What we're going to talk about today, is goals.

So just by way of a little warm up exercise, I'd like you to look at that integration problem over there. The one that's disappeared.

So the question is, can you do it in your head? Probably not. The question is, if a program can do that, is a program, in any sense of the word, intelligent? That's a background task I'd like you to work on as I talk today.

So today we're going to be modeling a little bit of human problem solving, the kind that is required when you do symbolic integration. Now, you all learned how to do that. You may not be able to do that particular problem anymore, but you all learned how to integrate in high school 1801, or something like that. The question is, how did you do it, and is the problem solving technique that we are trying to model by building a program that does symbolic integration, is that a common kind of description of what people do when they solve problems.

So the answer to the question is, yes. The kind of problem solving you'll see today is like generating tests, which you saw last time. It's a very common kind of problem solving that we all engage in, that we all engage in without thinking about it, and without having a name for it.

But once we get a name for it, we'll get power over it. And then we'll be able to deploy it, and it will become a skill. We'll not just witness it, we'll not just understand it, we'll use it instinctively, as a skill.

So there you are, you've got that problem, there's your problem, and what do you do to solve it? I don't know, look it up in a table? You'll never find it in a table because of that minus sign and that 5. So you're going to have to do something better than that.

So what you're going to do, is what you always do when you see a problem like that. You try to apply a transform, and make it into a different problem that's easier to solve. And eventually, what you hope is that you'll simplify it sufficiently, that the pieces that you've simplified to will be found in some small table of integrals. So how long is this table? It's not the case that we're going to look at a table with 388 elements, because this is not a big table of integrals. This is what a freshman might have in a freshman's head, after taking a course in integral calculus.

One of the interesting questions is, how many elements have to be in that table to get an A in the course? We're interested in how much knowledge is involved, that's one of the elements of catechism that I've listed over there, that will be part of the gold star ideas suite of the day.

So we'd like to take that problem, and find a way to make it into another problem that's more likely, or closer to being found in the table. So what we're going to do is very simple, graphically. We're going to take the problem we're given, and convert it into another problem that's simpler. And we're going to give that process and name, and we're going to call it problem reduction.

And so, in the world of integral calculus, there are all sorts of simple methods, simple transformations, we can try that will take a hard problem and make it into an easier problem. And some of these transformations are extremely simple and always safe. Some of them are just, well let's try it and see what happens. But some of them are safe, and I'd like to make a short list of safe transformations right now.

Now I'm going to be going into some detail. And that detail will be grungy. And the question is, why do I do it? And it's educational philosophy, is why I do it. So here's the educational philosophy. At one level, you want to have a skill. But if you're going to have a skill, you have to understand it. So if you're going to have a skill you have to understand it one level down. If you're going to understand it, you have to have witnessed it on a level lower than that.

So I'm not just going to talk about the idea of problem reduction, because if I were just going to do that, then we could all go home now. So I'm going to show you a particular example of it, so you understand it better, and I'm going to show you the detail at an even lower level than that. So you will witness the stuff that makes it possible, to understand the stuff that makes it possible, to build a skill. So that's why I'm going through the grungy detail.

So I don't know, let's see. Maybe we can get some hints from that example, but I wonder if somebody could volunteer a simple transformation that always is a good thing to do. Yes, Sebastian.

AUDIENCE: Take the constants out.

SPEAKER 1: Take the constants out. So we'll make that number two. And we'll say that the integral c f of x dx is equal to c times the integral f of x dx. Other suggestions? Yes.

AUDIENCE: Trig substitution.

SPEAKER 1: Trig substitution. Now this is-- no, that's for day two. We don't do trig substitution here under stuff that's safe, always works, never any doubt, there are simpler things.

These are the safe transformations. What you're giving me is a heuristic transformation. Often is helpful, doesn't necessarily always work. We're going to divide our transformations into those two categories. So I need another safe one.

AUDIENCE: [INAUDIBLE] SPEAKER 1: The architects are sitting over there. Divided not only by nationality, but by course. What?

AUDIENCE: The sum of integrals is the integral of the sum.

SPEAKER 1: The sum of integrals is the integral of the sum. Now what's missing? What's number one? You're probably thinking it's already there, because you've given me the transformation that involves a constant. And you can think of minus 1 as a constant.

But whether you use a separate transformation or not, of course depends on how you represent the knowledge. And all of this knowledge, all of this whole thing, was written in an early form of Lisp. As a consequence, the way in which minus was represented is different from the way minus 1 is represented. So we need one more transformation. Or rather, Jim Slagle needed one more transformation, when he wrote his famous transformation program.

And that was that if you have the integral of minus f of x, that's equal to, minus the integral of f of x.

So that almost completes our safe transformation set. There's one more that I'm going to supply you, because I don't think you'd guess it. Why should you? It's number four. There are more than this, this is a sample. And these are the ones we're going to need in order to solve that problem, by way of illustration.

So the fourth one is that, if you have the integral of p of x, over q of x, then you divide. If you can reach way back into high school and figure out how to divide polynomials.

But if the degree of the numerator is greater than the degree of the denominator, then it's a knee-jerk always win, you must do it, divide it out.

So this, then, forms the core of an integration program, that will integrate almost nothing.

But actually, almost nothing is integrable anyway, so it's a good head start. So let's see how we would put this into some kind of procedure. Some kind of framework for deploying the knowledge that we're beginning to develop.

What we're going to do is, apply all safe transforms. That's our first step. Then we're going to look in the table, and then we're going to do a test to see if we're done. And if we are, we report success. But, we're not likely to get done with just that stuff.

But you know what, there was one transformation up here, which breaks my little diagram. Which one is it? It's the third one, right? Because this picture does not reflect what happens when you apply number three. Because it breaks the problem up, not into just one problem, but into a whole bunch. So we have to extend our graphical device for talking about this by a little bit, and show what is called an "and node".

So we've got a program core, we've got a table of integrals, we've got a few transformations, we've got an architecture, a way of putting that stuff together. And now we can try it out on our sample problem. So let's have a go at that.

Let's see, this one immediately transforms into 5x to the fourth over 1 minus x squared to the 5/2 dx. And that in turn, immediately transforms into the integral of x to the fourth over 1 minus x squared to the 5/2, dx.

This program, by the way, is a dawn-age program. This was written by a nearly blind, and subsequently completely blind, graduate student by the name of James Slagle in 1960, a long time ago. The reason I gave it to you today is because, that by describing it, I am giving you a one-lecture course in artificial intelligence. He anticipated so much of the subsequent 20 years, that talking about his program, which is possible in one day, is a miniature introduction to the whole field.

So Slagle, as he was doing this on an antique computer, almost no memory, almost no speed, only slightly faster than mice running around on a treadmill. He was able to write a program that did extremely well when benchmarked against freshmen. And the way you benchmark against freshman, of course, is you give it an examination, drawn from the previous MIT finals for four or five years, the hardest problems. And this was the hardest problem that it solved.

So at this point, with what we've got so far, we would be stuck. We have no transformation that can take us further, so we need something else. And what we need by way of something else, is some transformations that we will describe as-- perhaps we'll call them, heuristic transformations. A funny word, meaning a method that often works isn't guaranteed to work.

It's not an algorithm in the usual sense that we talk about algorithms. But rather, it's an attempt.

So these things I'm going to talk about now, are sometimes useful, not always useful. Sometimes take you into a blind alley, don' always work. But you can't get an A in calculus without knowing some of them. So you said, some kind of trig substitution. So here is some kind of trig substitution. We'll call this heuristic transformation A.

You have a function sine x, cosine x, tangent of x, cotangent of x, secant of x, and cosecant of x. And we all know from high school trigonometry, that we can rewrite that as a function of sine x, and cosine x. Or we can rewrite that as a function of tangent of x, and cosecant of x. Or we can rewrite that as function of cotangent of x, and the secant of x. So that's a transmission from trigonometric form, into another trigonometric form. It's not always a good idea, sometimes it helps.

Well that's just part one of our suite of heuristic transformations. Stop. There are others that we need to have in our repertoire, in order to solve the problem. One of them is a family of transformations, which I'll show you only one. It goes like this, if you have the integral of a function, of the tangent of x, then you can rewrite that as the integral of a function of y over 1 plus y squared dy. So that's a transformation from a trigonometric form into a polynomial form. So it gets rid of all that trigonometric garbage we don't want to deal with. And there's a whole family of things like that, just as there's a family of transformations like so, but this is enough to give you flavor.

Now there's a C that we need as well. And that's going to be your proper knee-jerk reaction when you see something of the form 1 minus x squared. What do you do when you see that?

AUDIENCE: [INAUDIBLE] What's that Rhana?

Rhana: 1 + 6 * 1 - 6 Well wait a second. We could do that. But there's another thing we can do.

Christian, have you got something you can suggest? Where's our Hungarian? Our Turk, our young Turk. Yeah, what do you think?

AUDIENCE: I actually don't remember. I mean, I think it might have been 10.

SPEAKER 1: Well, let's see. Cosine squared plus sine squared equals 1. So, what's that suggest to you? So it suggests that we make a transformation that involves x equals sine y. So [? Silla ?] doesn't actually have to remember that anymore because going forward, she will never have to integrate anything personally in her life, she can just simulate the program.

So these go from polynomial form, back into trigonometric form. So you have three of these heuristic transformations. We've got four safe transformations. Let's see if we can make any progress on our integration problem.

OK so keeping track of what we've been using, this is safe transformation number one, this is safe transformation number two. What do we do next? We decided there were no more safe transformations that apply. But now we can look at our heuristic transformations and behold, we see what?

AUDIENCE: C SPEAKER 1: What?

AUDIENCE: Applying transformation C.

SPEAKER 1: Transformation C suggests that we do x equals the sine y.

And now we get the integral of sine to the fourth y over cosine to the fourth y dy, right. All good, I see some confused, worried, concerned looks. Maybe I've made a mistake, perhaps I should use notes. Well no, wait a minute.

For those of you who have a concerned look, remember that if x equals a sine y, then dx is equal to cosine y dy. That's why it's cosine to the fourth not cosine to the fifth, as you were perhaps thinking it might be.

So now we've made some progress. We look at this, we say, are there any safe transformations that apply? And the answer is, no. Now we look for a heuristic transformation that might apply, and I say, what do you see? Which one? What's that?

AUDIENCE: [INAUDIBLE].

SPEAKER 1: She said something unintelligible, but what she probably said is, that this looks like a pattern that might match with the heuristic transformation A, right? Because we have a function in which the variable is buried, universally in sines, or cosines, or tangents, or cotangents, or secants, or cosecants. And we know we can rewrite that in one of three ways. It's already written as a function of sine and cosine. But we can also rewrite that in terms of tangent and cosecant. Or cotangent and secant.

So when we do that, we can go this way, and we can get the integral of 1 over the cotangent of x dx. That's g3 up there. Or we can do it down this path, and get the integral of tangent of x dx. And of course, those are both to the fourth.

But know what, I've broken my little graphical diagram again. Where did it go, it's disappeared.

There it is. How have I broken it? Because with transformation A, I've introduced a possibility that a particular problem can be transformed into more than one kind of problem, any of which will be the solution to my problem.

So far I've got an and node, but now I've got to introduce an or node. Because now we have an example of something that can be solved one of two different ways, and we don't care which one it is. Now you'll notice that there's already some confusion here, because how can you tell the difference between an and node and an or node. So the universal convention is, you draw an arc over the and nodes. And that makes it look like an A, so it's easy to remember. So those are and nodes.

And now, we have the method of problem reduction, and this is sometimes called a problem reduction tree. Sometimes it's called an and/or tree, and sometimes it's called a goal tree, because this tree of problems is a tree that shows how our goals are related to one another.

So these are items for your vocabulary that are all synonymous. Problem reduction tree, and/or tree, goal tree, all the same thing. Now you have a name for it, you've got some power over it. So when we get a situation like this, unlike the previous situation, which we suggested might come up in transformation A. Let's see, we've got one, two, C, and this one is A, it's an or node. Which one of these problems do we work on?

Well Slegle, who considered himself to be modeling a freshman, modeling the intelligence of a freshman, modeling something that, after all, you have to be pretty smart to do, right.

Most people don't know how to do integration. Everybody at MIT knows how to do integration.

You would think that somebody, therefore, that knows how to do integration is pretty smart. What would a smart person do, when faced with this choice?

Well, a smart person would say, which of these two problems is easier? So how do you think you might determine which of two, or many algebraic expressions is the easiest to integrate?

What's your name?

AUDIENCE: Andrew Carrol.

SPEAKER 1: Andrew, what do you think?

AUDIENCE: Based on whichever one feels more familiar.

SPEAKER 1: Feels.

AUDIENCE: Yes.

SPEAKER 1: Feels.

AUDIENCE: You asked, how would I decide.

SPEAKER 1: Yeah, how would you decide? How would you feel it?

AUDIENCE: I would feel that the tangent is more familiar.

SPEAKER 1: Which one?

AUDIENCE: I feel that the tangent [INAUDIBLE].

SPEAKER 1: Yeah, but I wonder how we could make it a little bit more precise, this idea of simplicity. The young Turk has a suggestion. What?

AUDIENCE: I had a suggestion until you said this idea of simplicity. So then I realized that what I was about to suggest wasn't going to clarify simplicity, but I was going to say, whichever one we've had more encounters with, or more experience with.

SPEAKER 1: Yeah, if there was something here with a hyperbolic tangent, you might say, well, stay away from that. [? Yinid ?]?

AUDIENCE: To which one of those the easier transformation is applied on the next step.

SPEAKER 1: Like, somebody do a little look ahead, and see which kind of thing would be next to you? I don't know, maybe. Oh, we've got lots of people, all at the same time.

I don't know all your names yet. Shoot. Erica, I know you.

AUDIENCE: What's look it up in the table and see [INAUDIBLE].

SPEAKER 1: Oh, you could look it up in the table and see if something is in it, you could do that. But this is tangent to the fourth, so that's not in the table. Ariel?

AUDIENCE: I choose the one without the reciprocal.

SPEAKER 1: Why?

AUDIENCE: It is because when people see one it's like, oh man, it jut not going to work.

SPEAKER 1: Yeah, we're on the right track. Claire?

AUDIENCE: On an extremely simple level, I choose whichever one has the least symbols in it.

SPEAKER 1: The fewest symbols in it. Now we're really getting somewhere, because you can measure that, right, there's a little program Why Brett, there you are.

AUDIENCE: I would say, every [INAUDIBLE] expression can be written as, having a number of functions, we could say all these functions, multiplied together, divided, and you can just choose with the least amount of [? iterations ?].

SPEAKER 1: Well I heard it, perhaps others didn't but what Brett said, is he suggested that we should measure depth of functional composition. So the number of symbols may not matter, because if you have x plus x plus x plus x, out to a hundred, that would not be hard to integrate. But if you've got something that is really deeply nested under a lot of functional compositions, that could be a problem. And that's in fact, what Slegle decided to use, after trying several alternatives.

So if we measure the depth of the functional composition, this is the winner, and we put the other one on the shelf, at least for the moment. And now we have tangent to the fourth x dx. Do I need the safe transformation supply?

No. Which of the-- you know something has to apply, otherwise it wouldn't be up here as an example. So what of the heuristic transformation supply? Elliott.

AUDIENCE: [INAUDIBLE] SPEAKER 1: Yeah, B bravo. Military background or something like that. Maybe he flies airplanes. OK so B says, it is in fact a function of the tangent. And when we do that, we've got to make a substitution, that y is equal to the tangent. So that means that this becomes the integral of y to the fourth over 1 plus y squared. And that's by transformation B, and the transformation is y equals tangent of x. The tangent-- I guess I've lost track of the fact that I've already transformed a y, but relabeling doesn't matter.

All right so that's progress, maybe. But don't see this in any of the heuristic transformations, what do I do now? I didn't have to look in the heuristic transformations, because one of the safe transformations applies. Because this thing is a rational function and the degree of the numerator is greater that the degree of the denominator, so I have to divide.

And when I divide, and that by the way is number four, I get what? Is anybody good high school algebra that can help me out with that?

AUDIENCE: Y squared minus 2 plus negative 2 over 1 plus y squared SPEAKER 1: Exactly, y squared minus 1 plus 1 over 1 plus y squared, I think. Now what?

Now we're really getting close to getting through this, because that is a sum. And by virtue of the fact that it's a sum, that divides into three pieces, and the top piece is the integral of y squared, the middle piece is the integral of minus 1, and the bottom piece is the integral of 1 over 1 plus y squared dy in all cases.

Gosh, if I look this up, I've found it. That's up there, that's letter B. So I'm done with that. This one I can transform again, by virtue of 1, and now I get the integral dy. That's in there, that's B as well. As this one, I don't know. But I'd better keep track of what I'm doing here. This is in the and node, so I've got to do all of those. I can't give up on that last thing. And that and transformation is transformation number 3. So this is in the table, this is in the table, we still have this to do, but that's C, heuristic transformation C. We have 1, plus y squared, then with the transformation C, with y-- this is y squared-- y equals tangent of z And then we get to the integral of dz and that's in the table and, we're done.

So now we've solved the problem. It's the hardest problem that appeared in that half decade on MIT 18 01 finals. This is exactly the problem that was given, except that it started here. I put the other two pieces on just to illustrate a couple of the transformations.

But that's a problem that it solved.

And now that we've seen an example, we can finish up what we talked about a little bit ago, having to do with the architecture of this thing. So far, all we've done is talk about the safe transformations, but now we know that if we're not done, we need to find a problem to work on using that depth of functional composition business. And then after that we apply heuristic transformation.

And the way Slagle designed his program is, he found just one problem to work on, did one transformation, then went back around the loop. Because these heuristic transformations are a little harder to apply than the safe ones. So I'll given you an accurate portrayal of what this program did, except for one thing. Which I would like, now, to go back and patch up. And that thing is over here. What to do with something like this. Well we got to that in a board that's disappeared, but when we tried to deal with this, we had to find a heuristic transformation.

And when we decided to work on this, it must have been the case that this was the simplest problem at a leaf node that has not yet been solved.

So what's the functional composition depth of this? It's 3. Back over here, we have something that has a depth of functional composition of 2. So when the program actually ran on this particular problem, it stopped a few inches short of the finish line, And went back and screwed around with that other problem for a little bit, before it gave up and came back here.

So it's always looking across the whole tree, the leaves of the tree. Whenever it has to find a place to work on with the heuristic transformation, it happened to look at all the leaves of the tree that had not yet been dealt with, tried to find the easiest one, and that could involve a lot of backing up and starting over on a branch of the tree that it had previously ignored. A small detail, not a particularly important one.

Now where are we. We've got that guy there. We've got our complete architecture. We've got our solved problem. And now we can start reflecting on what we've done. We can say, for example, how good an integration program is this? And the answer is, it was pretty good. This machine that Slagle was using was a machine that was over in building 26. And we were so proud of it, that it was behind glass, and you could go there and watch the tape spin, it was really a delight. 32k of memory, that's 32k of memory. It's amazing that he was able to do anything with a machine of that size.

Let's see, let's get us a clean one. Can't do board geometry and talk at the same time. We can now ask some questions about how well the program performed. It was given 56 of the hardest problems, and it got 54 right. What happened when it didn't get the other two? Well, you might be right if you said, oh it probably ran out of memory, since it had 32k. But in fact, it just was lacking 2 transformations that were needed, in order to solve the whole entire set of final quiz problems. So when a program fails, that's often the most interesting question you can ask. This is an exception. This failed for uninteresting reasons on 2 of the 56 problems that it was given to.

And now the next question you can say is, what is the depth of the tree in the maximal case? And the answer is, it's that case we just worked out. And since I've once again lost the whole tree, I'll tell you that it's depth was 7 when you take off that minus 5.

So in the worst case, this thing had to get down seven levels.

That's the worst case, a more interesting question is what was the average depth? And that was approximately 3. And now we're beginning to say something, not only about Slagle's model of how a freshman works, but we're beginning to say something about the nature of the domain.

In the domain of calculus problems, integrals expressions that are given to freshman, in that domain, the average depth of problem reduction needed to solve the problem was 3. So that's not very complicated. If it were 10, you would say, wow, how can anybody ever do those problems? If it were 5, you'd say, well only people destined to be math professors are going to get anything right. If it's 3, us ordinary mortals can do a pretty good job.

Another question of even greater interest is, how many branches were unused? Here's a branch that turned out to be unused, it didn't pursue that. And so you might say, well maybe there are a lot of unused branches. Maybe you have to be pretty smart about your method for determining what problem to work on, because otherwise you'll go down a lot of rat holes.

And guess what, here's another statement about the domain. In the domain of problems that freshmen could work on a final, the number of unused branches is about 1. So that means this tree keeps itself together, and doesn't run down to a very large, bushy, useless tree.

So this means that the depth of functional composition, which Brett suggested as a technique for recognizing the right problem work on, was a choice that didn't actually matter.

Because the tree doesn't grow deep, it doesn't go broad. It doesn't matter what you use to decide what to work on, because in the worst case, you'll just generate a couple of extra, useless nodes. But they very quickly run to find a dead end, so you don't have to do anything more with them.

So now the next thing we need to do is back even further away from this program, and ask ourselves some questions about the nature of what we've been doing. And that brings me to the things I've got on that upper right-hand board. One of those things as a catechism having to do with knowledge.

And what we've done informally as we went through this program was, we've asked questions such as, what kind of knowledge is involved in doing this? Well knowledge about transformation.

Knowledge about how goal trees work and when we're done with a problem. Knowledge about what things don't need to be transformed, because you can look them up in a table. That's the kind of knowledge that is involved in doing 18 01. And if you do 18 0 circuit theory, 6 0 circuit theory or 6 0 Maxwell's equations, this is the same thing.

You have to ask questions of this sort, about the nature of the knowledge involved, and question number one is always, what kind of knowledge is involved? Is it Kirchhoff's laws, Maxwell's equations, what is it?

The next question is, how is the knowledge represented? And our answers here are, well all this stuff, ultimately was represented in list best expressions. Some of the knowledge was recorded in a table [? of best ?] expressions to show what transformations there are. There was a similar table of integrals. Knowledge about goal trees was embedded in the procedure, so it was procedurally represented. And so for each of the categories of knowledge, there's a way it gets represented. How is it used? Straightforward, transformations are used to make the problem simpler. The table is used to trim off and to serve as the bottom of the tree. Those are the ways in which the knowledge is used.

And then there's the question of course of, how much knowledge is required. Something that's useful to know if it's late at night, you have 2 finals the next day, and you're not sure which course you should study. So how much knowledge might you suppose was actually in this program? I've shown you a glimpse of the kind of knowledge that's involved in the program. I've answered a little bit of question 5, what exactly. But how much knowledge was involved. You might be surprised by the answer.

First of all, the table of integrals. I've listed only 3 things there. There are lots of other things you can think of, like integral of e to the x is e to the x. But in the end, what Slagle found is, a table only 26 elements was enough to solve all of these problems.

How about the transformations here, the safe ones, about 12. How about the heuristic ones, about 12. So just a few bits and pieces of knowledge, here and there, are sufficient to do everything you need to do, in order to do the integration problems on a calculus final. That was a surprise.

Another surprise of a similar kind, also about knowledge, is that the relationship between the method to be used, and the characteristics of the problem, was almost a diagonal table.

That means that you could, in this domain, make the right transformation almost all the time if you're a little bit smart, and never back up. That was an observation made by Joel Moses, who became subsequently our provost here at MIT for a while. And he wrote a program that could solve anything. It would beat the most dedicated mathematicians at integration.

And its descendents are in MATLAB today.

But this is how it all works. And now you can write one of these things yourself. Partly because you now have this catechism. This is the kind of stuff you should ask any time you're dealing with a new domain. It will make you smarter. And this is of course, meta knowledge, this is knowledge about knowledge. So this tired aphorism isn't quite what we are going to complete ourselves with. We're going to say that knowledge about knowledge is where the real power is.

Now there's one final thing that this program does for us. It tells us something about our appreciation of what it means to be intelligent. You know that in the beginning of this hour, I asked you to think about whether a program that could do symbolic integration would be, in any way, or should be considered to any degree, intelligent. And I'm imagining that even in these days of MATLAB, and whatnot, many of you said well, yes, I learned how to do that at MIT, or late in high school, so it must be smart.

But now that we've completed this discussion, I also expect that your feeling of intelligence in this program is somewhat diminished. Because what happens is that, when we understand how something works, it's intelligence seems to vanish. You've seen this in your friends, right? They solve some problem, they seem super smart. Then they tell you how they did it, and they don't seem so smart anymore.

So let's conclude our discussion today was a little story. A long time ago I was talking with a student who said, computers cannot be intelligent. And I said, OK, maybe you're right, but let me show you this program. So I showed him the integration program, working on problems like this. And after I showed him a couple of those examples, he says, well, all right, I guess maybe they can be intelligent. I'm learning how to do that, and it's not always easy. Then I made a fatal mistake. I said let me show you how it works, and we spent an hour going through it like this. And at the end of that time, he turned to me and said, I take it back, it's not intelligent after all. It does integration the same way I do.