EPISODE 775

[INTRODUCTION]

[0:00:00.3] JM: React Native allows developers to build native applications for iOS and Android, using components written in the React JavaScript framework. These ReactJS components render to native application code by going over a JavaScript bridge, a message bus that communicates between JavaScript code and native iOS or Android runtimes.

For most mobile application use cases, React Native works well. In some cases, the platform suffers from performance issues due to the functionality of the JavaScript bridge. For example, mobile games with high demands on graphics and networking and fast real-time updates to the UI, these can stutter when using React Native.

To address the performance issues of React Native, the core team working on React Native at Facebook is rearchitecting the React Native runtime with a project called Fabric. Fabric consists of changes to the threading model, the data handling system and the JavaScript bridge.

Chris Severns and Lee Johnson work at G2i, which is a group of React and React Native specialists. Chris and Lee join the show to discuss the rearchitecture of React Native, including the engineering history of React. They also talk about the technical debt within the React Project and the vision that the React team has for the future.

In addition, we talk about Google's Flutter Project; a cross-platform native framework with a different architectural model than React Native. It was a great conversation with Chris and Lee. I hope to do it again some time.

[SPONSOR MESSAGE]

[0:01:51.3] JM: MongoDB is the most popular, non-relational database and it is very easy to use. Whether you work at a startup or a Fortune 100 company, chances are that some team or someone within your company is using MongoDB for work and personal projects. Now with

MongoDB Stitch, you can build, secure and extend your MongoDB applications easily and reliably.

MongoDB Stitch is a serverless platform from MongoDB. It allows you to build rich interactions with your database. Use stitch triggers to react to database changes and authentication events in real-time. Automatically sync data between documents held in MongoDB mobile or in the database back-end. Use stitch functions to run functions in the cloud.

To try it out yourself today, experiment with $10 in free credit towards MongoDB's cloud products; MongoDB Atlas, the hosted MongoDB database service and Stitch. You can get these $10 in free credits by going to mongodb.com/sedaily. Try out the MongoDB platform by going to mongodb.com/sedaily, get $10 in free credit. It's the perfect amount to get going with that side project that you've been meaning to build.

You can try out serverless with MongoDB. Serverless is an emergent pattern. It's something that you want to get acquainted with if you're a developer, and getting that $10 and free credit to have a serverless platform right next to your Mongo database is really a great place to start. Go to mongodb.com/sedaily, claim that credit.

Thanks to MongoDB for being a sponsor of Software Engineering Daily. I love MongoDB. I use it in most of my projects. It's just the database that I want to use. Thanks to MongoDB.

[INTERVIEW]

[0:03:59.1] JM: I am here with the G2i team. You guys are all React Native specialists and I'm looking forward to having a conversation about React Native. Welcome to Software Engineering Daily.

[0:04:10.0] CS: Cool. Thanks for having us on, Jeff.

[0:04:12.3] JM: The main topic today that I want to discuss is React Native and its rearchitecture that it's undergoing. Why is React Native undergoing a rearchitecture?

[0:04:24.2] CS: Well, I think there's a lot of things that they're trying to accomplish with the rearchitecture. It might be helpful to start by going into explaining some of the different pieces of that. Actually, Rom had a good blog post that was a glossary of terms that explained all the different initiatives that they're doing from the lean core project to Fabric and the JSI. At the high-level part of why they're redoing this is to really make React Native more maintainable moving forward by using lean core, moving a lot of the modules that are currently in the React Native package out to the community. It makes it easier for the core team to focus on what's really important inside of React Native and for the community to be able to support the extra pieces such as the web view when needed.

They've seen a really great just response and show of support from the React Native community and supporting that web view component, which is really cool and I think really encouraging for the future of React Native and just where it's headed. Then when you look at the other pieces such as the JSI, it's really about improving performance and making React Native just continue to feel more and more like a true native experience and remove the so-called Jenkiness that you sometimes get with having to travel over the bridge.

I'm sure Lee and Andrei can speak personally to some of the experiences they've had with React Native on that front.

[0:06:02.0] JM: Yes. Can you guys talk about the JavaScript bridge and the associated Jenkiness?

[0:06:08.2] LJ: Jenkiness. That's a technical term, right? I think one important note to the rearchitecture is just the fact that Facebook uses this all the time and they have big boy problems, right? They have lots of data and lots of users and they're using React Native and some key things. The rearchitecture is from pain points and issues and things that they have learned through using it at a level that most people will never approach, right? They have so much just bandwidth and things going on. Then they're proving it as they're working on it.

Then the bridge is a lot of that and JSI is a lot of that rearchitecture. In the bridge, one of the first things I did in React Native was write a bridge. It was because in the early days, you couldn't

just NPM install modules and they work. Most things had not been created for React Native, because it was a new platform.

We had a messaging SDK as a messaging service where we're using – they had an iOS SDK, but there was nothing for React Native, because React Native was new, so we had to integrate that into our application through the bridge. Basically what it is is you have your native side, which is iOS in my case. Then you have your JavaScript side, which is your React.

For those things to talk together, they have to communicate over the bridge. It's typically a asynchronous skewed message, so our app would talk to that messaging app and say, "Hey, send a message." Then iOS would do its thing and tell our JavaScript when it was done and say, "Okay, I sent a message. Here's the response." It does that ping-pong back and forth. That's the bridge. Then that is actually how it works with almost all of the native elements that you use in your React Native application. It sends a message over the wire, the native code does what it needs to do and then asynchronously sends back a response or data or whatever.

Then you say, you can send multiple messages and it will queue them and do them as it can, which was great. I mean, when it came out it was great because those two things ran on different threads. Your interface is extremely smooth, because it's not stuttering and blocking as your JavaScript is calling and doing these things, which before we did a lot of things in PhoneGap, or Cordova, or different methods like that and that was the whole problem was the feel, right? The stuttering just webby feel that you got, it didn't feel like a native application.

That's what React Native really solved for us in the early days was like, "Man, this feels like a native application, because it doesn't have those pauses." Every time you do something in your "web code" in PhoneGap, like scrolling. It never scrolls smoothly ever. Didn't matter how many – it would always stutter and act funny and it's just something you had to deal with. Then React Native came along and put them in their own separate buckets and it's like, "Wow, this thing looks – that's exactly like a native app."

[0:09:09.1] JM: Right. Now I remember using I think it was the food delivery apps in 2008 or 2009 around the time when all these companies were starting to build their native apps and they were all using these cross-platform things and it was really stuttery. Then React Native came out

and it was it was leaps and bounds ahead of what had come before it. All due respect to React Native despite our criticisms of the current architecture, and I think we should talk about the ramifications of the current architecture.

The current world as I understand it, there are instances where the programmer needs to write native code instead of React Native code. There are certain instances where you really need high performance, and in those cases you will have to write native iOS code or native Android code and that code will run differently than the React Native code itself. Can you describe this dichotomy?

[0:10:14.4] LJ: Sure. I mean, I can or Andrei, if you want to weigh in on that. I know you guys have been working on some of that stuff recently.

[0:10:20.6] JM: Yeah. I just want to illustrate what kinds of performance issues do you have to address by writing native code.

[0:10:26.8] LJ: Yeah. Rom does a really good simple explanation of that. One of his was when you're typing really fast, you know that every keystroke actually goes to the JavaScript side, updates and then that sends back to your native size, so that it shows whatever you're typing on the screen. Then there's a loop there. There's a trip for every one of those. They are smart and did some debalance things inside the actual code, but there's this travel for something really small that can get stutter. That's a weird example.

Another common example is scrolling. You have lots and lots of lots of things on the scroll list and you just fling your thumb and zoom through them. It can't refresh it fast enough to keep up with it. Those are two common areas where the bridge struggles. Then a third would be if you were doing some 2D game type thing, right? If you had an application and had a 2D component, maybe this is Pokemon Go where you're doing this really neat app, map type app thing, but you had other elements that were menu-driven and whatever.

Obviously, that also it requires a lot of render cycles and real-time updating, so all those instances are when you would look at doing native codes. When it comes down to especially game, you don't want any delay, right? You don't want anything in between you and the lower

level, because of performances at a premium. You won't update as fast as possible. You want them instant. Anything that gets in the way of that is going to cause slowdown, or the technical term Jenkiness I guess, is what we're using.

You wanted to write those in native code. In React Native you can do that. You can define it and you can go back and forth. You're going to have a completely native application that you wrote years ago and have a tab that's a React Native application. I mean, so you can definitely mix it up, but those are the instances anytime you need some really low-level, high-performance thing. There's instances where you have to get down into the actual native code.

[0:12:26.7] JM: Chris, anything you want to add to that?

[0:12:28.6] CS: Yeah. I mean, I think that is a good summary. Not so much I guess about performance, but one thing that I think is interesting when it comes to native code is a few years ago, you really had to write native code to do a lot of stuff, because there just weren't solutions out there. I think it shows how far React Native has come and that a lot of people can honestly just use an expo project and it has so many extra bindings for a lot of the native stuff you want to do that are written and are just super effective at the JavaScript layer.

It's pretty rare outside of I think the performance issues that Lee was talking about to actually even need to dip into the native layer these days. There's just so many really stable, solid solutions out there in the community that are being provided, which I think is pretty cool to see.

[0:13:24.3] JM: Those solutions, are they going over the bridge in a more efficient manner, or are they taking some other route to rendering and updating the components?

[0:13:35.7] LJ: Well, as far as implementations go, most of them yes, go over the bridge. In a way, a lot of the built-in modules go over a bridge. Everything has to go over the bridge. The bridge is how – in one form or another. Now there are some native module views that give you a little bit lower level, but in one form or another, JavaScript has to toss over stuff to the native side and it tosses it back, which is the point of the whole rearchitecture that is going away.

[0:14:05.0] JM: I guess, let's start to talk about the characteristics of the bridge and some of the issues with the bridge as it stands today. The JavaScript bridge was designed to be asynchronous, serializable and batched. As we've said, it's basically this message bus that is communicating the state of JavaScript components to underlying native UI state. In order to communicate between this JavaScript artificial layer and the native UI state underneath, you have this message bus, this JavaScript bridge, and it was designed to be asynchronous, serializable and batched. Can you guys describe those characteristics in more detail?

[0:14:52.5] LJ: We've talked a lot about the asynchronous part. That's when you send a message over. It's a lot like node, right? Node came along being a non-blocking single threaded application. Lots of things ran asynchronously, so that the JavaScript side can make a call to the native side and keep doing things. Each side can work independently of the other. Then the synchronize is just because for those – for each side to access data that has to be synchronized or it has to be dumped out so that the other side can pick it back up, because there's no way to really share that data between the two different technologies.

Then and the batch, just like the queueing like we were talking about earlier. It's smart enough on the native side to handle things as it comes. The JavaScript side can fire things pretty quickly. In most cases, right? We're talking about this like it's this terrible performance problems. 99% of the case is you won't even notice any of this stuff going on. You use a term of a bridge and I think of it like driving to work over a river. You drive across the bridge at work and you drive across to go back home, but that's if you to drive across a bridge in what? 20 something milliseconds. It's extremely fast and the way it does all this, but we were talking about with the gaming, or lower level stuff, any delay can be a problem in certain instances. I guess, that's how it works.

[0:16:19.0] JM: Okay. Fabric is React Native's effort at rearchitecting this UI layer. Describe Facebook's plan to rearchitect React Native.

[0:16:30.9] CS: Basically at a high level, what they want to do is blow up the bridge and really go to a model that's much more like what we have today in our browser. If in JavaScript you set a variable equal to a document.createelementdiv or whatever, it's actually pointing to something that's being implemented at a lower level in something like say C++. That is really what we want

to do with the new React Native architecture and the JSI is that instead of taking actions at the JavaScript layer, serializing them, passing them over the bridge and dealing with them on the UI thread at native, actually there will be this JavaScript interface that you can interact and block the UI thread synchronously from actions that are being taken at the top level and JavaScript is listening to and delivering those across to the native layer.

That will go a long way in solving the performance problems that Lee was talking about when you get those super long scroll lists that you're flicking through and you end up with just a blank screen and then the list items start populating in. Well, because you're blocking the UI thread on your scroll event, you can't scroll faster than the UI thread can update anymore, right? You're actually dealing with it in more real-time. That's the idea.

You have this glue layer now in between your native code in your JavaScript that's written in C++, because basically node has this support for running C++ in the JavaScript VM. It can take your JavaScript code and do code gen down to the native code that needs to be run on the UI thread. It's just constantly running in between your JavaScript layer your native layer.

[SPONSOR MESSAGE]

[0:18:46.0] JM: How do you know what it's like to use your product? You are the creator of your product, so it's very hard to put yourself in the shoes of the average user. You can talk to your users. You can also mine and analyze data, but really understanding that experience is hard. Trying to put yourself in the shoes of your user is hard.

FullStory allows you to record and reproduce real user experiences on your site. You can finally know your users' experience by seeing what they see on their side of the screen. FullStory is instant replay for your website. It's the power to support customers without the back and forth to troubleshoot bugs in your software without guessing. It allows you to drive product engagement by seeing literally what works and what doesn't for actual users on your site.

FullStory is offering a free one-month trial at fullstory.com/sedaily for Software Engineering Daily listeners. This free trial doubles the regular 14-day trial available from fullstory.com. Go to fullstory.com/sedaily to get this free one-month trial. It allows you to test the search and session

replay from FullStory. You can also try out FullStory's mini integrations with JIRA, Bugsnag, Trello, Intercom. It's a fully integrated system.

Full Story's value will become clear the second that you find a user who failed to convert because of some obscure bug. You'll be able to see precisely what errors occurred, as well as the stack traces, the browser configurations, the GO, the IP, other useful details that are necessary not only to fix the bug, but to scope how many other people were impacted by that bug. Get to know your users with FullStory. Go to fullstory.com/sedaily to activate your free one-month trial.

Thank you to FullStory.

[INTERVIEW CONTINUED]

[0:21:08.1] JM: Let's drill down into that scrolling example in a little more detail. Chris, can you just rephrase what you said about the scrolling issue as it exists today with the current JavaScript bridge model and describe how the improvements to I guess synchronicity or asynchronicity are helping with that use case of the scrolling through?

[0:21:33.4] CS: Yeah. In today's world with the bridge – and by the way Rom goes through this example in his talk from React Conf, so anybody who is interested in getting into more details should definitely check that out. Basically in today's world, you have a scroll event that's happening on the JS layer. As we said before, the JavaScript needs to now translate that into some big string or whatever that it's going to translate it and to serialize it, give it to the bridge and the bridge has to hand it off to the UI thread and the UI thread has to unserialize, unpack it, get the information it needs about the updating of the UI state, reconcile all of that and display it onto the screen.

Now obviously as Lee said, that usually happens really, really fast. For something like scrolling, we can just input so much into JavaScript faster than that transaction can happen, and so it is updating the UI thread without being able to pull down all the extra data you need to show these cards or whatever happens to be in your scroll view.

Where they're moving to with the JSI is, as you actually are hitting that scroll event in JavaScript, you are making a call that basically immediately goes into the UI thread layer through this JavaScript interface and starts updating the code there. You're not dealing with asynchronicity anymore, this is a synchronous blocking call to the UI thread. You're not able to pile up events on one side and have to wait for them to be resolved, instead as the events are happening, they're actually updating the UI thread on your phone and you're seeing a much smoother real-time interaction. Lee, I don't know if there's anything that you could add to that or clarify in that.

[0:23:31.4] LJ: Well, that's a great explanation. Just people you hear synchronized – I mean, we're JavaScript developers. You hear blocking and synchronized and you immediately say that is bad. In this case, right? On an iPhone, it updates 60 frames per second. If you can block it for less than one-sixtieth of a second, which is perfectly doable, that's not as crazy as it sounds, then you can't visually see it, right? You're injecting your updates directly into that scroll view, you're basically pausing it for one-sixtieth of a second to almost force the data in there, instead of having it wait and pull it off of a cue when it feels like it, right? You're forcing it in there in one-sixtieth of a second. It's so fast that UI and the device can't to.

[0:24:17.4] JM: The scrolling example is such a good example here, because scrolling is something you want to be synchronous. If you think of scrolling, if you're scrolling through a webpage and you think of a scroll action as a series of frames that the page is displaying to you, you don't want to see those pages out of order. You don't want to skip any frames in that scrolling action. You want them to be displayed to you smoothly and in sequential order, which demands a synchronous pattern of display. The C++ thing that sits on the side, I guess this is an alternative rendering unit to the JavaScript bridge? Am I understanding that correctly?

[0:25:05.0] CS: It's called host object, right? A host object is the default C++ thing that talks. It's like instead of a bridge of communication, it's almost an exposed wrapper of the object itself, if that makes any sense. You can call it directly, instead of like, I could talk to you directly instead of sending a message through somebody that says, "Hey, come meet Lee at the coffee shop." We're talking directly now to you, instead of going through your assistant or whoever's taking messages for you.

Fabric is for the UI portion and it uses JSI. There's other parts of it that also use JSI and host objects. It's the way it is today. The bridge is used in lots of places beyond just when you actually have to write a bridge. That implementation is used internally in a lot of areas, including UI and updates and things like that. All of that is being converted from this disconnected messaging to a connected implementation.

[0:26:05.9] JM: Why wouldn't you just make all of your modules, like if we talk about some scrolling module and we can instantiate this C++ thing that is talking to the underlying native UI layer, instead of going over the bridge. Why wouldn't we just do this for every JavaScript component that we have?

[0:26:28.7] LJ: On the React Native side, it will be that way, right? That's what the turbo modules are and some of the code glue, all of the things that exist will have these wrappers. Everything becomes direct. There is no more bridge that's the future.

[0:26:44.1] JM: Interesting. Does that require developers to significantly rearchitect their applications?

[0:26:52.8] LJ: It depends, right? Chris had mentioned earlier how an expo type – for most use cases and we develop a lot of React Native apps. It's rare that we have to get into the native code. When we do, it's something simple like there's this logging tool that they want to use, so we have to implement that on the native side. We rarely come into these issues in real life and those applications, this won't make any change at all. The JavaScript side doesn't change. It's just the native side, the bridge side changes.

However on the flip side, if you have created customized bridging implementations and tools, then those things will need to be converted to the JSI. Now I'm assuming and everything that I've read seems like there's probably going to be a period where both exist for a while until they migrate fully to the JSI model.

[0:27:48.8] CS: Part of me thinks that there's just so many on the React Native team that really loves C++ and wants people to write more of it and that's the whole plan here.

[0:27:56.6] LJ: Well actually, you don't have to write C++. That's a joke, right? The host object is a interface that you can write and you can implement that in your whichever side.

[0:28:06.2] JM: Just to make sure people that are listening understand what we're talking about here, could you define the term turbo module?

[0:28:14.9] LJ: Yeah. Turbo modules is what the React Native team calls it and they came up with it. I'm trying to get all of these architecture things straight in my brain, because now – so fabric is the UI part, right? JSI is the interface, the direct interface. Then so turbo modules are the – those are the wrappers of the native modules that exist today. Then there's the code glue, which is – then there's a code gen part and the code gen part is where it will actually automatically generate this wrapper implementation that I had described earlier automatically for certain types of objects.

[0:28:53.8] CS: I think that the turbo modules basically are your way of actually accessing the native module that you want to access. It's the implementation that basically when JS calls this specific native module, it knows how to convert all of that into – from JavaScript to JNI and Objective C to target the specific platform you need. Rom again has a good example of this, which we actually reposted on our blog. If you actually want to take a look at some pseudocode that goes into the turbo modules, we have that up on blog.g2i.co.

He walks through an example. You would expose a JSI object at the top level, something like a native module proxy that's a global.turbomodule, whatever the name is. To access this, it actually will require the turbo module file, which inside of that there is basically the registry that happens. Then from there, it will trigger the JSI function, which is where it can do the platform-specific code, right? Actually translate your JavaScript into Java Objective C, whatever the case is and target the target platform that you're working on.

Basically, you have turbo modules for – at the end of the day, dealing with things like some native module that you want to access; maps or maybe push notification or something like that are all wrapped up in these turbo modules that will be able to translate the JavaScript through the JSI into the Java and Objective C code that you need at the native layer to actually make it happen.

[0:30:49.4] LJ: Yeah, and they had mentioned too about the finding those things with [inaudible 0:30:51.6] group to help, so that observation step works, right? There are types, so that when it does the observation of the object, it knows what types to create.

[0:31:02.4] CS: Yeah. They just recently did that. That was a pretty big community effort, where basically I think it was Eli who put that on an issue on the React Native repo and is basically like, “Hey, we need help flow typing these things.” I think it was in with a day or a couple of days, all of them had been done. I mean, it was a massive, massive amount of work that the community just stepped in, took on and got done to push this project forward.

That actually goes back to my point earlier about why does Facebook want to do this with React Native. Part of it is to actually make the project more maintainable moving forward. They're seeing that actually be proven out by the way the community is responding to these calls to actually implement some of these changes that need to happen to get the new architecture going.

[0:31:59.0] JM: What is the state of this rearchitecture in its current form today? Because I understand that some of this is planned and some of this has been prototyped and some of this is in-flight, what's the state of the React Native rearchitecture today?

[0:32:14.0] CS: Yes. Fabric was the first part that they announced, which I think that was back in June or July of last year. Sophie had put out a blog post on the Facebook blog announcing that and dropping some hints about the other things are moving forward. I'm not actually sure if they've released that into a stable, or even a minor release yet, or where that's at in terms of the actual React Native repo, but that was one of the first things that they announced that people were aware of that people were already ready to account for.

The rest of this stuff really was just announced at for the first time at React Conf. A lot of this stuff is definitely still in development. There's a lot of things that are going to have to be figured out. I mean, if you go into the React Native repo right now, all the code for the JSI should be in there, but I don't think it's actually doing anything. They've just been prepping it essentially up to this point. There's a road map that they're figuring out on how they're actually going to start to

make the switch. I'm sure as Lee mentioned, there will be a period where both worlds will be active, so that people can start migrating without apps breaking entirely when you update React Native.

They have a lot of the road map is up on the repo. I think they actually have a separate repository that just tracks in React Native community. I think it is is where they have a tracking of where everything is going, what their plans are. I know they have a spreadsheet that shows which modules they want to move out to the community for instance and what the roadmap for that is going to be. They went through every primitive in React Native and made decisions about which one will remain in the core, which ones they'll spin off, which ones they should just deprecated because there's already community-driven solutions for that. A lot of that can already be found on the React Native community repository. You can look into some of those issues.

[0:34:32.4] JM: Since we are talking about the community dynamic, React Native is an interesting project, because it's open source, but it's led by Facebook. Can you describe the community dynamic in more detail and how that's playing out with relation to the React Native rearchitecture?

[0:34:51.1] LJ: Yeah. For me, that's the most exciting part of the rearchitecture is that like Chris mentioned with all the typing and everybody pitching in, because of the way that architecture is happening and things are being broken out into modules, it makes it easier for the community to get involved in certain areas. It makes it more organized. Like before, it was an internal project for Facebook and the code was really dense and really hard to follow. Now with this new architecture and different modules, things are going to be separated and cleaner and we lost Jeff again.

It's just cool to me that now, this allowing reorganization, which anytime you refactor something, right? You get to redo something. You can use what you've learned and usually things come out more organized than they did the first time, so that's another benefit we'll have with any rearchitecture, especially this one is that things will be more organized, more modular and it allows the community to get involved in an easier way, which is the thing I'm really am most excited about.

[0:35:56.8] CS: Yeah. I think that's really changed from where React Native was, where they've moved to much more of a like a RFC process. There's a lot more involvement from the community. I remember hearing complaints from people about how hard it was to update React Native, because they would just – they would work in the dark and release these changes and it would break people's code and it was just a really rough process.

I had heard for some people who talked about they just had their own internal forks of React Native, because they had specific things they needed that Facebook wasn't providing them, and so they just went off and did their own fork in order to get what they needed working. I think this whole effort towards lean core, this whole effort towards making a much more of an RFC-driven process like they have done with React is really going to improve things.

Actually, the React Native Twitter account had a little thread the other day that described how already the efforts have got in the core library into a much more manageable state. They have gone down to about 140 pull requests from 280 where it was in early December. That's getting 2 to 5 PRs a day. There's already tons of community interaction and already just where they're at, they've been able to really start to respond to that more.

Their goal ultimately is to be able to be even more real-time and get to pull requests even quicker than they have been able to and really make it a super responsive, super community-driven library, whether that's the core, or whether that's the modules that they're spitting out with the lean core effort. They're definitely looking to the community to support the continuous development of React Native and the community has really stepped up and done that so far.

[SPONSOR MESSAGE]

[0:37:58.5] JM: Triplebyte fast-tracks your path to a great new career. Take the Triplebyte quiz and interview and then skip straight to final interview opportunities with over 450 top tech companies, such as Dropbox, Asana and Reddit.

After you're in the Triplebyte system, you stay there saving you tons of time and energy. We ran an experiment earlier this year and Software Engineering Daily listeners who have taken the

test are three times more likely to be in their top bracket of quiz scores. Take the quiz yourself any time, even just for fun at triplebyte.com/sedaily. It's free for engineers. As you make it through the process, Triplebyte will even cover the cost of your flights and hotels for final interviews at the hiring companies. That's pretty sweet.

Triplebyte helps engineers identify high-growth opportunities, get a foot in the door and negotiate multiple offers. I recommend checking out triplebyte.com/sedaily, because going through the hiring process is really painful and really time-consuming. Triplebyte saves you a lot of time. I'm a big fan of what they're doing over there and they're also doing a lot of research. You can check out the Triplebyte blog. You can check out some of the episodes we've done with Triplebyte founders. It's just a fascinating company and I think they're doing something that's really useful to engineers.

Check out Triplebyte, that's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte, byte as in 8 bytes. Thanks to Triplebyte and check it out.

[INTERVIEW CONTINUED]

[0:39:48.4] JM: This brings up one of the biggest, highest profile migrations away from React Native, which was the Airbnb movement away from React Native. I think when I did my show with Airbnb, they discussed the fact that I think – I think they had to maintain their own version of React Native. I could be wrong about that actually, but – Now Airbnb is a total edge case and yeah, it's just a gigantic organization.

It's unsurprising that this thing that was made for the masses and made for Facebook, eventually Airbnb maybe outgrew it. Is there a chance that these kinds of fixes, the Fabric and the changes to the community management, would these have kept Airbnb in a more comfortable state with their React Native usage?

[0:40:39.2] CS: I think there's a few different issues that Airbnb encountered, which they talked about very openly on a series of blog posts, which I think was really a great insight into beyond just the edge cases of performance we've talked about what you have to consider when you bring in React Native into a team that already has a significant amount of native code and a

significant number of native engineers working on that code. That's really what Airbnb was before they got into React Native, right? It wasn't a React Native greenfield project. They had large native code bases already.

That also meant they had large native engineering teams. There was just a lot of conflict in the culture of those teams, where the iOS engineers in general were perfectly fine to move, because as we know the performance on iOS for the most part is actually pretty solid and the Android teams for the most part hated it, because the performance on Android is just where you run into all the issues a lot of times.

There's definitely a dev culture piece of this that regardless of solving the actual performance issues through the new architecture that you just have to deal with and that goes down to just choosing the right tools for your team and what's going to get the job done. To the point of the Android performance, they definitely are going to solve a lot of the Android performance issues with the new architecture.

In fact, Eli White mentioned how much Android has driven this rearchitecture, that they spent almost all their time over the last six months focusing on Android, because there's a huge number of Android users who use the Facebook apps. He talked about how on Android, one of the things that really matters a lot more than on iOS is disk access. Reading from the disk is really slow on Android. You pay a penalty for every single time you have to load a java class.

They spent a bunch of time on the team investigating and optimizing React Native startup on 2014 Android devices to make marketplace really fast for those users. That's what actually led to the design of turbo modules and some of the features of Fabric as well. Definitely, Android was at the forefront of the Facebook teams' thought process when they were thinking about the turbo modules project and the architecture in general and they want to make that performance really good for so many of their users who are on older devices, underpowered devices. That's really a huge motivation for them.

Yeah, as it relates to Airbnb, maybe the rearchitecture would help get more buy-in from the Android teams. There's also probably just an aspect of being comfortable with your tools and wanting to create a really good developer experience for your teams and maybe it just wasn't

right for them. You compare that to something like discord, for example. Discord really started with React Native from the beginning and really just focused on iOS and made intentional choices about how they were going to deal with performance issues as it related to the native code.

I think they have an engineering team of 40 people that supports a 150 million users on an app that really is – has a lot of complexity in it, but runs just beautifully on iOS for sure. You can definitely see how the path differentiate between a company that has a project that is greenfield React Native, versus trying to bring in React Native into the brownfield and the different challenges you're going to associate that with that from a technical standpoint, but also just from a developer experience standpoint.

[0:44:44.8] JM: All right, as we begin to wrap up, I would be remiss if I did not discuss Flutter with you guys, because – We did some shows about Flutter recently and they were pretty popular. People are very curious about this cross-platform technology solution out of Google. It was funny, because I was reading the transcript of this discussion with the core React Native team, they didn't really want to talk about Flutter, which is which is totally fair. I mean, there was a discussion around React Native, but it was just – there's no clearer sign that something is a competitor when somebody refuses to talk about them by name.

What do you think about the Flutter architecture where it's writing directly to, as I understand, directly to the GPU, or directly to the screen or something? It's a different architecture than the React Native architecture. It's funny, because this is so iconic of the Google versus Facebook, because Google has this beautifully architected solution that seems so fundamentally better. Then Facebook has this Jenky thing with much more attraction that works and that is popular and has a gigantic community around it. Any perspective on Flutter?

[0:46:00.0] LJ: I can't also explain that. I'm assuming you're talking about the chat we had on discord and it was insane in a good way and a little bit of a crazy way, but there were so many people there. It was really a surprise. When we opened – Gabe and the guys opened the floodgates literally for questions, it was so fast that you couldn't really keep up with it. There were some wise guys that immediately jumped in there and just like, "Why don't you just use Flutter?" They were just trying to be funny. I think that's drove the –

[0:46:31.6] JM: Okay. Maybe I misread. I misread the dialogue.

[0:46:34.1] LJ: Well, it was crazy. People thought it was funny to get in a dig about Flutter at the get-go, so I think that forced like, all right guys, let's just get serious and have –

[0:46:45.4] JM: Fair enough.

[0:46:46.1] LJ: Because they weren't questions about Flutter. It was just, “Are you guys going to move to Flutter? Why aren’t you using Flutter? How many of you guys use Flutter in your free time?” I think that's why it was avoided, because people were being jokesters about it.

Yeah, so Flutter has widgets and those widgets mapped to lower level APIs, which render at a lower level. A lot more like you had discussed. In a lot of ways, this sounds a lot like this JSI method that React is moving to. It definitely does give nice UI performance and rendering and things like that.

For me, especially as a lead and somebody where efficiency and stuff is important, the biggest thing of for me – I like options, right? I like that there are other things to try and experiment with and learn. As a manager or somebody who is concerned about developer efficiency and has a team that already knows JavaScript and React, that's the biggest issue I have with Flutter is written in dart, the structure of the code is completely different.

If you're a JavaScript developer, or web developer, or in the area that we live in, it's completely foreign to you. I’m not saying you can't learn it, but the whole point of React and React Native was to learn once you write anywhere and it actually does work. I mean, it's the same code. You substitute the word div with the word view and you're 90% of the way there. You're making progress. You're building a mobile app.

Your team, you get way more efficiency out of a team of JavaScript developers that can do both of those things, instead of okay, now you need to go learn dart and the layout structure is completely different and everything is completely new. That’s my hang up with it. Performance-wise, they're doing neat things. I'm glad that they're doing it in a different way. I think we need

different perspectives to big problems that we have like mobile development. I'm glad that it's there. That's my take, I guess.

Lee, do you know, does Google actually use Flutter internally for any of their projects?

[0:48:57.0] LJ: I actually don't know.

[0:48:57.9] JM: They do. They do.

[0:48:58.6] CS: They do? Okay. Because I know that was one of the big problems with angular was Google wouldn't actually use angular for their project. It wasn't driven by the real needs of a specific dev team, which I think is interesting. I watched the talk the other day from Lee Byron about the creation of GraphQL. One of the things he talked about with GraphQL was how much originally when they created it it was just specifically to meet the needs of one particular team at Facebook.

They weren't trying to solve all the world's problems. They just wanted to help this particular team do their job better. It evolved as a real product. If they are using it at Google, that's an improvement I think where they can actually have real users who can give them feedback on how to improve it as a product, versus just putting something out there to have another tool in the ecosystem.

[0:49:54.3] LJ: Like I said, I like competition, I like options. I guess the other issue I have with it is I've got an e-mail in my inbox right now tell me about how my Google+ account will soon be migrated because they're not doing that anymore. I have one from before where this thing called – was it wave, right? Then I probably got a couple dozen e-mails from Google about projects and products in the past that no longer exist.

[0:50:19.0] JM: Well sure. I mean, the – what was the Facebook, Graph API or something? What was that API that a bunch of people built stuff on top of? Can't do that anymore.

[0:50:30.3] LJ: Yeah. Well then that's what I like about though with where React Native is going is you can fork it, right? If you like the way it is and Facebook says, "You know what? We're not

doing this anymore. We're going back to native code." The community can completely take over, which is great. That from a higher level like CEO, or executive type person, you're not putting all your eggs in this basket that can just be taken away. It is community-driven. You can certainly fork it if you want to and not worry about that force – a force of change.

[0:51:01.6] JM: Well, it'll certainly be interesting to see if this evolves into React versus Flutter is the new Android versus iOS. I think we're very far from that being a reality, but it's clear that people want cross-platform UI layers. The world in which we live is so fragmented and it's way too fragmented. This is not going to be the end state. We're asymptoting towards I think easier cross-platform development.

We're basically out of time. I could talk to you guys for a lot longer, because there's a lot of stuff I would want to discuss. I do want to give a quick plug for G2i, which is where you guys both work, because I am a user of G2i. I'm a client. I have hired people out of G2i. You guys run a great business. I'm speaking entirely honestly. I've used you guys for years at this point and I think you're building something really significant. People who are listening and are looking for react developers, or mobile developers, it's really a great place to go for finding contractors, finding hires. I think now you guys are playing more of a community role also, so probably some content. I just want to congratulate you guys on really building something meaningful.

[0:52:21.9] LJ: Cool, man. Thank you. It's fun. We get to hang out with cool people and talk about stuff like this all the time. We like what we do. We meet a lot of great people, a lot of great engineers. We have a pretty difficult process for bidding people, but once they get in everybody helps each other out. We have a huge Slack channel where people communicate and then share knowledge. Yeah, it's fun.

[0:52:43.6] CS: Yeah. To the point about getting more involved in the community, we definitely have been talking about some ways to continue to do that and give back to open source as a business. We'll be announcing some things this year about how we can continue to push that forward and give back to the open source community, which has given us so much.

We basically built our business on the open source community around React and React Native, so we're excited to continue to find opportunities to support the community that's building so

many of these great products and just making the development experience so pleasant for so many people.

[0:53:23.6] JM: Okay, guys. That's great. Well, thanks for coming on Software Engineering Daily.

[0:53:26.5] LJ: Thanks, Jeff.

[0:53:27.1] CS: Yeah, thanks for having us.

[END OF INTERVIEW]

[0:53:32.1] JM: DigitalOcean is a reliable, easy-to-use cloud provider. I've used DigitalOcean for years, whenever I want to get an application off the ground quickly. I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU-optimized droplets perfect for highly active frontend servers, or CICD workloads.

Running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily. As a bonus to our listeners, you will get a $100 in credit to use over 60 days. That's a lot of money to experiment with.

You can make a $100 go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure and that includes load balancers, object storage, DigitalOcean spaces is a great new product that provides object storage, and of course computation. Get your free $100 credit at do.co/sedaily. Thanks to DigitalOcean for being a sponsor.

The co-founder of DigitalOcean Moisey Uretsky was one of the first people I interviewed and his interview was really inspirational for me, so I've always thought of DigitalOcean as a pretty inspirational company. Thank you, DigitalOcean.

[END]