Rashmi Mohan:
This is *ACM Bytecast*, a podcast series from the Association for Computing Machinery, the world's largest educational and scientific computing society. We talk to researchers, practitioners, and innovators who are at the intersection of computing research and practice. They share their experiences, the lessons they've learned, and their own visions for the future of computing.

Rashmi Mohan:
If you've ever wracked your brain, trying to gauge the performance of your software program in a production system, and used DTrace to the rescue, you have our next guest to thank. Bryan Cantrill is a computer scientist and a software engineer with a rich career in system software and building solutions to key problems in distributed computing. His path breaking work on DTrace won him multiple awards and a spot in the MIT TR35 list. Today, he is the co-founder of the Oxide Computer Company. Bryan, welcome to *ACM Bytecast*.

Bryan Cantrill:
Thank you for having me. It's great to be here.

Rashmi Mohan:
Great. I'd love to ask you the question that I asked all my guests because I always get fascinating answers is, if you could please introduce yourself and talk about what you currently do, and also give us some insight into what drew you into the field of computing?

Bryan Cantrill:
Yeah, sure, we'll get to what I'm doing in a bit. As you said, I'm the co-founder of Oxide Computer Company. We're taking a very big swing at an old problem. In terms of how I got into this, I'm definitely a child of the 80s and born in the 70s, but grew up in the 80s. And so, I think personal computing was such a revolution in the 80s. And I think, like many in my generation, my first computer was a Commodore PET that I used in third grade and playing Oregon Trail on the Apple IIe. Computing was so revolutionary. You could kind of feel this revolution happening.

Bryan Cantrill:
I always enjoyed computing, as a, I would say it was more of a hobby, honestly, I enjoyed writing basic programs and so on. And I took programming classes in high school. I didn't really view it as an academic domain, honestly. This is a little bit embarrassing to say, I just didn't really know what computer science really was. And on the one hand, that sounds ridiculous.

Bryan Cantrill:
On the other hand, if you think that most computer science programs were really only born in the 70s and 80s, it's a new discipline and I just didn't know that there was something really academic there until I arrived at university. So, when I went to school, I was actually very convinced that I was going to become an economist, believe it or not. I was going to get a PhD in economics. I wanted to be a mineral economist, a very strangely directed idea what I wanted to go do. And I always enjoyed programming.

Bryan Cantrill:
And I really credit when I was looking at schools, I was looking at ultimately, two different schools. One had an extremely strong economics program and a more or less non-existing computer science program. And one had an extremely strong computer science program, and a pretty good economics program, but not necessarily especially notable. And parents think that their teenagers never listened to them and that is only mostly true.

Bryan Cantrill:
I definitely listen to my mother in particular at very key junctures. I definitely remember my mom saying, "You really owe it to yourself to really understand what computer science is about." And my mom is not a technologist at all. So, I'm not sure where, I guess it was just the wisdom of years that gave her this perspective, but that turned out to be very prescient. And when I really arrived and took my first real computer science course, as an 18-year-old, it just felt like a very big light was going on.

Bryan Cantrill:
For me, the study of algorithms, the fact that you could reason about algorithms, the fact that this was a mathematical machine effectively, and the fact that something that I had always enjoyed and had fun doing, namely programming, now suddenly, I could program for credit. I could actually do this fun thing that I enjoyed and get course credit doing it was just a tremendous rush.

Bryan Cantrill:
And I really fell in love with computer science ever since. I went through kind of a period where I guess the economics washed out to me faster than the academia because I definitely believed that I would go to grad school and get a PhD in computer science, and was very invested in that path. But then, fell in love, again, with operating systems. I took the operating systems course as a sophomore on school.

Bryan Cantrill:
And what I loved about the operating systems course, is the program of an operating system kernel, it's a really hard program. It's a really challenging program to get right, and I love that. I loved, it's very easy to get this kind of arrogance as a programmer that you can kind of like program anything and that arrogance leads to a kind of boredom, which is ridiculous because there exists these programs, many programs that are really hard programs to write. And I loved that in the operating systems course. I loved how hard the program was, because it's so frustrating when it's not working. And it's so gratifying to finally get it working and to get it right felt very gratifying.

Bryan Cantrill:
So, after taking this OS course, I really wanted to go do systems work. When I was looking for a job after my sophomore year, I really decided that I did not want to go spend that summer back at home. I wanted to go work somewhere.

Bryan Cantrill:
So, I'd read in, if you remember, Byte magazine back in the day, but Byte magazine had an operating system round up issue. And this would have been in December of 1993. And I remember, on the way

back home to Denver, I grew up in Denver, in O'Hare, reading this article. And in particular, they were talking about all these interesting operating systems that were out there. And one of the operating systems they talked about was an operation called QNX, which is a microkernel-based system. And I ended up just getting an email address. I must have found an email address for someone at QNX, and basically cold emailed them, saying that I would love to come work for them for the summer.

Bryan Cantrill:
So, much to my delight, they were willing to have me and I worked in, this is in Ottawa, in Canada, and worked there for two summers. But especially coming after that second summer, where I again, had done really hard programming and had done the bring up of the operating system on a symmetric multi-processor, which was so much fun and so hard.

Bryan Cantrill:
By the way, define what do you mean by hard. Hard is a program that you think you might never get working. That's an example of a hard program. If you have that voice in your head, worried that you're never going to get it working, that's hard. Wherever it is in the stack, whatever it is, that's a hard program. And I definitely had a hard program there at QNX. I really enjoyed getting it working. I was very satisfied. And I just found it satisfying, and really had decided at that point that I wanted to go do operating systems development. That's where I wanted to go, spend my career doing.

Bryan Cantrill:
And at that time, and now we're up to like 1995, getting into 1996. If you wanted to go do OS development, really there was one company that was doing OS development and that was Microsoft with Windows. And then a bunch of other companies that were rapidly trying to figure out how to run Windows on their computers. And I did not want to do that.

Bryan Cantrill:
There were a bunch of reasons why I did not like the OS development coming out of Microsoft. And I really loved Unix. And there was really only one company that wanted to bet it all on Unix and that was Sun Microsystems. So, I went out to Sun in 1996, and that kind of launched my career. So that's how I got into it anyway.

Rashmi Mohan:
That's incredible. There're so many nuggets in there that I want to tap into, right? I mean, one finding, I really love what you said about the fact that a hard problem or a hard program is one that you may never get right, to actually use that as something to fuel your interest rather than lead to frustration in itself is a very unique trait. I don't know if many of us have that.

Rashmi Mohan:
And secondly, I think the piece that you said, I mean, I'm wondering if you were in an area of software development that was not nearly hotly pursued by so many others, maybe because it was hard, or maybe because it was not as sort of flashy. And that in itself may have led to so many great opportunities that

you could have pursued. But the third and most important thing that you said was that teenagers listen. I have two teenagers, I wonder if they ever listen sometimes. At least, I know I won't stop talking now.

Bryan Cantrill:
I've got teenagers as well. And yeah, I think the challenge is you don't know what they're listening to. And they're kind of never listening on your schedule. They're listening on their schedule. But I think they definitely do listen, and it's true. And I think that, I mean, you raise a bunch of good observations there.

Bryan Cantrill:
I do think that it's a real danger, not that I ever did. And not that I don't necessarily see this as too endemic, but you definitely want to follow what's trendy or what's kind of in demand. I think it's much more important to find the things that are satisfying to you. And I know that this becomes, people are kind of derived the idea of following your passion. And I actually don't like the word passion for a bunch of reasons. But I do think it's really important that we are intrinsically motivated by the problems we're solving. We should find solving those problems intrinsically satisfying.

Bryan Cantrill:
And if there's a problem domain, in which one finds the problems intrinsically satisfying, that's a problem domain that you should pursue. And I had many people tell me that OS development was dead. People who were domain experts saying that, and this is in 1996, that there's really nothing new to be done in operating systems.

Bryan Cantrill:
And I'd love to believe that my technology was one that helped prove that wrong. DTracing and, but not just DTracing, but ZFS. And we did a bunch of things at Sun at that time. But there's been a bunch of things since then. I mean, you go back to, no one wants to run an operating system from 1996. Because operating systems have advanced, a lot, many different operating systems. It's not just one technology.

Bryan Cantrill:
I think that history is littered with folks who thought that a domain was not worthy of a young person's time and energy, because they themselves didn't see new opportunity in it. I think it's dangerous to talk young people out of their enthusiasm for a domain, because that enthusiasm is often required to get to real breakthroughs. And yes, the domain may be different than it was two decades prior or what have you. But that doesn't mean that innovation is dead in it. Quite the contrary, that may mean that the domain is really ripe for a new leap forward.

Bryan Cantrill:
I do think it's, it's important in terms of how we counsel one another. And I definitely tell people don't make decisions based on fear of what you think is going to happen. You should make decisions based on the things that you find motivating, and exciting, and worthwhile. And honestly, don't worry about the rest, yet. Focus on that intrinsic motivation. And that, to me, is a path to a much more satisfying career.

Rashmi Mohan:

Very, very true words. And I think it's relevant even for folks who are later on in their career. Many times you come across opportunities, where you think that the new shiny project in your organization is where all the excitement is. And there are so many opportunities for innovation, even in what you do.

Rashmi Mohan:
I remember one leader in one of my previous organizations would say, "You can potentially rewrite your existing well working software every couple of years and radically change it and learn so much from it." But one thing I did want to touch upon Bryan is, I mean, of course, it's very hard to read about your varied accomplishments without mentioning DTrace. How did the idea for that come about? And somebody who's so early in your career, when you joined Sun, how did you get the opportunity to work on something that was so fast breaking or was that something that you identified when you got into it?

Bryan Cantrill:
Yeah, it's funny that you should mention that. So, I had kind of the original foundational idea, namely, the dynamic modification of running kernel text in a way that was not actually stopping a program as an undergraduate. And I remember asking folks about why it seems like this is the way you would do it. Like, "Why wouldn't you do it this way?" The word I got back was like, "Well, if it were possible to do this safely, it would have already been done," which I actually accepted. I'm like, "Oh, okay."

Bryan Cantrill:
So, when I actually came out to Sun, the day I was interviewed at Sun, and this just speaks volumes in many different dimensions. And this is one of those moments where I remember exactly where I was in my life. I was in [Bart Smolder's 00:13:22] van on the 101, talking to Jeff Bonwick, coming back from lunch. And at that point, I was really excited about the opportunity. And I was describing this kind of nugget of an idea to Jeff. And the way I was describing the idea, honestly, was not in the sense of like, "I want to go do this" or "This should be done." It was more, "I don't understand why this isn't done and there must be a good reason for it. I basically have been told that this doesn't exist for a good reason."

Bryan Cantrill:
So, my question was, "What is the good reason because I just don't know what it is? I again, I don't doubt that it exists. And Jeff just said, "Yeah, I can't think of a reason why we can do that. You should do that. Actually, you should come here, and you should do that." And boy, was that empowering to hear as a 22-year-old, that not only was there not a good reason why it can't be done, but that, that I should go do it. I mean, it was very empowering.

Bryan Cantrill:
And so, when I came to Sun, it was always with that idea at the back of the brain. Now that, I came to Sun in 1986, we didn't start until 2001. I spent the next five years more or less, really doing a bunch of work that I felt needed. There was a higher priority work in the operating system and then learning a lot of what we needed to go do to actually tackle this problem. But it got to the point where I got such a clear idea of what we wanted to go do. Someone would have a problem. And I would say, "Actually, DTrace would solve that problem." And of course, DTrace didn't exist at all at this point. And this is just a ridiculous thing to say, especially to somebody who's just suffered their day through a painful debugging exercise.

Bryan Cantrill:
And again, one of these moments I remember exactly where I was, who I was speaking with, senior engineer, Tim Marsland just debugged a challenging bug. He'd been struggling with a challenging bug. And I remember saying to Tim, outside of the restaurant that we used to frequent, "DTrace would solve that problem for you." And Tim just gave me this kind of exasperated look and said, "Well, if he's English, I'm not going to imitate his accent, but it's a very distinguished accent." But he was very exasperated.

Bryan Cantrill:
He said, "If Dtrace would solve this problem for me, then why don't you just go write DTrace?" And I was like, "Okay. Yes. We need to stop talking about what the DTrace is going to go do and actually go do it." We started shortly thereafter. November of 2001, is when we started working on it.

Bryan Cantrill:
And it still took, I would say, on the order of six months before we got to... Whenever you start something bold, there's always the fear of failing. I don't care who you are, what you're doing. That fear that this is going to consume you, but there's always a fear of I'm going to invest a bunch of time and energy into a dead end. And with DTrace too, there was a little bit of that. And we really needed to negotiate for extended time to do something that might not work, that might actually not be viable for some reason.

Bryan Cantrill:
And it took on the order of six, a little bit less, probably on the order of three months really. Three to four months, before we had these proof points that indicated, "Okay, this is definitely the right direction." And I think as an engineer, especially when you're doing something bold, you really look for those proof points, almost omens, right? You look for these good omens early, indicating like this is actually the right path, that this path may be hard, arduous, long, rocky, but the payoff will be substantial. We did get those, but it took a couple months to get there before we got those proof points. And then spend another three, four years really fleshing out the technology completely.

Rashmi Mohan:
I think what you said earlier about being empowered, especially as somebody early in their career is so true. That's what I was sort of tapping into earlier saying, "How do you get those opportunities?" A lot of times you feel when I'm someone new, I'm going to get to work on a small piece of a small puzzle. And I may not even know how it fits into the larger sort of initiative or the goals of the company. As you were working on this problem, I mean, one thing was obvious with your conversations is that you already had your "first customer" who is saying, "Go out and build this." I'd at least experiment with it or try it out.

Rashmi Mohan:
What were some of the other considerations or even constraints as you, and obviously, Sun was a massive company at that time and you were almost working on this research project? What were the other things that were on your mind as you were wrangling this problem?

Bryan Cantrill:
Well, honestly, I was, and I think this is where we always do our best work, I had one customer in mind, and that was me, I was really developing the thing that I wanted to be able to go solve the problems that I needed to solve as part of my job as a kernel developer. And I'm originally joined in the Solaris performance group.

Bryan Cantrill:
So as someone who is focused on the performance of a system, I was developing the tooling that I needed to do that job. And when you're developing for yourself, when you are the customer, you've got such a crisp idea of what to do. And I always believe that we do our best work when we are our own customers. And you're ultimately, I think that even though some of the legends in the field, I think someone like Steve Jobs. Ultimately, Jobs was designing for himself. He happens to have good taste that was applicable to a broad cross section of folks. But I think we do our best work when we design for ourselves.

Bryan Cantrill:
And so, you want to kind of feel and you hope that you represent a larger demographic than just yourself. And fortunately, in this regard, I think I do represent a larger demographic than just myself. And in particular, another light that went on a couple of years later was realizing that actually, while I was originally developing it for myself and doing kernel performance work, the much larger value of it was actually finding system level problems, applications that were misbehaving. The symptoms of those misbehaving applications, were often so deep in the stack that you actually needed to be able to instrument the operating system kernel in order to actually reason about this higher-level problem.

Bryan Cantrill:
I think that the other thing I would say is that all along, and I am a strong believer in this as well, we were using our own technology. And I think we as a discipline have gotten actually quite a bit better about this over the years. We have kind of grown out of this, throwing it over the wall, big software release kind of a thing. People kind of frequently deploying into production and so on, which is good, but it definitely yields a higher quality artifact when we ourselves use the software that we are creating or something we definitely did as part of Dtrace, for sure.

Rashmi Mohan:
Yeah, I think it almost should be on a plaque, you are your customer. I absolutely agree with you where if you feel the pain of a certain problem enough that it disrupts your day to day sort of peace, you will go out to find a solution for it and possibly the best solution for that problem.

Bryan Cantrill:
Right. And I think the balance to strike there is you want to be using your technology, but you also need to still have the freedom to improve it. So, there's this kind of idea that you want to make a software engineer carry a pager, so they kind of feel the pain of anything that's broken that they deploy, which only works if they are similarly empowered to fix that brokenness.

Bryan Cantrill:

So, what you want to avoid is having punishing engineers for technical debt that they can't fix, which is the kind of extreme of this isn't actually productive, that just makes miserable people. And we had this kind of luxury of being able to use the technology, but then also being able to fix the things that needed to be fixed.

Bryan Cantrill:
And there were years, I have to remind myself of this, because this was so many years ago, but there was a really protracted period of time, where every time I used DTrace to find a problem, I would find the problem, but I would also find a bug in DTrace. And I would think to myself, "Jesus, am I ever going to be able to use this tool?" And oftenly, these are small bugs, not huge issues. But you think to yourself, "Am I ever going to get to the point where this thing is just robust that works?" And of course, it did get there. But it took a little while to get there. And it got there because of that feedback loop and coupled with the freedom to actually fix the problems that we found.

Rashmi Mohan:
That is really, really good point. I think that's the ability to identify the problem, but also have that power to be able to go and fix it and not be constrained by other restrictions by an organization or a technology choice is very important.

Rashmi Mohan:
One of the things that I was reading about, as I was reading about DTrace and the history, of course, now the word observability is bandied about. We use it a lot. It's supposed to be super critical for every business to be considering. What do you make of it? And how do you feel, has it grown or has it changed and morphed in our understanding of it?

Bryan Cantrill:
Yeah. More so. It's funny you mentioned that because I'm not claiming that I necessarily invented this term with respect to software, but I definitely don't remember anyone using it before me.

Rashmi Mohan:
Wow. That's amazing. I did not know that.

Bryan Cantrill:
So, again, it's hard for me to know. I mean, I talk about it a lot in this 2006 paper I had. I definitely was using it long before then. And in part because DTrace was a new kind of a thing that we did need to have a new way of talking about. And we said the DTrace was concise answers to arbitrary questions. That property of being able to answer questions about the system, I called observability.

Bryan Cantrill:
It's the kind of thing that, I don't know, maybe I did hear from someone else, I don't think so. I certainly, but I don't actually know. One of the things that drives me a little bit bananas is that kind of people go to the Wikipedia article for observability, which has really got nothing to do with software observability. It's

a control theory construct that has got zero to do. And they kind of provide this Wikipedia definition for observability. Considering that's actually not what observability is at all.

Bryan Cantrill:
Observability to me... Software observability, system observability is the property to actually observe what the software is doing. Because without dynamically modifying software, we don't actually know what it's doing. If I show you a CPU executing software without actually modifying it, you don't know what instructions are being executed. And to me observability is the property by which we can understand what the system is doing.

Bryan Cantrill:
And again, not that I was ... I couldn't do this. So, I meant the, certainly not the concept but the nomenclature, I don't know if anyone has used it before me. Again, I would be, I'm sure someone did or I don't know. I don't think they care, actually. But I do think it's kind of funny that now the term has become much more broadly used, and how come people are fighting over, what is the true definition of observability? Which I think is honestly, I think it's great, actually.

Bryan Cantrill:
I mean, I think it's great that there are so many people that are concerned with how we understand these systems, that there is disagreement over the terminology, it's probably a good sign. So, I think it's probably a good thing. But it definitely has taken on a bunch of different shades and tones and so on. Different people have got different definitions. But to me, it boils down to the ability to answer questions about the system.

Bryan Cantrill:
And then there's a very kind of important implication there, which is that you need to have somebody who's asking the questions of the system. And one of the challenges that we have is asking questions of the system is hard. It requires an understanding of the system. So, how do we allow people to easily ask questions of the system and explore and have those answers? Then prompt new questions that allow them to get to understand the behavior of the system, which is what we're really trying to understand. We're trying to understand, putatively, presumably, pathological behavior in the system, but we're trying to understand dynamic system behavior. That's what observability is about to me.

Rashmi Mohan:
Yeah, I completely agree with you. One question I do have, though is, has it become more sort of crucial, more of on the minds of business owners now, as the systems have gotten to be more complex and more distributed, cloud computing? Why is it talked about more today as compared to maybe 20 years ago?

Bryan Cantrill:
Well, yes, I mean, it is much more important. And these distributed systems make it really important, because when the systems misbehave, especially we are now have got companies that and we saw certainly early shades of this in the late 90s. But we have many companies that are entirely software as service companies. And when their software is unavailable, their business is dead. So, it serves a real

focus folks. And the idea of, you have outages, but then these kind of transient failures... I think, as Marc Andreessen 2011 prophecy, its software eating the world. As software is eating the world, our observability of that software has become ever more important.

Bryan Cantrill:
So, it is more important, which is great. There are a bunch of companies and projects, and so on. Technologies that see that opportunity, and that's great. I do think one of the challenges that we have, and this is just an evergreen challenge is, people generally, they don't want to have outages, certainly. They don't want to have poorly performing systems. But it can be more of a challenge to get people to invest upfront in what's required to not have that.

Bryan Cantrill:
People are more reluctant to make longer term investments. And that may be investments in terms of money, and maybe investments in terms of time, in terms of energy, in terms of thought. And getting people to think about the debug ability of a system, as a system is being developed is really challenging. Because everyone is focused on getting the system deployed, and not necessarily as much on what is going to happen when the system fails after deployment.

Bryan Cantrill:
So, I think there are a bunch of folks that are focused on the problem. And I hope that we can get a kind of a longer term thinking that allows an increasing number of people to be willing to make the necessary investment for the systems to be robust.

Rashmi Mohan:
Right. I think that preventative, sort of hack that you need to wear when you're sort of building software systems, that's yeah, its always hindsight seems to be 2020, in that case.

Rashmi Mohan:
Going back to the point about asking the right questions of your system. Do you have any thoughts at all in terms of, I mean, I've heard of metrics, events, logs, and traces to be the gold standard of measurement in this field of observability, would you help us understand, what do you think are the crucial maybe questions, we need to ask all these systems? And in turn, what should we be measuring in order to be able to answer those questions?

Bryan Cantrill:
Yeah, well, I think we don't want to kind of falsely dichotomize what we need. And I don't think it's kind of a, is it metrics versus logs or traces versus events, what have you? I think that is, that's us operating across purposes. I think that when a system misbehaves, we need to understand what it's doing. And all of those things can serve as valuable steps or valuable sources of data or information that can allow us to ask questions.

Bryan Cantrill:

So, metrics can allow us to ask interesting questions. But if I can't actually ask the followup question, I see this several 100-millisecond outliers that happened. Why did that happen? It's like, "Well, we don't know why that happened." Or not being able to kind of ask that next question, not being able to dig deeper to get that question answered, we run the risk of having performative metrics, for lack of a better term, where we are gathering these things to give ourselves dashboards, but it's actually not giving us real insight into our systems.

Bryan Cantrill:
And in fact, to the contrary, we are overloading or being overloaded with stimulus. And sometimes, we have so many metrics in the system, that it can be hard to see what is actually going on. And I always think that history can teach us a lot. And in the Three Mile Island disaster, there was a relatively simple problem that was very hard to see, because the operators were so overloaded and overwhelmed with all of the cascading failures of that, that this relief valve was in the wrong position and the solenoid was effectively telling them something that was the opposite of what they thought... The light rather, was telling them something that was the opposite. The indicator was the opposite of what they thought it was and they became effectively overwhelmed, and it took a shift change.

Bryan Cantrill:
And someone taking kind of a clean sheet of mental paper, if you will, and thinking about what they knew about the system to be able to ask this question. The metrics were not guiding them to the right question. In fact, they were overloading them.

Bryan Cantrill:
So, I think we have to be careful about looking for a panacea in there. There's not a panacea. The systems are complicated, and it is about how do we answer questions about the system. And I think also, we should serve, and I'm not sure to what degree this happens in all circles, it certainly should. It should also serve as a check against complexity.

Bryan Cantrill:
For those software engineers who haven't already figured it out, they will at some point in their career. And that is, complexity is the enemy. We need to make things as simple as we can for them to function. And there have been many revolutions in computer architecture, computer science, computer systems, that are effectively revolutions in simplicity and making the system much simpler, and having less, not more. And when you have less, it is easier to reason about easier to fit your head and easier to ask the right questions.

Rashmi Mohan:
That's an excellent mantra to sort of live by, especially in terms of in any sphere of life, I would think, but yes, simplicity will solve a lot more problems before they actually become problems. It seems like this would be a question that comes up in any software discussion nowadays is the use of ML, right? It seems to be the charter of every business out there to use that intelligence and to make their systems better or to be able to use machine learning to really enhance what they can offer. Where do you see the applications of ML in this space?

Bryan Cantrill:
I don't know. I mean, I think that part of the challenge, and this is not my domain and expertise, just to be clear, but I do think that part of the challenge is, I don't know how you would train that. Training data set is so important. And to me, these systems are so complicated and their emergent behavior is so difficult to reason about, that I don't know how we would train for assistance in pattern identification.

Bryan Cantrill:
I mean that's what we're talking about. We're talking about ML. We talk about learning and artificial intelligence, I think we do ourselves a great disservice by thinking of it as intelligence and not thinking of it as pattern recognition, which is what it is. And it is sophisticated pattern recognition. So how can sophisticated pattern recognition help us in asking questions of systems? And I don't know the answer to that, which is not to say that it can't. I'm sure it probably can but it is not obvious to me what that answer would be.

Rashmi Mohan:
Got it. I mean, it seems like you said, it's not that it can't be done. Maybe it hasn't been done yet or hasn't been thought about enough. And so, maybe the next young student that comes in from university, trying to solve this problem will have a shot at it.

Rashmi Mohan:
But I did want to talk about one other part of your career that I observe, Bryan, which is, you have a particular interest in open source. I mean, it seems like a lot of the solutions that you build are for the open source community. How did that come about? And I'd love to hear more about your philosophy on that.

Bryan Cantrill:
Well, I mean, as a software engineer. I mean, I remember being as a student at school, just eagerly awaiting the source license to the operating system. So, this is in the era of strictly proprietary operating systems. And we had a research agreement with Sun that allowed us to peek inside the source code, and boy, was it valuable. It was so essential to understanding some of the problems that we had to see inside the source code. So, I certainly never forgot that feeling, and as an engineer, just found that source code availability is essential.

Bryan Cantrill:
But when I arrived at Sun, it was still a proprietary system, and I got to participate in something that was pretty fun from 1996 to 2004. And that is, we open source the system. The system became open source in, I guess, early 2005. I guess it would be January 2005. And DTrace was actually the very first thing that we open sourced in the operating system. And it was really fun to get the source code out there and to make a contribution, make something that was meaningful. It was really hard to open source proprietary software.

Bryan Cantrill:

I think that sometimes outsiders underestimate the complexity and the organizational willpower it takes to open source the operating system, open source anything, open source a body of software. In part because, when in all proprietary company that is not paying attention to making their source code available, we'll sign all sorts of agreements and incorporate all sorts of other proprietary code that prevents them from open sourcing.

Bryan Cantrill:
And indeed, that was the case at Sun. I can't remember, it was on the order of 300 different agreements that had to be chased down. And there were parties that couldn't be located, companies that had gone out of business that we couldn't negotiate with to open source their elements that we had long since licensed in perpetuity inside the operating system. So, there were some things that when we open source the operating system, there were actually some things that we could not open source. Little things, stupid things that we can open source, that as an open source community, we later developed afresh.

Bryan Cantrill:
But that whole experience, which was an exhausting experience, really seared in my mind, the importance of open source. And we do better work when it is open source. We do proprietary software, allows for this kind of natural monopoly that the software vendor has. And they just begin to behave as a natural monopolist behaves. And there is good proprietary software out there certainly. It is the exception rather than the rule.

Bryan Cantrill:
So, I've always believed that software, being open source, allows for a really important kind of check. And as a technologist, it allows me to integrate someone's technology without fear, because I know that I've got the source. So, it kind of came up in this era of proprietary software, that then flipped to become open, and it felt like the open source revolution was done, was over. But then with the rise of cloud computing, we saw a different kind of angle where open source projects, were now being run as proprietary services, because the licenses didn't preclude that from happening. And that I felt was dangerous and corrosive.

Bryan Cantrill:
And as I feel some of these, eCos for years, Google and Facebook and Amazon didn't really open source very much. It's hard to believe now because Google has contributed so much. But there was a protracted era where Google contributed really very little. And they consumed a lot of open source, but they didn't actually produce anything. That changed over time. And I feel it has been changing over time, certainly at Facebook and even at Amazon. Amazon was not open sourcing anything, but has begun to change in that regard.

Bryan Cantrill:
And as those changes were happening, meanwhile, these other software companies that had grown up as open source software companies now are viewing the license as a way to effectively monetize their software, namely, by restricting the way it's used. And that's a really unfortunate development that is antithetical to the spirit of open source and federal letter of open source, actually. It's just not open source.

Bryan Cantrill:
I am optimistic that this will be a blip in history. Open source software, part of the beauty of open source software is that it's information, it survives in perpetuity, it can't be deleted, it can't be removed. And as a result, these open source in the limit, open source will effectively conquer every domain. That is say, every software domain will have open source components at its core. I very fundamentally believe that. And I think this regrettable recent trend in the last two years or so, I think will be a blip. But I certainly believe in the power of open source more than ever.

Rashmi Mohan:
And that's incredibly heartening and encouraging. But as an entrepreneur, Bryan, and now as a co-founder of a company, how do you see open sourcing the work that you are creating, working alongside the fact that you do have to run a for profit company?

Bryan Cantrill:
Well, so, we're a computer company. So, we've got God's own open source revenue model. Namely, we make computers, that's where you're going to make and sell. And we are incentivized to open source our entire stack. Because our entire stack is designed to run on, honestly, our computer. That's how we make money. We make money with the metal.

Bryan Cantrill:
We hope to serve as a model in that regard, because I'll tell you the true last domain, the last holdout for proprietary software is in this very critical, but not well-known layer of firmware. And firmware right now is broadly very, very proprietary, and it is causing mayhem, all sorts of mayhem. It's a security problem. It's a reliability problem. It's an economic problem.

Bryan Cantrill:
And we are really hoping to serve as a model with an open source stack, that includes not just all the system software, we're building rack scale machine. So, we are building, the kind of the minimum buy from Oxide will be a full rack of compute and storage and networking of our own design using off the shelf silicon. So, using x86 based silicon, but we are developing all the software, our own hardware root of trust, our own service processor, and so on. All of that software will be open source.

Bryan Cantrill:
And from our perspective, it's easy. It's not an economic argument at all, because I am just not at all concerned. In fact, if someone wanted to take our software and get it running on some other hardware platform, sure. I mean, good luck. But great, honestly, great. Because the key is that a customer that would do that, take our software, and go through all of the technical challenge of getting it to run on other hardware, that was never going to be a customer of ours anyway. Their contributions to our community is great. That's a contribution that we wouldn't otherwise have.

Bryan Cantrill:

And I think that's what folks lose when they think about the, "How do I build a business around open source?" It's like, that's the wrong question. When you're building a business, you are developing a product that people will pay for. The open source vector, it's not about monetizing open source. It's about using open source as a way of broadening your community, broadening your funnel, accelerating your existing business, not building a business around that.

Bryan Cantrill:
In that regard. I likened open source to a remote friendly workplace. We all know that a remote, especially now, a remote friendly workplace is the right answer. It allows us to find talent wherever that talent might be, it allows us to be more family friendly, and so on. So, there are lots of reasons to be remote from the workforce.

Bryan Cantrill:
Well, how do I monetize a remote friendly workforce? Well, wait a minute, what? It's kind of orthogonal, a remote friendly workforce is a tool that we use, as part of running a business, it's not the business itself, if that makes sense. And that's what I feel people have, if I may just be generational for a moment, that's why I feel this kind of younger generation a little bit. I'm going to sound like the old man that I am. But I feel that those folks that did not grow up in proprietary software are recreating a bunch of the problems that open source software solved. They're not seeing the fundamental problem here. And the fundamental problem here is you are trying to directly monetize open source software, that's not going to work.

Rashmi Mohan:
Thank you. I think that was an incredibly lucid answer. And certainly so, it gives us a lot to sort of think about. I have one other question for you, Bryan, and its somewhat sort of, it's a little bit different from just your overall career as software developer. But one of the things that I noticed through some of the conversations of yours that I've heard, is that Jeff Bonwick, played a really important role in your growth as a software engineer, right? That mentorship and that guidance in your early career seems to have really lost in view into some of the initial principles of software engineering that you learn. How do you feel that that has really changed your life? And how do you feel you've passed it on to others?

Bryan Cantrill:
I think it's more than just mentorship. It's more influence actually, I would say, that is that, really the service that, I mean, Jeff [inaudible 00:43:33] for me, was opening up and allowing me to flourish, contribute, take on hard problems.

Bryan Cantrill:
So, I think that in some regards, the most I mean, more formal mentorship is important too, of course, where you are much more in the kind of the mechanics of showing people how to do things and so on. But I think we really lead by our example. And it's very important that we are inclusive and encouraging of new thinking of folks that are outside the domain. People that don't know this stuff yet. I always loved what Jeff had to say about... It was very important to Jeff that we are able to explain things to new people. As he said, "People aren't born knowing this stuff."

Bryan Cantrill:
The thing that I really see that striking to me is kind of the way people treat knowledge and is knowledge something to be shared, or is knowledge something to be hoarded? And the people, especially if you're a young person, but the people you want to be around are the people that want to share knowledge that are excited to help that when they know something, it doesn't threaten them if you know it too.

Bryan Cantrill:
Sadly, that's not everybody. Sadly, there are people that want to use their knowledge, and they want to hold on to their knowledge, because God forbid, you get the same knowledge that they have, then what's their role, which is terrible to say. But there are a lot of people do implicitly or explicitly do that. And the engineers that I want to work with are those engineers that are very giving up their knowledge, that they are incentivized to me, and to make things easy to understand.

Bryan Cantrill:
I think that, you get this kind of perverse incentive, where people who want to hold on to their knowledge are also incentivized not to actually document what they know, not to explain it very well. You don't want any of those incentives. You want people to be incentivized to share what they know, to document what they know, to uplift other people, to help other people.

Bryan Cantrill:
So, to me, it's way more than just like mentorship. It's all about creating an environment in which newcomers can thrive and can people who are excited and motivated and willing to work hard, can uplift themselves and pass that on to other folks into another generation. And I think that, that is more than anything, it's that zeitgeist, that I very much got from Jeff and have tried to pass that on others over my career.

Rashmi Mohan:
That's such a golden nugget of advice, both to folks who have that knowledge to share as well as the ones who are seeking that knowledge. It's so important for us to so co-exist and be able to grow from that experience. This has been an incredibly eye-opening conversation, Bryan.

Rashmi Mohan:
For our final byte, I'd love to know, what are you most excited about in the field of computing or observability or whatever we may call it, over the next five years?

Bryan Cantrill:
Well, I mean, and this is where I'm going to talk my book, as the VC say, I'm very excited about the problem we're solving at Oxide. So, we are developing this RAC scale machine, as I mentioned. We are really trying to develop the system that I've wanted to develop for my whole career, which is, RAC scale computing, integrated hardware and software, taking a real systems approach. And this is a really hard problem.

Bryan Cantrill:

This is something that, honestly, that the origin story here is pretty funny in that, first thinking about kind of what we wanted to go do next. And I was recalling a venture capitalist that I'd known from years prior. And going back to my email just to remind myself of his e-mail address. And the last email he sent to me, maybe two years prior was, "Hey, look, Bryan, just as a reminder, I will fund anything that you put in front of me." I don't know, thinking like, "Is that true? Is that actually a literal statement?"

Bryan Cantrill:
And if that's true, then as I'm thinking about what's next, maybe I should actually be, and it sounds trite to say, but I should actually solve the problem that I would solve if I were unafraid to fail like, "What problem would I solve if I knew that I could raise money to go solve this problem? What problem would I solve?" The problem that I wanted to go solve is I wanted to actually go bring all these innovation from the Googles and Facebooks and Microsoft's of the world, I wanted to bring them to the mainstream enterprise market and actually develop a new computer company. That's what's in my marrow.

Bryan Cantrill:
And fortunately, and just blessed by the fact that lots of other great technologists have that same fervor, that same passion, that same mission, and we "were able to raise the money. Ironically, by the way, the VC that sent me that was one of the very first to just totally puke on this idea. He was just like, "Absolutely, I don't know. I would never find that. That's a terrible idea." I'm like, " Oh, okay." I'm actually really glad, clearly you were lying to me when you sent that email. And I'm really glad that you did, because that actually gave me the confidence to go do this. That's kind of a little irony. But I am really, really excited about it. And I feel that it is by far the hardest problem I have been associated with my career, so ambitious, so much at so many different levels of the stack.

Bryan Cantrill:
I joke that like, "Apparently, today is not going to be the day where I know how computers work." Because every day I'm learning something incredible, some incredible new domain that all of us are effectively built upon, that isn't well known. I mean, there's so many, it is such a miracle that these machines work at all. It's amazing.

Bryan Cantrill:
The hardware in the software, there's just so much, and to me, that is so exciting. It's exciting too, because this is on the one hand, it's kind of how I got into this business of going into OS kernel development at a time when I was told there was nothing left, and we are now a computer company. When people have kind of implicitly said that the innovation there is dead, and it's definitely and emphatically not.

Bryan Cantrill:
The innovation is very much alive at every layer of the stack. And it's really, really exciting. There's a lot of really exciting stuff happening. I think I've never been more excited for the next five years than I am for these coming five years. It's really going to be a terrific time.

Rashmi Mohan:

I mean, your excitement is absolutely palpable. And it's wonderful that you're sort of come a full circle in terms of that opportunity and being in that situation where you've seen the opportunity that many others haven't. We wish you well. And thank you so much for joining us at *ACM Bytecast*.

Bryan Cantrill:
You bet. Thank you so much for having me. Great conversation.

Rashmi Mohan:
ACM Bytecast is a production of the Association for Computing Machineries Practitioners Board. To learn more about ACM and its activities, visit ACM.org. For more information about this and other episodes, please visit our website at acm.org/bytecast.