facebook

iOS at Facebook

Simon Whitaker

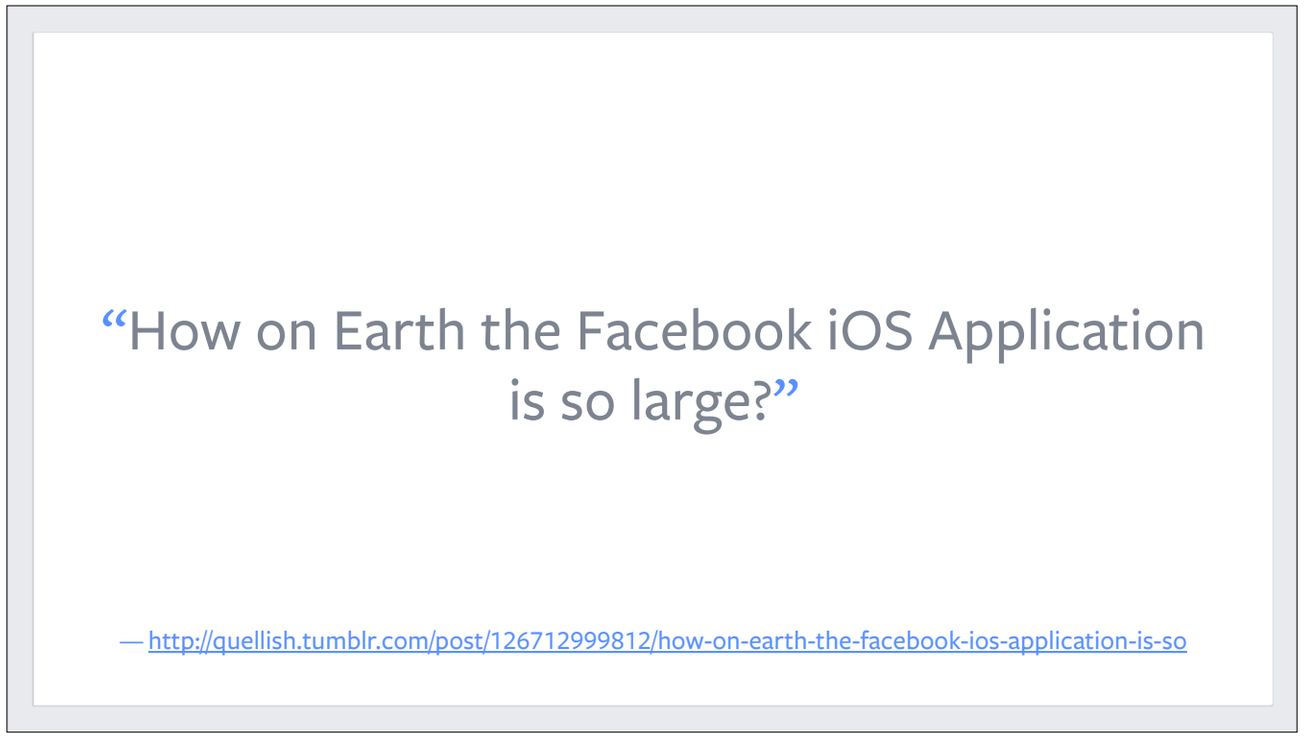
Software Engineer, Facebook London

Thank Chris/Neal
Introduce myself - quick bit about Protect and Care

We prevent bad stuff from happening on Facebook:
- Spam, revenge porn, child exploitative images, etc

And make sure people feel cared for when bad stuff does happen:
- Reporting flows, etc

I work on iOS within Protect and Care, so I find myself spelunking through various Facebook iOS codebases; the main Facebook app, Messenger, and a handful of others. So today I'm going to talk a little bit about what iOS engineering at Facebook looks like.



But let’s start by addressing the elephant in the room. Like, literally, the elephant.

Many of you probably saw this blog post when it did the rounds a couple of weeks ago. It even featured in the “and finally…” slot in the iOS Dev Weekly newsletter, which is a rare honour indeed.

The blog post was a response to a question on Reddit, asking why the Facebook app binary was so large. So this person did a class-dump on the binary to find out more.

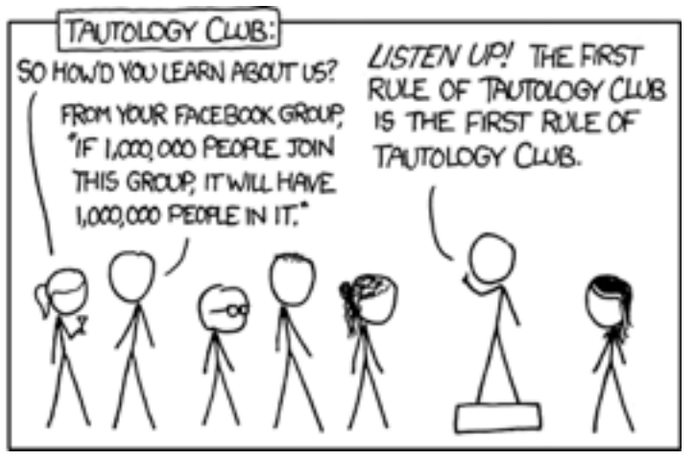
I was pretty excited when I saw this. I’ve wanted to know the answer since I joined Facebook last year, so I read with great interest!

> “There are more than 18,000 classses in the application [...] There is a LOT of crap in there. Even a “FBFeedAwesomeizer” - which alone is a collection of 74 classes and protocols. This is why the application binary itself is over 114Mb.”
>
> — http://quellish.tumblr.com/post/126712999812/how-on-earth-the-facebook-ios-application-is-so

So I scrolled (and scrolled, and scrolled) to the bottom for the conclusion.

And here it is. [Read slide.]

But it left me wanting more.



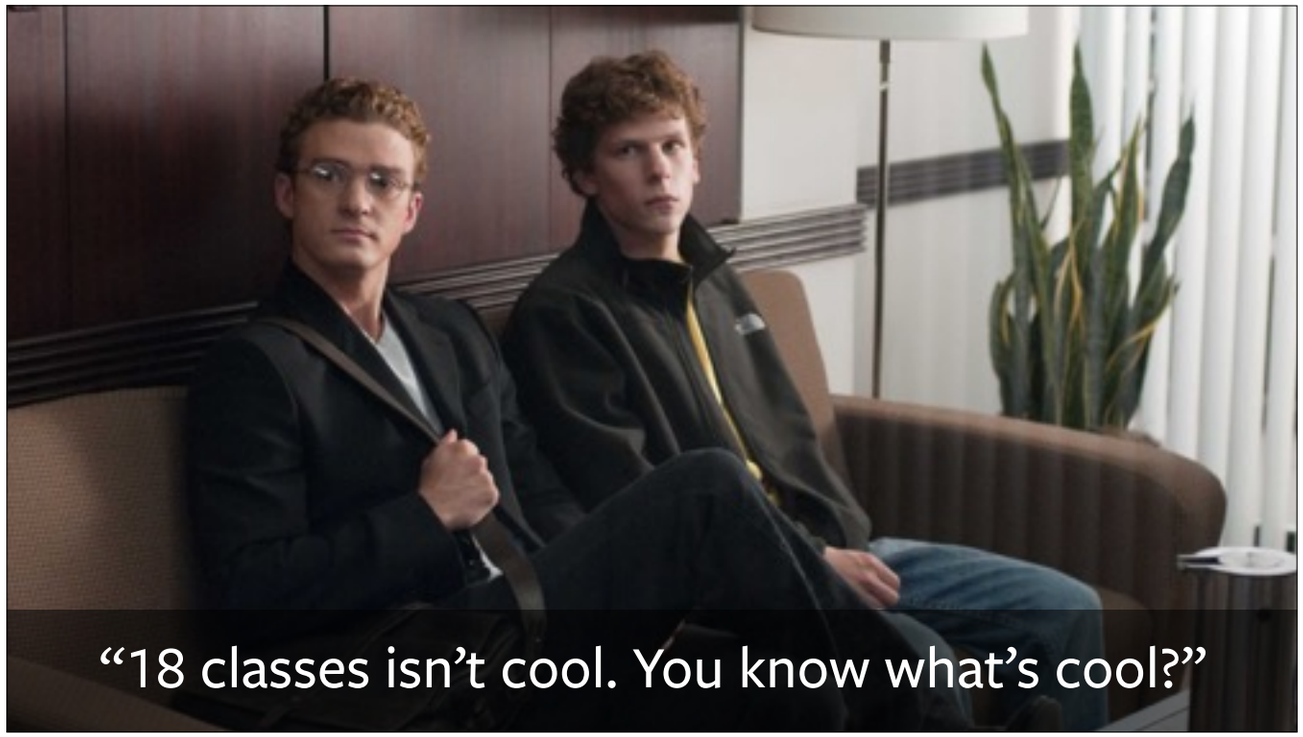
xkcd.com/703/

I mean, “the app is large because it has lots of classes” is something of a tautology. And honestly quite a relief, considering the alternative…

So I suppose the real question is,*why* are there 18,000 classes?

So in the next 18,000 slides I’m going to go through those classes for you. Are you sitting comfortably?

Well one theory I had was that someone might have goaded Zuck into it…



But clearly this isn’t the case; Justin Timberlake left Facebook long before we started writing Objective-C.

So if it’s not down to Justin Timberlake, then why?

Well, it helps to consider Facebook’s culture. The first clue is in the fact that Facebook has managed to scale to 1.5 billion people on…

PHP! Seriously! Clearly this is a company with a pragmatic approach.

There are plenty of people, most of whom wear a suit and charge more for an hour's work than you do for a week, who would tell you that you couldn't possibly scale to that size without an enterprise framework, something like J2EE or .NET. And yet Facebook scaled to that size on the back of the clowniest programming language known to man.

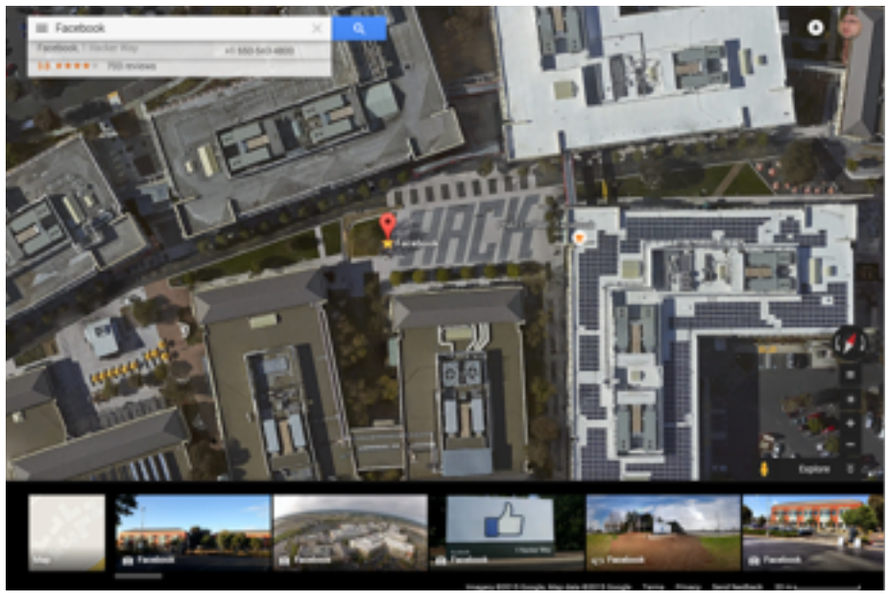
Although to be fair, Facebook don't use vanilla PHP any longer. We invented our own language that looks like PHP but has nullables, type hinting, generics and better collection classes. And we called it…

Hack. And that also gives you a bit of a clue regarding Facebook's culture. Only a certain type of company would invent a PHP-like language and call it Hack.

And lest you think this name was just a fun in-joke among a couple of developers - does anyone know the address of Facebook's headquarters in California?

1 Hacker Way, Menlo Park, CA

1 Hacker Way. I’m pretty sure it wasn’t called that when Sun Microsystems owned the campus. And that campus…



There’s an open square in the middle of the campus, and there’s a single word written there, visible from low orbit.

Hack.

OK, the hacker way isn’t just some idea that Facebook has been running with for a few weeks – it’s deeply ingrained in the way we develop software.

The hacker spirit is exemplified in posters that pop up all around the office like this one

And this one: the famous Facebook slogan.

18,000+ classes

All different, all special

- Duplication?
- Old code?
- Awesome stuff?

So why 18,000 classes?

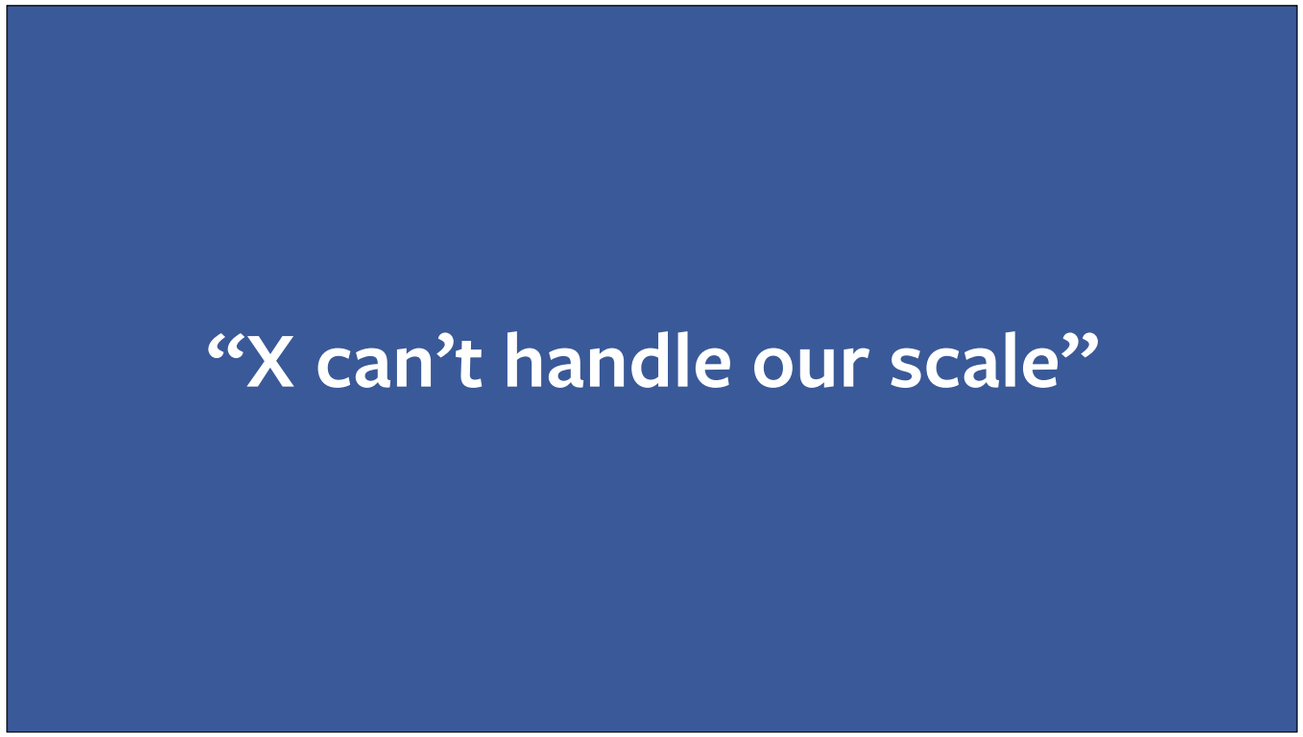
We're an engineer-led company, and at our core is the belief that hacking on things yields fast, awesome results.

We don't have software architects, at least not that I've found yet.
We don't have a committee who decides what can and can't go into the app.

Duplication: almost certainly.
- Two engineers writing essentially the same thing for two different teams - I'll show you an example of this later
- Intentional: A/B testing
Old code? Of course. Chat heads is an example.
Awesome stuff? Yes! e.g. Lots of really nice UI work, pushing the boundaries of what's possible in iOS.

There's also a fair amount of code that re-implements stuff that Apple gives us for free, and that most apps don't have to re-implement themselves.

So, why would we re-implement stuff Apple has already provided?



Next, let’s talk about scale.

“X can’t handle our scale” is a phrase you hear a lot at Facebook. It explains something of the culture, and also some of the binary size. It often leads to us having to reinvent a lot of stuff.

But you might be wondering: how does this assertion even make sense in the context of a mobile app? Mobile is the very opposite of “programming at scale”. Surely the scale is all on the server side.

Well, yes and no.

We’re not talking about scaling load here.

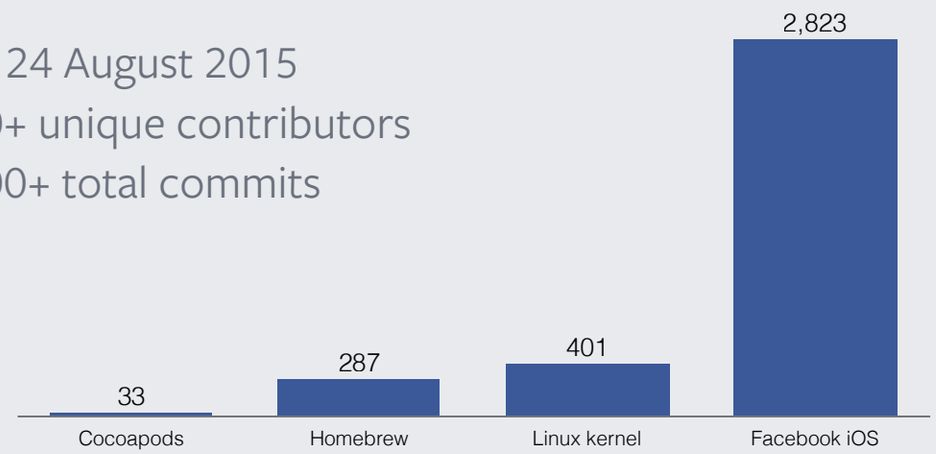
We’re talking about:

- The scale of our employee base: when hundreds of engineers are all working on the same codebase, some stuff doesn’t work so well any more
- The scale of the app’s complexity: like the increasing complexity of the UI widgets that appear in your news

Source control at Facebook

iOS codebase

- w/c 24 August 2015
- 400+ unique contributors
- 2800+ total commits



Here are the stats for the week starting 24 August this year.

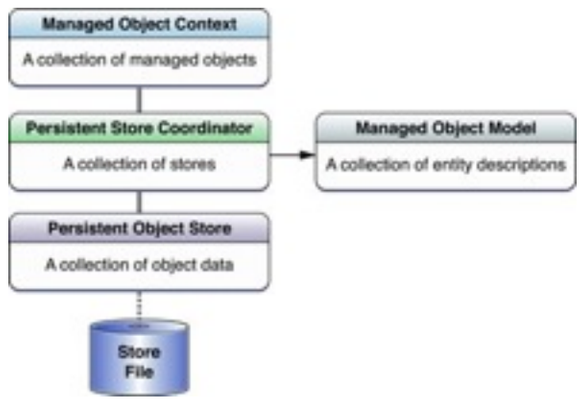
In that week:

429 unique contributors
2823 total commits

Doesn't include commits to bump version numbers, add translated strings etc - actual number is over 4,000.

So it's big. It's an order of magnitude bigger than the same figure for the Linux kernel in the same week, and bigger again than popular repos like Homebrew and Cocoapods.

It is frankly the biggest, busiest codebase I've ever seen. And LOTS of things can't handle this kind of scale.



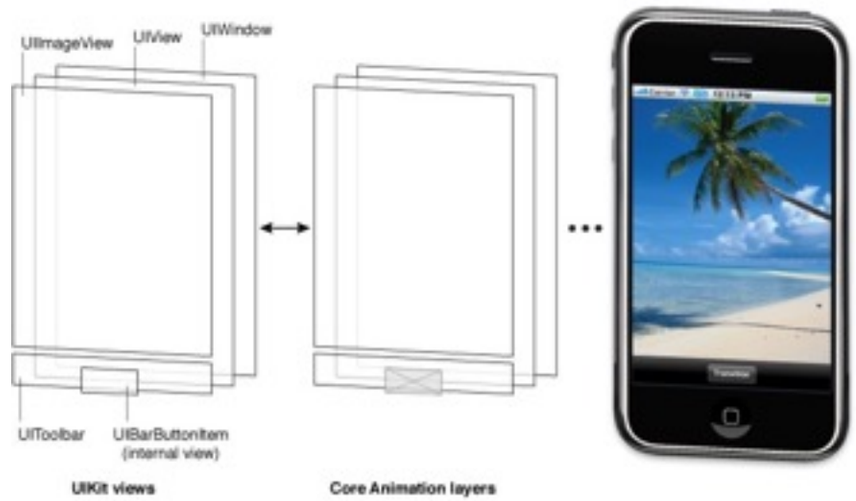
https://code.facebook.com/videos/859635244094281/f8-2015-facebook-on-ios-inside-the-big-blue-app-/

Core Data can’t handle our scale.

A couple of performance issues, but mostly this is around having a framework that vends mutable objects and relying on hundreds of developers per week to do the right thing with that mutable state.

So instead we use a framework developed at Facebook called Mem Models, which vends *immutable* objects and has some clever voodoo to handle the case when those objects genuinely do change (e.g. someone likes a new page).

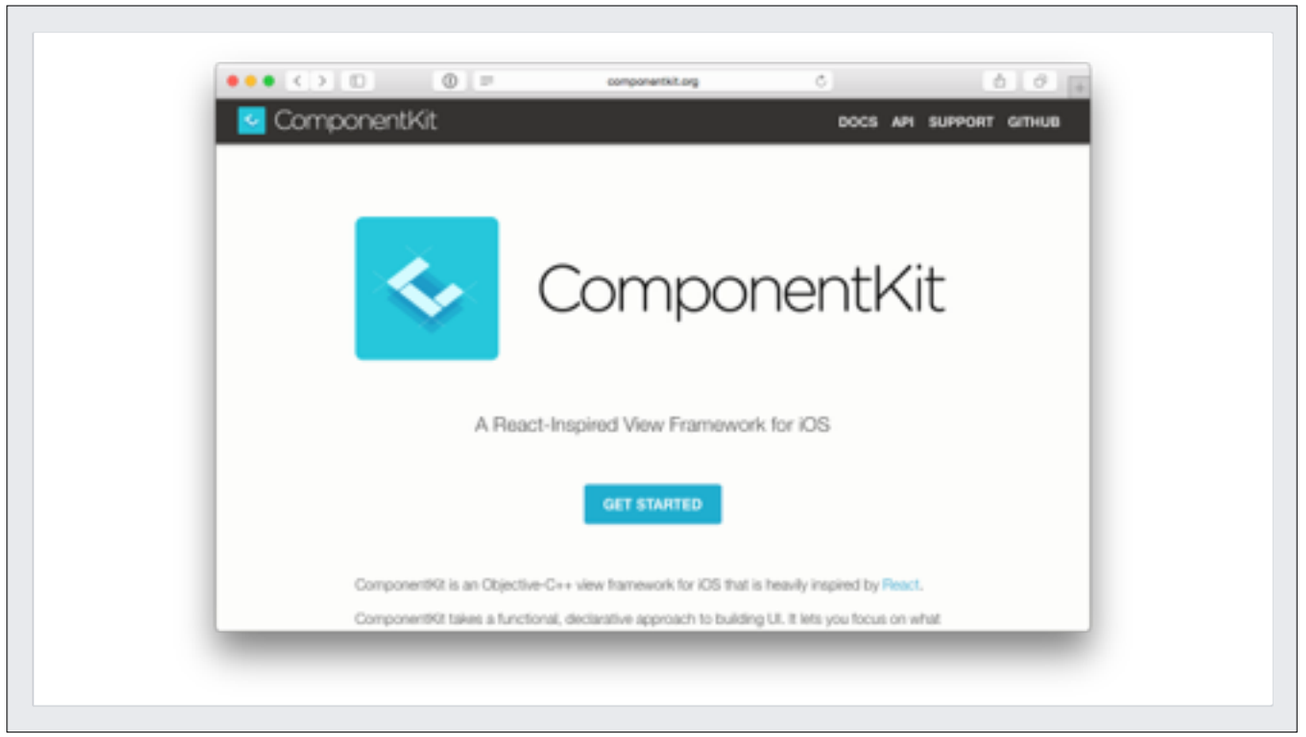
So going back to the 18,000 classes, there’s a bunch of classes in there implementing mem models, which those of you using Core Data don’t need to worry about.



https://code.facebook.com/videos/859635244094281/f8-2015-facebook-on-ios-inside-the-big-blue-app-/

UIKit can’t handle our scale.

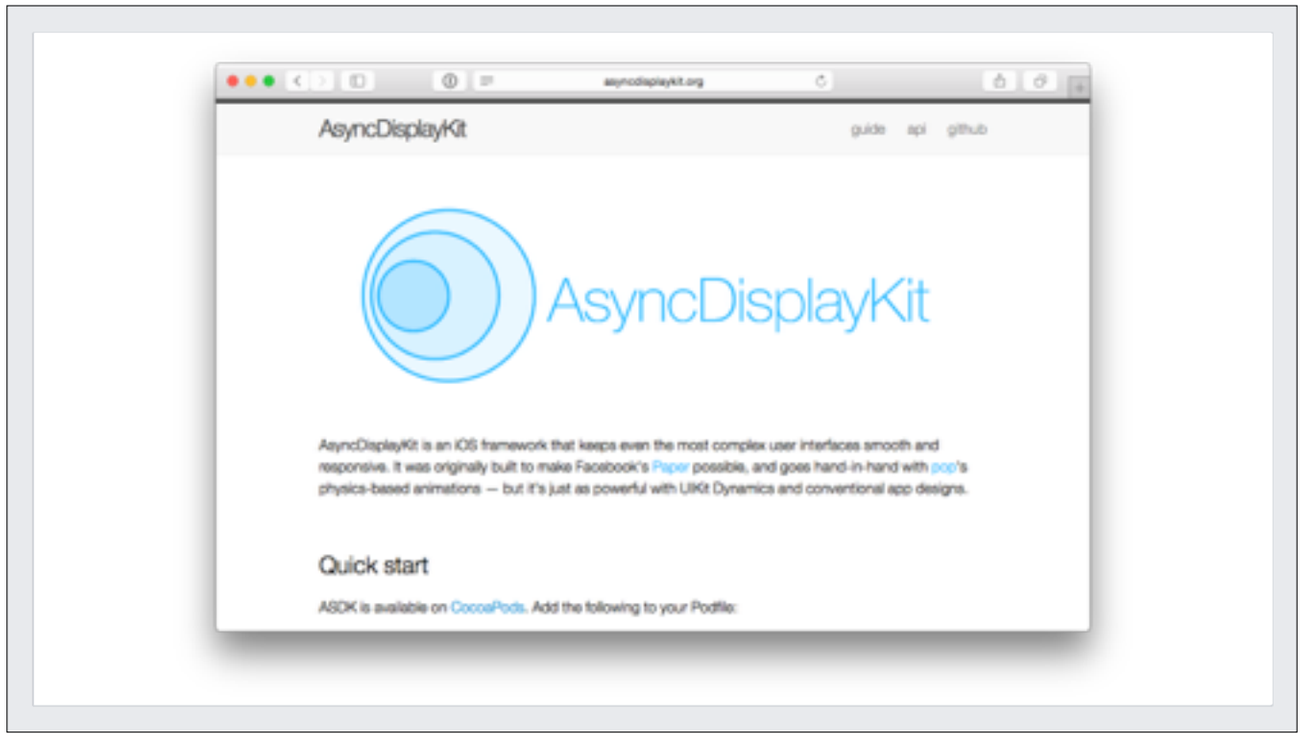
Specifically, since so much of the layout in UIKit happens on the main thread, it’s a bad fit for news feed, where we need to lay out increasingly complex feed units really quickly as people scroll.



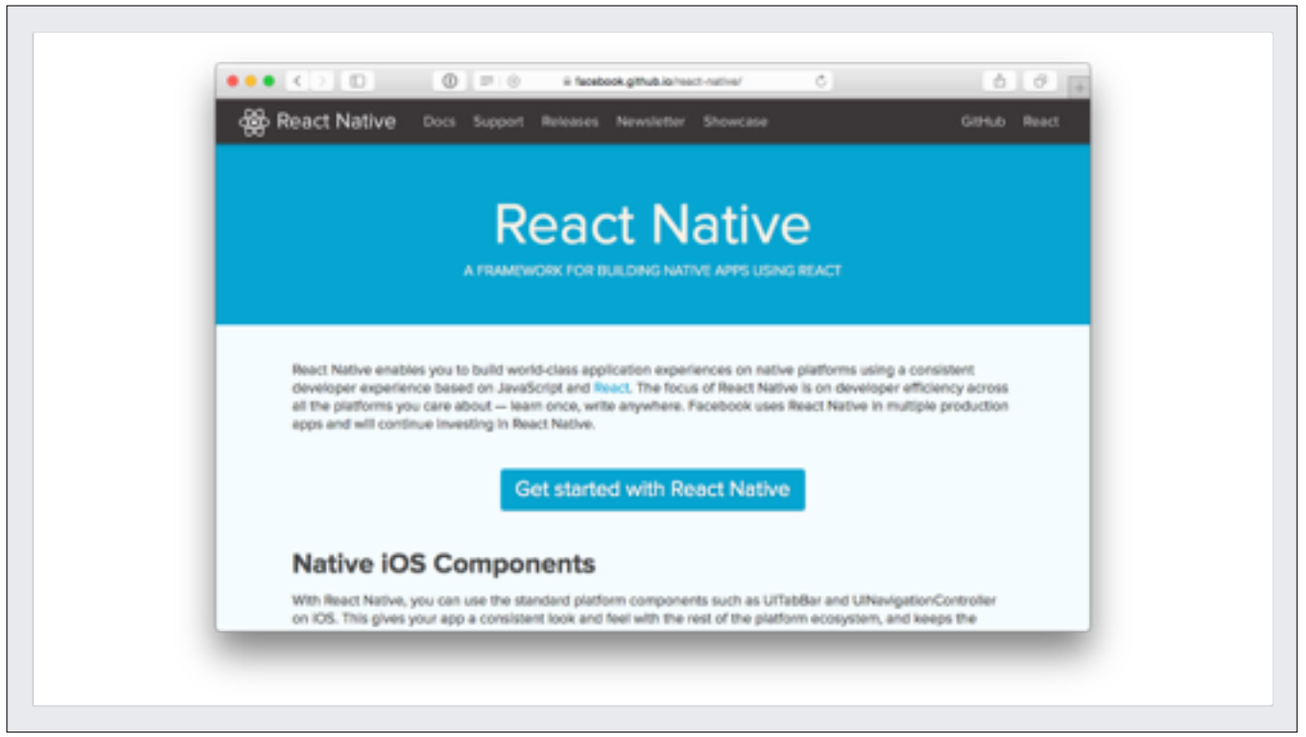
So we wrote (and open-sourced!) ComponentKit, which takes a functional, declarative approach to building UI.

All layout is done on a background thread, making 60fps achievable even with complex UIs on older devices.

And because this is Facebook, we have at least two other UI frameworks offering alternative takes on background thread compositing.

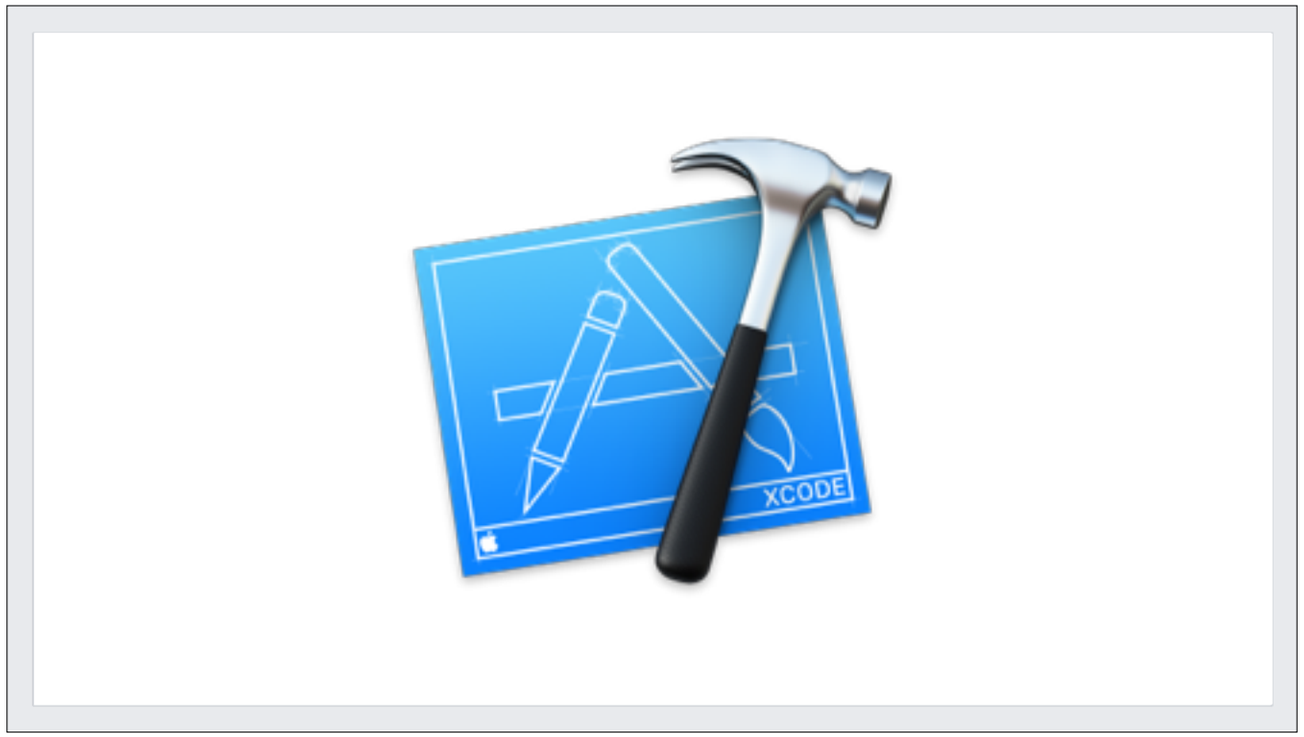


AsyncDisplayKit was written for Facebook’s Paper app



React Native focuses on developer efficiency across all the platforms you care about — learn once, write anywhere.

For the same reason, we don’t make much use of Auto Layout; it has unpredictable layout scheduling and does its work on the main thread, so it’s not a good fit for news feed and other parts of the app where 60fps is important.

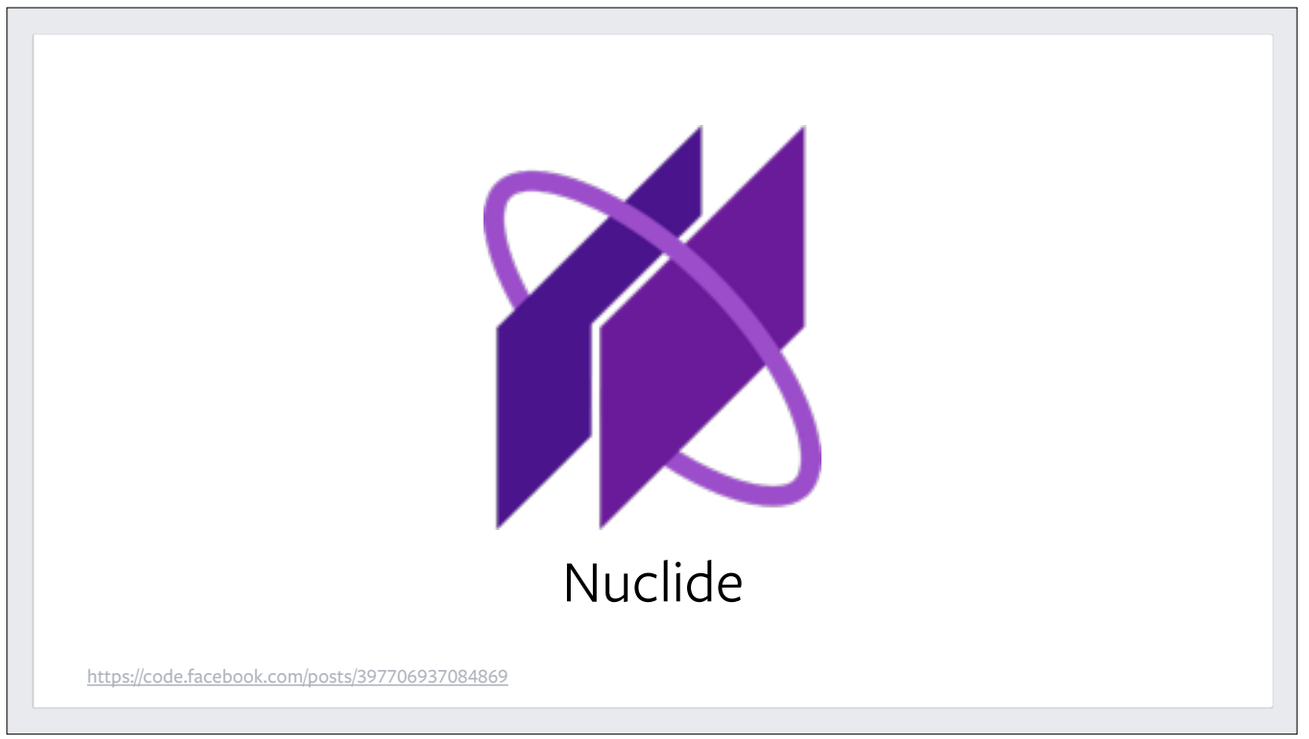


Xcode can’t handle our scale!

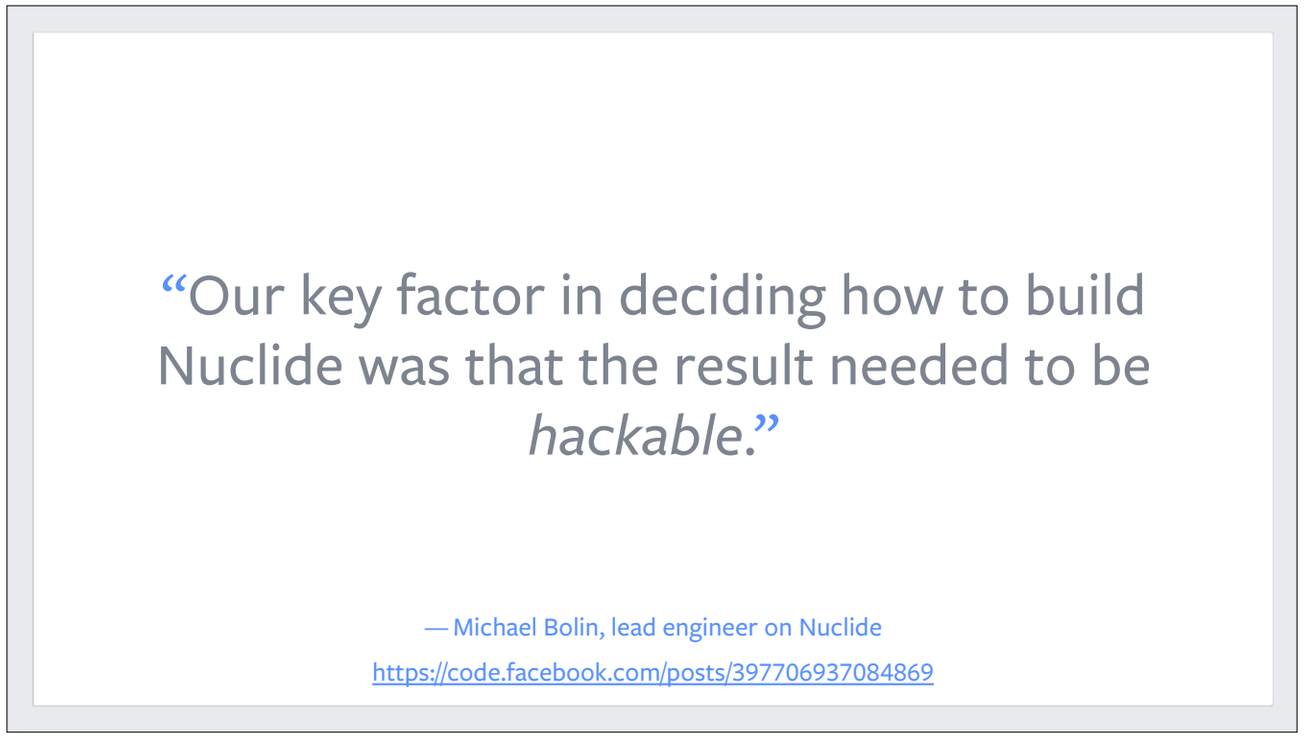
http://xkcd.com/303/

18,000 classes takes a really long time to load and compile!

Now you might think the solution here would be to simply remove some of those 18,000 classes. In which case I would humbly remind you that we are hackers. So instead…



We wrote out own IDE. Seriously. It’s called Nuclide, it’s based on Github’s open-source Atom text editor.



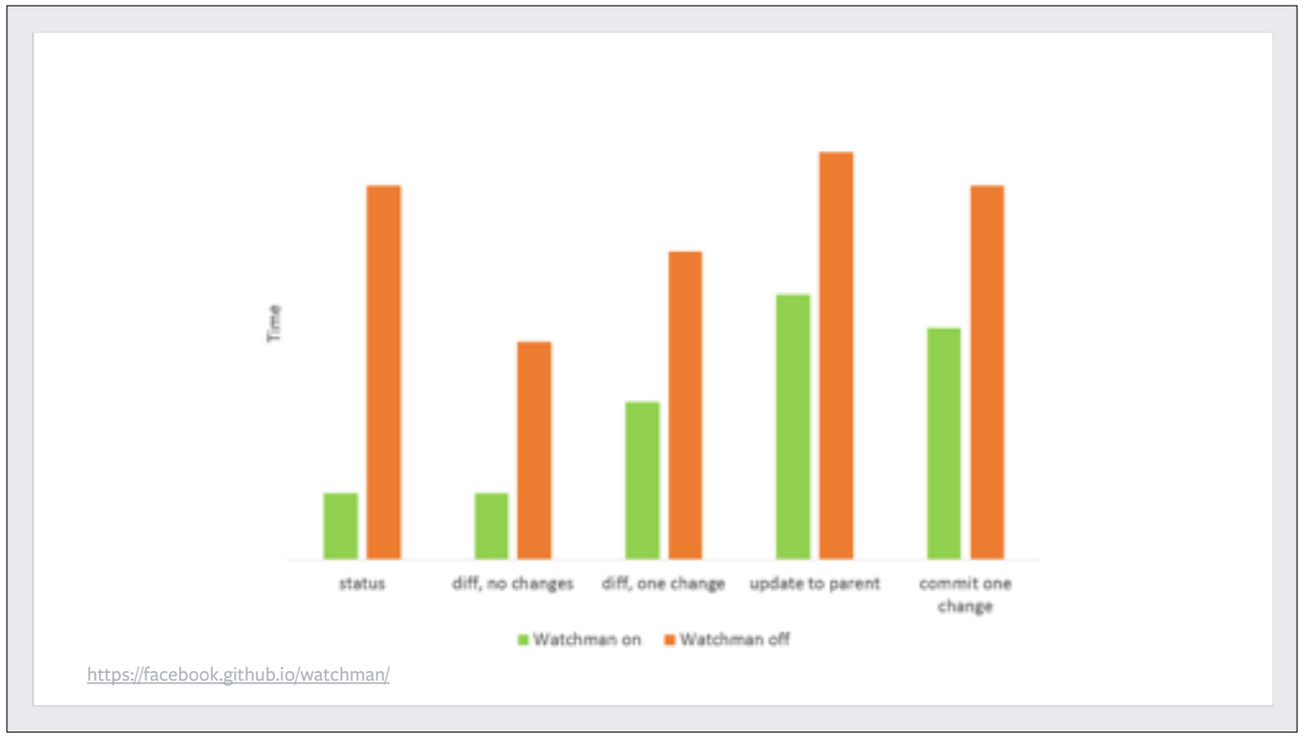
And here’s an interesting quote from Michael Bolin, the lead engineer on Nuclide. Once again, it’s all about hackability.

OK, so:
- Core Data can’t handle our scale
- UIKit can’t handle our scale
- AutoLayout can’t handle our scale
- Xcode can’t handle our scale

What else can’t handle our scale?

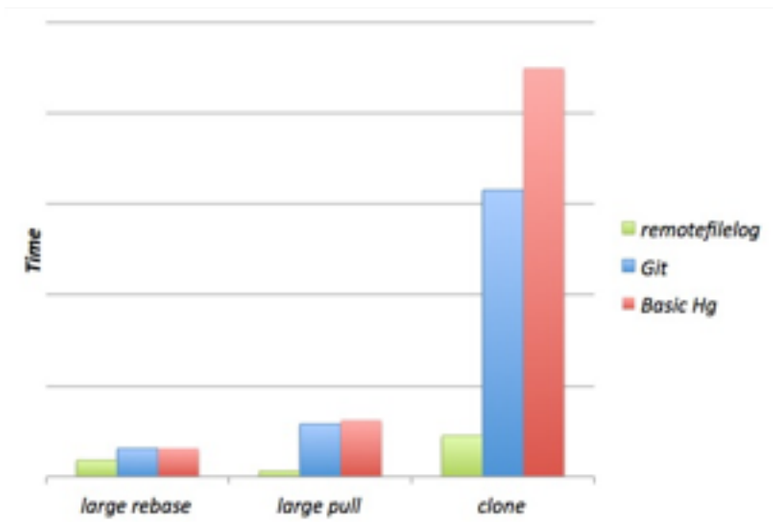
Git can’t handle our scale!

Mercurial works better for us. It’s more hackable, so we’re more easily able to dig in and make changes that suit us.



Using watchman, a local file system monitor that we wrote and open-sourced, we improved performance of operations that need to determine which files in a directory tree have changed.

Things like:
- hg status
- hg diff
- hg update (like git checkout)
- hg commit



https://bitbucket.org/facebook/remotefilelog

And using remotefilelog, a Mercurial extension that we've also open-sourced, we reduced network I/O by only pulling down metadata for the majority of commits when pulling or cloning, then fetching the file data on demand.

So that's it; a brief glimpse into just some of the reasons that we might have ended up with 18,000 classes in our iOS app.

Open source at Facebook

So, we have a lot of unique challenges at Facebook, and we open-source a lot of our solutions. I just want to highlight a couple of other open-source projects of ours that you might not be aware of.

Infer

http://fbinfer.com/

A tool to detect bugs in Android and iOS apps before they ship. It reports memory leak problems in Objective-C and C code. We run Infer as part of our continuous integration suite.

pop

https://code.facebook.com/projects/642915749111632/pop/

An extensible animation engine for iOS and OS X. Written to power the animations in Facebook’s Paper app.

brew install chisel

https://github.com/facebook/chisel

Chisel: a collection of LLDB commands for debugging iOS applications.

Loads of handy stuff like pviews (prints the current view hierarchy), pvc (prints the current view controller hierarchy), flicker (quickly show and hide a view to visualise where it is) and many more.

So low-level it doesn't have an icon, so I designed one.

brew install chisel

https://github.com/facebook/chisel

Good eh?

code.facebook.com/ios

- Blog articles
- Videos (including “inside the big blue app”, which talks in more depth about mem models and component kit)
- Open source stuff

Welcome to Clowntown

So some of you may now have the impression that Facebook is staffed by superhumans, people who aren't afraid to rewrite iOS from the ground up to squeeze that last bit of performance out of the system.

People who never make mistakes.

Let me disabuse you of that notion.

At Facebook, we talk of a mystical land called Clowntown. Many are called but few are chosen. Clowntown is the place you visit…

```
// Hack to support launch, will fix next week
```

… from 2013

…when you write comments like this – and someone finds them two years later

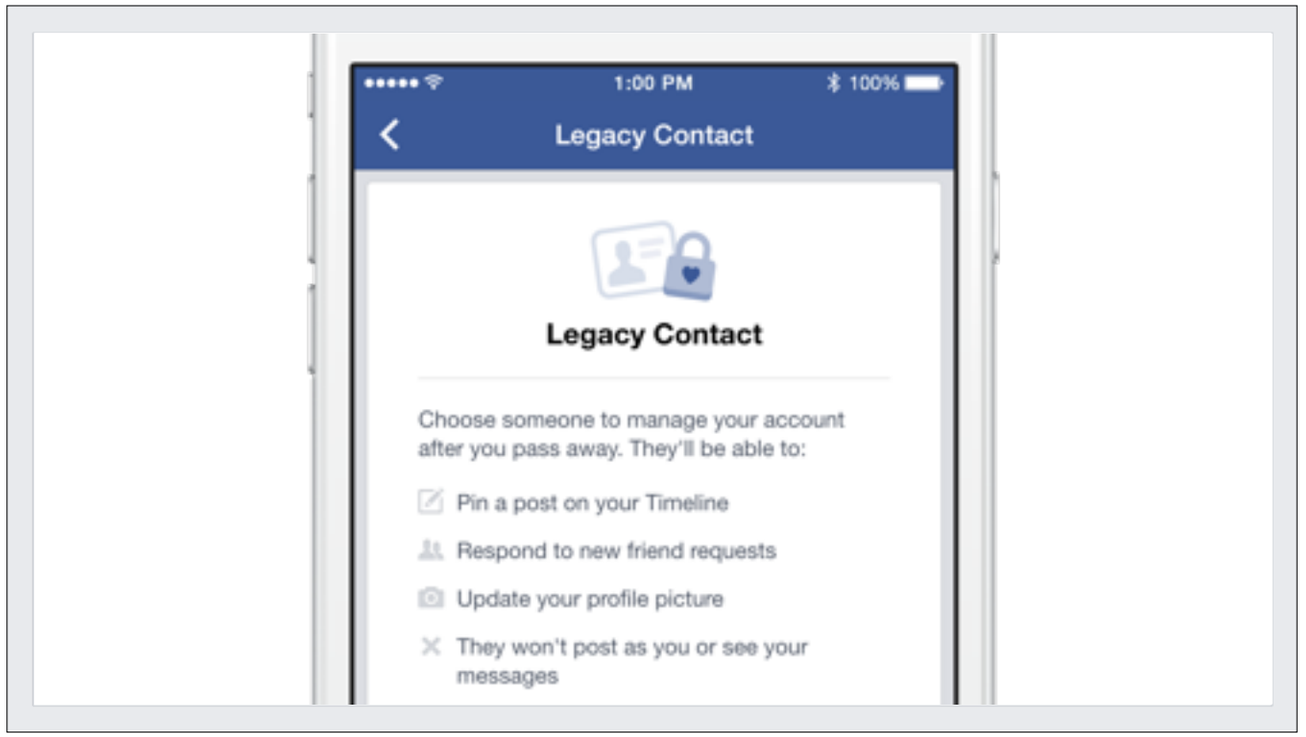
```
@implementation RCTMethodArgument

- (instancetype)initWithType:(NSString *)type
                  nullability:(RCTNullability)nullability
                      unused:(BOOL)unused                    ⚠ Unused parameter 'unused'
{
  if ((self = [super init])) {
    _type = [type copy];
    _nullability = nullability;
  }
  return self;
}

@end
```

or when your unused parameter called unused is unused.

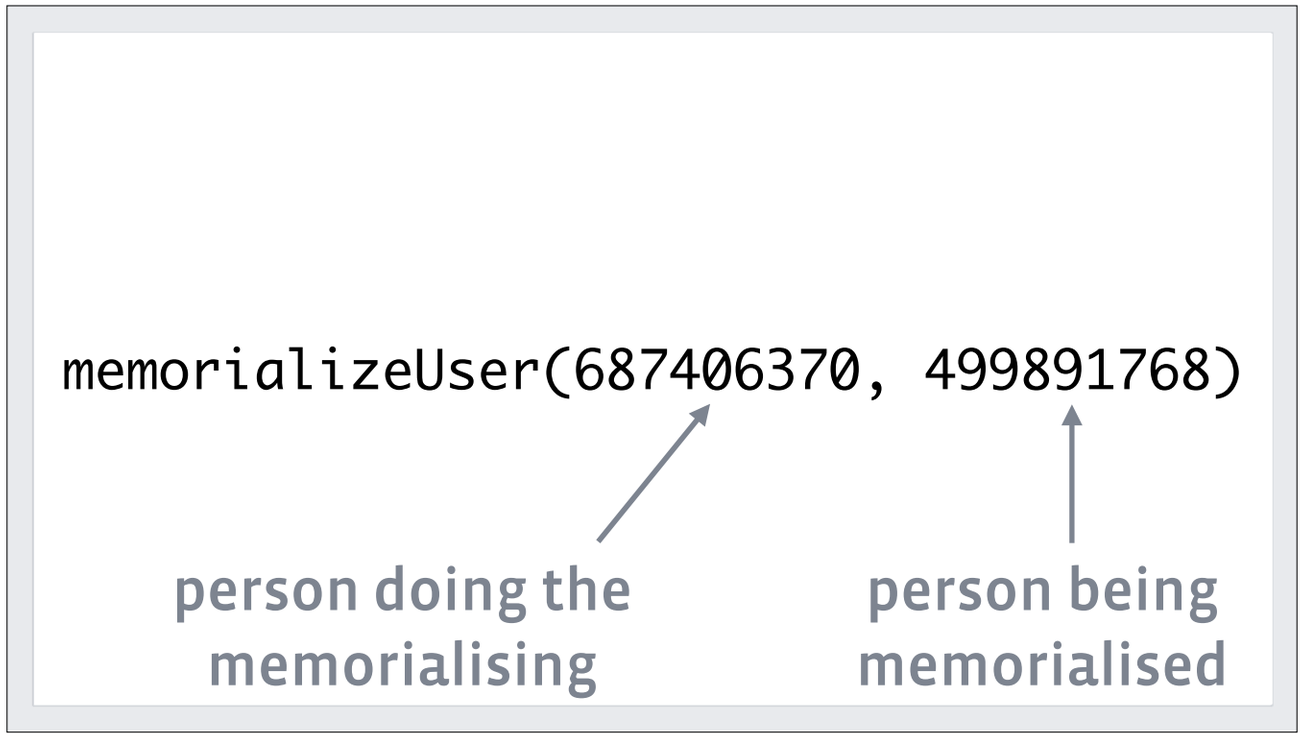
But some of us aspire to greater things. Some of us want to be the Mayor of Clowntown. And for that, a mere comment won’t do.



Earlier this year I worked on the iOS implementation of a feature called legacy contact. It's where Facebook lets you nominate a friend or relative to manage your account after your death. Initially I just rushed out a quick hack to get the basic functionality in place.

Then this summer I had an intern come to work with me for 3 months, so I tasked her with re-writing my hack using lots of nice new stuff like ComponentKit and make it properly production ready.

As part of her work, she needed to test her code. And to test her code, she needed create a dead test user.

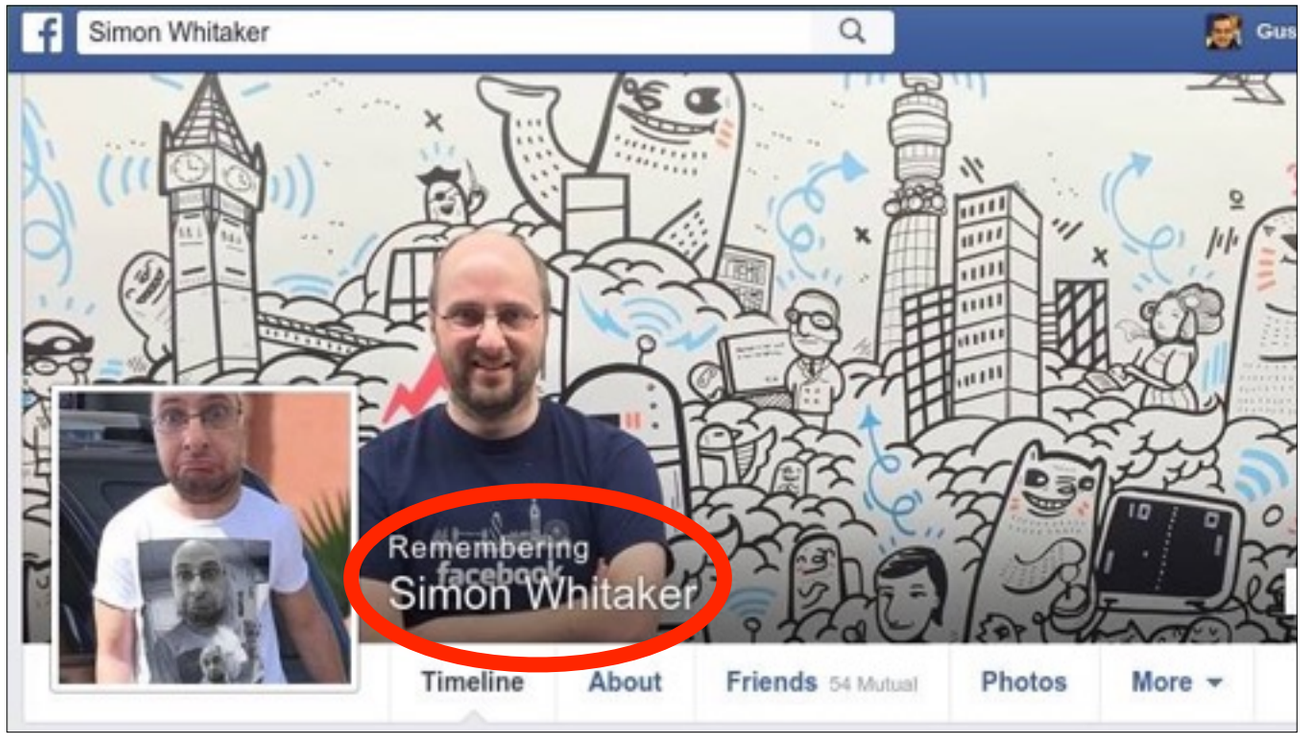


There is a UI for memorialising users, but I assured her that the pros simply ran a bit of code in the PHP debugger. There's a function that takes two parameters: one the ID of the person being memorialised, the other the ID of the person doing the memorialising. I gave her a demo to show her how easy it was.

And that's when I entered Clowntown.

I first realised something was wrong when I went back to farting around on Facebook and got prompted to login.

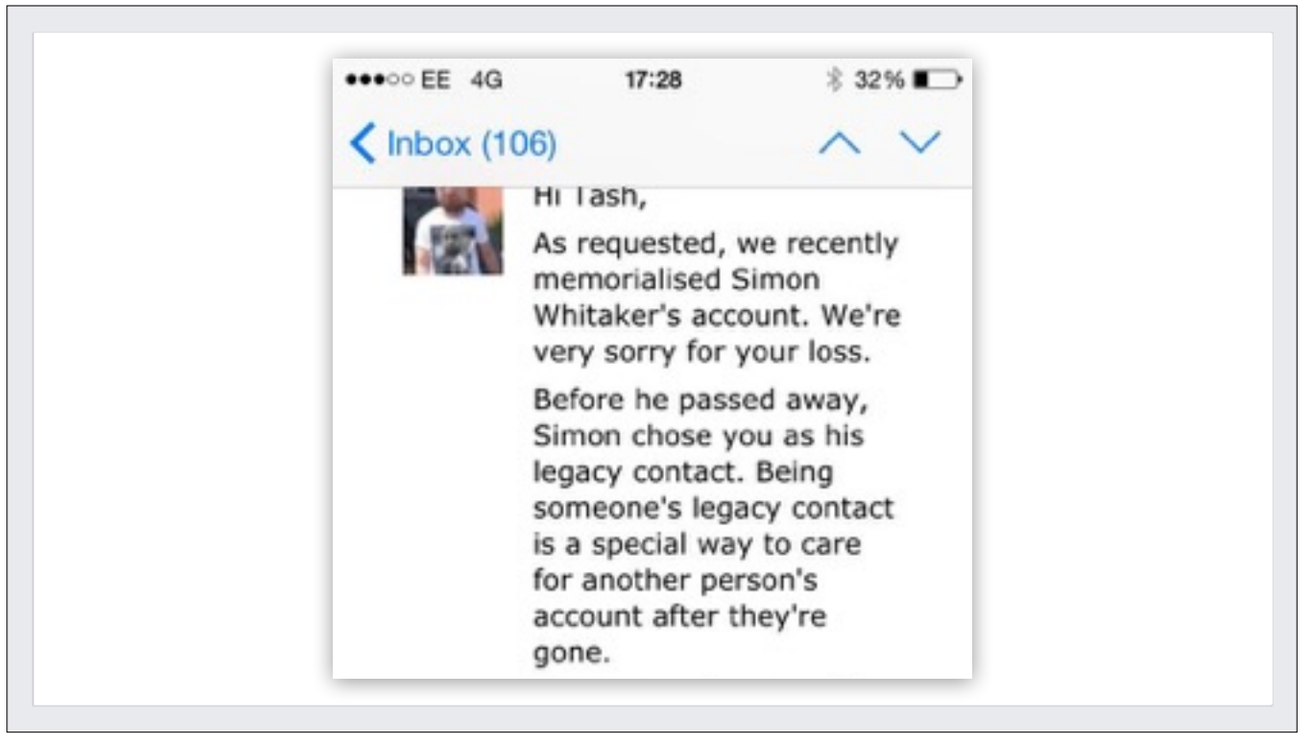
Then when I tried to log in, I was told I couldn’t log in to my account because it had been memorialised.

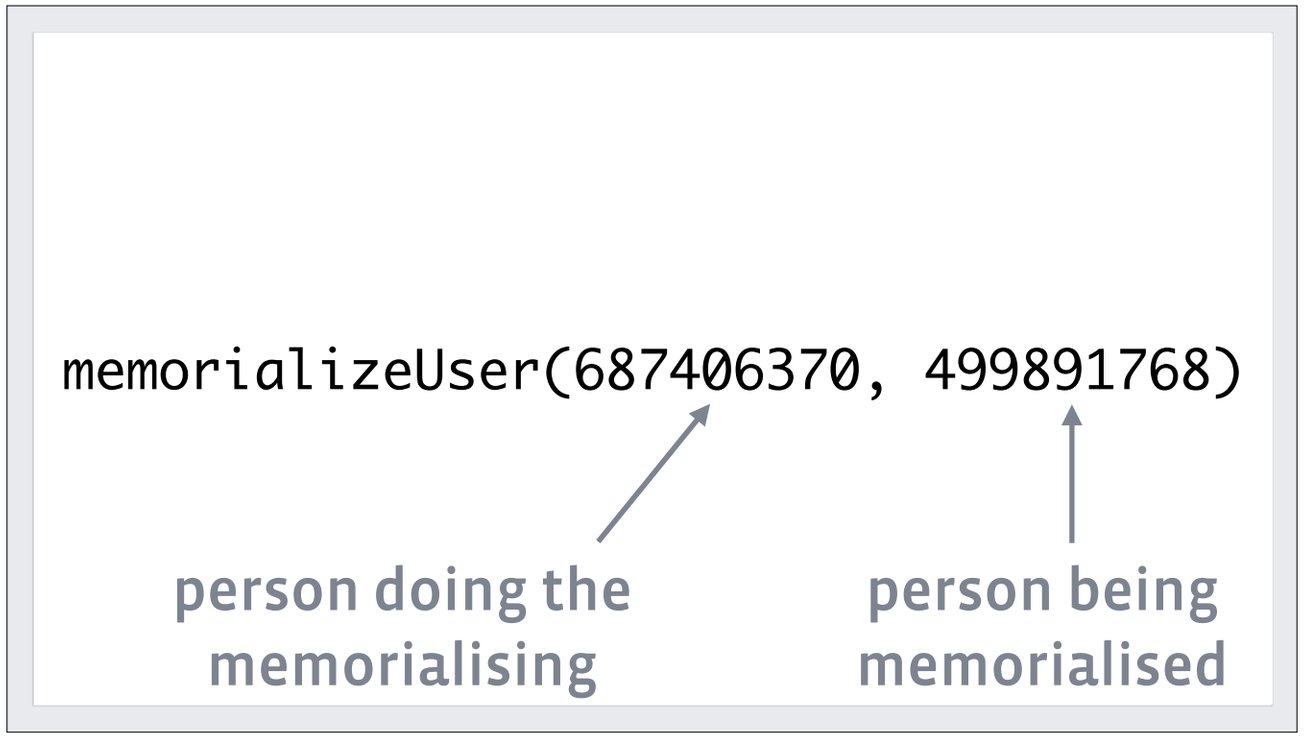


My co-workers started laughing, because they'd discovered that my account now showed "Remembering Simon Whitaker".

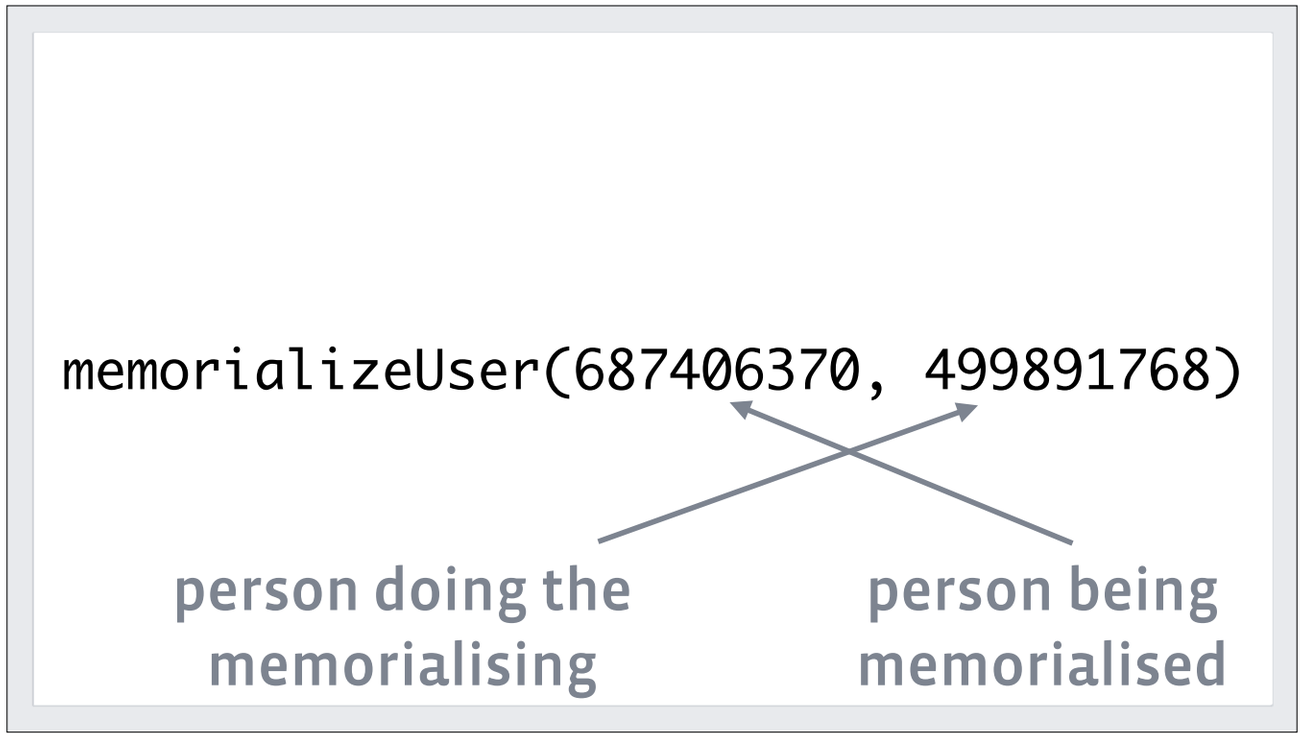
And then I thought; Oh god. My wife's going to get the email.

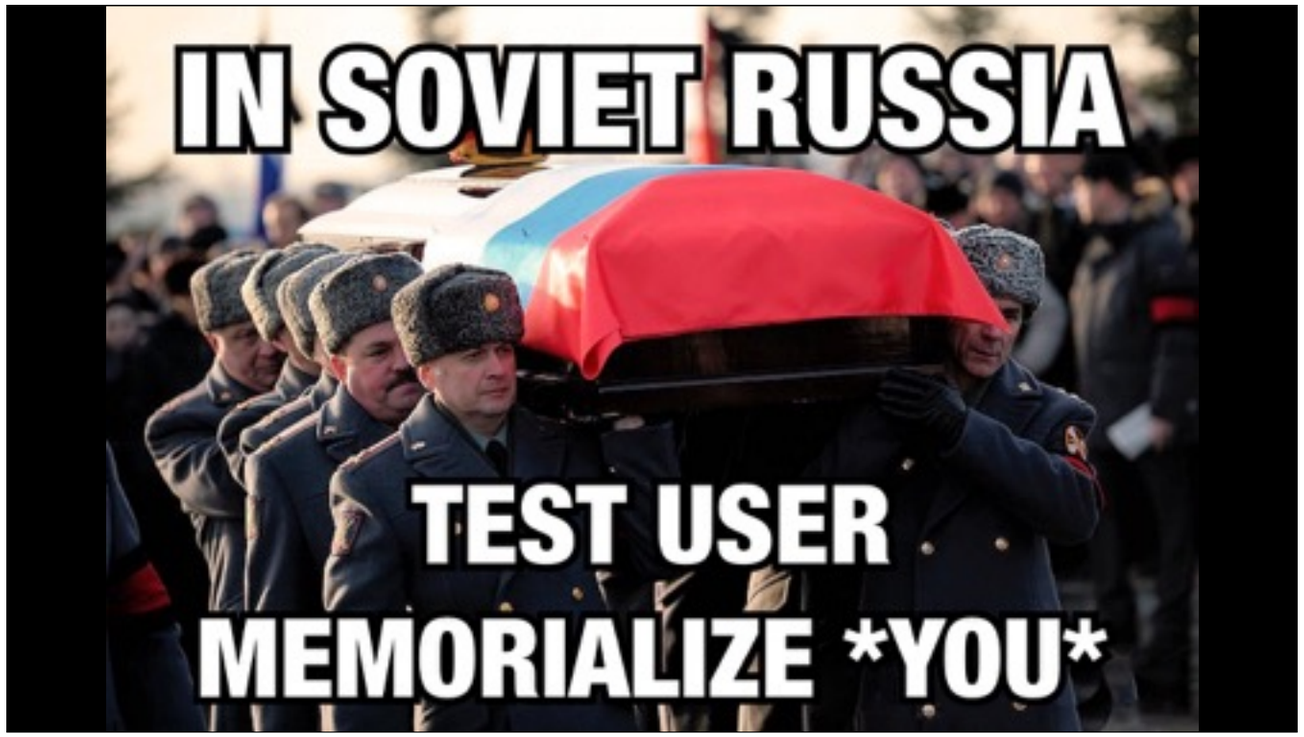
I had set my wife as my legacy contact.

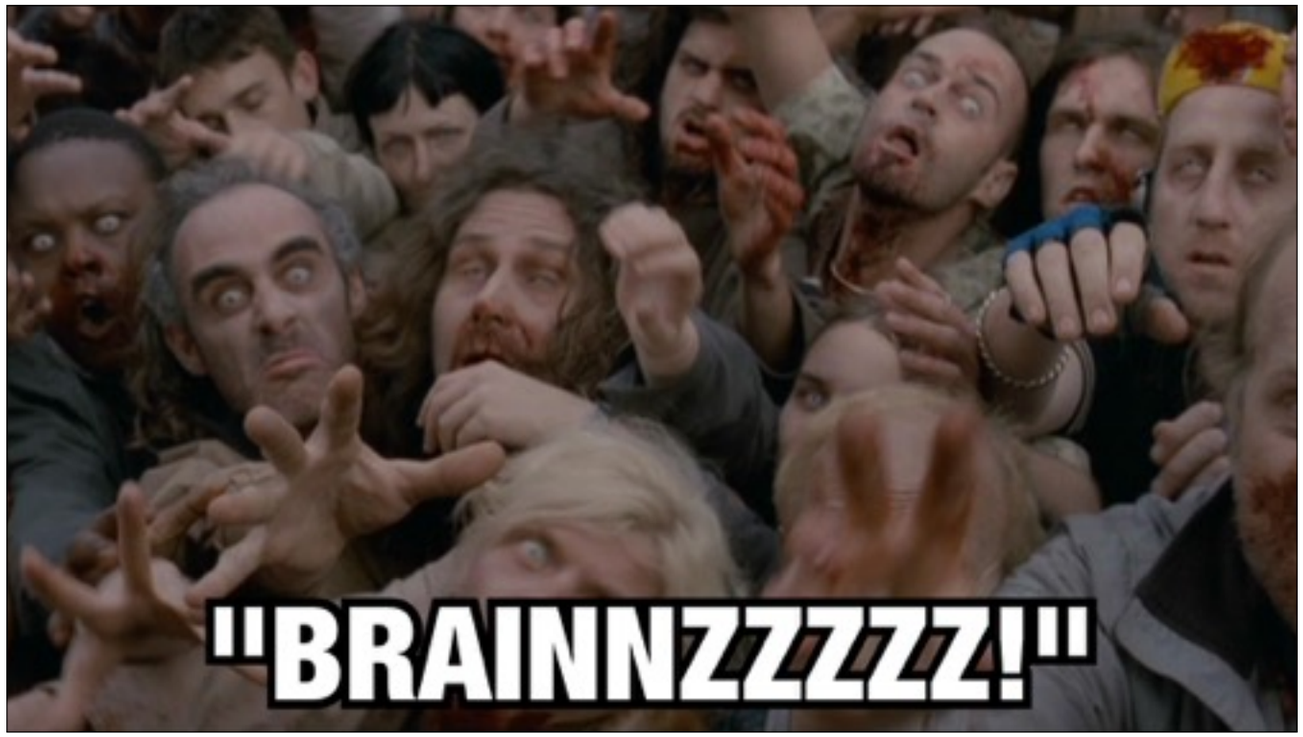

EE 4G
17:28
32%
Inbox (106)
Hi Tash,
As requested, we recently memorialised Simon Whitaker's account. We're very sorry for your loss.
Before he passed away, Simon chose you as his legacy contact. Being someone's legacy contact is a special way to care for another person's account after they're gone.



So in case you haven’t guessed what I got wrong yet, I managed to get the arguments the wrong way round. Instead of me memorialising my test user, my test user memorialised me.


memorializeUser(687406370, 499891768)
person doing the memorialising
person being memorialised


IN SOVIET RUSSIA
TEST USER
MEMORIALIZE *YOU*


"BRAINNZZZZZ!"

Simon Whitaker

Zombie Mayor of Clowntown

Simon Whitaker

www.facebook.com/simonwhitaker
@s1mn

Questions?