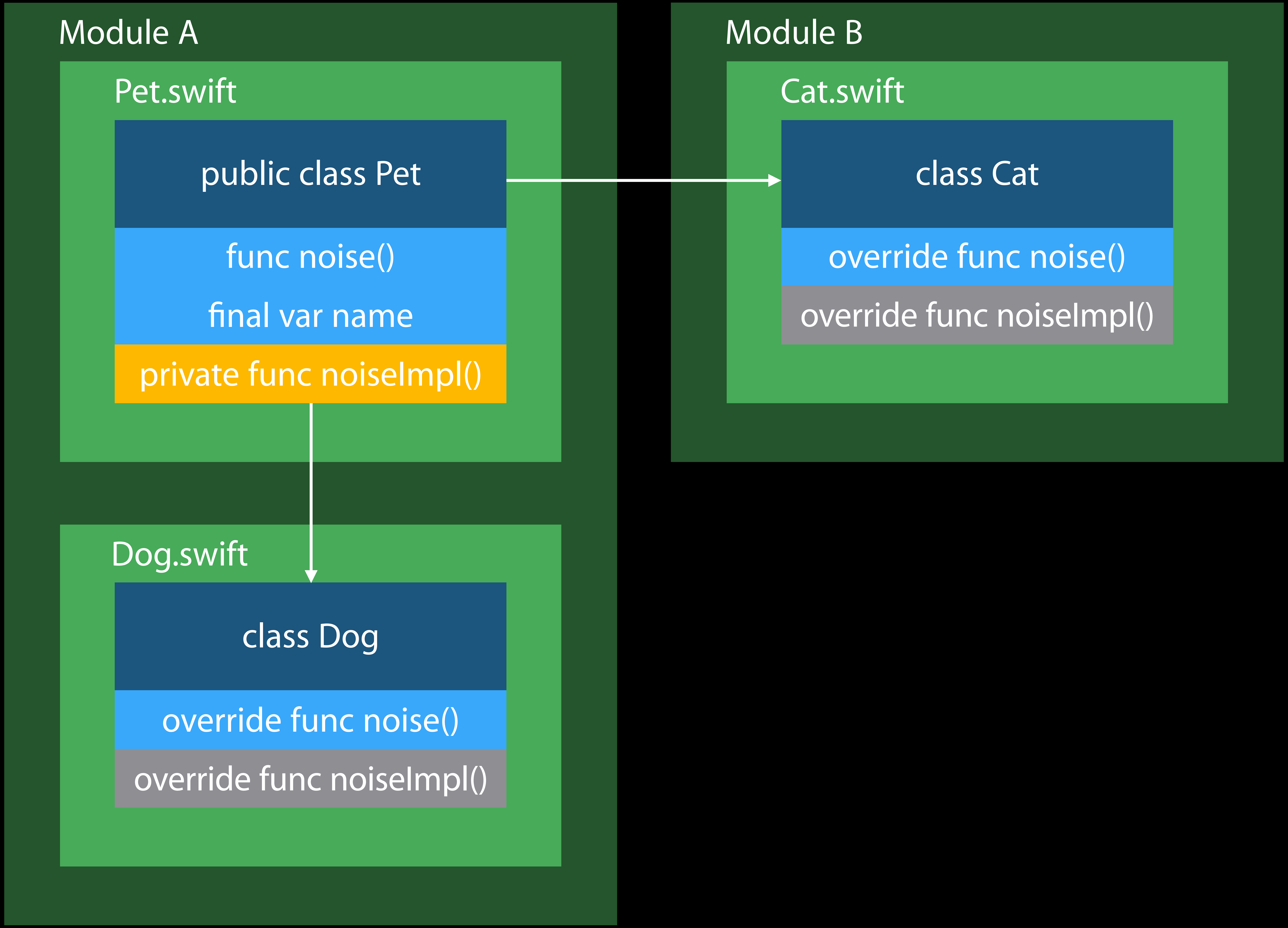
Cat.swift

class Cat

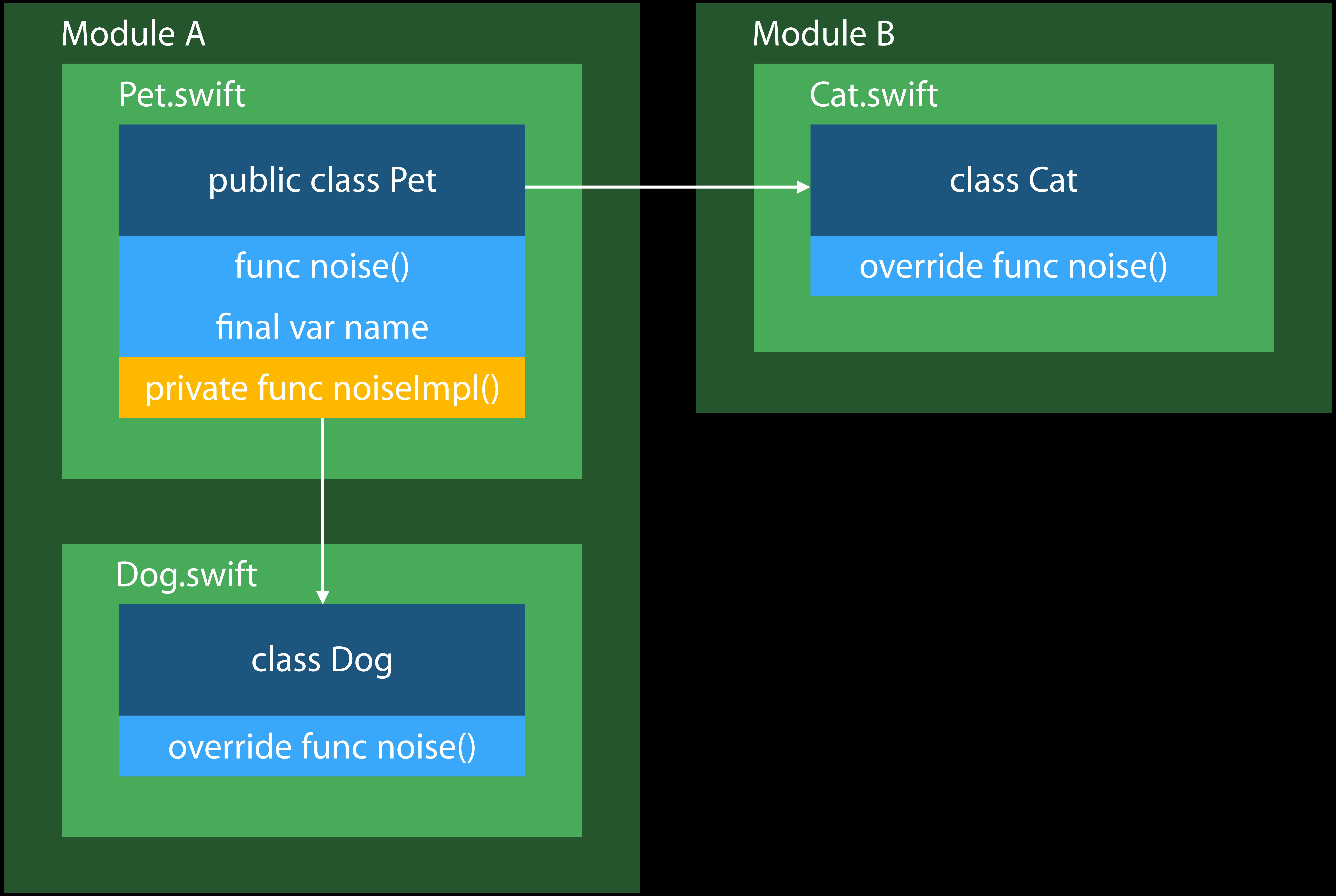
override func noise()

override func noiseImpl()

Access Control



Access Control



Module A

Pet.swift

public class Pet

func noise()

final var name

private func noiseImpl()

Dog.swift

class Dog

override func noise()

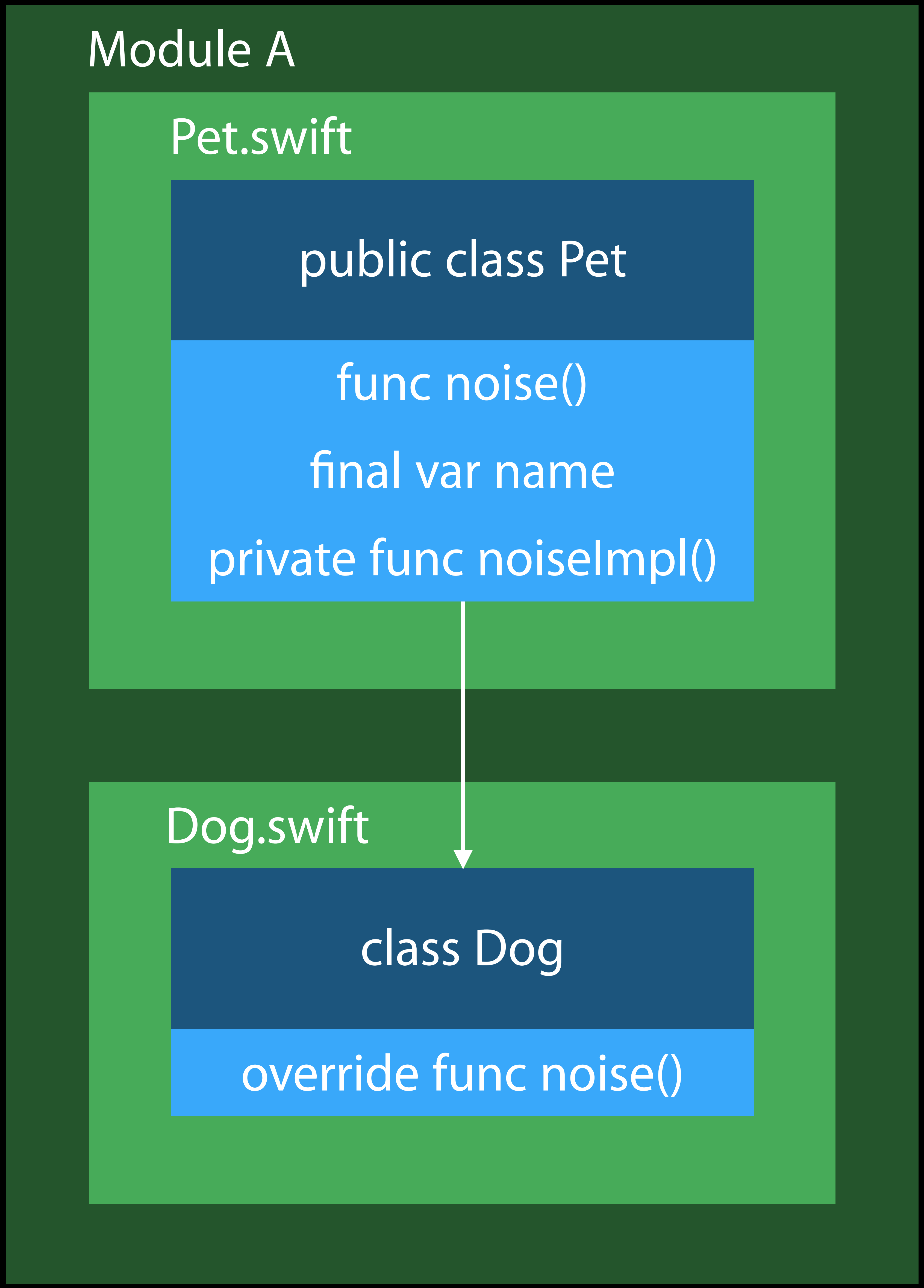
Module B

Cat.swift

class Cat

override func noise()

Whole Module Optimization



Module A

Pet.swift

public class Pet

func noise()

final var name

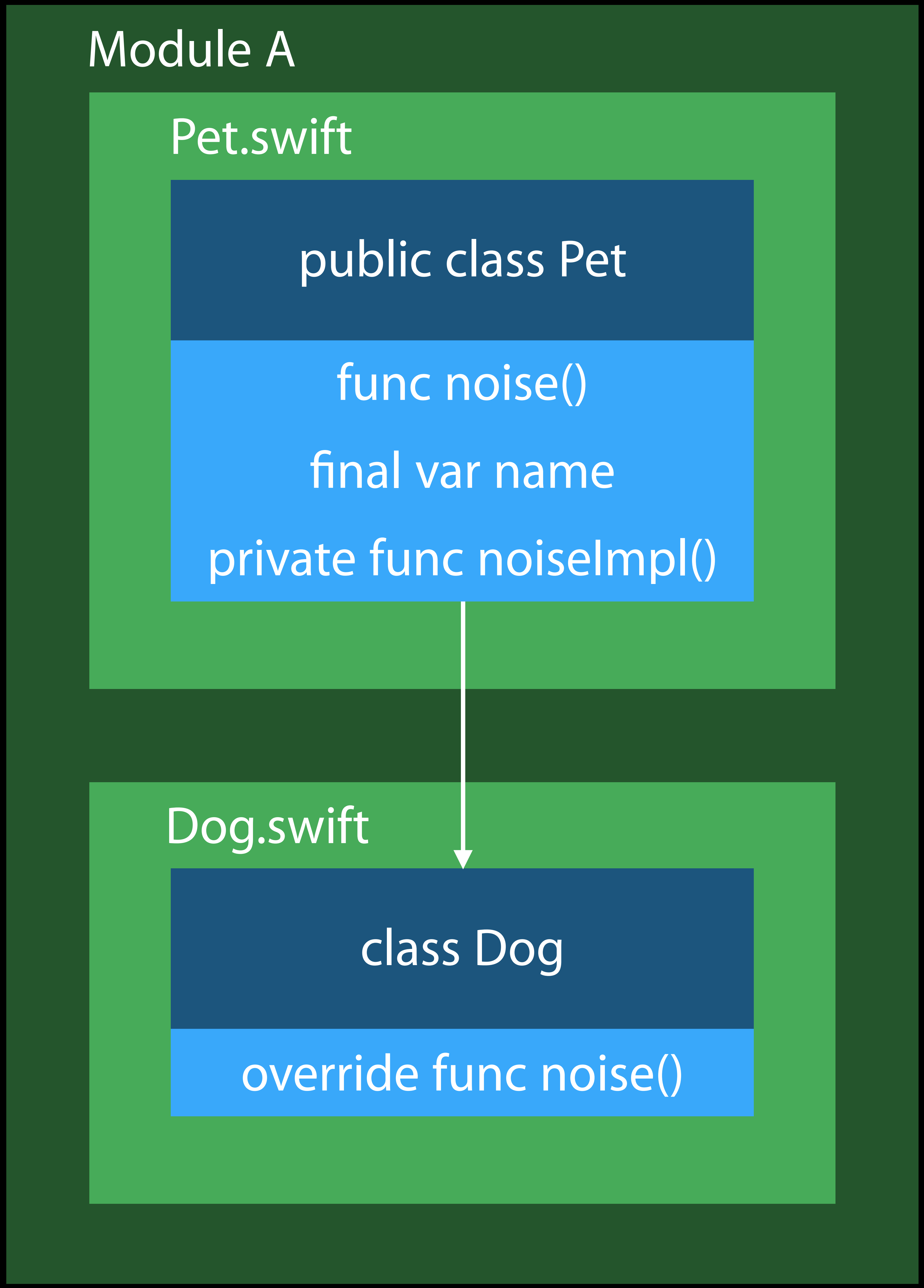
private func noiseImpl()

Dog.swift

class Dog

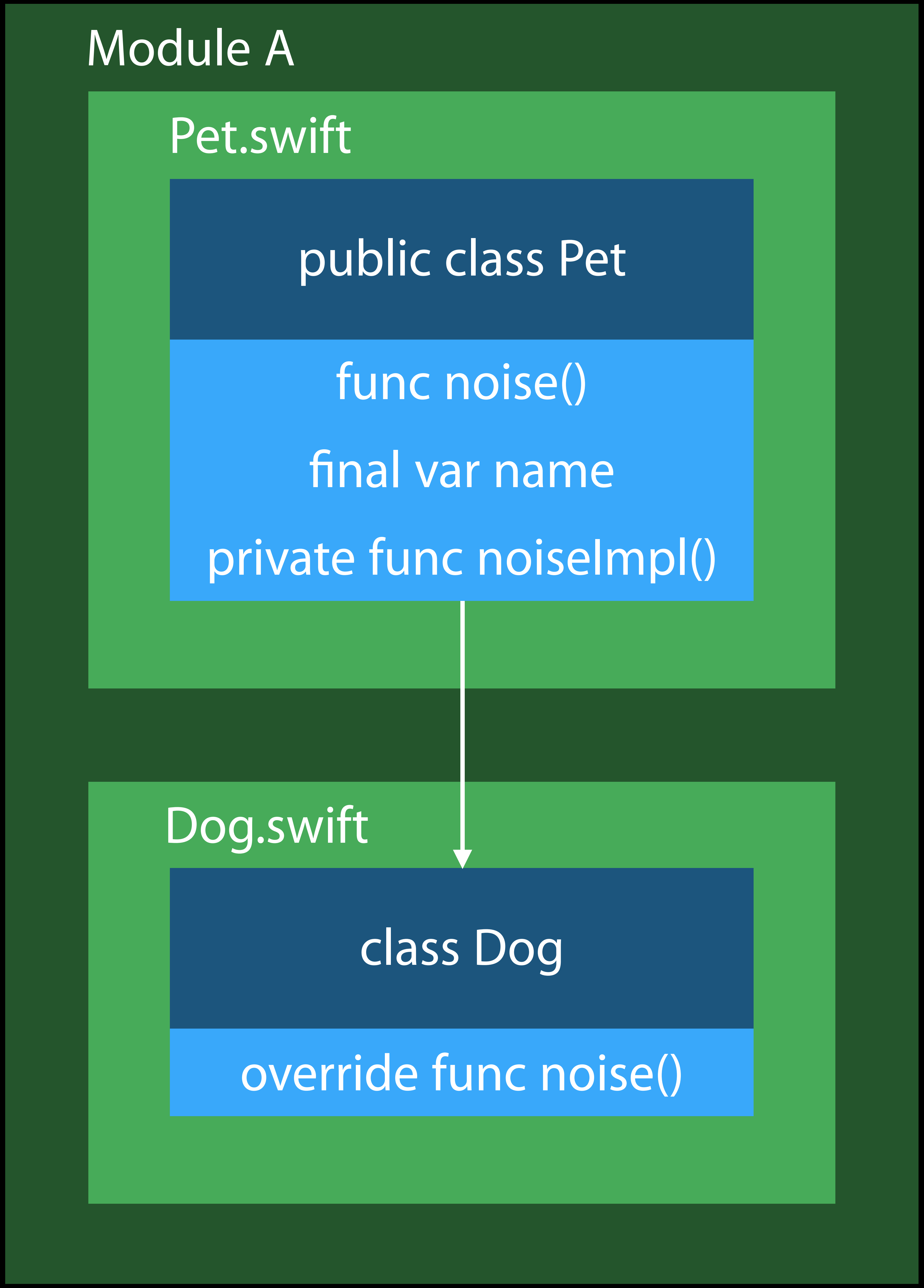
override func noise()

Whole Module Optimization



```
func bark(d: Dog) {
    d.noise()
}
```

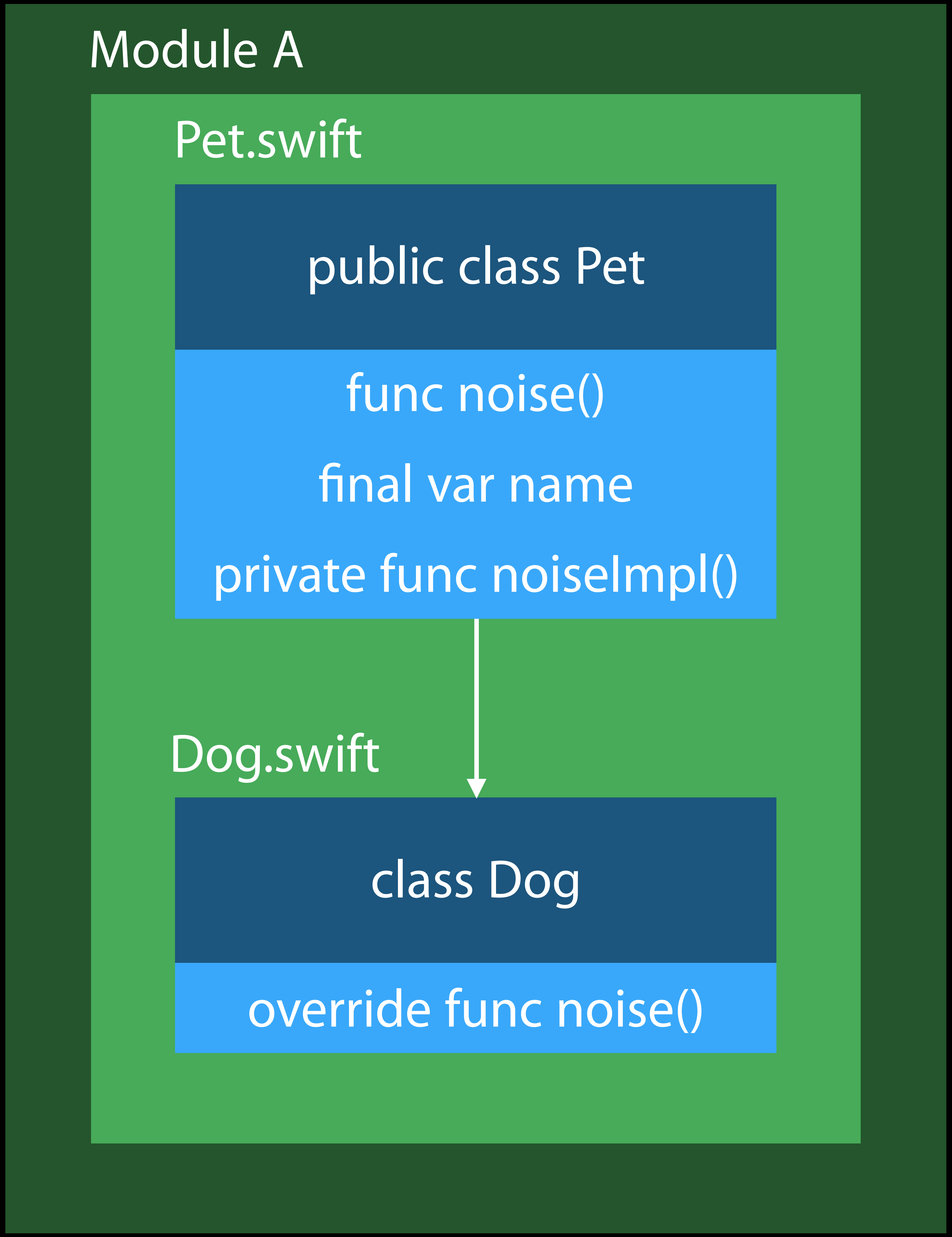
Whole Module Optimization



```
func bark(d: Dog) {
    d.noise()
}
```

```
func bark(d: Dog) {
    let noiseMethod = Dog.getNoiseMethod()
    noiseMethod(d)
}
```

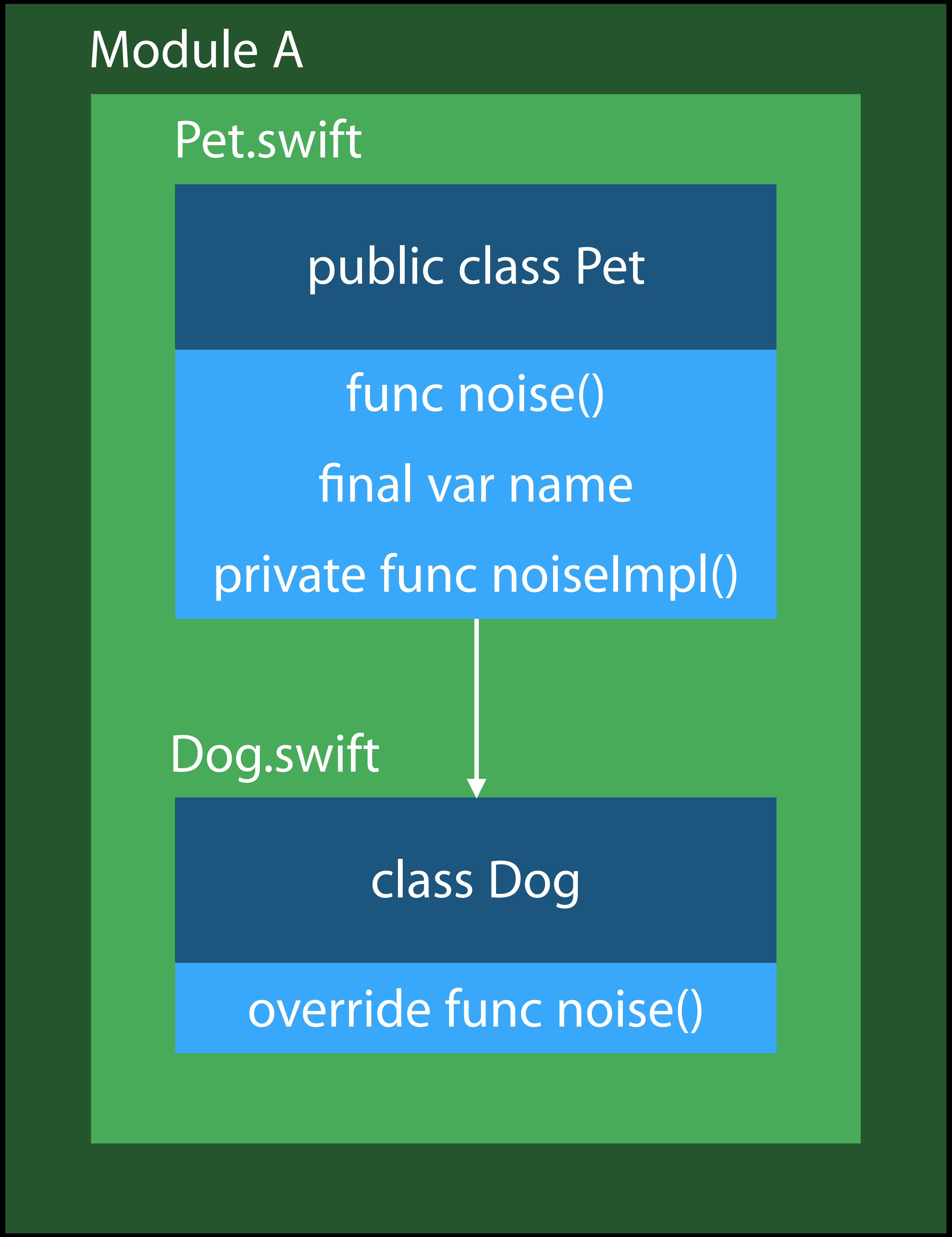
Whole Module Optimization



```
func bark(d: Dog) {
    d.noise()
}
```

```
func bark(d: Dog) {
    let noiseMethod = Dog.getNoiseMethod()
    noiseMethod(d)
}
```

Whole Module Optimization



Module A

Pet.swift

public class Pet

func noise()

final var name

private func noiseImpl()

Dog.swift

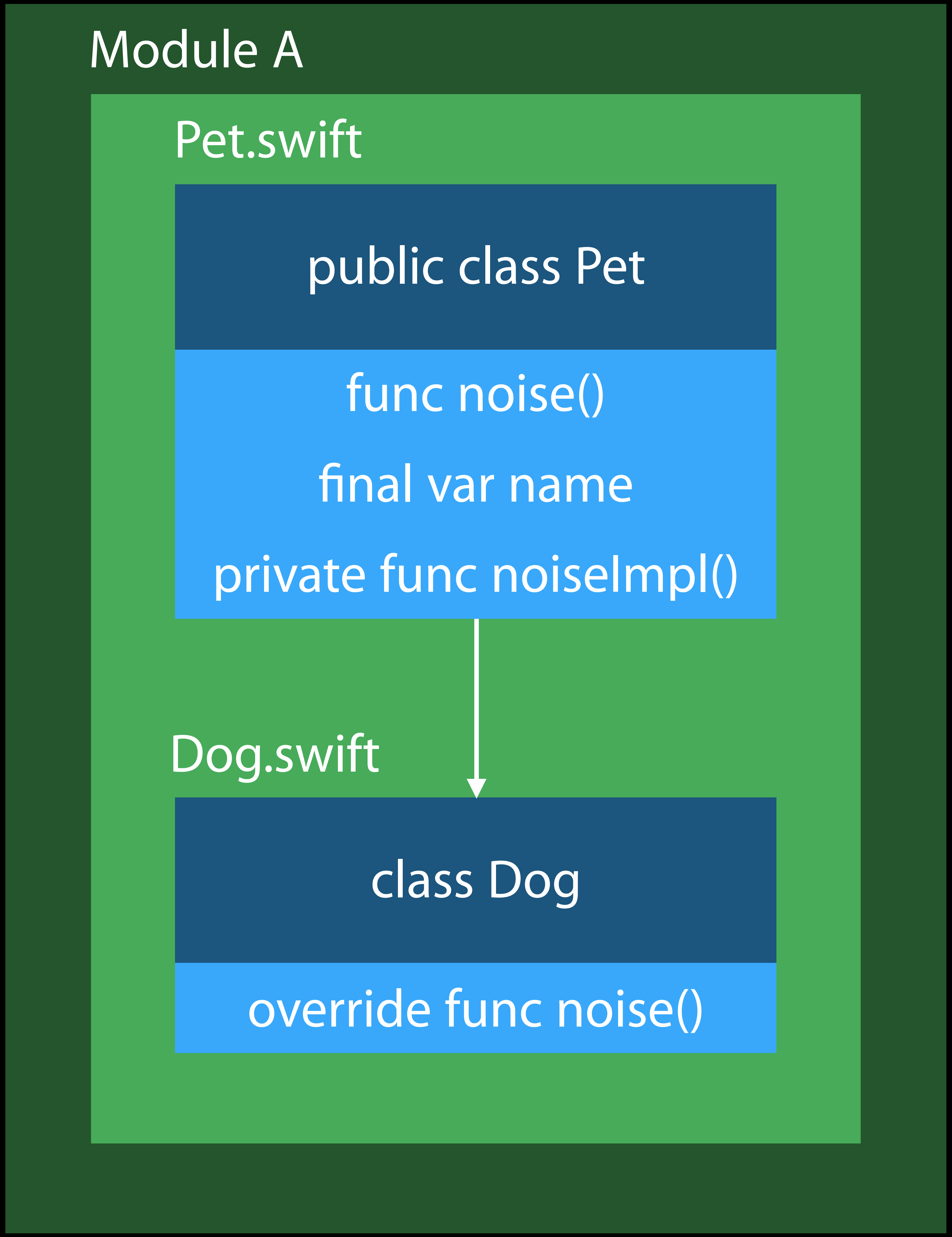
class Dog

override func noise()

```
func bark(d: Dog) {
    d.noise()
}
```

```
func bark(d: Dog) {

}
```

Whole Module Optimization

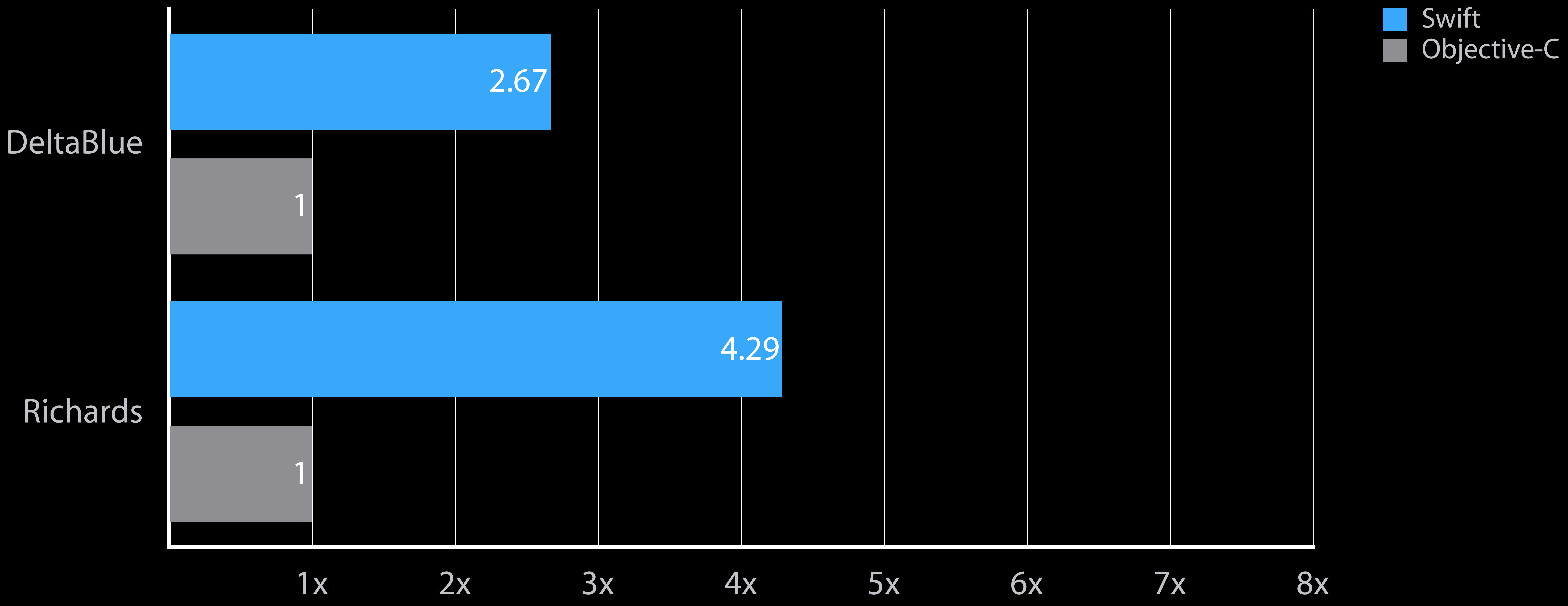


```
func bark(d: Dog) {
    d.noise()
}
```

```
func bark(d: Dog) {
    d.noise()
}
```

Swift vs. Objective-C

Program speed (higher is better)



Communicate your API Intent

Use the final keyword and access control

- Help the compiler understand your class hierarchy
- Be aware of breaking existing clients

Enable Whole Module Optimization

Demo

Joe Grzywacz

Engineer, Performance Tools

Summary

Swift is a flexible, safe programming language with ARC

Write your APIs and code with performance in mind

Profile your application with Instruments

More Information

Swift Language Documentation

http://developer.apple.com/swift

Apple Developer Forums

http://developer.apple.com/forums

Stefan Lesser
Developer Tools Evangelist
slesser@apple.com

Related Sessions

Profiling in Depth	Mission	Thursday 3:30PM
Building Better Apps with Value Types in Swift	Mission	Friday 2:30PM

WWDC15