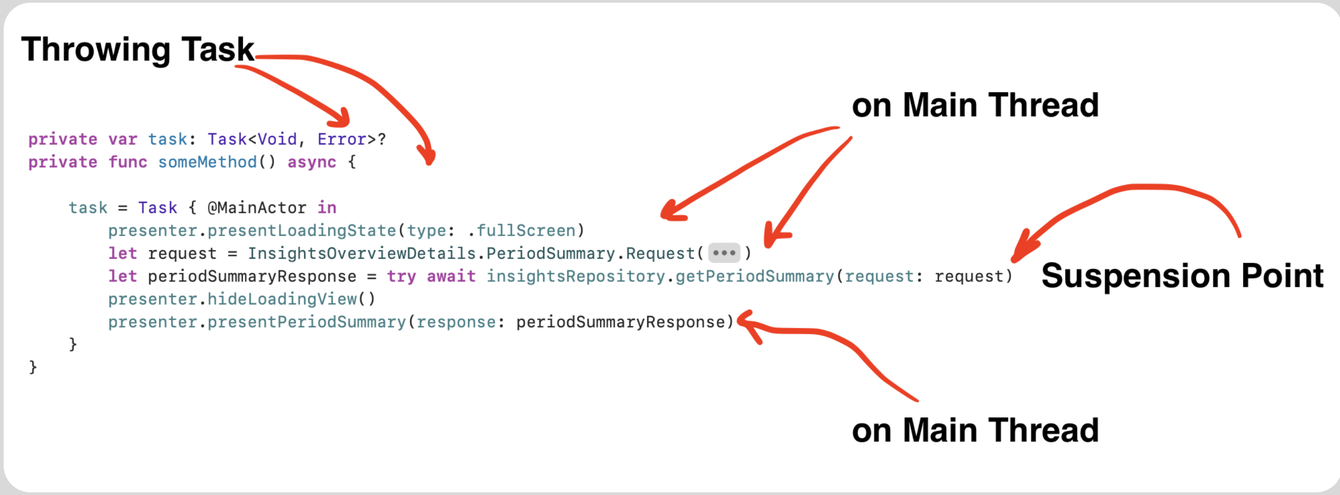
5 Donts:

- Don’t be deceived by “selfless” tasks. Self is captured implicitly
- Don’t believe in try without catch. Body of a task may be throwing and compiler won’t complain.
- Don’t forget about the main thread when dealing with UI. Mind the suspension points.
- Don’t mix GCD with Swift Concurrency. Your tasks may lose context.
- Don’t mark your whole classes as MainActor unless completely necessary or makes sense (like Coordinators)

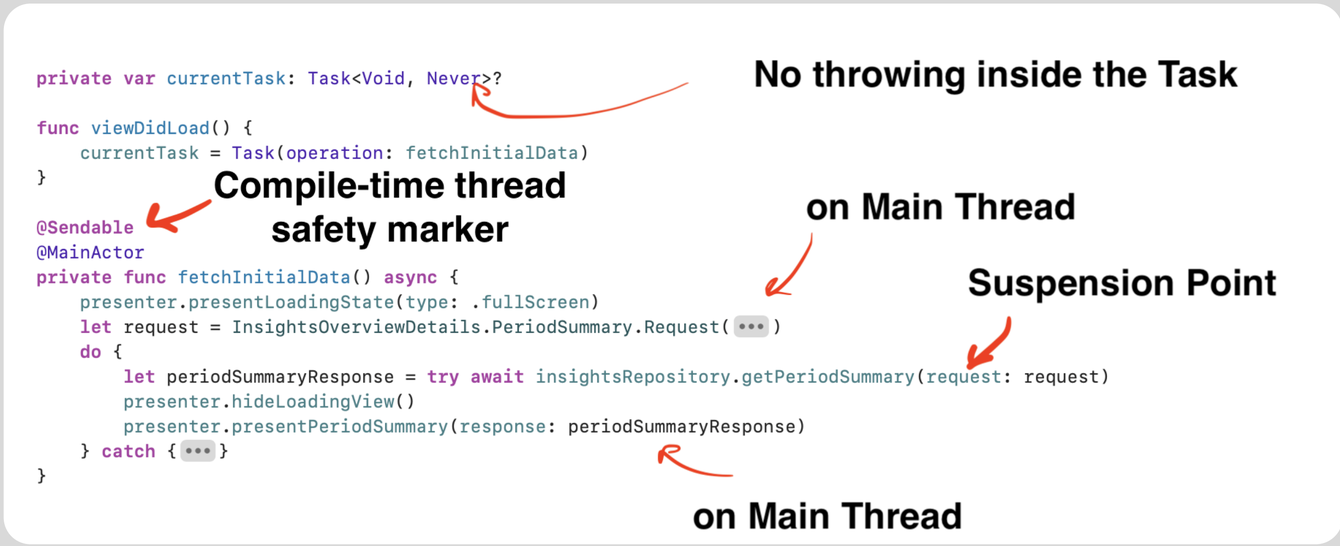



```swift
private var task: Task<Void, Error>?
private func someMethod() async {

    task = Task { @MainActor in
        presenter.presentLoadingState(type: .fullScreen)
        let request = InsightsOverviewDetails.PeriodSummary.Request( ••• )
        let periodSummaryResponse = try await insightsRepository.getPeriodSummary(request: request)
        presenter.hideLoadingView()
        presenter.presentPeriodSummary(response: periodSummaryResponse)
    }
}
```

Some highlights:

- The idea behind Swift Concurrency is to make Thread safety a compile-time matter.
- Sendable is only an empty protocol (or a marker for a closure/function). There’s no magic behind it.
- Simulators and real devices handle context switching differently.
- Try to bump your Concurrency check for the sample apps to Targeted 💡




```swift
private var currentTask: Task<Void, Never>?

func viewDidLoad() {
    currentTask = Task(operation: fetchInitialData)
}

@Sendable
@MainActor
private func fetchInitialData() async {
    presenter.presentLoadingState(type: .fullScreen)
    let request = InsightsOverviewDetails.PeriodSummary.Request( ••• )
    do {
        let periodSummaryResponse = try await insightsRepository.getPeriodSummary(request: request)
        presenter.hideLoadingView()
        presenter.presentPeriodSummary(response: periodSummaryResponse)
    } catch { ••• }
}
```