TADGE DRYJA: Last class, I talked about payment channels and lightning network. So today we'll talk a bit more about payment channels. We'll recap a little bit, and we'll talk about some optimizations, one of which is key addition, the other is hash trees. And then we'll talk about cross-chain swaps or cross-chain atomic swaps.

OK, so last time, the sort of main idea of these payment channels is you've got this commitment transaction and each party holds a sort of different transaction that has the same amounts but slightly different scripts. So it's spending from a shared funding transaction. And it's sending out to-- the one that's held by Alice on her hard drive sends to Bob in the clear.

Bob just gets his eight coins with no ifs, ands, or buts. However, when Alice broadcasts this, she needs to wait 100 blocks. She needs to wait a day. And if she reveals this key to Bob, now Bob can use Bob's key in this revocable key, and Bob can get these coins immediately.

And the opposite is for Bob. So when Bob has his transaction, it's got Alice's signature. He doesn't store his own, because he can just compute it quickly. The two coins that go to Alice go to her in the clear. However, the eight coins that Bob gets for himself, he has to wait 100 blocks. And during the 100 blocks, Alice can sign if she has Bob's revocation key. OK, so that's the basic idea of these outputs. And then you construct these.

So when I say, Alice, one bitcoin, Bob, nine bitcoin, it's of the format in these. The person who holds that transaction has a transaction where the other party gets the coins immediately, and their own coins have this lockup period, and can be grabbed by the other party if they revoke them.

So initially, the state is Alice has one bitcoin, Bob has nine. There's two different state 1 transactions, one held by Alice, one held by Bob. They build a new transaction together where Alice has five coins and Bob has five coins, and then they revoke the old one by sharing these secrets. So now, since they both have handed each other these revocation private keys, both of these parties cannot broadcast this. If they do, the other party takes all the money. And then

you can build these indefinitely. OK, so any questions about that? That was covered last time, but it kind of can be complicated. Questions? Good?

OK, so some optimizations we'd want to make-- yes, so basically, either party broadcasts and has to wait. Alice gives Bob the AliceR private key, Bob gives Alice the BobR private key. If they broadcast, the counterpart takes all the funds. So what are some optimizations here? The basic idea of the script is, OK, you can have a signature from KeyA, and you have to wait a day, or two, or five, or whatever. Or you can have a signature from KeyB and a signature from KeyC.

So what are some optimizations here? The revealing part-- any way to make that smaller, more efficient? So one idea is a pre-image instead of a signature. You're just revealing something, you're not actually committing to any data. So when Bob reveals his secret, he could just be, here's a pre-image. And instead of like 70 bytes for a signature, you get down to like 20 bytes for a hash and a pre-image. So how would it work? You'd just say, OK, Alice signs if Alice knows BobR's pre-image. BobR doesn't have to actually sign, she can just know it. That would work. So actually that was the initial idea. I think that's what's written in the paper.

But you can do even smaller. So an even smaller way to do it is combine keys. So in the elliptic curve that we're using in all these groups, all these signature schemes, if you add two keys together, you can get a third key. And it's using the addition operation that's defined on this group. So in the case where you have KeyB and KeyC, and you add them together to get, say, KeyR, what's the private key for KeyR? Anyone?

You've got KeyB-- private key-- KeyC, which has a private key, you add the two public keys up together-- yeah, it's the sum of the private keys. So public KeyB is just private KeyB times the generator point, G. But-- and then C times G equals R times G. And then you can just move the parentheses around. So B plus C, the private keys added together times the generator point is still the same.

So this is a really nice property. Actually, it's sort of the main property that we use for all signatures and all this stuff. What that means is you can say, here, I take your public key, I'm going to come up with my own private key, compute the public key for that, add them together, and say, hey, we've got this new shared public key. And neither of us can sign with it yet because we don't know each other's private keys. But if either of us reveals the private key,

then the other party can sign with it.

So if this is Bob's and this is Alice's, they can both compute little r times G by doing this operation. They both share public keys and compute the shared public key. But they cannot sign, because neither of them knows little r.

And then let's say Alice says, OK, Bob, here's this little c, gives it to Bob. Bob says, OK, now I know a little r. But Alice doesn't. So this is a really nice way to do it, because it saves a lot of space. So instead, the script is now just KeyD-- oh, I should have put R, oops. KeyD or KeyA and time. So instead of having this key and time passes, or these other two keys both sign, you just have a single key that can sign immediately, or another key that can sign after a period of time.

So the actual script used is this. You say OP_IF, the KeyR, OP_ELSE, delay, OP_CHECKSEQUENCEVERIFY, OP_DROP, then the KeyA, then the ENDIF, then the OP_CHECKSIG. So I'll walk through this because it's a little confusing. The OP_IF takes an item off the stack, and it's either true or false. So the first thing you have to do is you have to indicate which of these two paths you're going to use. Either you're going to put KeyR on the stack, or you're going to put this delay number, which is, say, 100 blocks, put an OP_CHECKSEQUENCEVERIFY to verify that it has been 100 blocks since you created the transaction you're spending. And then you have to drop the delay. This was to ensure compatibility, since it's a soft fork.

So old nodes would see this. Old nodes would see this, they would see a no-op, and then they would see the drop to get rid of this whole thing. So the old nodes wouldn't see it at all. Then you have a KeyA put on the stack, and then you end the IF statement, and then you check signatures. So really the IF statement is just determining which key gets pushed onto the stack. And then after the IF statement, there is a CHECKSIG to check the signature from that key.

So for example, if you're using the revocation key, the key that has been combined and revealed to you, you just put a 1 on the stack, and then a signature from R. And then the OP_IF results to true, so it executes this by just pushing the key onto the stack. Then it executes OP_ELSE, skips to OP_ENDIF, and then does OP_CHECKSIG. So basically you say 1, execute OP_IF, KeyR, skip to CHECKSIG. So it's just, you need to sign with this key.

Or you say 0, and signature from A. And then so the OP_IF takes a 0, it skips to OP_ELSE

because it's not true, it pushes delay onto the stack, performs CHECKSEQUENCEVERIFY to make sure that enough time has passed, drops the delay number off the stack, which is a little ugly-- it would be nicer if OP_CHECKSEQUENCEVERIFY did that itself, but it doesn't-- then pushes KeyA onto the stack, ends the IF statement, and checks the signature. So it's a little more complicated than in C, where you just can say, KeyD or KeyA and time. But this is stack-based. And it's not too hard when you get these things to work. So any questions about this setup? Yes.

AUDIENCE: So what's OP_DROP again?

TADGE DRYJA: OP_DROP takes the topmost item on the stack and deletes it, and all the lower items come back up. Yeah, it's just so that, compatibility-wise-- there would be ways to-- since OP_CHECKSEQUENCEVERIFY basically, this used to be a no-op where it did nothing. And then the soft fork was, OK, we're redefining no-op 3, I believe, to OP_CHECKSEQUENCEVERIFY, where it looks at, basically, the age of your inputs, and how many confirmations your inputs have had, or really the specific input you're spending in this case.

Old nodes would just see it as a no-op, and so you could put something like delay, OP_CHECKSEQUENCEVERIFY, and then an OP code around here that says, make sure the top item on the stack is greater than 5. And if this consumed the delay value by taking it off the stack, the old nodes might say, oh, this isn't valid. Even though this is a no-op, the subsequent OP code causes this to fail. Whereas in the new version, it would be like, no, this is good because this consumes that value.

And that would actually make it a hard fork, because then there would be transactions where the old nodes thought it was not good and the new nodes thought, yes, this is OK. So to ensure that this only restricts the validity of scripts, it doesn't pop that off. So you have to manually drop it. And DROP is there from before. So to an old node, it would say delay, no OP_DROP. And it's like, OK, you put something in the stack, you do nothing, and then you take it off the stack-- fine. So that's just a compatibility thing.

If you were going to make it from scratch, you'd probably do it differently. Other questions about this? OK, cool.

So then you reveal keys. You keep revealing things each time you make a new state. And you can do these several times a second. You can do it 10 times a second, something like that. So

you can have millions, billions of different old transactions. The problem is, you do have to remember all of these old private keys that your counterparty has given to you. Because if you forget one, they might broadcast a transaction where you really need to know that private key in order to grab the money immediately.

So for example, you forget-- this was revoked, but you forgot about it, and you're Alice, and then Bob broadcasts this. You really need to know that private key so you can grab Bob's five coins with your combined key. So you need to keep track of all of these little things.

So it's 32 bytes each. It's not great for scalability. So there's actually a couple of different ways you could do this. How can you remember a chain of 32-byte secrets? A merkle tree itself would have a root here, and then a bunch of little hashes that go up to the tree's root. So if these were secrets-- you can't go back down, right? So if you're like, OK, I got secret 0, secret 1, secret 2, secret 3, well, I've got the parents of these two things, but I can't actually remember them any better.

So a real simple way-- say, OK, I've got-- and you can construct it however you want. Obviously if your counterparty is not cooperating and these secret numbers are just completely random, there's nothing you can do. If he just says, hey, I'm going to give you secret 0, secret 1, secret 2, there's no special properties about these things. I can't do anything.

The simplest way-- what's a real simple way to say, OK, I'm going to reveal secrets sequentially, and then you can store fewer of them? It's sort of like a blockchain, right?

[CHUCKLING]

So you could just say, OK, well, I create secret 3. Secret 2 is the hash of secret 3. Secret 1 is the hash of secret 2. And so now I reveal 1. You don't know what 2 is, and so on. So 0 equals hash of 1. So you could just have a linear chain, like this, so 0, 1, 2, 3. If you remember 3, you can compute 2, and 1, and 0. If you remember 2, you can compute 1 and 0, but not 3.

So you can do that. That would actually probably work fine. But we always want to make unnecessary complex optimizations to things. So the problem with this is if you've got a billion of them, and you need to go back to the first one, you need to compute a billion hash operations. That's kind of slow.

So what we do-- or the basic idea, there's multiple implementations, slight variations-- you have a hash tree. So you reveal secrets one at a time, you store only log(n) secrets, and you re-compute any secret you want with something around log(n) operations. I called it Elkrem. I think there's an actual paper that basically is this, called GGM. And that's, I think, written by people at MIT. But I called it Elkrem because it's basically a merkle tree backwards, because the arrows point the other direction. So in this case, you start with a route-- you start with something up here. And you say, OK, if I wanted to send to the left, I append a 0 to the hash value of this root, and I hash to get that one. And if I want to go right, I append a 1 at the end, hash that, and then I go down here.

So what that lets you do is-- there is a sender and a receiver. The sender just computes a route, and stores that. They make a random route up, store it, and then they also keep track of, OK how many secrets have I given? The receiver just receives the first one, number 0-- can't do anything. They receive 1. OK, still can't do anything. They receive 2. Now they can say, OK, well I can throw away 0 and 1, because if I want to compute 0 and 1, I can just store 2. And if I want them to send to the left, append to zero, and hash if I want to send to the right--

So I don't store these, but I know they're still computable for me if I ever want to grab them back. I receive 3, can't do anything. I receive 4, can't do anything. I receive 5, now I can do the same thing with 3 and 4, remove those from the disk, and only store 2 and 5. So if I get a request for 3, I can easily compute it. And then I receive 6, and I can delete those two as well.

So as I go up the tree, sometimes I only have to store one thing, sometimes I have to store three things, because that's the height of the tree. Well, it's basically you have to store at most one on every level, and then sometimes two if there are two leaves.

So basically log(n) storage, and computing hashes is pretty quick, too. So that's a fun way to only have to store a very small amount of data. Any questions about this? OK, so we're going quick.

[INAUDIBLE] quick. OK, so we'll talk about cross-chain swaps for most of the bulk of this day. So there are altcoins. I mostly work on Bitcoin. I think a lot of the research is on Bitcoin. But there's a lot of different coins, too. People saw the idea of Bitcoin and said, hey, let's copy it.

So initially, most of them were just copies. There was a site called coingen.io. So like 2012, there started to be a lot of all coins, like Litecoin, and Dogecoin, and things like that. And most

of them were real small changes, where they took the Bitcoin source code on GitHub, and they changed a couple numbers here and there, and they said, OK, well, it's a new coin.

And then BlueMatt made a site called coingen.io, where you could do it all very easily. For people who weren't programming-savvy, they could just make their own altcoin by menus. There's a website, you say, I want it to be called Tadgecoin, I want there to be 25 coins every three minutes, and I want-- you know, just a bunch of things. You could upload a little picture, and it would make a program for you based on those things. And then I think he sold the site. I think it was the first blockchain technology company, as far as I'm concerned.

OK, so then recent ones are very different. People trade altcoins for bitcoins, and trade altcoins for altcoins, and they use exchanges. I don't know if-- have people here used exchanges to trade altcoins? Some. I actually have. I hadn't until last year. But there was these sort of hard forks from Bitcoin, like Bitcoin Cash, and Bitcoin Gold, and Bitcoin whatever. And so I actually opened an account, and sold them, and got more Bitcoins.

[CHUCKLING]

How do you trade? Well, you use an exchange. And so the current exchange model, pretty much without fail, is give us all your coins, post orders on our site to swap with other users, and then ask for your coins back. So the "give us all your coins" part, it works fine. They give you an address, you send your coins to their address, and it'll appear on their site, generally. But you definitely can send your coins.

The "ask for your coins back" is where the model tends to fail. There's many, many cases where they don't give them back. And there's a lot of reasons for this. So Mt. Gox was one of the most prominent early ones, but it certainly wasn't the first. Probably one of the first was called MyBitcoins.com. And then it was sort of-- I don't know if it was intentional, but the joke, after, was like, well, it's his bitcoins now.

[CHUCKLING]

It was more of a web wallet that had private keys than an actual exchange. But a lot of these exchanges-- they're really easy to set up. You just write some software, and people start sending you coins. And then either there's internal problems where you've got some rogue employee, or the rogue employee is the only employee, and someone just realizes, hey, this

computer in front of me has $20 million worth of stuff on it. I think I'll just unplug it and keep all the money.

Also hackers, right? They're probably the best targets for trying to intrude and hack into a system that you can find. Because there's a whole ton of money on this computer. It's not like data or, oh, I get someone's Facebook pictures or I get someone's credit card numbers. Yeah, that's nice, but bitcoins are sort of money, and you just steal them.

OK, so other problems with the exchange model is that people tend to trust it a lot, and they think, well, this is-- they assume that it's like a bank, and banks are very trustworthy, because in recent memory there haven't really been bank failures, at least in this country. So they sort of carry that trust forward, and bad things happen. So the idea is, wouldn't it be cool if we could have cross-chain swaps, where we have some kind of way where, I've got a bunch of coinA, you've got a bunch of coinB, and we can swap between them without giving custody to some third party or even giving custody to each other.

So you can do this using Hash Time Locked Contracts, just like in the Lighting Network. So you get this coin if and only if I get this other coin. You want them to be atomic. Even if it's very quick, you don't want any period of time in which one party has all the assets or more of the assets.

So the interesting thing about this is the Lightning Network sort of does this already. The Lightning Network is a way to make atomic payments between different channels. And then if you look at these channels and say, well, do the channels have to be on the same coin, do they have to be in the same blockchain? And basically, no, not much difference has to happen for them to be different blockchains.

OK, so this is the HTLC slide from last time. It's similar in that you have these two outputs, the same as in a regular lightning payment channel that's at rest. So for example, Bob holds this, where if he broadcasts this, Alice gets two coins right off the bat, Bob needs to wait 100 blocks, or Alice can know Bob's revocation key, and she can get seven coins or he gets the seven coins.

And then you've got this HTLC, Hash Time Locked Contract, where Alice needs to a pre-image [INAUDIBLE]. Well, yeah, Alice needs to know R, this receipt, or Bob has to wait. But the difference in waiting is this is a relative time lock, so that Bob has to wait 100 blocks after he broadcasts this transaction. Whereas in this case, Bob has to wait until height 500,000,

regardless of when he broadcasts this transaction. So it may be that he can broadcast this transaction and immediately spend by signing here, because that time has passed. So it's seems like a small distinction, but it's actually pretty important for this to be just absolute time.

So that's how an HTLC works. This output's Alice's, no question. This output should be Bob's. And by should, I mean something's gone pretty seriously wrong if Alice takes this money. This only is due to fraud. And fraud is not like in a legal, defined sense. But we sort of can think of, basically, this is always going to be Alice's, this is approximately always going to be Bob's.

This one, we're really not sure. This can fail, and it's not due to any malice. This can-- it should be Alice's. You're trying to pay Alice here. But it may well be the case that Bob takes it back. And that's not due to fraud, that's just due to computers crashing, or networks not working right, or something like that. This can sort of innocently be broadcast and go back to Bob, whereas this can't, really. Any questions about the HTLC basic idea? So Alice needs to know this secret, or Bob has to wait until a certain time.

OK, so the cross-chain swaps is basically the same as a routing payment in Lightning. Before, we said, OK, Alice has a channel with Bob, Bob has a channel with Carol, and now Alice wants to pay Carol without opening a new channel. In the case of lightning, where it's all on the same blockchain, Alice could open a payment channel to Carol and spend directly that way.

In the case where these two are different blockchains, you can't actually do that. You can't say, oh, I'm going to-- there's no defined channel that will link two different blockchains.

The other aspect is, although this can work with three parties, in the case of swaps, you're generally not saying, hey, Bob, I will pay you five Dogecoins if you pay Carol 100 Vertcoins. You're usually saying, if you pay me 100 Vertcoins. So usually it's Alice sending to Bob, who's sending back to Alice. It still works if this is the third party, Carol. But it sort of doesn't make as much sense, because generally you're saying, hey, I'll give you this if you give me that.

So in this case, we have two different blockchains. We've actually reduced the number of parties, where it's just Alice and Bob. But we've got two different blockchains. We've got Dogecoin, which I think is yellow, and then Vertcoin is greenish.

AUDIENCE: [INAUDIBLE]

TADGE DRYJA: Oh, because they don't have SegWit. Well, say Dogecoin upgrades pretty soon.

[CHUCKLING]

So you could, but you'd have to write a lot of code to deal with that. And to us, it's way easier to just merge and--

AUDIENCE: They don't have CHECKSEQUENCEVERIFY on? There's probably ways around it, but yeah--

TADGE DRYJA: Just upgrade-- who's in charge of Doge? I don't know. Tell them to upgrade.

So a lot of these altcoins, there's sort of archaeological aspects to them, where you can see where they branched off of the main Bitcoin codebase. Because a lot of times they don't get updated because no one's really working on them. Despite no one really working on them or looking at them, they can still be worth a billion dollars, as I think Doge was a few months ago. So sometimes they catch up, and they're like, oh, let's take all the code from Bitcoin that they've developed from the last few years and apply it.

Sometimes the opposite happens, where an altcoin will make something cool, and people in Bitcoin will merge it in, but it's generally the opposite direction.

OK, so you've got these two channels that have been set up. And Alice wants to sell her Dogecoin and buy Vertcoin. So you have the same HTLC construction, where Alice-- what's interesting in this case is that it's the same person on both sides. So there's actually some optimizations you can use. But forget those for a second.

Alice computes a random number, R, hashes it to make H. She sends it to herself, which doesn't really make any sense in this case. So you can skip that step. Alice already knows H and already knows R, so-- but Alice doesn't have to be Alice, it could be Carol. So there's optimizations that you can do when they're the same person.

OK, so she constructs an HTLC to Bob. She says, OK, if Bob knows R, he gets this money. Otherwise, Alice gets it after 5 o'clock. Bob says, OK, I don't know R, so I can't get these Dogecoins. Bob then forwards it to Alice, saying, OK, if Alice knows R, she gets the coins, if Bob waits till after 4 o'clock, he gets the coins.

So it's sort of like why make this construction? It actually still makes sense, because even though Bob knows that Alice knows R, Alice will have to reveal R in order to take the coins, right? So it's saying, OK, you need to know R, and you need to not just know R, but you have to show everyone.

Because the actual script says, OK, you need to know the pre-image of this 20-byte hash. And when you actually spend, you have to put that pre-image in the blockchain. You have to put it in your input for your transaction, and then Bob can see it. So the idea is, if Alice closes this channel in Vertcoin, takes the coins by using R, Bob will know R, and Bob will know it's the same R that he can use to take the coins on the Dogecoin network.

So you add these two HTLCs. You then clear them out. So instead of revealing R on the blockchain, Alice can just reveal R directly to Bob, say, I'm clearing out this HTLC because here's R. You know that I can do this, so just give me the coins. You also know R, so you can take the coins yourself if I don't cooperate on that end. But hopefully she does. So then you can get rid of the HTLC. Now it's back down the two outputs.

And then Bob says, OK, I've lost the Vertcoin and I haven't really gotten the Dogecoin yet, but I know I can close the channel and take it if need be. And then Bob goes to Alice, and she already knows R, but look, you know that I know R, let's clear this. And then Alice can just send the coins that way. OK, so any questions about this setup? Yes.

AUDIENCE: Because there's latency, can Alice mess with Bob, basically, take the Vertcoin before Bob has a chance to get the Doge, and basically screw Bob?

TADGE DRYJA: Yeah, well, that's why you have different times here. So yes, that is possible, that you could get to this step, and then Alice goes offline and shuts the channels down, closes the channels. But the idea is that Bob can grab these coins back after 4 PM, whereas Alice has to wait until 5 PM to grab these coins back.

So if Alice closes and tries to wait till right before 4:00 PM and then immediately grab these coins, Bob's actually OK. Because Bob says, OK, well, you grabbed the Vertcoins, but I actually have another hour until I have to-- I have an hour to reveal R where you're not able to get these coins back. So having this incrementing time can help against that.

An hour is probably not enough. The trade-off to having these times is, if you have it very far in the future, with very large gaps, those HTLCs can get stuck there. And if Alice just says, look, I'm leaving this third output in the transaction, that can be annoying. Or if it gets stuck on chain, it might take a long time to get it back. So you want long enough times that you don't worry about these timing attacks, where someone immediately broadcasts, and now you only have an hour, and maybe the blockchain is congested. Versus the annoyance of, oh, it closed,

and now I have to wait until 5:00 PM. So those are the trade-offs there. Any other questions?

OK, so cross-chain swaps-- H can be revealed-- sorry, R-- well, H just goes into either chain, R is then revealed on either chain, so that both parties need to watch both blockchains. This sort of makes sense, in that the channels are on two different blockchains, and both parties have a channel on each. So the receiver doesn't need to be the initiator. So it doesn't have to be Alice on both sides, but in most cases I think it probably will be. Alice has a Vertcoin channel, Alice also has a Dogecoin channel, as does Bob. So they need to be operating on both networks.

There are possibilities where if you had, say, this was Carol, and she sent to Dave using Litecoin-- so if you have Alice, Bob, Carol, Dave, Doge, Vertcoin, Litecoin-- Alice would not have to know about anything but Dogecoin. So you actually only have to worry about the networks that you have channels in, even though there are secrets that could appear in other-- well, no, sorry, that's not true.

So if this is Carol, Alice does have to look at Vertcoin, because the Vertcoin network could show things. So you have to look at one additional chain. So that's one of the issues with cross-chain swaps, where you may be routing through a network that you're not aware of, essentially.

I don't think that's likely in practice. So like it could be, I'll send you five Dogecoins if you send her 20 Litecoins, and then you send back Dogecoins to this other place, but it's possible. Yes.

AUDIENCE: I just want to make sure that I'm understanding this. So when they're both Alice, essentially that scenario is Alice is exchanging Dogecoin for Vertcoin.

TADGE DRYJA: Yes.

AUDIENCE: And then if on the right is Carol, that's saying Alice is going to pay Carol in Dogecoin, even though she's receiving Vertcoin.

TADGE DRYJA: Right. Carol receives Vertcoin, Alice loses Dogecoin. So it's an exchange [INAUDIBLE].

AUDIENCE: So it's like you're paying [INAUDIBLE] never see [INAUDIBLE]

TADGE DRYJA: Yeah. So like you go to some other country-- you go to England, you pay with your credit card, you lose dollars but they get pounds. And there's somehow-- there's an exchange rate in the middle. And so from Bob's perspective, it works the same either way. He's getting more

Dogecoins and losing Vertcoins.

AUDIENCE: So Bob needs to have a lot of-- well, in this case, he only really needs Vertcoins.

TADGE DRYJA: Right. So it's possible that you set up this by initially funding it. So Alice says, I have a ton of Dogecoin. Alice creates this channel, and she has like 100 dogecoins, and Bob has zero. And same with Vertcoin-- Bob could have a lot of them, and say, OK, I have 20 vertcoins and Alice has zero, and then we can make those payments in that direction.

You can pay them-- so actually, in all the software now, I think, there's only single-funded channels. Because having both parties funded at the same time is kind of-- software-wise, a little harder, and it's also, interface-wise, going to be complicated. Because how do you know when to match someone's funding? So usually what you do is you say, I'm Alice, hey Bob, what's your public key? Give me a new public key. OK, I'm going to construct this channel. Here, sign this. And then Bob's like, OK. And Bob has no money at stake when this process happens.

And then Bob also creates a channel with Alice, saying, OK, I'm going to put a bunch of Vertcoin in this channel. And they usually start with all the money on one side so they can then trade. Other questions?

So they have to look at these different channels on different blockchains in order to make sure this works. There's still a lot of open questions here. How do you actually trade? This is really good for trade execution, but what about discovery? So the current exchanges, they're not just "put all your money in here and then ask for it back," there's that middle step where you find all the other parties. So post the orders on the blockchain, and maybe?

There's problems-- I mean, it's not too crazy of an idea, but there's some problems with it. If you just say, hey, I want to sell dogecoins and I want to buy vertcoins, and you sort of put that in an OP_RETURN on the blockchain, well, it's non-binding. You don't have to actually do anything. There can be frontrunning, where the orders can go in the wrong sequence or in time. And it's not scalable, because if everyone's posting orders on the blockchain, that can take up a lot of space, probably more so than actual transactions. I don't know what-- I mean, I'm sure-- maybe.

Total number of orders placed on a order book versus total number of actual order executions, usually there's a lot more orders that then end up getting canceled. Because in

most exchanges that I've seen-- and I believe this is the case in regular New York Stock Exchange kind of things, you can say, hey, I want to sell this at this price, and then a few minutes later you say, yeah, I changed my mind. I don't to sell it at that price anymore. The price has moved, and I've changed my mind. And a lot of times-- yeah.

AUDIENCE: The New York Stock Exchange runs about 100 messages for every executed trade.

TADGE DRYJA: OK, so about 1% of the actual trades get executed. Yeah, that sounds probably about the same as in a lot of Bitcoin exchanges. Usually in the Bitcoin exchanges, they don't charge you to post an order and then take it back. That's free. Whereas the trade actually executing, usually they have some kind of fee.

So posting the orders themselves on the blockchain seems like a cool idea, but you've got, now, this scalability problem of all these transactions. Now it's probably 100 times worse. And frontrunning could also be tricky. Because the blockchain does provide sequence and timing, but it's up to the miners. So if you have a 10-minute window, during that 10-minute window, the miners can basically shuffle things around however they want. And so if the miner is also trading, they can see, oh, I see all these different orders, I will match these orders, and let those happen afterwards. So this is an issue.

So there's a bunch of different models people are looking at. And to be clear here, this is not something that really-- I don't want to say it doesn't exist yet, it sort of does. There are these decentralized exchanges popping up, mostly on the Ethereum network right now. Because then it's not actually cross-chain. You can have cross-asset, because there's all these ERC-20 tokens. But the cross-chain swaps themselves are a bit more complicated to program. So I don't know of any live, not-Mainnet operation like this.

So one model would be you have a central order book and a central counterparty. So the exchange says, OK, we're still the exchange, we still have a website, we still have order-- buys, and sells, and stuff like that. And not only that, but the exchange is one side of every trade. So they say, OK, open a channel with us, put all your vertcoins in, put all your dogecoins in. And then we'll open a different channel with you, and we'll put litecoins in it. And you trade with us.

But then what they can do is they can say, well, we're not actually going to trade unless there's someone else performing a similar, opposite trade. And then they can keep the spread. So they say, OK, we're selling Vertcoin at this rate and buying it at this other, slightly different rate.

So that seems the most feasible. I would venture that that's going to happen first, because it's not too crazy. But it has similar centralization to the current model. But there is less counterparty risk. So if you are worried about an exchange shutting down and running off with all your Dogecoin, this can help. Say, OK, well, I have a channel with my Dogecoin in it and a channel with my Vertcoin in it, and I can sell some Dogecoin, get some Vertcoin from the exchange, and I still have a nice order book. It doesn't solve any kind of frontrunning problem. The exchange can still order things how they want. And the exchange can still shut down. But it's much more difficult for the exchange to somehow steal your coins, which is a problem now.

Another model-- you say, well, we have a central order book, so the exchange still exists and aggregates all the orders of everyone who wants to buy and sell. But there's multiple counterparties that you interact with directly. So the exchange says, OK, I'm E, and I will tell Alice about Bob. Alice says, hey, I want to sell Dogecoin and buy Vertcoin at this rate. And then the exchange tells Alice and Bob, OK, you guys matched. Bob, you want to sell Vertcoin, Alice, you want to buy it. Here, connect to each other and perform this trade.

This is cooler, but there's questions, in that if you're connecting to many counterparties, it's costly. You have to open a new channel, and that is on chain. Also, how do you enforce trade execution? If it's a trade on a current exchange, and you say, OK, I want to buy bitcoins, And you click, OK, it just happened. They are aren't even your bitcoins. You don't really have control of them. So when you say you want to do something, they actually execute it.

In this case, if they don't have a channel with you, they just are observing the trade occur. So they're saying, OK, Alice and Bob, do this. It might not even be possible to enforce. You'd have to have some kind of penalties if Bob says hey, Alice, said she was going to sell, but then didn't, or Bob complains, things like that.

So there's a lot of ideas here. Another is, OK, if you're building channels, each channel that you built has these unchained costs and takes quite a bit of time. So maybe you could have a multi-hop network from different coins. So instead of just Alice, Bob, Alice, you've got Dave and Carol. And as long as I'm connected to someone in this network, I can route payments to them and they can route payments to me via different channels. So I'm connected to this multicoin network kind of mesh. That would be really cool. It's not implemented. It's not even really researched that well yet. So these are sort of ideas.

But that would be really cool. Then you wouldn't need as much liquidity. So one of the issues

with the central counterparty model is the exchange still needs to have a bunch of money. They still need to have a bunch of dogecoins and vertcoins in order to be a counterparty to every trade. And then the question is, well, where did those come from? And if you say, oh, well, it comes from customers, well, now that's back into the custodial model, where they still have coins that aren't really theirs.

You could say, well, they-- and what I'm guessing is going to happen is they're going to say, well, we charge a spread, and we'll accept investors. And so if you want to invest in this exchange, give us a bunch of Dogecoin, give us a bunch of Vertcoin. We will then use it to trade with different parties, and then we'll keep some kind of spread, and then, hey, you got a return on your invested Dogecoin. You get more Dogecoin at the end, and then you can withdraw it.

I think this is still better than the current model, where you leave a whole bunch of coins on the exchange, and you get nothing. Now you can choose-- OK, do I want to give them my coins and take that counterparty risk, but I get some return in exchange for that risk, or do I just want to trade without counterparty risk. So, another problem with that model. But that's probably what will happen.

This doesn't have that issue, in that there is no central counterparty who needs to have a bunch of coins. I'm trading directly with the people who own the coins and want to sell. Yeah.

AUDIENCE: My question is, even if you have this central role and multiple counterparties, wouldn't there still be these trader nodes that are going to go out there and essentially be kind of the same thing? They're going to be well-funded, they're going take fees by doing all these different transactions, and they're going to get bigger and bigger.

TADGE DRYJA: Yes, I guess the only distinction is-- so yes,

AUDIENCE: It just allows it to happen--

TADGE DRYJA: Over the not even long term-- medium term-- these, as a user interface, they'll look the same. Because whether it's the exchange itself saying, OK, I'm aggregating, everyone deposit, I've got a ton of capital, I'm going to be a counterparty to these trades and a market maker, versus, oh, there's no defined market maker, but the people with a lot of money or who are really trying to do this end up in that position.

I guess the only difference is, in this case, the counterparty is also running the order book. And in this case, they may be different. There's an order book server, and then there's this big counterparty who has a ton of money who's doing all the trades.

So does that make much of a difference. I don't know. Yes.

AUDIENCE: In the history of markets, the second example is the one that you see more.

AUDIENCE: Right, yeah, right? New York Stock Exchange, London Stock Exchange, and so on. The first model has the centralized party collecting huge economic rents because they're centralized, which is kind of interesting, where Bitcoin was supposed to be decentralized.

TADGE DRYJA: Yeah, so this probably ends up with a small number of pretty big nodes, mostly because of the efficiency. If you're connecting to 100 different counterparties, and then you never actually trade with any of them, it's very costly to build those channels, and which probably has analogies to, in real life, it's costly to maintain these relationships with all these different companies and stuff. It's much more efficient if we just all meet under this buttonwood tree.

It's a different issue in that, enforcing trade execution, you might not have legal-- you might not know who any of these people are legally. So you have some kind of penalties or some kind of proofs that the trade occurred the correct way, things like that. Yes.

AUDIENCE: How about the enforcing-- [INAUDIBLE] what we'd call it, but the way the trades are set up, do you need to enforce them, or before you enter into them, do you already know what [INAUDIBLE]

TADGE DRYJA: So the example is, before you-- OK, so the example is, Alice goes to a server. So you have an exchange, but this exchange is only the order book. So there's little buy and sell kind of things. And then Alice says, OK, I want to buy VTC. And then-- this is Alice-- Bob says, OK, I want to buy Dogecoin. Then the exchange says, oh, you guys crossed. You want to buy Vertcoin at a rate that he's willing to sell at, or slightly higher, or whatever, so trade.

So the exchange tells both people, hey, you matched, now you have to trade. And then as soon as A sees, oh, you matched, you have to trade, Alice disappears, and says, oh, I didn't mean to. So the exchange only can provide information. They can't actually force Alice and Bob to do the trade is the problem.

AUDIENCE: Right. I guess you can broadcast-- in broadcasting your potential trade, there's information for

somebody to come in at those-- so if it's at a specific price, and you're actually publishing what you would need to actually execute the trade.

TADGE DRYJA: Ah, so that-- yeah, at least in Bitcoin, I don't know how to do that. Because you need to know who your counterparty is in order to either use these channels, or even if it were on chain, you have to share these secret-- it's interactive with your counterparty. So at least in Bitcoin, I don't know a way to say, Alice is posting her buy Vertcoin order in a way that anyone can then go and fill it without additional interactivity from Alice.

I think there's ways to do that in Ethereum, for Ethereum tokens, and there are people who've written software like that. But in the Bitcoin model, there might be a way to do it. We haven't figured it out. So it's sort of-- you put in your orders, they return and say, OK, trade, and now--

AUDIENCE: You have to create the [INAUDIBLE].

TADGE DRYJA: Yeah, now they have to continue to interact. So it's pretty straightforward for Bob or Alice-- Alice could just unplug her computer right at that point. And that introduces timing sort of issues where Alice just puts in all these fake trades. Maybe they're kind of out there. But sometimes the market moves during the time between I put my trade in and the execution comes back. And so I unplug most of them. I ignore most of them, but the ones that are beneficial, I actually go through with.

AUDIENCE: Which isn't too far from the way the markets work.

TADGE DRYJA: Well, yeah.

[CHUCKLING]

AUDIENCE: The difference is, you're saying these are not live hot bids and hot offers, meaning executable. They had another step if they were executable. So this is a new form of spoofing. This spoofing goes on in the US swaps market all the time, because [INAUDIBLE] bids and offers. But in the market where there's executable bids and offers, that's the distinction-- this is not hot. [INAUDIBLE]

TADGE DRYJA: Right. I guess so "hot" in this case would be-- if Alice's offer contains all the information needed by Bob to execute the trade, then it would be sort of hot in that sense, in that Bob doesn't have to interact with Alice anymore to execute. With these construction-- with the

cross-chain swaps, using these channels, you still have to talk to each other. Bob generates-- no, Alice generates the R value, shares the hash with Bob. They're talking to each other.

So we don't know-- if someone figures out a cool way to do it, then great, that will solve this issue. Yes.

AUDIENCE: Is 0x and Radar Relay, are they hot [INAUDIBLE]

TADGE DRYJA: I believe 0x is, from what I've read. I don't know about how [STAMMERING] whatever that is-- Radar Relay works. But 0x, yeah, basically, your order is executable by any party without further input from you. But it also has scaling issues.

So that's a good segue way to the next part, which is having some kind of distributed order book, where there is no central order book, there is no central counterparty, the whole thing is a big mesh network kind of thing. That would be really cool. That seems the furthest out, in terms of, how do you do this. I mean, people are working on it, but it seems hard to make practical in that how do you ensure fairness, right? Who's ordering these things? And you could say, well, the blockchain is ordering these things. But that ends up being that the miners order it.

So if Alice is buy Vertcoin, she's like, OK, I'm buying Vertcoin at a price of 5. And then he says, OK, I'll buy Doge at a price of whatever-- basically, you cross, and you have an order where there's some kind of overlap. The miner can say, OK, well, I'll instead ignore this order, execute my own, and then execute with this guy and take the spread. Because the miner really gets to sequence things however they want. The blockchain does provide ordering, but at the block level. Within a block, it's totally up to the miner. So the miners could swap things around.

But yeah, how do you ensure fairness in terms of timing? How do you enforce trade execution? And probably the biggest for a distributed order book is just the scalability of the orders. You probably don't want to put all the orders on a blockchain, because it just ends up being a ton of data. So you have to have some kind of broadcast network where you've got these orders, they probably should have some way to expire, but expiry is also an issue in that, how do you ensure that-- hey, I'm canceling this order, how do you ensure that the cancellation actually propagates if someone can say, OK, I've got pretty good control of the network, and I can just block these order cancellations, and wait, and then execute. So this is sort of-- there's papers, there's ideas, but it's kind of out there.

OK, so that's basically-- yeah, that was it. That's a little earlier. But yeah, this was basically the stuff for today.

The idea works, right, the cross-chain swaps. But there's still a lot of issues, in terms of, OK, how do you actually get these to work? How do you build a scalable trading platform? And there aren't really any yet. There are some things on Ethereum that work OK. The fees end up being kind of high. And that's also Ethereum, where it's not between different blockchains. It's all running on the Ethereum network, and the Ethereum network allows you to specify, well, this is actually something else, this isn't ether.

You can do that in Bitcoin, but-- so there are protocols in Bitcoin, like Omni, and Mastercoin, and things, where you can sort of create different assets by using OP_RETURN. And basically all of those have moved to Ethereum now, because Ethereum is a much nicer design for those things. And most of the Bitcoin developers are sort of like, great, now all these non-Bitcoin transaction data have left, and Bitcoin is just focused on moving bitcoin around.

Yeah, so the basic idea works, there's still a lot of unsolved questions-- further research required, which is great to hear when you're a researcher.

[CHUCKLING]

And there's people working on this here. I'm working with an MEng student on cross-chain swaps this semester. There's other people working on all sorts of decentralized exchange things. Neha is also very interested in it. So if you want to work with Neha, that's a chance.

And yeah, I think it'll be big, right? Because if you look at these exchanges, there's a lot of profit. So I don't know, a year or two ago, you look at like Poloniex, they were making like $1 million a day. There's so much volume. And they have very few employees. So and they're like-- I think the market is getting sort of more established now, in terms of its going to be larger players and stuff like that. But it was pretty much like Wild Wild West a few years ago, where you're just like, I'm going to open an exchange, and I don't even need a bank account.

The big exchanges that have bank accounts I feel like are less profitable. So Coinbase makes a lot of revenue, but they have a lot of fees-- a lot of costs. They have a ton of employees, they have a ton of lawyers, they have to worry about banks. Whereas if you're like Poloniex-- well, Poloniex is now-- it lasted for like a year or two where they could do that. But if you're just

some random altcoin exchange, you don't need a bank account. You just say, look, I'm an exchange where you can trade Dogecoin and Vertcoin, and that's it. There's no dollars involved.

So those kind of places, they can open up on any server, anywhere in the world. They don't even need to be companies, they can just be someone's server. And the fact that they have custody means they keep getting shut down or just running off with money. Crypsy-- Crypsy was that guy, right?

AUDIENCE: [INAUDIBLE] exchange wallet as like his own personal bank account. Yeah, spent all the money one day.

TADGE DRYJA: Right, so-- and then he said-- I remember there was a blog post where he just sort of said he did that. And it was just like--

AUDIENCE: He was trying to justify it. Like yeah, we're making fees and stuff, so it was OK, we though we could [INAUDIBLE] It's like where the two graphs just diverge, unfortunately, and it went to zero.

TADGE DRYJA: But it was just an amazing blog post, where it's like, wait, you're just admitting to a whole bunch of crimes here, you probably shouldn't have written this. And then he fled to China, but I think they got him. He's in jail now or something.

So there's a lot of you know pretty bad actors in the exchange space. It's not well-run generally. But there's enormous amounts of profit, which, to me, seems like an inefficiency, right? Wait, why are these people making $1 million a day for something that is sort of just running automatically on this computer? That seems like ripe for disruption. And so there's a lot of people interested in, OK, how do we decentralize the exchanges, make them less trust-involved, less risk-involved, and probably make less money than Paloniex, or OK coin, or whatever. But that's sort of better, because an exchange where there's less profitability means more people are going to want to trade on it because there's lower fees.

So there's a lot of research into this. I think it's pretty cool. So next class, on Wednesday, I'll talk about discrete log contracts, which share some of these issues, in that it's a different way to execute a trade, but it doesn't really define how you find people to execute that trade. I personally am more interested in the discrete log contract stuff, probably because I came up with it, but also because I'm not that excited about altcoins in general. I don't know, people like

altcoins, but I don't really feel that, oh, we need to have all these different altcoins that sort of act like money-- I don't know-- which is mostly what cross-chain swaps let you do.

So the discrete log contracts, though, has the same issues with how do you find each other. And it's mostly a scalability thing, right? You can do this sort of OTC, where you talk in a chat room, and you sort of know each other, and you trade that way. And that's the level it is right now. But how to scale it up is another question.