[SQUEAKING]

[RUSTLING]

[CLICKING]

PROFESSOR: OK, actually, before I continue, one administrative announcement-- OK, there's a sign up sheet to Scribd. You guys saw this very fast Scribd. It's all taken, so that's great. If anyone-- how many people here are taking this class for credit and did not sign up to Scribd?

So come talk to me. I want mainly to gauge the numbers, but that's fine. So let me tell you two options. Either you can Scribd with someone. That's OK with me, but it's OK if you don't Scribd. There's other things, more things in the scribing regime that I can happily hand out. So just come talk to me if you didn't Scribd. And thank you so much for scribing, for picking up the spots so quickly.

So before I explain the two to one-- the problem with from one claim to two claims, I want to recap a little bit because I had some questions in the break, and I want to make sure to try to clarify. OK, so what are we doing here? The verifier-- the prover is arguing that in layer I, starting with layer 0, the value at a certain point in the layer-- eventually, it's in the extension. At the beginning, it's not in the extension. The value is some Vi star.

Now we know that-- so he's actually claiming that Vi star is the value of the extension. But by this calculation, we know that in the-- in other words, he's claiming that Vi star is this sum of this gigantic thing. So he's claiming Vi star is sum over P in each of them, sum over w1, w2 in each of them, blah, blah, blah, blah, blah, blah, blah, blah. OK, so the claim is that Vi star is equal to sum of multivariate low-degree polynomials.

What we're going to do, we're going to do this exact black box, sum-check protocol, where beta, the value we're claiming-- we're claiming is Vi star. So the prover is claiming that the sum of this entire thing is going to be f. OK, this function is f. And he's claiming that the sum-- so if you call this entire function, this thing f, the prover is claiming that Vi star is equal to sum and pw1w2 of fpw1w2 in h to the n-- each one in h to the n.

So the prover is claiming that Vi star is this sum. And this f is some kind of low-degree polynomial. So what we're going to do, we're just going to do black box, the sum check. You don't need to remember what the sum-check protocol is. But if you remember, you just-- the prover constantly checks-- sends univariate low-degree polynomial, being checked. But if you don't want to remember what it is, don't remember.

The only important thing to remember is at the end-- and during the sum-check protocol, the verifier just sends randomness. He doesn't need to know anything. He sends a random field element, gets back a univariate polynomial, does some checks, and the random field element. He never actually needs to know what this f tilde is.

The only point where he actually needs to verify-- compare something with f tilde is where he needs to check that f tilde of z0z1z2 is equal to some t-- I don't know-- to some element t that came out from the sum-check protocol, some field element t that came out.

But the thing is, so how does he check it? In the sum-check, we said let's assume he has Oracle access to it. But now, he doesn't have Oracle access to this gigantic mess. So what do we do? So first thing, let me help-- say, OK, he does have Oracle access to add tilde. Let's assume that. But even then, he doesn't know what this Vi-- he doesn't know the-- he can't compute the low-degree extension of one layer below, so he doesn't know the extension on z1 and z2.

So these are really the only things he does know, beyond the tilde, which we assume he knows. If he knew them, that he could check, and he can catch you, the prover. But he doesn't know them. So now we say, OK, so we're not done. We don't just do sum-check protocol. I mean, I said only sum-check, but it's many sum-checks, not just one. So we're not done.

So now we say, you know what, the prover will give the verifier the values, these two values. You know what, you can't compute them unknown. I'll tell you what they are. Now, of course, he's going to cheat because if he tells honestly, he's going to get caught because by the sum-check protocol, there's a very high probability this t is actually not the right value.

So because of the soundness, he's going to catch him. This is not the correct value. So now when the verifier-- so now the verifier tells the prover, give me these two values. If he gives the correct value, then the verifier will compute the correct f, and he'll notice the cheat, and he'll reject him. So now what we know is if the prover wants to win, he must give false answers. One of these answers or both must be false. Otherwise, he's going to be caught.

And therefore, we reduced a claim from layer I to two claims and layer I plus 1. Is there any question about this? Does anybody want me to repeat what I just said, maybe in different words, or is this clear? OK, so now what we do?

So let me actually-- OK, so what do we do? So the idea is-- actually, in the GKR paper itself, what we did is, we did kind of another protocol for go to two claims to one claim. But actually, there's a much, much simpler idea, which is actually Rachel's idea. Well, you don't need to do anything, actually. So here's the idea. Don't do anything. Actually, it's not a problem.

I know it's funny, but it's actually not a problem. Why is it not a problem? So we reduced from one claim to two claims. Fine. Continue with two claims. Now you're saying, what do you mean continuing with two claims? You're going to get four claims and eight claims. That's terrible. Yeah?

AUDIENCE: It's not my idea.

PROFESSOR: What?

AUDIENCE: It was Lisa's idea.

PROFESSOR: Oh, it's Lisa's idea? OK, Lisa's idea. OK. So you go-- right, it was Lisa's idea. I forgot. So you go from-- so you just continue with two. And now you're saying if you continue with two, you're going to get to four. So the point is, you're not. You're going to stay with two, and you're going to stay with two and stay with two and stay with two. Why?

So here's the idea. We reduced it to two claims about layer I plus 1. So now when we go down, we're going to do two sum checks. We're going to do again the exact same sum check, but twice because now we have two claims. So now we're-- OK, so let's see. Let me maybe write it, so it'll be a little clearer.

So let me actually write a little bit the notation. So now let's-- let's call this-- I have this function. Let's call fi, the sum-- fi is a function of p-- yeah, p, w1, w2, and all it does is it does a add-- it reduces the value of checking on p to checking two elements in layer I plus 1.

I want to have a neat notation, so we'll be able to see it. So it does sum-- it does-- sorry, it checks. If it's add, p, w1, w2, so it checks add, and it adds a Vi-- well, we'll do the extension-- but Vi and a w1 plus Vi plus 1, Vi plus 1 and w2 plus mult. So I want to give this a name, so we'll have it. This is i. This is i, p, w1, w2 times Vi plus 1, w1 times Vi plus 1, w2.

So for each and every w1 and w2, we check whether-- we want to go to 1 layer below, so we check whether-- if it's an add, we do an add. If it's a mult, we do a mult. And at the end, we also multiply this with chi of p and w0. No, one second. That was the claim.

Let me call it zi. This is fi of zi. Good. And now we can think of this as a polynomial because it's all extended. And when we check, we check-- we get a value Vi, and we check-- or Vi star. We check that Vi star is the sum of f. This is the i star. That was the star. We check that it's fi, zi star of p, w1, w2 of p, w1, w2 e, h to the n

So this is where we start, right? We have a value on layer-- we have a fixed zi star in layer i. There's a claim that the extension that equal Vi star, and we know that the sum, just by this calculation, that Vi star should be-- if you were honest-- it should be sum of this polynomial f. And we do a sum check.

And when we do a sum check, we reduce. And now we reduced it to a claim after this-- so now in the GKR protocol, first we do a sum check for-- so for each layer. what do we do?

Sum check corresponding to fi for proving that Vi star, which we got from the previous layer-- OK, we got it-- so it originates from-- it's the claim. V0 star is the claim of the prover-- the original. He claimed that C of x equals V0 star. That's the original V0 star. And then we check. We do sum check that this is equal to fi, zi star. These are both the claims that we started with.

Sorry. So the sum p, w1, w2 and fi, zi star of p, w1, w2. So we do-- we do the sum check. At the end of this sum check, we reduced it to checking two things. We reduced it to checking V tilde-- two things-- Vi plus 1, 1 star, and Vi plus 2, 2 star. So we have two V's now, and two--

AUDIENCE: [INAUDIBLE]

PROFESSOR: Wait, what?

AUDIENCE: The second to right V is V--

PROFESSOR: Oh, one. Sorry, sorry, I plus 1-- I plus 1. And then two points, we called it z1, z2, but these are random points from the prover. That these are-- so these should be z1 and z2 respectively. These are now star because they are going to be fixed. OK, so now we have a new sum check.

So this boils down to-- so now what we want to do? We want to do two sum checks. First sum check that Vi plus 1 star is equal to sum fi plus 1, and we call them star. So we have z. Let me call this now-- we called it z star. OK, z1 star and p, w1, w2. But we have another one-- Vi plus 1, 2, which is fi plus 1, V2 star sum p, w1, w2, p, w1, w2. So now we need to prove this.

We have two sum checks. And note that it's not even-- it's even two sum checks of different functions too. This is with z1 star, and this is of z2 star. It's kind of a little different, the functions. Yeah?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Exactly. You got it. That's it. It's as simple as that. So what do we do? We something-- do two sum checks. How do I do two-- won't it grow to four? No, the simple, trivial observation, do to sum checks with the same randomness from the prover. So the GKR says when you have this, do two sum checks with same prover randomness-- verifier-- sorry, sorry, verifier randomness.

So at the end, what do we have if we use the same randomness? We need to-- it reduces to verifying fi plus 1, z1 star, and some randomness from the verifier, which we call-- let me call it-- I'll call it again, z0, z1, z2. It's fresh. And also, he needs to check fi plus 1, z2 star and z0, z1, z2. How does he check them? What is this f? Let's go back.

We have all-- this is the f, right? It's like the [MUMBLES] all this. So we said, look, add tilde, don't worry, he can do on his own. Mult tilde, he can do on his own. What does he need from the verifier? And sorry-- and this chi, he can do on his own. What does he actually need from the prover?

What he needs from the prover is v plus 1-- v plus 2 tilde because we're going one less-- and z1 and on z2. But it's the same z1 and z2 on both. So all he needs from the prover is the value of V tilde and z1 and z2. And he's done. So yeah, two values-- we started with two. We ended with two. That's it. You continue like this all the way down.

So to summarize, what is the GKR protocol? It's just a bunch of two sum checks in parallel, D of them. That's all. That's what we do. And in the middle, the prover gives information. The prover needs to give the v tildes to help him out, but that's it. Yes?

AUDIENCE: You need to interleave the [INAUDIBLE], right?

PROFESSOR: You do it in parallel. Yes, it's very important-- good observation, Gabe. It's very important that when you do-- when you go from layer I to layer I plus 1, the two, sum checks are done in parallel. If you do it sequentially, then now, the second subject you know the coin, so you can cheat. But you do two-- you do-- so here's the GKR protocol.

You start with the output layer. You do-- so the output layer is two sum checks, but it's just one. It's for the same thing. So you can think of it as just one sum check. So you do the sum check twice for the sake of it, but really, one. You do sum check. You reduce to-- so you do the two sum checks. It reduces to two values in layer I plus 1.

But not just it reduces-- it reduces to a point here, which you can compute from two values in layer one below. So I ask the provers, give me the two values. He must cheat if he-- I mean, if he cheated originally, he must continue to cheat. So I have two values, and then I do another sum check with the same randomness, same exact randomness.

So I do a sum check. I go one layer below. Again, I need to check. I get a value of f, but the two values of f to check them. I need to ask the prover, give me two values on this layer I plus 2 below. I get these two value. By soundness of the sum check, they must-- they should not be correct. And then I do another sum check, another sum check, another, until I reach the leaves-- the input.

Once I reach the input, what do they give me? Give me a point in the low-degree extension of my input. That I can compute in linear time. It's just a sum of-- a weighted sum over my input values. So I just compute it on my own. That's it. That's the protocol. So it's really just two sum checks in parallel with same randomness. Prover sends two values in the low-degree extension of the next layer. Two sum checks, present two values. Two sum checks, present. That's the protocol. Yeah?

AUDIENCE: So those are not the same that, right? So you're saying it's the same as each other, not the same as the previous layer.

PROFESSOR: Yeah, exactly. Yeah, yeah, yeah. These are actually per layer, right. This is like I plus 1. This is like I. Exactly. So there's the same for I, but in Is, they have different-- in different layers, you choose the randomness fresh. Yes, thank you. Yeah?

AUDIENCE: Is there any security proof of the parallel version that-- does it just come from the sum-check protocol? Would there be any issue with the security [INAUDIBLE] randomness?

PROFESSOR: Good. So let's do the security proof. I'm trying to think what's the best to erase here. I'll erase this. OK, so let's do the security proof, the soundness. So first, I guess, the completeness really follows from the completeness from the sum check. If you do things correctly, the sum check should work. You just get-- the completeness is pretty trivial. The complexity y-- the complexity-- we'll do the soundness in a minute. Let's first do the complexity.

So what are we doing? We're doing two sum checks per layer. We have d layers, so we run in time-- so, OK, let's first do the complexity. So what is the communication complexity? So we do d pairs of sum-check protocols. So we already run in d. What is the communication complexity of each sum-check protocol? Number of variables, degree, and log F. Number of variable is 3m, order m. The degree is h minus 1, but-- oh, I just erased. So the depth-- the number of variables like log S over log, log S, the degree is log S, so everything is like poly log S.

So we don't need o. And times the log f, but we'll take f to be poly log s2, so everything is d times polylog s. OK. So I'll put it aside. We'll take F it can be very big, but it will be poly log S. That's the field. OK. Sorry. It's log-- poly s. Yeah, so log s is-- yeah, log f is poly log s. Great.

So this is communication complexity. What about the verifier runtime? So of course, he does the-- he does the-- so in the GKR, all he does, a bunch of sum-check protocols-- so he just sends randomness and does whatever-- does these-- I guess the runtime of the sum-check protocol. And then he gets at each time two elements from the prover. And he just needs to check consistency of this f, right?

Assuming he has f tilde and multa as an oracle, the only thing he checks is that-- this times this plus this is equals to some value. So that's just a bunch of field elements. It's nothing. So really, it's the same as this, not much more. And what does the prover do? The prover just does a bunch of sum checks.

So the prover runtime-- well, he does d sum checks or 2d, but we'll do a lot of order-- times the number of variables m, which is less than log s times h to the m, which is s. So already, he's like poly s, so the d doesn't matter actually. Essentially, d poly s because the d is poly s, h to the m is poly s.

And the time to compute-- he also runs in the time to compute this polynomial, this kind of function, but the time to compute this is also like poly and h to the m because each one of them is like sum of h to the m terms times a very simple polynomial. That's kind of the low-degree extension. So overall, he runs in time poly s. So really-- so this is what you get, and this is what we wanted. So the verify runs in time d, the number of sum checks times polylog s because each time, he does something very, very small.

So let's talk about the soundness. That's kind of the interesting part. So for soundness-- the soundness analysis goes as follows. It says if the prover cheated-- successfully cheated, so he lied, but we accepted him-- we want to argue that this happens with a small probability. So for this to happen, there must exist a layer when he went from a false claim to a true claim. Because at the end, he gave us a true claim because I'm checking. In the beginning, he gave me a false claim. So somewhere I had to go from false to true.

So let me-- I want to look at the notation, so I won't mess it up this time. So here's the soundness. So let's denote-- so suppose we have a cheating prover that cheats with probability epsilon. And then we'll see also what the epsilon is, what's the best probability you can cheat. So suppose there's a p star that cheats with probability epsilon-- cheats, meaning he gives me a false statement, but I accept him.

Then I'm looking-- so then for every-- actually, for every i in d, I want to denote the randomness that was used by the sum check, by the two parallel sum-check protocols. Let's denote by Zi0, Zi1, and Zi2 the randomness use of a V in the i-th sum-check protocol-- in the two kind of parallel-- use the same randomness.

So let's just denote by Zi0, that's kind of the Z0, Z1, Z2. But in each layer, it's different. OK. And let's denote the resulting claim. So denote resulting claim of after the i by a Vi tilde, and Zi1 equals Vi1 and Vi tilde. And Zi2 equals Vi2. OK, so this is the claim, what the verifier sends me after the sum-check protocol. So the sum-check protocol gave him a bunch of Zi0, Zi1, Zi2. This is in the i-th pairs of sum-check protocol.

And then I can-- for the next layer, he gives me the values. This is just notation. So with this notation, what does it mean for the prover to cheat? So let's denote by kind of a bad event, Bi, the event-- so I'm going to denote by Bi for bad-- the event that in round i minus 1, he was faulty, namely he gave me cheat.

But miraculously in round i, everything is fine. So denote by Bi is the event that what? That in round i minus 1, Vi minus 1 and Zi minus 1, 1, either this is not true, or Vi minus-- or the other one is not true. So one of them is not true.

So bad is the event that in round i minus 1, at least one of the claim is false. But in round i-- and in round i, both of them are true miraculously. This is also an equal. And Vi and Zi2 is equal to Vi2.

Note that if the prover cheated, it must be the case that the probability-- because the prover cheated with probability epsilon by our assumption, it must be the case that B1 one or Bd is at least epsilon because he cheats with probability epsilon. So the probability of epsilon-- he starts with false and gets to true.

So either he already got the true after round two, after the first one, or went to true. So one of the Bs must hold. One of the bad event must happen for him to cheat. OK, let me denote the epsilon here. But by union bound, this is just sum on d of probability of Bi. OK, so we know that he can cheat with probability and with probability at most sum of the probability of bad and i.

Now what is it bad-- so let's look at probability of bad and i. So let me actually even this-- there's two types of bad i. There's bad i where this is bad, bad i 0, or bad i 1 and bad i 2, or this is bad. So just let me denote this by sum and i and d, sorry.

The probability-- so this is just notation. I want to call this Bi1 or Bi2. So let me define. Bi1 is that bad where-- this is bad where the 1 is bad. So this means that Vi minus 1, Zi minus 1, 1 is bad. It's not-- it was false. And this means that 2 was false.

So Bi says one of them was false. Let's partition it. Either the first was false or the second. Now maybe both-- I said or. But I know one of them is false. Now I'm going to union bound. So I'm going to say again, this is equal to some and i and d, sum and B in 1 or 2. Probability of Bi, B.

Now let's look-- what is the probability of B-- what is this bad event, Bi, B? Let's think of Bi for 1 for simplicity. This is false, but the layer below is true. If this is false and the layer below is true, it means that he cheated in the sum-check protocol. This exactly means he cheated in the sum-check protocol because he managed to start with a false claim in the sum-check protocol and get you to a true claim at the f, at the bottom.

Because if he didn't give you a true claim-- if he gave you a true, you wouldn't accept him. You would actually call him out. So if you accepted him when he gave you a true claim in i, it must mean that he cheated in the i-th in that sum-check protocol. Yeah?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Oh, sorry, yes. Yeah. Sorry. i union bound, everything is more equal to. Thank you.

AUDIENCE: And the middle one.

PROFESSOR: Sorry. This is equal. This is definition. Yeah, this is just definition.

AUDIENCE: Bi 1 again.

PROFESSOR: OK, yeah. So Bi 1-- OK, so let me explain. Bi is the event that this or this and both of this. Bi 1 is just the event that this is false and this. Bi 2 is the event that this is false and this.

AUDIENCE: [INAUDIBLE].

PROFESSOR: OK. Exactly. So that is the-- Bi is the event that in layer i minus 1, one of the answers is false. But in layer i, both are true. That's Bi. Bi 1-- I'm partitioned into two. Bi 1 is that actually, it's the first claim that's false, and the next layer is true. Bi 2 is that the second claim was false, and the layer below is true. So I'm kind of partitioning the-- there's two sum-check protocols.

I'm saying, look, you cheated in one of them. Either you cheated in this one, or you cheated in this one. So Bi 1 said you cheated in the first sum-check protocol because this was false, and you managed to get a true for some reason. Bi 2 says you cheated in the second protocol. You started with false, and you got true.

So going back to the question about-- OK, let me just finish this. What's the probability of cheating in the sum-check protocol? Well, we have it-- m times d divided by f. M is less than log s. d Is less than log s, so let's say, log squared s divided by f. So really, the probability to cheat epsilon is smaller than log squared f divided by epsilon. Squared f is just kind of m times d for us because m is like log s over log, log s. d is H minus 1, which is log s, so essentially, log squared s over s.

So as long as you take f to be-- oh, sorry. Yeah. Yeah. So as long as you take f to be significantly-- now just take f to be-- you pay with polylog f in the communication and everything. So take a big f, and you're good. Now about the comment of isn't it dangerous that we use the same randomness for two sum-check protocols-- it's not dangerous. And the reason it's not dangerous because we're doing a union bound.

The point is we're doing two sum-check protocols, but actually, we don't-- look, let's forget that we're doing with another. I don't care. Let's just look at one of them. We know that on this, you can't cheat even if you're doing a lot of other things in the side. On this, we can't cheat.

And we know that on this we can. Now, it may be correlated. I don't care. It may be that if you cheat on one if and only if you cheat on the other. The fact that you cheat on one is good enough for me. I don't-- so the union bound takes care of the fact that we use the same randomness.

Questions? Yeah.

AUDIENCE: So in the sum-check protocol, is it-- is the GKR apply in like a black box way?

PROFESSOR: Exactly. So GKR uses the sum-check protocol completely in a black box way. You don't need to ever open the box to know how it works.

AUDIENCE: Oh, so there's Z0, Z1, Z2's randomness [INAUDIBLE].

PROFESSOR: Yeah, exactly. So I'll tell you what you need about the sum-check protocol. The one property you need-- so you can use any quote, unquote sum-check protocol as long as it has the following property. The property is that the protocol is public coin, namely the verify-- actually, that even doesn't matter. Let me even-- the verifier sends some randomness.

Let's call it T1 to TM. I don't care what else he sends, actually, whatever. But at the end of the day, it reduces to the verifier needs to check f and T1 up to TM. So the only thing I'm really using about the sum-check protocol beyond the guarantees is the fact that at the end of the day, the verifier checks f and random points generated by him because that is what allows me to do the two sum-checks in parallel.

And as I said, by the way, in the original GKR, we didn't need that. We could use any sum-check because we actually didn't do two GKRs. There was kind of a 2 to 1 trick that went in there. But the way I presented it here, which is simpler, we need-- the property we need about the sum check is that-- the protocol itself is that at the end, it reduces to verifying a single point of the verifier's choosing. Yeah?

AUDIENCE: What happens at the very end when you're like-- you get to the input layer?

PROFESSOR: Good.

AUDIENCE: How do you define the extension of that or compute that?

PROFESSOR: OK. So what happens in the input layer? In the input layer, at the end, the verifier needs to compute-- the verifier needs to compute Vd, which is the input layer on-- I don't know-- zd1-- it kind of equals Vd1 and Vd and Zd2 is equal to Vd2. He needs to check this. So the verify will check this.

How will he check it? Oh, he can check that on his own. Why? Let's see. How long does it take to check this? Let me see where the low-degree extension is buried. OK, let's see. What is the low-degree-- computing the low-degree extension-- he knows the-- so v-- this is low-degree extension of x1 up to xn. It's just the extension of the input. That's all.

AUDIENCE: So n is small?

PROFESSOR: n is small. N is much smaller than s. So n is very, very small, much smaller than-- of course, the-- look, the verifier needs to read the input. Yeah, he needs to run time n. He needs to know what he's verifying. So he reads the input, and you should think of it as much smaller than s. Otherwise, there's no point in all this.

And to check this-- to check a point in the low-degree extension of x takes time, linear in n times polylog n the field, so quasi linear in n. Because to check the sum-- to check low-degree extension, you just take only n of them because you're doing sum check over n. You can pad it by zeros to get s as you want, but there's only n non-zero f's here. So you're doing sum of n elements, x1 to xn, and each one times this small polynomial that you can compute. That's a polylog computation.

AUDIENCE: When you do the dummy variables for padding, [INAUDIBLE].

PROFESSOR: Exactly, you just set them up to 0. Yeah, exactly. Actually, in the original paper, the way the notations work is more complicated because we didn't set them all to 0. So we had another s and another. But setting all to 0 is just simple because you have less notation. Other questions? Yeah.

AUDIENCE: So [INAUDIBLE] this, but the V1 and V2, do they come from the fan-in 2, left inputs and right inputs or--

PROFESSOR: No, the fan-in 2 came from the fact that add here has only three things. It's like the input and two output. P is the gate above, and it only has two children. That's the fan-in 2. Otherwise, you'll have many. The reason-- OK, yes. The answer is yes actually. Because the-- OK. Yeah, sorry. Yes, because they have only fan-in 2, there's only-- otherwise, you'll have Vw1 plus Vw2 plus Vw3 plus Vw4. Sorry.

AUDIENCE: So if you had a higher fan-in with lower depth, could you do just more sum checks in parallel and [INAUDIBLE] the depth?

PROFESSOR: Yeah. Yeah, yeah, yeah. You can-- OK, good point. OK, Leo has a very good point. What Leo is saying-- he's saying, look, you assumed fan-in 2, but you paid for it in another log number of rounds. Don't assume it. Just don't have fan-in 2. Have p and many children-- w1 up to w whatever.

But then reduce it to more. True, but you can't have too big fan because you're going to pay for it in the communication complexity and everything. So as long as it's most polylog, you're OK. So it's-- yeah. But you're right. You don't-- there's no magic about 2.

OK, we're out of time. Thank you so much and enjoy your break next week for some reason, your holiday.

[LAUGHTER]