



1

Who is Ray Fix?

Source: https://en.wikipedia.org/wiki/La_Jolla

2

RayWenderlich.com

3

ECHO
LABORATORIES

iosdc.jp Ray Fix

4-1



ECHO
LABORATORIES



4-2



5

Stack vs Heap

iosdc.jp Ray Fix

6

Stack

The stack is really fast!

iosdc.jp Ray Fix

7-1

Stack

The stack is really fast!

```
func candy() {  
    let ramune = 10  
    let pocky = 12  
    let gummi = 20  
}
```

iosdc.jp Ray Fix

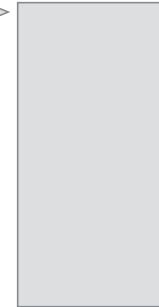
7-2

Stack

The stack is really fast!

```
func candy() {  
    let ramune = 10  
    let pocky = 12  
    let gummi = 20  
}
```

Stack



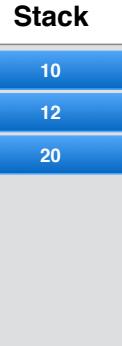
iosdc.jp Ray Fix

7-3

Stack

The stack is really fast!

```
func candy() {  
    let ramune = 10  
    let pocky = 12  
    let gummi = 20  
}
```



iosdc.jp Ray Fix

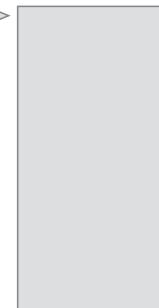
7-4

Stack

The stack is really fast!

```
func candy() {  
    let ramune = 10  
    let pocky = 12  
    let gummi = 20  
}
```

Stack



iosdc.jp Ray Fix

7-5

Stack

The stack is really fast!

```
func candy() {  
    let ramune = 10  
    let pocky = 12  
    let gummi = 20  
}
```

No need to use any special locks.



Stack

iosdc.jp Ray Fix

7-6

Heap

```
class Dish {  
  
    var name: String  
  
    init(name: String) {  
        self.name = name  
        print("⭐ Dish \(name)")  
    }  
  
    deinit {  
        print("☒ Dish \(name)")  
    }  
}
```

iosdc.jp Ray Fix

8

Heap

```
let softCream = Dish(name: "Soft Cream")
```

Heap

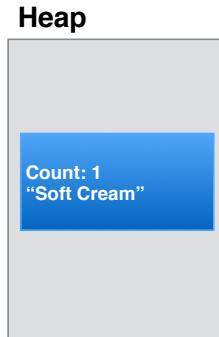
iosdc.jp Ray Fix

9-1

9-2

Heap

```
let softCream = Dish(name: "Soft Cream")
```

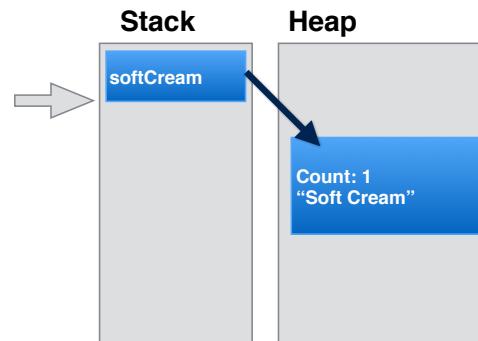


iosdc.jp Ray Fix

9-3

Heap

```
let softCream = Dish(name: "Soft Cream")
```

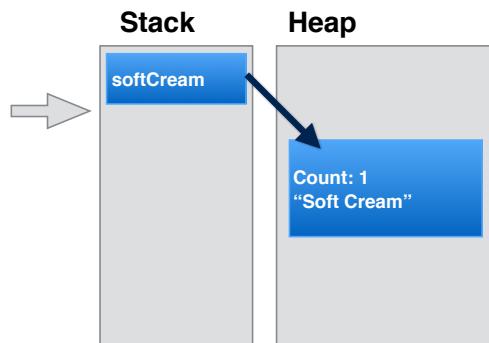


iosdc.jp Ray Fix

9-4

Heap

```
let softCream = Dish(name: "Soft Cream")
```



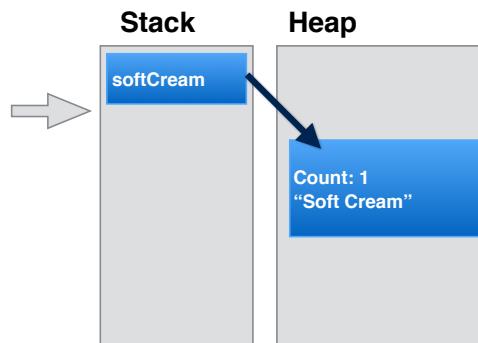
iosdc.jp Ray Fix

10-1

Heap

```
let softCream = Dish(name: "Soft Cream")
```

```
let special = softCream
```

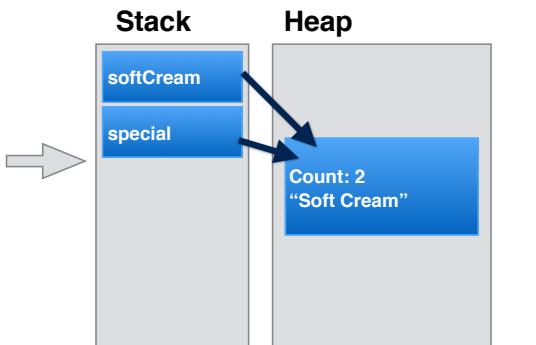


iosdc.jp Ray Fix

10-2

Heap

```
let softCream = Dish(name: "Soft Cream")
let special = softCream
```



10-3

Sharing

```
softCream.name = "Choco Soft Cream"
special.name // "Choco Soft Cream"
```

iosdc.jp Ray Fix

11-1

Sharing

```
softCream.name = "Choco Soft Cream"
special.name // "Choco Soft Cream"
```

Wow, this is super convenient!

iosdc.jp Ray Fix

11-2

Sharing

```
softCream.name = "Choco Soft Cream"
special.name // "Choco Soft Cream"
```

Wow, this is super convenient!

But...

iosdc.jp Ray Fix

11-3

Sharing

```
softCream.name = "Choco Soft Cream"  
special.name // "Choco Soft Cream"
```

Wow, this is super convenient!

But...

```
special.name = "Green Pepper"  
softCream.name // "Green Pepper"
```

iosdc.jp Ray Fix

11-4

Sharing

```
softCream.name = "Choco Soft Cream"  
special.name // "Choco Soft Cream"
```

Wow, this is super convenient!

But...

```
special.name = "Green Pepper"  
softCream.name // "Green Pepper"
```



Unintended sharing is a problem with reference types

iosdc.jp Ray Fix

11-5

Solution: Use a constant

```
class Dish {  
  
    var name: String  
  
    init(name: String) {  
        self.name = name  
        print("🌟 Dish \(name)")  
    }  
  
    deinit {  
        print("💔 Dish \(name)")  
    }  
}
```

iosdc.jp Ray Fix

12-1

Solution: Use a constant

```
class Dish {  
  
    var name: String  
  
    init(name: String) {  
        self.name = name  
        print("🌟 Dish \(name)")  
    }  
  
    deinit {  
        print("💔 Dish \(name)")  
    }  
}
```

iosdc.jp Ray Fix

12-2

Solution: Use a constant

```
class Dish {  
    let name: String  
  
    init(name: String) {  
        self.name = name  
        print("🐣 Dish \(name)")  
    }  
  
    deinit {  
        print("☠ Dish \(name)")  
    }  
}
```

iosdc.jp Ray Fix

12-3

Solution: Use a constant

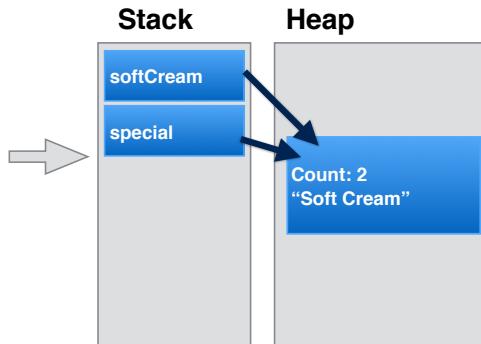
```
class Dish {  
    (X) special.name = "Blah"  
  
    let name: String  
  
    init(name: String) {  
        self.name = name  
        print("🐣 Dish \(name)")  
    }  
  
    deinit {  
        print("☠ Dish \(name)")  
    }  
}
```

iosdc.jp Ray Fix

12-4

Heap

```
let softCream = Dish(name: "Soft Cream")  
let special = softCream
```

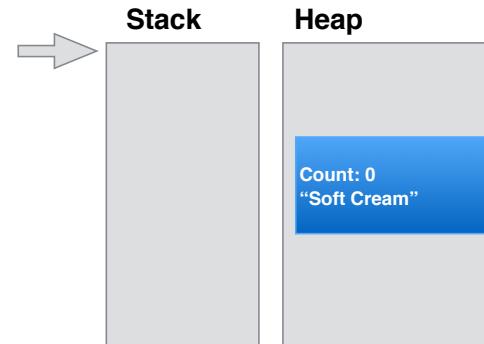


iosdc.jp Ray Fix

13-1

Heap

```
let softCream = Dish(name: "Soft Cream")  
let special = softCream
```

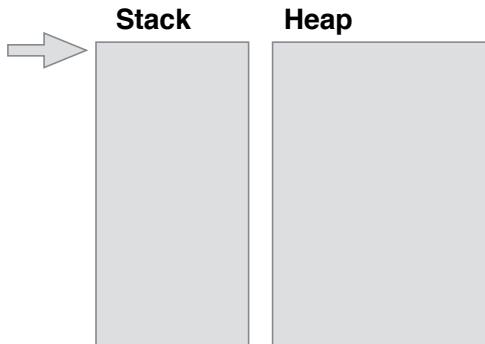


iosdc.jp Ray Fix

13-2

Heap

```
let softCream = Dish(name: "Soft Cream")
let special = softCream
```



13-3

Problem: Circular References



iosdc.jp Ray Fix

14

Reference Cycle

```
class Customer {
    var orders: [Order]

    func add(order: Order) {
        order.customer = self
        orders.append(order)
    }
}

class Order {
    var customer: Customer?
    let dish: Dish
}
```

iosdc.jp Ray Fix

15-1

Reference Cycle

```
class Customer {
    var orders: [Order]

    func add(order: Order) {
        order.customer = self
        orders.append(order)
    }
}

class Order {
    var customer: Customer?
    let dish: Dish
}
```

iosdc.jp Ray Fix

15-2

Reference Cycle

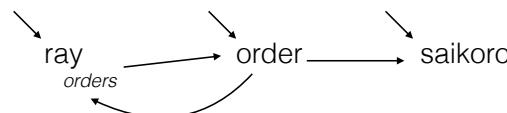
```
let ray = Customer(name: "Ray")
let saikoro = Dish(name: "Saikoro Steak")
let order = Order(dish: saikoro)
ray.add(order: order)
```

iosdc.jp Ray Fix

16-1

Reference Cycle

```
let ray = Customer(name: "Ray")
let saikoro = Dish(name: "Saikoro Steak")
let order = Order(dish: saikoro)
ray.add(order: order)
```



iosdc.jp Ray Fix

16-2

Reference Cycle

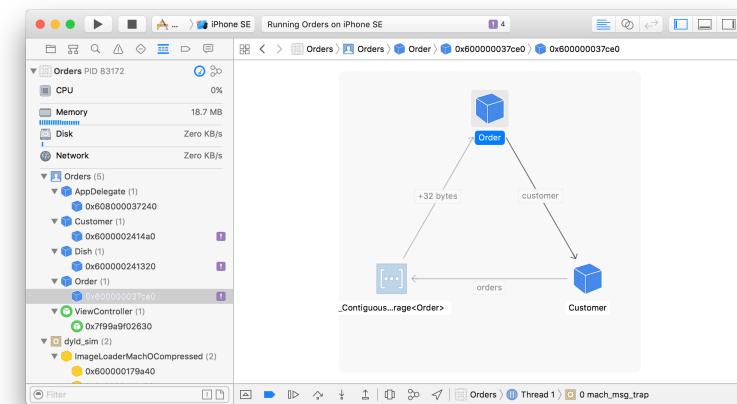
```
let ray = Customer(name: "Ray")
let saikoro = Dish(name: "Saikoro Steak")
let order = Order(dish: saikoro)
ray.add(order: order)
```



iosdc.jp Ray Fix

16-3

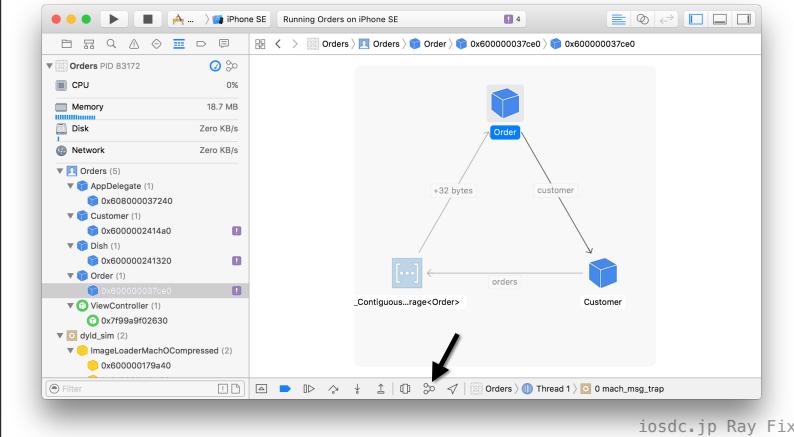
Xcode 8 Memory Visualizer



iosdc.jp Ray Fix

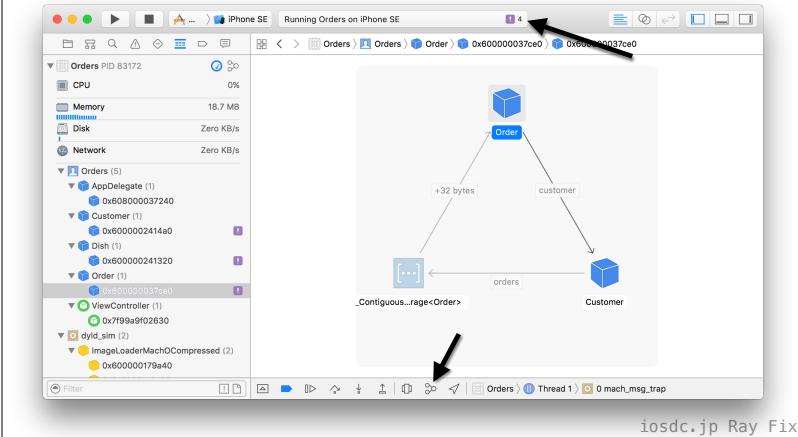
17-1

Xcode 8 Memory Visualizer



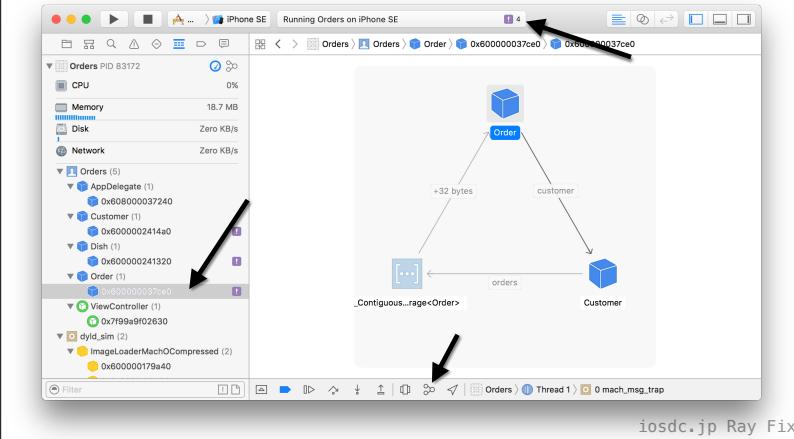
17-2

Xcode 8 Memory Visualizer



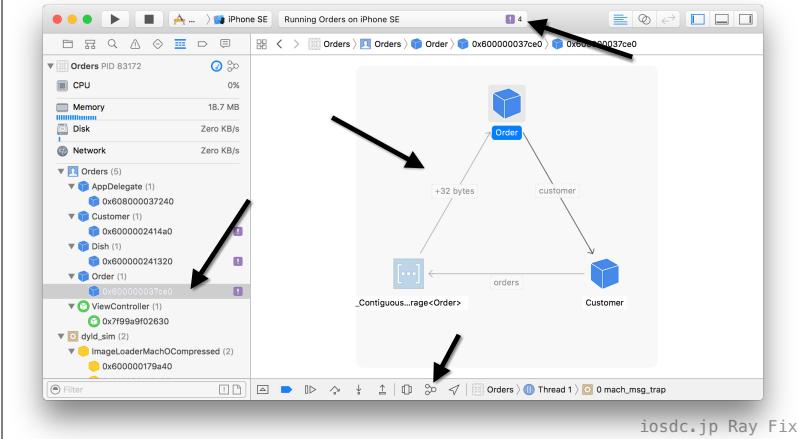
17-3

Xcode 8 Memory Visualizer



17-4

Xcode 8 Memory Visualizer



17-5

Reference Cycle



iosdc.jp Ray Fix

18

Reference Cycle

```
class Customer {  
    var orders: [Order]  
  
    func add(order: Order) {  
        order.customer = self  
        orders.append(order)  
    }  
}  
  
class Order {  
    var customer: Customer?  
    let dish: Dish  
}
```

iosdc.jp Ray Fix

19-1

Reference Cycle

```
class Customer {  
    var orders: [Order]  
  
    func add(order: Order) {  
        order.customer = self  
        orders.append(order)  
    }  
  
class Order {  
    weak var customer: Customer?  
    let dish: Dish  
}
```

iosdc.jp Ray Fix

19-2

Reference Cycle

The diagram shows two versions of the code. The top version is identical to slide 19-1. The bottom version shows a fix where the 'customer' variable in the Order class is declared as 'weak'. A solid arrow points from the Customer class to the Order class, and a dotted arrow points from the Order class back to the Customer class, forming a cycle. A curved arrow points from the original code to the 'weak' declaration in the Order class.

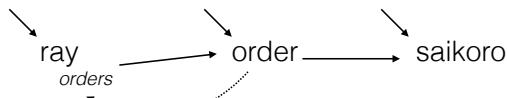
```
class Customer {  
    var orders: [Order]  
  
    func add(order: Order) {  
        order.customer = self  
        orders.append(order)  
    }  
  
class Order {  
    weak var customer: Customer?  
    let dish: Dish  
}
```

iosdc.jp Ray Fix

19-3

Reference Cycle

```
let ray = Customer(name: "Ray")
let saikoro = Dish(name: "Saikoro")
let order = Order(dish: saikoro)
ray.add(order: order)
```



iosdc.jp Ray Fix

20-1

Reference Cycle

```
let ray = Customer(name: "Ray")
let saikoro = Dish(name: "Saikoro")
let order = Order(dish: saikoro)
ray.add(order: order)
```



iosdc.jp Ray Fix

20-2

Reference Cycle

```
let ray = Customer(name: "Ray")
let saikoro = Dish(name: "Saikoro")
let order = Order(dish: saikoro)
ray.add(order: order)
```



iosdc.jp Ray Fix

20-3

Reference Cycle

```
let ray = Customer(name: "Ray")
let saikoro = Dish(name: "Saikoro")
let order = Order(dish: saikoro)
ray.add(order: order)
```

saikoro

iosdc.jp Ray Fix

20-4

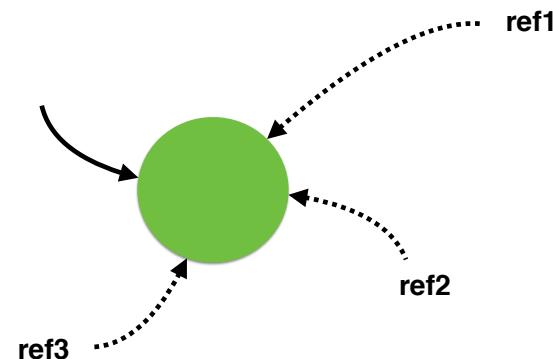
Reference Cycle

```
let ray = Customer(name: "Ray")
let saikoro = Dish(name: "Saikoro")
let order = Order(dish: saikoro)
ray.add(order: order)
```

iosdc.jp Ray Fix

20-5

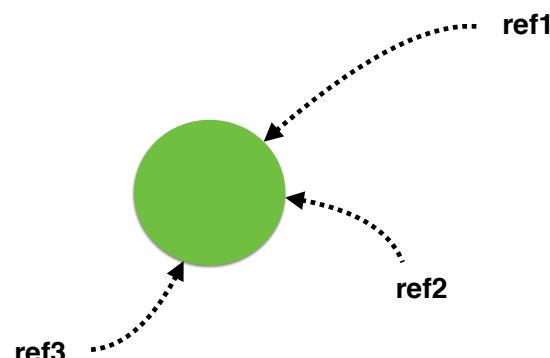
weak



iosdc.jp Ray Fix

21-1

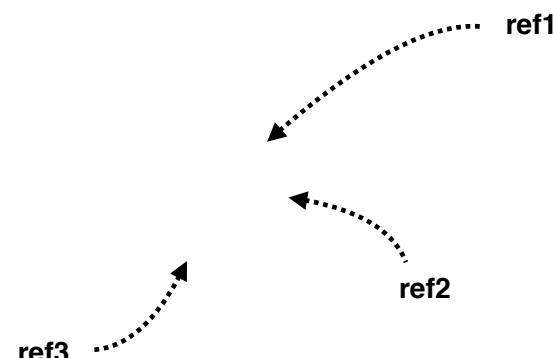
weak



iosdc.jp Ray Fix

21-2

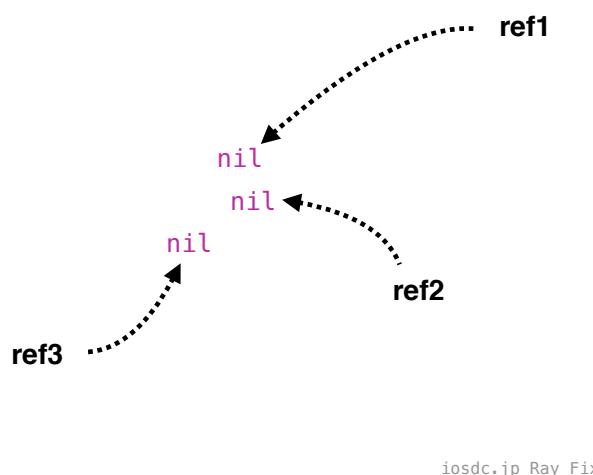
weak



iosdc.jp Ray Fix

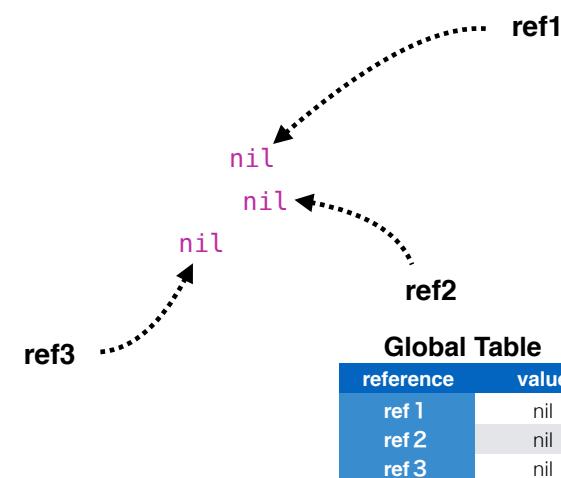
21-3

weak



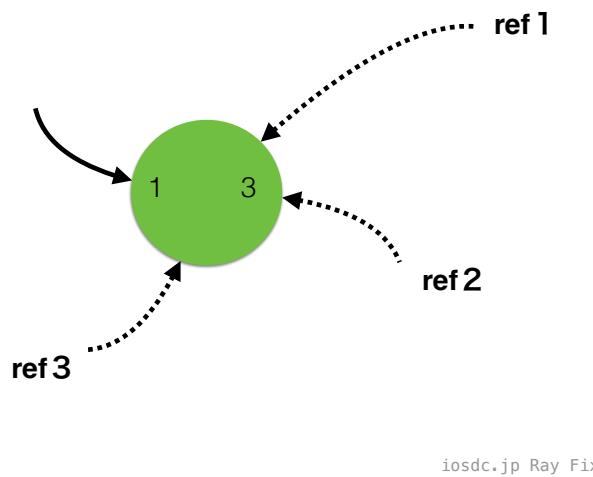
21-4

weak



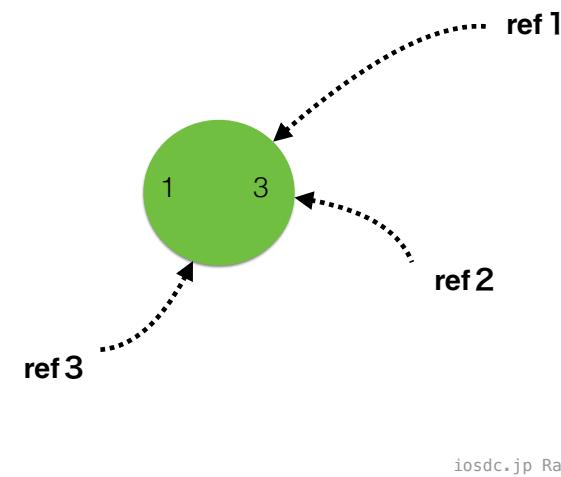
21-5

Swift weak



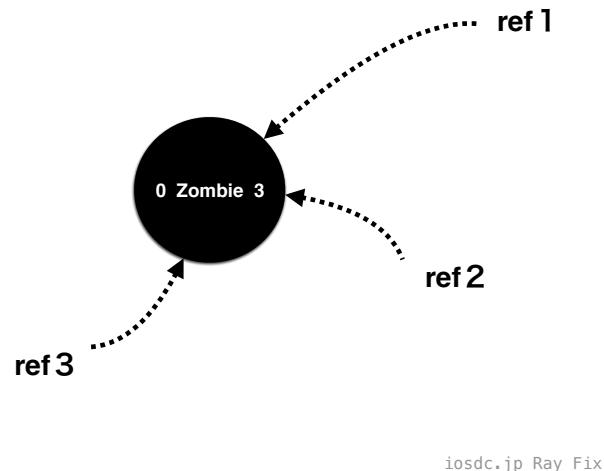
22-1

Swift weak



22-2

Swift weak



22-3

When the strong count goes to zero



23-1

When the strong count goes to zero



23-2

When the weak count goes to zero



24-1

When the weak count goes to zero



24-2

unowned

- unowned is another type of weak reference
- unowned must always point to something
- if it doesn't, your program will stop on access

25

unowned

```
class Customer {  
    var orders: [Order]  
  
    func add(dish: Dish) {  
        let order = Order(dish: dish, customer: self)  
        orders.append(order)  
    }  
  
    class Order {  
        let customer: Customer  
        let dish: Dish  
    }  
}
```

26-1

unowned

```
class Customer {  
    var orders: [Order]  
  
    func add(dish: Dish) {  
        let order = Order(dish: dish, customer: self)  
        orders.append(order)  
    }  
  
    class Order {  
        let customer: Customer  
        let dish: Dish  
    }  
}
```

26-2

unowned

```
class Customer {  
    var orders: [Order]  
  
    func add(dish: Dish) {  
        let order = Order(dish: dish, customer: self)  
        orders.append(order)  
    }  
  
    class Order {  
        unowned let customer: Customer  
        let dish: Dish  
    }  
}
```

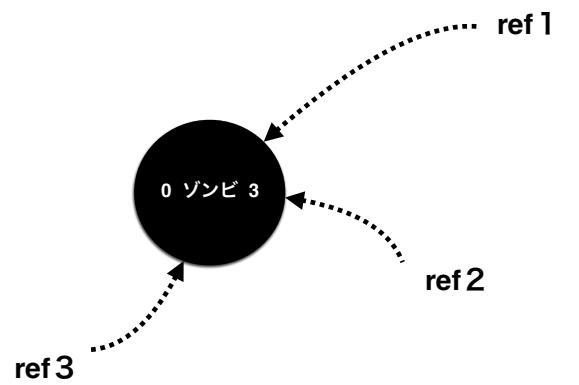
26-3

unowned

```
class Customer {  
    var orders: [Order]  
  
    func add(dish: Dish) {  
        let order = Order(dish: dish, customer: self)  
        orders.append(order)  
    }  
  
    class Order {  
        unowned let customer: Customer  
        let dish: Dish  
    }  
}
```

26-4

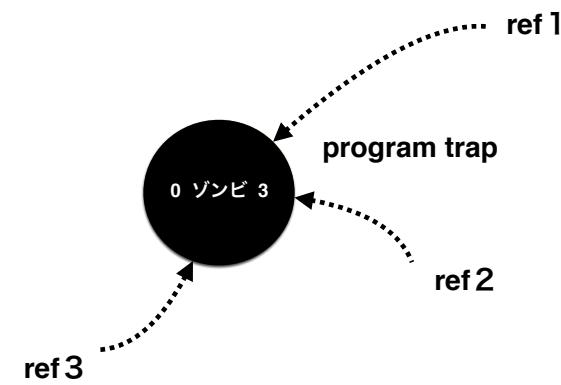
unowned



iosdc.jp Ray Fix

27-1

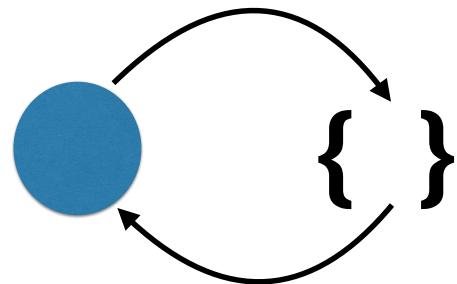
unowned



iosdc.jp Ray Fix

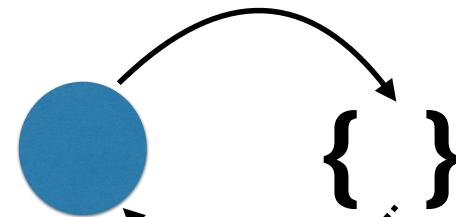
27-2

Functions and closures are reference types



28-1

Functions and closures are reference types



28-2

Example: Sushi Shop

```
enum MenuItem: String {  
    case toro, ebi, anago, uni, ikura, hamachi  
}  
  
typealias Action = ()->()
```

29

Example: Sushi Shop

```
class Sushiya {  
    lazy var menu: [MenuItem: Action] = [...]  
  
    func prepare(_ menuItem: MenuItem) {  
        menu[menuItem] ?()  
    }  
  
    private func serve(dish: Dish) {  
        print("Now serving \(dish.name)")  
    }  
}
```

30

Example: Sushi Shop

```
menu =  
[  
  .toro: {  
    let dish = Dish(name: "Toro")  
    serve(dish: dish)  
  },  
  
  ...  
]
```

31-1

Example: Sushi Shop

```
menu =  
[  
  .toro: {  
    let dish = Dish(name: "Toro")  
    self.serve(dish: dish)  
  },  
  
  ...  
]
```

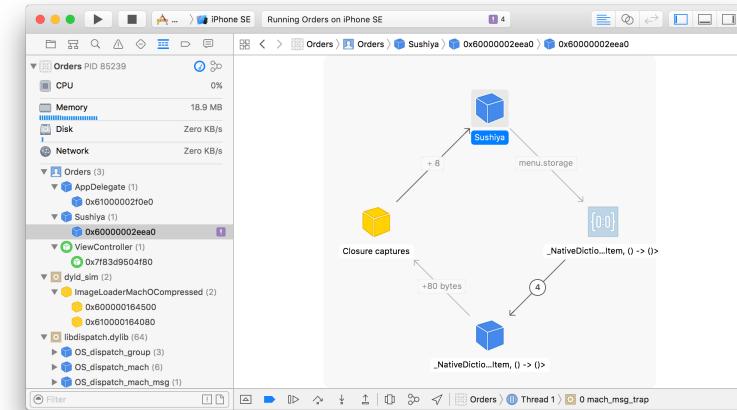
31-2

Example: Sushi Shop

```
menu =  
[  
  .toro: {  
    let dish = Dish(name: "Toro")  
    self.serve(dish: dish)  
  },  
  
  ...  
]  
  
let sushiya = Sushiya()  
sushiya.prepare(.toro)
```

31-3

Memory Leak



32

Solution: Capture List

```
var value = 0  
  
let showValue = {  
    print(value)  
}
```

33-1

Solution: Capture List

```
var value = 0  
  
let showValue = {  
    print(value)  
}  
  
showValue() // prints 0  
value = 10  
showValue() // prints 10
```

33-2

Solution: Capture List

```
var value = 0  
  
let showValue = {  
    print(value)  
}  
  
showValue() // prints 0  
value = 10  
showValue() // prints 10
```

34-1

Solution: Capture List

```
var value = 0  
  
let showValue = { [value] in  
    print(value)  
}  
  
showValue() // prints 0  
value = 10  
showValue() // prints 10
```

34-2

Solution: Capture List

```
var value = 0

let showValue = { [value] in
    print(value)
}

showValue() // prints 0
value = 10
showValue() // prints 0
```

34-3

Solution: Capture List

```
menu =
[
    .toro: {
        let dish = Dish(name: "Toro")
        self.serve(dish: dish)
    },
    ...
]
```

35-1

Solution: Capture List

```
menu =
[
    .toro: { [unowned self] in
        let dish = Dish(name: "Toro")
        self.serve(dish: dish)
    },
    ...
]
```



35-2

Async Problem

```
private func serve(dish: Dish) {
    print("Now serving \(dish.name)")
}
```

36-1

Async Problem

```
private func serve(dish: Dish) {  
    DispatchQueue.main.async {  
        print("Now serving \(dish.name)")  
        self.served += 1  
    }  
}
```

36-2

Async Problem

```
private func serve(dish: Dish) {  
    DispatchQueue.main.async { [unowned self] in  
        print("Now serving \(dish.name)")  
        self.served += 1  
    }  
}
```

36-3

Async Problem

```
private func serve(dish: Dish) {  
    DispatchQueue.main.async { [unowned self] in  
        print("Now serving \(dish.name)")  
        self.served += 1  
    }  
}
```

✖ CRASH!!!!

36-4

Async Problem

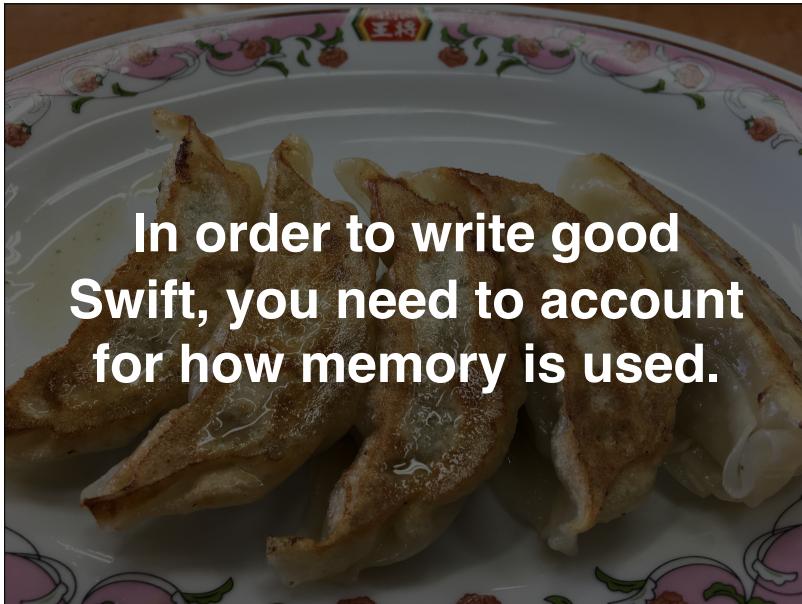
```
private func serve(dish: Dish) {  
    DispatchQueue.main.async {  
        print("Now serving \(dish.name)")  
        self.served += 1  
    }  
}
```

36-5

weak

```
private func serve(dish: Dish) {  
    DispatchQueue.main.async { [weak self] in  
        print("Now serving \(dish.name)")  
        self?.served += 1  
    }  
}
```

37



In order to write good
Swift, you need to account
for how memory is used.

39

strong weak dance

```
private func serve(dish: Dish) {  
    DispatchQueue.main.async { [weak self] in  
        guard let strongSelf = self else {  
            print("Cancelled \(dish.name)")  
            return  
        }  
        print("Now serving \(dish.name)")  
        strongSelf.served += 1  
    }  
}
```

38

Thank you

40

Links

Ray Wenderlich Memory Tutorial (by Maxime Defauw) Scheduled for release mid-September

<http://raywenderlich.com/>

ECHO LABS Microscope

<http://echo-labs.com/>

Swift Weak References Blog Post

<https://www.mikeash.com/pyblog/friday-qa-2015-12-11-swift-weak-references.html>

Japanese Linux meetup on glib malloc (Japanese language)

<https://www.youtube.com/watch?v=0-vWT-t0UHg>

Sushi picture

<https://ja.wikipedia.org/wiki/寿司>

WWDC Swift Performance

<wwdc2016/416/>

Picture of southern california

https://en.wikipedia.org/wiki/La_Jolla

Source code and slides

<https://github.com/rayfix/MemoryDish>

LINE Stamp by my friend

<http://bit.ly/etystamp>



Links

Ray Wenderlich Memory Tutorial (by Maxime Defauw) Scheduled for release mid-September

<http://raywenderlich.com/>

ECHO LABS Microscope

<http://echo-labs.com/>

Swift Weak References Blog Post

<https://www.mikeash.com/pyblog/friday-qa-2015-12-11-swift-weak-references.html>

Japanese Linux meetup on glib malloc (Japanese language)

<https://www.youtube.com/watch?v=0-vWT-t0UHg>

Sushi picture

<https://ja.wikipedia.org/wiki/寿司>

WWDC Swift Performance

<wwdc2016/416/>

Picture of southern california

https://en.wikipedia.org/wiki/La_Jolla

Source code and slides

<https://github.com/rayfix/MemoryDish>

LINE Stamp by my friend

<http://bit.ly/etystamp>



41-1

41-2