EPISODE 634

[INTRODUCTION]

[0:00:00.6] JM: Java Script has been the exclusive language of the web browser for the last 20 years. Whether you use Chrome or Firefox or Internet Explorer or Safari, your browser interprets and executes code in a virtual machine and that virtual machine only runs Java Script. Unfortunately, Java Script is not ideal for every task that we want to perform in the browser. Think about the use cases where you need to use software outside of the browser today?

If you're like me, you do almost everything in the browser except for things like video editing, music production, 3D art, video games, these applications require a high degree of performance that is hard to get from raw Java Script. Web Assembly was created to get better performance on the web. Web Assembly allows code from other languages to be compiled and run in the browser. With Web Assembly, languages such as C, C++ and Rust can be used to achieve major performance gains.

Web Assembly is still under development and eventually more programming languages will be accessible as well. Lin Clark is an engineer with Mozilla and she's been working closely on the Web Assembly project. She's the author of a detailed series of illustrated blog posts that explain how Web Assembly works.

In this episode, we discuss how Web Assembly came to be and talk about its advantages over a web purely driven by Java Script. We also explore what is possible with Web Assembly as well as its engineering implementation in some granular detail. I really enjoyed this episode and I am looking forward to doing more shows about Web Assembly in the future.

[SPONSOR MESSAGE]

[0:01:52.2] JM: Cloud computing can get expensive. If you're spending too much money on your cloud infrastructure, check out Do It International. Do It International helps startups optimize the cost of their workloads across google cloud and AWS, so that the startups can spend more time building their new software and less time reducing their cost.

© 2018 Software Engineering Daily

Do It International helps clients optimize their costs and if your cloud bill is over $10,000 per month, you can get a free cost optimization assessment by going to doit-intl.com/sedaily. This assessment will show you how you can save money on your cloud and Do It International is offering it to our listeners for free.

They normally charge $5,000 for this assessment but Do It International is offering it free to listeners of the show with more than $10,000 in monthly spend. If you don't know whether or not you're spending $10,000. If your company is that big, there's a good chance you're spending $10,000 so maybe go ask somebody else in the finance department.

Do It International is a company that's made up of experts in cloud engineering and optimization. They can help you run your infrastructure more efficiently than helping you use commitments, spot instances, right sizing and unique purchasing techniques. This to me sounds extremely domain specific, so it makes sense to me from that perspective to hire a team of people who can help you figure out how to implement these techniques.

Do It International can help you write more efficient code that can help you build more efficient infrastructure. They also have their own custom software that they've written which is a complete cost optimization platform for google cloud and that's available at reoptimize.io, it's a free service if you want to check out what do it international is capable of building.

Do it international are experts in cloud cost optimization and if you're spending more than $10,000, you can get a free assessment by going to doit-intl.com/sedaily and see how much money you can save on your cloud deployment.

[INTERVIEW]

[0:04:15.6] JM: Lin Clark, welcome to Software Engineering Daily.

[0:04:17.8] LC: Thank you so much for having me on.

[0:04:19.7] JM: You're an engineer at Mozilla and you have been working on Web Assembly and you've written some excellent post that describe how Web Assembly works. I want to start by talking about Java Script in the browser because I think that will gradually get us towards the motivation for Web Assembly and what it actually is.

When Java Script first came out, what did the browser do to process it?

[0:04:46.1] LC: Well, initially, browsers used interpreters and I want to explain a little bit about what an interpreter is in case folks that are listening aren't so familiar with them. Your browser basically needs to go from the source code, the source Java Script code that was run by the developer and turn it into machine code that your computer actually understands.

This is how running an application in your browser is different from installing that same application on your computer. When you install the application on your computer, what you have is compiled code, it's already machine code. But when it's coming into your browser, it's in a human readable, you know, it's the source code and you need to translate it into the machine code.

There's this translation process that's happening in your browser and in the early days, the browsers were doing that completely on the fly. When you load the page, doing this translation process is part of what happens during the loading of the web page. These browsers, the would get the file and then start going line by line and doing that translation and then running the code at the same time and executing it.

That's called an interpreter, when it does it on the fly. The nice thing about an interpreter is that it starts running your code really quickly. But, there's an alternate approach, an ahead of time compiler that will actually do that translation to machine code before you start running your program.

That has the nice benefit that – with an interpreter, if you are doing a loop, if you're in a loop, you have to keep doing that translation to machine code over and over again for the code that's in that loop. With an ahead of time compiler, you don't actually need to do the translation over and over again, you just have your already translated code that's in the loop.

© 2018 Software Engineering Daily

[0:06:32.8] JM: Right, when people are learning to code in Java for example, kind of productive to C to describe it this way but their experience is they compile the code, there's a compilation or objectification step and then the code can actually run, it just runs, it might as well be in binary as the programmer understands it and it's fast in contrast to interpreted languages where line by line, the code is turned into executable machine code.

Then somewhere in between the interpretation mode of processing and the compilation mode of processing, there's the just in time compilation process. What is a just in time compiler and when did just in time compilation come to Java Script?

[0:07:23.2] LC: Exactly, yeah. Just in time compilers were created to weigh these tradeoffs. In about 2008, both us at Firefox and the Chrome team, it was right around when Chrome was released, we were both working on just in time compilers. Ours was called Trace Monkey and it came out a few weeks before Chrome's did.

The idea behind the just in time compiler was that first you would run the code to the interpreter and you would see, which parts of the code were being run a lot, where you would actually get a lot of performance gains from compiling that code and run it through a – well, this architecture has changed a bit over the years.

I'm going to describe the architecture that most of the browsers have now. Once you run it, maybe about 10 times, that's what it is for us in Firefox. Then, the parts have been run, you know, 10 times or more will be sent to a baseline compiler and that baseline compiler is going to compile to machine code and store it.

For a functional, that's being called more than 10 times, we'll compile that to machine code and store it. Then we'll actually do this because you have to have different compiled versions for different types. When you have a string passed into a function, the compiled code you get for it is going to be different than when you have a number passed into that same function.

You'll need to have two stubs, two bits of compiled code, it's called with either of those types. That's what the baseline compiler does. The interpreter will keep her, the jip will keep monitoring

to see how often the functions are called, if they get called a whole lot I think for us at Firefox, it may be a thousand or 10,000 times. Then it will go to an optimizing compiler and the optimizing compiler will you know, take the time to figure out what the most optimal way to translate this into machine code is.

Now, optimizing compilers are kind of slow because they have to take a lot of time to figure out, what is the most efficient way to do this? Once they've done that and they've stored it, then they run a lot faster. The one thing about optimizing compilers is, to make those kinds of shortcuts, to make those kinds of optimizations, you really need to have fixed types and so optimizing compilers will usually require that this function be called with the same types all the time.

And if one call happens, that's not of that type then the optimizing compiler will actually throw out the compiled code that is compiled. Start monitoring again to see if it should optimize this function again, it will keep checking the types. But this process can actually end up slowing down your code if it happens enough times, if you keep optimizing and then throwing out the code that you've optimized. There can be some real tradeoffs to weigh here.

[0:10:12.6] JM: Originally, Java Script was just interpreted, and this is sub-optimal because the code has to be reinterpreted every time a loop runs. If you have a four loop and it's going to run 10 times and then naively interpreted world, it's just going to be converted to binary 10 different times which is pretty expensive and a just in time compiler can improve that by understanding that the code is going to be executing 10 times so we can just compile it to binary and then rerun it 10 different times which is going to be an optimization.

But the downside is that this just in time compiler has to do a lot of overhead, it has to be very aware of what is going on in the code path. It needs to understand what is a hot code path and it needs to understand what is a hot code path. I think if I understood you correctly, you are eluding to the fact that it's actually harder to do in an untipped language like Java Script because it's harder to make a smart compiler when you have untyped code.

Because the system has to – kind of infer what type something is when it's doing the compilation. Am I understanding things correctly?

[0:11:28.0] LC: For the most part, yes, the fact that Java Script is untyped is a part of why it's slower than it could be. The heart of the matter is the dynamic types. The fact that the types of variables can change at any time is the real problem there because that optimizing compiler, if it could depend on when it sees this function being called with a string or with a number, that it will always be called with a stringer and upper.

You would just need to monitor the first time it was called. Then you could actually optimize based on whatever it was called with. You could see what the type coming in was at that point. In Java Script, you can't make those kinds of assumptions because it could be called with a number of 10,000 times and then called with a string the 10,000th and first time.

That is where you get into those optimization and re-optimization, bailing out is what it's called, where you bail out and re-optimize over and over again and that can be a performance cliff. Also, with Java Script, because you can't have the type changing on you, you have to add these guards, you actually have to check the types that are coming in before you pull out a compiled bit of code and make sure that they're the types the same as what you expect. Those guards can be expensive too.

[0:12:40.6] JM: One way of looking at the Web Assembly project is that it's going to get us to a way of executing code in the browser faster. Another way of looking at it is it's going to get us to a place where we can execute different languages in the browser other than Java Script. As we move towards a conversation about Web Assembly, how does the past of how Java Script in the browser has evolved –

How does that inform where we got to with the browser vendors to side, "Okay, we need to start working on this Web Assembly thing?"

[0:13:18.4] LC: Well, Web Assembly was very informed by the history of the web, that is very true. There were a couple of different approaches that different browser vendors were pursuing around the same time. At Mozilla, we had this thing called ASMJS which was basically taking that problem that Java Script had of these dynamic types and saying that in ASMJS, you would just use a character to say, "No, this isn't a dynamic type, this is just a number, this is just an

integer" and by signaling to the engine that this is not dynamically typed then the engine couldn't make those optimizations, do those shortcuts.

That was the sieve for Web Assembly, there was also another project going on at Google at the same time called Pinnacle that was kind of similar in its aims but with a slightly less web focused. It didn't build so much on existing web technologies.

Once we had ASMJS, we proved that it could be done, we started talking with the other browsers, with the folks on the pinnacle team and seeing, could we do something that takes this basic principle of reducing the dynamism of those types and could we use it in the same way as we would use Java Script where you ship it down, you know, you fetch it, you execute it in the VM in the browser.

But, it gives you this speed, this performance, predictable performance that you don't have with Java Script usually.

[0:14:56.4] JM: When we're talking about other languages that we would want to run in the browser, What's a concrete example of something from a language outside of Java Script that I would want to run in the browser?

[0:15:09.8] LC: There are a bunch of different examples, the earliest examples were PC games. One of the reason that was a really good use case for Web Assembly is because when you have animations in games, you really are very sensitive to making sure that you get the framerate correct, that you aren't dropping frames. Because if you drop frames, that makes your animation really jumpy and that's not a great game experience.

The PC game industry was the first partner that we worked with when we were working on Web Assembly. But there are lots of applications that can benefit from this kind of performance, from this ability to have consistent performance and to manage memory directly instead of through Java Scripts memory management.

One application that you could see coming to the web is something like Adobe's Photoshop. That would make it possible for you to, instead of installing Photoshop on your computer, just

fire it up on the web whenever you wanted. And these other kind of large applications, that you traditionally think of as applications you have to install are applications you could, in the future, see being web applications.

[0:16:13.4] JM: Actually, that's an incredible example because I use a program for writing music on the computer and it's mostly – I think it's a C++ application and writing music on the computer, something that the browsers have not been able to do, I think for the very same reasons that you can't run Photoshop in the browser.

[0:16:34.8] LC: That is exactly correct, yes. That industry, the folks that make those tools for making music are really starting to get very interested in Web Assembly. I think it's a company called Propeller Heads maybe, recently released some of their tools in browser versions. The great thing about Web Assembly is that they can just take their existing code base, take all of the business logical of that back end stuff, compile that to Web Assembly and then put an HTML and CSS front end on top of that.

Which often is actually easier to work with than the toolkits that you're using for user interfaces and their native applications. It actually not just makes it available more widely but also can be a better developer experience.

[SPONSOR MESSAGE]

[0:17:25.2] JM: At Software Engineering Daily, we're always analyzing data to determine what our listeners care about. We actually have a lot of data, even though we're just a podcast. It always reminds me that organizations with much more engineering going on have an order of magnitude, more data than a podcast like Software Engineering Daily. That's where the job of data scientist is such a good job to get. Flat Iron School is training the next generation of data scientists and helping them land jobs.

Flat Iron School is an outcomes focused coding boot camp that offers transformative education in person and online. Flat Iron School's data science program is a 15-week curriculum that mixes software engineering, statistical understanding and the ability to apply both skills in real life scenarios.

All of the career changing courses include money back guarantees. If you don't get a job in six months, Flat Iron School will refund your tuition and you can visit their website for details as a Software Engineering Daily listener, you can start learning for free at flatironschoool.com/sedaily.

You can get $500 off your first month of Flat Iron School's online, data science boot camp and you can get started with transforming your career towards data science. Go to flatironschool.com/sedaily and get $500 off your first month of their online data science course. Thanks to Flat Iron School for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:19:17.7] LC: As we move towards a conversation around how this kind of code is going to execute in the browser, how C++ code or Rust code is going to execute in the browser, we need to have a conversation about intermediate languages and virtual machines. Different programming environments have implemented virtual machines.

Java Byte Code for example, there's a virtual machine where your Java code compiles down into byte code and then that byte code gets compiled into machine code if I understand it correctly. And then there's this huge ecosystem around other languages that's called the JVM ecosystem.

Other languages such as Closure or Scala that compile into JVM code and so you get kind of economies of scale because people make optimizations to the JVM and that ends up optimizing all of these different languages that can compile to the JVM code and you saw a similar phenomenon with LLVM, that's another virtual machine ecosystem.

This idea of the intermediate language, the intermediate byte code representation and the virtual machine that can run that intermediate language, what are the advantages of that? What are the advantages of having this intermediate run time?

[0:20:40.3] JM: Well, intermediate language and intermediate representation, actually have a long history. Even proceeding things like java, they were used in compilers as a way to make it easier to reuse different parts of the compiler. Early compilers, you would have to rate a new compiler when you wanted to go from one programming language to a particular target architecture because different devices have different machine codes that they understand.

The machine code that runs on your phone is going to be different than the machine code that runs on your computer probably. That's why you know, when you're going to download a program from the internet, that will ask you, "Do you want the Mac version or the Windows version?" Because they actually speak different machine code languages.

When you're writing a program in a source language and you want to compile two particular machine code language. Early on, you would have to have a specific compiler for each pair. But what having an intermediate representation does, is it makes it so that you can compile from your source code into intermediate representation.

Then that single intermediate representation can go to any one of the different machine code targets that the compiler targets. That gives you a lot more reusability, you can have the thing that goes from source to the intermediate representation, that's called a front end so you can have a bunch of different front ends for different languages and then the thing that goes from the intermediate representation down to the machine code, that's called the back end.

That is why you have these and then they were used also for virtual machines, for these things that don't necessarily compile ahead of time but actually are running your code at one time.

[0:22:17.4] LC: Why is it beneficial? Why would I want to go to an intermediate language, why would I want to go from Java to the JVM instead of going from Java straight to the processor's language or go from Rust or to the LLVM's of going Rust directly to the processor?

[0:22:32.5] JM: Well, like I did mention, it gives you the ability to have one tool chain so if you write a front end that can go from Rust to a particular intermediate representation, then you can go from that intermediate representation to all of the different processors, all of the different machine codes that that particular compiler support.

It gives you that extensibility. But, there's also – in the case of language agnostic compilers, things like LLVM, the nice thing is that there are these optimizations that happen when it's in the intermediate representation and because they're language agnostic, they can be applied to code that's run in all different kinds of languages, so you get to reuse those optimizations across different programming languages as well.

[0:23:12.9] JM: We're going to eventually get to Web Assembly. But LLVM, well, it is the first intermediate language that can translate into Web Assembly, or at least that the browser vendors and the other people who are working on Web Assembly are trying to get implemented. They're trying to build a way to convert LLVM code into Web Assembly code and Web Assembly code is code that can run in a very performant fashion in the browser. First of all, did I get that right? Is that the right way of articulating it?

[0:23:46.3] LC: Well, LLVM actually does support this at this point, it's less that we're actively working on making it work as now it's more optimizing things. LLVM can already target Web Assembly. Rust is actually using it as t heir back end to go from Rust to Web Assembly.

[0:24:06.9] JM: Okay, I could go from Rust to the LLVM and then from the LLVM to the Web Assembly, and run my Rust code in the browser today?

[0:24:16.9] LC: Yes, you can do that. This is work that is currently in progress, there are still a few things that – kinks that need to be ironed out but if you're adventuresome, you can try it out. It is running in production in a number of different tools. For example, the source maps, well, the source maps Java Script library that is used in tools like Web Pack and in our Firefox developer tools. That currently has some Rust compiled to Web Assembly that sped up performance by a great deal, there's a great article by Nick Fitzgerald on exactly how that optimization happened, which I can add a link to the show notes.

[0:24:54.9] JM: Yeah, that sounds really interesting. The process of going from LLVM to Web Assembly, this is going from one intermediate language to another, right? Because Web Assembly itself is an intermediate language before hitting the compiler's instruction set, is that right?

[0:25:15.0] LC: It's kind of hard to exactly pin down what Web Assembly is because Web Assembly, you could really think of it as an ISA in and of itself. But it's just a ISA for a virtual machine, not for an actual physical machine. But it's fairly low level, I would consider it lower level than LLVM's intermediate representation.

[0:25:35.5] JM: I see. You want to get there as supposed to – you want to get to this other instruction set, the Web Assembly instruction set rather than going, getting the LLVM intermediate representation because it's going to be even more highly optimized. For example, it would not be a viable solution to the problems of running code in the web to just say, "Let's try to build LLVM for the web?"

To take the kind of counter factual, what would be the problems with trying to implement just like the LLVM and the browser?

[0:26:08.2] LC: Well, you know what? It's funny, they actually did consider that and so if you look, I think in the facts for Web Assembly, maybe in the design repo, there is a question of exactly that. Why not just use LLVM's intermediate representation? I'd say that there are a couple of reasons why Web Assembly is better than just shipping down LLVM's IR. One of them is that because Web Assembly is –

We kind of know what we want to do, we have kind of a more restricted problem set. Some of the Web Assembly stuff has been tailored, some of the instructions, the way it runs have been tailored for performance particularly in these virtual machines that are running this code. More importantly I think, LLVM's intermediate representation would just be a lot larger.

It's not as compact, Web Assembly is designed to be compact. That is important because part of the user's experience loading a page is the downloading of the files. I think that downloading a byte of code is something like a hundred times more battery intensive than parsing that same byte of code. Don't quote me on that because I've never actually checked that way.

[0:27:21.9] JM: I'm sure is directionally right.

[0:27:25.1] LC: You really want to minimize that cost of downloading. Web Assembly is quite compact and a lot more compact than LLVM's IR would be.

[0:27:34.7] JM: Fascinating. One direction this takes us is this makes there an incentive or maybe that there already was an incentive that I'm not aware of but it gives an incentive for people to write C and C++ and compiler to the intermediate representation of LLVM because the tool chain work has been done on LLVM to Web Assembly. If people want to get their C++ code running in the browser, it first has to hit LLVM code.

[0:28:01.5] LC: Well, you know what, there are actually other toolkits that are currently being worked on as well. We're working on one, besides LLVM, there's one called Binarian. It has an intermediate representation that's similar to Web Assembly but not identical to it and it's easier to optimize for Web Assembly on that.

One of the nice things about it is that it's language agnostic, so there's a lot of language, agnostic optimizations and it's particularly a lot easier for high level garbage collective languages to target it. Now, Web Assembly doesn't have support for garbage collection for integration with the browser's own garbage collection quite yet.

But it's actively very actively being worked on in the Web Assembly community group. Having a tool like Binarian which is good for those kinds of languages and is optimized for compiling to Web Assembly could be a really good – we could see this maybe as being the LLVM for those kinds of languages.

[0:29:01.4] JM: LLVM to Web Assembly. That is a compilation step that has been figured out. Now it's being optimized. Somebody could conceivably write a JVM to Web Assembly transpiration process, right?

[0:29:18.9] LC: Java's probably going to be one of the harder languages to make work in Web Assembly. I don't know that I can actually – I'm not the best person for explaining why that is.

[0:29:28.7] JM: It's okay, I can give you a pass on that.

[0:29:33.0] LC: But we don't' think that that one's going to be one of the ones that can easily target Web Assembly.

[0:29:37.3] JM: Someone's going to do it, it sounds like it's –

[0:29:39.5] LC: Yeah, someone – I think it's actually already been worked on by adventuresome people, but I think that because of some of the garbage collection language features, I think there might be a difference in the way that finalizers are being spec'd currently versus the way that Java requires uses them but like I said, I'm not the best person to talk about those details.

[0:30:01.3] JM: Okay, I guess this is an interesting point. The main language that people are working on getting their code from current client side applications into the browser today are I guess C and C++ and Rust and these are not garbage collected languages, these are languages where you manually manage the memory and this is one thing we want out of our Web Assembly modules, right?

If we're building a browser-based application, we want to make a call to this compiled Web Assembly module that's been compiled into Web Assembly and we want this thing to, this little module that we're going to be calling to manage its own memory, right?

[0:30:44.3] LC: Well, there is a linear memory, this basically a ray of bytes that emulates the heat in a Web Assembly module, so that is definitely what people use today for managing state and Web Assembly modules. But part of the integration that we are working on with garbage collection in the browser will also give Web Assembly developers the ability to work with managed objects from Java Script and the idea is that you'd be able to use them both these objects across the boundary. To be able to use them both in Java Script and Web Assembly. So eventually, you will be able to have a Web Assembly module that doesn't necessarily manage any stuff in linear memory.

But linear memory is definitely a big part of that manual memory management. It is definitely a big part of Web Assembly today, yes.

[0:31:36.1] JM: Right and for people who don't know I think this is accurate but Java Script, it's a garbage collected language and so you are not manually managing memory. Your application is just the memory management is taken care of by I guess the Java Script engine or?

[0:31:52.6] LC: Yes, the Java Script engine will figure out if you still have things in scope and if you don't, it will clean them out for you.

[0:32:00.6] JM: So this topic of a Java Script engine, we probably should have defined this earlier. So there are different Java Script engines. There is a V8 on Chrome and then I think on Firefox. Does Firefox use, is it Spider Monkey?

[0:32:14.0] LC: Yeah, we have Spider Monkey and then Edge has Chakra and V8 has JSC.

[0:32:20.0] JM: Got it, okay. So that's four major Java Script engines, what's the spec for a Java Script engine? What does a Java Script engine have to do for you in the browser?

[0:32:28.8] LC: The Java Script spec is very large. It is hard to encapsulate in a quick answer but basically, what happens when your code comes in is that it will be parsed into an abstract syntax tree and then that abstract syntax tree will be a lot of time. It's turned into what's called the byte code. So like you said, like an intermediate representation and then that byte code will be executed by the interpreter or compiled.

[0:32:57.7] JM: So anybody who has worked at a company that's building a web application has gone through this torturous process of testing on different browsers. They go like, "Why doesn't my application run the same in all of these different browsers?" Is it running different in different browsers because the Java Script engine that is running in each given browser is making subjective decisions? It's making different decisions around how to run that same web application? Is it the Java Script engine that is making that different decision?

[0:33:28.2] LC: So yes, the different Java Script engines – if you see something that is acting very differently in Java Script across different browser engines, that is because the implementations are different and usually it's that one of them has a bug. It's usually the case that one of them is not spec compliant or compliant with the specification and the reason that

this happens is because one of these engines might have done an optimization to make things run faster.

But that optimization might end up breaking things in the process. So, it is usually the case that you want to if you find one of these issues, you can actually – we have a web compatibility group at Mozilla and you can file issues on their repo saying, "I see this different behavior in these different browsers. Can you fix it?" And those folks will figure out which of the browsers is not acting in the spec compliant way and put an issue on the right repo, on the right issue tracker.

[0:34:23.6] JM: And so, because there are different Java Script engines in each of these different browsers, there is going to be some different opportunities for how Web Assembly could be implemented in these different engines, right? Or is this an opportunity to actually create some standardization around how Web Assembly is implemented?

[0:34:43.6] LC: I would say both is actually true. We are all implementing our Web Assembly support independently. So you do still have opportunities for there to be these differences and there definitely has been these differences between different browsers. At the same time, because Web Assembly itself doesn't require as many of these optimizations and it certainly doesn't require as much guessing and checking kind of optimizations, that makes it more consistent.

It's easier to implement Web Assembly as a compiler developer than it is to implement Java Script. So it reduces the number of these compatibility issues.

[0:35:18.8] JM: Okay, so we need to be able to compile the LLVM intermediate representation into Web Assembly files and as you've said, this part has been accomplished but it's not completely optimized as it is, as it stands today. What's hard about that? The process of writing a LLVM intermediate representation to Web Assembly compiler?

[0:35:47.8] LC: I don't know that it is actually that difficult. I think it is more that because Web Assembly is just so young, it doesn't have the accretion of the optimizations that have happened

over years for targeting other ISA's. So, I think that it's more just a matter of Web Assembly needs to mature than there's anything inherent to Web Assembly that makes it difficult.

[0:36:11.4] JM: So let's imagine we've got that whole compiler tool chain in our browsers today, is it in the browsers today? Can Rust actually run in my browser today?

[0:36:22.2] LC: Yeah, like I mentioned that source maps library that we have ported somewhat from Java Script to Rust, that is actually running in browsers today. A number of different codebases are running in browsers today.

[0:36:35.3] JM: So, let's give the full soup to nuts explanation. If I am going to get Rust code deployed in my browser running on an application, what's the whole tool chain look like? Getting Rust into the intermediate representation then to Web Assembly and then eventually into my browser, just kind give an overview and then we'll dive into it in more detail.

[0:36:59.2] LC: Okay, well there's I think two different overviews I can give you. For your super simple Hello World, all you need to do is compile your Rust file or your Rust project with the Rust compiler and with a flag. I think it is target unknown, unknown something, I forget the exact compiler flag but that will give you a Hello World. You can actually even do this without downloading the compiler, without installing the compiler through a tool that we have called Web Assembly Studio.

And that's an online IDE for playing round with compiling to Web Assembly. So if you just go to Web Assembly Studio, start a new Rust project and hit build and run, that will run that in the browser for you. Now if you have a more serious project, the tills you are going to use for that are somewhat different. We are working on trying to make it as easy for developers to incorporate Web Assembly modules, Rust to Web Assembly modules into their developmental workflows.

So, the tools that we are working on for that are there's this one called Wasm bindgen because one of the issues that you have when you are working with Web Assembly is that currently, the only types that you can pass into functions or that you can get as a result from functions are

integers or floats. So that's not super useful for certain different kinds of computation. You might have something like a string. Wasm bindgen will help you.

It will basically wrap your code in things that can turn your string into numbers, put it into the linear memory that goes into Web Assembly and then when you get a return from Web Assembly that is a string, it will take the numbers out of the linear memory and turn the back into a string. So, we have tools for making that a lot more ergonomic for the users. Then there is another tool we have called Wasm pack that automates the process of packaging up your Web Assembly.

Your Rust to Web Assembly project and pushing it up to MPM and then we're also working with the different bundlers to make it easy for those bundlers to bring Web Assembly modules into Java Script projects. So that's the tool chain that you would be using if you are doing something more advanced than just a Hello World.

[SPONSOR BREAK]

[0:39:23.4] JM: Stop wasting engineering time and cycles on fixing security holes way too late in the software development life cycle. Start with a secure foundation before coding starts. ActiveState gives your engineers a way to bake security into your language's runtime. Ensure security and compliance at runtime.

A snapshot of information about your application is sent to the ActiveState platform. Package names, aversions and licenses and the snapshot is sent each time the application is run, or a new package is loaded so that you identify security vulnerabilities and out of date packages and restrictive licenses such as the GPL or the LPGL license and you identify those things before it becomes a problem.

You can get more information at activestate.com/sedaily. You want to make sure that your application is secure and compliant, and ActiveState goes a long way at helping prevent those kinds of troublesome issues from emerging too late in the software development process. So check it out at activestate.com/sedaily if you think you might be having issues with security or compliance.

Thank you to ActiveState for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:40:53.5] JM: Interesting. So the last mile there, so if I get my C or C++ or Rust module compiled and ready to deploy to web applications, it still needs to be pulled into the web application. I think you mentioned something about MPM integration there. So, what's the process of getting a double Web Assembly module that has been compiled from C or C++ or Rust pulled into the web application, what's the process at the last mile?

[0:41:24.2] LC: Well it is really very similar to a Java Script module. You add it to your package like JSON file and the bundlers are supporting something that's not quite natively supported in the browser yet. I am actually championing this project through the Web Assembly community group and TC39 and the project is to make it possible to load Web Assembly modules using the ES Module syntax. The reason why that's nice is because one, it gives you an easier way to instantiate Web Assembly modules that we have today.

Today you actually have to do this boiler plate with Java Script to instantiate a Web Assembly module. So, with this, you'd be able to just say, "Import function" through or whatever from Bardot. Wasm just as you do with a Java Script module. So that is one benefit is that it gives you that better ergonomics. It also allows you to then use the Web Assembly module in the same way that you'd use a Java Script module and use them in the same module graft as other Java Script modules.

So, your users might not actually realize that they are using a Web Assembly module or a Java Script module. They would just be importing and exporting as they normally would.

[0:42:38.0] JM: What's the interface between somebody who has written a Java Script application and they now want to use a Web Assembly module, how are they interfacing with that Web Assembly module? How are they calling it?

[0:42:51.2] LC: There are a few answers to this. You might have some people, so if you're developing a Web Assembly module, you might want to give a Java Script API. So, you might wrap it in a Java Script so that it will handle converting data and stuff like that but if you are just working with numbers, then you could just have "Import random" or whatever from food out Wasm and then just call the function the same way that you would with Java Script passing in numbers.

[0:43:20.1] JM: When that Web Assembly module is loaded, I know it is giving a fixed space of memory, what else goes on? What's the boot strapping and the instantiation, the allocation process of that module getting called?

[0:43:37.4] LC: So, you have different kinds of imports that you can pass into a module. You can pass in things like values and I shouldn't say pass in, you can import things like values and those will be globals within that module's scope. So, when you are instantiating, you pass in all of those things that you want to be a part of the module. So those are the imports like you would have in Java Script. Those get allocated but not in the linear memory.

So, with linear memory, the program, the code that is actually in the Web Assembly module and the functions can really read and write that memory at will. They can see the bytes there. The imports that get passed in, the Web Assembly functions can actually see the bytes directly for that but during the process of loading you will set aside, allocate those values and then you will instantiate the linear memory. So, create that object that had – is that a ray of bytes.

And if the module had any preset data that should go in that memory, that will be stuffed into that linear memory. You can also have these things called tables which will get initialized during this point. I won't go into depth on those because they are a little bit more of an edge case at this point and then once that's all done, it will call what's called the start function. So that is if you are coming from Java Script, it is the top-level code.

It is the code that's outside of functions in Java Script. The code that gets run immediately after the module has been instantiated but that's pretty much it. Once that is all done, then the only way that the Web Assembly starts doing anything is if you actually call the functions that it exports.

[0:45:19.8] JM: These Web Assembly modules that get instantiated in my browser, they're operating within a stack based virtual machine. So, this is a virtual machine that's super minimal. It's super low footprint just like if anybody has taken a course in electrical engineering or just computer architecture where you learn about how the processor instruction sets work. I mean I think people who are curious about that can look into that.

I mean it is pretty interesting, not super relevant if you're a high-level software engineer except to know why this kind of system actually runs quite efficiently even though it sounds kind of lame. It's like you are doing stuff with a register's I think, and you have to do everything in a stack. Are all of these modules, are they self-managing or is there some overseer that's managing all of your different Web Assembly modules that are instantiated in the browser?

[0:46:17.8] LC: Well I'd say that your overseer is actually Java Script. Java Script is the one that at least currently is your using a Java Script API to load the Web Assembly module and then to call any functions on that Web Assembly module. Now that will change probably when we get the ES module support because then you'd be able to have a script type equals module and foo.Wasm and you may not have any Java Script in there.

Currently if you did that then you wouldn't actually have any side effects in the browser. You won't actually be able to see it doing anything, so that would be useless but once you can actually pass dom functions and all that kind of stuff into Web Assembly functions that could then call them, you could see the possibility of having a whole module graft of Web Assembly modules that doesn't require any real Java Script.

[0:47:09.2] JM: Which is pretty incredible to think about. You talked in your post about how Java Script can call Web Assembly modules and then Web Assembly modules can also call out to Java Script. Why would you want to do that? Why would you want to have it? I mean because I can see how I could write a web application where I would want to delegate functionality to Web Assembly modules to do low level operations or to run Photoshop but why would I want Photoshop to be able to call back out to Java Script?

[0:47:43.1] LC: Well currently, you actually do need to call back out to Java Script for a lot of things because Web Assembly doesn't actually have direct dom access. So if you want to make a change to something in the dom, you need to call a function from Java Script that could make that change in the dom for you. I mentioned, we're working on making it so that you can pass built ins from the web platform or different kinds of dom objects into a Web Assembly function.

And then use those directly from Web Assembly. So, in that case, you wouldn't necessarily need to call a Java Script function in order to do that for you but for now, you do have Web Assembly functions calling Java Script functions a lot for that kind of side effect-y kind of stuff.

[0:48:30.8] JM: When do we start to see Web Assembly impacting the average user and affecting the average user's application experience?

[0:48:39.5] LC: Well, when you are talking about the average user, do you mean the average person who is browsing to web pages or the average developer?

[0:48:47.2] JM: Let's give both time wise.

[0:48:48.4] LC: So the average user, you are already seeing it in a lot of applications. I can't remember, Google Earth I know is at least working on using Web Assembly. I can't remember if they're using it but AutoCAD has released their CAD software in the browser. So, a lot of people like I mentioned, we have the source maps library in our developer tools that is using Web Assembly. So, a lot of users are being impacted by the use of Web Assembly without realizing that they are using Web Assembly.

For developers, I think that it is probably going to be fairly similar. A lot of developers are going to end up using Web Assembly without realizing that they are using Web Assembly because the package that they depend on, like the source maps library, is using Web Assembly under the hood.

[0:49:31.0] JM: I should have asked this earlier but are there additional security considerations that Web Assembly introduces to the web developer or I guess to the browser vendors?

[0:49:41.9] LC: You know, Web Assembly interestingly enough is actually interesting in the opposite way. It actually can close security holes and is being considered for cases where you want to run untrusted code but code that you are downloading at random from the internet but don't want to give access to your systems resources. Now people have had concerns that Web Assembly would open up on the attack surface, but it uses the same concept that Java Script does.

You know, it basically uses all of the same security modelling that Java Script does and the reason that Java Script has some pretty – the browser vendors have really worked on making Java Script secure to run is because you are downloading this untrusted code from who knows what kind of person and you don't want that person to be able to take over your system. So that's why for example the linear memory, is not really –

People when they hear that you have direct memory access with Web Assembly, they are concerned that the attacker is going to be able to get access to the global object or something like that but Web Assembly doesn't have access to random parts of the platform. The only thing that Web Assembly has access to is that linear memory which is just an array of lights. So, it is an array buffer which you already have in Java Script and anytime somebody tries to access something else outside of that array buffer, there is an array balance check.

That will throw in an exceptionable trap whenever Web Assembly module tries to access memory outside of its array buffer, this is linear memory and as far as functions you can call, you only could call things that can pass into Web Assembly either as imports or as a function arguments and that means that you can't just reach out and touch random parts of the platform. As a developer who is working with a Web Assembly module.

You are intentionally giving them exactly the access that you want, exactly the functions that you want this module to use.

[0:51:41.2] JM: To talk about one of the broader implications for Web Assembly and I do want to start to zoom out because I know we are running out of time here but the idea of using a wider variety of languages in the browser is very exciting and you wrote that a goal for 2018, one of

your goals, is to make Rust more of a web language. Why is Rust a good compliment to Java Script? How do you see it fitting into the web?

[0:52:07.6] LC: Well I should clarify that that isn't just my goal but it's also the goal of the entire Rust to Web Assembly team and that work is being led by folks like Nick Fitzgerald to work done that source map, that library that I was talking about. Actually, Williams who is working on Wasm pack, Alex Creighton who is working on Wasm bindgen. So a lot of those folks are really driving this work to make Rust more of a web language.

And the reason why Rust is really suited to be a web language is really going to be a very good tool in web developer's tool boxes, is that there are some problems that system programming languages are better suited for. So, there are times when managing your own memory yourself is going to be a lot more performant than allowing the Java Script engine to create objects willy-nilly and just letting the Java Script engine clean them up for you.

But managing memory as a developer is really pretty hard and if you get it wrong, you can open up some pretty big security issues. So, Rust is nice because it was designed in a way that allows you to manage your memory yourself but also guards you from the kinds of memory safety bugs that open up these security issues. So, the reason that this is good for the web and good for web developers is because a lot of web developers don't have the time to really learn about C and C++ proper memory handling.

And Rust gives you a way to get that performance of a systems programming language without giving you the foot gun that programming language like C or C++ might.

[0:53:36.7] JM: I haven't built many web applications, but I imagine that memory leaks can be a less significant issue if you're working with a language like Rust that emphasizes the memory management.

[0:53:46.5] LC: That's true. One thing with Web Assembly is if you have a memory leak, that memory leak will only impact your module because of the fact that you have that kind of sequestered off linear memory but yes, within your module that memory leak will be better with something like Rust.

[0:54:06.7] JM: This is a world that seems like it's so different than what web development looks like today because the early examples of what we're talking about with "Let's get Photoshop in the browser or let's get AutoCAD in the browser," because these are our best projections of the present-day applications and how they would fit into the browser. But undeniably, the coolest applications of Web Assembly are going to be the ones that we are not thinking of today.

But you are thinking of it from a more fundamental perspective, if you get Rust in the browser what do you get? What's the state of tooling around Web Assembly? So, if I am a developer that's not on the cutting edge, my younger brother for example is always tinkering with whatever new thing has made it to the top of hacker news today. You know he is building a flutter app, he is probably doing something in Web Assembly and why not?

And people are like that, but the average developer is probably waiting a little bit for the tooling to develop. So, what's the tooling that you see as barriers between the super cutting-edge developers and the average developer?

[0:55:12.0] LC: Well I think we are getting there as far as tooling that you can produce Web Assembly, a solid Web Assembly program with. Like I said, the Rust team has really been working on this a lot this year. They are going to be working on it through the rest of the year very – going even more heads down on this. The problem that we have, the area that I think needs the most work is in debugging and that is something that is actively being worked on but it's just hard to find a way to represent the debugging information.

There has been some thinking that we can piggyback on source maps and there's been some work around that, but source maps might just not – we may never be able to get all the way there with source maps. So, there is also discussions about should we use something based on this thing called dwarf. I have to say, I am not actually that well-versed in the distinctions for these different debugging tools, but I think that that is the area that's probably going to hold back developers who are a little bit more cautious.

[0:56:12.5] JM: Okay, I know it's the end of our time, but I just want to ask you, what about progressive web applications? Do you think Web Assembly will make progressive web applications more compelling?

[0:56:21.8] LC: I do think so. Because I think that you know, Web Assembly gets you closer so the performance of native code. That's why a lot of people who have developed platform specific mobile applications have done that is because they want that speed. With a PWA, you have the ease of installation of a web app or you know, no installation really that a web app has but if you're using Web Assembly, then you can get close to the performance of that native application still.

It also makes the transition from a native mobile code base to a web code base easier because you can take all of the logic that you have in your native app written in something like Swift and just reuse it and if you compile it to Web Assembly then you can just reuse all of that code that you already have, all you need to do is put a new front end on using web technologies.

New HTML and CSS but that a lot of times, I mentioned this before with the propeller heads example, a lot of times using the HTML and CSS for your user interface is actually better than the toolkits that you're given.

[0:57:21.2] JM: Yeah, look at Electron.

[0:57:22.2] LC: Yeah, exactly. I think that this definitely makes using web technologies a much better choice for these native experiences or these mobile experiences.

[0:57:35.0] JM: I did a show, a couple of shows about Flutter recently and it sounds like Google is really trying to get away from the JVM for their mobile application. I wonder if Flutter and Dart gets them closer to this same world where a progressive web apps could be doable?

[0:57:50.9] LC: I do know that the Dart folks have been talking about Web Assembly and so I could see them possibly ending up using Web Assembly as part of their story as well.

[0:58:02.2] JM: Okay, well Lin, thank you so much for coming back on the show and your posts were super helpful for me understanding Web Assembly. It's taken like three or four Ed vats for me like sitting down and saying, "I'm going to understand what this thing does and I think I got the closest with your article." Thank you so much.

[0:58:20.0] LC: Well thank you. I'm so happy to hear that, thank you so much.

[SPONSOR MESSAGE]

[0:58:24.5] JM: Over the last two years, I spent much of my time building a video product. We had issues with latency, unreliable video playback, codecs. I was amazed that it was so difficult to work with videos. As it turns out, video is complex and figuring out how to optimize the delivery of video is not easy. Especially since there is both mobile and desktop and mobile users might not have as much bandwidth as desktop users.

If you are an engineer working on a product that involves video, you just don't want to think about any of this. I could tell you that from firsthand experience. That's why Mux exists. Check out Mux.com and find out how Mux makes it easy to upload and play back video. Mux make video hosting and streaming simple and today, you can get $50 in free credit by mentioning SE Daily in the signup process.

Even if you aren't working on video right now, if you think you might work with video in the future, Mux is a really useful tool to be aware of. Check out Mux.com and if you're an engineer who is looking for work, you can also apply for a job at Mux.com. On Software Engineering Daily, we've done two shows with Mux and I know that Mux is solving some big difficult problems involving lots of data and large video files.

To find out more, you can go to Mux.com, you can get $50 in free credit by mentioning SE Daily and you can apply for a job, if you're interested in working on some of these large challenges. Thanks to Mux for being sponsor of Software Engineering Daily.

[END]