[WEAK SELF]

AND

WHY IT CAN BE MADE WRONG

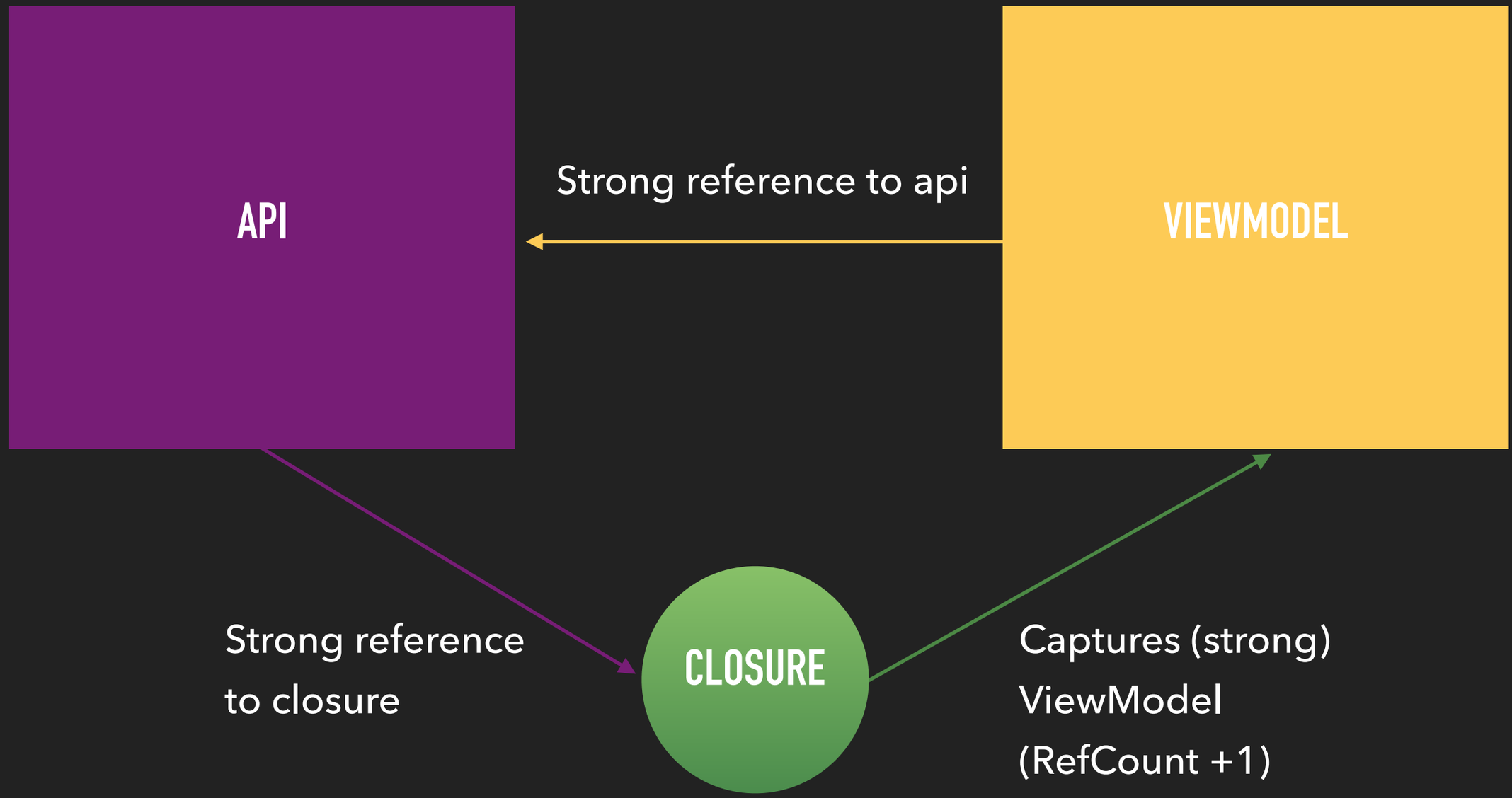
IOS DEVELOPERS ARE AFRAID OF RETAIN CYCLES & MEMORY LEAKS!

```
class ViewModel {
    let api = API()
}
```

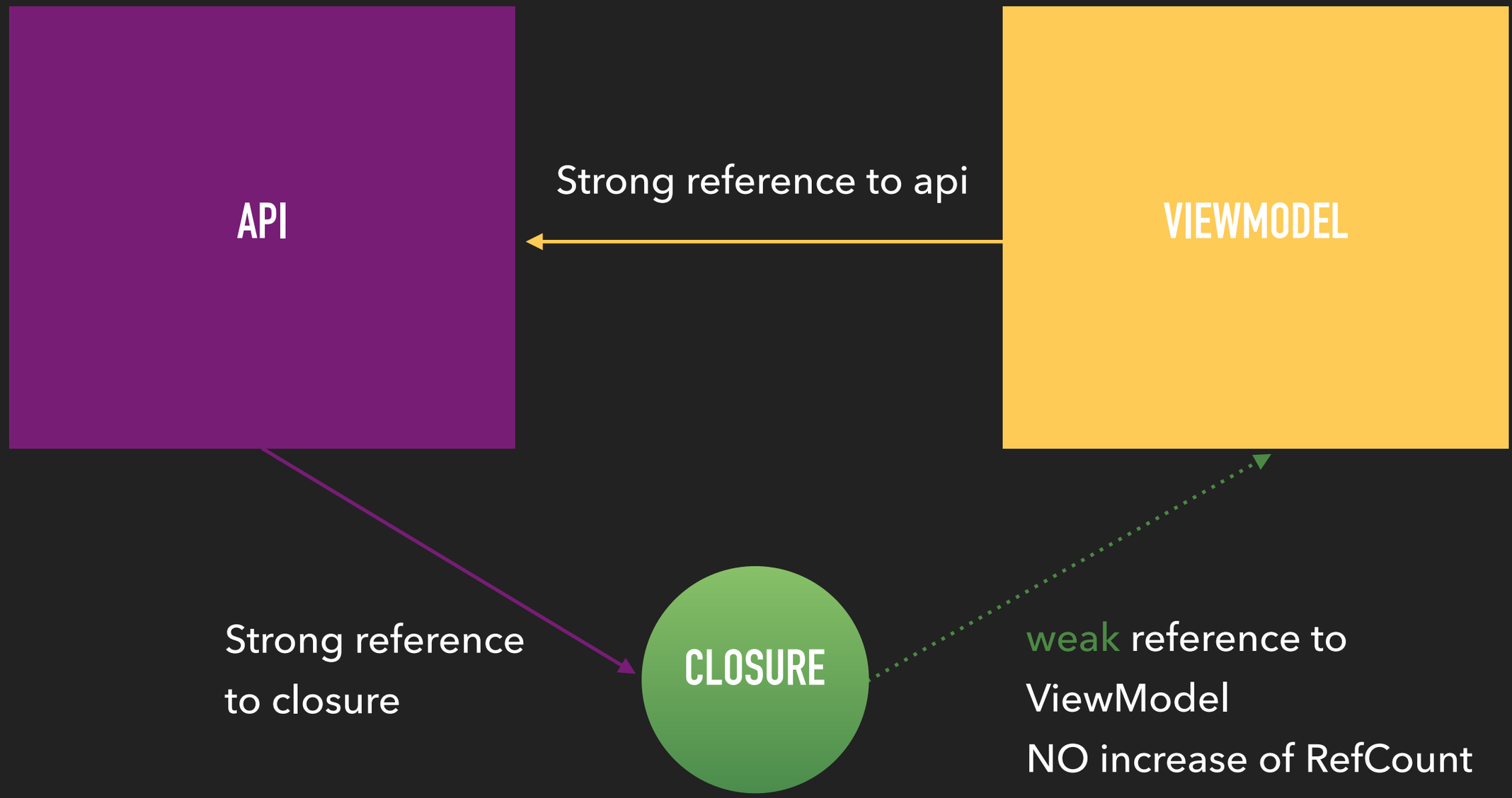
```
class ViewModel {
    let api = API()

    func update() {
        api.load {result in

        }
    }
}
```

```
class ViewModel {
    let api = API()

    func update() {
        api.load {result in
            self.log(result: result)
        }
    }

    func log(result: String) {
        print(result)
    }
}
```

RETAIN CYCLE



BREAK RETAIN CYCLE



```
class ViewModel {
    let api = API()

    func update() {
        api.load {[weak self] result in
            self?.log(result: result)
        }
    }
    func log(result: String) {
        print(result)
    }
}
```

Capture List

Breaks the retain cycle

FIXED?!

NEW BUGS?

DEMO

```
[weak self]  &  self?.
```

UNPREDICTABLE

```
class ViewModel {
    let api = API()

    func update() {
        api.load {[weak self] result in
            guard let self = self else {return}
            self.log(result: result)
        }
    }

    func log(result: String) {
        print(result)
    }
}
```

BECAUSE
IOS DEVELOPERS
ARE SO AFRAID

[weak self]

AT ALL PLACES ?

```swift
UIView.animate(withDuration: 0.1) {[weak self] in
            self?.log(result: "animating")
        }
```

```
UIAlertAction(title: "Error",
                      style: .default) {[weak self] action in
                      self?.log(result: “action handled”)
        }
```

```
DispatchQueue.main.async {[weak self] in
    self?.log(result: “run on main thread”)
}
```

DON'T USE [WEAK SELF], IF . . .

- Closure is non escaping (default)
- Closure doesn't capture anything that creates a retain cycle
- UIView Animations
- DispatchQueues

DON'T USE [WEAK SELF], IF . . .

Reference count of Object (API)

with associated closure will be decreased

before the function returns

```
func update() {
    API().load {result in
        self.log(result: result)
    }
}
```

DON'T USE [WEAK SELF], IF...

If singleton nils out the completion closure

```
func update() {
      API.shared.load {result in
          self.log(result: result)
      }
}
```

WHEN IS

[weak self]

NECESSARY?

NEEDED WHEN YOU CREATE A RETAIN CYCLE

```
func update() {
    closure = {[weak self] in
        guard let self = self else { return }
        self.log()
    }
}
```

USE [WEAK SELF]

NEEDED WHEN YOU CREATE A RETAIN CYCLE

```
func update() {
    let api = API()
    api.load {[weak api] result in
        guard let api = api else { return }
        api.log(message: "Some")
    }
}
```

USE [WEAK SELF]

NEEDED WHEN YOU CREATE A RETAIN CYCLE

```
class ViewModel {
    let api = API()

    func update() {
        api.load {[weak self] result in
            guard let self = self else { return }
            self.log(result: result)
        }
    }
}
```

USE [WEAK SELF]

IS THERE A BETTER WAY?

HOW ABOUT MOVING OWNERSHIP?

```
class ViewModel {
    let api = API()

    func update() {

    }
}
```

HOW ABOUT MOVING OWNERSHIP?

```
class ViewModel {
    let api = API()

    func update() {
        api.load(context: self) { (self, result) in
            self.log(result: result)
        }
    }
}
```

HOW ABOUT MOVING OWNERSHIP?

```
class API {
    var complete: ((String) -> Void)?
    func load<T: AnyObject>(context: T,
                            completion: @escaping (T, String) -> Void) {
        complete = {[weak context] result in
            guard let context = context else { return }
            completion(context, result)
        }

        DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
            self.finish()
        }
    }

    private func finish() {
        complete?("Done")
    }
}
```

WRAP UP OF [WEAK SELF]

- Don’t just use always **[weak self]**
- If [weak self] use **guard let self = self else { return }**
- Or transfer the ownership management to the API
- And don’t use **self?.doSomething()**