PROFESSOR: Hello and welcome. So today I want to start to talk about discrete time systems. But before beginning there, I want to say just a quick word about homework. I tried to stress this last time that the way you get to know this stuff is by doing the homework, and we'd like to provide positive feedback in a direction that will help you get the most out of the homework.

So to that end, we've already talked about there's two kinds of problems. Each homework will have tutor problems that are intended to be problems like you will see on an exam, but they will also have the feature that they can be automatically graded so that you can get immediate feedback about whether you're doing it right or not. There will also be harder questions questions that look more like real world questions so you don't think that this class is only about toy problems. The idea is to give you problems that you can imagine running into. Those are harder to grade automatically so we won't. Instead they'll be graded by a human.

Those problems often admit several different lines of reasoning. So there isn't necessarily a unique right way to think about it. So what we would like to do is encourage you to-- after you finish the homework-- read the solutions. Now we think it's kind of an impediment to wait for a week till you get your graded homework back because people-- in my experience-- just don't bother. So what we would like to do is post the solutions immediately after your solutions are due. Your solutions are due-- if you submit electronically-- at 5:00 p.m. on the due date. So homework one is due tomorrow, 5:00 p.m. electronic submission.

You can also turn in paper. You have to do that in your recitation. Immediately after it was due at 5:00 p.m. we'll post the solutions, and we'll encourage you, in quotes, to read them, because it's good for you, right? We hope that that will help you find the errors that you made, but also expose you to other ways that maybe will be simpler, maybe will be harder. Your feedback on your coming up with a simpler way would be greatly appreciated, so please send me email anytime you think you have a better way of doing things, which is entirely possible.

But as further encouragement, if you mark up all of your errors and resubmit, we'll give you back half of the points you lost. So you get to submit every homework twice. Once for the due date 5:00 p.m. on Wednesday. Immediately after that, you'll be able to see the published solutions. You can circle all the errors and say algebra, or conceptual, or I'm an idiot, or whatever is the right way of thinking about what you did. And if you get all of the errors, you'll get back half of the points you lost. If you say, I'm right, you're wrong, maybe you won't get all your points back. Yes?

AUDIENCE: [INAUDIBLE]

PROFESSOR: The resubmission is the same rules. If you want to submit by paper, you can turn them in on Friday's recitation. You have until Friday at 5:00 p.m. otherwise. Please notice that, if you're using paper, we will not have returned your solutions in time for you to write on them, so make a copy. If you want to do the paper route, make a photocopy before you turn it in the first time, so that you have something to write on for the second time. If you submit electronically, there's a feedback button on the tutor, so you can see the thing that you just submitted. So that's not a problem if you're doing the electronic thing, but if you're doing paper, please, make a photocopy before you submit. Otherwise, it will be hard to mark up your solutions. Yes?

AUDIENCE: [INAUDIBLE]

PROFESSOR: No, but you should probably tell us what you'd like us to do. We would normally try to make our best guess. So if you turn in everything in the paper copy, we will look at your previous submissions, but it would be best if you said some remark like, this overrides the electronic submission, or please ignore problems 1 to 4, just make sure we know. We're going to try to do what you want us to do. Other issues about logistics? Yes?

AUDIENCE: So just to make sure, we don't have to submit homework both electronically and in paper?

PROFESSOR: No, absolutely not. If you do the electronic route-- which is the preferred, we like electronic because it won't get lost-- the graders will absolutely have access to it because they're all in one place. We like the electronic route, but if you'd prefer paper, just make sure you give us paper and one or the other suffices. Yes?

AUDIENCE: Did we ever submit the tutor questions on paper?

PROFESSOR: You don't need to. If you submit the tutor problems online and it says you're 100% correct, you're done. We don't ever need to see anything else. You've got full credit. Other issues? Yes?

AUDIENCE: If people are having technical problems with the tutor, please tell us. Don't just feel like you have to kick out and do the paper version. We're trying make sure that everything is in place this week.

PROFESSOR: We will try to be very responsive. As I told you, this is a brand new tutor. We're very optimistic about it, but we're also realistic. Software has bugs, but don't tell any of my software people-- my software friends-- that. Software has bugs, so we're trying to get rid of all of ours. Yeah?

AUDIENCE: [INAUDIBLE]

PROFESSOR: I'm sorry?

AUDIENCE: [INAUDIBLE]

PROFESSOR: You have to shout because my hearing is gone.

AUDIENCE: What happens if you run out of submits on tutor?

PROFESSOR: If you run out of submits, that's bad. During this sort of ramp up phase, just send us an email, because the goal is not to have you run out of submits yet. Yes?

AUDIENCE: So if you run out of submits and your last submission was an incorrect answer and [INAUDIBLE] that has the correct equation for that answer, will you still get full credit?

PROFESSOR: If you continue to hit submit, we will continue to be aware of it. It's all in the grade. We know everything you've done. So just go ahead and hit submit some more times. You can always turn in paper to override it, or you can always send me an email and say, I ran out of submits, the answer is blah. We're going to try to be very cooperative. This is new. This is supposed to be helpful. Yes?

AUDIENCE: [INAUDIBLE]

PROFESSOR: That's not supposed to be on your website. That's supposed to be on my website. I'll have to look into that. If it works, use it. OK, other obvious errors that we've made?

OK, let's go on and talk about 003. What I want to do today-- last time, I introduced what I call the 6.003 abstraction which is the idea that we represent a system-- whatever it is-- by the way it transforms an input signal to an output signal. The goal for today is to start to develop machinery that takes advantage of that representation.

We'll start by thinking about DT. DT is conceptually easier than CT. That's why we start there. Algebra is slightly easier than calculus. That's the idea. What we learn in DT will transfer almost verbatim into CT. CT is like DT but a little harder so there is some extra stuff, but the things we learned in DT will still be applicable in CT. Discrete time, continuous time.

Furthermore, DT is just important. So, increasingly, applications of this field are in discrete time. That wasn't necessarily true 20 years ago. It's still the case that there's lots of applications that are intrinsically married to the physics that underlie it. That makes it intrinsically CT, but a lot of stuff is done in DT. So for a lot of reasons, we're going to start with DT first, and we're going to look today at several different representations of DT.

In particular, we're going to look at difference equations and block diagrams. And we're always interested in verbal descriptions, because that's the way the real world is described to you. So what we want to do is understand these. Understand there are strengths and weaknesses, so that we can capitalize on their strengths, and go around their weaknesses. That's the idea.

So the easiest possible representation is a difference equation. Difference equations are good because they are absolutely precise, and they are the absolutely most concise representation we will have. Precise and concise. Those are good features. So here's an example. What if my output Y were my input X minus X, n minus 1. Let's say that the input is a unit sample.

We will often ask for the unit sample response. The unit sample signal is the most simple non-trivial signal that we can have. It is 0 almost everywhere, and the only place it's not 0 is at 0, and when it's not 0 it's 1. So that's sort of the easiest thing we can imagine. So what we're going to want to do then is think about what happens if we put this signal into that system. OK, that's completely trivial. You've done this a gazillion times before, and now is your chance to prove it. How many of those statements are true about the output of this system when that's the input? Talk to your neighbor. You've got 15 seconds. Is it too simple? Figure out how many of those statements are true.

OK, you're either all bored out of your minds or paralyzed with fear. I can't tell. So everybody raise your hand. How many of these statements are true? More hands, more hands, c'mon. OK, less than half correct, so I'll just assume that you ran out of time, and I'll go on. So this is trivial, right? What do I do? How do I figure this out? Yeah?

AUDIENCE: Step by step?

PROFESSOR: Step by step, of course. Yes, so if I think about applying this equation to that input, then I can think about just substituting n equals minus 1, n equals 0, n equals 1, n equals 2, et cetera. And you can sort of see that I'm only looking at the input of two instances of time. So there's a finite range over which the input-- there's a finite range of inputs that can contribute to the output. And in particular, if I were to say-- if I were to substitute n equals minus 1, I would see that the output of time-- y minus 1-- depends on inputs before the delta function started. So therefore the answer is 0. If I look at the output of time 1, the time 0 input is 1, the time minus 1 input is 0. So my total answer is 1, et cetera.

This is very straightforward. It's just plug and chug, and what you see is a waveform. From which is pretty easy to infer that Y2 is bigger than Y1, so Y2 was bigger than Y1-- bigger in a math sense. And Y3 is bigger than Y2-- not. Y3 and Y2 are identical, so that's not true. Y2 and, in fact, everything above that is 0. That's true, and if you do a little bit of manipulation, you can show that this must have also been true. Everyone see that?

I'm assuming you've seen all this sort of stuff before, so I'm going to go through it kind of quickly, but the answer was 4. 4 of those statements were true. It was just a refresher on how to think about difference equations. By comparison, I want to contrast a difference equation on a block diagram.

This is the same system but represented graphically. Why would you want to do that? Well there's advantages and disadvantages. This is a direct way of expressing the system if you wanted to implement it in hardware. So you can imagine building a delay box, building an adder, building an inverter, and constructing a piece of hardware that does the computation. So this is more like a hardware realization. It's not as compact as the difference equation.

It has another idiosyncrasy that we have to have, the notion of rest. When we say the system starts at rest-- which we will say all the time, and in fact, you can always assume that unless we tell you something to the contrary. When we say a system is at rest, what we mean is that all of the delays in the system start out being 0. So at rest means that this signal starts out at 0, because that's the only-- there's only one delay in the system. Its output is initially 0.

So now we think about the input chugging along. So the input starts out being 1. That means the inverted input is minus 1. That doesn't propagate through the delay, because we're still at the instant when the input just became one for the first time. So this one goes through, and adds to this 0 to give us 1, and our answer is 1. Now the clock goes kachunk, and at that instant, the input changes from 1 to 0, because that's what happens here. And at precisely the same instant, the delay signal went from 0-- which it was because it started at rest-- to minus 1 which was its input just before kachunk. Then-- after the signal settle down-- this becomes 0, and this becomes minus 1, and that's our output. Then, again, kachunk. The input doesn't change, because it was zero. It's still zero. The output of the delay goes from minus 1 to 0, and after things settle down, we get 0 at the output, and that is a state that persists.

So we have these two different representations. We have difference equations and block diagrams, so here's a question. How are they different? In particular, I've told you the difference equations are concise and precise. Are there advantages to a block diagram? Talk to your neighbor. Come up with a good hypothesis. What's good about a block diagram, anything? So can anybody think of something good about a block diagram? Yeah?

AUDIENCE: It changes the [INAUDIBLE] and it's easy to see what's happening.

PROFESSOR: So it's easy to see. So that could be something to do with a graphic representation, but it could be deeper too. Can somebody say a deeper reason why it might be easier to see? So that might be a very good-- that might be a deep comment too. Why might it be easier to use the block diagram than it is to use that difference equation? Yeah?

AUDIENCE: Because equations are broken down into steps?

PROFESSOR: How so?

AUDIENCE: So there's [INAUDIBLE] and then there's the feedback [INAUDIBLE] difference equations are [INAUDIBLE]

PROFESSOR: So that's kind of right. I'm wondering if you could say, I mean, you just do this, right? You do this, and then you do that. What's the difference? There's something very different about these two representations. There is more information in the bottom than there is in the top. Can somebody tell me the information that's available in the bottom that's not available in the top? Yes?

AUDIENCE: Actually solving circuits it becomes a lot easier to use graphical representation.

PROFESSOR: Circuits, graphical, but is there additional information here that wasn't available there? Yes?

AUDIENCE: The number of signal paths?

PROFESSOR: Number of signal--

AUDIENCE: Paths

PROFESSOR: Paths. Signal paths is a good idea. Yes?

AUDIENCE: Is it that X and negative 1 is 0?

[INTERPOSING VOICES]

PROFESSOR: Well that sort of goes with specifying X, yes?

AUDIENCE: The structure is more emphasized, so you can just look at it, and [INAUDIBLE] or not, and it's more easy to see structure [INAUDIBLE]

PROFESSOR: It is easier, but there's a very precise reason why it's easier. There's arrows. Arrows tell you what to do next. So the thing that's different is that. From the structure of the arrows, everything starts here, and you should worry about what goes that way, and what goes that way, and what goes that way, and how that comes together there. There's more information. In computer science terms, we would say that the difference equation is declarative. It tells you a statement of fact, not necessarily what you can do with it. The block diagram is imperative. It tells you how to do something, and those are very different.

So the difference equation is concise, precise, and declarative. The block diagram is imperative. Do this, do this, do this. It shows you the signal flow path. There's no question about who causes what. In this, we could be computing X from Y or Y from X. There's no indication other than some convention-- some place-- that the x is the input. That make sense? So there's a very big difference.

There's another step we want to take. So imperative and declarative is one step. Another step is lumping things in the same sense that we lump the coordinates of a three space point into a single thing called a point, or in the same way we make a vector in linear algebra. The idea is going to be that we're going to want to think about whole signals at a time. In the notation that we will use, we will think about-- this will be the X signal, this will be the Y signal, and we can re-interpret the block diagram to be operations on signals rather than operations on samples. So I'm trying to develop a level of abstraction that helps us predict the way systems will behave, so the next level of abstraction I want to think about is thinking about whole signals at once.

So if we take this signal idea, we would say there's an X signal. It gets operated by this inverter to give an inverted signal, then this signal goes through a delay box to give us a delayed inverted signal. The delayed inverted signal gets added to the original signal, and that gives us an output signal. So we think about the nodes being whole signals, and the boxes being operators on signals, not samples.

That's a huge difference, and to make sure to be explicit about that we'll develop a notation. We will say there's an operator R-- the right shift operator-- that operates on signals. So we'll think about if X is the whole input signal, not the nth sample, the whole thing. If X is a signal, we can operate on it by the R operator. We can apply R-- the right shift operator-- to the signal X, and generate a whole new signal which I'll call Y which will be a shifted version of the X signal. OK, you all got that?

And that leads us then to a concise representation similar to the difference equation, but that has the features of the block diagram. Because it's imperative, R operating on X tells me start with X and apply R to it. There's a direction. R applied to X is imperative. It tells me what to do. So this is a way that I can make a notation that is as concise as difference equations, but has the same imperative feature as a block diagram. So it's an improved notation in that sense.

Make sure that you know what it means. If I said that Y is RX, how many of the following is true? Now you have to redeem yourself. You were less than 50% right, so now I want 90% right. Oh, excuse me, not how many, which? OK, everybody raise your hands, c'mon. This is redemption time. Wonderful, you're redeemed. So you're about 100% correct.

So to think about that. One easy way to think about it is by example. Examples can't prove that something's true, but they can quickly prove that something's false. So if you think about the simplest possible signal-- say you had a unit sample. If you applied R to that, it would shift it to the right. If it shifts it to the right, it's pretty clear that Y at time 1 is the same as X at time 0, and that's a feature of this equation and none of the others. So that proves the others are wrong, and if you think about it a little bit, you can generalize that argument to prove that the second one is actually true for all the cases.

So the idea then is that we want to think about this hybrid notation, this operator notation, and we want to think about how to represent simple systems that way. So here's an example. Say I have this system. I can think about that as the cascade of a system that starts with X, X as a signal, and this whole thing gets turned into an operator. That operator can be written mathematically as 1. This path added to minus 1 times a delay minus 1 times a delay, so one minus R operating on X gives me Y1. Then a very similar system operates on Y1 to give me

Y2 and if the rules of math apply, I ought to be able to substitute 1 minus RX for Y1 and get that expression. And if the world were nice and it is. If the world were nice, then this expression, since it looks like a polynomial in R, might behave like a polynomial in R, and it's pretty easy to convince yourself that that's, in fact, true.

So think about the primitive definition. Sample by sample, difference equation, what should this do? Well, there's a difference representation for this, which is here. There's a difference representation for this, which I can substitute twice so I can generate an equivalent difference equation for the whole thing, and then I can think about term wise translating all of those into R expressions. And I see that I have proved the idea that this particular system at least behaves as though the operator expressions follow the rules of polynomials.

So that's important because you know how to deal with polynomials. If there's an isomorphism between these kinds of discrete time signals and polynomials, the fact that you know how to deal with polynomials translates into you already know how to do systems, right? So it's a very powerful kind of thing to draw an isomorphism between two systems, especially when you already know how to do one of them.

So let's think about equivalence a little more. Here's another example. So here is that same system that's composed of two cascaded difference engines, and the hypothesis is that this is the same. Two things I want you to see. First, it should be pretty clear that you can see how you would derive that thinking about the system as operators R. You should also be able to derive the equivalence by thinking about the individual blocks as difference equations, but more importantly, I'm hoping that you can see a caveat.

Assuming both are initially at rest-- that will come up over and over and over and over again for exactly this reason-- the equivalence is only true if they started that rest. That's not quite the right way to say it. The equivalence is always true if they start at rest. That is precisely right. If they start at rest it will always be equivalent. Can you think of why they have to be at rest? Why do they have to be at rest? OK, smile. It's not torture. Yes?

AUDIENCE: [INAUDIBLE]

PROFESSOR: So if they weren't at rest, the outputs of the delays wouldn't necessarily be zero. There is no simple correspondence between these two delays and those two delays. If I told you what are these two delays, it may or may not be possible to make an equivalent system where there are two different numbers here. Say this delay starts at 1, and this delay starts at 7. There may or may not be a set of numbers for the bottom delays that will have the same behavior as those numbers did in a top system. But what is guaranteed is if they all start at 0, everything's fine. That's the reason we put in the caveat initially that both systems start at rest.

The equivalence that we talk about is equivalence starting from rest. So think about these systems. They are equivalent. You can try to convince yourself they're equivalent by thinking about operations on block diagrams, but what I'd like you to think about is, they are a statement of what property of polynomials? The equivalence of those two systems is a statement about what property of polynomials? Is it the commutative property, the associative property, the distributive property, the transitive property, or none of the above?

OK, you're far too quiet. Take a break. Turn to your neighbor. Tell your neighbor what living group you're in. Become accustomed to talking to your neighbor, and then figure this out.

[INTERPOSING VOICES]

PROFESSOR: a equals b means b equals a.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Or 5.

AUDIENCE: [INAUDIBLE]

PROFESSOR: OK, what property is being described here. Everybody raise your hand with some number. Excellent, excellent. More than 90% correct. So the idea is that we write this as an R expression. So this system says, start with X, apply this operator to X. It's a little confusing that one goes left and one goes right. So start with X, apply this to X-- that's the 1 minus R operator-- then take whatever comes out, and apply R again. Similarly, down here, start with X. You've got the sum of two things, an R in the top, and an R squared minus 1 in the bottom, and the equivalence of those is the distributive principle. R distributes so multiplication over addition.

OK, in the interest of time, let me skip that. That's an exercise that you can work on on your own. It's very straightforward. What I want to do is talk about another more slightly deeper issue, and that's the difference between recipes and constraints for whole block diagrams. So we can think about this as a recipe in the sense that we can take the R operators, and think about signals as the composition of other signals. So this says, add this signal to that signal.

This signal is just the delay of that signal. This signal is just the minus 1 of that signal. So you can think about, there's a recipe start with X, invert it, delay it, add it to the original, that's the output.

Let's contrast that to this. This is harder. So let's think about the R operators. What we can say here is that this output is composed of the sum of two things, the input and the delayed version of the output. That's a recipe for finding the input from the output. That's not what we want to do.

So there's a recipe here that says, start with the output, shift it to the right, subtract that from the original, and that will tell you the input. But I don't know the output, I know the input. What that says is if you knew the output, it would have this property. It's declarative. Whatever the output is, it has to have that property but it didn't tell me how to get it. The R operator is imperative. Start with the input, apply R to it, you get the output.

But you can compose systems using an imperative operator that is not-- where the overall system is not imperative, so what are you going to do with something like that? Well it's a constraint, so the question is how do you think about that? The thing to do in this case is the thing you should always do in this course. Fall back to a simpler method. The idea is we're going to start by teaching simple methods, then more abstract methods and more abstract methods. The reason being that as the methods become more abstract, they become more powerful. You can do things quicker, but if they fail, revert to the easier method.

So here the easier method is sample by sample, so let's just think about how signals would propagate through this system. So think about this as a sample by sample block diagram system. Say that it starts at rest. So the output before any inputs come in have to be 0, because the initial condition was that delays were all 0. Now the input comes in at time 1. So X of 0 is 1, so the output becomes one. And now because of the way the feedback path works, the output as a second signal is-- the second sample is also one. Everybody see that? And in fact, it's stuck.

So the idea in this system-- if you think about it from a sample by sample point of view, the signal at the output persists long. In fact, infinitely long after the signal as the input goes away. That's very different from what we saw in the simpler systems that were imperative. If in the previous system you had a finite length signal at the input, any number of operations that you did to it and then combined those would give you a finite length output. Maybe bigger than the last one because of extra delays in the system, but if it started finite, it's finite. This is different here. I'm having a persisting response persists to infinity from a transient input and that's because-- it's because of feedback. It's because there's a path that wraps back on itself.

How do we think about that in terms of R operators? Well having figured out what the response looks like, there's a perfectly valid R expression for this signal. This is just the unit sample plus R times the unit sample plus R squared times the unit sample plus R cubed times the unit sample, ad infinitum. That's that system. So I've done something very powerful and not all that obvious.

I've taken a system that was not imperative, and I've turned it into a system that is imperative. This system gives me a recipe for how to do things. Start with X, delay it by one, and add it delay by another one, and add it delay by another one, and add it delay it by another one, and add it. I've converted a declarative statement about how the system should work into an imperative statement about how the system should work, and I've got an equivalent system representation like this.

And you should be saying to yourself, do I really buy that? So is it really going to be the case that a system that was previously described this way? Tell me the output operate on it by 1 minus R, and that'll be the input should that be the same as this system. If you think about that mathematically, here's a proof. It is OK to think about it that way. Start with the idea that X2 equals X1. This is a statement of the right hand side. Substitute X1 for X2. Substitute this equivalent expression for X1, and multiply it out. When you do that, you find Y2 equals Y1. It's true, and what we've done is showed a very important thing. This and this are reciprocals.

When you multiply two polynomials together and the answer is one, they're reciprocals. So that implies that we can treat that first expression-- we can think about this-- let's see, backup. So here, we had an expression that said, start with the output, operate on it with this, get the input. We can think about that instead as operate on the input by the reciprocal operator. Now I've defined what this reciprocal mean. Reciprocal means that thing in the last slide, so now I know from the last slide the reciprocal of 1 minus R is 1 plus R plus R squared plus R cubed, blah, blah, blah, and that follows the normal rules of polynomials.

This is not a surprise. You could have done that. You could have derived that same expression by thinking about synthetic division. You could have thought about it as a Taylor series. Anything you know about polynomials should have worked. So this idea of representing a signal by a system-- by an operator R-- is extraordinarily powerful. We've been able to take advantage of non-trivial properties of polynomials by having done that, and we'll continue that way.

So the overriding feature of this system that was not-- that could not-- so this system was not a recipe. It was a constraint. The feature that made it that way is something we'll call feedback, and we'll be able to visualize feedback very simply by looking at the block diagram. Feedback means that there's a cycle somewhere. So if there's no cycles then it has to be-- so we call this the network that results when there's no cycles, we say it's acyclic. What a clever word, right? So no cycles. And when it's a acyclic, it will have the property that transient inputs give rise to transient outputs. If it has a cycle in it, then it has feedback and feedback has the potential-- doesn't always do it but has the potential-- to generate a persisting output to a transient input.

So if we think about a slightly more complicated example where I've put a number p0 in the feedback loop that was not in the previous example, we can see that the way you can conceive of this-- since there's a single time when we're thinking about the unit sample response-- there's a single input that's not 0. And what you can do is think about the feedback-- the block diagram-- you can think about it separating times, because there's one unit of delay in every cycle.

So you can think about that as the only way you can get from the input at times 0 to the output of times 0 is through the straightforward path. The one highlighted by red. The only way you can get from the input of times 0 to the output of times 1 is to make the circle at least once-- to make the circle exactly once there. If you make the circle exactly once then, instead of being height one, it's height p0, because you had to have gone through the p0 once. If you go around a third time, it's p-not squared, p-not to the fourth, et cetera.

So there's a way of thinking about why the way is there this persistent response by looking at the block diagrams. What I'm trying to emphasize is that it's useful to be able to think about difference equations, block diagrams, R operators, they're all useful. What you want to know is the strengths and limitations of each. So here what I've showed is a very good way to think about feedback in terms of block diagrams.

How many of these networks are cyclic? How many of these networks are acyclic? Is this cyclic? This network? No, so none of the paths wrap that back on itself. Here's a path. Here's a path. Here's a path. None of those wrap on itself. Is this network cyclic? Sure, this is cyclic over here. This one, Yeah. The input, this is cyclic. This one? Doubly cyclic. So that's the idea, and this gives rise to a notion that we will call a pole. A pole is the base of the geometric sequence that results in these simple sequences whenever you have feedback, and we'll be saying a whole lot more about those in upcoming times.

In the interest of time, I have to stop now, but so the idea from today is supposed to have been lots of representations for DT signals, DT signals and systems, strengths and weaknesses of each, difference equations-- concise, precise-- block diagrams, visualizing signals flow paths, operators, thinking about things as polynomials, and what we'll do in the next few sessions is try to build on that polynomial representation to gain further insights into the way that systems will work. Thank you.