SOLVING CALLBACK HELL
WITH GOOD OLD FUNCTION COMPOSITION

VINCENT PRADEILLES (@V_PRADEILLES) - WORLDLINE

CALLBACK HELL?

CALLBACK HELL?

```
typealias CompletionHandler<Result> = (Result?, Error?) -> Void

func service1(_ completionHandler: CompletionHandler<Int>) {
    completionHandler(42)
}

func service2(arg: Int, _ completionHandler: CompletionHandler<String>) {
    completionHandler("Result: \(arg)")
}

func service3(arg: String, _ completionHandler: CompletionHandler<String>) {
    completionHandler("🎉 \(arg)")
}
```

CALLBACK HELL?

```
typealias CompletionHandler<Result> = (Result?, Error?) -> Void

func service1(_ completionHandler: CompletionHandler<Int>) {
    completionHandler(42)
}

func service2(arg: Int, _ completionHandler: CompletionHandler<String>) {
    completionHandler("Result: \(arg)")
}

func service3(arg: String, _ completionHandler: CompletionHandler<String>) {
    completionHandler("🎉 \(arg)")
}
```

CALLBACK HELL?

```
typealias CompletionHandler<Result> = (Result?, Error?) -> Void

func service1(_ completionHandler: CompletionHandler<Int>) {
    completionHandler(42)
}

func service2(arg: Int, _ completionHandler: CompletionHandler<String>) {
    completionHandler("Result: \(arg)")
}

func service3(arg: String, _ completionHandler: CompletionHandler<String>) {
    completionHandler("🎉 \(arg)")
}
```

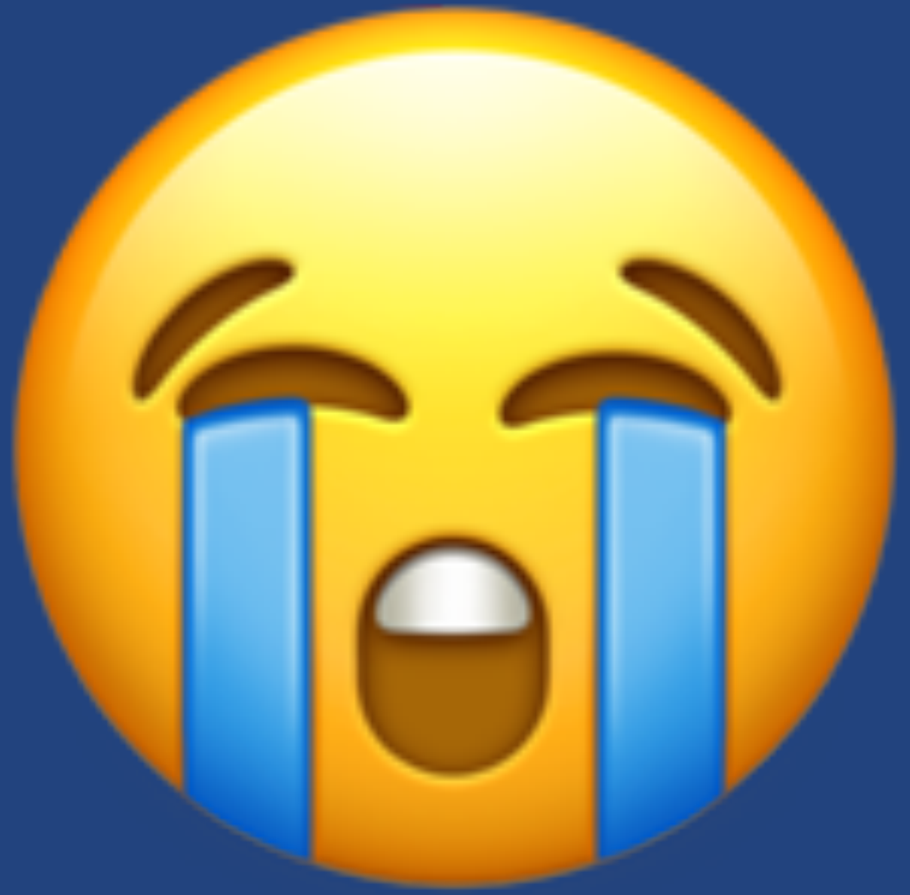
CALLBACK HELL?

```
typealias CompletionHandler<Result> = (Result?, Error?) -> Void

func service1(_ completionHandler: CompletionHandler<Int>) {
    completionHandler(42)
}

func service2(arg: Int, _ completionHandler: CompletionHandler<String>) {
    completionHandler("Result: \(arg)")
}

func service3(arg: String, _ completionHandler: CompletionHandler<String>) {
    completionHandler("🎉 \(arg)")
}
```

CALLBACK HELL?

```
typealias CompletionHandler<Result> = (Result?, Error?) -> Void

func service1(_ completionHandler: CompletionHandler<Int>) {
    completionHandler(42)
}

func service2(arg: Int, _ completionHandler: CompletionHandler<String>) {
    completionHandler("Result: \(arg)")
}

func service3(arg: String, _ completionHandler: CompletionHandler<String>) {
    completionHandler("🎉 \(arg)")
}
```

```
service1 { result, _ in
    guard let result = result else { return }

    service2(arg: result, { result, _ in
        guard let result = result else { return }

        service3(arg: result, { result, _ in
            guard let result = result else { return }

            print(result) // 🎉 Result: 42
        })
    })
}
```



Combinators

NSSpain 2018

Daniel H Steinberg

dimsumthinking.com and editorscut.com

"A combinator is a higher-order function that uses only function application and earlier defined combinators to define a result from its arguments."

https://en.wikipedia.org/wiki/Combinatory_logic (August 18, 2018)

```
infix operator ~>: MultiplicationPrecedence
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ second: @escaping (T, CompletionHandler<U>) -> Void)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ firstResult, error in
            guard let firstResult = firstResult else { completion(nil, error); return }

            second(firstResult, { secondResult, error in
                completion(secondResult, error)
            })
        })
    }
}
```

```
let chainedServices = service1 ~> service2 ~> service3

chainedServices({ result, _ in
    guard let result = result else { return }

    print(result) // 🎉 Result: 42
})
```

```
let chainedServices = service1 ~> service2 ~> service3

chainedServices({ result, _ in
    guard let result = result else { return }

    print(result) // 🎉 Result: 42
})
```

CALLBACK HELL SOLVED!

REALLY?

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ transform: @escaping (T) -> U)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ result, error in
            guard let result = result else { completion(nil, error); return }

            completion(transform(result), nil)
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ transform: @escaping (T) -> U)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ result, error in
            guard let result = result else { completion(nil, error); return }

            completion(transform(result), nil)
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ transform: @escaping (T) -> U)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ result, error in
            guard let result = result else { completion(nil, error); return }

            completion(transform(result), nil)
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ transform: @escaping (T) -> U)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ result, error in
            guard let result = result else { completion(nil, error); return }

            completion(transform(result), nil)
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ transform: @escaping (T) -> U)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ result, error in
            guard let result = result else { completion(nil, error); return }

            completion(transform(result), nil)
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ transform: @escaping (T) -> U)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ result, error in
            guard let result = result else { completion(nil, error); return }

            completion(transform(result), nil)
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ transform: @escaping (T) -> U)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ result, error in
            guard let result = result else { completion(nil, error); return }

            completion(transform(result), nil)
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ transform: @escaping (T) -> U)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ result, error in
            guard let result = result else { completion(nil, error); return }

            completion(transform(result), nil)
        })
    }
}
```

```
func ~> <T, U>(_ first: @escaping (CompletionHandler<T>) -> Void,
               _ transform: @escaping (T) -> U)
               -> (CompletionHandler<U>) -> Void {
    return { completion in
        first({ result, error in
            guard let result = result else { completion(nil, error); return }

            completion(transform(result), nil)
        })
    }
}
```

```
let chainedWithMap = service1
    ~> { int in return String(int / 2) }
    ~> service3

chainedWithMap({ result, _ in
    guard let result = result else { return }

    print(result) // Prints: 🎉 21
})
```

```
let chainedWithMap = service1
    ~> { int in return String(int / 2) }
    ~> service3

chainedWithMap({ result, _ in
    guard let result = result else { return }

    print(result) // Prints: 🎉 21
})
```