PROFESSOR: All right. Welcome back. Today is probably our last lecture about 3SAT, but this time it's about planar 3SAT, which we've sort of alluded to in the past. And we saw a version of what you might call planar 3SAT with planar CircuitSAT last class. But planar 3SAT is a very useful special case of 3SAT. It is just like 3SAT, but you also are told that the bipartite graph, let's say of variables versus clauses is a planar graph.

So the idea is you have some vertices representing variables. v for variable here. And then you separately have some clauses. I guess I should really have more variables. You have an edge between a variable and a clause whenever that clause includes that variable, either in positive or negative form. So maybe some of these edges, I'll draw a red to indicate, actually, that's v bar that is included in this clause. So this clause is vi or vj bar or vk. And so on. And this graph should be planar, shouldn't have any crossings. Something like that would be a valid input to planar 3SAT. Question.

AUDIENCE: So if you're given the fact that a particular bipartite graph is planar, is there a polynomial operation to find a planar arrangement?

PROFESSOR: Yes. Deciding whether a graph is planar and finding a planar embedding when it is is linear time. So here I'm just saying it's given as a planar graph, but not with an embedding. So we're going to use the embeddings in our reduction. Yeah, good point. So we do actually need the ability to draw these things without crossings. Good.

So first thing I'm going to do is prove that this is hard. And usually I don't prove base problems are hard, but this is a sufficiently sophisticated base problem. I mean, we

do lots of reductions from planar 3SAT. But the proof is actually from 3SAT, and I think it's a representative example of how to build a crossover in this kind of setting, and what kind of extra properties you can get out of it.

So here it is. This is the proof in a single diagram. So the notation here is that the little green circles are clauses. And the bigger circles are the variables. Purple circles are variables. So the idea is it in your initial instance-- so we're going to reduce from 3SAT. We're given an arbitrary 3SAT instance. So that has a corresponding bipartite graph. Variables versus clauses. Just not planar.

So consider any drawing. Actually, we'll look at a specific drawing in a moment. But it's going to have some crossings like this. This is a clause-variable connection, another one. They cross here. We're going to replace that with this picture, which is technically not 3SAT, because there's a four-variable clause in the center. But we can fix that. That's not too hard.

Let me first convince you-- and this'll take a little while-- that this is equivalent to this. So the ultimate claim-- so there's four Greek letters in the center as extra variables. The main claim is that a1 equals a2 equals a. And b1 equals b2 equals b. If that's true, then, of course, this connection simulates that connection. And that connection simulates this connection. And the claim is as long as you satisfy that, you can always set the stuff in the middle to be happy. So those are the only constraints, basically.

So let's start with over here and down here. These little sort of length two paths-- sorry, so the coloring is similar to what I did on the board. The red are negative connections, meaning that clause has not a and this clause has not a2. And the blue connections are positive connections.

So when you have this sort of alternating four-cycle here, you can read that as not a or a2, which is the same as saying a implies a2. And also, a2 implies a. That means they have the same value. a if and only if a2. Or I'll just write equality of their truth values. Same thing here. So that part is just to get warmed up.

Now the fun part is on the inside. And I think the next step is to think about these sort of triangles in the corner. If you stare at them for a little while, you get these kinds of constraints. So let's work through this one. You'll see there's a certain symmetry among what the alpha, beta, gamma, deltas are. They're going to be all sort of all patterns of a2 possibly complemented, anded with b2 possibly complemented. So these are supposed to be the four cases of a versus b.

The trick is that some of them are 2's and some of them are 1's. So that's what makes it a little bit annoying. It would be nice to make it perfectly symmetric, but that would, in terms of these labels-- but that would destroy planarity. So this is the planar version.

So let's look at this. We have, for example, not alpha or b2. So alpha implies b2. Also, alpha implies a2. So if alpha is true, then they must both be true. And conversely, I want a2 and b2 to imply alpha. Right. So if these are both true, then this guy's unsatisfied from those two red clauses. So alpha must be true. Good. So it works in both directions. You get equality.

It's exactly the same in each of these four corners, just the coloring is different. And you're connected to a different b. So here if you reflect, you're connected to b1 instead of b2. And there's a complement here basically because these two colors switched from here. So it's exactly the same argument, but you just imagine flipping the value of b1. And then reflect over here. Switch the value of the a's. Reflect over here, switch the coloring of the b's. So that's what's happening in the corners.

And at this point, I had to draw a table of all the cases. Oh, sorry. This is a little bit about what happens in the center. So of course, the center clause says that at least one of these is true. We want exactly one to be true, because exactly one of these cases should happen. But that's a weak constraint.

And then we also have some constraints like if alpha is true, then this is not satisfied from alpha. And similarly, this. And therefore, beta must be false. So if alpha is true, then beta and delta are false. I think that's actually all you need. I think these might not be necessary. But they make for a nice symmetric diagram. And they don't hurt.

So now I drew a table-- there may be a more direct argument-- of all the possible settings a2 and b2, and what you can derive from that. And everything except this part is really easy to derive, assuming I can remember what I did last night. So let's say a2 and b2 are false. I claim alpha is false, because we have this equation. That's easy. And in general, you can compute alpha directly, because it's just a function of a2 and b2. It's the end function. So that's good.

What about beta? I claim if alpha is false here, then in this particular case-- well, OK. Here's one thing we wrote. Alpha implies not beta and not gamma. So in this last row, if alpha is true, we know that beta and gamma are false. That's good. We hope also that beta and delta are false. We hope also that gamma is false. But I don't think we need that, because if you look at this equation on delta, which we know is 0, that must mean that one of these two things is 0. But b2 is true.

Sorry. I'm going to use true and 1 interchangeably, and 0 and false for whatever reason today. So this needs to be 0, but this is 1. So this must be 0, which means that a1 is 1. And symmetrically on the top, this is a symmetric between a and b. So up here, you know that a2 is 1. And therefore, b1 bar must be 0. So b1 must be 1. So we transfer the information.

And the only other thing to check in all these cases is that actually everything is satisfied. But that's sort of boring. I want to do the other direction, which is that you are forced to communicate the copies of a and the copies of b. So that was the last row.

I think the first row should also be easy for a slightly different reason. So a2 and b2 are false. Alpha is false. Beta is false. That's as far as we've gotten so far. Let's see. Delta is a1 bar and b2, right? Right. Because b2 is false, we know that delta is false. So it doesn't matter what a1 is. So it's sort of the reverse direction.

And similarly, we should be able to-- oh, because from this, we know at least one of them is true, at least one of the Greek letters is true, then we've got three down. So it must be gamma that is true. And once you know gamma is true, you know exactly

what a1 and b1 are. So that's another good case.

These are slightly more annoying, at least as far as I could see. Maybe there's a cleaner argument. So let's say they're symmetric, though. So let's do the second row. So a2 is 0. b2 is 0. I think just like this argument, we get that beta is 0, because beta involves a2. We know a2 is 0, so beta equals 0, just like this case, actually. OK. So that's good for that.

I think it's harder to figure out delta. Or here's one way to argue it. Suppose that gamma were 1. It should be 0, because gamma's supposed to be only 1 in this situation. If it's 1, then I claim that alpha 1 and beta 1 must be 0. That's from this equation. And then we should get a contradiction. Help me find the contradiction.

AUDIENCE: Delta becomes 1.

PROFESSOR: Delta becomes 1.

AUDIENCE: And then the clause between gamma and delta is--

PROFESSOR: And then this is unhappy. Good. OK. So we do need a couple other constraints like this one. Cool. So that's the idea. Therefore, gamma is 0. And then by that, delta must be 1. And then we're happy again. OK? So that's the idea in a nutshell. There's maybe other ways to see it.

And this is an old proof by Lichtenstein in 1982. It's a cool paper. It has a lot of results in it. We'll be covering a few different results. But at this point, we have shown this. You just apply this to each crossing. Eventually you get a planar bipartite graph that's satisfiable if and only if the original is a planar 3SAT is NP-hard. Question? So many questions.

AUDIENCE: I guess this is probably silly. But [INAUDIBLE] 3SAT, every clause had to have three variables.

PROFESSOR: Yes. OK. Right.

AUDIENCE: Two or four.

PROFESSOR: I cheated a little bit here. OK. Yeah. Some have two. So for that, I need to be able to construct the value false. And some have four. The four, I believe the claim is you can take a little clause like this and convert it into a clause with another variable.

AUDIENCE: And one of the two edges connecting the variables [INAUDIBLE].

PROFESSOR: Right. Like this? So the claim is this four-way clause to the 4SAT clause is equivalent to these two 3SAT clauses. Let's call this new variable x. So the idea is x is going to represent whether the left side satisfies the clause or the right side satisfies the clause. If x is 1, then it satisfies this for free. But it doesn't satisfy this one, so at least one of these must be satisfied.

On the other hand, if x is 0, then it satisfies this one. And so at least one of these two should be satisfied. OK? So that's a reduction from 4SAT to 3SAT, so to speak.

Now, I didn't create any degree-two clauses, but I have tons of degree-two clauses over there.

AUDIENCE: Just use parallel edges.

PROFESSOR: Just use parallel edges. So you could just repeat the same variable in the clause multiple times. That sounds good. I don't know if there's an easy way to avoid multiple edges. I think it's a little tricky. You can't construct false in the CircuitSAT sense, because everything you build should be satisfiable. So you have to be a little careful. But it might be possible to pair them up, because there should be an even number of those guys. And so maybe you pair them up and connect them in some way. But let's just allow multiple edges here. Yeah?

AUDIENCE: In the original instance of 3SAT, does it allow two clauses with two variables? For getting planar?

PROFESSOR: Yeah. I mean, in general, this would work for any kind of SAT, any kind of CNF-SAT. We're not relying on three-ness at all in this picture. But it's not going to split things up, of course.

AUDIENCE: But the definition of 3SAT has three distinct variables in each clause.

PROFESSOR: There are a couple definitions. One has exactly three, but we never, I think, specify that they are distinct. I think that is hard when they're all distinct. But I don't know the reduction offhand. I think the way we define it is there's exactly three, but we didn't say any uniqueness. So let's say that. And then we're still happy here, because we can duplicate variables. Yeah?

AUDIENCE: Why did we include the a equals a2, and b equals b2?

PROFESSOR: I think it's for the next slide.

[LAUGHTER]

The next claim is that not only is that bipartite graph planar, but it also remains planar if we connect all the variables together in a cycle. So let's say we number the variables 1 through n. It doesn't matter how you number them. Well, in a certain sense. You can restrict to instances of 3SAT that this cycle plus the variable clause adjacency graph is together planar. OK? So that's less clear here.

But let me prove it. This is proved in the same paper. So here's a top-level diagram of what we're trying to do. So we have clauses on one hand. Let's put them all on the y-axis. We have variables on the other hand. These are the original variables in the original 3SAT instance. Put them on the x-axis. Then draw all the connections in the bipartite graph. And furthermore, draw this cycle on the vertices down here. OK?

And obviously, we put them in the order that they appear in the cycle. Otherwise we're making our lives harder. And here is that crossover gadget. This is how it's originally drawn. In this case, it's already been expanded in the center into that picture over there. This is where I got that reduction from. And then there's no color here, but you'll notice the curvy line is an attempt to connect together all of the variables that appear in the crossover gadget in linear order without introducing any crossings. So that diagram is still planar when we add all of those edges.

And so what this says is that if we-- we're trying to build a global cycle that connects all the variables. So obviously, we have this for these variables, but we need to add into this cycle all the variables that appear in all these crossings. And so what this says is if we are coming in either here or here into a1, we can follow this path and then leave either here or here on a2. OK?

And because the diagram is symmetric, also, if you're coming into b1, you can end up in b2. And I'm guessing this is why we separate b2. The reason we separated b2 from b in general is to basically add this connection, so we can separate this gadget from this gadget. So just showing that we can copy and extend things. It may not be necessary, but it definitely matches this diagram. OK.

So this diagram is this diagram, but where every intersection plus a few extra places, like over here, we add these little diamonds. The diamonds indicate that gadget. Then sorry there's no color, but in curved lines is the attempt to connect all the variables together in one single path. Question.

AUDIENCE: So what are the extra crossover gadgets that don't cross anything over?

PROFESSOR: They're just to do this path thing. So what are they? I mean, they are exactly this gadget, just there's no connection on the left.

AUDIENCE: So they're a crossover of the path on the variables. Is that right?

PROFESSOR: Yeah. So instead of going straight here, you just put a crossover gadget in the middle. You can still go through. You're copying some arbitrary information here to some arbitrary information here. So you don't preserve the number of solutions, let's say. But who cares? The reason for that is so that we can take this path, connect to this vertex, and then get out over here.

AUDIENCE: Yeah, that's what I meant by crossing.

PROFESSOR: So that we can do this kind of loop thing. So how do we follow the path? I mean, we basically do it in scan-line order. It's like some printers used to do this, where they print back and forth, left and right, line printers.

So we come in here. I don't know quite why they draw it coming in the middle here. But it's essentially coming to this vertex. Then ending on this vertex on the left side. Then going into this vertex, which brings us to here. Let's say coming on the top. Doesn't matter. And then we go to the right till we get to the end. Here we add an extra crossover, so we can basically cross over this line with our path. And add another one. And whenever we visit an intersection, we can just go. And then we go to the next line and so on. We clear?

Then at the very end, we're either on the right or left, depending on parity. Probably we want to be on the right. So if we're on the left, we'll just add extra crossovers here to end up on the right. Then we can go through all these variables in reverse order and then come back up here. So that will be one non-crossing path in this planar embedding.

Therefore, we've constructed a new formula where the bipartite graph plus the path through all the variables is planar. And so this new version of planar 3SAT is hard. This version is usually called planar 3SAT, but if you want to distinguish it, you could call it planar 3SAT with a variable cycle or something. Yeah?

AUDIENCE: Why is it that we care about this?

PROFESSOR: Why do you care? Aha. We'll get to that a little bit later. Well, we've seen a lot of proofs, actually, where you visit a variable and make a choice, and then visit the next variable and make a choice, and so on. And when you make a choice, you go visit the clauses and things like that. So when we're reducing from planar 3SAT, it's often handy to have this traversable path. If we have a robot or somebody moving around, then we can guarantee planarity of that motion still.

And the connections between the variables don't have to cross anything. In general, our goal here is to avoid crossover gadgets when we're reducing from 3SAT to something else. So this gives you kind of a generic crossover, so that as long as you can do variables and clauses more or less independently and you can build connections, then you don't have to worry about the connections crossing. Yeah?

AUDIENCE: When you're given this bipartite graph, how do you actually tell whether an edge is negated or not?

PROFESSOR: You're still given a formula, let's say. And then you're just told this extra fact that the bipartite graph plus this path and the variables, in that order, is planar.

AUDIENCE: The variable nodes, you have one for every variable and its first negation? Or do you just--

PROFESSOR: No. Sorry. There's only vertices for variables and for clauses, not for literals. But the next version is actually about that case. The next one is-- so the idea, if you were just given the bipartite graph, you'd have to be told the coloring of the edges here to know which literal you're talking about.

But also, the problem remains planar if we use literals instead of variables. So if you were wondering about that version, that will also be planar. It doesn't follow immediately. But the idea is suppose we have vi. So in the regular bipartite graph, we have some positive edges and we have some negative edges.

Instead, I want to represent that as vi and vi bar. And the positive connections are over here. The negative connections are over here. Now I don't need the two coloring, because I have the labels to tell me which is which. And furthermore, I can add this edge connecting them. That will still be planar. So just split vi into two parts. What this tells me is that the positive connections to vi appear as a contiguous chunk. And the negative connections appear as a contiguous chunk. So it's not like alternating white, red, white, red. So far, it is if you look at the reduction. But we can fix that. OK.

So I just want to add, with an edge between the xi and xi bar. Yeah?

AUDIENCE: When you say you add an edge between them, what does that mean? Because all the edges are supposed to be in between literals and clauses.

PROFESSOR: Well, I mean, the graph already wasn't bipartite. We start with a bipartite graph between variables and literals. Then we also add the edges between variables and

their complements. And then we also add a cycle among all the variables, among all the literals, let's say. So I guess this could be part of the cycle, for example. And that graph, which is not bipartite, must be planar. That's the constraint. The claim is that all such problems are NP-hard.

In general, we want to make as small a problem as we can, as special a case as we can NP-hard, because then we're reducing. We have more structure that we get to exploit. This is a special case of this is a special case of this. So the more we can confine it, the better. Cool.

So this is proved also in the Lichtenstein paper. Although, we'll see a cleaner proof in a moment, so I won't cover this in too much detail. But the idea is this. So we're taking each variable, we're replacing it with a big cycle like this. I think it's the same kind of trick as over here. It doesn't matter. All these a's should be the same or sort of alternating. And we're connecting the negative side-- so this is really two vertices connected by an edge. But I draw it like that to make clear that that represents the same variable to literals of it. And do this thing.

And then this is the path. So suppose the path used to go through this way. Then the path now bisects. And you see that all of the positive copies are on one side, and negative copies are on the other side. That's the idea. We will see another proof of a stronger statement, so I think let's not worry about this too much unless there are questions.

Here's the stronger statement.

So in general, the variable cycle-- so I'm going to not subdivide into literals for the moment. But we could put that back. So here's the variable cycle. This decomposes the plane at two regions-- the interior and the exterior. Or course, I could flip the edge, and it doesn't matter which one's interior and exterior. But there are two regions. And what I'm going to require is that every clause that I draw here should have entirely positive connections on this side. And every clause that I draw out here should have entirely negative connections. OK? This is monotone 3SAT again.

So this implies monotone.

We know monotone 3SAT is hard. Now we're claiming that planar monotone 3SAT is hard when all the positive clauses are on one side of the variable cycle, and all the negative clauses are on the other side. This is really helpful.

I haven't yet given this problem a name, because it has a name that uses another word, which I should first introduce. So that's the next page. That word is "rectilinear." so this is a relatively simple modification to what we have already. I want the variables to live on the x-axis. You can think of them as horizontal segments on the x-axis. I prefer this drawing.

So imagine each variable is like a little horizontal segment, or in this case, a box. And the clauses are horizontal segments. And then they're connected by vertical segments between the clause and the variable. And all of the clauses that are above the x-axis should be all positive. All the clauses that are below the x-axis should be all negative. That would be planar monotone rectilinear 3SAT, and I jumped the gun. Planar rectilinear has no such constraint. Each of these could be positive or negative. OK.

So variables are segments. All right. So each variable is a segment on the x-axis. And each clause is another horizontal segment, plus vertical connections to the three variables that it includes in positive or negative form. OK. And then planar monotone rectilinear 3SAT is a special case of monotone 3SAT, where all positive clauses are above the line and conversely, above x-axis. Every clause above the x-axis should be all positive. And every clause below the x-axis should be all negative. OK.

So I guess this one you could call monotone planar 3SAT without rectilinear. But of course, rectilinear doesn't change much. I didn't say why, but if you have a planar drawing like we had before, this is just a particularly nice way to make that drawing. To go here, I just used the fact that there are no crossings. And I stretched out the variables to decent lengths, so that clauses can just go straight down. You can prove any planar graph can be drawn in this way.

Any planar graph that has the vi cycle and has some degree-- three vertices that connect to variables. They have to nest in this way, because if you think of a clause, they could either be in this pocket or outside of it, or down in this region. And so you can just figure out how big this thing needs to be, and then make the next one a little bit bigger. And so you can represent the nesting in this nice orthogonal, or rectilinear structure. Rectilinear just means horizontal and vertical lines. Cool? So that's planar rectilinear 3SAT first observed by Knuth and Raghunathan.

Now we want to-- let's prove that planar monotone rectilinear 3SAT is also hard. Question.

AUDIENCE: Wait. If rectilinearity just is a nice way to draw the graph, can we use the fact that it's rectilinear in our reductions? I mean, is that going to help us any more?

PROFESSOR: The rectilinear helps us mainly when we're reducing to a problem that lives on a grid. It's just a convenient way of thinking about it. It doesn't directly help us. I mean, it's a linear time reduction. I mean, all of these are reductions, so of course, none of them help us in theory. But the cleaner you can make the problem-- I mean, this looks pretty. And we'll see, I think at the end of class, a proof that directly mimics this structure. So that's the only reason. You could, of course, as a first step, say, hey, draw it rectilinearly. But this saves you that step. Cool. All right.

So planar monotone rectilinear 3SAT. Here's an explicit example. Although it's not going to be modified, it's just another example of planar rectilinear 3SAT without the monotone. You see there's negations above, lack of negations below.

And here is the trick to fix things. So suppose you have a variable xi on the x-axis, you have a clause above it that uses that variable in the wrong orientation. So of course, or the reflective picture. But let's say it's above. It's U is negated. So in general, we have a bunch of connections up. We have a bunch of connections down. This one is somewhere in the middle. These bold lines are just to distinguish these are the ones to the right, these are the ones to the left. They don't mean negation, necessarily. Because this one should be negated. OK.

So we're just going to replace it like this. This is the trick that we saw in the right and bottom corner of the crossover gadget. We're going to duplicate xi, way except we'll negate it at the same time. So the other one had a negation here and one negation here, not here. This xi or a, together with xi bar or a bar-- again, these are 2SAT clauses. You have to duplicate an edge to make it 3SAT. That forces xi to be not a, I think. Maybe I even have that as a fate. OK, xi is not equal to a. Similarly, a does not equal b, which means b equals xi. OK?

So we just duplicated xi to be used in all the old scenarios. And maybe these guys still all go down. We could move some of them over here if we wanted to. The ones to the left have to be over here. The ones to the right have to be over here. This one gets to use a negated copy of xi. So boom, one negation down. One negation that was on the wrong side. Repeat this n times.

Where you have a negation you're not supposed to and everywhere you don't have a negation that you're supposed to, apply this trick. You'll increase the number of variables by some constant factor. And now you're in the monotone case. So that's actually really easy. It's our first proof that monotone 3SAT is hard. But it's a pretty easy one. Yeah?

AUDIENCE: Can we use a similar construction to recover splitting the variables into literals?

PROFESSOR: Oh, to do this?

AUDIENCE: Yeah. Because we dropped that when we moved over here, right?

AUDIENCE: But you said it was just [INAUDIBLE].

PROFESSOR: Well, we sort of did. No, actually, this is supposed to be a generalization of this, because you could just split apart-- I mean, you can carve vi in half to the positive side and the negative side. And all the negative sides are below, and all the positive sides are above. So yeah, we could already do that. Right. Because this was not about having them-- it was about having the vertices labeled as literals instead of as variables. So we don't need to explicitly say that this one does not equal that one. That's already in the problem.

So we can combine all of the things I've said so far. That may not always be the case, but so far, we can combine all. Yeah?

AUDIENCE: Silly question. If we're carving them into positive side of the variable on one side and negative on the other side, wouldn't that make all the connections below that positive connections instead of negative connections?

PROFESSOR: Sorry. I don't mean split it in the way of that diagram. I mean that we could, in this sense, we could have vertices represent a positive vi and a negative vi, and have the connections. This was saying that we could separate the positive connections from the negative ones. And I'm just saying we already have that here. This is a stronger property to say that they're also all in the same direction. If you just had this property, then you could have some white here and some white here, and red here and red here. But this is saying you can get them all aligned in the right direction. Yeah.

In fact, if we look at this diagram, maybe you didn't understand this diagram, but it has the same issue. Or this is exactly that kind of picture. So here we have a5. The positive is over here, the negative is over here. Here we have a4. The negative is there, and the positive is there. So this was getting the weaker property. Again, you don't have to worry about it, because we got the stronger one. But that's just to show that there's a difference between the two. Good. So that was planar monotone rectilinear 3SAT. OK.

I have a governor general's warning. So here's another problem. Suppose we have the variable cycle. And I say, well, every clause is inside the cycle. And you are free to have positive and negative connections. It's going to get hard. OK. Something like that. And it's planar. That problem is polynomially solvable.

So I'll rephrase this. If all the clauses are on one side, of the variable cycle, then that version of 3SAT is polynomially solvable. And the intuition is that this kind of nesting structure is essentially a tree. If you look at what nests inside of what, what clauses fall underneath other clauses, that is a nesting relation, which corresponds to a tree.

And then you just do dynamic programming over the tree. That's easy.

The sort of surprising thing is that just having two of these trees, one on the inside of the cycle, one on the outside, is enough to make the problem hard. That's a typical thing, actually, because the trees interdigitate. They have no real relation to each other, and there's no way to keep track of both of them simultaneously with the dynamic program unless P equals mp. OK.

So that's one gotcha to be careful of. So when you're doing your reductions, you have to worry about things above the line and things below the line.

Also, here's a special case of this. Suppose I phrase the following problem. I want the bipartite graph on variables versus clauses, plus the cycle v1 through vn back to v1, plus another cycle on the clause's c1 to cn and back to c1. I want that whole graph to be planar. OK?

If you think about that for a second, that would imply all the clauses are on one side of the variable cycle. And so we're in this case, which is easy. So this is a special case of that case. Now, this one I mentioned, because if this were hard, life would be so easy. It's not so life isn't easy. But you already knew that.

Recall, say, proving push-1 is hard in 2D. Well, first we did it in 3D and then we had a crossover gadget. And most of the work was in the crossover gadget. The base proof was trivial or very easy. So we can get all of these connections between variables and clauses to be planar. We can get all of these connections, the variables in a line, to be planar together, all of those things together.

But we cannot also have a path connecting all the clauses together. Even a path. So I said a path here, which is what we want. So that's annoying, because if we could make all of these things planar, we wouldn't have to worry about these crossovers. Same thing with all the Nintendo proofs, like Super Mario Brothers again. The player is going from one variable to the next, so you need those edges connecting all the variables in a path. You need the edges connecting the clauses in a path for the final check. So planar 3SAT basically doesn't buy you anything in these proofs.

Now, if we could somehow avoid adding these connections, everything else could be planar. That's sort of the point of planar 3SAT. So if you can avoid these things-- and there are a lot of games where that would happen. Just you have to get a point here, and it's just added to your total score in some Never Never Land not in the plane, then life would be much easier, because you wouldn't need a crossover gadget, because you can just reduce from planar 3SAT. OK?

That should give you some intuition why we care about planar 3SAT, but also why we didn't use it for those problems. As far as we know, there's no way to use planar 3SAT and make a simpler Super Mario Brothers proof or whatever. OK.

So next topic. We will see some planar 3SAT proofs, reductions from planar 3SAT. But let me first tell you about other versions of 3SAT. Remember there are three main versions-- planar 3SAT, planar 1-in-3SAT, and-- sorry. There are three versions of 3SAT-- 3SAT, 1-in-3SAT, and Not All Equal SAT. Turns out 1-in-3SAT is hard when its graphs are planar. Not All Equal SAT is easy when graphs are planar. So I will show you both.

AUDIENCE: For this easy case, do you have to have both the variables in the cycle and a clauses in a cycle? Or is it just like seeing the clauses in a path or cycle immediately triggers this easy condition?

PROFESSOR: So I don't know. I think if the clauses are in a path and the variables are not connected at all, it should still be hard. But I couldn't find a proof of that. It's just a vague recollection. So I'm pretty sure if variables are in a path and clauses are in a path, then that's enough to be toast. But all I proved here was cycle. But I think a path is enough. It will still behave like a tree. Mm. Maybe not. I'm not sure. Yeah. So it's a good question. All right.

Let's do planar 1-in-3SAT. So it's pretty much the same thing. I won't duplicate all of the text here. We take the bipartite graph of variables versus clauses. And we also add a path, or cycle, connecting all the variables. And that graph should be planar. And otherwise, it's 1-in-3SAT, exactly 1-in-3SAT.

So here's the proof. It's one of the easiest proofs we'll cover. Suppose you have a clause. It's a reduction from planar 3SAT to planar 1-in-3SAT. OK? So this is also our first proof of 1-in-3SAT being hard.

We are going to take a regular 3SAT clause and turn it into three 1-in-3SAT clauses. And that transformation preserves planarity, so done. So all of our proofs are going to just keep building up in this way. We've already proved this problem's hard when the graph is planar. So let's do this.

At least one of them should be 1. We don't want them all to be 0. So let's try when they're all 0, see if something should break. Let's say this variable is 0, easier to reach. So that makes this guy satisfied, which means both of these must be false. This variable's just hanging out. It's just to make everything degree three.

So if this is 0, this is 1, so this must be 0. So this is 0. This is 0. That means exactly one of these should be 1. If cp is 1, obviously, we're happy. But this will end up forcing this guy to be 0. Or sorry. Actually, it'll leave this guy free to do whatever. So if this is 0, 0, 0, then this must be 1. This is 1. This is 1. Therefore, both of these are 0.

If zr bar is 0, that must mean this one is 1. So if both of these were 0, it forced this guy to be a 1. If this was 0 and this is 1, for example, this is 1. Then this must be 0/ OK. This is 0. But we're free to set either one of these to 1. So zr is free, which is what we want to do. We're simulating regular 3SAT. If this is 1, then both of these guys should be able to do whatever they want. And that's what these guys let you do. They're free floating. This guy can toggle if it's not constrained to be the only guy that's 1. Yeah?

AUDIENCE: What's the dashed line represent?

PROFESSOR: Dashed line is the outline of the gadget. It's just saying you replace that blob with this blob. I mean, slightly more formally, I think it's saying something like, if you contract this to a point or if you contract this to a point, then it's exactly the same diagram. So it's one way to argue you preserve planarity, something like that. So

ignore it, basically.

Now, I guess you would have to argue that you can still have a path that visits all the variables. Another question?

AUDIENCE: Yeah. What's preventing you from setting all three to false?

PROFESSOR: That is one of the cases I just did. If these two are false, then these have to be true. No, sorry. These have to be 0. So these are both 0. And then this has to be a 1 or satisfy this. It's easy to get confused, because that one is 3SAT, this one is 1-in-3SAT. So these all have exactly 1 constraints. OK.

Now, this proof has negations. And you may recall that monotone-- I'm going to switch to saying positive 1-in-3SAT where you have no negations is also hard. So we can also define planar positive 1-in-3SAT. This is also hard by a slightly more complicated proof, more recent proof also. Surprisingly, this planar monotone rectilinear 3SAT, or just planar monotone 3SAT, is also very recent. I think 2010. So a lot of these just fell recently but are a natural culmination of all these simplifications.

So this is a version that-- well, these are some gadgets, actually. This is one way to force the two variables to have opposite values and force the two variables to have equal values, using-- I guess this is really planar positive rectilinear 1-in-3SAT. It's the 1-in-3SAT version of this problem. So again, variables are on the x-axis. Everything above is implicitly all positive. Everything below is implicitly all negative. Oh, sorry. Here, everything is all positive. Yeah?

AUDIENCE: Another nice thing to note about that particular construction is that each of the variables in the clauses are unique, so it's exactly 3. And they're all unique, which someone was asking about before.

PROFESSOR: Good. So in this construction, 1-in-3SAT, all of the clauses do not repeat any variables. Exactly three distinct guys, yeah.

AUDIENCE: So to be clear, this problem is hard, but in this terminology, planar positive

rectilinear 3SAT, not 1-in-3SAT, is easy. Because if everything was on one side of the line--

PROFESSOR: I mean, positive 3SAT is easy. You set all the variables to true.

AUDIENCE: I'm confused. Positive means that--

PROFESSOR: Positive is sort of the 1-in-3SAT version of monotone. So 3SAT monotone means every clause is all positive or all negative. In 1-in-3SAT, you just need everything positive. And it's still hard. But with 3SAT, that doesn't work, because you can set everything to true. 1-in-3SAT requires some false things. So it's sort of the analog. This is sometimes called planar monotone rectilinear 1-in-3SAT. But to avoid confusion, because they are somewhat different, I'm calling it positive. I think this paper actually calls it positive, which makes me happy.

Anyway, you can check these gadgets that they force. In particular here, you're basically forcing a to b 0 by this little construction. You can see as 1. Therefore, this clause forces exactly one of these to be true, the other to be false. So it forces them to be different. You repeat that twice, you get equality. Then you do this.

First, you get rid of all negations, because you have this way to force two things to be different. These Xed out ones are the red lines. Those are connected to negated copies. You just duplicate the variable in negated form and then use a wire there. So this wire becomes that one. x2 is different from x1. x3 is different from x2. So it's equal to x1. Same trick we saw before. Now you have no negations.

So now this type of clause is a 3SAT clause. It's a little weird. Right of the arrows, these are 1-in-3SAT clauses. Left of the arrows are regular 3SAT clauses. So we replace this with this construction. It's sort of like the old one but a little bit spread out, with the duplications, with these equal gadgets. We get a copy of x over here. That's so that if we have more connections below, we can easily access x still. And yeah. These 1-in-3 gadgets constrain this. Yeah, there is an equal gadget--

AUDIENCE: The equals and not equals are not part of the problem, they're just referring to the earlier gadget.

PROFESSOR: No. Right. That's shorthand for these gadgets. So you would plug in each of those things into those little pictures. Yeah. OK. So let's skip those details. But you get that planar positive rectilinear 1-in-3SAT is hard. And that was 2008. Cool.

Let's do planar Not All Equal 3SAT. This is polynomial. Important thing to remember-- in the planar world, you have to be careful with all sorts of things. It would be nice to have a Schaefer-type dichotomy theorem for planar graphs, but I don't know of any such theorem. So meanwhile, this is the sort of main characterization. You have those three standard problems. This one falls in the planar case. Same setup. We have Not All Equal clauses and the connections between those and the variables negated or not. Even when you allow negations, this problem is polynomial.

The fun thing is the proof of this theorem is a reduction, and it looks exactly like a hardness proof. I mean-- [LAUGHS] It just happens to be from this problem to a known polynomial time problem, namely, planar max cut. So we're going to reduce to-- this is one of the few times we will reduce to something-- planar max cut.

So max cut is I give you a graph, and I want to color the vertices two different colors. Think of them as the two sides of a cut. Say red and blue. And I want to maximize the number of edges that are red, blue. So I want to maximize the number of, here, white-black edges, bichromatic edges. That's max cut.

In general graphs, that's NP-hard, but in planar graphs, it's easy, because if you look at the dual graph and then solve the Chinese postman tour problem, which is the shortest path that visits all of the edges at least once, which you can do in polynomial time by perfect matching and a bipartite null in a cleek. Anyway, you do that. And then all the edges that you double are not in the cut, I think. And all the edges that you don't double in that tour are in the max cut. Fun fact. So this is a known fact. Planar max cut is easy.

So if we reduce from planar Not Equal 3SAT to max cut, the problem is easy. And here's the proof. It's got a variable gadget and a Not All Equal clause gadget. So we

want to represent Not All Equal. And so the idea is we're going to represent a variable with this alternating chain. And if you want to maximize the number of red-blue edges, you should alternate, because this have even length. If you don't alternate anywhere, you're going to not get as many edges. In general, we'll say, well, the target cut size you're trying to get, the decision problem is, is there a cut of size at least something?

The something is going to be, in particular, it's going to be a sum of things, but for this gadget, it is two times the number of occurrences that the variable would say, or basically the length of this cycle. We'll just make the cycle big enough, so we get lots of copies of xi, lots of copies of xi bar. So this is one possible setting. If blue means true, this is like saying that we set xi to true. We could do the reverse. We could set xi to false and set xi bar to true. But they have to be opposites.

So now we have-- this is basically a split gadget. We have several copies of xi and xi bar. Then we're going to connect them with this Not All Equal clause, which is just a triangle. And the idea is if these are-- so this is really a negated version of xi, but it doesn't matter. If you negate all the variables, they'll still be not all equal. And exactly when they're not all equal, we get two points out of this gadget. Well, we also have to get the three points from these connections. That forces this alternation.

So we say the target in this gadget is to get a cut of size 5. We want five red-blue edges. That's the most you could hope for because of an odd cycle here. And if you get the sum of all these cut sizes, in total, you must have Not All Equal in every clause. So that's simulating Not All Equal-- planar Not All Equal 3SAT. This also preserves planarity.

And we just contract the variable to a point, the clauses to a point. And it is the bipartite graph of the Not All Equal picture. And we've simulated that planar Not All Equal 3SAT instance with planar max cut, which here, we actually have a polynomial algorithm. So that gives us a polynomial algorithm for planar Not All Equal 3SAT. So that's kind of fun. Similar style. Yeah. Good. So many problems.

So I guess now we're going to do some reductions from various planar 3SAT things to problems you might care a little bit more about. I'm going to start-- I mean, it's sort of a vague transition, because these, you could think of just more problems of the same type.

X3C is something we talked about in the context of 3PARTITION. This was exact cover with sets of size 3. I think Exact Cover with sets of size 3. So this was-- you could think of it as a hyper-graph.

Or you could think of it as a bipartite graph. You have sets of size 3 which cover variables. You want every variable to be covered exactly once. But you can only choose sets of size 3. Yeah. So this is closely related, I guess, to planar 1-in-3SAT.

Here, each of these things could have arbitrary degree-- each element can appear in many different sets. Whereas over-- and you're covering these with these. I think this is the reverse setup, where every-- let's see. The clauses, again, have to be exactly covered by these things. And these things have arbitrary size. So these are the things that you're choosing. I'm choosing to make this variable true, which covers this guy. These have arbitrary size here. The thing that I'm choosing has size 3. OK? So it's sort of the dual of--

AUDIENCE: What are you choosing?

PROFESSOR: --of planar 1-in-3SAT. So the goal-- maybe I should write down the problem again. You're given three sets. These are sets of size 3. And you want to choose, let's say, k of them. Actually, it would be choose n/3 of them that are disjoint. And therefore, every element-- they're n elements-- every element is covered exactly once by exactly one 3SAT. So it's, I think, sort of complementary to planar 1-in-3SAT. The planar version is that this bipartite graph is planar.

And Dyer and Freeze, this is same people that approved one of these many problems is hard. I think the-- I'll look back. Planar 1-in-3SAT. So no surprise.

So we're going to reduce from planar 1-in-3SAT to this problem, prove that this is hard. And here's a very simple proof. They make it more complicated. But for

starters, let's make a variable by this kind of even cycle trick. And the picture here is that the big circles are 3SATs, the dots are the elements you're trying to cover. And every element should be covered exactly by exactly one set.

So this looks good, because there are exactly two ways to solve this thing. You could choose these guys and cover those points exactly once, in which case, this is covered, this is covered, and this is covered. But these other guys are uncovered. Those are going to attach to other gadgets. Or you could do the reverse. So that's going to correspond to a true or false setting. And I think-- this is not in their paper, but I think this would be a clause for exactly one 3SAT.

Just connect-- you could have even negated versions of your variables. But let's say we have all positive planar 1-in-3SAT. 3, So I'll just take a positive copy of xi, a positive copy of xj from somewhere, and then just bring them together at a common dot. Then that point should be covered by exactly one of them, which means exactly one of those is true. Done. Very easy.

Now, they want to prove more things, so they end up using-- I think I will just show you for fun. They end up using a more complicated clause and a more complicated way to connect these things into the clause, because they want to prove another problem hard, which is planar three-dimensional matching.

Three-dimensional matching was a generalization of numerical three-dimensional matching, which was closely related to 3PARTITION. In general, three-dimensional matching, it's like this problem. But you also have the extra information that for every set of size 3, one of these is red, another one is blue, another one is yellow. This paper uses the additive primary colors. I don't know why. It could be green if you prefer.

So there are three types of elements. And you're told that every set has one of each type. So that's extra information that's useful in some proofs. So this is a better construction, because it's going to end up being three-colorable. Yeah. I don't know how much you care about this reduction. I think I will skip the details, although I spent a lot of time understanding it.

The rough idea is that these connectors-- there's a positive and a negative version, because they're worried about negations. But these connections will attach to these three terminals for one variable, these three terminals for another, and these three for another. And just by a counting argument in here, there is 1, 2, 3, 4, 5, 6, 7, 8, 9 of these guys. And there are however many-- I guess I should really be counting the black dots. 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12 points here.

So the best you can do is to choose three of the sets inside that will cover all of the black dots except for three of them. And so from that, you get exactly one 1SAT. Exactly one of these should be covered from somewhere else. That would correspond to this situation where you cover these three points as opposed to this situation, where none of them get covered. That's like these guys. And then you can always satisfy the rest by adding three in there. So that's roughly how that clause works.

Then that picture is three-colorable. And this way, you can color all the dots with three colors. And it pretty much works. The variable alternates red, yellow. So we always get blue connections, which is good. So we can attach to anything, except that the way that we attach is like this. And these are three colors which match here. And in this case, these three colors match here.

But you might want to attach here, for example. So you need another connector, which is slightly different. This does exactly the same thing, but now this color pattern matches here, but in negated form. And if you switch these two colors, it matches here, I think when it's upside-down. So there's this version, and there's the reflected version. And then there's also this coloring of the same gadget.

And so you do all these things. You know what you're supposed to connect, and so you just choose one of these three gadgets, and it connects to one of the three appropriate places. That's their proof. It was fun to read. And that proves that planar three-dimensional matching is hard. OK.

Plus, in their diagrams, oh, they have dashed lines through everything in order to

illustrate that you can have one path that visits all of the elements, just like in planar 3SAT. So also, both planar three-dimensional matching and planar exact cover by 3SATs, you can have one cycle that visits every element. Yeah. Cool.

Here's another relatively simple proof. This is in the original Lichtenstein paper, so this is one of their motivations for planar 3SAT. Planar vertex cover. So what's vertex cover? You're given a graph. You want to choose a set of vertices that cover all of the edges. So for example, if you have a triangle, and I choose this vertex, it covers this edge, and it covers this edge. I have one edge uncovered. So I'm going to add this one and cover that. So I don't require exactly one cover. Just every edge has to be covered by at least one of its endpoints. So you can think of it as an oar of its two endpoints. OK.

But this problem is NP-hard even for planar graphs. And here's one way to do. This is like the whole proof in one little picture. You have a variable gadget, which, again, is just a even cycle. And in an even cycle, the vertex cover-- let's say this has size k. And I give you the budget of only k/2 for your vertex cover, then you have to alternate. So either these three guys are in your vertex cover, or these three guys are. This proof actually looks the same as the last one we saw, right? Then those things are just connected to triangles.

Same deal here. I'm going to give you a budget of two for this clause, because to cover all three of these edges, you need at least two vertices, like we did over there. And if you're going to get away with only two-- so if I choose, for example, these two, that also covers this edge and this edge for free. But it does not cover this edge. This would be the one edge not covered by the clause alone if you only get a budget of two. And that's the one that better be covered by this side, which means this variable is set to true. And in general, at least one of these three things better be covered. Otherwise you won't have enough budget to finish that triangle clause. Question?

AUDIENCE: So the hexagon versus square here is just to give you more connection points into other clauses?

PROFESSOR: Yeah. Just make these cycles big enough to connect up to all the things you need it in. So if you have ni occurrences, you're going to do, like, 2 ni or something. Cool. So that's planar vertex cover.

And because this reduction preserved planarity, we get planar vertex cover's hard, because planar 3SAT is hard. Here we didn't need any connections between the variables.

Here's one where we do need connections between the variables. So this is planar-directed Hamiltonian cycle. So I think you all know what Hamiltonian cycle is. And you're given a directed graph here. You want to find one path that visits every vertex exactly once. Don't care about edges.

So we can do a similar kind of thing. The proof is a little bit-- but checking this works is a little bit more annoying. But here's Lichtenstein's idea for variable. Essentially, a wire. The graph is directed here. So we get to say there's an incoming edge here. Then you get a choice. So which way it'll go. But then you basically have to alternate. Because of these vertices in the middle, you've got to visit all of them. And the only way to do that is to alternate.

So you get this alternating pattern, which means if you look at these edges, a1 and a1 bar, exactly one of these is in. And then the opposite of these are in. The variable here is called a. So there's many instances of a. And these are going to be in-- they're going to be both in or both out. And then these are going to be both out or both in and so on. So because I have to zigzag, I made a bunch of copies, ni copies of the variable a. But you're free to choose one setting or the other.

And then the clause is just going to be a single vertex. And the idea is if we want b bar to satisfy this clause, we're going to add in those two edges. So instead of going straight here, you could have done this and grabbed the clause for free. You don't have to. But this is going to get grabbed if and only if exactly-- at least one of these chooses the appropriate edge. If you're using this edge, there's no way to cover this clause. But if using this edge, you can do that. You have to check that you can't let go from here to there to over here. That's what this figure is about.

So basically, if you're alternating here, you switch sides, and then you're alternating here, this guy is uncovered. And apparently, this proof works even when the graph is undirected, but that's even less clear. We'll see other Hamiltonian cycle proofs that are stronger than this. But it's a nice illustration. Here we're using that we can connect all the vertices together. That's what these connections are in the cycle.

So there's this big vertex loop on the outside taking these gadgets, and pasting them together in a big loop, because we want an overall Hamiltonian cycle. Because we know that preserves linearity, life is good. OK. So there'll be some clauses inside the cycle, some clauses outside the cycle. But that doesn't matter. OK. That's planar-directed Hamiltonian cycle.

Time for a Nikoli game. I bumped into this slide that I made a while ago. So for fun, these are all the Nikoli games I know that have been proved hard except the one I'm going to talk about. These are the references. We covered one of them, right, the Light Up. But there's a lot. And lots of papers proving them.

So here's a relatively new game. It came out a couple years ago in Nikoli land, although it was invented before that. Shakashaka, which is like a shaking sound. So we have a square grid, blank squares, and obstacle squares. The obstacle squares, some of them have numbers, some of them are wild cards. And what you're allowed to do for a blank square-- or some of the blank squares, you can fill in one half.

So you have like four different halves of the square you can choose. There they are-- this one, this one, this one, and this one. And you can fill it in half black, half white. So you cut along a diagonal, and then you choose one of the two halves to fill in black. You don't have to. You could just leave it white.

And when you have a numbered square, then the number of filled things must be exactly that number. So here, there's one adjacent to it, and nothing down here. This guy has exactly two half-filled squares next to it. This one has exactly one. This one has exactly zero half-filled squares. Think of blank squares as zeros and these as counting as one.

Plus, the goal is that the regions you make-- this would all be easy so far, I think. But the extra constraint is that all the regions you make must be rectangles. OK? This is a 45-degree-rotated rectangle. This is a regular rectangle. So you can use either one. So it's a funny constraint. It's actually quite a fun game to play. You're not allowed to fill in a square 100%. That one was already filled. That's the rules.

Here's a reduction from planar 3SAT. Pretty simple. Looks a lot like Minesweeper, in some sense, a little bit thickened. But this is a wire, because the 1 says that one of the two sides is half-filled. Once that's half-filled, you have to make this a rectangle. You can't just leave it as a pentagon. So you've got to fill that in to a rotated square. But because of the 1, this must be an empty square. So it alternates. You can do this or this. OK.

You can easily split just by connecting those things in the obvious way. You can do 90-degree turns. No problem. Here's a slightly more sophisticated-- slightly more sophisticated gadget. You can think of this as a terminator. It would be hard to just stop this somewhere. Well, maybe you could just go all black. I think that's OK as well. This could also be a terminator. You can also think of this as a negator. Of course, also, the splitter's like a negator. This guide's just probably not so necessary.

Let's get to Clause. So Clause is bringing three things together with one of these blocks. But we do it in a funny way. There's no 1's here or here. So we end up with this L blank shape. Everything else is now drawn as black. That's just obstacles. So this is not happy the way it is. There's lots of ways to resolve it. The one case where we can't resolve it is when all of these wires are 0's, because there's already stuff here, and stuff here, and stuff here, so we can't put anything here. Can't put anything here, or here, or here, because of those 1's. Then you're forced to have the L shape, and you're toast.

In every other case, we can decompose into sometimes a big rectangle, sometimes little squares, sometimes rotated squares. But that's OK. So this is a regular 3SAT clause.

The one thing that's missing at this point is a parity shift gadget, because a lot of these gadgets have very specific lengths, modular 3 or whatever. So it turns out this is a way to shift things slightly. It can be filled in two ways. You can fill in all of these guys like that and make this giant rotated square. Or if these are not allowed because these are here, then you can do this weird filling. You end up with a rectangle there, and two rotated squares, another rectangle. And you count the total length. This is length 4, whereas most of the things are length 2?

AUDIENCE: 3 [INAUDIBLE].

PROFESSOR: 3. OK. So that makes me happy. Shift the parity. And that is Shakashaka. Here's an overall picture where we actually plugged some gadgets together. That was fun.

So I think I have one more proof to just sketch. So this is a problem of so-called fixed-angle chains. So you have this forced 90-degree angle. We had a similar thing with the Hamiltonian path with cubes and elastics between them. You could twist one edge around the other, one block around the other, but you couldn't adjust this angle. OK?

So the problem is I give you such a chain, which is specified by lengths and then 90-degree angles. So it can be-- you don't know whether the 90-degree angle goes up or down. And you want to just put it into the plane without crossings. And this is an old proof from 2000. The problem is weakly NP-hard. It's basically a partition proof, so it is a partition proof. You get to choose, for each of these things, whether it goes left or right. And you need to line up this with that. Otherwise you get collisions. You would just need to argue. This outer structure has a unique embedding, roughly. Unique enough. It always looks like that.

So we proved a few years ago that it is strongly NP-hard. And I will just sketch the proof, because it's fun and cool. So here's the rough idea. Imagine this is like a piece of wire. And you can flip it up or down. All the angles stay 90 degrees. These are the down-case. Here are some example up-cases. One of them's going to represent false, the other true. Question?

AUDIENCE: What's the difference between this and the Carpenter Ruler problem?

PROFESSOR: Which Carpenter?

AUDIENCE: Like you have to fold the ruler to fit.

PROFESSOR: Oh, to fit inside a given length. Well, the goal is different. Here, the goal is to draw it in the plane without crossings. Before, the goal was to draw it in the plane with minimum span. To fit inside box. Yeah. I mean, this proof is pretty much the same. This proof will be totally different. That problem was also only weakly hard. There was a pseudo poly-algorithm. This problem is strongly NP-hard. There is no pseudo poly-algorithm unless P equals np. So just a slightly different goal. That one was more one-dimensional. This is going to be much more two-dimensional That's the other difference. OK.

So this is our some kind of variable gadget. This is the variables gadget. All n variables are represented here. This represents one variable. At this point, I only have one copy of each variable, because there's no constraint between these. Each of them can independently flip up or down. Next, we have a clause. This is where things get fun. So there's some infrastructure, but the main action is here.

Each of these is sort of independently pop in or outable. It's like some kind of pleated form. It can either pleat back and forth, like it's doing here, where it pleats in and out and out and in. And that gets this to this position. With a slightly different pleating, it ends up in this position. Or a slightly different pleating, where they're all out, you end up in this position.

And notice this structure. So I'll go through it again. Here's the left one. And then the middle one. And then the last one. What ends up happening is that these pegs, in order to avoid hitting that peg, I occupy either P1 or P2 or P3.

So I'll go through it one more time. Here it's hitting P1, because this basically pushes this thing down. Whereas the other ones can be up-- this is up and up and up and up. Or if we shift it into the middle, then this one has to be down. But this guy can be up and avoid P1 but hit P2. Or this one's down, these are up, and we hit P3.

We want to avoid collisions, so we have to collide with exactly one of these things, or at least one of them, I guess. So that's the sort of picture. This is a bunch of those clauses. There's one here, there's one here, and there's one here. Look familiar? This is planar monotone rectilinear 3SAT, I think. So far so good, except this chain is separate from this one, so we're going to connect them together. Is a little more, I think. Yeah. Connect them together.

So now it is one big chain. What I ended up doing, there's still this big thing to represent the entire variable. But I basically made a whole bunch of copies of it, so that I could attach things here. But for example, when this is up, all of these guys have to simultaneously be up. So it forces all the copies of the variable here to be one way or the other. This is one variable, this is another variable, this is another variable, and this is another variable, just like before. So same thing, but now I can connect them all together.

Up here, these are all the negative, I think, clauses. Doesn't matter. And then down here, you put all the positive clauses. And that's the proof. Question.

AUDIENCE: --all those three dots in a row? Like, it seems to me that if you have two dots [INAUDIBLE] straight line, they might as well just merge into one dot.

PROFESSOR: I think the idea is, yeah, this could be a single line. We subdivide it just to point out that you can attach things to it. Yeah. There's extra dots, but just for consistency's sake, I guess. So now everywhere here, you can attach one of the clauses. And that way, they all kind of hold to the same infrastructure. Yeah. Clear? [LAUGHS]

So obviously, you get pretty complicated pictures in the end. But this is a proof that clearly benefited from the rectilinear aspect and the monotone aspect that all the true thing-- we could not get some true above and some true below. They all wanted to go the same way. And the rectilinear was nice, so we could draw this all on a horizontal line and only have to worry about vertical connections between the top and the bottom. So that gets to your question about, why do we need rectilinear? But there you go.

So that was just a few examples of planar 3SAT. But in general, the idea is that a lot of 3SAT-- or many 3SAT proofs get easier when you use the appropriate planar version. You just have to be careful of ones that are not hard. As long as you avoid that and you have the right sort of graph setup, you don't have to add too many extra edges. Then you'll have planarity and not have to worry about crossovers. And crossover, it wouldn't have a clue how to do a crossover in a problem like this. And other times, it just makes your life a little bit easier, because you skip one gadget. But that is planar 3SAT.