EPISODE 603

[INTRODUCTION]

[0:00:00.3] JM: Moore’s law states that the number of transistors in a dense integrated circuit doubles about every two years. Moore’s law is less like a law and more like an observation, or a prediction. Moore's law is ending. We can no longer fit an increasing amount of transistors in the same amount of space with a highly predictable rate. Dennard scaling is also coming to an end. Dennard scaling is the observation that as transistors get smaller, the power density stays constant. These changes in hardware trends have downstream effects for software engineers. Most importantly, power consumption becomes much more important.

As a software engineer, how does power consumption affect you? It means that inefficient software will either run more slowly, or cost more money relative to our expectations in the past. Whereas, software engineers writing code 15 years ago might have been comfortably able to project that their code would get significantly cheaper to run over time due to Hardware advances. The story is more complicated today.

Why is Moore's law ending? What kinds of predictable advances in technology can we still expect? John Hennessy is the chairman of Alphabet. In 2017, he won a Turing award along with David Patterson, for their work on RISC, which is the Reduced Instruction Set Compiler Architecture. From 2000 to 2016, he was the President of Stanford University. John joins the show to explore the future of computing. Well, we may not have predictable benefits in Moore's law, or Dennard scaling, we now have machine learning. It's hard to plot the advances of machine learning on any one chart, which we discussed in a recent episode with OpenAI, but we can say empirically that machine learning is working quite well in production.

If machine learning offers us such strong advances in computing, how can we change our hardware design process to make machine learning more efficient? As machine learning training workloads eat up more resources in a data center, engineers are developing domain-specific chips which are optimized for those machine learning workloads. The tensor processing unit, or TPU from Google is one such example. John mentioned that chips could become even more specialized within the domain of machine learning.

For example, you could imagine a chip that is specifically designed for a long short-term memory machine learning model. There are other domains where we could see specialized chips; drones, self-driving cars, wearable computers.

In this episode, John describes his perspective on the future of computing and offers some framework for how engineers can adapt to that future.

[SPONSOR MESSAGE]

[0:02:56.3] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes.

You can quickly provision clusters to be up and running in no time, while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked-in to any one vendor or resource. You can continue to work with the tools that you already know, so just helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications offline. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

To learn more about Azure Container Service and other Azure services, as well as receive a free e-book by Brendan Burns, go to aka.ms/sedaily. Brendan Burns is the creator of Kubernetes and his e-book is about some of the distributed systems design lessons that he has learned building Kubernetes. That e-book is available at aka.ms/sedaily.

[INTERVIEW]

[0:04:31.8] JM: John Hennessy, you're the chairman of Alphabet and the former President of Stanford. Welcome to Software Engineering Daily.

[0:04:37.0] JH: Thanks. Delighted to be here.

[0:04:39.1] JM: Moore's law is the observation that transistor density on a circuit doubles about every two years. We've been hearing for a while that Moore's law is ending. What does it really mean to say that Moore's law is ending?

[0:04:51.9] JH: That's a good question Jeff, because of course what it means for an exponential rule to end is not that it necessarily comes to abrupt halt, but that it begin slowing down. We've begun to see that slowed down, probably more acutely in the last decade or so, but we're facing a continuing slowdown that will affect the ability of designers to design next generation processors, for example.

[0:05:16.6] JM: Do we have an explanation for why that's happening?

[0:05:18.9] JH: I think we can see a set of factors. One is that the – some of the advantages that were gained by voltage scaling and other things are starting to hit fundamental physical limits. Obviously at some point, the number of electrons that you're switching has to be large enough that thermodynamic noise doesn't interfere with that. Otherwise, you get unpredictable results. We're getting closer and closer to that limit. We're also starting to see much more difficult manufacturing problems. One of the reasons things are slowing down is that the cost of a next-generation fab has gotten so expensive, that the time between deploying new fab lines is beginning to slow down, and that's causing a slowdown in Moore's Law.

[0:06:00.2] JM: This other trend that's ending is Dennard scaling. Dennard scaling is the observation that as transistors gets smaller, the power density has stayed constant, as Dennard scaling has been true. In a world with Dennard scaling, you can fit more transistors on the same surface area while consuming the same amount of power. This is unfortunately no longer the case. Now if we want more compute, we have to pay with more power. What's the consequence of the end of Dennard scaling?

[0:06:29.3] JH: Well, I think this is a really great question, because I think people talk a lot about the end of Moore's law. It's easy for people to understand that. Dennard scaling actually much more acute and is already upon us, fully upon us. Though what's really happening here is because the power density as you said was constant, it meant that the energy per computation was actually dropping at a rate equal to Moore's law basically, right?

Now we're in this situation where we've seen the end of Dennard scaling. In fact, it probably ended a decade ago. The result is the power consumption is soaring. Look, we have Intel microprocessors that slow their clock down, or actually even turn themselves off, or parts of the processor off. Why? Because otherwise, they're going to burn up, they're going to burn too much power, they're going to overheat, right?

That becomes a real critical factor. Obviously, when we begin to look at things like cellphones, or battery-powered things, then power is really important, right? You like your cellphone to last at least 24 hours between charges. If it only lasts three hours, remember the early cellphones, which you basically if you use them heavily during the day, they'd run out of power in the middle of the day? Well, that's the problem that we face. It requires a whole new approach to thinking about design in order to deal with that power problem.

[0:07:50.5] JM: What are some of the consequences of power consumption becoming more important? How does that change our design thinking?

[0:07:57.5] JH: Well, in several ways. It means that designers have to think about power and energy as their primary limitation. It also impacts lots of devices. In the age of the internet of things and ubiquitous processors, processors are everywhere. We like the notion and we can begin to see systems, which might operate on a battery for a period of 10 years off a battery charge, by using energy harvesting and other techniques. Well, that doesn't work if you can't tame the power consumption of the processor, or whatever other logic you have involved.

The other thing that people are surprised by is the impact of power consumption in the cloud is actually significant as well. Sure, the cloud everything is plugged in, but it turns out that getting the power in and the heat out is one of the biggest cost factors in building a large-scale data center.

[0:08:53.2] JM: How does that change the economics of the data center operator?

[0:08:57.2] JH: Oh, dramatically. I think if you look at a large Google data center for example, what you would discover is that the most expensive item is of course the servers. The second most expensive item is the electrical distribution network to get power in, and the cooling network to get the heat out. That's the second biggest cost item. Obviously, you have to continue to pay power, so your power bill has to continue to be paid. That's just the infrastructure cost. You've also got to pay the power bill.

The result being that power becomes a really key design parameter and using power and energy efficiently becomes a key factor. When you're talking about data centers that have a 100,000, 200,000, a million processors in them.

[0:09:42.7] JM: With that data center power consumption issue, we saw the creative approach of the machine learning with DeepMind being able to reduce the cost dramatically of a data center, the power consumption cost. Is there a similar opportunity for deep learning to reduce power consumption on smaller devices?

[0:10:05.1] JH: Possibly. You could possibly think about ways to do that. Perhaps an even more important approach might be to think about building hardware for machine learning, deep learning kinds of applications. The thing we've discovered about machine learning and the reason it took us quite a bit of time to actually make this technology viable was we didn't realize how much computation was required. We were probably off by a factor a 100,000 times when we were estimating what it would take to make machine learning a really viable, impressive technology that did creative human-like things.

Lots of compute power. If you have to provide that compute power using traditional general-purpose processors, it's very expensive, not only in terms of processor utilization, but also in terms of power. Now the question has become can you design, begin to design architectures which do the deep learning, machine learning kinds of things better and critically more power efficiently.

[0:11:09.5] JM: Computational improvements can also be achieved through parallelism. Your emphasis on energy efficiency in computing, how does that apply to our approaches to parallelism?

[0:11:21.3] JH: Well, it certainly applies there as well. I mean, we all recall from early days learning about Amdahl's law and the implications of what happens in a parallel application and the way in which a relatively small amount of serial code can dominate the execution time and dramatically reduce slowdown. Well another way of thinking about that is that's an inefficiency. If you've got an application that's parallel and runs most of the time on a hundred processors, but spends 10% of its time running on only one processor, it's only going to get a speed-up of at most 10. What you're going to have in that situation is you're going to have a 100 processors burning power, but you're only going to get the throughput of 10.

Again, this comes down to how to get the software and hardware to work collaboratively so that the efficiency is very high. The move to parallel computing and multi-core doesn't eliminate the need to pay attention to efficiency, it just shifts where that burden is between programmers and the architects. The programmers now get more of the burden of figuring out how they're going to ensure that Amdahl's law doesn't become a bottleneck for them.

[0:12:34.5] JM: I think we've set the stage at this point for the importance of looking more closely at our hardware and looking at how can we build hardware that is well-suited to the specific applications that we are trying to design at this point. Since we've laid that out, what are some hardware design principles for these domain-specific application architectures that designers could keep in mind? What are some of the principles they should keep in mind?

[0:13:05.9] JH: Well, I think the key principles are you first want to get an architecture that's a match with whatever the application characteristics look like. That's the key thing. We basically need a different design paradigm, rather than having the hardware and the software guys completely separated and maybe not talking to one another very often. We need a design paradigm where they're going to interact a lot more, where people who are application designers are going to work with people who build compilers and operating system technology and work with the underlying architects to reintegrate those things, to almost have a

Renaissance mankind of a Renaissance woman approach to attacking these problems. I think if you do that, then what you try to do is make sure that the hardware is being used efficiently.

Let me give you a simple example, right? Look at caches. Caches have worked wonderfully for so many years. They're a terrific idea. They work in architecture, but they also get used in operating systems, in other words we do caching, right? Caching has a downside. It has two downsides. One is when the application characteristics don't obey our normal modes of locality, then the caches can actually cause the program to run slower than it would have run if it were accessing main memory. Doesn't happen very often, but certainly can happen in some applications. Happens a lot with streaming, graphics kinds of things for example.

The other is caches have the same diminishing return property that other hardware optimizations have. Look at L3 caches now, they're on the order of 2, 4, 8 megabytes. Well, when the cache is close to the size of the program, it generates a lot of good improvements, but as the cache gets bigger you're actually using less and less of it, even though you're paying the energy cost and the silicon cost of this larger cache. You might ask, well does the app – can the application and the software people get together and engineer a more effective way to use that silicon than just building a cache out of it? That's what we're seeing people do with things like tensor processing units, or GPUs. I mean, we're seeing them rethink how they architect the machine and rethink how they use memory.

[0:15:29.4] JM: Will you talk about the emphasis of different categories of people getting in the room together? You've spent a lot of time in your career trying to bridge gaps between these interdisciplinary roles. You can imagine the straightforward way of doing this, get an electrical engineer in a room, get a computer scientist in a room, have them mediate their shared understanding. What are the things that have gone wrong in your experience trying to bring cross-disciplinary people together? What are the anti-patterns when you try to bring people from different disciplines together?

[0:16:02.7] JH: Well, I think you're exactly right, that sometimes things just don't work. They often don't work, because there's not enough common vocabulary. That's a starting point; people have to be willing to have a common vocabulary, or shared vocabulary, some notion of what the other people do. They've got to also have respect for what the roles of each one of the players in the team has. They all have – if they're designing something that stated they are, they all have a significant role to play, and they all have a lot of hard design work to do.

You've got to bring them together, and then you've got to have an environment that allows them to brainstorm. If you want that brainstorming of approaches to really succeed, you've got to be able to let people throw ideas out on the table. At first, just let's get all the ideas up on the board, and then let's start hashing through what's good or bad. It doesn't mean that you want an environment where people won't ask hard questions, or will challenge people, but you want one that works in a sense of in a respectful community, otherwise you're not going to be able to hold the group together and really drive it to where you want to go.

[0:17:08.1] JM: Talking more about these domain-specific architectures, different units on the chip what you're going to emphasize. You're talking about domain-specific architecture. This is not a – this is not like you have a specific application that you want this piece of hardware to accomplish, you want this piece of hardware to be able to accomplish a domain of applications. For example, you would want a TPU to be very good at doing a broad range of machine learning operations, rather than something as specific as a chip for image recognition. Why is that? Why wouldn't you want a chip just specifically for image recognition?

[0:17:52.2] JH: Well, certainly you can build a chip just for image recognition. I mean, I think the problem with building a custom silicon targeted at one application is, in order to justify that design cost and the fab cost and everything, you've got to be able to make a lot of versions of those, right? They don't need to be programmable. I think the right way to think about this is that different applications demand different approaches, right? Take your cellphone for example. Right inside your cellphone is a modem that communicates either with a community – you have Wi-Fi support, you also have supports of the cellular network.

That doesn't really need to be a programmable chip. It may have a little processor on the side to help direct stuff and deal with the fact of as you drive down the highway, you jump from one cell to the next, you've got to reconnect, it's got some stuff for that. You really want the core of the signal processing that's handling that cellular call and doing all the radio processing to be basically hardwired, or semi-hardwired anyway.

That's very different than let's say, deploying a data center where you know you're going to be doing machine learning, but you're probably going to be doing a wide range of machine learning programs. You want a programmable, but a domain-specific process, or processor targeted at doing deep learning content applications. I think we have both of them. Then you see this in the graphics community, right? GPUs are programmable, right? They don't just run one application, they run a variety of different applications, because you want your computer to both be useful for drawing nice pictures on the screen, but also maybe you want it to play games, so it's got to be able to do two different things and it's got to be able to be programmable.

[SPONSOR MESSAGE]

[0:19:45.3] JM: Citus Data can scale your PostgresSQL database horizontally. For many of you, your PostgresSQL database is the heart of your application. You chose PostgresSQL because you trust it. After all, PostgresSQL is battle-tested, trustworthy database software.

Are you spending more and more time dealing with scalability issues? Citus distributes your data and your queries across multiple nodes. Are your queries getting slow? Citus can parallelize your SQL queries across multiple nodes, dramatically speeding them up and giving you much lower latency. Are you worried about hitting the limits of single node PostgresSQL and not being able to grow your app, or having to spend your time on database infrastructure instead of creating new features for your application? Available as open source, as a database, as a service and as enterprise software, Citus makes it simple to shard PostgresSQL.

Go to citusdata.com/sedaily to learn more about how Citus transforms PostgresSQL into a distributed database. That's C-I-T-U-S-D-A-T-A.com/sedaily, citusdata.com/sedaily. Get back the time that you're spending on database operations. Companies like Algolia, Prosperworks and Cisco are all using Citus, so they no longer have to worry about scaling their database. Try it yourself at citusdata.com/sedaily. That's citusdata.com/sedaily.

Thank you to Citus Data for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:21:30.4] JM: Does this ever reach a world where we would want our fabs to be able to produce even more specialized chips? For example, 10 years ago I couldn't get a t-shirt custom printed for Software Engineering Daily, but today it's quite easy to get Software Engineering Daily custom printed t-shirts. Does the same apply to chips?

[0:21:52.5] JH: Yeah, so there's actually a new idea out there taken from the software world, called agile hardware design, named after the notion of agile software design, right? Where you, rather than put together a giant team and try to coordinate the whole thing and become the next mythical man-month disaster, you do smaller designs, you iterate more, you possibly prototype, so for hardware designers right now, you can prototype a lot of things in FPGAs, and you can even sometimes ship FPGAs. If they're not shipping in large volume and it's a limited amount of time, you can put it in a field programmable gate array, very fast design time.

The idea is this agile hardware development is a way to liberate more people to be able to design smaller chip. This isn't the way you're going to design a big Intel processor, right? It's the way you're going to design a smaller chip designed for a particular application space. I think, we see a need to improve that technology. I mean, hardware designers are still a long ways behind software designers in terms of the quality of their tools and their ability to get leverage. We've got to improve that, because with the explosion of internet of things and your cellphone wanting to have 50 different functions in it from being a camera, to being a listening device, to being a voice synthesis device, to being able to do machine learning, I think we're going to see an explosion in demand for these domain-specific architectures to tackle various problems. We've got to get the cost of the hardware design down.

[0:23:25.7] JM: You mentioned a number of different applications of camera, machine learning. What are the other categories that we will see domain-specific chips be built for?

[0:23:36.7] JH: Well, I think image recognition is clearly one of them. It's probably one of the most important; intelligent voice interpretation. Ideally, in a lot of environments you'd like to be able to distinguish one voice from another and be able to analyze sentences. You'd like to be able to put a device down in a conference room while there's a conference going on with a bunch of people and unravel a transcript that could tell who's speaking when and what they said and how that interaction actually happened, rather than something that runs it all together and doesn't distinguish.

I think there are a variety of different applications like this that are really exploding as we think about ways to do that. Of course, there's also all the navigation stuff; self-driving cars, these kinds of technologies I think which will continue to grow and blossom.

[0:24:25.0] JM: Which of these chips will be in the conference room with us and which of them will be in the data center?

[0:24:31.4] JH: My guess is that mostly what will be in the conference room is domain-specific chips for doing inference, right? Here we're not doing the – you're not going to try to do training in this environment. The problem with training, the training which is the other half of building a machine learning system, training is probably depending on the application, a 100 to 10,0000 times more computationally intensive. That's not something you're going to do in the local environment.

The other thing is inference in many applications, you can do inference with quite effectively with short integers, right? 8 and 16-bit quantities, while when you start talking about the training process, you need to do a lot of floating-point intensive computations, you do a calculation called back propagation. It's very computationally expensive. That's going to probably have to happen in the data center.

Although, we may build hybrid systems that maybe do a little bit of training every now and then, but mostly they do inference. I think we'll see how all those work out as the applications become more obvious. One of the interesting things about the computing profession is that, it's the people who invent new uses of computers that really change the world. All of us that are technologists, yeah, we're contributing to that because we're providing the technology that makes it possible. As that person that comes along and creates that new application that you never thought of before. The first time you see, right? The first time you see a windows-based system, you see WYSIWYG, you see pulldown editors, you see all these things. The first time I saw an iPhone I said, “Yeah, this is going to change the world.”

[0:26:12.3] JM: The inference chips, those will be fairly straightforward?

[0:26:17.2] JH: Yeah, fairly straightforward probably and probably very inexpensive. You can see the first generation in them in some of the new cellphones for doing image processing, so they're doing really intelligent imaging and everything from autofocus kinds of things that you don't even know that are happening. For example, rather than take one photograph, take several photographs and then decide which one is the one that's in focus. Well, how do you decide if something's in focus? A little bit of machine learning goes a long way for telling whether a face for example, is in focus, right?

[0:26:48.4] JM: These are not the kinds of chips that you're talking about, where we really need to rethink the architecture?

[0:26:53.6] JH: No. They're the first generation of machine learning chips. I think we'll see more and more as people get more experience trying this out and understanding how to make the tradeoffs. Even relatively simple machine learning inference problems, swamp conventional processors in terms of their efficiency and their power efficiency. We're already beginning to see, you want to do any reasonable amount of machine learning. You're going to have to probably build a domain-specific.

[0:27:23.5] JM: For a specific machine learning application.

[0:27:25.1] JH: Yeah, for a specific set, or a set of machine learning algorithms, right? I mean, if you're doing image recognition things, they're probably CNNs. If you're doing speech recognition and natural language processing, there may be LSTM. Different kinds of deep neural network structures that are appropriate for different kinds of applications. Probably will benefit by having domain-specific architectures that have some things in them that are specific to that class of neural networks.

[0:27:54.6] JM: Oh, so we could look at the different neural network topologies today and be able to maybe draw a mapping between what's going to lead to specific architectures? Ship architectures.

[0:28:04.1] JH: Yeah. Out of the architecture, the interesting question is to figure out how much of this is going to be done in software and how much is going to be done in architecture, and how do they go together, because in the end if I'm coding for example the structure of my neural network in something like Tensorflow, and I've got to get it into a domain-specific architecture, I've got to be able to go from that Tensorflow representation to the domain-specific representation. They've got to be close enough that I can bridge that gap efficiently. Right now, I think we're just beginning to understand how those things will work together.

[0:28:42.7] JM: If I was an enterprising compiler engineer today, what should I be working on?

[0:28:49.0] JH: Well, I think there are two interesting things to think about. The last 20 years in software development and software engineering has been about really creating productivity by going to more flexible programming languages, running scripting languages, going to Python, eliminating types, getting ease of reuse, and that has generated tremendous improvements in productivity, but it also generates programs which run a 1,000, a 100 times slower, a 100 to a 1,000 shall we say.

There have been some experiments. There's a nice little paper by some colleagues in MIT called this plenty of room at the top, where they point out that they can take a Python version of code and just rewriting it in to see diverted runs 47 times faster, just rewriting into C, without any specific optimization to the hardware yet.

One direction is to ask, is there a way to get all that productivity associated with these high-level scripting languages and yet get efficiency? Maybe just a little closer to see. Maybe it doesn't – for a compiler writer to be able to say I'm going to make your program run 10 times faster, you're a hero in the compiler world if you can do that. I think that's one direction.

The other direction will be figuring out how to build the compiler technology that goes from some high-level domain-specific language. Maybe it's Tensorflow, maybe it's something else, to the underlying hardware, in such a way that we preserve some portability. We want the architectures to be different and yet, you don't want to have to rewrite the code every time just because the architecture changes a little bit. The same problems that compiler writers have to solve today just solved in this domain-specific environment, rather than in a more general environment.

[0:30:34.0] JM: You alluded earlier to the fact that we had these machine learning techniques for a long time, but they didn't work with less computational power. Did we know back then that they were going to work with more computational power, or like why was it not obvious that, oh, if you just amp up the computational power, you're going to have better results.

[0:30:55.6] JH: Yeah. We had to amp up two things. We had to have a lot more data, because to initially train the network for supervised learning, which is what probably the biggest impact, well supervised from reinforcements learning. The amount of data that we needed for training, we didn't understand how much data was necessary. There we were off by a factor of a 100 to a 1,000. Then in order to do that training with a 100 to a 1,000 times more data, we needed 10,000 times more compute power.

I think people just didn't realize it. It was probably just a bridge too far initially, and people were not thinking, let's build the system that does this. Ed Feigenbaum, who shared the Turing Award for work, he had done way back when and knowledge-based AI systems made an interesting observation in to me. He said, what's happened with the rise of machine learning is rather than cognition, which is where a bunch of the AI people were, it's focused on recognition of extremely complex patterns. After all, deep learning is basically statistical analysis of things right at a very deep way.

It's based on recognition, and what's happened is it used to be recognition was regarded as a small part of the AI space. Now the deep learning techniques are making recognition-based approaches more and more and more of the solution into artificial intelligence problems. That's changing how we think about the field. Makes it interesting.

[0:32:23.9] JM: Yeah. I mean, I think about MapReduce, like MapReduce was do you hear about it on the first day and then you say, “Okay, this is a pretty useful parallelization technique,” but over time we realize how broadly applicable it is. How powerful it is and how flexible it is. I think if you are making the ball case for machine learning, it's that, oh, we can take these fairly straightforward classification statistical inference techniques and go very far with them.

The bare case would be that we're engineers, we're maybe getting overly excited about technology that's showing signs of working, but of course, need to be aware of our own human tendency towards bubble thinking and irrational exuberance. How do you guard against that? How do you know that you're not getting irrationally exuberant about the developments in machine learning?

[0:33:18.5] JH: Yeah. Well, I think there is some irrational exuberance about machine learning, and I think, to say that machine learning on its own gives you general AI is today anyway an overstatement, right? We don't have general AI, and I think my colleague Fei-Fei Li had a great way of explaining this. She said we have this great program for recognizing cats and cat breeds, but if you ask the program why is that a cat and not a dog, a question which a five-year-old can probably tell you the difference, right? Pointy ears versus not pointy ears, whiskers, furry, the program can't tell you why that's a cat and not a dog, right?

We still have a lot of work to do to get beyond what I would call statistical complex recognition of complex statistical patterns. We've got to figure out how to do that, and that's going to require a lot of work. It's probably going to require more focus even on reinforcement learning, because after all, people's brains are essentially reinforcement learning mechanisms. That'll just require us to rethink what we have to work on and what we have to do. I mean, it's an exciting time, because for a long time AI didn't produce the breakthroughs. Now it's producing the breakthroughs, where at the beginning there's obviously a lot of excitement about it. I think there's a now enough energy and focus and belief that this technology can work, that we're going to see significant ongoing progress.

[0:34:48.5] JM: When you spend enough time talking to the experts in the field, it's probably easier to get the right dose of temperance to –

[0:34:58.7] JH: Yeah, I think so. I mean, I think they are still enthusiastic. The people who are really working on thinking about general AI, they're thinking 10 to 20 years, 30 years. That's different than the situation I think a decade or two ago, where most people thought they weren't going to see it in their lifetime. I mean, we've already seen it. If you look at things from AlphaGo or AlphaZero playing either Go or chess, the program makes moves which by any measure were creative, and that's really a step forward in terms of what the system is doing. It's just the beginning. Game playing is only a small piece of the – small part of what you might call human intelligence, but we're beginning to get there and I think we're beginning to see things which the system begins to – maybe it doesn't pass the most rigorous Turing tests, but it passes a simple one perhaps, and that's an interesting step forward.

[0:35:58.0] JM: I think what's a good point to make about that result is that the fact that we could define a reward function in that domain made things a lot easier, and what you just said was that we need to get better at reinforcement learning, because if you can define this reward function, then it's the perfect abstraction over what you're trying to optimize for. Do you have any ideas for how we get better at defining reward functions for more complex operations like driving?

[0:36:26.1] JH: Yeah. We're going to have to do that. I mean, I think the challenge in driving is you've got to build in a layered set of decision-making processes, because obviously first and foremost you have to deal with safety, right? Don't get in an accident. You've got to build the system, do that avoid accidents. You've also got to build other sophisticated things in it. There's a great example, in California the law about crossing the double yellow line is rigid. It says, you shall not cross the double yellow line.

Now all of us have driven down the road and seen a situation where a big truck is pulled over to the side of the road and there's no way to pass that truck without crossing the double yellow line, so what do we all do? We all temp – we make sure nobody else is coming in the lane, we pull out, we cross the double yellow line, we get back in, right? We just violated the traffic laws actually. Now if I build my system too rigid, my self-driving car is going to come up behind that truck and it's going to sit there and it's going to sit there and it's going to sit there.

We've got to be able to build common-sense reasoning into that. I think back to when my colleague Sebastian Thrun was building Stanley and they won the DARPA Grand Challenge, one of the key things, the key thing they used machine learning for in the end was to decide how fast to drive when they were uncertain about what was ahead of them. They saw something in the camera, so it's further out than the Lidar can reach. You saw something you weren't sure what it was, maybe it was something blocking the road, maybe it was something else, and what they did was they watched what human drivers did. If human drivers began to slow down given that uncertainty over what was there and being prepared to stop, then they could learn from that process on what to do under uncertain circumstances.

I think it's that process of reinforcement, or supervised learning that enables you to then make that intelligent decision and begin to act in a way that's reasonable that exhibits what we think of as common-sense.

[0:38:28.6] JM: Okay. We start off the conversation talking about lower-level advancements we need to make to make things more efficient. This sounds like a class of problem that might be so difficult that we need better higher level abstractions that are better for the programmer, to have better ergonomics, because if you're trying to program judgment into a machine, do we have the right abstractions? Do we have the right languages of –

[0:38:54.7] JH: Yeah. I mean, I think we're going to have to figure out how to build them. I mean, I think it's the same, you know, there's a really interesting lesson, an analogy we could say is look what's happened with spam filter, right? If I had to define a set of rules as a programmer for distinguishing every piece of spam from several of the piece, this is a really hard piece of probe code to write, because yeah, some easy things I can figure out, but other things are more difficult, right?

On the other hand, if I have the benefit of all your classification of every single e-mail has come in this is spam, this is not spam, I can learn what you think of as spam. In fact, if I'm writing – if I'm the spam writing the spam filter for Gmail, I not only have all your knowledge, but I have everybody else's collective knowledge about what's spam, and I can build that in. We can create intelligent behavior by observing how other systems operate. That allows the programmer then to come up a level to use the data to write the – essentially, what you're doing is you're using the data and the classification of the data to write the code, rather than have to write all those rules by hand. If we have to write all the rules for common-sense reasoning in the world by hand, we're never going to get there.

[SPONSOR MESSAGE]

[0:40:21.9] JM: Today's episode of software engineering daily is sponsored by Datadog, a monitoring platform for cloud scale infrastructure and applications. Datadog provides dashboarding, alerting, application performance monitoring and log management in one tightly integrated platform, so that you can get end-to-end visibility quickly. It integrates seamlessly with AWS, so you can start monitoring EC2, RDS, ECS and all your other AWS services in minutes.

Visualize key metrics, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast. Try it yourself by starting a free 14-day trial today. Listeners of this podcast will also receive a free Datadog t-shirt. Go to softwareengineeringdaily.com/data dog to get that fuzzy, comfortable t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[0:41:28.8] JM: We've done a few shows about this idea that as we get self-driving cars, as we get consumer drones, as we get IoT becoming a reality and we have more and more devices around, these devices have spare compute cycles. When we get more and more devices, we will have more and more spare compute devices that are between us and the data center. If we could make use of those compute cycles, that would be great. We would have lower latency response times, perhaps we can offload some of that intensive training to those devices that are sitting outside your window, or sitting on the street outside to idle. How do we make use of those?

[0:42:10.1] JH: Well, certainly we can do that and some people have done it. I mean, there's been this folding at home right, where people are doing protein folding on people's home computers that they want to share their computers, right? My colleague, Professor [inaudible 0:42:22.2] did. I think the thing to remember in order to make this a viable approach, first thing you have to remember is that communicating data is the most energy expensive thing you do, so in order to make it effective for me to use a remote computing resource, I've got to be able to send it enough work with a small enough, in a small enough packet of information to make that viable.

That's why the folding at home works, right? I'm going to send you a description of a molecule that has a 100,000 atoms in and it's going to take a megabyte to describe that, but it's going to take you 10 hours of computation time once I give you that. There's a good trade-off in terms of computation to communication ratio. I think your other key point which is a great observation is because of the cost of that communication, to the extent that I can do get you an answer without having to go all the way up a hierarchy may be transmitted across the country through the internet. If I can get you an answer locally by caching or some other mechanism, it's going to be much more energy efficient than sending something all the way across the country and getting the data back from a far remote location. I think there are ideas like that bouncing around that people are thinking about how they use that caching in some form to achieve that.

[0:43:43.3] JM: You talk about building a bridge to the post silicon world. This is perhaps quantum, or biological. This bridge is important, because an interface with a quantum computer, or a biological computer is likely to be quite different from traditional computer interfaces. What are the bottlenecks to getting to post silicon computing being usable?

[0:44:07.9] JH: Well, I think the biggest bottleneck is you've got to get a technology that is reliable, scalable and general-purpose enough. That's the magic of silicon, right? These transistors are magic. They're just switches, they're the most basic logic element you could build, they scale like a bandit, you can have millions and millions of them. We've seen steady, steady progress year after year. You've got to get to whatever the next technology is going to be. It has to have some of those same properties.

I mean, quantum's got to scale and it's got to be useful for a wider range of applications that currently people at least are talking about. They're talking about a fairly narrow set of computationally hard applications, but it's going to be very hard to build a general-purpose computing structure unless we can do a lot of things with it. We've got to figure out what that replacement is. I mean, we probably still have a good 10 years' worth a silicon, maybe 15, maybe even 20. After that, we're going to hit really fundamental limits in our ability to push silicon forward. If our dreams about software continue to grow, which they have, it's the one thing that's been absolutely true in the history of computing, people's dreams of new applications and writing new software systems has been magical for providing great opportunities for users, assuming we're going to continue to do that, we've got to be able to build a hardware infrastructure that will enable us to run that software.

That's going to be a challenge and I think today, the interesting work is going on in the laboratories of physicists, right? They're trying to figure out what this next technology might be and how to make it work. Eventually, it'll find its way, something hopefully will find its way into the mainstream and become the next generation implementation technology.

[0:46:01.7] JM: It's quantum, I think I heard you say this quite promising for something like protein folding.

[0:46:07.9] JH: Yeah. Protein folding is a natural thing that relies on the – it does rely on the same properties that quantum relies on, so it could be used for protein folding problem. Whether or not you can justify building a lot of quantum computers just to do protein folding is a more difficult question and certainly, that's not what I do most of the day. I want a computer that does the things that I'm interested in as well.

[0:46:32.5] JM: Yeah. Now with biological computing, the promising result that I've seen is that you could store a lot of data in a small region of space.

[0:46:41.6] JH: Yeah, you can store a lot of data in a small region of space witness DNA and because it's truly three-dimensional and that's its big advantage, whether or not you can capture that, get the data in and out and get it stored consistently will be the question. Of course, there'll be other alternative more conventional technologies challenging it, right? So-called memristors, or phase change memories that promise at least considerably higher density than either DRAM, or even than flash possibly. There are some ideas out there floating around that people are investigating that I think could be another part of our computing technology spectrum.

[0:47:23.7] JM: Other developments in material science?

[0:47:25.9] JH: Yeah, developments in material science. These phase change materials are very interesting, because they have some of the same properties that flash have. Namely they're non-volatile, so you turn the power off, they maintain the state of whatever you stored, but you might well be able to make them even denser than flash. They also have properties, they also have advantages in terms of longevity. Flash has the problem that it actually wears out over time. Getting a technology, it doesn't wear out that has better properties for writes would make a big difference, and might mean that more and more systems would be shipped without magnetic disk and magnetic disk would be something that was very rarely used as the final archival storage, as opposed to something you use regularly.

[0:48:10.7] JM: What applications do you see for decentralized ledger technology?

[0:48:15.1] JH: I think this is an interesting technology, and cyber currency is just one small use of it; having ways to get decentralized agreement on what is truth and what is reality. I just heard this story last night about this being done with CryptoKitties and blockchain being used to maintain CryptoKitties so that I can prove that I own a particular CryptoKitty and I am the owner of it right, and everybody can agree on that. That's a interesting indication of your ability to do things.

I mean, I think we would like to have situations where we could verify transactions, we could verify ownership of things in a way that was far more distributed than it's traditionally been the case. They'll be good uses of that technology, I think to verify things. For example, enforcing copyright, worrying about multiple copies of things, who owns the copy of it? Where is the archival copy? That can all be done in a distributed manner using these technologies. We're just at the beginning, right? We've got the first implementations of it and we'll see how it continues to develop. Got to be able to make it efficient, but I think we now have real brains and computer science going into doing distributed ledger technology. People who understand algorithms in a deep way who will make it efficient and fast, which the first implementations weren't necessarily to be expected. That'll change its usefulness for lots of other things.

[0:49:46.7] JM: One observation about the development of decentralized ledger technology, I was going to ask you about the classic question, okay, industry is investing more in basic research, so what is the place of the academic institution for basic research at this point if the industries are investing in basic research? What's funny is cryptocurrency was neither industry nor academia and it looks a piece of basic research.

[0:50:14.5] JH: Yeah. I mean, I think there were other ways of doing distributed ledger agreement that we're out there, right? I mean, what blockchain represents a different approach to the problem, one that leveraged off of lots of work that had come before in terms of cryptography in particular, right? Complex hash functions, things like that. I think one of the nice things about our field is there's constant interaction between what's happening in industry and what's happening in the academy.

In some fields there's a big gap between what happens in industry and what happens in the academy. One of the great things about computer science information technology is that they're interacting all the time. In fact, people move back and forth and they have an appreciation for the roles that both the academy play and the roles that industry play, and that makes it a marvelous field to be in.

[0:51:09.2] JM: Yeah. If I was a person that wanted to develop a brain computer interface, if that was my dream, 10 years ago the only place I could have gone would be a place like Stanford. Today there are companies that are actually investing in that super far-flung research. What are the types of pro problems that are best suited for a student to come to Stanford and think about as opposed to going to an industry these days?

[0:51:33.8] JH: Yeah, that's a good question. I mean, I think the advantage of the university is that it's clean sheet of paper, lots of serendipity. I mean, think back when Larry and Sergey started working on what became Google eventually, lots of us thought web search, there's AltaVista out there, there's InfoSec, it's pretty well-done, doesn't seem to be a lot of opportunity there. Along comes Larry initially and then joined by Sergey and said, “We can do this much better. We can do this much better.”

It was right. I mean, I remember the first time I went to see the demo of Google and this is when it was still Stanford prototype, and I immediately said, “My God. This is so much better. It's so much better than what I used to get before.” I mean, AltaVista was great. I thought it was such an improvement. We had web crawling, we had all these things. The problem was I get back 10,000 hits in an order that wasn't particularly insightful about what I was really looking for and Google solved the problem, just changed the way the problem was solved. That's what universities are really great at. They're really great at that serendipity.

Both Dave and I, Dave Patterson and I, when we started what became our risk project set at Berkeley and Stanford, we started them with a graduate student class that was a brainstorming class. We started with a simple question. We said, “You know, computers are going to be built on a single chip and increasingly what's on a single chip is going to be a real computer, not just a little microcontroller or something. Should they be designed the same way they've always been designed, or should we rethink how we do the design? Does the single chip setting create a different set of tradeoffs?” The answer to that was you created a very different set of tradeoffs.

That was the right way to approach it in the academic setting and to think about the problem. I think they're complementary and I think there are lots of good things that'll go on in both places. One of the exciting things that we're seeing happen is these large companies, right? Google and Microsoft and Facebook and Amazon are investing in research, in ways that previously only IBM and Bell Labs did and now here come these new generation, the new generation of tech Titans and they're making these kinds of investments in fundamental work, everything from cryptography, to quantum computing, to machine learning, and that's really exciting to see, makes it exciting to be in the field.

[0:54:00.1] JM: You mentioned Google. You're now the chairman of Alphabet, so the role of chairman is to provide a leadership and independent counsel. What are the new skills you've tried to develop as chairman?

[0:54:12.3] JH: Well, I think clearly the question for any company is how does the board bring its expertise and knowledge to help the management team achieve what they're really trying to achieve in the company? We've got a board of people with a wide range of different experiences and backgrounds, and my job as chair is to try to bring their – help their knowledge and experience come to the forefront to help the management team really do – really do great things and think differently, and it provides an outside set of eyes. I mean, when you've got a management team that's so engaged as the Google Alphabet teams are, so engaged in driving things forward, getting that external perspective that’s maybe one step back and can say, this is not necessarily the best thing to do in the long-term, that's really important.

I think the thing we try to do, which certainly if you read the Google founders letter, you get this message that this is a company that wants to be here for the long-term, we're playing for the long-term. That's why we invest in quantum computing and machine learning and all these other technologies. Now machine learnings move very fast. It's moved from a technology that was a research investment to one that's now well into development, but that's exciting to see and I think we try to help the management team look for those opportunities and evaluate them and think long-term.

[0:55:41.3] JM: When I think about that role of chairman of Alphabet in contrast to something like president Sanford, I think of president of Stanford as a role where diplomacy is much more important. Not that diplomacy is not important as chairman of Alphabet, but when I hear Larry and Sergey talk, Sergey in particular you almost feel like diplomacy gets in the way of reality. Do you feel like you get to be less diplomatic and more straightforward? How does the angle of diplomacy compare to other leadership roles you've had?

[0:56:14.2] JH: That's an insightful question. I mean, clearly universities are different kinds of environments. You've got a different set of constituents. You naturally, things are generally more – you have higher visibility for actions inside the university, just the nature of the university, it tends to be a fairly open environment. I think it's slightly while Google Alphabet is not a very hierarchical company, universities have almost no hierarchy at all.

If you think about there's the president and provost, right? Their deans, their department chairs, and then their faculty, there's almost no levels to the hierarchy, right? The faculty are really the cornerstone in much the same ways that the engineering teams are the cornerstones in a company like Google or Microsoft, right? Your job as a university leader is to really enable the faculty and students to do great things. To learn, but also to do great research right, in a research university.

The objective function is a little more vague, in that Google's objective function is to do great things for its users, its shareholders and its employees. It's a little more complicated than a university, because you also have – not only do you have, to have the faculty and students and the staff, you also have a 100,000 alumni out there that care about the future of the university. For universities, this is different companies too, and I think increasingly true for all great organizations. Your reputation is really crucial.

For Stanford, that reputation is absolutely a crucial thing about the University, its reputation for excellence and quality and accuracy and truth, but it's also true of companies. People really want to know that when they get search results from Google, they're getting what the algorithm says is the very best answer, not corrupted by some external outside influence.

[0:58:14.3] JM: Last question, risk was a fairly heretical idea at the time. I think it was pretty against the grain. What are the heretical ideas in computer science today?

[0:58:27.0] JH: That's an interesting question. Yeah, risk was you're absolutely right, it was heretical. I mean, in fact I still remember situations where Dave and I were on panels and we were accused of doing harm to the computer industry by promoting these ideas. Yes, exactly. If you'd ask me this question five years ago, I would have said a big breakthrough in AI, but now that we've seen the big breakthrough in AI, it's there, I think they're heretical ideas are probably going to be a rethinking of how software systems and hardware systems work together.

One clearly heretical idea is going to be focusing on all the issues in security. How are we going to have solved security problems now that we've created all these meltdown specter, all these other gigantic security problems? I think we're going to have to dig out some ideas, which we threw, which we passed by in the 70s and 80s, that were ideas for solving security problems, and which we put on the back shelf. We didn't end up actually deploying and scale. My view is security is a really important problem. It's getting more important, it's going to require software designers and hardware designers to work together in a really radical new way, and that's something we've got to focus on. Given the trust, all these people trust us with all their data about their lives. We owe them a secure environment that provides whatever level of privacy they want and that's important.

[1:00:01.7] JM: John Hennessy, thanks for coming on Software Engineering Daily. It’s been great talking to you.

[1:00:04.9] JH: Thanks, Jeff. I've really enjoyed it.

[END OF INTERVIEW]

[1:00:09.7] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It’s open source and free to use and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plugins. Use the value stream map to visualize your end-to-end workflow. If you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on the fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team, who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations.

You can check it out for yourself at gocd.org/sedaily. Thank you so much to ThoughtWorks for being a long-time sponsor of Software Engineering Daily. We're proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]