Javascript Jabber

124 JSJ The Origin of Javascript with Brendan Eich

JOE:

Do you have a phone in your shoe?

BRENDAN:

...No.

[This episode is sponsored by Frontend Masters. They have a terrific lineup of live courses you can attend either online or in person. They also have a terrific backlog of courses you can watch including JavaScript the Good Parts, Build Web Applications with Node.js, AngularJS In-Depth, and Advanced JavaScript. You can go check them out at FrontEndMasters.com.]

[This episode is sponsored by Codeship.io. Don't you wish you could simply deploy your code every time your tests pass? Wouldn't it be nice if it were tied into a nice continuous integration system? That's Codeship. They run your code. If all your tests pass, they deploy your code automatically. For fuss-free continuous delivery, check them out at Codeship.io, continuous delivery made simple.]

[This episode is sponsored by WatchMeCode. Have you been looking for regular high-quality video screencasts on building JavaScript done by someone who really understands JavaScript? Derick Bailey's videos cover many of the topics we talk about on JavaScript Jabber and are up on the latest tools and tricks you need to write great JavaScript. He also covers language fundamentals, so there's plenty for everybody. Looking over the catalogue, I got really excited and I can't wait to watch them all. Go check them out at JavaScriptJabber.com/WatchMeCode.]

[This episode is sponsored by Component One, makers of Wijmo. If you need stunning UI elements or awesome graphs and charts, then go to Wijmo.com and check them out.]

CHUCK:

Hey everybody and welcome to episode 124 of the JavaScript Jabber Show. This week on our panel, we have Joe Eames.

JOE:

Hey, everyone.

CHUCK:

Aaron Frost.

AARON:

Hello.

CHUCK:

AJ O'Neal.

AJ:

Yo, yo, yo, coming at you live from Boston.

CHUCK:

Jamison Dance.

JAMISON:

Hey, friends.

CHUCK:

Tim Caswell.

TIM:

Hello.

CHUCK:

I'm Charles Max Wood from DevChat.TV. And this week we have a special guest, Brendan Eich.

BRENDAN:

Hello.

CHUCK:

Do you want to introduce yourself real quick for us, Brendan?

BRENDAN:

Hi. I perpetrated JavaScript in 1995. And I've been making up for it ever since.

[Laughter]

JOE:

Awesome.

AARON:

Still perpetuating the lie.

JAMISON:

[Chuckles]

AARON:

That's awesome.

CHUCK:

Awesome. So, we brought you on to talk about where JavaScript came from. I'm curious to know just for my own edification. I have clients and others who ask me all the time, "So, you do Java?" And so, I want to know why it was called JavaScript [chuckles] for one. But yeah, maybe you can give us a little bit of backstory on JavaScript.

JOE:

Yeah, that whole story I think is totally worth telling, even though I'm sure you've told it a hundred times. I know that I've been doing JavaScript for quite a while before I finally heard the story of how JavaScript was created. So, I'd love to hear it again.

CHUCK:

[Imitating old man's voice] Once upon a time…

BRENDAN:

Yeah, you guys should cut me off because it will go on and on. So, the thing you have to know about Netscape is it was a Jim Clark, Marc Andreessen joint. So, it was basically the union of NCSA Mosaic principals plus Lou Montulli from University of Kansas who did the Lynx browser. I think that was spelled L-Y-N-X, which is a text-based browser. But everybody else at Netscape on the first floor was either from NCSA Mosaic or NCSA HTTPd. They were all at the National Center for Supercomputing Applications at my alma mater, University of Illinois at Urbana-Champaign. So, Marc Andreessen recruited there heavily. And he paired up with Jim Clark, the founder of Silicon Graphics.

So, I was at Silicon Graphics out of grad school in 1985, which was great. It was pre-IPO. It was a hot valley company. This was back when you had technical Unix workstation companies building their own basically CPUs or building their own CPU boards around the Motorola 68020 or 68030 chip. And Sun and other companies have licensed the Stanford University Network Sun-1 architecture. And Andreas Bechtolsheim had given up his PhD to go found Sun with Bill Joy and others. It was a pretty awesome time in the valley. It was before the PC. The PC was out. There was the 8080 or 8086. There was IBM PC but it was kind of a joke. So, for real industrial computing you needed a workstation or a Minicomputer even. There were still minicomputers around then. Digital Equipment Corporation was a thing then.

And SGI was turned into a company from a Stanford research project Jim Clark was the professor of, which was building basically what became the GPU. They were building the Graphics Processing Unit as a whole graphics board using VLSI technology. Carver Mead at Cal Tech had written the book. People could make lots of transistors on a single silicondie. And they could build something that was really good at doing lots of graphics operations in parallel. And that's where Silicon Graphics came from.

But by the time the 90s came along, Clark was I think kind of squeezed out of management politics at SGI. He was the chairman but not otherwise empowered. He was annoyed. So, he wanted to do something new. And I'm not sure exactly how, but he got introduced to Marc Andreessen in NCSA. There might have been some venture capitalists involved there. And they hit it off and they thought about doing something, which became Netscape. And the weirdest thing was they went through various ideas. I only know of one that I heard about, they were serious about for a few days or weeks which was, let's build Nintendo 64 software for modem-connected N64 boxes. And that wasn't…

[Chuckles]

BRENDAN:

Looking too good after a few days. So, they decided, no, let's go make the internet commercially [viable]. Let's kill Mosaic by making a Mosaic killer browser which will be the killer app, which will actually have security for commercial e-commerce. And that's what they did. That was Netscape1.0, 1.1. They did things like SSL. They did kill Mosaic. They took all its market share. At first they were called M Com, not Netscape, Mosaic Communications. And I think NCSA's lawyers came calling and there was a rapid rename to Netscape. And of course when they were founded, because they had Clark as cofounder, he drew from Silicon Graphics' early talents. So he drew Tom Paquin to manage engineering and Kipp Hickman who was my senior partner when I joined SGI out of grad school as a kernel hacker.

And Kipp called me up while I was at MicroUnity Systems Engineering, which is a crazy company I'll tell you about later. And Kipp said, "Do you want to come join us? It'll be fun. We're going to do a Mosaic killer." And I said, "Oh, I've still got some things to finish at MicroUnity." So, I stayed a year like an idiot and missed out being on the first floor. So, I joined in April 1995 as year into Netscape. When I came onboard, Netscape 1.1 was heading toward release.

And because of some weird financial shenanigans, they couldn't hire me into the group they wanted to hire me into. They had tempted me there, all these Silicon Graphics people. I knew Kipp, and John Giannandrea. They said, “Come and do Scheme in the browser. We need a programming language in the browser. Come and do Scheme.” And Scheme was a language that I only learned through book learning, through Sussman and Steele’s SICP, famous book, ‘Structure and Interpretation of Computer Programs’.

AARON:

Did you like it though? Did you go, “I like Scheme. I’ll go do that”?

BRENDAN:

Yeah, I said [chuckles] big idiot me, I said, “I like Scheme. I’ll go do that.” And of course, when I got there the answer was, “Oh, well wait. We’re doing a deal with Sun. And they have something called Java, renamed from Oak. And we can’t really let you do Scheme in the browser now.” [Chuckles] Plus because of financial machinations I don’t fully understand to this day, they hired me into the server team instead of the client team. [Laughs] So, that was weird. And they ended up having me work on HTTP 1.1 or what we thought would be HTTP 1.1. Way back then in April 1995 it had a lot in common with SPDY.

And I was working with the McCool twins, Rob McCool and Mike McCool who wrote NCSA HTTPd which then forked can do Apache. And they were fun to work with, and Ari Luotonen who worked on the proxy server that Netscape product-ized. So, for a month I was screwing around with server-side stuff but I was thinking about what to do to rescue the idea of a language for the browser, which wouldn’t be Scheme. And so, in May I got switched to the client team finally. They got, a headcount opened. Even in a company of 150 people, they had headcount shortages and little games where people had to trade requisitions.

So, I ended up in May starting hitting the ground and running, really moving fast to create JavaScript. And it was codenamed Mocha. I wish I kept a diary, but I could tell you that involved intense meetings with people like Marc Andreessen, not only at Netscape but also at the Peninsula Creamery in Palo Alto where they had greasy burgers, fries, milkshakes. And at that point Marc, he was a big boy. [Chuckles] We used to go there and get, he used to like Midwest food.

[Laughter]

BRENDAN:

He’d get a milkshake. He’d drink milk at work. There were a lot of junk calories being consumed. And Netscape also had something I was horrified to learn about that has become standard in the valley, which is the sleeping room. So, Tom Paquin, the engineering first floor manager, would take the sheets home every other day to get cleaned, which was good.

CHUCK:

Oh no.

BRENDAN:

[Laughs] But people were actually…

AARON:

That sounds horrible.

BRENDAN:

Sleeping at work. And I was actually working around the clock, because I realized in order to do what became JavaScript, I had to not only show that we could make a language work in the browser, I had to show that it wasn’t redundant with respect to Java, which was the big deal with

Sun that was brewing that led to this whole, “Oh sorry. We didn’t really mean Scheme in the browser. We’re doing something with Java.”

And the best I could pull off was, okay so you have a language, Java, which is compiled. You have to learn what a compiler is. You have to run Java C. You have to write Java source code which consists of classes including a class with a main method, a static method. And you have to compile that into bytecode. Right there, you’ve lost a lot of people who could write programs if they were written in an interpreted language, if they were… in Microsoft’s stack at the time, they could use Visual Basic and they wouldn’t have to worry about C++. But with Netscape and Sun cobbling up this deal, it was only Java and it was this compiled language. It was kind of hard to use. It was typed. I’m not saying it was bad. It was just not for the casual amateur or beginner programmer.

AARON:

Can I do a timeout and ask you a question?

BRENDAN:

Yeah, absolutely.

AARON:

Did they come at you and say, “Brendan, you have a hard stop in 10 days.”

BRENDAN:

Yeah.

AARON:

Or did the 10 days things would just be like, “I worked on it for 10 days and we had the initial prototype”? Or did someone say to you, “You got 10 days, otherwise this is dead”? What’s the 10 days thing?

BRENDAN:

So, the thing that happened was, if you might gather from the story so far, they were trying to do a deal with Sun for Java. Sun thought Java should be the only language. They realized that it was too hard to use for amateurs and beginner programmers. They needed a language like Visual Basic in Microsoft’s stack at the time. They needed JavaScript. And Marc Andreessen saw this clearly. So, he and I were partnered up. That’s why we would meet up at the Peninsula Creamery in secret and we would plot our own destiny independent of whatever Sun was going to do with Java. And if we hadn’t done that, there wouldn’t be any easy-to-use language that you could embed right in the HTML.

I remember Marc specifically said, “No. You’ve got to be able to write the language, the source code, right in the HTML. You’ve got to be able to write it directly in there inside some kind of container tag.” And so, I went off and figured out how to make that work, which was horrendous because HTML, supposedly based on SGML but its own grammar, doesn’t have the ability to embed a less than operator in the middle of a tag unless you use a special content model for that element. And so, I figured all that out. I made the script tag. I made it all work.

But it was really a rush, not 10 days that reckoned, but pretty damn, quite fast in order to prove not only to Netscape but to Sun, and there were people in Sun and Netscape who doubted, that there was a value in having an easy-to-use language you could embed directly in HTML that amateurs could write. And that it could be done quickly enough that it could be shipped in the same release that Java was shipping in. And that release was Netscape 2.

CHUCK:

So, we could have wound up with Java? I love you even more now. That's all I have to say.

JOE:

Yeah.

BRENDAN:

If it was only Java. The fact is, Java bombed in the client. It took forever to die, but it's pretty much dead. It's a source of malware.

[Chuckles]

BRENDAN:

Chrome and Firefox blacklisted it. It's gone.

AARON:

So, it wasn't called JavaScript. Was it called LiveScript?

BRENDAN:

Originally, Marc wanted to call it Mocha. And Marc actually [chuckles] I'll tell you something I haven't told anybody. So, Marc was actually not sure about Sun, so he thought, "Maybe we should do our own Java VM." So, Kipp Hickman, my friend from early days at SGI who was a senior kernel hacker when I hired, he started writing his own JVM. And this was before we had done a source license for Sun's Java Virtual Machine. So, he started writing his own. He was trying to self-host the Java C compiler that Arthur van Hoff wrote, which is written in Java, a very nice compiler written in Java. So, if Kipp's VM could run it, we could conceivably do our own Netscape VM. It wouldn't be the Sun VM. It would run Java. And then we'd be independent of Sun.

And Marc was so ambitious he thought, "Okay. We'll have Kipp's JVM. We'll have your JavaScript VM. Now, we need a graphics library, like for canvas, for 2D and 3D graphics. Who can we get to do that?" And I said, "Uh, I worked with somebody really smart at SGI. I'd like to hire him, but he's at MIT being a grad student," and that's Andrew Myers who's now a professor at Cornell. And so, Marc's like "Great. We'll hire him. Throw stock at him." [Chuckles] So, there was this grand plan which never came to fruition where we would have an entirely Netscape-created codebase. It would be Kipp's Java VM, my JavaScript VM, and Andrew Myers', who we never hired, graphics library. And so, that lasted a few weeks.

Also when the rubber hit the road, what we had to ship was the JavaScript VM wedged into Netscape 2 in a way that actually embedded in HTML you could have a script tag with inline content that was code, including the less than operator. You could then access document.links sub zero, or document.forms sub one, or document.forms.myform. And so, I created this primitive version of the DOM that became known as the DOM level 0, all in those early days.

And the worst part of it was that Netscape the browser, which was written the previous year in 1994 in six months as Jamie Zawinski has described "in order to kill Mosaic fast" was not designed to have a scripting language embedded into it. Because when you write JavaScript you can extend the lifetime of a form. You can make a reference to document.forms sub zero and suddenly you can keep that alive longer than the document. Well, what happens inside the old Netscape browser code was as soon as you navigated away from that page, they freed all that memory. [Chuckles] They just wiped it out, returned it to the C malloc heap, and if you kept pointers to it you were in big trouble as far as vulnerability to crashes, security bugs, all sorts of things.

So, I was juggling with chainsaws trying to embed this arbitrary lifetime scripting language into a mandatory lifetime just so lifetime C codebase. And that meant Netscape 2 was a nightmare of bugs. But in spite of that, to finish my statement before I run on too long, there were people using it even in the beta releases who were doing amazing demos. They would write forms inside tables and they would write spreadsheets or calculators or various form-based applications, even what we might call single-page applications today. They didn't have XHR but they could hide a form element or a link that they would auto-submit or auto-click inside a hidden frame. And they made it all work.

And I had to help them on a workaround because there were tons of bugs. And they were like, "I have a demo next Monday and I'm going to get my contract cancelled if I can't demo. Can you help me?" And I said, "Okay." I figured out a workaround and I gave it to them and they delivered it. And that helped JavaScript prosper.

JAMISON:

So, it sounds like, did you have a vision of the potential of JavaScript? It seems like in the last five years, well more than that, maybe the last 10 years, it started to come of age more and become more widely recognized as a platform for building mature applications. Did you always hope that would happen and think that would happen? Or did it happen as a surprise to you?

BRENDAN:

So yeah, you're talking about Web 2.0 or the Ajax revolution?

JAMISON:

Yeah, yeah.

BRENDAN:

2004 or 2005?

JAMISON:

But some of the stuff you describe sounds like that happened in the early days, too.

BRENDAN:

It did. In 1995, early adopters, and I can't claim sole credit for this. A guy named Bill Dortch who's still online, he had a company called Hidaho design, like Idaho with an H in front. And he built all these essentially single-page applications using frames and framesets, hidden forms or links. He even wrote image generation using the XBM image format, which is the X Bitmap format, which is an anti-compressed image format. It uses a subset of the C programming language, including the C preprocessor, to specify a char array initializer containing the bits of the image. [Chuckles] It's a crazy format. And he would generate this from JavaScript and load it using the right MIME type into image elements in the DOM, the primitive DOM level 0. And suddenly, you get these black and white drawings he could create. It's like an early canvas.

So, this was all '95, '96, dodging lots of bugs. One of the bugs was funny. It was HTML tables, as you might know, can nest. And so, Eric Bina who I like a lot, I really worked well with him, he was Marc Andreessen's coding partner. He was the real programmer on NCSA Mosaic and then he joined Netscape. His layout engine was very brute force about tables. It would basically do a table layout by laying out its contents, measuring the heights that you got from the widths, measuring how much space it needed, and then laying out the parent table. And so, if you nest tables n levels deep, you get 2^n passes to do all this measurement.

And so, when I integrated JavaScript into that, I didn't realize this. So, I started saying, "Okay, I'm going to put a callback here in the middle of Eric's layout routine that creates a form element. And I'm going to create a JavaScript peer object to match that element. It will be the peer of that internal C data structure." Unfortunately, because he was doing table layout repeatedly to measure things, I get 2^n of these things. But I also realized because HTML at the time had no restriction on the name element, you could have things named "foo" three or five or a hundred times in the same document, that I would automatically array them.

So, if you said one “foo” you get a singleton. If you said two, you an array of two elements named “foo”. And three would just append to that array. And so, totally accidentally his layout, multi-pass 2^n characteristic, interacted with my JavaScript automatic arraying to create a form element that was basically an array of individual form elements. Only the last one was the real one. The rest were all throwaways used for measuring the table dimensions.

And so, this particular contractor I mentioned, he was saying, “I’ve got a demo on Monday. I’m going to get fired if I can’t do it,” he couldn’t make the element come to life. And I said, “Oh, you’re using the first element or you’re using the array. That’s not the one to use. You want to actually drill down one level deeper and go to length-1 element. That’s the real element.” And when he did that and suddenly it started responding and showing that it was the real element, you could actually change its attributes and have effects on the presentation, he was overjoyed and so was I. So, there’s a real kick in helping people get workarounds to these things.

[Chuckles]

JOE:

That’s funny.

AJ:

That’s awesome.

AARON:

That’s awesome.

JAMISON:

Did any of these workarounds become enshrined into the language and then you regretted them later?

BRENDAN:

Yes.

[Laughter]

BRENDAN:

Many of them.

[Laughter]

BRENDAN:

You can’t avoid some of the sort of DOM level 0 stuff is codified in HTML 5 thanks to the work in the WHATWG that we started with Apple and Opera when I was at Mozilla in 2004. And some of it has been warped through the IE4 lens. But a lot of the early stuff’s there, like the name attribute instead of the id attribute we all know from more modern HTML came via XML. The name attribute is kind of funky and you can get automatic arraying of things, like radio buttons automatically array. And I’m not sure if the old property that all form elements automatically become arrays if there’s [inaudible] instances with the same name, is still there. But that was there then and that was part of the workaround I mentioned.

The vision I had that was bigger, I had maybe a few weeks while I was still on the server team thinking ahead trying to plan. Because I knew I was going to transfer to the client team. And I thought, “How could I make JavaScript and demonstrate its value against Java and get it into the browser really fast?” And I thought, “How can I make it easy to use for beginners?” because that was the big pitch that Marc Andreessen and I, also Bill Joy of Sun Microsystems who signed the trademark license in December 4th I believe, that gave Netscape the right to call it JavaScript instead of LiveScript. The Mocha name was eventually contested because there's software that uses Mocha in some sort of name. And so, Netscape chickened out on Mocha.

And they hired somebody who liked LiveScript. And that was a lame name in my opinion. And so, what they really wanted was to call it JavaScript. And they finally got the license through Bill Joy. And what Bill Joy and Marc Andreessen and I all saw was you have to make this really simple for beginners. And so, did you even want it to look like Java which is a C-like language? I started looking at languages like Logo and Smalltalk and Self and HyperTalk which was Bill Atkinson's language for HyperCard.

JOE:

What about Visual Basic?

BRENDAN:

Visual Basic, no.

[Laughter]

BRENDAN:

I knew Basic from when I was a teenager and my friend had a Commodore PET and he wrote a Star Wars trench game. It was a 2D vector graphics trench game where you're flying down Death Star trench trying to shoot at the exhaust port. And I knew Basic, but I didn't think that would work in the modern era. And I was attracted to Self in particular because it was taking ideas in Smalltalk and simplifying them, trying to minimize the number of concepts, and then maximize their utility. And that was David Ungar and company. And it was good work.

They also did optimizing [inaudible] for it, which they founded a company out of Stanford about called Animorphic. And actually, a few years later Netscape, on loan to Sun, did some due diligence when Sun acquired Animorphic. And Animorphic had Lars Bak and others as the young programmers doing all the heavy lifting for the senior people like Dave Ungar and Craig Chambers on what became the Self with classes, jitting VM and the Strongtalk VM which was Smalltalk with types. Gilad Bracha did the work on the type system. Eventually that open sourced. And that led to V8, if you can believe it or not. From the names I've dropped, you probably can believe it.

So, there's a lineage here that goes from JavaScript through Java because Lars went to Sun and worked, Lars Bak went to Sun and worked on the Hotspot VM, to JavaScript and V8, and now to Dart. But long story short, there was an idea in JavaScript that I was pursuing, and maybe a few others saw it, too, of a language that wasn't C-like. It was easy to use. It was meant for people who were building things inductively. They were learning programming for the first time. And they didn't necessarily have to know where semicolons had to go, or even curly braces. I lost the curly brace front. On the semicolon front I said, "It's ridiculous to reject a program because of a missing semicolon. We should do some kind of error correction procedure." So, I made one up on the spot and that became automatic semicolon push.

[Laughter]

CHUCK:

I kind of hacked this in and it stayed.

BRENDAN:

Yup.

AARON:

That's awesome.

JOE:

And 15 years later, people are still fighting over it.

JAMISON:

That's every programmer's nightmare, right?

CHUCK:

It's so true.

JAMISON:

Just the hack [inaudible] that you made.

AJ:

I felt a sudden tremor in the force as if a thousand semicolons suddenly screwed up everything.

[Laughter]

BRENDAN:

Yeah, there are definitely some issues there. But in fact, I remember Jamie Zawinski was writing some JavaScript and he had a long return expression so he put it on the next line with no semicolon after the return. And he was outraged that ASI would insert a semicolon after the return making the 'return' return the undefined value and the next line becomes dead code, a useless expression. It's unreachable in the control flow of that function. He was totally outraged. [Laughs] But I said, "It's too late. I can't change it." Once you ship things on the web, it's very hard to change it.

AARON:

Yeah. So, I want to skip up and talk about, so you got this thing going. It's JavaScript. It's successful. How many other people adopted it before you guys decided to give it to Ecma and let Ecma standardize it?

BRENDAN:

Netscape was the hottest name and the name that put the browser and things like URLs or the web on the map. It wasn't Mosaic. We killed Mosaic and then we were the hot thing for a year and a half. And during that time it was awesome because Microsoft was behind. Bill Gates realized they made a mistake. They were doing an AOL killer called Blackburn, which was a proprietary content system for a dial-up network.

And when Gates saw what was happening with Netscape he said, "We're going to lose unless we jump on this and embrace, extend, and extinguish it." They actually tried to buy Netscape in late '94. And they offered a hundred million or something paltry. So, they were told to get lost. And that meant they went and cloned it. And everybody knew it. When I joined Netscape in April '95 it was like, "We're doomed." People were just thinking Microsoft was inevitable going to kill the company. It was only a matter of time.

And at that point, they'd acquired Spyglass and they started digesting the source code and then creating IE. I don't know if there was an IE1. IE2, IE3 in 1996. IE4 was coming out in '97 in prereleases, preview releases. And that's when it got really good. So, there was a real concern about Microsoft killing Netscape fast. But before that got huge in late '96, they were putting moral pressure on Netscape. Bill Gates would say, "Netscape keeps changing the semantics of JavaScript," and I'm like, "Oh yeah, like you never do that to Visual Basic?"

[Laughter]

AARON:

Yeah.

CHUCK:

And like they didn't do it later on in IE?

BRENDAN:

If you follow Visual Basic, there's this crazy mistake where they blended it into .NET and they called Visual Thread or VB7 and it completely flopped. But even VB6 was incompatible and weirdly different from previous versions. And I was like, "Pot, kettle, black, hello."

[Chuckles]

BRENDAN:

But the pressure through developers was enough and the fact that they were coming, Mike Homer the marketing VP at Netscape said the monster truck was in our rearview mirror. Netscape's in its little Yugo, the pedal to the metal going, topping out at 50 miles an hour heading toward distant horizon. And behind us is the monster truck that's creeping up. And it's far away and it's a little dot and it's getting bigger. And suddenly, it's like its bumper, you can read the print on the bumper. And then you look ahead and you realize it isn't the horizon you're heading toward. It's a wall. This is all Mike Homer's metaphor. So, what do you do when you're heading toward this wall? Ultimately, the executives decided to do [27:24]. They said, hard left and hope for somebody to survive. Don't let the monster truck run you into the wall. Release the Netscape source code as open source.

But before that happened, that decision was late '97 leading into April 1st of '98, before that happened there was, "Let's standardize JavaScript." Okay, we don't like the moral pressure from Microsoft, hypocrites they are. We don't like being held to account for not standardizing the language. Let's take it somewhere. And so, this guy Netscape hired to be their standards guru named Carl Cargill, he said, "I know how to do this." He was friends with Jan van den Belt who was Secretary General of an outfit called ECMA, recently called the European Computer Manufacturer's Association, but they decided they wanted to be worldwide and the "European" was too limiting.

So, they say, "It's not an acronym. It's a proper noun. Capital E, lowercase C-M-A, Ecma." And so, they [chuckles] they started renaming themselves and saying, "Let's do software standards." And if you look on their sort of core agenda of standards at the time, it had things like Fortran standards and COBOL standards. These were wall long-standing ISO standards they embraced or wrapped up and repackaged. They also did a bunch of novel standards based on optical disks at the time. So, Philips was a big Ecma supporter. And getting into software with Netscape was good for them, so they went for it. Carl liked it because Microsoft didn't have any [inaudible] power with them.

And in fact Ecma, I think before Netscape did JavaScript through Ecma, Ecma had done the [inaudible] standardization of the Windows 3 API, Windows 16-bit API. Because the European governments, this was pre-EU, had demanded that that be standardized since they were all dependent on it for their infrastructure using PCs. They said, "We must have a standard for this." Microsoft wouldn't do it, so Ecma did it based on open knowledge, open docs and specs and man pages, whatever the equivalent Microsoft had was. And Microsoft actually sued them over that and lost, I think. So, Carl thought, "This is a great way to romp with Microsoft. We'll take JavaScript to Ecma.

[Laughter]

BRENDAN:

And Jan van den Belt was a [raconteur] and a very learned fellow. He's got a Belgian surname, or Dutch surname actually, van den Belt. Dutch surname, born in Belgium, lived in Geneva, knew everybody. We had great dinners out in Europe, I have to say that, when we went to standardize JavaScript in November '96 through June '97. When we finally finalized the standard in June, in was in Nice. And it was a good event. It had people from IBM, like Mike Cowlishaw, IBM fellow now retired. It had people from Borland, Microsoft, Netscape, other companies. Sun loaned Bill Joy, Bill Ga-, sorry, I'm saying Bill Joy but I mean to say Guy Steele, total CS god from the Scheme era. Guy Steele and Gerry Sussman did Scheme. So, I was in awe. It was great to work with them.

And Guy knew everything about things like floating point. In fact, if you wanted to do a floating point number to string and string to number conversions, the canonical paper still is one by Guy Steele and David M. Gay on this. And there's still some C code that everyone uses. It's ugly 70s C code that David Gay wrote. [Chuckles] But it's the code to use for converting accurately because it turns out to be very hard to convert decimal strings of digits after the decimal point to binary fixed-precision IEEE double. You have to carry extra precision around to round correctly. So, that code and that standard was very helpful in ECMAScript.

And Guy was there. And Guy even brought Richard Gabriel, a Lisp god, also a poet. He was associated with Stanford at the time. I saw him at OOPSLA a few years ago but I haven't kept up with Dick. But he was a lot of fun. He wrote Clause 3 of the Ecma standard, which is the intro. It tells you about what JavaScript's about, objects and functions. You know, objects without classes.

AARON:

Wow. So, AJ wants to know where the name Mozilla came from.

BRENDAN:

[Laughs]

AJ:

Yes, I do want to know that.

BRENDAN:

That's actually on the web. So, if you read Jamie Zawinski's many posts on his site, he named it based on the idea of a Mosaic killer. When Netscape realized they weren't going to do a Nintendo 64 modem networked software, they said, "Let's do a Mosaic killer," and Jamie started musing about Mosaic killer. Giant monster that kills Mosaic, Godzilla, Mozilla. So, that's where the name Mozilla came from.

AJ:

Oh, that's funny.

AARON:

That is cool.

BRENDAN:

I feel like I should explain more of the DOM level 0, like setTimeout. Wacky because people use the form that takes a function as the first argument. And then it's awkward because you put the timeout in milliseconds after that. And then if there are any arguments, actual arguments to the function, you pass them after the timeout. So, they're interrupted by that timeout value. The original version of setTimeout in Netscape 2 did not have the function argument. It only had a quoted string which was eval-ed. So, you wrote an expression to be evaluated and milliseconds. And in Netscape 3 we added the function form and at that point it was too late to reorder the arguments because you can't break the web. And so, that's why it's function, timeout, arguments. And you could still use the string form.

AJ:

I always thought that that was better anyway, to put the function first, because then it's composable.

BRENDAN:

It's a mixed bag. If you want to have something that applies or does partial application, you end up having to juggle both timeout and the arguments list. But you can do it.

JOE:

So, I have a question. Why did you make functions first-class citizens?

BRENDAN:

So that was [chuckles], people say JavaScript is powered by Self. There was very little Self in fact, because I had so little time. And Self has things like lexical scope, so you have no conflation of objects and scope ribs, which JavaScript had and has does to the global object and the width. But I did at least have the sense to make functions first class, which Self does, Lisp does. And I realized if I didn't do that, I didn't have much. I was in such a rush. It was ten days until the demo in front of the Engineering team. And I thought, "How am I going to do this and make it useful?" I had to lean on a very few concepts that were powerful. One was functions and the other was, and closures were only latently there in the very first version, the other was objects you could create ad hoc.

So, functions were huge. And not making them first-class meant I was going to have to lock them down and get it all right and make the global object and built-in functions and the map object, make that all perfect from the first release. And I knew there was no time to do that. I realized that I needed to allow people to mutate the environment. I needed to let everything be mutable in order to patch, monkey-patch, polyfill, prolyfill. I didn't see the whole evolution where people would actually anticipate standards by prolyfilling. But I did know that I could get it right and I had to leave it open. It is basically one bit of decision logic. Do I make it open and mutable, unsealed? Or do I make it closed, locked down, and sealed? And I said, if I make it closed [chuckles] I'm screwed. And in fact, Java's world was much more closed and that hurt them.

CHUCK:

So, you mentioned Java again. I just want to clarify because I'm not sure I got my answer out of your story.

BRENDAN:

[Chuckles]

CHUCK:

Did you call it JavaScript just because Java was popular and you partnered with Sun?

BRENDAN:

Like I said, Marc wanted to call it Mocha. I didn't care except I liked Mocha because it was different. I didn't know of any prior art but apparently there were software trademarks involving the name Mocha. But they were not related to a scripting language. So, Netscape could have probably fought for Mocha. When they hired the marketing guy who said, [inaudible] it's going to be LiveScript, we have LiveWire which is like a server PHP-like project that would allow you to do configuration management and simple database query-based apps, but [inaudible] really PHP. He wanted to call that LiveWire. So suddenly, everything was Live this and Live that. And I was throwing up in my mouth a little bit. I didn't like that name at all.

[Laughter]

BRENDAN:

So, when they came around in December and got Bill Joy to sign the trademark license for JavaScript, I said okay. I'm not sure this thing's going to make it because I was the only programmer working on it throughout '95. I was it. I had no support other than part-time help from the frontend folks who did the form and the presentational stuff that had to update when you have JavaScript poke at the DOM. And the DOM wasn't totally two-way. If you poked certain things, it wouldn't update the online presentation. You couldn't do text ranges. You couldn't do arbitrary layout. You had to poke at text in a text area or a text input. So, I was little worried JavaScript wouldn't make it and I thought if we're going to have some juice here, we might as well use JavaScript's juice. And then also, I liked the fact that Joy signed the trademark agreement because everybody else in Sun hated JavaScript and it really cheesed them off.

[Laughter]

AARON:

That's funny.

CHUCK:

Yeah, nowadays Sun, you mean Oracle, you mean, nah, never mind.

BRENDAN:

Some of the people there are still people working there I know from the old days, like John Rose who did Nashorn and made sure it invoked dynamic, which is what we would call a jitted, polymorphic call site instruction, got into the JVM. So, my hat's off to them. They're long old-timers who want to keep evolving the JVM. And in fact, the JVM's a great VM. It has lots of languages like Clojure and Scala. But it's a server-side VM. There's no way it could have survived in the client.

Sun thought they had a shot when they took, after they saw JavaScript getting standardized, they said, "Okay, we're going to take Java to Ecma." And this was '98 or something like that. They sat down with Guy Steele again. They ordered Guy to go do his duty and standardize Java. And they had Microsoft for a time and others sitting around the table. Sam Ruby at IBM was there. And Guy said, "Okay, we have one year. We have 7,000 pages. We have 39 seconds per page. We have to go now." [Chuckles] And so, we have to rubber stamp every page in the spec. And it didn't happen. That was not a good way to do it because there was too much to specify, not just the language but the VM and the memory model which wasn't even fully understood then. I think Hans Boehm at HP Labs had to help figure it out.

And meanwhile, Microsoft decided they didn't want to play ball with Sun. And so, at some point they yanked Java from Windows, if you guys remember the history. And that led to something we all know and love called the XML and HTTP request that because Outlook Web Access needed Java to do asynchronous XML HTTP requests, once Microsoft decided they were breaking up with Sun they were going to yank Java and there was a lawsuit around that, that they needed a replacement for the Java async I/O class. And so, they just hacked in XHR. That's how XHR came about. It was totally based on this fight between Microsoft and Sun.

JAMISON:

It's so crazy, because it's such a convoluted story.

JOE:

Yeah.

JAMISON:

You could imagine ways where XHR wouldn't exist and then the web would be totally different.

BRENDAN:

I'll tell you, having a hidden form or a link you click was a pain. XHR for all its funkiness and crap like the synchronous option, it's better than [chuckles] what was there before. But all of it's random. It's like evolutionary biology. You don't really get to choose. You just take what works and you're lucky if you survive.

JOE:

So, was XHR all of Microsoft's invention?

BRENDAN:

It was. But again, it was cloned, it was filling a vacuum that was left when they kicked Java out and they needed to keep Outlook Web Access working. And some of it had the flavor of Microsoft and Visual Basic informed API. Some of it was based pretty closely on the Java async I/O class that they booted.

JOE:

Did you ever look at this baby that you created and at certain points wonder if something was going to kill it off, like say Flash for example?

BRENDAN:

Oh, absolutely. [Chuckles] First of all, I was done working on, as the lone gun on it. I had some help. Ricardo Jenez who was working on some kind of Hadoop startup now with friends I know from Yahoo, like Raymie Stata. I forget the name of the company. But Ricardo came in to help me. He ported the David Gay floating point number to string and back code.

And early on, there was a guy named Ken Smith from Netscape acquired from Borland who did the date object based directly on JDK 1.0, the Java Developer Kit 1.0 version of java.util.Date. So, all that crap people hate about the date object, like months are numbered from 1, numbered from 1 in normal, human calendars, but are indexed from 0 in the java.util.Date API and therefore in JavaScript date object, people hate that. They Y2K bugs getYear instead of getFullYear on two-digit date offset from 1900, [chuckles] that was all from Java. And it was a Y2K bug. It was likein Office Space. [Chuckles] That came directly from Java. It was Ken Smith basically transliterating, transporting Java code implementing java.util.Date into JavaScript or into the C part of the engine that implemented the date object.

But other than those guys helping out on pieces, I had no help. Until late '96, one of the Netscape founders Chris Houck decided to join me and I felt pressure in joining the Ecma group to make the language cleaner. So, I started to rewrite the VM, pay off technical debt, and I wrote what became SpiderMonkey. The original Mocha VM was not SpiderMonkey. It was much slower. It was a rush job. When I rewrote it as SpiderMonkey I had a garbage collector. I had things that I thought were better done. I had tagged values, machine words with low order bits that told you what the type was.

I took two weeks away from work where the VP in charge of me was saying, "You better get in here. We're trying to standardize this. You have to write a language spec," and I said, "No, get lost. I'm re-implementing it." I have to implement it before I specify it because it really was based on the code. It was based on what I actually shipped. It wasn't a matter of writing down some platonic, perfect JavaScript because there wasn't any such thing. It was all about what actually was in

Netscape 2 or 3 at that point. And so, the JavaScript engine got rewritten in ’96. It was finally shipped in Netscape 4 in ’97 or ’98.

And that helped me keep my head up in the JavaScript standards body, the Ecma TC39 group. Because the first meeting of that group, even though our guru Carl Cargill had thought we would have the advantage inside Ecma, Microsoft set in a lot of people. They were the B-team. So, one of the guys, he actually gave me the joke that I’ve used since then, that “Well, we can’t really call it JavaScript. That’s a Sun trademark,” and Sun was suing people whose surname was Javanko, like a middle European last name.

[Laughter]

BRENDAN:

This guy is on the web. He’s got his website called Javanko.com and he’s named after his ancestors going back to thousands of years and Sun lawyers come up and say, “You must cease and desist using the name Javanko. It’s not your name. It starts with J-A-V-A. It’s our name.” And he said, “Get out of here. It’s my ancestor’s name.” So, Sun was stupid. They didn’t want to give the name away to the standards body. So, Ecma said, “Oh, we’ll call it ECMAScript.” And once of the Microsoft guys, I forget his name, said, “That’s not a really good name. It sounds like a skin disease.”

[Laughter]

AJ:

It does.

BRENDAN:

And he was right, yeah. But because I was actually re-implementing and I was all over the language semantics, I easily defeated them in these various little debates about how to standardize it. And there was a wacky guy from Borland who thought he’d implemented JavaScript. And he had it all differently implemented, not inter-operably implemented, because he wasn’t building a browser. And there was this company called Nombas, No MBAs that was founded by the son of Ray Noorda who was running Novell at the time. And they said, “Oh, we’ve been doing JavaScript for years.” And I said, “Really? I only created it last year. How can you be doing it for years?” And what they meant was they were doing the C-like scripting language. And of course, all the details, all the detailed semantics differed.

But we finally got the committee to codify mostly what was shipped in Netscape. Microsoft after that first meeting where they didn’t send their A-team sent their A-team and that was Shon Katzenberger who’s very smart. He went to work on .NET with Andres Hejlsberg after that. And he actually told his bosses, he did what I did only more so, he told them to get lost and he spent six months working on what became the first edition of ECMAScript as a spec. And Guy Steele did a fair amount of work and others contributed pieces, but Shon did the backbone and the main part of the work. Six months to work on what became the first edition of ECMAScript. And he came up with the formalism for the date object, because the Java-based date object was crazy. He just [said], “Let’s make an extrapolated Gregorian calendar that goes plus or minus 200,000 years because we have that much accuracy, precision within IEEE double if you count milliseconds.”

And he was super very smart. He did a good job. And he didn’t really seek advantage. There were a few things that I thought that he wanted to push that weren’t right, but it turns out, I dug through a mail archive when I met him and I realized he’d mailed me [chuckles] months earlier in ’96 saying, “Hey, there are things in JavaScript. Maybe you can change them quickly before it gets too hard to change on the web.” And when I found that mail I realized I’d skipped it quickly because I realized I couldn’t change it even then.

Marc Andreessen tells the story that NCSA had 80 web servers in the whole world, including CERN, the original web servers from Tim Berners-Lee, and they couldn't change anything. So, Netscape 2 beta, we shipped JavaScript and [inaudible] LiveScript. And even then with early adopters writing code, that contractor with his demo I helped rescue, you couldn't change it. So, there's just survival advantage to keeping compatibility and it's real. And that affected us all the way through this. And it affected Microsoft, too.

So, once… here's the punchline. When I redid the implementation and created SpiderMonkey I said, double equal operator, I was being spun around by the Borland guys who wanted very loose implicit conversions. I was looking at Perl 4 which had all sorts of crazy. That was a mistake. I want a real equivalence relation except for NaN. I want something where you have a reflexive, transitive operator. And it partitions all values into equivalence classes which are as small as possible. Let's make the double equal operator be like what we know as the triple equal operator. And I said, "I know it's incompatible. I'll have people call it JavaScript 1.2 and they'll have to opt into it. They'll say script type equals application/javascript1.2." Microsoft guy said, "Ugh. Well, you're right. You can't do this without any type equals opt-in. And nobody's going to use the opt-in or they're going to get it wrong. So, it's too late. Let's just add triple equals." So, that's what we did. Even then, it was impossible to fix mistakes in the language. This was '96 and '97.

CHUCK:

So, why'd you call it SpiderMonkey?

BRENDAN:

[Laughs] There's a Wikipedia page which obviously somebody from Netscape has over-edited because it half-explains this but it doesn't quite link to the correct explanation. If you look for Beavis and Butt-head Tom Anderson…

[Chuckles]

BRENDAN:

Spider monkey.

CHUCK:

[Laughs] Okay.

BRENDAN:

You will be enlightened.

AARON:

Alright.

BRENDAN:

The name was given by Chris Houck, Netscape cofounder.

AARON:

So, I want to go back to this Ecma thing. I watched some talks at Yahoo where you and Douglas Crockford both gave two different explanations of this falling out. And I think that you compared it to the Lord of the Rings.

BRENDAN:

Oh, this was the fourth edition falling out?

AARON:

Yeah, yeah.

BRENDAN:

Yeah, alright. [Sighs]

AARON:

And he compared it to something like The Good, The Bad, and The Ugly, didn't he?

BRENDAN:

No, that was me also. [Laughs]

AARON:

Okay.

BRENDAN:

That was my TXJS 2010 talk, I think.

AARON:

Okay.

BRENDAN:

So yeah, Doug, I don't know, Doug didn't have a good metaphor for it. I made it Lord of the Rings. I actually had lots of fun trying to find Doug's avatar. And I started with Yoda, but I realized he was too short.

[Laughter]

BRENDAN:

And I said it has to be Gandalf. And so, it was Gandalf. He was on the bridge of Khazad-dûm spiting the Balrog, the ES4-rog, and the hobbits were, the JS hobbits were being defended from having to use this horrible classy, fancy complicated language. ES4 was an attempt, it was several things. It was an attempt by Macromedia and then Adobe which bought Macromedia to standardize the fork of JavaScript called ActionScript that they put in the Flash player.

And they'd actually done two versions. One was ActionScript… maybe they had three. The original ActionScript, now ActionScript 1 was pretty much JavaScript, probably some minor differences. ActionScript 2 had a few more things. It was still mostly a dynamic language. And then ActionScript 3 implemented by the Tamarin engine that they had finally open sourced through Mozilla in late 2006, was quite different. It had classes, packages, namespaces, types, optional type annotations, machine types. You could do n32. And it felt like they were responding to a lot of Java envy or Java angst among their Java-trained developers in the Flex/Flash world of 2006 or 2007. And they wanted to standardize that through Ecma as the sequel to the third edition of ECMAScript.

I wanted to get Microsoft to play ball and evolve JavaScript as it was. When I started Mozilla and started Firefox and we launched Firefox 1 in November 2004, I realized this was going to restart browser competition. So, I rejoined Ecma on behalf of the Mozilla Foundation as a not-for-profit member, which is awesome because you don't have to pay. [Chuckles] And then I started collaborating with Macromedia, soon to be Adobe because they bought up Macromedia, to evolve JavaScript. And as a warm-up, we did E4X which was this crazy XML syntax inside JavaScript that was done by BEA. If you remember them, they were a Java app server company. And John Schneider, then at BEA who had been at Microsoft and his buddies at Microsoft like Rok Yu who was actually running the Ecma liaison for Microsoft decided, “Let’s do E4X.”

So, we did this ECMA-357 standard that tried to standardize ECMAScript for XML. It was an add-on to the JavaScript standard and it put XML literals into the language. It also put namespaces in, including the double colon qualifier to separate the namespace from the name, all sorts of crazy stuff. And it had lots of foot guns in my opinion. It was very much based on an implementation that was done in Java on Rhino, the Mozilla Java implementation of JavaScript. And [it became a bad] standard ultimately. It wasn’t really implemented by anybody else, not even by Microsoft.

But Microsoft for a time, they were talking a good game. They had said while Netscape was still alive, “Yes, we’ll do JavaScript, too. And yes, it will be classy. It will have classes and packages and namespaces.” And they even implemented some of those things, their own spin on them, in JScript .NET in 2000 era. And you can find webpages to this day that talk about JScript .NET and classes in JavaScript. But they backed off on it when they got serious about C# and they reneged on the whole thing. And they also, other than Rok Yu, this fellow who’s left Microsoft since then long ago, but he was a fun guy, he was the liaison to the ECMAScript standards body. And he was doing E4X even though he was not selling it inside Microsoft and nobody was implementing it.

So, I used E4X as a lever to get Microsoft’s engagement and attention and to get JavaScript’s standards body back in business. So, we reformed what was basically disbanded in 2003. Microsoft had crushed Netscape in the browser wars. The Mozilla Foundation was spun up in the summer of 2003 from an AOL layoff. And the person I’d given the keys to the kingdom for JavaScript to Waldemar Horwat at Netscape. Very smart guy, I think he won the Putnam Math exam in 1986, if you know what that is. That’s pretty impressive. Waldemar was from MIT. He has a PhD. He was working on a Java Virtual Machine. This was not the original early days. He was going to do a full jitting, hotspot sell JVM at Netscape in ’97 called Electrical Fire.

But Netscape was running out of money [chuckles] because Microsoft was killing them by taking the price of the browser to zero and also doing a better job on the server side. Netscape could never make server business pay. Waldemar was looking for a job. So, in late ’97 I said, “Hey, why don’t you take over JavaScript and standardize it and take it to the next level?” So, Waldemar worked on JS 2 or early ES4, what you might call as primitive ES4. And you can still find his design documents in the web archive on this. And that fed in some way into E4X and into ES4 the final real ES4 that we tried to do in 2006 through 2008.

So, we tried to evolve JavaScript aggressively to absorb and embrace and extend the Flash player version of it, the fork of the language that was in Flash, the ActionScript language. We also tried to do it, this was my own agenda, to get Microsoft to start evolving it again and stop sitting on the web. Because you guys all know this, right? After IE6 they took the team down to a skeleton crew and they stopped fixing bugs. They had horrendous security bugs and page layout bugs and DOM bugs and networking bugs. And they just said, “Ah, the web’s over. We’re going to do Windows presentation foundation,” which became Silverlight. “We’re going to do .NET. The web, that was a passing fad, sort of like television.” So…

[Laughter]

AARON:

It’s a toy. The web’s a toy.

CHUCK:

That’s right.

BRENDAN:

And until Firefox started taking market share back from IE, they were confident that was true. And then they were like, “Oh shit. We better actually do IE7. Oh wait. We haven’t really done IE7 right.

Let’s do IE8.” And then finally, when Firefox triggered enough response from Microsoft and Google that Google did Chrome, then they really woke up and they did IE9.

But think back to 2006 or 2005 when I was doing E4X. Microsoft was just asleep at the switch. And so, I said, “How am I going to poke them? I have to stick the stick into the hornet’s nest and stir it around.” And the best way to do that was to use Adobe, to use E4X, to use ES4. And so, the thing you mentioned with Douglas Crockford was at first Douglas Crockford was on board. He was at Yahoo. He thought, “Well, we have to repair the defects in JavaScript,” because Doug wants just the good parts to be there. He wants to get rid of the bad parts.

[Chuckles]

BRENDAN:

And like I said early on, it’s very hard to remove things from the web. You can’t really break compatibility. Nobody wants to do it. It’s biologically unsound, unfit. But Doug was up for it at first. So, we met summer in Oslo 2006 and we said, “Let’s do ES4, fourth edition.” The third edition was in ’99, maybe 2000 was the ISO version. It’s been a long time. Time to upgrade it, what can we do?

And so, various people had their own Rashomon-like interpretations of what happened. It was like, the Macromedia/Adobe guys, at that time Macromedia said, “Let’s do classes. We want classes.” And they leveraged Waldemar Horwat’s design work from six years earlier at Netscape. And they put that into what became Tamarin and AS3. It wasn’t shipped yet. And they were making a big bet that it all gets standardized. People like Douglas were like, “Let’s take out the bad parts.” People like me were like, “Let’s get Microsoft to actually work generatively on JavaScript.” So, you could tell it was not going to work out. But it did get things moving. And so, we had ES4. And it got so much design attention, almost, it was overdesigned frankly, that it became a big threat.

And so, by 2007 January, Douglas decided, “No, this is too big. And it looks like Java. It’s classy. And I like things small and simple and I want to take out the bad parts. So, this is all wrong.” And he forged an alliance with Microsoft. Now it may be coincidence, but at the time Microsoft was considering buying Yahoo. [Laughs] So, when we were heading into a spring meeting hosted at Microsoft Redmond, Washington near Seattle, Douglas was there early. He was having meetings on his own with Microsoft people. And we’re like, “Wait. What’s up?” And then Chris Wilson, now on the Google Chrome evangelist team, was at Microsoft still on the IE team saying, “Oh, we don’t really want ES4. It’s too much. It’s too incompatible,” by the way I agree with some of that, and, “We can’t have any of this.”

And Doug was there. [Laughs] I think Jerry Yang broke his heart because Jerry did not accept what probably was the best offer Yahoo could ever get. So, that all fell apart. But still, it let to this public fight over ES4 which you guys probably my blog post at the era. And it was, I can’t remember, an open letter to Chris Wilson. [Laughs] It was not totally serious because what I was trying to do was get JavaScript to evolve. What Microsoft was trying to do was still push Silverlight, maybe help JavaScript. And it turns out they didn’t know what they wanted to do. So, they had people like Allen Wirfs-Brock who then jumped to Mozilla.

But at the time, Allen was working for Microsoft and he became the editor of ES3.1, which became ES5. And that was their, “Let’s do something less than ES4. Let’s do 3.1. Let’s do a no-new-syntax version of ES3.” So, Allen did that for Microsoft and he had Doug on board and others. And again, Doug couldn’t remove the bad parts, so I think he lost interest.

[Chuckles]

BRENDAN:

But Allen plugged away. And between ES4 being overdesigned and Allen finding the APIs that were missing, like object.defineProperty, a lot of us realized, “Hey, this is a good intermediate step. It almost gives us a compile-to language for ES4.” And so, we folded up the ES4 tent in July 2008 at Oslo. And Opera, which was an ally of Mozilla on ES4 along with Macromedia come Adobe, Opera had somebody really strong, Lars Hansen, who had written their Futhark JavaScript engine. And he was all on board. He jumped to Adobe because he wanted to keep working on ES3. But he was also instrumental in it. So, I think Adobe realized it wasn't going to get standardized and they were feeling it was a waste of money to invest in the standard. They're feeling maybe, some of them felt betrayed by Mozilla. But we didn't actually betray anybody. We just said, "We can't say no to ES3.1. It's as good intermediate step. Let's do that."

A year later, we said, "We'll call it ES5 and we'll just mop all ES4 as something that never shipped and onto ES6 and the modern Harmony era." So, I forged the Harmony agreement. I was the rainmaker and the peacemaker, the group [hugger/ranter]. And I wrote the email that's on ES Discuss that talked about ECMAScript Harmony and what was excluded and what was permissible to include in the future versions based on this new peace, this new era of peace. It was like the Treaty of Westphalia, if you will. So, that was all 2008.

The funniest story, which I'll also share here exclusively with you guys, is Allen Wirfs-Brock jumped to Mozilla and I worked with Allen and he's still the ES6 editor. Allen said, "Yeah, right before you guys folded on ES4, Microsoft was about to fold on it." [Laughs] They were about to say, "Ugh. I guess we better implement ES4. It looks like it's going to happen."

[Laughter]

BRENDAN:

So, it was like "Dang it." It was like poker and we didn't bluff hard enough.

[Laughter]

BRENDAN:

But it would have been a mess. It would have been a total disaster to have this overdesigned language of classes, packages, namespaces, all sorts of quasi-static typing. We weren't sure exactly how to make it work. We had dynamic checks based on annotations, not just like Dart or TypeScript but actual runtime checks compiled from like annotations and colon annotations. This stuff was half-baked. There's still interest in this in the committee. There are obviously things like Dart and TypeScript. They erase their annotations. So, they have a checked mode option where you can turn them into runtime checks. That's only for testing. When you actually deploy, you erase them all and you just generate plain old JavaScript.

So, they type system is formally unsound. There are other reasons it's unsound, but at the very least it's unsound because it lets stuff escape into runtime that could have an exception, a type error. But you know what? That actually has found a lot of adoption. So, I think it's useful. And what's useful may not be sound. And so, there's actually a chance, this is controversial in the Ecma committee right now, but maybe in a future annual edition of Harmony era, JavaScript will have optional types. I don't know.

They won't be like ES4. ES4 tried to make them actually mean something. And that's extremely hard on the web, because you're always loading code and you load new code that renders your old type judgments invalid. What do you do? Do you recompile the world? Do you throw a type error about the mixture being incompatible when the old code was already running? Like the old code was the basis of your Gmail user interface. Just because you load new code that invalidates those old type annotations doesn't mean you should break your user Gmail experience. It doesn't make any sense.

So, JavaScript is fundamentally dynamic. And you see all the value in TypeScript and Dart coming from the tool-time advisory checking, the warnings. When I was at JSConf AU in 2012, I made the joke that, "Well, it really isn't a type system, right? They should have called it WarnScript. But WarnScript's not a very good name."

[Laughter]

CHUCK:

So, I'm curious. We've talked about where JavaScript came from. We've talked about where some of the features, some of the controversial features have come from. I'm a little curious where Mozilla comes in with all of this.

BRENDAN:

So, Mozilla was the thing that I worked on after JavaScript and Netscape. In some ways, I've always worked on the same thing. It was JavaScript, the SpiderMonkey rewrite. I realized that I had all these early adopters like that contractor who had that demo that needed to know that table layout was replicating his form elements and he really needed the length-1 element. And I realized this would all be better if it were open source. I shouldn't be hiding anything from these people. They need to have workarounds. I need to have code fixes or bug reports. So, I got very interested in open source.

So did Netscape, because they wanted to do a life pod. It was partly executive vanity project and partly Mike Homer, his metaphor, the Yugo being chased by the Microsoft monster truck into a wall. Literally, you better, he did do a hard left and tried to bail out from [inaudible]. So, Mozilla was that bailout. It was that hope that you could dive from the Yugo, roll in the dust, and get up and then have another sequel movie that was even more exciting later. And so, Mozilla in 1998 launched with the Netscape codebase, which was all this student code plus my SpiderMonkey engine re-implemented in '96 and '97 and integrated into Netscape 4. And it was spaghetti. It was a mess. It was mostly C and it was rushed and it was full of technical debt.

And early on, we had very few contributions to the Mozilla project from outside the Netscape employee base. So, we started with everybody who was a committer, this was in the era of CVS, before Mercurial, before Git, before the modern distributed version control era. We had CVS server that was hosted by Netscape. We had a few contributors from outside who had submitted patches and gotten the permission to actually commit directly. Some of them had done things like replace the crypto module. Because in 1998 the US government ostensibly through the commerce department but really through the NSA would not let you export a crypto module. You could not export the crypto algorithm suite required for SSL.

So, when we released Mozilla codes in April 1st '98, there was no SSL, there was no HTTPS. And pretty quickly, some people down under, Eric A. Young and Tim Hudson, somebody I had met when I was at SGI coincidentally, had hacked an early version of what became OpenSSL which was at that point called SSLeay, Eric A. Young. SSLeay, they'd hacked that into the open sourced Mozilla code to provide the missing crypto algorithms to provide HTTPS. That was one of the biggest, most impressive hacks that were done by open source contributors.

But other than that, it was such spaghetti, second-generation student code. Hardly anyone could to anything with it. And so, I was the technical leader of Mozilla. I was the architect. I said, "We can't have this. We have to do something that's more greenfield that allows homesteadingby new contributors that allows people who care about standards to implement things correctly instead of hacking on this very non-standard codebase."

And Netscape had acquired a company called Digital Styles that was known for rendering engines of some kind. And they started doing a next-generation engine in '97 I think based on Java. And they thought, Netscape's doing the Javagator, Netscape and Sun are going to kill Windows, Java's going to be the future on the client side. Let's build a Java engine. When Java got the plug pulled from it in late '97, when the Electrical Fire JVM that Waldemar Horwat was building at Netscape got cancelled, when Sun went away because Netscape was basically going out of business slowly, the team that was doing this Java engine, this Java web engine, rendering engine called Raptor said, "Oh, we better rewrite it in," maybe it was called Xena, I forget. They said, "We better rewrite it in C++." And then they said, "Let's sell it to Mozilla."

And I was looking, as a buyer I was not just a chump. I was also looking for the greenfield, homesteadingspace for the open source standards-oriented contributors. So, we made a deal to reset the Mozilla project in October 1998 around rewriting everything from the ground up based on this Raptor engine which got renamed Gecko. And around the same time, the KHTML project, KDE was getting going. Now, we could have picked that, but it was also looking kind of raw and it wasn't web-compatible enough. And Raptor was pretty raw. It had a lot of bugs. Gecko, early Gecko had a lot of mistakes that we had to rewrite over the years. But we did get the greenfield advantage. We did get the web standards folks interested. We did get new contributors.

And we did what Joel Spolsky describes as the big mistake. You don't rewrite your codebase if you're in the competitive market. You do Netscape 5, Netscape 6. You do Netscape every year. You keep trying to fight for your market share. Netscape didn't do that. They basically trusted Mozilla to rewrite the codebase. Some of the executives thought it could be done in two years. And I was like, "No, four years." It turned out to be, Netscape 6 was crapped out prematurely based on Mozilla 0.6. We wouldn't call it 1.0. It was so bad, you guys may not remember it, or you may not have been born yet. Netscape 6 was a disaster.

[Laughter]

BRENDAN:

It not only had black and blue theme with circular buttons, it had ICQ and AIM and other AOL incrustations. But fundamentally, it was based on this crashy, buggy, slow version of Gecko. And so, the Mozilla leaders, like Mitchell and myself, were like, "No, not 1.0. You shouldn't ship this yet." The Netscape people living in fear of the AOL masters in Dulles, Virginia were like, "We have to ship or morale will fail," and we were like, "No, morale's going to be bad if you ship." And what they really meant is their necks were on the line as basically the acquisition that had to deliver some value to the mother ship. So, they shipped and it was crappy and it got panned. And I think there were some executive beheadings.

[Chuckles]

BRENDAN:

And Netscape lived on for another year or two, 2001, 2002. And finally we got Mozilla 1.0 done in 2002. And the best I can say is that it didn't suck. At that point, you're not looking for ultimate victory. You're looking for something that you can build on. It was a stable base for API compatibility. It was a stable release. And Netscape did a version 7 or a 6.1 or 6.2. I can't remember the number, that actually didn't suck either but it was too late for them. And then a year later in 2003, AOL pulled the plug on Netscape.

CHUCK:

Did the ICQ integration say, "Uh-oh"?

BRENDAN:

I think it did. There were a lot of things that were jammed in that were closed source. It was not good. And the funny thing was, Mozilla doing open source only, at first we didn't even provide binaries. You had to get your own compiler. You were using GCC or EGCS, if you guys remember what that is, a fork of GCC on Linux.

AARON:

Wow.

BRENDAN:

Or you were using MinGW or Microsoft Visual C, or this was pre-Xcode. You'd use the Apple tools and PowerPlant, the user interface toolkit from CodeWarrior. You used CodeWarrior on Mac. This was old school. So, at some point we decided we're going to do builds because we need testers who don't have compilers to get the builds and test them for us. We don't want to make everybody be a developer who knows how to compile their own bits. And so, by producing builds Mozilla started producing product. We didn't even know it. But our Mozilla browser suite didn't have ICQ in it. And people actually liked it better. So, we started getting adoption.

And at some point, I can't remember when, 2001 or 2002, we actually had more users than Netscape did. [Laughs] So, at some point we were getting the signal, very real signal from the market, the users, saying "You should do your own browser." And that's when I think David Hyatt who went to Apple in 2001 who was one of the senior engineers there who did a lot of what became XUL, he and Blake Ross who went to Facebook and a few other people decided, screw all those Netscape browser suite 90's era application suite where you have mail and news and ICQ and browser and editor and address book, like a Swiss army knife application. But just do a browser and make it really awesome. And they called it Mozilla/Browser. And then I think I was involved in this, that we said let's call it Phoenix, like from the ashes and…

AJ:

I remember when it was called Phoenix.

BRENDAN:

Yeah. We kind of incubated it inside Netscape and sheltered it from management. And they were doing things, even Dave Hyatt who wrote Chimera which became the Camino Mac-only browser, he was practicing how to build a tab browser, how to build widgets, UI widgets in a cross-platform way, stuff that he then went to Apple and worked at Safari on. That all was making fools of Netscape management because they were still polishing the turd that was their application suite.

[Laughter]

BRENDAN:

So, we ended up building what became Firefox as this pirate ship inside Netscape. And then when AOL finally laid everybody off, Mitchell and I knew that was coming through backchannels, because Mitchell had already been laid off and gone to work for Mitch Kapor, the guy who created Lotus 1, 2, 3, big investor, very well off. Mitch was employing Mitchell to work part-time on Mozilla, part-time on the Open Source Applications Foundation, which was basically, I think Mitch was inspired by Mozilla and also Mitch always wants to recreate Lotus Agenda, if you know what that is. It's a PIM, personal information manager. So, OSAF was trying to do that as open source 12, 14 years ago. And Mitchell was working on it, but she was also working on Mozilla.

Mitch Kapor knew one of the early AOL guys, Ted Leonsis, very nice guy. Business guy, not a technical guy. Very well off, of course he owns the Washington Capitals. Whatever. I like Ted, but Mitch and Ted ran into each other at the very first D Conference and Ted was like, "Mitch, I have this thing called Mozilla. I don't know what to do with it inside AOL." And Mitch who coincidentally, or maybe this was fate, hired Mitchell Baker when she was laid off by Netscape/AOL, said "I'll tell you exactly what to do with it," because Mitchell had been talking to him. So, through these backchannels also through IBM we knew that AOL was going to drop the axe on Netscape. They were going to lay off everybody.

Netscape had hundreds of employees in this building in California. AOL's back in Dulles, Virginia. And they sent out some hatchet man VP. He had security goons behind him. They didn't have individual meetings with people. They just did it Jonestown-style. They had everybody in one room, except for the people who went to the other room. The other room was much smaller. So, you had this weird day of event where the email went out and people were saying, "We're all supposed to go to the big room," and then somebody said, "Well, I'm going to the little room." And then people said, "Wait, is that good or bad? I'm going to the big room. You're going to the little room. Which one's good?" [Laughs]

It pretty soon became clear that going to the little room was good and going to the big room was bad, because in the big room the hatchet man said, "You're all fine people and you're all fired." And the little room involved people like me and David Baron and the build guy, Leaf and I think somebody else still at Mozilla. It might have been Asa Dotzler. We were all sitting there and we were not going to get laid off right away. We were going to have three more months and we were going to transition the Mozilla assets to a non-profit which involved the trademarks and some build machines. This is pre-Amazon Web Services, so we have an Amiga or something. I don't know. We had a SunOS machine. We had an IRIX machine. We had PC. We had Macs. We had a very few rack-mounted machines we had to do our continuous integration on.

And by the way, Mozilla did pioneer that with Tinderbox. It's something that's a staple now and it's been done better. But at the time, Tinderbox and continuous integration was a Mozilla feature that I think was new to the open source world. A lot of open source…

AARON:

Brendan…

BRENDAN:

Repositories wouldn't even build if you went to their CVS. I guess I'm coming to another punchline, which is when we spun out Mozilla, we only had 12 people and we had Phoenix. Firebird, remember that name, also was contested, so we became…

AARON:

Yep. I remember Firebird, yeah.

BRENDAN:

And we went to Firefox. But when we got into 2004 we knew we had something that was big. And it was like, you guys remember Chrome in 2008 was hot. Firefox against IE was even hotter. And it was just amazing to be there then to do it, because you saw this opportunity to do a new browser and take back market share from Microsoft, which no one had ever done.

And it was a large part also associated with JavaScript because people were realizing then, the Ajax revolution was starting. People, not Jessie James Garret, but this guy in Sweden, what was his name, Daniel Brockman. He'd written the paper about how you can do partial, like function prototype bind, partial application, partial application of functions, in JavaScript. So, people were rediscovering functional programming. So, things were starting to look really hot as Firefox 1.0 launched. And AOL I think missed out. They could have had that, but they didn't keep it. They got rid of it.

And that helped us like I said, to connect to my earlier story, helped us rejoin Ecma and partner with Macromedia then Adobe, and get Microsoft's attention and do ES4. And even though it didn't actually result in a standard, it provoked ES5 and Harmony. So, you look back and you see all these crazy accidents and these things that you ship that you can't fix later. And long ago, I used to let it hang in my head. It was like, everything bad in JavaScript, some of which I've forgotten [laughs] thankfully, cannot be fixed. But the good there is that that's the web we have.

And you wouldn't have this common asset around the world if you didn't have some amount of compatibility and interoperation and some amount of actual innovation. Both things like V8 and the other jitting runtimes making it super-fast and the actual work to make the language complete, fill the gaps and make it a better language for people writing code and also for things like Emscripten.

So, I feel like JavaScript's still got a lot of go. It's still up-trending. And there are still things to fix in it. And it's still generating returns for all developers who benefit from that shared common good.

AARON:

That was awesome.

BRENDAN:

[Laughs]

AARON:

That's a lot of good history that I think…

CHUCK:

Yeah.

AJ:

Brendan, I don't know if you know this, but you're kind of a big deal.

AARON:

Yeah.

[Laughter]

BRENDAN:

In some ways, I'm the person who was there at the time and had to do it. It's true that nobody else in Netscape would have done it. I was the language buff and the fast implementer. But all those mistakes, I'm sure there are people elsewhere in the world who could have done a better job. But they weren't there. Also, maybe they would have insisted on more time, and that would have failed. Or, maybe if they got that time, would have been better for it. But I didn't do that either. [Laughs] So, it's an accident but it's I think a happy accident in terms of the developer community. A lot of the good ideas and the bug fixes and the optimizations and the extensions, the API extensions especially, came from developers. So, I couldn't have done it without the developers.

JOE:

That's a great point.

JAMISON:

So, I have a couple of questions, just looking back on where JavaScript has gone since its inception. What are the things that have most surprised you? What are the uses that have most surprised you for JavaScript?

BRENDAN:

Robots?

AARON:

Johnny-Five?

CHUCK:

[Laughs] Awesome.

BRENDAN:

I think Node.js was a surprise, too. Ryan Dahl was looking for, he's a C/Unix guy like me, so I think I kind of get him. He was looking for something that was close enough to the Unix system call table and didn't have blocking in it. And JavaScript was it. V8 was it in particular for him. And that's fine. But that was a bit of a surprise.

And it shouldn't have been a surprise. Once you have open source as a commodity, as a means for companies to hire people and get better QA, I think a lot of open source is like that. It isn't necessarily about the community. It's about how can I get my code better tested? How can I recruit talent that I can't find otherwise that just comes to the open source project? And that's fine, too. That's really where a lot of value in open source comes from. V8 has benefitted from that and they've benefitted Node. And Node has benefitted people.

I worry that Node is kind of, you've seen people defect to Go lately, like Felix and then TJ. And you realize Node and JavaScript itself are not the last word in systems architecture or programming languages. So, evolution goes on and there should be other languages, especially where the switching cost is low enough. On the server side, there are other languages. People are doing awesome things with new programming languages. And people are building great languages on top of JavaScript on the client side because that's the low switching cost path, compile to JavaScript. You're not going to easily wedge into their VM in the browsers in the standard way, or even in any browser. Dart is going to have a hard time getting into Chrome. But Google can do it, but will it pay off? I doubt it.

And in the meantime, you can compile to JavaScript. You can co-evolve JavaScript and these new languages like Dart, TypeScript, the Haskell influence languages that are coming along like Elm. And there's mutual learning there, and there's co-evolution. That's where I think we can actually accelerate things. The web can be so much better than it is if the various participants leaned into it more instead of being distracted by their own ability to innovate in a proprietary way, essentially proprietary way, even if it's open source. It's like Yehuda Katz said. Every time somebody does something like a Chrome innovation that could only be in Chrome and it's very hard to standardize, it's a big new codebase and it's a big team at Google, it's like, "Cool story, bro. But what good is it? I can't target that…

[Laughter]

BRENDAN:

In iOS Safari?" It doesn't mean anything. And meanwhile, iOS Safari is investing in its own things but also Apple's investing in native iOS widgets and stuff like LLVM and Swift. But I see some convergence. LLVM is this great compiler framework that's used by Emscripten. Emscripten is basically a JavaScript backend for LLVM that allows you to cross-compile C++ apps like Unreal Engine apps or Unity games to JavaScript and WebGL.

So, we're starting to see convergence through compilers, cross-compilers, and through people working intentionally on the web to fill the gaps and do the extensible web manifesto thing of building up the lowest level possible that's still efficient when you use it at scale so that you don't have the standards bodies inventing these big sky castle high-level standards or high-level APIs. That should be done on GitHub. The standards bodies should be building up from below. And that I believe strongly in. If there was more intentional, coherent, collaborative work that way or even competitive work that way, I think the web could be even better than it is. And I think it will be.

AARON:

That's awesome.

JOE:

Angular or Ember?

BRENDAN:

[Laughs]

JAMISON:

Oh, geez.

[Laughter]

BRENDAN:

Like I said, it would be a disaster. If you imagine the W3C trying to pick a winner and standardize it, it would be like they would pick Dojo right when jQuery was coming out. It would be a mistake. So, I don't really have a dog in that fight. I actually like things about Ember that I've seen. I also like things about React, which is in some ways less ambitious because React just says, "We're the view. You can use us with Ember or Angular." So, people are starting to realize that MVC, the C is vestigial. The controller's vestigial. And the view, there are various ways to skin that cat and I like the React way of using virtual DOM diffing and immutability. And the work Rich Hickey did with Clojure which has borne fruit in many areas based on lots of prior art about immutability solving the time versus state problem, that's really appealing to me. Swannodette, if you guys [inaudible] this.

JAMISON:

David Nolen.

BRENDAN:

David Nolen has done some good work in JavaScript and in the compile to JavaScript version of Clojure, ClojureScript. So, I think all that stuff is too early to standardize. Some of it looks very promising. It needs to be pursued. It needs to be adopted and then battle-tested. And then we can standardize the pieces that deserve to be standardized and make things super-fast.

JOE:

That's a very intelligent answer to a silly question that I asked as a joke.

[Laughter]

JOE:

But I love that answer. That's great. I do want to ask you specifically, what do you think with what Angular does with customizing HTML, writing your own HTML?

BRENDAN:

So, I actually like that. I like also, believe it or not this is a blast from the past with respect to E4X, I like the JSX syntax that React has. I think it turns out Sweet.js, which is a macros for JavaScript project started at Mozilla Research when I was at Mozilla…

JOE:

Right, yeah.

BRENDAN:

Can do that. It can do JSX at least. And I'm not sure about the Angular extensions. But everything in HTML and JavaScript should be ultimately taken out of the committee's hands, the standards bodies like TC39 or the WHATWG or the HTML Group. It should be in the developer's hands, which means things like hygienic macros and extensible custom elements. So, I'm in favor of that in principle. And custom elements among the many things in Web Components, which Google I think with Mozilla is spearheading, that looks pretty promising and should be doable.

In fact, you can do it, Sam Ruby demonstrated this years ago, you can make, HTML lets you make up unknown tags. And they turn into elements in the DOM that just don't have any particular magic to them. And then you can go into them and inspect and rewrite them. So, Sam had a demo where he loaded MathML as HTML and it made a bunch of unknown elements. But then he went through with a tree-walker that recognized that these were MathML elements and it made them render MathML. You can do that with SVG.

We want custom elements. We want ultimately, I hope at least offline if not online, hygienic macros. And then JavaScript is done. At that point, you've filled all the semantic gaps. You have typed arrays, typed objects. You have probably lots of ways of controlling memory allocation in a precise way that doesn't run into garbage collector faults or performance problems. You probably have all the affordances of classes that you want in more concise functions. I think you've got all the statements and expressions you could ever want. We'll have thanks to Rick Waldron the exponentiation operator, star-star. How many more do you need?

At some point, Sweet.js ties the knot and people can write hygienic macros and extend the language. It can even do JSX. It can do markup, angle bracket delimited markup. And then it's not quite the promise land, but it's pretty close. It puts all the power in the developer's hands. It lets people… The complaint is then macros allow you to do things like in Lisp, Common Lisp or Scheme, where you have basically dialects that different adherents cannot understand. They're mutually incomprehensible because they don't understand each other. There's macrology, but they should just learn.

And it's all based on a common set of primitives and that's the important thing. You can reason about the primitives. You can compose them. They're compositional. They compose orthogonally. That's the goal here. JavaScript has a lot of warts in it, but if you weed those out or deprecate them you end up with stuff that is compositional. And that's where I think we're heading.

JOE:

Awesome.

CHUCK:

So, it's interesting that you're talking about all of these different capabilities. I've found or I've noticed that the more recent larger JavaScript applications, they really do feel more like applications and less like webpages or websites. Does that make sense?

BRENDAN:

Yeah.

CHUCK:

Is that the direction we're going here with this?

BRENDAN:

It is. So, originally the vision was JavaScript would be the glue language, like the stupid little brother sidekick to Java. And you write all your components in Java and you glue them together. Maybe you [inaudible] different programmer, a less experienced programmer, would take the components invented by the high-priced experienced programmer and glue them together with HTML and images and CSS to make an app. And that is still done and that will continue to be done.

But you're also seeing SPAs and people building, whether they're using JSX or some consolidated syntax, they're building applications that are very large. And JavaScript as a scripting language or as a language that was easy to use, JavaScript wasn't meant to scale up to hundreds of thousands of lines. So, even things at Mozilla Research like the Shumway Project, which is a Flash player in JavaScript which is pretty awesome, have started using TypeScript, because you want that kind of ahead of time warning system. You want WarnScript. You want modules before they're actually implemented fully in SpiderMonkey. You can get the benefit of them using TypeScript.

So, I do see building large applications requires different tools and different affordances in the language. It involves problem solving among a larger team that isn't just about the expressiveness of the language or whether it's complete by some measure. It involves whether you can actually understand the other guy's code or write tests for the other person's code, or can you actually work together at scale? And that's also coming. Again, TypeScript is smart because it's embracing and extending. Let's hope Microsoft doesn't try to extinguish. That's the third E.

JAMISON:

[Laughs]

BRENDAN:

I don't think they will. I think they actually bought into JavaScript at some level. And I've talked to people like Anders Hejlsberg about this. And they also, coming with little mobile market share compared to iOS and Android, I think they have incentives to work on JavaScript as a platform language instead of…

JOE:

Yeah.

BRENDAN:

Trying to start over.

JOE:

Well, they almost killed it with IronRuby, right?

BRENDAN:

The whole thing where they were doing Silverlight in multiple languages and a .NET light, it didn't work out. That was the same era as the Flash/Tamarin ActionScript era.

JOE:

Right, right.

BRENDAN:

The web endures. And yet, you have Zuckerberg two years ago saying the web was a big mistake for us for two years. We wasted a lot of money on it. But he buried his own lead in the same paragraph. He said the web's still pretty important and pretty optimistic. It's still bigger for us on the m site than the sum of all our native apps at that time. Now, they've gone on to have better native apps. But their native apps aren't even fully native. If you look at the Facebook iOS app, it has a lot of web views in it. So clearly, the web's important.

You look at the Amazon app on iOS. It's got a native home screen of course. But once you get into the catalog, the marketplace, it's web. It has to be, because Amazon has this incredible information architecture that they already mapped HTML in very detailed ways, like the cross-through-ed price with the cheaper price or the Kindle price. There's no way you're going to pay somebody to recreate all that presentation in Cocoa Touch. It's just insane. You wouldn't do it. You get it wrong…

AJ:

To me, it seems like a lot of this native stuff is a step backwards. And I hear a lot people saying that the prediction is that the web is going to bow down to native. But it's like, well then, there are so many different native frameworks. To me, it seems like their time would be better spent in optimizing the browser on that mobile device than coming out with Swift, for example. What are your thoughts on that?

BRENDAN:

So, there's atension. Obviously, that's true, because Apple did the web a solid or two in WWDC in June. They announced WebGL support enabled in iOS 8 which had been turned off. That's the last WebGL domino to fall after IE 11 shipped WebGL support. They announced full, four-tier, optimizing JavaScript core JIT support for apps. That even helps Chrome for iOS. But Apple, I think can't afford to sandbag the web. They have to worry about Google flanking them by taking over the web. So, Apple's still in there uplifting the web, even though they're also uplifting the native side.

JOE:

Right.

BRENDAN:

And like I mentioned, Swift, you read Chris Latter's blog, not the public Apple's propaganda, but Chris actually gave Rust at Mozilla Research credit for influencing Swift. And he said Swift started as his own personal project in 2010. So, I believe that. And Swift's just a better language than Objective-C by far. And it's an LLVM frontend language. So, the real asset there is the open source LLVM compiler framework for ahead of time compilation. And again, that benefits not just Swift but Emscripten. It benefits Epic games in Unity, Unreal Engine, and the Unity Engine. It benefits everybody else doing cross-compilation to the web.

And it's no secret it's being used to place Autodesk and I think Adobe's looking at it. So, Emscripten is a big deal. Compile to JavaScript is a big deal. Mapping C++ "native" code to JavaScript, that's happening. Even Google's portable native client folks will reference Emscripten as… Oh yeah, if you use PNaCl, it only works in Chrome. But to work in other browsers, you run a separate Emscripten pass and you get JavaScript out. You can run that in other browsers. So, there's this clearly the web is not dead, far from it. It's in everyone's interest who's not dominating the web to uplift it.

That's where Apple and Microsoft are cooperating in spite of their native frameworks. And even Google has a lot of solid web people, but they also have their native stack. And maybe they have too many [laughs] engineers. I don't know. They seem to sometimes conflict with each other more than anybody else when I see them fight in standards bodies. But I think the web is too great an asset. The number of web developers, estimated, this is rough, 8 million or more now, far more than iOS and Android developers.

So, you see companies like Collin Jackson, a friend who was at Stanford under John Mitchell, he and Adam Barth were doing browser security back in the day when hardly any other academic CS researchers were looking at browsers. Collin has a company called Apportable which is cross-compiling iOS Cocoa 2D, Cocoa Touch apps, mostly games, to Android. Because a lot of those games get developed for iOS first because that's where the high monetization user is, and then when you get to Android which is the big volume OS, you don't want to rewrite it by hand. You just press a button, Apportable generates the code. So, I think that's awesome. I think compilers and the web are a perfect marriage for keeping up with native.

And if the web just gets those gaps filled, like obviously I'm not going to lie to you guys. WebGL is based on OpenGL ES2. It should be OpenGLES3. It should be, WebGL, it should be based on

OpenGL 4, the desktop version, if the desktop GPU is there, because we're all used to sorting through old browsers and [inaudible] hardware in small screens and not retina screens. Why can't we have the best of the GPUs instead of the lowest common denominator? And I think anybody working on the web, and this is a problem at Mozilla too frankly, who thinks of the web as only the lowest common denominator is doing it wrong. You want the web to be the best of the native. You want to compete.

JOE:

So, why aren't we hearing more about ASM lately?

BRENDAN:

I'm not at Mozilla, but I think they're still plugging away. The Unreal Engine 4 is coming up in it. There's work, I don't want to get ahead of myself here but there's work even with Googlers on exposing all the missing low-level APIs that are needed for native code, like threads frankly. SIMD is public. This is in the Ecma TC39 meeting notes that you'll see on ESDiscuss.org.

John McCutchan who's an ex-game developer who went to Google and worked on Dart's SIMD intrinsics, single instruction, multiple data, short vector instructions like for the SSE unit on Intel and the NEON unit on ARM, this is a way of parallelizing by 4x using parallel data operations on short vectors, like four 32-bit floating point numbers at a time, that's great for games and DSP hackers and lots of people, crypto hackers. John did a nice job on Dart and he tweeted about it. And I tweeted back and I said, "Hey, how about let's put this in JavaScript so you can actually make Dart to JS compile to something that works instead of having it be really slow?" And he said okay. And he's joined the Ecma group. And Intel and Mozilla and Google have thus collaborated on SIMD for ES7, which is implementing in SpiderMonkey and in V8 thanks to Intel folks.

So, there's a lot of stuff that people think, "Oh, JavaScript can't do X." Can't do definite types. Well, can do typed arrays. Can't do no garbage collection allocation, again through typed arrays, through Emscripten. Can't do SIMD. That's coming. Can't do threads. Even threads are coming. The trick is to avoid or somehow manage the risk of data races. But all that stuff can happen. And then at that point, JavaScript is almost this complete target language for C or C++. And at that point, I think you will see lots of games. Even now, you'll see Unreal Engine 3licensees, like the Monster Madness folks that released on the web. Unity has said that the web is first-class target for them. Epic, with Unreal Engine 4 has the web as a first-class target. And this means not just Firefox but Chrome.

And it definitely means IE. I can't say more, but the Channel 9 video from last year with Anders Hejlsberg and Steve Lucco and Luke Hoban at Microsoft all talking about ASM, you can tell they're interested. And it makes sense. They don't want to be left behind. They want to make the Chakra engine in IE12 or 13 run compiled native code, C++ to JavaScript, super-fast. They've done WebGL. They swallowed that. It was so funny. Microsoft said, "Oh, Direct9, Direct11, Direct-whatever. We're not going to do WebGL. No. No. No. Yes." And then they said [chuckles], it's like down the memory hole. Suddenly WebGL's in IE11 and it's like all is forgiven. It's WebGL.

1:

32:35argcomputeshader, general computing on the GPU. JavaScript will be the shortest path to massive parallel computing power. And it will allow you to take legacy codebases. Legacy doesn't mean old, boring. It means hot games that can only be written in C++ because that's the way to get them to the metal on the Xbox, and cross-compile them to the web.

CHUCK:

So, I also wonder a little bit. Does your role with JavaScript change now that you're not with Mozilla anymore?

BRENDAN:

So, I'm on the TC39 standards body still. And I think that will continue. Ecma wants me to remain and they were kind enough to give me an award last time, along with Waldemar Horwat because I think they recognize people who'd been doing this for a long time. [Laughs] And so, Waldemar and myself even longer than Waldemar had been plugging away on JavaScript standards for 18 years in my case. And so, I'm going to continue with TC39. And I'm still very interested in the web. I don't know where I'll end up or what I'll be doing. But I think trying to do something really new. People can say, "Oh the internet of things require mostly powered off devices and very lightweight protocols and very simple processors," and that's true. And that's not JavaScript or HTML. It's going to be much simpler.

On the other hand, that's a far out bet. I think the nearer term thing is people experimenting with Arduinos and other such systems or people doing things like their own Nest-like things but not locked down or tied into Google's mother ship. I'm much more confident in that sort of hobbyist, early adopter path. That's what the personal computers in the 70s with the Apple and the Apple II and the S-100 bus computers people built, personal computers before the IBM PC, all that stuff was incredibly generative in its day. And obviously the biggest success there was Apple. It's still with us.

But I don't think internet of things is going to be a clean break from the internet. I think there has to be continuity and evolution. And if we end up finding a path that miniaturizes part of the web to fit on very small sensors or networks that are mostly powered off, I wouldn't be surprised. But I think the web with all of its developers, 10 million developers, however many it is, that's just too much to just press the reset button on. So, I'm still pretty committed to the web.

JOE:

Awesome. That's probably a good note to wrap up on, don't you think?

BRENDAN:

Yup.

CHUCK:

Yeah, probably.

JAMISON:

This has been fantastic.

CHUCK:

Yeah, absolutely.

AJ:

Huzzah!

[Chuckles]

AARON:

Epic.

CHUCK:

Thanks for coming, Brendan.

BRENDAN:

Oh, look at the time. [Laughs]

CHUCK:

Alright, well should we go ahead and do some picks?

AARON:

Yeah.

CHUCK:

Alright, Aaron, do you want to start us with the picks?

AARON:

Yeah. I got three picks this week. The first one is I tried it out just to see what a what, but it's called hapi.js. And it's Walmart's version of Express. I hate to say it like that. Walmart guys probably are like, "Screw you, man." But it's Walmart's framework on top of Node to do REST API and I thought it was awesome. I really liked it.

JOE:

Yeah, you normally don't want to call something the Walmart version of anything, right?

AARON:

Yeah.

JOE:

That's not a selling point.

AARON:

Yeah, no, no, no.

CHUCK:

[Chuckles]

AARON:

I meant, yeah. So, it's a really, really good framework. If anyone's looking, maybe check it out.

Disabled:

Should I Care?' It was by Valhalla. And I'll put the link in the show notes. But it just wasn't good. I found myself cringing as he would point out the obvious comebacks and then be like, "But I don't really care because here's what I think." And so, it was painful but I think everyone should read it. Any JavaScript developer should check it out. It's called 'JavaScript Disabled: Should I Care?'

And the last pick is self-serving. I'm doing a Frontend Masters course next month. It's in Minnesota. It'll be, you can listen live from home. But it's on ES6. So, I won't do it nearly as much justice as Brendan would, but Rick Waldron is going to go over most of the content to make sure that I'm not telling too many lies and that it looks good. But it's going to be a great course. It'll be really, really good. So, Frontend Masters next month. Those are my picks.

CHUCK:

Awesome. AJ, what are your picks?

AJ:

First of all, I'm going to pick that just perfect moment when Brendan said, "Cool story bro" talking about Chrome Apps.

[Laughter]

AJ:

Because that was epic in ways that only the long term listeners will understand, because Frosty is on here. And he's always "Cool story bro"-ing people about stuff. And Chrome Apps are his thing.

AARON:

I loved it. I thought it was awesome.

AJ:

[Laughs] Oh, that was hilarious.

My other pick for today, actually I'm going to pick two more things. One, I'm going to pick 'Don't Stop Me Now' by Queen.

JAMISON:

Queen.

AJ:

Yes, because I also felt that that was very applicable in that moment. Brandon was just passionate and going and going. And then Frosty just had to cut him off. He was having a good time and he just had to cut him off. I'm sorry, I'm picking a little much.

And then the last thing I'm going to pick is Trending.fm. I don't even think it's in public beta yet, but I know these guys that are in this incubator and they're working on Trending.fm. And it's this cool music site where you can tune in and you basically create a group radio station kind of thing. One guy is the DJ and then everybody else can throw song suggestions in the chat window and the DJ guy can click add. And so, great for the office type scenario or something like that, or if you just know somebody who's really into music. You love their style and you just want to listen to what they listen to all day.

CHUCK:

Alright, Jamison, what are your picks?

JAMISON:

Oh, man. I have more music. This is a music-themed episode, or music-themed picks at least. One of them is a soundtrack for a little arena shooter game. The game is pretty sweet, too. But the soundtrack is my real pick. It's called 'WE ARE DOOMED' and it's just really good synth-y electronic music. It's my programming soundtrack for the past week.

My other pick is another video game soundtrack. There's this PlayStation 3 game that just came out called Hohokum I think. It's one of those weird artsy games that make you feel superior playing it even though it's not very much fun. [Chuckles] But the soundtrack is another fantastic piece of music. It's not as synth-y, but I don't know. They're both good listens for programming. Those are my picks.

CHUCK:

Awesome. Joe, what are your picks?

JOE:

Alright, so I better stick with the music theme and pick some music. I'm going to try to be a little less pedantic than the rest of you, though. [Laughs] I'm going to pick Nashville Outlaws.

AARON:

Oh, geez.

JOE:

A tribute to Motley Crue, a new album that I'm not sure when it came out. It came out in the last year. I just noticed it and it's got a whole bunch of country artists singing the hits from Motley Crue. And I absolutely love it because I was a huge Motley Crue fan back in the day. And so, hearing it done by these country artists with totally different type of sound to it but still the songs that I know and love, I absolutely loved listening to it. Not really great to program to, because you just want to sing. But still a great album. So, I'll pick that.

And then my second and final pick is going to be Audible. I was recently using AudioBooks.com. Bought some audio books. It was nice, because they were $12 every time I wanted to buy an additional one and didn't have any kind of screwy plans the way that Audible does. But their app completely blows. And after I got the Audible app, I was like, okay, I'm never listening and using the other AudioBooks.com app again until they improve it. So, I'm also I guess in essence calling AudioBooks.com out on the floor saying, "Make your app better," because the Audible app really rocks. And this would be my second and final pick.

CHUCK:

Alright. I just have one pick. I've been working on some subscription stuff. I should have it out here within the next few weeks. Anyway, what I've been using to collect payments is Stripe. And I just love Stripe. It is really nice to integrate with, does subscriptions and one-off payments. And I actually have it set up right now at DevChat.TV/donate. But anyway, it's just really easy to put together and integrate with. And they have libraries for freaking every language. So, go check them out. Stripe.com.

Brendan, what are your picks?

BRENDAN:

Oh, Guardians of the Galaxy.

CHUCK:

[Laughs] I love that movie.

AARON:

[Chuckles] Which hero are you?

BRENDAN:

I am Groot.

[Laughter]

AARON:

I was hoping you'd say that, bro.

JOE:

Nice.

CHUCK:

[Laughs] Is that all of them?

BRENDAN:

Yeah. I'm not nearly as hip as you guys. I'm old school. I still use vim in the terminal.

CHUCK:

I use Emacs in the terminal. We can fight later.

BRENDAN:

I used Emacs in 1979 and '80 on DEC minicomputers. It was Richard Stallman's Emacs. It didn't have Elisp. It had TECO, this crazy stack language. You should look it up. I switched to vim and I can't go back. I can do both. I can do Emacs if I have to.

CHUCK:

Yeah, that's pretty much what I say about vim, so.

BRENDAN:

[Chuckles] It's true. Typing your name in vim is very destructive whereas in Emacs it's constructive.

CHUCK:

Yeah.

[Laughter]

CHUCK:

Alright. Well, thanks for coming on the show. We really appreciate you taking the time.

JAMISON:

Yeah, this was incredible. I'm really glad I got to hear the history lessons.

BRENDAN:

My pleasure. I look forward to seeing it all online.

AARON:

Cool.

CHUCK:

Yeah, next week.

[Working and learning from designers at Amazon and Quora, developers at SoundCloud and Heroku, and entrepreneurs like Patrick Ambron from BrandYourself, you can level up your design, dev, and promotion skills at Level Up Con taking place October 8th and 9th in downtown Saratoga Springs, New York. Only two hours by train from New York City, this is the perfect place to enjoy early fall and Oktoberfest while you mingle with industry pioneers in a resort town in upstate New York. Get your ticket today at LevelUpCon.com. Space is extremely limited for this premium conference experience. Don't delay. Check out LevelUpCon.com now.]

[This episode is sponsored by MadGlory. You've been building software for a long time and sometimes it's get a little overwhelming. Work piles up, hiring sucks, and it's hard to get projects out the door. Check out MadGlory. They're a small shop with experience shipping big products. They're smart, dedicated, will augment your team and work as hard as you do. Find them online at MadGlory.com or on Twitter at MadGlory.]

[This episode is sponsored by RayGun.io. If at any point you application is crashing, what would that cost you? Lost users, customers, revenue? RayGun is an essential tool for every developer. RayGun takes minutes to integrate and you'll be notified of your software bugs as they happen with automatic notifications, a full stack

trace to detect, diagnose, and fix errors in record time. RayGun works with all major mobile and web programming languages in a matter of minutes. Try it for free today at RayGun.io.]

[Hosting and bandwidth provided by the Blue Box Group. Check them out at Bluebox.net.]

[Bandwidth for this segment is provided by CacheFly, the world's fastest CDN. Deliver your content fast with CacheFly. Visit CacheFly.com to learn more.]

[Do you wish you could be part of the discussion on JavaScript Jabber? Do you have a burning question for one of our guests? Now you can join the action at our membership forum. You can sign up at JavaScriptJabber.com/jabber and there you can join discussions with the regular panelists and our guests.]