EPISODE 02

[0:00:00.0] JM: Brendan Eich created the first version of JavaScript in 10 days. Since then, JavaScript has evolved, and Brendan has watched the growth of the web give rise to new and unexpected use cases. Today, Brendan Eich is still pushing the web forward across the technology stack with his involvement in the WebAssembly specification and the Brave browser.

For all of its progress, JavaScript struggles to run resource-intensive programs like complex video games. With JavaScript falling short on its charge to be the assembly language for the Web, the four major browser vendors started collaborating on the WebAssembly project to allow programming languages a faster, lower level compile target when deploying to the Web.

Brendan is the CEO of Brave, which aims to provide a faster and safer browsing experience by blocking ads and trackers by default in his new browser. The Brave browser is also helping publishers monetize in interesting new ways while also giving a share of ad revenue to its users. Caleb Meredith is the host of this show. He previously guest-hosted a popular episode on Inferno, a fast React-like JavaScript framework.

As we bring on more guest hosts, please send us feedback. We want to know what every host is doing well and what we can improve on. Thanks again for listening to Software Engineering Daily.

[SPONSOR MESSAGE]

[0:01:30.0] JM: Life is too short to have a job that you don't enjoy. If you don't like your job, go to hired.com/sedaily. Hired makes finding a new job enjoyable, and Hired will connect you with a talent advocate that will walk you through the process of finding a better job. It's like a personal concierge for finding a job. Maybe you want more flexible hours or more money or remote work. Maybe you want to work at Facebook, or Uber, or Stripe, or some of the other top companies that are desperately looking for engineers on Hired.

You deserve a job that you enjoy, because you're someone who spends their spare time listening to a software engineering podcast. Clearly, you're passionate about software, so it's

definitely possible to find a job that you enjoy. Check out hired.com/sedaily to get a special offer for Software Engineering Daily listeners. A $1,000 signing bonus from Hired when you find that great job that gives you respect and salary that you deserve as a great engineer.

I love Hired because it puts more power in the hands of engineers. Go to hired.com/sedaily to get advantage of that special offer. Thanks to Hired for being a continued long-time sponsor of Software Engineering Daily.

[INTERVIEW]

[0:02:53.9] CM: I am here with Brendan Eich, the creator of JavaScript programming language and the CEO of Brave Software. Brendan, welcome to Software Engineering Daily.

[0:03:01.0] BE: Hi, thanks for having me.

[0:03:02.0] CM: Let's get started talking about WebAssembly. The lowest human-readable instruction format that we can give to a machine is some assembly language, the details of which depend on the machine. JavaScript is a much higher level programming language, yet it is still often has been described as the assembly language for the Web. Recently, a W3C Community Group, which you, Brendan, are involved has been working on a WebAssembly specification. Why is an assembly language for the Web useful and why is JavaScript insufficient in its current place as the WebAssembly language?

[0:03:37.5] BE: Sure. First of all, I'll start with how JavaScript came to fill this role. JavaScript, I did it in a hurry in 1995 in 10 days in May at Netscape, and it was partly rushed because of internal politics around whether Java was enough. It turns out, JavaScript, because it loads with the HTML and can be written by anybody and start from small scripts and grow to big programs, kind of vanquished Java from the client's side of the Web over the last 22 almost years.

JavaScript was a rush job. It had bones borrowed from other languages put together in a Frankenstein body in a hurry by me. Yet, because it got out early enough not just through the Netscape politics but due to the Microsoft Windows OS tying of Internet Explorer that was bearing down on Netscape, eventually, convicted monopoly abuse in the U.S. through Microsoft

case; we knew we had to go fast. If we got something up in Netscape 2, and there were several things like JavaScript, Java was halt for. There were new innovations in the HTML, a version of the time that Netscape was driving. If we got them out early enough, they would become part of the Web standards.

Really, JavaScript was one of the very few things that survived and endured. Therefore, it's this zero-install language runtime that is available almost everywhere, it's on mobile devices, it's embedded in apps through web views, and of course it's in the browsers, which are still economically important on mobile and especially on desktop, but on both.

Now, coming at it from a point of view of someone developing software, in the '90s, you would compile maybe for Windows PC. You would use Microsoft Visual C++. If you needed a database, you'd use Rogue Wave or [0:05:23.4] or something. Fast forward through the Web and suddenly, nobody does that anymore. Everyone's doing zero-install web-based things.

The game developers were still using C++, because they needed to get to the metal. To be fair, I exaggerated when I said nobody does that. There are other fat apps still being built or just fewer, and the browser has absorbed or eaten most of them and turned them into server side, server-based service software model businesses.

Yet, games and your sort of other hardcore, down-to-the-metal apps and your — If they have to use JavaScript, you have problems, because JavaScript is, besides being a rush job, we've made it better over the last 20 years in standards. It got better. It got bigger. It grew affordances and smoothed out some of the awkward spots and it has done well, but it is a garbage garbage-collected language. It has performance uncertainty. Performance unpredictability is one way to put it, where you may be giving something at 60 frames a second for a game and suddenly, you run out of real-time because of a garbage collection that has to happen to reclaim memory, or you've been using the latest engines that you just-in-time compiling, and your program has changed phase, of types that its speculated on through that just-in-time compiler.

Now, all that code that was generated according to certain assumptions or inferences has to be recompiled. That can take a lot of wall time, too, and can push you out of the current 160th of second frame of animation budget that you have, push you out of that soft real-time limit. Then,

the game gets to a slower, laggy, or the animation doesn't update properly. That's a competitive drawback for the games, so they like to stay in C or C++ so they like to stay close to the metal.

What we discovered, because JavaScript became so ubiquitous and so fast aside from these unpredictable performance cliffs, we found that if you use a subset of JavaScript — This was at Mozilla starting in 2012. If you focus just on the parts of JavaScript that looked like the C programming language and used memory in a flat, large, contiguous array, this was a feature edit for WebGL called Typed Arrays, then if you use that subset of JavaScript in that memory model, you could go as almost as fast as C or C++. You could go as fast as any safe, native run-time approach to such low-level languages. There were some predecessors in this phase, going back into the '90s, no one's heard of, but also notably of the portable native client work at Google, which was on-going at that time in 2012.

At Mozilla, we developed this subset of JavaScript. We called it asm.js in honor of Assembly language. We run a type system for it. Dave Herman did that. Luke Wagner of Mozilla who's still there did this amazing compiler back-end for that type-checked subset of JavaScript so that while you were parsing the JavaScript, you could decide that it was in that asm.js subset and you could very quickly generate really good machine-level code but with memory safety. That was the crucial requirement.

Anything you're going to have zero-install ability to execute from a website has to be safe. It cannot be running in a way that could just trivially own your user identity and possibly take over your machine. Securities are never done. There are always flaws in every piece of civic and software, so browsers have vulnerabilities, but having a memory-safe language like JavaScript or the asm.js subset was a requirement.

When we did this work with Mozilla, we also used a compiler that have been developed by someone there, Alon Zakai. He'd written Emscripten as an LLVM-based C or C++ compiler that generate a JavaScript. To tell you the truth, asm.js was formalized as a type system by Dave Herman, but alone actually — I think he spoke in 2011 JSConf EU. Alon had already developed an intuitive understanding of the subset in this type system for Assembly-like JavaScript by building this Emscripten compiler and making it generate JavaScript to making that JavaScript

go fast and using this typed-array approach to a very flat contiguous memory region for the usual C Global variables in the heap.

JavaScript stack is for the stack variables in C. The translation is very tidy to be optimized, and alone did this work ahead of us understanding what asm.js was. Again, there were precursors to this. There was somebody at Adobe Labs who did a system, I think it was called Alchemy, that used an earlier version of the LLVM compiler firmware to do similar thing targeting ActionScript 3 in the Flash Player. We were aware of that work and it was in this school of safe native code runtime techniques that we were studying.

When we've realized asm.js could be just a subset of JavaScript with the Typed Array extension. It was becoming standardized as WebGL, finally got into all the LSS including iOS. We realized, "This is likely to be the new safe portable native code runtime. This is going to kill portable native client at Google." The pinochle people at Google didn't want to admit that, but we actually made it happen. Luke Wagner wrote his fast compiler back-end. Alon Zakai kept working on his Emscripten compiler front-end that generated the code, the JavaScript code. Dave Herman made sure the type system was sound.

By late 2012, we contacted Epic Games, maker of the Unreal Engine, had a small team visit them. In less than a week, in four days, the fifth day they rested, was bringing the Unreal Engine 3 to the web by cross compiling its C++ code base, multimillion line code base, into JavaScript and to the asm.js subs in JavaScript. You had to do things like match their audio APIs to the web audio interface, and they had adapters for OpenAL and other audio libraries that was not hard.

They were already using OpenGL for mobile GPU optimized rendering. WebGL is based on OpenGL, so that was a quick adaptation. They had to fix a bug Emscripten or two, they had to fix in the compiling just-in-time back-end, which runs on the whole asm.js module, so it's ahead of time in some sense. That weekend, they had Unreal Engine 3 running full-frame rate in a prototype version of Firefox. Tim Sweeney, the founder of Epic, was stunned. He said, "I thought this would take years. Suddenly, it's here."

From that point at Mozilla, we just kept working on it, making sure that it worked for other games. We worked with the unity folks. I think they had announced the following year. We let the Microsoft know, and they got interested, so I had invited to talk at Microsoft to their engineers. After that, I met with Anders Hejlsberg, creator of .NET and C#, and Steve Lucco, who was then heading the JavaScript engine team, the ChakraCore team at Microsoft. They were very interested in asm, and they got on board at that meeting and pretty soon, announced it.

Over time, the last three years or so, asm.js just became this inevitability that you could make it super-fast. The runtime you already have I your browser instead of trying to add the second runtime is not only portable native client, but Dart had proposed to do and it pretty much failed. Even in Chrome, to succeed at doing. It's very difficult to add a second runtime after JavaScript. I can get into why. Even Flash, which was a second runtime as a plug-in, was going down. Steve Jobs had banned it from iOS, and therefore, it was withdrawn for android around 2011, I think.

There was nothing else in town if you wanted to have a fast machine-level optimizing runtime for languages than the browser. The only problem was everyone said it only understood JavaScript. Now, with asm.js, there was this subset that could be used as a target language for compilers. People had already been building hundreds of compilers from various languages like Python, or Java, or Ruby to JavaScript over the last six years. This was a topic at conferences. People were aware of Emscripten being just one of many among this large set of compilers.

Some of these compilers didn't have to work hard to optimize, because they were taking a language that was not particularly fast and mapping it to JavaScript. Often, they were mapping languages that were of similar semantic level and style to JavaScript, so it was a good fit. It came to be known as transpiling. It's even done for our future versions of JavaScript to older versions. Emscripten was the one that pushed things to the metal and got used by the game developers. Especially, when Unity 5 got their tool chain around, that suddenly, you could take any Unity game and press the button and out comes what they call the WebGL port, which really meant asm.js WebGL, web audio, and you can embed it in HTML. Just think you can have a game that had been on Xbox, have a second life on the Web, and you could put it on a

store that have try-before-you-buy, just load it in your browser and play. That was a great success.

How did that lead to WebAssembly? It was clear that JavaScript could be the one VM, or the virtual machine for JavaScript could be the one VM for mobile languages. JavaScript was still load at source, even asm.js's source. When I designed JavaScript, I didn't pick the shortest keyword for function, it's eight letters. They're just an inherent cost to parsing JavaScript. Even if you compress it when you're transporting it, and you transfer transport layer compression, which saves bandwidth on the networks, saves data cost you still have to uncompress it in memory on the target device and then parse it.

It turns out that was why running a Unity game or even a game that you can play today, like Ski Safari, is having an afterlife on Facebook as an asm.js compiled game. You can go there and play it. It's a lot of fun. It was originally a mobile game. It uses Unity, I believe. Kind of 2D-ish. It has a third dimension, but it's a fun downhill cartoony skiing game. It runs at full-frame rate, but the load time is a little slow.

It runs best in Firefox because of Luke Wagner's great work. The VA team at Google didn't quite want to do the same full program compilation that Luke did. The Microsoft folks did. They actually took Luke's and they changed the style, but they kept the copyright credit to Mozilla, and it had to be licensed in license they could use, so you'll actually find in ChakraCore, which is on GitHub. That's the engine in edge and IE now, you'll find a version of Luke's OdinMonkey compiler for asm.js. That's why some browsers are better than others at Ski Safari, but the load time is a pain.

If games change levels, there's a hit there too. Even before I left Mozilla in 2014, we were thinking, "How are we going to fix this? Maybe there is a binary syntax that those make sense for this asm.js subset." That's the glimmer of WebAssembly came into our eyes. Since we got Microsoft onboard with it, at some point, this might have been early 2015. I'm not sure when I heard rumors. There was a fight inside Google between the VA team and the Portable Native Client team. There was "blood on the ground." Then, the end, the VA team won. It was, just as I said, clear to everyone. You're not going to second safe native code runtime. Rather, let's take the JavaScript engine and give it a second input language, which is WebAssembly.

Apple also — Apple never announces early or agrees early, they always like to control their story at WWDC, but Apple got onboard. I can say that now in hindsight. I didn't want to disclose it then. People on Hacker News were doubting. Now, you've got all four browser vendors onboard with WebAssembly. Some of them are even saying, "Yeah, asm.js is just a subset of JavaScript. We happen to run fast. We don't even need to recognize it as such to make it fast. You give us some C-like, low-level JavaScript that uses a big Typed Array of memory. We'll make it fast. Don't worry about it."

Asm.js became less of this sort of whipping boy from the point of view of, "Why don't you have a proper binary syntax?" and just a stepping stone, a convincing argument to WebAssembly. Once all the browser vendors were onboard, WebAssembly was going forward as a community group project as you mentioned in W3C, and really on GitHub where the code and the design DOTs are.

[SPONSOR MESSAGE]

[0:18:16.8] JM: Release the Kraken! GitKraken, that is. Are you tired of feeling like you're sailing the stormy seas, because you have a clunky, old Git user interface? Unleash the beast, that is Axosoft's GitKraken. Voted 2017s most popular Git GUI for Windows, Mac, and Linux.

GitKraken is designed to make you a more productive Git user. The app offers efficiency, elegance, and reliability. The UI equips you with a visual understanding of your branching, merging, and commit history, and features multiple profile support, one click undo and redo, a built-in merge tool, and fast search.

Run the installer, open the app, and set sail with GitKraken. Easily setup integrations with GitHub, GitHub enterprise, Bitbucket, and GitLab. That's one high performance sea monster.

Visualize your version control and code on into the sunset sailors. Visit gitkraken.com/sedaily and use promo code sedaily to get $10 off GitKraken Pro. GitKraken is free for noncommercial use, so if you're a solo developer, don't worry about the cost, but we'd still love it if you went to gitkraken.com/sedaily.

Thanks to GitKraken for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:19:52.7] CM: From a basic level, what is WebAssembly? What is it look like? You open up a WebAssembly file, what's in there?

[0:19:57.8] BE: I hope I will be able to — Why it's useful and why it happened, but it is useful because it loads faster and gives you this low-level model of memory and instruction so you have definite performance, you can do 60-frame a second first-person shooter games.

What it is? If you think about JavaScript for a second as a C language, its expression grammar, it has all the bitwise logical and shift operators, it has some messed up precedence due to history of C language, and it has the arithmetic operators. JavaScript doesn't have integer types explicitly, but through the Bitwise operators, you can cast to integer 32-bit, un-sign its signed integers in temporaries, in expression evaluation. That's the key to asm.js. That's how asm.js could have integers that are fast that go through the floating point unit, don't turn into double-precision IEEE 754 numbers that people like to blame me for that are in JavaScript and Java. They were a problem with JavaScript kind of it only has that number type, so people sometimes want to use a different type and it just isn't there yet. We're working on fixing that.

In any language, it has IEEE 754 double binary precision. If you add .1, plus .2, and you print the results faithfully, you'll find that it's not .3, it's .3000000000000008. That's because you're actually doing the math in binary decimal and there's extra precision that's needed to compute how to round, but you do round with error, because of the finite precision, and you end up with that 8 at the end after a lot of zeroes. That's IEEE 754.

With JavaScript — The bitwise logical shift operators give you the ability to deal with 32-bit signed and unsigned integers if you use those operators as if they were typecasts. It's easy for a compiler like Emscripten to do this, so that's part of the magic that Alon Zakai discovered that led to a most formalized as asm.js to make really fast integer math in the subset of JavaScript.

WebAssembly just gives you better syntax for that instead of writing out infix operators in JavaScript, you can have — I think they've gone with a sort of little stack machine language for the expression part of the grammar and it could have a bits syntax. It's very efficient to parse as well as to transfer.

For the larger program structures that control structures in the programming, like if-then-else, and while loops, or other kinds of loops. Obviously, JavaScript has those. WebAssembly has them too. They're in a structured form. When Java was developed, it was mapped to bytecode, and then that bytecode is sent around the web. The problem was you could get someone untrusted sending you bad Java bytecodes. You had to run what's called the verifier on it.

For the longest time, absence in type information, they finally added verification could have pathological behavior. I think my friend Michael Franz, a professor at UC Irvine showed that they have order end of the 4th power complexity if you send that torture test, sort of a travesty of bytecode at a verifier, it would suddenly take a long time. You could do denial of service attacks on verifiers.

With WebAssembly, we learned from this — The WebAssembly binary syntax, even though it's a mix of prefix and post fix for the expression part, SAC machine code, it's sort of self-verifying in the sense that if you parse it, it's not going to do anything crazy. You don't verify it that it's not tricking you. Because the verifier was trying to check that this malicious bytecode didn't try to do type confusion, where an if-then-else, and the then part, you send the value through memory, through a variable of one type, and in the else you attack using a bad type pond, or a different type, and then after the if-then-else, the values are still used, but they're used under the wrong assumption that they have only one type.

That verification was important for security. With WebAssembly, you don't have the sort of nasty control flow analysis problem that Java bytecode had for the longest time, where you can have type confusion unless you do this careful analysis of the spaghetti go-tos that are possible within Java bytecode.

As far as I know, there is no unrestricted go-to at WebAssembly. They've been changing it a bit, but I think it's still a structured code in the old '70s sense of structured programming. No go-to

consider harmful. It is more concise. It loads faster, parses faster, uses less battery. The games you play like Ski Safari now can just be that much faster and better when compile to WebAssembly instead of asm.js.

[0:24:37.8] CM: That transitions well into our next question. You've been talking a lot of video games. Frankly, most WebAssembly presentations also demo some kind of video game run in C ++ and compiled with WebAssembly. What are some cases for WebAssembly besides enabling graphics-intensive video games thrown on the web, or is that the only use case, and that use case is just incredibly compelling with the rise of WebVR or for other reasons?

[0:25:05.1] BE: I think the latter is a good point, but it's not the only use case by a longshot. First of all, games are very sophisticated. I used to go to the Game Developer Conference. I remember hearing it top from one the Valve CTO. He made the point that games — It took over for practical systems research from operating systems. They went into the GPU when it was still possible to have GPUs not manage malicious code while we're at sandbox, different threads of execution. They went into multicore. They went into vector processing units, the so-called SIMD units, single instruction multiple data units that we have on pretty much all company hardware now.

Games were really using everything; they're using threads. They were doing DSP. They're doing physics, so that's not rendered, but it requires a lot of computation. They're doing AI. AI is now a big buzz phrase everywhere.

WebAssembly is good for all those use cases. It's good for machine learning, algorithms on your device, which can be quite practical. They don't have to be in the cloud using super computers in the cloud. They can sometimes run using simpler algorithms very effectively on your local data. If something were interested Brave — People need AI not just for games, but for all sorts of things now, for assistance and helper apps, voice to text and vice versa.

WebAssembly is just good for all of these computationally intensive workloads. Games are great, because they're fun to demo and people like them. Like I said, they really do put it all together. The one thing that we can't pixelate is the sort of the dynamic language workload of JavaScript. People think, "Oh, WebAssembly, what means we can get rid of JavaScript." Well,

not yet. WebAssembly is starting as this target language with the fixed memory space for C and C++, and there are things on the road map that will add garbage collection, which has to be done in cooperation with the one true garbage collector of the one true language runtime, the JavaScript engine.

Otherwise, you end up with this guest-host cycle collector problem, where any time you have two garbage-collected systems that can form references together at heaps, they can form cyclic references. If the garbage collectors don't have special protocols for talking to each other, or there is no super collector that runs, those cycles become uncollectible. That's an argument for what .NET has, which is a multi-language, a polyglot, a virtual machine that has one memory manager. That's what JavaScript is growing now, it grows the polyglot support to have assembly.

ebAssembly in its current minimum viable product or MPV form is really abo a being a great target for C and C++ and maybe some other languages that can fit the model. Over time, you have to add garbage collection in this consolidated way so that everyone's using the host GC and Java Search Engine not running their guest GC and the — Or the assembly code. You can do that, too, if you want, but if you ever form references, you're going to have uncollectible cycles.

They also want for dynamic languages fast, dynamic dispatch of a method that might or might not be on an object. Like JavaScript moving in Python, sometimes you have no type system, no stable in the code of what the type of an object is, and you call a method of it. You call the same method pretty much from the same call side all the time. This is how just-in-time compilers test will speculate. Yet, WebAssembly doesn't have the instructions for that yet. Even the JavaVM guru invoke dynamic if you know what that instruction in the Java bytecode does.

That can be put on the road map and that's going to happen, I think, but it's down the road. A few other things have to done to really support truly dynamic untyped languages as well as these statically-typed languages C and C++ and Rust from Mozilla and others that I think what will work well with WebAssembly as their compiler target language right now or very soon. That means you can't get rid of JavaScript that fast. In fact, JavaScript's going to be obligatory for so long. There's no definite schedule, which you can say, "Oh, yes. By fourth quarter of 2018, we

will take out JavaScript and just have a WebAssembly engine, and all of the JavaScript in the Web will have to be compiled maybe by the browser into WebAssembly." That's not a credible plan. There is no such plan yet. We'd evolve there over many years, and think it could happen, but it's just over the horizon.

[0:29:30.7] CM: Will code compile to WebAssembly eventually have access to all the browser APIs that JavaScript has access to, and will you be one day be able to write and entire application in a language other than JavaScript that compiles to WebAssembly?

[0:29:44.8] BE: I think the answer in principle is yes. The APIs that JavaScript can access should be accessible WebAssembly. You can do that any way since WebAssembly is designed to work in a module, which is something that's coming to JavaScript in the ECMAScript 2015, or so-called ES6. We're still getting the way that modules work in browsers down as a separate spec out of the working group. It's taking a while, but the WebAssembly idea is modular. That means you'll have — Like I said, you could have a JavaScript app that starts using a WebAssembly module for machine learning or for a little bit of physics engine, or a little bit of intensive numerical computation. You could do that right away of you use C or C++ to write the source of that algorithm that you're putting into WebAssembly.

That means you'll have a mix-and-match model for JavaScript to WebAssembly, so it's always possible to sort of proxy API calls that JavaScript can do to the WebAssembly code. There is good thinking going on right now in WebAssembly in the design group, which concentrates some big brains among the four browser vendors on how to make it easy and safe to call any API efficiently through WebAssembly.
I think the general answer is, yes, you'll get all those APIs, and yes, you could like the games that are already cross-developed are entirely in the target language. It may be asm.js today, because WebAssemblies were just turned on in some top browsers.

In a year, those games like Ski Safari on Facebook, Heroes of Paragon— Those are games I've played on Facebook compiled to asm.js, they can be compiled to WebAssembly. When they are, pretty much, the whole programming as WebAssembly. You can do modules, you can mix and match, you could do a whole program.

[0:31:31.9] CM: In a world where a developer can write an entire program in any language that compiles to WebAssembly, then JavaScript, the language, must make the argument that it is actually a good programming language. After all, a developer could write their entire app in Rust or Haskell or some unknown future language without touching JavaScript. As the creator of JavaScript, I'd like to ask, why would anyone actively choose to write their web app in JavaScript? Is there some merit to JavaScript besides the fact that it's part of this evolutionary system and it has the share it has today because being the only dynamic language in the browser?

[0:32:09.4] BE: I think opinions vary on languages. You go to Hacker News and you still see people wasting [0:32:15.4 globs] of time arguing about blub. Blub —

[0:32:16.9] CM: I really want to hear your opinion though.

[0:32:19.9] BE: I will give it. Paul Graham's view of things is people just fall in love with one language that they learned first, or that they think is best, and they argue chauvinistically for it. I'm not going to do that for JavaScript. I use many languages. JavaScript has virtues apart from being ubiquitous through getting on early enough due to that Rust job in May, 1995 that I did, but JavaScript has evolved and the evolution has been important.

You hear of armchair, Monday morning quarterbacks in the 22 years sense say, "Oh, it could have been Python." "No, it couldn't have, because Python would have been frozen like a flying ember in 1995." That was Python 1.3. You don't want that.

Also, Python had unsafe foreign function interface. It still does. You don't want those on the Web, so it would have been a subset of Python. Then, if it was frozen as JavaScript was for too long, and there was this standards process installed, and then we have to have Firefox to restart the browser market; you would have had evolution of Python along a different lineage. It would have been Web Python. It would have gone from very old, out-of-date. People would have been cursing it for not being up just enough with Python 2, and then it would have tried to jump forward, but it would have to go sideways of it, because the Web is a different environment.

Evolution matters. You can have the dinosaurs complaining, “Oh, if only the oxygen level hadn’t dropped, and the temperature hadn’t dropped, and those free critters wouldn’t have won. I’d have won.” That’s sour grapes.
I say JavaScript actually has benefitted from that evolutionary niche it’s been in, which is, in terms of space, a huge space, not a niche. It definitely meant this monopoly that people resent, where JavaScript was the only language. There, before the grace of God, goes Python, because otherwise you get this flying ember, broken distaff version of Python that people would be cursing instead of JavaScript.

The other thing about JavaScript, I’ll say, is that I did, in spite of the rushing and some blunders, get some things right. It has this least authority model. If you get rid of a few back doors that are pretty wide, it can be an object capability model. That’s what Mark Miller at Google’s research with Google Caja announced secure atmosphere pretty much folding into the standards, so you can — With strict mode and a few other things, you can control trust in a mixed trust environment. You can use object capabilities, security models with your basically safe pointers and proxies and membranes to moderate authority. That’s a good thing in JavaScript. That’s not in every other language, and that also co-evolved with the Web.

Now, there is a concern I have with WebAssembly, because it’s now in W3C. There’s this sort of cargo cult thinking in W3C that says, “You must have the cross-origin request security model.” I think that — I forgot what it stands for — CORS. For anything that goes cross origin. By default, you can’t load WebAssembly that’s bad. There’s also this knee-jerk reaction against JavaScript’s eval feature, that anything like eval or the function constructor in JavaScript that can take a string of code and turn it into executable objects is bad.

WebAssembly should only come from URLs. They should never come from a string. I think that’s just nonsense, too. We compose programs out of strings all the time. We do it various ways on the Web even if you try to neuter eval, it comes back. It rears its head. It’s very useful, and the security properties are not all or nothing. It’s a tradeoff. You can still secure programs with the use eval. You have to be careful to analyze the code that you’re taking in as a string. That’s always true, because you’re taking in the code as a string for URLs, too.

There are risks, I think, in the modern world. Security never have been even solved and certain problems vexing big companies like Google. You get some kind of — As I said, cargo cutting or all-or-nothing, thinking about security, which is actually, I think, not good for either security or usability. I don't like the old security usability tradeoff. I think the best solutions I've seen have included both.

On the topic of languages though, and security, Rust is a language that has a static safety model. You cannot have a null point of your reference at runtime. You cannot have your memory errors out of bounds array in the Cs. You cannot raise conditions in multi-threaded code, which is a theorem for free that comes with the ownership system.

Rust is very interesting being compiled to WebAssembly, and I would choose that over JavaScript whenever I'm doing some heavy machine-intensive but safety-oriented code. Yes, JavaScript is not the hammer for all nails. It never really was, that's why some people would use Java or Flash's ActionScript or even compilers that generate code of ActionScript. Those other runtimes kind died. They became malware vectors, both JavaVM, the JRE plug-in and Flash.

Because they were owned by one company that took its eye off the ball and didn't update the install-based fast enough, or if it did, still didn't come after security bugs hard enough like Flash. It became this easy, exploitable bug farm for the bad guys, and they didn't have to worry about the different browser versions.

The diversity of browser implementations actually has been a hindrance to those bad guys. I think that's going to be true of WebAssembly. There'll be different engines to attack, so it's better to have some diversity in your system and implementation level. That's a deeper point I've made. This is a research by Constantine Dovrolis of Georgia Tech, I believe, that in any network system with layering protocol stacks, or language stacks, or both, you tend to get these hourglass-shaped evolutionary structures where the wasp waist through the hourglass is like JavaScript or HTTP TCP/IP. It becomes this evolutionary kernel that is in everyone's interest to be backwards compatible and evolve slowly and carefully with security in mind. Then, you get great diversity above and below. Like in Dovrolis' work on network stack, you get lots of link layers from the old 10-megabit Ethernet on Coax all the way to Metropolitan Ethernet on fibr to satellite or other communications, radio communications through our phones.

hose link layers are all different and innovative in their own time and space, but they all funnel this old protocol TCP/IP through. That's evolving, too. We're getting HTTP 20:38:48.5 HTTP/2 and click from Google. There's always slow evolution there, but it's very hard to replace those older protocols. We still use the DNS, the Domain Name System. These form of evolutionary kernels, they are stable, they can serve their valuable DNA. The same is true of JavaScript. Since the browser won over each other's runtimes, either on mobile now, it just seems like this is going to continue for a while. Again, I can't put in a stop date on JavaScript.

[0:38:54.8] CM: Last question, and it'll help us transition into our conversation about Brave. Now, with WebAssembly, are we about to see a new era of browser wars as everyone competes to have the best WebAssembly implementation?

[0:39:27.7] BE: It's a good question. I don't know. Browser wars are hard to predict, but they do come about when there's something really broken in the browser market like the eye in monopoly convicted in the US. Genuinely, a case of neglect from Microsoft, which I think felt burned out by the U.S. v. Microsoft in a trust case after having questioned that stupid thought, standards are hard. It hadn't yet been punished in the EU for the Windows Media Player and further browser monopoly behaviors, so it hadn't form its standards bureaucracy and diplomacy arms that Microsoft has now. Microsoft does standards much better now.

At the time, they said, "Let's go back to Windows. Let's do good old Windows lock in. we use .NET because Ander's done a good job with C# and the .NET runtime." That's set the stage for a new browser world, which was started by Firefox. When I was one of the people who spun it out, I picked all the technical staff and was one of the general managers in the sense of the project when we got out of AOL. We knew that the browser market was soft, because Microsoft had walked from it. They left I6 as a sort of — Or at that time, 5, but 6 was no better; a skeleton-crewed terrible browser that had huge security problems due to ActiveX and a lot of popups as a form of knowing Window-level advertising, so we did things like popup walking.

Browsers get soft for a lot of reasons. I don't think WebAssembly might be the softness of it, because I think all of the browser vendors can do the Java WebAssembly. I'm actually concerned that after the MVP, the minimum viable product level that's good for C and C++ or

Rust or languages like, that the browser vendors might go their separate ways, because they can't really agree much under competitive pressure; it will take some disciplined work and some good social networking to keep WebAssembly going toward that further opposed MVP road map of garbage-collected language support, dynamic language that make fast method call support, other features that they want.

Maybe they'll get there. I hope they do. Maybe they'll get there more slowly. They might have a bit of a pause on WebAssembly after the MVP, where it gets absorbed and people figure out on the developer's side how to use it best, how to do those modules from machine learning, or physics, or any other numerically-intensive work, how to mix and match the tools, because people are using tool chains now with JavaScript, which is a big breakthrough.

16 years ago, people would say, "Hurrah, JavaScript! I want to write to the metal only. No stinking tools." Then, slowly through Lint and JSLint and ESLint now, and then Babel 6 [inaudible], people are now used to compiling, and it's actually been helpful to smooth over browser differences. So having people learn, as developers, how to mix in WebAssembly and compile from another source language two of assembly mix it with their JavaScript. Whether it's the whole app or part of it, that will take time.

Really, I don't want to sound like I'm speculating darkly about the browser vendors failing to cooperate. Maybe they'll do great in keeping going on the WebAssembly road map. In truth, it does take time for developers to absorb these things, so we have to give it time. We have to not try to over-predict it or over-constrain it.

That makes me think the browser wars, if they are heating up again, are more about higher level concerns such as ads and tracking and where the browsers are failing their users right now, because the main browser vendors actually have conflicts of interests with their users.

[0:42:54.0] CM: That moves us into talking about Brave. The web has moved a lot towards centralization since the web's early more open days where a control over-identity and data was distributed. Many people think this centralization combined with other trends have broken the Web. Is the Web broken, and how does Brave, your software company who's producing a new browser, plan to fix the Web?

[0:43:17.2] BE: It's a great question. I think anybody who studied the Internet realized it was designed to withstand the nuclear strikes, and it was essentially and potentially a peer-to-peer network. Peering is important. You have to route traffic for other people sometimes. This became a hot topic with Netflix allegedly poaching them with Comcast, or Comcast was trying to build out and cover the costs of its last mile where the video over the top was taking a lot of the bandwidth. You have to work on the common infrastructure, but it isn't all peer-to-peer and it need not all to be all peer-to-peer. We still have trust relationships. Humans have evolved over 10,000 years. We're used to some amount of centralization, and it's inevitable.

I'm not one of these radical decentralist who think, "Everything will be decentralized, and it will all be different, and you'll have to learn how to be a decentralized user," because, for one thing, it's harder. You don't trust anybody, so you end up having to do more proof examination or confirmation. It's not for everyone.

The other thing is, since humans are used to some amount of trust relationships in their lives for good or ill, they will want to use those. They will want those to be reflected in the Web. Like I said about evolutionary kernels, it's inevitable in the network from physics to biology to the Internet to have first and second place winners for a time come to take a large share with the market.

The problem that I'm particularly struck by though is the data, and this is something Tim Berners-Lee is also working on with Solid and other people who care about this certainly in the decentralized Web movement that [0:44:48.7] folks beat your browser. They care about this. Why should your data be tied into your app. You have this awesome app, all of your friends are on it, you want to go there to socialize and chat, but don't you own your own data? Shouldn't you keep your data separate from the app? No business will get investment if it force war that data.

All of the businesses that keep data in the server side have terms of use to pretty much make you this farm animal for them holding your data and harvesting value from it mostly for advertising. Maybe some other services there, they could charge for.

That also seems inevitable, but Brave is trying a different approach. We don't want to have your data on this server. We don't want to see your data in the clear. We don't even want to have the temptation of it. We want your data to be owned by you on your device, encrypted when its synchronized cross device for your use as a shared bookmarks and tabs and history and things like that. We propose to offer you—You would choose this, so it would be opt in, the ability to add some local machine learning to add value, mind that data for things like the opportunity to get ads that are anonymous that pay you a revenue share.

There is no cookies, no tracking. The ad matching happens only on your device. How that would work is pretty simple to describe. Ads don't come out every minute, and they don't change all the time. They take some creative effort to produce and they sometimes run campaigns for weeks. You've probably seen them follow you around longer than you'd like.

That means we can have a catalog that to download that can be delta-updated of just the ad cache URLs. These don't telegraph a lot of information except to whole network actors like the NSA. They can come up with keywords to match against. Again, if the user of Brave doesn't opt into this, none of this happens. If they opt into it, there'll be a local agent in the [inaudible] machine learning, studying everything about you that would be transparent. It would use naïve base or other simple machine learning that can explain itself.

One of the problems with machine learning that uses convolutional neural nets deep learning. it really explain why it made decisions. There is, I think coming in Europe, the GBPR, this right to explanation where you're supposed to ask your machine learning, "Why did you decide that?" If it can't tell you, there's going to be trouble.

In the Brave philosophy, we keep your data local. It's encrypted if it goes anywhere near servers that we control, and it's encrypted with your key, not ours. It's studied by algorithms that can be explained. Those algorithms, yet, in spite of that relative simplicity and transparency, I think, can do a really great job, a much better job than a lot of the parasitic ad-tech players do today.

We've all seen not only ads that retarget you unknowingly but really bad ads that just mistarget you, and you have no idea why. You can say, "Oh, I'd like to opt that as ad." Good luck. A friend,

Rob Leathern, tried this, and he went to 300 different company screens to opt out. Some of them were just broken forms, they 404ed.

It's a joke to say that you're in control of your data right now if you don't defend it. Brave's posture is to defend by default. We block third party ads and trackers, we use various techniques for this. Some of them are shared with other ad blockers tracking protection software. It's a shared resource like the disconnect.me tracking protection list or the EasyList so-called that a lot of blockers use that Adblock Plus pioneered.

We do not do anything like the white listing or what might be suspected of being paid to play where you take money from advertisers, and then long and behold, sometimes, those ads get through your ad blocker. I've had conversations with people who do this at AdBlock Plus. They say, "No. No. The left hand doesn't know what the right hand's doing. We only allow acceptable ads. By the way, we sustain ourselves with a little Robin Hood, Robin from the rich." I'm not convinced. No one's convinced. It looks like they're double ending the system in a way that creates inherent conflicts and temptations, and B0rave doesn't want to do that.

Anything we do in this face of decentralized is going to be user controller. It's going to be on device in the cleared data only, no in the clear data on servers. It's going to block cookies and trackers. This is important for security, because you may be aware malware has come into ad exchanges, its third party ad exchanges. By the malware vendor buying cheap ad slots, they've been able to get malvertisements, so-called onto, even the New York Times last March and BBC Online AOL, other sites.

It's an on-going problem. It's not a well-understood problem, because these look like ads. You cannot judge them by their pixels and say, "That's malware." They even hide the malware loader script using steganography, small perturbations to the color and the luminance of the images in the ad. They extract those perturbations with a little innocuous looking JavaScript that creates a string and then evals it — Let's go back to my eval pointer earlier. That evals to a loader that loads in the exploit kit.

The exploit kit is just a recipe for trying to this, trying that, trying the other, see if the Flash Player's there, see if it's got this known vulnerability or this dark net no yet disclosed your past

vulnerability. It tries Silverlight maybe, it tries browser vulnerabilities, and it can often clone your system, because you shouldn't have Flash. Brave turns Flash off by default. These old plug-ins that aren't well-maintained, they come from a single vendor, like I said, become a honey pot of bug forms overtime. Just very attractive, because they're so widely distributed from a single vendor, and they're not well-maintained.

Even if they're well-maintained, every sniffing in piece of software has security volumes like I mentioned, so they have to be patched up. That's the other thing that exploit kits look for is older versions of plug-ins. When they find them, they can put ransomware on your system, or they could suddenly corrupt it to make it part of a botnet.

There's a real danger to advertising that I think motivates defending your data. For a decentralized web in the future, to say that we're going to go from where we are today toward you owning your data by gentle or even harsh language at Facebook or Twitter to say, "Give me my data. I own my data. You can have your app and you can absorb my social graph next edge connectivity to my friends, but I should own my direct friend list. I should own my tweets or my posts." Good luck with that.

I know people who use RSS to syndicate their posts to Facebook, and they also have sent it to Twitter. They have to run their own servers. They have to be very technically savvy. Most users will never do that. If Facebook doesn't like the way you're interacting with it through a tool or a proxy, they will shut you down. They've done that before. They want you to stay in the farm as a farm animal harvesting attention.

The economics behind attention are interesting, because they go to advertising but also to search. They create a lot of value for users if they're well-done, not just ads that keep some people — Some of them will do like it, so let's face it. Some people hate them and never want to see them, but search results, in some sense, require noticing attention.

Google made it clear and demonstrated very well that sometimes that they can correct your queryj, but based on other people's queries faster than you can, or they predict your query. That can be helpful. There's a privacy trade-off there that we don't like at Brave, but even with your

own search history, your own query log, you can do a lot. That's where I find this enormous interest in the future for Brave in looking at personal query log on device and studying it, because when I'm searching for something, I've often — I'm refining a search last week that I've forgotten about, or I'm doing a search that I've done before.

The idea of decentralized in the Web, it think, has to start aggressively from the other end, which is the user's client devices, the end that's currently just treated as a dumb terminal in the worst case. Make it smart, make it hold the data, and then we'll have a more level playing field on which to decentralize appropriately. Use peer-to-peer protocols. Use block chains. Use the techniques that are evolving still to decentralize the data of the Web.

[0:52:59.7] CM: Yeah, It's fascinating, but how does Brave plan to monetize this platform?

[0:53:04.9] BE: We talked about this about when we launched a year ago at January. If we get users who choose to take these private zero-knowledge ads — I didn't mention the other part of this, we privately match against that ad catalog without any cookies. The machine learning that you would opt into would study what you're surfing for in this group of tabs and possibly keep it separate from other windows where you do other tasks, like research or work, and develop a model for predicting a few keywords to match against the catalog that's been downloaded and delta-updated.

That's a very private way of getting the right ad at the right time in front of you. With the right publisher progress, we'd even replace the ad on the publisher's page. We wouldn't want to do it without the publisher's permission. There are other ways to get these to you though.

Then, the problem is confirming that the ad was viewed or acted upon. Some ads only pay better if you actually click on the download this app or game at the end link in the ad. There's a video ad. We could do those two with high privacy deterministic anonymity, anonymous identity, through the same zero-knowledge proof protocol we're using for what's already in Brave as a beta program, Brave payments, so-called a way of automatically donating your top sites to give back $5 a month, $10 a month in lieu of the ads that you blocked. People are using that. We have over 48,000 user wallets created right now.

That's something we're going to turn on even more as we get a better system for getting funds into that user wallet. That's just part of our ambition. Once we have that donation system at scale, we can then start paying users who choose to see zero-knowledge ads for their attention. When you choose to take ads, even if they're well-targeted at you, and you like them, and they offer you fine things that you want to buy; they're still taking up some space on your screen or some space somewhere. It could be email if you want to get it through email promotions once a day. It could be a chat bot. It could be a full screen videos channel that's outside the pit.

However they go, they take some of your attention, so you should get a share of that advertiser's spin that comes in from the brands and their servants, the agency. That money currently goes to a very inefficient fraud prone system where people slice away at the pie until the publisher is left with a tiny slice of pie. We want to give a bigger slice to publishers. We want users to get a slice.

We talked about this last year, and of course, the first thing that happened was the publishers who are often zombified, as if they were a roach that had been bitten by a jewel wasp. Jewel wasps will zombify roaches and ride around them and I've never seen this. There are videos on YouTube. The publishers started howling as if they were the AdTech partners that were actually threatening, kind of what we're proposing with Brave would get rid of all the parasites and the middle players who are taking out to much of the pie.

The user would get a share of the revenue, and the revenue is there, 70 billion plus was spent on digital advertising last year in the U.S. I dare say most of it was wasted. There's an old maxim in advertising called Wanamaker's Dilemma, attributed to Jude Wanamaker who owned a chain of department stores in Phili over a hundred years ago. He's alleged to have said, "Half my advertising budget is wasted. I just don't know which have."

This was in the days of newspaper ads and catalog ads and you didn't have computers with JavaScript to tell you when someone viewed an ad, so you were sort of guessing what would sell and guessing where to buy the ads based on the newspaper. If half was wasted then, you would think would computers it'd be more efficient now, but it's actually worse. There's fraud, there's retargeting where people make ads follow you around that don't work for you and they

drive you to get an ad walker and then your lost through the system, so called a negative externality, sort of like pollution, attention pollution.

There's also this sort of problem that they had to guess what websites will go to. If you're advertising Ford trucks and you try to identify likely buyers of Ford trucks, you still don't know exactly where you're going to hit them on the web, so you buy ad slots on 10,000 sites. That's inevitably wasting more of your money. If you just could get the likely Ford truck buyer at the right moment in their browser, which is what we propose to do with Brave, you could spend far less and there'd be more money to go to the publisher and there'd be some for the user, and that's our model.

It basically takes — If you think about advertising online now, there's the buy side, the brands, and the agencies that help them that put money into the system to buy ad spaces. There is the supplier sell side, that's the publishers who give up ad space on their page. In the middle is this incredible sort of ecosystem of middle players, ad servers, data management platforms, optimization services, measurement services, anti-fraud services, and they all take a cut.

Take that parasitic middle ecosystem out, put the user in the middle, get them tools like Brave with block chains and zero-knowledge proofs, give them privacy, let them control their data, that's a much simpler and more efficient system. It guarantees that there's a real user, not a robot viewing the ad. It guarantees that they saw the ad. That's what we're trying to build, and that's definitely a viable business, because we know right now, for advertising, there's somewhat viable business for somebody there. It turns out increasingly, it's Google and Facebook.

If you look at how the 7u0 billion is spent, 80% of it going to Google and Facebook is not really great. After the increment over 2015 where more 60 billion was spent, almost all of that added 10 billion is going to Google and Facebook. It's becoming a duopoly of Google and Facebook that's absorbing all the spending, because they own not only a lot of the middle tracking powers as parts of their businesses. They also own, in the case of Facebook, this really is their business, they own the publisher side, or the app side. They have the user facing property on which to advertise, which in Facebook's case is your feed, or your Instagram.

Google has the search engine result page, but they also bought DoubleClick in 2008, so they sell ad serving, and they sell this sort of ad exchange business to publishers and advertisers use it, so they take a cut when the ads go through. That means there's sort of a conflict of interest on their part. I mentioned this earlier, whenever you have big companies start to find their business interest in conflict with their user interests, especially browser user interests, like we did when we're doing Firefox against Internet Explorer 6, you find an opportunity for a new browser war.

[0:59:23.9] CM: Is it morally okay to block online advertisements and replace them with your own Brave advertisements given that for some websites it is the only way they make money?

[0:59:33.7] BE: Let's take then two parts. First of all; is it morally okay to block ads? Absolutely yes, and here's why, Doc Searls talks about this. There was a great Medium post he wrote where he talked about his sister who's a 20-year plus NAVY veteran. It would take the New York Times Sunday paper edition and she would field-strip it. She would take out everything she wouldn't read, ads, news, definitely some ads she would not read. Other things she wanted to read, like the travel section, and she would see the ads there. Maybe she would take a classified insert that was all ads because she knew she wanted to look through those. Ad blocking existed long before digital ads and browsers and ad blocking extensions. People were always free to field-stripping these papers to ignore things like that.

Ad blocking is actually woven into the web standards. The web standards are not a VRM video stream or image where you cannot tamper with the ad, because of the digital millennium copyright act. As Cory Doctorow explains, DRM is really a legal hack to try to control supply of playback devices to jack up prices a little bit. That's all it is.

All the technology for it turns out to be evadable. People pirate movies all the time, even if you have very strong so-called high definition DRM, or ultra-high definition DRM that users secrets computers inside your computer running in a hypervisor, in hardware ring minus two. People put in HDMI dongle on and they get the high-definition pixels out that way and they put them on the darknet.

DRM doesn't want technically, but it has legal teeth, thanks to acts like the DMCA in the U.S. and its counterparts around the world. That's sort of the nasty truth about DRM. The web is not like that. The web is a set of hyperlink text, hypertext and multimedia embedded in that. It was always designed intentionally so you can mix and match pieces of it. You can pullout just a text and use it in reader mode. If someone is visually impaired or handicap, they can use a screen reader that turn texts to speech. They can throw away the ads, and that was always part of the web design. To op that back or try to DRM the web is, in my view, heinous, it's like depraved. It's completely wrong.

I think most consumers agree. Nobody really wants that. People want their Netflix. They do not want the web to suddenly say, "You can block ads." Consumer sentiment was entirely on Brave's side last year when we sort of poke the stick at the AdTech sort of parasites and ended up with the poor roaches that are riding in the back. Some of the publishers wrote us a letter. It was legally meaningless, but it was threatening, because it said, "How dare you replace our ads?"

Let's answer the second part of your questions. Blocking is legal, and legal, and ethical, and in my view, necessary, for safety due to malvertisements. Can you replace ads? That's a gray zone legally, but also we wanted to pay the publishers. We always said we would give the publishers 70% I full of that revenue share, and that's more than they make from their third party ad partners now.

The IAM, the Interactive Advertising Bureau, in 2014 did a study and said, "Oh! Publishers make 45%. Isn't that great?" No. In the app store, the Apple App Store set the standard, 70%. In old media, like television and print, 85%. Going down to 45% is not good. Actually, publishers doing third part or programmatic ads will tell you, "45%, I wish. I make 30%. I'm making 20%," because they don't necessarily know the full gross spend that's coming their way. All these middle players, these parasites, take so many cuts out of it that what's left to them is not 45, it's less.

With Brave, we proposed to do replacement giving them 70% in full, 50% directly, write direct to them as fast as possible. Yet, that wasn't good enough. They kind of had a knee-jerk reaction, because I think they've been captured by the current ecosystem and its parasitic middle players.

The other thing we did was we didn't build this, we just talked about it. If we built it, we would deal with publishers as partners. We would want the publishers to refer us. Think about it, Brave users coming to the publisher are going to block the ads by default, the third part ads, so publishers lost that portion of the revenue. Publishers do — Sometimes do what's called direct sales. They take their best ad slots and they sell directly that space to a brand, like el.com sells to Louis Vuitton to put a beautiful handbag custom video at it.

That we like not to block, because that ad is almost like a sponsorship ad, it's almost like an image on the publisher's page. Why would we mess with it? Unfortunately, it's placed through Google's DoubleClick ad server and it's full of tracking for various reasons, but that could be improved through zero-knowledge proof, so that's okay.

That's a direct sold ads. The indirect, or the programmatic ads are the ones that we proposed to replace. We're going to block them anyway. Saying to a publisher, "Hey, you're starting from zero with Brave users. Why not try our alternative to your unsafe third party ad partners who are taking too much of the revenue and sometimes letting malware through. Why not use Brave as your third party ad partner?" We would do it with publishers and partners. That, we didn't exactly say, because we were trying to make a point.

That's why it was so funny. I heard this story from other people. When you do poke the ecosystem this way, usually you get this zombified roach, the publisher is screaming loudest at you, but, really, the person who's threatened — The actor who's threatened is the AdTech middle player. They always lead with copyright. The letter that we got said, "How dare you infringe our copyright?" I thought, "Copyright? Interesting." You hold copyright on the third party ads, they're placed in slots in the New York Times by JavaScript. You don't write. That JavaScript occasionally can insert ransomware, the Angler exploit kit. It has happened on New York Times in March 2016. You hold copyright on ransomware. That's very interesting. I didn't know that New York Times wanted that copyright and liability. Of course, they say, "No. No. No. That's not us. That was a mistake. We're not liable. No one was harmed. We never heard about that," but it happened.

It shows the fallacy of treating digital ads like they're ink on paper. Now, if 100 years ago, in the era of Jude Wanamaker, I went up to your grandmother's door and I took the Sunday Times and I secretively sort of did paste off of my own ads on top of the ink on paper ads that were on that newspaper page and then tried to get some revenue from doing that fraud of ad replacement, that would be immoral and that would be illegal, I'm sure. It would be, at least, something called tortious interference in the business as New York Times. I would be trespassing on grandmother's doorstep.

That's not how digital ads work. They are not ink paper. They are not DRM pixels in video, they are loaded separately from different servers. The think that loads them is a script written not by the New York Times, but not even by its direct contracted ad partners. Sometimes the script is written by somebody in Russia, seven degrees of separation away from New York Times.

Do not confuse the essential differences here. There is a system people are used to thinking about which is very visible and even tactile, which is ink on paper ads. There is the digital world of scripted third party ads. Completely different, completely unsafe, subject to parasites, malware vendors, basically, [inaudible 1:06:54.9] who take too much money out in these middle fees that they extract while they let anything through.

Because they extract fees on anything that goes through and on any clicks that happen on ads, they have perverse incentives. They will let malware through, because, "Hey, they got a fee," whether it's a good ad or malware. If there's a click that was done by a robot to steal ad revenue, because you have that kind of fraud too. You have actual fake users and fake publisher sites being set up as a meth bot to steal revenue. The middle player still makes the fee. What do they know, "Hey, pay me. It's all good."

There's a conflict of interest again, and that's something that we definitely are targeting at Brave, because nobody works for free. There's no free lunch. Somebody has to lose if there's going to be a better internet advertising system, and I think that's the 2000 or so companies that parasitically infest the middle of the AdTech system.

I would say, also, Google and Facebook have to shape up. They don't get off free, and they are heavy trackers, especially Google, and we'll have to shake that up.

[SPONSOR MESSAGE]

[1:08:04.1] JM: Good customer relationships define the success of your business. Zendesk helps you build better mobile apps and retain users. With Zendesk mobile SDKs, you can bring native in-app support to your app quickly and easily. If a user discovers a bug in your app, that user can view help content and start a conversation with your support team without leaving your app.

The conversations go into Zendesk and can automatically include information about the user's app information, device information, usage history, and more. Best of all, this is included with Zendesk for no extra charge. Use the out of the box iOS UI to get up and running quickly, or build your own UI and work with the SDK API providers. Keep your customers happy with Zendesk.

Software Engineering Daily listeners can use promo code sedaily for $177 off. Thanks to Zendesk for supporting Software Engineering Daily, and you can check out zendesk.com/sedaily to support Software Engineering Daily and get $177 off your Zendesk.

[INTERVIEW]

[1:09:26.1] CM: Brave is built on Chromium, a browser engine developed mostly by Google. Are there any concerns around using Google's browser technology to create a competing browser?

[1:09:36.2] BE: Google is dominant with Chrome. It's not going to reach the Internet Explorer level of 95% market share that you read about in Wikipedia. Some say 94, some say 96. The way that browser market share was measured has changed and nobody is sure, but it was high.

95% is pretty much monopoly. By the time standard oil was subject to anti-trust action back in the Gilded age, it was actually in the 20th century, they were down to 70%. They've already declined. Unfortunately, I think, Google getting to 80% or 85% with Chrome wouldn't get them anti-trust attention in the U.S.

In Europe, it already has and things about Android and search have gotten them in the anti-trust regulator sites in Europe. My point in all of these is that Chromium, which is built around Blink, which is a fork of WebKit. You'll still find WebKit references all over the source code. Apple's engine that was forked from KHTML.

Chromium Blink is dominant. If you want to compete as a browser you pretty much have to use that code. I say this as a founder of Mozilla. We had our own engine. Mozilla has it still, called Gecko. They're slowly adding the compatibility that's needed for not just standards that evolve, but the de facto standards that were set on the mobile web by the iPhone, by WebKit in iOS in 2007 and on. Those standards — Some of them were quite brilliant innovations for mobile, touch response, and rounded corners. Some of them were in the web standards. Some of them were still coming along.

What's worse — I think most vexing, but part of the nature of the web. There's old web content out there that just assumes it was the iPhone and uses sort of a WebKit prefix on the CSS property, Slack, or whatever. Those de facto standards need to be engineered into any new engine that has to compete. Now, there will be new engines up and coming.

There the Servo engine, which I was actually a sponsor of in Mozilla research at Mozilla. It's coming along and it's got some amazing innovative components. It has web render component that uses seven shader programs on the GPU to render all of CSS. It decomposes CSS rendering into those seven shader programs and composes them in parallel for maximum speed up.

There's also a font renderer that can render sort of rasterize font cliffs, font vectors and do it on GPU again in parallel and safely using Rust and shader programs. There's just some amazing work there, but Servo is not web compatible enough, otherwise I'd be using it in Brave. I'm using Chromium.

On iOS, the rules Athlon poses on developers require you to use their versions of web engines which are either the old one, UIWebView, or the new one, WKWebView, the Safari WebView, and that's the WebKit engine that Apple is attending from which Chromium Blink forked Blink.

Really, you don't have much choice right no due to this market structure. The winners of the two iterations of the browsers wars, Chrome kind of won on desktop through Google's wealth on market power and distribution power, and Microsoft sort of subsiding from the force, it used to be on the PC and the PC era. Apple, of course, very strong at the high-end of the smartphone market. Google Android on the smartphone market helped too.

Chrome is number one, 80 something percent, maybe 70 something percent. The iPhone is strong and it's influential. I carry one and a lot of people do. Invaluable market, so Apple can kind of control the ball a little bit in web standards when they choose to and not let the Google run the show. This really drives Google crazy, I know. Some of them used to be in Microsoft. Some of the know from that era, Microsoft felt it could do anything it wanted.

In IE 4, when Microsoft got the upper hand over Netscape where it was still working, like watching a train wreck in slow motion. Couldn't do a thing about it, kind of Netscape had gone off on an acquisition bender with the mad money it made from its IPO, and acquiring a bunch of companies never worked, just as Yahoo.

Microsoft did an incredible job at least on Windows with IE 4. It was not very secure with ActiveX all over the place, but they really did kind of embrace the JavaScript idea that I had in Netscape 2. They elaborated my work to create the document [inaudible] model into something they called DHTML, and they did kind of own the web standards. They made friends with W3C in that era, in '96, '97.

Google is trying to control the ball, as if they had 95% market share, but they don't. They do control a lot of the ball, and maybe they should. Again, there this sort of conflict, macroeconomic conflict of interest against their users, because they're an advertising funded business who will not put ad blocking into Chrome by default, not easily or lightly. Maybe I can get them to sort of help standardize stuff from Brave. That will be good.

There's a new market structure and a new duopoly just like we saw — I had spoken earlier with AdTech, with Facebook, and Google, there is this sort of Apple-Google duopoly. Microsoft's browsers, Edge and IE as a pair, are slowly losing market share last I looked. It's still

happening, even though they've worked hard in Edge and they're very proud of it, and they cleaned up a lot of the code and got rid of all the ActiveX letter junk.

It's just kind of too bad that there was a new browser engine coming in. I would like to Servo get there. It needs a product, and it needs a tip of the sphere market in which to get that product to users. Maybe it's VRAR, maybe it's something outside of just the browser model where you face the huge billion plus website array of content. You have to be compatible with including its WebKit of Chrome only versions on mobile.

Then, if Servo can't be compatible with that, it's just not going to get into those mature product categories. It might get into a new product. It might make it through other means. They will get more machine learning to engineer compatibility linear code, write our compatibility code for us.

I'm a technological optimist, so there will be a new image in some say and it will replace the sort of WebKit lineage or sort of supersede it somehow. Like I said, perhaps, there's some machine learning. That would be cool, and I look forward to it. I just can't bet the farm on it with Brave. It almost doesn't matter. The web is — Google is doing all right with some of its innovations. We had to teach them not to waste time on Dart and portable native client when I was at Mozilla. Got them to see the light with web assembly after asm.js.

JavaScript is evolving, Google is participating. The service worker work at Google is good and helps you have an offline model and it's a sort of way of intercepting network requests locally, so you can do things that were impossible before, do offline, do smart apps that act more like native apps on mobile. That's all good.

The real innovation, I think, has to come in serving users in more of this decentralized web mission we spoke to earlier, where the user isn't just an attention farm animal, his attention to being harvested and degraded. By the way, people talk about attention as a commodity, but it's actually scarce resource. It's information that's too plentiful and it degrades — You need to resynthesize dopamine or you get tired, you get blindness to banner ads.

I think the innovation in the engine space will happen, but it needs a hot new product like we have with Firefox when we got Gecko to market. By the way, taking Gecko into Firefox in 2003

and 4, Firebird Phoenix. Firefox in 2004 took off and it had web compatibility even though it had the Gecko engine, because Netscape had been so powerful. A lot of the web was still feature testing for, is it Netscape or IE. They would say, “If document.all,” and then they would assume it’s IE else, and they would write for Netscape.

Other new engines like WebKit, or KHTML in Safari in 2002 sort of drafting off the else clause that it was Netscape based content. That lineage, that sort of patrimony of Netscape content helped Gecko succeed in Firefox. Firefox got to 27% market share in 2011 I believe, it peaked, and then it fell to Chrome. It’s maybe stable, but still kind of low compared to that.

You have to get something new up there that users get, and users don’t really understand a little tweak in HTML or a little difference in JavaScript. Even WebAssembly, like you asked earlier, might be a point of difference if some browsers are really slow or just doesn’t do it. If they’re all good at it and it’s been designed to have a deterministic performance model, then WebAssembly won’t be the differentiator. It will be something with higher order.

With Brave, we’re blocking not just ads, but the invisible tracking scripts. Mobile saves you half your data plan. It speeds up on the benchmarks we’ve shown. I’ve tweeted them recently, three to seven times against Chrome on Android and three to eight times against iOS Safari. There is a higher level of user value there that you can sell without having to get down into the WebAssembly details.

[1:18:39.7] CM: Okay. Brave blocks the accumulation of data, and for some companies, this is blocking their accumulation of power. As in the information age, data translates to power through data science processes such as the training of advanced artificial intelligence algorithms. Does it hurt societal progress to keep data away from these companies who want to train algorithms from that data?

[1:19:02.7] BE: Yeah, I don’t agonize about this one, because, first of all, we leak data all the time. It’s very hard to enforce perfect confidentially. Even when you encrypt something, there are side channels, timing channels. Fingerprinting is endlessly innovating to use different measures, different bits of entry that you leak.

With Brave, the fingerprinting defense we do, which [inaudible 1:19:26.6] has been working on is still opt-in by site. It protects against canvas fingerprinting, web audio fingerprinting, WebGL fingerprinting, batter status fingerprinting. It's worked on some sort of clever sort of HSTS fingerprinting, so-called. [inaudible 1:19:45.2] clever thing called Sniffly, you can look up. There are a lots of ways to fingerprint.

What matters is the ones that are used in practice at a large scale, because the so-called data science or the AdTech companies that claim they make societally beneficial use of data don't have the wits to do their own custom fingerprinting. They use script. There's a fingerprint to .js script and it's commonly used. That's what you want to block.

In spite of all that, as I say, people leak all the time. People give up data voluntarily. I mentioned earlier, decentralization isn't for everything. We have trusted relationships in our lives. We will give up data, and we probably should give up data, in some cases, to a legitimate authority for a legitimate value exchange. The problem with giving up data on just browsing the web to trackers who promised to get better ads to publishers is; A, they haven't done that. The ads have gotten worse. B, they promised better yield, that is revenue per ad slot for publishers. That's gotten worse. Publishers are going out of business.

If the middle players are actually parasites whispering in your ear, "It's good when I eat some of the food you ate." But they're actually making the host wither and die. That's not societally beneficial. That's not good data. That's not voluntary, really.

This is the other point I think Doc Searls made in his Medium post that I mentioned earlier that when you go to a department store and you see a flier for something for sale, the flier doesn't go fly to your car, stick to your windshield, follow you home, stick to your pants as you walk in your house. Attach itself to your wall and install a spy cam. That's what the trackers do in the current AdTech system. That is not voluntary.

In fact, there's an open question, some privacy advocates are arguing the European emerging privacy regulations, E-Privacy in the GDPR, or the General Data Protection Regulation, require that users must consent to any kind of tracking. It cannot be done invisibly without consent. It cannot be done under duress where they say — The publisher says, "You must turn off your ad

block or tracking protection extension in order to view my content that otherwise is freely available. In fact, it's not behind the pay-wall." That would be duress. That would violate consent.

We talked about moral philosophy earlier. Here is where it hits reality. You're being tracked against your will. It's not helping the publishers. It's not even helping the advertisers. There's a lot of fraud, a lot of waste. Who is it helping? It's helping a bunch of [inaudible 1:22:08.9] and parasites. Let's fix that problem and then let's talk about the [inaudible 1:22:14.3] value of giving out data.

I said it's inevitable that any network who'll get the first and second place, [inaudible 1:22:21.1] optimal winners, 80-20 or the 75-15, and then the 10% is a bunch of little guys struggling to be the next big thing. It's stable for a while. The problem is monopolies tend to buy politicians and make themselves 100 year institutions by corrupting politics. That's a problem I don't propose to solve. I don't quite see that happening with ads, so we've been called un-American at Brave. When I was at Mozilla and we talked about tracking protection, the Interactive Advertising Bureau started saying, "It's un-American." Sorry. I don't buy it. Let me, as an American, defend my data. You could have if I think I'm getting value for it. I have to see who I'm dealing with. I have to know that I can trust them. they can give it out to their cousin in Russia.

[1:23:05.2] CM: All right. Thank you so much. This was a great interview. Thanks for coming onto Software Engineering Daily.

[1:23:09.8] BE: Welcome. A lot of fun. Thanks.

[END OF INTERVIEW]

[1:23:15.5] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning from each other. Check it out at symphono.com/sedaily. That's symphono.com/sedaily. Thanks again Symphono.

[END]