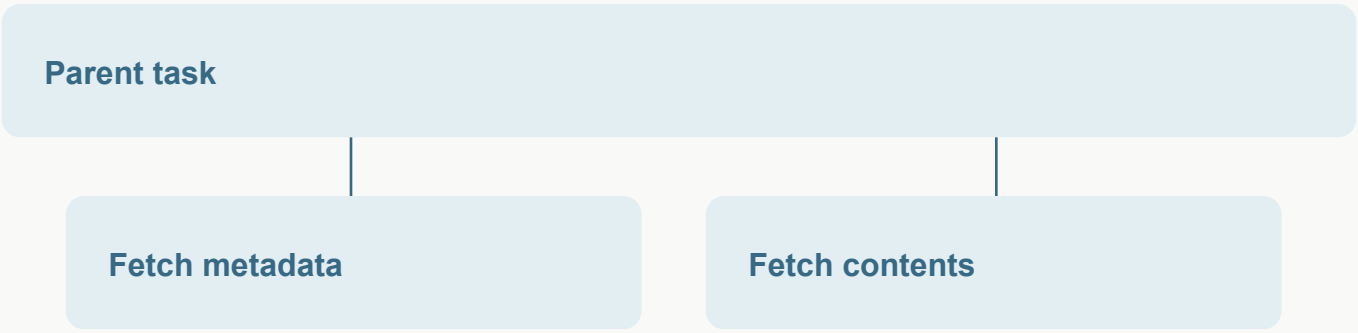
Swift concurrency

A small guide to tasks, suspension, and shared state.

01 STRUCTURE FOLLOWS LIFETIME

A task is a unit of asynchronous work. Structured concurrency gives child tasks a lifetime bounded by their enclosing scope, making the work easier to reason about.



Both child tasks complete before the scope returns.

02 SUSPENSION IS NOT BLOCKING

An await marks a possible suspension point. While a task waits, its thread can do other work. Asynchronous code does not necessarily run on a background thread.

03 ISOLATE SHARED STATE

Actors protect their isolated state. Code on the main actor is appropriate for updating the user interface; expensive work needs deliberate placement.

Two requests. One scope.

Use async let when the independent work is known up front.

```
func loadArticle() async throws -> Article {
    async let title = fetchTitle()
    async let body = fetchBody()

    return try await Article(
        title: title, body: body
    )
}
```

READ THE BOUNDARIES

The child tasks may run concurrently. The result is assembled after both values are ready. Errors propagate through try await; the structured scope also accounts for unfinished children.

CANCELLATION IS COOPERATIVE

A cancellation request is a signal. Long-running work should check for cancellation at useful boundaries and stop promptly when it can.

Keep the model simple.

A short review before introducing more concurrency.

01 Find the independent work

Only overlap operations that do not depend on each other.

02 Name the owner of mutable state

Choose the actor or isolation domain before sharing a value.

03 Plan the exit

Decide what errors and cancellation mean for this operation.

Further reading: The Swift Programming Language, Concurrency.
docs.swift.org/swift-book/documentation/the-swift-programming-language/concurrency/