Thanks to

BARTER
COMMUNITY HUB

puzl
coworking

Thanks to

Follow us - Xcoders Sofia

https://www.facebook.com/profile.php?id=61565246821995

https://www.meetup.com/xcoders-sofia

https://www.linkedin.com/groups/8323933

Swift concurrency

The good, the bad and the ugly

Radoslav Radenkov 19.11.2024

About me

- Principle software engineer @ Paysafe
- 10 Years experience writing mobile apps
- ❄️⛄ 🏂 ⛰️

Swift concurrency

- Easy
- Clear
- Safe

```
func loadRecipe(_ completion: @escaping (Recipe?) -> Void) {
    let request = URLRequest(url: URL(string: "example.org")!)
    let task = URLSession.shared.dataTask(
        with: request,
        completionHandler: { data, response, error in
            guard let data else {
                return
            }

            DispatchQueue.main.async {
                completion(nil)
            }
        }
    )
    task.resume()
}
```

```
func makePizza(_ completion: @escaping (...) -> Void) {
    let dispatchGroup = DispatchGroup()
    dispatchGroup.enter()
    getDough { result in
        dough = result
        dispatchGroup.leave()
    }
    dispatchGroup.enter()
    getCheese { result in
        cheese = result
        dispatchGroup.leave()
    }

    dispatchGroup.enter()
    getTomatoSauce { result in
        tomatoSauce = result
        dispatchGroup.leave()
    }

    dispatchGroup.notify(queue: .main) {
        guard let dough, let cheese, let tomatoSauce else {
            assert(false, "Missing ingredients")
            return
        }

        // Make pizza
    }
```

```
func makePizza() async {
    async let (dough, cheese, tomatoSauce)
        = (getDough(), getCheese(), getTomatoSauce())

    cook(await dough, await cheese, await tomatoSauce)
}
```

```
@MainActor
func makePizza() async {
    async let (dough, cheese, tomatoSauce)
        = (getDough(), getCheese(), getTomatoSauce())

    cook(await dough, await cheese, await tomatoSauce)
}
```

Data Race Safety

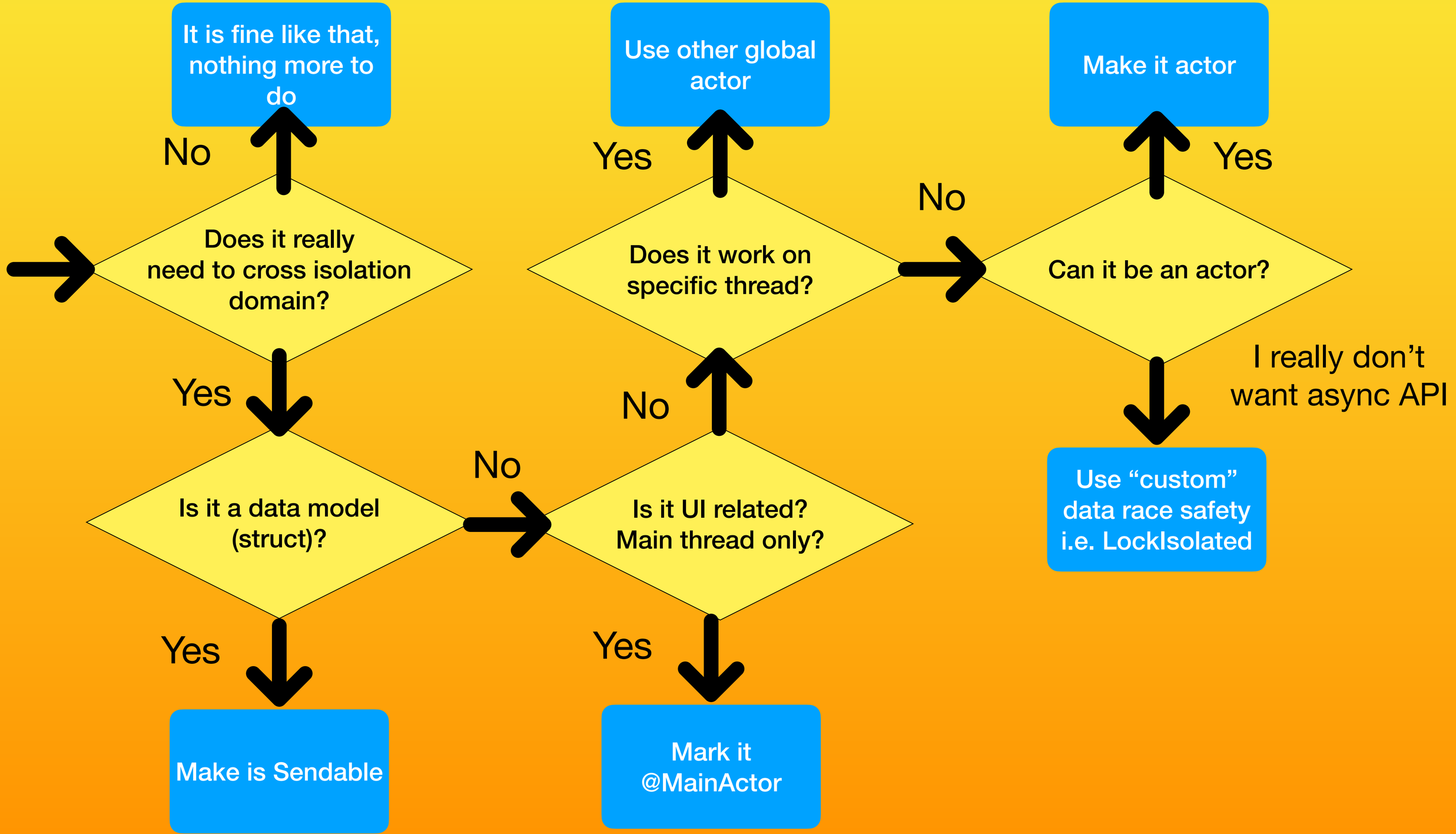
```
...completionHandler:
{ data, response, error in
    let decoder = JSONDecoder()
    let recipe = try? decoder.decode(Recipe.self, from: data)
    DispatchQueue.main.async {
        completion(recipe)
    }
}

@MainActor
func makePizza() async {
    let cheese = await getCheese()
}
```

⚠ Capture of 'recipe' with non-sendable type 'Recipe?' in a `@Sendable` closure

Data Isolation

- Complier is checking for data race safety - **static** checks
- Sendable
  - Structs
  - Final classes with non-mutable state
- Actors
- Global actors


It is fine like that, nothing more to do
No
Does it really need to cross isolation domain?
Yes
Is it a data model (struct)?
Yes
Make is Sendable
No
Is it UI related? Main thread only?
Yes
Mark it @MainActor
No
Does it work on specific thread?
Yes
Use other global actor
No
Can it be an actor?
Yes
Make it actor
I really don’t want async API
Use “custom” data race safety i.e. LockIsolated

```swift
class DataFormattersCache {
    private var cache: [String: DateFormatter] = [:]

    func formatter(for format: String) -> DateFormatter {
        if let formatter = cache[format] {
            return formatter
        }

        let formatter = DateFormatter()
        formatter.dateFormat = format
        cache[format] = formatter

        return formatter
    }
}
```

```swift
final class DataFormattersCache: Sendable {
    private var cache: [String: DateFormatter] = [:]
    ⚠ Stored property 'cache' of 'Sendable'-conforming class 'DataFormattersCache' is mutable
    func formatter(for format: String) -> DateFormatter {
        if let formatter = cache[format] {
            return formatter
        }

        let formatter = DateFormatter()
        formatter.dateFormat = format
        cache[format] = formatter

        return formatter
    }
}
```

```swift
final class DataFormattersCache: @unchecked Sendable {
    private let lock = NSLock()
    private var cache: [String: DateFormatter] = [:]

    func formatter(for format: String) -> DateFormatter {
        lock.lock()
        defer { lock.unlock() }
        if let formatter = cache[format] {
            return formatter
        }

        let formatter = DateFormatter()
        formatter.dateFormat = format
        cache[format] = formatter

        return formatter
    }
}
```

```swift
final class DateFormattersCache: Sendable {
    private let cache = LockIsolated<[String: DateFormatter]>([:])

    func formatter(for format: String) -> DateFormatter {
        cache.withValue { formatters in
            if let formatter = formatters[format] {
                return formatter
            }

            let formatter = DateFormatter()
            formatter.dateFormat = format
            formatters[format] = formatter

            return formatter
        }
    }
}
```

```
public final class LockIsolated<Value>: @unchecked Sendable {
    private var _value: Value
    private let lock = NSRecursiveLock()

    init(_ value: Value) {
        self._value = value
    }

    func withValue<T>(_ body: (inout Value) throws -> T) rethrows -> T {
        lock.lock(); defer { lock.unlock() }
        return try body(&_value)
    }

    func setValue(_ value: Value) {
        lock.lock(); defer { lock.unlock() }
        self._value = value
    }
}
```

```swift
import Synchronization

@available(iOS 18.0, *)
final class DataFormattersCache: Sendable {
    private let cache = Mutex<[String: DateFormatter]>([:])

    func formatter(for format: String) -> DateFormatter {
        cache.withLock { formatters in
            if let formatter = formatters[format] {
                return formatter
            }

            let formatter = DateFormatter()
            formatter.dateFormat = format
            formatters[format] = formatter

            return formatter
        }
    }
}
```

When designing tasks for concurrent execution, do not call methods that block the current thread of execution. When a task scheduled by a concurrent dispatch queue blocks a thread, the system creates additional threads to run other queued concurrent tasks. If too many tasks block, the system may run out of threads for your app.

Xcode Documentation, Dispatch Queue

```swift
func makeSync(_ asyncOperation: @Sendable @escaping () -> Recipe) -> Recipe {
    let semaphore = DispatchSemaphore(value: 0)
    var recipe: Recipe!

    DispatchQueue.global().async {
        recipe = asyncOperation()
        semaphore.signal()
    }

    semaphore.wait()
    return recipe
}
```

... code written with Swift concurrency can maintain a runtime contract that threads are always able to make forward progress. ... This is in the form of a new cooperative thread pool to back Swift concurrency as the default executor. The new thread pool will only spawn as many threads as there are CPU cores, thereby making sure not to overcommit the system.

WWDC 2021, Swift concurrency: Behind the scenes

Protocols that haven’t adopted Swift concurrency

- Apple APIs - WKNavigationDelegate
- Fixed in Xcode 16
- Suggestion works on Xcode >= 15.3 || >= iOS 17
- Swift 6: Suppress errors about isolation mismatch
  - *@preconcurrency* conformance to protocol

```
public protocol WKNavigationDelegate: NSObjectProtocol {
    ...
    optional func webView(_ webView: WKWebView,
                          didStartProvisionalNavigation: WKNavigation!)
    ...
}

extension MyViewController: WKNavigationDelegate {
    // ⚠ Main actor-isolated instance method
    // 'webView(_:didStartProvisionalNavigation:)'
    // cannot be used to satisfy nonisolated protocol requirement
    func webView(_ webView: WKWebView,
                 didStartProvisionalNavigation navigation: WKNavigation?) {
        showLoading()
    }
}
```

```
extension MyViewController: WKNavigationDelegate {
    nonisolated func webView(_ webView: WKWebView,
                             didStartProvisionalNavigation navigation: WKNavigation?) {
        // ❌ Call to main actor-isolated instance method ‘showLoading()'
        // in a synchronous nonisolated context
        showLoading()
    }
}
```

```
extension MyViewController: WKNavigationDelegate {
    nonisolated func webView(_ webView: WKWebView,
                 didStartProvisionalNavigation navigation: WKNavigation?) {
        Task { @MainActor in
            showLoading()
        }
    }
}
```

⚠️ Think three times before doing it ⚠️

```
extension MyViewController: WKNavigationDelegate {
    nonisolated func webView(_ webView: WKWebView,
                 didStartProvisionalNavigation navigation: WKNavigation?) {
        MainActor.assumeIsolated {
            showLoading()
        }
    }
}
```

Mixing GCD and Swift concurrency

- Swift concurrency uses compile time checks
- GCD is dynamic (runtime)

```
class MyViewController: UIViewController {
    func loadRecipe() {
        DispatchQueue.global().async { [weak self] in
            self?.updateUI()
        }
    }

    @MainActor
    func updateUI() {}
}
```

⚠ Call to main actor-isolated instance method 'updateUI()' in a synchronous nonisolated context; this is an error in Swift 6

Task cancellation

- Cooperative task cancellation
- Each task checks whether it has been canceled at the appropriate point
  - `Task.checkCancellation()`
  - `Task.isCancelled`
- `Task.withTaskCancellationHandler(operation:onCancel:isolation:)`

```swift
func loadRecipe() async throws -> Recipe? {
    let request = URLRequest(url: URL(string: "example.org")!)
    let (data, response) = try await URLSession.shared.data(for: request)
    let decoder = JSONDecoder()
    return try decoder.decode(Recipe.self, from: data)
}
```

```
class MyViewController: UIViewController {
    private var task: Task<Void, Error>?

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        task = Task { @MainActor in
            let recipe = try await loadRecipe()
            // update UI
        }
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        task?.cancel()
    }
```

```swift
class MyViewController: UIViewController {
    private var task: Task<Void, Error>?

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        task = Task {
            let recipe = try await loadRecipe()
            // update UI
        }
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        task?.cancel()
    }
```

```
class MyViewController: UIViewController {
    private var task: Task<Void, Error>?

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        task = Task {
            do {
                let recipe = try await loadRecipe()
                // update UI
            } catch is CancellationError {
                // do nothing
            } catch {
                // show error
            }
        }
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        task?.cancel()
    }
```

Implementing cancellation

```swift
extension URLSession {
    func myData(for request: URLRequest) async throws -> (Data?, URLResponse) {
        try await withCheckedThrowingContinuation { continuation in
            let task = dataTask(with: request,
                                completionHandler: { data, response, error in
                if let error = error {
                    continuation.resume(throwing: error)
                } else {
                    continuation.resume(returning: (data, response!))
                }
            })
            task.resume()
        }
    }
}
```

```swift
extension URLSession {
    func myData(for request: URLRequest) async throws -> (Data, URLResponse) {
        var task: URLSessionDataTask?
        return try await withTaskCancellationHandler(operation: {
            try await withCheckedThrowingContinuation { continuation in
                task = dataTask(with: request,
                                completionHandler: { data, response, error in
                    ...
                })
                task?.resume()
            }
        }, onCancel: {
            task?.cancel()
        })
    }
}
```

Reference to captured var 'task' in concurrently-executing code

```swift
extension URLSession {
    func myData(for request: URLRequest) async throws -> (Data, URLResponse) {
        let taskLock = LockIsolated<URLSessionDataTask?>(nil)
        return try await withTaskCancellationHandler(operation: {
            try await withCheckedThrowingContinuation { continuation in
                let task = dataTask(with: request,
                                    …
                task.resume()
                taskLock.setValue(task)
            }
        }, onCancel: {
            taskLock.withValue { $0?.cancel() }
        })
    }
}
```

```
func loadRecipe() {
    let task = Task {
        let request = URLRequest(url: URL(string: "example.org")!)
        let (data, response) =
            try await URLSession.shared.myData(for: request)
        ...
    }
    task.cancel()
}
```

```swift
extension URLSession {
    func myData(for request: URLRequest) async throws -> (Data, URLResponse) {
        let taskLock = LockIsolated<URLSessionDataTask?>(nil)
        return try await withTaskCancellationHandler(operation: {
            try await withCheckedThrowingContinuation { continuation in
                if Task.isCancelled {
                    continuation.resume(throwing: CancellationError())
                }
                let task = dataTask(with: request,
                                    ...
                task.resume()
                taskLock.setValue(task)
            }
        }, onCancel: {
            taskLock.withValue { $0?.cancel() }
        })
    }
}
```

Adopting Swift concurrency

- Identify Sendable and @MainActor objects
- First enable *targeted* concurrency checking and then *strict*
- Module by module

Resources

- Getting started - WWDC21, WWDC22
- Common Compiler Errors from Migrating to Swift 6
- Matt Massicotte’s blog
- ConcurrencyRecepies by Matt Massicotte
- swift-concurrency-extras by PointFree

Q&A

Thank you!

CODERS SOFIA

Make an interactive spatial app

with Reality Composer and Xcode

w/ Nikola Kirev

DATE

15 Jan 2025, Wednesday
19:00

VENUE

Barter Community Hub
47 Cherni Vrah Blvd, Sofia

Feedback

https://forms.gle/6g3Je61WHbEbnF1u6