- Table of Contents
- Participants
- The Marshall Scholarships

- Home

Address

Philip Power: Good morning. In remorseless pursuit of our timetable, which envisages a busy and crowded day, I think that it's time to get going.

Welcome to the second session of the Marshall Symposium. I'd like to get some housekeeping matters out of the way. For those of you who don't have programs, they are available in the lobby. There will be a fifteen-minute break between the first and the second panel, at approximately ten forty-five. There will be coffee and pop available in the lobby, but we will resume our panel discussions promptly at eleven. We are hopeful of provoking as much interplay between our panels and the audience as we can. To facilitate that, there are microphones at each aisle, and so at the end of panels, people are encouraged and invited to ask questions.

Our morning speaker is a most distinguished trans-Atlantic visitor who suffered the indignities inflicted on many airline travelers at the hands of Northwest Airlines and its associated unions. Roger Needham, who is a pro-vice chancellor of Cambridge University, was born in 1935 and has been in computing at Cambridge since 1956. His Ph.D. thesis in 1961 concerned the application of digital computers to problems of classification and grouping. In 1962, he joined the computer laboratory, which was then called the mathematical laboratory, and has been on the faculty since 1963. He took a leading role in Cambridge projects in operating systems, in time-sharing systems, in memory protection, in local area networks and in distributed systems over the next twenty years. He was made head of the computer laboratory from 1980 to 1995, was made professor in 1981, was elected to the Royal Society in 1985, to the Royal Academy of Engineering in 1993. He is a member of the Technology Foresight panel and became pro-vice chancellor of the University of Cambridge in January 1996. And he has been the director of Microsoft Research Limited since 1997.

Please join me in welcoming our distinguished guest to open the symposium.

Roger Needham: Good morning, and it's a great pleasure to be

here.

Given my somewhat ill-assorted dual affiliation, I ask myself on occasions like this, "Who do you want to be today?" Today, I'm an academic, and I come here with fraternal greetings from Cambridge University, which has been privileged to host a good number of Marshall students over the years.

A while back, I was visited by two civil servants. And they said to me that people in high places doubted that computing research existed and could I help them do something about this. This was a bit of a surprise, because when you've devoted yourself to something for 40 years, to be told it doesn't exist is a bit odd. But I subsequently found that this strange view is not confined to the British Civil Service. Therefore, when I had done what I was asked, which was to write one page of A4 on what computer science research was - one side of 8½ by eleven to you - I thought it might be worth saying something about it here.

I think people get strange ideas about computing research first because computer science isn't science. Scientists seek to unpick the secrets of nature. Computing researchers are trying to make something happen and obtain the necessary knowledge and understanding so that they can.

Computing research is there to set the agenda for the future. Research doesn't make products, it doesn't sell things, but it provides the insights, the skills and the understanding that enables people to make the things that will be the stuff of the future. If you're a small company, you can get your agenda out of the intellectual atmosphere around you just by being alert. If you're a big company, or a university, you can't. And this is one thing that both my employers have in common. It's probably the only thing they have in common - that they're concerned with making the agenda for computing of the future.

When I was writing my one side of paper, I divided computing research into four kinds, and I'm going to say something about these in turn.

First, algorithms. Algorithms are what computers do. You can't get a computer to do something unless you've got an algorithm for doing it. Conversely, a computer that is doing something which isn't an algorithm is considered to be broken. This means that the study of algorithms is absolutely central to progress. Making computers do better what they either can't do at all or don't do very well - that's what the study of algorithms is. And it's a sobering thought that only 35 years ago sorting things into order came under that heading. Only 35 years ago. Only 30 years ago, public-key cryptography was thought to be impossible. It was thought to be such an absurd idea that it wasn't worth anyone spending their

time seeing if they could find a way of doing it. It's a fascinating story how it was noticed in a completely different context, not even digital, that it was possible for two people to communicate confidentially without sharing a secret. And as soon as that had been noticed, that it wasn't a self-contradictory idea, people got down and discovered good ways of doing it.

Today, there are still challenges. And quite a few of the challenges still relate to cryptography and its applications. Not so much how you do the cryptographic algorithms, which is pretty well understood, but in relation to a challenge faced by all algorithm research. There is a temptation to say the problem is what my algorithm solves. Where somebody came to you with a problem, you work away at it, you do your best, you get an algorithm which does something like the problem and you say, "That's what your problem really was." Which is a very offensive thing to do, if you look at it that way. And it's almost always wrong.

Butler Lampson, a well-known and distinguished American computer scientist, says, "Anybody who asserts that a problem is readily solved by encryption, understands neither encryption nor the problem." Curiously enough, he attributes that remark to me, as well as me attributing to him, which may say something about the nature of the evening when it was thought up.

I've indicated some other areas where algorithmic progress is needed.

Image retrieval is one. We heard yesterday about the prominent place of images in computer applications of the future. Something we can't do now is to write a program, write an algorithm such that you come and get in front of a digital video library and say, "Find that scene where the pretty redhead rides off bottom right on the white horse." Nobody has the faintest idea how to do that. But we could all do it if only we could look at the whole video library in parallel. That is a challenge for the future, and there are many more of the same sorts. It is actually possible to give an entire talk around that subject. Do you need to be able to distinguish a horse from a genetic quadruped, because people don't write movies in which people ride off on white anything other than horses. Not cows or polar bears or things like that, and other comments which are not quite politically correct.

Handwriting recognition is another. We don't do that very well with cursive handwriting. Nobody actually knows how useful it would be to be able to do good quality recognition of cursive handwriting, but it's certainly a challenge. It's an example of something that is regarded as a challenge to computing research, although people are not very good at it. I can't read my wife's writing at all. Frequently, I pick up the shopping list and I say, "What the devil is this?" Then, quite often, she can't, and her

writing is not unusually bad. Trying to get computers to do things we're not very good at is a slightly fraught art.
The last thing I've put down here is steganography. Steganography is a slightly fancy word for hiding information. It's come to the fore because of the desire to watermark images and sounds, or fingerprint them in some way, for the protection of intellectual property, or at any rate for the detection of misuse of intellectual property. This is an algorithmic piece of research, though people often don't think of it that way. And it is certainly a pretty difficult one. My academic colleagues at Cambridge believe that they can destroy any copyright watermark that has been put on the market and published in sufficient detail for it to be possible to do the experiment.

I hope I have given you some of the flavor of the range of algorithmic research. Some of it can be quite abstract; some of it's highly mathematical. But what makes it belong to computer science is that it's goal-directed to enable you to write an algorithm to make a computer do something.

The next thing is process. What I mean by this is principled improvement to the process of implementing programs and systems. Techniques that will make it easier to get them right, or make it quicker to get them right enough, depending on where you sit in the marketplace. Theory of programming languages, types and the like, all fall in this category - showing that this program does that algorithm, showing it, that is, by formal reasoning, often itself computer-supported. The goal of all this sort of stuff is to make it less likely that we shall get unpleasant surprises from what computer programs do for us. A very current issue is code that will work everywhere; "mobile code" is what the practitioners call it. It was remarked yesterday what a change it is for code to get executed in an unknown environment. You pull it off the Net, and you execute it and you hope. You first of all hope that it won't do anything nasty to you. But then you hope that the environment in which you place it is sufficiently like the environment it expects to be in, that will actually do the right thing, quite apart from issues of malice. In order to do that, you've got to have proper formal descriptions of what the environmental assumptions are and what the code object in hand is. You can, if you like, look at a whole range of development of this sort as being successive relaxation of assumptions. You start off with programs which will run on one particular computer. The first programs I wrote would run on the Edsac and nothing else, and there was very little else for them to run on then. Then you have programs that will work on any Unix machine. Today, you want programs that will work on any machine, pretty well. And the more it comes to be the case that there's diversity of environment, the more you have to be precise and inclusive about what you're assuming. This business, mobile code, is an extremely active area of research at the moment. Probably will be for a year or two. And then, it will be done and

people will move on.

The third point was collectives. This again was mentioned yesterday, computers in very large numbers. Coping with unreliability and reasoning about it. This is not a very advanced art yet. People have made all sorts of mistakes, like saying that replication produces reliability. I learned that was not true at Xerox PARC in about 1980, where I was concerned with setting up a message system which spread all over the Xerox corporate net. And we were very proud of the fact that if you were trying to send a message using one message server, and it failed, then it would try another one for you, hopefully seamlessly, and you would sit at your work station and never know that anything had gone wrong.

Well, that was all right, until a poisoned message arrived. It arrived from Berkeley, but it wasn't their fault; it was an accident. The first server in Palo Alto died. The second server in Palo Alto died. The one in El Segundo down in L.A. died. One in Dallas died. One in Stamford died. One in Tokyo died. One in southern Britain died. And by this time the first one had started up so it could be killed again. And so it went on. Replication does not equal reliability. Replication is good for reliability if things die for casual reasons. If they die for certain, predictable reasons, replication has contributed nothing to you. We still don't really know what to do about that.

Even if you could have some centralized agency which looked down on the whole thing - call it Galactic Central - and could see everything that was going on, you wouldn't quite know how to write the program in it which would detect that something very strange is happening, and a stop needs to be put to this nonsense while you find out what it is. Even if you could do that, Vinton Cerf would quite properly detonate at the idea of having such a Galactic Central, and we know even less about how to do such a thing on the basis of limited local knowledge - everyone looking around and coming to some sort of conclusion that something strange is happening. When one's talking about huge nets and uncertainty, one needs to engage in probabilistic modeling. And very remarkably over the last two or three years, it's turned out that some of the methods developed for statistical physics purposes can be applied to this sort of thing. There's collaboration between Microsoft Cambridge and the Dublin Institute of Advanced Studies, precisely in this sort of area.

It's going to get more serious and important if, as Vint suggested last night, the Internet becomes really all-pervasive and includes the light switch and other appliances like that. We will be dealing with such large numbers that it will never be the case that it all works at once, that any assertions you make about its behavior have to be made on the basis of counting and inference. And also,

an observation: a great deal of study of networks that has gone on has been done on the basis of assuming statistical distributions of the behavior of things, which frequently are completely unwarranted. Assuming for example that if you've got a switch, at every input to that switch packets arrive randomly distributed for all the other outputs. The world isn't like that. How much can you do by measurement and inference? How long do you have to look at the behavior of a network in order to be able to predict its behavior in the future? The answer turns out to be not very long, unless you are interested in seeing very far into the future. And, if you always knew what the weather was going to be like tomorrow, with great accuracy, you'll be less interested in what it's going to be like next week. If you can see a bit of the future, it may not as good as being able to see a lot of it, but if you could see all of it, life would be very dull or very alarming.

The last of the four areas is applications. This is a rather different kind of research. I'm not so much here talking about specific applications here such as working out your tax return and things like that. I'm talking about styles of application, styles of interaction. People have to think of these things, and this is where my original title of "Think it Up and Try it Out" came from. But you can't predict what people are going to find helpful. You can't predict what specific sorts of people are going to find helpful and natural and comfortable, and you have to try things out and see. And this is a very different style. It gets away from algorithms, from mathematics, and gets involved with people.

Much of our present image of computing - windows and mice and pull-down menus and that sort of thing, things that we now regard as just the way it is - derive from research done at Xerox PARC in the '70s. What we tend to forget if we were there, or not to know if we weren't. is that a huge amount of experiment went into that sort of thing. A huge amount of approaches to doing things at Xerox were tried out and rejected because people didn't like them. We're still mining the research of the 1970s. And it's about time to move on. It's not at all easy to know how you do it.

A particular point about all that research was that it was done by computer research people in order to find a good environment to help them do what they spend their lives doing. So, the people who thought of it were perfectly good people to try it out on, and so were their colleagues. You would try somebody's idea out who was down the hall, and you would say, "I find this a very crummy way of doing it for the following reason." And they would say, "Well, I suppose we could fix that," or maybe, "No, that completely violates the structure." "If you think in that way there's something wrong with you," I've heard said, which has a slight tendency to be arrogant about it, but arrogance is a characteristic of computer people, as many have noticed. So we have a problem. It's not nearly so easy when the users aren't like the experimenters.

It's a methodological issue. It's easy to imagine what's better for me than to imagine what's better for them, particularly when "they," and what they're trying to do, are about as unlike me as they come.

I don't actually think this methodological problem is well-solved and well-understood. And what it tends to lead to is a huge number of options for the user to configure to match his particular quirks and preferences. This might be all right if he didn't find getting about 157 options so daunting that he has to get somebody to do it for him, and the idea of tailoring it to yourself is completely lost. I think this is a challenge for computing research in the future, and it's going to continue to be a challenge, particularly as computers get to be used in the sort of pervasive way we heard about yesterday.

I'll close with an interesting question. I've called it, Where Do You Live? I'll try to convey what I'm talking about. When you sit and use a computer, there's an environment, a software environment to which you are typing, assuming that typing is what you are doing. In the sixties, it was a command-line interpreter. Whenever you wanted to do anything, you typed the appropriate command and you entered the program that did what you wanted. In the seventies, and later, people said, "We don't want this at all. What people spend all their time doing is editing documents, whether they be programs, texts or what have you, and you should spend your time living in an editor." Unix people lived in Emacs, a practice that I've always thought totally perverse, but most of them did. And you can look at that, sitting in an environment which is helpful for reaching out in your local filing system and doing all the various things you might want to do. Today, when we want to reach out further than the local filing system, reach out into the Internet, the fashionable idea is that where you sit, the cockpit in which you act is a Web browser, and this is why they're thought to be such important things.

What's it going to be in ten years' time? I haven't the faintest idea, but I think it's a major challenge to computing research to try to see that coming, be ready for it, or even, of course, create it. An observation was made in connection with UK government's Technology Foresight program - I can give it you in Latin, but then I'll translate it. Futura formare prospicere melius - "It is better to create the future than to try to foresee it." And that's what research is about. Thanks very much.

Audience member: Would you like to answer questions?

Roger Needham: Sure, if there's time.

Philip Power: I would only say this, that Professor Needham's lecture is a magnificent demonstration of the British tradition of

the powerful and succinct lecture.