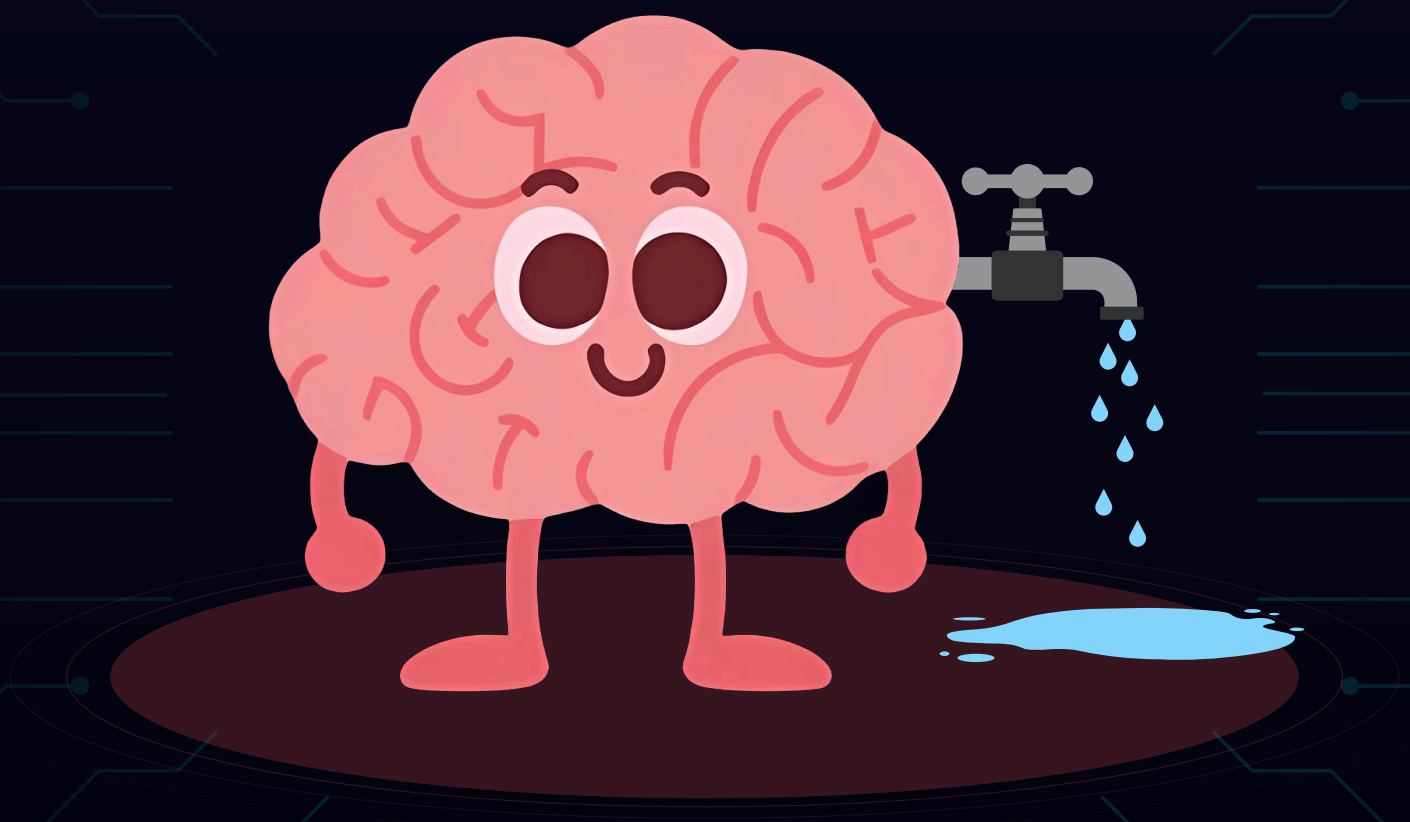
Mobile
Innovation
Network

1 min read



Memory Leaks in iOS

— Quick Fix Guide —

Mobile Innovation Network

Swipe

What is a

Memory Leak?

👉 A **memory leak** occurs when an app retains memory that is no longer needed. This can result in:

- Increased RAM usage
- Poor performance
- App crashes

👉 iOS uses **ARC** to manage memory, but developers must still:

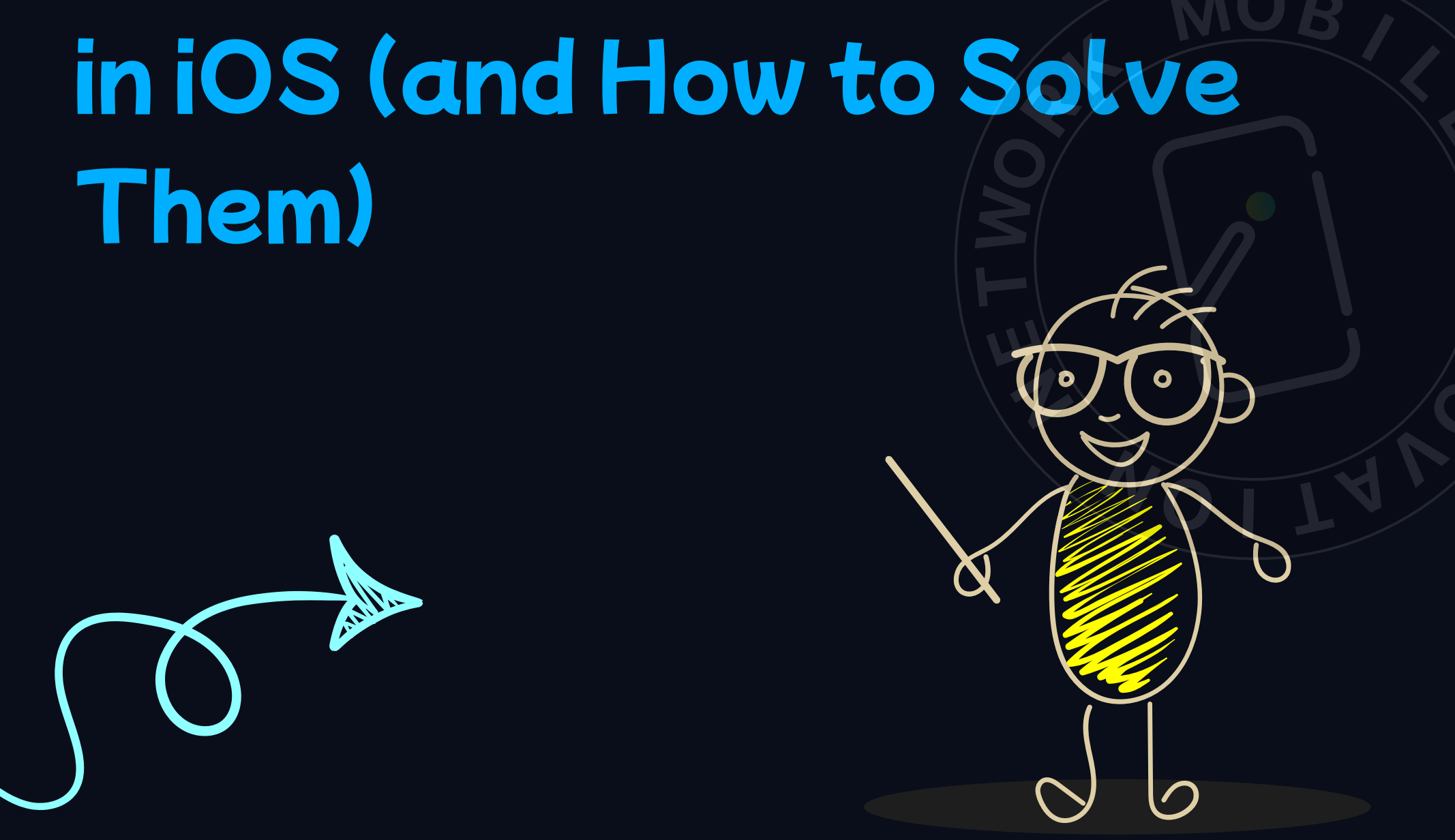
- Write code carefully
- Avoid retain cycles and other memory management pitfalls

Mobile Innovation Network

Swipe

Common Causes of

Memory Leaks in iOS (and How to Solve Them)



Mobile Innovation Network

Swipe

1. Retain Cycles in Closures

Closures can strongly capture self, causing retain cycles.

❌ PROBLEM

```
class HomeViewController: UIViewController {

    var onDataLoaded: (() -> Void)?

    override func viewDidLoad() {
        super.viewDidLoad()
        // ❌ Leak: Closure strongly captures `self`
        loadData {
            self.updateUI()
        }
    }

    deinit { print("HomeViewController deallocated") }
}
```

✅ How to Fix

```
loadData { [weak self] in
    self?.updateUI() // ✅ Weak capture
}
```

- Use [weak self] or [unowned self]

Mobile Innovation Network

Swipe

2. Delegates Not Marked as Weak

Properties are strong by default. Forgetting weak on delegates can cause retain cycles.

❌ PROBLEM

```
protocol DataManagerDelegate: AnyObject { /* ... */ }

class DataManager {
    var delegate: DataManagerDelegate? // ❌ Should be `weak`!
}

class ViewController: UIViewController, DataManagerDelegate {
    let dataManager = DataManager()

    override func viewDidLoad() {
        super.viewDidLoad()
        dataManager.delegate = self // Retain cycle!
    }
}
```

✅ How to Fix

```
class DataManager {
    weak var delegate: DataManagerDelegate? // ✅
}
```

- Always mark delegates as weak

Mobile Innovation Network

Swipe

3. Timers Not Invalidated

If you use a Timer and don’t invalidate it or break the reference, it will keep firing and retain its target.

❌ PROBLEM

```
timer = Timer.scheduledTimer(withTimeInterval: 1.0, repeats: true) { _ in
    self.updateCountdown()
}
```

✅ How to Fix

```
timer = Timer.scheduledTimer(withTimeInterval: 1.0, repeats: true) { [weak self] _ in
    self?.updateCountdown()
}
```

- Also, always call:

  timer.invalidate()

- when the timer is no longer needed (e.g., in deinit or viewWillDisappear).

Mobile Innovation Network

Swipe

4. Observers Not Removed

If you add an observer to NotificationCenter and don’t remove it, the system will retain your object

❌ PROBLEM

```
class ProfileController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // ❌ Observer not removed
        NotificationCenter.default.addObserver(
            self,
            selector: #selector(handleUpdate),
            name: .profileUpdated,
            object: nil
        )
    }

    // Missing removal in `deinit`
}
```

✅ How to Fix

```
deinit {
    NotificationCenter.default.removeObserver(self)
}
```

Mobile Innovation Network

Swipe

5. Strong Self Captured in Async Tasks

Async APIs like DispatchQueue, Combine, or async/await can capture self strongly.

❌ PROBLEM

```
DispatchQueue.global().async {
    self.doWork()
}
```

✅ How to Fix

```
DispatchQueue.global().async { [weak self] in
    self?.doWork()
}

Same for Combine:

publisher
    .sink { [weak self] value in
        self?.handle(value)
    }
```

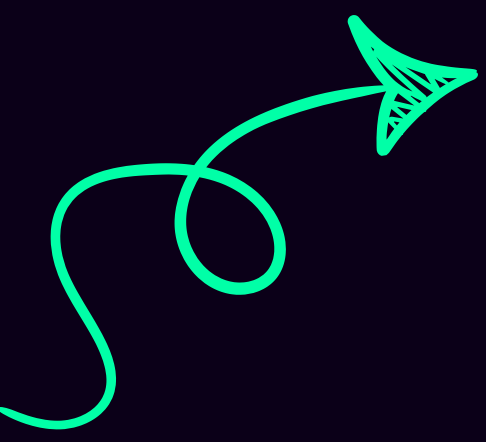
🛠️ Tools to Detect Leaks

👉 Xcode Memory Graph Debugger

- Analyze retained objects in live app

👉 Instruments (Leaks + Allocations)

- Profile memory over time



Mobile Innovation Network

Swipe

Best Practices

👉 Use **weak/unowned** for closures, delegates, and reference cycles.

👉 **Avoid lazy var** closures capturing self without [weak self].

👉 **Leverage deinit:** Add print statements to confirm deallocation.

👉 **Test Navigation:** Push/pop view controllers repeatedly to catch leaks.

📌 Final Thoughts

👉 Memory leaks in iOS may not cause immediate crashes.

👉 They silently degrade app performance over time.

👉 Understanding how ARC works is essential.

👉 Be aware of common traps like:

- Retain cycles
- Strongly captured closures
- Forgotten observers

👉 This knowledge helps build faster and more reliable apps.

Thank you for your Attention!