NEHA NARULA: So to start, we're going to recap what happened last class. And last class we were talking about how traditional payment systems work and some of the downsides when you had a bank in the middle even if you were using techniques like Chaumian e-cash to try to manage what the bank was doing.

So just as a quick reminder, we had this kind of setup where there was Alice and there was Bob. In order to actually make transfers, Alice had to interact with the bank. And in the case of the traditional payment system, we saw that the bank could basically just say no, right? That it was always possible that a bank could decide that it didn't like Alice and it didn't want to process Alice's transactions.

We got a little better than this with Chaumian e-cash, but even with Chaumian e-cash, the bank was required to do something. They were the ones minting the tokens, and they were the ones redeeming the tokens. So the banks still played an integral part in this system. And that actually really does make a difference. It's not just a theoretical problem.

There are many accounts of people with, for example, PayPal accounts, who have their funds frozen, so they can't access the money that they have in PayPal. So this often happens to merchants, actually, who end up accepting customers from different countries, or things like this. They can get flagged by people's algorithms as being potentially fraudulent, and that can put a hold on their funds.

In the case of Chaumian e-cash, you'll notice that we're not all running around using digital tokens issued by banks. Chaumian e-cash didn't really work out. And that was in part due to the fact that banks weren't really interested in implementing this technique.

So-- OK. So this is where we ended last class, which was we're in this situation where the bank has a lot of control over payments and how payments happen.

And something I want to emphasize here, as well, is this isn't-- it's not just that we worry about

this because we want to do illegal activity. Having an institution in the middle of payments and being able to facilitate payments can be a problem in other ways, as well, because it can just simply make things slower. It can make things-- it's harder to innovate. It's harder to introduce new features. There's a sign up process. So there are reasons why we don't want to have an institution in the middle, even if we're totally honest actors.

So the question-- so where we ended up was how do we build a decentralized digital token transfer? So we ended with Chaumian e-cash. And now we're in this position where what we want to do is we want to figure out how to make payments without a bank in the middle.

So there's a few problems that we're going to have to address in order to make this happen, OK? So first of all, we really don't want it to be possible for other users to intercept a transfer of payment and steal funds. If our system can't support this very basic thing, then no one's going to use it. No one's going to trust it. No one's going to rely on it. So it's very important that we have security, right?

Another aspect of security is that we don't want to allow people to spend money without authorization. There has to be some mechanism by which we're really sure it's Alice when Alice wants to make a payment.

Another problem that we want to avoid is the double spend problem. So what is the double spend problem? Well-- oh, and an equivocation problem. So what is a double spend problem? Well, when you have digital tokens, the problem with digital information is that it can really easily be replicated, right? So you actually need to think pretty carefully. The naive solution of just having a unique string that you blast around as your a digital token doesn't really work. You have to do something extra to think about how to ensure that these-- once a token is spent, it cannot be spent again.

Now, what I mean by the equivocation problem is you want to make sure that once a spend happens, it can't be taken back very easily, right? So if I'm being paid by Alice, I have some confidence that once I actually receive the money I've received the money. Alice can't renege on that promise, OK?

So these are some of the problems that we want to solve with a decentralized digital payment system. And here are some of the features that we want to have, which is kind of related to some of the problems.

So first of all, we want to make sure that anyone can use it. That's what we mean by the word decentralized. There's no single point of control. There's no one metering access to this system. If you want to make payments, if you want to join the system, you can join the system. So anyone can use it. We need authoritative transfer so that we can be sure that when Alice produces a transaction it's really her, and we believe that. And we want to make it so that you can't equivocate. You can't undo spends.

So that translates into the following terminology that we often use with these systems. We want it to be permission-less. It doesn't require permission to enter the system. Anyone can join. Authoritative transfer, no double spends, and you can't undo. We call that being tamper-proof. So we don't want it to be the case that someone could go back and undo a spend. Another way to think about that is we don't want people tampering with history.

Something that would be really useful here, and something that the bank was keeping implicitly, was a database of who is transferring money to whom, right? That was sort of the little thing in the corner which said how much was in Alice's account and how much was in Bob's account.

There are actually techniques out there for maintaining a decentralized database among a set of participants. This problem of needing to agree on a value amongst many different participants is a problem known as distributed consensus.

So here's what distributed consensus means. And if you've taken A24, then you've probably seen this before. But distributed consensus is the problem of multiple computers agreeing on a value, particularly in the presence of faults. So some of these computers might fail or go away.

You can use this technique, distributed consensus, to build a log of operations with something called state machine replication, and essentially what you have when you do this is a distributed database, which is what we're trying to obtain here. We need to keep track of accounts and who's spending what, and what payments have happened in the system. And so distributed consensus is a tool that we might be able to use to do this.

So the reason that we might be able to use distributed consensus to do this is we could keep a log of all of the transactions in the system. So Alice pays Bob. David pays Charlie. And what this log provides is it provides ordering. So if Alice tries to pay Carol the same thing that Alice pays Bob, because we have this primitive, this log, this globally ordered log, everyone can see

that this came after that one, right? Everyone can see that Alice was trying to double spend. Alice was trying to spend the same coin twice.

So this primitive is going to be really useful. This idea of this globally-ordered log of transactions is what fundamentally underlies all of these things.

So we could say, if we were to see this log and we were to see this transaction pop up, we can decide that this doesn't really count. That this is-- Alice already spent her coin. A property that we want to maintain with these things is that you can only spend a coin once. So therefore, we're not really going to allow this to happen.

So as I said, the problem of distributed consensus is that when you have a whole bunch of people working on this log, you can tolerate faults. So if some of the people go away, if it's only up to a certain number, it's OK. You still have a consistent, globally-ordered log.

So there's a lot of existing systems that actually provide this property. Some of them are Paxos, ZooKeeper, and Raft, if you've ever heard of them. So the way that these protocols work is that all of these participants are executing the protocol to agree on the log. If a few fail, the protocol can still operate. Usually they can tolerate up to a minority failing.

So these protocols, the ones that I just listed, operate in something called the crash fault tolerance model, meaning they can handle computers going away, but not really much else. And when we're operating in decentralized digital payment land, we want to be able to tolerate more than crashes. We want to be able to tolerate actively malicious behavior. Because if you're going to create a digital payment system, and it's money, then people are going to try to steal that money.

So like I said, if people go away, if it's up to a minority, you can still have the log. So there's stuff from the literature that actually supports something close to this. There's something called Byzantine fault tolerance. And Byzantine fault tolerance tolerates more than crashes. It actually tolerates actively malicious participants. It can tolerate a smaller number of actively malicious participants, up to a third.

So here, nodes might not just fail. They might actually try to subvert the protocol to keep from landing on a single globally-ordered log.

And like I said, we have protocols that can tolerate this. Great. However, and these protocols

are very, very old, OK? Like the Byzantine general's problem, which was a basic formulation of what led to Byzantium fault tolerance, was formulated in 1982. Paxos and view-stamped replication also came about in the '80s. Those were two distributed consensus program protocols. Practical Byzantine fault tolerance, so an actual, very practical protocol that you can implement and that you can use to create a globally-ordered log, that was in the '90s. And ever since then we've had a lot of different protocols. So-- And then in 2009, Bitcoin.

So this idea, this idea of a globally-ordered log that can tolerate malicious participants, the thing I want to impress upon you is that it's an old idea. Decades old.

So what's new? What happened with Bitcoin? Well, a downside that all of these systems have is that you need to know exactly who is participating in the protocol. So the way that these systems work is that the protocol works based on identity. So you have to know who is involved. You need to know everybody in the system. You know everybody in the system. You can tell who's sending you messages and who's not sending you messages, and you can make decisions about what everybody else thinks the globally-ordered log looks like.

The thing that these protocols did not tolerate was when you don't know everybody in the system. And the problem when you don't know everybody in the system, and you're not doing participation based on identity, is that an attacker could create a whole bunch of identities. Well, there's nothing preventing an attacker from creating identities.

And remember, we want a permission-less, decentralized system. We don't want to have some gatekeeper saying who can enter into and out of the system. And given that we don't want to have that gatekeeper, we can't just let people join willy-nilly and manufacture their own identities, because if they do, all of these protocols, all of a sudden, no longer work. This is what's known as a Sybil attack, OK? When an attacker can create identities, essentially for free, and can swarm the good guys and take over the protocol.

So where I'm going to leave you is with this problem, which is we want to figure out how to address the Sybil attack problem when creating a globally-ordered log. And one way of doing that is to think about making identities costly. So good guys can create new identities and can join the system and can execute the protocol. Bad guys could potentially create identities and join the system and execute the protocol, but because identities actually cost something, there's a limit. There's an inherent limit. They don't come for free. The bad guys can only create so many. And that's where I want to end. Are there any questions about that as we

transfer over to Tadge?

AUDIENCE: I have a question.

NEHA NARULA: Yeah?

AUDIENCE: Does that model you presented to deal with distributed consensus assume you have synchronicity in the system?

NEHA NARULA: Not necessarily. So that problem is unsolvable in the case where there is a totally asynchronous system. But there's a lot of different ways to create a model where, for example, if you have random coins, then it is solvable in an asynchronous system. So this is something that-- you might have heard of something called the CAP theorem. This is sort of a predecessor to that, which indicates that in a totally asynchronous system, without random coins, consensus is actually impossible.

But that's-- there's different ways to get around that. And oftentimes, people assume that the system is semi-synchronous.

AUDIENCE: Awesome.

TADGE DRYJA: Neha.

AUDIENCE: I have one more question.

NEHA NARULA: What? Oh, sorry. Yeah.

AUDIENCE: So with Bitcoin, in the paper they mentioned that they moved away from IP-based identity to CPU [INAUDIBLE] identity. Is that where the cost comes from?

NEHA NARULA: Exactly. And that's what Tadge is going to talk about.

AUDIENCE: OK.

NEHA NARULA: Yeah.

TADGE DRYJA: Anything else? Cool. OK. Hi, everyone. I'm going to talk about proof of work, which is, in my estimation, sort of the thing that started all of this, that, you know, there's all sorts of cool cryptography involved in Bitcoin and different cryptocurrencies, but it was the proof of work consensus that kicked all this off, and that was the big difference.

So we want a cost, Neha was mentioning. We want some kind of cost to create identities or cost to participate in the system. How do we prevent Sybil attacks? And this is a very hard problem, because it tends to be an arms race in many ways, where Twitter or Facebook or things like that have tons of bots. Because as soon as Twitter makes a cool new algorithm to identify what's a bot and kick them off the system, the various people who were trying to make millions of Twitter bots start to figure it out.

They're like, hey, they banned all these bots. Ah, well, I'll switch this around, and make my bots more human-like and better.

I was actually kicked off of Twitter for being a bot, and I am not. I totally am not.

[LAUGHTER]

They said I exhibited automated activity. So this is a hard problem, and this sort of gets into like Turing test, CAPTCHA-kind of thing as well. But Bitcoin's a fairly different solution. And as Neha said, you don't want really anyone in charge. You don't want a set identity list of people who can participate. So that rules out things that other system use, like let's use social security numbers. Let's use phone numbers. Let's use CAPTCHAs.

Well, those don't really apply. Like, what's a social security number? What if I'm in Germany and I want to use it? What if I'm in somewhere else, you know? Phone numbers as well. Phone numbers are just administrated by a phone company who would then become the de facto controller of the system. And CAPTCHAs are kind of an interesting idea, but they-- you know, the identify the signs in this picture or the cars in this picture or whatever. But it doesn't really apply, because it's not easy for people to verify, and not easy for computers to verify, that the people were actually doing it. So those don't work.

So what does work is work. So I don't know if-- I think some people have said they've-- some people said the first day, oh, I did problem set one. I was, like, oh, OK. Cool. Some people have yet to start, which is fine as well.

But if you have started and looked into problem set one, great. I will talk about it a little bit in the next few minutes. The problem set needs many attempts to forge a signature. So if you've done it, you've-- I don't know what the fastest-- probably someone got it to work really fast. When I completed it, I got it to run on eight cores of a fairly fast machine, and I got it to run in like three minutes. And it took me something like 2 billion attempts. Which it should, right? 2

billion is a run.

So this is because the hash functions have random output, right? I'm looking for a string where certain bits are set and certain bits are free. And since the hash functions have random output, I have no shortcut to be like, well, I want an output where these bits are ones and these bits are zeros, and the rest I don't care. I can't really specify that as the input to my hash function. So I just have to keep trying different numbers, all right?

We know what we want, but the only way to get it is just keep throwing in different numbers to this hash function, getting the output, and checking if it matches our criteria. So what do you want out of work in this case? You want it to be time-consuming, kind of like the homework. But the example in the homework has some problems, because it's this weird trusted setup, right?

In fact, it's-- so in the homework case, you're trying to forge a signature where I have the private key. I've made four signatures, and you want to forge a fifth signature without my private key. The problem is it seems like I would have an advantage in this work. I actually don't, because you can verify that I haven't revealed any new bits. But it's a weird setup, right? Where it's like, OK, well, there's one certain actor that like sets up this whole work, and we shouldn't have that kind of thing.

So you want it to be deterministic. You also want it to be scalable. And you want it to be memory-less, where everyone gets a chance and you're not making progress. Because if you are-- I'll get to that later, but if you're making progress, it tends to be that the fastest person always wins, right? If there's no randomness involved. Like, if it's, you know, let's have a race between a bicycle and someone running. It's like, well, that's not really race. The bicycle is always going to win.

Whereas in this case, if it's memory-less, every attempt is a new shot at getting it, and so someone who is 10% as fast will still win 10% of the time.

OK. So in the case of the homework. So here's an example where five bits of eight are constrained. So let's say these first two, you've got the pre-images for both. This is the zero row. This is the one row. And the signatures I've revealed-- OK, you get both bits here. However, here, in this place, you only have the one bit, the one pre-image. Here you only have a zero. Here you only have the one. Here you've got both, and then you're also here, one, zero.

So the idea is these three bits are like x. Like, don't care. Either a one or a zero will work. These five bits are constrained. So it must match 101 and then 10. So you're going to have to keep trying messages. And since five bits are constrained, you're going to have to try 2 to the 5 messages, which is, what, 64? I think? Yeah.

Which is totally doable, right? 64 tries, and you'll probably get it. So for example, you try once, and you say, OK, I got, 11001100. Doesn't work. These are OK, because they're OK no matter what. This one's wrong. This one's OK. This one's OK. This one's wrong.

Let me try again. OK, I got it, right? These can be anything. I got a 0 and a 1. Sure. These all line up. The red ones all line up, so I've found a valid solution.

And so that works as a proof of work, and in this case, I found a solution, which was Tadge forge 1 154262107. And that's a message that can be forged, given those four signatures in the problem set. And anyone can, you know, whatever homework solution you have, you can probably copy and paste that string and check that if you hash that, you'll get a hash that matches up and can be signed with those four signatures.

So then the question is, did I do 154 million tries to get that? Like did I start at zero? Maybe I did more. Maybe it was random. Maybe I got lucky and I started counting at 150 million. So it's not really a proof, is the thing. Like, you're not exactly sure how many attempts someone did. Not quite a proof, right? There's a lot of chance.

However, over many proofs, it averages out and you can be pretty sure that someone's doing all the work, right? So if-- you know, if someone gets lucky once, fine. Like, maybe they tried a couple times and got a valid proof of work, but eventually, if they keep doing proof of work and they're iteratively showing them, then you can be pretty sure that they're not going to be lucky the whole time, right?

And so you have to estimate the collision difficulty. The best way to estimate the difficulty and the amount of work that people have done isn't to look at their proof, isn't to look at their nonce, but to look at the constraints of the system before they start, right?

So in the case of the homework, 31 bits are specified, right? So with four signatures, you'd guess that about 32 bits would be-- and in this case, there's 1 bit difference. So in the case of the homework, there's 31 bits that are specified, so that means you're going to have to do

about 2 billion hashes, right? 2 to the 31 is like 2.1 billion, or something.

And so that's the best estimate of how long it took, regardless of what the nonce is, regardless of what it looks like they did, you could say, well, it's going to take on average 2 to 31 attempts. Let's just-- and so if any valid solution they give me, I'm going to credit that with 2 to the 31 work.

So actually, in the case of mine, the one is the thread number, and there were eight threads, so this Tadge forge 1 154 million. So I had eight threads going in parallel, and that happened to be thread one. And so they all went about the same speed. So it's really like, 154 million times eight, which is about 1.2 billion.

And so that's actually kind of lucky, right? Like, it should-- you would estimate on average you would take 2 billion, and I got it at 1.2 billion, so pretty good luck, but within 2x of what you'd expect. And since you can look at the probability distributions of like-- that might be kind of interesting with the homework, like OK, everyone's attempt, some people it takes 2 billion. Some people get lucky and it takes 1 billion. Some people get unlucky and it takes 5 billion. And you're going to have this curve to see it.

But this is-- that's a kind of weird application of this signature forging algorithm, all right? It's not what it's made for. We can have a simpler proof of work that has a specific target that's reused forever, and it's more universal in that it's not, like, oh, here's this private key. Here's this key pair and here are these four signatures. Where are these things coming from? That's weird. We have to match these weird strings of bits and stuff. That's kind of annoying.

So it's better to do something similar, but much simpler. We'll just try to collide with a fixed string. So there's some defined string of ones and zeros, and we just say, OK, well let's try to have a collision with this. And the simplest is collide with zero, or you could also collide with like FFFF. But in the case of Bitcoin and the case of many of these systems, we say, well, let's just try to have a low number.

So let's interpret the hash output as an actual number. In most cases-- do we have any hashes out here? No. You know, it's a 32 byte string, and we think of it as just, like, here's a blob of bytes.

But you could cast that into an integer, right? You could say, oh, let's interpret these 32 bytes as a uint 256. Right? A 256-bit unsigned integer. And let's look for really low ones.

And you can do that. I think with some CPUs, they'll actually let you do giant integers that way. But even if not, you have some kind of big int library in your computer, which just interprets it that way and says, OK, let's try to find collisions with zero. Basically, let's try to find low numbers. Yeah, OK.

So this idea-- there's a paper before hashcash, but I don't think hashcash was the first software. This idea is actually really old. In 1997, Adam Back, who still works on this kind of stuff now with Bitcoin, the idea was to stop email spam, which-- I don't if-- you know, in the late '90s, was actually a huge problem. They didn't have all the cool like different machine learning stuff, and there's tons of spam.

And the idea was, OK, put a nonce. And a nonce is this-- we call this a nonce, right, the random number you're throwing in to keep changing things. Put a nonce in your email header, and you try to get a low hash output when you hash the whole thing. So within the UI for email, you don't necessarily have to see, OK, what's this person's nonce? The computer just does it all automatically for you. But the idea is you need a partial collision with zero. You need a low hash output for your email, before your email, the receiving person, will display the email.

And since the header includes the destination address, if you're a spammer and you want to spam a million different people, you're going to have to come up with a million different proofs of work. And that could take quite a while. Like, let's say an average CPU can do it in two or three seconds. That means you're going to have to be waiting months to spam all these emails.

However, for a normal person, if you're just sending an email to your friend, takes a few seconds, right? Takes two seconds for your CPU time, which is not a huge deal. So this was a cool idea. It never really got off the ground just because, I don't know, Gmail happened, Hotmail happened. A lot of web mail things happened. People don't really use--

AUDIENCE: I think there was a paper that the economics of it doesn't work.

TADGE DRYJA: Yeah, I mean, spammers often have bot nets. So, yeah, there's a lot of other issues, right? Like there's-- it didn't take off for various reasons. But yeah, also, spammers have botnets, which also sort of hurts the whole idea of proof of work, in some cases, which we'll talk about in a second.

OK. So the simpler proof of work. So for example, if you go onto your computers, and you say,

echo Tadge 4233964024, and pipe that to sha256sum, you will get this output. And that's eight zeros in the front, so that's 4 bytes. I did 2 the 32 work, and it's-- you know, that's me, right? And so that proves that I'm a hard worker, and I'm going to put this nonce on my resume, and everyone can see that I do work.

That is kind of silly, but it's a really easy universal way to have a proof of work. Anyone can put their name or any message, and any kind of nonce, and we all just pipe it into the sha256 function and see how low the output is. In this case, it's actually 2 to 33, right? Because the first digit is a 7, so that's 0111.

But this is what we use in Bitcoins. It's what they use in a lot of these things. I mean, it's really straightforward.

OK. So partial collision work. It has a lot of the properties that we're looking for when we want this consensus system, right? It increases the costs of equivocation and helps with Sybil resistance, in that if I want to fool you, I'm still going to have to do a lot of work. And I need time to do that. I need electricity to do that. I need a bunch of computers to do that with.

Also scalable, because a ton of work still takes the same amount of time and space to verify as a little bit of work, right? So I can do O of n work, right? I can put as many zeros as I want, and the nonce doesn't really get any bigger. The nonce never has to get bigger than the hash output, certainly. And the time to verify is the same no matter what, so that's really cool.

So why do we do work, in the case of Bitcoin? The idea is you use a chained proof of work as a distributed time stamping mechanism. So it's to determine what happened first. So in the case of a double spend, where someone is saying, oh, transaction A happened first. And someone else says, well, I think transaction B happened first.

You can't just rely on what other people are telling you, because who are these other people? There's no real identities here. So you look at what got into the chain of hashes first. And you say, OK, well, that one-- this one, this data, has a hash of this data, so it must have happened after it, right? And the things that happened before are pointed to by the things that happened after. Because the idea is I can't include the hash of something that I haven't seen yet, right? It should be impossible to do that.

So the idea of the blockchain. You've got some message m, some nonce. Let's call it r, and some target, t. And this actually changes, but we're not going to talk about the changes yet. So

you tack the hash of your message and the nonce-- like I was doing with Tadge-- and the number, and you say, OK, that's h. And h has to be less than some number t. So in the case where I just said my nonce, I had 2 to the 33 work. So say the same thing here. In the example, I'm going to only require 2 bytes for the target, so that target needs to be fairly large.

And then the message for block n includes some data, and also the hash of block m minus 1. You refer to your previous block hash in your current block data. So for example, message number two would be data 2 concatenated with the hash of data 1 and r-- well, there should be r1 there. OK.

So I'll show you a little diagram this. So the idea of a blockchain, which-- yeah?

AUDIENCE: You'll need to back up. Could you explain why it's less than?

TADGE DRYJA: In this case, we want low numbers as the hash outputs.

AUDIENCE: Does it have to be less than?

TADGE DRYJA: You could do greater than. You could have, I want something that starts with a lot of Fs. It would work the same, you know? You could-- yeah?

AUDIENCE: The idea here is you want to constrain, right?

TADGE DRYJA: Yeah.

AUDIENCE: So you can do that with less than or you can do that with greater than. You could do that with a fixed sort of--

TADGE DRYJA: Yeah, you could--

AUDIENCE: Little bits have to be--

TADGE DRYJA: Well, there's something-- OK, so less than and greater than are actually nicer than just fixed bit. So you could say, I want it to start with the repeating pattern 10101010. That would work, too, but then you're constrained in how fine tuned you can turn the knob, in that you're going to either have to add another bit of constraint or remove a bit of constraint. And so that, you know, you're going to have to go in 2x jumps.

What's nice with this is you actually have very fine-grained control over how much work you're requiring, because you don't-- so in this example, I said, hey, I have this many zero bits. But I

can also say, well, I needed to have a number less than 7f-- dah, dah, dah, dah, dah. Like, less than 7f 000 here. And this satisfies it, because hey, that's 7e. That's less than 7f. right?

So I can actually have a pretty specific thing, not just in terms of number of bits specified. So greater than or less than is kind of nice in that sense, because you can have very small changes to the target. Oh. OK. Any questions about all this crazy stuff? Cool. OK.

Yeah. So you can you can specify. So in the case of the homework, it's just number of bits, right? Because they're all scattered throughout the field, whereas in this one, you can say, well, it's the targets greater than or less than. In this case, less than. So I have fine-grained control. Which, we're not going to-- we'll get into later. But, yeah, in the actual case of Bitcoin, the target can vary, right?

So when Bitcoin started, you needed 4 bytes of zeros, right? You needed to do 2 to the 32 work. Now you need to do way way, way more, and there's an algorithm in Bitcoin that adjusts what that target is. And I'll show at the end of this-- like a current Bitcoin proof of work.

AUDIENCE: We've got to talk about that. I was going to ask you, [INAUDIBLE] time? How do you know [INAUDIBLE] solve?

TADGE DRYJA: Right. Right. So time is sort of a tricky, semi-ugly subject in Bitcoin. It'd be really cool if they could get rid of it. But time is used, and only to vary that target, right? So when you mine a block, you say, I'm going to put what time it is. And you look at the last two-- when every two weeks, or so, every 2016 blocks, you look at them, and say, OK, what's the first timestamp of this 2016 period and the current timestamp?

And if that took two weeks, cool, we don't have to vary the target. If it took less than two weeks, let's crank up the difficulty, or lower the target. And if it took too long, let's make it easier by raising the target. And so really, you're only comparing the first and last thing.

During the process, when everyone's downloading stuff, I think the general rule is if you see something that's off by more than two hours, you reject it, but it's very lax in terms of synchronization, right? If someone's 10 minutes off, you're like, yeah, sure. I'll take it. Even if block eight says it happened before block seven, refers to block seven, and yet has a timestamp that's before it, you're still like, yeah, OK. Fine. Whatever. People's clocks are off.

So that the time is fairly lax.

AUDIENCE: What happens if, like, the person who has the last block-- who decides-- who timestamps it?

TADGE DRYJA: You just time-- the person who's doing the work timestamps it.

AUDIENCE: Could an actor timestamp it wrong to like--

TADGE DRYJA: Yeah. To screw around with the difficulty? Yeah.

AUDIENCE: OK.

TADGE DRYJA: You're fairly constrained, right? So it should be two weeks, and you can say, well, I bet if I'm off by 20 minutes everyone will be OK with it. But that's not a ton of control, right? If you're the last one of this difficulty adjustment thing, you could say, hey, I'm going to say it's-- I'm going to say-- what would you probably want to do?

I'm going to say it's 20 minutes in the future, because I want the difficulty to not go up too much, because I'm trying to get more blocks. So you could try to push it to 20 minutes in the future from what it actually is, and people, all the rest of the nodes in the network will probably accept it. But you get 20 minutes difference over a two week period, which is way less than a percent, so--

AUDIENCE: And you said two hours is-- like, it would not take a block that says it's a week in the future?

TADGE DRYJA: Yeah. Yeah. So-- the thing is, it's hard to-- you don't really necessarily know everyone's consensus rules in these systems. But I think the default is two hours from your clock. And every client I've seen will reject something that seems to be more than two hours off. Which is really lax, right? For most types of distributed systems, two hours is crazy. Most of the time they're going to be within a few seconds.

So in general, I don't think it ever is a problem in terms of that. So yeah.

But what people do-- OK. So one of the things they do, which I can get to here. Since there is a timestamp in the message being hashed, some chips that are purpose-built for doing this work, will actually flip bits in the timestamp, flip the low order bits, because they're like, well, no one cares what second it is, so I have a bit here that I can just turn on and off and use as a nonce. So the last, the lowest few bits, they just randomize. They're like, well, if I'm off by five seconds, no one cares whether future, past, so I'll just use this as nonce space.

OK. So-- but I'll get into that a little more later. So anyways, so a block. Let's call this. A block

has a bunch of data, and then we hash it, right? And in this case, it's got three things. It's got a previous hash, so a pointer. It specifies the thing it's building off of. It's got a message so that you can add your message that you're adding to the blockchain. And it's got a nonce, so that you can prove you're doing work.

And then these are the things, the actual data, but then the block itself has a hash, right? And this hash is not included anywhere in the block, because it couldn't be, right? This is the hash of this data. And so we can compute it, but we can't we can't stick this 00db thing here, because that would then change the data itself.

So the idea is, use these block hashes as identifiers. And in most of these systems, in general, the hash of something is its identifiers, its name, the way to point to it.

OK. So the next block includes a hash of the last block, right? So here you're saying, OK, the previous block is 00db. You're pointing to it. And then you're adding your own message, oh hi. And you're adding your own nonce to make-- you know? And so these two things you start out with. You start out with the pointer to the previous block, and then you say, OK, the message I'm going to try to add here is oh hi.

And then you start grinding through these, right? You start iterating or randomly attempting different nonces, until you found one where it's low, right? It's got a bunch of zeros. And so this is the nonce you need to specify to get a lot of zeros. And in this case, there's four characters of nonce. And you only need two characters of zero, so it's way more nonce space than you need. Yes?

AUDIENCE: How many possible nonces are there that would result in this?

TADGE DRYJA: In this case there would be, on average, 255 nonces that would work, right? Because-- I'm, OK. I haven't actually specified these things, but assuming the target here is must be less than 00ff, right? So you need eight zero bits in the front, and all the rest can be Fs. And you can have any nonce you want.

And this is an actual hash function, which is 2 byes long.

That means you've got 2 bytes here and only 1 byte specified, so you've got two to the 16 nonce space, and your output is constrained in 2 to the 8, so you're going to have 2 to the 8 different nonces that will work.

In the case of Bitcoin, there's nowhere near enough nonce space, right? Because your actual constraint on your proof of work is a lot of bits, and you only have 4 bytes, or 32 bits, of nonce space. So what you actually end up doing is using your message as further nonce space, right?

So the thing is, I could put oh hi 2, and now I've changed my message, which will change my hash. And so if I go through my entire nonce space and don't find a valid proof of work, I can edit my message and then go through it again. So that-- it's a little ugly, right? It'd be nicer if your nonce space was big enough that you'd never had to do that. In the case of Bitcoin, we don't know who designed it, and whoever designed it didn't put enough nonce space, and so it's kind of annoying. But you can get around it.

In theory, you could just eliminate the nonce space entirely and say, well, just put it in the message, right? But it's nice-- it's much easier to think of when you're like, here are these separate things. Like, this is random. It has no real meaning. This is the thing that has meaning that we're going to use. Anyway.

So yeah. The chain keeps building. You add work each time, right, by finding different nonces that match up with your message. How are you? Oh, I'm good. OK. And the idea is if you flip a bit in any block that's come out. So, for example, you change oh hi to oh hey, and try to leave everything the same, you're going to change the hash. So most likely, the proof of work will no longer be valid, right? Oh, this starts with a 9 now. No one's going to use it.

And more importantly, actually, these pointers stop working, right? Because this block was pointing to 002c. That's not this. That's this other one. So basically, you flip any bit in the entire history of this chain, and it breaks, right? So you can't go back and edit things. You can't change messages after the fact. So it gives you the nice property of immutability that once a message is included in the system. So in the case of money, once a transaction has moved funds from one place to another, you can't go back and change it. You can't undo something.

Unless. Unless you fork the chain. So the idea is everyone is running this system. Sometimes inadvertently, or sometimes on purpose, you can have two branches, right? You can't have something point to two different previous blocks, right? There's one specified-- so in this case, when I say previous block, it's a single, specific hash that I'm pointing to.

And since we're pretty sure we can't find collisions, that means if I'm pointing to 00db, it's only this one, which has the message hi. I should not be able to find two different blocks which

hash to 00db, because then it's ambiguous, and like, wait, which am I pointing to? So I shouldn't be able to do that.

However, it's easy to have two blocks here that both point to the same ancestor, that both-- we call this a parent, call this a child, call this a branch.

So this is easy to do, right? Two different people can just come up with their own messages and own nonces and point to the same ancestor. They could do this inadvertently, because they're not aware of each other's work, right? Or maybe these two things happened at the same time, and they're like, oh, well we both found the answers at about the same time.

Or they could do it, maliciously, where I saw that you made this block, and I saw that you made this block, and I'm going to make these two, to try to confuse things.

So this can happen. And the rule in Bitcoin, and most all of these systems, is OK, well, the highest blockchain wins. The most work wins. So that's just our metric for what we all determine as the state of the system. So everyone uses the chain with the most work.

And that can change, right? Let's say we started off with the 008a being the chain tip. That's what we call the current state of the system. And then someone came along and said, well, I'm want to get rid of this thing. Something happened in 94 that I want to rewrite. I want to get rid of the how ru message, and I'm going to replace it with my own message.

And so the way I do it is I branch off from 002c, make this one, make this one, and now, I've made another one. So now, this branch here has the most work. And everyone's going to use it. And they sort of forget about these two things. They're like, yeah, that was the tip. That was the most work state of the system, but now this is.

So now we have to go back. We have to erase these two messages, and add these three messages, and now everyone agrees that that's the next state the system. And this is called a reorg, a reorganization of the block.

This is sort of bad, but yeah. Yeah?

AUDIENCE: How does the whole process of-- so no one realizes they're in the wrong chain, [INAUDIBLE], right?

TADGE DRYJA: Mm-hmm.

AUDIENCE: How do they say, OK, I need all the information to build the actual chain?

TADGE DRYJA: Yeah. They request it, or basically, someone says, hey, I have these three. And you're like, really? What are those three? And they give them to you. And you're like, OK. Yep. That works.

So the actual network-- I think in a few weeks we're going into the actual network messages that are used in Bitcoin, but they're not like-- you-- actually, if you wrote it yourself, you'd probably write something more sensible and better. It's a little weird. But basically, you sort of-- nodes in this network are all connected to each other, and they report on what they have. And they say, hey, I have 0061. And you're like, really? I don't have 0061.

And you think, maybe 0061 builds off here. You request 0061 and you're like, oh, it builds off of here. And then they're like, I have f2. I have a3. And so they build these-- you know, they request them all and build them.

And then they actually keep these in memory for a little while, and then-- but I don't think they save them on disc. So you can have multiple different branches, concurrently, in your software, and you just use this and show this to the UI.

Yeah?

AUDIENCE: Who specifically do you ask for the chain? Because if it needs only one, then you could just sort of--

TADGE DRYJA: Needs only--

AUDIENCE: If only one person had a longer chain, and then everyone starts replicating it, then the network starts believing it, right?

TADGE DRYJA: Yep. Yep. So it's a gossip network. So basically, there's no real identity for all the different nodes of computers. You also will report all the-- you basically do it by IP address, right? So when I connect to a node, I don't think I ask. I think they just tell me, like, hey, here's a thousand IP addresses that I know of, that I've connected to, that are running this software.

And you're like, OK. I'll add that into my list. And then you just randomly connect to people. I think the default is randomly connect to seven peers. And then when you-- when anything comes in, you're like, hey, I got a new-- you know, I got a new block message. You will tell all

seven of your peers. Hey, I got 00a3. And if they request what the data is, you'll give it to them.

So in practice, things get propagated pretty well. But yeah, partition attacks. If you can isolate-- so like, if you can isolate this off the network, you may be able to prevent this kind of thing from happening. So if someone built these three blocks but isn't able to broadcast it to the network, then people will still think that this is the most work and build off of this.

So yeah, those are attacks that can happen. Yeah?

AUDIENCE: What happens if you reorg transactions, for instance? Like if the messages were transactions?

TADGE DRYJA: Yeah. They stop being-- they get undone. So this is a very real risk. If you have a-- Alice is sending Bob 5 bitcoins in this block. And in this block, you have a conflicting transaction, Alice is sending Carol 5 bitcoins, the same 5 bitcoins, Bob could think at this point, hey, I got some money. I have some money. Oh, shoot. I, whoops, don't have the money anymore when this gets reorged out.

So that's-- that is an attack, right? And you have to be aware of that in using Bitcoin, or any of these kinds of systems. They're not-- you have consensus, but I think it's like eventually consensus. So generally, the rule of thumb people use is wait six blocks. I think that's in the original Bitcoin white paper. Which is fairly arbitrary. Like, you know? The longer you wait, the more certain it is. The more difficult it will be to perform this kind of attack, because you're going to have to do more work, right?

And the idea is everyone's building off of the one that they think is the tip. So everyone's going to-- you have to get pretty lucky or outrun everyone else to perform this kind of attack. So. OK. Yeah?

AUDIENCE: Are those known as uncles?

TADGE DRYJA: Oh. So like, in Ethereum and some other systems they have uncles where you can point to multiple ancestors, which is weird, because your uncle is not really your ancestor, but anyway.

[LAUGHTER]

So that would let-- for example, that would let 00f2 to point to both 0061 and 0094.

NEHA NARULA: The answer is no. These are not uncles.

TADGE DRYJA: Yes. So yes.

NEHA NARULA: [LAUGHS]

TADGE DRYJA: Sorry. Uncles is a separate thing. This is not uncles, but related, right? You could draw, like-- you could draw an arrow there, and-- I don't know.

OK. So I'll talk about pros and cons, and some of this gets into not as objective and somewhat subjective ideas. But it's an interesting thing to talk about, especially in context of money.

OK. So pros. Anonymous, member lists, scalable, non-interactive. I'll go through all of these. Tied to the real world.

Cons. Pretty much all nonces fail. It uses watts. It uses chips. There's 51% attacks. And people hate it.

[LAUGHTER]

So the pros. They're quite good. It is anonymous, right? You can mine without any kind of identity. Even after you've mined, you don't really have to say who you are. There's no pre-known keys. There's actually no signatures involved, right? There's no public key system at all. You're just trying different nonces. Anyone can go for it.

Every attempt is equally likely to succeed, since it's random, and it's not limited to humans. So you could have-- you don't need a social security number. Bots are welcome. Anything can mine. And everyone doesn't really care who mined it. It's not about who. It's just oh, there's a nonce. There's work. OK, this is valid. So that's really cool.

It's memory-less. This is important, and a little counter-intuitive in some ways. There's no progress. So if you have 10 trillion failed nonces, your next nonce is just as likely to fail, which is extremely likely. So that makes it a Poisson process, right? Which is-- you know, it's a certain probability distribution, kind of thing. You will always expect the next block in 10 minutes from now.

So, hey, it's been 20 minutes since the last block came out. When's the next one coming out? Probably 10 minutes. And there's no-- it's sort of like-- you know, there's no progress being made. You always have no memory of what the last few minutes have had. So the fact that it's been 10 minutes, the fact that it's-- or it's been two seconds, you still think it's going to take

about 10 minutes.

Which is nice, because it means there's a linear trade off between how many attempts you're making and what your chances of finding the next block are. So if you attempt twice as many, you have twice as good a chance of finding the next block. Which is good, because if there is progress, right, if you had some kind of function where the probability was not just linear number of chances, but if it was super linear, exponential or something like that, you-- the fastest person would always win, right?

So if it's like, OK, I need to compute this thing, and it takes a couple trillion computations. And so for some computers, it takes five minutes. For some computers, it only takes two minutes. The computers that can do it in two minutes are just always going to win, because the people who take five minutes to finish it, well, they just did it in two minutes, and I can't keep up with them.

So this is really nice, and it makes it competitive. Otherwise, it would end up being like whoever is fastest just consistently can perform the work.

Any questions about memory lists? It's kind of-- this-- I mean, you guys actually know computers, and stuff, and math. You know, MIT. But this is a fairly common misconception in Bitcoin and other systems like this, where people get mad. Like, it's been an hour and no block's come out. It's like well, yeah, you know? Poisson process. On average, every day and a half there will be a one hour gap.

And then people have been, like, well, the last three blocks came out in like, two minutes. So there's going to be a awhile before the next block. It's like no, no. There's--

[LAUGHTER]

There's no memory. So there's a lot of misconceptions when it's a purely random system, that people are not used to this kind of thing. But it's counter-intuitive, right? Even I think, like, oh, well-- no, no. Memory-less. Got to remember that. Got to remember that it's memory-less.

OK. So scalability is actually amazing. So this is a block hash from this morning, or late last night, that actually happened in Bitcoin, and there's a lot of zeros. There's 18 of them, and 9 bytes, and actually more, because it starts with a three instead of an f or something.

So what's really cool about this is it takes just as long to verify this work as it did with the one

with my name, right? Which only had 4 bytes of work, you know? And the one with my name, it took my computer, I don't know, 20 minutes or whatever to do. This took computer-- you know, this one took 10 minutes, right? But it took the network 10 minutes, because everyone's participating at the same time.

So yeah. And it's a trillion times more work, and takes no more time to verify. That's really cool, and it is kind of amazing, because this is almost a mole of attempts. Like, Avogadro's number. So you're getting into numbers that like, you never deal with these scales of numbers in-- I mean, you do in chemistry, but, like, whoa. So that's a nice property.

Also, it's non-interactive, and that helps as well. So you never have to report your failed attempts, right? So you can have a thousand different computers all trying one nonce with different messages, or one chip trying a thousand times, and you don't need any communication between them. So the only communication between in the network is when a block is actually found. So there's no real setup needed. So it's a very simple message protocol where you say hey, here's a block. Everyone just broadcasts. That's really cool, too.

So these are all really useful properties for our network and our consensus system. Oh, also. It uses real world resources. So when you want to go back and rewrite history, even if a majority of participants want to do that, there's still going to be a cost. So that's pretty unique compared to many different says consensus systems. In many systems, say, OK, well, we have unanimous agreement. All participants in the system are going to rewrite history. We're going to go back, pretend that never happened, and branch off and create a new history.

Even if everyone in Bitcoin, every participant, tries to do that, they're still going to have to spend that energy and that work. And so that really discourages reorging and trying to rewrite history, even when everyone wants to. So that's cool in that you're tied down not just by the other participants in the system, but by physics itself and the way the world works. We are going to have to do work to rewrite this. So that's-- and also, it shows that people are sort of invested, and they've done work. So that's cool, too.

OK. So any questions about all these pros? Or if you want to say some of these pros are actually cons before I get to the cons? OK. We can go to cons. And this-- it's always more fun to complain about stuff. So we can go into cons, and I'm sure people can jump in with other things that, oh, yeah, this is bad.

So one problem is that it's inefficient. Almost every attempt fails. So that's no fun, right? So the

proof of work from a few hours ago, this is 10 to the 22 attempts. And all-- you know, there's 10 to 22 failed attempts that were required, that happened, in order to get that one. So that's extremely inefficient, you know? I don't know how to express that as a percentage, but the actual things moving the system are some incredibly small percentage of the actual things happening.

It's also a problem because you're not going to be able to get a valid block, right? If you have to 2 to the 72 attempts, you're just not going to do it. Like, ever. You can run your laptop for your entire life and you will not be able to do two to the 72 work, which is kind of depressing. Even if you buy specialized hardware, yeah, you're not going to-- you're not going to find a block. It consolidates into-- you need a factory. You need a warehouse full of equipment, basically, in order to do this now.

So that's kind of annoying, right? It's not as fun. Like, the small players have been pushed out of the game because you need to do so much work in order to have-- you can't get, hey, I came close. Can I get partial credit? Can I-- you know, I did 2 to the 50 work. Can I get anything for that? The system doesn't recognize partial work.

OK. Other cons. It uses watts and chips, uses lots of electricity. You could use that electricity to charge your car or do something cool. It uses fabs to make microchips, which could make more CPUs or make cool phones or something. Once it gets big enough, it actually starts affecting markets.

So I don't know if-- I just know in Micro Center, down there, many times-- because I live near there. And you go around, and there's the GPU section, and everyone complains about Bitcoin in Micro Center near the GPUs. It's a Bitcoin. It's usually Ethereum and other coins that use GPUs for mining, but that has had a really big effect on the market for GPUs. A lot of people-- because you can make money with them. So people are like, yeah, I'll buy one of these things for $500, because I'll make back my $500 in a year. And then I'll still have the GPU, and maybe I can sell it used, and get a couple hundred dollars back, and then I profited. So these things affect-- it's big enough that it affects markets. Initially it didn't, because it was just a couple of nerds running Bitcoin, and so like, who cares? You know, a couple of people buying GPUs. But now it's big enough. These are like billion dollar markets, and it's getting to be the same sort of scale as the gamer market-- or gamer, or whatever-- CUDA, or whatever people use GPUs for.

But you know, mining is significant, and so it affects the market. Who knows? Someday you could start to affect electricity prices if you get big enough, where a significant portion of all the electrical usage in the world or the country is going into doing this sha256. And so that makes electricity prices go up. And then everyone starts complaining like, man, electricity is twice as much as it used to be because all these people mining Bitcoins.

And it's possible if it gets big enough, instead of just people complaining about GPUs. So that's sort of a con, right? That's-- it starts affecting the real world in big ways.

The corollary of it being a memory-less process is that it's irregular. It's a Poisson process, which means sometimes you get in a few seconds, and sometimes it takes an hour. And people don't like that, right? I can see how you'd rather have something come out regularly. A lot of people think, yeah, blocks come out every 10 minutes.

Well, on average, every 10 minutes, but it can be hours. It can be seconds. And so that's hard. In some cases, you can't really use it. Like, hey, I want a regular clock, or I want messages that come out in some kind of-- I want some assurance, right? If I make a transaction, I want it to be included in the system. I want some assurance it will be included in the next 20 minutes. You don't get that assurance, right? It can be 30 minutes before anything gets included. And in fact, we'll get into it later, but there's even worse assurance properties, in that just because five blocks came out doesn't mean your transaction got into any of them. It might have been ignored. It might-- there's all sorts of reasons.

So you can deal with this, but it is annoying, and it precludes some use cases. And other consensus systems do not have this problem. Other consensus systems can have much more regular clocks. So this is kind of annoying as well.

Any questions about the irregularity aspects? OK.

OK, big con. It's like 51% attacks. OK. Part of the problem with anonymity is you have no idea who's doing this stuff, right? Even if they say who-- even if they claim-- so one of the interesting parts of anonymity, despite this being designed as a really cool hacker-y, cyber punk, anonymous system, one of the first things people did was put their name in it, right?

So if you mine a block, you've got space where you can put 32 bytes or so that are undefined, and people would start putting the names of their mining pools or the names of their cat or whatever. They just put their names in it. So it's kind of like they're showing who they are, and

you can look at distribution of who are the different miners, and stuff like that, even though it's supposed to be anonymous.

So they claim-- but they can claim a name, and they can claim this block was mined by F2Pool, or this block was mined by BTC.com, you're not 100% sure, right? It's all-- I could also mine a block and claim that it's from F2Pool. And it's hard to disprove. So you don't really know who's mining, and an attacker with 51% of the total network power can write a chain faster than everyone else combined. So that means they can rewrite history, right?

So like in the example before, where hey, I made this three blocks longer than these two blocks, I had to either get lucky or everyone else stopped mining for a second, but that's not really possible, long term, if I have a minority of the hash power, right? Because everyone's just going to start-- you know, if the majority is honest and playing by the right rules, they're just going to see what the most, the longest worked-- the most worked, the longest chain is, and start throwing blocks on top of that.

And if I go back and try to make a branch, the main branch will grow faster than my sub branch, and so I'll never overtake the main branch, and so I'll never be able to reorg out a transaction.

However, if I am faster than everyone else put together, it's assured that I will eventually catch up and overtake the rest of the network, and then everyone will have to reorg out onto mine. So here's the main chain, and then I branch off here, and I'm faster, so I overtake them and then, that's sort of-- everyone's stuck, right? Even if they want to keep mining, I can just reorg them out every time, because I'm bigger.

This is called a 51% attack. It's pretty bad. An attacker can rewrite history, can undo transactions. They can't forge transactions, right? So they can't necessarily reorg out a message and replace it with a message of their choosing. All they can do is, you know, if they-- but they can do that with their own messages. So you need to have this as part of an attack where they reorg their own messages out.

By default, when a reorg occurs, so like-- I can-- go here. So by default, when a reorg occurs, like say, these two blocks of messages have been invalidated, software will attempt to include all the transactions in these blocks into the next blocks, right? It's not-- just because this got reorged out, doesn't mean it was invalid. It just means it didn't get into a block. And so we can try to include this message later, as long as nothing in here conflicts with it.

So just doing a 51% attack to reorg on its own doesn't actually do very much damage. However, it's not too hard to combine that with other attacks where whoever's doing this mining says, oh, I'll pay someone here, and then pay myself here. And then undo the payment that they made to someone.

So that's a big, big con. 51% attacks. Not only is it very disruptive to the system, it's very hard to predict when it will happen, since it's anonymous. It could be the case that all the people mining Bitcoin now are friends and they're a 51% group, and they could just get together and say, hey, let's reorg out these things.

That said, there is a cost to do so. Yeah?

AUDIENCE: There seems to be some requirement of momentum of messages to not able to trick the system. So if you're setting up a new currency, it must be much, much easier to work the chain.

TADGE DRYJA: Yeah. I mean, if there's very little work being done in the system, it's probably pretty cheap to get a bunch of hardware and be 51% of that system. True. So like-- and in the case of Bitcoin, if you have to 2 the 72 work, well, every 10 minutes, that means if I want to be 51%, I have to do 2 to the 71 work every 10 minutes, which-- that's a lot of work. And so you're going to need enormous amounts of resources to become a 51% attacker.

There's two ways this attack can happen. Either some attacker comes out of nowhere and says, OK, I'm just going to build up an enormous amount of attack power, or the existing actors in the system go bad, which seems more likely, where they're like, OK, we're already-- you know, I'm 5%. He's 10%. And she's 20%. And they all get together and try to do an attack. So both of those different ways are possible.

Yes?

AUDIENCE: But [INAUDIBLE] mining downsides? Like, we join some other pool?

TADGE DRYJA: Wait. Wait. Sorry. Who does?

AUDIENCE: Like, small players.

TADGE DRYJA: Yes.

AUDIENCE: So like, [INAUDIBLE] of [INAUDIBLE]. If the miner pool rollback-- like for example, I'm in a miner [INAUDIBLE], and the software is malicious, so one of the players can roll back simultaneously?

TADGE DRYJA: Yeah. So pools, we haven't mentioned pools, but the idea-- part of the con of you're never going to find a valid proof of work, is the way they mitigate that, is people get together in pools and they will prove partial work to each other, and there's one entity that's then-- controls a lot of different miners and gives out partial rewards.

So OK, there's a thousand different people mining, and one of them found the block, but I'll distribute that reward to all the thousand people for all the work they've done. And so that does concentrate power in that mining pool, whoever's running it. So that can also be a big problem. Yeah.

Because if you're doing it solo now. So solo mining is saying, I'm just mining on my own. I'm going to try to find a valid proof of work. That's very difficult, because of the amount of work needed. So many people pool together. So yeah, that's another issue. Yeah?

AUDIENCE: So [INAUDIBLE].

TADGE DRYJA: Yeah. There's also protocols, one called P2Pool, where you can have another layer of block chain-y kind of messages, which allows you to pool together resources without having a single entity responsible for getting the rewards and distributing them back up. I mean, I haven't really talked about the rewards. The Bitcoin-specific stuff will be in next week, probably.

But yeah. There's all sorts of-- this gets really complicated and crazy. And like, pools, there's so many attacks on pools, and it's a mess. OK.

Last con. People hate it. And this is not a quantitative, objective reason, but it is a problem. And people don't like proof of work. Some people are fine with it, but a lot of-- especially in academia. I've talked to a lot of people. They're just like, ah, this this is horrible. The whole point of sha256 is you can't find collisions, and you've designed an entire system where all we're doing is trying to find collisions in this non-collidable system, and you've got giant warehouses full-- uses so much electricity, and you've got giant warehouses full of chips that are just doing this pointless thing.

And it's-- you know, it offends the sensibilities of many people. And it probably is getting worse, because it keeps getting bigger, and so a lot of people don't like it. And I can understand that.

When I first read about this stuff, I was like, that's kind of cool. That's kind of stupid. And then I was like, oh, man what if this really takes off? There's just going to be so much power usage and so many chips dedicated to kind of a pointless thing.

I would say that I acknowledge this as a big problem, and I think the best analogy is something like gold or silver, precious metal mining where you could make very similar arguments, that all these you know Spanish or Portuguese people sailed over to South America 500 years ago, and it was sort of-- it was horrible and all these people were working in these mines, and it's a mess, just to get silver or gold or whatever.

And it's like, why? Why are you doing this, right? Just to get some gold? So similarly, the bitcoin and the proof of work in general has a lot of people who don't like it. And so that's why there's a lot of research in, OK, can we have the same kind of system, but without the work? Or alternatively, can we have some kind of proof of work that's also useful for other things? So can we have some kind of proof of work that cures cancer?

And it's like, well, I mean-- there's actually, right, papers about protein folding proof of work. And I haven't seen a ton-- there's one where you like find sequences of prime numbers, and it sort of worked. It sort of has most of the properties you want. And it's like, OK, but it's like, well. You found five prime numbers in a row. Like, eh. Yeah?

AUDIENCE: The fundamental problem with those [INAUDIBLE] that's different from this is generally the same amount of time to verify as they do to generate, whereas the advantage of this is it takes ages to generate, but it's really quick to verify.

TADGE DRYJA: Yeah. So that's another-- yeah. People want useful proof work. A lot of times with useful proofs of work, it's hard to tell that it's a valid proof, or there's not as much of a gap. Like, the gap in Bitcoin is enormous, right? O of n takes O of 1 to verify, which is perfect. Yeah, you can't get better than that.

So yeah, it's hard to verify. There's also-- that also applies to many different proofs of work. So one of the things with 51% attacks is if, like you were saying, if you want to build your own coin, if you use the same proof of work algorithm as Bitcoin, you are in a very risky situation, because you've started your own new coin that-- your own new message network that very few people are using, and there's very little work being done in this network.

And then there's this huge Bitcoin network. And as soon as they see your network, they could

easily overpower it, right? One actor, one person who's mining, could become more than 51% of your network, just with the flip of a-- pressing of a button. And so it's very dangerous to be running a very small network that shares a proof of work with the larger network.

And so most of the different coins, when people say, hey, I'm going to make my own derivative of Bitcoin or my own new coin, I'm going to have a different proof of work. And so there's all sorts of different hash. Maybe I'll use sha3 instead of sha256. OK. Or I'll use some other hash function, or some other usually iterative hash function, like for password hashing, where you actually end up hashing several million times.

And the disadvantage there, as well, is it can take quite a while to verify the work. Not too much, but noticeable, where if you're running different clients like Lightcoin, for example, it uses a proof of work which takes thousands of times longer to verify. Still fast enough, but some of them are pretty heavy.

So those are all the different-- you know, it gets pretty messy, and I'm-- ask-- probably James and some other people are much more familiar with all the horrible intricacies of proof of work.

AUDIENCE: [INAUDIBLE] if you back to the part about [INAUDIBLE], like back when Bitcoin was first made, it just spawned the longest chain, by absolute number or blocks.

TADGE DRYJA: Oh, yeah. Well, yeah.

AUDIENCE: [INAUDIBLE]

TADGE DRYJA: Yeah. So there's one-- wait. So I said the longest-- so if you say the longest chain is the valid one, I think I mostly said most work, right? They're not necessarily the same, right? In this case, where we have a fixed target, a fixed amount of work per block, they are the same.

But in the case of Bitcoin and most of these networks, the target and the amount of work required for a block can change with that 2016 period. And so there are attacks where you can say, well, I'm going to make something that's longer, but actually has less work, because I've forged all these--

You know, I go back in time. I pretend it took me a long time, and I changed the timestamps so it looks like it took a long time and the difficulty decreased, but I actually have more blocks, but less total work. And then I can have some weird attacks where I reorg out things.

So in the case of Bitcoin, I think-- yeah. Pretty early.

AUDIENCE: Like, version one point--

TADGE DRYJA: Version one point-- Version 0.1, it actually just looked at number of blocks, because it's so much easier in the code to just do number of blocks. Otherwise, you have to treat all the hashes as big ints and figure out how much work was done on each one, and this annoying stuff.

But that attack was thought of. I don't think the attack ever happened, but whoever designed it was like, oh, wait. Yeah. Longest is not quite right. Has to be most work. And so they changed the code.

OK. So proof of work. People hate it. Yeah, but the thing is, it does work. It's been working for nine years. The blocks keep on coming. Not regularly, but over the course of a day, about 144 will come out. In practice, it's infeasible to write old messages, right? There are all these known attacks, but they're very costly, and either you have to do an enormous amount of work and get all these resources, or you have to have a lot of collusion between the different actors.

One nice part is that in these systems, people are very adversarial, and so they don't want to collude, and they always are attacking each other and hate each other. So that keeps the system working. In practice, with Bitcoin, there are very few block reorgs. This happens-- I don't know. When's the last time an actual reorg happened, instead of just two? Like, years. This just never happens. Most, one or two blocks.

So you can have a one block where two blocks come out at the same time and point to the same parent, but that's not a reorg, right? Because then your software sees it, and it's like, well, which is going to be right? And then one of them pulls ahead. So that happens every couple days. It used to be more frequently, but now it's fairly infrequent.

And like, two blocks reorg sometimes. It has happened, but it's very rare now. So you can build on these things.

OK I'm almost done. We'll do one last fun fact, and then questions.

OK, so fun facts. How to estimate the total work done throughout history of the network. Well, you just look at the lowest hash. This is kind of counter-intuitive, like whoa, kind of thing. You can prove all the work ever done in the history of the system with just one hash, the lowest.

Because probabilistically, well, I've been doing work the whole time.

Probabilistically, I guess the way this works is that whether we were all trying for the same block or whether we're trying for all these different blocks, it doesn't matter, right? We're still doing attempts. And the best attempts will have the most zeros in front of it. And that holds true for whatever configuration, graph configuration, whether there's forks, whether there's a single chain, whether we all just work on the same thing and never make any progress and just admit the best hash.

So that's kind of cool. You can just look at a single hash and see all the work that's ever been done in the system. Kind of cool. With pretty crummy accuracy, of course, but actually not too bad.

And so there's a paper that's pretty interesting. If you're interested in it this and want to look into the math, there's a paper called HyperLogLog that uses this fact for something completely different, to count set cardinalities and stuff, and it's kind of a cool math paper. But I think the Bitcoin software does do that, and it'll say best hash. And it's in there, and you can say, like, well, this is how much work we all have done.

There's also some papers in Bitcoin where you can try to use these facts, like the lucky blocks that happen to have much more work than they needed, and you can look back through history and give an abridged version of history that still took as much-- that would still take about as much work to rewrite, without providing all of the different blocks and stuff.

You can also prove close calls. So with pools and mining, one way you can do it is you can prove that you were trying but didn't quite make it. So if the requirement is you need 2 to the 40 leading zeros, and you get 2 to the 39, you can show. Like, hey. I got 2 to the 39 and I didn't quite make it, but I tried.

And then people can verify that work as well, and credit you for it. So the way a mining pool works will be you give what's called shares, which is like close calls that were not sufficient to meet the proof of work, but you keep giving them every few seconds, or every few minutes. And then the central pool operator says, OK, well, this person did this many shares. This person did this many shares. I have-- the central operator has a very good estimation of how much work different people are doing, and so credits them. And said, OK, well, he's doing this much work. She's doing this much work.

And then when someone actually finds the block, distributes the rewards from the block proportionately to how much work everyone's been doing. So that's trusted, and not great.

And there's also a bunch of attacks. The sort of sneakiest of which is keep submitting shares, submitting close calls. But when you actually do find a valid proof work, immediately forget about it. Drop it, and never tell anyone. The reason that's a good attack is fairly counter-intuitive, but it's a good attack, even though it seems like, well, you just found the block. Tell everyone. You get this reward. You get--

It's like no. No, I found a block. Throw it away. But oh, I got a close call. OK, keep telling the central operator. What that does is that really screws over the central operator, who now thinks you're doing a lot of work, but you never find an actual block. And eventually, the central operator will be like, this person is really unlucky, huh? He's found so many close calls, but never seems to actually get over the line and find a valid proof work. And so maybe they can kick you off, eventually, but it's not something you can prove. You're like, well, I got unlucky.

So there's all sorts of things. There's all sorts of fun things you can do with the proof of work and just these hash functions.