PROFESSOR: OK. So yeah, problem set 2. There's a lot of work done. Congratulations, all the workers. 16 trillion hashes performed. How can we prove that? So this is a personal gripe that I hear a lot. That people say that proof of work doesn't scale. And that really bugs me because sometimes I think I sort of know what they're talking about, and they mean like, bitcoin doesn't scale, or like, these block chains have poor scalability properties, which sure, sure. They definitely do.

But proof of work, it scales perfectly, in a theoretical sense. There's nothing that can scale better. You can prove an arbitrary amount of work in 0 of 1. Right? So in this case, well, how big were these headers? They were less than 100 bytes, right? 100 characters.

And with that space, you can prove actually the entire work that happened over the entire problem set. So yeah. block chains, whole bunch of scalability problems. There's complex systems, all sorts of scaling issues. But proven work itself, in this pure form, scales really great.

OK. So question. Not super intuitive, but how do you prove all the work ever done throughout the entire problem set in one line? Does anyone have any intuition about this how do you prove all the work from all 1,800 blocks with just one piece of data?

AUDIENCE: Well you know that for each block, 2 to 33, work had to go into it. So you just need to know the number of blocks produced times the--

PROFESSOR: OK. Yeah, so the thing is how do I prove the number of blocks without showing all of them, right? So OK, it's a weird trick question. Andrew-- I think-- I remember Andrew Miller, who's now a professor at somewhere, Cornell? Who was not, during the time, wrote about this initially in the bitcoin forums.

What you do is you just show the luckiest block, and I have not yet defined luckiest block. But in this case, it was mined by turtle. This is the block, the previous block reference was of 0065a2 turtle 1654244 and the hash of that block is 000c49a414 blah, blah, blah. So anything interesting or novel about this particular block that you can see? It's not--

AUDIENCE: [INAUDIBLE]

PROFESSOR: What?

AUDIENCE: It's better than...

PROFESSOR: It's better. There's more work. So we didn't count the things as having more or less work just by a sum number of zeros. We just looked at the threshold, did it have enough 0 bits and accepted or rejected. But in this case, these 4 bytes, right? You needed 4 bytes and then 1 extra bit. You needed 33 bits. So these 4 are always worth 0. But then in c and red, there's another extra byte that's all 0s. And another extra half a byte that's all 0s. And then a c means for that nibble, that the highest bit was 1, right?

So you've got this red stuff. There's a byte and a half extra-- or almost a byte and a half extra. So, yeah. So what you can do for a compact proof work. So if you look at this again, there's 4 green bytes, byte and a half is red, right?

So that's 5 and 1/2 bytes, so 44 bits. And 2 to the 44 is 17 trillion, right, if you do out 17 something, which is what we expect, that's our proof, right? We did 16 trillion hashes for our calculation, and it shows up here. And another way to look at it is we needed 33 bits for a valid block. We have 44 bits here. That's 11 bits extra. That's 11 bits of being lucky.

And so that's 11 bits to the 11, that's 2,048, which is pretty close to the 1,862 that we actually performed, right? So in this case, we're a little lucky. We might have only had 2 of the 10 and then we could only prove that we'd done like 8 trillion work instead of 16 trillion work. So there's probabilities here, but this is an interesting property that actually does work.

You can prove all the work, to some approximation usually within a factor of 2, that the system has ever done just with 1 block header. So, yeah. This is fun because another way to look at it is that you've got this metagame, where for every block found, take the, I'm doing a hash and I need to find a hash with a lot of 0s to prove that I've done work.

And you go a level deeper and say, OK, I'm finding a block. And I want to prove that I found an even better block than everyone else, right? The entry for admission here is find a valid block. And then from those blocks, since it's a uniform distribution with 1s and 0s, you're going to have this tail-end of like, happened to have lots of 0s that you didn't need in the beginning.

And that can prove all the work ever done. There's a really interesting paper called, HyperLogLog, that uses this for non-bitcoin applications that uses it for set counting. Where, like on a website, you want to see how many unique visitors you've gotten or something. And you can store that in 0 of 1 space because you could keep track of OK, let me keep track of every IP address that's ever visited or something like that, or cookie.

But instead, you hash them. See if the hash starts with a bunch of 0s or any arbitrary character, and then just store the lowest. And then, every time someone visits, hash it, compare. If it's lower, replace. If not, ignore. And then you have a very compact indication of how many visitors have visited.

Anyway, that's like a super high-level view of it. But if you're interested in this stuff, HyperLogLog is a paper. It builds off of some other things. It has nothing to do with bitcoin, other than this property, where you've got this random function and you see how many 0s are in it. But I think these are cool, and so to me, this is a fun-- this is not used in bitcoin, right?

In bitcoin, you actually download all the headers, but people have written papers about how you could use it. If long term the headers are big, you could prove. Have some checkpoint, where look, I proved all the previous work and now I build from there. Any questions about this idea? Yes?

AUDIENCE: It's not really a proof. It's just a probability weighted.

PROFESSOR: Yes, but the proof of work itself is the same. Not of real proof because you might have gotten lucky. So finding a block, it's like, well that's not a proof. There could be luck involved, there could be probability, but it's the exact same luck and probability that the underlying proof of work uses. So there's no further reduction in security or certainty because of this. Not really.

And so I remember talking about this with someone a few years ago and saying, yeah proof of work is a misnomer. It's not really a proof, right? Maybe it's an argument of work or some probabilistic argument of work. How could you make it more certain as a proof? There's a bunch of ways. One way would be to have multiple nonces, where instead of just finding one nonce that satisfies it, you have to find several replaceable nonces and then iterate through them.

That would be a much more certain proof. It would remove the idea of probability, to some extent. It would also completely break the system in a way that's like fairly unintuitive, and so I always was sort of joking like, I should make an altcoin, where you've got multiple nonces, and you could be like, yes, for more security and then people would buy it, but it completely breaks.

The completely broken incentives and system, maybe I'll get to-- I'll let you guys think about it til Wednesday, and then draw it out and be like, why wouldn't this work? It's fun. It breaks in subtle but bad ways. This is talking about proof of work optimization. So if you look at this slide anyway, you've got these headers or blocks or whatever we're mining. So you've got kezike17, tomriddle, Thalita, all these people mining.

It's interesting to see what people use. Some people use looks like base64, some people just use a decimal number, all sorts of different things. Who knows. There was also a lot of invalid blocks with valid work submitted, where there was four or five different things in my spaces. So there's been count, but actually a lot more work was done, and that's not even counting the human work of doing all these assignments.

So sending this over the wire, or storing it on disk, some inefficiencies may jump out at you. What do you think you could do to make this more efficient or compress it or? Yeah?

AUDIENCE: [INAUDIBLE]

PROFESSOR: OK. That's an easy one. Oh, this doesn't work when I gave you the slides because you can just look at the next slide. Whoops, OK. Never mind. Yeah, so the first 8 characters are always going to be 0s by definition. If it's not, it's an invalid block so don't even bother sending it. So the first characters are 0, don't send them over the wire. Just have that implied and just start at the ninth character. That saves, well, really you'd have the serializing binary, so it only saves 4 bytes. In our case, it would say 8 bytes. That's cool.

And then, also the entire previous hash. When you're sending a list of 5 here, like here, in order for it to be valid, this has to be the hash of the line above it. So just don't send the whole thing, right? Just send name nonce. The hash is also computable from anyone receiving it. That takes almost all of this space and we could compress this entire blockchain to just the first line, and that's the only full line we need. After that, it's just named nonce named nonce, and we get rid of 70-something percent of this base. That's pretty cool, right?

Yeah this kind of header optimization is also very much possible in bitcoin, and it's not implemented. It's not done. If you want to, you could program it, change bitcoin, make a pull request. I think-- I mean, we've discussed it, and people are like, yeah that'd be cool but, no one's done it. So if you want to leave your mark and be like, I'm a bitcoin core contributor, and I optimized the proof of work propagation system or something sounds cool, you can do this.

It's not too hard. You got to learn how bitcoin works and all the different messages and stuff, so it's a little annoying. But I think the main reason people haven't done is because it's-- this is not the slow part, right. This is not the critical path, not a bottleneck in the actual system. Generally, the proof of work verification is pretty quick. The headers are a total of 40 something megabytes now, 50 megabytes maybe.

So you could you could definitely reduce that by a significant extent, but no one's bothered because there's so many other scaling issues that are more pressing. But it's kind of cool, I think. So, yeah. That'd be a fun thing to do. If you did that would that be a soft or hard fork? If you said, OK, I'm going to now send header messages that are truncated. I'm going to leave off the 4 bytes that are always 0s. Bitcoin is also the first-- the difficulty requirement here is basically the same as it is in bitcoin.

I'm going to have the implied previous block hash, things like that. Would that be a fork? Actually, it wouldn't, right? It's a non-fork. There are lots of changes you can make. So I know Neha talked about soft forks and hard forks changes you can make in the system that affect consensus, but there's a lot of changes you can make that optimize it that don't really affect other people.

So in this case, it would just be a wire protocol change and you could easily maintain backwards compatibility right? So in this case, you say, the header optimization is not a fork, right. What you do is you'd have a new message type like, truncated header or something, and then, when you connect to nodes you say, hey, you know about this new message type I'm using? And if they don't know what you're talking about or they usually they say what version they are when they connect. You're like, oh you're an old version, you don't know about this. I'll just keep saying the old header type.

And even if I store the new truncated headers on disk, I can recreate the old one pretty quickly by performing the hashtag and then on sending it to you. So I can be backwards compatible and forwards compatible. No soft forks needed. The old nodes, they don't even see that this happens. They might see that there's oh, there's a new version or a new message I'm not aware of. They ignore it. Everything seems fine. So these are the easiest-- they're not forced--

the easiest changes in the system get through because there's no real coordination needed and it's backwards and forwards compatible, so that's cool.

So some example non-forks. A lot of them are internal only. You can't even see from outside. So for example, compressing blocks or compressing your database. That's fairly straightforward, right? Intuitively it seems like, well, these are all random numbers and hashes. You can't really compress those because they're random. In practice, you actually can. People reuse public keys a lot, and so you just see the same pub key over and over. So you do some pretty simple encoding and you can make those smaller.

Also, the amounts is 8 bytes. So if you're sending someone one bitcoin, that's 100 million. And people like to use round numbers, and so those get compressed pretty well. And generally, they're much smaller. So the top bytes are usually 0s. So you can compress it a decent amount.

But no one has to know that you're compressing, right? That's all transparent. When someone connects to you, they have no idea if you're compressing or not on disk. Something like faster signature verification, where there's been enormous amounts of work in optimizing the code for that. Making assembly, stuff like that. Nobody knows you're doing it, they're just like, oh, he's asking for blocks quicker than this other person. Maybe his network's faster, maybe his CPU faster.

So these are changes that are purely internal. Nobody needs to know. That's cool. Other non-forks are peer-to-peer non-forks. So the truncated headers, maybe, where you can say, hey, I'm going to send you less data over the network. You identify at connect time and you default to the old behavior. People don't know what you're talking about.

So there's one called compact blocks. I didn't describe it, but you can probably guess what the idea of compact blocks is. Anyone want to venture a guess what those do? Block. So it's not a header that's come back, but the whole block. How would you compact a block?

AUDIENCE: Get rid of all the fields that aren't necessary. Like version...

PROFESSOR: Yeah, actually, that would work. But that's not what they do. There's a really big 2x redundancy. And so the basic idea is transactions are propagated, and then a block's propagated. Where's the redundancy there?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yes, the transactions the block. You've Probably already seen them, right? You see the transactions, and the block comes out. Most of it, in general, 90-something percent, it's like, yeah we're going to see all this. So compact blocks is a way to say, hey, here is the block. Here are all the transactions in it, but I don't show the whole transaction. I just show the TXID the hashes.

And then you can say, OK. 90% of those I've already seen so we're good. Here's these 50 transactions I have not seen, please give them to me. So it's interactive here's the blocks with just the transaction identifiers. OK, what do you need? OK, I need these 10. OK, here's the 10, and now I can reconstruct the whole block.

So the block goes from being a megabyte over the wire to something like 10 kilobytes? But it is a little slower in that it's like a multi round thing, right. It's like, here's the compact block, OK, I need these extra things, OK, here's the extra things. So a little bit more complexity.

If you're really optimizing for latency, then you don't want to use this. But in general, it's a pretty big gain in terms of bandwidth, which can be taxing on full nodes. I run a-- there's a full node on the first floor in one little rack and it uploads three terabytes a month or so. Depends on how much people are using bitcoin. In December, everyone starts downloading it and installing it, and there's a lot of bandwidth needed to sync people up.

Another non-fork was the Bloom filters, which note full nodes can then say, hey, I will perform Bloom filter calculations for you. And light nodes can connect in, like I said two weeks ago with SPV. Light nodes can submit a Bloom filter say, hey, when I download a block from you, first, filter the block. Match all the transactions against this Bloom filter and only send me things that match. That's not a fork, but it's a fairly involved change in the peer-to-peer code.

OK, any questions about these peer-to-peer non-forks? Cool. There's another aspect called standardness, where you haven't soft forked something out, you haven't declared something invalid, but you can declare it non-standard.

And what that means, is when you're node sees a transaction coming over unconfirmed, the transaction being propagated through the network. And it's got this property, and you say, oh, that's non-standard. I'm going to drop it, I'm going to ignore it. I won't propagate it onto my peers. I won't ban, I don't-- it depends. I don't know. Do I ban the person submitting it to me? I think you don't, but I ignore it. I don't propagate it, so it doesn't really get around the network.

When most of the peers on the network have these rules of non-standardness it's going to be very difficult to get your transaction out there. However, if you see this non-standard transaction in a block, you accept the block. You say, OK, well that was this weird thing that I didn't like, but since it's in a block and someone did a lot of work on it, I will accept it.

It's a little weird, right? Why have this? It's something that's not quite a soft fork, right? It's showing that we're discouraging this, we think it's non-standard. The miners software, by default, will also consider this non-standard and not mine it. But if someone else is mining it, we're OK with it. And so what you can do, is you can stage future soft forks this way, right.

So for example, in SegWit, oh, I didn't talk about SegWit at all. I'm going to have to do that next class, next week. OK, so SegWit was the biggest soft fork ever in bitcoin, and it occurred last year. It changed the output scripts to say-- so before, you said, OP_DUP, OP_HASH160, the hash, OP_CHECKS, OP_EQUAL, whatever.

Here, it just says 0. Just pushes a 0 byte, and then pubkey hash, and that's it. And if you actually interpret that in the stack, no signature is needed, right? You push a 0 to the bottom the stack, you push a pubkey hash on top of that, and then your execution halts, and you're like, well, there's a non-zero piece of data on the top. I interpret non-zero data as true, same way he does, so it's true. You don't need a signature at all.

So that's the weird SegWit soft fork where they said, no, what used to be considered true without a signature, we now template and we say, this means check pubkey hash, right? This means the same as OP_DUP, OP_HASH160, hash, OP_EQUALS, OP_CHECKS, OP_CHECKSIG, right. So what you actually do, is you need to provide the pubkey that this hashes into, and then check a signature.

It also defined 1, 2, 3, up to 16, and left this undefined, and said, look, these are now non-standard. Before, if you-- I think they were already non-standard, but the idea is that if you just push a 1 on the stack, and then push some data, well, I guess no signatures needed. But now they're non-standard because it means we're going to use these next. The next soft work will define what one, some piece of data means.

Maybe it's a new signature scheme, maybe it's a new program where you put some data here, but it's non-standard. So if you try to make a transaction that's using 2 and then a data push, all the nodes will be like, yeah, I'm not ready for that. I haven't I haven't seen that. And if you see, I think in your air logs, if you see a block with a bunch of these kinds of things, it'll give you a warning. It's like, warning. People are using stuff that your software doesn't know about. You might need to upgrade.

There's a bunch of warnings like that where like, warning some percentage of the last few blocks had these things, so people are doing stuff that you're not considering invalid, right? You're not going to refuse the block, but you're also like, this is something I don't understand and I've specifically coded it as nonstandard.

OK so any questions about non-standardness? Neha talked about soft forks and hard forks, and I will go through a bit more detail about how these end up working and how these interact with miners and notes. Did people have questions about soft forks and hard forks before we start? Sort of got the general idea, right? Soft forks add new rules, hard forks remove rules, in general.

And this is minors. So the miners have a unique role here. It's not just the same as a full node. A miner decides what to put into a block that they're mining. And so they do have a bit more influence in this fork decisions. OK, so a soft forks would be, for example, saying, OK, all output amounts must be odd, right. You can't send an even number of coins to anyone anymore. That would be a weird, silly fork. Wouldn't really impact the usability system, but it'd be dumb, but you could do it.

And you could say, OK, well, if I see a block, if I see a transaction, which outputs, if any of the outputs have an even number of Satoshis, invalid. You've got to do odd. Potentially leading to the loss of 1 Satoshi per transaction-- with the fees, 1 Satoshi per block may end up being lost due to this. So here I'm saying, an A for adopter and I for ignorer.

Now people may ignore the fork because they disagree with it. They don't want to do this fork, or they may do it because they may just not even know that this software exists. It's a giant decentralized system, and it's hard to know how to communicate with everyone, right? There is bitcoin.org. There's also bitcoin.com, where the guy doesn't like the bitcoin developers, and says they're bad. Anyone can just register these things. There's a bitcoin Twitter account that was purchased recently by someone who wanted to argue about these things.

So there's no one really in charge of this. And then there's also different implementations. There is the real bitcoin, which is run by this bunch of crazy people who say they control bitcoin, and that everyone has to pay them taxes in bitcoins, yeah. But they're all running bitcoin-- they all are in consensus and doing these transactions. So ignoring could be any number of things.

If you have a soft fork, where you say, OK, we're now adding this rule, but none of the miners are enforcing it. None of the miners even know about it, potentially. Here, it just stops, right? You say, no, I require that all output amounts are odd. And then every block has these even amounts. And you're just like, OK, no that's not a valid block, that's not a valid block, you will never see a valid block again, right? None of the miners are enforcing this rule, but you are.

Everyone's ignoring it, and they say, everyone's ignoring it. Everything seems fine. You just self-imposed this new rule, making you incompatible with the rest of the network, and from your perspective, everything stops and no more blocks occur and the system is over.

Or potentially, if you're soft fork is some weird rule that nobody knows about and nobody breaks anyway, we say, OK, the sum of the outputs of all-- the sum of the outputs in a transaction must not be a Carmichael number. OK, you could have that rule. Probably no one's break-- wait. There's a lot of small-- something like that, right? Where no one's breaking it anyway. Then, from your perspective, everything's cool because everyone's already obeying your rule even though they don't know about it, it's silly.

Another possibility is let's say a minority, somewhere 1 to 50% of the miners adopt this rule and say, yeah, we're going to enforce this new rule, right? All output amounts need to be odd, so the idea is you say, yes, only odd numbers. And a bunch of the majority, actually, of people don't care about odd or even. So the majority, they still go off on their own chain, but you split off into your own faction, you say, no we're the odd bunch. And both of those chains are viable.

Blocks come out here maybe quite slowly, if it's only a few percent. Blocks still come out here. The fact, so you've got these odd thing, and then you've got regular. And the regular is going to be longer, right? The regular is going to be potentially a lot longer, but from the people here, they're like, we don't care if it's longer. It's wrong. They use even numbers, that's just plain, old wrong.

And these people are like, yeah we can sometimes see it, but actually we lose track of it very quickly. After here, we start seeing block advertisements like this, and we're like, we're over here now. We're way past that. Why are you talking about this stuff from like weeks ago? So they just disconnect. It's pretty ugly, but that can happen.

Now, if you have the majority of the hash power, this ends up being longer, the odd blocks end up being longer, and everyone gets dragged along, right? No split, and now we have a new rule even though they didn't know about the rule potentially. So they're like, what the heck, half my transactions don't work. Some of them do, Some of them don't, I don't know what's going on. I just randomly adjust my fees until it seems to work, and then my transactions go through.

They should probably find out from someone, oh, yeah, there's a new rule. Only odd numbers, and then they-- that rule is imposed on them from the miners, essentially, in the rest of the network. And essentially, the same thing here. When you get to 100%, there's none of these orphans, but it worked-- oh, sorry-- these orphans would actually be like this because everyone agrees that this is valid, and then some of the people aren't aware of the rule and keep mining off of these what they consider valid blocks. So it's a little different topology.

OK, so that makes sense, right? Any questions about soft fork, mining power rules? One other aspect, is if you split here, and then later on you get a majority and you pull ahead, you will reorg out the ignoring side, so we split off with 10% of the hash power. We've got our much shorter chain, where we only have odd numbers. At some point, we convinced the rest of the miners, that, no, this is the way to go. The even numbers are really screwing up the system. And we get the majority of the hash power, and then we overtake the even and odd mix chain.

The people who have not yet updated their software, and are ignoring the fork, they will reorg out because from their perspective, OK, I was on a longer chain now there's this other longer chain. They both look valid, and when I see two valid chains, my way to decide is who has the most work? And so this one pulled ahead, in terms of work, so I switched. So it's a weird-- this has never really happened, that I'm aware of, they were threatening to do it last summer [INAUDIBLE] I don't know.

So yeah there is all sorts of stuff on the internet, and Reddit, and Twitter about doing this with a minority of hash power. They didn't though, or they did. They say they did, but everyone else says they didn't though. A lot of arguing. OK. So that's another weird aspect of it.

OK, hard forks. No minor support. What happens to those adopting it? Nothing, right? Everything just keeps working. If you say, OK, we're now going to allow every transaction output. Every transaction can have 1 extra Satoshi gets created. It's just 1, it's no big deal. It's quite limited, but we want to compensate people for using bitcoin, so when you have your inputs, you have your outputs, you can add 1 Satoshi. You get a free Satoshi per transaction.

The previous software, absolutely does not allow that, right? If you're just generating money out of nowhere in these transactions, not OK. But these guys, they'll see that the transactions they do that with are not confirming, but otherwise, they're OK if you don't add a Satoshi. And so the system works. Nothing happens, nothing happens. They just see everything.

With a minority of the hash power, something like 10%, 20%, you get all these orphans, get all these dead ends, where you see a block, OK, and it's got this 1 Satoshi per transaction bonus. Great, but it keeps getting orphaned out because you still consider-- all right, so you see OK, here's this longest chain without the bonus.

And then you say, oh, here's a block with the bonus, cool. Maybe someone builds 2, great. But this keeps getting longer and you keep trying, but you keep getting overpowered because you see both of them as valid. You're not requiring that there's this 1 Satoshi bonus per transaction, you're just allowing it. And so you say, oh, this is cool. Cool, oh, no. Got reorged out, got reorged out. So you see all these little starts, they get reorged out. And you basically stay with the same chain. You don't split.

These people also see a bunch of invalid blocks, right? You'll see it on the network, hey, someone keeps sending these invalid blocks much more frequently than usually. I don't know why they're doing that, but they've got invalid transactions, I ignore them. Here, majority of hash power is split, so once the majority and these are the bonus, the plus ones, they pull out ahead.

These guys don't actually care that they have a majority. After the first block, they don't see the rest because they ban. So if someone submits to you an invalid block, you ban them. You ban their IP address for 24 hours or something. You're like, I don't know what you're doing, you're crazy. Disconnect. So you won't really see this pretty quickly.

These guys don't have the bonus. They're still on their same old blockchain. It may be much slower because a lot of the harsh power now moved to this other chain, and these guys say, oh, it worked, cool. We've got our new bonus coin chain. Now we're stimulating the economy, everything like that.

Job creation. OK, so and then these guys slowers. Slows down. And then if you have 100%, well stops. For the non-adopters, no more blocks come out. This is the end. Everyone's gone to the job creation train. And there's not really a split anymore, it's just the new rule. OK. So any questions there? This grid so far?

Then another way you can do it is combine this to say OK we're going to do a soft fork and a hard fork at the same time, and actually, many times that people say hard fork, they actually mean this. So the nomenclature is pretty ambiguous. I like keeping these terms very distinct and pure, so like a soft fork is purely increasing the number of rules, where it must be odd, and a hard fork is just reducing rules. And yes, we will allow but not require a transaction to have this property.

And then to combine them would be something like saying, we allow this new thing that was not allowed before and we require it to be true. So for example, every transaction must introduce 1 new Satoshi bonus. That would be both hard and soft work because now if you're doing this, you no longer consider the old rules appropriate. And there's a complete mutual disagreement on the rules.

Some people have called this full fork, who called it that? I forget. Greg called it a bilateral hard fork. We don't have good terms for these things. A lot of people refer to forks that have both as hard forks, so it's somewhat ambiguous. I think it helps to keep these different terms, but it's a different setup. It's the union of these two things in some way in that, we allow this new thing that was prohibited before, and not only that, but we require it.

So if 0% enforce the new hard fork, well the adopters, it just stops, right? There requiring this new thing, it's not showing up, it ends. These guys nothing happens, right? When there's a minority, it will split off. It'll split off with the new rule set and the new bonus coins or whatever. And the ignorers, they see that it's slower because some people have left some mining powers left.

When you have a majority, you also split off. And ignorers, again, it's slow. They're not going to adopt because they see the new fork is invalid. In this case, as well, these guys won't adopt here because they see it as invalid. And then for the full thing-- for the 100%, the adopters, new rule, and these guys system halts.

OK so any questions about full fork or bilateral fork, or whatever you want to call it. Hey, it works. OK, cool. Yes. Bilateral, hard, full, we don't know the good names for these. But yeah. It's essentially a soft fork and a hard fork coupled together. And this is much simpler or easier to produce, and if you just start changing the code, you're probably going to create one of these, right?

You have to be very careful. If I want just a hard fork, I'm very careful that everything that used to be valid is still valid and I'm just rescinding one rule or one set of rules. It's very easy to inadvertently create this when you're trying to make consensus changes.

Yeah, there have been-- did Neha talk about the 2013 fork? I don't think so. OK, so a little anecdote. I remember I was in the airport. I got to Nagoya airport and opened my laptop and bitcoin went down to like $20 from $30, and it was all over the internet like, oh, no. And there was a inadvertent hard fork due to the Berkeley DB to level DB transition in the software.

AUDIENCE: [INAUDIBLE]

PROFESSOR: No, no, no. It was 0.7 was using Berkeley-- it used to be everything was using Berkeley DB for the UTXO set in the blocks. And then they switched to level DB, and then, it was OK for a month or two, and then someone wanted like a block that had a bunch of inputs or something. I don't remember exactly the reason. And the new software was like, yeah, that's cool, and the old software said it wasn't OK.

The thing is it wasn't clear why, right? There was no defined like, this should be invalid, like, this looks valid. But the Berkeley DB layer gave an error. And so it's like, well, the database says it's bad, so it's not a good block. So that was a weird unintentional consensus change.

What happened was-- so it seemed like it was a hard fork. The thing is it was like a compiler time option dependent hard fork, if you compiled it with a different Berkeley DB cache setting, then it would work. It was sort of ambiguous and people rolled back though.

They were on IRC and they were talking to the different miners and are like, what's going on? There's two different forks being built. And they told the people who were, to some extent, in the right, the new version, which seemed more correct, to stop mining. And they did, and then the old version with the Berkeley DB caught up, and then they restricted their block size. It was something to do with like having too many file locks open or something like that.

So that was essentially a hard fork. They stopped it, and then went back so there wasn't a hard fork, but then months later they're like, OK, we're going to all transition to level DB because we're not even sure what the rules are. Yeah?

AUDIENCE: [INAUDIBLE] the number of blocks update they produce block was way too many, so level DB was like, OK, [INAUDIBLE]

PROFESSOR: Yeah. And the thing is, they definitely did-- at the time, people were running around screaming like, why-- after the fact, they're like, that's why. But at the time, people were talking really quick and like, what's going on? Bitcoin has failed, sell all your bitcoins, the system doesn't work because no one really knew what was going on or why some versions weren't working. So yeah, that was an unintended fork. It was a little scary, and then the price went back up after that, It was like, hey, we can work through this guys.

So that was a hardcore, right? The old software would not allow these blocks that opened a bunch of file locks, and the new software did. And so that was probably the one real hard fork that bitcoin has been through. There were a few maybe in 2009 that like are fairly ambiguous because there were no there was no actual split. I think there's also a hard fork that's happened, but it didn't, it's a little weird. It has to do with the timestamp in the block header and how it like expires in 2106, once you run out of bits from Unix time. 1970 plus 2 to the 32 seconds is like 2,106 in January, or something.

And so they actually did a hard fork, but the thing is the hard fork wouldn't diverge until 100 years from now. So it's like whatever, everyone will have updated by then. If someone's still running software from 2015 in 2106, they will diverge, but that seems unlikely. So there's a lot of weird stuff like that.

OK. Firm variance. Some people call it firm fork, people call it evil fork, I'm [INAUDIBLE] call it evil and firm, I don't know. This is a fun one that has not been attempted, and I remember talking to people and they're like, don't talk about this. The miners don't know they can do this, so just shh. I'm like, come on, they don't know they could do this? That's kind of like--

OK. So how would you do this? Make a soft fork. It's a hard fork, right? It completely changes the rules, however, it looks like a soft fork to non-adopting notes. It's kind of crazy. Well, the proof of work for the new chain is a empty but valid block on the old chain. So instead of your proof of work being, hey, have a header that hashes to this, say, have a header that hashes this. Also a block that is valid but completely empty. We'll take that as our header, right? Our header now gets bigger. Instead of 80 bytes, it's going to be 200 something bytes, but that's doable.

Now our header chain is a chain of empty blocks. And our actual blocks point to that, right. We can put our Merkle root in the output address or something. We can put our Merkle root somewhere in this empty block transaction-- the one transaction in the block. The old nodes will see it and say, yep, that's a block. Someone's mining, here's where the money went. But there's no transactions in it. My transactions never confirm.

You don't actually have to connect to the old network to do this, it's totally deterministic. You just say, OK, my new proof of work is a valid but empty block on the old one, and I commit somewhere in this to my new block. That's evil because what happens is, here's the chart for this. It's basically a firm fork plus this little evil thing. The adopting, if you have no hash firewall system holds, then nothing changes here.

If you have 1 to 50%, the adopting split off with the new rule, so in this case, it's like a hard fork. However, the main difference from a hard fork, if you have majority hash power, the ignoring the system essentially halts. It doesn't halt in that the blocks keep coming out, right. You'll still see your software won't give you any warnings. It'll just be like, yep, block height keeps progressing as normal, as expected. We keep selling all these blocks, however, no transactions occur. And you can never receive or send money, but according to your software, everything's working fine.

So this is the firm, evil, sneaky, whatever part, where if you're able to get 51% of the mining and you implement this new fork, you're basically forcing everyone to update because if you don't update your software, if you're ignoring this fork, you can't do anything. You've got to adopt a new rule.

So this is scary. And I can see why some people are like, don't tell miners they can do this. Also I remember last year with SegWit2x stuff, I think, they were arguing about this. And it really seemed that they were not aware of this possibility, which was like, huh, they don't know about this. Cool, I guess that helps keep things safer because they were arguing about how they were going to rent like hundreds of millions of dollars of hash power to mine empty blocks on the old chain. It's like, you know you can do that for free if you just change your software to make your new proof of work and empty block on the old proof work, but I don't think they were aware of that, so we're like, OK let them go.

Anyway, and so the thing is it you can see how it would really quickly turn into that, right? If You've got 75%, well why would anyone try to mine on this? You could keep making blocks that did contain transactions, but they would get orphaned out. And everyone would, you might occasionally see, hey, a block came out with transactions in it, I just got reorged out, and nothing would ever come of it.

So you can see how the miners can never really get paid on the minority fork and minority chain, so they're all going to start switching to the new thing, if they want to get paid. So that's a little ugly. This has not happened yet. Hopefully, this doesn't happen, but on the other hand, despite it being called evil, I know Luke Jr, he's kind of crazy, but he's cool. He was saying this is how we should do a hard fork, right? We should also give people the option to say, look, there's going to be a fork. We give you the option to adopt a different system or doing this so that no one's inadvertently left behind.

So we say, hey, a year from now, this is going to happen. You can either say we actively refuse this new rule and we've got our own chain now. We make a soft fork before that point, or we adopt this new rule, update our software, and now I've got this new proof of work.

The thing is, the new proof of work, you don't need new chips, right? You just need to do a little bit different software. OK any questions about evil forks? Kind of fun. Yeah?

AUDIENCE: [INAUDIBLE] an empty block?

PROFESSOR: Well, OK every block has to have one transaction, so you'd have the coinbase transaction, but any user created transactions would not be. And, yeah, in the case where there's only one transaction the Merkle root just becomes the TXID of that single transaction, and you can put arbitrary data in that single transaction. The input field-- yeah we said-- the input field for that coinbase transaction that generates new coins can be any arbitrary data you want.

So you could put a Merkle root from your real block in there, and then write your new software and say, OK, the new proof of work is the header with the nonce, also it's got to have a coinbase transaction, and then a real Merkle root in the coinbase transaction, and then build out your treat from there.

So it's a little ugly. It's a little more complex, it would totally work though. And all the old software would just be like, huh, no transactions. And all the new software knows that that's not a real coinbase transaction. It's just a part of the extended header. OK. Cool. Don't try-- well, I mean, try this at home, if you want. I don't know. Yeah, seems coercive, thus evil, people call it.

OK, yeah, fork coordination. How do you go a level up from this. How do we know to do all these things? Reddit, IRC, Twitter, there's-- these systems exist on the real world and people talk. Like the meeting I was at last week. We actually didn't talk about forks much, we sort of did, but the developers get together, companies using bitcoin, all sorts of stuff.

Yeah, no, on Wednesday, people were arguing about MAST versus Schnorr which is more important, which we should try to soft fork in first. Not much gets done. So it used to be called BIP9. That's still there. Bitcoin Improvement Protocol 9. It was the idea in the header field in the version field, you can set these flag bits for which soft forks you are adopting, right so you indicate before adopting a fork which one you're ready to adopt. And then you don't actually implement the adoption until you see some threshold to say, OK, once 95% of people are signaling that we have this new operation, we'll all enforce it.

Because quite likely the software people don't want this. A lot of times you say, look, I want this new rule. I want this new signature system or I want this new even, odd thing. It would be really better if everyone only used on numbers. However, I'm not willing to split off because of this, right. I want everyone on board, or at least the majority on board. So we have a new split. We get the new one, this is what I want. I like the rule but I'm not willing to put a stake in the ground say, look, I'm making my own network if you don't like it.

So in order to do that, we say we want to get a majority of mining power to adopt it before we start doing it. And so that way we can signal in the header that, hey, I'm aware of this new rule, and I will enforce it if everyone else says they're going to enforce it so a staging process. So that was called BIT9, and the idea is once 95% are signaling it, then you activate it.

This didn't actually work in practice. I think it worked once with the OP_CHECKS sequence verify, and then last year, people were just arguing in the miners are like, no we're not going to activate any new soft forks, or then, they started making all these deals-- it was a mess. Governance, yeah.

OK, so, yeah, the future of soft forks is definitely unclear. This is very much in flux. How is this going to work in the future? How are people going to agree on these things, right? It seems, a lot of the times, from the developer perspective, it seems like, why not? Like, hey, we made this cool new signature system. It's faster, it's more secure, it saves space, let's use it. And then people say, no and you're like, well, why? Things like that.

But on the other hand, if it's like, no. We only have odd outputs. It's like, why? Who cares. Let's use, even and odd numbers. OK so another aspect with the forks. Transaction replay. So this is tricky. So a split happens. Let's say in the case of a minority soft fork, or a majority, but not unanimous hard fork, or a full fork, something like that. There's a split, right? There's two chains that are now being extended. You make a transaction on the old chain what happens on the new chain? Yes?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. [INAUDIBLE] it and it happens on both, right? In many cases, these transactions are valid on either. And so they can be rebroadcast or relayed between the two networks. And if it can be relayed between the two networks, it will. Someone's going to set up a little script that grabs all the transactions on one chain, broadcast them on the other, even if you don't want them to someone will do that, right?

And if it's valid on both, it gets confirmed on both. And so you say, OK, it now splits. There's now two different histories. At the time of the split, now I've got coins on both, right? I don't usually think of it when it's a short term split as I now have two types of coins, but I do. If these extend indefinitely and they're never going to reconverge, well, I've still got the UTXOs on both. I can make a transaction here, and maybe it gets relayed here and now they move on both sides.

Or maybe it doesn't. So you can you can potentially-- and eventually the UTXO sets will diverge, right. How do you diverge these things? Well, if you mix it with coins that have been mined, so you know that the mine, the new coinbase, the new coins here, are definitely different, right? Those are going to have different TXIDs. Coins that got mined here will not exist on this one, and vise versa.

So eventually, more and more coins start getting mixed in with each other and they will diverge. That takes a while though. Another thing you can try to doing is a spamming double spends. I'm going to send this-- I'll make a transaction, Alice pays Bob, I also make a transaction, Alice pays Carol. I just send one to one place, one to the other, hope they get in. Eventually, they'll start diverging just by chance.

You can also try exploiting locktime deltas. So in many cases, the heights will be different, and you can say, OK, I'm here. I'm going to make a transaction that's only valid after block 5. And if someone replays it here, they're going to have to wait 2 blocks before it's valid. And then this gets confirmed here, and then I make a different transaction spending the coins here without a time lock. And so I can try to exploit the fact that there's timing differences between the two chains to make a transaction A occur on the top one, and transaction B occur on the bottom one, and split my coins off that way.

So those are potential ways to split your coins, despite these transaction replays. And then you can now say, OK, I have two separate wallets, two separate keys on the different chains. However, yes, this is expensive. This is ugly. It's possible, but it's ugly because you're basically going to spam the network and in many cases, you're not actually trying to send money, you're just moving your own money around internally.

So if everyone does that in the system, it can overload the system, have tons of transactions, also if you're doing this, it might not work. And you're like, OK, I just confirmed a transaction for no point whatsoever, got to keep trying. It's pretty ugly. Also, people don't know, so why is this a problem? Well, in many cases, you want to sell one and not the other.

In addition to these software rules, such as bonus coins for each transaction or only odd numbers allowed, there are often philosophical and cultural rules that get associated with it, and people hate each other and yell at each other and insult each other on the internet all the time. This is-- I don't think this is unique to bitcoin or cryptocurrencies, I think it's just, you got money involved, you got trolls on the internet, it's a rich mixture of the best parts of humanity.

So a lot of times people want to sell one or the other. So they say, I think the odd coin is stupid. I'm going to sell it, and someone wants to buy it and I'll get these new coins. That's difficult if transaction replays are occurring. Another problem is that many users could be unaware of the forks. I know a lot of people-- there have been a bunch of full forks in bitcoin recently, where different rules have to been adopted. In many cases, entirely different proofs of works, things like that.

I'm not aware of all of them. I know of some of them. Most people I know don't know of all of them or even any of them. So users might unknowingly send both or not be aware of these things. That's an issue. There's also all sorts of crazy legal issues. Talking to exchanges, where like, OK, this fork happens. Do we owe our customers both? Do we only owe them the one that-- and which does the exchange have to let people decide to adopt or ignore new rules set? There's all sorts of weird legal issues that are still being settled.

And for one example, you can do a replay attack on exchange and this is not a theoretical example, this has happened. So let's say the network splits, right? You get bonus coin and regular old coin. And the bonus coin has a majority hash rate, and there's no kind of replay protection. All the transactions that are valid in one are valid in the other.

OK, so the network splits into coinA and coinB. And the exchange is only running coinB. They say, look, this has the most hash power and that's what defines the system. They adopt a new rule, fine. There's this new rule that you can generate a coin out of nothing. So the user says to the exchange, OK, I'm going to deposit coinB. The exchange says, yes, I acknowledge your deposit of coinB. That's the network I'm running on, I see your transaction, it's in a block. Cool, you've got a balance. And the user says, changed my mind. I'm withdrawing coinB and the exchange-- so what happens next?

The exchange says, sure. Here's coinB. Oh, and coinA, right? The exchange doesn't implement any replay protection. They don't know. They don't acknowledge the existence of this other chain. They don't know they have coinA, and they should because they actively split but whatever. And then the users like, cool I got both, right. I relayed this transaction between the two networks. I was now able to deposit only coinB and withdraw both coinA and coinB. And now I redeposit-- I split again, I redeposit coinB, and I keep doing that.

And so I can drain the exchange of all of their coinA with the same amount of coinB just looping through depositing and withdrawing. So, yeah, this happens. Does anyone-- I mean, I'm not going to say which exchange was susceptible to this attack. Does anyone know? It shares a name with the first transaction in the block. Anyway, so yeah, that was almost two years ago, a year and a half ago with the ethereum, ethereum classic hard fork. That happened.

So that was-- it happened. I'm not saying, yeah, it's not obvious, right? These are some attacks that are like huh. In retrospect, it wasn't that hard to find out. I'm sure the people at the exchange were like huh, shoot. Yeah, we probably should've seen that coming, and we lost a couple of million bucks. Shoot. It's also weird because all of their users like generally are identified, and they have their password or where they live, and so you could probably tell like, hey, come on dude, give it back. But maybe they didn't because they like, well, tech-- because they might have a program in a way where they could deny it. And say, look, I just deposited and withdrew a couple times. That's what I always do, I don't know.

But yeah, there were definitely warnings. There were there were a lot of people saying, hey, this is dangerous. You need to really implement replay protection. If there is a fork without replay protection implemented, the exchanges really need to honor and try to split both before offering both for a deposit and withdrawal, things like that. And so there's been a lot, last year as well, there's a lot of argument because in bitcoin, one group of people wanted to SegWit2x, was what it was called.

And they wanted to implement a hard fork and not implement replay protection. And so that was a big argument where people were saying, look, if you're going to make a hard fork, make it a full fork. Make it implement so that you're going to go off, but also implement replay protection. Make it so that transactions that you guys signed are slightly different than the old way. And it's not too hard to do. What you can do is when you're making a signature, flip some bits.

Well, OK. You can't flip bits in the signature itself because those can be flipped back, but what you can do is you can like flip a bit or two in the hash that you're signing, and then the old software won't be aware of that flip and say, look, this doesn't look like a valid signature because it's trying to compare it against a different message. Or you can like a pen, something at the end of the message you're signing, things like that. So that on the new network, the signatures look different.

And that helps in terms of safety because then the old software that's ignoring will not inadvertently send transactions. And also for the new network, they will not inadvertently send transactions on the old network. So that's-- and there's a lot of ideas of opt in versus opt out replay protection, where you can like allow the option to sign differently, but not require it. All sorts of weird ways you can do it.

But yeah, this is a fairly recent mostly last summer, last fall, people were trying to do different things. And so in practice-- I think this is the end of it. Let me go two more minutes. But yeah, consensus change is hard. In practice, there's been some full forks recently. The last soft fork was Segregated Witness SegWit happened sometime in September last year. It was it was a mess, and there was also some full forks.

Bitcoin Cash, and then later Bitcoin Gold, which is a lot smaller, and they completely changed the proof of work. And now they're a bunch that are like pushing the definition of full fork, where they're basically like also called airdrops, where it's sort of a completely different coin that just happens to have the UTXO side of the old coin. And so it's like, why even bother with the history. We're just like look, it's a new coin that you inherit all these other coins.

Yeah, so there's a bunch of those. It's a mess, it's fun being-- I could not have given this lecture a year ago. A lot of these things had not happened. A lot of these terms were not well defined. A few years ago, the idea of soft, hard forks were not even defined. It's pretty clear that Satoshi later, after releasing bitcoin started to understand this system. But in the beginning, there was not a clear understanding.

Probably the biggest, contentious software in 2009 was that Satoshi added a 1 megabyte block size limit. And to reverse a soft fork, is a hard fork, and so this blocks out his hard fork. And then there is a very clever way with SegWit to make it a software, but also increase the block size in a weird way that the old software wouldn't recognize.

I might have to explain a little SegWit to you next week. OK, so yeah. It's a feature and a bug, right? Consensus changes in these systems can be very difficult. On the one hand, you want your coins to stay put. You don't want your money to change. You want to be able to just have a bunch of money, and a year later, you still have a bunch of money, and that's what you want to do.

On the other hand, new features are cool. And these are not-- you don't want these to be ossified legacy systems, you want this to be like new, cool technology and you go make all these new cool things. And make it faster, and better, stronger. And the role of miners is also a big point of contention, right? They seem to have outsize influence in some ways, right?

Up here is the mining power and how that affects things. And why should these miners have outsize influence? Shouldn't the users themselves be able to vote? But they can't, right? If the users could vote, maybe you wouldn't need mining at all to verify block chain. So there will continue to be a lot of debate on this stuff going into the future. I hope this helped explain the general thinking as of early 2018, but it'll probably change. Cool.

Any other questions about this whole thing? If you have a light-- like I don't know, James helps develop Vertcoin, right? Are hard forks and soft forks difficult in Vertcoin? No, you're like, hey, we're doing a hard fork.

AUDIENCE: [INAUDIBLE] the [? exchanges ?]

PROFESSOR: Yeah. So in smaller communities, smaller coins, where there's not as many people involved and people are all on the same page, these changes can be made fairly regularly, not a huge deal. Bitcoin is very messy. Bitcoin everyone hates each other, they're always trolling each other on the internet and hacking each other, death threats, all sorts of stuff. So yeah, future forking methods-- there's probably new, cool ways you can add to the bottom of that chart some new idea that maybe works better. So stay tuned.