EPISODE 698

[INTRODUCTION]

[00:00:00] JM: Google Search is a highly interactive JavaScript application. As you enter a query, results are being automatically suggested to you before you even finish typing. When you press enter, some of your search results may be widgets that represent the weather, or the price of a stock, or a recipe for green bean soup, or a language translation for praise. These complex frontend components are loading dynamically. The Google search application is not pre-fetching every single possible widget that you might ask for. It's not prefetching the weather widget, and the price of a stock widget, and the currency conversion widget.

All of these things are a little bit expensive and they would slow down the page of Google were to pre-fetch the JavaScript for all of these different components, but the results do load very quickly. If you search for a currency conversion, Google loads the results of that currency conversion widget quite quickly. So you know that Google is doing some advanced JavaScript engineering to deliver you the code for that currency conversion widget very quickly, and Google has many other examples of advanced JavaScript programming.

The company is mostly known for backend engineering inventions like MapReduce, and TensorFlow, and Dremel, and Spanner, and to turn these backend tools into user-facing products, Google develops its own JavaScript frameworks and infrastructure to deliver information from the backend to the frontend. Backend and frontend are terms that are not very precise. A backend is a backend only relative to a frontend, and at Google there are so many layers of infrastructure between a user and the data center, that if you're an engineer working on a service at Google, you probably have several frontends on one side of you and several backends on either side of you. It’s a multilayered, multi-tiered infrastructure.

Malte Ubl is a senior staff engineer at Google and he's heavily involved in Google's JavaScript infrastructure. He has written about managing large JavaScript applications in detail and he also works on AMP, which is an open-source project for delivering webpages in a fast performant format. Malte joins the show to describe Google's history with JavaScript frameworks, the process of building frontends and middleware to deliver JavaScript applications and the

engineering behind the AMP project. There are criticisms of AMP, but some of them misunderstand how the AMP technology actually works. AMP allows pages to be cached and pre-fetched and served to a user more quickly.

In the case of Google Search, when you search and you get some results that are AMP pages, accelerated mobile pages, all that's going on is that the page is being pre-fetched. So when you click on the page, it's going to load instantly, because the page is actually already loaded on your phone, and the reason it can do that is because AMP pages are conforming to a specific format that is easier to cache and pre-fetch and have predictable performance characteristics. So the criticisms are often the AMP centralizes pages around being served from Google Search, or some other confused accusation towards Google like that, but AMP does not necessarily centralize pages around being served from Google Search.

Another good example of amp speeding up pages in a realm that is not Google Search is Reddit. If you click on a sub-Reddit about programming, for example, and one of the pages listed is an AMP page, then if you click on the AMP page, you're going to get it loading really quickly, because the AMP page has been pre-fetched by Reddit, and that AMP page is not necessarily being pre-fetched from Google servers. There are Cloudflare AMP caches. I believe there is now a Bing AMP cache, and these are places where these pages are hosted that are outside of Google.

So this is a pretty interesting show. We talked about a lot of different subjects, and I really enjoyed having Malte on the show. Before we get started, I also want to mention, we launched a new podcast recently, which is Fintech Daily. Fintech Daily is about payments and cryptocurrencies and trading and the intersection between finance and technology. You can find it on fintechdaily.co. It’s fintechdaily.co, or on Apple or Google podcasts, and this show is going to have multiple different hosts. We are looking for other volunteer hosts who want to participate right now who want to cover areas of fintech. If you're interested in becoming a host, you can send us an email to host@fintechdaily.co. That's host@fintechdailiy.co, and I hope you like this show. I hope you like Fintech Daily.

[SPONSOR MESSAGE]

[00:05:50] JM: Your audience is most likely global. Your customers are everywhere. They're in different countries speaking different languages. For your product or service to reach these new markets, you'll need a reliable solution to localize your digital content quickly. Transifex is a SaaS based localization and translation platform that easily integrates with your Agile development process.

Your software, your websites, your games, apps, video subtitles and more can all be translated with Transifex. You can use Transifex with in-house translation teams, language service providers. You can even crowd source your translations. If you're a developer who is ready to reach a global audience, check out Transifex. You can visit transifex.com/sedaily and sign up for a free 15-day trial.

With Transifex, source content and translations are automatically synced to a global content repository that's accessible at any time. Translators work on live content within the development cycle, eliminating the need for freezes or batched translations. Whether you are translating a website, a game, a mobile app or even video subtitles, Transifex gives developers the powerful tools needed to manage the software localization process.

Sign up for a free 15 day trial and support Software Engineering Daily by going to transifex.com/sedaily. That's transifex.com/sedaily.

[INTERVIEW]

[00:07:38] JM: Malte Ubl, you are a senior staff engineer at Google. Welcome to Software Engineering Daily.

[00:07:43] MU: Thanks for having me.

[00:07:44] JM: You've been heavily involved in building frontend infrastructure at Google for several years. You write about JavaScript. How has JavaScript usage at Google evolved since you started there?

[00:07:56] MU: That's a very good question. I think you have to go back to maybe even the launch of like Google Maps and Gmail where we very much innovated and what people thought could be done on the web. You have to think back to the world of like [inaudible 00:08:11] and stuff like that, but we definitely didn't yet know how to build web applications. It was all kind of invented in midflight.

Then over the years, I think we just like got it wrong over and over again and then eventually came to a state where we actually – I think right now we're pretty confident saying like we actually kind of know how to build applications in a way that is really scalable and has like good outcomes in a high-percentage of attempts. That's how I will put it. That's what we're trying to do today that basically always good. It's not great, because there's some like superstar engineer who really knows what to do. It's that the teams are really successful at scale.

[00:08:54] JM: I have seen a wide variety of ways that people manage the frontend of their applications. It seems to be more of a Wild West than perhaps backend development. There seems to be more standardization around how projects are managed, the different roles in those projects, the interactions between frontend engineers amongst themselves I think is less standardized than perhaps the interactions between backend engineers. You're broadly speaking maybe not at Google. Do you think that's the case? Do you think that's accurate?

[00:09:31] MU: I actually think it's not accurate. Where I have seen big problems in the past is when the communication between frontend and backend engineers is not good. That is a very real problem. I think that from within those disciplines, it's always just that the other one might look more or less organized. But I think that's more of a myth.

What's very real is that often these disciplines don't talk to each other much. I think that's kind of where the notion of the full stack engineer kind of came in who at least the hope is capable of kind of navigating the disconnect and building solutions that work end-to-end. I think that was necessarily all that successful. So this gap of communication is absolutely real and a problem.

[00:10:18] JM: You can organize entire teams of frontend engineers working on the UI layer or the middleware layer. You can also organize engineers where a frontend engineer works within a product team and the frontend engineer works with backend engineers and you're just trying

to ship a product. At Google, is there a standard model for how frontend engineers are organized?

[00:10:48] MU: Google is a very big company. So we definitely do it all of the possible ways, I guess, but there is strong bias, and I think by far most common way is to have a product team in which people just take on different roles, but they all work together.

We also very much use a multilayer architecture. Famously every layer basically is a bunch of protocol buffers talking to the next level of protocol buffers. I think at that stage, it's just such that effectively everyone at Google is a frontend engineer, right? Because there's always a backend that you're talking to, whether you're working on Spanner and you're just a frontend to Bigtable, or you're working on an application and you're a frontend to Spanner, or you're working on a UI and you're a frontend to that application. This pattern actually repeats itself quite dramatically, and because the teams are so big, it definitely happens that you do have an organizational split that is just the classic frontend and backend, but you very much often have what we then call more infrastructure teams that work on kind of the fundamental problems that don't necessarily are very low level, but definitely disconnected from the product itself.

[00:12:02] JM: At the level that you spend most of your time, I think it is the frontest end, because it is the layer that gets rendered to the user. It's the frontend rendering engine and the JavaScript that's being executed in the user's browser. Is that correct, or is there also some layers of middleware and backend programming that you spend your time in?

[00:12:30] MU: I think I'm my favorite anecdote is that, for example, like Chrome has a backend team, and the backend team is the blink team that works in the rendering engine, right? Because of rendering engine that makes the pixels is a backend to Chrome, which is the actual UI. Again, this metaphor scale, it's very much to every level.

My work that I currently do on AMP thus span between different layers, but possibly not as much, definitely not in the form of doing. However, my previous work on JavaScript infrastructure very much spans the whole stack and we were mostly actually thinking about these things in an end-to-end fashion where what happens on the client side is just a small part, but everything has to fit together end-to-end.

[00:13:19] JM: How does the tooling for a JavaScript developer inside of Google differ from what developers outside of Google have available?

[00:13:29] MU: I think there's a few key differences. So one of them is that Google has a standardized build system. It's open sourced as Bazel, Bazel, however you pronounce it, and this system is just used for building everything at Google, from Java, to C++, probably some Haskell in there as well, Go and JavaScript. You never have to wonder, like, "How do I build this project?" Just always exactly one command to do it and it does the whole thing. In the fundamental level, it's the biggest difference.

Then going up a little bit, we currently – I mean, we use a bunch of JavaScript frameworks. You will find a lot of Angular, especially for internal apps. You will find a lot of Polymer. You'll find a lot of React if you look a little bit closer, but you also find a lot of the stuff that I've been working on. It's a framework. It's called Wiz. It's not open source, and it's currently powering most of our consumer-facing applications. So it's very much designed for something that – It's not a rare type of application, but at Google it's very common, which is a highly-tuned and highly UX-refined custom application targeted at a lot of users.

[00:14:46] JM: You said it's called Wiz?

[00:14:47] MU: Yeah.

[00:14:48] JM: So these JavaScript frameworks, there've been a number of frameworks that we've covered historically, things like Angular. We covered React a lot, Vue.js. Is Wiz in the same category of frameworks, or is it a framework at a different level of abstraction?

[00:15:10] MU: We have a bigger system that kind of goes further vertically up and down the stack, but Wiz is at the same level of distraction, like very roughly speaking, because you named to very large category of frameworks. But it's in a few ways fundamentally different and uniquely suitable for what we want, but it certainly opens up a lot of tradeoffs.

[00:15:34] JM: There was this Kubernetes open source project, got started as an open source version of Borg, and my understanding was the main reason that they didn't open sourced it, open source Borg itself, was because Borg was so tightly coupled to the Google specific infrastructure, that it was a project that just wouldn't make sense to open source. Is that been the case with Wiz where it's just so tightly coupled with the Google infrastructure that it just doesn't make sense to open source?

[00:16:05] MU: It's a different case, and we've certainly discussed open sourcing it. We just felt like we didn't necessarily add enough value to justify the cognitive load for the community for yet another framework. Based on that, kind of we made the decision not to open source.

I actually think that this was a mistake, and it might still be a mistake. Maybe we will change our mind. What I didn't realize at the time was how unique the framework actually was and its capabilities and how attractive that would be for the use cases.

When we started launching apps in it, it was kind of the time when React got popular and that seemed fine and people seemed happy. So why would you give them something else if they're already happy? What I didn't realize, for example, is how much startup time is a problem for React, because you have this process that's called hydration, where you basically – Even though you have the services that render your app, you still have to – Effectively, clients that render it again on the client, even if you don't expect any changes. I didn't realize at the time how big of a problem this might be, and that's why I didn't realize how big of an advantage Wiz would have, because it was very much designed for having like a nice program model that people would enjoy, but on the other hand, have effectively a zero startup time where you just kind of have the app in front of you and there is never this kind of called uncanny valley, where you see it, but you can't use it. There was something that we very much wanted to not have, and I didn't, again, realize at the time that this was such a big problem in the open source framework space.

[00:17:47] JM: When did the focus at Google shift from Angular as the most leading edge framework to Wiz if Wiz is now the most popular one?

[00:17:59] MU: Again, it's a very big company, and whenever you ask for any technology, you'll find someone using it. When we actually decided to build Wiz, we had experimented with Angular, and I think definitely at the time, because I must say that they very much pivoted and now have technology that if we had had it like five years ago, probably have built on it actually, but –

[00:18:22] JM: In Angular, you mean?

[00:18:23] MU: Yeah, but the situation was definitely in a state back then where I would classify Angular as aimed as a highly productive framework that makes compromises in the fidelity of the application, and this is certainly true for application startup times. At the time, I don't think that Angular actually supported service at rendering, and that was an absolute must have for us, definitely not negotiable.

[00:18:46] JM: So I've done a lot of coverage of this Kubernetes community recently, and what was interesting about the Kubernetes world was there was a time where there were a bunch of different container management orchestration open source systems that were competing for mindshare, and then eventually people settled around Kubernetes, and that settling around a standard led to much, much faster developments and much more benefits to the open source community. Do you think that will happen with the frontend frameworks where eventually there will be some kind of centralization, or do you feel that frontend is – Again, obviously there's that dodgy term, frontend versus backend, but these JavaScript frameworks, maybe that would be the term you'd want to use. Will there be more centralization, more standardization at some point?

[00:19:35] MU: I don't think that's the way it's going. There are a few things that are going to happen though. One of them is that many of these frameworks are modeled around some notion of components, and we have seen the standardization of web components, which are now appearing in all the rendering engines, finally. So there's one thing that I am very sure of, which is that we will see web components as the basically only technology used for what I would call leaf components.

Your application is made up of all these application components, but it has like a button in it and it has a slider and stuff like that. For now, these leaf components are typically rewritten every time someone makes a new framework, and then that just doesn't make any sense, right? You can encapsulate them as work components and get a much better usability across frameworks, and I think that's something that will absolutely happen and makes a lot of sense. We'll, somewhat ironically, lower the bar for new frameworks to enter the market, because right now there's a mode, because you have the best components, which also, by the way, is why some frameworks might not be super happy about this development. But basically I think that layer of the frontend stack is going to get commoditized by work components and frameworks will move up the stack to be kind of responsible for the application itself, but not for every last component. That's a thing, it's something is definitely going to happen and it's going to have an effect like you describe here, which is that because not everyone has to reinvent everything all the time you get much better quality, because people can focus on the stuff they actually are good at as an application framework.

[00:21:21] JM: Those leaf components, those are things like date pickers and buttons. We don't need to reinvent these things.

[00:21:29] MU: Yes, but we are very much reinventing them in a one to two-year cycle. We want to make them look different every year. That's fine, right? But then you don't have to like make them look different in five platforms.

[00:21:40] JM: Yes. So the reason that that happens today is because the leaf component idea is tightly coupled to however those leaf components are getting rendered and updated on the page?

[00:21:53] MU: Yeah, and basically they're implemented in terms of the component notion of the application framework. I think the web [inaudible 00:22:00] made two mistakes over the last three, four years. One was that web components were introduced and people gave them two jobs. One was here you can be something that you can reuse, and the other one was – And you should also use it to build applications. You did the same thing with all the application frameworks saying like, "This is really good for building applications and you can use it to make buttons if you want to," and that turned out to be, in my opinion, a mistake.

So web components very often sold for the good average is to make reusable leaf components, and application frameworks should stick to what they're good at what, which is making applications. So merging these two together I think is actually a very powerful combination, and I'm pretty bullish on that being very successful over the next few years.

[00:22:45] JM: You wrote this piece that was a tour de force. I've read a couple of times about managing a large JavaScript applications, and it suggests that there are things that can go very wrong when a JavaScript application gets big, and if you're not managing it with the mind to the scale of that application. What starts to go wrong when a JavaScript application gets big?

[00:23:12] MU: So I do want to say, and I think in the title of my talk and my medium piece, I put JavaScript in parentheses, because many of these things apply to all large applications. Why I did focus on JavaScript, because first of all I was at JSConf, where I game the talk. Second of all, because examples I'm giving are out of a JavaScript world. I think some of the learnings are universal, and in all honesty, I also copied the learnings from micro=service framework that Google was building and that our architecture is embedded into. So these are very much things that I basically ported from more stack independent systems to be very frontend specific.

I think in the talk, one of the biggest pattern that I have identified that goes wrong in a large applications, is if you have central configuration, so the universal things. In JavaScript, it might apply to the routes of the application, or it might apply to having a single CSS file. These are very concrete examples. But I think every application, at some point, has a notion of configuration. What I've seen is that a single file then it really has a few negative effects. One is that either this is a point of contention and of ownership. Who gets to make edits? People might not be feel empowered to make edits. It is a point of kind of a deterrent to deleting code, because you have this pretty isolated thing where you just want to get rid of it, but because there are the central configuration files that you have to also make edits and it makes simple change suddenly much more complex, and that deters deleting code, and code that is what really in the end leads to large applications, that there's lots of stuff in them that you really don't want.

Where this is a bit worse on the client side compared to the server side, is that if your service application gets too large in a very physical sense, you notice, because you suddenly have to buy more RAM. Maybe that's fine. Maybe you can buy more RAM. But if you're on the client side and you run out of RAM, then your application crashes and there's nothing you can do.

[SPONSOR MESSAGE]

[00:25:36] JM: This podcast is brought to you by wix.com. Build your website quickly with Wix. Wix code unites design features with advanced code capabilities, so you can build data-driven websites and professional web apps very quickly. You can store and manage unlimited data, you can create hundreds of dynamic pages, you can add repeating layouts, make custom forms, call external APIs and take full control of your sites functionality using Wix Code APIs and your own JavaScript. You don't need HTML or CSS.

With Wix codes, built-in database and IDE, you've got one click deployment that instantly updates all the content on your site and everything is SEO friendly. What about security and hosting and maintenance? Wix has you covered, so you can spend more time focusing on yourself and your clients.

If you're not a developer, it's not a problem. There's plenty that you can do without writing a lot of code, although of course if you are a developer, then you can do much more. You can explore all the resources on the Wix Code's site to learn more about web development wherever you are in your developer career. You can discover video tutorials, articles, code snippets, API references and a lively forum where you can get advanced tips from Wix Code experts.

Check it out for yourself at wicks.com/sed. That's wix.com/sed. You can get 10% off your premium plan while developing a website quickly for the web. To get that 10% off the premium plan and support Software Engineering Daily, go to wix.com/sed and see what you can do with Wix Code today.

[INTERVIEW CONTINUED]

[00:27:34] JM: You've written about some specific examples of heavy applications or large applications and how you manage them. One that is explored in that article is the Google search application and the fact that when you search for weather in San Francisco you get a little widget that pops up, and then if you search for convert currency from $5 to rubles, you will get another little widget that pops up and you know that the Google search application is intelligently managing the appearance of those different components, because there's probably thousands of little utility components that can potentially get rendered. How does that management work of all those nice little utilities that can be prompted to appear with just the Google search box?

[00:28:36] MU: I like this example, because it completely intuitively shows that if you just load it every possible widget to the client, then it will get primitively large, because you can always think of more. Yes, as you say, there're thousands of them, and many applications actually have cases like this. So what we do is very much that on the server side, when the search is assembled, some systems as, “Okay. We should this special widget,” in this case for that currency conversion, and then it’s included in the output and instructions are sent to the client to load the associated JavaScript, or in some cases it will be sent with the HTML response for that page.

Then as this widget renders, it kind of bootstraps itself and says, “I need these three components to actually make that pull-down menu work.” So kind of the application is initialized incrementally. So the way I would describe it is more of a pull pattern. So it's not that you have to statically configure an application with all that's in it, and then you load some subset of that to the client, it’s like the part of the users actually using is what's being loaded the client and usable. What this kind of leads to is something you can actually express very well in like O notation. In terms of algorithmic complexity.

In this world, the JavaScript code is actually loaded to the client that’s executed as a function of what the user is using as supposed to some other things. Is not a function of what we thought the user might use, or it’s not a function of just everything. So it’s just a function of what the user is using. In my experience, that's the only actually scalable way, because while apps get really big, the subset of that app that the user is actually using, is relatively constrained in any case I’ve seen. So it's always better to bias the JavaScript loading to the client based on what the

user is actually using, then just try to be smart and predicting it for them or even loading everything at once.

[00:30:55] JM: I want to shift our conversation to talking about AMP, and since we're already talking about loading results from search, maybe we could use that as a segue. So I search for articles all the time in Google Search, and sometimes those articles that I click on are extremely slow to load. Other times they're very fast to load. Sometimes they're fast to load because of AMP, the accelerated mobile pages project.

Explain what AMP is. What is the AMP project?

[00:31:32] MU: Yeah. Difficult question, because it's a lot of things. But very quickly, it's a web components library that form a consistent way how you can publish webpages on the internet and bundled with a validator, which if you can think of like a compiler that tells you whether your thing is good or bad, just like any compiler tells you have a compilation error or you don't. So the web components library combined with that compiler lead to a system where if the compiler says, "Everything's good," then we have very high confidence that the outcome is a fast loading webpage.

Then besides the loading, there's all kinds of other things. But I think that's the easiest to understand part, where you just say, "Validate your page, and if you're successful, then the page you have has very high likelihood of being a very fast experience.

[00:32:26] JM: Softwareengineeringdaily.com is a WordPress website. If I wanted to have AMP support for softwareengineeringdaily.com, what would I need to do? The best thing you can do is install the new WordPress plus plug-in that Google is working on together with Automatic and the company called XWP, which we invested a lot of resources recently.

So there was a WordPress plugin for a long time, but it made some really ugly pages, like a blue header and like no customization. So please don't do that. Saw the current version 1.0, or release candidate one, which I think came out yesterday, and it has a mode that's much more advanced where you can retain the entire like layout of your blog and is compatible with lots of

plugins and stuff like that. That's where the data turns a WordPress blog or website into an amp website.

[00:33:19] JM: Can you talk in more detail about the more general process of adding amp support to a webpage? What that involves?

[00:33:29] MU: Yeah. Maybe I should spend a little bit in terms of software engineering practices as well. So what AMP introduced was a different way to make webpages, and we could've told people like, "Hey! You should publish AMP instead of your normal webpage." But our experience is that that's not something people want. Also, certainly, when we first launched our developer preview, AMP was like 11 weeks old and it wasn't very stable. The New York Times shouldn't have switched everything they have, like to publishing app. That wouldn't have been a good idea at all. It'd be very bad idea at the time.

So we thought about how can we introduce a migration path so that folks can adopt AMP without throwing away everything they have. So what we did was basically say – We introduced this new link tag, link [inaudible 00:34:21] AMP HTML, where you can say, "Of this webpage, there's an AMP version and it's at this URL." So that's basically what you do. You make an AMP version of your webpage and then you link your so-called canonical page. The notion of the primary URL of any particular resource says, "The amp version is over there." I think that was very helpful and that enabled folks to just like kind of smooth migration. What we're seeing now much more is that there's a problem with this approach, which is if you have two versions of everything, then you have to maintain it twice and maybe they're both not very best they could be. And so what we're seeing a lot is that folks, now that AMP is three-years-old, say, it's actually working really well for me and I use it as the primary technology to build my webpage. But that's is one way to do it. We certainly still support the so-called paired mode, where you we have both something built not based on AMP and something built based on AMP.

[00:35:16] JM: Okay. So the old version of support, well, that people can still use, is I've got software engineeringdaily.com/podcasts/javaScript with Malte Ubl, and I might have a link rel AMP HTML. So I've got a specific tag on that page that tells a search crawler where my AMP supported page is located so that the search crawler can associate that, my original URL, with the AMP page, which I have somewhere else.

So let's look at this from the search crawler's point of view. So if the search crawler is – The Google search crawler is crawling softwareengineeringdaily.com and it finds one of my podcast episodes with an AMP page associated with it, what does the crawler do?

[00:36:12] MU: Yeah. It will also basically crawl that other version of your page and then associate those URLs to be in one canonical set, which isn't a new concept. I mean, it's very common for websites to have multiple version of the same URL. The most common one would be like having a mobile version or a desktop website. So the notion of having more than one URL for the same resource is something that just happens, and it's very common on the web. So this link type for AMP is just adding one new way to associate in another URL with that same canonical set. So from that point of view, it's not something very special for the search caller.

[00:36:52] JM: Then when does the AMP cache get involved in the process of putting that AMP page in a place where it can be accessed more quickly?

[00:37:04] MU: Yeah. Maybe I should step back a little bit. So one of the design properties of AMP is that it's designed so that pages don't make assumptions about the origin they're run-on, origins on the idea of schema plus a domain. So you can put them in different places and serve in an optimized way. So that could be like be a BitTorrent, or in a native app or just on a different URL. So we this concept of AMP caches which are effectively CDNs that serve AMP documents, and what is a big architectural shift on the web is that these AMP caches are owned and operated and associated with the platform that's referring traffic as supposed to the website that's posting the content.

So on Google links and AMP page, it goes with the Google AMP cache. What this does is it has one very obvious effect both that it's typically faster even in cases where folks have very nice CDNs, of which there are quite a few providers obviously. Because it's on Google's infrastructure and if you're already on google.com, you have an active connection that can be served through and so forth. So that's one fact of just a physical improvement of speed.

Then on the other side, what you get is something we call privacy preserving preloading, and that's actually key to the instant loading aspect of AMP that we haven't talked about yet. So AMP

can load pages faster than possible, let's say, by the speed of light, and the only way how that works, obviously, is that when you search for something on Google search, and Google thinks it's actually very high likelihood you will actually eventually click on a result, then the pages and resources are already preloaded in the background.

Then once you click the result, the browser doesn't actually have to go to the network to make that navigation, and which is obviously faster and which is effectively the only way to get to instant speed on the typical connection people are actually on.

The privacy preserving aspect that comes in here is that it's not cool and it's not expected on the web, but just because, for example, you search for something that, let's say number one search result, page learns your interests, because maybe you clicked the second link, which was Wikipedia and you didn't want the first one to learn that you're interested in this topic. So that's why we only preload from the mcache, which is a Google-controlled serving system and it doesn't leak any information to the publisher.

At this point, basically, Google already knows you searched for something. So the preloading doesn't leak any information to any third party. So we can like basically do this in a way that's both really fast, but also respects user privacy.

[00:39:49] JM: The spec for an AMP cache that you've described makes complete sense. It makes sense why that would be useful, but it also centralizes the notion of loading these accelerated mobile pages. Well, I guess, no. It only centralizes the notion of loading these AMP pages in a – This is just for a search specific application. So it would make sense for Google to maintain an AMP cache, because Google has a search application and you would want to do this kind of privacy preserving prefetching in response to a search query. There may be other kinds of mobile applications where you want lists of AMP pages that are served from different places other than a search index.

For example, I think Cloudflare has applications that they can serve AMP pages for. How would the usage of Cloudflare caching vary from the example you gave for Google Search doing AMP caching?

[00:40:58] MU: Yeah. I think you make a good point, and that centralization is actually not the right notion here, because what really happens is that specifically for traffic referred from entity X, that next hop is still served from entity X. With that, either user was already there, right? So there's no additional centralization going. You can't publish AMP documents by like publishing them to the mcache. You have to put them on the web. They always have to be on the web. They can't not be on the web, and I think it's a very important design feature of AMP. It's really only the idea that that serving is being kind of facilitated by the application you're using, and you can definitely think of other use cases where you might want something like this and it would make sense.

As you say Cloudflare actually offers an mcache, there is – Just actually a couple of weeks ago, being launched, there's obviously Bing's us cases very similar to Google's. You could imagine that apps, like news reading apps, Google News, Flipboard would take advantage of something like this and other places that refer a lot of traffic, like for example Reddit, might be very different use cases that kind of fall under the same idea.

[00:42:18] JM: So just to be clear, every page that has an AMP page associated with it, it's also going to have a normal URL associated with it. Is that right?

[00:42:30] MU: No, that's not right. You can use AMP just for your normal URL depending on your choice. Basically, that's what we talked about earlier, where you either pair them together or you just use Amp for the main URL.

[00:42:42] JM: Got it. There are some examples where I have loaded an AMP page that is still super slow. So the worst actors that I always deal with are recipe sites. I'll search for fried okra, and I'll click on the fried okra AMP page, because it's the top result, and I think it's going to load quickly, and it still loads extremely slowly. What's going on? What are they doing wrong on the recipe sites where AMP still loads slowly?

[00:43:15] MU: I don't know the concrete example. The main thing that you can certainly still do with AMP is just to have in a lots of images that are unnecessary and that may be hide the content. Otherwise, I really don't have good examples for why this might go wrong. I think we

certainly have seen a lot of success in the recipe space, as you say, has some issues and haven't seen it get much better.

[00:43:39] JM: I kind of glossed over how AMP actually works. So we should talk about that in more detail. Is all of the content from an AMP page – So if we're talking about the Google search example, is all of the content getting pre-fetched or is there still some content on the page that is not fetched until you actually click on the link?

[00:44:01] MU: Yeah, good question. So what we do is that AMP controls all resource loading. So it controls for each image, for each I-frame, for each video. The sub-external resources controls when they're loaded. What we'd use this and which is very common technique obviously on the internet, is we use it for lazy loading, so that resources that might be further down the page aren't loaded until you're more likely to actually scroll to them.

For the pre-rendered phase, we have a special version of this where only resources that match two requirements are actually pre-rendered. So the first one is only staff in the first uPort, which you can actually see post click where it'd be pre-rendered. I think that makes sense, right? You get the impression of everything being there, but everything below the full isn't there yet, and that kind of I think is the best tradeoff between resource usage, because there would be a lot of wasted bites if you load more images from the same page, so you can't see right away the user appearance.

The second thing is we only loads resources that we can load from that privacy preserving point of view that I explained earlier. What that really means is only resources that themselves can be loaded from the mcache. So AMP components have to opt in to being pre-renderable, by default they're not. So, for example, what you might get is the images are pre-renderable. For videos, we only pre-render the poster frame, but we don't the video. For I-frames, we can provide a placeholder, but by default we would do nothing. So you can have this degenerate AMP pages that are really just an I-frame. So those would be a good example of potentially having bad performance, where you load them, but they really just say, "Load that other webpage inside of me," and that will obviously be arbitrarily slow depending on how fast that I-frame goes.

[00:46:01] JM: That's sounds like what the recipe says you're doing.

[00:46:04] MU: Yeah, I wouldn't say that that's not impossible. For sure, yeah.

[00:46:09] JM: What are some other best practices for designing AMP pages? Maybe we could just talk about what kinds of components are typically found in AMP pages and how the construction and the design of an AMP page differs from a non-AMP page.

[00:46:28] MU: The main goal of AMP is that however you use it, the so-called HTTP waterfall to render the page should be as small as possible. So the way to think about that is the main enemy of fast loading is actually latency. It's not bandwidth. So even though on LTE, you're often in situations where your latency to talk to network is around, let's say, 400 milliseconds, and that's fine. 400 milliseconds is a fine latency if you need to make a single connection and get a response, but what you often see on the web are so called waterfalls, so where you fetch a file and the file says, "Oh! I actually need that other file." Then that file comes and says, "Oh no! I also need that other file." With these three depths waterfall, you have 1200 milliseconds, or already at one second load time. So vast majority of webpages on the internet have like this very long waterfalls. Again, every time you go through that full on latency, it just multiplies.

So AMP is designed to keep the maximum depth of the waterfall to be as small as possible. It depends on a few factors, but its worst-case three, and that is I think what really makes a difference, is that you can like do whatever you want to that page, but you'll never be worse than that. Then we're also trying to make that even better than this three, but that's kind of where we start. We have some optimizations of the mcache where we actually get it down to reliably only exactly one resource, which is the HTML page, right? Obviously you have to download the HTML page. There's nothing you can do, but that's kind of what we do to really optimize performance.

So that that's kind of the baseline, right? That's the baseline architecture, minimizing the depth of the HTTP waterfall at scale. You then have basically the whole web platform at your disposal. You can use HTML, you can use CSS. The one missing piece, at least for now, is there's no JavaScript that you can write. We're actually changing that, but maybe we can find some time later today to talk about what we're going to do. But for now, no JavaScript. If you need to implement it into your activity, you have two options. You can use either one of the AMP

extensions, which are basically web components. The other option is you use a technology that we call [inaudible 00:48:43], which is a simple way of adding interactivity to your page. Basically, you can listen to click events and change some state and then you can express in the page how your page should change in response to those state changes. So that's really what we do. It's normal web programming. You're just using those AMP extensions for functionality. I think that's basically it.

There's one other interesting – I don't really want to call limitation. So we only allow you to have 50 kilobytes of CSS per page, and I think it's an interesting limitation, and that 50 kilobytes is really enough for any page you might conceive. People are going to disagree with me on that, but it's quite a bit actually. If you handwrite 50 kilobytes, it's a lot of JavaScript, lots of CSS, but is not enough if you have a large website to have the same on every page. So it's kind of a forcing function to what I would call CSS hygiene. So you have to – If you want to do this at scale, you have to put a system in place that knows what CSS page actually needs and then limits the CSS to what the page actually needs or uses, and then you can easily make it under 50k.

So while this seems like a very static limitation, it really is just kind of a motivation for folks to really implement the system that has reasonable amounts of CSS at scale, and that's the kind of thing that we try to do with AMP, is to kind of put these little motivations into places where you could've done this all along. You can obviously do the same optimization to an online page, or for some reason people don't. In AMP, you just don't have a different choice, and obviously that then results in better outcomes.

[SPONSOR MESSAGE]

[00:50:28] JM: I have learned a ton from QCon. QCon San Francisco takes place this year, November 5th through 9th, 2018, and I will be going for my fourth year in a row. I always love going and seeing the talks, and in between the talks I hang out and eat some mixed nuts, chat with other engineering leadership about the latest talks and stuff that they're seeing, the 50 different stream processing systems that we're seeing, the different databases we're seeing, and QCon is a place where senior software developers and team leaders and managers and senior leaders, they all gather together, and you have fantastic conversations. You have

fantastic presentations, and it's extremely high quality. Its technical. A lot of it is also cultural and about management, and you can get $100 by using the code SED100.

QCon is a great learning experience. The presentations this year include 18 editorial tracks with more than 140 speakers from places like Uber, Google, Dropbox, Slack, Twitter. They are curated high-quality talks, and you can get $100 off if you use code SED100 at checkout for your ticket. Again, QCon San Francisco takes place November 5th through 9th, 2018, and the conference is the 4th through 7th, November 8th through 9th are the workshops. You can go to qconsf.com to find out more.

Thank you to QCon. If you are going to buy a ticket, please use code SED100, and we really appreciate the sponsorship of QCon.

[INTERVIEW CONTINUED]

[00:52:27] JM: Since you hinted at it, let's go there in terms of the JavaScript restrictions so you said if I heard you correctly. JavaScript is not usable in AMP pages today, but you would like to change that. You would like to allow people to add some dynamism to their amp pages. Why is there such a restriction on JavaScript and what are you doing to change those restrictions?

[00:52:52] MU: So the reason why there is this restrictions is once you add something that's touring completes to a system, then it's much hard to reason about the system, and AMP is all about being able to reason about the low properties of a system. Again, you can implement very sophisticated interactivity today, but people really want JavaScript and they should have it, like AMP buy-in, for example. It does have function, loops, if statements, or even classes. There's no abstractions, and obviously you need abstractions. So we do want to add JavaScript support.

What we've done and already with AMP buy-in is, one, people say like, "Wouldn't it suddenly load much slower?" We actually have very simple answer to this, which is we – With AMP buy-in today and with [inaudible 00:53:37] in the future, we won't allow the JavaScript to change the page when it loads. So you have to, on the server-side, render out everything you want, the way you want it, and then you can use JavaScript on the client to react to user action. So it, basically by definition, doesn't impact load performance, and then you can basically let the state machine

that you implemented be driven by user action, and I think that's a model that works really well for the vast majority of our pages. There are some things you can't implement with it, like Google docs would be very hard with like collaboration.

But in general, for the vast majority of webpages, this is a very good model. So I think it's a good step forward for us. I mean, we're taking this one step at a time, recurrently implementing the technology behind this, and we'll see how it goes.

One thing we're were betting on we actually created this technology called WorkerDom, which is a way to expose subset of webpages Dom into a web worker, which is a different threat that JavaScript runs in. So that's another thing we're doing, because it's in a different threat, it can't directly impact any of the performance aspects of the main page. Also because of the different threat, AMP gets to control when it starts and when it gets to accept messages from that threat, right? Basically, because these are shared nothing threats, the threat can say, "I would really like to like change that headline to a different font," and the page will just say, "Well, you didn't get a user action. I'm not accepting your change," just as one example.

[00:55:11] JM: That interactivity to the page – Sorry. Maybe you already said this, but does that prohibit network calls?

[00:55:18] MU: It doesn't prohibit network calls. What it does, what we currently not planning on doing at least. Again, like we're taking one step at a time, is if you have in a network call that's more of like event source, where you get events from the server, which you might want like for some kind of life updating feed, or you want some kind of collaborate editor, stuff like that. You couldn't build this with this technology, because you always need the user to click something to change the page.

But I think as a user, I think that's great. I think the page can change all at once, but it's always nice to have the user acknowledge that they want something changed. We have some specific technology for life feats and so forth, but those all have to be declarative so that we can kind of have a uniform UX. If you want to use JavaScript, basically you have to always wait for the user to do something and then you can react to it.

[00:56:07] JM: Now, one thing I know you wanted to mention was the change of governance in AMP. AMP is, I think, moving to a fully open source. Well, I guess it's always been open-source, but the governance model, the direction, the steering of the open-source project has now been delegated to a wide variety of the industry interests, I think, including Cloudflare and several people who are not affiliated with Google. So we could talk about that, but we only have like five minutes left, and I really wanted to ask you about web assembly. So I'll leave it up to you. Can we move on to web assembly or do you want to say anything else about governance?

[00:56:49] MU: I don't need to say much about it. I think the only thing I want to say is basically AMP was more successful than we ever imagined. So it was time to really make sure that it's not just like us caring about users and all the constituencies, but like them to get [inaudible 00:57:05], because it's such a big impact on the overall on the web. So we just wanted everyone to have a voice. Yeah, it's a deep topic. So let's talk about web assembly.

[00:57:14] JM: Great. We just did three or four shows about web assembly, and I am pretty excited about it. So I think people listening will probably have some perspective on what web assembly is. How do you think it will change the web and the way that applications on the web are built?

[00:57:32] MU: It'll be a process. So where it will have the most immediate impact is in the gaming sector, where it has direct applicability. We've seen, for example, AutoCAD ported to the web. So that's another obvious case, so where you have an existing code base written in language-X and you don't really want to rewrite it. So in that case, web assembly is a great technology. What I am definitely excited about is the future use cases that are yet to be discovered, because it's opening up a whole world of possibilities, and it's difficult to predict what that actually is going to be.

Just yesterday, Mozilla had a blog post that they massively reduced the time it takes to call between JavaScript and web assembly, and I think that is a game changer, because it enables to build applications that are more classic Dom-based web applications where the core application logic is living inside of web assembly. So I think that's the biggest development that I'm excited about. The way browsers work today, Mozilla announces yesterday, it puts a lot of pressure on other vendors. In my view, when you put on a two-year hat, that basically means

everyone will catch up. So it's time to plan for a world where every browser is fast with those respect and it really enables you start using Rust, or if you really want C++, or Go will build these type of applications.

[00:59:01] JM: Google has this history of on the backend languages, there are a set of blessed languages. I think it's like Go Lang, Python, maybe Dart, Java. There's some list of blessed languages that backend developers can use.

[00:59:22] MU: I think the most common ones are Java, C++ and Go.

[00:59:24] JM: Okay. Java, C++, Go. Now, I can imagine like web assembly might make it so that there are similar constraints necessary on the frontend, because you could conceivably use just about anything in a web assembly module on the frontend. Do you see that kind of explosion of potential frontend web application module compositions? Do you see that potentially happening, or I guess that would just be a slow slow transition anyway? So maybe not anything to worry about today.

[00:59:58] MU: I think what you can look at is android and NDK, the native development kit, because it's used but it's not that much used. You have a lot of applications in Java and Android, and then you have a few applications written in C++ or whatever someone liked, and I think for the time being, it's a good bet that the distribution will be very similar, if that makes sense.

[01:00:21] JM: Okay. I guess to close off, what do you think about progressive web apps? Because if you get web assembly, if you get AMP pages, really making the web up much better experience. Because we're still in this era where so many webpages, or just the loading experience is really awful. I'm glad that AMP kind of pushed people to really update their standards for how things look. But we are getting to a place where people really have all the tools they need to build progressive web apps, but progressing web apps still don't have too much traction, I don't think. Do you think they'll ever get traction?

[01:00:55] MU: I actually think they have a lot of traction. Where there was a bit of a mistake is if you ask five people what progressive web apps are, you get five different answers. So it wasn't the greatest, like branding exercise in the history of the web. It was basically just like new

technologies in 2016 that happened to be [inaudible 01:01:15] in that year. So I think there was a bit of a missing coherent vision, what that would actually be. Having said that, the way I usually group this is most of the stuff in the progressive web space are all about like making web applications as engaging and as re-engaging as what we've seen with native apps. So where there's lots of stuff, like notifications that pull people back into your app after you've tried it once.

I think that really is a very strong story, especially combined with AMP, where you have this like completely seamless onboarding where like, "Okay. You just have to click on this link and have the app done," and then bring the things that we learned from native apps or how to keep users engaging into that lifecycle. I think that combination is an absolute killer feature, and I can't see how it wouldn't be successful.

There certainly is a lot of room for improvement, is that the apps the people are building, while they perform from business point of view, I think there is a big gap in terms of user experience. So that's something I'm certainly very interested and how we can solve it at scale. We need to make it so that people say, "I want to build a PWA," and then just physically read the manual, follow the instructions and the outcome is as good as some native iOS app. I don't think we're there yet as a web community, where that kind of flow is possible. Right now, people like start, I want to build a PWA, and they start off being confused by all the tools and the manual stuff and that really like leads to the outcomes not being as great as they could be, and I think that's kind of the biggest challenge that we're facing right now.

[01:02:43] JM: Well said. Well, Malte Ubl, thank you for coming on Software Engineering Daily. It's been really great talking to you, and I look forward to seeing how the AMP project progresses and any other stuff you work on in the future.

[01:02:55] MU: Thanks for having me. This was great.

[END OF INTERVIEW]

[01:03:00] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the

focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance.

The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company. So thank you, DigitalOcean.

[END]