Memory Management Buffet

Ray Fix
@rayfix

画像提供 Yumi Kimura (CC2)

1

Who is Ray Fix?

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

2

RayWenderlich.com

3

ECHO LABORATORIES

4-1

ECHO
LABORATORIES

Stack vs Heap

Stack

The stack is really fast!

Stack

The stack is really fast!

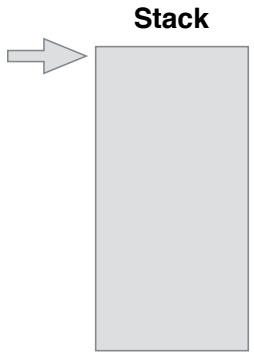
```
func candy() {

  let ramune = 10
  let pocky = 12
  let gummi = 20

}
```


7-2

Stack

The stack is really fast!

```
func candy() {

  let ramune = 10
  let pocky = 12
  let gummi = 20

}
```

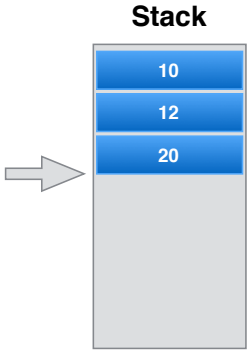


7-3

Stack

The stack is really fast!

```
func candy() {

  let ramune = 10
  let pocky = 12
  let gummi = 20

}
```

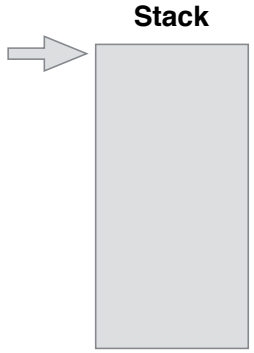


7-4

Stack

The stack is really fast!

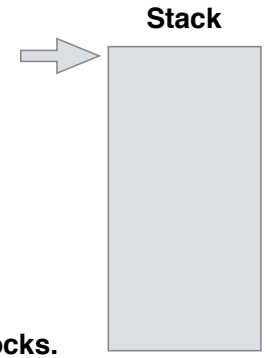
```
func candy() {

  let ramune = 10
  let pocky = 12
  let gummi = 20

}
```



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.

Heap

```
class Dish  {

    var name: String

    init(name: String) {
        self.name = name
        print("🍻 Dish \(name)")
    }

    deinit {
        print("☠️ Dish \(name)")
    }
}
```

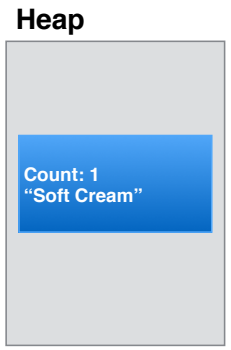

Heap

Heap

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

Heap

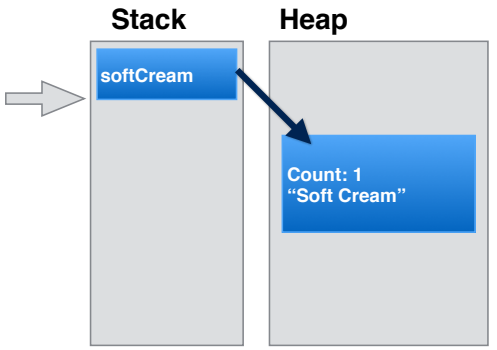
```
let softCream = Dish(name: “Soft Cream")
```



9-3

Heap

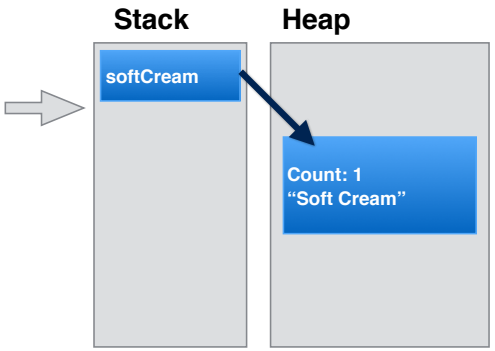
```
let softCream = Dish(name: “Soft Cream")
```



9-4

Heap

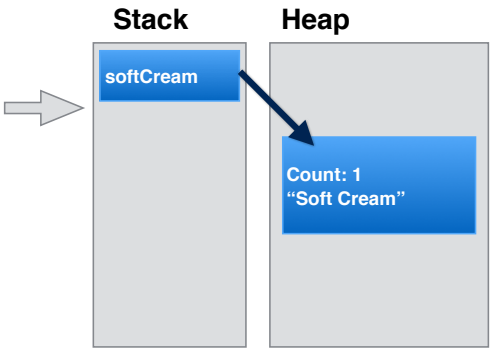
```
let softCream = Dish(name: “Soft Cream")
```



10-1

Heap

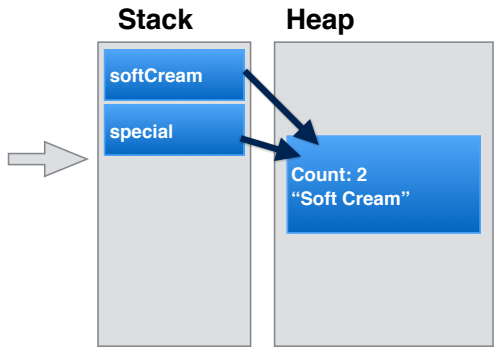
```
let softCream = Dish(name: “Soft Cream")
let special = softCream
```



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"
```


11-1

Sharing

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

Wow, this is super convenient!

11-2

Sharing

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

Wow, this is super convenient!

But…

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”
```


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

Solution: Use a constant

```
class Dish  {

    var name: String

    init(name: String) {
        self.name = name
        print("🐥 Dish \(name)")
    }

    deinit {
        print("☠ Dish \(name)")
    }
}
```


Solution: Use a constant

```
class Dish  {

    var name: String

    init(name: String) {
        self.name = name
        print("🐥 Dish \(name)")
    }

    deinit {
        print("☠ Dish \(name)")
    }
}
```

Solution: Use a constant

```
class Dish  {

    let name: String

    init(name: String) {
        self.name = name
        print("🍟 Dish \(name)")
    }

    deinit {
        print("☠️ Dish \(name)")
    }
}
```

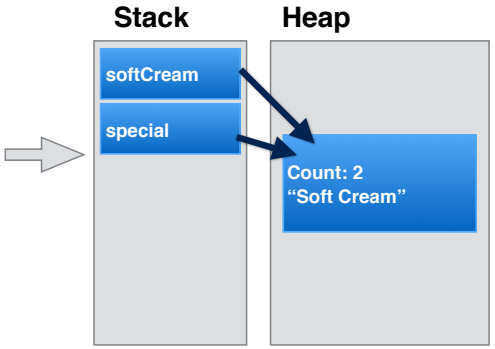

Solution: Use a constant

```
class Dish  {

    let name: String

    init(name: String) {
        self.name = name
        print("🍟 Dish \(name)")
    }

    deinit {
        print("☠️ Dish \(name)")
    }
}
```

⊗ `special.name = "Blah"`

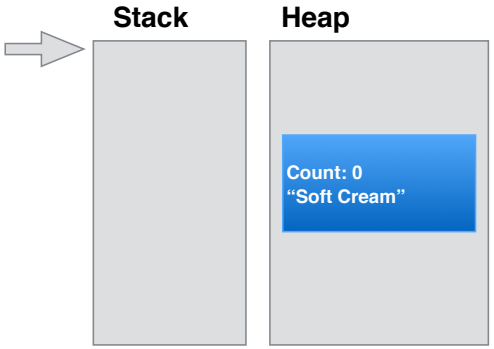
Heap

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



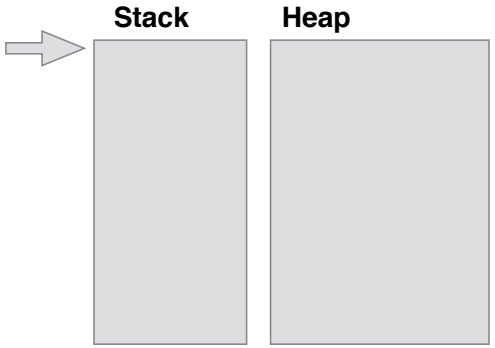
Heap

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



Heap

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



Problem: Circular References

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
}
```


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
}
```

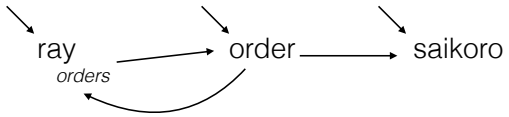
Reference Cycle

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


16-1

Reference Cycle

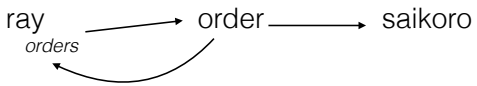
```
let ray = Customer(name: "Ray")
let saikoro = Dish(name: “Saikoro Steak")
let order = Order(dish: saikoro)
ray.add(order: order)
```



16-2

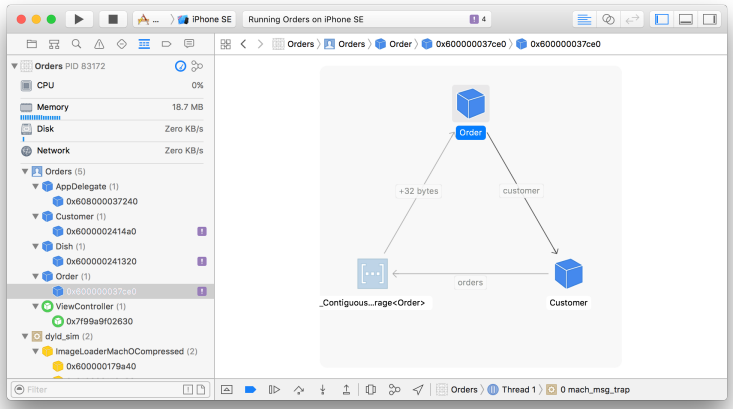
Reference Cycle

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



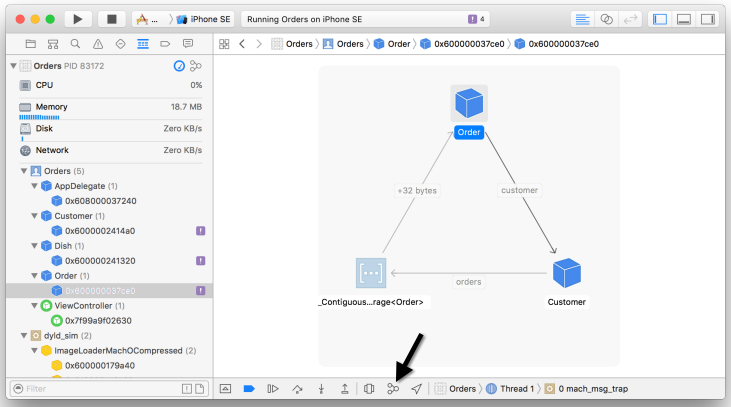
16-3

Xcode 8 Memory Visualizer



17-1

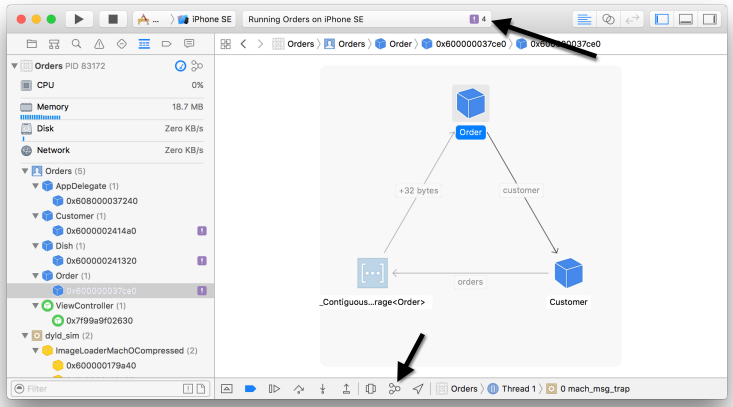
Xcode 8 Memory Visualizer



iosdc.jp Ray Fix

17-2

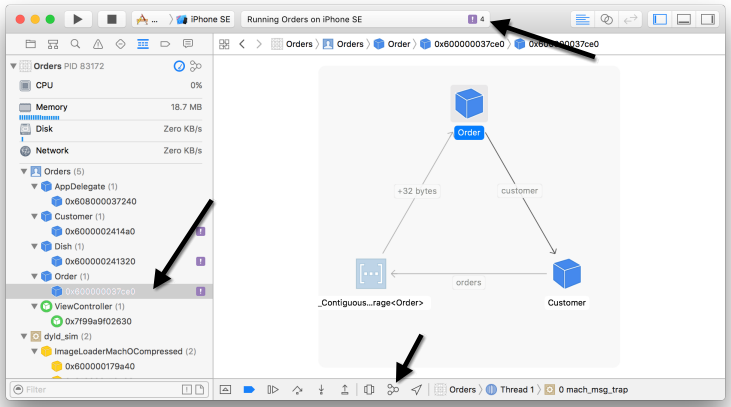
Xcode 8 Memory Visualizer



iosdc.jp Ray Fix

17-3

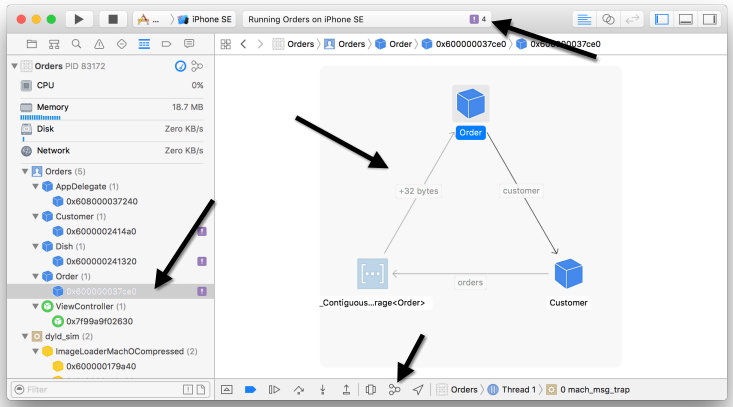
Xcode 8 Memory Visualizer



iosdc.jp Ray Fix

17-4

Xcode 8 Memory Visualizer



iosdc.jp Ray Fix

17-5

Reference Cycle

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
}
```


Reference Cycle

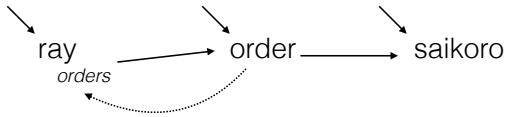
```
class Customer {
  var orders: [Order]

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


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


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
}
```

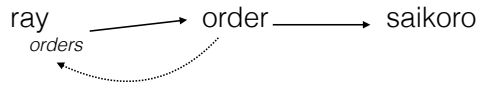
Reference Cycle

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



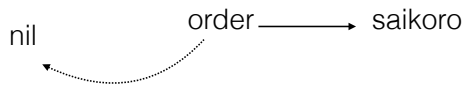
Reference Cycle

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



Reference Cycle

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



Reference Cycle

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

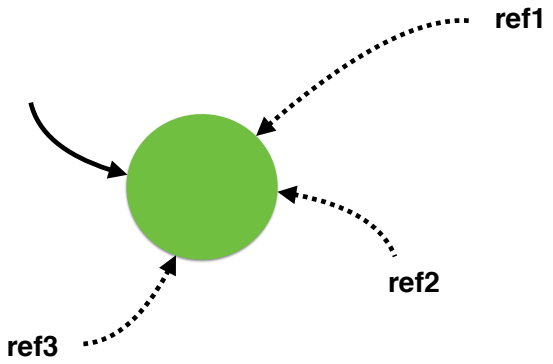
saikoro

Reference Cycle

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

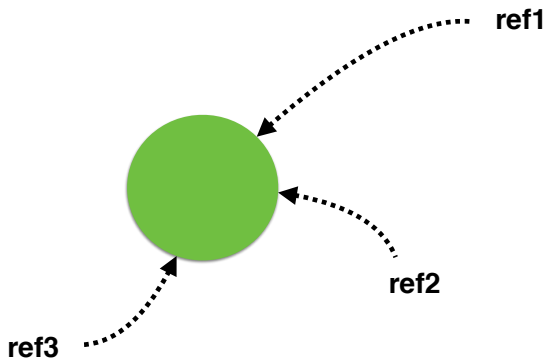

20-5

weak



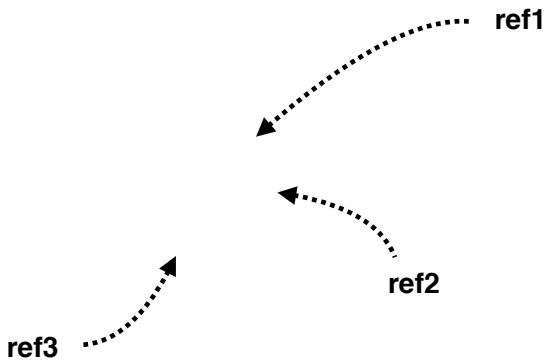
21-1

weak

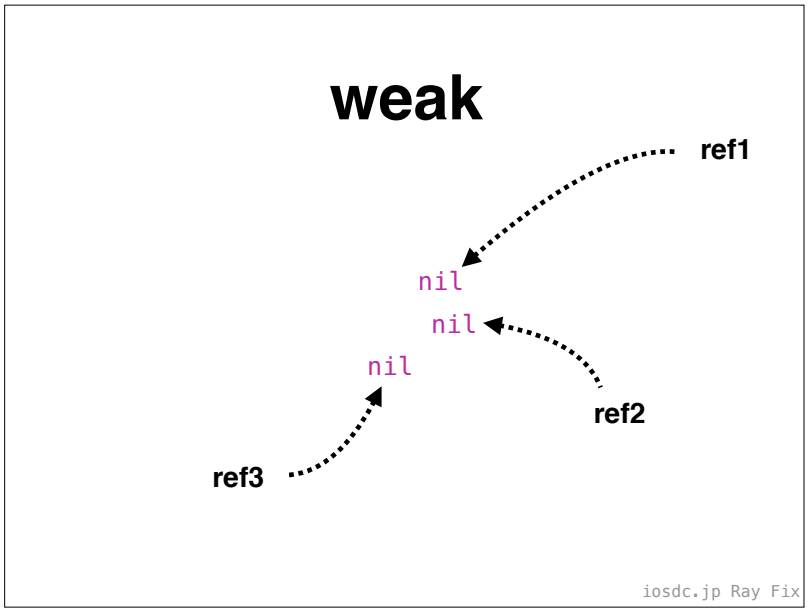


21-2

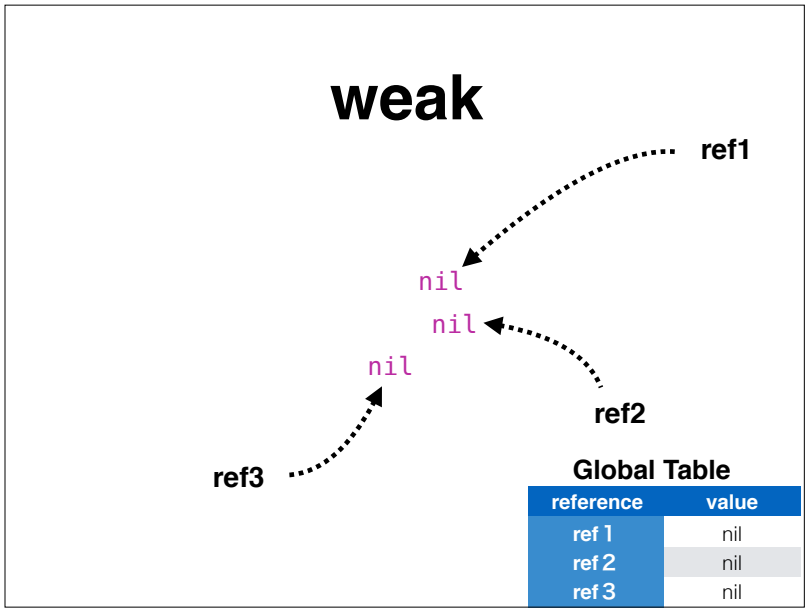
weak



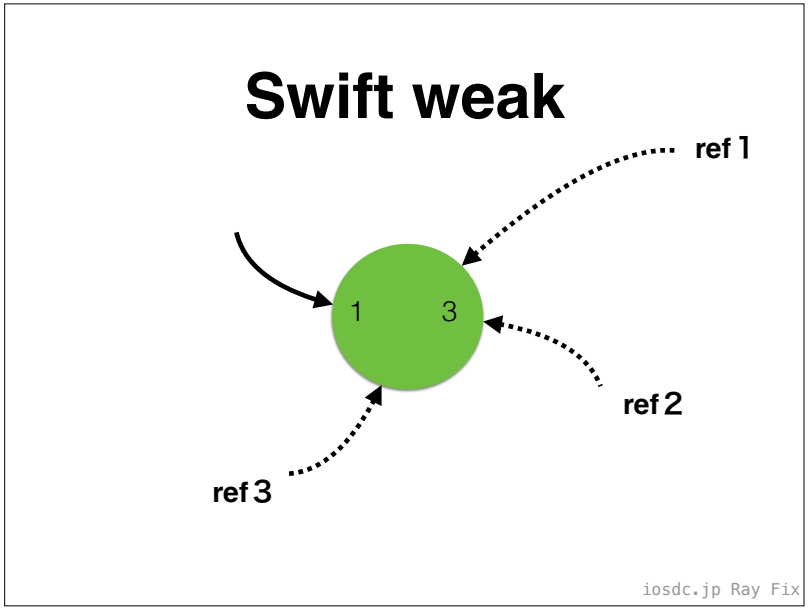
21-3



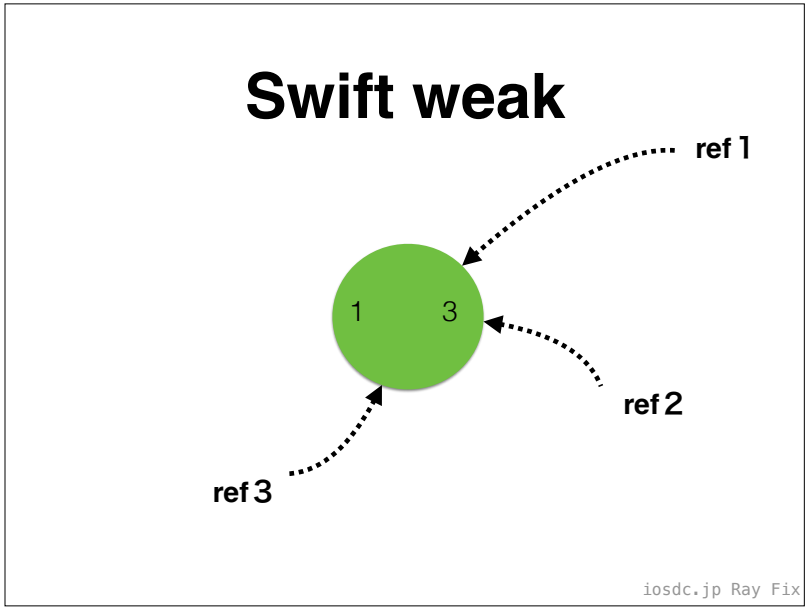
21-4



21-5

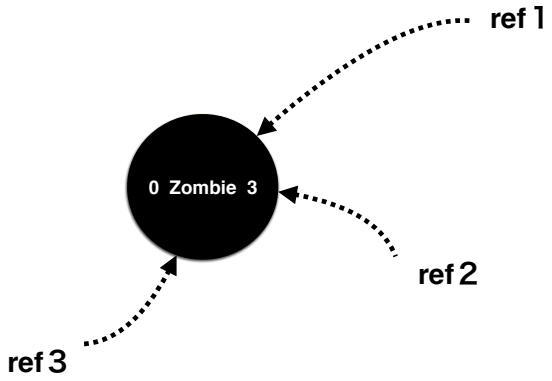


22-1



22-2

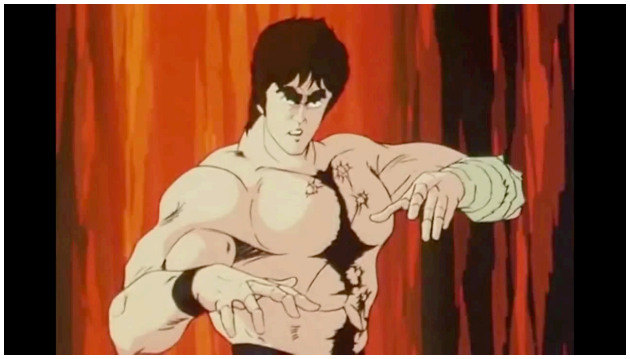
Swift weak



iosdc.jp Ray Fix

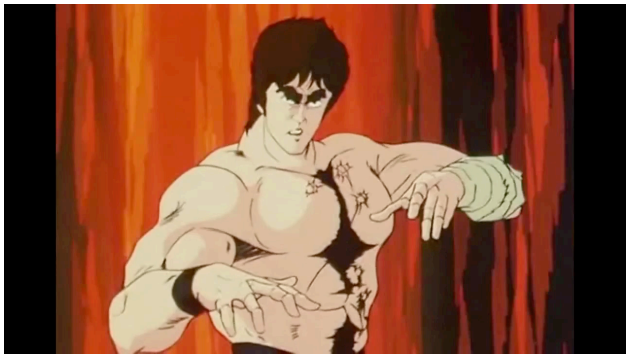
22-3

When the strong count goes to zero



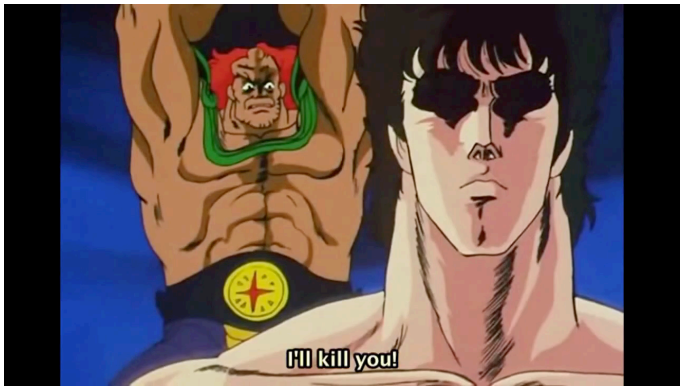
23-1

When the strong count goes to zero



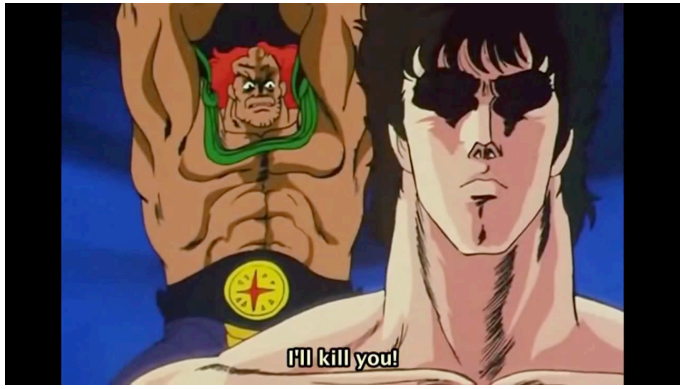
23-2

When the weak count goes to zero



24-1

When the weak count goes to zero



unowned

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

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
}
```


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
}
```

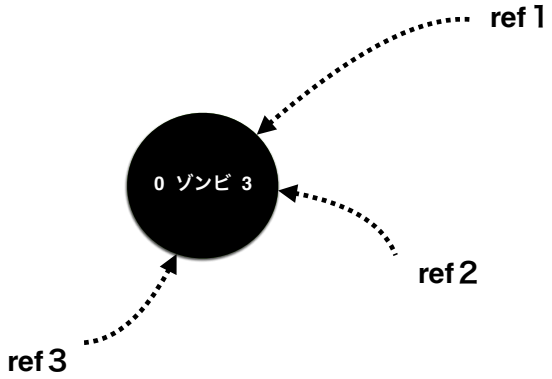
unowned

```
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
}
```

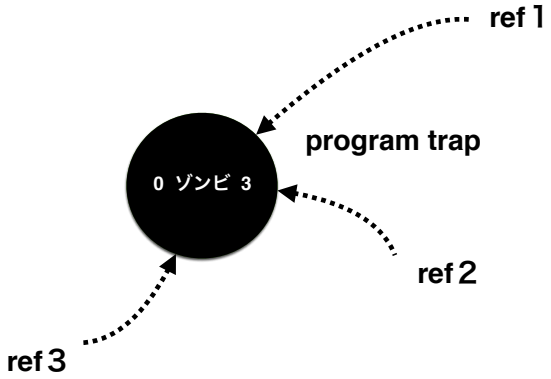

unowned

```
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
}
```

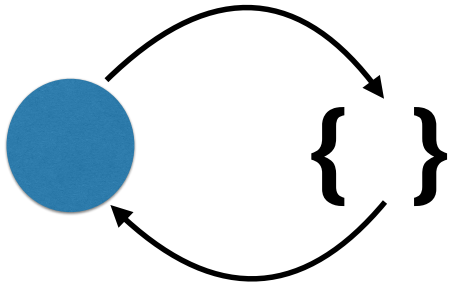

unowned



unowned

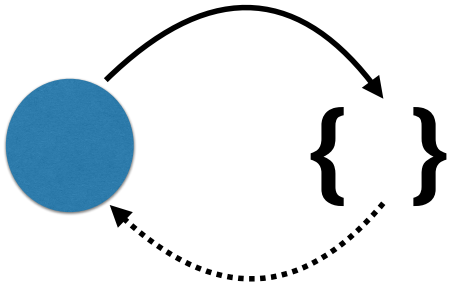


Functions and closures are reference types



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)
    },

    …
]
```

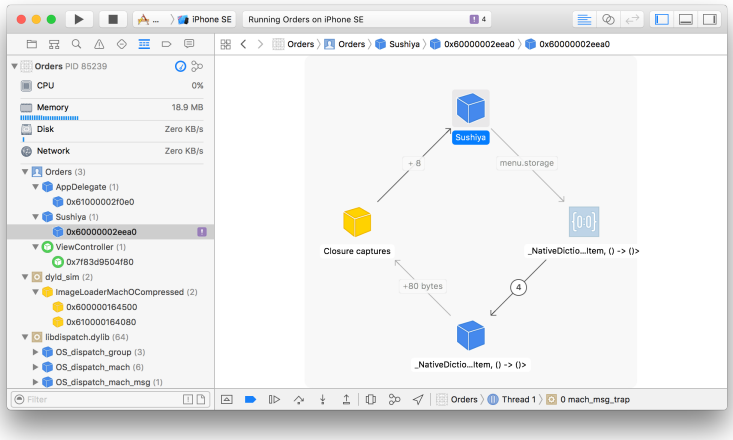

Example: Sushi Shop

```
menu =

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

    …
]
```


Example: Sushi Shop

```
menu =

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

    …
]
                let sushiya = Sushiya()
                sushiya.prepare(.toro)
```


Memory Leak



Solution: Capture List

```
var value = 0

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


Solution: Capture List

```
var value = 0

let showValue = {
  print(value)
}


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


Solution: Capture List

```
var value = 0

let showValue = {
  print(value)
}


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


Solution: Capture List

```
var value = 0

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


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

Solution: Capture List

```
var value = 0

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


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


Solution: Capture List

```
menu =

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

    …
]
```


Solution: Capture List

```
menu =

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

    …
]
```

Async Problem

```
private func serve(dish: Dish) {

   print("Now serving \(dish.name)")


}
```

Async Problem

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


Async Problem

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


Async Problem

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

⊗ **CRASH!!!!**

Async Problem

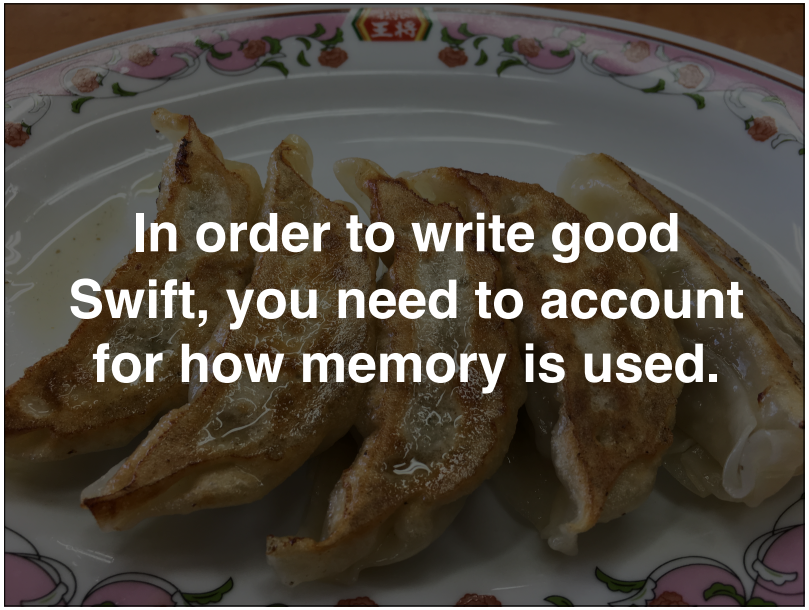
```
private func serve(dish: Dish) {
  DispatchQueue.main.async {
   print("Now serving \(dish.name)")
   self.served += 1
  }
}
```

weak

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


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
  }
}
```

Thank you

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

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

41-2