Welcome!

Professional Development Seminar Series

raywenderlich.com

Our mission is to create a world-class educational platform and community for mobile developers.

Connect With Our Experts & Other Devs!

* Help with raywenderlich.com tutorials, books and videos

* Career advice and job opportunities

* Support for your apps and projects

linkedin.com/company/rwenderlich/

@rwenderlich

discord.com/invite/ZTUmDxX

facebook.com/raywenderlich

instagram.com/rwenderlich

raywenderlich.com

WELCOME!

Modern Concurrency in Swift

FEBRUARY 23, 2022 • 11AM EST

Marin Todorov

AUTHOR

MODERN CONCURRENCY IN SWIFT

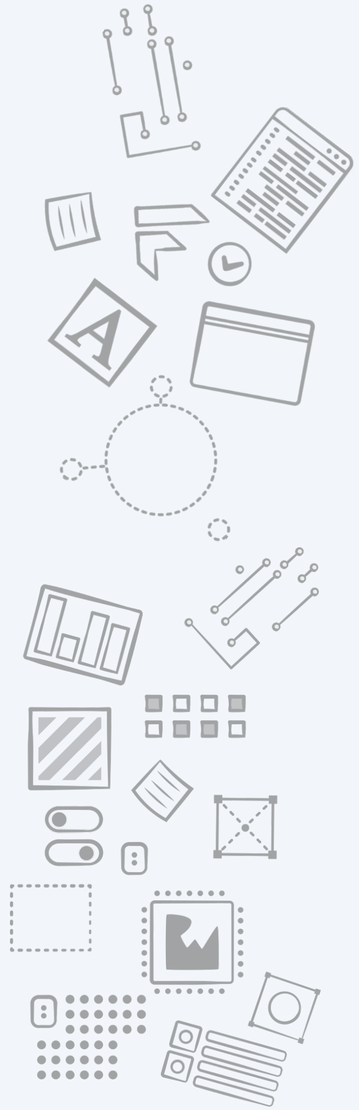
ICANZILB

Agenda

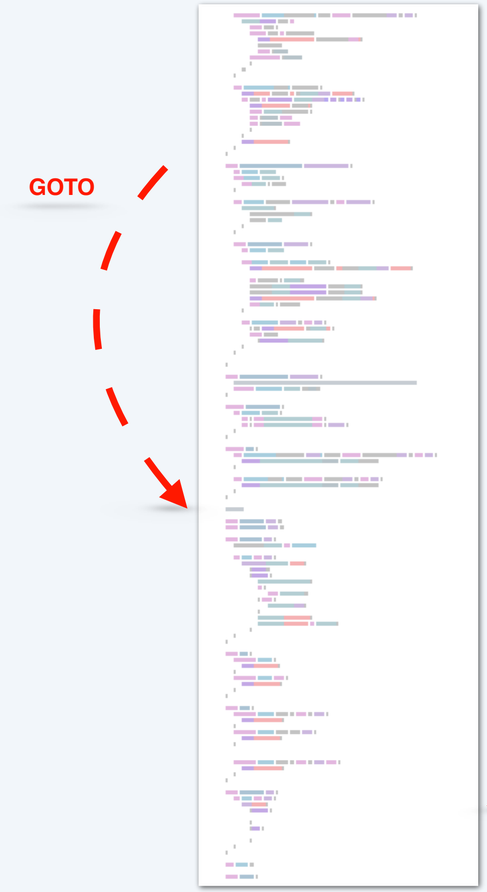
- Why a new concurrency model
- Asynchronous code
- Concurrency language design
- Where to go from here?

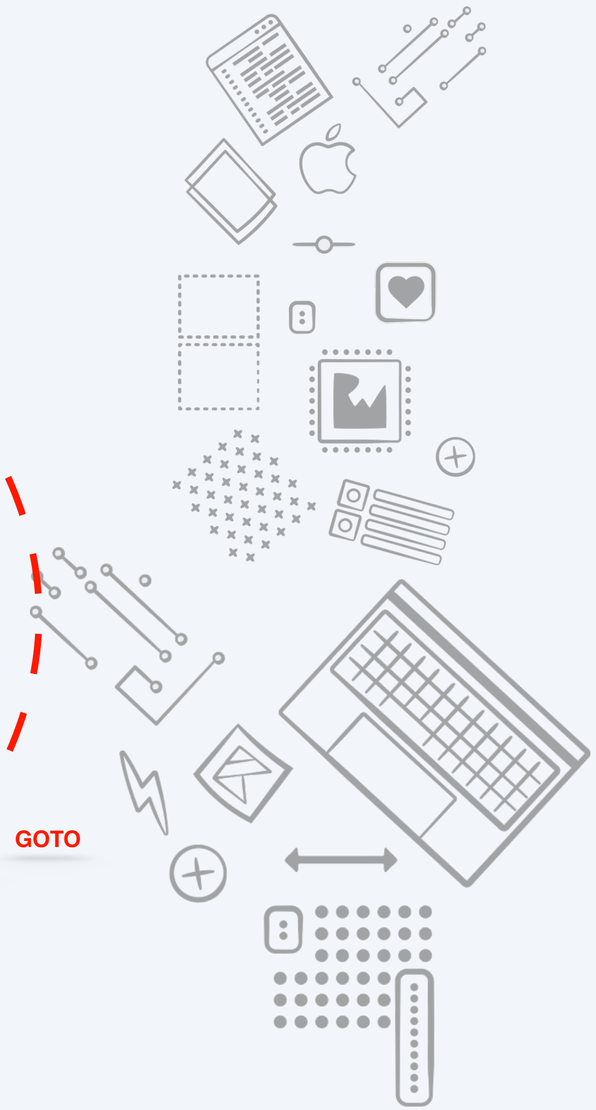
Structured Concurrency

- Threads
- Grand Central Dispatch
- Modern Concurrency



```
174
175     func fetch(callback: @escaping () -> Void) {
176         print("Hello")
177         DispatchQueue.main.async {
178             work()
179             callback()
180         }
181         print("World")
182     }
183
```

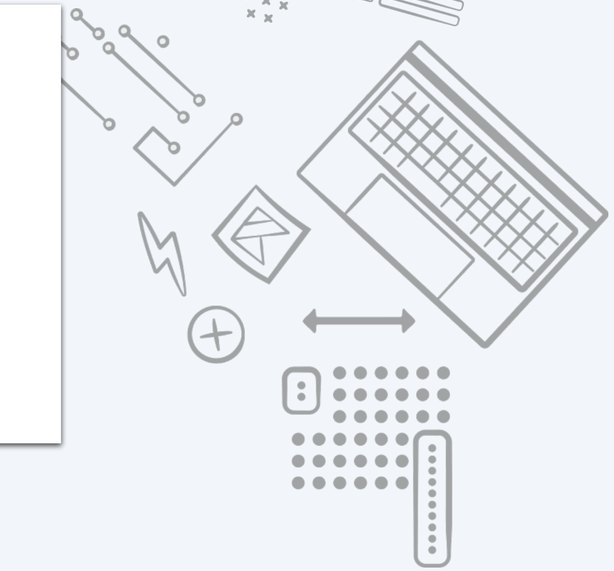




Async/await

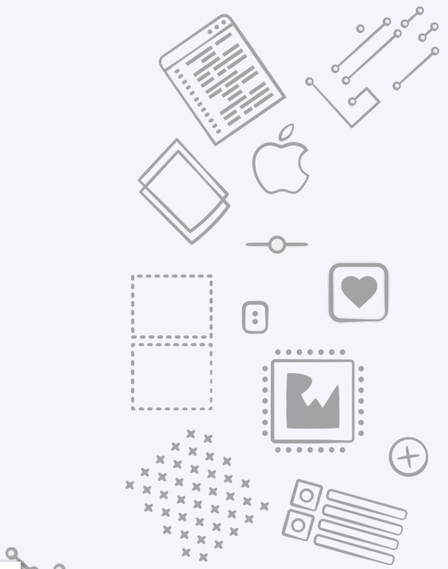
- Clean code syntax
- Tells the compiler what you're doing
- Enables runtime optimization

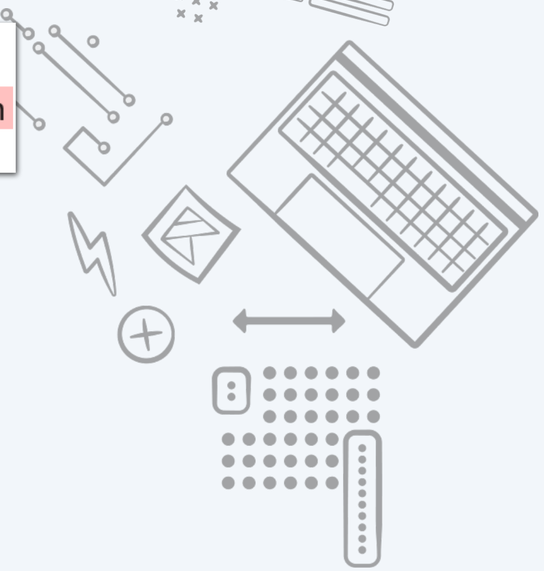
```
func loadItemsFromServer() async throws -> [Item] {
    return Server.allItems
}

let items = try await loadItemsFromServer()
print(items)
```



Concurrency with Actors

- Multi-core execution
- Shared mutable state
- Compile time guarantees



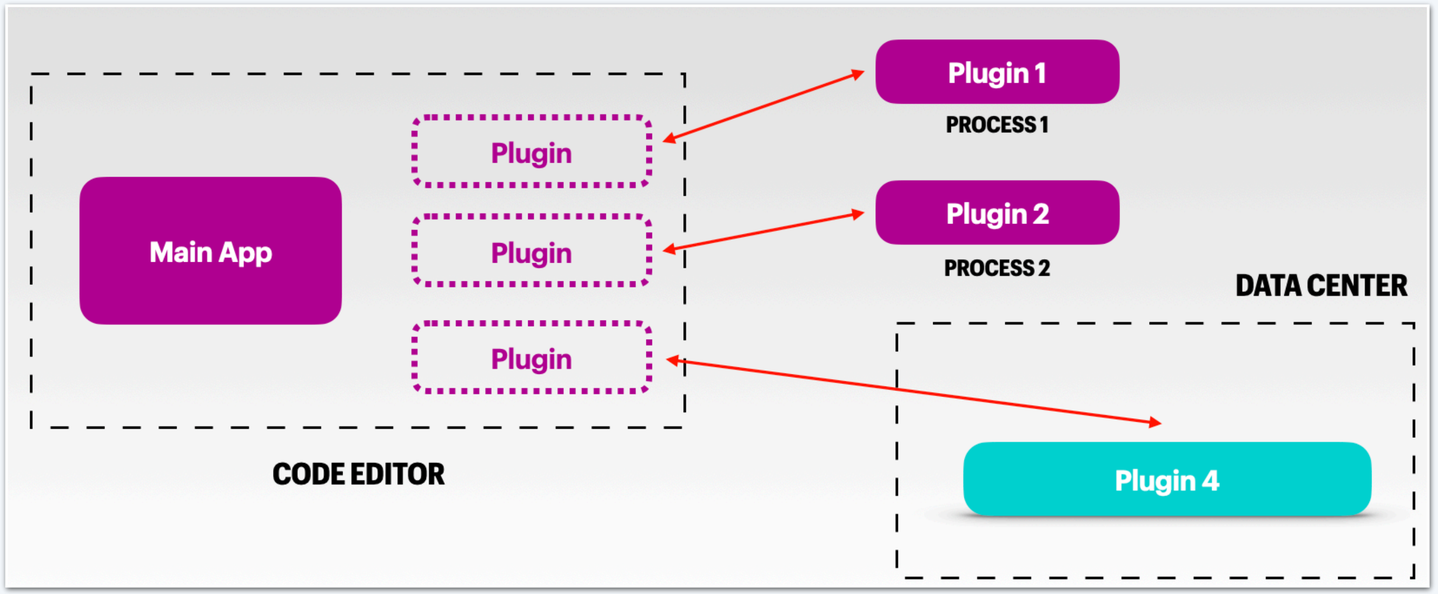


```
let database = DatabaseActor()
database.initialize()
```
actor-isolated instance method 'initialize()' can not be referenced from

```
let database = DatabaseActor()
await database.initialize(&data)
```
mutation of captured var 'data' in concurrently-executin...

Thank u, next

- @Sendable, @Sendable everywhere
- Ownership control
- Distributed actors



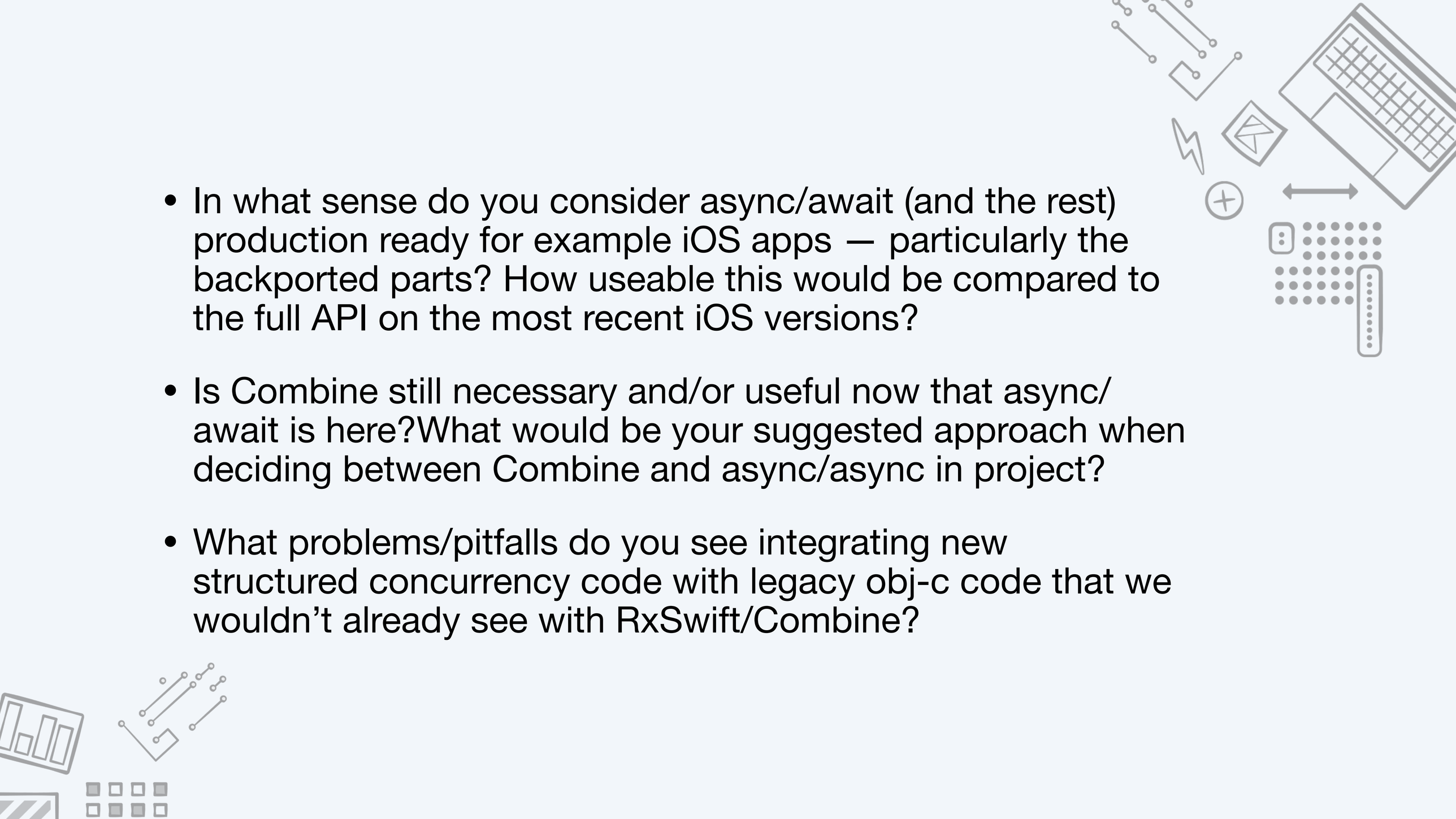
Key Takeaways

- Just like native error handling, async/await will change common Swift code patterns
- Will require, at minimum, familiarity at the point-of-use
- The goal is concurrency-safety at compile time
- Opens up new fields of Swift applications like distributed computing

Recommended Resources

- WWDC 2021 Concurrency videos
- https://forums.swift.org
- "async/await in SwiftUI" at rw.com
- "async/await in Server-Side Swift and Vapor" at rw.com
- "Modern Concurrency in Swift"

Questions



- In what sense do you consider async/await (and the rest) production ready for example iOS apps — particularly the backported parts? How useable this would be compared to the full API on the most recent iOS versions?

- Is Combine still necessary and/or useful now that async/await is here?What would be your suggested approach when deciding between Combine and async/async in project?

- What problems/pitfalls do you see integrating new structured concurrency code with legacy obj-c code that we wouldn’t already see with RxSwift/Combine?

- If a task is created explicitly (Task { ... }) from an async context (say from inside an actor's function, or any other async function), who is the task's parent task? The task in which the current async function runs? Does it inherit the parent's priority, actor (global or custom) etc?
- When will Actors Distributed System will be out?
- What is next for async/await?

Thank you!