DAVIS: Anniversaries are a time to reflect, first looking back at history, examining the origins of our work, and taking stock of where we are and how we got here, and, finally, imagining the future. We've asked three eminent scientists to kick off the symposium in that spirit, with Professor Tom Leighton looking at theory, Ed Lazowska discussing systems, and Patrick Winston considering artificial intelligence.

We've asked each of them to spend about 20 minutes talking and then taking five minutes for questions. There are microphones set up in front here, so, at the conclusion of their talks, please feel free to come up and ask questions. Their detailed bios are in the program, so I won't go into details. I'll simply ask each of them come up here.

Our first speaker is Professor Tom Leighton, cofounder and chief scientist of Akamai Technologies and professor of applied mathematics at MIT. Tom.

[APPLAUSE]

LEIGHTON: Thanks very much. For the next 20 minutes or so, I'm going to talk about theoretical computer science-- what it is, how the field got started, how it has transformed our lives, and where we're headed.

Computer science is all about methods for doing things, usually with the help of some sort of computational device. Today, of course, computers are ubiquitous. Each of us owns a variety of devices that contain a microprocessor. And we spend many hours every day interacting with those devices.

Worldwide, there are now more computer-equipped devices sold each year than there are people. Over 5 billion of these devices are connected to the internet. And more than 2 billion people use the internet on a regular basis.

To say that computation is transforming practically everything is, if anything, an understatement. Theoretical computer science is the branch of computer science that lays the foundations for the field. This is accomplished in part by establishing facts about computing systems and algorithms through the use of mathematical analysis and the notion of a proof.

For example, say that you've developed an algorithm for doing something, and you want to know how long it's going to take to run that algorithm on a particular problem. In theoretical computer science, you would generally use mathematics to analyze that algorithm, and then you would prove a theorem saying something like, the algorithm takes x steps.

If you're working with a computation that spans multiple processors, then you might try to prove a theorem that tells you how many units of hardware that computation's going to require. Or maybe you're working in an environment with noise, in which case you might want to prove a theorem the confirms that the algorithm is going to work even if z bits of your input are somehow corrupted by the noise.

As a more concrete example, let's look at a problem that many of you may be familiar with-- that is, finding the prime factors of a number n. A number is said to be prime if it is divisible only by itself and one. For example, two, three, five, seven, 11, and so forth are the first few primes.

Now it is well known that if a number greater than one is not prime, then it can be uniquely factored into primes. For example, four is two times two, nine is three times three, and so on.

By the way, if you're not so familiar with prime numbers, don't worry, I'm not giving a quiz today. And you're not alone. In fact, a few years ago the *Boston Globe* used nine as the prototypical example of a prime number in a front-page article on great advances in mathematics.

[LAUGHTER]

We have a lot of fun with the *Boston Globe* in my course for mathematics for computer science. Anyway, suppose that we want to define the prime factors of a number. Lots of algorithms have been discovered to do this. One of the simplest starts out by first factoring out all the factors of two. Just divide the number by two, if you can.

For example, if we wanted to find the factors of 78 we'd divide by two and find out that 78 is twice 39. Then we would divide out the factors of three.

We're going to need a new battery in this clicker, here. [LAUGH] We're going to need a new clicker. There we've got one! All right. [LAUGH]

So you divide out the factors of three, and you see 39's a multiple of three. So we get two times three times 13, and we keep working.

You got another one? It's taking about a few clicks, here, per slide. Thank you. You don't know if it works? We'll try this one. Okay.

Then you divide out the factors of five. There are none here. Hmm. (LAUGHING) I'll stick with the first one. And you keep on going for all the primes up the square root of the number you're trying to factor.

So in this case you'd go up to seven. And you'd divide by seven, in this case, and there's no-- it doesn't divide through. And then it turns out if there's anything bigger than one left over, what's left over is a prime. And in this case 13 is left over, and so you know that 13 is a prime. So, a pretty simple algorithm.

Now the first question you might ask about this algorithm, or any algorithm, is, does it really work? In other words, will it get the correct answer for every number n that you apply the algorithm to? The answer in this case is yes, but to be sure you would need a proof, which is the sort of thing that theoretical computer scientists do for a living, although usually for more complicated algorithms.

Now once you're convinced the algorithm is correct, then you might want to know how long it takes to find the factorization for some number n. Once again, this is a common task for the theoretical computer scientist. In this case, the answer is easy. The number of steps is simply the number of primes up to square root of n. That's because we do one of these divide-out kinds of steps for every prime up to the square root of n.

Now actually determining the number of primes up to square root of n turns out to be not so easy. This requires a bunch of mathematical analysis. But by using number theory, it is possible show that the number of primes up to the square root of n grows roughly proportionally to the square root of n over the logarithm of n. It's not exactly that, but it's pretty close. So that means our simple algorithm takes about square root of n over log n steps.

Now this kind of result is typical, in many respects, of what is done in theoretical computer science. First, the result was obtained and proved using mathematical analysis instead of computer simulation. For example, you could have taken the algorithm and run it on 78 and see how long it took. But that might not give you a good idea of how long it's going to take on a larger number.

Second, the result is stated as a function of n. So you know how long it takes as a function of an arbitrary input-- the number n, in this case. Third, we didn't worry too much about the details of what kind of machine was being used to run the algorithm. Obviously, if it used a state of the art processor today, it would be a lot faster than if it used a machine that Victor talked about maybe 30 years ago. And this is why the result is expressed in terms of n, or a number of steps, instead of milliseconds. Lastly, we didn't get the exact answer-- only an approximate answer that sort of tells us how the running time grows as a function of the input-- the number you're trying to factor.

Now that we know the running time, roughly, the next question for the theoretical computer scientist is, how good is it? Is this a good algorithm or not? And let's try to answer this question by plugging in some values of n. For example, if n is a number less than 100, this is a very good algorithm. It only has, at most, four steps. That's because there's only four primes less than the square root of 100-- namely two, three, five, and seven-- and you have one step for each of them.

But if n is much larger-- say that n is a 100-digit number-- so a number like 10 to the 100-- then the algorithm is not so good. It will take more than 10 to the 40th steps. That's because the square root of 10 to the 100 is 10 to the 50, and then you divide out by log of 10 to the 100, and you've got a very big number. In fact, it's at least that big. And I don't carry around the names of these numbers in my head, but if you look it up online you can find out that's 10 duodecillion, which means you're not going to be able to use this algorithm to factor 100-digit numbers.

Now this algorithm gives you a glimpse into the kinds of things that someone who works on algorithms does for a living. But algorithms is just one of many subfields of theoretical computer science. There's also parallel and distributed computing, where you're taking lots of processors and trying to get them to work in a coordinated way to solve a problem even faster.

There is complexity theory, which is all about grouping classes of problems together in the same class so that you can leverage tools you develop to solve a problem in that class. And there's examples of things like NP-complete problems, of which there are now thousands of practical problems that have been put into this class. So that if somebody's clever and solves one, you'll get a solution to all of them. Or if somebody shows that one of them is not solvable, we'll instantly know that none of them are solvable.

There's computational biology, which Susan referred to earlier, where computer science is being used to greatly accelerate the job of the biologist and the medical researcher. There's cryptography, where computer science is being used to develop codes which we hope are unbreakable. And at least we hope to be able to have tools that tell us how much effort the adversary is going to need to use to decipher a message without the key.

There's quantum computing, which is a whole new way of thinking about computers. Everything you're going to hear about today, and everything we do today in computer science, basically has a processor doing one step after another. Sure, there's parallel computing, where, if you have 100 processors, you can do 100 things at once. Quantum computing is a whole different game, where lots and lots of things can all be captured in a gate-- in a notion of a quantum gate. Could completely radicalize and change how we think about computing over the next 50-plus years.

And there's a lot more. And I don't have time to go into all the areas that people in theoretical computer science are working on.

Where does computer science come from? Many people say it dates back to the days of Church, Turing, and Kleene, who invented the early models of computing machines, today referred to as a "Turing machine." And they proved fundamental results on the limits of computation.

In particular, Turing proved that no computer, however powerful and however big, is always capable of deciding whether or not a given machine halts on a given input, known as the "halting problem"-- a very fundamental result then and even today. Basically the Turing machine is a model of computing that can do anything that is computable. But there are some things that are not computable.

But actually computer science goes back earlier. In fact, mathematicians have been working on the problem of factoring a large number for centuries. In fact, that's a who's-who list of mathematicians that spent many years of their lives trying to figure out faster algorithms for factoring large numbers. And in fact not just for centuries, but in fact for millennia. The ancient Greeks were also obsessed with the problem of determining if a number is prime, which Eratosthenes has a good algorithm for, and finding prime factors of numbers, dating back to Euclid.

In fact, Euclid discovered a very clever algorithm in 300 BC for finding the largest common factor of two numbers. That's not quite the same thing as finding factors of a number. But if you have two numbers, you can find common factors-- the largest common factor of the two numbers.

Now, for example, seven is the largest common factor of 21 and 70. Euclid's algorithm provides an exponential speedup over using the simple algorithm I described earlier to factor each number and then find the biggest common factors. And instead of using 10 duodecillion steps to find the largest common factor of two 100-digit numbers, Euclid's algorithm takes 200 steps. So that's a monstrous improvement that makes it easy to do in practice. And the amazing thing is that, thousands of years later, Euclid's algorithm is still widely used.

Now, despite Euclid's algorithm, which we use today to do factoring other things, we still don't know how to factor numbers this large easily, because you can't use Euclid's algorithm to do it directly. But at this point you might be asking yourselves, why on earth would you care about factoring 100-digit numbers? After all, when's the last time you went to a party and somebody came up to you and said, hey, I got this 100-digit number. Can you factor it for me? You know, I'm guessing it's been a little while.

Now I will explain why that's important in just a moment. But first I want to pose an even more basic question. Does theoretical computer science matter?

Now, as you might imagine, this question is often posed to research in this field. I've been asked a question many times. In fact, I was asked to answer this question today. And it can cause a fair amount of angst, especially if the person posing the question happens to be on your promotion committee or the head of a DARPA or NSF panel deciding how to allocate grant dollars. Or if you're preparing a talk for several hundred people.

One way to answer this question would be for me to describe to you some of the many brilliant and fundamental discoveries made by researchers in this field, many of which have been done here at MIT. But that's hard to do, since it would take way more time than I have left to explain the central advances in even one of the subfields that I listed for theoretical computer science.

Unfortunately, the complexity of explaining our most important advances is the same challenge faced by our community when it comes time to justify their share of the funding pie. So, instead of explaining or describing the advances that our field considers to be the most fundamental or the deepest, I will mention some of the examples where work in theoretical computer science has impacted your lives. It's a little strange that these examples are generally not considered to be even close to the most difficult or important results in the field. It's just that they turned out to be incredibly useful.

The first example is the RSA encryption protocol. Now this is an algorithm for encrypting data. And it's based-- its security is based on the fact that multiplying two large prime numbers with hundreds of digits is easy-- computers can do that quickly-- and the belief that splitting a large number into its prime factors is hard. Okay, we already saw the simple algorithm is not going to do the trick for 100-digit numbers. And, in fact, we still-- well, we could do 100-digit numbers today; we don't know how to do 1,000-digit numbers today.

This algorithm was developed by Rivest, Shamir, and Adleman at MIT in the 1970s. And that's a picture of Ron Rivest, who's here at MIT today. That's for RSA. And you use this algorithm today, whenever you shop or bank online. Pretty much all the stuff that's done over the internet that's sensitive is protected using the RSA algorithm or a variant derived from the RSA algorithm.

And this is why factoring large numbers is so important. If you were to figure out an algorithm to factor large numbers quickly, you could use that algorithm to crack the most widely used cryptosystems today. And that would give you access to all sorts of interesting secret data. So a lot of people spend a lot of time looking for better algorithms for factoring. And today they've figured out algorithms that are good enough to factor 100-digit numbers but not 1,000-digit numbers. And so today that's why the cryptosystems have keys with at least 1,000 digits.

The Viterbi algorithm-- I'm not sure how many people out there have heard of that algorithm-- but it's an optimal algorithm for reconstructing a signal that's been corrupted by noise. It is used in pretty much all your cell phones, modems, and digital TVs. Now the neat thing about this algorithm is that when Viterbi discovered it in the 1960s, he had no idea it would be useful. In fact, he thought it would never be practical. Of course, they didn't have great computers in the 1960s, compared to today.

He actually invented it while he was preparing to teach his class, because he wanted to get the optimal solution to explain how to do decoding optimally for his students. Later, it led to the creation of Qualcomm, which you probably have heard about.

Linear programming is actually-- it's a body, a family of algorithms for solving common optimization problems that are governed by linear constraints. Comes up in a lot of areas. And not widely known is it was actually invented during World War II to optimize military planning. And it was a closely held secret till it was published in 1947. This is the simplex algorithm.

Google's page-rank algorithm. Google models the web pages out there as a graph. Every page is a node of this graph. And the links on a page are modeled as edges. So if this page has a link pointing to this page, you put an edge there.

Now page ranks, used to determine your search rank, are determined by computing what's called the "stationary distribution" of a random walk on that graph. And basically that means that if a lot of websites point to yours, you get a high rank. And if a big, popular website points to yours, you might get an okay rank. And, more or less, that's how it worked.

The particular algorithm called PageRank was discovered by Larry Page. In fact, the research here goes back to Andrei Markov, in the early 1900s. And the application to web graphs was done by Jon Kleinberg in the 1990s-- a theoretical computer science researcher

The last example is one that I have some personal experience with, and that's Akamai's Content Delivery Network. Akamai runs the largest on-demand distributed-computing platform on the internet. You use this every day if you go online, because we deliver about 130,000 websites, including all the major websites.

The development of Akamai's platform spawned from work done in theoretical computer science here at MIT in the 1990s, where we developed algorithms for load balancing, packet routing, server assignment, and caching. Now, as a theoretician, I'd love to explain the theoretical results that led to Akamai in greater detail, but my time is running out.

Let me just quickly say that one of them involved a new approach to hashing that is well suited for use in dynamic and unreliable environments such as the internet. We called the algorithm "consistent hashing," and it formed the core of Danny Lewin's prizewinning master's thesis here at MIT. That's Danny Lewin, shortly before he was killed on September 11.

Now, at the time, we had no idea that we would ever form a company or that anybody would really use the algorithm. In fact, forming the company was an act of a last resort. We had tried to get people to use the technology. We went to a lot of the major ISPs and the major web enterprises and couldn't convince them that this made sense. The reaction was, distributed computing is a theoretical concept that has no use in practice, so please go back to your ivory tower. Okay.

Anyway the story of Akamai is a long one that I don't have time for. Let me get back to the question of, does theoretical computer science matter? And, in my opinion, at least, the answer is yes. But it's also true that we don't often recognize it at the time. Often, when we do things in theory, it's not till later that we realize, wow, that turned out to be pretty important.

So where do we go from here? Well, I can't predict the future better than anybody else, but here are some things that might be coming, and when we all get together in 50 years, maybe we'll have found out that nanoparticles can be used to cure diseases. Now this might not take 50 years, because that is a real picture of a bacteria that was destroyed by a nanoparticle built by IBM, recently released. Now, of course, to make this work in real people and to kill lots of bacteria, a lot of research has to be done, and not just by computer scientists but, of course, by medical researchers-- people who fabricate the nanoparticles.

And, within theory, distributed control systems, for example-- distributed computing. Maybe we'll find that computational biology has been used to stop cancer and slow aging. After all, those processes really are computational. They are very big data problems. The human genome is a string of three billion characters with 30,000 genes that have complicated folding patterns. And really we're going to need computer science and computational biology to figure out how all that really works so we can stop cancer and slow aging.

Maybe by then quantum computers, which are a reality today, will become a reality at scale. And, in fact, it's been discovered that quantum computers can factor numbers very easily. If you could build a big quantum computer, factoring is a piece of cake. Maybe we'll find out in 50 years that factoring is an easy problem because computers are quantum computers.

And maybe we'll have learned that computers really do you think. Maybe not, but already they're beating us at things like chess and Jeopardy. And, in fact, this one I'll actually leave to the last speaker in this session, Pat Winston.

There might be time for a question or two. And I can't see you, so you may have to shout out if they don't turn the lights on.

[APPLAUSE]

Thanks. I think we also have microphones up front, here, if you're brave enough to come down to the microphone. Okay, no questions. So the theory guy gets off easy. And [LAUGHS] we'll introduce the next speaker.

[APPLAUSE]

Okay, you've got to press this, like, three or four times. The other one doesn't work. That's my--

DAVIS: Okay, next I'm going to ask Professor Ed Lazowska to come up and discuss systems. Ed is the Bill and Melinda Gates Chair in computer science and engineering at the University of Washington, in Seattle, and we're delighted to have him with us this morning. Ed.

[APPLAUSE]

LAZOWSKA: Well, I'm truly honored to be here, to be with you here today, and to participate in this wonderful event at the place that I think arguably, more than any other organization, has contributed to the advances of computer systems and computer science over the past 50 years. Let me go back 40 years ago. As Tom said, I didn't realize I wouldn't be able to see you at all.

But if you go back to 1969, I'm going to claim that there were four big events in that year. Think about what they were. What happened in 1969? Can any of you, up here in front, where I can see, you remember what happened that year?

AUDIENCE: ARPANET.

LAZOWSKA: ARPANET was one. We'll get there in a sec. What else?

AUDIENCE: Woodstock.

LAZOWSKA: Woodstock. Great. Anything else?

AUDIENCE: [INAUDIBLE].

LAZOWSKA: Man on the moon. And? John Guttag, you know the answer to this one. The Mets won the World Series.

[LAUGHTER]

And the Red Sox won a game last night. So remember, every journey begins with a single step.

[LAUGHTER]

So we have the Mets. We have Woodstock. It was a great time, although no one remembers much about it.

LEIGHTON: Press it hard.

LAZOWSKA: Have to press it hard. We have the moonshot, a monumental engineering achievement. And we have the first packet flowing over ARPANET.

ARPANET had four nodes at that time. That's Charlie Klein's log on the right, in which he describes sending the first packet over ARPANET, from UCLA to SRI. The first packet over ARPANET contained the characters L and O. It was an attempt to login, and it didn't quite make it all the way. So--

[LAUGHTER]

--that's another thing that hasn't changed. But the second attempt, later that night, worked.

So here's the question. With 40 years of hindsight, which of those four events had the greatest impact? Right? And my argument is that, unless you're really big into Tang and Velcro, the answer is completely clear. It was the internet, which surely received far less attention and required far less funding at the time. Right?

Now this is not to diminish the importance-- both the inspirational and aspirational and scientific importance-- of sending a person to the moon. But it is to say that the exponential power behind computing has caused the internet to have dramatically greater impact on our lives and our world than anything else that happened 40 years ago.

So let's talk about those exponentials for a second. We spent 100 years with mechanical computers. That's the reconstruction of Babbage's Difference Engine 2 on the left and Vannevar Bush's differential analyzer on the right. You heard about that earlier today, and I'll come back to a little bit later. He looks not entirely happy in this photograph. I think it's because he was trying to invent foosball and didn't quite make it.

[LAUGHTER]

Then we spent 10 years, 15 years, with vacuum-tube electronic computers. Concurrently, the transistor was invented. That led, 10 years later, to the integrated circuit. That led, 10 years later, to the formulation of Moore's law and the exponential growth that we've all taken advantage of.

The result of that is that the power of early vacuum-tube machines is now literally embodied in electronic greeting cards. And the computational power that got people to the moon is now literally embodied in a Furby.

[LAUGHTER]

So I say it's not clear this is the best social use of that computational power, but it's still utterly remarkable what we've been able to achieve. And the same is true of the internet. As you heard earlier, about the number of hosts on the internet, it really has experienced the same sort of rough exponential growth that everything else in our field has experienced.

So now this is a bit of an eye test for you. But this exponential growth makes it very hard to predict what the future is going to yield. But there are people who see the future with enormous clarity.

Paul Baran, who was one of the many parents of the internet, passed away a couple of weeks ago. And that caused my attention to be brought to a technical report that he wrote in 1971. So that's just two years after that first packet traveled over the internet-- the four-node ARPANET, at the time. So it's long before any of what we imagine today.

And this is a technical report on the implications of home communication, home networking, that Paul Baran wrote in 1971. And let me read you a few of the things he talked about. So each of these is a phrase supported by a few sentences.

"A cashless society. A dedicated newspaper. Computer-aided school instruction. Shopping transactions. Plays and movies from a video library. Transit reservations. Banking services. Library access. Even mass mail and direct advertising."

[LAUGHTER]

So he had it all in 1971, and it's taken us 40 years to get there. But we have gotten there. It's entirely remarkable.

So let me talk now about the most recent 30 years, having started 40 years ago. About a year ago, a set of faculty members from the Wharton School at the University of Pennsylvania were asked to identify the most important 20 innovations of the most recent 30 years. And looking at the most recent 30 years is important. If you go back too far, you're competing with fire and the wheel and things like that. It's a little hard to hold up.

But these are the top innovations of the last 30 years, according to judges from the Wharton School. And I'm going to assert that the vast majority of these were computer-science innovations. And, in fact, the vast majority of those were computer systems innovations.

So if you look at this, the internet, the PC, word processing, email-- down the list. These are the top-- asserted top 20 inventions of the past 30 years, by a set of people who aren't computer scientists. The ones I've identified there are hard-core computer science innovations and, in fact, for the most part, computer system innovations.

These are at least 50% ours, and these are at least a quarter ours. So the life-changing innovations of the past 30 years are due to the progress of computer systems.

Now, like a number of people in this room, a couple of years ago at the turn of the decade I was asked to say what had changed in the past 10 years due to advances in computing. And this was my list of things that were totally different in 2010 than they were in 2000. Search, scalability, digital media, mobility, e-commerce, the cloud, and social networking and crowdsourcing.

And I'm going to say a word about just a few of these. But again, I consider this to be the march of computer systems, driven forward by a marriage with the theory of computation, which Tom talked about, and artificial intelligence, which Patrick will talk about next. So let me talk about scalability for a second.

This is an ad from a Compaq AlphaServer, from just about a dozen years ago. And in all of Compaq's ads at that time, Jeff Bezos from Amazon.com was featured as sort of the poster child for this system. Now this system was a very large-scale, shared-memory, multiprocessor system. And you can see some quotes from Jeff-- a picture of the AlphaServer, there.

Here's what's important about this. Amazon.com's scalability was limited by the largest reliable shared-memory multiprocessor that Digital Equipment Corporation-- subsequently Compaq, and subsequently HP-- could build. All right? Today, that's an absolutely laughable way to think about constructing a scalable web service.

Today, just a bit more than a decade later, we build systems of unimaginably greater scale. We use the cheapest imaginable components. We accept failures as a fact of life. When you have hundreds of thousands of disk drives and hundreds of thousands of processors and power supplies, you can't conceivably have a reliable hardware system. You couldn't afford it. You couldn't do it.

So software, software algorithms, many of them invented here, make these systems fault-tolerant, available, recoverable, consistent, scalable, predictable, and, to a great extent today, secure. All right? So today, hundreds of thousands of blades make up these data centers, made reliable by software. And the way in which built Amazon.com built its infrastructure just a dozen years ago is inconceivable at the scales we operate today. So the world has changed totally, in terms of how we build large-scale systems.

Operational efficiency is part of this. John Guttag and I were undergraduates at Brown University more years ago than we'd care to count. And the machine we grew up on was an IBM 360 model 67. That machine came with its own full-time service person. All right?

So he would arrive every morning, just as we were heading off to go to bed, because we used the machine from midnight to 7:00 AM. And he would spend the day there, sort of praying around the base of this thing and keeping it working. He was the hardware maintenance person. And that was how this one million instruction per second machine was kept working.

Web services of just five or 10 years ago typically had one support staff member for 250 servers. Today, Microsoft, Amazon.com, Google, Yahoo-- they're all about the same. Microsoft Azure requires 12 support people to support 35,000 servers. All right? And those 12 are really six and six. They're located half on this continent and half on the other side of the world to provide easy 24-hour support.

So, again, these systems are largely self-maintaining, self-healing, self-reconfiguring-- require a very modest amount of support, or you wouldn't have been able to do it. Jim Gray, when he won the Turing Award a bit more than a decade ago, talked about 10 or so problems for us to tackle in the future. And one of them was these systems that were not just scalable but had scalable operations. And again, we've moved a long distance towards achieving that.

You heard about digital media a few minutes ago. And again, this is a great marriage between theory and systems. Today, the fact that text and audio and video are all digital is completely transforming, and really it's a huge change in just the past 10 years. The idea that all of your photographic images and all of your videos, all of your audio, are created and edited and consumed digitally is a complete change in a decade.

Mobility-- the fact that your computing and your communication is with you everywhere-- is, again, a complete change. So that's just a look at three of these things I identified a year ago as having changed the world.

Butler Lampson talks about computing as going through three stages-- 30 years of simulation, 30 years of communication, and the future focused on embodiment. So let me say a word about that. We've seen the 30 years of simulation. That's really from the 1940s until the 1970s, in which the idea was to use a computer system to model the world that you couldn't observe.

We then, starting in maybe 1980, spent 30 years with computers as communication devices. So these were devices, but they still were largely thought of as computers. And they hook us to the world in a variety of ways, whether it's word-processing equipment or video or audio chat-- cellphones-- things like this. And what we've been entering in recent years is this notion of embodiment, in which computers connect us to the physical world.

And that was my story last year for what we're going to be seeing over the next 10 years. Instead of referring to "embodiment," I referred to "smart." And again, this is the junction between artificial intelligence and computer systems that you'll be witnessing over the next decade. We're going to have smart homes and cars and health and robots and science, and smart crowds and smart interaction-- virtual and augmented reality. Let me say a word about a couple of these.

You're familiar with the DARPA Grand Challenge, the DARPA Urban Challenge. These remarkable robot vehicles that navigate at over 100 miles of open terrain or dozens of miles in a city landscape. Of course, we're seeing this work in our automobiles today. We have lane departure warnings, we have adaptive cruise control, we have self parking, we even have, God help us, Google vehicles driving on 101 in California. This is a reason to stay in Boston, I think.

[LAUGHTER]

But this is utterly remarkable. And what's happening here-- you can think of this as robotics or artificial intelligence. I think of it as systems, not in the operating-system sense but in the sense that all of computer science today, the vast majority of computer science, is building systems. You're no longer the sensor person, or the effector person, or the speech person, or the language person. You're building a system. And building that system is driving all aspects of the field forward.

Smart crowds and human computer systems is another thing I identified. Whether it's Trapster-- how many of you use Trapster? If you're a speeder, Trapster is the greatest thing in the world. It's crowdsourced speed-trap location. Right? So it gets you out of the normal iteration in which they invent radar, so you get a radar detector, so they invent lasers, so you get a laser detector, so they make detectors illegal, so you get a little one that fits under the hood. All right?

[LAUGHTER]

Here, what happens is you just drive along, and when you come to a place where someone has reported a speed trap, it flashes at you and you slow down. And when you pass, it gives you a thumbs up or thumbs down, and you tell it whether the speed trap was still there or not. All right?

That led, through a complicated path, to the DARPA balloon challenge of just two years ago, in which DARPA one morning deployed 10 10-foot-diameter red weather balloons at 10 more-or-less random sites in the United States, some of them highly visible-- Union Square in downtown San Francisco-- and some of them in the middle of nowhere, and challenged a set of teams-- there were originally 4,000 teams registered for this competition-- to crowdsource the locations of those balloons. An MIT team won the competition.

This is something that the intelligence community told DARPA simply could not be done. All right? So the question of crowdsourcing intelligence is really interesting.

A third version of crowdsourcing, in the lower right, is a program from the University of Washington called Foldit, in which hundreds of thousands of people are using a web-based game for protein structure calculation. Okay? Protein folding and protein structure calculation. So it's games for science.

Luis Von Ahn, the wonderful young person from Carnegie Mellon who, in some sense, pioneered these online games, begins with the observation that the number of person hours spent in a day playing computer solitaire is the number of person hours it took to build the Panama Canal. So if you could turn some tiny fraction of that into useful work, the world would be a better place. And this is just one example of that. All right? So this question of humans computer systems that combine the capabilities of both is an area we'll be seeing a lot of the future.

Smart interaction. I'll give you two examples, here. Let me talk about the Kinect, first of all. This is an utterly remarkable machine. And it contains a set of advances from four different Microsoft research laboratories-- advances in speech, where, of course, MIT has been extremely active, identity recognition, tracking, and system performance. And again, what was dramatic here was combining all of these advances into a single system.

I have a photo I have seen-- the box that was the original sort of prototype for the speech system. It was an array of microphones. It's about the size of a desk. And that's what they did the software prototyping on. And this was eventually reduced to a four-microphone array that costs $0.30. Okay?

So a remarkable job of integrating a number of technologies to create a system. Lo and behold, when the speech people and the vision people get together, they're able to build a system that provides dramatically greater capabilities than either could do on their own. You know? Not surprisingly, maybe, the location information from the microphone array helps the camera array figure out who the player is at this point in time.

You heard about Watson before today. Again, another remarkable advance of AI, but really in the systems context. It's well known that Watson beat Ken Jennings and Brad Rutter. It's less well known that a couple of days later there was a Watson demo on Capitol Hill, and Rush Holt actually beat Watson. Killed it. Right? Which shows that we need more physicists in Congress. Rush is the only one left. But, uh--

[LAUGHTER]

--he actually-- he killed the system. He, by the way, I think, is a five-time Jeopardy champion. So he was a bit of a ringer. But nonetheless it was quite remarkable. So let me talk about a few trends, and then I'll wind down.

First of all, everyone is building systems. Everyone is building complete systems. As I said before, you're not the speech person or the identity-recognition person or the tracking person or the performance person. You're collaborating to create a Kinect or an autonomous vehicle. And that systems view of the world is propelling us all forward.

Secondly, AI and systems are converging. All systems have to be smart. All systems have to deal with uncertainty.

Third-- again, you heard this from Victor earlier today-- ubiquity, embodiment, invisibility. Systems are really part of the fabric of our lives. They're not thought of as computers. The ones that are thought of as computers are the small number-- the 1% that are kind of annoying. The ones that are part of the fabric of your lives are the ones we're building as we go forward.

Two more, and these arose from discussions with Butler Lampson over the past couple of weeks. Increasingly, best efforts is good enough. Now banking systems have to be reliable, although maybe the ATM doesn't have to be up all the time, as long as it at least records your transactions correctly.

But many systems-- Facebook, Amazon, Google-- don't have to get it totally right. It's more important to be there, to be fast, to be cool if you're a game or something like that. And you really couldn't afford to build a 100%-reliable system at the scale we're building. So this notion that best efforts is good enough--

And it's something we've been practicing for decades. Right? One of the breakthroughs of Ethernet, back in the 1970s, was it was a best-efforts network, and its best effort was good to a couple of [? lines, ?] and that was good enough. You could always retry at a higher level. The end-to-end principle-- another contribution of MIT. It's telling me I'm done.

Secondly, reusable components are finally catching on. Now we've had the notion for decades, in software, that you would create modules that could be reused. The truth is, I think people always viewed the understanding in reuse as more complicated than just building it again.

And it's the large modules that have been successfully reused. So databases get reused, operating systems get reused, but we're seeing more and more reuse of large components. So Watson integrated a large number of independently developed components, some of them from other companies and many of them from research labs at Carnegie Mellon, Microsoft, and elsewhere. Kinect is going to be a component in countless amazing systems over the next few years.

So I want to conclude with a left-coast view of MIT's role in the advances of computer systems. And you've seen a few of these early today. But I just want to go through this. I'm going to stop 15 years ago, because it's hard to predict the impact of recent things.

But I think if you take my word for it or take the Computer History Museum's word for it, you'll see that MIT has been driving this field forward since the beginning. We've talked about the differential analyzer. The concept of the memex was absolutely transforming. Again, like the Paul Baran material I showed you earlier, this is someone who saw the future with unbelievable clarity-- in this case, 60, 70 years ahead of reality.

The notion of cybernetics, control theory, Shannon's work on communication. Whirlwind, which led to core memory a couple of years later-- the first core-memory machine. TX-0, I believe you saw earlier-- the first transistorized machine.

Now the TX-1 was not so successful. There's a thing we call "second-system syndrome." Second-system syndrome is where you build something successfully and you overreach.

But out of TX-1, two important things happened. One was the DEC PDP-1. All right? And, of course, back at MIT, Space War. All right? So, the first video game.

The second was TX-2 and Ivan Sutherland and Sketchpad, which, again, foresaw an enormous amount of computer graphics. You'll hear about computer graphics from Tony DeRose and others later today. Right? So these are, in some sense, by-products of TX-0 to TX-1 which didn't quite make it. Okay? The DEC machine and the TX-2 and Sketchpad.

CTSS, the Compatible Time-Sharing System. I had always wondered what it was compatible with, since it was the first time-sharing system. And what it in fact did was to use essentially the prototype of today's virtual machine monitors to run, in the background, the Fortran batch system that the 709 and then the 7090 had been running. Right? So CTSS-- I use this in my operating systems classes to discuss.

But it invented virtualization. It had one of the first text-processing programs. It had user messaging.

There was Project MAC. Project MAC begot Multix, which is the source of every good idea in modern operating systems. Now Multix also overreached, in the sense that it tried to do too much for the hardware of the day. A beneficial side effect of that was the Unix project. There was sort of a divorce between Bell Labs at MIT, and the Bell Labs folks did Unix, a simple operating system whose name was, in some sense, a parody of Multix-- so, a direct derivative.

Bob Metcalfe's work that defined Ethernet. Metcalfe's thesis was an analytical thesis. It's one of my favorites. This is a performance graph. And what Metcalfe showed was that if you had what we think of today as binary exponential backoff, the ALOHA network would be stable. So all those catastrophic, decaying curves below are a fixed backoff strategy ALOHA, the way it was implemented. And the nice curve with the arrow pointing to it up above is the stability that you get. Okay?

There was Emacs. There was RSA, which of course was great theory work but has led to enormous impacts in systems. There's the GNU project. There's the Connection Machine.

So there's the World Wide Web Consortium. You may be surprised that I've listed this here, but I think it was critically important in bringing unity and focus to something that was having the danger of becoming a dozen different vendor-specific systems. So the power of the web is that there's one language we all use. And I think of that as coming from the World Wide Web Consortium.

Finally-- and I really am done-- I recently participated with David Shaw, who'll be speaking tomorrow, and others in assisting the President's Council of Advisors on Science and Technology in assessing the federal IT R&D program. And here's what PCAST said. Computer Science "is arguably unique among all fields of science and engineering in the breadth of its impact. Recent technological and societal trends place the further advancement of Computer Science squarely at the center of our ability to achieve all of our priorities and address all of our challenges."

That's what's computer science and computer systems are achieving. Thanks very much.

[APPLAUSE]

Again, if there are one or two questions, I'd be happy to take them. And if not, we will move on. We have a question.

AUDIENCE: Would you care to speculate on the relative roles, going forward, of universities in industry in computer science, or particularly in computer systems research?

LAZOWSKA: So that's a challenging one. The question is the relative roles of universities in industry. My view is that we have gone through cycles, from the university point of view, of believing that a huge proportion of the important work was coming from universities, and then troughs of despair. And my belief right now is that we are, in some sense, coming out of a trough of despair. Right? And the trough of despair is based on industry having access to vast amounts of data that we have not had access to and systems of a scale that we haven't had access to.

But I see that shifting. I see access of data, access to data, being much more-- data being much more easily available than it was in the past. I see scalable systems being available to universities much more than in the past.

I despair of the state of industry research, to be honest. In going back 30 or 40 years, our nation had AT&T Labs and the Xerox Corporation and IBM as a big part of the GDP pie in computing, and each one had a large, fundamental research organization. Today, that pie is a large amount bigger, and the companies like Salesforce and Oracle and Cisco, from my point of view, invest almost nothing, looking more than one product release ahead. Microsoft and a few other companies are notable examples.

So I think the opportunities are huge and, in fact, essential. It seems to me the role of universities is more important than it's ever been. Yeah.

AUDIENCE: Hi. I would argue that nature actually came up with computers and information processing before people, in the form of a cell. And it also has the ability to adapt to its environment. And I think we've already made progress with biological computers and bacteria that can have biological clocks and things. And I just wanted your opinion on the future of sort of biological computing and maybe using things like bacteria and cells to do computational stuff.

LAZOWSKA: I'm going to have to pass on that one, because I'm simply not nearly enough an expert to talk about that. Maybe Patrick will have something to say. But, again, this is a convergence that's going on, and it's just not something that I do. It's a great question, and I'm sorry I can't answer it adequately.

AUDIENCE: That's all right.

AUDIENCE: You talked about how the transistor revolution led to exponential increases in power, and now we've got multiple machines and multicore machines. What do you see as the pragmatic limit to computational power? Is it going to be literally the power that these things require, the density of the power? Or will be the complexity of programming them?

What's going to keep us really taking advantage of all of this marvelous hardware? Or will nothing get in our way?

LAZOWSKA: Right. So the question is, what will get in our way? And the answer, Randy, is everything that you've described will get in our way. All right? So I think we cruised along for many years, because Moore's law, which really talks about increasing transistor density, was able to lead to increasing single-core processor performance. And that powered this cycle in which, to state it crudely, Intel processors would get faster, so Microsoft could add more features to Word, so you would buy the next processor, then you'd add more features to Word. And we sort of rolled forward.

And when these single processors ceased to get faster because of thermal considerations, in particular, we're then in a situation where we need to use parallelism on a very large scale in order to get performance increases. And for embarrassingly parallel problems-- Monte Carlo simulation, things like that-- this is easy to do. Similarly, most web services.

But for word processing, for example, and many other applications, it's much less clear. And we've never done an adequate job of figuring out how to allow programmers to express parallelism-- how to extract parallelism from problems. The techniques we've used over the past decade, two decades, to tackle this are just not adequate. So we're facing a number of barriers right now.

I view this as the Full Employment Act for computer scientists and as another argument for why you can no longer be in your niche. These are problems that are going to be solved by architects and systems hardware people and systems software people and applications people working together.

DAVIS: Any other questions?

LAZOWSKA: Thank you very much.

[APPLAUSE]

DAVIS: Our last speaker is Professor Patrick Winston, the Ford Professor of Artificial Intelligence and Computer Science at MIT. He's going to talk to us about artificial intelligence. Patrick.

[APPLAUSE]

WINSTON: Let's see-- it's, um-- it's April. I think it was about a year ago when Victor asked me if I could give a talk today reviewing the past 50 years of artificial intelligence in a balanced way. And above all, he said, without insulting anyone, especially my senior colleagues here at MIT.

[LAUGHTER]

I've been trying for the past month to find a way to make that talk interesting. And last night, or rather early this morning, I gave up.

[LAUGHTER]

So my talk, in the end, will be about history but not just the past 50 years. It won't be balanced. And as far as insulting people, I'm not sure about that. I have to-- we'll wait and see how it goes.

I want to center my discussion this morning on a trinity of hypotheses-- what I call the "strong story" hypothesis, the "perception" hypothesis, and the "social animal" hypothesis. Because I think that, in reviewing the history, we see evidence that this-- that these hypotheses are what ought to take us forward into the next 50 years. So my talk will center on this.

And as I work toward these hypotheses I'll talk about some dark ages, some golden ages, some frustrations, some pioneers, some mistakes. And I'll also, by the time I'm finished, show how artificial intelligence has had consequences that I think we can use to make ourselves smarter. So by the time you leave here today, you'll be smarter than you were when you came in, as a little side benefit of attending the symposium.

So that's the direction I'm going with my discussion of the history of artificial intelligence. And I suppose we might start at the beginning. The problem is, it's hard to figure out exactly where the beginning is.

Everybody has to show a slide of Watson these days, and here's mine. It is what might be regarded as the beginning 100 years from now, because, after all, it does do something that's understood by the general public to require intelligence. The philosophers, however, are quick to point out that although Watson is a tremendous engineering achievement, there's some things that it can't do.

For example, if there were a conference on Watson, Watson couldn't attend. It has nothing to say about itself. It doesn't have a model of itself, so it can't participate in discussions of how it works.

And it isn't, of course, the first program that performed at a level that you would call "human expert." There have been many programs that perform at that level. And to find the first one, you have to go back at least 50 years, to the integration program written here at MIT by Jim Slagle. It was a tremendous achievement because it worked freshman calculus quizzes at a level that was right in there with the very best MIT undergraduates of the day. It was a testament to what one person can do in the course of a PhD program, just as Watson is a testament to what 30 people can do-- 30 smart people-- over a course of several years.

So it's not surprising that programs like Jim Slagle's integration program led us all the things that within a few years, maybe a couple of decades, we would have general intelligence. And what apparently we forgot or overlooked was the idea that it's much harder to produce programs that have common sense than it is to produce programs that behave at an expert level on very narrow technical domains.

So that was a wonderful year, 1961, because of the integration program. And it was also a wonderful year because that's when Marvin Minsky published a paper called "Steps toward Artificial Intelligence," a paper which laid out in great detail a program of research that he felt would lead to artificial intelligence. It talked about search, it talked about matching, it talked about probability, it talked about learning. It was an extraordinarily visionary paper.

But it wasn't the beginning, because perhaps the beginning was in 1957, when the Dartmouth conference was held. We call it a "conference," even though it was more of a workshop, because it took place over the entire summer at Dartmouth. It has, as you can see, extraordinarily ambitious goals that the participants felt might take a decade or two and which, in retrospect, perhaps they should have allocated a century or two for.

But it created a great deal of excitement. I wasn't there. I can't tell you much about it. I wasn't invited. I'm a little sore about it, but I was in eighth grade at the time, so maybe it's not a surprise.

[LAUGHTER]

But it wasn't the beginning. Perhaps the beginning was when Alan Turing wrote his paper "Computing Machinery and Intelligence." And we all know it for the Turing test, but actually its importance lay in the nine counterarguments to the possibility of artificial intelligence that Turing debunked. So, together with Marvin's paper, these form the seminal papers of the birth of artificial intelligence. Turing told us that it was possible to make a machine intelligent, and Marvin told us how.

But this wasn't exactly the beginning, because we can go back to Babbage's time and Lady Lovelace and see that they were already speculating on the possibility that computers could be intelligent. And Lady Lovelace wrote-- if I can put it in my own patois-- she said, don't worry about a thing. Computers can only do what we program them to do. Of course, she might have added, programs can do only what we program them to do, and then what we teach them to do, and then what they discover how to do on their own. But that, of course, wouldn't have had the soothing effect that she was looking for.

[LAUGHTER]

But that wasn't the beginning, either. Because really we have to go back to the Greeks. Aristotelian logic, after all, was invented by Aristotle. And those of us in the field sometimes say that Aristotle was the John McCarthy of his day.

And while he was the John McCarthy of his day, Plato might have been the Marvin Minsky of his day. He wrote the famous dialogue in which he talked about the organization of well-formed states, which was really an allegory to a well-formed mind. And he talked in terms of our mind being composed of parts which have to work harmoniously together.

And he called that piece of work *The Politeia*-- which is generally translated into English as *The Republic,* but everyone agrees that that's a mistranslation. It might have been called something more like *Society*-- the translation. And so Plato could have called his famous dialogue *The Society of Mind,* anticipating Marvin Minsky's book by 24 centuries.

So that takes us back to the beginning of thinking about thinking. But why stop there? We might as well go back to the beginning of thinking.

On the right we have one of us, and on the left we have what looks like a somewhat distorted one of us. And some think that that's exactly what it was. More commonly, though, it's labeled the Neanderthal.

And the Neanderthal was an amazing species, because it controlled fire, it made stone tools, it did all sorts of things. But what it didn't do was paint the caves at Lascaux and build, manufacture, design, make up the figurines at Brassempouy. In fact, we didn't do that either for 100,000 years after we became anatomically modern.

About 50,000 years ago, after we'd been around for 100,000 years, something very magical happened. Probably in southern Africa. Probably in a [INAUDIBLE] community of a few hundred or perhaps a couple of thousand individuals, somehow we got something that made us different. And the question is, what was that?

The great paleoanthropologist Ian Tattersall, from whom I borrowed these images, thinks it's that we became symbolic. But that's not a very complete technical view. What does it mean?

Noam Chomsky thinks that what it means is that somehow we developed the ability to take two concepts and put them together to make a third concept without damaging the component concepts and without limit-- an interesting point of view that I'll return to later in my discussion this morning. But rather than pursue it now, I think we'd best go back to the modern beginning of artificial intelligence, which was dominated by these pioneers-- Marvin Minsky, John McCarthy, and Newell and Simon.

Marvin taught us that we should think about heuristic programming. John McCarthy said we should bend logic to our will. Allen Newell was interested in building cognitive models of problem-solving, especially the models of what we do when we solve [? crypto ?] arithmetic problems and the like. And Herb Simon, in this very auditorium, explained to us that when we see something that looks complicated in behavior, we may discover that it's a consequence more of a complex environment rather than a complex thinker.

So all of these pioneers contributed a great deal by way of their own personal insights, but even more because these are the people who founded the original laboratories at MIT, Stanford, and Carnegie Mellon. And out of those laboratories eventually came a field overflowing with a cornucopia of ideas. Here are 50 ideas. These are just the first 50 that came to my mind when I decided to make a list.

And there are all manner of things on this list, from whole fields like machine learning to individual ideas like cross-modal coupling and intermediate features. So this is the stuff of which the field is made. But, before going into the applications of artificial intelligence, let me go back to the pioneers and add to it someone from the sort of middle dawn age, around 1970.

Having set up laboratories in the 1960s, the 1970s was an era in which research flowered. And one of the principals in that era was Roger Schank. Roger is a very interesting kind of person. If I were to characterize, I'd say that Roger is sometimes visionary, always arrogant, and never in doubt. I'm not sure why he never ended up at MIT, but that seems like it would've been a natural thing.

[LAUGHTER]

But Roger was the first to recognize the importance of understanding stories and developing any kind of account of human-level intelligence. He talked about restaurant scripts. Here it is.

John and Mary went into a restaurant. They sat down. They ordered hamburgers. They got up and left. They got angry and left.

And when we hear that story, we know that something went amiss-- that perhaps their food didn't come. And that's because we understand the story of the restaurant. And that's the sort of thing that Roger and his students worked on.

Of course, we at MIT were not idle. This was the era of the four Ws. Terry Winograd built a model-backed system for dialogue understanding. I built a model-backed system for one-shot learning. Dave Waltz built a model-backed system for understanding line drawings. And Gerry Sussman built a model-backed system that learned how to manipulate blocks. Of course, Sussman doesn't begin with a W, but we often suggest that he change his name to Wussman, just to make it consistent.

[LAUGHTER]

But that was an era of great excitement, because we were making a great deal of progress. And you notice that there are some common themes to these four pieces of work. I used the word "model-backed" in each case. It was easy to build models of the blocks world. And perhaps when we went on to more difficult worlds, it wasn't so easy.

It was also an era in which the computer that we worked on, the PDP-6s and the PDP-10s, were approximately 10,000 times slower than the laptop that's currently running this presentation. And with all the mega numbers that we've seen go before us this morning, 10,000 doesn't seem like a very big number. But just think about this. What if you could think 10,000 times faster? What advantage you would have.

Or, if that doesn't impress you, if you had 10,000 times as much money, think what you could do. So 10,000 times is a big deal, and that computer has 10,000 times as much memory, too. So it's a miracle that we got anything done, back in those days, working on a computer that could barely be called a computer by today's standards.

But now this dawn age, from 1960 to 1980-- it includes the work of David Marr. Any discussion of the history of artificial intelligence has to talk about David Marr, has to include him, because he was an extraordinarily charismatic, transformational figure in the field. But strangely he's known less today for the particular work that he did than he is for the way he did it.

He championed a view of cognitive science and artificial intelligence that emphasized three layers of explanation. You start at the top, with the computational problem you're trying to understand. Then you move on to thinking about the constraints and the overall shape of a solution. And only when that is in place do you start thinking about the algorithms and the hardware that implement the method that deals with a computational problem.

It seems so obvious. It's just the scientific method. In fact, so obvious to us computer scientists that David Marr is known more in cognitive science than he is in artificial intelligence. There's a Marr Prize at the Cognitive Science Society every year.

But we should pay more attention to this, because we computer scientists love our gadgets. And when someone develops an extraordinarily cool idea like boosting, it's awfully easy for us to become excited about it and look for problems to solve with it, rather than saying, here's my issue, here's my problem, what has to be brought to bear to get on with it?

Now that takes us to 1980, when David Marr tragically died. And the early '80s were characterized by-- well, let's see. I suppose I could call "enthusiasm for entrepreneurship." One man's enthusiasm, of course, another man's hysteria. This is the five-year period when Ed Feigenbaum was teaching that the Japanese would soon overwhelm us because of their happy embrace of rule-based expert systems and expert-system technology.

What did we learn from that era? Perhaps mostly that if you're going to start a business, it better solve somebody's problem or provide somebody with a new toy. You can't build a business around a technology. There were vast conferences held in which people were trying to sell expert-systems technology to each other. And so that era naturally could be described as a kind of bubble.

But at the same time most people were distracted by this entrepreneurship phenomenon, it wasn't a time when everyone was idle. In a remarkable two-year period, 1987 and 1988, five seminal works were produced, three of them in book form. Marvin Minsky wrote his book *Society of Mind,* 24 centuries after Plato could have written something with that title, and refocused his attention on how humans think, instead of an exclusive attention to how thinking might work in the abstract.

Judea Pearl wrote a seminal work in which he stressed the value of probabilistic methods in artificial intelligence-- an idea which remains increasingly popular today. And finally, McClelland and Rumelhart wrote their classic two-volume work on neural nets. Wasn't the first time neural nets were discussed. McCulloch and Pitts did it in the '40s. It seems to be an idea that comes around every 30 years or so.

So these three books were landmarks in that era. But they were also joined by two papers in the *Artificial Intelligence Journal* that were equally influential. One was the paper on the Soar architecture, written by Newell and his students and colleagues at Carnegie Mellon-- an architecture that laid out their view of how short-term memory, rules, long-term memory, preference analysis-- how all these things can be put together to provide an architecture for human cognition.

And finally, last but not least, the well-known paper by Rodney Brooks in which he taught us that we were all wrong and that our work was wrongheaded because all of the attention to representation and models and so on was something that he felt was less material than it should have been viewed as. Those of you who are looking at the slide with binoculars might note that although the paper was submitted to the *Artificial Intelligence Journal* in 1987, it wasn't actually printed until 1991. In the interim, the journal had dropped the practice of including the names of the recommending reviewers on the title page of the article. Probably a coincidence.

One year after that, by the way, there was another significant event, not causally related, and that was the fall of the Berlin Wall, which had a profound influence on how funding for artificial intelligence was done. And for that and other reasons, we were drawn more and more toward the world of applications and less and less toward the science side of artificial intelligence, in which we inquire into, what is the nature of the computation that takes place in here?

In some ways, it was a golden age, because all of this stuff became the stuff of computer science. And George [INAUDIBLE] once told me, you've won, because it's impossible to build a big system today without its incorporating some aspect of artificial-intelligence technology. So there've been thousands of such applications that use some aspect of artificial in technology. Here are three.

There's Rod Brooks's Roomba robot, by his company iRobot. It uses subsumption architecture to do its work. The second image is of an airport. It's from my company's work. We develop resource allocation systems that allocate expensive resources, like people and airport gates. And then the lower right-hand corner is the work that brings tears to my eyes whenever I think about it. It's the work of Eric Grimson and students on image registration, on the registration of video images with fMRI images, with MRI images, so as to make it possible for brain surgeons to do their work with more precision and less collateral damage.

So a lot of all this. The past 20 years have been a golden age. On the other hand, in terms of how much progress we made toward understanding what goes on here, I think it's fair to say that all of us feel the progress has been less rapid than we would have expected. Not exactly a dark age, but when you combine these advances and applications with the slow pace of understanding of human intelligence, maybe it's a grey age, and it's time to do something about it.

And many people believe it's time to do something about it. We have, at MIT, the Intelligence Initiative, after all. But my personal view of what specifically ought to be done about it is that we ought to go back to 50,000 years ago and ask more questions about what makes us different.

Chomsky says it's putting two concepts together to make a third without limit, and without damaging these two down here. And I would add that if we can do that, then maybe what we can do is that we can begin to describe events. And maybe when we can describe events, we can string them together into stories. And when we can string them together into stories, perhaps we can combine and merge and mix them together to create new stories.

And when we can do that, perhaps we can introspect in how we did it and build models of ourself and models of someone else. So that's what I think ought to be done. And it leads me to what I call the "strong story" hypothesis. I suppose I call it the "strong story" hypothesis because I mean to say that there's no such thing as a weak story hypothesis. Maybe a weak story hypothesis would say that this is merely important.

When you call it "strong," it means it's supremely important. This is what we do. This is what makes humans different. And our stories, the stories we work with, come from all over the place, from our microculture, our family, our personal experience, our macroculture, our country, our religion.

We start getting washed in stories with fairy tales, in childhood. And then in high school we have more stories in the form of literature and history. And then we end up in professional school with case studies.

Roger Schank says that all of education is about storytelling. Of course, I couldn't believe this without doing something about it. So here's a glimpse of the genesis system, which I have my students work on, analyzing a short English description of the plot in *Macbeth.* It's being analyzed two ways, because two variants on the system have been put to work on the same description, producing different perspectives.

All that stuff up there in white is stuff that was explicit in the story. Everything in grey was inferred. And everything in yellow was discovered by looking across the story and finding patterns in it and looking at this more carefully.

What we see is that the system on one side looks at it from a kind of dispositional point of view that says that one of the killings was a consequence of bad behavior. Another one is more situational and says that it was a consequence of the situation in which the actors were immersed. So one is a kind of Western point of view, our point of view, very individualistic, and the other's much less so. It's more situational.

And Randy is telling me I must come to an occlusion. I wish this laser were 50,000 times more powerful. Then I could talk as long as I wanted. [LAUGH]

[LAUGHTER]

So I'll race to an end and fill in the details of the bicentennial. So the strong story hypothesis is not alone. We think with our perceptions, too. And they're importantly integrated into the entire package.

So what do I mean by "direct"? I can say, is there an organ in this hall? And once you've done the disambiguation, your eyes will go over there and say yes. So your language system has commanded your visual system to do something in its behalf.

What do I mean by "imagine"? Imagine what it would be like if I ran across the stage with a full bucket of water. You all know what would happen. My leg would get wet. But you've never read it, you've never seen it, you've never experienced it. But you can still know that.

So our perceptual systems provide tremendous problem-solving capability. And it's interesting that they're related.

Here's a puzzle for you. What am I doing? This is not a trick question. The one-word answer is I'm drinking.

Now what am I doing? Also not a trick question. I'm toasting.

What's the cat doing? It's drinking. And how do we know that? How do we know that, in spite of the fact that left and the right look more alike than the one in the center, but the center is more like the one on the left? And we give it a name, because we have some internal language that tells a story. And the story on the left and in the middle is the drinking story and on the right is the toasting story.

So we have an inner language-- an innerese. And maybe that innerese accounts for this labeling, and it accounts for our storytelling capability. Yes, okay. So we have an innerese. Which came first-- our communication language, the one I'm using now, or the inner one?

Well, plainly, the inner one, because there's no point in talking unless you have something to say. Of course, this is not observed in all walks of life. We have politics and so on. But, for the most part, that's true.

And then, after all, another thing is that there's no point having something to say if there's no one to say it to. Hence the third of the hypotheses in this trinity.

So I was once talking with Danny Hillis. He came into my office and said, have you ever had the experience of having difficulty explaining an idea to Marvin Minsky? Sure, I said. Is it the case that you've noticed that Marvin is somewhat impatient? Absolutely, I agreed.

Then he said, has he ever guessed your idea before you've had time to actually get it out? Every time, I said.

[LAUGHTER]

So he said, so how could you go about making yourself smarter? And the answer is, talk to somebody, preferably someone smarter than you are. And if you can't find anybody smarter than you are, a friend will do.

Sometimes students come to me and say, well, what if I don't have a friend?

[LAUGHTER]

In that case, I tell them, you can talk to yourself, because it does the same kind of thing that you see happening when you talk to someone else. Now to get the full experience you have to talk out loud. But this, of course, has to be done with some care in social situations.

So there it is. The way forward, in my view, is this trinity of hypotheses. And so when I come to give this talk at the bicentennial, I think I'll be able to say that a focus on these hypotheses, challenge problems that come out of these hypotheses, work done in support or in refutation of these hypotheses will have tremendous benefits, because we'll have these contributions come out of that kind of work in the next decades.

We'll be able to understand ourselves better, because we'll know how we work. We'll be able to understand other people better, because we'll be able to understand how they are the product of the stories in their culture, which will be different from the stories in our culture. And finally, we'll be able to make ourselves smarter in important new ways, because we'll be able to use educational technology to better engage our problem-solving apparatus, instead of merely increasing the pressure in the fire hose and spraying the bits past us at an ever-increasing rate.

So that's the story of artificial intelligence as it has unfolded so far. And you might say, well, where are we, exactly? Well, for that I have to borrow a phrase from Winston Churchill and say that where we are is not the beginning of the end, but it might well be the end of the beginning.

[APPLAUSE]