



# NEEDS

July 28, 2016

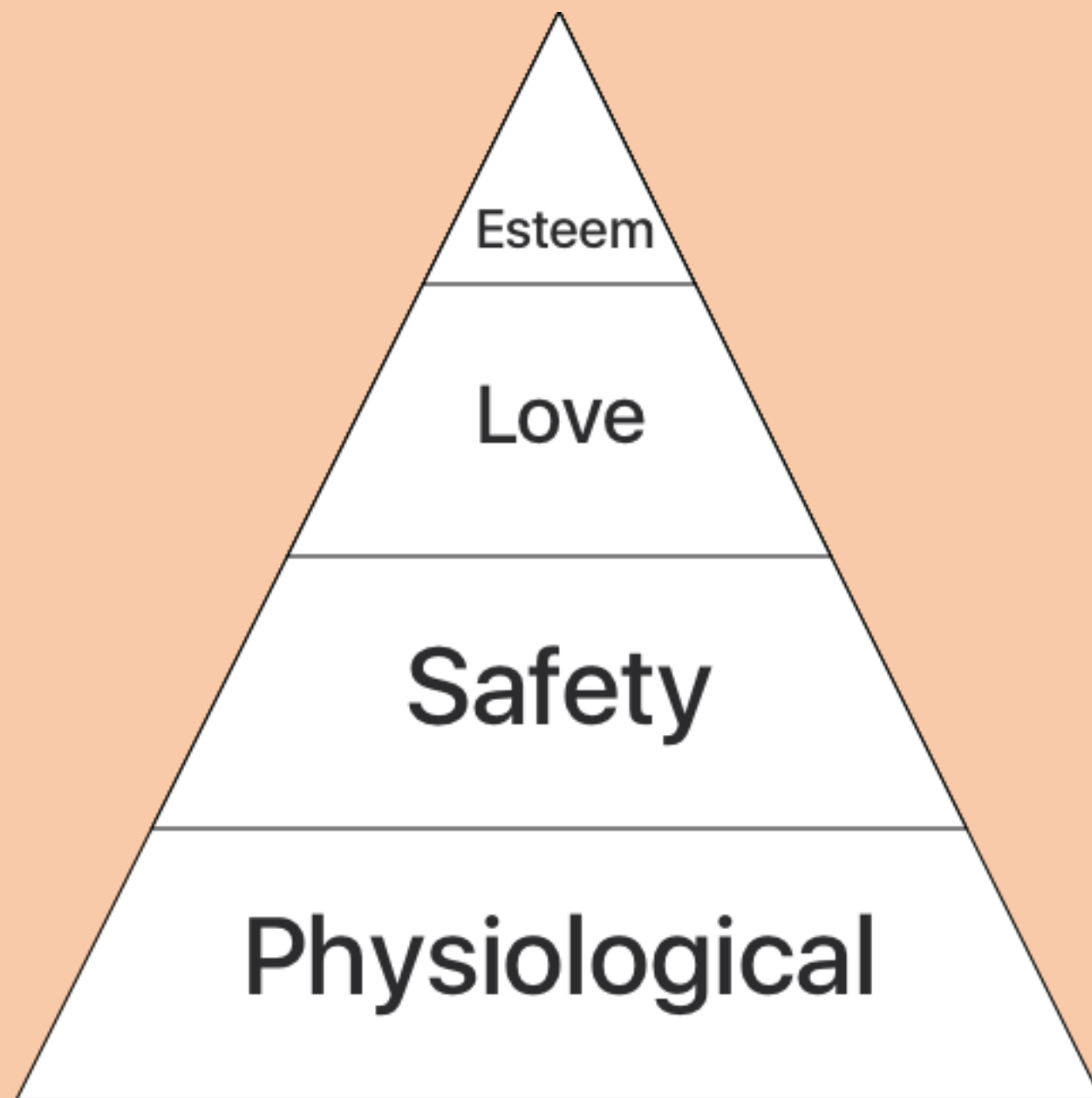


I'm Jack, I'm an iOS engineer here at Stripe. We have an iOS SDK.

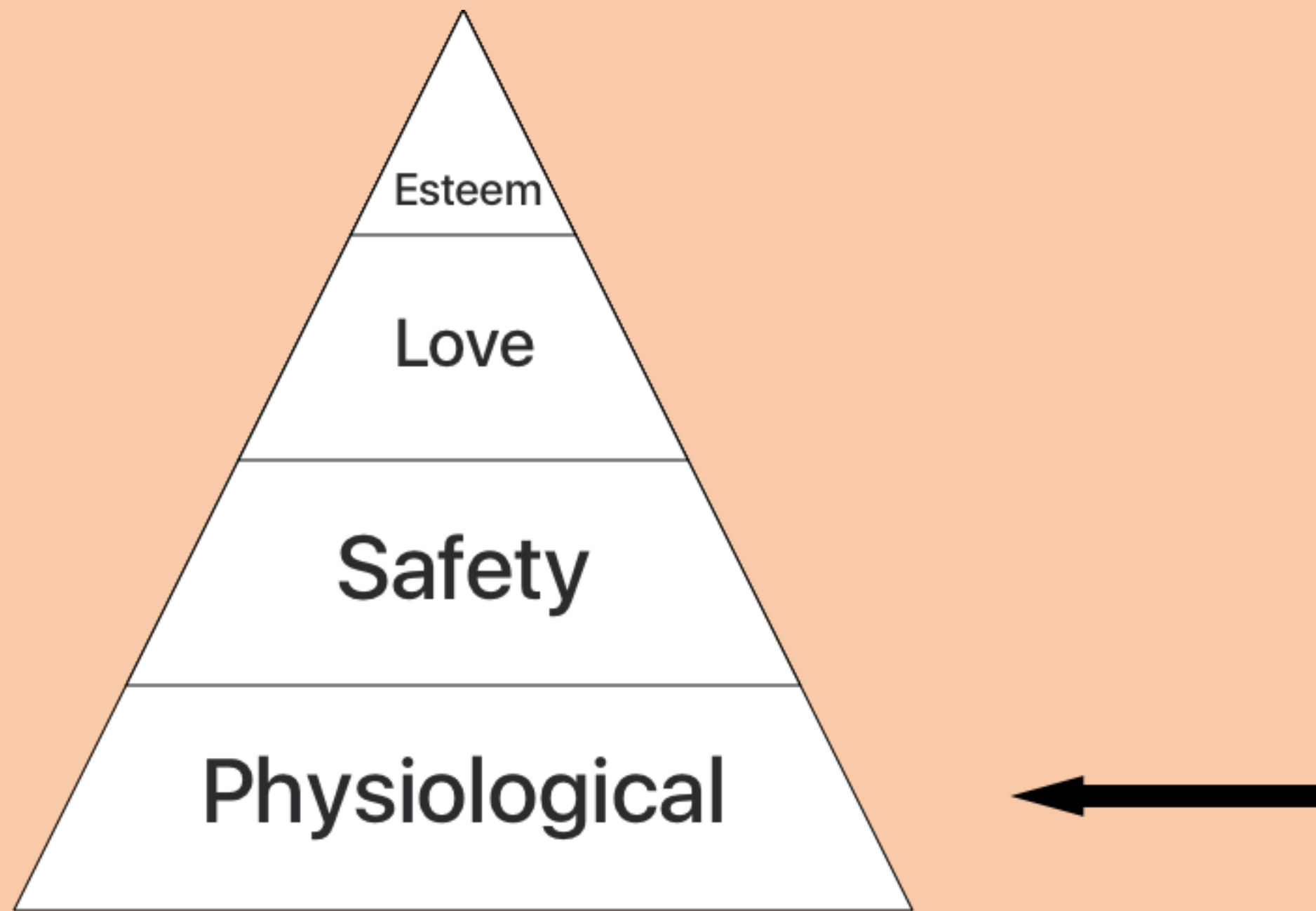
Choose a template for your new project:

iOS		
Application	Cocoa Touch Framework	Cocoa Touch Static Library
Framework & Library		
watchOS		
Application		
Framework & Library		

This has been my first time working on an SDK instead of an app. From a terminology standpoint, I basically define an SDK as a library that you are distributing as part of a paid service. I want to highlight some of the differences we've seen between that and developing first-party iOS apps.



So when someone is thinking about using your SDK, they're having an internal conversation with themselves. And what they're really doing is, they're thinking about their needs. And so you, as an SDK maker, just need to make sure those needs are met. And I really think a useful metaphor for this, from a priority standpoint, is Maslow's heirarchy of needs. This is a psychological framework developed by Abraham Maslow in 1943, and like most psychological frameworks from that era, it has tons of things that are super problematic and is primarily just useful as a metaphor in talks like this.



So, the most foundational thing in the hierarchy - the thing that you can't have anything else without - are physiological needs. Food and water.

# Installation

And I think the analogy here is installation. People can't evaluate your SDK if they can't install it. This sounds trivial, but we've spent a really surprising amount of effort here to make sure it's as easy as possible to install the Stripe SDK and keep it up to date. I'll tell you about some of the challenges here.

# Support ~everything.

- Cocoapods
- Carthage
- .framework

First, you have to support every possible way your users might want to install your SDK. This, at the minimum, means supporting Cocoapods users, and providing a prebuilt .framework. This means you get support for Carthage for free.

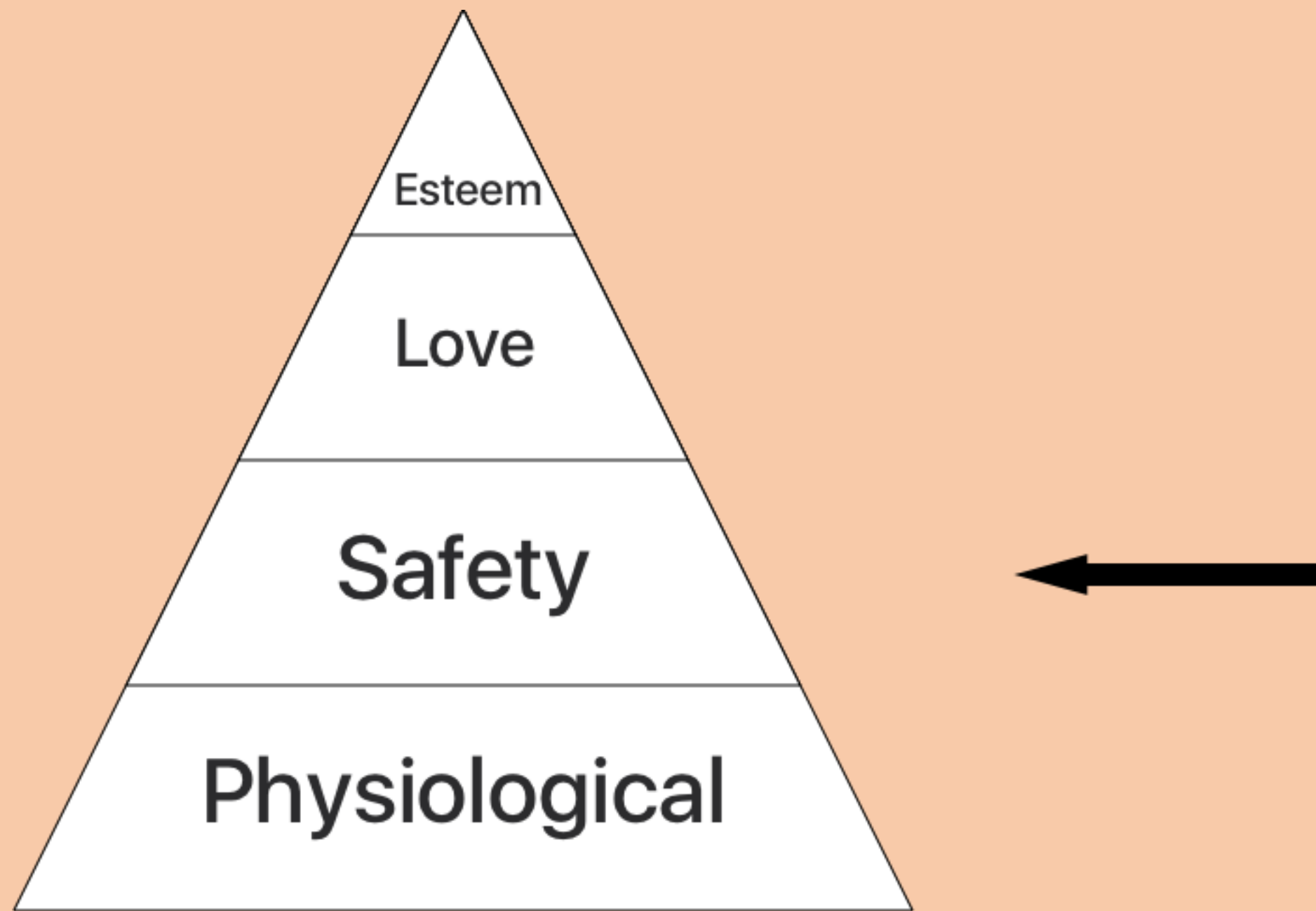
# Objective-C.

Next, simply put, given how often Swift changes, if you have any regard for your own sanity you have to write your SDK in Objective C.



# No dependencies.

Unfortunately, because people install your SDK via different means, you also can't use any third-party code. The exception here, which is an enormous pain, is if you "vendorize" code - i.e. you replace all of the class prefixes, and manually audit the code you want to use to make sure it would play nicely if someone else were using that code in their app.



So onto the second need in the hierarchy - safety. This translates pretty literally to SDK world - people want to be sure that your SDK isn't going to crash their app. There are a few essential ways to do this.

# Make it open-source.

This is by far the easiest way. Let developers peruse the source at their leisure. If they really really want to, they can fork it and change stuff they don't like (although you don't want them doing this, because it makes it harder to get them to upgrade and use new features).

# Write lots of tests.

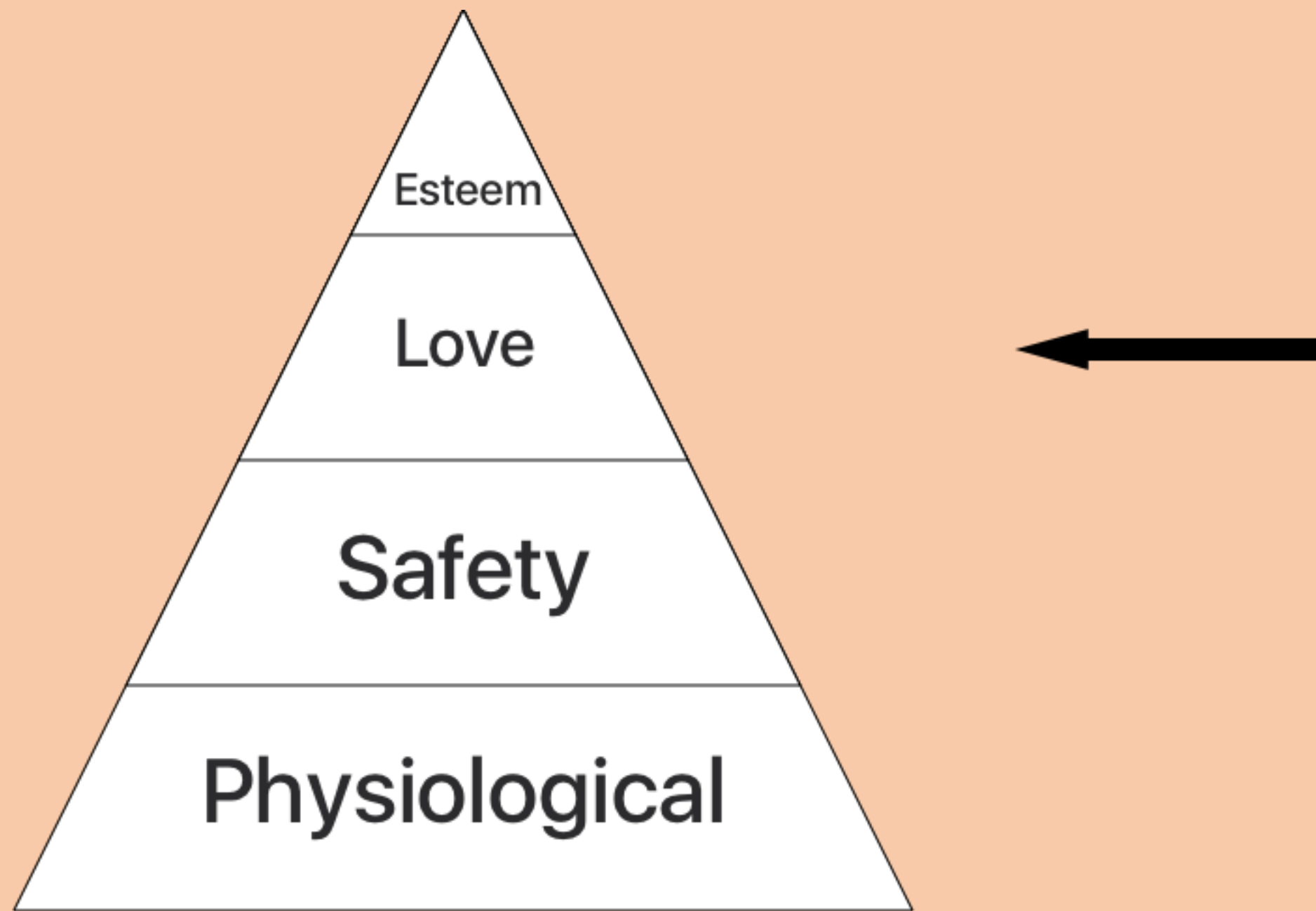
So, this should seem obvious. You should absolutely adopt a test-first mentality. The standard for your code is just way higher here. If you write a bug in an app, you see a few crashes. If you write a bug in your SDK, you see an enormous amount. So as much as I don't like forcing development practices on teams, I really do advocate using TDD when developing an SDK.

**\_\_IPHONE\_OS\_VERSION\_MIN\_REQUIRED**

We also have started to write a few unconventional tests. First, since our SDK supports back to iOS 7, we're super careful to not accidentally use an unavailable API. We use a really fantastic linting tool called FauxPas to enforce this, which integrates right into our Xcode and CI builds.

# Installation Tests

We actually also write CI tests to make sure installation methods don't break. (They fail all the time.)



Love/Belonging - API Design & Documentation

# API design

Atoms, not molecules (if you're going to ship molecules, ship the atoms too)

^ Don't force design patterns onto your users

^ Don't break your API (modocache talk)



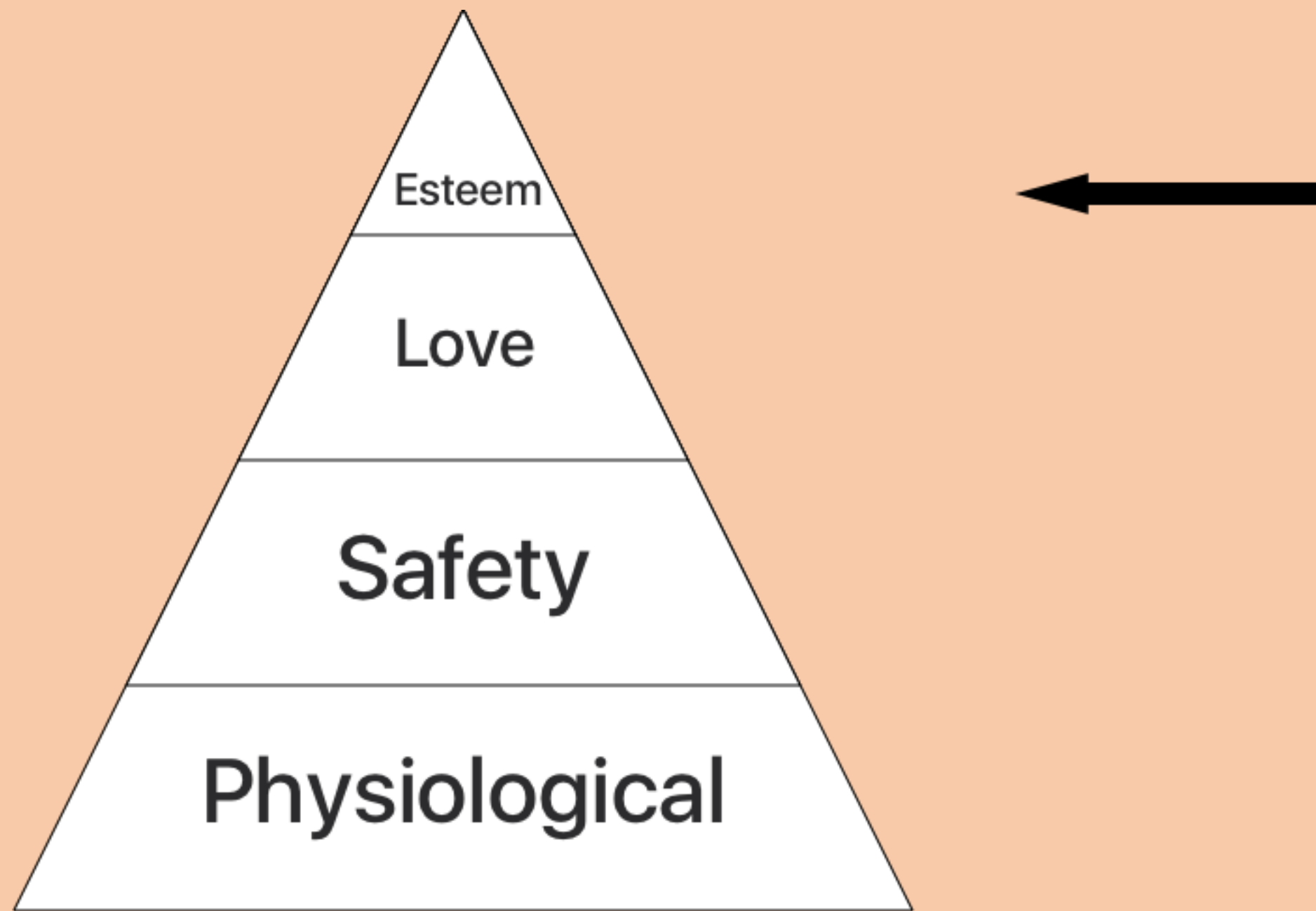
# Anticipate failures

You never know how people are going to actually call your APIs, so be super careful. Don't assume you're on the main thread, etc.

# Documentation

**You can't write  
enough.**

I don't have a lot to say here, other than that you can't write enough. We recently started using a fantastic tool from Realm called Jazzy, which will actually autogenerate a full static site for you based on documentation it generates from your headers.



Esteem and self-actualization

^ This is probably the weakest metaphor here, but cut me some slack. I think this basically correlates to having a dialogue with your users. It's also the thing that I think is hardest, and has the most room for growth in the industry.

# Features + Feedback

# How do you collect feedback?

- Normal channels
- Internal analytics
- Crash reporting?

# Thank You

jack@stripe.com  
@jflinter