The following content is provided by MIT OpenCourseWare under a Creative Commons license. Additional information about our license and MIT OpenCourseWare in general is available at ocw.mit.edu.

PROFESSOR: OK. Welcome. Welcome to the second discussion of networks and the second to last talk of lecture of this course. There's a very important set of lectures after that from the students I'm looking forward to.

So last time we talked about the use of computers to model various complicated systems in mainly cellular systems, ranging from highly cooperative, even bistable bifurcation so that where a cell makes a decision, a discussion of chromosome copy number and its implications and a large chunk of time on flux balance optimization where we can look at many of these things from the standpoint of optimization.

Now, today, to paraphrase a famous local politician, we will ask not what computers can do for biology, but what biology can do for computers. And of course we'll go back and forth and try to look at what this interesting dynamic between the experimental and the computational side.

And so how can biology aid algorithm development, to return the favor and aid biology? But not only algorithm development to not only inspiring algorithm development, but actually implementing hardware and software in biological systems. We'll talk about that. And ranging from molecular computing to cellular computing and then back to inspiring algorithms again.

So slide 3, what is it that we're really talking about, that computation needs in terms of aid? How do we get-- what are the real issues here? And we've mentioned a couple of times in the course, probably initially, when we were talking about dynamic programming and related topics, of how a problem in computer science scales.

Typically, you'll have one key input, size in, and then the running time or sometimes the memory has an upper bound which we've been referring to as the order of some function of that end or end as the length of a string or the size of the problem. There are other symbols that are slightly more rarely used. There's in addition to the upper bound, there's lower bound. Sometimes you can get an exact bound or equal bound.

And how does this play out when you have specific instances of n? Now of course, this when you say this something is on the order of or upper bound of a function of n, you don't always necessarily describe the constants. But to give you a ballpark, when you have n ranging from 1 to 1,000 and the polynomial ranging from linear to quadratic to 10th power, and then the exponential or factorial.

And you can see that for very small n, you can get cases where a polynomial performance computation time can be longer than an exponential computation time. So exponential isn't necessarily bad news, depending on the size of your problem. But it gets to be bad news very quickly. As N increases, you quickly get over large numbers on the far right hand side that are computationally intractable with any known computer.

So we mentioned computational complexity is one of the various definitions of complexity at the beginning of course. This is not the one we chose to be closest to the definition of living complexity. But it's one that's very frequently used in the computer science field. I just want to introduce a few of the terms here just so you've heard them in this context. We have-- and it basically refers to the issues brought up in the previous slide of whether it takes, whether it's scales by a polynomial, which is generally desirable, or whether it's scales exponentially or worse.

And so we have P are problems that we can solve in polynomial deterministic time. Deterministic is just described the fact that the algorithms will do the same thing time after time, which is the typical computer that we feel comfortable using. And an example of problems in the P class, the dynamic programming, which we've used numerous times. It scales polynomial and the polynomial just depends on the problem slightly from square to the sixth power we've seen.

NP has a number of subsets, but overall, it means it's not the deterministic polynomial time to get the solutions. The solutions are checkable in polynomial time, but are generally felt to not be currently feasible in polynomial time for the determination as opposed to checking.

An example of this were actually the inventors of various encryption schemes such as RSA. The R and the S and the A, referring to the last names of the authors. In a certain sense, they're banking on the difficulty of cracking these codes in polynomial time. You can use the codes if you know it. You can check it, that is to say, but you can't crack it, unless someone comes through with breakthrough on the NP problems. Because if you solve one of them, you can solve all of them.

And then they're subsets of this. This is a little less critical for today's discussion. But there's NP complete, an example of which is the traveling salesman. Can you get through all the vertices on the trip with a mileage below some threshold. And an MP hard version of that same thing is what's the minimum mileage that you can get, not just less than x, but how much less than x?

And the worst case scenario is undecidable, where you really, even given an unlimited amount of time and space, you can't tell whether it is. And the classic one is the program halting problem where you don't know whether your program is going to halt and probably all of you have run into that problem. I'm just being funny. So but I mean, it is a real-- the program halting problem is a serious mathematical construct. OK.

How do we deal with this? How do we start thinking about ways that we've dealt with it before and ways that biology could change the landscape a little bit? Usually, what we do when we're faced with an NP hard problem is cheat in some way or another. You redefine the problem, so it's in class P, sometimes sacrificing something.

So you might have-- if you're interested in tertiary structure, you may be defined as a secondary. And we showed that secondary structure for RNase can be solved with the dynamic programming algorithm with N squared or worst, N to the the sixth algorithm. Whether that is as precise as the most precise tertiary structure that one could get given infinite exponential time is an open question. Probably not.

If N is a small enough, we showed that exponential times can be reasonable. And so you just do an exhaustive search. Or if you can't do that, then you use some clever heuristic way of pruning things. And that's in a certain sense, that's what most of the approximations are.

So what can biology do? We'll talk about three examples in today. One is DNA computing. One is genetic algorithms and neural networks. The first one in a certain-- none of these really actually solve the problem. The first one is a way of just obtaining a lot more raw computing power.

I'll show you a quote where they say they've solved an NP complete problem, but it's in the same sense that you can solve any exponential problem by brute force. That's not really finessing it out of NP and into P. Genetic algorithms and neural networks are definitely heuristics. They're beautifully inspired by two of the greatest algorithms in the history of life on Earth and that is evolution and complex brain networks. And we'll get--

And so genetic algorithms is based on the adaptation that occurs during evolution and recombination and mutagenesis. And neural network is also about adaptation, but this is on the time scale of learning. So we'll first dedicate ourselves to molecular computing and associated just kind of put it in the context of nanocomputing in general.

And you can see all the issues in computing, not just the math module. So all the steps are assembly of the requisite hardware. This is some kind of factory operation, typically. Then there's some of input module, some hardware and software that's required for getting the data in. Then there's some sort of memory. And then there's a central processor, might have math components and output. This is from our first lecture, assembly input, memory process, and output.

And what we want to do with biology is we want to harvest things from genomics and from just biological research in general, use to design better computers either in silico or in a biological, biochemical sense. And then, harness evolution, either to make devices or as part of algorithm development.

We have-- different people have different opinions about how much longer Moore's law, the scaling for large scale integrated circuits or curves files version that goes back to 1900s, about doubling every two years, the ability to calculate, calculations per second, per 1,000 [INAUDIBLE]. There might be another decade left in the silicone, a large scale integrated circuit. That's what some people say, or maybe more, maybe less.

So there are three real options here for that next step, electronic nanocomputing, optical nanocomputing, molecular nanocomputing and you could add to this quantum computing, so maybe four different options for beating Moore's law or extending Moore's law, depending on how you look at it.

So let's just walk through them quickly, one at a time. Optical computing you can think of as already here in a sense in that our optical fiber networks have very fast switches that are required for a good deal of our fastest internet. And there are various demonstrations where you can do optical computing for many of the tasks, not just the data transfer.

And just like many other things, there's a desire to shrink this down for cost of manufacturing and quality. And so forth. The advantages of optical over electronic, typical computers is that they are, for a given set of operations, there's the general sentiment that it might be lower heat generation. It goes at the speed of light, rather than the slightly lower than speed of light that typically comes into actual implementations in electronic circuits.

And here's two examples taken from the literature of getting natural sort of self-assembly, just as we've seen in many biological systems, we get self-assembly of membranes, self-assembly of multi-protein complexes. Here you have, you want to make particular kind of size of optical particles that have the right rank of index and spacing and shape and so forth. And you can use self-assembly here for that thing. These are examples of where we're getting in the nanometer range here. This is a 5 micron scale object here.

I've chosen this particular example, there are a number of examples of electronic nanocomputing where the electronics is getting down to the size of molecules. Here the molecule chosen is a polymer of carbon, not a hydrocarbon, but carbon. It's like the buckyballs which is carbon-60 or graphite rolled up in the tubes, these nanotubes, can be used as transistor like elements in very tiny circuits.

This is just a schematic. This isn't actually a micrograph, right. And here are four circuits that kind of reflect what might be your first four projects in an introductory electronics course, ignoring the fact that you might not use nanotubes. But you would have voltage in and voltage out. In other words, this is a transistor like circuit or an inverter like circuit in the upper left hand corner of slide 10.

And the nanotube here is in the middle in series with a resistor going from high voltage to the ground at the lower part and the voltage in essentially modulates the voltage out in the nonlinear curve. And you can see this highly cooperative curve just like the ones that we've been talking about in a number of other biological and physical systems.

The upper right, you have a NOR gate. Almost every circuit can be made by combinations simply by [? not ?] [? OARS. ?] This is an inverter plus an [? OAR. ?] And so you can see that there now have two inputs on the left and right in 1 and 2. And they can have states 1-1, 1-0, 0-1, 0-0, going from left to right. And you can see only when they're both 0, does the whole circuit now, output go down to 0 for low voltage for output.

So input, the three possible input combinations and the last one gets the [? low. ?] And this is all done this is all done with these kind of molecular scale nanotubes. Here's a RAM, required two nanotubes in order to store, whether it's either where you flip it from open to flip it from low to high.

And the last example, in the lower right, you now need 3 nanotubes. You see, we worked our way up in complexity from 1 to 2 to 3. And you need 3 in order to get a ring oscillator use. I mean, that's the way you would think about. Most easily think about it, where we have the first one's output affecting the second one. The second one affecting the third and third one looping back to the first.

And the result of this as you get a series of peaks and troughs here, which you used for synchronizing circuits or generating other useful sinusoidal processes. Now that's one example-- so the optical electronic. Molecular includes a number of different possibilities, including DNA which is what we'll focus on, DNA computing. And this was started by a person, a physicist famous for thinking out of the box quite a bit.

Feynman in 1959, when he was still fairly young, gave a talk entitled There's Plenty of Room at the Bottom. And by that he meant that we can-- just as we can machine and manufacture objects under fairly automated fashion, we should be able to scale that down to the point where we're manipulating individual atoms. And he couldn't think of any physical reason, such as the uncertainty principle or anything like that, that would prevent one from doing that, manipulating individual atoms.

And many years later now, he's been proven right and that we are doing that, albeit not in any really high production method. Drexler and his thesis and subsequently has championed, is probably the right word, this notion and given it the name nanotechnology and nanosystems and really fleshed out some of the things that one might be able to do if you had a much higher throughput way of dealing with manufacturing at atomic scale.

However, even he did not really connect all the dots between where we are now and where and how we get to the very first nano-assemblers and nanotechnology. Since then, there's been kind of a Renaissance of interest in this with a recognition that biological systems actually are naturally doing nanotech scale, atomic manipulations. And you've seen a few examples of that in this course.

But the particular instance that we'll use as a jumping off point for discussion of all the steps that could be where biology and molecular mining could give us new tools, whether it's from assembly inputs, memory computation output. We'll start with Len Adleman's pace changing paper in 1994. This is the A of the RSA that we talked about a few minutes ago.

He was obviously a hardcore and is algorithmics expert and decided in 1994 to actually do a paper that required not only algorithmic, but a huge change in the way you implemented the algorithms. And then actually to go into the lab into a biochemical laboratory which he was not previously trained. And to author-- it's a single author paper that included such things as PCR. That was in 1994. There was no literature on the subject of DNA computers before that.

A mere six years later, there were 520 references on the subject. So he obviously hit some kind of nerve. The first few years of that after that were mainly theoretical. But I'll show you some examples. His paper had an experimental component, and some others that I'll show you.

Since this course is really about that interface, constantly checking the theory with reality, those are the ones that I'll emphasize here. So the question that he asked in 1994 and is still fresh today is, is there a Hamiltonian path through all the nodes in a network? So we've been talking about interesting biological networks. But here, just in any network where the black nodes are connected by directed edges in this directed graph.

You want to go from the start, S, to the terminus T, from 1 to 6, obeying the arrows and going through every point once. How do you do this? And how do you do it in DNA? So an example of one here is going from S to 3 to 5 to 2 to 4 to T.

So the way you do it, first in broad strokes, you encode the graph, both the nodes, black spots, and the edges into single-stranded DNA sequences. Then you create all possible paths as by using overlapping sequences to indicate which node is connected to which other node by an edge in which direction. So you can actually have directionality, just because DNA has directionality.

And you use DNA hybridization now to do that step. Now, the first step is linear. Encoding the graph is linear with the number of points in the graph, the number of places in the Hamiltonian path. The second step is not something-- and that you would do by having your computer program a DNA synthesizer which is an automated machine that we've described a couple of times.

But the second step is out of your hands. This is something that happens automatically when you put DNA in the solution. If you design these sequences carefully so they don't cross hybridize very much, then the only way they can assemble is the way you want it. Then, you finally determine where the solution exists, and this is something which is almost constant in complexity.

So the entire thing scales very gracefully instead of scaling exponentially as the Hamiltonian path problem normally would. This gives the appearance of scaling linearly in time, which is really one of the best case scenarios for polynomial time and certainly better than exponential.

So how do we actually do that? That was broad strokes. This is a more detailed view. And you can see how this really seems like it's going to work. You have each of the nodes encoded by sequences, let's say, red and tan here on the top left of slide 14.

And you have, if you want to connect node 3 with node 4, so that you take the right hand in, the 3 prime end which is sorta greenish tan. And you connect it now to the other end of, that is to say the 5 prime end of node 4 just below it, which is blue. So now you have this hybrid which is ordered. So it's an arrow going from 3 to 4 and that edge has this particular sequence going from 5 prime to 3 prime.

Now, in practice, you want the edges to be complementary, not identical to the nodes. And so all the nodes are actually represented as reverse sequences in the lower left hand part of slide 14. And so represent all the edges that connect nodes in this directional manner.

And then an example of how you would connect three nodes, 3 to 4 to 5 by two connecting directional edges, 3, 4, 4, 5 is shown here. All the nodes are in reverse compliments and all the edges are in the forward direction as arbitrarily defined here. And you can see how they stitch these together and you make firm connections that are unambiguous, non cross-reacting and have a direct directionality.

Now that you are starting to get the idea that now we can encode this in DNA. But how are we going to actually do the computation and how are we going to find out who the winner is? So what is done, remember, we want to create all the paths and then to ask whether any of them go through all the points? And then do any of them go through all the points exactly once?

So the first thing is to create all the paths from start to terminus. And by just throwing in this mixture of all the edges and all the nodes, you will create in principle all the paths. Now you want to write, you will go from the prefix of one into the suffix of the other in the same way that illustrated on the previous slide.

And here are some examples of some of the paths, some are very short. This only goes through the 1, 2 4, 6, only goes through 4 the nodes, that's not all 6 of them. The bottom one goes through too many nodes and some of them it's going through repetitively. But you get the idea is that you can define the path in terms of all these edges and the reverse complements which represent the nodes.

But here's the actual algorithm as encoded in DNA and implemented by practical methods that a computer scientist can do without too much help, at least not enough help required for co-authorship. So we've already encoded the graphs in the DNA sequences. And this is done by automated oligonucleotide synthesis. You create all the paths from ST by PCR amplifying from the S end of the oligo to the T end.

So that means, the mere fact that they PCR amplify means they must contain node 1 and 6, the start of terminus. So that's good. Now, you want to get the ones that visit every node, so by serial hybridization, you can have nodes 2, 3, 4 and 5 immobilized. And you bind the PCR products to it, and you'll loop from 2 and then in series you bind to 3, loop from 3, bind to 4, loop for 4, 5. So now you know it has 1 in 6 because that's the PCR primers. It has 2, 3, 4 and 5 because it balances them in series by hybridization.

Then, but that, you could get some of those long paths that went through multiple nodes multiple times. If you want it to have exactly N nodes, then what you do is sort of of electric [? thread ?] [? excising. ?] And here if you have a calibration curve as shown on the bottom of slide 16, we have known DNA size markers and known PCR products going, showing you have one of these DNA nodes, 2, 3, all the way up to 6.

And if you find a solution that has all these properties, PCRs, does serial hybridization to all the nodes and is the right length, such as the one in column 6, then you know you've got a solution. And that was Len Adleman's argument that he had DNA computing working.

Six years later, we now have, or as of six years later, there were now over 500 examples of this. I'll show you this example as both as an introduction to the satisfiability problem and also as showing that you can do RNA computing and that you can encode two-dimensional objects and it illustrates a number of things.

The problem here is a test problem, very simple test board. It's not 8 by 8, but 3 by 3. And you've got an artificial number of knights here. And those of you who know chess know that these knights can attack in a curious combination of straight and diagonal. And it doesn't really matter. The point is there are a variety of arrangements of any number of knights such that none of them can attack each other and they're all kind of at peace here.

And that the object of this algorithm is to find those combinations. And here's-- and you do it by cloning. So that was something that was not in the previous example. By cloning, you can find each of the solutions. And you then determine what is present and sits along that clone. It's kind of like haplotyping or splice form analysis. You can really only analyze these things by looking at the product of a single molecule. And that's what cloning is about, looking at, amplifying that single molecule up to the point where you can analyze it.

So that's one thing that's new. The other thing is new, is you start with an RNA in order that you can use this powerful method, this enzyme called RNase H which will-- it has a property that when you bind a DNA oligonucleotide to an RNA and their complementary RNase H will destroy the RNA at that point, at the point of hybridization.

So it's a way of eliminating an entire molecule if it happens to have a particular sequence in it. And so one of the ways that you can ask logical questions about each molecule in a large complex mixture of molecules is using this RNase elimination. And in a way, it's a way of designing an infinite number of restriction enzymes. The RNase plus the DNA oligonucleotide provides in a certain sense, a custom restriction enzyme.

Any case, the other thing that's unusual here is the idea of using split and pool oligonucleotide synthesis. We introduced this in the lecture where we were introducing drug protein interaction and ways of synthesizing drugs and other molecules by Poole synthesis. And the idea behind split Poole synthesis in this case is that each of the squares in this 3 by 3 matrix can have two states, either has a knight or it doesn't.

Each of those two states, you can consider a 0 or 1. You can represent them as two different sequences. Sequence A or sequence A prime, representing presence or absence of something. And so you basically have 2-- you have 8 squares and so you have 2 to the 9th different possibilities.

And so down below they synthesized a set of polymers where you have every possible binary state for this 3 by 3 grid. And that's done by-- you're synthesizing along and you come to where you're going to synthesize either A or A prime. You split it. Half of this pool gets a A. Half of it gets sequence A prime. Pull back the other half gets B, half gets B prime. Pool them and split them and C and C prime and so forth, all the way out. You really only need nine of these. They did 10 for some reason or other. But the point is to get 2 to the ninth power, you need nine of these.

And then as you can-- then you can read them out electrophoretically just as the electrophoretic readout gave a sizing in the first DNA computing, you can use it to get the sizes of these. And you can see-- you could read out-- Here's two solutions, BEFH referring to the squares on a 3 by 3 grid and EFC, these are both solutions. BEFH, the way of reading this off this combinatorial synthesis from the bottom is you've got A through I as the possible bit binary signatures.

And then you have two columns, either the 0 column or the 1, two states, the two sequences. And as you PCR from the end tag out to each of these tags, A or A prime, B or B prime, and so forth. Then you get-- these are all-- you'll get this graduated series. It will tell you, you've got a little-- you've got, A is in the 0 state. B is in the 1 state. C and D are in the 0 state. E is in the 1 state so forth.

And so I've circled B being in the 1 stage showing that there's a Knight in position B. And you can go through the same thing for the other solutions. Each of these to develop as a clone. And there's an and-- the neat thing about all these problems is they have multiple representations. You represent them in DNA. You represent them in data.

You can represent them as a Boolean logical set of operations where these things represent ANDs and ORs at the bottom of the slide and so on. And you will see that in the last examples where I've just kind of breezed through, where that set of logical operations is probably one of the favorite points of attack for DNA computing still today.

So what are the problems and the advantages? The problems are that yes, it is polynomial time. In fact, it's close to linear time with a number of inputs in. In terms of synthesis needs, you know, your computer tells a synthesizer what to make. And then enzymes will or hybridisation will do this highly parallel reaction independent of N. So basically constant time. So it's linear time for synthesis, constant time for the computing, and constant time for the getting the answer right out.

But you have exponential volumes. For example, a hundred-node graph, we've been talking about the various graphs solutions might be 10 to the 30th molecules. And if any of you have ever tried to synthesize a mole or 10 to the 24th molecules, you realize it would bankrupt our planet to make 10 to the 30th molecules.

So in addition, the elementary steps are slow. It's highly parallel. You can imagine having trillions of molecules computing in parallel, but the elementary steps of hybridization and DNA polymerase or RNase H and so forth, typically are in the millihertz range. That is to say, it might take 1,000 seconds, rather than gigahertz, which would be a billionth of a second. So there might be a 10 to the 12th gap in rate of executing the commands, but there's more, the hope is that there's more than a 10 to the 12th advantage in parallelism.

In addition, experimental errors mustn't be swept under the rug. You've got issues with mismatch. There's a limit that just how cleverly you can design all these sequences. As the graph gets bigger, you need to have more and more sequences involved. And that means that you're going to get more and more cross hybridization, incomplete cleavage, and so forth.

There are-- when this slide 18 says non-reusable, there are reusable forms. And we'll get to those in just a minute. So those are the disadvantages. What are the promises or the possible advantages? High parallelism, could be much more than the 10 to the 12th fold loss in speed.

When computer hardware and people dream of the next generation of computers, they hope to get away from the current record, which is around 10 to the 9th operations per joule for conventional computers. Maybe that's not a record but it's conventional computers.

Closer to the 34 times 10 to the 19th operations per joule that you should be able to squeak out near the thermodynamic limit. And as it turns out, many DNA enzymes such as DNA polymerase are already within a factor of 10 of that goal, while conventional computers are off by 10 factors of 10 or more.

If one can quote, solve, or one NP-complete solve problem, you can get many. The improvements that we'll just kind of briefly talk about that keep people excited about this are that this is a natural way of talking to biological problems. If there are biological problems, you can--

It may be a smaller step to get to DNA computing, and there are faster readout methods just as there are faster and faster computational methods. And natural selection, evolution, is something that you can use on DNA computers, which so far has not been extremely powerful in conventional computers, although we'll talk about genetic algorithms shortly.

So one way of getting reuse is to have a so-called sticker-based model or something where you're basically just using the hybridization properties without being destructive. And here's an example. I'm not going to walk through it too much. I should point out the all in here includes Adleman again. And I'll have one more example of his work in just a moment here.

But there are examples now of work trying to consider seriously the amount, the volumes of DNA that are needed and ways of dealing with fault tolerance or error reduction algorithms where you actually go through and consciously say, OK, if we had an error here, how would we compensate for it? How would we dedicate a little more, a few more bits to take the next step?

So just like in the night test problem we had a little bit before, the idea of using ANDs and ORs of Boolean variables these X's in slide 21 which can have two states 0 or 1 similar to the Boolean variables that we run across from time to time in generative models.

And you can have clauses, which are basically logical operations on the set of Boolean variables. We have, NOTs and ORs and ANDs. And this kind of problem is a very general problem, very interesting one. And it has been tackled just in very analogous ways to the ways we've been talking about the previous ones where you encode the graph in DNA sequences, thereby creating all the paths by hybridization or something like that. And then you read it out with PCR and solid phase.

And you can say, you see here's a quote where they say, here we solve a NP-complete problem. They have indeed solved one, but they haven't really turned it into a polynomial time problem. It's simply brute force it out. And this is the most recent one, just it came out in *Science* again, Len Adleman on it, and now up to 20 variables in a three-set problem.

So that's all about computing. It may not-- since DNA is relatively slow at computing per step, highly parallel, it may not be the best of the various steps in computing assembly, input, memory, computation, and output. So we'll explore some of the other ones, in particular assembly. To a certain extent, assembling computers is slow and so it's something where molecular assembly might have some advantages.

Now this particular example, I'm going to emphasize the assembly aspect of it, but you can think of this as a way of mapping, these authors have mapped assembly of an actual two-dimensional piling onto an abstract but important and powerful computer science concept which is the general Turing machine, a machine, kind of a tape-like structure that can compute, do any kind of general computing, deterministic computing.

And so they mapped a kind of a physical tiling as you might have a periodic or a periodic mosaic of tiles onto this computing machine and onto this kind of logical operation, XOR on a string of binary bits, just as we had in the previous couple of slides on the three stat problem.

I want to emphasize the geometric, physical, version of it because we're trying to make a transition now in this paper which combines the DNA computing with DNA assembly. Now up it, we have three more or less equivalent ways of representing the same sequences. So these are called triple crossovers, because you can see, let's take a look at say Y2 here, right in the middle of the slide.

You can see that there are these multiple crossovers where you have a kind of a recombination event where you've got two double-stranded DNA molecules that are exchanging a strand. And this is not a natural homologous recombination of N in the sense they're non homologous, and this thing is kind of trapped in this crossover.

And the crossover, when you have multiple crossovers, you can make a piece of DNA that now has more than just two ends. You can have multiple sticky ends. You can have, say in this case, in the upper part, you can see four different ends with 1, 2, 3, 4 crossover strands here.

And each of these, you can see on the far right, a 5 base 3 prime overhang, and a 7 base 5 prime overhang and some flush ends on the far left. And so each of these things has ability to stick to other tiling elements. And you can see that you can put together a fairly-- a two-dimensional structure, which can be as intricate as a mosaic. It does not have to repeat itself, or it can repeat itself if you want to use visual tools such as Fourier transforms to look at a repeating structure. And you can see, you can engineer in restriction sites to help you analyze the structure on a gel-based assay on the far left.

We've seen enough gel-based assays tonight already, I won't belabor, we're not going to walk you through it. But you can see that there's-- even though this is not straight DNA as the previous two or three examples were, this is a much more complicated branched structure.

Nevertheless, you can turn it into a linear readout with electrophoretic sizes. But you can also look at it as a truly two or three-dimensional object here. Here we use atomic-- they use atomic force microscopy where a probe with a single atom at its tip is responding to the force.

That allows that as you touch an object, the feedback in the system with a scanning tunneling microscope, usually as part of the feedback, tells you that you just touched the surface and you've got to back off a little bit. And then just scan along and just profiling the surface.

And so here, with two different tiling methods, this is more a repetitive rather than aperiodic tiling. You can see that these little pink protrusions, you can engineer into the tile, not just the two-dimensional stickiness, sticky tags, but a third dimension, which is a bump, which might be a stem loop as we've seen in other secondary structures. There's engineering into this DNA.

So now that these bumps will stick up and be easy targets for the atomic force microscope. And here you have a bump every second tile, and that's on the left. And on the right of slide 27, you have a more complicated tiling where you have four different types of tiles and a bump every fourth one.

And so you expect, you can calculate from the Watson-Crick model for DNA, or more advanced models of DNA, even though this is a lattice of brain structures, you can calculate, it should be about 33 nanometers between the bumps. And that's what's observed. And 65 nanometers is calculated and observed. You can see the bigger lattice spacing in these admittedly somewhat fuzzy atomic force micrographs.

But you can see, you get something, an experimental confirmation of the two-dimensional structures here. Now this is self-assembly nanofabrication. And to some extent, it is inspired by and can be combined with microfabrication. This is something where you basically use optics, you go to the limit of current optical manufacturing as a microfabrication where it's typically hard to get below say 100 nanometers or so in feature size.

Here you can see this is-- and this is the microfabrication used to make your computer chips. But it's, in this case, it's used to actually make moving parts, parts that move relative to one another. And that motion actually has useful applications.

The first such useful application that I'm aware is putting these things into air bag sensors. And the idea is that when you're driving your large automobile and run into a even larger object, you suddenly decelerate, either by brakes or some other method. And when you do, this little bitty device, a sensor inside your-- somewhere in your car, will shift one of its parts by at least 0.2 angstroms, meaning a tenth of an atomic diameter.

That doesn't seem like very much. And it isn't. It only causes 100 femtofarads, that means 10 to the -13 farad capacitance change. But that's quite enough to signal that a collision has occurred or will occur and with a very low false positive rate, those of you who have driven large automobiles know how infrequently the airbag opens up accidentally. But when it does deflect by 0.2 angstroms then it does open up the airbag.

OK, so this is a payoff of microfabrication. But now we want to combine the nanosystems that we saw in the DNA computers and in the tiling with microfabrication. And I picked this as just one of the very few examples where microfab meets nanofabrication. And we'll call it a nano-electromechanical system.

And here, so you've got the microfabrication of both the posts upon which these things stand, which is these nickel columns of 80 nanometers, and the little bars, metal bars, which can be as a micron range. And you can see them visualized on the left hand photograph where you have these bars at regular spacings, and if you look at this publication or the webs that go along with it, these little bars will spin around.

And what they're spinning on is not a microfabricated motor, but it is a nano biotech motor, which and actually it's a protein that most people didn't think of as a motor when it was first discovered. It is the protein present in almost all organisms that is responsible for ATP generation.

Usually, you think of this as making ATP for motors to use, but this actually, since it is capable of rotary motion, does move this 1 micron bar around at the rate that you would expect for the nano-machine to be generating torque.

So we're now, we've talked about assembly. That's an example of an output device, what sort of input devices we have that will work at the single molecule level. Here is, there are many examples. We've mentioned a few of them in the sequencing, genotyping lecture where we were talking about single fluorophores.

Here, you can use another aspect of biology, which is the self-assembly of membranes to make a very, very tight, low conductance seal which is only on the order of 2 nanometers thick, but it's enough to make a gigaohm seal, multi gigaohm seal. And then you poke a little hole in it with a single molecule of a protein.

And there are growing ways to do this on inorganic substrates as well. So when you have a single protein pore which itself might be a 1 nanometer opening, it will allow in the presence of electric field, indicated by these negative and positive charges here, yellow negative and positive ions to go through like sodium chloride. And they will go through it about up to a million ions per second, easily.

And when you have a larger molecule, that's say, a polyanion in the electric field, it will slowly migrate through this pore. And while it does so, it blocks the rapid movement of the smaller ions like sodium. And you can record, the-- it doesn't necessarily completely block the channel. You can record the rate at which sodium will go through and how it's influenced by the composition of the polymer going through.

And so here's an example of a bacterial protein. This is [? Miller ?] et. all, referenced down below. A bacterial toxin, actually from staph, staphylococcus, whose goal in life is to kill other organisms, it's not to provide a handy conduit for RNA to go through a cell. But nevertheless in this experiment, when it does, when a nucleic acid does go through, it blocks the little red water molecules and similar sized ions are blocked.

And they're blocked in a way, which is sensitive to not only the molecule, but parts of the molecule. So you can see here, you've got a molecule with a oligo(A) part and an oligo(C) part. And you can actually discriminate between these, both in terms of the rate at which they go through, each of the parts goes through, and the conductance.

So if you look down in the lower right hand part, you'll see a 5 picoampere here to the -12 ampere, 20 picoampere and 120 picoampere levels for these individual molecules. And each of these spikes is first the 830 half, and then the C-70 half going through.

And you can see that the two different conductance levels that you get, reproducibly go, it's going through a typically in one direction, first the A, then the C, and different conductance levels, different rates. And then 125 amperes is in between molecules where you're getting the full conductance capability, lots of sodium is going through.

And just like with other methods that we've seen before, the use of two dimensions helps get you better statistical resolution. Here the two dimensions are the conductance and the time, time, and the vertical axis and conductance on the horizontal axis.

And you can see how you can discriminate different types of polymers by this method. One molecule of time, each of these dots represents a single molecular event. And so we'll take a short break. And then when we come back, we'll talk about not molecular computing, but designing cellular computers and revisit some of these same themes. Thanks.