RR007

What happens when Moore's law is over?

Capgemini

RR007

What happens when Moore’s law is over?

Disclaimer: Please be aware that this transcript from the Realities Remixed podcast has been automatically generated, so errors may occur.

Alright, Varen. Choose your vegetable. well, if you were a vegetable, what would you be? That's the end of now, isn't it? That could be the new\... I mean, what are you excited about doing next? Becoming a carrot. Yeah, if you're a vegetable, what vegetable would you be? (00:29.23)

I'm Dave Chapman. I'm Esmee van de Giessen, I'm Rob Kernahan. And this is Reality is Remixed, an original podcast from Capgemini. And this week we are going to talk about the hardware-software mismatch and how we are still living with limitations that were developed in the days of Alan Turing.

Now joining us later in the show to talk about this, I am delighted to say is an old friend and colleague of mine, Peter Richards, who advises startup and medium-sized tech companies and AJ Guillon, who's the founder of Yetiware. Now Yetiware specialize in developing parallel programming software and middleware designed to optimize software performance across heterogeneous processes.

Now. That's a mouthful, isn't it? It is indeed, David. And what I love is this interview is going to test your technical capability. I love it. I'm waiting closely I it will test many of us. Now, we go really deep on this one. Every now and again, we do do a bit of a tech deep dive on the show, but I think we've rarely been right to the heart of the compute processor in the way that we do in this episode, right, Rob? Yes. We're not going to be talking about gates and flips and flops and things, but there is a sort of issue with processing new style of highly parallelised computing. It's fuser. Indeed. Anthrophomorphise. And the sort of the developer is trained to be sequential and we understand programs in sequences of events and things. But when we do these highly complicated processes, it's very difficult to get efficiency out of the system. This has been an issue for developers for a long time where you don't know how your code is going to perform until you've plonked it where it needs to go and you turn it on, go, didn't do what I quite expected. And that's as much functionally and non-functionally. So I think sometimes it's a very specialized thing to get the non-functionals working well and we need all the help we can get. So looking forward to the conversation. Well, I think what's interesting about it is perhaps, I mean, it may be arguably even up to about five years ago, (02:40.706) this wasn't perhaps as big a problem as it's presenting right now. And the way it manifests itself is more and more critical, isn't it, as we're creating ever more power hungry kind of applications. It was always a problem for academics with extremely high performance computers. I think if you look back 20, 30 years, we were able to build, you remember the Cray systems and those big super computers. hugely parallel, needed very specialized code to make them work. But it was like very niche, very small group of people who were highly specialized in it. I think now with the way compute's going and the things we're doing and the thirst for compute for things like AI, this has now become almost commonplace and the average software developer is starting to be challenged by it. So I think it's just rising up the realms of everybody needs to get to be more capable with this sort of stuff. What are you making of all of this, especially given the view you normally take on what we talk about, is sort of got human centric and human system centric and things like that. This is a bit of a deeper dive technically, but how does this sort of subject resonate with you? Well, for me, as a transformation coach, I often say that my job is to help the system reveal itself. What if AI... Nope, Robert, steady on. on there. David, don't go there. David. Thanks guys. Okay, so what fascinates me is that modern software and AI might actually be starting to do something similar, like revealing the hidden structure of both technical but even also organizational systems. So what if AI and software and whatever is able to pinpoint what is actually not working in the physical world? And that fascinates me, like the organizational system, the human system, what if AI is actually able to see patterns that, you know, live out in the hardware instead of the other way around. So certainly, you know, one of the things that Yetiware, I think, work on is how you optimize software to run on very specific

types of hardware. Now, AJ can explain that sort of far more deeply and eloquently than we perhaps can. (04:54.552) But what you're talking about there reminded me of Conway's law. And for those who don't know Conway's law, like an organization effectively starts to represent its software architecture. It's IT architecture. Is that what you're thinking that AI could run across all of this stuff and start to identify how organizations are working despite itself sometimes? Yeah, it's mirroring, I think. So we see in the software now where it's actually, where the bottlenecks are in the physical world or in the organisational structure, what if we can let it help us identify where the bottlenecks are? That's an interesting idea, isn't it, where you're sort of using the analogy of getting software to work better with the hardware. You shine a light on the human and the organisation. Although, to be fair, with most organisational issues or structural disadvantages, Bleedingly obvious and we just go, why aren't you cheating? Okay, we're going to live with it. That's fine. Other factors are in politics and personalities and emotion creeping. Whereas software to hardware is purely logical. So it's easier to deal with when we find an issue. You mean like booking a hotel or an Airbnb? Well, maybe the producer told the analyst to say, yeah, we'll all go for a hotel. So I did. And then I get told I've done the wrong thing. I mean, that would be farcical organisational design, wouldn't it? Hey Marcel. Marcel's giggling for everyone who's listening. For once, I feel like I should leap and defend Marcel's honor. I know it's unusual, but I think there was an explicit conversation about what we were going to do, wasn't there, Dr Mike? He's nodding. He's nodding. just like... Anyway. Don't gaslight me. Don't stop gaslighting me that with your cabal and we never said that. I got an email that said, you need to book this today because that's what's been agreed. So I what I was So you automatically did it without any questioning. I'm compliant, Dave. I am compliant to the process. I was asked to do something. I undertook that process. Anyway, back to the subject of the day. I think one of the things that... (07:04.308) It really resonates with me about today's subject of this notion of very low level optimization and continual optimization is like we all know really that there is an issue with the sheer amount of power that's being consumed by the training of LLMs. that's rattling all the way back up the energy supply chain. It's having an impact on the sort of scale and size both physically and from a processing compute point of view of data centers. And it sort of feels like I haven't actually seen a graph. There must be show somebody's produced a graph that sort of sort of performance against the amount of energy it's sucking in. And there's a big conversation about whether it's worth it and whether the outcome is worth the amount of energy and power that's going into it. This feels to me like there could be like a real fix in here somewhere, doesn't there Rob? Yeah, so rather than constantly eke it out, that's sort more fundamental release of performance that resets it a little bit and maybe takes us back to a more or easier to deal with level. Because the pressure's on, isn't it? I mean, for a long time, we outsourced power to a way we tried to be more energy efficient, but now the AI race is pushing energy consumption up again. power grids take a long time to respond to that, generation takes a long time to respond. Maybe it's things like this that might make the big difference. think so. It has felt to me for a while that there's an element of brute force that's going on with LLMs that needed a smarter kind of... Sophisticated. Yeah, yes, a more sophisticated way to deal with the problem. And the same is in the chip design, right? There's the thing about the GPUs are good at AI, but they're not brilliant. They're still very powerful. mean, there's 100 kilowatt racks being installed in data centers. That is absolutely nuts. mean, the heat... there are very, very... I mean, if you remember the Hyperscale events, there are very, very clever engineers who were designing cooling systems, the liquid systems, to keep everything in, to be able to cope. (09:14.156) with the heat generated from the power. I heat loss itself is a waste of the energy and the power usage, maybe solutions like this that are going to make a difference, then we can go back to something a little bit more sensible. Or recycle in the same that they use the heat to warm houses. Yes. I think those are solutions that are a plus plus if you're able to do that. Which is the Nordic country where they're using... No, I've got that wrong edit point. I've got that

completely wrong. That's a bit of a disaster. got the wrong end of the stick, sorry. I was wondering where you were going to go with that. There's something about heat transfer that goes around the systems and it's linked into data centres, but I can't remember the facts. Excellent. Brilliant, isn't it? It's the best podcasting. That is the best podcasting you're to see in a while, Dave. Tell you what, here's a question. What's your favourite type of chips? Potato chips. sweet potato chips. Not sweet potato chips. Go away. the sweet potato chip is an abomination, right? It is a disaster. I don't know why people like this. It's like people over the age of eight eating ketchup. We've been here before, I know, but this is, I've got another thing about it. ketchup chips even. my God. Hang on a second. How does that work? It's really good. How does it work? No. tastes like ketchup. A chip. you mean a crisp? Yeah. Yes. So I was using the English vernacular of, you know, like a French fry as a chip. know, like a... Chips or French fries. Is that called a crisp? That's crisps. We call chips crisps. So we have tortilla chips and then we have crisps and then we have normal chips like as in fish and chips. Just to make life easy. Potato chips. It's a nuanced language. It's a nuanced language. I don't think we've got to the bottom of this. Sweet potato. Why do need, why do chips need to be sweet? Stop making all your food sweet. Where do you, where do you stand Esme on the sweet potato? It's way more healthy. So yeah, sweet potato chip. When I feel like I've been, you know, I need to adjust my food, just to get things in balance. Yeah. Then sweet potato chips is better. But just don't eat chips. If you don't, if you don't want the unhealthiness of chips, just don't eat them. It's fine. (11:32.088) Just avoid them. You have to. It's better. You're talking a lot about the placebo pills or whatever you take or drinks just to make you feel better, right? Here's your answer. Go for real vegetables. And we actually had this conversation lately. Like what type of vegetable are you? thinking. You are actually a sweet potato. I've never seen Robert as a sweet potato. I see Robert Moore as a sprout. A sprout if you were a vegetable god. Well, mean, this conversation's taken a turn in a direction I wasn't expecting when I came onto this podcast. Well, we knew the episode was about chips, right? Chips, my word. just... Right, okay. Now I think we've got to the bottom of that debate. So anything more on this? I think I need to sit down for a while and process the fact that I'm a Brussels sprout. Yeah, well, you've done very well. Popular at Christmas, Brussels sprouts. Yeah, I know. That's the only time they're popular. That's true. That's true. All right, back to the subject in hand. One of the things I think is particularly fascinating about this conversation is how far back it reaches into the history of compute. And Alan Turing is a fascinating character for anybody who hasn't seen much on Turing or read much about Turing. There's a good film called The Imitation Game with Cumberbatch plays Turing, think, in that. Some hideous inaccuracies in it and whole characters removed, but it is quite a good drama. Robert, you said you... I mean, that felt like you weren't approving. I am a fan of the man and his background. I went to Manchester University where he studied and created all the good AI stuff that he did and I'm a fan of Bletchley Park and the tour. let's just say that story has some editorial changes that leave out quite significant things. Interesting. Well, we'll leave that there again. But on Turing specifically, yet the father of the modern computer, would you say? (13:47.178) Yeah, well he was there in the department when they created what was called The Baby, which was the first stored program computer, which is basically the genesis of what we know as computing today. So yes, the father of computing is a good term, although again at Manchester there were lots of other very clever professors involved in that, but his name is synonymous with that process. If you go back to your time as being a student, what would have helped you back then in terms of software development or knowing what you're doing. the biggest bit of advice going from academia to industry is stop seeking perfection. Everything's slightly messy. It's got to work and it's got to create the outcome, right? So at university you get told to be perfect about your analysis and your approach. you arrive in industry and everything's imperfect. And that for me was, it took me a while to realize that I was seeking to be too good at creating software when it just had to work and get out of the door and create the next sequence. It had to work, it had to be available and

performant and all that good stuff, but I kept trying to refine it and refine it and refine it until I realized that actually good enough is often good enough. Don't always think the perfection is the answer and you'll waste a lot of your time doing things that don't release value basically and getting your head around that I think was a bit of a difficult thing for me so that's the one bit of advice I would give to anybody in the situation. so let's jump to our conversation now with AJ and Peter from Yetiware where they're going to walk us through a conversation that kind of brings a lot of this in from Alan Turing to AI and what they might be doing about it. (15:36.878)

All right, and we are here with AJ and Peter from Yetiware. AJ, how are you doing today? I'm great, very good. Very good, thanks. I understand you've got snow where you are. Yeah, 15 centimeters. We're enduring the winter still. 15 centimeters. Rob, what's that in inches? You remember your school ruler, divide it in two. There you go, that'll do about six. 2.2 something, anyway. Look at that. It's like fast, it? It's like a calculator. It's like razor sharp. Peter, how are you doing? Sun streaming in behind you? It is, yeah. How is life on Barbados? I'll let you know when I get there. He's actually on the beach with a pina colada as we record. exactly. Well, look, let's start the journey of unpacking what Yetiware is and does. So AJ, take us back to the beginning of your observations that ultimately led to the founding of Yetiware. Yeah, so I mean, I was just like really weird, awkward teenager who went to university and kept my roommate up all night with weird old computers that nobody cared about. And a couple of years later, I found on eBay this titanium server that was going really cheap. And I didn't really ask myself why nobody wanted to buy an titanium server at the time. And so here's this like really cool computer, it around 2002, 2003, that nobody could figure out. how to program this thing. And to me it was kind of puzzling because you can boot Linux on it. As you know, you have a C compiler. So why is it everyone's saying you can't program it? And so that kind of curiosity led me to really focus my career on what became the hardware software mismatch. We're producing this amazing hardware, but we can't figure out how to program it. And then I was fortunate enough to come all across with GPUs around 2007. And here's another program, something, another processor that fundamentally really hard to program and it's, you know, parallel and we can't figure it out. And so I started to really hear kind of these whispers in the hallways that nobody really wanted to talk about, which is what happens when Moore's law ends? And so this was kind of a question of what's the software response when you have this hardware chaos and things are changing all the time under you. How do you actually get an application to keep running? How do you get resiliency that you can port to, you know, new stuff coming out? (18:03.77) And what's interesting about the current market is we're kind of repeating the mistakes of ITAM. We have all of these really unprogrammable processors like NPUs, AI things, things are doing in memory computation. And this mission of how do we actually continue to write software, not have to refactor it and keep running after hardware is continuously changing drove me to today. And now it's kind of this. you know, emergency need like, no, Moore's law is dead. What do we do? And all these physical laws are kind of coming to an end or flatlining. And that's how we came to this journey was actually 20 years ago and a lot of work to figure out what's the world going to be like after today. perhaps before we get into Yetiware and the solutions that you guys are coming up with, let's just zoom in a little bit for those who might not be familiar exactly with the, the the software into hardware stack. When you talk about they're not programmable or they're not programmable effectively or efficiently enough, just describe the layers of things that are going on there and where the main bust is, if you like. Yeah. So it should be clear there that when you talk about something not being programmable, what you get from the hardware vendor is they say, this thing has amazing performance. And then what you get when you run your application is one one thousandth of that. And so what I should be clear on is that it's number one, very challenging

to write your code in the form required for that processor. Anyone who's done GPU computing will tell you it's actually really hard to restructure your algorithm, your memory, all this stuff to get performance out. And it really is about how do you get the theoretical potential in the sticker that other people can get versus you in your office trying to figure out how to not get an application shipped while you have this other queue of bugs to go through and stuff. And AJ, from a perspective of... So, software developers... I'm a bit older, so I remember maybe having to learn the hardware, but software developers are getting very used to and trained up on abstraction. So there's this thing that says, I don't have to care about the hardware, I just have access to compute, off I go. And now, occasionally we have to go, no, but you have to really be aware of how it works. You have to get into that flow state of everything working together harmoniously, so it flows around the system effectively across the full stack. (20:23.574) It sounds like you're moving into that to continue to enable the abstraction and keep the simplicity, but actually be aware of getting the best out of the hardware underneath. Is that the sort of strategy that you're starting to move towards? Yeah, absolutely. I mean, if you have to write low level code for every hardware device, like you're never going to come back. It's extremely complicated. And those abstractions are important because as much as we talked about the software hardware interface, the application developers kind of the customer interface and they change their mind. They want different features. They change things and If you tell them, it's gonna take me two years to refactor your code because I'm not getting the performance out, they're just gonna say, you know what, we're kind of good with performance being relatively slow. So the trick is that we need to have those abstractions. They have an important need for software engineers. But how do we allow you to have your object-oriented architecture, your service-oriented architecture, all these abstractions, while allowing the... processor to give you good performance. And that's the really hard part. And if you look at how we got here, historically, people would exchange notes and books on low-level tricks to improve code. Then we had the 80s and 90s where we had this huge surge of improvement. And the software developer said, you know what, I'm just going to use tools that are kind of wasteful because the processor will kind of make up for me in the next generation. And they never really got the memo that that ended. 20 years ago, we just kept going with these layers of abstractions. And so now you look at like, you have this huge GPU that's doing this stuff and you have layers and layers and layers and layers of stuff to actually encode something to it. And I've seen projects where you're spending, you know, one second encoding data and then the GPU is like less than a millisecond, right? So you have this huge tower of abstraction that has to then get to the fast processor. Do you have a sort of a B-hack, like a big hairy audacious goal? if you really get going to if that ever was possible, is there something that you're like so intrigued by what drives you day in day out? Well, the big audacious goal I have is to figure out, you know, how do we write software in one way so that for the next hundred years we don't have to really care about how hardware changes. (22:36.93) because we know that hardware is gonna change all of the time. The chaos is already here, it's continuing. And my goal is, is there some way that we write software, is there something we're missing that we write it one time in a simple format and it just keeps working? Historically, we had that. You're still running code on sequential machines from Fortran from the 70s and 80s and stuff like this. Cobalt's still here and going strong, right? But in parallel, we can't do that. And that became a mystery. So the audacious goal is, how can we build something so that you write software once and for the next 20, 30 years, 100 years, matter how hardware is changing, it just keeps working and works efficiently on the target hardware. One other way of thinking about this or one other way of stating it is, how do we get to a point where software does the optimization for you in a way which gets good efficiency out of whatever hardware happens to be around and what you're running. Would you say that? True, AJ? Yeah, absolutely. The inefficiency that's here is caused at what point in the stack for you, like when the problem started, was it, if I, I dummies guide this a bit, for me at least, and you know, correct me where I get this wrong, cause I probably

will, you have these abstracted programming languages, you then compile that your code into something then that can then run on that particular processor. And then, you know, the compiled code then, whether it's assembler or MOS fiscated versions of such a thing is then driving the CPU. the problem in the abstraction? it in the compilation or is it in the, how the underlying kind of low level language has been? has been written way back in the day that we're still belaboring today. If that makes sense, if I've managed to piece that together a bit. Yeah. So let's look at that from, you know, what is the problem and what's kind of the solution and what's the underlying, you know, root cause versus the symptoms, right? So what you described is exactly how a 486 works. And that's the last processor that kind of works how programmers think processors work, right? (24:51.8) then what happened is to get more and more performance out of sequential programs, which you write stuff in a certain order and it just goes through in sequential order. We added caches, out of order execution, all sorts of complexity to make that go as fast as possible. But when you pivot to this kind of parallel programming model, where you have multiple things happening at the same time, humans do that really naturally. If you're in a grocery store, you just kind of switch lines. If we're in a grocery store and see this long line, we divide it and a new cache year comes and we kind of parallelize like that. But for processors, it's really hard to make software do that. So the inefficiency partially comes from the fact that we really super optimized a sequential model. And we pivot to hardware, this becomes inherently parallel. Everything that makes sequential software fast makes it very hard to write in parallel, if that makes sense. So there's this mismatch between the two worlds. and that mismatch makes it causes the first source of inefficiency. The second is that compute actually isn't the problem anymore. Compute's actually relatively free. It's memory transfers, the amount of time it takes to get data from one place to the other. And so our programming languages or abstractions weren't really designed to help us figure out how to change the memory structure of an application. And for developers, we're happy just to get it running at all. Now we said, well, you're have to reorganize it. for every potential processor, it becomes very complicated. So the first source of, it's not necessarily the abstractions that are the inefficiency, it's a mismatch between this concept of a sequential world and a parallel world, and now an increasingly heterogeneous world where you have specialized processors, and this mismatch of the fact that we don't really care about just optimizing operations, we care about optimizing the memory structure because the rate of memory transfer hasn't really changed in what, like 10 years. So AJ, there's obviously an awful lot to do to try and get from the problem issue that you're trying to deal with to some form of usable solutions. So tell me about the team that you've pulled together in Yeti where you had to sort of help with this. Yeah, so this is kind of this grand challenge problem of how do we change the compute stack? And it's not something that one person can do because you may kind of fool yourself and be wrong with how you're approaching the problem, you need new ideas. (27:15.98) So over time, we brought together a whole bunch of people, including Paul Chow, who was one of the, on FPGAs and worked with Hennessey and Stanford on RISC. We brought in Maurice Lerhy, who built a lot of underlying concurrent theory. He's won the Dykstra Prize three times, other Dykstra Prize winners, and people who created OpenCL, C++, OpenMP. and kind of create the modern world we have now to say, would we do it better? What can we fix? How do we change things? And so those advisors have worked with us very closely to critique each step of our development, to make sure that our architecture makes sense and now to help figure out how do we actually scale this up and build it. there anything, I mean, that's obviously an august body of individuals. Was there anything that they said that you felt particularly burnt by? like, the... The guy who invented C++ has just called me on this. What was the worst one? There must be one. You know, I would think it'd be easier to make forward progress than to have to go in a room with everybody who created everything and tell them why they're wrong. It's a bold opening. I love it. I know you're all experts in your field, but you're a bit shit. This was the most, I spent a decade actually working

on the open CL standard. and working on specifications, there was the same feeling, you know, tell every major employer who may ever give you a job why they're completely wrong. Right. And so, yeah, I mean, there are many things. What we started with, everyone went through this kind of same stages. You know, I'm kind of tolerating you, but that can't possibly work as heresy to say that turning is maybe not quite right. And the right approach for programming and so on. So. Everyone was like, yeah, but I can't see a reason why it won't work, but there has to be something. And then it became this kind of exhaustion of I can't see a single reason, no matter how much I've complained about it, that it's not going to work. And that's what we call the engineering process, right? There still may be issues with our work, but we've looked at it very aggressively and there were issues that were found and we had to fix them. (29:31.886) And that's exactly the process we followed. We tried to follow this methodically and scientifically by bringing in ever increasingly smart people and understanding, you what should we do differently? I think, you know, to me, one of the things which sort of confirm we're ready for commercial process now is the last three people that joined. Right. So there's a guy called Tim Mattson who for years was Intel's leader for parallel programming. And he's been one of the fiercest critics of what we've been doing. And he's Now believes that he doesn't know why it won't work. Right. there's a guy called Michael Wong, who was at CodePlay and at Intel and has been, you know, in the center of, um, all sorts of standards around language, including C++. And through Michael, we've now brought on Bjorn Struestrip, who was the inventor of C++. And, you know, they're all sort of working with us now on ensuring that. that this is both valid and commercial. And so to me, that was a very big tick in the box that we're ready to go. Now, Peter, to the casual listener, this might seem like very low level detail and very technical, but why is this important for businesses? how does this roll up to then causing us problems either for businesses or industries or even more broadly? Well, I think you sort of touched on it a little bit, right? There are orders of magnitude delays in the way that processes and programs work today. And so if you can gain some of that efficiency back, suddenly the cost of utilizing compute goes down, right? The footprint that you need for data centers goes down. (31:43.054) the things that you can do with the same amount of energy, the same amount of funding goes up, right? So if we improve the low levels of efficiency, then it's beneficial everywhere. I used to run very large farms of processors to do risk analytics at various large banks. And it was frustrating that we were using one, 2 % of the capability. and you would do overnight runs, whereas you could really do sort of 10 minute runs. And so you were constrained, right? And you weren't constrained by necessarily the investment that you were making, you were constrained by the fundamentals. So things had to change or things have to change. But it's, you know, it would be valuable everywhere. It'd be good for your phone. It'd be good for, you know, home entertainment. If you increase efficiency, it's a good thing. Yeah. And of course, one of the biggest challenges that we're facing at the moment that is changing energy policy even is how power hungry and processing hungry and memory hungry AI is of course. And you you'll see in the energy world, for example, nuclear is now back on the radar as it's a, you know, a source of clean power and can better harness now that wasn't there. necessarily as very visible in the conversation pre-AI. And it's all because, isn't it, of just having to layer up data centers in the way you were describing in your analytics days? Yeah, exactly. It has become more imperative that we gain efficiency. And the way that the Yetiware is approaching this is really quite revolutionary, but it's one which once adopted will be beneficial for many, many years. But you'll also see there are many companies that are focused on incremental improvements. and so you have lots of places where they're doing very good focused work, but you're getting percentage improvements rather than orders of magnitude improvement. Because you actually can't fix the fundamentals by an incremental improvement. It's a bit like the paradigm with battery technology. (34:08.994) which is they've been eking out little tiny gains in battery technology for years and years years and years and years. But the fundamentals of what happened with lithium ion technology hasn't really

advanced. And everybody's waiting for, when we talk about electrification of cars and transport generally, it needs that revolution. So many कंपनियाँ have been saying, yes, we've got it, we've got it, we've got it. know, the charge state that's really can be acquired really quickly and the 1,000 mile range car, et cetera. But it's not arrived yet because it's just like there's a breakthrough missing. And it feels like in this particular conversation, we're getting towards that breakthrough of really bringing software much more optimized on top of the hardware and get that efficiency and performance. I think that's a very good example, right? Where you can also think it is climbing mountain ranges, right? So you can go in the foothills and you can ascend a summit. and you're always going up and you get to the top and there's nowhere else to go. Right. You actually have to go down to get up higher. Right. And that we're in the evolutionary part of just climbing up a little bit higher on the summit. the height of that summit is like a hundred times lower than where you should be. And therefore you have to go down to get up. What is the effect on the, on the engineers? Do we see it? Is this being, is it a different scale or? different mindsets. AJ. Yeah, so let me add to the energy efficiency that we talked about these abstractions, but software engineers, we're really just concerned about getting something to market, getting it to work at all, and then moving on to the next feature. And now the concept that the abstractions, these layers actually have a cost, an energy cost, a carbon footprint, is something I think you'll find is very foreign to most software developers. Let's face it, we're building these data centers (36:30.766) to really run one or two applications. And even there, we're struggling to get efficiency of the hardware. So what hope do you have in most enterprise businesses to get the software efficiencies out where these trilling dollar companies can't figure it out? And on energy efficiency, what's interesting isn't just how to get the performance out, but it's the commoditization of hardware. If we're in the same situation where we had one car that everyone had to buy, the miles per gallon would be a footnote at the back that this is kind of useful for you. But because this is the only car, there's no incentive to improve your fuel efficiency. When you commoditize hardware, when we're able to be completely portable that this quarter I can swap out a new device for this device, all of a sudden there's now a competitive incentive for hardware developers to push that energy efficiency up and it can't push your prices down. But right now you kind of get what you get from the vendor. It's on the sticker, you can read it. But there's no incentive for competition because you'd have to rewrite your entire compute stack. And we haven't even been able to do that for AI with the enormous investments. haven't succeeded by that. Well, I feel like I need some form of analogy to keep going in this conversation because we've had mountains, cars, batteries. So I feel like before the end of the show, as both of us are on point to try and come up with our own analogies, but before we get to that, I think what I'm hearing in in the conversation AJ is revolution or evolution here. So give us an insight into how then you've gone about attempting to solve this problem at Yetiwa. Yeah. So if we look at how we wound up in the world we are today, we've known for a long time that hardware is going to change in the ways that it has changed. You heterogeneous computing isn't new. It's from around 2005, like that. GPU computing isn't new. So question of what's kind of a better model in As an industry, we've invested the billions of dollars into this concept of automatic parallelization. How do we take simple sequential code and make it just kind of work? And so, there's been all of these efforts. I have a bookcase full of ideas for how do we possibly make it easy to program all of these processors? And so, the question is, how do you do that? And when you go back and look at what all of these things had in common, the insight we had is that It's not possible because we need to be able to analyze code. We need to be able to look at a program and understand before we run it, what resources is it going to consume? And so the root insight is how do we build software so that we can actually analyze what that code is going to do? How much energy is it going to use? What's the AWS bill going to be if I run it on this cloud system? How much memory and so on? So the key insight is how do we make software to be analyzable? (38:53.196) So that once we have that, we understand the rest of how we actually

map to hardware and so on. What does the product look like? Is the product the analyzer of the code that then ultimately does what with it? And how does understanding it like that make it portable in the way that you describe? Yeah, so let's take a step back to say that it's not possible with current methods to do what we want to do. And the reason is that all of computing, all of how we build software is really based on pillars of two pioneers, Alan Turing and John Boyd Neumann. And what Turing did is he established how software should roughly work. You he talked about languages being Turing complete and all that stuff. And we have these really powerful languages. And von Neumann said, yeah, but software developers should be explicitly in control of memory movement and stuff. And those two things is what's in common with basically everything about how we write code today is Turing completeness and so on. we know as a fact, you you learn this very early in your computer science training. we can't analyze any program that's Turing-complete. When you look at a compiler, what does it have to do? I have to look at this code and figure out what it's going to do to figure out how to map best to hardware. So the core insight from us was to say, what happens if we make code from the ground up that intentionally doesn't cross the threshold to be Turing-complete? It's intentionally designed to not trigger these impossibility results. And when you do that, you wind up saying, OK, now for the first time we can analyze code. But this now, because it's foundational, has an implication on how do we design compilers? How do we design hardware? How do we design the operating system? How do we design the programming language? So what it looks like is a programming language that you would use or an interface that looks very similar to how you write code today. And if you look at it, we have these really powerful languages. But companies come up with style guidelines saying, don't use these features of the programming language. There's no reason to start thinking about Mersenne primes or Goldbacks conjecture in your accounting software or something like that. But the compiler can't exclude you going to suddenly decide to do that. So what it looks like is we come up with this model of how software should look, and then we had to rebuild all the pieces around the software to understand what we can now do that code can be analyzed. And when you look at the implications, (41:09.026) we can analyze code, start to get rid of many security vulnerabilities, because I understand the permissions, I can compare this organizational policy. You can start to understand resources usage, and also more importantly, when we talked about heterogeneity, what processor would work best on this application, so you can make that decision before you run it, if that makes sense. So is that just a, you're looking at the code, you're profiling it and saying the hardware profile for this to be optimized the best is X. Are you also, is that correct? then can you provide guidance on the way through to prompt say, no, it should be, don't do that, do that over there. That's a crime against computing, stop that. Bit like the minority report where you can stop a hardware incident from occurring long before it ever gets to production type thing. Yeah, that's another analogy. was another one. full of it today. Sorry, AJ. Yeah, no, absolutely. Like that's exactly the path we're going down, except the difference here is it's not a manual process. Once we have the analysis, we can let ourselves do extremely aggressive optimizations and we can take the human out of the loop. And so the vision for what happens beyond Moore's law is that we allow a system to autonomously improve itself as it executes. And part of it, yes, is machine learning, which is we take that profiling data you've mentioned, and we can say, okay, we've tried this on this processor, but someone's running the microwave right now, so it doesn't quite behave as we want, and we can start to adjust and adapt, right? The trick with us is we've been looking at how do we do this in a way that the program will always be correct, deliver correct results, will never deviate from the true answer as it explores the space of how do I represent this program? How do I run efficiently and study this? So the solution absolutely is How do we run this profiling data and get the human in the loop so that can, Moore's law becomes software efficiency, right? And it's that thing about Moore's law where they, it was a very easy calculation about number of transistors per whatever it was. forget the actual calculation. And

slowly over time you cause the parallelism starts to create. It was excellent attention. Brilliant wasn't it? But they've started to change the way. (43:19.454) Yeah, it's got transistors in it and stuff like that. But then the way they've calculated it, they've not corrupted it, but changed it, changed it, changed it. What you could argue is now with the profiling you're doing, it's an extension to how we conclude how Moore's law can continue to be calculated. You're onto the next stage of, we've gone into the software layer now to say we've optimised that to make sure we can continue to get the doubling of performance over the period. Well, you know, you're talking about Moore's law. If you say Moore's law in a room for computer architects, you get very upset and they start correcting you. Like, no, you're talking about DÃ©nard scaling. And so they keep having these laws are pulling up that are still going. Right. But they're all failing. Right. So the trend lines are all still flattening. So, you know, regardless of what happens with Moore's law, we know, let's say it keeps going. We have more transistors, more processors that we still can't program. But now we're scaling up 1 % and we're still at 1%. Right. But I think the key to all of this is that for all of our lifetimes, computer programs have been unanalyzable. You actually don't know what happens until you run them. The change that we're proposing that we've done that we can make happen is we can now analyze it using software. So you know how long it's going to take to run. And you can build the dependency graph so you can do automatic parallelization. Suddenly that releases software to do what it does well, which is do things quickly, do it automatically. And you can do that optimization without relying on the programmer saying, you should optimize this bit or this might be able to run in parallel, or you should put this data over here because that's where the memory is. All of that goes away, which actually coming back to you, what does it mean for engineers? is it becomes a more focused job of solving a business problem rather than sort of twiddling around in the lower layers of compute and sort of bringing things together rather than, right, I actually want to solve this accounting problem rather than understand the depths of computing. (45:42.574) I could have desperately used this many years ago. I sent many crimes against performance to production and then discovered it was woefully non-performant and then had to spend months on picking the threading model or something like that. where was this when I needed it? I mean, it's absolutely a massive issue because sometimes you do... It's really good point. You don't know what it's going to do until you actually get there onto the hardware that's the target hardware for production. Now, cloud... You can reconfigure your hardware, but actually under the covers is that is a very complex job to try and understand what it actually needs and you're solving what is it if you've ever been in a room trying to eat more performance out of a production system. This is, this is exactly the answer. that's as I was saying, once you have that ability to analyze, then you can understand how to map that onto any backend, which means that processor upgrades are essentially free. You don't have to recompile the code. You don't have to adjust the code. Changes in how big the platform is you're running on or how small it is, essentially free. And you end up with the focus being on resolution of business problems rather than the intricacies of compute problem. AJ, in the sort of testing that you've done, what kind of performance uplifts are you actually seeing and a measurable. Yeah. So first I want to just kind of respond to the sharing application. I to add one point that I'll mention. or for engineers who are actually debugging code, the other advantage of analysis is that we can also help you understand bugs without just running them in the field where someone's reporting something that's a weird issue. So when you take the analysis and add it to the developer tool chain, we can better understand before you ship code. if there's a problem rather than having to debug afterwards. So we can start to use this as a safeguard to help developers improve their productivity, because most of our time spent debugging weird issues that nobody understands. In terms of performance, where we're at as an organization is that we had to prove that the model would work, and we had to show that this would work at a small scale. Now we're ready to scale up to build this into a platform. Now we're ready to start to work with strategic to understand (47:58.21) How do we integrate this with the hardware

and so on? So I can't share with you direct hardware numbers because our first test was to show that the analysis can work at all. So if you're make an analogy to say the Manhattan Project or something, first you have to come up with nuclear physics. That's what we had to do with figuring out the theory. Second was to show a nuclear pile. Can the atom be split? That's where we're at. Next is we scale up. And so what we have shown, which closes the loop, is we have now a static analysis running. that we can show and we can actually generate the execution kind of dependencies and graph that we would have targeted a titanium processor. have a simulated processor running and that shows us that the technology works, but it needs to be scaled up to get those benchmarks and the whole system needs to be put together. And that's what we're doing right now. Very good. And I think in the, in the analogy competition, got another one now, Dave, properly taking the crown with that. yeah. The Manhattan project. As Dave. where we're doing the heavy lifting. Where's your analogy? Come on piloting. We're still waiting. Come on. He's trying to think. Come on. can see it on his face. could see. I'm watching the agonizing look on Esme's face. I'm thinking about an onion, right? It's with the layers. my analogy that you can't have an onion. The more compute stack is very much an onion and at the center still remains cobalt. AJ, so another thing that occurs to me as we have this conversation about hardware platforms, maybe the deepest dive we've done on hardware platforms in the entire run of the reality shows. So really interesting to get into like such low level, but it seems to me that one of the things that's happening at the moment with quantum is that there's a whole other platform that thinks very differently right on the horizon. It's in, you you can use it in the Google cloud now on special request. And it seems to be fundamentally different in so many different ways. So how does that fit into the picture for you? Well, I think it's just a demonstration of just how exotic and strange hardware is about to become. We're very, very far away from, know, we're not in Kansas anymore, right? Like we now have this really strange processor that you have to sit through a lecture from a bunch of physicists talking about qubits and you can't really figure out how to print something to the screen or how to add to numbers. (50:19.416) without going through all of these details. Quantum is one of those things we've been lucky enough to have a number of people come on the show and tell us about how quantum functions. And for the fraction of a second it's being explained to you, you get it. then literally it's gone. And I just couldn't explain it again after that, save my life. Well, it's like Heisenberg's concept, right? Like, is it working or is it not? You know, we don't really know. But I think the issue with quantum is going to be one of programming models. So let's say I give you a quantum computer right here that runs infinitely fast. Well, how are you going to program it? What are you going to run on it? How are things going to change? So they actually have the same problem that we're trying to tackle, which is how does software change with really exotic hardware coming down the pipe? And so for us, quantum computing has been this kind of test. Is our approach, is our... Is our work sufficiently generalized that we could target a quantum computer if it suddenly was brought to us and put on our desk? And that's what we've been looking at. It's not, you know, we expect it to be shipped next quarter. But if it does come out in the next hundred years, could programs written in our model conceptually target it? And do you think when we train computer scientists, we teach them about the sort of art form, a sequence, choice, selection, iteration, et cetera? Feels like there's maybe an educational shift that we need to bring in. You need to think differently to get all this to work. mean, you see sort of like the system changing that that material is available and people are moving to maybe try and train people to have a different mindset around it? Or do think we're still sticking dogmatically to the old style? I think it's actually a problem that you're in computer science trying to learn software and all of sudden you have to degree in quantum physics to understand how to write a program and now you have to figure out what computer architecture does. I think it's more a symptom of the disease, right? Which is software hasn't really changed. What we need to do is look at the programming languages we know and love and already use. You're taught not to use the exotic

features, right? But on the other hand, the compiler, the systems has to assume that maybe you did use all the exotic features. So the solution that as I see it is not to increase the training because this stuff is complex and hard and worse the time to market is very long. (52:40.984) What a Python developer would do in a day would probably take me two years to do in parallel, right? And that's not going to scale up no matter how you train people and the customer is going to change their mind and you just throw it all out, right? So I think the answer is to figure out how, you that's what we did was how do we take these style guidelines, how we actually write code and be compatible with industry best practices? How do we make it work with what people already do? And if you look at it, when's the last time you intentionally wrote an infinite loop in your accounting package? you're really mad you might do it, but you're not going to do it on purpose. So we have all of these results from the theory, but they all require really weird programs nobody actually writes. So when we actually start, we actually fix those things. We wind up being able to give you something that's more compatible with what software developers do. So the training, if anything, would basically be as it is, but not having to worry about all of this extra detail of what's coming down the pipe with hardware and memory placement and stuff like that. So to bring us to a bit of a close today, both Peter and AJ, help us understand what the time to market is here. So obviously you're developing something that's very new, a revolution as you say Peter. When are people going to be able to get their hands on it and what steps are going to be required to be able to consume something like this? Well, I think, you know, as we've been describing, this is a complete reimagining of the compute stack. And as such, you know, spent many years going through the theory, making sure that it will work. We've got a demo that shows it will work. We've proven that the more contentious parts of it. So when you line up all the pieces that you need to put together, we think that there's a 36 months evolutionary path to a full platform. We think in 24 months, there'll be available libraries that can be linked into current applications and those will perform or provide performance boosts. But it really is, you know, we think 36 months away because, you know, you've got a lot to do. It's unfortunate but true. What are the things people could do to get (55:04.224) involved a bit earlier is we do have an early access program we're spinning up that if you go to our website yetiware.com there's a link that you can sign up and be in the loop to be one the early testers or users of this thing if you want to try it out and as we make progress you can reach out to see if you're interested in trying it out. you (55:32.908) Now we end every episode of this podcast by asking our guests what they're excited about doing next. And that could be they've got a great restaurant booked at the weekend, or it could be something in their professional lives, or it could be a bit of both. So Peter, what are you excited about doing next? I'm not sure I'm excited, but I know what I have to do next is a lot of spring cleanup. So there's all sorts of branches fallen off and I'll be out there with a rake. I'm sure. But, it's kind of Spring cleaning. Like it's a thing, isn't it? My, my head immediately went to cupboards and things like that. is all like outdoor stuff. know, if you recall, I do own five chainsaws. Impressive. What? I'm sorry, but why do you need five? Or is it you collect them like motorbikes or is there a utility for each of the five? You just like chainsaws. A young chainsaw. Yeah, no, they're all different. I horrifically dangerous as well. I understand. Have you injured yourself with one, No. See, safe. Safe day, if you see. Responsible adult in charge of equipment, unlike ABU. I don't trust myself with a chainsaw. I'm not going to lie. don't think anybody should trust you with a chainsaw, Dave. I think I'm with you on that. Do you know if you're a woman and you work with a chainsaw outside, how many men come to you and actually give you advice? Mansplain? Mansplaining. Please be careful because now I have wireless, but the first one did have a wire to say, please be careful because before you know it, you cross the wire. So I must be more accurate. I only have four chainsaws. My wife has one. A wireless one, right? No, no, no, they're all, they're all petrol powered. And she and her friend used to wander around one of the local parks with their chainsaws clearing paths. (57:42.196) Wow. Like sort of chainsaw vigilantes. Any trouble at all. Just sort of going out into the world and doing good

with chainsaws. exactly. Were the police ever involved for an interaction to say there's a person in the park with a chainsaw? They have an occasion. Why? By questioning what are you doing? Why are you here with a chainsaw? No, no. We have a very good relationship with the local constabulary. Well, didn't think we'd end up there. AJ, help us out. What are you excited about doing next? I'm not really the chainsaw type, I will be, you know, I'm looking forward to as things start to go cycling again and go for some long rides and maybe throw my kid in the back of a trailer and haul them across the trails and stuff back here. Are you a mountain biker or a road biker? Mountain biking is too scary. I have a hybrid that I go on rail trails so I don't have to deal with cars. So there's old rail trails that you can go on that are fairly long. And what's your favourite brand of bike? Well, I only have the one, it's a Norco. So I'm not like some big fancy bike person. It's a generic yellow bike number three hybrid. Was that how you bought it in the catalogue? The guy the bike shop said it was great. So I got that one. Excellent, excellent consumer advice there by that guy. Brilliant. Guys, well look, thank you for spending a bit of time with us this morning to tell us about what is really a fascinating inquiry at the real heart of what we all do. So like really best of luck as you come to the end of the development process and the launch. It's been great talking to you. Thank you. Thanks.

If you would like to discuss any of the issues on this week's show and how they might impact you and your business, please get in touch with us at realitiesremix@capgemini.com. We're on LinkedIn. We'd love to hear from you. So feel free to connect and DM if you have any questions for the show to tackle. And of course, please rate and subscribe to our podcast. It really helps us improve the show. A huge thanks to AJ, Peter, our sound and editing wizards Ben and Louis, our producer Marcel for not being here. No kidding. He's on a boat. He's on our way to come come back to us. (59:52.61) We missed you. And of course to all our listeners. See you in another reality next week.

About Capgemini

Capgemini is a global business and technology transformation partner, helping organizations to accelerate their dual transition to a digital and sustainable world, while creating tangible impact for enterprises and society. It is a responsible and diverse group of 340,000 team members in more than 50 countries. With its strong over 55-year heritage, Capgemini is trusted by its clients to unlock the value of technology to address the entire breadth of their business needs. It delivers end-to-end services and solutions leveraging strengths from strategy and design to engineering, all fueled by its market leading capabilities in AI, generative AI, cloud and data, combined with its deep industry expertise and partner ecosystem. The Group reported 2024 global revenues of €22.1 billion.

Make it real.

www.capgemini.com

Capgemini