EPISODE 655

[INTRODUCTION]

[0:00:00.3] JM: WebAssembly allows developers to run any language in a sandboxed memory-controlled module that can be called via well-defined semantics. That might sound confusing, but we're going to get into it in this episode, and we discussed it in much detail in previous episodes with Lin Clark and Steve Klabnik from Mozilla, as well as an episode a while ago with Brendan Ike.

WebAssembly is changing application architectures both inside and outside the browser. So it's worth paying attention to WebAssembly is being adopted by all of the major browser vendors, including Google.

Today's guests are Thomas Nattestad and Ben Smith from Google. Thomas is the PM for V8, which is the JavaScript engine of Chrome. He's also PM for WebAssembly, storage and games on the web; and Ben is a software engineer on the Chrome team. Ben and Thomas join me to talk about the state of WebAssembly, what the different browser manufacturers are doing and some cool uses for WebAssembly, from games, to CDNs, to cryptocurrency infrastructure.

As in the previous episodes, there was discussion of WebAssembly security and memory benefits from being a bounded context, flexible, modular place to run code, and I am excited about WebAssembly. I can start to see so many applications for it, and there're just a lot of open questions as to how it's going to roll out and how it's going to impact people, but I plan to cover that in as much detail as possible, and you can always send me suggestions for episodes if you've seen some cool use cases or developments around WebAssembly in the wild.

I also want to mention, I'm hiring for a new company that I'm starting, and I can't talk about the product quite yet, but I'm excited about it and I'm looking for an engineer in the Bay Area who has a significant experience in React.js and also some sort of cloud, whether that's Heroku, Firebase, Google Cloud, AWS. You can email me at jeff@softwareengineeringdaily.com. You can check out the job posting at softwareengineeringto.com/jobs, and I would love to hear from you.

[SPONSOR MESSAGE]

[00:02:20] JM: At Software Engineering Daily, we have web app, have an iOS app, and android app, and a backend that serves all of these frontends. Our code has a lot of surface area and we need visibility into problems that occur across all of these different surfaces. When a user's mobile app crashes while playing a podcast or reading an article, Airbrake alerts us in real time and gives us the diagnostics that let us identify and fix the problem in minutes instead of hours.

Check out airbrake.io/sedaily to start monitoring your apps free for 30 days. Set up takes only a few minutes. There's no complicated configuration needed, and Airbrake integrates with all of your communication tools, from Slack, to GitHub, to Jira, and it enhances your current workflow rather than disrupting it. You can try out Airbrake today airbrake.io/sedaily.

If you want to monitor and get visibility into the problems that may be occurring across your application, check out Airbrake, at airbrake.io/sedaily. Thank you to Airbrake.

[INTERVIEW]

[00:03:40] JM: Ben Smith and Thomas Nattestad, welcome to Software Engineering Daily . It's great to have you.

[00:03:45] BS: Great to be here.

[00:03:46] ND: Thank you, Jeff.

[00:03:47] JM: Today we're talking about WebAssembly. We've done a couple shows about WebAssembly recently, and on those shows, people have defined WebAssembly in a number of different ways. It's a low-level compilation target that allows any language to run in the browser if they compile down to WebAssembly. It's also described as a virtual machine for running WebAssembly modules, which are these sandbox execution runtimes. How do each of you describe WebAssembly?

[00:04:18] BS: Sometimes we talk about it as being an instruction set architectures. So similar to x86, or x64, or arm. It's a target that's like a machine. So when people say virtual machine, sometimes you think Java or maybe the.net runtime, but we think of WebAssembly closer to something like what you would actually run on an actual machine. Now, there're a lot of differences, but I think that's kind of like how I would think of WebAssembly.

[00:04:46] TN: I completely agree with that assessment, and just from the perspective of a developer and how they can kind of perceive it, I think the most useful framing that I've seen developers describe is a byte code format that I can compile something into and then kind of just ship it to have it run in the browser. Especially when you look at some of the languages that it supports right now, like C++ and Rust, what it really does is give the developer the ability to write in those languages, and of course port existing code written in those languages as well.

[00:05:14] JM: JavaScript is a Turing complete language. It's got ton of community support. In what ways is a WebAssembly complementary to what we have today with JavaScript in the browser? Ben, maybe you can answer that.

[00:05:29] BS: Yes, so JavaScript is great actually. It's an excellent language to use, but there are some places where it's maybe not as appropriate. So actually what we found was there are a lot of people who want to bring applications to the web that just didn't really work in JavaScript. So if you were compiling something like C++ to JavaScript and use some script and do that, you would end up with just reams and reams of JavaScript to do it.

So as a result, you end up actually taxing the JavaScript engine a lot more than you should be, because the JavaScript engine doesn't really expect code to be written that way. So WebAssembly actually feels that niche very, very well, and its original purpose was to provide a compilation target for low-level execution that isn't written by a human.

[00:06:21] JM: This paradigm shift to having WebAssembly in the browser as a complement to JavaScript's execution, this requires some refactoring of the platforms that process JavaScript, where we want to have WebAssembly also existing.

Thomas, what adjustments do we need to make to a web browser in order to get it to execute WebAssembly?

[00:06:47] TN: Yeah, that's a great question. Within Chrome, the WebAssembly team and the V8/JavaScript team are actually the exact same team. So this is really a very joint collaboration, and the original ideas for WebAssembly actually came out of engineers on the VA team. So they're the ones that are the real experts on how this stuff actually works, and Ben can probably talk about the very nitty-gritty details of what's changing. But at the high level, V8 is the engine that’s still doing the interpretation of the WebAssembly code as well as the JavaScript code.

We just recently launched a new, what's called a baseline compiler. It’s called Liftoff, and that's a compiler that is entirely responsible for just WebAssembly. So the team, the V8 team had to go and write that.

[00:07:33] BS: Yeah, I can add a little bit more to that. So the idea is basically that you already had in every high performance JavaScript engine a low-level compiler that would basically take code and then compile it down to native machine code and be able to execute it very quickly.

So the original sort of conception WebAssembly was that it would actually be relatively simple for people who already have a JavaScript engine to add a WebAssembly engine, because we would sort of hijack the same high performance compiler. It turns out that we ended up needing other compilers too, such as the one that Thomas has described, the baseline compiler, and that's to make it so that you can execute code very quickly, like you can compile it very quickly. But I think the original conception of sort of integrating directly with the JavaScript compiler is still accurate, and basically all the WebAssembly engines do that.

[00:08:25] JM: In talking to Steve Klabnik, from Mozilla, I had him on the show yesterday, and he gave one of the earliest examples of WebAssembly being used in production. He gave the example of Dropbox. When you upload a file to Dropbox, there are some WebAssembly that runs, I suppose, on the browsers that do support WebAssembly today. It compresses your file. So it gives you a client-side compression. I guess it's useful to have that compression algorithm written in Rust, or perhaps C++, whichever it's written in.

Can you give a few other examples of where we will see WebAssembly deployed early on in the early days?

[00:09:11] BS: So the Dropbox case is definitely one great example of where you can take advantage of the extremely fast performance of WebAssembly to integrate into kind of an existing app or functionality.

One of the things that I'm especially excited about WebAssembly enabling is, first of all, being able to import large existing applications. The biggest example that we've seen of this so far is probably AutoCAD, which took their 35-year-old code base and compiled it to WebAssembly to enable all of their web users to access a ton of the same functionality that you previously could only get through the native app. So that’s one that we've already seen.

In addition to that, SketchUp, which is a 3D modeling software, did the exact same thing. They were able to port their native application to the web, which is really exciting. In additions that, you also see some other large applications, such as Figma, which is a design tool for the web. Really heavily lean into WebAssembly and use it for its highly increased performance over something like asm.js.

[00:10:05] JM: This is an illustration of two almost polar opposite types of applications, because with the case of Dropbox needing a little compression module in the browser, you have very modular component of an application. Then in the case of something like AutoCAD, you have a giant old monolithic application that's probably written in C++ and was written in a time where people couldn't even envision the focus on web and the emphasis on browser applications and applications that feel like they are browser applications. So you had these two almost polar opposite use cases.

So, can you talk about how WebAssembly is used in those two different types of use cases and why it's useful for both of those types of use cases?

[00:10:55] TN: Yeah, absolutely. So you're exactly right, that the Dropbox use case is very modular and it's as a library, and I think this is one of the spaces that is going to grow the very most when it comes to WebAssembly and where will seal a lot of really exciting use cases where you can take a small library that was either written already, or that you've written to do some small task and integrate it into your existing standard.

Figma, the design company that I mentioned earlier, is an interesting illustration where they do a little bit of both. So they take some libraries and they modularly integrate that into their otherwise JS components, but then they also have large areas of their application where all of the logic is run entirely throughout WebAssembly. So I see it more as a gradient with two ends than kind of a just two different options.

[00:11:39] BS: It's interesting actually, because with a lot of like, say, JavaScript frameworks versus libraries, right? This is a conversation that people have a lot. Like, does the library sort of integrate nicely into your already existing application, or does it sort of take over and want to control everything I think one of the things that's pretty exciting about WebAssembly is that it can actually fill both roles. Because it's so sort of sandboxed and modular, it allows you to sort of say, "Well, I want to have everything in this, or maybe just a little piece of it."

[00:12:07] TN: Yeah, it's very well-said, Bed. Speaking of JavaScript frameworks, I'll mention that the ember framework is currently looking at, and has already started making commits to adopt WebAssembly to do some of their more computationally-intensive aspects. So that's one example of actually integrating WebAssembly into an already existing library.

[00:12:26] JM: Okay, we've got three different case studies here. We've got Dropbox's compression algorithm. When you upload a file, it gets compressed using some WebAssembly module, or modules. We've got the example of AutoCAD, a large 35-year-old monolithic application that's probably written in C++ that was ported to WebAssembly and can now be used in the browser. We've got the example of Ember potentially wanting to use WebAssembly for some features.

So maybe we could run through these three as some case studies in helping people understand how a module or an application gets turned into a set of WebAssembly modules or applications. So let's start with the Dropbox compression algorithm. So I assume there's somebody at Dropbox that wrote some Rust code for compressing a file, and they want to put that Rust compressor to WebAssembly so that the client on their browser can compress the file before it gets sent over the wire. You obviously want it to be in the browser so you wouldn't want to send the file to Dropbox and have them compress it. You would want to save on the bandwidth, so compress it beforehand.

So, talk about what is the compilation path? By the way, we've done a show where we really dove in deep into this whole compilation path and the intermediate representation and LLVM stuff and so on. But maybe you could just walk through it, because this Rust or this compression algorithm is a nice first example.

[00:13:59] BS: With Rust, it's a bit different than with C++. With C++, you would probably use some script in, which is the toolchain that we usually suggest, which is essentially wraparound LLVM, allows you to compile C++ and then integrates a lot of functionality that you would want if you're compiling to the web.

With Rust, it's a bit different because they control their toolchain little bit more closely. So what you would end up doing is using the – Well, I mean, if you had Steve on yesterday, he can explain way better than I can. But basically you would use the Rust toolchain, pull that down, and then that would give you the toolchain to compile.

Yeah, you would have a compression library that you'd compile, and what you'd get out of the end is a WebAssembly. What that module would do is it would have a collection of imports and a collection of exports, and the idea is that a module by itself can't really do much of anything. So you have to sort of tell it what – You have to sort of give it the imports to say, "Here are some things you can ask me, the embedder, about," and then it gives you some exports saying, "Here's some stuff that I can do." Then when you call those exported functions, it will do what you want.

So, for example, in the compression case, you would have an expert function that maybe is called compress. When you call that, the module might call back to you and say, "Well, give me some data," or maybe pass it down. There's a number of different ways you can actually configure it, but that would be basically the idea.

Then you'd take that module and integrate it and call it through JavaScript. So there's a JavaScript API for WebAssembly modules, and you would use that to load the module into a JavaScript object and then be able to access it directly.

[00:15:38] JM: So the import/export constraints, is that how you draw a line around – That sounds like a sandboxing limitation, where it defines quite succinctly and rigidly what this module has permissions to do.

[00:15:54] TN: Yeah, exactly. Yup.

[00:15:56] JM: Okay. I think we could also talk about the example of the AutoCAD. So if you had a large application that you wanted to migrate to WebAssembly in order to run it in the browser, what would that procedure look like?

[00:16:09] TN: So it's an interesting question when you talk about a large existing application that was written without the web in mind, and there are various things that you, as an application developers, do have to do in order to move that application to the web. People always ask how magical the conversion processes is to – So to open that black box a little bit, one of the primary things that you as an application developer have to do is figure out the parts of your application that can be modularly converted into WebAssembly and which parts won't.

Some of the primary components that you won't be able to convert directly are any operating system, system interfaces. So if you do any Windows-based window management, then that's something that you have to rip out and replace with some new UI. But all of the kind of core code components that you have, you can convert those directly into modules and then hook those modules up in whatever makes sense for your architecture. That's exactly what AutoCAD did. They were able to take – I think it was something like 95 or 98% of their total code base, and then they just replace the UI components with a little bit of stuff locked up in React and then connected the components together to get their application working.

One of the good rules of thumb that I've heard is that if you have a Windows-based large application and you want to bring it to the web view of WebAssembly, it'll probably be about as much work as bringing your application to Linux.

[00:17:28] BS: Yeah, it’s very similar. I think the other thing I would add about that is that if you already have a cross-platform application, something like you have a Windows version and a Mac version, then you probably already have some level of abstraction over the UI and all the sort of OS components that will be separate, and those will be the same sort of layers that you will need to separate for bringing your application to the web via WebAssembly as well.

[00:17:54] JM: Cool. Are there any gotchas that have been encountered by these other people – Because I think this kind of procedure has been done by a number of different companies. I think Lin was also telling me that the reason team, which is like a digital audio workstation, I think they've also gone through this procedure. What kind of gotchas have these large applications that have been ported to the browser, what have they encountered? What has not worked? What's been difficult?

[00:18:19] TN: I think the only real thing that we've heard is that some of the largest applications hit up against the 2 gigabyte memory limit, but luckily we are planning to increase that to four. But outside of that, I don't think we've heard too many negative things. Oh, one thing might be the state of debugging right now is definitely something that we could improve on and that we’re currently working on improving. But those are the only two that come to my mind.

[00:18:41] BS: Some of the other things are limitations in web APIs. So because the web doesn't allow you to access all of the functionality of your computer deliberately, right? Because you don't want, say, a webpage to start sending network packets for you, for example. There are things that are not actually – It is not able to do.

So actually one of the biggest ones is probably file system access. For a lot of applications, that's okay, because they want to – If they’re running on the web, they want to have some kind of cloud storage, but it might be very different than what you are currently implementing in your native application, because people don't usually think of cloud storage there.

But I think one of the things that’s kind of cool about WebAssembly in that regard is that it can take a native application that may be was limited in that way, and because it's now converted into a web application, there are certain things that you just sort of take for granted. For example, the fact that you do have a server that you can use as to provide any pieces of functionality there or the fact that you do get to just update your server and then provide a new application to the client directly.

[SPONSOR MESSAGE]

[00:19:55] JM: Leap.ai matches you with high quality career opportunities. You are more than just your skills and a job description and a resume. These things can't fully capture who you are. Leap.ai looks beyond these details to attempt to match you with just the right opportunities. You can see it for yourself at leap.ai/sedaily.

Searching for a job is frustrating, and Leap tries to reduce the job search from an endless amount of hours, days, weeks to as little as 30 seconds trying to get you match to a job instantly by signing up based on your interests, your skills and your passions.

Leap works with top companies in the Bay Area, unicorns and baby unicorns, just to name a few; Zoom, Uber, Pony.ai, Cloudera, Malwarebytes and Evernote. With Leap, you are guaranteed high response and interview rates. You can save time by getting direct referrals and guaranteed interviews through Leap.ai. Get matched to jobs based on your interests, your skills and your passions instantly when you sign up.

Try it today by going to leap.ai/sedaily. You would also support Software Engineering Daily while looking for a job. Go to leap.ai/sedaily.

Thank you to Leap.ai.

[INTERVIEW CONTINUED]

[00:21:34] JM: So for applications today, applications like AutoCAD, for example. So this is a 3D rendering engine, and this is the type of application that people buy that super high-end that Apple computer, where like it’s a giant cylinder that kind of looks like a trashcan. It's like $6,000, and it's really expensive, it's really performant, because you're doing some kind of 3D rendering or movie production on your laptop, and it would be great if even if you are using the

WebAssembly version, you could access all the horsepower of that computer. But my understanding is it that's probably not doable today, and I think to your point, someday, that kind of stuff will be accessible via server, or maybe not, or maybe you'll always need a giant trashcan computer for some particular kinds of applications. Maybe it's a VR, or AR or something like that.

But if we assume that we do need some kind of heavy client device and we want to be able to take advantage of all of the resources on that heavy client device, Thomas, maybe you could talk about overcoming the memory constraints that exist in the browser today and kind of the memory-constraint model of these WebAssembly modules.

[00:22:50] TN: Yeah, absolutely. So I think, first of all, I want to make it very clear that our goal with WebAssembly is exactly as you say to expose all of that underlying horsepower of any machine that you're on and to make sure that any web application can take as maximal advantage of everything that your CPU/computer has to offer. Right now, today you have a huge boon on JavaScript, but we're also adding a lot of things to let you get more control over all of the resources that your computer have.

One of the big ones is threads, of course, which gives you fast access to multiple cores on your CPU. Then, additionally, also SIMD, single instruction multiple data, to really optimize for that, like very high mathematically intensive use cases. So like our overall ethos with WebAssembly is exactly as you describe, is getting all that horsepower.

Specific to the memory limits, we know that we can go fairly easily from the 2 gigabytes to 4 gigabytes, which should be sufficient for the vast majority of use cases, but I think there are options in the future to go even beyond that if we find that it's needed. It's just a matter of prioritizing, getting the fastest bang for your buck.

[00:23:58] JM: Sure. Ben, anything you want to add?

[00:24:00] BS: Yeah. So about the last point, the reason there's a 4 gigabyte limit is that currently, Wasm, is what we call Wasm32, which just means it's a 32-bit architecture essentially. That actually has a lot of benefits in the sense that if you're running a 32-bit Wasm module inside a 64-bit CPU, which is what most of us have these days, you can actually use tricks to make it run fast and to perform the sandboxing that's necessary.

So the option to sort of breakthrough that four gigabyte limit would be to add a Wasm64. We have sort of thought about that and how that might work, but we haven't actually put any serious effort into it yet. But I think it's an interesting avenue to explore in the future definitely.

[00:24:44] JM: All right. The third example that you mentioned was Ember.js. Ember is a web application framework for doing – Making JavaScript-based applications, the higher-level JavaScript framework. How would Ember use WebAssembly?

[00:25:02] TN: So as I understand it, Ember right now is looking at using WebAssembly to do some of the [inaudible 00:25:06], but they currently do for their rendering. So that's one component of the Ember framework that is kind of bottlenecked on performance. So they should be able to moderately take that component out and then replace it with a WebAssembly, one, to just get faster improvement. That means that they'll have to figure out what language they want to use, like C++ or Rust, write the actual code in that language and then compile it to WebAssembly and integrate it through JS API calls to their existing framework.

[00:25:33] BS: The cool thing there though is that means that you, as an Ember user, if you did have this, would sort of not have to even know that you were using WebAssembly. It wouldn't really even matter to you really. You would use the same JavaScript API, or you would use the same APIs that you normally do, and it's just sort of running WebAssembly module in the background, and it just means that your diffing is faster.

[00:25:57] JM: And that's, of course, the beauty of a lot of this WebAssembly potential, is that the power of WebAssembly will be hidden in the packages that we import. It won't be such that you need to rewrite your entire application to support WebAssembly. It'll be that maybe you, instead of doing NPM install compression algorithm V1, you install compression algorithm V2 featuring WebAssembly, and you just get to take advantage of that.

[00:26:24] TN: That's exactly right, and I think that the two components there is, one, as we already described, existing frameworks being able to swap out the parts that are performance bottlenecks with WebAssembly, and then also being able to swap out entire libraries. If there's a library that does something like physics, or artificial intelligence, or visual analysis, then you can just take those entirely, compile something down to WebAssembly and then just wrap it in a JS API. Like you said, just swap it in. It could even be the exact same APIs, a previous version, but now it's just running hopefully several times faster.

[00:26:56] BS: So I talked to some of the Mozillians recently about this, who work on Rust and WebAssembly, and they were describing this as being a sort of an interesting process where you might want to try and find where that proper dividing line is. So it might not being necessarily at the library level, but it maybe it’s at sort of like a little bit larger than the library level. It's not exactly clear. I think the future we want to get to is where it really could just be individual libraries. But at least with the sort of plans that I've heard recently, there is going to be some amount sort of decision on the part of the application developer.

But I think that actually provides you with a lot of benefits. It means that you sort of bring a bunch of these components together and then its own little component for your application that's written in WebAssembly.

[00:27:43] JM: I'd like to revisit some of the lower level aspects of WebAssembly's runtime. Again, we went through this in detail with Lin, but can you talk about – I'm not sure who this is a better question for, but maybe Ben. Can you talk about the role of the LLVM in this process and the role of intermediate representations in making this all work in the browser?

[00:28:11] BS: So LLVM is the suite of tools actually. It used to stand for something, but now it just stands for LLVM, and the ideas that LLVM itself takes an intermediate representation, an IR and converts it to native code, perhaps that’s x86, perhaps that’s x64.

So on top of that, there's a compiler clang. So what clang does is it converts C++ code and C code into LLVM IR, which then LLVM converts to native code. So what we've done with WebAssembly is we have our own backend, which is basically converts from LLVM IR to WebAssembly. You could call it IR, WebAssembly bytecode. So that's sort of the role of LLVM in this case, is to provide that sort of pieces that are necessary for converting from IR to WebAssembly.

The benefit there is that it means that any other frontend languages can use the same backend to WebAssembly. So Rust uses the same backend to do its conversions as C++ does, and other languages could do the same, D, For example, or perhaps even languages like Go. Although with some languages, it actually ends up being a little bit more complicated. If your language maps very well to something like the same type of thing that C++ and Rust do, where it's kind of a low-level systems like language, then it maps much more cleanly.

If your language is more like C# or Java, you might not use LLVM in that case, because you have your own toolchain and you have your own sort of compiler infrastructure that you would use instead. And so they've taken a different attack.

[00:29:56] JM: I think I see. So LLVM is – Correct me if I'm wrong, is a way of writing interpreters. It's not – Or is it may have been referred to as an interpreter itself in the past. Now it's more like you can use it to interpret things from one language into another, like a higher level language into assembly code.

[00:30:18] BS: Yeah. I mean, we usually use the term compiler. But yeah, it's the same sort of idea. It's just translating from one representation to another, and the question is whether that representation is something like a programming language, or it's something like the internal representation that LLVM uses itself.

[00:30:35] JM: So in this case, we would take the C or C++ or Rust code. It would be compiled down to the LLVM intermediate representation, and then something would compile that intermediate representation down to the WebAssembly format?

[00:30:51] BS: Exactly, yeah. That's the WebAssembly backend. So essentially – I mean, I don't how in-depth we're going to get here, but there's multiple level levels of – Well, there's just one IR, but there's actually different formats that it takes sort of at it goes through the process. But the whole idea is that you can sort of write the IR in, say, more of a normal way, more like a programmer would do, and then the whole point of the sort of LLVM infrastructure is that it can take that format and then it can do optimizations on it and improve the code and produce better output than you would be able to – You could do it by hand, but you wouldn't want to is sort of the question. So that's kind of what LLVM is doing for you there. It’s providing that level of optimization over the code that you write.

[00:31:36] JM: So in managed runtimes, I think you have this hot code path runtime situation where your virtual machine can look at the code that's executing and it can look at, for example, the code that's executing in loops. Since it’s executing in loops, it can optimize that code to run faster. But I think in these modules, that's not exactly what we want, because we have a sandboxed amount of memory. Is there some degree of management that's occurring over these modules? Once they are fully compiled into the WebAssembly format, i there a management layer that's doing things like hot code path detection?

[00:32:24] BS: You could do that, but WebAssembly is actually specifically designed so that you don't have to do that. So one of the things that makes writing a JavaScript VM so difficult, it's like you say, it is a dynamic language. So you have to – If you converted JavaScript directly into machine code, and there actually are JavaScript engines that do this, it wouldn’t run as fast as a lot of the engines that we have today.

The reason that it runs fast is that it does exactly what you're saying. It sort of analyzes the code as it's running. Tries to determine things that are currently the slowest or the things that are the hottest, and that a trust optimize that.

But what WebAssembly has is it has full information about the types of the system and all the layouts of – And all the structures. So like C++, or very similar to C++, it can actually do much more ahead of time compilation. So what happens is when you write your code in C++ and you compile it to WebAssembly, it actually will do a lot of the same optimizations that your C++ code would do when it's compiling to x64 or to ARM.

The difference is that there's a separate compilation process that happens when you take WebAssembly and you convert it to the actual native code, and that happens on the client side in the browser. So the WebAssembly engine itself can sort of choose what it wants to do. It could just take the WebAssembly code and convert it to assembly code directly. Since it's already been optimized by something like LLVM, that's maybe not such a bad choice actually, but you can also choose to take that WebAssembly code and then perform more optimization still on it, and there might still be optimizations you can do, because, let's say, the generated WebAssembly, there was already some optimizations done, but there are certain optimizations you could only do once you have a native architecture, like x64. So that's kind of what is happening there, is there’s actually two compilation steps that happen, and two potential points for optimization as well.

[00:34:23] JM: In terms of the security and the sandboxing of WebAssembly, can you talk about – Maybe Thomas can talk to this as well, the standards that you need to impose around whatever that sandbox is and developing that API for the modules to make sure that they have enough privileges, but they are also highly constrained, because web is such an insecure place to play in.

[00:34:54] TN: Yeah, I can speak to this. So when I think about security and API access of WebAssembly, there're kind of two components. There’s, first of all, the kind of computational sandboxing, which makes sure that you can't access any part of the stack that you aren't supposed to, and WebAssembly has very strong kind of boundaries there. We do what’s called bounce checking on every operation so that you can't snoop on any parts of the application that your WebAssembly code isn't supposed to be snooping on.

In addition to that, there's something kind of very separate, which is what APIs that they have access to. These can be things like WebGL, or kind of other APIs, and all of that is currently mediated through JavaScript. So WebAssembly, as I think we’ve mentioned really, it doesn't get access to anything today that JavaScript doesn't have access to, except faster compilation.

So, right now, WebAssembly funks out to JavaScript when it wants to make a web API call, and then you can return back into the WebAssembly with the result. In the future, we’re hoping to be able to cut out the JavaScript middleman through something called host bindings. But even then, you will still only have access to all the things that you still had access to through JS. So in that sense, WebAssembly in no way breaks any of the existing security model around the web. Ben, maybe you can speak to a little bit of some of the like module import/export rules.

[00:36:07] BS: There's not much to say there except for that you really can only import and export functions and globals, and – Well, actually, you can import WebAssembly memories and

WebAssembly tables as well. But sort of functions and global source, sort of the most natural thing to think about, and the idea there is just that, like, you can take a JavaScript function and you can pass it as an import to WebAssembly module. Then whenever the WebAssembly module wants to call it will just call out through that function, and then there's a translation layer that basically pops that over back to JavaScript. So that's sort of the way that you interact.

[00:36:44] JM: And these different modules, the different WebAssembly modules, I want to understand a little bit better how they are interacting with each other and also how they are getting created and destroyed. Are the different WebAssembly modules sharing memory with each other or are they copying values and passing them to each other? Are these different modules being managed by some overseer in the browser that can know about the different modules that you're running? Tell me a little bit more about that overview of the different modules.

[00:37:19] BS: Yeah, I'll talk about this a little bit. So there is no really overseer. I mean, you might say that the browser engine itself sort of knows about all the different WebAssembly modules and knows perhaps how they want to be interacting. So if you did have two WebAssembly modules that, one, exported the function and the other imported that function, the sort of browser engine knows that it can call straight through from one function to another. In general, it's sort of the responsibility of the JavaScript application to provide that sort of glue. Now, that's currently the case.

What we’re trying to work on is better, ES6 module integration. There's a proposal for that, and the idea there is that then you could actually have something like ES6 module, which is actually a WebAssembly module. Those – They both use the term module, and so things that they're similar, and they are similar in some ways. But there are some differences as well.

But then the goal there would be that you would actually have a module that you can import just like you would import a JavaScript module, and it might be more transparent. As it is currently, the way that data is shared between modules and even between a WebAssembly module and JavaScript is through what we call WebAssembly memory, which is we also call linear memory, the ideas that it's just a linear collection of bites.

So you might almost think of it as being the same way that you take some data and you serialize it to a file when you're writing at your file system state. It's very similar. You would have to have a format that you write out that both sides know about. So the WebAssembly module writes out to the memory and says, “Here's my data,” and then the JavaScript side says, “Oh, I know how to read that data,” and there are tools that actually make this simple so you don't have to do it yourself, but that's basically the idea.

[SPONSOR MESSAGE]

[00:39:14] JM: If you’ve ever been a tech news news junkie, you've probably checked out Techmeme. Techmeme.com is a great website for finding out what's going on every day in the tech world. It's a highly-curated set of content around technology news, business news, and now they have a podcast, The Techmeme Ride Home. It's a short 15 to 20-minute podcast the you can hear on-the-go Monday through Friday.

It might be a good complement to Software Engineering Daily and it’s certainly more concise than Software Engineering Daily. You can tune into techmeme Ride Home to find the latest headlines, context and conversation around what is happening in the world of tech. You get everything from the top news stories, to hearing about the top posts, conversations and tweets, because who has time to get all of these stuff from the top of the funnel? It's great to get it condensed down into a 15 to 20-minute podcast. The podcast is hosted by Brian McCullough, who has been running the Internet History podcast for several years. That show is awesome as well.

By the way, I have listened to a lot of Brian McCullough on the Internet History podcast and I really enjoy his ability to give perspective, both historical and contemporary on what's going on across the internet.

So check out the Techmeme Ride Home podcast and subscribe today if you like it. You can give your eyes a rest from the many news and information aggregation sites across the web and get it in podcast form. Check out the Techmeme Ride Home, and I hope you like it.

[INTERVIEW CONTINUED]

[00:41:11] JM: Let's zoom out a little bit. I think we've given people enough of the low-level guts of WebAssembly for now. The use cases for WebAssembly outside of the browser, it's easy for people to understand how this is useful inside the browser. But I think the vision for WebAssembly is much bigger. I think the vision is that WebAssembly is going to be permeating your whole experience as a computer user. Your experience navigating your file system, for example, will feel a little bit more interconnected with the web itself instead of feeling like this very just client society thing.

Am I understanding that correctly? Can you tell me more about what is going to be the sensation of the computer user when WebAssembly is permeating more of our application surface area?

[00:42:02] TN: So when we think about WebAssembly and not web contexts, we often times have seen interesting use cases on the server and through blockchain. So one of the interesting things that have been done on the server are things like CDNs, like CloudFlare, has started adopting WebAssembly and they've adopted it because they're interested in its security properties. Meaning that they can let users write WebAssembly or a language that compiles WebAssembly and then have guarantees that they can run that securely in sandbox on their server to do like conditional CDN hosting, for example.

The second example is blockchain, which, again, values it for that same kind of security property. An interesting tidbit is that in the last several months, we've lost two of the main team members on WebAssembly to Blockchain startups. So certainly there are people out there who are excited about this use case for WebAssembly, as are we. In terms of having it run on kind of client machines, I don't think that's a use case that we've really seen so far. It's not like you get any additional ability to interface with the browser by running WebAssembly inside of, say, a native application any more so than you could with any other language.

[00:43:04] BS: I'll add a little bit to that actually. So WebAssembly has a lot of really interesting properties that make it very attractive to people. Actually, it sort of fills a sweet spot that hasn't really been covered by any other format as far as I can tell. So a couple of those things are, it's very, very highly specified and it's almost completely deterministic. There are few corner cases that are nondeterministic. So that's actually one of the biggest things for blockchain people, is that they need a very deterministic runtime and they had their own VM, but it was very convenient to reuse the WebAssembly VM, because we were already doing a lot of the work on the tool chain support and it's and it's a format that's specified. So that's one of the other really big things for them, is just that they can say, “Oh, this already exists, and we can just use it.” So they’re very, very excited about that.

Another thing that's really great about WebAssembly is that it's low-level. So it’s a little less opinionated than other formats. So in the past you'd have a format like, say, Java or.net, which provides a lot of the same sort of ideas, but it's more opinionated and that it has its own object model. Currently WebAssembly doesn't have that. So what that means is that you can sort of use it for more things than you might use Java for. So in the case of like taking C++ code, you could compile that to a Java module, but people mostly don't, because it's not it's not a great match, whereas WebAssembly was specifically designed for that.

So another thing that's really pretty valuable about WebAssembly is that it's a portable executable format. So in the same way as Java or .net, you can use it as sort of a way to say, “Well, write some code and hand it off to somebody and it can run on any architecture. So that actually gives people who are running an OS or some kind of platform a lot of power, because it means that they can, say, “Well, right now our platform only runs on, say, MIPS. But if we want it to run on x64, we don't actually have to change anything, because we already have this portable executable format. So we actually have started to see people using it for that purpose, using it as sort of a format that you can just sort of handoff and just assume that it will work and it will run fast too.

[00:45:22] JM: As a sandbox, why is WebAssembly preferable to Docker, for example?

[00:45:31] BS: Docker is actually really, really cool. Obviously, it allows you to take something like an application that's written for Linux and provide sort of all the infrastructure around it that's necessary so that you don't really have to know what the application needs.

So in a way you might say that it's very similar to WebAssembly. The biggest difference is that WebAssembly is much, much, much lower level than that. I guess sort of it sandboxes is more on the level of a process sandbox, whereas Docker is more on the level of being like an operating system sandbox.

One of the big benefits there is you can take a WebAssembly module and you could run 10,000 of them probably, and I think that would be fine. I think you maybe do the same with Docker. I don't really know enough about the specifics there, but you might end up being hitting limits. Whereas we already see people run WebAssembly modules. You could run multiple WebAssembly modules in one process, and because you have the sandbox, you don't really have to worry about crosstalk as long as there's mutual trust between those in the process. So there're a lot of benefits there.

[00:46:38] JM: That is cool, and I need to do blockchain WebAssembly show. Although I hesitate to say that, because whenever I do like, “Hey, email me with your blockchain related ideas.” I get a bunch of crazy ICO companies. But I'm sure there're some legit ones that would be more than welcome to talk about WebAssembly.

So in terms of developments at Google, Ben, so you're in Chrome team. What does that integration – From the from the point of view of a browser manufacturer, and then, Thomas, I know you can talk to this on kind of a diplomatic level and an industry-wide level as well. Ben, maybe you could give me your perspective on how support for WebAssembly is being integrated into Chrome and your outlook for how that will develop in the future.

[00:47:30] BS: Yeah, I can talk little bit about this. Although I think Thomas actually is a bit better person. Mostly the ideas that – I mean, so WebAssembly is really exciting for a lot of people inside of the Chrome team. I think one of the things that's very cool about it is that it sort of opens the door to a lot of new applications like we were discussing before.

So I guess the question was mostly about sort of integration and how we think about it inside Google. I would say – I don’t know. Maybe Thomas can fill this better actually.

[00:47:58] TN: If there's interest in kind of peeking behind the curtain of how these teams are organized, I’m more than happy to give that kind of detail.

[00:48:03] JM: Yeah, do it.

[00:48:04] TN: Yeah. So I'm the product manager of the V8, which is also consists of the WebAssembly team, and Ben is one of the co-chairs of the community group for the WebAssembly CG group, which the cross-browser group.

Within Chrome, you have the full list of everybody who's working on Chrome, and then within that you have what's called the web platform, which is the team that's responsible for all of the exciting underlying technologies, like JavaScript, CSS, web APIs, etc. Within that, you have the V8 team, which is led by someone named [inaudible 00:48:36], who is fantastic. Within the V8 team you have the team that's responsible for some of the tooling. You have a small team that's responsible for integration with Node.js, which we've been investing in a lot more recently. Then you have WebAssembly, which is a sizable portion of the overall V8 team, and that's team that Ben sits on.

[00:48:55] JM: And tell me more about the cross-browser cross-company discussions and how consensus it has reached on stuff. Do different companies focus on different aspects of this? Just tell me more about that diplomacy.

[00:49:09] TN: Yeah, I can talk about this little bit. So we actually are a W3 group. So we have a working group and a community group, and the differences there are mostly technical. So I won't go into them too much. But the basic idea is that we have a community group, which anybody can join, and I encourage people to join if they're interested in WebAssembly, and the ideas that that's sort of where we do our technical steering.

So currently it has a lot of people from the web browsers. So we have representatives from Google, and Mozilla, and representatives working on JavaScript Core, which is the WebKit toolkit, and also Microsoft and many others, blockchain people. The way the process works is that you basically – Well, we use GitHub issues and we use – So you basically file a ticket and you add a little item to the agenda, and then we meet every two weeks and we discuss basically interesting topics, things that we want to do to try and move WebAssembly forward and just make it better.

So the consensus process is usually just that we discuss the topic and then we have a vote; strongly favor; in favor; neutral; opposed and strongly opposed. We use that to help sort of drive the direction. Then when the community group is done with that, then sort of has an idea about a technical idea. Then we move that over to the working group and then we try and standardize it.

[00:50:31] JM: I'd like to have a little bit of speculation on this show, because web assembly is something that's fun to speculate about, because it's almost like the blockchain stuff as well. You know it's going to impact our world, but it's not quite clear how and to what degree it will.

I mean, look at taking application – So I was thinking like Gmail, or Google Docs, or Hangouts. I can imagine all of these Google applications that I access is in the browser and all of these ways that they could be improved by WebAssembly. We could even take Zencastr, the application that we're using to record the podcast right now to streaming audio application. It's recording our audio. What are some of the different ways that WebAssembly could improve the performance of our different highly resource- intensive web applications?

[00:51:24] TN: I think for all these application it really comes down to their specific architecture and where their bottlenecks lie. So Zencastr, for those of you listening, is an amazing piece of software. I've been really enjoying using it, because you can kind of just share out a link and then I can join the application and jump right into the action.

Of course, recording audio, especially multiple screens, can oftentimes be a bottleneck. So our hope is that WebAssembly, for example, just to take Zencastr, could integrate with maybe encoding the audio, uploading, or minimizing the audio files so that you could better uploaded it faster and have kind of less pauses and [inaudible 00:51:56] in it. But I unfortunately don't know enough about the architecture of Zencastr to say exactly where that bottlenecks lie.

But certainly we've had some explorations with internal Google teams, such as the photos team, about spaces and points in their application where they see [inaudible 00:52:10] and where they see a degrading experience, and WebAssembly could very potentially help decrease those.

[00:52:17] BS: Yeah, I agree. I think one of the things we're sort of seeing now is that there are a lot of applications that are maybe not so interested in WebAssembly, because it just doesn't really fit into their sort of the workflow. It really is currently very much more useful for people who have C++, or Rust, or code like that, sort of low-level systems codes, which means that it works great for things like if you already have a code base in that form or if you're doing something that's highly performant, like compression or something like that.

But we are planning and proposing many new features that will make it useful for people who are just doing every day average coding and providing features for languages that do that sort of thing as well. So I think that will sort of be the big turning point, and I think that's one of the things I'm most excited about in like the next year, two, three, four years or something, which will be when we sort of open the gates to all new sorts of languages that can be compiled to WebAssembly and then integrate it into the web, I think that will be very, very exciting. I think you'll see a lot of people start to just jump on it and use it for any number of things, like things we can't even imagine right now. I think that's very, very exciting.

[00:53:31] JM: Yeah, why hasn't Go been more widely used for web assembly modules? Why has the focus just been on Rust and C++ and C?

[00:53:38] TN: This essentially comes down to whether or not the language is a managed language, which means does the language have its own internal garbage collection system or does it rely on manually managing the memory? And something like C++ and actually Rust as well does have that manual memory allocation, and that means that when you write Rust code, we can translate that very directly into WebAssembly without having any concern around memory leaking.

The issue comes in when you have an internalized garbage collection system that you then have to compile. You can end up having some weird cycles with the actual web browser that can cause some of these applications to potentially weak memory, unless you are extremely careful. So it can be done in some cases, but it's something you have to be very careful about.

We're working on a proposal called the managed objects proposal, that's part of the WebAssembly CG group. What hopefully that'll enable is better integration of external languages, their garbage collection system with a garbage collection system built into WebAssembly so that they can better manage their memory and prevent memory leaks.

[00:54:38] JM: I didn't know Go was a garbage collected language.

[00:54:41] BS: Yup. Yea, it is. Actually, there is a version of Go that you can compile to WebAssembly currently. I think it's maybe sort of just more experimental at this point, but our goal is definitely to make it so that languages like Go can use WebAssembly and people can just not really have to think about it. Yeah, I think WebAssembly as it currently exists is going to be a little bit of a rough place for them for that, but we're definitely moving in a direction where we can make it easier for them.

[00:55:07] JM: JavaScript is necessary in this interaction between application, web applications and WebAssembly modules. As WebAssembly becomes more and more important, how do you think it will change JavaScript?

[00:55:21] TN: I always expect JavaScript to exist in the browser, and I don't expect that people, at least in my foreseeable future, what I can imagine, I don't imagine the people will drop having to write some amount of JavaScript. My hope is that for those who really don't want to, they can write extremely minimal JavaScript, but WebAssembly as a goal is in no way trying to replace JavaScript. So I think that they will continue to interoperate.

[00:55:45] BS: JavaScript, because it was the only language of the web for – What? 20 years or something? Ended up having to sort of put on a lot of different hats. So because it's a language that people right by hand, it has to have a lot of features that make it nice to write by hand. So a lot of the features that the technical group ads are features that are useful, but they are mostly useful for people.

So in some ways, I think WebAssembly takes a lot of pressure off of JavaScript as a language. You can push it in more of a direction to make it useful for people and you don't have to worry about making it a language that's useful for as a compilation target for programs.

The hope I have is that when people think, “Well, I want a language to run on the web.” They think, “Oh, I can use WebAssembly,” and they don't have to use JavaScript. I think for JavaScript, that'll be great. That means that they can focus on making it the language that they want, that language that people want, as opposed to a language that people are sort of cramming their ideas into.

[00:56:49] JM: Okay. Well, let’s wrap up with just a little bit of forecasting. So in about a year, where do each of you think WebAssembly will be? What kinds of progress will have been made and what will be doing with WebAssembly?

[00:57:03] TN: So I'll speak maybe to my hopes about the ecosystem, and Ben can speak a little bit more to the format. But I think WebAssembly is going to really continue to take off in the next year. I think we’re going to see a large number of large applications similar to the AutoCAD sketch up and thickness of the world start to come to the web, and especially existing native applications.

I'm also extremely excited about what the kind of grassroots developer community will do with libraries and what kind of libraries you will see. I think you'll start to see WebAssembly integrated both as kind of libraries by their own right and also integrate it into existing libraries and frameworks, such as the example of what Ember showed.

On the note of libraries, I’m especially excited about what people will do with the existing wealth of C++ libraries. You can take things that were part of CANoe or other like amazing open source C++ projects and libraries and you can just now get to utilize those in the browser. If you are an extremely awesome and kind developer, you will wrap that library in a JS API, upload its NPM, and now suddenly the entire developer ecosystem has access to that new library and functionality with minimal amount of work on any given one developer. That's where I see WebAssembly going.

[00:58:11] BS: Yeah. So I'm excited about all of that stuff too. I think I focus much more on the technical side of things. So one of the things I'm most excited about, and I think we’ll definitely see in a year, is the threads, and probably SIMD as well. That just means it'll be much more possible, like we were talking about earlier, for you to basically use all the functionality that your machine expects. So you could basically load up an application in the web and you could run it and not even have to think about it. You could you could basically have applications that are as powerful as things like Photoshop running in the web. I think that's pretty reasonable to assume.

One thing we actually haven't talked about at all in this podcast, which is really surprising, is we'll probably see a bunch of games too. We've been talking to a ton of people who are interested in games and bringing their games to the web. I think we'll see a lot of that. I'm very, very excited about that.

[00:59:05] TN: You're exactly right, Ben. It's surprising we haven't mentioned that. Just to maybe end the podcast on that note, we have been working really closely with Unity and Unreal Game Engines, and they both now support and export target that is the web so that you can take any of your existing unity or unreal content and then just compile it to WebAssembly and ship it in the browser and take advantage of all the advances in WebGL technologies to run an extremely performant game experience and then leverage all of the things that make the web awesome, like the fact that I can start a game, copy the URL, send it to my friend and he can just click it and instantly be in the action with me without having to worry about downloading or doing any of that kind of upfront labor. I'm really excited to see what kind of games will be enabled by that.

[00:59:45] JM: Very cool. Well, you guys have given me a lot of ideas for future shows. I think I can – Basically, I can almost envision a series of shows on WebAssembly in practice with some of these different verticals. So, Ben and Thomas, thank you for both coming on the show. I've really enjoyed this conversation.

[01:00:00] BS: Yeah, thanks for having us.

[01:00:00] TN: Yeah, thank you so much.

[END INTERVIEW]

[01:00:04] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested. With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers.

I know that the listeners of Software Engineering Daily are great engineers because I talk to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many, newer, hungry software engineers who are looking to level up quickly and prove themselves. To find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineeringdaily.com.

If you're a listener to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. Send me an email at jeff@softwareengineeringdaily.com. Thank you.

[END]