EPISODE 654

[INTRODUCTION]

[0:00:00.3] JM: WebAssembly is a low-level compilation target for any programming language that can be interpreted into WebAssembly. Alternatively, WebAssembly is a way to run languages other than JavaScript in the browser, or yet another way of describing WebAssembly is a virtual machine for executing code in a low-level, well-defined sandbox. All of these different ways of defining WebAssembly are accurate, and WebAssembly is reshaping what is possible to do in the web browser.

A developer can write a programming in Rust or C++, compile it down into a WebAssembly module and call out to them module via JavaScript. This is very useful for running memory intensive workloads in the browsers, such as 3D games, or really anything where you want access to Rust or C++, and we go over some examples in this episode. But WebAssembly is also useful for running modules outside of the browser. Why is that important?

You can already run C++ or Rust code outside of the browser by executing the program from the command line. Why would you want to put that code into a WebAssembly module before executing it? Well, one answer is security. WebAssembly modules have well-defined semantics for what memory they can access. WebAssembly could provide more reliable sandboxing for un-trusted code.

Steve Klabnik is a software engineer at Mozilla and he joins the show today to play the role of a WebAssembly futurist. We revisit the basics of WebAssembly, which we touched on in an excellent previous episode with Lin Clark, where she really dove down into the guts of WebAssembly.

In this episode we really talk about the applications more than the guts of WebAssembly, that we do talk about the guts somewhat. Steve also gives us some historical context, comparing WebAssembly to some past web technology, such as Flash. What did flash do right and what did it do wrong? What were the lessons that were learned from Flash that are hopefully averted in WebAssembly? We're also going to talk more about WebAssembly in the next episode of

Software Engineering Daily, which is with a few people from Google who are working on WebAssembly.

Before we get started with this episode, I want to mention that I am hiring for a new company that I'm starting, and I can't talk about the product quite yet, but I'm very excited about it. I am looking for an engineer in the Bay Area with significant experience in React.js and either AWS or Google Cloud, and you can email me at jeff@softwareengineeringdaily.com. You can check out the job posting at softwareengineeringdaily.com/jobs, and I'd love to hear from you. Let's get on with the episode.

[SPONSOR MESSAGE]

[0:03:05.9] JM: In today's fast-paced world, you have to be able to build the skills that you need when you need them. With Pluralsight's leaning platform, you can level up your skills in cutting edge technology, like machine learning, cloud infrastructure, mobile development, dev ops and blockchain. Find out where your skills stand with Pluralsight IQ and then jump into expert-led courses organized into curated learning paths.

Pluralsight is a personalized learning experience that helps you keep pace. So get ahead by visiting pluralsight.com/sedaily for a free 10-day trial. If you're leading a team, discover how your organization can move faster with plans for enterprises. Pluralsight has helped thousands of organizations innovate, including Adobe, AT&T, VMWare and Tableau.

Go to pluralsight.com/sedaily to get a free 10-day trial and dive into the platform. When you sign up you also get 50% off of your first month. If you want to commit, you can get $50 off an annual subscription. Get access to all three, the 10-day free trial, 50% off your first month and $50 off a yearly subscription at pluralsight.com/sedaily.

Thank you to Pluralsight for being a new sponsor of Software Engineering Daily, and to check it out while supporting Software Engineering Daily, go to pluralsight.com/sedaily.

[INTERVIEW]

[0:04:44.8] JM: Steve Klabnik, you are a software engineer at Mozilla. Welcome back to Software Engineering Daily.

[0:04:50.7] SK: Thanks so much for having me. It's good to be back.

[0:04:52.9] JM: Yeah, last time we talked about Rust, and that show was really popular. People are very fascinated by Rust. You work at Mozilla. Mozilla is doing a lot of interesting things these days. There's Rust. There's WebAssembly. What is it like to be at Mozilla these days?

[0:05:11.1] SK: So it's interesting, because I sort of see myself as working on Rust first and at Mozilla second. I'm actually not honestly that great of an employee. My focus is mostly on Rust itself. So a lot of times – Mozilla is a pretty relatively big company. I think we're 1,300 people, 1,500 people right now. So people would be like, "Oh, do you know my friend that works at Mozilla," and I'm like, "I have no idea actually, because I pretty much talk to my team and don't do a whole lot of stuff come across the organization."

Although now with the WebAssembly stuff, I'm sort of branching out in some places. But in general, Mozilla is very interesting place to work and I really like it. There's just so much going on, and I like the fact that it feels more like a sort of values-driven place than a profit-driven place.

[0:05:54.4] JM: Certainly. WebAssembly is a topic that people have been really wanting to hear a lot about. I think Rust is I think easier for people to wrap their minds around, because Rust is a newer systems level programming language. People are comfortable with the idea of systems level programming languages, and you can just tell them Rust is like C or C++ in some sense, or Go in some sense, but it's got different semantics and it's interesting because of those different semantics.

WebAssembly is a topic that's a little bit harder for people to wrap their mind around. So I'm glad to have you back on the show to discuss it. So we had Lin Clark on a while ago, and she gave a really good explanation for WebAssembly. But I'd like to go over the basics, the foundations of WebAssembly for a little bit with you, because I want to give people another chance to catch up.

[0:06:45.9] SK: Yeah, totally.

[0:06:46.8] JM: So WebAssembly is a low-level compilation target for languages with controlled memory. At least that's how it exists today. You need to have controlled memory structures, things like C and C++ and Rust where you can do memory management. WebAssembly is currently adapted by the major – Or adopted by the major web browsers. Why do we want that? Why do we want to be able to run things like C and C++ and rust in the browser?

[0:07:14.4] SK: So before we get to that slightly, I actually prefer a slightly different framing for WebAssembly. Since you want to talk about what it is, like maybe we should dig into that. So you're definitely not wrong. However, it's deftly true that many people do not know a lot about WebAssembly and I think that that's because, overall, the crew WebAssembly people has not done as much of a job like communicating what is interesting about WebAssembly to the outside world. Everyone who works on it is fantastic, but a lot of them are involved in technical details and not sort of like more marketing kinds of things. So I've been trying to sort of change that sort of aspect.

So one of the things I think is most interesting about WebAssembly, and I think sort of the proper way to think about it, is its own virtual machine, and by some sense, like programming language. That you can always program in the VM directly. However, it's a VM that other languages can target.

Like you mentioned, C, C++, Rust, all sorts of other languages. I'm sure we'll talk about that more. But it's also not as exclusive to the web. It was designed very carefully so that it can be put in other environments, not just web browsers, and I think that's also interesting area of development for WebAssembly. But it is true that like most usage of it at the moment will definitely be inside of web browsers.

So when you think about it as sort of like an additional VM in the browser to target, that's like I think the sort of like right kind of framing to some degree. It is a particularly low-level virtual machine and does not have a lot of like fancier stuff that high levels ones do, and there are sort of good reasons for that.

So when you start thinking about that way, I think it opens up really interesting possibilities. So for example, you can write applications that work in a web browser, but also outside a web browser and sort of like allows the web to get into these replaces that it wasn't like traditionally kind of able to go. I think that's mostly what's really exciting about it for me. Most platforms have sort of like a high-level language and a low-level language, and the web has only ever had a high-level one. So I'm sort of really excited about having a low-level one too.

[0:09:11.0] JM: Tell me more about the concept of virtual machines. We've got a number of virtual machines that are in our everyday vocabulary as software engineers, like the Java Virtual Machine, the LLVM, the low-level virtual machine. We've got the VH, which I think is a virtual machine for JavaScript, and then there's the Mozilla's equivalent version of the V8, which is the Spider Monkey JavaScript virtual machine. Virtual machine does things like execute bytecode. You can interpret a language down into bytecode and then you can manage that code within the virtual machine so that certain code paths run more effectively. You can do things like garbage collection in a virtual machine.

Talk about the idea of a virtual machine more broadly, and why we need multiple virtual machines. What purposes they fill in the various facets of our computing?

[0:10:02.2] SK: Yes. So what's interesting is, is that like a virtual machine has a bunch of different – Like many terms. It's kind of overloaded a little bit. So some people think one thing and some people [inaudible 0:10:11.5] other things. The sort of biggest distinction between the two is I'm not talking about a virtual machine, like Docker is not a virtual machine in any sense of the word, but like things like that where you're sort of emulating a whole another computer. There's a reason that we refer to them both this way, but like I'm not talking about that kind of thing.

The ones that you brought up is definitely more along the correct lines, but like fundamentally, a virtual machine just means that you're writing your code for something that is not a piece of physical hardware. You're writing it for a piece of software that sort of presents some sort of execution model for you to be able to code against.

What's really interesting is that not a lot of people know this, but C is actually implemented, like the spec defines a virtual machine. MMC is implemented in terms of that machine itself, not in terms of the hardware, because in some level, a virtual machine is the only strategy by which you can be portable. Because sort of by definition, if you target a specific kind of hardware, you're not portable to other kinds of hardware.

So VM is like an abstraction over multiple kinds of machines. So that way you can like target it and it does the job of targeting the individual machines specifically. This has a wide variety of like implementation strategies. Like I said, C does not run in what we traditionally think of as a virtual machine, like the JVM, because it's kind of like a compile time construct. The same thing with LLVM, They actually – You're totally right, that they originally were named at, because it was called low-level virtual machine. They felt that that conception of a VM was a little too confusing. So they've backfilled it to something else now, because LLVM is very similar. It's not a runtime virtual machine. It's sort of like the compile time concept.

Things like V8 and Spider Monkey and Chakra is the one in Microsoft Edge, the JVM. These are kind of runtime virtual machines, and this is extremely common in programming languages today, where your programming language executes in some sort of like interpreter/JIt compiler that provides you with all these other sort of facilities.

So WebAssembly sort of sits somewhere in the middle, and that you can run it in an interpreter if you want to, but you can also have it JIT compiled. So that's also like a very interesting aspect, and part of that is due to the fact that it is very simple virtual machine. So there's not a whole lot of like stuff going on inside of it.

[0:12:27.7] JM: Yeah. You've given a great survey there, and I think one other one that we didn't mention is the Ethereum virtual machine. What we can see with these is each of them is a domain-specific machine for doing some kind of program translation/runtime management or runtime optimization, but it's not like you can just take a virtual machine off-the-shelf and then tweak it to do things that you want out of WebAssembly. This is why we need to build a new virtual machine for the purposes of WebAssembly, for getting this web portability that we're trying to achieve.

[0:13:04.6] SK: So the Ethereum example is actually perfect, because the folks at Parity, one of the companies who works on the production of Ethereum implementation actually have a test network where the WebAssembly VM is integrated with the Ethereum VM and you could write smart contracts in WebAssembly. So that's like a good example of outside of the browser possibly usage of WebAssembly.

[0:13:26.6] JM: Yeah, I saw that in one of your posts. Let's scroll back a little bit though and stay at the foundational levels. So you've talked about the fact that we want to be able to run these WebAssembly modules outside of the browser as well as inside of the browser. So the first idea is we want to be able to run lower level languages inside the browser. So things like C++, and I can think of several useful examples, like my digital audio workstation has been written in C++. Right now you can't have a web version of that, because there's no way to, as far as I know, run C++ in the browser managed by the browser. But what WebAssembly does is it is able to interpret C++ code, or Rust code into code that can run in the browser in this WebAssembly lower level language.

So that's one thing. Correct me if I'm wrong there. But why would I want – Since you touched on the ability to run outside of the browser as well, I can already run C++ outside of the browser. Why do I care about being able to run it as a WebAssembly module outside of the browser?

[0:14:39.7] SK: So there's a couple of different aspects to this. One is which it's easier to target one platform than many platforms. Like while it's totally true that you can run stuff outside of the browser, you still need to get some tweaks going to be able to take an existing application and get it to run inside a WebAssembly. So if you do that first, then you kind of get the benefits of both without needing to port it.

Additionally though, there are some other aspects of it that are very interesting, and those rely around or revolve around WebAssembly's sandboxing and safety situations. So WebAssembly is sort of follows the JavaScript model of security, which is a really important property. So all of the access inside of it is completely sandboxed, and there are also certain other sort of deep technical aspects of the way that WebAssembly is defined, such that many of the serius security vulnerabilities that can happen in your C and C++ code cannot actually happen inside a WebAssembly interpreter.

So one of the sort of benefits of taking C that would run on your normal machine and compile it to WebAssembly instead and then running down your machine is that you kind of get this exploit mitigation, and that's like really important. You obviously pay a little bit in performance for this to happen, but like if it's the difference between like I get owned and I don't get owned, then a little bit of performance might be worth it.

[0:16:08.1] JM: That's pretty cool. So that means that instead of essentially having my terminal run my C or C++ program, then the program has the permissions that terminal session has. I am running it in what? An electron app or some other kind of web-based sandbox?

[0:16:32.1] SK: I mean it would depend entirely on how they actually implemented it. So there are multiple WebAssembly interpreters that are completely outside of web browser and have nothing to do with any of that. You could also adopt something like electron and use the ones that are built in to the web browser. But you also don't have to, because that's a pretty huge dependency. Since WebAssembly is so simple, it's not that hard to like write one, and there's a bunch of people who have written a bunch of them for various purposes. So you could sort of do either.

Taking this idea to the extreme, there's some projects people have where they have like operating systems, where these are obviously toys right now where they only run WebAssembly programs and no other kinds of programs. WebAssembly is the binary output format, which is like a really cool extremely experimental kind of thing.

[0:17:18.8] JM: Well, that is cool, because I can see the appeal of that, where when I think of a WebAssembly module and I think of my program as a C or C++ or any other language that ends up compiling down to WebAssembly into a WebAssembly module, and then think of composing applications out of these modules, that's kind of a composability abstraction set up that I can get comfortable with as a programmer.

[0:17:48.9] SK: Yeah. Definitely, there's a lot of possibilities in this space. So there's a lot of ways that this can go. I'm not saying that like everyone is going to run all their code in WebAssembly in the future necessarily. I'm saying there is a possible future in which that is true.

I don't know if that will ever come to pass, and it probably won't. But like this area is so new and so underexplored that there's just so much possibility out there. It's very interesting.

[0:18:13.4] JM: Yeah. It does remind me of Docker in some ways, because what was nice about Docker is not only was it a really nice new composability layer and there was obviously the ease-of-use aspect of it, but it fit into the present day reality of the programmer. So that's the beautiful thing about WebAssembly, is these WebAssembly modules, you can just access them from JavaScript. So it fits into your everyday workflow as a web developer, but you can also imagine this future where things are just composed entirely of WebAssembly modules.

[0:18:49.1] SK: Definitely. I think that the relationship between JavaScript and WebAssembly is definitely big topic to – I don't know if you want to go there.

[0:18:57.0] JM: Sure. Let's go there.

[0:18:58.1] SK: [inaudible 0:18:57.6].

[0:18:58.1] JM: Let's go there.

[0:18:59.3] SK: All right. So another thing about WebAssembly is that lots of people hear the sort of one sentence summary of what WebAssembly is and does, which is run other languages in your browser and then they go, "Oh my God. This is going to destroyed JavaScript," and they either sob or tackle, depending on whether they want that to occur or not.

What's interesting is that this kind of opinion is completely not shared by basically any of the people who actually work on or advocate for WebSssembly. None of us think that WebAssembly is going to be replacing JavaScript, and that's not the intention of the technology either. I think that people hold that opinion for several different reasons that are all kind of wrong. So it's sort of really important that people understand that especially in the near term, WebAssembly is like almost more useful to augment your JavaScript than it is to write applications entirely in not JavaScript, and there a bunch of different reasons why that's the case.

[0:19:56.7] JM: Go through some of those reasons. How does WebAssembly augment my ability to write applications as a JavaScript developer today?

[0:20:06.0] SK: So here is one of the reasons why JavaScript is not going to go away in this world. You already have that JavaScript virtual machine installed in your computer, because it's in your browser. But if I want to write, say, Ruby, a language I love, and compile it to WebAssembly and put it into the browser, because WebAssembly is so low level. Its semantics are more similar to like C or C++, you, in order to run Ruby code of the browser, you need to compile the Ruby virtual machine to WebAssembly and your Ruby program to WebAssembly, and then you run your Ruby program in the Ruby VM, which is running on the WasmVM.

So that has a couple of different implications. That generally works, although there are some reasons why it's little slow at the moment. WebAssembly could grow some stuff to make writing these kinds of virtual machines on top of it a little bit easier. But beyond that, that means that you now need to download a RubyVM. I haven't actually compiled Ruby to Wasm, so I don't know how big it is. But in native code, the RubyVm is about 30 megabytes. So I'm assuming it will be roughly the similar. So you're talking about downloading 30 megs when you hit a web page.

The Go programming language has [inaudible 0:21:19.1] we support out, and my understanding is that their Hello World is about two megabytes, for sample. So download speeds and by proxy size is extremely important web applications. There's that study by Amazon on a lot of people site, where if you take more than the second, half of your traffic falls out, right? So binary size is a really, really big sort of issue, and for a lot of languages, they need to produce large binaries. Whereas with JavaScript, the VMs aren't included, and so that's like a very big step for making JavaScripts like a valuable thing, because you're going to have smaller sizes in some cases. I think that's definitely like the biggest one on the list.

The other thing is, is that regardless of the people on the internet tell you, there are a lot of people that really like JavaScript and like writing in JavaScript. So I think a lot of this anxiety comes from sort of both directions. Like some people hate JavaScript and think there's no reason anyone running a program in it, and some people who like JavaScript to know that those people exist, and they fear that this is going to cause an Exodus away from JavaScript.

However, if you look at the server side of the equation, like a lot of people believe that like people only write JavaScript because it's the only language you can write in your browser. If they had any choice, they would never to choose JavaScript. But on the server, people have that choice and many of them still choose Node.js today. It is massive. For those reasons too, like it's not just as simple as like there are alternatives, and now JavaScript is dead. JavaScript is the most popular programming language by far today.

So programming languages just don't go away overnight just because like new options are introduced. That's not really how things work. So those are the two big reasons why I definitely think JavaScript is sticking around and why Wasm is not like a JavaScript killer.

[SPONSOR MESSAGE]

[0:23:17.5] 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, and 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, and 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, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars 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, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder 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. So thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[0:25:24.5] JM: I think we've given that good introduction to WebAssembly at this point, and one thing I'm having trouble wrapping my mind around, I think a lot of people are happy having trouble wrapping their minds around is where exactly we are in the evolution of WebAssembly? What impact is WebAssembly having on the everyday user today and what are going to be the early applications of WebAssembly? When is it going to impact the everyday user?

[0:25:52.2] SK: Yeah. So I think every day, we're maybe not quite there. One of the things is about Wasm right now, as I said, is still very early on. So there's a lot of stuff happening, but a lot of it is experiments. So you can go to some webpages and see some really cool stuff using WebAssembly and you can play some games in your browser and do things like that, but we're not at the point where like you are using WebAssembly without realizing it most of the time. Some of that is just because it's so new.

The first sort of – They call it the MVP release has shipped all browsers in like September of last year. So there hasn't actually been time for people to have built significant production applications with WebAssembly just simply because it was not ready until very recently. So you're only going to see lots of small demos, because it takes – Say you want an application that's like the million lines of code. It literally takes time to write those million lines, right? So there has not being enough time for people to write larger things. This is slowly starting to change.

So, for example, Dropbox announced recently that when you sort of like upload a file, I believe only in certain circumstances right now, they do client-side compression using WebAssembly before shipping the file off to their servers. This is a good example of like where I think WebAssembly fits into people's daily lives, is there are some tasks that are useful that you want to do client-side that need this sort of efficiency and stuff.

So Dropbox is going to save a whole lot of bandwidth and a whole lot of money by only transferring compressed things over the network instead of transferring the full file and then compress it on their servers. So that's like the earliest sort of like thing that I can think of that that like fits into sort of every day model. But you wouldn't know that by using Dropbox itself. This is also sort of one of the problems that these sort of like foundational technologies, is you kind of only notice them when it's broken, right? So you ever notice how much JavaScript is running on a page until it does the work anymore for some sort of issue. I think the same is sort of with WebAssembly. You'll see it sort of sneaking in to places where you don't even realize that you are using it. It just happens that everything kind of works.

A lot of people are excited about client-side cryptography for WebAssembly, and I think that's also going to be a really big area in the future, because WebAssembly has integers, and integers are very important for doing cryptography correctly. Right now, sort of cryptography and JavaScript is sort of relying on the optimizers to do the right thing, and you never want to trust your security to an optimizer. So that's like another area where I think that there will be some stuff, but there hasn't been a whole ton of major production usage yet.

[0:28:33.8] JM: JavaScript has less accurate number formats?

[0:28:37.3] SK: So JavaScript only has IEEE7 whatever. I can't remember the number freight now for some reason, but floating-point numbers. The JavaScript language itself only has floats. So what JavaScript VM's will do is if they see you using a number like an integer, they will optimize it into an integer so that you have reasonable performance. But on a language level, you only have floating-point numbers. Floating-point numbers have inaccuracies because of the way that floating-point number works. So, yeah, that's kind of an issue.

[0:29:08.7] JM: So does it just bloat the – You get the integer effectively, but it just bloat your memory for it?

[0:29:14.4] SK: It's more like you have to be relying on the optimizer understanding that your code only does integer stuff, and then doing the right thing of internally representing it as an integer. So it has to kind of like recognize that by – I don't know what heuristics they use

specifically, because I'm not a JavaScript VM engineer, but like I talk to them a lot. They basically will see like, "Oh, you're never using the decimal aspect of this. So we'll turn it into an integer internally so that you get more speed."

[0:29:44.8] JM: The example he gave with the Dropbox compression. So I upload a file to Dropbox, and if dropbox could get me to compress that on my client-side in my browser, that would be fantastic, because then they would both save on bandwidth in the transmission of that file to Dropbox's servers. Why can't they just write that compression algorithm in JavaScript? Why do they want to use Rust or whatever to compress my file?

[0:30:12.3] SK: Yeah. So this kind of lead to one of the big areas where WebAssembly is supposedly going to impact people's day-to-day lives, and that's performance. So even as we were just saying about this whole integer situation, right? The people do that optimization because it's faster using numbers integers instead of floating-numbers for math.

So this is kind of just true with WebAssembly in general, is that while performance is sort of fluctuating really wildly at the moment, because this technology is very new, it's roughly expected for it to be kind of like generally in the same order of magnitude as C. You should not be losing much performance if you compile C code to Wasm and then execute it. Whereas JavaScript is a very fast dynamic language, but still ultimately is on sort of a different category of performance than lower level languages.

So the primary reason is that, like if it – We think about numbers. Sometimes it's hard to like imagine the differences. But say WebAssembly is 100 times faster than JavaScript, just to make up a number. I'm not going to say this is literally accurate, but like if something were to take one second in WebAssembly, then it would take 100 seconds, which is a like little over a minute and a half in JavaScript. So performance is itself kind of like a big feature, and that's why people are using Wasm in these circumstances.

[0:31:29.6] JM: Okay. So let's say I am working with my browser and I've got several different web applications open in different tabs that have various WebAssembly modules. I've got Dropbox open. It's doing some compression. I've got a game open with some WebAssembly modules that are managing the graphics, etc.

So I've got these different WebAssembly modules that are doing various things in my browser. Each one of them is sandboxed and is, I guess, to some degree controlling its own memory. What is the browser doing to manage those WebAssembly modules from the overall perspective?

[0:32:05.5] SK: Do you mean like in the security aspect, or like when you say manage, what do you mean more specifically?

[0:32:11.4] JM: Does the browser have to allocate some fixed amount of space that the WebAssembly modules can potentially occupy?

[0:32:20.0] SK: So, yeah. Thanks. I get it now. So when you write a WebAssembly module, you can say in sort of the metadata of the module, this is how much memory that I want to have at startup. When you instantiate that model inside of the browser from JavaScript, you can sort of inspect that value before you actually do the instantiation and then make a decision to say like if this is enough memory that you want to allocate or not.

Wasm is a 32-bit platform. Excuse me. So that means that it can ask for up to 4 GB of memory, but nothing more. I believe that browsers today have sort of a cap built in and that you can't literally get all 4 GB. But I'm not actually sure, but conceptually speaking you could allocate up to 4 GB. Then the module itself, also, there's sort of an instruction on WebAssembly. It says, "Hey, I would like to grow the amount of memory that I have by this s much," and the browser is then able to say yes or no and decide to grow it or not.

Interestingly enough, there is no shrink operation currently. So you can only ask for it to grow, but that's kind of like how it works. So each one of those modules in each of your tabs would be asking for a certain amount of memory and the browser decide to give it to it or not, and they are completely isolated from one another. So there's no way that to kind of like interfere with each other's stuff.

[0:33:37.0] JM: How well-developed is the memory management story between the browser and the WebAssembly modules? Are there any outstanding memory leak issues? Are there any other issues related to memory?

[0:33:47.8] SK: There's not really issues around things like leads. What there is sort of issues around – So one of the things that WebAssembly wanted to do, and sort of hit this in the bigger picture slightly. Now we have a low-level place for certain kinds of APIs, while WebAssembly is restricted to do the same things that JavaScript can do today, WebAssembly is going to grow some features that JavaScript does not have in the future.

One of those major features that was sort of on the thoughts of the designers when they made it was multithreading. So the intention was for WebAssembly to grow full multithreading with shared memory concurrency, and the idea would be that your JavaScript could read that WebAssembly memory and write to it and then WebAssembly can like figure that stuff out in a multithreaded context.

Then specter and meltdown happens, and that became a problem. So that's definitely an area where those plans have sort of been put on hold while we investigate how to fix those kinds of issues. So regarding memory stuff, I think that's the least mature aspect of it, is that like WebAssembly is also still single threaded today and a lot of it comes down to these sort of exploits happening. That's why it's not enabled yet.

[0:35:00.3] JM: What was the specter and meltdown –I know what that was, but actually only faintly. Maybe you could remind us what that is and how it affected the development of WebAssembly.

[0:35:09.6] SK: Okay. So I'll preface this, but the fact that I'm a hobbyist operating systems developer and not a security expert. So I'm going to give you some broad stroke explanations that may have some details wrong. So please do not send me hate mail, any listeners.

Basically, specter meltdown were a family of attacks on CPUs, specifically Intel CPUs. Although I don't know if they got found in AMD eventually as well. Initially the AMD's CEO denied it, but I

don’t know if they’ve figure out that he was lying or not. Anyway, it doesn’t matter. The point is, is that it was a bug in CPUs. So you’re like, “How can there be a bug in my CPU?”

Well, one of the things that CPUs do in order to process things efficiently is that there is this concept called pipelining in a CPU, where your single instruction actually does a number of different steps. So you can imagine something like an if statement. If you are executing a bunch of instructions sort of at the same time and you run into an if statement, you don't know whether to execute the true side or the false side correctly first.

So hardware has this concept called branch prediction that sort of tries to guess the future and see if it can think that it's going to be true or false and it will start preemptively executing the one that it thinks is going to happen. So normally what happens is if, okay, if thought that true is going to happen, but false actually happens. So it does this thing called a line stall and it throws away the code that was executing and starts executing the code that it should have been doing.

So the code that was executing should not have ever been observable to outside people, but it turns out that due to shenanigans, you could actually get it to observe these kinds of situations. That’s kind of like the core idea of it, is that like the CPU was supposed to be – It’s called speculative execution, because it’s speculating on the future and what should be correct and what should not be correct. So you could do this attack where you would be able to recover this kind of information that you weren’t supposed to be able to see, and that can cause all sorts of like problems.

The reason this was such a big deal, is this model of execution is kind of like the foundation of the way that we’ve been making CPUs fast for the past like 40, 50 years. So removing those optimizations makes them very slow again, but leaving them in causes security issues. So the similar situation could happen with multithreading in your browser, where the JIT would sort of like speculatively execute certain code and then you’d be able to recover sort of things, and that's kind of like the sort of the idea. So it’s the situation. Any time there’s this like multithreading stuff where you can accidently expose that information.

[0:37:40.9] JM: Okay. So because of that general risk of multithreading WebAssembly, multithreading got put on hold.

[0:37:48.7] SK: Yeah. That's sort of like stops that idea that you can be executing two things at once and sneak peek at stuff. So I guess there is like one subtlety that I left out of this thing, which is like it's not just like in an if statement and the code you're running, because you're running multiple threads. Each thread is executing speculatively. So once thread could peek on the speculative execution of another thread and leak information from a different program of that. So that's like the issue.

In your scenario where you have a game running in one browser and you have your like dog running in the other tab, the game could, in theory, if we implemented multithreading poorly, would be able to peek information in your DAW thread and like spy on whatever's going on there. If that's, say, PGP encrypted email, instead of just making some audio, that would be a big deal.

[0:38:38.5] JM: Indeed. Okay, well we can put multithreading on the back burner. There are other features of WebAssembly that are being worked on. So garbage collection, for example. When we talk about garbage collection, the context of WebAssembly, what kind of garbage collection would be useful to WebAssembly in the browser?

[0:38:57.7] SK: So what this actually means is that [inaudible 0:39:00.9] providing an API. So maybe I should start using more precise terminology. So the embedding environment for WebAssembly is called the hsot. So that's usually web browser. But like I said, it could be the Ethereum VM or any other thing that embeds WebAssembly inside of it. So the idea for the GC proposal is not to like pick a GC and put it into the WasmVM. It's to provide a GC API that the host would be able to implement a GC of their choosing and then expose that to WebAssembly programs for their use. So it'd be kind of like relying on whatever GC the host has made to be your GC implementation.

The reason that this matters is, basically, instead of Wasm getting its own GC, it would be reusing the GC from the JavaScript VM, and that would be helpful for being able to like share objects across the boundary, because like a big problem in this sort of area is if you have – It's sort of like why you don't extend Ruby apps with Python. You extend them with C, because like you don't want the Python and Ruby GCs both fighting with each other over who controls what

memory, right? The world is much simpler when there's only one GC running inside of your process.

So this would allow Wasm programs to reuse the JS GC, which would be much, much nicer conceptually and there's less contention and all that kind of other things. This would also slim down what I mentioned earlier about like sticker binaries of interpreted or dynamic language on WebAssembly. This would allow them to not ship their own GC, and so it'd make the binaries smaller. That's another reason why this would be exciting to people.

[SPONSOR MESSAGE]

[0:40:51.8] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes. You can quickly provision clusters to be up and running in no time while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked into any one vendor or resource. You can continue to work with the tools that you already know, such as Helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications offline. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

To learn more about Azure Container Service and other Azure services as well as receive a free e-book by Brendan Burns, go to aka.ms/sedaily. Brendan Burns is the creator of Kubernetes and his e-book is about some of the distributed systems design lessons that he has learned building Kubernetes. That e-book is available at aka.ms/sedaily.

[INTERVIEW CONTINUED]

[0:42:27.7] JM: We've done this sort of survey of different aspects of WebAssembly. Tell me, where are we in terms of development? What are the big development bottlenecks and what are the difficulties in terms of what's being worked on right now?

[0:42:43.4] SK: Yeah. Fundamentally, the difficulty is, is that it is still early on, and so it just feels vabuely Wild Westy in a number of different ways. So depending on where you are personally, it can either be very exciting or like far too early for you to consider using. I think it's really important as developers to sort of understand where you personally fit on that sort of adoption curve. I'm in a moment in my career where I like emerging new technologies where there's sort of a lot of space to figure things out, and I'm less excited by mature or well-supported technologies.

If I was the VP of engineering at a Fortune 500 company, I would have a totally different opinion, right? So that sort of fits with WebAssembly broadly. So there's a lot of stuff that's kind of like rough around the edges.

So, for example, debugging is the area where people are thinking a lot about right now. You can do some WebAssembly debugging inside your browser, but sort of similar to if you take JavaScript and you obfuscate it or do all your tree shaking and module compiling and all that shenanigans. The JavaScript that comes out on the other end is very different than the code that you wrote, right? Because it's minimalized. It's obfuscated. It's optimized. So you need source maps to be able to get from like debugging in your browser back to the original code that you wrote.

The same thing is kind of true of web assembly, except for that it hasn't really fully worked yet. There's like some stuff where it kind of works, but people are exploring options and they want to do something a little better than the source maps that are used in JavaScript. So figuring out a debugging experience is something that's kind of like an area of active work that is like not as mature as maybe you would like it to be at the moment.

The other one is that since you usually don't write WebAssembly by hand, you write in some other language that compiles two WbAssembly. The support for doing so is drastically different depending on what language you're choosing. Some of them are significantly easier than others

and some of them has to do with maturity or like the desire that a particular language has to sort of expose WebAssembly stuff.

Some of them are driven by the teams of language themselves. Some of them are people from the outside being like, “Look at the cool weekend hack I did to add support, and I’m not planning on submitting it upstream.” It really just kind of depends. Which language you choose can also have an impact on kind of like how nice the experience is at the moment.

The third one, and the last one [inaudible 0:45:04.9]. The second is that crossing the boundary between WebAssembly and JavaScript is sort of an area that is – Like it's well-defined at a low level, but has varying degrees of like support and tooling based on the language that you’re working on as well. In some sense, WebAssembly only has numbers. If you want to pass a string between JavaScript in your language, like how does that work? That's up to the language to implement and some of them have better implementations and easier implementations that other ones do. That's also sort of an area of maturity that’s kind of lacking at the moment.

[0:45:40.8] JM: I kind of wanted to run by several technologies and just get your opinion on how you think they might interact with WebAssembly in the future. By the way, I think we’re glossing over a lot of the lower -level elements of WebAssembly, and I'm doing that because we already covered that in that episode with Lin Clark. We talked about pretty much the low-level, like basically the compilation path of WebAssembly.

Like I said, that episode is great. But you've actually been writing a lot about the future of WebAssembly and the past. Bbefore we dive into the technologies of the future and the present, we could contrast WebAssembly a bit with things of the past, because you've written these articles about these. So we have these older web technologies, like Java applets and Flash programming. How do you see WebAssembly as a contrast or a similarity to these older technologies that gave us more flexible web development capabilities?

[0:46:42.2] SK: Yeah. So you referenced this blog post that I wrote. So I’ll kind of give you the summary of this blog post. One thing I will say if anyone goes and reads that particular poster posts, is that I’m actually planning on doing a follow-up talking about like how these

technologies were good as well as bad. So I'll give you sort of the like mix of the post I already wrote and the one that I want to write.

So a lot of people are like, “Is this the return of Java applets, or is this the return of Flash, or [inaudible 0:47:07.1] or all these other technologies. So the web has been a very different place throughout its history, and for a long time Flash was a very, very big part of using the web. But flash was plagued with a number of security vulnerabilities and some other problems. The same thing with Sun Microsystems at the time, who owns Java, really wanted to push Java as sort of the language of the web. Java and JavaScript were sort of designed at the same time basically. 1995 was a huge year for programming language incidentally. So much of our technologies today was released in 1995. It’s just kind of absurd.

But these, specifically Flash and Java Applets had a number of different sort of problems when it came to the web, and they were good solutions at the time and had a lot of like innovative aspects to them. But they sort of didn't really like fix these kinds of issues they had, so eventually sort of fell by the wayside. The core issue was that they were not particularly well-integrated into the web as a platform. They sort of served as a platform for these two separate companies, Adobe and Sun to sort of exert their corporate power over the web instead of being a true open kind of player.

So what this means in practical terms is if you used the Flash website or one that is using Java Applets, you kind of got this blackbox. You sat there and you waited for it to load, and then it would eventually load it over your 56k dial-up connection, or 14.4, or whatever. It would end up drawing its own kind of little universe. It felt very different than the rest of the web, because you’d be using different buttons and different menus and like all these things.

So it wasn't really so much like part of the web, as it was kind of this Trojan Horse of these environments into a web context. Sort of by contrast, WebAssembly is an open specification that has all of the major web companies coming together working on it. So it's not really owned by one organization, and that's really important. It also fits into the existing web much better. Part of that is because the existing web was a much more like complicated place back in the heyday of the Flash and JVM Applets kind of thing. But like the integration with JavaScript is very clear and the integration with the DOM is very clear. While you technically can write a

WebAssembly app, where everything is one big giant canvas and you draw everything into it. It's not really where we're seeing it being used. We're seeing it like a sort of a Dropbox thing being like augmenting your existing application that's written in HTML and CSS.

So all those things are kind of like reasons why it's significantly different. I guess the last one is – So I talked earlier about the GC and WASM being like integrating with the JS virtual machine. In your browser, the Wasm wise implementations are actually in the JavaScript virtual machines themselves. So the WebAssembly runs on the JSVM, and that's really important for terms of developing the web. The fact that the Flash VM and the JVM were separate VMs, meant that if we wanted to include those into the web platform, every browser would be required to ship to VMs, and it's hard enough already to ship one. So the fact the WebAssembly plays nicely with the existing JSVM is a huge like maintenance and also security win. Those security issues happen due to this lack of integration and they often happen at the points where universe of these technologies would interact with sort of the rest of the web. WebAssembly is kind of baked into it. It doesn't have these kind of rough edges the same way.

[0:50:36.1] JM: Yeah. Well, the corporate side of it, as well as the historical side of it, I could see how it's so different. WebAssembly is so different, because we're now in this time where companies are more engaged in co-op petition and jujitsu of making technologies that maybe even complementary. Not necessarily directly competitive with other cloud providers, or even if they are competitive with cloud providers, it's a competitive edge if you make yourself interoperable, because if you make yourself embittered to developers, then you're just going to like loose support over time.

So it's a much more savvy, competitive world, instead of the, "Let's just try to take everything and make a technology that's entirely disjoint from everything else." WebAssembly is more of a platform. Also, I think we've just had so much more time to develop with the web and developers are just getting better and better. So the abstraction of WebAssembly is so much more sophisticated than I think the older technologies.

[0:51:39.3] SK: Yeah, it's like true and not true at the same time, to sort emphasize that slightly before we move on. The other reason WebAssembly has succeeded is that it is tiny. So the spec is actually like very well-written, but it's also very small, and that's what enabled everyone

to agree on it. If we wanted to put java applets into the web platform, we would need to specify all Java and all of the Applets and have all of the major web companies all agree on that spec and come up with independent re-implementations, and that will be a massive undertaking.

Whereas you can feasibly, as an individual, write a WebAssembly interpreter in, let's say, like a week depending on how good you are at writing interpreters. But like the spec is very small, and that's a huge benefit in terms of actually shipping a thing, getting everyone to agree. So by starting small and then adding in a backwards compatible way, that's also the model the web has taken, rather than like, "Here's a big giant thing." Start with this. You start tiny and you add things.

[0:52:39.8] JM: Okay. Well, let's begin to wrap up by doing a rapid fire of how WebAssembly will impact some other technologies. You can just pass on some of these, or you can say that you have no idea. But let's start with progressive web apps.

[0:52:55.0] SK: So this is like a half answer, because I've not written that many progressive web apps. But a lot of people are looking into using things like web workers, where WebAssembly runs in the web workers and using these kind of technologies that are part PWAs. So I think that it will be part of an implementation of some of them, but is not necessarily going to revolutionize the entire concept.

[0:53:11.9] JM: What about traditional mobile apps?

[0:53:14.5] SK: So traditional mobile apps, one of the interesting things about speed, is that speed is also better on your battery, because the longer a CPU can sit idle, the less power you're using. So I think that Wasm will be important for mobile, because performance is a battery life feature.

[0:53:30.3] JM: And it would also let you run, like if you're running an android app, you could use any kind of code, right? You just use a WebAssembly module?

[0:53:39.2] SK: Yeah.

[0:53:39.8] JM: What about Flutter, like the dart Flutter project? Have you looked at these at all?

[0:53:43.9] JM: From the outside, and I'm very excited about them. But I do not know specifically how WebAssembly would change them.

[0:53:49.7] JM: What about serverless runtimes? The functions as a service?

[0:53:54.1] SK: So serverless is very interesting. WebAssembly can be a hack around the fact that serverless platforms only accept certain languages, because you can run WebAssembly and code in node. For example, AWM Lambda does not support Rust directly today, but it does support node. So you can compile your Rust to Wasm and then run it inside of node and then upload that as a node serverless application instead. So it's kind of this Trojan Horse into using other like languages as a platform. That's sort of a little lockdown.

[0:54:19.6] JM: Have you looked at GraalVM?

[0:54:22.0] SK: Yes. I am a huge fan of GraalVM.

[0:54:24.8] JM: Okay. Tell me about a GraalVM.

[0:54:26.4] SK: All right. So GraalVM and this companion technology Truffle are coming out of Oracle Research, who now owns Java. Unlike Sun, as we talked about earlier. It's this sort of idea that you can write a virtual machine that knows how to do all kinds of shenanigans. So it is extremely impressive work. They have done things like – So for example, if you want one problem – Okay. So my entry point into this is JRuby, which is Ruby running on the JVM.

So one of the problems of using JRuby as opposed to regular C Ruby is that you can't use C extensions, because of the JVM. You need Java specific extensions. What the Graal people have done is they've had a way that it can read in a C and then JIT compile it to JVM bytecode so that you now have like Java extension instead of a C extension. They've even gone so far as there was the experiment a couple years ago to be able to take and read x86 assembly code and then compile it to the JVM bytecode and then have the JVM JIT it back out appropriately.

But it's sort of this idea that you can have this VM that's able to understand a whole host of technologies and do all sorts of things. So I don't think WebAssembly will change GraalVM so much, as it's just one more thing for them to kind of be able to understand and [inaudible 0:55:43.1] with all the other sort of stuff. But it's one of the most exciting things happening in compilers today.

[0:55:49.6] JM: What about cryptocurrency infrastructure, such as Ethereum? I know you touched on the parity implementation a little bit earlier. But I think, today, the world of cryptocurrency is almost disjoint from the rest of the development world. Do you see WebAssembly as any kind of like interoperability layer between the two?

[0:56:08.8] SK: Yeah. The closest is that stuff I was talking about with Ethereum. I'm not really that big of a believer in cryptocurrency, and so I don't follow it very closely. So I'm not really sure.

[0:56:17.7] JM: Okay. All right, fair enough. I guess let's zoom out and talk more generally. What are the other future ramifications of WebAssembly and how do you see them evolving overtime?

[0:56:28.3] SK: I think it depends. So in your term, it's going to be that you'll be using packages from your JavaScript that happened to be WebAssembly and you don't even necessarily know about it, because you can just use them like they were JavaScript packages. future ramifications of WebAssembly and how do you see them evolving over time. so in the near term.

In the long term, writing more things completely in WebAssembly is like a thing and we'll see how that goes. Then in theory, in the far, far future, having everything happen in be that you will be using packages from your javascript that happened in the WebAssembly is a possibility, but less of a probability.

[0:56:53.9] JM: You said at the beginning of the conversation that you're spending a lot of your time on Rust these days. Since you're looking at Rust and WebAssembly, what makes Rust a compelling language for WebAssembly modules?

[0:57:07.4] SK: So there's sort of two big reasons that I can posit. One is that on a technical level, Rust does not have a runtime and a garbage collector and all that stuff. So we produce extremely tiny modules. For example, the smallest module that's been increased by the Rust compiler was 114 bytes. Not kilobytes, bytes. So in terms of getting that sort of speed and like low overhead, that's an area where Rust really shines.

Additionally, the other reason why I think Rust and WebAssembly is going to be awesome, is because we on the Rust project hear about WebAssembly, and we're investing tremendous amount of time and resources to make the experience excellent. Whereas other other languages maybe are quite so into it yet, but we really think that Rust is a fantastic language for writing WebAssembly and we kind of want to it to be one of the first languages that you think of [inaudible 0:57:52.5] consider writing WebAssembly code. That involves like developer experience and all that other stuff.

[0:57:58.3] JM: Are web developers going to have to learn to write in a language like Rust or are they just get to be doing something like NPM install X module that is actually going to be a WebAssembly module?

[0:58:09.9] SK: In the near future, I think that the latter is mostly true. I think that this is largely a feature for library authors that want to have a library that does certain really awesome things. Most developers are just going to use it.

As an example of that developer experience, there's a tool that we've developed called Wasm Pack, and it allows you to compile your Rust to WebAssembly and then upload that WebAssembly to an NPM mobile and then your users just use it and don't even realize that it exists. That's really trivial to do in Rust and it's sort of a great example of how like people will end up using these packages without even realizing it.

[0:58:42.1] JM: Yes. You can also imagine a world we're still using things like React to compose our user interfaces. We're still using JavaScript to glue together are different modules, and the only thing that really changes is the performance of those modules and the web just gets more performant.

[0:58:59.9] SK: Totally.

[0:59:00.8] JM: Okay. Well, Steve, thank you for going back on Software Engineering Daily. It's been really great talking to you. I had a bunch more questions, and I'm sure we'll do this again sometime.

[0:59:07.7] SK: Yeah. Thanks so much for having me again. It's always a pleasure.

[END OF INTERVIEW]

[0:59:12.9] JM: You listen to this podcast to raise your skills. You're getting exposure to new technologies and becoming a better engineer because of it. Your job should reward you for being a constant learner, and Hired helps you find your dream job. Hired makes finding a new job easy. On Hired, companies request interviews from software engineers with upfront offers of salary and equity so that you don't waste your time with a company that is not going to value your time. Hired makes finding a job efficient and they work with more than 6,000 companies, from startups to large public companies.

Go to hired.com/sedaily and get $600 free if you find a job through Hired. Normally, you get $300 for finding a job through Hired, but if you use our link, hired.com/sedaily, you get $600, plus you're supporting SE Daily. To get that $600 signing bonus upon finding a job, go to hired.com/sedaily.

Hired saves you time and it helps you find the job of your dreams. It's completely free, and also if you're not looking for a job but you know someone who is, you can refer them to Hired and get a $1,337 bonus. You can go to hired.com/sedaily and click "Refer a Friend".

Thanks to hired for sponsoring Software Engineering Daily.

[END]

