TADGE DRYJA: Today we're going to talk about synchronization-- how all these different nodes on the network come to consensus, how they link up. So it's sort of bringing it all together.

So so far in these lectures we've talked about signatures, mining and blocks, transactions and scripts. And now we're going to put it all together. How does this all actually work? What do all these components come together to do? And how does this make this cool money?

Quick recap on signatures that I think you got the idea from the homeworks, from the lectures-- you have these public and private keys. This key pair where you generate the private key, you distribute the public key. The private key can sign a message. And anyone can verify, given the triple public key, message, signature group.

This is useful for lots of things-- providing identity, ownership, all sorts of things. And it's better than paper signatures. Paper signatures don't really sign the message. They just sort of sign the paper. So if you change some part of a document, the signature still sort of looks OK.

Maybe you can see-- you want the paper to not be tampered. But you can sign a blank piece of paper and then add all this text on afterwards. You can't do that with these systems. So signatures are really cool. And they are sort of a necessary thing for this network to work.

So mining and blocks-- I think you sort of got the idea with the Lamport signatures. If you've looked at the current problem set, it makes a lot of sense. You change it nonce, hash a bunch of times while you change a nonce a bunch of times. Try to get a low output.

So you get the idea of mining. You're proving work by repeatedly going through different nonces, trying to find a certain hash output that there's no shortcut to. You just have the guess and check.

And now if you include the previous data as part of your input to that hash function, you can make a chain of work. And that's what we call a blockchain I guess. Any questions on this so far? It's the basic idea from two lectures ago. Cool.

And a recap of what Neha said yesterday-- there's transactions and scripts. I'm not going to go into scripts too much yet, because in practice, 99% of the scripts are just checking a public key signature. You can do all sorts of other crazy things with the scripts. But almost nobody does yet.

Basically they say, OK, I'm sending to a public key hash. When I spend from it, I reveal the public key, check that the hash matches, check a signature. And transactions have inputs and outputs. We went over this yesterday.

Just a sort of real-world numbers, this is how big things are, "ish," so a transaction ID and index to point to a previous transaction. The transaction ID is 32 bytes. The index is encoded as a four byte, so 32-bit integer, which is kind of overkill, because I think the most that there's ever-- the most outputs there's ever been is like 1,000 or something.

So you really don't need it to be able to go up to 4 billion. But that's how it is.

The signature ends up being about 100 bytes because you have to provide the public key, which is either 33 or 65. And then the signature itself is encoded to something like 70 bytes. Both of those things could be much more efficient. But they're not yet.

And the output is actually a lot smaller. An output, the script-- the main thing is your public key hash, which in Bitcoin is 20 bytes. And then you have those other little opcodes, which it adds a few bytes, and then your amount, which is-- so all amounts in Bitcoin are encoded as 8-byte, signed 64-bit integers. So you can have pretty high precision.

And that's also overkill, since all the bitcoins ever mined, put together-- if you added them all up, it's nowhere near 64 bits to the 64. It's like 2 to the 40 something. So it's also kind of overkill but yeah, whatever. So this ends up being for a one input, one output transaction, less than 200 bytes.

So that's a message, pretty small. You can broadcast it around the network. Inputs point to old outputs, have signatures. Outputs have scripts and coin amounts.

So what do we do with all these things? What is the mining process? So in the homework, you're mining your name. You connect to the server, figure out with the last block was, put your name on, put a nonce, and continue to mine. That's not super useful, unless you want to prove that you're-- hey, this is me.

In Bitcoin, the basic idea is users are making these transactions. Transactions are moving coins from one place to another, from one key to another. They make the transactions, they sign them, and they broadcast. I'll get in to what "broadcast" means.

So in the current problem set, there's one server, which is not really a robust distributed system, as people may have seen yesterday from about 1:30 to 3:00 PM, when the whole thing went down. In Bitcoin, it's completely peer-to-peer. Every node is the same.

They're all listening for people connecting in. They're all connecting out to other nodes. And they broadcast. So if someone sends you a message, you'll pass it on to all the other people. So it's called a gossip network.

In practice, it works OK. It's fairly heavy load on some network traffic, but you can make transactions, sign them, broadcast them. And then someone, the miner, takes all these recent transactions that they've seen, and puts them into a block, and then does some work. So those transactions are now confirmed, and people can build the next block.

So the only difference from the current problem set is instead of putting your name in, you put-- you can put your name-- but you also put all these messages that you've seen recently. And so you commit to them that way.

You could do it by just sticking them all in, but instead, there's a bit more advanced way to do it. You use what's called a block header. So yeah, the block header itself is the message. So similar to the problem set, it's the block header that you need to hash to get a low output value, not the block itself, which is kind of interesting.

And the headers have a hash of all the transactions in the blocks. So you don't just put all the transactions into one big megabyte data structure, hash the whole thing, and then try to get a low output. You actually do some intermediate steps first.

And what's interesting is it's actually just the headers that make a chain, not the blocks themselves. So instead of blockchain, you could call it a headerchain.

So I'll talk about headers. The headers are 80 bytes. And they're actually quite similar to the blocks in problem set 2. So the main three components are you've got the previous hash, the Merkle root, and the nonce.

And so this is like in the problem set. You start with the previous hash. Then you have data that you're actually committing to. And then you have some data that doesn't have any actual meaning, just to get the work done.

So I can-- to reference, if you look through the work people are doing right now, this is all public, the current problem set. There's a previous hash, which is basically the hash of the line above it.

And then there's some data you're committing to, in this case people's user names, and then some non-meaningful data here with just random numbers and stuff it looks like-- so very similar. Any questions about this idea? Cool.

So we use a Merkle root, which I think I talked about last week Monday, instead of just concatenating all the transactions and hashing them in together. You could do that. That actually would work. It wouldn't make a huge difference. But this is a little nicer if you want to prove that a transaction was in this block, without giving the whole block.

So the idea is I have these TXIDs. And a TXID, transaction ID, is just a hash of the transaction. Stick all the components of the transaction into bytes, hash that, and you've got what's called a TXID. And that's how you refer to transactions.

You hash these two together to get this intermediate point. Do the same thing up to the root. And so you can't change any of these little transaction IDs without changing the Merkle root. So it commits to all the transactions just the same way it would if you just concatenated them all together and hashed that.

So it really-- we'll go in to, later, why this is useful. But in many cases, it really doesn't help too much. Any getting questions about Merkle, Merkle tree? Good.

So the actual Bitcoin headers, which many things use, has a couple of fields. Some of them are actually not very useful. But the main two are previous hash, Merkle root, nonce. And then there's-- I'll talk about the other things.

So there's also a version field right at the beginning. It's 4 bytes. It indicates block version. It's not clear what that's going to be used for in the future. It used to be used for sort of signaling protocol changes. I'm not sure that's going to be the case going forward, because it didn't really work very well for that.

So right now I think they all start like 02 and then a bunch of zeros. And that's the current version, whatever that means. And if you mine something with a different version, everyone will accept it. But there'll be like these warnings that show up in your Bitcoin log files that say, warning, unknown version detected.

The idea is maybe, well, if the inversion increases or changes, maybe there's some new rules in this system, or new opcodes, or new something going on. And you're not aware of it, so you might need to upgrade your software. That was the idea anyway.

In practice, what happens is-- you'll see in your logs all the time, that like, unknown version detected. And it's just someone just set random numbers in the version field. And it doesn't seem to mean anything, so not super useful.

Previous hash, just like in the problem set, it's the hash of the previous block, 32 bytes. Merkle root, as described a few slides before-- hash of all the transactions in the block. Time, actually kind of complex-- I'm not going to go into the whole thing right here. So far we haven't really talked about time. Does anyone know why we'd want time in these headers? Yeah.

AUDIENCE: You had mentioned earlier that you don't accept blocks between a certain interval if they were too late.

TADGE DRYJA: Right. So it makes sense intuitively that like if someone says, hey, I mined this in 1987. It's like well, that seems crazy. Or if someone says, here's a block. It came out in 2046. Like, this doesn't make any sense.

So intuitively, yeah, you shouldn't accept things that have some crazy date that's clearly wrong. But why? Why do we need time at all? Yeah.

AUDIENCE: If you want to lock the transaction until a certain time.

TADGE DRYJA: Yeah, you could say, here's a transaction, and I don't want it to be valid before August 1. And so then, you could say, if it goes into a block, and that block has a timestamp before August 1, consider the block invalid. You could. You can also do timestamping based just on block height. But what's the main-- does anyone know the main reason to have height here? Yeah.

AUDIENCE: Is it because if they're a competing transaction? And then you would pick one or the other.

TADGE DRYJA: So the competing transactions, when they get in, they sort of get into a block. And that sort of solves that competition. So you have two transactions that both are mutually exclusive.

Well, if they're both in the same Merkle root and both in the same block, then that block is considered invalid, because it's like, hey, you've given me a block. It's got two things that can't both exist here. So throw the block away.

If you find two blocks that both seem to have-- that sort of collide. They're both pointing to the-- they've both got the same previous hash. So they're both in the same height, we call, of the chain.

You could say, oh, well, whichever one came out first. I'll look at the timestamp and say, OK, the block that came out first will be the valid one. But the problem is this is claimed block creation. You can put whatever 4 bytes you want in there. And so you can always say, oh, I just wanted it exactly 1 second after the previous block. It just took me a while to broadcast it.

So you can't really trust the timestamp to see which came first. If you could, you wouldn't need all this crazy mining stuff. And transactions themselves could just have a timestamp. And you wouldn't need this whole structure.

So the fundamental reason you're mining is we can't trust people to say when they did something. You can always say, no, this transaction came first. No, this came first. So the real reason for this blockchain is, OK, we know which came before what.

AUDIENCE: In practice, which one happens? Do people just lie and say it happened a second later? Or is it [INAUDIBLE]?

TADGE DRYJA: Oh, in practice the timestamps are pretty unreliable. They can be off by minutes. It can be before the previous block's time. And that's OK.

It seems intuitively like, well, that should just be a rule. And it probably-- it would have been cool if it was a rule and made things simpler from the beginning. But if you're pointing to a previous block, I'm building on top of it. And the previous block came out at 10:15. And I set my timestamp to 10:12, 3 minutes prior to the previous block.

Logically, that's impossible. I'm referencing something, and I'm saying I'm coming before it. But the software says that's OK.

AUDIENCE: So if we're creating a version, would it be useful to just get rid of version and time, like if we're creating a new blockchain?

TADGE DRYJA: So version, maybe you could get rid of, or you could put it somewhere in the Merkle root or something. Time actually does have a really useful purpose. Does anyone, maybe if you know?

AUDIENCE: I don't know, but does it play into the difficulty of the mine? Does it?

TADGE DRYJA: Yeah, so the main reason for time here is to adjust the difficulty. And that happens every 2,016 blocks. You just look at, OK, how long did this 2,016-block period take according to these timestamps? And if it took two weeks, OK, we're good. The difficulty doesn't have to change.

If it took three weeks, that means the blocks were coming out very slowly. And we need to reduce the difficulty. If the 2,016 blocks came out in one week, that means, wow, people were mining really fast. And so we need to increase the difficulty. So there's this negative feedback mechanism based on this time.

And it can be tweaked. It's not accurate. You can have things coming in the wrong order. The general rule of thumb-- the rule in the software is about two hours. If you see something that's two hours off from what your internal clock says, you will reject it.

But that's a huge gap. Most network systems, everyone's got their clocks to the same second at least, or millisecond. Two hours is like kind of enormous gaps. But the system works OK, because you've got these really long-term difficulty adjustments that only happen every 2,016 blocks, which in practice is something like two weeks.

So if someone gets something a few minutes off, it doesn't really affect things too much. And it's really only used for, OK, look at the last 2,016 blocks, two weeks-ish of work, of all these blocks, and see how fast we need to make things.

So that ties into the next field, which is difficulty. It's in a sort of weird floating pointlike format with a mantissa and exponent, which is totally custom. And you kind of have to write your own code to deal with it. But it basically says, OK, what does the number have to-- what does the hash have to be below?

It's not just number of bits. So in the problem set, I said 33 bits of work. So that's fairly easy to detect, because you just look for 33 0-bits in the front. In Bitcoin, it's not just number of bits. It's actually a number that it must be below.

If it were just number of bits, the problem then is your adjustments are fairly coarse, because you can only adjust by a factor of 2. You can double your difficulty or half it. But with this, you can have much smaller difficulty adjustments of like a fraction of a percent.

Yeah, this field is pretty much useless since you can calculate it from the time fields of the previous blocks. So you could just have it be implied. But it's in there. And you can just whatever. It's an extra 4 bytes. I don't think you actually-- like, you don't have to store it on a disk if you want, because you can just figure it out from the other things.

AUDIENCE: Wouldn't you need it for [INAUDIBLE]?

TADGE DRYJA: No, because you can figure out what difficulty is just from the headers. I mean, it's in there. I guess it's nice if you just want to validate whether a single header has enough work. But it's like, how much work does it claim it needs? And then you can validate it. But I don't know. It's in there. It doesn't-- you could take it out and reorganize the code a little if you wanted to optimize it. But that would change so much that no one bothers.

AUDIENCE: So when we talk about the adjusting difficulties and even just showing the problem or proof of work, who [INAUDIBLE] for the problems that will go to the central server?

TADGE DRYJA: So in this, it's just everyone broadcasts their blocks. So if you've received a block or if you found a block yourself, you just send it to all your peers that you're connected to. And so there's no like, oh, this is the canonical block.

There can be competing blocks where you have two at the same time and just stochastically, one of them will pull ahead, because, well, randomly. So you can have conflicting things. Yeah, and then the adjustments-- also, everyone computes the adjustments.

And this is an actually very quick computation, because you're just looking at-- you're not even looking at 2,016 timestamps. You're basically just saying, OK, if height-- so height is just what block number it is. So if you're-- right now, it's about 500 million. No, sorry, 500,000.

So you basically in the code just say, well, if height modulo 2,016 is equal to 0, check height minus 2,016's block. Compare the two timestamps. Subtract them. Get a duration. And then compare that duration to two weeks. And then change the difficulty proportionally.

So it's actually, like, super quick for everyone to compute the new difficulty. And they only do it once every two weeks. And it, yeah, it's pretty straightforward.

[illegible]

There are weird attacks and stuff. And it's kind of some weird off by 1 errors, where you're-- I don't remember. Like, it's kind of confusing. It's also confusing because the test network, which I haven't gone into but will use probably in two weeks. There's a Bitcoin test network, which operates pretty much exactly the same as Bitcoin, except everyone agrees that the coins are not worth any money.

What's interesting is it's actually called testnet3. The first two test networks have the same setup. However, the agreement that they were not worth any money broke down. So at testnet1, someone said, hey, I'll pay you a bitcoin for a million testnet coins. And once people saw this happening, they said, oh, well, you just ruined testnet. Now they're worth money. So we'll go to testnet2.

It happened again. Testnet3 has had some staying power. I think people realized that if they try to buy testnet3 coins, everyone's going to leave and go to testnet4. So it's kind of fun. I'd actually be OK with testnet3 coins being worth money, because I have many, many thousands of them.

But yeah, so one difference, though, between the test networks and the real network is the difficulty adjustments. So I think in the first test network, it just worked exactly like Bitcoin.

But one of the problems was people would mine, and the difficulty would increase. And then people would stop mining, say, oh, I'm going to test out my mining software. I'll mine a couple thousand blocks. Maybe it only takes me a day or two to do so, because I have a very fast computer compared to the rest of the network. And then I say, OK, well, it works, cool. I'm going to go to the real network now. And I leave the test network.

And now the difficulty increased, because let's say 2,000 or 4,000 blocks came out. And they came out very quickly, so the difficulty went up. And then all the mining power left. And so now blocks aren't coming out.

And since the adjustment can be up or down but happens based on number of blocks, not based on time, if you have a very high difficulty and the very low hash rate relative to that difficulty, it can take weeks or months or years for the difficulty to reduce. So testnet3 put in this sort of difficulty nerfing code, which is probably wrong and not what they intended. And it has this thing where like if 20 minutes have gone by, the difficulty lowers. And it's kind of ugly. So that's the main place I've dealt with this field.

One other rule with the restriction-- the difficulty can go up by at most a factor of 4 and drop by at most a factor of 4. So if you mine 2,016 blocks in one day, the difficulty goes up 4x but does not go up 14x or whatever the implied would. Any-- go ahead.

AUDIENCE: So the difficulty is definitely constant for two weeks then?

TADGE DRYJA: Yeah. Well, sorry, not two weeks-- 2,016 blocks, which is generally around two weeks, but yeah.

AUDIENCE: So it unblocks and blocks, within literally an almost two-week period, that difficulty would be the same.

TADGE DRYJA: Yeah, so if you actually look at the headers, this is just the constant. It just is always the same. So it's kind of a silly field to be in there. You never need it, and it's always the same. Any other questions about-- yeah?

AUDIENCE: How many transactions are usually [INAUDIBLE]?

TADGE DRYJA: Oh, I'll get to that-- right now, a couple thousand, 4,000-ish. We'll get to that I think. But yeah, in the Merkle root-- so the height of the Merkle root's like 12-ish. And it goes out to maybe 4,000 transactions, sometimes more, sometimes very few.

You'll find empty blocks that just have one transaction in them. And that transaction ID just becomes the Merkle root, because a height-- it's like a height-zero Merkle tree, but yeah, something like that.

And then last-- pretty easy-- there's a nonce, 4 byte. Anything you want goes in there. You can think of it as a you went 32, there's no meaning to it. So does anyone see a problem with this nonce field? Yeah.

AUDIENCE: It's too small.

TADGE DRYJA: Yeah, it's too small. 4 bytes-- even in the homework, people are using 12 something bytes for a nonce. With only 4 bytes of nonce, you can go through 2 to the 32 possibilities, which is not enough to mine in almost all cases, because you're going to need to go through 2 to the 70 possibilities to find a block.

So what are some ideas for how do you deal with this problem? Like, it would be nice if it was just 8 bytes. That'd make things simpler. But the system is what it is. It's very hard to change. How, as a miner, would you work around this issue?

AUDIENCE: Adjust the version and time.

TADGE DRYJA: Yeah, so you can adjust version. So that may be why sometimes weird version numbers come up. Time is a good one too, since time-- so yeah, adjust time and also Merkle root.

So time, if you're off by a few seconds, nobody cares. So use the low bits of this time field as part of your nonce. It's kind of in the wrong place, but you can make chips to sort of fiddle-- twiddle these bits as well.

What's also nice is that every second you can sort of-- you can do it the wrong way. And you can say, oh, I'm just going to take the least significant 4 bits of my time field and just use them as nonce space randomly.

What's nice is that the actual time progresses by one bit every second. So as long as your chip has enough space-- so you're like, OK, I've got 2 bit to 32 here, another 4 bits here, so I'm at 2 to the 36. If you're chip only goes through 2 to the 36 hashes every second, you're good because the actual time progresses.

The other way you can do it is modify the Merkle root. And you can do that-- so can you think of ways to modify that without breaking things?

AUDIENCE: Add or drop the transaction.

TADGE DRYJA: Yeah, you could add or you could drop a transaction. So you say, OK, I have all these transactions. I'm going to drop one. That's got some disadvantages, because it may pay fees to the miner.

AUDIENCE: Changing the order?

TADGE DRYJA: Yep, you can swap them. You can just say, OK, well, these two are independent. I'm going to swap them. This will change, which will change that. So you can swap transactions around.

You can also edit what's called the Coinbase, which I think is in like one more slide. So yeah, so there's a bunch of ways that you can change things. And so this is really where you're going to have all the variation. You have 32-bit bytes here. And just even if it was just swapping, and if you have 1,000 transactions, swap in whatever order you want-- there's enough sort of entropy there that you'll be able to find it.

So what's interesting is that the nonce is there. And it's important, because that's where sort of the high-speed mining occurs. But most mining chips will also have circuitry to modify this, because they're operating so quickly that they will exhaust the 4-byte nonce space in a fraction of a second.

And so they'll have to swap two transactions, recalculate a Merkle root, which involves a few dozen hashes, and then go back here. So it actually doesn't hurt their efficiency too much, because, OK, I just did 4 billion hash operations. And then I need to do a few dozen more to get to the next 4 billion. So it doesn't hurt things too much.

Also, in Bitcoin-- a sort of weird quirk-- it's called SHA-256d. They do SHA-256. And then from the output, they do a SHA-256 again, not sure why. I think in one person's first problem set, they inadvertently were doing the same thing and was like, yeah, that works. Satoshi, whoever he or she was, or they, just put that in there.

Any questions about this header in this format and anything about it? It's pretty compact. It's 80 bytes.

AUDIENCE: [INAUDIBLE] including the Merkle root. So if you end up mining something, you don't have to put anything there. So the only incentive is the transaction fees?

TADGE DRYJA: In the block reward, which I'll get to in a second, but yeah. That's a good question. Can you put nothing in as a Merkle? I don't think so. I'm pretty sure you need one transaction.

AUDIENCE: Oh, that's right. I mean, you need the base for the-- because even if just did it, you need a [INAUDIBLE] transaction.

TADGE DRYJA: You can do that. And there's many blocks. So in the first year or so in 2009, almost all the blocks are empty and only have one transaction, because no one was using it. But people were mining. So it was very similar to the problem set, where everyone's just mining, and they're not actually using the system.

AUDIENCE: Bu then right now--

TADGE DRYJA: Right now there's tons.

AUDIENCE: People aren't doing that just because there's transaction fees.

TADGE DRYJA: Right now--

AUDIENCE: It's their only incentive? People just decided--

TADGE DRYJA: No, you still get more bitcoins. So you still get a reward. The reason you'll see empty blocks now is a little tricky. We're not sure, because who knows? But it's probably because of blind mining, where you receive a block, and you haven't actually looked through the contents of the block yet. But you want to mine the next block. But you're not sure what transactions to put in.

You see this 80-byte header, and you're like, oh, someone find a block. But you only have the header, and you want to build on top of it. You have a bunch of transactions you'd like to put in, but they may have already been put into the previous one. And so you're like, well, I have no idea what's in the previous one. I'm just going to mine a block with nothing in it, that way I'm sure that I'm not going to conflict with my previous one.

You'll often see a block with only one transaction very soon after its predecessor block.

AUDIENCE: So like every once in a while, it would check. So you're saying if it happened and went to a block really quickly, there's just more likely to be more transactions.

TADGE DRYJA: Right, right, because a miner might-- it's actually an optimal strategy for a miner to say, look, first thing I'm going do, download the 80-byte header. Figure out if it's got a valid proof of work. If it does, I'm going to just assume it's valid for the next few seconds, because it probably is.

And then I'm going to try to mine a block on top of that. Reference this previous block. Mine a block on top. The thing is, I have no idea what's actually in the block. I have no idea what contributes to this Merkle root, because I haven't even downloaded that data yet. But I can build on top of it, just from the header.

But I can't include transactions, because I have no idea what transactions are in here, so they might conflict. So I'll just mine sort of blind for a second or two. And then download all the transactions, validate it. And now I can include my transactions that haven't been included.

And so that happens. It can be an OK strategy. Sometimes it can lead to you mining an invalid block. If someone produces an invalid block, and you just see, oh, well, it's got proof of work, you grab that header, start mining on top of it.

No matter what you mine, it's going to be invalid, because it's pointing to an invalid block. That happened 2015 or 2016. The summer of 2015 it happened. And it was like quite extensive. It was like seven or eight blocks in a row that were all invalid, because none of the miners were actually verifying anything. They were just downloading the headers from each other and being like, yeah, I mean, you did the work. So they were just assuming everyone else was verifying the Merkle roots.

And yeah, so then it ended-- so they lost 25 times 8, however many bitcoins. They lost hundreds of bitcoins, which at the time was still worth quite a bit-- and now it's worth millions of dollars-- just because they weren't actually checking things.

AUDIENCE: If I mine a block and it's empty, do I decrease my chances of being mined afterwards?

TADGE DRYJA: No, actually, I would say you'd, sort of game theoretically, you would increase it, because you're not depleting the mempool. I need to talk about the actual mining coinbase and mempool and stuff. But yeah, it's a tricky question. And I'll try to get back to it. If I don't, bug me again.

TX-- so in this Merkle root, you've got all these transactions. They have a specified order. Transaction 0 is the coinbase transaction. And it's special. It generates new coins, and it takes fees from all the other transactions in the block.

I think Neha mentioned this yesterday, where if you have a difference between the input amounts and output amounts, that's implicitly a fee. So if your input-- so here, Neha's thing, you're spending 20 coins. You've got 5, 10, 4. Well, there's only 19 coins in the output, so there's an implicit fee of 1 coin.

That one coin can go to transactions 0. So transaction 0 has essentially no input. Transaction 0's input field is just empty, anything you want to put, any bytes you want to put in there.

Its output field generates new coins. So that's currently 12 and 1/2 coins. So currently, if you mine a block with only TX0, you get 12 and 1/2 coins.

If you mine a block with thousands of transactions, you get the 12 and 1/2 coins plus the difference between the input and outputs of all the other transactions in the block, which can be even more than 12 and-- which can recently, like in January or December, there were quite a few blocks where they're getting 25, 26, 27 coins, because the total fees for the entire block were more than 12 and 1/2 coins, which is hundreds of thousands of dollars now. So it's kind of cool.

The fees have since decreased. Fees are highly variable. And we'll talk about fees in a few more lectures. And it's kind of a mess, but it's an evolving area in this whole network thing.

So you've got your coinbase transaction. That's important. That's why people are doing this stuff, because they want money. All the other transactions can be shuffled around.

However, they can only spend outputs from previous transactions. And previous means they have an index within the block that's lower. So for example, you have transaction B spends an output of transaction A. Transaction A must come first in the block ordering.

This makes it so that you can go through in a linear fashion and validate every transaction in order. Otherwise, you'd go through, see, OK, transaction 0, I don't have to validate that. That's coinbase-- transaction 1, transaction 2. And then you see transaction 3. It appears to be spending something you've never heard of.

So that would at first-- that would appear to be invalid. And then maybe you go through another few transactions, say, oh, this creates the output that this thing I just saw before spends. It's sort of out of order. It makes things very complicated to validate. And so this rule ensures that if you go through and just check every transaction in order, it'll all make sense. Yes.

AUDIENCE: Is there any benefit moving earlier or later?

TADGE DRYJA: In the block?

AUDIENCE: Yeah.

TADGE DRYJA: No, I don't think so. I mean, I can't think of one. Yeah, it's just sort of random. A lot of times they'll organize it by fee rate. Or by default, they'll just organize it by when they saw them first. So it's pretty arbitrary.

AUDIENCE: Does that mean that you have to wait until the transaction that the major output has been mined, in order spend that again?

TADGE DRYJA: No, although if they did, that would make the software a lot simpler and easier to deal with. But that is not how it works.

So for example-- I'll draw it-- you can have a block where there's-- let's do it this way. So you have TX0. There's coinbase transaction, transaction 1, transaction 2, transaction 3. And transaction 3 may be spending something that was generated in transaction 1. That can happen.

So if you make transaction 1 broadcast it. it's unconfirmed. You then make transaction 3 broadcast it, spends transaction 1. The miner can put those in the blocks. They must put it in order. So if this happens, you can't switch them in order. But that's considered OK.

It makes parallel-- it makes multi-core validation more annoying. If you said, no, you can only use outputs that have already been confirmed, then the block validation becomes embarrassingly parallel, because you can just validate every transaction independently. That would be kind of nice.

There's other interesting reasons why this is also useful. I mean, if I were designing it, I would say you have to confirm, because it just makes things simpler. But I was not Satoshi.

So yeah, the order is fairly arbitrary. Any other questions about block ordering, Merkle root stuff? And then we're going to have a quick intermission right at the halfway point. Sounds good, so 256-second break.

So now I'll talk about the synchronization process. How does this actually work in the software when you download Bitcoin? So first, you download Bitcoin. You go to bitcoin.org. of Your friend hands you a USB drive and says, hey, I got some good stuff, man, this new thing called Bitcoin.

And so you've got the Bitcoin EXE file or DMG file, or the binary, or the code. And you want to know what's been going on for the last nine years?

So first, you download the binary, or you compile the code. And you verify all the GPG signatures of this code, if you want to do this securely. So I'm sure everyone has their PGP keys on the MIT PGP server and goes to key signing parties held on the weekends, right? Yeah, no?

AUDIENCE: Keybase is cool. Base

TADGE DRYJA: Keybase is also useful, yeah. So I have my PGP key hash on my business card. I don't think anyone's actually ever used it. But the Bitcoin nerds actually do do this. And they're very sort of annoying about it, because a really good attack vector is to get someone to download compromised Bitcoin code. It's like the best attack vector ever if you're trying to do something sneaky, mainly just to steal all the money.

If you get them to download a Bitcoin binary that you control, that you put some backdoor code in, the code can be like two lines. It's like, open a TCP connection to my computer. Send me all the private keys. We're good.

Or if you want to be more sophisticated, every time they click Send and type in their password, I just change all the addresses and all the outputs to me, and like, every time they try to send money. To UI sort of shows that they're sending money but actually just send to me. And they won't find out for a little while.

There's a lot of things where you want to be running the right Bitcoin code. And that's a hard problem. Because we're sort of operating in this Trust List, decentralized network, how do you get into this in the beginning? If it's your friend and saying, hey, here's the Bitcoin I'm running. I know this is good. Then it works. But just a website-- what if someone hacks the website-- things like that?

It's like a huge rabbit hole. And you can try to worry about it for years. But anyway, you download the binary, assume you've somehow gotten the binary, and you're pretty sure it's the right software.

So how do you connect to this network? Well, there are these hardcoded DNS seeds in order to find peers in the beginning. If you know how DNS works, it's how you look up IP addresses based on a hostname.

There are some servers that will return multiple different IP addresses every time you query them. And those are IP addresses of currently running Bitcoin nodes. So the idea is, OK, someone's running a Bitcoin node. They've got their DNS server. You query that DNS server, and it will hand you out some IP addresses.

This is also sort of centralized, slash trusted, slash whatever, in that if someone compromises these four or five DNS servers, you might not be able to connect to the Bitcoin network. So in practice, it's not completely mathematically secure in Trust List.

There's all these real-world issues that's like, how do I know I've got the right software? How do I know I'm connecting to the actual Bitcoin network? What if my ISP is blocking me and sending me to some other network, or things like that?

So in practice, it sort of works OK right now. You connect to the DNS seeds. And then you connect to a Bitcoin node, and you ask for headers.

You say, hey, I just showed up. I know about one header. There's a hardcoded header in the code called the genesis block that Satoshi did. And you say, hey, I've got this genesis block. Do you know anything that builds above this genesis block, that comes after?

And they say, yes, I actually know 500,000 headers that come after it. And they'll start sending it to you. They send it to you in a couple of thousand of headers at a time. And then you start to download all those and verify them.

The header chain, you get it first. And it's actually very quick. You can do it in under a minute if you have a good internet connection. And you verify all the work before you do anything else.

So this is nice in that the attacker, in order to sort of make you do more work here, would have to do a lot of proof of work. But for you, it's very quick to verify everything.

Even half a million headers, 30 seconds if you've got a good internet connection, something like that, because all you're doing is one hash per header. You just download the header, check the bits, check the time, make sure the times are like progressing reasonably. If the times keep going backwards for like, I think it's 10 blocks, then you consider it invalid.

But your computer can actually do this. It's 500,000 hash functions. And I'm sure if you've seen for the problem set, you can do that in a few seconds in many cases. So you can verify the work done throughout the entirety of the Bitcoin's existence pretty quickly.

So then you've got 500,000 headers. And now you need to actually download the blocks. Any questions about header synchronization? Seems pretty straight-- oh, yeah.

AUDIENCE: Can you catch any of that work, since you're going to see some of these every time you sync?

TADGE DRYJA: Well, yeah, you save it to disk. So you don't have to, like, if you shut your computer off, turn it on the next time, you've already got all those headers on disk. Basically, you save them to disk once you've verified them.

So you download a couple thousand. It builds linearly, so it's nice for you to like download them, validate, and as you validate, write them to disk. And then when you start backup, they're on disk. You trust your own disk.

If someone goes in and modifies things on disk between running of Bitcoin, all bets are off. So you sort of implicitly trust that. So yeah, that's pretty quick, works well. Then you get to the real hard part, where you now have to validate all these signatures and download all these transactions. Any other questions? Good?

So then it's called IBD, initial block download. So you get the headers first. That's quick. Now you start asking your peers, hey, here's this header from 2009, block height 1. Here's the header. Can you give me the full block? I have the header. What are all the things that go into the Merkle root?

So you request blocks from peers. You match the transaction lists, the Merkle root and the header. And you process each transaction in order.

So download it. Say, OK, here's all these transactions. Let me take the hash of all of them. Compute the Merkle root. Make sure it matches the Merkle root I see in the header I've already gotten. And now process each transaction. So what do we do to process transactions?

So you've got this UTXO DB. So this is unspent transaction output. So all the cool-- I'm sure in like 2030, there will be a new slang term where we'll just call money UTXOs, like, hey I've got a lot of UTXO. I mean, I'm already doing that. And I'm pretty ahead of the times, so.

So you've got this database, which is basically a key-value store. And it just has transaction ID index-- so this sort of how you reference inputs in Bitcoin, the transaction ID index as the key. And then the value is just the output, the scriptsig and 8-byte amount.

So it's pretty compact. You've got all these key values. And it's using level DB. But you could use some other key-value store database.

And the idea is, OK, every time you get a transaction, validate all the inputs. Make sure all the signatures are good. Make sure it's spending things that actually exist in your UTXO set.

And delete those inputs from your UTXO DB. You say, OK, this transaction is spending these inputs, so delete. So [INAUDIBLE], sorry.

First, make sure the transaction's valid, given your current UTXO DB. So validate that all these inputs exist. Validate all the signatures are correct.

Then you're saying, OK, this transaction is good. Now I modify my database by deleting all the inputs that are consumed and adding all these new outputs for the transaction. So this modifies the database in place. And you're sort of constantly reading from it to validate inputs, and then writing to it to delete inputs, and then writing to it again to add outputs.

So it doesn't seem too bad. But there's a lot of disk access. And the UTXO DB is a key-value store with a lot of keys. The values are very small.

So it's not like a crazy database problem, if anyone's interested in databases and stuff. But it can be slow. And we want to really optimize it.

So when you think the initial block download, you're doing this 300 million times. So there's about 300 million transactions historically. So you're validating signature, deleting input, adding output, 300 million times. It ends up being about 170 gigabytes of downloads.

And then the end result, when you're done modifying this database, is that you have 55 million transaction outputs remaining. And it's about 3.2 gigabytes of disk use.

So yeah, but you had to download that 170 gigabytes to get to the 3.2-gigabyte end state, because most of the transactions that have been created and most of the outputs have then later been spent. So there's a lot of churn.

So yeah, of the 300 million-- sorry, these are not the same numbers. I was actually looking. How many transaction outputs have been created throughout all of Bitcoin? And I couldn't find the number. And I didn't want to write software to figure it out. But you can certainly figure it out from the blockchain.

But yeah, this is transaction outputs. How many total transactions have TXOs? I'm not sure. But yeah, so it's pretty big. But it's reasonable. Like, we can do this on today's computers. If you've got a decent laptop, this is possible.

This total time taken depends on a lot of factors. Has anyone actually done initial block download and synced to Bitcoin node, and like, want to say about how quickly they did it or? OK, James, how long did it take?

AUDIENCE: For 0.15, it's actually quite quick. On a spinning disk it will take about six hours maybe.

TADGE DRYJA: On a spinning disk?

AUDIENCE: Yeah, yeah, with the new one, it's really quick.

TADGE DRYJA: Because I run 0.15.1 on a laptop with a spinning disk, and it'll take like overnight to just sync up a week or so. It's really slow, but I don't know.

AUDIENCE: Like, my mum tried to start it, and it did it from scratch. It did it in like eight hours.

TADGE DRYJA: Wow, cool, so eight hours to do the whole thing-- anyone else have tried it? Yeah.

AUDIENCE: A while back it took me a week.

TADGE DRYJA: Yeah, a while back it took a week. So the software has been improved quite a bit. So if you downloaded it-- like, I first downloaded it in 2011. And it took overnight to download everything. And the download was vastly smaller. It was less than a gigabyte to download the entire blockchain.

So what's interesting is that the time taken for initial block download over the last seven years has been somewhat constant in that the blockchain gets bigger and bigger. But there's all these optimizations to the code and the databases. And so that sort of keeps pace.

Although actually, I'd say recently it's gotten faster, because like 0.15-- wait, 0.11 or 0.12 had a big speed-up as well.

AUDIENCE: They completely refactored the net web code. It used to be couples of the synchronization. And then they decoupled it.

TADGE DRYJA: Yeah, I think that was mostly Cory, right? So Cory Fields, who also works for the DCI, helped to refactor the code, make it a lot faster. There's definitely still optimizations, a lot of cool-- a lot of it's pretty low-level tweaks kind of stuff. But some of them are pretty big things. Most of the big things, low-hanging fruit, has already been gotten.

The worry is that long-term, this just keeps going up. As the blockchain gets bigger and longer, it's going to take heart. It's going to be harder to validate.

It can be parallelized to some extent. But there's also network I/O concerns, things like that. So it's tricky but doable. Any questions about initial block download? Good?

So here's a question. You've got this UTXO DB. What about this 170 gigabytes? Do you have to store it? Or can you delete it? This you can't delete, right?

So you think this is OK? Yeah, you can maybe delete some of this. Actually, there's a lot of research into maybe we can delete this, accumulators, cool stuff like that.

It would be really cool to have some kind of data structure where we can keep adding these-- we can add, remove, and prove, and then seek, and see if something's in there, where it either is like constant size, or login size, or something like that. That'd be really cool.

There are constructions like that, but they don't work for what we're trying to do right now. But there's a lot of research into that. If anyone here finds some cool data structure that you can use for the UTXO DB that doesn't keep growing linear at size of the number of keys, everyone in Bitcoin will sing your praises forever. But it's an active research area.

So pruning-- by default-- oh, that should be a K, not an M, sorry. There's only 500K blocks, not 500M. Anyway, by default, your client will download all these blocks and store them on the disk. And that's important because what if someone else requests them from you?

Everyone starts out as a noob. Someone else comes and says, hey, guys, I just downloaded Bitcoin. What's going on for the last nine years? And you might want to give them blocks to let them into the system. So you can serve to others who are doing IBD.

However, if you want, and your hard drive's small, or you have an SSD or something, you can prune and delete the blocks after you've done IBD, with no loss of security. Anyone think of downsides doing so? Not really, right?

The only real downside is sort of this. Well, not everyone can prune. If everyone prunes, no new entrants to the system. So it's a little bit of a tricky sort of seed versus leech kind of problem, where someone's got to be there to serve up these blocks.

You don't have to trust them. You're still validating all the work, validating all the signatures. They can't do anything bad. But someone's got to be there to provide the network capacity.

And so it is tricky. Like, most of the nodes on the network are behind people's cable modem firewall kind of thing. So you can't actually connect to them and download. And if you run a node that does allow people to connect in and serve them blocks, people will download quite a bit.

So I have one in the office over there. It ends up sending out about three terabytes a month, which is a lot. Like, it's dozens of gigabytes a day, 20, 30, I don't know. So yeah, people are doing this. People are connecting in and downloading all the blocks, either through IBD or just keeping up with current transactions.

So yeah, pruning is possible. But not everyone can do it, so it's sort of an unsolved issue there.

There's a lot of research into how we can do partial pruning, where, OK, I'm going to only store the last month's worth of blocks, which is mostly what people do, because a lot of people have intermittent connectivity, where they'll turn off their node and then start it back up again a few days later. And they just need to catch up with the last few blocks.

So pruning's cool. That's been in since 0.12 or something.

So I'll go through-- in practice, if you go to your Bitcoin node, what does that actually store? And if you just go to your Bitcoin folder, which in Unix-type OS's is like home directory /.Bitcoin-- total random aside, I don't like how they put a dot in front of all the really important folders.

It's like they hide all the important things, like your GPG folder, It's got a dot. And your Bitcoin folder's got a dot. But like, downloads doesn't. And like, who cares about that?

Anyway, so if you just ls in your folder, here's all the files. And we'll just go through it real quick. Here's the files, and I'll describe.

So there's a banlist.dat. This is a list of IP addresses that you have banned, because they're bad. They're doing something weird. So I'll get to it at the end.

I sort of am thinking of making a ban list for the problem set, because there are some nodes that are doing non-good things. That was what caused yesterday's outage. Someone was connecting. Although it was really my fault, because the server code was not verifying inputs correctly.

But yeah, in Bitcoin, you verify everything. If people start sending you nonsense data, or they say, hey, here's a block, and it's wrong, or hey, here's a transaction, and the signatures are wrong, you'll pretty quickly ban them, because it's like, well, if they're making a mistake-- it's computer. There's no excuse for making a mistake.

So either their software is just different than mine, or something's wrong with their software or their hardware, I don't know. But they're wasting my time. They're sending me nonsense-- ban. So you have your own ban list.

Then the blue ones are folders. I'll talk about those later. But you have peers.dat, which is good nodes. So it's quite a bit, 4 megabytes. And you keep track of here's all the different nodes I've connected to for the duration of however long I've been using Bitcoin.

I keep track of all their IP addresses, how much uptime they've had, what I've downloaded from them. And so I sort of sort them and put the good ones at the top. And like, OK, here's all the different Bitcoin nodes. So next time I start up Bitcoin, I'm going to try to connect to them.

So this makes the network very robust, because everyone remembers everyone else. And then when they need to-- if there's a network disruption, maybe half the nodes go off the network, you can still try to connect to all the rest. And also peers will share their peers files, not directly, but they'll sort of take random samplings of this file and share it with each other, so that everyone sort of knows about everyone else.

Then there's a wallet.dat, which is very important, because that's got all your precious UTXOs. And we'll talk about wallets Monday, I think. There's a bitcoin.conf, little config file. You can set some settings and things like that; a debug file, which shows all these weird messages; and a mempool.dat.

So the mempool is a transaction you've seen that you've not seen in a block yet. So people are broadcasting transactions. And you store them.

It used to be just in memory, hence the word "mempool." Now it's more like disk pool, because you actually store them on disk, because it saves a little speed when you shut down and start up again.

So any questions about just what all these files are doing? Makes sense, so now the folders.

Chainstate blocks and database-- so any guesses onto how big these things are based on previous slides or? So how big is chainstate, for example? Yes?

AUDIENCE: 3 gigs.

TADGE DRYJA: Yeah, 3 gigs. This is a UTXO set, 3-ish gigs. This is all the blocks. What? Oh, no.

And then database, actually, I have no idea. Does anyone know what that is? There's a database folder, and it's got one little log file. And it's like 80 kilobytes. I don't know what it is. Do you guys know? Yeah, I don't know.

But there's a blocks folder, and that's got all the blocks. And that's your huge amount of data. And this is the UTXO set, not too bad.

So yeah, you can look in it. It's reasonable but yeah, it's kind of big.

So any questions about the data stuff? I'm going to go into blockchain as a database, real quick at the end.

So it's 186 gigabytes, or alternatively, you can think of it as just 3 gigabytes. But it's a really crummy database. So I've heard a lot that blockchain is going to change the world. And it's like a database that's shared among everyone. And you can query things. It's a really bad database.

So for example, I'm going to have some fun interactive questions, where some of these are answerable. Some of these are not. And I'm posing the question to my Bitcoin node.

So I posed this question to my Bitcoin node. Hey, remember transaction 9e95c3 dot, dot, dot, from back in 2014? And how do you think the Bitcoin node will answer? Will it answer, or will it not be able to? Any ideas? Yeah.

AUDIENCE: Wait, where does one be easiest [INAUDIBLE]?

TADGE DRYJA: 183 plus 3, so the total data usage on this computer is 186 gigs. The rest are kind of small.

AUDIENCE: What do you mean?

TADGE DRYJA: So I mean like, when you're using Bitcoin, you've got 186 gigs on your hard drive or your SSD devoted to Bitcoin. So you've got this 186-gigabyte database, essentially. But it's a really crummy database. And it can't do a lot of the things you might expect it to.

So for example, this-- arbitrary transaction from the past-- you say, hey, there was this transaction a couple of years ago. Give me the information about it. And what do you think the response from the full node is?

AUDIENCE: It's valid.

TADGE DRYJA: What, sorry?

AUDIENCE: It's valid or not valid.

TADGE DRYJA: It's valid or not valid. Any other ideas?

AUDIENCE: What's the header?

TADGE DRYJA: What, sorry?

AUDIENCE: What's the header of [INAUDIBLE]?

TADGE DRYJA: It asks instead for a header. Any other ideas? Yeah, so sort of that-- it'll say, remember TX disk? No, it's somewhere in the blocks maybe, but I have no idea where. It's not in the chainstate.

So it just stores the blocks. Like, here's this block. Here's that block, in line. And if you say, hey, there's this transaction. OK, go look for it. Oh, 2014, well, that might be somewhere in the middle.

But yeah, if you don't know what block it's in, forget it. So it does have an index of blocks. It'll tell you a block, but transaction, no luck. Yeah.

AUDIENCE: So it pretty much tells you if it exists, and that's it.

TADGE DRYJA: It won't even tell you if this exists. It has no idea.

AUDIENCE: What's the [INAUDIBLE] in the block though?

TADGE DRYJA: I might have made that up. So if you're saying, hey, here's this transaction. Do you remember it? Does it exist? I don't know. Yeah.

AUDIENCE: If you ask-- if you query about a certain block, will it be able to?

TADGE DRYJA: Yes, and I'll-- yeah, good question. But I'll get to that. I think it's in the later slides. But yes, if you create your base on a block hash, then it does have that in the database. And it'll be able to get it for you. Yeah, James.

AUDIENCE: I know what the database directory does.

TADGE DRYJA: You know what the database directory does.

AUDIENCE: Yeah, it's the journaling for the other databases.

TADGE DRYJA: Journaling for other databases-- OK, I didn't-- yeah, cool. It's very small, so I guess it helps things work. So this one-- do you know this transaction? No. How about this?

Well, I've got this output. It's still there in the UTX-- like, someone spent here. It's like, this a transaction in the first one. It's an op_return output. So it's got some extra data. But op_return means it's invalid. It can't spend it.

Can you tell me what the data is? Do you think it'll be able to? If you query, hey, here's this output, what do you think the response will be? Yea, nay? Nay, OK, I'm seeing nays. Yeah.

Nope. If it's an op_return output, even though it's unspent, well, it's unspendable. So you don't put it in the UTXO database, because you just see, oh, this output, op_return is in there. Don't bother putting it in. No one will ever be able to spend it. So there's no reason to put it in.

So op_returns are used to sort of commit to data and all these different protocols. But the actual normal code won't store them. Anything else?

Next one, this one-- hey, I have a public key. And here's the hash of my public key. This is essentially an address. So we didn't talk about addresses. But the Bitcoin addresses that start with like a 1, and then have these alphanumeric stuff-- it's just a different encoding, slightly shorter than hexadecimal, for a pubkey hash.

So you say, hey, I've got this. I have a private key. I just computed the public key. I hashed it. I got this.

Do I have any money? I think I did. I don't remember. But I remember my private key. I backed that up. That was the important part.

Everyone says keep your private keys. So I have my private key. But all this data and all this blockchain stuff, I lost my computer. But I have my private, so I've got my money. How many coins do I have? How many outputs?

What do you think the full node will tell you? Yeah, any ideas? It'll say, I don't know.

Well, you're going to have to search through everything in chainstate. And it doesn't index based on the public key script, only the transaction ID index there. It's a key-value store, and the key is this 36-byte txid:index.

So this is a very real problem. Like, OK, I backed up my key. Or I took my private keys to some other computer or something like that. And this is fairly-- it's gotten a lot faster. It used to take hours, where you had a hard drive, and you're like, OK, import a key to this wallet.

And it's like, well, when did you do transactions? It has to look through the entire blockchain, [INAUDIBLE] and linearly, to see if any of these transactions have an output that matches that, and then says, oh, yeah, you got money back in 2013. Oh, then you spent it-- and sort of replays things, because it doesn't have an address index.

Next one-- this is an example. How many coins-- so you say, hey, here's this output, this transaction:1, how many coins does it have? Will the full node be able to tell you this? Yea, nay? I'm seeing a bunch of nays. No, it will. Yeah, this is the one thing it can do.

So if you say, hey, 7434, dot, dot, dot, colon 1, it'll know that. That's in the UTXO DB, because that's the key that the UTXO DB sorts by. So yep, this is a UTXO. It's unspent. And it has a bunch of coins.

And this is fairly recent. I was just looking through. Someone got a couple million bucks worth, cool. Is that a million? Man.

Yeah, it's a new UTXO set, hasn't been spent, and you can sort quickly based on txid:index pair. So I think this is in some software called an outpoint, where it's like, you concatenate them. And it ends up being 36 bytes.

This is 32 bytes. This is 4. So you sort of have a 36-byte outpoint, which describes what goes into the UTXO database.

AUDIENCE: But once it gets respent, it's hard to find it again.

TADGE DRYJA: Yeah, once this is spent, you delete it from the UTXO, and you won't remember it anymore. It'll just be, hey, you, how many coins does this have? You're like, well-- well, you can still answer it. You say "none."

If it's spent, and you say, how many coins does this guy have? None. It's not in the UTXO set.

Yeah, so the previous one-- I just copied these randomly. So any questions about what is stored and what is not stored? Basically, keeps track of UTXOs, and it keeps track of historic blocks in order to give them to people.

And it keeps after the headers. The headers ends up being small. All the headers total is like, what? 40 megs, something like that.

So yeah, you can add further indices. You could write software to answer all these questions very quickly. But that's not what Bitcoin does by default. Those types of indices would take a lot of extra space and add a lot of CPU or things like that.

So a very common thing is an address index, so people can ask if they have any money. So the second to last one, where you say, hey, I have this key hash. Do I have any money? Do I have any transactions? Having an address index is actually pretty useful for a lot of things, for example, importing keys or like web wallet kind of things.

But Bitcoin by default doesn't do it, because, well, why? You can make arguments that it would be actually useful to have in the normal code, but we don't. Any other questions about what indices, what it can do, what it cannot do?

Somewhat counterintuitive in many cases, where you say, hey, here's this transaction. And you can't actually find it. Or you have to scan through 180 gigabytes in order to find it.

So wait, James, I have a question. So how big is an address index for what you were working on?

AUDIENCE: Usually, equal to the size of the chain inside.

TADGE DRYJA: Wow, so it could be hundreds of gigs. Yeah.

AUDIENCE: It only takes what? At least for Bitcoin, it takes usually multiple weeks to generate.

TADGE DRYJA: Weeks to generate? I bet you, well--

AUDIENCE: Although [INAUDIBLE] usually takes a few days on the inside, which is what [INAUDIBLE] takes-- weeks.

TADGE DRYJA: Well, that's inside. So the other thing is, these are like fairly involved sort of CSE software engineering problems. And optimization really works.

If you download like Bitcoin 0.9, it'll still work. But you're never going to catch up. Maybe not never, I don't know. If you have a fast computer, but it'll take months and months.

And as people have been updating the software and making it faster, making it more efficient, now it's quite fast. And you can sync the whole thing in a few hours on a good computer.

So address index is one of those things where it hasn't had like the full force of all these Bitcoin protocol coder people on it, because it's sort of seen as like, well, yeah that's kind of a fun feature if you want. But it's not like a core utility of Bitcoin.

So yeah, it is a database, maybe not the best way to think of it though. Don't think of the blockchain as like a global shared database, because it sort of is. But it's a fairly specific database that isn't useful for many other things.

Yeah, and it's also untrusted. Another part of why is it's untrusted. Most of these things exist so that they can be used over the peer-to-peer peer network. If you request a block, I'll give it to you. If you give me a transaction, I'll match it against my UTXO set.

But an address index doesn't work that way. It's sort of trusted. I can easily omit things. If you say, hey, I've got a key. What are the transactions involved with this key? I can omit things very easily, and there's no way for you to prove it or verify it.

So your DB queries aren't really given out to network peers. And network peers are scary. And you need to ban them if they act funny.

And this happens all the time. If you look through Bitcoin logs, and you have a node that's up, every few seconds, you're going to be disconnecting from someone or banning them, because they're doing something crazy, trying to hack into you or whatever.

So basically, all you're doing is you're providing headers, blocks, transactions. And you're sharing the other IPs and nodes. You try to simplify it. Other questions?

Yeah, bad database, good for consensus, it kind of works. Everyone's got the same UTXO set, even though they all really would like to change that UTXO set. I would much rather everyone had a UTXO set where I had those 27-coin UTXO.

So almost everyone in the systems would rather there was a different UTXO set. And yet, they all managed to agree on a single UTXO set, so pretty cool.