EPISODE 436

[INTRODUCTION]

[0:00:01.5] JM: Ethereum is a decentralized transaction-based state machine. Ethereum was designed to make smart contracts more usable for developers. Smart contracts are decentralized programs that usually allow for some kind of transaction between the owner of the contract and anyone who would like to purchase something from the contract owner. For example, I could set up a smart contract where a listener sends my smart contracts on ether and I send the listener a podcast episode automatically.

Smart contracts can also interact with each other to network together complex transactions. In the same way that web development has been made easier by platform as a service and software as a service, smart contracts will make building financial systems much simpler. It's going to be a really bright future, but in the meantime it's going to be rocky to get there because the infrastructure has a long way to go.

Today's guest, Preethi Kasireddy, is no stranger to blockchains. She's a blockchain developer who writes extensively about cryptocurrencies and she joins the show to describe how the Ethereum platform works, including the steps involved in a smart contract transaction.

This episode covers some advanced topics of Ethereum. Like yesterday's episode, I was out of my comfort zone and there's a good chance that you will be too. If you're looking to get acquainted with the more basic aspects of cryptocurrency, you can look at our back catalog. We have tackled the basics of cryptocurrencies in some of our episodes and you could find them in the Software Engineering Daily app for iOS or android. You can hear all of our old episodes and they're easily organized by categories, such as blockchain.

As you listen, the Software Engineering Daily app gets smarter and it recommends you content based on the episodes that you're hearing. We've done 600 episodes and there's a whole lot of content to find. If you don't like this episode you will easily find something that you do like using our recommendation system. These mobile apps are open sourced at github.com/softwareengineeringdaily. If you're looking for an open source project to hack on, we would love

to get your help. We're building a new way to consume software engineering content. We have the android app, the iOS app, the recommendation system, web frontend and more projects coming soon. We would love to get your ideas for how you want to consume media about software engineering.

If you're interested in contributing code, of course, check out github.com/softwareengineeringdaily. You can join our Slack channel. We've got a nice little community going. There's a link on softwaredengineeringdaily.com for the Slack channel, and at any time you send me an email; jeff@softwareengineeringdaily.com. I would love to hear from you. If you have any questions on the open source project or feedback on the podcast or suggestions for shows, I always would love to hear from you, so send me an email.

With that, let's get to this episode.

[SPONSOR MESSAGE]

[0:03:00.5] JM: The octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like .NET apps, Java apps and more. Ask any developer and they'll tell you that it's never fun pushing code at 5 p.m. on a Friday and then crossing your fingers hoping for the best. We've all been there. We've all done that, and that's where Octopus Deploy comes into the picture.

Octopus Deploy is a friendly deployment automation tool taking over where your build or CI server ends. Use Octopus to promote releases on prem or to the cloud. Octopus integrates with your existing build pipeline, TFS and VSTS, Bamboo, Team City and Jenkins. It integrates with AWS, Azure and on-prem environments. You can reliably and repeatedly deploy your .NET and Java apps and more. If you can package it, Octopus can deploy it.

It's quick and easy to install and you can just go to octopus.com to trial Octopus free for 45 days. That's octopus.com, octopus.com.

[INTERVIEW]

[0:04:31.5] JM: Preethi Kasireddy is a blockchain developer and she joins the show to discuss blockchains and Ethereum specifically. Preethi, welcome back to Software Engineering Daily.

[0:04:43.0] PK: Thank you, Jeff.

[0:04:43.9] JM: You have been studying the Bitcoin and blockchain space for a while. You worked at Coinbase for a while and then you have been writing more recently and you've gone off on your own. I want to get a picture for the basics of Ethereum in this episode, and I think most of the people listening to this understand what Bitcoin is. They've probably heard a little bit about Ethereum.

We know that Bitcion is decentralized money and we've probably heard that Ethereum is decentralized compute. It's a bit virtualized machine that's spread across the world. The question that I think is confusing to some people is we get why you need a currency in Bitcoin, because it's decentralized money. That seems like a currency would be core to that idea. Why do you need a currency in the Ethereum space?

[0:05:35.7] PK: I guess taking a step back, I'm not completely sure I agree with the parallel that Bitcoin is decentralized money and Ethereum is decentralized compute. Fundamentally, they both are under — The underlying technology for both of them is the blockchain. Bitcoin has a blockchain. Ethereum has a blockchain, and a blockchain is just kind of — The most general way to think about it is like this journal or ledger which stores transactions that are happening on the network.

In Bitcoin's case, one application that could sit on that blockchain is decentralized money. Of course, Bitcoin also does enable other types of applications, like smart contracts and so forth. Fundamentally, Ethereum tries to go above and beyond what Bitcoin offers in a few different ways. Ethereum also has a blockchain. Similar thing, it contains — It's like a ledger with a series of blocks. Each block contains a certain number of transactions and so forth.

The underlying difference between Ethereum and Bitcoin is that Ethereum now tries to create, what you said, like a decentralized compute platform essentially by making the language that

runs Ethereum [inaudible 0:06:57.1] complete and has state. It has the ability to manage state overtime. That's how they are kind of different.

The main question you were asking is, "Why does Ethereum need a currency?" Ethereum needs a currency, because, again, underlying both blockchains, there's a consensus mechanism to ensure the security of the transactions. To avoid things like double spend, they both have a consensus mechanism, where miners do something — Miners provide their compute power to solve a cryptographically difficult hashing algorithm and that consensus mechanism is called proof of work. Both Bitcoin, blockchain and Ethereum blockchain use the same consensus mechanism called proof of work. The reason Ethereum has a currency is because it creates new Ethereum every time a miner provides a compute power to verify and validate these blocks. That's how new Ethereum is generated. Similarly, that's how Bitcoin is generated. Yeah, that's one of the main reasons for the currency.

[0:08:09.0] JM: I get it. IN Bitcoin you've got all these financial transactions that are taking place and they get recorded in everybody's copies of the blockchain and overtime the blockchains are verified by the miners who are putting their hard earned compute into verifying those transactions. Those are miners are rewarded with new Bitcoins. In Ethereum, the thing that you're verifying, the set of transactions that you're verifying are programmed. You're verifying that the same programs have been run — That everybody has the same ledger of computation that has existed in the past. Is that correct?

[0:08:58.7] PK: Yeah. I think you're trying to, again, shove Ethereum into this compute model, and I get it, but both still have — It's not programmed. There are still transactions. Ethereum has the concept of transactions as well, but the difference is that — In Bitcoin's case, they follow this model called a UTXO model. For every transaction, it's essentially a set of input UTXOs which are unspent transactions, and then it's a set of outputs that spend that transaction. It's basically a ledger that kind of keeps track of transactions moving from one account to another, to another, and all the unspent ones then become spent, then those become unspent when they go to the next sender and so forth.

With Ethereum, they still have the concept of transactions, except the difference is that these transactions are allowed to interact with code, because Ethereum has this concept — Unlike

Bitcoin which has one type of account, Ethereum has two types of accounts. One is called like an externally owned account, and you can think of this as like a user account. This is an account that you would have, for example, to maintain your own ether balance, any kind of wallet that you own. That would be associated with an externally owned account. These accounts live outside of Ethereum's ecosystem.

Then you have a contract account, and a contract account has code associated with it. A transaction has the ability to execute code on a contract account, and that's the kind of fundamental difference between how transactions work in Ethereum versus Bitcoin.

[0:10:37.6] JM: In a contract account and an external account, you both have — In both situations you have some ether sitting there, and then the external account is more like just like your external wallet or your store of value area, and then the contract account is this repository for — It's like code together with ether so that that code can execute in transactions involving ether. Is that right?

[0:11:07.7] PK: Not necessarily. The contract account doesn't actually have to have any ether and neither does — You can have a zero balance on a contract, right? It's just that when a transaction occurs, it basically — It's like you can think of a contract as this like autonomous object that exists in the Ethereum ecosystem and every time you send a transaction from an externally owned account to a contract, it's like poking that contract and say, "Hey, run this function."

The function can ensure that function might transfer ether to another contract or to another account, but it can do so much more than that. It can store something in the world sate. It can do calculations. It can do for loops. It can do a lot more than just an ether back and forth across accounts.

[0:11:54.0] JM: Okay. Let's just go ahead and dive into the idea of a transaction, because this is a deep subject. Why don't you define that term transaction a little more holistically for us?

[0:12:06.1] PK: Yeah. Again, you can think of a transaction, it's basically this assigned piece of data or assigned data package and it stores the message that needs to be sent from the

externally owned account to either another externally owned account or a contract. This transaction contains a few different things. Main things are like obviously the recipient of the message or contain the signature that actually identifies the sender to verify that the sender is a valid sender. It will contain — If there's an amount of ether that the sender wants to transfer, it will contain that amount value. It also has the ability to contain extra data. Let's say an externally owned account is sending a transaction to a contract and it wants to input some data into that contract, it can also send data along with it.

Two other things that it contains that are very important are something called a start guest value, and this kind of represents — You can think of it as like the gas limit, which is basically like how much is the sender willing to pay to execute this transactions. How many computational stubs are they willing to take and pay to execute this transaction, and they set a limit on that.

The other thing it contains is called like a gas price, which is like a representation of how much the sender is willing to pay per step. This is kind of what composes a transaction. Yeah, again, the transaction can then be sent to either another account or a contract and it gets executed.

[0:13:40.2] JM: Can you give us an example of a contract? Just walk us through the transaction steps.

[0:13:48.5] PK: Yeah, sure. Let me see, it's a good example. Yeah, let's say I have a contract that has just one function and it basically — Let's say it accepts some piece of data and it stores in storage. What happens is I'll send a transaction and it will contain all the fields that I just said, which is the recipient, the signature that identifies that you are the sender. Some ether amount if you want to send it. Some data fields. In this case, data field will be what data you want to store. Then the gas limit value, and then the gas price.

You send that transaction across, and first what happens is we first check if the transaction is well-formed, like it has all of the fields that it needs to have. Then what's done is like we calculate the transaction fee. This is calculated by taking that start gas amount that you've provided times the gas fee, because, again, in the [inaudible 0:14:47.0] gas, you stated how many computational steps you're willing to provide. What is the maximum number of

computational steps you're willing to take? Then the gas price is how much you're willing to pay per step, and so you just calculate what the transaction fee is by multiplying those two.

Then amount you subtract from the sender's account balance, and then once you subtract that balance, you increment something, a value called nonce on the sender's account. The nonce is basically keeping track of how many transactions the sender has actually sent overtime. The reason we do this is to avoid double spend. We can't do like replay attacks, where the sender tries to do the same transaction twice. The nonce keeps track of that.

One those two steps are done, then we initialize gas to the start gas. So far we've spent the start gas times gas price amount, and what happens at that point is then the data is sent to the contract, and to store that data perhaps it costs X-amount. We, again, subtract that amount of gas from the total amount that you're willing to pay and we keep subtracting for each computational step that you'd take from that total gas that you're willing to pay until either the transaction fails or succeeds.

It can fail for various reasons. For example, one reasons it might fail is that it runs out of gas. You didn't provide enough gas for all the computational steps to run on that contract to store that piece of data. Then in that case it will fail. What will happen in the failure case is the gas will still be paid for. You won't get your gas back, but if you sent an ether along with it, that will be refunded to you. Then if it succeeds, if the transaction succeeds, then that gas get paid to the miner and any gas that you didn't spend based on the maximum amount that you've provided will get refunded back to you and then the state will move to the next state based on what you stores in storage.

[SPONSOR MESSAGE]

[0:16:53.8] JM: Every second your cloud servers are running, they are costing you money. Stop paying for idle cloud instances and VMs. Control the cost of your cloud with Park My Cloud. Park My Cloud automatically turns off cloud resources when you don't need them. Whether you're on AWS, Azure or Google Cloud, it's easy to start saving money with Park My Cloud. You sign up for Park My Cloud, you connect to your cloud provider, and Park My Cloud gives you a dashboard of all your resources, including their costs.

From the dashboard, you can automatically schedule when your different cloud instances get turned on or off, saving you 65% or more. Additionally, you can manage databases, auto scaling groups, and you can set up logical groups of servers to turn off during nights and weekends when you don't need them, and you could see how much money you are saving.

Go to parkmycloud.com/sedaily to get $100 in free credit for Park My Cloud for SE Daily listeners. Park My Cloud is used by corporations like McDonalds, Capital One and Fox, and it saves customers tens of thousands of dollars every month. Go to parkmycloud.com/sedaily and cut the cost of your cloud today. That's parkmycloud.com/sedaily.

[INTERVIEW CONTINUED]

[0:18:28.8] JM: Okay. That's a really, really descriptive example of how an Ethereum transaction works, and because there's so much detail there, I think I want to revisit this a third time. I've done some shows about Ethereum, but I have not gone into this level of detail. I want to present like — You gave a really good exemplification, but I want to get at even more concrete example. What I'm thinking is something that's like an actual application where people could use this in. One example I'm thinking of — You can tell me if this is a bad example and we can discuss something else, but one example I'm thinking is let's say I want my Software Engineering Daily podcast contract where basically the user can send some amount of ether and I'm going to send them back an MP3 to a podcast episode or maybe they can also send me a different amount of ether and then they subscribe to the podcast, that means I'm just going to push MP3s to them all the time. I'm going to store their user data so that I'm learning about what episodes they listen to. Would that be an interesting example to explore, or shall we do something more traditional like — I don't know, a mortgage back security or something like that?

[0:19:41.7] PK: That's fine. It will be the same explanation again, so I'm happy to repeat it.

[0:19:45.8] JM: Yeah.

[0:19:46.5] PK: Let's take the example of you're trying to have a contract that builds a subscriber list, let's say. I'm want to keep it simple. I'm not going to do the second part where

you said I send back stuff to the contract, because then it gets into a whole new ballpark, because then you're doing contract to contract calls and that adds an additional layer or complexity.

Let's say it's just one way, right? An externally owned account is trying to say, "Okay, register me for your subscription list. In your contract you're storing some list of addresses that represent who is subscribing to your list. This externally owned account would — I guess I'll put it from an end user perspective how they would send this transaction in. You'd log in to some software, whether it's MyEtherWallet or MetaMask or some kind of wallet that allows you to send transactions to Ethereum contracts. Then you say — Alright, you get the contract address and you say, "Here's who I'm sending it to. Here's how much ether I'm sending with it," if you decide to send any kind of ether with it. Here's some data I send with it. Maybe the data might be like you ask for — I don't know, their age or their birthday or their credit card number. I don't know. Whatever it is. You might ask for some data fields when they subscribe. They want to send some data along with it.

Then they go ahead and they hit send, and then the client's software will say, "Alright, how much gas are you willing to spend on this?" Then the user can state how much gas they're willing to spend. Then at that point they can hit send, and that transaction will be sent to your contract. You contract will then execute that function and it will say, "Alright, store this at —" Then Ethereum has a calculation to determine, "Alright, to store this, this is how much it's going to cost." Ethereum will calculate that and every time you do some kind of computation in the contract, it will calculate the gas and deduct it from the maximum gas limit that the user is willing to spend.

Then it will keep doing that until it finishes. In this case, you're just storing the user's address in some array, let's say, so that you'll store that in your storage. The Ethereum will calculate how much gas you use for that. Deduct it from the gas that the user is willing to spend. Anything that the user didn't spend will be refunded to them into their account, and then you have a list of people who are just — Now, you have an updated list of addresses with the new subscriber. Then the miner will get the gas fee that was charged for doing that computation.

[0:22:14.0] JM: That computation gets logged on everybody's blockchain eventually?

[0:22:22.2] PK: Yeah, basically.

[0:22:23.0] JM: We should go into that. We'll take a step back and say I'm the customer, I've stated my gas amount that I'm willing to spend in order to subscribe to this thing and if the network has enough liquidity, then my gas price is going to — The gas that I'm willing to pay is going to meet the liquidity amount in the network and my transaction is going to go through and the contract is going to execute. I'm going to get back anything that I'm supposed to get back. The contract is going to — I'm working this worse than you could, so I'll stop rambling. The contract is going to execute and then the execution is going to get logged in the blockchain somehow and then it's going to get shared with everybody else's copy of the ledger. Let's just go into that. Explain how that verification happens. How the ledger is getting shared with everybody else.

[0:23:22.9] PK: Yeah, sure. I guess the best way to think about this is — The best way to explain this is to understand how blocks on Ethereum get — How a transaction gets validated is through this block validation algorithm. Meaning every block holds a set number of transactions. Every time you want to validate the state of Ethereum, you have to validate the blocks. To validate the blocks, it's like this entire process where you have to check that the previous block is valid and then you check that the timestamp is valid. You check all these different parameters, and then what you do is basically you take all the transactions that are in that current block and then you apply them and then you check that that once you apply them, the end state of that block matches the end state that you expected.

There's block validation algorithm. Basically, I think the question that you're trying to ask is like where is this contract code executed. How does that actually work? The answer is the process of actually executing that contract code is actually the definition of the state transition function itself, which is part of the block validation algorithm.

If a transaction gets added to some block, then the code that's executed by that transaction will be executed by all the nodes, like from now and into the future. Anyone that tries to validate that block will have to execute that transaction as a result of validating their block.

[0:24:55.9] JM: They have to execute the entire thing. There's no thing where like a preponderance of them executes it, then it gets like rolled into a Merkle tree and then the verification in the future is easier. You literally have to run all of the transactions?

[0:25:12.2] PK: No. The code execution that's run by that transaction will be executed by all the nodes that download and try to validate any block. The purpose of the Merkle tree is that it efficiently stores all state that exist in Ethereum. It's an official way of storing past date and then whenever a state gets updated, the Patricia Merkle tree is what Ethereum uses. It just has to update part of the state that got updated and it references that state, but that doesn't mean that the blocks are not getting validated each time. The state is not necessarily being recreated each time, but the blocks are being validated.

[0:25:48.5] JM: Let's say the — Ethereum has been running for a really long time and I want to join Ethereum, the Ethereum network with a new node. Do I need to run all of the past transactions or can I just take the Merkle tree at TZ0 and that's —

[0:26:07.0] PK: Yeah. There's basically — To run the full chain, if you're a miner, you definitely have to. You're going to want the full node, so you're going to have to start from the [inaudible 0:26:14.1] block. For example, for me to sync the blockchain, when I first got started with Ethereum, it took about a day and a half for me to sync the full node and I have like a brand new one terabyte Mac laptop. Yeah, you have to basically run the entire chain.

Of course, Ethereum now offers like clients and this is where they kind of prune certain parts of the state that you don't need to revalidate over and over and over again and they let you start from basically a simpler state and that way — The lighter clients took me like about less than an hour, for sure, to sync the whole thing, which is not bad at all. Again, that's what I'm saying. You have to revalidate at least some part of the state. There's options to do lighter clients versus full nodes. I think you're confusing the Merkle tree. Merkle tree is more to manage the state. It's not necessarily the validation of that state.

[0:27:10.7] JM: Right, but what I was saying was like if you join the Ethereum world as a new node and everybody has an agreed upon state of the present day, why would you need to rerun

all of the past transactions? Shouldn't there be no difference between the state that you would observe after running all of the —

[0:27:34.0] PK: That's the purpose of — That's why Bitcoin and Ethereum are so secure, right? Every single node is verifying every single transaction.

[0:27:40.8] JM: Right. But if there's a hundred nodes in the network and I want to be node 101, if all —

[0:27:45.8] PK: Yeah, that's the purpose. It's like every node adds an extra layer security to the fact that it's harder and harder and harder to manipulate the chain, because every single node is validating all of the transactions. Every single full node — I'll correct myself. Every single full node is validating every transaction. Of course, again, like I said, there's light clients and so forth. They're not validating every transaction, but every full node is.

[0:28:13.5] JM: Sure. What I'm still confused about is if all the hundred nodes, before I joined the network, have validated everything in the past and they all agree on the state of the Ethereum blockchain and they all agree at the transactions on the transaction chain has occurred up until T0 when I'm joining and they have a Merkle tree that represents the current state of affairs. Why can't I just copy the Merkle tree and then copy all subsequent transactions and then I process all subsequent transactions?

[0:28:46.6] PK: Again, this is what light clients let you do. It is possible, but for a full node, you have to execute all the transactions. Of course, a developer right now, I'm using a light client just because I don't want to have that burden. If I had to download the full chain on Ethereum, I think it's like over 10 gigabytes right now. Whereas like the light client I can only download — It's like two or three — Between two and four gigabytes, so it's much lighter and I don't have to processes every transaction.

[0:29:17.0] JM: Let's go just a little bit deeper into the way that a transaction gets validated. I get that the Merkle tree represents the state right now that everybody agrees upon. Well, not necessarily. If we all have the same Merkle tree, then we all agree on it, but the Merkle tree as a representation of state, and Merkle tree, by the way, for people who don't know, are just this —

It's like a tree of hashes. It's like if you know the data structure or try, it's just an efficient compression mechanism for — It's like how Git repositories are managed. You can look into the past with a Merkle tree, but it's a compressed data structure.

What I'm curious about, and you probably have said this like three time already, but I'm just going to ask you again, because I still don't get it. This transaction occurs and a user sends ether in exchange for a contract executing — The ether that they send is an amount that they're willing to pay, their gas limit, and the gas limit is associated with the amount of liquidity in the network. Then that transaction executes —

[0:30:36.1] PK: Can you explain what you mean by liquidity in the network?

[0:30:38.6] JM: Oh, sorry. Yeah. I'm probably misappropriating terms, but liquidity in the network I mean the gas — Like you state your gas limit and the gas that actually gets used, the ether that actually gets used to pay for your transaction executing across the network is based on how many nodes are active in the network, right? If there's more nodes in the network, then it's going to take less gas to spend — Actually, I could be totally wrong about that.

[0:31:04.5] PK: No, not at all. Gas is, again, simply just — Ethereum has in its white paper, in its yellow paper. It strictly defines how much gas each computation step cost regardless of which node —

[0:31:19.2] JM: Oh! Okay.

[0:31:21.3] PK: Yeah.

[0:31:22.1] JM: Interesting.

[0:31:22.9] PK: Of course, there's a gas price you can set, which determines how much you're willing to pay per step, but that — Yeah. The gas price and the gas limit are different. Gas limit is like how much in total gas are you willing to spend, and then the gas price is basically like per computation step how much — What's the per unit gas?

[0:31:47.4] JM: Okay. Well, then let's go deeper on that gas stuff. You said that the price — So much to unpack here. The gas price is set for each computation step. What are you talking about there? Can you explain that a little bit more?

[0:32:08.0] PK: Yeah, sure. I know this is kind of confusing. Gas price is basically the fee that the spender is willing to pay per computation step. Ethereum has — For example, for — I'm looking at the yellow paper right now. Let's say if someone wants to get the balance in a content, the value of cost is 400 gas and the user determines — The gas price is basically representing the fee that the — For every computational step — I guess I'm not doing it — It's like confusing trying to describe it.

[0:32:55.3] JM: It sounds like there are primitives in the language of Ethereum transactions. There are primitives that have set prices, but I guess I'm not understanding — Okay, so there's perimeters there. Is that right?

[0:33:11.6] PK: Yeah. I guess the fundamental — You can think — For every computation, the fundamental unit is gas, and usually most steps cost like, let's say, one gas. Some more heavier operations will cost more gas, because they're just more computationally expensive. I think that's the best abstract. That's the best way to kind of think about it at a high level. Then, for example, to store something, it costs more gas than to add to two values together, just because storage is much more computationally heavy on the entire state of the network than adding two operations. That's kind of what gas represents, and then the user just determines how much they're willing to spend on gas to execute that transaction.

[0:34:01.7] JM: Why do they have to set that number? If I have a contract and you want to execute that contract, it seems like the contract price would be set based on the operations that I'm running in my contract. Aren't you just paying the price that's set based on that?

[0:34:21.0] PK: Yeah, that's a great question. What a lot of these clients do is they offer the user, "Hey, here's a recommended on a gas spend," and usually it's like 21,000 gas. The reason the user has the ability to set more is because some things just cost more gas. For example, sending a simple transaction, like for example transferring ether from one account to another, as

much of those gas, then creating a contract, for example. You want the ability to increment how much gas — You want to be able to spend more gas on transactions that just require more gas.

Then the second reason is because it's also a protection mechanism. Let's say, because Ethereum is [inaudible 0:35:07.5] incomplete, it allows for loops. There's a possibility that some of these contract calls or some of these function calls can go into like literally infinite loops or there might be people, malicious attackers who are trying to create contracts that just go through like ridiculous numbers of computational steps. In that case — Like as a miner, if you see that some attacker puts like 17 million gas in a transaction, it's pretty easy to spot that, "Hey, this person is probably trying to do something where they're trying to run out of — Basically, to something that either goes into infinite loop or just do an incredible number of computational tasks. It kind of gives that visibility into that.

[SPONSOR MESSAGE]

[0:35:58.8] JM: Who do you use for log management? I want to tell you about Scalyr; the first purpose-built log management tool on the market. Most tools on the market utilize text indexing search, and this is great for indexing a book, for example. But if you want to search logs at scale fast, it breaks down. Scalyr built their own database from scratch and the system is fast. Most of the searches takes less than a second. In fact, 99% of the queries execute in less than a second. That's why companies like OkCupid and Giphy and Career Builder use Scalyr to build their log management systems.

You can try it today free for 90 days if you go to the promo url, which is softwareengineeringdaily.com/scalyr. That's softwareengineeringdaily.com/scalyr. Scalyr was built by one of the founders of Writely, which is the company that became Google Docs, and if you know anything about Google Doc's history, it was quite transformational when the product came out. This was a consumer grade UI product that solved many distributed systems problems and had great scalability, which is why it turned into Google Docs. The founder of Writely is now turning his focus to log management, and it has the consumer grade UI. It has the scalability that you would expect from somebody who built Google Docs, and you can use Scalyr to monitor key metrics. You can use it to trigger alerts. It's got integration with PagerDuty

and it's really easy to use. It's really lightning fast, and you can get a free 90-day trial by signing up at softwareengineeringdaily.com/scalyr, softwareengineeringdaily.com/scalyr.

I really recommend trying it out. I've heard from multiple companies on the show that they use scalyr and it's been a real differentiator for them. Check out Scalyr, and thanks to Scalyr for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:38:29.0] JM: When I make my contract, I'm sometimes saying there is some non-determinism from the — The person who's going to call that contract, it's not like the contract is going to have some fixed price. There's some variability, and that's why you want to set the amount that you're willing to spend.

[0:38:49.6] PK: Yeah. Also, let's say — It can be accidental too, right? Maybe you accidentally started an infinite loop. You want to be able to set the max limit of gas that you're willing to spend to the sender on that transaction so that your infinite loop doesn't spend all your ether. There's a lot of protection mechanism here too for why you want to setup a gas.

[0:39:08.1] JM: If I set a gas limit and the contract tries to execute for more than the gas limit, do I lose the amount of money up to that gas limit if the thing doesn't fully execute?

[0:39:21.8] PK: Yeah. What happens is you do lose a fee. You lose a gas amount, but let's say you were trying to send — Let's say you were trying to send me five ether, but you didn't put enough gas. You'll lose the gas, but you won't lose the ether. The ether will be sent back to you. That part of the state transition will be rolled back, but the fee won't be rolled back, because Ethereum did spend X-amount of computational power to run that transaction until it ran out of gas.

[0:39:48.9] JM: Okay. Interesting. Getting back to our example of I pay to subscribe to something. I call this contract to subscribe to something. I pay some amount of gas. I'm maybe paying ether to get on that subscription list as well. I've sent gas and ether. Am I correct there so far?

[0:40:09.3] PK: Yeah.

[0:40:09.5] JM: Okay. Sent gas and ether along with some arguments that state maybe who I am and how I want to subscribe to this thing, other metadata, and the person who's running the contract, the subscriber list, they are going to execute their contract across my arguments. Is there anything interesting there we should discuss? I guess the execution, that code execution is written in solidity. Is that correct?

[0:40:41.3] PK: It can be written — There's a few different clients or like higher level language. All contract code is written in something called EVM code, which is like Ethereum virtual machine code, but obviously no one wants to write in a staff-based language, and so Ethereum has a few higher level languages.

Solidity is one of them. It is probably the most popular and most well-maintained one, and then it also has others, like Serpent. Serpent is more closer to Python. Solidity is closer to JavaScript in terms of how it looks and feels. They also have something called LLL, and then I think Vitalik is working on a new language called Viper. It seems like Vitalik loves writing new languages.

There's a few different ones, but basically the main one you should know about is Solidity, because that's what most of the industry seems to standardizing on.

[0:41:33.9] JM: Okay. My contract is written in something and it is going to execute. Is the first execution of that contract just taking place on the contract that the node that is hosting that contract?

[0:41:49.6] PK: What do you mean?

[0:41:50.4] JM: Sorry.

[0:41:52.4] PK: No single node necessarily hosts the contract. Again, all the transactions occur — It goes back to this block validation thing that I talked about. A block has an X-number of transactions. Those transactions execute code, and that's how a code gets executed.

[0:42:10.5] JM: Okay. When I am sending my — When I have stated that I want to send this subscription request. I want to subscribe to this newsletter or something and I'm paying for it. Where am I sending it? What node am I sending it to? Is it I'm sending it to any node?

[0:42:28.2] PK: You're not sending it to a node. Again, going back to our content of account, we talked about how we have externally owned accounts and contract accounts. Those contract accounts have code associated with them. That's where the code lives. Every time you want to execute a code, you send a transaction and that transaction gets executed as part of the state transition algorithm. Every time a state needs to transition, it will execute that code.

[0:42:57.5] JM: Does that mean that I have a copy of the smart contract that I'm sending to — If I have a node, then I can just execute that contract basically on my own node and then it propagates to other nodes?

[0:43:14.0] PK: You can think of the world state obviously. Blockchains are replicated databases. Your node won't be any different than any other node in the network, it's not like your contract only exists on your node and it doesn't exist on every other node. Every node is the same, and you can think of the — The world state of every node has the entire state of Ethereum, right? You can think of the entire state as containing. You can think of it as — Again, a state is basically the set of accounts. It's like an address and an account. Again, there's two types of account. There's eternally owned account and the contract accounts. The contract accounts have code with them. Regardless of who created the account, how you send a transaction is you create a transaction. You send it and its broadcast to the network. Then those transactions gets executed by all the nodes.

[0:44:12.8] JM: Okay. The first one is probably going to — It is going to execute on my node, right? Does that transaction — I guess it needs to propagate to all the other nodes regardless of how successful it is, because I'm going to pay some cost to it regardless, and that cost paid has to propagate. Regardless of what happens during that transaction, whether it's successful or failure, the record of that transaction attempt is going to propagate to all of their nodes.

[0:44:45.7] PK: Yeah.

[0:44:46.8] JM: Okay. You've got the contract on your node, the smart contract that you're going to execute against. You're going to send your transaction — You're going to call that smart contract using your internal Ethereum account to pay for it and then I guess — Okay. Then —

[0:45:07.5] PK: It gets broadcast to the network, and basically it gets broadcasted. It's just off the internet. Then every node that's listening for transactions can hear that transaction being broadcast. Yeah, that's how it gets — It gets broadcast to the entire network, just off the internet.

[0:45:25.2] JM: Does it happen instantly or does it happen after the — Does the transaction have a serialized? It executes fully and then it gets broadcast? When does it get broadcast?

[0:45:38.6] PK: It gets broadcast, but that doesn't mean it got verified right away. There's a time difference between how long — Between the time you broadcast your transaction and the time that someone actually validates that transaction. The only time your ether is — Everything is like — The only time the state changes and updates is once your transaction is actually validated by some miner. Yeah.

[0:46:06.6] JM: Yeah. It gets broadcast and then it finishes running and then there's the verification or something. I guess walk me through this in a little more granular detail. I'm trying to understand the series of events in which this transaction gets executed and then it gets — Or it gets initially executed, and then if it's successful or failure, what happens. Then when it gets fully verified by the rest of the network, can you help fill the blanks in there?

[0:46:39.3] PK: Yeah, sure. Again, it gets broadcast and every network can hear that transaction. Miners who are willing to validate that transaction can put it into their block and execute that transaction and validate it. If they are first one to create the block and that block gets confirmed, then that transaction is validated and that transaction is now permanent.

[0:47:04.2] JM: The only — You have to explicitly say you're a miner on the Ethereum network to be one of the people who is going to mining so the other nodes are just like hanging out waiting for the miners to all come to a consensus on the most recent set of transactions.

[0:47:22.3] PK: Yeah. High level, yeah, you can think about it that way. You have to explicitly be a mining node if you want to be a miner.

[0:47:30.9] JM: Let's see. Can you talk more about the process — I think this probably agnostic of whether it's Ethereum or Bitcoin, but can you talk a little bit more about how the miners come to a consensus, because I get that they're all competing to solve some kind of mathematical equation that gets us to verification — Just talk a little bit more about how the miners come to a consensus. Every miner has been broadcast these set of transactions that is trying to say, "Hey, these are transactions that actually occurred on the network." How do they come to that consensus?

[0:48:10.0] PK: Yeah. I can explain to you — A simple example is proof of work in Bitcoin. Proof of work is the consensus algorithm. Essentially, what happens is the miners need to solve a computational — They need to hash the block header such that the value is less than something called the nonce of that block. The nonce is set so that the — The biggest thing to know here is that to solve that hash, to get to that nonce that's less than that amount, is really, really, really difficult. There a very tiny chance that it happens.

You need to run this computation many, many — You need to run this hash like randomly like many, many times before you're lucky enough to hit that number and be considered the valid — Or like the block reward, the person who gets rewarded for running that transaction.

[0:49:06.8] JM: Okay.

[0:49:07.6] PK: The main thing to understand, the other thing to understand also is that how difficult it is to create that hash and verify — You have to basically create that hash. Gets adjusted — At least in Bitcoin's case, it gets adjusted so that blocks only get created every 10 minutes. If blocks are to get confirmed sooner, the difficulty gets adjusted so that blocks gets verified every 10 minutes.

[0:49:37.3] JM: Okay. Let's go back to the contract example. Give us some more complex examples. You said that my original proposition of the users, like subscribing to a podcast type of program where they're getting sent back something. You said that was too complicated. Can

you ratchet up the complexity of the contract example that we've been working with where instead of me just sending my request to the contract and the contract fulfilling that request, there's a little more interactivity?

[0:50:10.5] PK: Okay. We talked about transactions and we said transactions have — They're sent from some externally owned account to either another account or a contract. Let's say, now, you get that subscriber list and now you want to call another contract that you own and it does something else. For a contract to communicate with another contract, it would send a message. Transactions are when externally owned accounts send messages to contracts, and then messages are when contracts communicate with other contracts.

You can think of messages as the same thing as transactions. The only difference is that it's basically like a sub-execution of the original transaction. The difference is that it's all within the Ethereum environment. It's not coming from an externally owned account. It's not coming from the external world. Messages only propagate between contracts.

Again, it's the same thing as a transaction and it would call the method on the other contract do the same thing with the gas. It costs a certain amount of gas to execute it every computational step and so forth. The difference now is that when you sent that original — When that subscriber sent that original transaction to the first contract, they set a gas limit number. That gas limit has to be enough to now also execute the sub-executions to other contracts that he was trying to do. If he tried to call — If the first contract ends up making like four other contract calls or something, that original gas limit should be enough to execute all of them, otherwise wherever — It will fail somewhere along the way.

Yeah, that's only other part to it. You can think of like — It's not any more complex. It's a little bit different in term of the fact that one is a transaction and then when contract communicates with other contracts their messages.

[0:52:09.6] JM: Okay. I want to begin drawing to a close, because we're almost near the end of your time. As you've said before the show, the amount of questions I had was way more voluminous than I expected. I don't know. Maybe we'll do another show about advanced Ethereum concepts.

[0:52:29.5] PK: Yeah, I think like a more interesting topic would be more like higher level applications or so forth, or if you have any questions about that.

[0:52:36.8] JM: Yeah. You have written a couple of articles about blockchain stuff. One was this really comprehensive explanation for Bitcoin and the Ethereum basics, and I used that article to create a lot of the material that we've been discussing. The next article that you wrote was about the blockchain scalability, and I think we'll hopefully do a show in the future about the scalability issues of blockchain and Bitcoin and Ethereum.

Let's just tease at that, because we've described this really beautiful system of Ethereum throughout this show. Why is it not the fantastic solution to everything yet? What are the scalability issues that you can tease at given what we've discussed so far?

[0:53:23.6] PK: Yeah. When we're going through the whole processing and validating transactions, you were confused of why every node needs to process every transaction. I kind of responded with the fact that that's a security feature of Bitcoin and Ethereum, right?

Again, the downfall of that is both Bitcoin and Ethereum suffer from the flaw that every transaction does need to be processed by every full node in the network, which means it's hard to scale, because as a size of the network grows. I told you that Ethereum's chain is like on my laptop was about like 10 gigabytes or something and it's kind of continue to grow. There's only a — The number of computers that can actually manage that kind of chain is just going to get smaller and smaller, and so you start to risk centralization. Only a few nodes can actually start validating these transactions and then it becomes like a tragedy where like only the big mining pools can actually validate instead of like it being a truly decentralized network. There's scaling challenges.

That's kind of why Bitcoin has had this like two year debate in the ecosystem and communities kind of disagreeing about how to solve the scalability issue. They had two different sides to it. One side wanted to just increase the block size itself so that it could fit more transactions per block, so Bitcoin had the one megabyte block limit and they wanted to increase to two megabytes to just fit more transactions per block.

The other side wanted to do something called SegWit, which basically moves the signature part of the transaction to a different part of the transaction so that it — Because the signature part just takes a lot of room, but it's only really needed at one time, which is to validate the transaction, but it's not really needed otherwise. They figured out a solution to basically move that to a different part of the transaction and lower the amount of bytes that each transaction actually eats up. These are kind of scalability solutions that Bitcoin has been coming up with and they are already in the process of kind of implementing those.

Ethereum on the other hand also has the same scalability challenges, and there's various solutions to these things like lightning networks are one that actually started in Ethereum and then now — Sorry, that started in Bitcoin and now also being included into Ethereum. There's something called sharding, which where you can basically — Instead of every node processing every transaction, you actually can shard it so that nodes only process the certain number of transactions that they care about.

There's other various mechanisms, like TrueBit which is trying to do off-chain computations. Yeah, and a few other that are kind of outlined in that host. It's a very interesting problem, obviously, and a very real problem. For blockchain to kind of reach the scale that we've wanted to reach and to really be able to host all the types of applications that we dream of or imagine of, we'll have to definitely solve the scalability challenges. But I'm not concerned that it won't be solved, because there's already all these kind of solutions being researched or being implemented currently and things like even IOTA, which are pretty interesting. I think we should definitely have a conversation about that in another —

[0:56:46.9] JM: Okay, Preethi. It's been great talking about the Ethereum basics and we will talk about more advanced topics in the future.

[0:56:53.1] PK: Awesome.

[END OF INTERVIEW]

[0:56:56.9] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning from each other. Check it out at symphono.com/sedaily. That's symphono.com/sedaily.

Thanks to Symphono for being a sponsor of Software Engineering Daily for almost a year now. Your continued support allows us to deliver this content to the listeners on a regular basis.

[END]