EPISODE 547

[INTRODUCTION]

[0:00:00.3] JM: Consensus protocols are used to allow computers to work together. A consensus protocol lets different servers agree on the state of the system. For decades, these protocols have been used to establish consensus among database nodes, application servers and other infrastructure that runs within an enterprise. More recently, new consensus protocols have been invented to allow crypto economic consensus to agree on the state of a financial system. The first crypto economic consensus protocol to reach wide adoption was a Nakamoto consensus. This was a proof of work system used for consensus in Bitcoin that is still in use today, and since then other systems have been developed with different trade-offs in security, speed and formal verifiability.

Ethan Buchman is the CTO at Tendermint; a consensus system for block chains, and in addition to working on Tendermint, Ethan works on Cosmos; a network of block chains. In this episode we talk about different consensus systems for centralized trustworthy systems, like Amazon, DynamoDB or S3 as well as trustless systems, like currencies.

It was a great episode. We went pretty deep on consensus protocols, and I hope you enjoy it.

[SPONSOR MESSAGE]

[0:01:31.3] JM: Software Engineering Daily is brought to you by Consensys. Do you think blockchain technology is only used for cryptocurrency? Think again. Consensys develops tools and infrastructure to enable a decentralized future built on Ethereum; the most advanced blockchain development platform.

Consensys has hundreds of Web3 developers that are building decentralized applications focusing on world changing ideas, like creating a system for self-sovereign identity, managing supply chains, developing a more efficient electricity provider, and much more. So, listeners, why continue to build the internet of today when you can build the internet of the future on the blockchain?

Consensys is actively hiring talented software developers to help build the decentralized web. Learn more about Consensys projects and open-source jobs at consensys.net/sedaily. That's C-O-N-S-E-N-S-Y-S.net/sedaily. Consensys.net/sedaily.

Thanks again, Consensys.

[INTERVIEW]

[0:02:47.5] JM: Ethan Buchman is the CTO at Tendermint. Ethan, welcome to Software Engineering Daily.

[0:02:52.3] EB: Thank you. Great to be here.

[0:02:53.6] JM: I think the root of what we're talking about today is consensus protocols, and then we'll get into the applications that can be built with consensus protocols, including cryptocurrencies and networks of blockchains. Starting from a base level, what is a consensus protocol?

[0:03:14.0] EB: Cool. Great question. A consensus protocol is any protocol that allows a diverse set of agents to agree on something. So I'd like to think of, really, any kind of shared state or kind of coordination that is successful as having achieved consensus and as implying some kind of underlying consensus protocol. For instance, we agreed on a time for us to meet today. So we had some consensus protocol that we were able to negotiate a time to come together. So humanity and human organizations and societies have, for hundreds or thousands of years, had consensus protocol running implicitly across them that allow them to have shared tradition, shared cultures, shared languages, allow them to understand each other and so on.

Starting around the late 1970s when computers were becoming more mature and engineers decided, "Hey, you know what? Let's use microprocessors to control aircrafts." Well, they realized they couldn't just use one microprocessor, because they needed more than one in case there was some kind of fault. When you're flying an airplane, it's a highly kind of adversarial environment. Your processor might — One of them might go down and there might be kind of arbitrary bit flips from the radiation, whatever. So they needed they needed multiple processors controlling the aircraft and they needed to make sure that those processors always agreed on the state of the aircraft and what to do next at all times. So that was kind of the motivation for the formal study of consensus protocols in computer science, and since then they've evolved into these algorithms and these protocols that enable computers spread out over the world or spread out over any kind of area to coordinate with one another to agree on some set of values or some set of actions to do next. Typically they're phrased in such a way that they're able to handle different kinds of faults so that even if some of the computers are faulty, the others are still able to agree on what to do next. So they've become very much a staple or a foundation of any kind of fault-tolerant computing.

[0:05:15.1] JM: How did you get interested in consensus protocols?

[0:05:17.9] EB: My background was primarily in biophysics, and I was studying the origin of life and kind of the emergence of complex, highly functional systems in this universe. To some extent, there's a little bit of a consensus mechanism going on when some highly organized system emerges out of the chaos of the progression of entropy in our universe and on our planet. So I was trying to make sense of that and understand how the physics and the thermodynamics could account for such, say, organization emerging out of chaos. This notion of the sum being greater than — Or the whole being greater than the sum of its parts, and the notion of kind of a reliable system from unreliable parts. So I was looking at it from a biophysical perspective, how kind of the chaos that is our internal physiology in this diverse set of cells we have managed to coordinate to this wonderful, beautiful dance that is life.

Somewhere around 2013, Bitcoin showed up on my radar, and when I tried to understand it and I learned about what was going on, I realized that here was the same phenomenon happening in the digital medium that I had been studying in the biophysical media. This idea of unreliable elements coming together to form a much more reliable organized whole. So from there I kind of just fell in love and went — Drop down the rabbit hole and started studying what was going on there and came to realize that at its foundation there was this consensus protocol that made Bitcoin tick. Basically, ever since then, ever since I realized that that's what underlie this whole thing and that it had an old history in computer science, I became infatuated with it and I've been studying that ever since.

[0:06:57.8] JM: Did you read a lot of the classic research, like Lamport, and Lampson, and Liskov, or did you mostly read the more recent applied stuff, like reading about maybe how a distributed database works or how the blockchain works?

[0:07:16.1] EB: So I read a lot of it. I remember having some epiphany. I think it might've been in late 2013 or something, because at that point I thought Bitcoin was kind of the only one that had solved this consensus problem, and then I started hearing about this thing called Paxos, which apparently had been invented in the 90s and solved the consensus problem long before Bitcoin did and I was horrified to find out that there is this whole academic literature that I hadn't known about yet thinking I was becoming some kind of Bitcoin expert.

So once I discovered Paxos, I started gobbling up all the academic literature on consensus that I could find, and by mid-2016, when I completed my master's thesis, it was on consensus protocol specifically on Tendermind and kind of consensus protocols in the age of blockchains, and there is a big section at the end of that thesis that does a grand sort of literature review over the history of academic studies of consensus protocols and how the more modern blockchain-based consensus protocols fit into that mix.

[0:08:10.8] JM: That's perfect if you've done a literature review, because I wanted to ask you; what are the milestones in the history of consensus protocols? What are the main — The important milestone developments? I know that there're actually a lot of them, but are there some specific ones that come to mind?

[0:08:26.8] EB: Totally. I love this question. So, like I mentioned, it all kind of started when humans decided they wanted to use microprocessors to fly airplanes. So that prompted — Leslie Lamport was working on that project and it prompted him to write the kind of paper which kicked the whole thing off, which is the time clocks and the ordering events in a distributed system. I think this is written in 1978 and it laid the foundation for consensus protocols and really for fault-tolerant distributed computing where he defined this notion of causality in a distributed system based on communication events between concurrent processes, and he drew a lot of analogies to relativistic physics and things like this and drew pictures very similar to the ones that we see in kind of quantum relativity in physics.

So in that paper he outlined the very first consensus protocol which worked in a fully asynchronous network, but it actually couldn't tolerate any faults. So even if one of the processes failed, his consensus protocol didn't work. So it wasn't quite good enough to, say, to fly airplanes. It was a few years later, I think in ’82, that he coined the phrase the Byzantine Generals Problem, and it was him and a few others that wrote this paper on the Byzantine Generals Problem, which put forth the idea that not only do we want to have consensus protocols in a context where some of the agents or some of the computers can crash, can go off-line, or can have no significant delays in message latency or in message delivery. We also want to be able to build consensus protocols that can tolerate explicitly adversarial behavior, or malicious behavior, or to generalize it, completely arbitrary behavior. So a computer acting in a way it wasn't programmed to act. We want to build consensus protocols that can tolerate that. That's really what you need in the kind of adversarial condition that exists on an airplane where it’s being constantly bombarded by radioactive particles from outer space and the guarantees that you typically have on a microprocessor in a safe space, like in the atmosphere on earth. On the ground, you don't have up there, because bits can kind of flip a little bit more arbitrarily.

So the Byzantine Generals Problem was coined, because he was thinking in analogy with the problem of a couple of or a set of Byzantine Generals trying to coordinate to attack Rome, and they’re spread out on the different corners of Rome and they can only coordinate by messenger, and one of them might be a traitor. So what kind of protocol could they use to coordinate to attack Rome at the same time communicating only by messenger even if one of them was a traitor? They came up with a solution to that problem, but it only worked in a synchronous network, in a network where you have an upper bound on message delivery. For instance, you know how long it takes a messenger to get to one general and back. That was in ‘82.

In ‘85 was kind of the landmark result of consensus science, which is known now as the FLP impossibility result. FLP stands for the authors of the paper; Fisher, Lynch and Patterson, and they showed that actually in an asynchronous network, that is in a network where there is no upper bound on message latency. So messages can take an indefinite amount of time to be delivered. You can't timeout and say, “Oh! It should have been delivered by now.”

In such a network, it's impossible to achieve consensus with a deterministic protocol, right? To just say that again; Consensus, the FLP impossibility result, 1985 landmark result, that consensus is impossible in an asynchronous network with a deterministic protocol. So this kind of shaped the landscape of consensus science ever since.

So, basically since then, and actually started a little bit before, because that intuition was around before it was formalized in that paper, there have been the kind of field of consensus science split in two based on those two assumptions within that result; one of an asynchronous network and the other of a deterministic protocol. Ever since of then — And, again, starting slightly before, there's kind of been these two branches to solve the consensus problem, because consensus is impossible in an asynchronous network with the deterministic protocol. But doesn't say anything about a synchronous network or a nondeterministic protocol, and that has kind of set the stage.

So in '83 and '84, even before the FLP result was published, there were a couple of results by Ben Orr and Michael Rabin that showed how you could actually solve the consensus problem in a fully asynchronous network by using a nondeterministic protocol, and that is very, very cool stuff and it is come to be known as kind of the coin flipping approach or the common coin approach to consensus.

The other, which really had its landmark results in the late 80s. I think it was 1988, by Dwork, Lynch and Stockmeyer. So they won a Dijkstra Prize for this. It's now known as the DLS Consensus Result. They introduced the notion of weak or partial synchrony, which is the idea that there is a spectrum between a fully synchronous network and a fully asynchronous network. If you can pick spots along that spectrum that aren't quite fully synchronous, that aren't quite fully asynchronous but kind of give you the best of both worlds, and you can solve the consensus problem with a deterministic protocol at those different spots along that consensus line. That produced this kind of natural question of; what is the weakest synchrony assumption required to solve the consensus problem? What is the weakest assumption we need to make about time within the network so that we can actually solve consensus?

There was a number of results in the 90s that really started to look at this that use abstractions around failure detection, or failure detectors, which tend to be implemented as timeouts that were able to show what that weakest assumption looks like and what it requires and what kind of previous results actually correspond to that. That’s sort of on the theory side.

On the practical side, it wasn't until Lamport came up with Paxos in around ‘89 or ‘90 that we really saw the first realistic, practical consensus algorithm that could be engineered. Of course, the problem with Paxos was that it was so complicated and so difficult to understand that it didn't actually get published until roughly 10 years later. So it wasn't published until about ’99. Halfway through, I think in ’95, one of Lamport's friends published a paper called Paxos Made Simple, or something like that, trying to explain to people how this thing actually worked and how they could implement it.

Yes, Paxos kind of started the field of practically engineered, practically deployable consensus algorithms even though it was very difficult to understand, and everyone that’s gone and implemented Paxos since then, and that includes Amazon and Google and various others, there’s something like Paxos sitting at the bottom of basically every major internet infrastructure provider. They all have various kind of black magic, black art kind of ways to make it work and to make it real and it’s very difficult bring new engineers into those systems.

So it wasn't until 2013 when Diego Ongardie I think his name is. I forget his last name, but he was at Stanford, and he decided for his Ph.D. he was going to try to reinvent consensus protocols from scratch where the primary goal was understandability. So he wanted to build a consensus algorithm where the most primary design criteria was that it could be understood by first year undergraduates, and that became the Raft Protocol, and it has been hugely successful and there's a large number of open source projects that are now using Raft as their foundation instead of Paxos, because it's much easier to understand Raft. It has a complete specification. It's been formally verified using a number of different tools. That's kind of a history from the non-Byzantine fault-tolerant perspective.

In the Byzantine fault-tolerant case, actually, the DLS paper from late 80s was Byzantine fault-tolerant, but it wasn't until ‘99 that Barbara Liskov came out with PBFT; practical Byzantine fault-tolerant, which was a fully-defined version. Kind of like a Paxos, but it was Byzantine fault-tolerant that could be implemented, that they did implement, that worked in an asynchronous network that actually solved this problem of Byzantine fault-tolerant and kind of became the industry standard.

Now, what was interesting about BFT, even though it was a very hot academic topic in after ’99, in the early 2000's, a lot of interesting innovations were done in that space. It was never really deployed widely, or practically, or at all, and there weren't really any usable implementations that anyone cared to use. It wasn't until Bitcoin showed up on the radar, it came out in 2009, that people started to realize that actually Byzantine fault-tolerant might have more wide applicability than we ever thought.

I think a significant reason why that was the case is because until then, and even until now, most consensus systems that are deployed, most internet infrastructure is controlled by a particular entity who has a set of employees with legal employment agreements that are responsible for the operation of the databases so to speak, right? So wasn't really this need, or as much of a need to be able to tolerate the kind of adversarial and malicious behavior that you do need in a global decentralized anonymous public cryptocurrency setting.

So I think it was in early 2011 that people started to realize, primarily Andrew Miller, that actually Bitcoin is another solution to the Byzantine Generals Problem, and can find its place in this much longer, decades long tradition of consensus research. Then in 2014, the kind of next step in the evolution of that, was when Jae Kwon, the founder of Tendermint and my partner in everything I’m working on now, realized that, “Hey, wait a minute, why don't we go back and take some of the classic academic Byzantine fault-tolerant protocols, upgrade them into the domain of blockchains and cryptocurrencies and launch a cryptocurrency that draws on this rich academic tradition that is well understood and can be formally verified and have something that is fast and efficient and safe as a foundation for building cryptocurrencies on?” So that was the genesis of Tendermint and many others have followed since then, including IBM and J.P. Morgan and many others.

[SPONSOR MESSAGE]

[0:18:10.5] JM: We are running an experiment to find out if software engineering daily listeners are above average engineers. At triplebyte.com/sedaily, you can take a quiz to help us gather data. I took the quiz and it covered a wide range of topics; general programming ability, a little security, a little system design. It was a nice short test to measure how my practical engineering skills have changed since I started this podcast. I will admit, although I've gotten better at talking about software engineering, I have definitely gotten worse at actually writing code and doing software engineering myself.

But if you want to check out that quiz yourself and help us gather data, you can take that quiz at triplebyte.com/sedaily and in a few weeks we're going to take a look at the results and we're to find out if SE Daily listeners are above average. And if you're looking for a job, Triplebyte is a great place to start your search, fast tracking you at hundreds of top tech companies. Triplebyte takes engineers seriously and does not waste their time. I recommend checking it out at tripledbyte.com/sedaily. That's triplebyte.com/sedaily.

Thank you, Triplebyte, for being a sponsor.

[INTERVIEW CONTINUED]

[0:19:41.5] JM: So you've ou really laid out the timeline in a detailed fashion. I appreciate that. To put some finer points on that, the consensus protocols are either Byzantine fault-tolerant or not, and Byzantine fault-tolerance allows computers to reach consensus, that is agreement, even when there are nodes in the network that are acting maliciously.

So it's important to distinguish; are you on a network where people are potentially acting maliciously? Because if you're talking about a production grade consensus system based on Paxos or Raft, these consensus systems that often power the Hadoop distributed file system, or distributed databases, or etcd powering Kubernetes, these are often running on trustworthy infrastructure.

Actually, I guess I should ask; do those kinds of systems allow for Byzantine failures if I'm running my Kubernetes cluster or my distributed database, do the algorithms that underpin consensus across those different nodes, do those typically tolerate Byzantine failures?

[0:20:56.6] EB: They do not at all. One single Byzantine failure within one of those networks can take down the whole system.

[0:21:02.3] JM: That's a characteristic of Paxox and Raft, or is that a characteristic of the implementations —

[0:21:07.2] EB: No.

[0:21:07.7] JM: It's a characteristic of the algorithm itself.

[0:21:08.5] EB: That's right, yeah. Paxos and Raft are not Byzantine fault-tolerant.

[0:21:12.6] JM: Okay. So Paxos and Raft don't really work for economic systems.

[0:21:18.3] EB: For sure, not.

[0:21:19.1] JM: Right. What you referred to there with the blockchain, the fact that the blockchain is — Well, the distributed Nakamoto consensus, this is another approach to consensus that is Byzantine fault-tolerant. Of course, the problem with the crypto economic consensus, the Nakamoto consensus as it was implemented first in Bitcoin is that it requires the agreement of all the nodes in the system, and I think what you're getting at there with the developments that Jae Kwon made and I guess the applications of, I think, PBFT; practical Byzantine fault-tolerance, is that you can accelerate the rate at which nodes come to consensus if you just, I guess, have an assumption that two thirds of the nodes have come to agreement. Maybe you could shed a little bit more light on the developments between Bitcoin, because we've talked about proof of work at length in a lot of previous episodes. So I don't think we need to rehash proof of work. Maybe you could talk about the developments between proof of work, the Nakamoto consensus, the original Bitcoin model for consensus and the applications that Tendermint developed, the crypto economic consensus that Tendermint implemented.

[0:22:38.8] EB: Sure. Yeah, I'd love to. So I think it might be useful to first actually cast Nakamoto consensus in light of what I've been talking about in terms of synchrony assumptions and Byzantine fault-tolerant and especially with respect to the FLP impossibility result. So because I find this kind of illuminating and kind of amazing that no one came up with it before, and when you put it in this light, it's kind of a weird solution to the consensus conundrum that was presented by FLP.

As a reminder, FLP said that consensus is impossible in an asynchronous network with a deterministic protocol. There are two ways to get around that, either you make a weak synchrony assumption or a strong synchrony assumption if you'd like, or you introduce nondeterminism. What's interesting about Nakamoto consensus is that it did both. So the proof of work that Satoshi Nakamoto introduced as a mechanism for achieving consensus actually makes a very strong synchrony assumption and it hedges the failure of that assumption using randomness. So that's a kind of very clever solution that is somewhere in the design landscape that no one had really explored before. So since then there's been an explosion of research into trying to better understand and to understand what happens to something like Bitcoin in case some of its assumptions are violated, for instance, it's assumption of synchrony, right? So it is very well known now that Bitcoin requires a strongly asynchronous network and that it does not work at all in an asynchronous network, and we see that empirically in the way that miners have set up these very, very tight relay links with each other that are on separate kind of overlaid networks so that they have direct links to each other so that they can get blocks from each other as absolutely quick as possible to mitigate any kind of asynchrony in the network.

So that's really interesting, but there's been a lot of other work in kind of exploring that Nakamoto design space, things like GHOST; the Greedy Heaviest Object Sub-Tree first, which is the proof of work that Ethereum uses, which is able to reduce the size of the synchrony assumption. I should say, the synchrony assumption that Bitcoin is making is that the time to find the block, i.e., the time to solve the partial hash collision, that is the proof of work puzzle, is much greater than the time to propagate the block over the network. Empirically, it takes about 10 minutes to solve the partial hash collision and it takes maybe 10 seconds to propagate the block across the network, but that it's very important that there's an order of magnitude difference in there to kind of shield against any variations in propagation latency.

It's well known in the Bitcoin community that if you try to reduce the time it takes to find the blocks, that you can have blocks faster. For instance, you want more throughput, you want lower latency, you're actually somewhat compromising the security, because the security of the system rests very much on that assumption that you can propagate blocks much faster than you can find them or that you can solve for them.

So something like GHOST, which is what Ethereum uses, is able to get around that a little bit by including additional information about the network behavior within the consensus protocol, right? So in Bitcoin, if multiple miners find the block at the same time, all of those blocks except for one are going to get orphaned. They're not going to get included in the main chain and it’s just wasted work. In something like GHOST, information about those other blocks is actually included and inform the longest chain rule. That actually allows you to reduce the overall block time and strengthen without weakening the guarantees of the consensus.

So there have been a lot of other approaches similar to those to try to understand how we can use similar kinds of techniques either to bring down the block time or to not require proof of work or to require different kinds of puzzles and so on. So it’s very interesting.

That said, all of those systems are still making some kind of a synchrony assumption, and none of them really went and said, “Well, what if we don't want the safety of our system to be dependent on a synchrony assumption?”

So I should mention that consensus algorithms are typically cast in terms of the key features or the key guarantees they provide, which are called — They’re known as safety and liveness. These are actually very common. They go much more beyond consensus protocols. They kind of relate to and and or duality in logic. These are kind of like duals of each other.

Safety is the idea that nothing bad will happen, that everyone will agree on the same value and no one will disagree, for instance. And liveliness is the idea that eventually something good will happen, right? That eventually the system will make progress. What FLP showed is that you can't get both of those with the deterministic protocol in an asynchronous network. So every academic between the FLP resolved and Bitcoin decided that the thing they wanted most was safety. That it was okay to compromise the liveness of the system so long as the system was always safe. That no two correct nodes ever disagree, because that was the whole point of a consensus algorithm in the first place, was that everyone agrees on the correct value.

So what Bitcoin did was it kind of turned that on its head because it said, "You know what? Actually, we don't care that much. We're okay with a little bit and disagreement in the interim, so long as we have a way to resolve it," and the whole proof of work — Or most difficulties chain as a resolution to that kind of solved that. But it requires for safety for it to actually work out that there is strong synchrony in the network, and if that synchrony is violated, for instance, if there is a long-lasting network partition, if the underwater cable, the fiber-optic cables between North America and Europe get cut and that network is partitioned on the order of a day or a few days, which is totally within the realm of possibility, then Bitcoin is — It's unclear what's going to happen, because North America and Europe are going to go on their merry ways minding their own Bitcoin chains for a few days engaging in commerce and so on. When the partition heals, the longest chain is going to win a days' worth of economic activity is going to get reverted, and that's going to be quite chaotic.

What Jae Kwon did — Hi realization was, it was, "You know what? If we're building financial infrastructure, we don't actually want to depend on synchrony assumptions for safety, because it's very realistic in this hostile adversarial world we live in that there are going to be long-lived network partitions, and if our system is going to have fundamental inconsistencies within it because of the failure of those synchrony assumptions, then we're going to be in a world of trouble. So why don't we go back to the old academic literature which puts safety first about liveness and all else and use those protocols that can operate in a fully asynchronous environment without compromising safety?"

So the trade-off there is, unlike Bitcoin where even if you're cut off from the whole rest of the network, you can continue mining on your own and it will get sorted out later when you reconnect. In a proper academic BFT kind of world, that's not the case. If you're cut off from the rest of the network, you can't make any progress. So there are strong thresholds, and this is where the two-thirds thing that you are referring to come in. There are thresholds that must be met to make any progress, and those thresholds basically imply that you have to have some level of connectivity in the network, that if too much of the network is partitioned, the whole thing will come to a grinding halt. That's the trade-off that systems like Tendermint and others like it make compared to Bitcoin in terms of what they do in the event of a network partition and how their assumptions about synchrony in the network kind of influence their take on safety and liveness.

To kind of summarize Bitcoin, require synchrony for safety, but not for liveliness, and Tendermint and systems like it require synchrony for liveness, but not for safety. So even in a fully asynchronous network, Tendermint will never allow multiple nodse to disagree about a result.

[0:30:03.4] JM: Okay, we've now explored a range of consensus mechanisms. Let's talk about an application of them, which is Tendermint. You are the CTO of Tendermint. Explain what Tendermint is.

[0:30:17.3] EB: Cool. The term is a little bit overloaded, because we use it to refer both to the consensus algorithm and to the implementation of it that we've developed and to the company. It's kind of many things in one. So in the firsthand, Tendermint is a consensus algorithm. That's modeled after the original DLS, Byzantine fault-tolerant protocol, but has evolved to be a little bit more like PBFT and to have been upgraded into the kind of blockchain paradigm. So I've been arguing that actually the blockchain data structure is really an optimization in the design space, a Byzantine fault-tolerant design. So if you take PBFT and you try to upgrade it into the blockchain model to inherit some of these optimizations, you end up with something that looks very much like Tendermint. So that's the Tendermint meant consensus protocol.

Now, what we've gone and done is implemented that in a high-performance secure open source software that is called Tendermint. You could find it on GitHub, github.com/tendermint/tendermint. It's written in Go, the Go Programming Language, and it can process thousands of transactions per second on nodes spread out across the world on hundreds of nodes all over the world, and we have been working on that for the past few years and trying to develop it in such a way that it's as general-purpose as possible.

So unlike many other implementations of consensus protocols where the application, the state machine is actually baked in to the rest of the software, we've separated that out completely from Tendermint. So Tendermint itself, the software, is completely agnostic to the actual application that runs on top of it. So it's a kind of general purpose, Byzantine fault-tolerant state machine replication engine is the way I like to pitch it. What that allows you to do is to build an application in any programming language you what. You can write your state machine in Python, or Haskell, or Java, or C or whatever language you like, run it in one process on your operating system, run Tendermint in another process and have Tendermint handle all of the networking concerns in the consensus algorithm and all of that kind of complex stuff and only forward messages to your application when it is certain that everyone agrees on them. Then your application can process those messages however it wants, decode them however it wants, maintain state however it wants, return results to Tendermint, and Tendermint takes care of everything else. That's Tendermint, the software. We've seen a lot of uptake on that, a lot of success. It's being widely recognized as pretty much the most mature implementation of Byzantine fault-tolerant state machine replication I think available. There's a diverse community of people implementing state machines in many languages, in Go, Java, C, Haskell, Erlang, Python, etc. If I didn't mention a language that you like writing in, we're always excited to see new implementations. That's Tendermint itself.

But the original vision and design for Tendermint wasn't just this general-purpose state machine replication engine. It was actually a cryptocurrency, and specifically a proof of state-based cryptocurrency. So even though over the course of a few years, Tendermint evolved to be more general than just that. We've decided as a company to kind of come back to that original vision and to say, "Now that we've built this general-purpose module, this state machine replication engine that's useful to many different people for many different things, that is kind of a database they can rely on, let's go build an application on top of that. Let's go build a proof of stake, a public proof of stake cryptocurrency network," but we don't just want to build kind of any random cryptocurrency. There're enough of those already. We want to design our cryptocurrency so that it can actually solve many existing problems within the cryptocurrency community, and those include things like speed, scalability, security and flexibility and interoperability, especially. Many cryptocurrencies today have no kind of capacity to interoperate with one another.

So we've put a lot of time and attention into designing this crypto currency network using well-defined subcomponents or modules that can be pieced together to give you a wide diversity of applications so that you're not locked into one particular application design. For instance, a huge amount of our motivation for doing that came from our experience working with Ethereum. So Jae and I have been working with the Ethereum technology since pretty much 2014. We've been developing on the Ethereum virtual machine since then. We've maintain forks of the EVM. We've implemented the EVM from scratch ourselves and it's part of the Hyperledger Project actually, and we kind of found that there are a lot of limitations there, a lot of weird design choices and that people are really locked into that design. It's very hard to experiment with new forms of cryptography, new forms of programming languages, new forms a cryptocurrency design and so on.

So we wanted to build our system in a way that it would be able to support a whole host of innovation and experimentation in cryptographic primitives and language design and virtual machine design and so on, but still enable these different innovations and different experiments to interoperate with one another. The set of protocols we're building to achieve that is what we now call the Cosmos Network. So that's what really our company is primarily focused on building right now, is this internet a blockchains. This network of blockchains that we call the Cosmos Network that will consist of many different independent blockchains most, if not all of them, at least initially, running on top of Tendermint and hosting a state machine that can be very application-specific and very different from the other state machines, but still able to interoperate with one another.

[0:35:29.8] JM: Okay. So I want to scale back a little bit and talk about Tendermint a little bit more before we get into Cosmos. So I want to make it clear to people what Tendermint is useful for. Tendermint has something called the application block chain interface. This allows people to build applications on top of the Tendermint consensus engine.

So people have other ways to build systems on top of other consensus engines. You could build a system on top of Bitcoin. You could build a system on top of Ethereum. Why would you choose to build something on top of Tendermint?

[0:36:09.3] EB: Great. Because it gives you the maximum amount of flexibility. So the reason we introduce the ABCI, the application block chain interface, was to allow developers to write their applications in whatever programming language they wanted. So when you're building an application on Bitcoin, you're confined to a very rudimentary constrained set of opcodes. The Bitcoin scripting languages is a stack-based scripting languages. It looks a lot like Fourth and it has a very limited functionality. If you're trying to build an application that uses the Bitcoin scripting language, you're extremely, extremely confined in what you can do, right? You can't use a nice high-level programming language like you're used to, because Bitcoin just doesn't support it.

So that was one of the motivations for Ethereum, which decided, “Hey, why don't we just go and take something like Bitcoin scripting language, but make it fully touring complete? We’ll define a full-blown virtual machine. We’ll give it all the opcodes we need to do all the computations we need. It will give it jump opcodes so we could do for loops and more complicated if statements. We’ll give it storage opcodes so that it can persist things in an internal state so that we can do rich kind of stateful operations.” Bitcoin can't do that all. Bitcoin has no actual persisted state.

So Ethereum was kind of this very exciting thing to be able to do that, but even still — Even within Ethereum, even though it’s this general-purpose thing, it's still very confined in terms of how you have to write your applications. You can't use existing programming languages. You have to use the programming languages that were designed to target the Ethereum virtual machine. There're a lot of very weird quirks within the design of the Ethereum virtual machine that you have to get used to. It wasn't really designed with safety in mind in the sense that it's very easy even for some of the most Ethereum developers on the planet to write code that isn't safe and that results in the loss of potentially millions of dollars. We’ve seen that a few times. We’ve seen losses of tens of millions or even hundreds of millions of dollars. So it doesn't quite have the guarantees and the testing infrastructure and debugging infrastructure that a mature professional software engineer expects from an application development platform, right?

So based on our experience with those kinds of applications, as much as I love Ethereum, and I love being part of the Ethereum community and I love the kind of passion and zest of it, the current design of the EVM leaves much to be wanted, right? It is very concerning to me and I'm very actually scared of writing applications on Ethereum because of how difficult it is to ensure their integrity. So I kind of have this this mantra now that is to write as little solidity as possible to get what you need done. Solidity, of course, the high-level programming language that is used to target the Ethereum virtual machine.

So what we were able to do is we said, “Okay, Look. Instead of defining a full-blown touring complete virtual machine, why don't we introduce an interface? A little bit lower in the stack that can actually separate the concerns of the consensus engine from the concerns of the application state, and that way we can allow the application to be written in any programming language you want. Literally, any existing language you want. You can take an existing code base where you've already written most of the state machine or, if not all the state machine, add a few tweaks so that it can fit into the ABCI format and then now suddenly your finite state machine which used to run on a single machine can now run in a distributed fashion, fault-tolerant on machines across the planet guaranteed to be kept in sync by the Tendermint consensus.

So that ABCI is that interface we introduced to make that separation to enable application developers to use existing programming languages with rich debugging and testing environments that they're comfortable with, that they've been programming in for 10 years to build their application logic that's going to run in this distributed context.

[SPONSOR MESSAGE]

[0:39:54.2] JM: If you are on call and you get paged at 2 a.m., are you sure you have all the data you need at your fingertips? Are you worried that you're going to be surprised by things that you missed, errors or even security vulnerabilities because you don't have the right visibility into your application? You shouldn't be worried. You have worked hard to build an amazing modern application for your customers. You've been worrying over the details and dotting every I and crossing every T. You deserve an analytics tool that was built to those same standards, an analytics tool that will be there for you when you needed the most.

Sumo Logic is a cloud native machine data analytics service that helps you run and secure your modern application. If you are feeling the pain of managing your own log, event and performance metrics data, check out sumologic.com/sedaily.

Even if you have your tools already, it's worth checking out Sumo Logic and seeing if you can leverage your data even more effectively with real-time dashboards and monitoring and improved observability. To improve the uptime of your application and keep your day-to-day run time more secure, check out sumologic.com/sedaily for a free 30-day trial of Sumo Logic.

Find out how Sumo Logic can improve your productivity and your application observability whenever you run your applications. That's sumologic.com/sedaily.

Thank you to Sumo Logic for being a sponsor of software Engineering Daily.

[INTERVIEW CONTINUED]

[0:41:38.3] JM: I'm going to explain Ethereum and Tendermint as I see them and I want you to tell me what I'm mistaken about. So from what I heard you explain — So first of all, you got Ethereum, where solidity is the programming language of choice. Solidity is used to write smart contracts. Solidity compiles down into EVM code, Ethereum virtual machine code, which is low level opcodes. There are other programming languages that could be built on the EVM, but it's just important to note that everything gets compiled down into EVM byte code and the nodes across the Ethereum blockchain are reaching consensus on the order of operations that are in EVM byte code. So that's Ethereum.

So if you want to write an application on top of Ethereum, you have to write an application that is compliant with EVM code that ultimately is probably going to be written in solidity or some other high-level language that compiles down to the EVM, and your critique of that is that, if I get it correctly, you — In order to do that, you are building applications that manage both the state and the computation all in EVM code, whereas in Tendermint you're saying, "Well, maybe we don't actually need to manage the computation and have consensus over the computation. All we need really is consensus over the state." So the Tendermint consensus engine manages consensus over state and whereas you can actually have the computation execute in any programming environment. Is that an accurate contrast?

[0:43:25.1] EB: It's close. I hesitate to say that Tendermint is managing the state, because actually the application you build is responsible for managing the state itself. So it's not so much that we're saying there's this nice separation kind of naturally in these consensus systems between the underlying replication engine, i.e., the actual consensus protocol and the application logic. Typically, we think of consensus protocol — The reason people are interested in consensus protocols is because they're trying to do state machine replication, right? They've built the state machine, it takes in transaction, it processes them, it updates the state and it outputs a result. They want to be able to run that state machine in a fault-tolerant context, which means they need to be able to run it on many computers simultaneously in such a way that they always agree on the order of transactions and the latest state and the results. So they always process things in the same order and get the same result.

So there's this nice separation then between the state machine itself and the replication component, and in most of the blockchain frameworks out there, they're very much confounded and it's hard to tease apart those distinctions within the software. What Tendermint went and did is made that absolutely explicit. It said, "Look. Tendermint is only concerned with the replication component. It makes absolutely no requirements, it has no opinions about what you're going to do in your state machine." Your state machine can be as general-purpose as you want. Your transactions can mean whatever you want them to mean. They can use whatever serialization or encoding format you want. You can keep state however you want to keep it in an SQL database or in a Merkel Tree or no state at all, completely up to you.

All Tendermint is going to do is handle the replication of some transactions, which contain arbitrary bites that Tendermint knows nothing about. Once it gets consensus on an order for those transactions, it will forward them to the state machine, which can run another process, be written in any programming language. That state machine processes the transactions, updates the state, returns the result to Tendermint and Tendermint carries on.

What that do is you can take the Ethereum state machine. So the Ethereum state machine consists of all the transaction semantics and the EVM and everything to do with Ethereum accounts and ether and so on. That's the state machine. In Ethereum, the replication is done by the proof of work in the peer-to-peer layer. So what you can do is you can take the Ethereum state machine exactly as it's written today in Rust or Go or whichever language you want and run it as an ABCI application on top of Tendermint. So you have the exact same semantics, the exact same transaction formats, all of the existing client libraries and client code works out-of-the-box and you're using the Ethereum, the literal code that the Ethereum team wrote, that's been audited, that's running on the main network, but instead of being replicated using proof of work, it's replicated using the Tendermint consensus algorithm. That's something we've actually done. It is a product we call Ethermint. You can find it on GitHub as well, github/tendermint/ethermint, and it's literally importing the code from Go Ethereum, wrapping it in the ABCI kind of interface and then plugging it into Tendermint so that Tendermint can handle all the replication.

[0:46:30.1] JM: Got it. So what's an example of something that people have built on top of Tendermint?

[0:46:36.7] EB: So there's lots. There's a foreign-exchange running in Europe that is apparently processing about half a billion dollars a day worth of foreign exchange transactions, and that's running on top of Tendermint. There's Ethermint. So people are looking at — There's huge scalability problems on Ethereum. There's limits to how much you it can process per second. Part of that is due to the proof of work. So people are looking at Tendermint as a possible solution to say, "Hey, let's run the same state machine, but on a faster replication engine, and that'll mitigate some of our scalability concerns." Not all of it. It's not a long-term solution. It helps a little bit, right? There are people building energy trading infrastructure.

[0:47:09.3] JM: Let's zoom in on that, that foreign-exchange. I'd love to learn more about it. Just as an example, for people to understand like how Tendermint is useful, explain how a foreign-exchange — That's a currency exchange, I guess?

[0:47:21.2] EB: Yeah.

[0:47:21.6] JM: Okay. So explain how a currency exchange would utilize Tendermint.

[0:47:25.4] EB: Sure. Again, I mean, Tendermint supports any kind of deterministic finite state machine. So that means any application that is deterministic and is guaranteed to terminate so it won't go into an infinite loop. Tendermint will support it. So pretty much — Almost any application you've interacted with will tend to satisfy that kind of requirement. For instance, in an exchange, you have buyers and sellers that are placing orders. Someone wants to buy euros for dollars at this rate and someone's willing to sell dollars for euros at this rate, and so you have this order book, and that's the state. Anytime someone submits a new order, that's a transaction. That gets processed by the application and added to the state. Then this state machine, this application is checking, "Are there two orders that overlap? Is there a buy order that matches to a sell order? If so, execute those two orders against each other so that they actually — The exchange actually takes place."

So that's a relatively simple state machine that is not too difficult to define that you can build and run it on Tendermint and thereby — The reason you'd run it on Tendermint is because that is what gives you the kind of fault tolerance so that you can run it on many computers and not be too concerned if one of them goes down. Furthermore, because Tendermint is Byzantine fault-tolerant, you can actually run that exchange in a more un-trusted environment, right?

For instance, we have many exchanges in the U.S. or all over the world that are kind of centralized entities that maintain their own order book. But if we wanted to, we could decentralize that so that we have a number of entities responsible for actually running the computers, but no one of them have complete control of the order book, because it is distributed across all of them. So then you'd have — The state machine would still be the same thing. It would still be an order book like you're used to on any kind of exchange, but it's running simultaneously on computers owned by many different stakeholders who are running the Tendermint consensus algorithm to make sure that they all stay in sync even if some of them become suddenly malicious or adversarial.

[0:49:19.9] JM: So talking more about where this is going in terms of Cosmos. So Cosmos is this network of blockchains that take advantage of Tendermint. Can you explain with the Cosmos network does?

[0:49:33.8] EB: Yeah. There's a number of things that go on into the Cosmos network. First and foremost is that it's proof of steak and, specifically, it's security deposit based proof of state. So unlike proof of work where you have a bunch of anonymous miners and anyone who was mining power can participate in the consensus protocol, in proof of state you have to have coins, you have to have the native coins of the system. In Cosmos, the coins are called atoms.

So if you have the atoms in a security deposit based proof of state, the way it works is you bond those atoms in a security deposit. That security deposit is maintained by the block chain itself. When you do that, you are eligible to be what we call a validator within the network. The validators are the ones who are able to participate in the consensus protocol. So they kind of take the place of miners. The reason we create that security deposit is because it's a guarantee over the behavior of the validators. So if the validators try to do something malicious or they try to misbehave or the try to fork the blockchain, anyone on the network can submit proof that that happened. For instance, that they signed for two different blocks at the same height. Anyone can submit evidence of that and the block chain will automatically slash that validator's deposit or revoke the deposit. They'll destroy it.

So this security deposit is what provides the economic security to the proof of stake network, that is commensurate to the kind of economic security you get from proof of work where it comes from the amount of electricity that has to be wasted basically on a daily basis to secure the blockchain. Here we have coins that — It's not electricity or value being wasted, its value that's locked up that is destroyed only if a participant misbehaves. So you can actually get a much greater amount of economic security within such a system, because you're not constantly wasting — You're constantly burning that value as you go. You just kind of lock it up in deposit and only destroy if there's a malicious event. That's one element of Cosmos, is the fact that it's one of these security deposit based proof of stakes.

Another element is that there's a governance mechanism built in. Cosmos is designed so that it's kind of an autonomous self-governing organism. I like to think of it as. The participants in the governance process are those who have bonded their atoms. So the reason atoms exist is to be bonded. They're not designed to be a currency. They're not designed for speculation or for paying for your coffee or anything like this. They're designed to be placed in security deposits and left there to guarantee the behavior of the validators. Anyone with atoms bonded into one of these security deposit is eligible to participate in the governance process, and that governance process is very simple. Basically, people can make proposals written in English and they can vote on those proposals, and there's a constitution that underlies the whole network that says, "Once a proposal passes, once a certain number of atoms have voted for it effectively, then it's passed, and then the network must implement it." So that way the governance can pass upgrades, or new software that should be run, or new ways to change the software, or new ways they want pool of reserve atoms to be paid out to. So some of the transaction fees that are earned are kind of pooled in this reserve, that can then be paid out to different projects using the governance mechanism. That's another thing.

Then one of the main kind of driving force behind some of the excitement behind Cosmos is the interoperability. Cosmos comes with the protocol that's called the inter-blockchain communication protocol, IBC, and this is basically a formalization of the light client mechanism supported by Tendermint.

So just to back up a little bit and explain what I mean by that, in most consensus protocols and in Bitcoin and Ethereum and so on, you have to have — To get the full security of the network, you have to operate a full node, which means you have to run sync with everyone and download all the transactions and execute all of them the same as anyone else participating in that consensus. Of course, that's massive overhead and your cellphone is never going to be able to keep up. What we want is to build our blockchains in a way that they support an extremely high level of security for clients that aren't running the full blockchain protocol. So we call these clients light clients, because they're not running full nodes. They're not processing every transaction.

We designed Tendermint very specifically with light clients in mind. So everything we do in the Tendermint protocol is designed in such a way that we can provide light clients with as much security and as much assurance about the state of the network as possible. So Bitcoin has mechanisms like this as well. They're called SPV; simplified payment verification, though they're a lot clunkier and a lot less efficient than what we've designed in Tendermint, and part of that comes from the proof of work. Part of that comes from the proof of stake in Tendermint. Part of that comes from certain other things we've done in the design with the Merkel Trees.

Now, the really cool interesting thing you can do with light clients is that not only can a mobile phone be a light client of a blockchain, but one blockchain can be a light client of another, right? So if you have a simple, efficient light client protocol, you can program it into a blockchain itself, and that way you can have one blockchain be the light client of another blockchain and thereby securely exchange data between themselves. So that's essentially what IBC is. The inter-blockchain communication protocol is a light client protocol programmed into a blockchain to allow multiple blockchains to securely interoperate with each other by reading each other's state without having to process the entire set of transactions existing on both blockchains.

So one of the motivating design goals of Cosmos was to facilitate this kind interoperability using light client proofs, and so the way we've set it up is Cosmos is going to consist of many, many blockchains, but we don't want each blockchain to have to be a light client for each other, because even though light clients are efficient and cheap, that's still a lot of overhead. So what we've designed is a system where we have what we call a hub block chain and many zones. Each zone is itself a block chain. The zones, to be able to interoperate with one another, they use the hub. So each zone functions as a light client for the hub, and the hub functions as a light client for all the zones, and thereby the different zones can interoperate with one another. They can send coins back and forth to one another by passing them through the hub. That way we can get a rich diversity of heterogeneous state machines running on these Tendermint blockchains and have many of them have them be able to interoperate by passing coins and eventually bypassing data through the central blockchain hub. Then the goal for that hub is basically to have the most secure, highly available, globally distributed validator set on the planet.

In a sense, it's like we're putting up a new core or a new foundation for internet infrastructure that consists of a set of validators that are running a consensus protocol between themselves to always stay in sync to process many different kinds of transactions in a way that is highly secure and gives very strong guarantees to people all over the world that want to use various kinds of state machines that aren't controlled by a central party.

[0:56:24.6] JM: What's an example of a pair of blockchains on Cosmos network that would want to interact with each other? And explain how those two block chains would interact with each other.

[0:56:37.2] EB: So there's a long-term plan and then there's a short-term plan. The long-term plan is basically to do everything and to facilitate kind of arbitrary interactions between blockchains. There's a lot of work we have to do to get there and some advancements in cryptography that will help achieve that that are kind of on the horizon and that we're working on funding and doing the research with.

Now, in the shorter term, and what we'll be launching with this year is the ability just to transfer coins between two blockchains. The idea there is that — Well, for instance, suppose right now the problem on Ethereum is that you have all these different applications, all these different smart contracts that are all competing for the same space in a block. Not only that, they all have to be totally ordered with respect to each other. So even if you have two applications that never interact with each other that are just completely independent, there they need to be processed sequentially in the Ethereum blockchain and they have to share the same space. They're literally in the same block.

Of course, anyone who is paying attention to Ethereum earlier or late last year saw a single popular application could take down the whole network.

[0:57:41.2] JM: Crytokitties.

[0:57:41.9] EB: Exactly. So idea with the Cosmos, or one application of Cosmos is kind of a first order approach to scaling for Ethereum, and what I mean there is that we can run — Because we already know how to run the Ethereum virtual machine on top of Tendermint, and that can give us a 10 to 100-X increase in throughput alone, we can then run many distinct Ethereum blockchains on the Cosmos network. We can have — Call it Ethermint 1, Ethermint 2, Ethermint 3, and maybe on Ethermint 1 we run Cryptokitties, and on Ethermint 2 we run decentralized exchange, and on Ethermint 3 we run a our prediction market, and on Ethremint 4 we run some other application, what have you. That way, each of these applications can operate independently of each other, because most of the transaction that they're processing don't concern the other applications. They're relatively isolated. They don't need that other information, right? That way you don't have all that kind of crosstalk within a single block within a single blockchain.

But then in the event we don't want those things to be completely isolated, because then we wouldn't have any interoperability and we wouldn't actually have solved the problem. So what we'd like is for — Even though they're relatively isolated and most of the transactions are happening in this kind of isolated environment, if you then want to take your coins from your Cryptokitties blockchain or even take your Cryptokitties from your Cryptokitties Ethermint blockchain and move them to some other blockchain so that you can trade them on the decentralized exchange or bet them in the prediction market or use them as payment in some other application on another blockchain, you can actually send them out of the Cryptokitties blockchain, pass them through the hub and then on to the other Ethermint blockchain where they can be used however you'd like.

[0:59:23.8] JM: So it sounds like Tendermint and Cosmos looks at building different blockchains interacting with each other similar to how Ethereum is now looking at Plasma, if I'm not mistaken, which is like this kind of where you have each smart contract on Ethereum be its own tree of different blockchains . Is that an accurate analogy? Are those related ideas?

[0:59:53.9] EB: For sure. For sure. They are complementary in a number of ways, and we're working closely with a number of groups that are working on implementing Plasma to see how our various approaches can complement each other. There are different emphases in Plasma versus a Cosmos on how things are done, but they are complementary in a number of ways and we anticipate being able to implement Plasma-like mechanisms within the Cosmos network to provide the kinds of guarantees that Plasma provides, for sure.

[1:00:19.7] JM: Is there a token associated with Cosmos and Tendermint, or is it just an application tool?

[1:00:27.0] EB: Tendermint, no. Tendermint is a general-purpose Byzantine fault-tolerant state machine replication engine that's totally agnostic to anything to do with tokens, right? You could use it as a drop-in replacement to Raft in etcd or in Console or in Apache [inaudible 1:00:40.1] or whatever you like. No tokens required.

In Cosmos, Cosmos is very much a token-oriented system and it has a native token called atoms, and those are designed to be staked in these security deposits that secure the consensus mechanism. But it will also have support for many other kinds of tokens that haven't necessarily been invented or created yet.

So for instance, we want to build a peg to the Ethereum blockchain as it exists today so that people can move their ether from the Ethereum blockchain into the Cosmos network and use it to drive applications and pay for gas on the ethernet zones within the Cosmos network. There will be other currency — There will be other tokens like that within Cosmos and there will be mechanisms for creating new tokens and for using them in various different ways and having new kinds of crypto economic mechanisms and so on.

[1:01:29.7] JM: Cool. Okay. Well, I think that’s a good place to stop, Ethan. I think we covered a lot of ground. I'm excited to see where Cosmos and Tendermint go next, and thanks for coming on the show.

[1:01:39.5] EB: Thank you so much. It's been a pleasure chatting.

[END OF INTERVIEW]

[1:01:43.6] JM: LiveRamp is one of the fastest growing companies in data connectivity in the Bay Area, and they're looking for senior level talent to join their team. LiveRamp helps the world's largest brands activate their data to improve customer interactions on any channel or device. The infrastructure is at a tremendous scale, a 500 billion node identity graph generated from over a thousand data sources running a 85 petabyte Hadoop cluster and application servers that process over 20 billion HTTP requests per day.

The LiveRamp team thrives on mind-bending technical challenges. LiveRamp members value entrepreneurship, humility and constant personal growth. If this sounds like a fit for you, check out softwareengineeringdaily.com/liveramp. That softwareengineeringdaily.com/liveramp.

Thanks to LiveRamp for being a sponsor of Software Engineering Daily.

[END]